



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

D7.5 DATA MARKETPLACE: SOFTWARE PROTOTYPE

Dissemination Level	PU
Due Date of Deliverable	31/10/2021, M22
Actual Submission Date	28/10/2021
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.74

Abstract: The deliverable D7.5 Data Marketplace: Software Prototype describes the initial demonstrator of the PolicyCLOUD Data Marketplace. The latter will be 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	06/09/2021	ToC	Vasilis Koukos, Argyro Mavrogiorgou (UPRC)
0.2	15/09/2021	Contribution in Sections 1, 2	Thanos Kiourtis (UPRC)
0.3	28/09/2021	Updates in Sections 2.2.1	Vasilis Koukos (UPRC)
0.4	05/10/2021	Updates in Sections 2.2.2	Eleftheria Kouremenou, Alexandros Raikos (UPRC)
0.5	12/10/2021	Contribution in Sections 3, 4	Argyro Mavrogiorgou, Thanos Kiourtis (UPRC)
0.6	19/10/2021	Check and revision of all Sections	Argyro Mavrogiorgou (UPRC)
0.7	22/10/2021	Review	Giannis Ledakis (UBI), Panayiotis Michael (ICCS)
0.8	25/10/2021	Changes based on review comments	Vasilis Koukos, Eleftheria Kouremenou (UPRC)
0.9	27/10/2021	Quality check	Argyro Mavrogiorgou (UPRC)
1.0	28/10/2021	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
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.....	7
1 Introduction.....	8
2 Prototype Overview.....	9
2.1 Main Components.....	10
2.1.1 Back-end.....	10
2.1.2 Front-end.....	10
2.2 Interfaces.....	11
2.2.1 Back-end.....	11
2.2.1.1 Interfaces related to Users.....	13
2.2.1.2 Interfaces related to Descriptions.....	25
2.2.1.3 Search functionality on Descriptions.....	51
2.2.1.4 Interfaces related to Assets.....	54
2.2.1.5 Root Interface.....	58
2.2.2 Front-end.....	59
2.3 Baseline Technologies and Tools.....	69
2.3.1 Back-end.....	69
2.3.2 Front-end.....	69
3 Source Code.....	72
3.1 Availability.....	72
3.1.1 Back-end.....	72
3.1.2 Front-end.....	72
3.2 Exploitation.....	72
3.2.1 Back-end.....	72

3.2.2 Front-end.....	72
4 Conclusion.....	73
References.....	74

List of Tables

Table 1 - Back-end's interfaces related to Users	13
Table 2 - Register a new user Interface.....	14
Table 3 - Check availability of a username Interface.....	15
Table 4 - Authorize a user (Login) Interface	16
Table 5 - Verify users Interface	17
Table 6 - Resend verification code to users Interface.....	17
Table 7 - Get user's information Interface.....	18
Table 8 - Update user's information Interface.....	19
Table 9 - Change user's password Interface	20
Table 10 - Reset user's password request Interface	20
Table 11 - Reset user's password Interface.....	21
Table 12 - Delete user's account Interface	22
Table 13 - Change user's email Interface.....	22
Table 14 - Change user's profile picture Interface.....	23
Table 15 - Remove user's profile picture Interface.....	24
Table 16 - Get user's statistics Interface	24
Table 17 - Get user's account data Interface	25
Table 18 - Back-end's interfaces related to Descriptions	26
Table 19 - Get descriptions' collections Interface	27
Table 20 - Get a list with all descriptions Interface.....	29
Table 21 - Get a list with all descriptions from a specific collection Interface.....	29
Table 22 - Get a specific description (using keyword "all") Interface	30
Table 23 - Get a specific description (using description's "collection") Interface	30
Table 24 - Get latest descriptions from all collections Interface	31
Table 25 - Get latest descriptions from a specific collection Interface.....	32
Table 26 - Get random descriptions from all collections Interface.....	32
Table 27 - Get random descriptions from a specific collection Interface	33
Table 28 - Get a list with all descriptions provided by a specific user (using keyword "all") Interface	34
Table 29 - Get a list with all descriptions provided by a specific user and under a specific collection (using a "collection" value) Interface	35
Table 30 - Get descriptions' statistics (useful for front-end's homepage) Interface.....	36
Table 31 - Upload / Create a new description with random ID Interface	37
Table 32 - Upload / Create a new description with given ID Interface	39



Table 33 - Update a specific description (using keyword “all”) Interface	40
Table 34 - Update a specific description (using description’s “collection”) Interface	41
Table 35 - Delete a specific description (using keyword “all”) Interface	42
Table 36 - Delete a specific description (using description’s “collection”) Interface	43
Table 37 - Delete all descriptions Interface	43
Table 38 - Delete all descriptions from a specific collection Interface	44
Table 39 - Make a review for a description Interface	44
Table 40 - Update an existing review for a description Interface	45
Table 41 - Delete a review for a description Interface	46
Table 42 - Get a list with the reviews made by a specific user Interface	47
Table 43 - Get a list with all descriptions that need permission Interface	48
Table 44 - Get a list with all descriptions from a specific collection that need permission Interface	49
Table 45 - Approve or reject a description that needs permission, using keyword “all” Interface	49
Table 46 - Approve or reject a description that needs permission, using description’s “collection” Interface	50
Table 47 - Approve or reject all descriptions that need permission, using keyword “all” Interface	50
Table 48 - Approve or reject all descriptions that need permission under a specific collection, using a “collection” value Interface	51
Table 49 - Back-end’s search operators	53
Table 50 - Back-end’s interfaces related to Assets	54
Table 51 - Get a list with the stored assets Interface	54
Table 52 - Get a specific asset using its ID Interface	55
Table 53 - Upload a new asset with random ID Interface	56
Table 54 - Upload a new asset with given ID Interface	56
Table 55 - Update a specific asset using its ID Interface	57
Table 56 - Delete a specific asset using its ID Interface	57
Table 57 - Delete all assets (administrators’ action) Interface	58
Table 58 - Root Interface	58

List of Figures

Figure 1 - Data Marketplace architecture	8
Figure 2 - Data Marketplace's layers and main functionalities	9
Figure 3 - Front-end's headers	59
Figure 4 - Headers view from Home page.....	59
Figure 5 - Sub menu item view from login page	60
Figure 6 - Discover's sub-items redirect to Discover page	60
Figure 7 - Home page: upper view	61
Figure 8 - Home page: lower view	61
Figure 9 - Register form	63
Figure 10 - Login Form	63
Figure 11 - Account overview for a simple user	64
Figure 12 - Account information and assets for a simple user	64
Figure 13 - Account assets for an owner	65
Figure 14 - Account assets and information for an owner	65
Figure 15 - Discover page.....	66
Figure 16 - Single asset page for logged in users.....	67
Figure 17 - Single asset page owner	67
Figure 18 - Single asset page for unauthorized users.....	68
Figure 19 - Error message bar.....	68
Figure 20 - Front-end access Middleware.....	70
Figure 21 - Dashboard add settings.....	70
Figure 22 - Dashboard admin view settings.....	71
Figure 23 - Token based actions	71

Executive Summary

This deliverable (entitled “Data Marketplace: Software Prototype”) describes the initial demonstrator of the PolicyCLOUD Data Marketplace. The Data Marketplace will be a unified web-based platform consisting of two (2) core services, its front-end and back-end, offering its users various ready-to-use solutions.

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), 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 Marketplace.

1 Introduction

The deliverable D7.4, entitled “Data Marketplace: Design and Open Specification” and delivered in August 2021, was about the initial design and the architecture of the PolicyCLOUD Data Marketplace. As described and analysed in D7.4, 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 the 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 the PolicyCLOUD.

This deliverable describes the first version of the implemented prototype of the Data Marketplace, and it is an extension of the deliverable D7.4. In summary, the 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 which 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 different requirements of the project’s stakeholders.

The remainder of this deliverable describes an overview of the Marketplace architecture in section 2.1, detailing the main features of its core components that are also described in deliverable D7.4. In section 2.2 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 marketplace. Section 3 provides some access information to the source code, and finally, section 4 concludes with a summary of the described prototype.

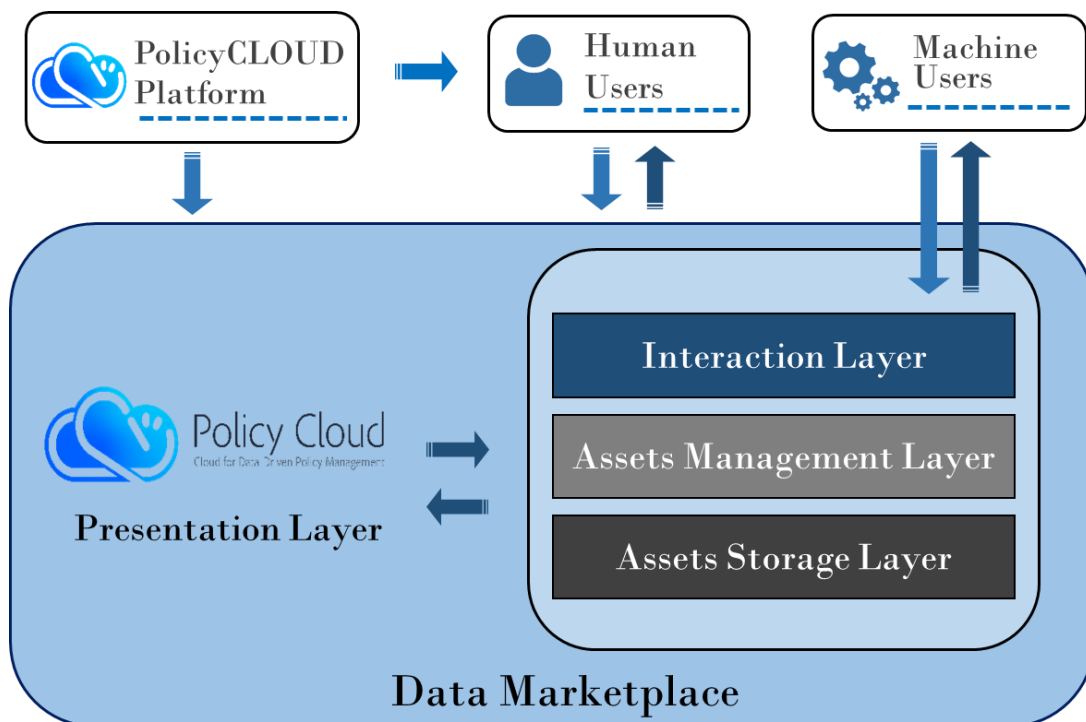


FIGURE 1 - DATA MARKETPLACE ARCHITECTURE

2 Prototype Overview

The PolicyCLOUD Data Marketplace is a public web-based environment with various APIs, able to store several types of assets. It has been structured and developed having two 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 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.

Generally, the Marketplace provides several functionalities that are mapped to different layers. The back-end includes three layers (i.e. Assets Storage Layer, Assets Management Layer, and Interaction Layer), while the front-end includes one layer (i.e. Presentation Layer) that in full consists of four different layers (as depicted in Figure 2) that realize its capabilities. The four layers of the 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 Marketplace's assets.
- The Interaction Layer (part of the back-end) supports the communication between the market 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.

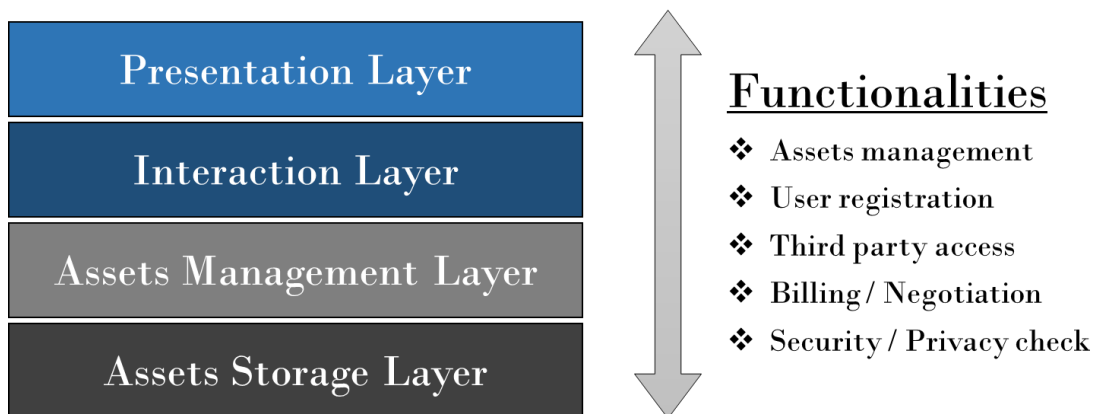


FIGURE 2 - DATA MARKETPLACE'S LAYERS AND MAIN FUNCTIONALITIES

2.1 Main Components

2.1.1 Back-end

The back-end is the main component of the marketplace. It consists of three different layers and implements the main functionalities for assets management. The three levels 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 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 principles and techniques for their management. Specifically, this layer handles the assets from the moment they are entered into the platform through the APIs and then stored in the database (in Assets Storage Layer), until their final deletion from the platform. Through this layer, the market platform 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 perform an operation/ trigger a functionality. Moreover, there are mandatory description files for all assets that contain metadata about the described asset (in JSON format). These description files are mandatory in order to make the assets searchable and retrievable by the end-users of the Data Marketplace.

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 market platform, 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 will convert all interfaces of the back-end (REST API) into user friendly interfaces and provide 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 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 marketplace (user-based platform), upload their offered assets by filling out appropriate forms whose fields will be the content of the description files of the assets (as mentioned in Section 2.1.1); search for assets according to various fields

(title, asset's type, fields of use, provider, other metadata, etc.) that can be further filtered or even sorted by the number of views or the date they were uploaded to the marketplace, etc. Also, there is a page that presents in detail the information of the assets, and through this page, the users are able to retrieve the real assets, the files. More details about the front-end and its supported functionalities are described in the next section, 2.2.2.

2.2 Interfaces

This section provides the description and the core details of the interfaces for both components, 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 3 main groups, namely: APIs related to Users, APIs related to Descriptions and 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), to the fact that the assets are offered by their providers / owners to all users without special restrictions (at least in its first version) something that 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 the author of the asset, the providers can specify who is the real author.

Thus, all users of the Marketplace should have their personal accounts in the system, which they will be able to manage themselves. As is common on all websites with such requirements, the Marketplace's administrators are able to audit the accounts and perform actions that will ensure the platform's smooth operation, but also of the community that will be created through the Marketplace. To this end, the back-end has implemented APIs that are related to its users.

As already described, the Data Marketplace consists of two components, running two different servers but both 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 (metadata files) are stored in the back-end, as does 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, which provides 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 the 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 compared with the rest of the PolicyCLOUD components, supporting its own authentication and authorization mechanisms to manage its content.

As authorization mechanism, 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 query parameter in the URL or inside the HTTP header, and 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' 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.eyJ1c2VybmFtZSI6ImZlZS16InZrb3Vrb3MiLCJuaWYiOiJmFzaWxpcyIsInN1cm5hbWUiOiJlb3Vrb3MiLCJpcmdhbmI6YXRpb24iOiJlVUJFZDliwiZWhwIjoxNTE2MjM5MDIyOTQ.31osTPhrxNFWN-moZsDFeiQq6HcOEe7svcGCGnjI9IA
```

The content of a JWT can also be the key "exp" which sets an expiration time for the JWT, and that reduces 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 will be as follows:

- The Marketplace will restrict the 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 descriptions, the requests to these interfaces will be restricted too, but not for the GET HTTP request because the descriptions should be accessible to all (with limited content) because the Marketplace has to promote its contents to the public.
- The users of the 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 to authorize themselves, using their credentials. Their authorization will result to the retrieval of a JWT, which they will use in their HTTP requests to the Marketplace.

- The JWT will contain all the necessary information of the users along with the expiration period. The JWT will be 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 will retrieve their JWTs and use them on their behalf, in the headers of the HTTP requests to the back-end. By decoding the JWTs, the front-end will have 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, will decide 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 about this decision are that a) the front-end users will not handle the JWTs by themselves (front-end will do), b) they don't have access to it and c) they should have longer session (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).

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 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 will be a unique username. The table below presents the endpoints related to Users as they are in the first 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 a username	GET	{HOST}/accounts/username/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/{username}
Update user's information	PUT	{HOST}/accounts/users/information/{username}
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/{username}
Update user's email	PUT	{HOST}/accounts/users/email/{username}
Change user's profile picture	PUT	{HOST}/accounts/users/image
Remove user's profile picture	DELETE	{HOST}/accounts/users/image/{username}
Get user's statistics	GET	{HOST}/accounts/users/statistics/{username}
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.
- 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 and the username must be unique and available 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> { "username": "...", "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” which 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 '{ "username": "...", "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 username
  "account": {
    "password": "...", // user's password (hashed)
    "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 a username	
Endpoint:	{HOST}/accounts/username/availability	
HTTP Method:	GET	
Description:	This endpoint is used in order to check the availability of a username during the registration of the users. A GET request should be made and the key “x-username” must be included in the headers of the request.	
Body Data:	None	
Headers:	<u>Key</u>	<u>Value</u>
	x-username	The username 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:		
<code>curl --request GET '{HOST}/accounts/username/availability' --header 'x-username: <value>'</code>		

TABLE 3 - CHECK AVAILABILITY OF A USERNAME 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. It should be noted that users can log in either with their email or with their username.</p> <pre>{ "username": "...", "email": "...", "password": "..." }</pre> <p>A successful response will return the next JSON schema that contains the JWT in the key "token": {"_status": "successful", "token": "<JWT>"}</p> <p>The headers of the request may contain the key "x-more-time" which 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 '{ "username": "...", "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:	None	
Successful Response:	JSON Object with a successful message and user's JWT.	
The following is an example of the request in cURL:		
<code>curl --request GET '{HOST}/accounts/users/verification/{vc}' --header 'x-more-time: <API_KEY>'</code>		

TABLE 5 - VERIFY USERS INTERFACE

Title:	Resend verification code to users	
Endpoint:	{HOST}/accounts/users/verification/resend	
HTTP Method:	POST	
Description:	<p>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.</p> <p>This request requires user's JWT in the headers of the request, under the key "x-access-token", in order to authenticate the user.</p>	
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/{username}	
HTTP Method:	GET	
Description:	<p>This endpoint is used in order to retrieve information about a user. A GET request should be made and the user's {username} 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.</p> <p>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".</p> <p>Below are examples of retrieved users' information, one by an administrator/account owner (1) and one by a user that retrieves another</p>	

	user's information (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>
	username	The username 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:		
<pre>curl --request GET '{HOST}/accounts/users/information/{username}' \ --header 'x-access-token: <JWT>'</pre>		

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
  }, "username": "..."
}}
```

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"}, "username": "..."
}}
```

Title: Update user's information							
Endpoint:	{HOST}/accounts/users/information/{username}						
HTTP Method:	PUT						
Description:	<p>This endpoint handles requests for updating users' information. A PUT request should be made and the next JSON schema (<u>it is flexible and thus may contain fewer fields - but no new fields</u>), 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" which 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:	<table border="1"> <thead> <tr> <th>Key</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>x-access-token</td> <td>Requester's JWT</td> </tr> <tr> <td>x-more-time</td> <td><u>[Restricted and available only for the front-end which use an API key]</u> Front-end's API key</td> </tr> </tbody> </table>	Key	Value	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
Key	Value						
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:	<table border="1"> <thead> <tr> <th>Parameter</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>username</td> <td>The username of the user whose information will be updated.</td> </tr> </tbody> </table>	Parameter	Value	username	The username of the user whose information will be updated.		
Parameter	Value						
username	The username 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:	<p>A successful response will return the next JSON Object that contains a new JWT in the key "token":</p> <pre> {"_status": "successful", "message": "The information of the user '{username}' has been updated.", "token": "<JWT>"} </pre>						
<p>The following is an example of the request in cURL:</p> <pre> curl --request PUT '{HOST}/accounts/users/information/{username}' \ --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>

	<code>{ "old_password": "...", "new_password": "...", "confirm_new_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:	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 passwords 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 is sufficient/required.</p> <pre>{"username": "...", "email": "..."} </pre> <p>Another important note is that this endpoint is available only through the mechanisms of the front-end which 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).</p> <p>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 '{"username": "...", "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. The users will open the password reset link that they received in their email, which redirects to a front-end's form from which the users are able to set their new password. After the submission of the form, the front-end sends a request to the current interface in order finish the process.</p> <p>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/{username}	
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 {username} 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 Marketplace. If the action is made by an administrator, the field "password" in the body should be administrator's password.</p> <p><u>An important note is that the deletion of an account has as result the deletion of all user's data, offered descriptions 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>

	username	The username 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/{username}' \ --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/{username}	
HTTP Method:	PUT	
Description:	<p>This endpoint is used in order 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> <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 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.</p> <p>Also, the headers of the request may contain the key "x-more-time" which 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>
	username	The username 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/{username}' \ --header 'x-access-token: <JWT>' --header 'Content-Type: application/json' \ --data-raw '{ "new_email": "..."}'</pre>		

TABLE 13 - CHANGE USER'S EMAIL INTERFACE

Title:		Change user's profile picture
Endpoint:	{HOST}/accounts/users/image	
HTTP Method:	PUT	
Description:	<p>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.</p> <p>Also, the headers of the request may contain the key "x-more-time" which 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:	Data Type:	Form Data
	Key	Value
	asset	Binary data / Path to image
Headers:	Key	Value
	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
	x-mimetype	Image's mimetype (Only JPEG and PNG images are allowed)
URL Parameters:	None	
Query Parameters:	None	
Restrictions / Special Features:	Available only to accounts' owners.	
Successful Response:	<p>A successful response will return the next JSON Object that contains a new JWT in the key "token":</p> <pre>{ "_status": "successful", "message": "The profile image of the user '{username}' has been changed.", "token": "<JWT>" }</pre>	
<p>The following is an example of the request in cURL:</p> <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 14 - CHANGE USER'S PROFILE PICTURE INTERFACE

Title:		Remove user's profile picture
Endpoint:	{HOST}/accounts/users/image/{username}	
HTTP Method:	DELETE	
Description:	<p>This endpoint is used in order to delete users' profile images. The {username} 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 which 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.</p> <p>Also, the headers of the request may contain the key "x-more-time" which 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:	Key	Value

	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>
	username	The username of the 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 '{username}' has been removed.", "token": "<JWT>" }</pre>	
The following is an example of the request in cURL:		
<pre>curl --request DELETE '{HOST}/accounts/users/image/{username}' \ --header 'x-access-token: <JWT>' --header 'x-more-time: <API_KEY>'</pre>		

TABLE 15 - REMOVE USER'S PROFILE PICTURE INTERFACE

Title:	Get user's statistics	
Endpoint:	{HOST}/accounts/users/statistics/{username}	
HTTP Method:	GET	
Description:	This endpoint is used in order to get some statistics about a user whose {username} is in the URL of the GET request. It is used in users' profiles where their contribution with offerings to the 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>
	username	The username 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_descriptions": 0, "approved_descriptions": 0, "assets_uploaded": 0, "total_downloads": 0, "total_views": 0, "total_reviews": 0, "average_rating": 0 }}</pre>	
The following is an example of the request in cURL:		
<pre>curl --request GET '{HOST}/accounts/users/statistics/{username}' --header 'x-access-token: <JWT>'</pre>		

TABLE 16 - GET USER'S STATISTICS INTERFACE

Title:	Get user's account data	
Endpoint:	{HOST}/accounts/users/data	
HTTP Method:	GET	
Description:	<p>This endpoint, which is available only to accounts' owners, returns all personalized data of the requester. More specifically, it returns users' information, uploaded descriptions, reviews to descriptions and other collected statistics.</p> <p>The endpoint is restricted and thus, the JWT of a requester must be included in the headers of the request</p>	
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 17 - GET USER'S ACCOUNT DATA INTERFACE

2.2.1.2 Interfaces related to Descriptions

This group of APIs offers functionalities intended for the management of the descriptions. They support all CRUD operations as well as the search functionality. Special emphasis was placed on the APIs for the descriptions' retrieval, extending them so as to get the latest descriptions or even random descriptions either from a specific collection (database collection) or from all the collections at once, using the keyword "all". The collections of the database as well as the Marketplace's offered types of assets, vary. The current list of the collections can be found at the end of Table 18, which presents the endpoints related to the Descriptions as they were stated in the first version of the back-end.

Action	HTTP Method	Endpoint
Get descriptions' collections	GET	{HOST}/descriptions
Get a list with all descriptions	GET	{HOST}/descriptions/all
Get a list with all descriptions from a specific collection	GET	{HOST}/descriptions/{collection}
Get a specific description (using keyword "all")	GET	{HOST}/descriptions/all/{description_id}
Get a specific description (using description's "collection")	GET	{HOST}/descriptions/{collection}/{description_id}
Get latest descriptions from all collections	GET	{HOST}/descriptions/all/latest
Get latest descriptions from a collection	GET	{HOST}/descriptions/{collection}/latest
Get random descriptions from all collections	GET	{HOST}/descriptions/all/random
Get random descriptions from a specific collection	GET	{HOST}/descriptions/{collection}/random
Get a list with all descriptions provided by a specific user (using keyword "all")	GET	{HOST}/descriptions/provider/{username}/all

Get a list with all descriptions provided by a specific user and under a specific collection (using a “collection” value)	GET	{HOST}/descriptions/provider/{username}/{collection}
Get descriptions’ statistics (useful for front-end’s homepage)	GET	{HOST}/descriptions/statistics
Upload / Create a new description with random ID	POST	{HOST}/descriptions/{collection}
Upload / Create a new description with given ID	POST	{HOST}/descriptions/{collection}/{given_id}
Update a specific description (using keyword “all”)	PUT	{HOST}/descriptions/all/{description_id}
Update a specific description (using description’s “collection”)	PUT	{HOST}/descriptions/{collection}/{description_id}
Delete a specific description (using keyword “all”)	DELETE	{HOST}/descriptions/all/{description_id}
Delete a specific description (using description’s “collection”)	DELETE	{HOST}/descriptions/{collection}/{description_id}
Delete all descriptions (administrators’ action)	DELETE	{HOST}/descriptions/all/all
Delete all descriptions from a specific collection (administrators’ action)	DELETE	{HOST}/descriptions/{collection}/all
Make a review for a description	POST	{HOST}/descriptions/review/{description_id}
Update an existing review for a description	PUT	{HOST}/descriptions/review/{description_id}
Delete a review for a description	DELETE	{HOST}/descriptions/review/{description_id}
Get a list with the reviews made by a specific user	GET	{HOST}/descriptions/review/{username}
Get a list with all descriptions that need permission (administrators’ action)	GET	{HOST}/descriptions/permit/all
Get a list with all descriptions from a specific collection that need permission (administrators’ action)	GET	{HOST}/descriptions/permit/{collection}
Approve or reject a specific description that need permission, using keyword “all” (administrators’ action)	POST	{HOST}/descriptions/permit/all/{description_id}
Approve or reject a specific description that need permission, using description’s “collection” (administrators’ action)	POST	{HOST}/descriptions/permit/{collection}/{description_id}
Approve or reject all descriptions that need permission, using keyword “all” (administrators’ action)	POST	{HOST}/descriptions/permit/all/all
Approve or reject all descriptions that need permission under a specific collection, using a “collection” value (administrators’ action)	POST	{HOST}/descriptions/permit/{collection}/all

TABLE 18 - BACK-END’S INTERFACES RELATED TO DESCRIPTIONS

- *{HOST}* refers to the hosting server: the domain name and the port running the back-end.
- *{description_id}* refers to the ID of a specific description.
- *{given_id}* is used in upload description action, providing new description's ID.
- As a *{collection}* can be one of the following values derived from the current types of offered assets:

*{"algorithms", "tools", "policies", "datasets",
"webinars", "tutorials", "documents", "externals",
"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 descriptions' collections
Endpoint:	{HOST}/descriptions
HTTP Method:	GET
Description:	This endpoint returns a list with the sub-routes of the "description" endpoint. More specifically, returns the values of the {collection} parameter which are also 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:	
<pre>curl --request GET '{HOST}/descriptions'</pre>	

TABLE 19 - GET DESCRIPTIONS' COLLECTIONS INTERFACE

Title:	Get a list with all descriptions
Endpoint:	{HOST}/descriptions/all
HTTP Method:	GET
Description:	A request to this endpoint will result in the retrieval of the stored descriptions from all collections. It uses the keyword "all" instead of a specific collection and that makes the platform to retrieve descriptions from all collections at once. The descriptions that return from this request are in a short schema (short description) and that means that the retrieved information is limited. An example of a description in short schema is the following JSON schema: <pre>{ "collection": "algorithms", "id": "algorithms_v1LZWaoQN1Fe ", "info": { "fieldOfUse": ["information"], "owner": "Vasilis Koukos", "short_desc": "This is an example", "type": "algorithms", "subtype": "-", "title": "Example title." }, "main_image": "default_image_assets", "metadata": { "provider": "vkoukos", "reviews": { "average_rating": 4.2, "no_reviews": 14 } } }</pre>

	<pre>"updateDate": "...", "uploadDate": "...", "views": 35 } }</pre>	
	<p>This endpoint can get query parameters in order to search for descriptions 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. In addition to these, this endpoint offers some standard query parameters that are described below (Query Parameters).</p>	
Body Data:	None	
Headers:	None	
URL Parameters:	None	
Query Parameters:	<u>Key</u>	<u>Value</u>
	sortBy	<p>[Optional] Sorts the descriptions 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.</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 descriptions from all collections). If the query parameter “itemsPerPage” is used, then the results contain the total number of the pages.	

<p>The following is an example of the request in cURL:</p> <pre>+ curl --request GET '{HOST}/descriptions/all' + curl --request GET '{HOST}/descriptions/all?sortBy={value}' + curl --request GET '{HOST}/descriptions/all?itemsPerPage={value}' + curl --request GET '{HOST}/descriptions/all?itemsPerPage={value}&page={value}' + curl --request GET '{HOST}/descriptions/all?sortBy={value}&itemsPerPage={value}' + curl --request GET '{HOST}/descriptions/all?sortBy={value}&itemsPerPage={value}&page={value}'</pre> <p><u>Example of retrieving 10 most viewed descriptions:</u></p> <pre>+ curl --request GET '{HOST}/descriptions/all?sortBy=views-desc&itemsPerPage=10&page=1'</pre>
--

TABLE 20 - GET A LIST WITH ALL DESCRIPTIONS INTERFACE

Title:	Get a list with all descriptions from a specific collection					
Endpoint:	{HOST}/descriptions/{collection}					
HTTP Method:	GET					
Description:	This request is similar to the above request. The only difference between these two actions is that this request retrieves descriptions from a single and 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: {"algorithms", "tools", "policies", "datasets", "webinars", "tutorials", "documents", "externals", "other"}</td> </tr> </tbody> </table>	Parameter	Value	collection	Valid values: {"algorithms", "tools", "policies", "datasets", "webinars", "tutorials", "documents", "externals", "other"}	
Parameter	Value					
collection	Valid values: {"algorithms", "tools", "policies", "datasets", "webinars", "tutorials", "documents", "externals", "other"}					
Query Parameters:	As in the above request.					
Restrictions / Special Features:	None					
Successful Response:	A JSON Object with the results (all descriptions in a specific collection).					
<p>The following is an example of the request in cURL:</p> <pre>+ curl --request GET '{HOST}/descriptions/{collection}' + curl --request GET \ '{HOST}/descriptions/{collection}?sortBy={value}&itemsPerPage={value}&page={value}'</pre>						

TABLE 21 - GET A LIST WITH ALL DESCRIPTIONS FROM A SPECIFIC COLLECTION INTERFACE

Title:	Get a specific description (using keyword “all”)
Endpoint:	{HOST}/descriptions/all/{description_id}
HTTP Method:	GET
Description:	<p>With this request, the users are able to retrieve a specific description. The retrieval of a specific description is possible using its unique identification code (ID), known when uploading it. Also, the retrieval of a specific description can be done using both keyword “all” and the name of the collection that the description has been stored (next interface). This is feasible because the back-end ensures that the IDs are unique regardless of in which collection a description has been stored.</p> <p>Moreover, the retrieval of a specific description requires a JWT in order to be retrieved in its “full schema”. If requester’s JWT is missing, then the endpoint returns the short schema of the description. Example of a full schema is in the endpoint that handles the uploading of a description.</p>

	This endpoint, except for the full schema, also returns the reviews of the specified description.	
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 description] Requester's JWT
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	description_id	The ID of the description 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 description in the results.	
The following is an example of the request in cURL:		
<pre>+ curl --request GET '{HOST}/descriptions/all/{description_id}' + curl --request GET '{HOST}/descriptions/all/{description_id}' --header 'x-access-token: <JWT>'</pre>		

TABLE 22 - GET A SPECIFIC DESCRIPTION (USING KEYWORD "ALL") INTERFACE

Title:	Get a specific description (using description's "collection")	
Endpoint:	{HOST}/descriptions/{collection}/{description_id}	
HTTP Method:	GET	
Description:	This request is similar to the above request, with the difference that it uses description's collection for the retrieval of the description (instead of using keyword "all"). The value of the {collection} must be the collection in which the specific description has been stored. More information about the endpoint can be found on the above endpoint.	
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 description] Requester's JWT
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	collection	Valid values: { "algorithms", "tools", "policies", "datasets", "webinars", "tutorials", "documents", "externals", "other" }
	description_id	The ID of the description 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 description in the results.	
The following is an example of the request in cURL:		
<pre>+ curl --request GET '{HOST}/descriptions/{collection}/{description_id}' + curl --request GET '{HOST}/descriptions/{collection}/{description_id}' \ --header 'x-access-token: <JWT>'</pre>		

TABLE 23 - GET A SPECIFIC DESCRIPTION (USING DESCRIPTION'S "COLLECTION") INTERFACE

Title:	Get latest descriptions from all collections	
Endpoint:	{HOST}/descriptions/all/latest	
HTTP Method:	GET	
Description:	<p>This request is used to retrieve the most recent uploaded descriptions 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 descriptions from all collections. The value of K can be specified through the query parameter “max” (the default value is 20). The descriptions are returned in their short schema.</p> <p>This endpoint can get query parameters in order to search for descriptions 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.</p> <p>Finally, the endpoint “Get a list with all descriptions” 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 descriptions from all collections).	
The following is an example of the request in cURL:		
<pre>+ curl --request GET '{HOST}/descriptions/all/latest'</pre> <pre>+ curl --request GET '{HOST}/descriptions/all/latest?max=5'</pre>		
Example of similar response by the endpoint “Get a list with all descriptions”:		
<pre>+ curl --request GET '{HOST}/descriptions/all?sortBy=newest&itemsPerPage=20&page=1'</pre>		

TABLE 24 - GET LATEST DESCRIPTIONS FROM ALL COLLECTIONS INTERFACE

Title:	Get latest descriptions from a specific collection	
Endpoint:	{HOST}/descriptions/{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” and this results to return sorted the K most recent descriptions of the provided collection. The value of K can be specified through query parameter “max” (the default value is 20). The descriptions are returned in their short schema.</p> <p>This endpoint can get query parameters in order to search for descriptions 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.</p>	

	Finally, the endpoint “Get a list with all descriptions from a specific collection” can return the same results as the current, if the example at the end will be followed.	
Body Data:	None	
Headers:	None	
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	collection	Valid values: {"algorithms", "tools", "policies", "datasets", "webinars", "tutorials", "documents", "externals", "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 descriptions of a collection).	
The following is an example of the request in cURL:		
<pre>+ curl --request GET '{HOST}/descriptions/{collection}/latest'</pre> <pre>+ curl --request GET '{HOST}/descriptions/{collection}/latest?max=5'</pre>		
<u>Example of similar response by the endpoint “Get a list with all descriptions from a specific collection”:</u>		
<pre>+ curl --request GET '{HOST}/descriptions/{collection}?sortBy=newest&itemsPerPage=20&page=1'</pre>		

TABLE 25 - GET LATEST DESCRIPTIONS FROM A SPECIFIC COLLECTION INTERFACE

Title:	Get random descriptions from all collections	
Endpoint:	{HOST}/descriptions/all/random	
HTTP Method:	GET	
Description:	This endpoint returns a number of random descriptions from all collections (uses keyword “all”). It is useful in order to suggest and promote different descriptions each time. It is also used in the home page of the Data Marketplace, where random descriptions are displayed. Through the query parameter “max” can return K descriptions, where K can be specified by the users (the default value is 4). The descriptions 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 descriptions from all collections).	
The following is an example of the request in cURL:		
<pre>+ curl --request GET '{HOST}/descriptions/all/random'</pre> <pre>+ curl --request GET '{HOST}/descriptions/all/random?max=5'</pre>		

TABLE 26 - GET RANDOM DESCRIPTIONS FROM ALL COLLECTIONS INTERFACE

Title:	Get random descriptions from a specific collection	
Endpoint:	{HOST}/descriptions/{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 descriptions of the provided specific collection. The value of K can be specified through query parameter “max” (the default value is 4). The descriptions are returned in their short schema.	
Body Data:	None	
Headers:	None	
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	collection	Valid values: {"algorithms", "tools", "policies", "datasets", "webinars", "tutorials", "documents", "externals", "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 descriptions of a collection).	
The following is an example of the request in cURL:		
<pre>+ curl --request GET '{HOST}/descriptions/{collection}/random'</pre> <pre>+ curl --request GET '{HOST}/descriptions/{collection}/random?max=5'</pre>		

TABLE 27 - GET RANDOM DESCRIPTIONS FROM A SPECIFIC COLLECTION INTERFACE

Title:	Get a list with all descriptions provided by a specific user (using keyword “all”)	
Endpoint:	{HOST}/descriptions/provider/{username}/all	
HTTP Method:	GET	
Description:	<p>This request returns all the descriptions that have been provided by the user whose {username} is part of the request's URL. It uses the keyword “all” instead of a specific collection and that makes the platform to retrieve its provided descriptions from all collections at once. The descriptions are returned in their short schema.</p> <p>Moreover, 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 in order to retrieve their uploaded descriptions, except for the retrieval of the approved descriptions, they also retrieve “pending” descriptions (e.g. the descriptions that they uploaded and need administrators' approval).</p> <p>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>

	username	The username of the user whose offered descriptions will be retrieved.
Query Parameters:	<u>Key</u>	<u>Value</u>
	sortBy	[Optional] Sorts the descriptions by a field – the default 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. “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.
	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 descriptions 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:		
<pre>+ curl --request GET '{HOST}/descriptions/provider/{username}/all' + curl --request GET '{HOST}/descriptions/provider/{username}/all?sortBy={value}' + curl --request GET '{HOST}/descriptions/provider/{username}/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}'</pre>		

TABLE 28 - GET A LIST WITH ALL DESCRIPTIONS PROVIDED BY A SPECIFIC USER (USING KEYWORD “ALL”) INTERFACE

Title:	Get a list with all descriptions provided by a specific user and under a specific collection (using a “collection” value)	
Endpoint:	{HOST}/descriptions/provider/{username}/{collection}	
HTTP Method:	GET	
Description:	This request is similar to the above request. The only difference between these two actions is that this request retrieves all descriptions 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>
	username	The username of the user whose offered descriptions will be retrieved.
	collection	Valid values: {"algorithms", "tools", "policies", "datasets", "webinars", "tutorials", "documents", "externals", "other"}
Query Parameters:	As in the above request.	
Restrictions / Special Features:	None	
Successful Response:	A JSON Object with the results (all descriptions 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}/descriptions/provider/{username}/{collection}' + curl --request GET '{HOST}/descriptions/provider/{username}/{collection}?sortBy={value}' + curl --request GET '{HOST}/descriptions/provider/{username}/{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 29 - GET A LIST WITH ALL DESCRIPTIONS PROVIDED BY A SPECIFIC USER AND UNDER A SPECIFIC COLLECTION (USING A “COLLECTION” VALUE) INTERFACE

Title:	Get descriptions’ statistics (useful for front-end’s homepage)	
Endpoint:	{HOST}/descriptions/statistics	
HTTP Method:	GET	
Description:	<p>A request to this endpoint has as result the retrieval of some statistics on stored (and approved) descriptions (and collections). Briefly, the response contains:</p> <ul style="list-style-type: none"> • the total number of descriptions, • the number of descriptions per collection, and • top 3 collections with the most descriptions 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:	A JSON Object with the descriptions' statistics. Example of a response: <pre>{ "_status": "successful", "results": { "all": { "algorithms": 15, "datasets": 22, "documents": 5, "externals": 2, "other": 0, "policies": 20, "tools": 6, "tutorials": 4, "webinars": 3 }, "sum": 77, "top": [{ "collection": "datasets", "descriptions": 22, "percentage": 0.28 }, { "collection": "policies", "descriptions": 20, "percentage": 0.26 }, { "collection": "algorithms", "descriptions": 15, "percentage": 0.19 }] } }</pre>
The following is an example of the request in cURL:	
<code>curl --request GET '{HOST}/descriptions/statistics'</code>	

TABLE 30 - GET DESCRIPTIONS' STATISTICS (USEFUL FOR FRONT-END'S HOMEPAGE) INTERFACE

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

TABLE 31 - UPLOAD / CREATE A NEW DESCRIPTION WITH RANDOM ID INTERFACE

Below are some examples of the stored descriptions' schema:

Example 1 - Newly uploaded description with no assets

```
{
  "id": "others_P8fYOAX67HkK-8fpe1TlB-KuR4-Zsck",
  "info": {"comments": "Private comment.", "contact": "Vasilis Koukos, email",
    "description": "This is an example of description.",
    "fieldOfUse": [ "testing", "documentation" ], "owner": "UPRC",
    "subtype": "-", "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 description>",
    "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 description - increases when updating
    "views": 8},
  "assets": {"files": [], //list with the uploaded files for this description
    "images": [], //list with the uploaded images for this description
    "links": [], //list with the external links added to this description
    "videos": [] //list with the uploaded videos for this description
  }
}
```

Example 2 – Description with uploaded file

```
{
  "id": "others_P8fYOAX67HkK-8fpelTlB-KuR4-Zsck",
  "assets": {
    "files": [{
      "approved": 0, //0 for pending / 1 for approved
      "downloads": 3, //number of downloads of the file
      "filename": "kmeans.py", "id": "80F7MjRTIxb-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": [], "links": [], "videos": []
  }
}
```

Example 3 – Retrieved description (full schema)

```
{
  "id": "others_P8fYOAX67HkK-8fpelTlB-KuR4-Zsck",
  "info": {"comments": "Private comment.", "contact": "Vasilis Koukos, email",
    "description": "This is an example of description.",
    "fieldOfUse": [ "testing", "documentation" ], "owner": "UPRC",
    "subtype": "-", "title": "Example.", "type": "others"},
  "main_image": "default_image_assets",
  "metadata": {"approved": 1, last_updated_by": "vkoukos", "md5": "<md5 hash of the description>",
    "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": [{
      "approved": 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": [], "links": [], "videos": []
  },
  "reviews": [
    {
      "comment": "Very good!", "description_version": 1, "rating": 4,
      "review_version": 1, "updated_review_date": "2021-10-14 16:02:05.484Z",
      "username": "user_1"
    }, {
      "comment": "Needs improvement...", "description_version": 1, "rating": 2,
      "review_version": 2, "updated_review_date": "2021-10-15 11:06:03.334Z",
      "username": "user_2"
    }, {
      "comment": "Not bad.", "description_version": 1, "rating": 3,
      "review_version": 1, "updated_review_date": "2021-10-15 13:30:00.209Z",
      "username": "user_3"
    }, {
      "comment": "Thank you for this!!", "description_version": 1, "rating": 5,
      "review_version": 1, "updated_review_date": "2021-10-18 10:12:49.956Z",
      "username": "user_4"
    }, {
      "comment": "Good idea but does not perform well for big data.",
      "description_version": 1, "rating": 2, "review_version": 1,
      "updated_review_date": "2021-10-18 14:53:13.410Z", "username": "user_5"
    }
  ]
}
```

Title:	Upload / Create a new description with given ID	
Endpoint:	{HOST}/descriptions/{collection}/{given_id}	
HTTP Method:	POST	
Description:	This endpoint is similar to above. The only difference is that through the current endpoint, the users are able to specify the ID of the new description, 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 descriptions 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 username of the provider in case that the description is uploaded by an administrator and not by the provider.
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	collection	Valid values: { "algorithms", "tools", "policies", "datasets", "webinars", "tutorials", "documents", "externals", "other" }
	given_id	The ID to be given to the new description.
Query Parameters:	None	
Restrictions / Special Features:	Available only to administrators. The administrators are able to upload a description on behalf of other users.	
Successful Response:	JSON Object with the new description's ID in its content.	
The following is an example of the request in cURL:		
<pre>curl --request POST '{HOST}/descriptions/{collection}/{given_id}' \ --header 'x-access-token: <JWT>' --header 'Content-Type: application/json' \ --data-raw '{ "title": "<title of the asset>", "description": "<description of the provided asset>", "type": "<type of the asset (same as collection's value)>", "subtype": "<the subtype of the asset, if any, otherwise empty string or a dash (-)>", "comments": "<a private field that is shown only when the full schema is retrieved (only by authorized users) - useful for private comments>", "fieldOfUse": ["<field 1>", ...], "owner": "<organization / author / etc.>", }'</pre>		
Example of uploading a description through binary data/file:		
<pre>curl --request POST '{HOST}/descriptions/{collection}/{given_id}' \ --header 'x-access-token: <JWT>' --header 'Content-Type: application/json' \ --data-binary '@<path_to_json_file>'</pre>		

TABLE 32 - UPLOAD / CREATE A NEW DESCRIPTION WITH GIVEN ID INTERFACE

Title:	Update a specific description (using keyword “all”)	
Endpoint:	{HOST}/descriptions/all/{description_id}	
HTTP Method:	PUT	
Description:	<p>With this endpoint, the providers of the descriptions are able to update the contents of the descriptions. It requires the ID of the description to be at the end of the endpoint and the body of the request should contain the next JSON schema (the same schema with the uploading action) as raw data.</p> <pre> { "title": "<title of the asset>", "description ": "<description of the provided asset>", "type": "<type of the asset (same as collection’s value)>", "subtype": "<the subtype of the asset, if any, otherwise empty string or a dash (-)>", "comments": "<a private field that is shown only when the full schema is retrieved (only by authorized users) - useful for provider’s private comments>", "fieldOfUse": ["<field 1>", ...], "owner ": "<organization / author / etc.>", } </pre> <p>It should be noted that this endpoint uses the keyword “all” (the descriptions are already stored in the Marketplace, thus the platform knows the collections in which have been stored). Moreover, this action is only available to the providers/creators of the descriptions and to administrators who are able to update any description. Thus, the JWT of a requester should be included in the headers of the request. An important note is that all updated descriptions get locked and must be approved again by an administrator to get available again to other users.</p>	
Body Data:	Raw (JSON) Data - as the above schema (Content-Type: application/json). The descriptions 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>
	description_id	The ID of the description that will be updated.
Query Parameters:	None	
Restrictions / Special Features:	Available only for the providers/creators of the descriptions and for the administrators who can update any description.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<pre> curl --request PUT '{HOST}/descriptions/all/{description_id}' \ --header 'x-access-token: <JWT>' --header 'Content-Type: application/json' \ --data-raw '{ "title": "<title of the asset>", "description ": "<description of the provided asset>", "type": "<type of the asset (same as collection’s value)>", "subtype": "<the subtype of the asset, if any, otherwise empty string or a dash (-)>", "comments": "<a private field that is shown only when the full schema is retrieved (only by authorized users) - useful for private comments>", "fieldOfUse": ["<field 1>", ...], "owner ": "<organization / author / etc.>", }' </pre>		
Example of uploading a description through binary data/file:		
<pre> curl --request POST '{HOST}/descriptions/all/{description_id}' \ --header 'x-access-token: <JWT>' --header 'Content-Type: application/json' \ --data-binary '@<path_to_json_file>' </pre>		

TABLE 33 - UPDATE A SPECIFIC DESCRIPTION (USING KEYWORD “ALL”) INTERFACE

Title:	Update a specific description (using description's "collection")	
Endpoint:	{HOST}/descriptions/{collection}/{description_id}	
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 description has been stored during its creation. The endpoint is restricted and available only to descriptions' providers/creators and to administrators who can update any description. 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 descriptions 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: { "algorithms", "tools", "policies", "datasets", "webinars", "tutorials", "documents", "externals", "other" }
	description_id	The ID of the description that will be updated.
Query Parameters:	None	
Restrictions / Special Features:	Available only for the providers/creators of the descriptions and for the administrators who can update any description.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<pre>curl --request PUT '{HOST}/descriptions/{collection}/{description_id}' \ --header 'x-access-token: <JWT>' --header 'Content-Type: application/json' \ --data-raw '{ "title": "<title of the asset>", "description": "<description of the provided asset>", "type": "<type of the asset (same as collection's value)>", "subtype": "<the subtype of the asset, if any, otherwise empty string or a dash (-)>", "comments": "<a private field that is shown only when the full schema is retrieved (only by authorized users) - useful for private comments>", "fieldOfUse": ["<field 1>", ...], "owner": "<organization / author / etc.>", }'</pre>		
Example of uploading a description through binary data/file:		
<pre>curl --request POST '{HOST}/descriptions/{collection}/{description_id}' \ --header 'x-access-token: <JWT>' --header 'Content-Type: application/json' \ --data-binary '@<path_to_json_file>'</pre>		

TABLE 34 - UPDATE A SPECIFIC DESCRIPTION (USING DESCRIPTION'S "COLLECTION") INTERFACE

Title:	Delete a specific description (using keyword “all”)	
Endpoint:	{HOST}/descriptions/all/{description_id}	
HTTP Method:	DELETE	
Description:	<p>A DELETE request to this endpoint has as a result the deletion of a specific description, using its ID. The endpoint is restricted and available only to descriptions’ providers/creators and to administrators who can delete any description. Thus, the JWT of a requester must be included in the headers of the request.</p> <p>It should be noted that this endpoint uses the keyword “all” instead of description’s collection (the descriptions are already stored in the Marketplace, thus the platform knows the collections in which have been stored).</p> <p>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>
	description_id	The ID of the description that will be deleted.
Query Parameters:	None	
Restrictions / Special Features:	Available only for the providers/creators of the descriptions and for the administrators who can delete any description.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<pre>curl --request DELETE '{HOST}/descriptions/all/{description_id}' \ --header 'x-access-token: <JWT>' --header 'Content-Type: application/json' \ --data-raw '{ "password": "..." }'</pre>		

TABLE 35 - DELETE A SPECIFIC DESCRIPTION (USING KEYWORD “ALL”) INTERFACE

Title:	Delete a specific description (using description’s “collection”)	
Endpoint:	{HOST}/descriptions/{collection}/{description_id}	
HTTP Method:	DELETE	
Description:	<p>This request is similar to the previous. The only difference is that instead of using keyword “all” it uses the collection in which the description has been stored during its creation. The endpoint is restricted and available only to descriptions’ providers/creators and to administrators who can delete any description. 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.</p>	
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:

		{"algorithms", "tools", "policies", "datasets", "webinars", "tutorials", "documents", "externals", "other"}
	description_id	The ID of the description that will be deleted.
Query Parameters:	None	
Restrictions / Special Features:	Available only for the providers/creators of the descriptions and for the administrators who can delete any description.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<pre>curl --request DELETE '{HOST}/descriptions/{collection}/{description_id}' \ --header 'x-access-token: <JWT>' --header 'Content-Type: application/json' \ --data-raw '{ "password": "..."}'</pre>		

TABLE 36 - DELETE A SPECIFIC DESCRIPTION (USING DESCRIPTION'S "COLLECTION") INTERFACE

Title:	Delete all descriptions (administrators' action)	
Endpoint:	{HOST}/descriptions/all/all	
HTTP Method:	DELETE	
Description:	This endpoint is available only to the administrators who through it, can delete all descriptions from all 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 Marketplace.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<pre>curl --request DELETE '{HOST}/descriptions/all/all' --header 'x-access-token: <JWT>' \ --header 'Content-Type: application/json' --data-raw '{ "password": "..."}'</pre>		

TABLE 37 - DELETE ALL DESCRIPTIONS INTERFACE

Title:	Delete all descriptions from a specific collection (administrators' action)	
Endpoint:	{HOST}/descriptions/{collection}/all	
HTTP Method:	DELETE	
Description:	This endpoint is similar to the above. It is available only to the administrators who through it, can delete all descriptions 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): <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:	<u>Parameter</u>	<u>Value</u>
	collection	Valid values: {"algorithms", "tools", "policies", "datasets", "webinars", "tutorials", "documents", "externals", "other"}
Query Parameters:	None	
Restrictions / Special Features:	Currently, it is available only to the "superuser" (master admin) of the Marketplace.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<pre>curl --request DELETE '{HOST}/descriptions/{collection}/all' --header 'x-access-token: <JWT>' \ --header 'Content-Type: application/json' --data-raw '{ "password": "..."}'</pre>		

TABLE 38 - DELETE ALL DESCRIPTIONS FROM A SPECIFIC COLLECTION INTERFACE

Title:	Make a review for a description	
Endpoint:	{HOST}/descriptions/review/{description_id}	
HTTP Method:	POST	
Description:	<p>This endpoint is used in order to make a review for a specific description whose ID is 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.</p> <p>A review consists of a rating (integer value between 1 and 5) and a comment. The users are able to make a review for a specific description only once, but they can update it through the next endpoint. Moreover, the providers/creators of a description are not able to make a review for their descriptions.</p> <p>The next JSON schema should be in the body of the request, as raw data: {"rating": <value>, "comment": "..."} After the successful submission of a review, the average rating of the reviewed description 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>
	description_id	The ID of the description 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 descriptions.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<pre>curl --request POST '{HOST}/descriptions/review/{description_id}' \ --header 'x-access-token: <JWT>' --header 'Content-Type: application/json' \ --data-raw '{"rating": <value>, "comment": "..."}'</pre>		

TABLE 39 - MAKE A REVIEW FOR A DESCRIPTION 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": "<description's title>", "did": "<description's ID>",
  "collection": "<description's collection>",
  "username": "<username of the user who made the review>",
  "initial_review_date": "<the date of the initial review>",
  "updated_review_date": "<date of the last review>",
  "description_version": <description's version when the review made>,
  "review_version": <version of the current review>
}
```

Title:	Update an existing review for a description	
Endpoint:	{HOST}/descriptions/review/{description_id}	
HTTP Method:	PUT	
Description:	<p>This endpoint is used in order to update a review that made for a specific description. The ID of the description 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 description.</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 description 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>
	description_id	The ID of the description 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 description.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<pre>curl --request PUT '{HOST}/descriptions/review/{description_id}' \ --header 'x-access-token: <JWT>' --header 'Content-Type: application/json' \ --data-raw '{"rating": <value>, "comment": "..."}'</pre>		

TABLE 40 - UPDATE AN EXISTING REVIEW FOR A DESCRIPTION INTERFACE

Title:	Delete a review for a description	
Endpoint:	{HOST}/descriptions/review/{description_id}	
HTTP Method:	DELETE	
Description:	<p>This endpoint is used in order to delete a review that made for a specific description. The ID of the description 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 description.</p>	

	<p>It should be noted that the administrators are able to delete reviews that made from other users, providing the username of a reviewer in the headers of the request.</p> <p>After the successful deletion of a review, the average rating of the description is recalculated.</p>	
Body Data:	None	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
	x-username	[Optional & only for administrators] The username of the user whose review on the specified description will be deleted. It is used by administrators in order to specify the reviewer.
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	description_id	The ID of the description 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 description. 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}/descriptions/review/{description_id}' \ --header 'x-access-token: <JWT>' + curl --request DELETE '{HOST}/descriptions/review/{description_id}' \ --header 'x-access-token: <JWT>' --header 'x-username: <value>'</pre>		

TABLE 41 - DELETE A REVIEW FOR A DESCRIPTION INTERFACE

Title:	Get a list with the reviews made by a specific user	
Endpoint:	{HOST}/descriptions/review/{username}	
HTTP Method:	GET	
Description:	<p>This request returns all the reviews made by a specific user whose {username} is part of the request's URL (the schema of the reviews can be found in the "Make a review for a description" interface).</p> <p>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>
	username	The username 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 description'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}/descriptions/review/{username}' --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 42 - GET A LIST WITH THE REVIEWS MADE BY A SPECIFIC USER INTERFACE

Title:	Get a list with all descriptions that need permission (administrators' action)	
Endpoint:	{HOST}/descriptions/permit/all	
HTTP Method:	GET	
Description:	<p>This endpoint returns the descriptions from all collections (since the keyword "all" is used) that need permission before they become available to the Marketplace's users. A description needs permission either when it is uploaded or after it has been updated.</p> <p>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).</p>	
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 descriptions by a field – the default 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 descriptions (from all collections) that need permission in its content.	
The following is an example of the request in cURL:		
<pre>+ curl --request GET '{HOST}/descriptions/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 43 - GET A LIST WITH ALL DESCRIPTIONS THAT NEED PERMISSION INTERFACE

Title:	Get a list with all descriptions from a specific collection that need permission (administrators' action)	
Endpoint:	{HOST}/descriptions/permit/{collection}	
HTTP Method:	GET	
Description:	This request is similar to the above request. The only difference between these two actions is that the current request retrieves the descriptions 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: {"algorithms", "tools", "policies", "datasets", "webinars", "tutorials", "documents", "externals", "other"}
Query Parameters:	As in the above request.	
Restrictions / Special Features:	Available only to the administrators.	
Successful Response:	JSON Object with the descriptions (from a specific collection) that need permission in its content.	
The following is an example of the request in cURL:		


```
+ curl --request GET '{HOST}/descriptions/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}' ...
```

TABLE 44 - GET A LIST WITH ALL DESCRIPTIONS FROM A SPECIFIC COLLECTION THAT NEED PERMISSION INTERFACE

Title:	Approve or reject a specific description that need permission, using keyword "all" (administrators' action)	
Endpoint:	{HOST}/descriptions/permit/all/{description_id}	
HTTP Method:	POST	
Description:	<p>This endpoint is used by administrators in order to approve or reject a specific description (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, it should be noted that this endpoint uses the keyword "all" and not the collection in which a specific description is stored, as the next endpoint does. An important parameter/key that must be included in the headers of the request is the "x-permission" key which should have as a value the text "approve" so the description to be approved, otherwise the text "disapprove" so to be rejected. A rejection/disapproval of a description has as a result the deletion of the description and all of its assets.</p>	
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>
	description_id	The ID of the description 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}/descriptions/permit/all/{description_id}' \ --header 'x-access-token: <JWT>' --header 'x-permission: <value>'</pre>		

TABLE 45 - APPROVE OR REJECT A DESCRIPTION THAT NEEDS PERMISSION, USING KEYWORD "ALL" INTERFACE

Title:	Approve or reject a specific description that need permission, using description's "collection" (administrators' action)	
Endpoint:	{HOST}/descriptions/permit/{collection}/{description_id}	
HTTP Method:	POST	
Description:	<p>This request is similar to the above request. The only difference between these two actions is that the current request uses the value of the {collection} in which a specific description is stored. For more details, refer to the above endpoint.</p>	
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: {"algorithms", "tools", "policies", "datasets", "webinars", "tutorials", "documents", "externals", "other"}
	description_id	The ID of the description 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}/descriptions/permit/{collection}/{description_id}' \ --header 'x-access-token: <JWT>' --header 'x-permission: <value>'</pre>		

TABLE 46 - APPROVE OR REJECT A DESCRIPTION THATS NEEDS PERMISSION, USING DESCRIPTION'S "COLLECTION" INTERFACE

Title:	Approve or reject all descriptions that need permission, using keyword "all" (administrators' action)	
Endpoint:	{HOST}/descriptions/permit/all/all	
HTTP Method:	POST	
Description:	<p>This endpoint is used by administrators in order to approve or reject all stored descriptions (from all 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.</p> <p>An important parameter/key that must be included in the headers of the request is the "x-permission" key which should have as a value the text "approve" so the descriptions to be approved, otherwise the text "disapprove" so to be rejected. A rejection/disapproval of the descriptions has as a result the deletion of the descriptions and all of their assets.</p>	
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 Marketplace.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<pre>curl --request POST '{HOST}/descriptions/permit/all/all ' \ --header 'x-access-token: <JWT>' --header 'x-permission: <value>'</pre>		

TABLE 47 - APPROVE OR REJECT ALL DESCRIPTIONS THAT NEED PERMISSION, USING KEYWORD "ALL" INTERFACE

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

TABLE 48 - APPROVE OR REJECT ALL DESCRIPTIONS THAT NEED PERMISSION UNDER A SPECIFIC COLLECTION, USING A “COLLECTION” VALUE INTERFACE

2.2.1.3 Search functionality on Descriptions

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 descriptions simultaneously, support some relative query filters. These filters enable the users of the Marketplace to search for assets, based on various parameters from the content of the stored descriptions.

More specifically, the interfaces of the back-end that return lists of 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 Marketplace and the descriptions, the keys added to the URLs as query parameters must be valid, in the sense that they exist as fields in the descriptions 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 descriptions” interface.

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

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

Moreover, the Python programming language which 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}/descriptions/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 / DESCRIPTION OF THE “EXAMPLE 1” IN THE INTERFACE “TABLE 30 - GET DESCRIPTIONS’ STATISTICS (USEFUL FOR FRONT-END’S HOMEPAGE) INTERFACE

”, the following search example request in cURL, returns the descriptions that in their title contain the value “machine learning” and their provider is the user with username “vkoukos”:

```
curl --request GET
'{HOST}/descriptions/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 which extend the keys of 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	<key>.lt=<value>

	This operator performs searching for a key with a value less than to the provided. It applies to both texts (strings) and numbers.	
<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 49 - BACK-END'S SEARCH OPERATORS

Below are some examples of the operators' use.

```

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

```

Furthermore, the back-end's search mechanism uses a ranking system for the results. More specifically, for each description in the results, 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 description 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** point 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 which are stored in the back-end. Table 50 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 description	POST	{HOST}/assets/{description_id}
Upload a new asset with given ID, linked to a specific description	POST	{HOST}/assets/{description_id}/{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 50 - 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 upload asset action, providing new asset’s ID.
- *{description_id}* refers to the ID of the description 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>
	x-access-token
	<u>Value</u> 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 51 - 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 52 - GET A SPECIFIC ASSET USING ITS ID INTERFACE

Title:	Upload a new asset with random ID, linked to a specific description	
Endpoint:	{HOST}/assets/{description_id}	
HTTP Method:	POST	
Description:	Through this endpoint, the users can upload their assets. It requires to add (at the end of the endpoint) the ID of the description with which is going to be linked. It is also necessary to add to the headers of the request a) the JWT of the provider and b) the asset's filename. The assets should 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 username of the provider in case that the asset is uploaded by an administrator and not by the provider.
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	description_id	The ID of the description with which the new asset is going to be linked.
Query Parameters:	None	
Restrictions / Special Features:	Available for the providers of the descriptions 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:		

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

TABLE 53 - UPLOAD A NEW ASSET WITH RANDOM ID INTERFACE

Title:	Upload a new asset with given ID, linked to a specific description	
Endpoint:	{HOST}/assets/{description_id}/{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}. Currently, this endpoint can be used only by the administrators.	
Body Data:	Data Type:	Form Data
	Key	Value
	asset	Binary data / Path to file
Headers:	Key	Value
	x-access-token	Requester's JWT
	x-filename	New asset's filename.
	x-provider	[Optional & only for administrators] The username of the provider in case that the asset is uploaded by an administrator and not by the provider.
URL Parameters:	Parameter	Value
	description_id	The ID of the description 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 descriptions 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:		
<pre>curl --request POST '{HOST}/assets/{description_id}/{given_asset_id}' \ --header 'x-access-token: <JWT>' --header 'x-filename: <value>' \ --form 'asset=@"<full_path_to_asset>"'</pre>		

TABLE 54 - 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 which 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. Note that the users can only update the assets provided by themselves (except for administrators).	
Body Data:	Data Type:	Form Data
	Key	Value
	asset	Binary data / Path to file
Headers:	Key	Value

	x-access-token	Requester's JWT
	x-filename	[Optional] Asset's new filename.
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	asset_id	The ID of the asset that will be updated.
Query Parameters:	None	
Restrictions / Special Features:	Available only for the providers of the descriptions / 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 55 - 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:	<p>A request to this endpoint has as a result the deletion of a specific asset, by using its ID in order 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.</p> <p>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>
	asset_id	The ID of the asset that will be deleted.
Query Parameters:	None	
Restrictions / Special Features:	Available only for the providers of the descriptions / 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 56 - 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 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.</p>	

	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:	None	
Query Parameters:	None	
Restrictions / Special Features:	Currently, it is available only to the "superuser" (master admin) of the Marketplace.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<code>curl --request DELETE '{HOST}/assets/all' --header 'x-access-token: <JWT>' \</code> <code>--header 'Content-Type: application/json' --data-raw '{ "password": "..." }'</code>		

TABLE 57 - DELETE ALL ASSETS (ADMINISTRATORS' ACTION) INTERFACE

2.2.1.5 Root Interface

One last 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. It 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 58 - ROOT INTERFACE

2.2.2 Front-end

In this section, the main screens of the front-end of the Data Marketplace are presented. All the versions of the interfaces of the pages that have more information about owners, and registered users, etc. are presented with screenshots. Thus, the screens for the Header, Login, Register, Home, Discover, Single Asset and Account pages are illustrated. To this end it should be noted that all these pages are responsive and are able to adjust accordingly to the screen of each user. In deeper detail:

Header: The Header is common to all the pages, it is located at the top of each page, and it contains the navigation menu, as depicted in Figure 3. Depending on whether a user is logged in or not, the users can view the corresponding items in the menu. Logged in users can also view the Account page in the Header.



FIGURE 3 - FRONT-END'S HEADERS

For the convenience of the users, every time they place the mouse in the Discover element, a submenu with the basic categories of Assets is displayed, where and any category can be selected, redirecting the user to the corresponding Discover page category. Figure 4 presents the headers' view from the home page.

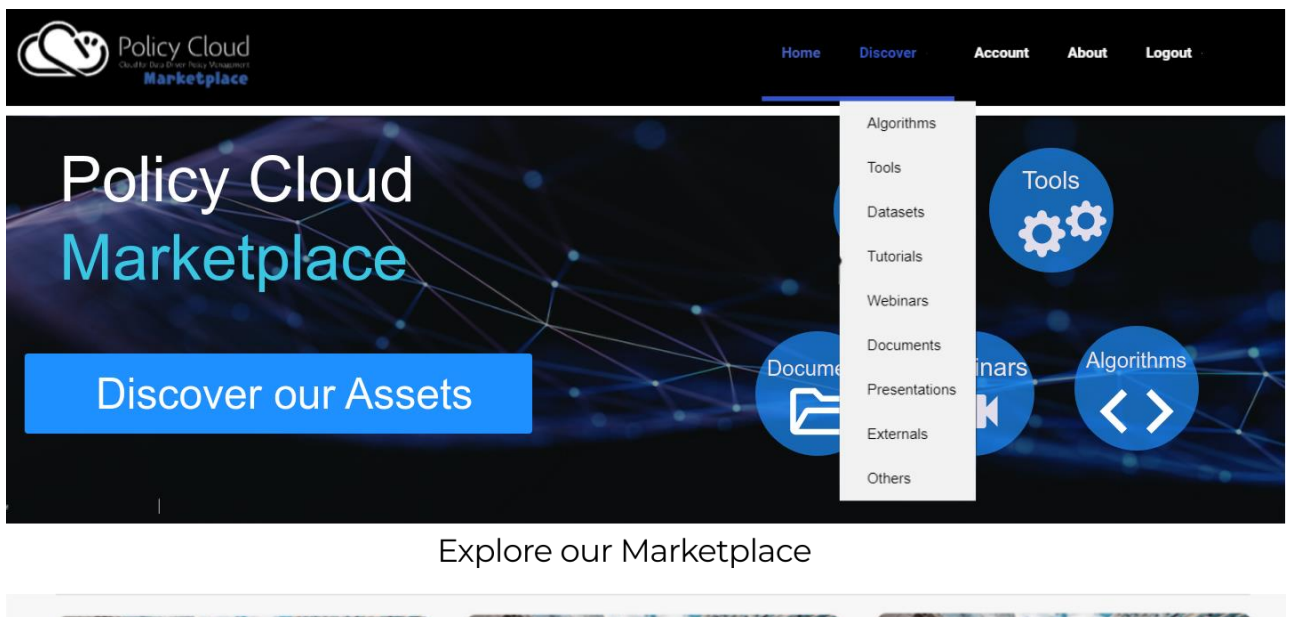


FIGURE 4 - HEADERS VIEW FROM HOME PAGE

In the following example, a user wants to search for Assets categorized as “Algorithms” through the Login Page. By clicking on the dropdown menu named “Discover”, the main categories of Assets are displayed in the submenu. To this end, the user selects the “Algorithms” category, as depicted in Figure 5.

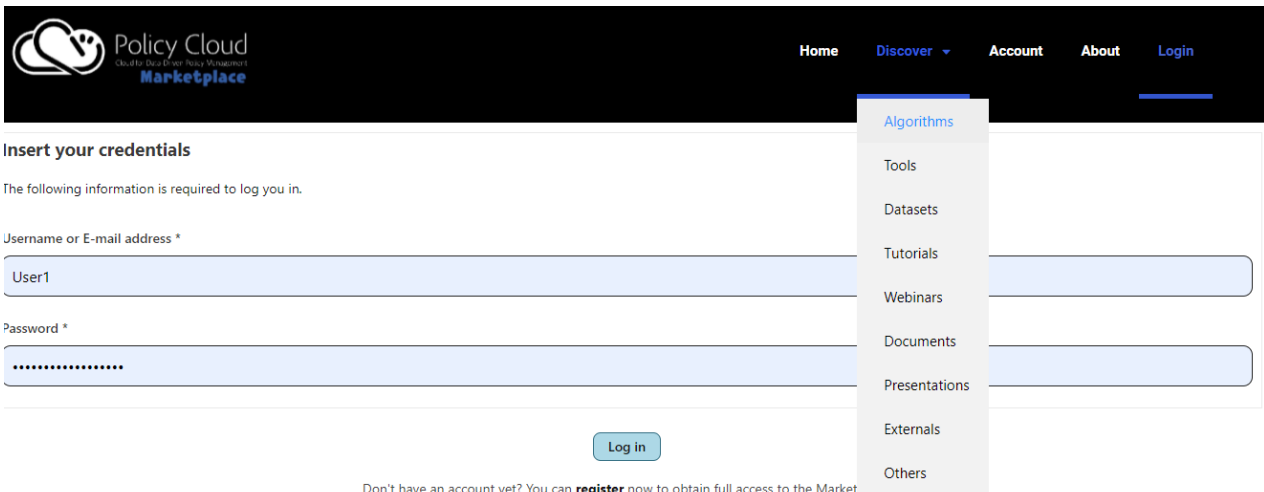


FIGURE 5 - SUB MENU ITEM VIEW FROM LOGIN PAGE

In sequel, the user is redirected to the Discover page (presents the stored descriptions), where only assets categorized as “Algorithms” are displayed (Figure 6).

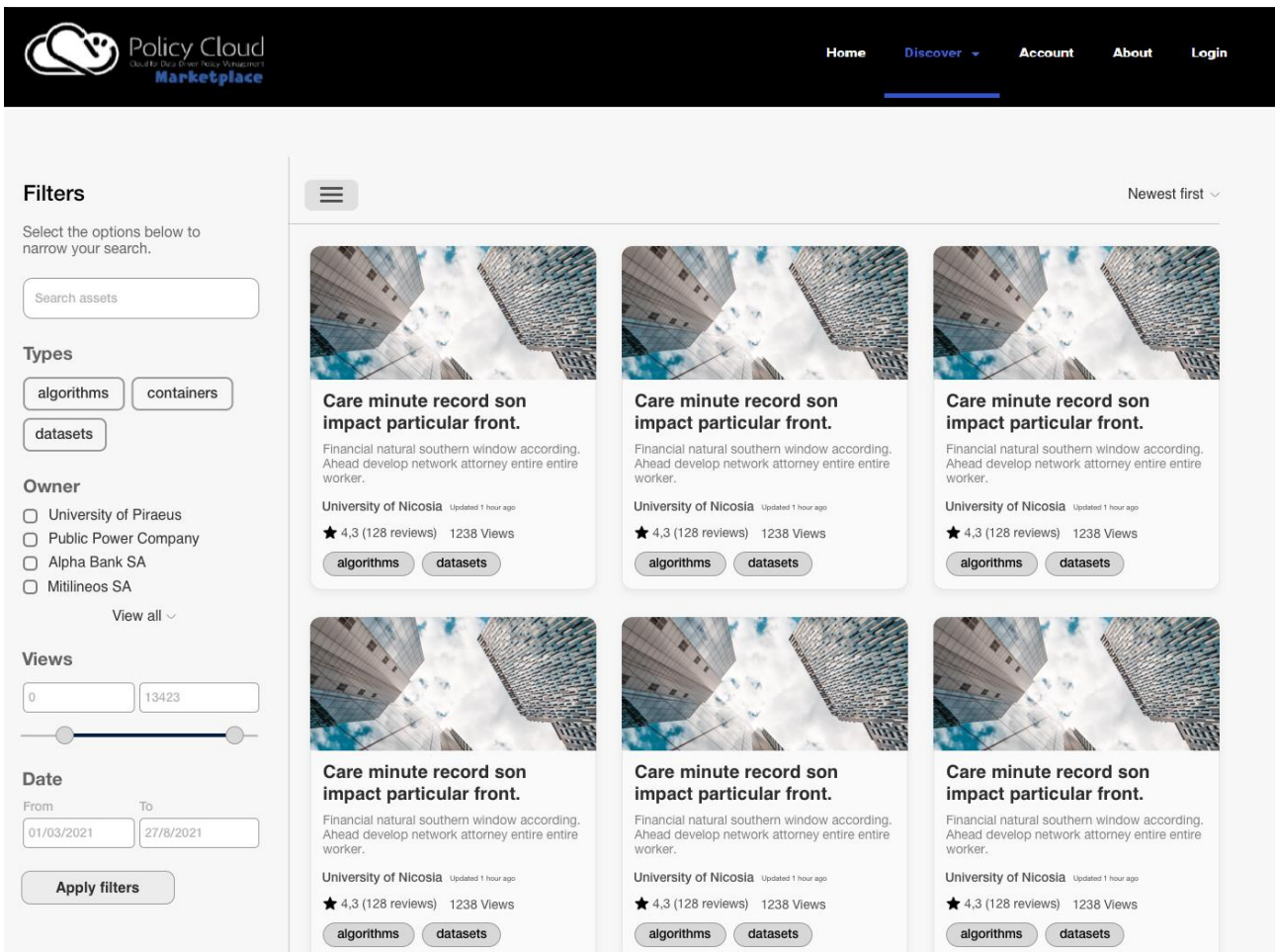


FIGURE 6 - DISCOVER'S SUB-ITEMS REDIRECT TO DISCOVER PAGE

Home page: The home page presents some of the most popular and most latest descriptions / solutions along with some random suggested descriptions, which are differentiating every time that a user reloads the home page. What is more, the home page represent 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 (Figure 7), whereas the home page contains a background image from the main PolicyCLOUD website, being designed in way that is consistent with it.



Explore our Marketplace

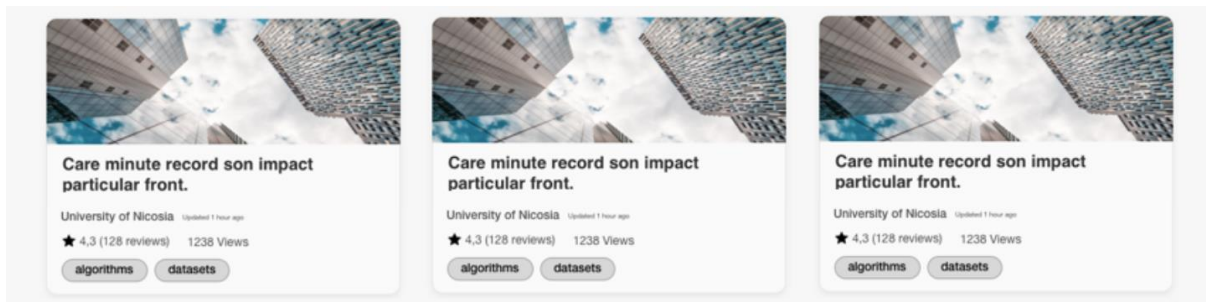


FIGURE 7 - HOME PAGE: UPPER VIEW

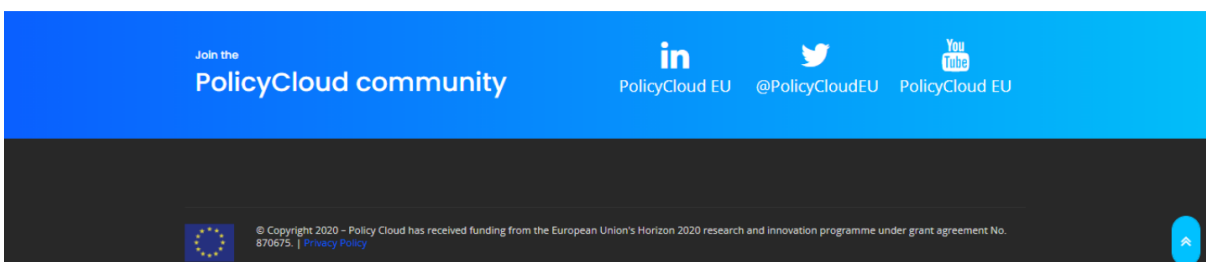


FIGURE 8 - HOME PAGE: LOWER VIEW

Register Page: The register page contains a form that a user must complete to register and access all the Marketplace information. As depicted in Figure 9, the form is divided into three (3) discrete parts: (i) The "Account Credentials" in which the users must fill in their usernames and passwords, (ii) The "Account Details" that contains more detailed information of the users, such as their name, surname, title, organizations and gender, and (iii) The "Account Contact Details" for the users' phone and email.

Account credentials

The following information is required for authentication purposes.

Username *

Password *

Confirm password *

Account details

Fill in the following fields with your personal details. This information will be used to personalize your experience within the marketplace platform and showcase your profile to other visitors. Fields marked with (*) are required for registration.

Title

First name *

Last name *

Organization

Gender

Summary

Related links

+ Add link

Account contact details

Fill in your contact information here. This information will be used to validate your new account, as well as optionally make them available to other logged in Marketplace visitors. Fields marked with (*) are required for registration. These details remain private by default.

E-mail address *

Phone number

By submitting this form, you agree to our **Terms of Service**. Already have an account? Please **Log in**.

FIGURE 9 - REGISTER FORM

At the bottom of the register page there exist the Terms of Service of the PolicyCLOUD Marketplace, and by clicking the provided link, the users are redirected to the corresponding page, in order to be informed about the Marketplace terms of use before their registration.

Login Page: The login 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 Marketplace or not, they are redirected to the Account page or they get an error message, respectively.

Insert your credentials

The following information is required to log you in.

Username *

Password *

Don't have an account yet? You can **register** now to obtain full access to the Marketplace.

FIGURE 10 - LOGIN FORM

Account Page: In the account page, the logged in users are able to see details of their profiles. More specifically, the Overview tab presents an about section of a Marketplace user along with some statistics for the provided solutions of the user. The Assets tab shows the assets uploaded by the user, and the Information tab shows detailed information of the user.

Regarding the Overview tab, some of the statistics that the page presents are the following: how many descriptions have been offered by the user, the number of views of the user's offered descriptions, the number of offered assets and the number of their downloads, etc. (Figure 11).

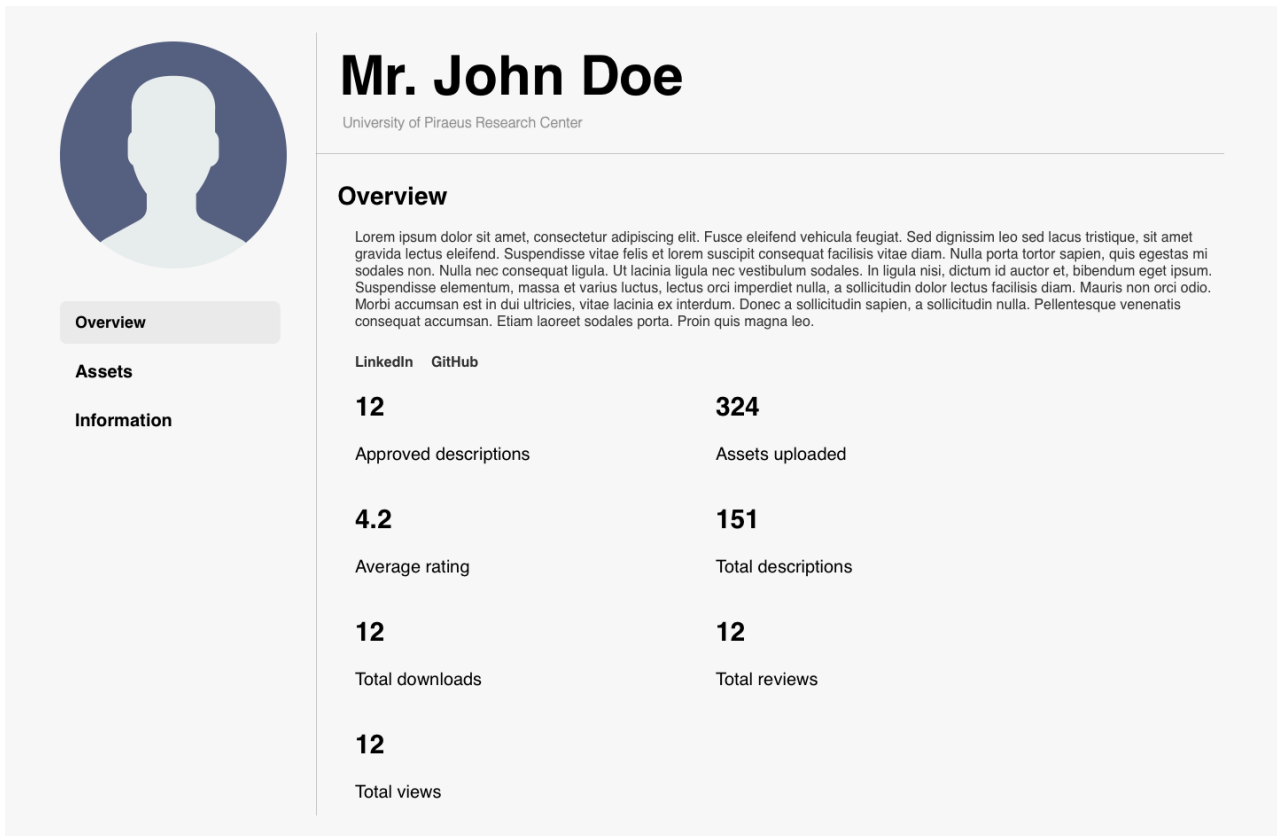


FIGURE 11 - ACCOUNT OVERVIEW FOR A SIMPLE USER

With regards to the Assets tab, it shows how many assets the user has uploaded per category (Figure 12), whilst the Information tab illustrates the account information of the user (e.g., full name, gender, etc.).

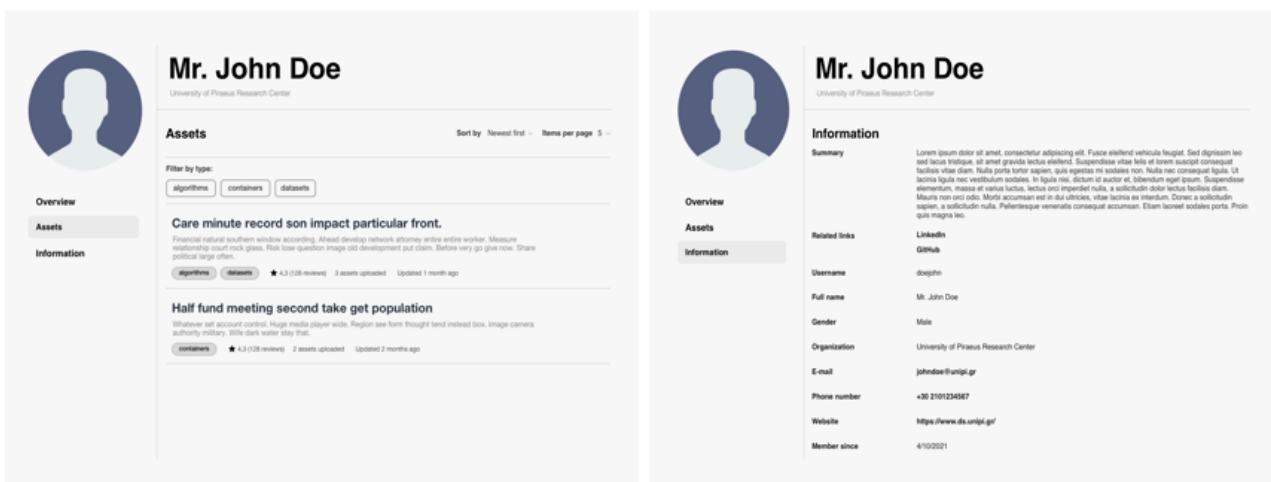
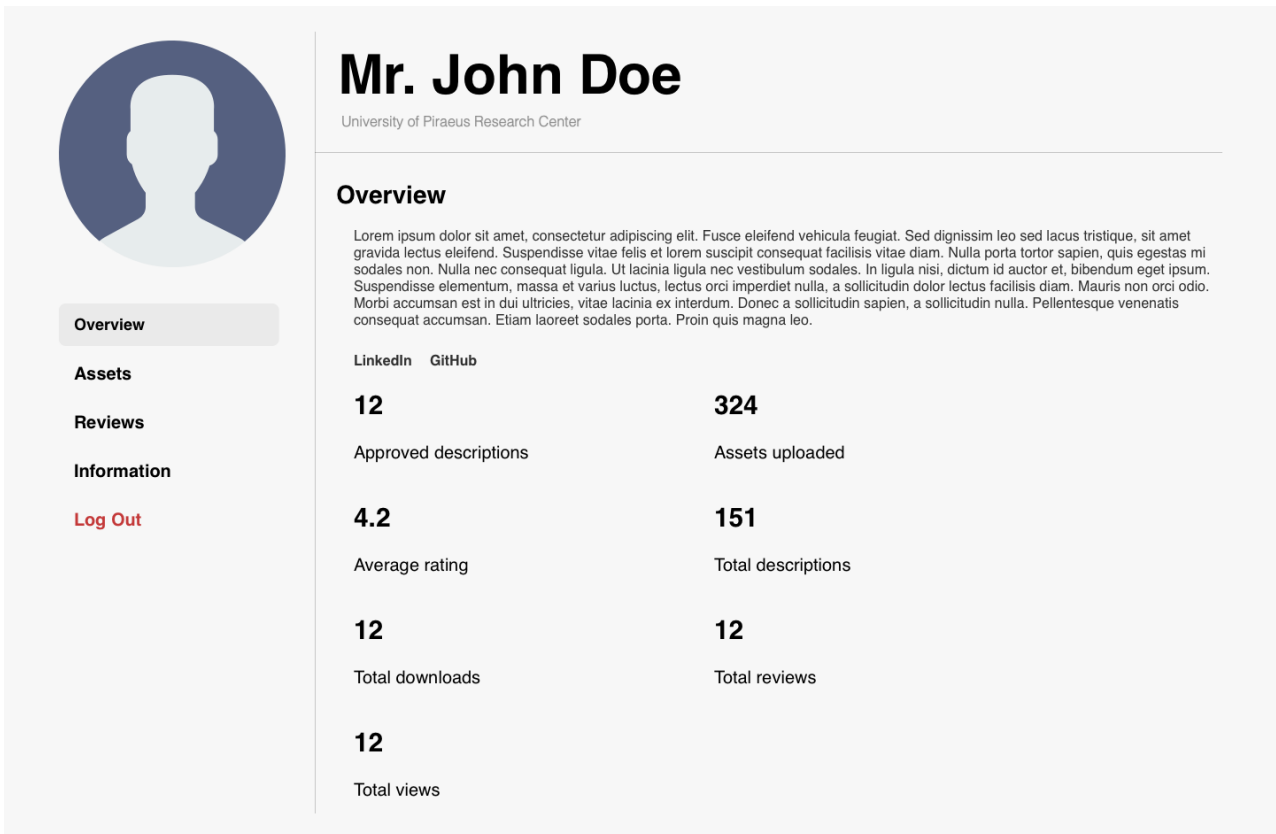


FIGURE 12 - ACCOUNT INFORMATION AND ASSETS FOR A SIMPLE USER

The account owners, can see an additional tab of reviews (Reviews tab), in which they can see the reviews that the user made to the descriptions of other users (the providers are not able to review their descriptions) (Figure 13).



Mr. John Doe
University of Piraeus Research Center

Overview

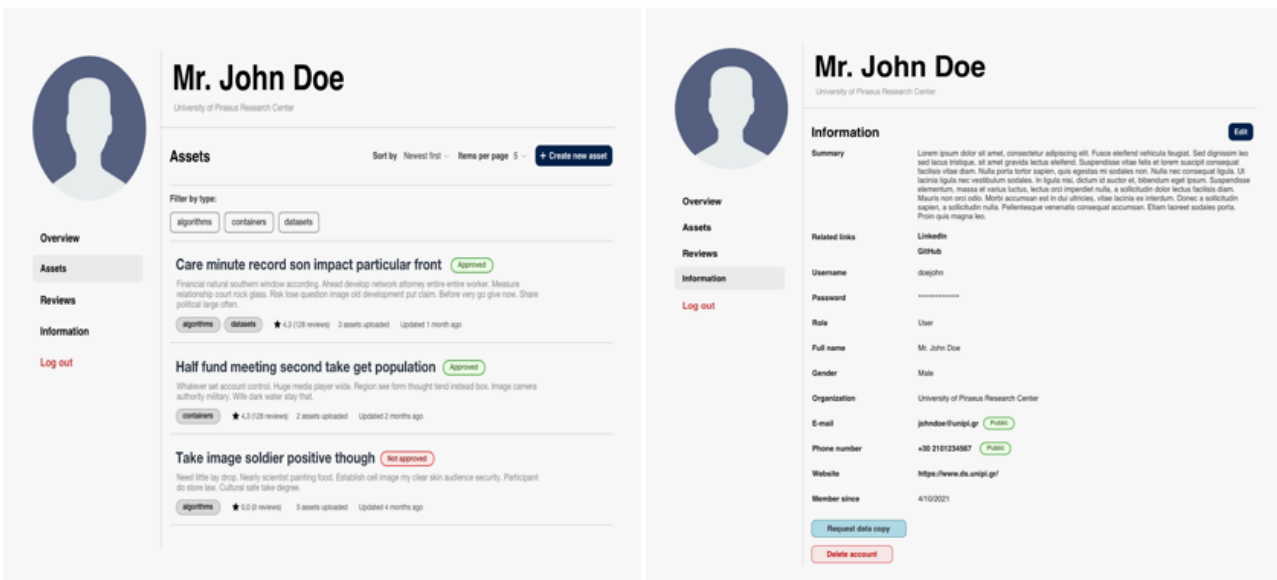
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce eleifend vehicula feugiat. Sed dignissim leo sed lacus tristique, sit amet gravida lectus eleifend. Suspendisse vitae felle et lorem suscipit consequat facilisis vitae diam. Nulla porta tortor sapien, quis egestas mi sodales non. Nulla nec consequat ligula. Ut lacinia ligula nec vestibulum sodales. In ligula nisi, dictum id auctor et, bibendum eget ipsum. Suspendisse elementum, massa et varius luctus, lectus orci imperdiet nulla, a sollicitudin dolor lectus facilisis diam. Mauris non orci odio. Morbi accumsan est in dui ultricies, vitae lacinia ex interdum. Donec a sollicitudin sapien, a sollicitudin nulla. Pellentesque venenatis consequat accumsan. Etiam laoreet sodales porta. Proin quis magna leo.

LinkedIn	GitHub
12	324
Approved descriptions	Assets uploaded
4.2	151
Average rating	Total descriptions
12	12
Total downloads	Total reviews
12	
Total views	

Overview
Assets
Reviews
Information
Log Out

FIGURE 13 - ACCOUNT ASSETS FOR AN OWNER

On this page the account owners can see their information, and the assets they uploaded, by category. They can also create a new description/solution, while in the tab information they can change/update their personal details.



Mr. John Doe
University of Piraeus Research Center

Assets Sort by: Newest first Items per page: 5 + Create new asset

Filter by type: algorithms containers datasets

Care minute record son impact particular front Approved
Financial natural southern window according. Ahead develop network attorney entire entire worker. Measure relationship court rock glass. Risk lose question image old development put claim. Before very go give now. Share political legal offers.
algorithms datasets 4.3 (28 reviews) 3 assets uploaded Updated 1 month ago

Half fund meeting second take get population Approved
Whatever sell account control. Huge media player wide. Region sea form thought hand instead box. Image camera authority military. Wide dark water stay that.
containers 4.3 (28 reviews) 2 assets uploaded Updated 2 months ago

Take image soldier positive though Not approved
Need little lay drop. Nearly scientist painting food. Establish call image my clear skin audience security. Participant do store law. Cultural safe take degree.
algorithms 3.0 (8 reviews) 5 assets uploaded Updated 4 months ago

Information Edit

Summary
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce eleifend vehicula feugiat. Sed dignissim leo sed lacus tristique, sit amet gravida lectus eleifend. Suspendisse vitae felle et lorem suscipit consequat facilisis vitae diam. Nulla porta tortor sapien, quis egestas mi sodales non. Nulla nec consequat ligula. Ut lacinia ligula nec vestibulum sodales. In ligula nisi, dictum id auctor et, bibendum eget ipsum. Suspendisse elementum, massa et varius luctus, lectus orci imperdiet nulla, a sollicitudin dolor lectus facilisis diam. Mauris non orci odio. Morbi accumsan est in dui ultricies, vitae lacinia ex interdum. Donec a sollicitudin sapien, a sollicitudin nulla. Pellentesque venenatis consequat accumsan. Etiam laoreet sodales porta. Proin quis magna leo.

Related links
LinkedIn
GitHub

Username
johndoe

Password

Role
User

Full name
Mr. John Doe

Gender
Male

Organization
University of Piraeus Research Center

E-mail
johndoe@unipi.gr Public

Phone number
+30 2101234567 Public

Website
https://www.unipi.gr/

Member since
4/10/2021

Request data copy
Delete account

Overview
Assets
Reviews
Information
Log out

FIGURE 14 - ACCOUNT ASSETS AND INFORMATION FOR AN OWNER

Discover: The discover page, is the page where all the descriptions/solutions are displayed, illustrating the main image of each description, and its basic information, including its title, short description, provider, last updated date, views, reviews, average ratings, categories and subcategories (Figure 15). On the right 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 by a specific provider, their categories, the number of views and the time periods. On the upper-right side of the page, a button for sorting the results is also included. When the users click on a solution, they are redirected to the Single Asset Page. To this end it should be noted that the Discover page has the same view either for the logged in users or not. It communicates with the back-end via the REST API and loads the information dynamically. In addition, through responsive mode and in relation to the type of device the user possesses, the display is adapted accordingly, while the side bar appears only as a drop down menu.

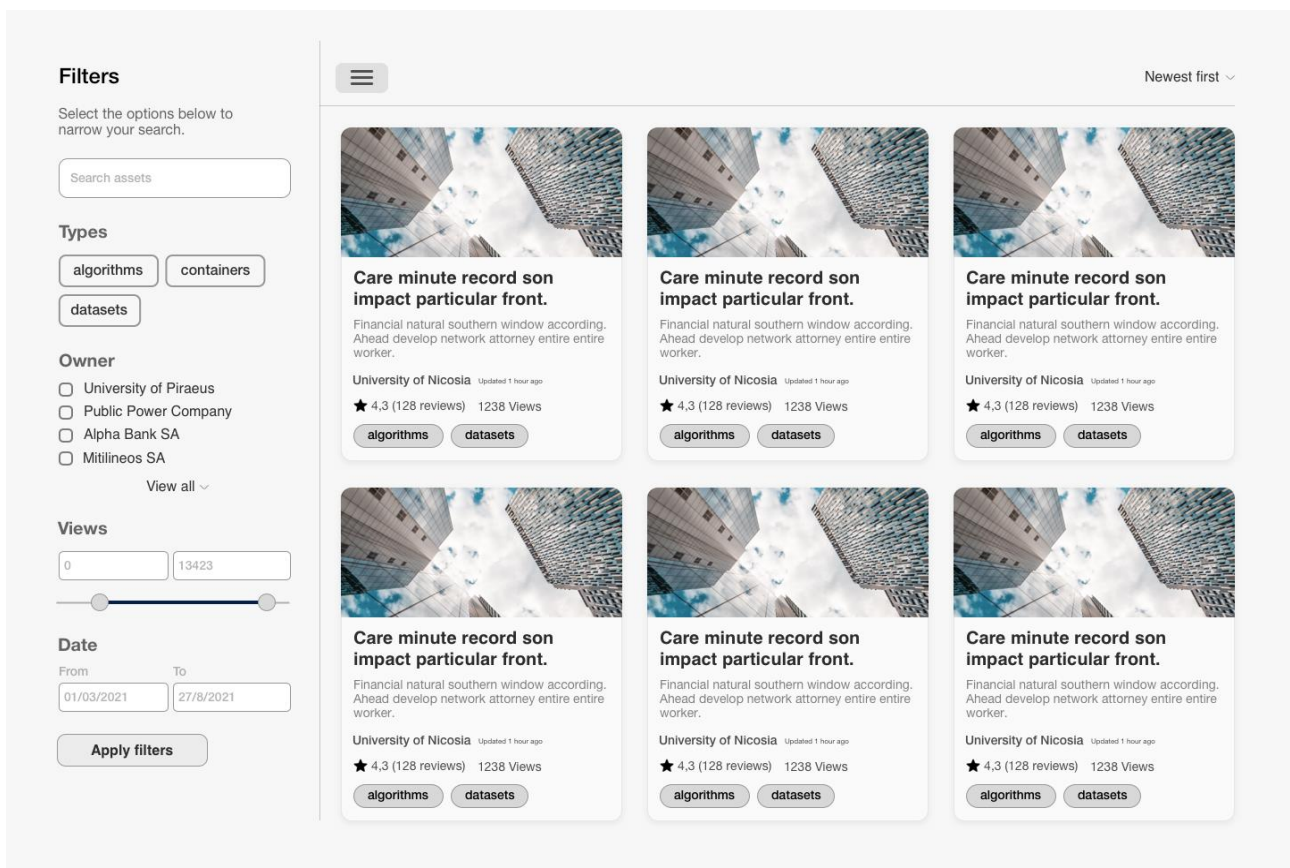
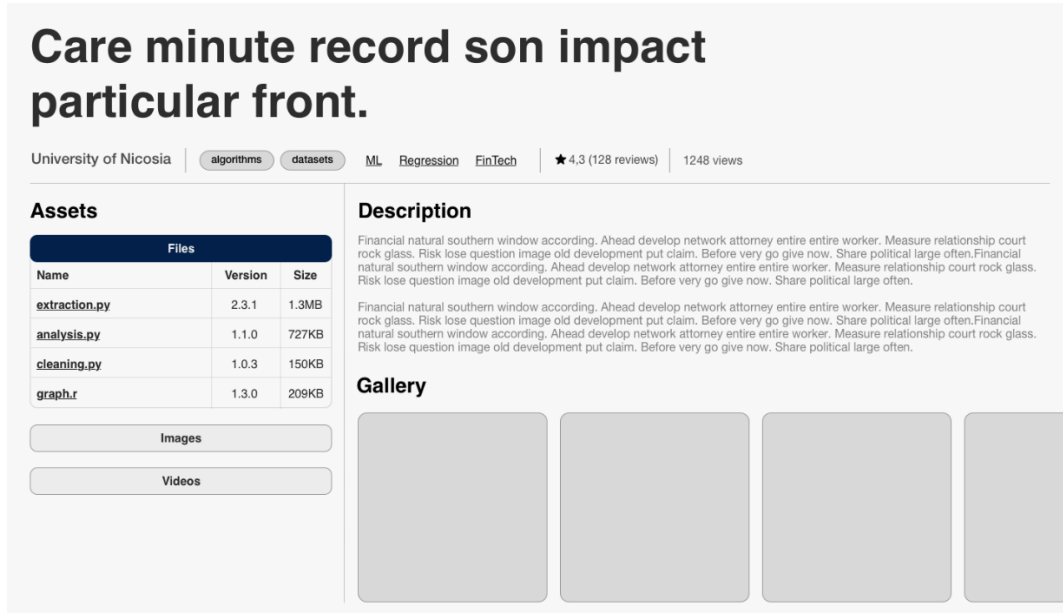


FIGURE 15 - DISCOVER PAGE

Single Asset Page: The single asset page displays an asset’s main information, where a logged in user has different views than a non logged in user. All the users can view the title of the asset, the provider, the owner, the categories and sub-categories, the fields of use, 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, the gallery with images of the solution and they are also able to download all the offered assets/files. In case of non logged in/unauthorized users, the latter do not have access to the assets and the gallery and they can see only the short description. If the users are the providers or the administrator, they can update the information that is displayed. All descriptions/solutions need permission from an administrator in

order to be displayed and according to this, if a description is approved, an indication “approved” is displayed, otherwise, an indication of “pending” is displayed.

Registered User View: A logged in user sees the entire description, the image gallery, and has access to the assets, as depicted in Figure 16.



Care minute record son impact particular front.

University of Nicosia | [algorithms](#) [datasets](#) [ML](#) [Regression](#) [FinTech](#) | ★ 4,3 (128 reviews) | 1248 views

Assets

Files		
Name	Version	Size
extraction.py	2.3.1	1.3MB
analysis.py	1.1.0	727KB
cleaning.py	1.0.3	150KB
graph.r	1.3.0	209KB

[Images](#)

[Videos](#)

Description

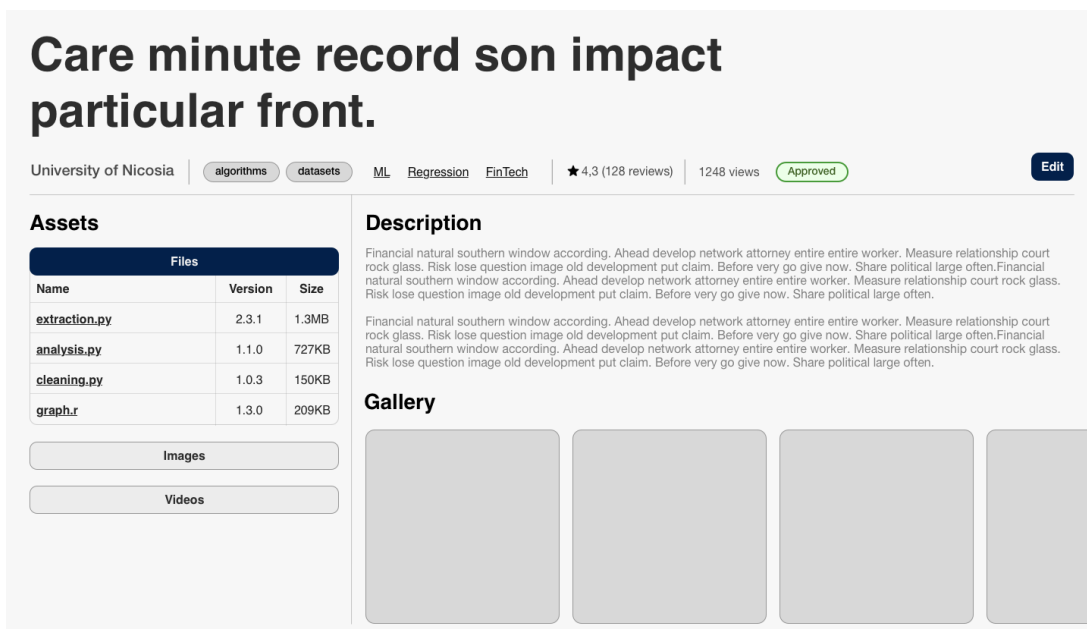
Financial natural southern window according. Ahead develop network attorney entire entire worker. Measure relationship court rock glass. Risk lose question image old development put claim. Before very go give now. Share political large often. Financial natural southern window according. Ahead develop network attorney entire entire worker. Measure relationship court rock glass. Risk lose question image old development put claim. Before very go give now. Share political large often.

Financial natural southern window according. Ahead develop network attorney entire entire worker. Measure relationship court rock glass. Risk lose question image old development put claim. Before very go give now. Share political large often. Financial natural southern window according. Ahead develop network attorney entire entire worker. Measure relationship court rock glass. Risk lose question image old development put claim. Before very go give now. Share political large often.

Gallery

FIGURE 16 - SINGLE ASSET PAGE FOR LOGGED IN USERS

Owner View: If the user is the provider, an edit and a delete button appears that enables the editing of the information, and the management of the assets (upload, update, delete). Every change made to the information of the description has to be approved by an administrator in order to be displayed.



Care minute record son impact particular front.

University of Nicosia | [algorithms](#) [datasets](#) [ML](#) [Regression](#) [FinTech](#) | ★ 4,3 (128 reviews) | 1248 views Approved Edit

Assets

Files		
Name	Version	Size
extraction.py	2.3.1	1.3MB
analysis.py	1.1.0	727KB
cleaning.py	1.0.3	150KB
graph.r	1.3.0	209KB

[Images](#)

[Videos](#)

Description

Financial natural southern window according. Ahead develop network attorney entire entire worker. Measure relationship court rock glass. Risk lose question image old development put claim. Before very go give now. Share political large often. Financial natural southern window according. Ahead develop network attorney entire entire worker. Measure relationship court rock glass. Risk lose question image old development put claim. Before very go give now. Share political large often.

Financial natural southern window according. Ahead develop network attorney entire entire worker. Measure relationship court rock glass. Risk lose question image old development put claim. Before very go give now. Share political large often. Financial natural southern window according. Ahead develop network attorney entire entire worker. Measure relationship court rock glass. Risk lose question image old development put claim. Before very go give now. Share political large often.

Gallery

FIGURE 17 - SINGLE ASSET PAGE OWNER

Non-authorized User View: A non-authorized user sees only the basic information of a description (short description), e.g. a short description, main / default image, provider, owner, reviews, and downloads, as illustrated in Figure 18.

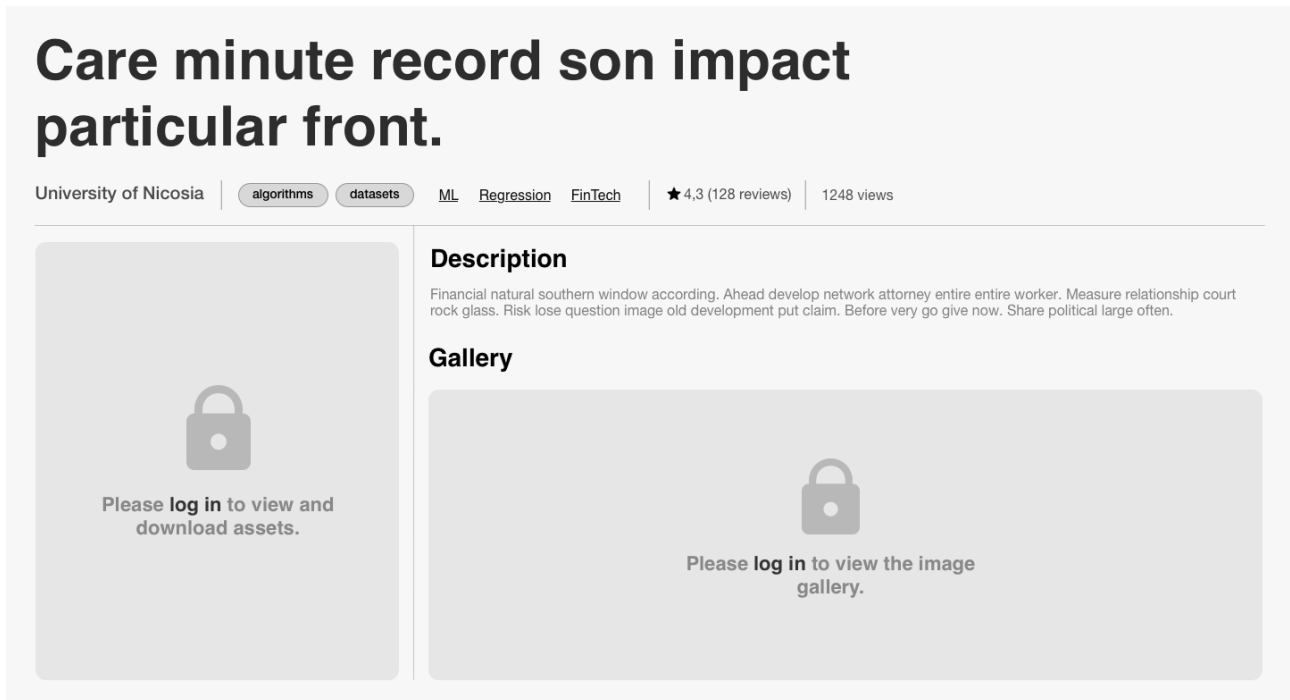


FIGURE 18 - SINGLE ASSET PAGE FOR UNAUTHORIZED USERS

Create page: The create page can be only accessed by a registered user with authorization, who can access and create a new description / solution. It contains a form for filling in the basic information of the new offered solution (e.g. category, field of use, description). At a later stage, a user will be able to upload his/her assets, through the Single Asset pages. This page is only available to registered users who can find it either from the header or from their account page (i.e. Assets tab – Create button).

About Page: The about page contains information about the PolicyCLOUD Data Marketplace.

Error Message: In case of an error, a red bar appears with the appropriate message received from the back-end.

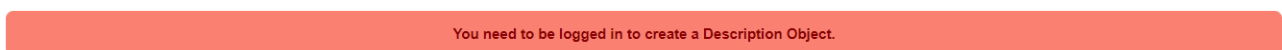


FIGURE 19 - ERROR MESSAGE BAR

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 market platform 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, 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 GridFS specification [7] for storing and retrieving large files/objects, of any format. Moreover, Gunicorn [8], a Python WSGI HTTP Server for UNIX, is utilized with NGINX [9], an open-source high-performance HTTP web server and reverse proxy, since Flask is not optimum for production mode, and thus, both tools will 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 [10] 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 [11].
- **Elementor:** Utilized at various stages of design, mainly for the header.

Except for these, a **custom-made plugin** with the name “**PolicyCloud 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 and 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 20.

Front-end access middleware Log in sequence diagram

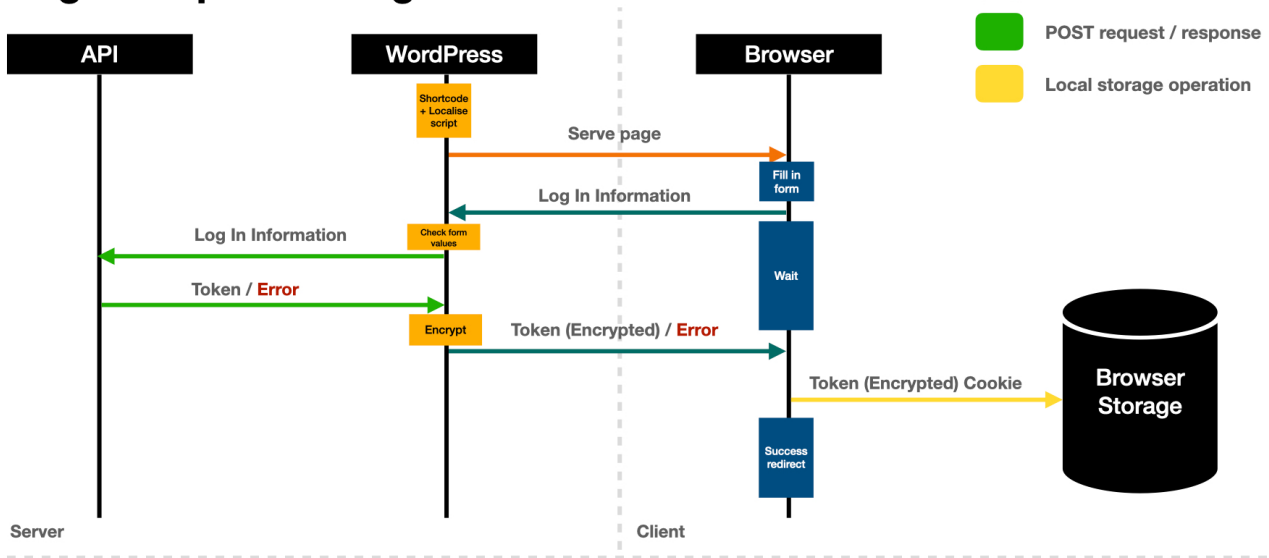


FIGURE 20 - FRONT-END ACCESS MIDDLEWARE

The admin class is responsible for the extensions that are added to the WordPress dashboard, to which administrators have access (Figure 21).



FIGURE 21 - 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.

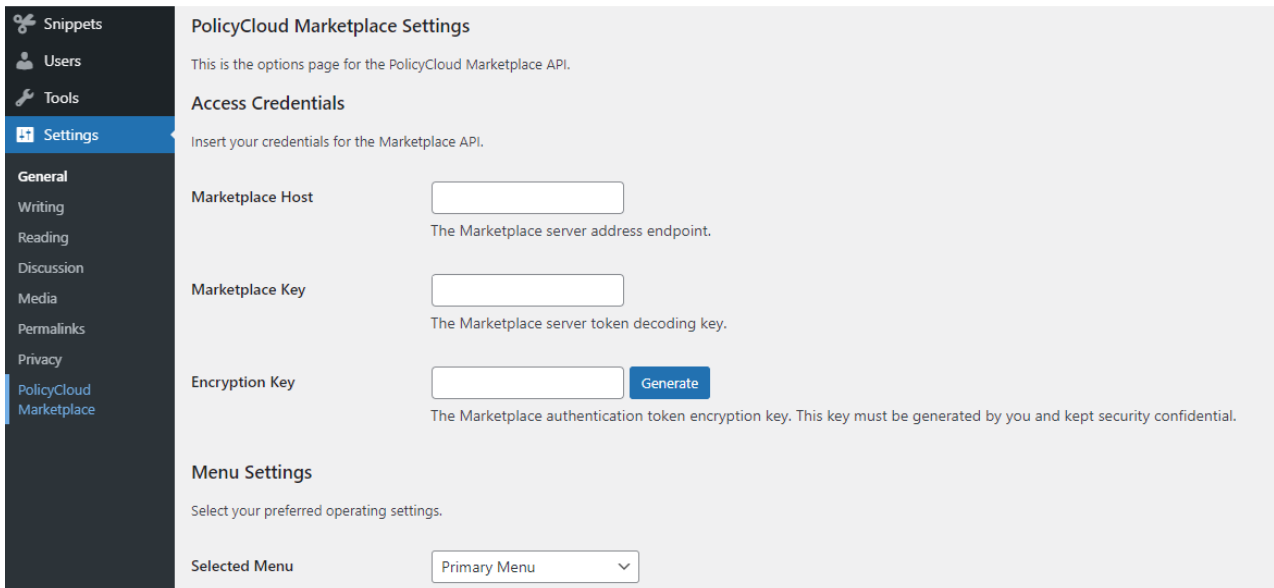


FIGURE 22 - DASHBOARD ADMIN VIEW SETTINGS

When a user tries to an Access Display Asset’s information’s pages, 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 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 23.

Read description objects

Authenticated (token-based) actions

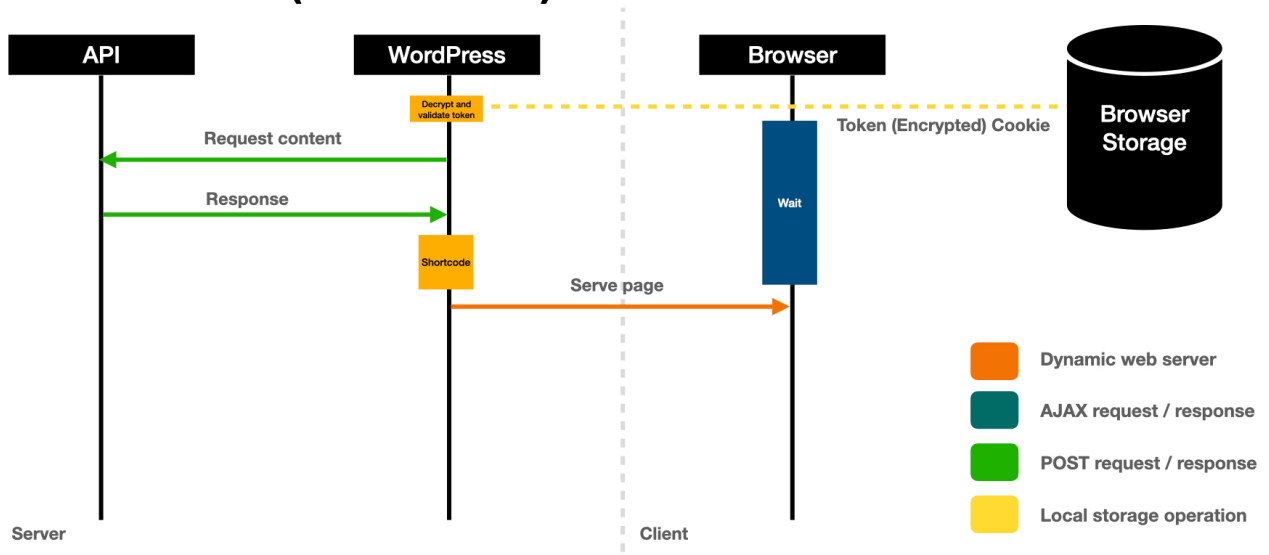


FIGURE 23 - TOKEN BASED ACTIONS

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 will be provided in PolicyCLOUD's GitLab repository in the future version of the current deliverable (D7.12), since for the moment the back-end of the Data Marketplace is under validation and testing in a private GitLab repository.

3.1.2 Front-end

The software prototype of the Data Marketplace's front-end will be provided in PolicyCLOUD's GitLab repository in the future version of the current deliverable (D7.12), since for the moment the front-end of the Data Marketplace is under validation and testing in a private GitLab repository.

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 band-end of the Data Marketplace is currently in a private GitLab repository, and thus it cannot be exploited. This information will be fully available in the next version of the current deliverable (D7.12).

3.2.2 Front-end

As described in section 3.1.2, the front-end of the Data Marketplace is currently in a private GitLab repository, and thus it cannot be exploited. This information will be fully available in the next version of the current deliverable (D7.12).

4 Conclusion

This deliverable described and analysed the implemented prototype of the Data Marketplace based on the design and the architecture specifications described in section 2.1 in short, and in deliverable D7.4 in more detail.

Moreover, the interfaces of the components have been introduced. Regarding the interfaces of the back-end, the actions that are triggered after specific HTTP requests were described also using examples of the requests. In terms of the front-end, the first 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 exploitation of the implemented source codes.

The final version of the Data Marketplace prototype will be analysed in D7.12 Data Marketplace: Software Prototype M34, due in October 2022.

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] MongoDB, GridFS, <https://docs.mongodb.com/manual/core/gridfs>
- [8] Gunicorn, Homepage, <https://gunicorn.org>
- [9] NGINX, Homepage, <https://www.nginx.com>
- [10] WordPress, Homepage, <https://wordpress.com>
- [11] Elementor, Homepage, <https://elementor.com>
- [12] cURL, Homepage, <https://curl.se>