


Tailored IoT & BigData Sandboxes and Testbeds for Smart,  
Autonomous and Personalized Services in the European  
Finance and Insurance Services Ecosystem



D8.3 – BigData and AI Solutions  
Marketplace - I

<b>Revision Number</b>	<b>3.0</b>
<b>Task Reference</b>	T8.2
<b>Lead Beneficiary</b>	UPRC
<b>Responsible</b>	Dimosthenis Kyriazis, Vasilis Koukos
<b>Partners</b>	ABILAB, AGROM, ATOS, BPFI, ENG, FBK, GFT, JSI, NBG, PRIVE, RB, UPRC
<b>Deliverable Type</b>	Report (R)
<b>Dissemination Level</b>	Public (PU)
<b>Due Date</b>	2021-09-30
<b>Delivered Date</b>	2021-10-11
<b>Internal Reviewers</b>	LXS, UNP
<b>Quality Assurance</b>	INNOV
<b>Acceptance</b>	Coordinator Accepted
<b>EC Project Officer</b>	Pierre-Paul Sondag
<b>Programme</b>	HORIZON 2020 - ICT-11-2018
	This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement no 856632

## Contributing Partners

Partner Acronym	Role <sup>1</sup>	Author(s) <sup>2</sup>
<b>UPRC</b>	Lead Beneficiary	Vasilis Koukos, Filipos Kabouris, George Galanos
<b>INNOV</b>	Contributor	John Soldatos, Eftychia Vorila, Ariana Polyviou
<b>UNP</b>	Contributor	Tiago Teixeira, Bruno Almeida

## Revision History

Version	Date	Partner(s)	Description
0.1	2021-07-05	UPRC	ToC Version
0.2	2021-09-17	UPRC	Information on the implementation of the backend of the market platform
0.3	2021-09-24	URPC, UNP	Additional details on the interfaces
0.4	2021-09-27	UPRC, INNOV	Addition of validation scenarios
1.0	2021-09-30	UPRC	Version for internal review
2.0	2021-10-05	UPRC	Version for QA review
3.0	2021-10-11	UPRC	Final version for submission

<sup>1</sup> Lead Beneficiary, Contributor, Internal Reviewer, Quality Assurance

<sup>2</sup> Can be left void

## Executive Summary

This document provides an overview of the INFINITECH marketplace, which is considered one of the most significant artefacts developed in the INFINITECH project. The multi-sided marketplace enables the utilization of Big Data and AI technologies, serving as a single end-point for the available assets found in the marketplace and includes algorithms, datasets, frameworks, webinars and lectures, docker containers and combined solutions, scalable and adaptable to fit into the business needs of companies and organizations, especially of the Fintech and InsuranceTech subsectors.

The assets accommodated in the marketplace have been offered by various INFINITECH members, developers and data scientists and can be leveraged internally, increasing collaboration and knowledge transfer among INFINITECH, or externally by developers, service providers, tech companies and organization, all while promoting the results of the Project and the impact made to the Finance and the Insurance sectors.

As more detailed presentation of the marketplace's specifications and architecture have been provided in past deliverables this document provides a brief overview of the marketplace's specifications and architecture and focuses on the accommodated assets, the developed REST API endpoints descriptions and relevant validation scenarios regarding the functionalities of the back-end.

## Table of Contents

1	Introduction.....	6
1.1	Objective of the Deliverable.....	6
1.2	Insights from other Tasks and Deliverables.....	6
1.3	Structure.....	7
2	The INFINITECH Marketplace .....	8
2.1	Marketplace Overview .....	8
2.2	Architecture and main components.....	9
2.2.1	Back-end .....	9
2.2.2	Front-end .....	10
3	Marketplace Technical Overview .....	12
3.1	Baseline technologies .....	12
3.2	Interfaces (APIs).....	12
3.2.1	APIs related to Users .....	12
3.2.2	APIs related to Descriptions .....	15
3.2.3	APIs related to Assets .....	20
4	Validation scenarios.....	23
4.1.1	Authentication Scenario .....	23
4.1.2	Upload Description Scenario .....	23
4.1.3	Upload Asset Scenario .....	24
5	Conclusions.....	25

## List of Figures

Figure 1 - Market platform's layers and main functionalities. ....	9
Figure 2 - Snapshot of the INFINITECH Marketplace Home web page .....	11

## List of Tables

Table 1 – APIs of the back-end related to Users. ....	12
Table 2 – APIs of the back-end related to Descriptions. ....	15
Table 3 – APIs of the back-end related to Assets. ....	20

## Abbreviations/Acronyms

Abbreviation	Definition
AI	Artificial Intelligence
API	Application Programming Interface
CRUD	Create Retrieve Update Delete - Basic Operations in DBMS
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
ML	Machine Learning
REST	Representational State Transfer software architectural style
UI	User Interface

# 1 Introduction

INFINITECH provides a complete integrated environment enabling the utilization of big data and AI techniques in the finance and insurance sectors. The latter has become feasible through a set of technologies that enable exploitation of various datasets (obtained from different sources), optimized data management for these datasets (e.g., across diverse data stores), analytics with innovative algorithms covering a wide set of scenarios in the finance and insurance sectors, as well as use of tailored sandboxes on the underlying infrastructure layer for the execution of the aforementioned algorithms. Additionally, the development progress of INFINITECH pilots and provided sandboxes and testbeds have increased the INFINITECH ecosystem's resources, ranging from ML models and AI algorithms, to IoT applications, Blockchain and a variety of ready-to-use solutions,

The INFINITECH solution goes beyond the utilization of analytics on specific datasets for a number of pilots / use cases, by aiming at a generalized approach that will facilitate the exploitation of various analytics algorithms (provided both by INFINITECH researchers / partners and by 3<sup>rd</sup> party data analysts) on top of different datasets. To this end, the analytics algorithms need to be made available, to be described in terms of functionality, parameters and offerings, to be accompanied with datasets that can be used by interested parties in order to validate their applicability and performance and to be offered as ready-to-be-executed solutions (e.g., containerized) in order to increase their utilization.

All of the above are representative functionalities of the INFINITECH market platform, which is at an operational stage (accessible at <https://marketplace.infinitech-h2020.eu/>) with various updates and functionalities introduced after the initial version. The marketplace holds and offers the solutions for realizing big data and AI techniques in the finance and insurance sectors.

Based on the above, the INFINITECH's multi-sided market platform aims at being one of the project's main ambassadors to the big data and AI communities. It is a single, public and hybrid system with many different APIs, covering all the different required perspectives of the platform. The various APIs developed related to users, descriptions and assets are described in this document.

The market platform offers big data and AI solutions, as well as IoT and Blockchain solutions, and VDIH Services. Thus, the INFINITECH market platform is a four-perspective, unified environment being able to store several types of assets (e.g., algorithms, descriptions of algorithms, evaluation and validation results, datasets, experimentation outcomes, etc.) in any format.

## 1.1 Objective of the Deliverable

The main objective of this deliverable is to provide details on the implementation of the market platform of INFINITECH, following its design as described in deliverable D8.2 - Market Platform and VDIH Specifications. To this end, it provides an initial summary overview of the already deployed INFINITECH marketplace, its architecture, functionalities and asset accommodation features potentials, while introducing a more detailed view and guidelines on the implementation and utilization of various APIs developed for various users, assets and their respective necessary descriptions.

## 1.2 Insights from other Tasks and Deliverables

This document is closely connected to deliverable D8.2 Market Platform and VDIH Specifications - II, which is in turn an updated version of D8.1, both focusing more on the specifications and detailed presentation of the marketplace's layers, consumers and offerings presentation. Thus, this document provides a brief overview of the marketplace's specifications and architecture, focusing on the accommodated assets, developed REST API endpoints and relevant validation scenarios regarding the functionalities of the back-end.

It should be noted that D8.3 (current deliverable) and D8.6 (IoT and Blockchain Solutions Marketplace) are complementing each other since the marketplace has been realized as one solution offering both big data and AI solutions (main target of D8.3) as well as IoT and blockchain solutions (main target of D8.6).

## 1.3 Structure

The remainder of this deliverable is structured as follows. Section 2 describes an overview of the marketplace architecture, detailing the main features of its core components that are also described in the deliverable D8.2 “Market Platform and VDIH Specifications - II”. Section 3 describes the baseline technologies of the marketplace, as well as the interfaces that have been implemented in the back-end component of the market platform. Section 4 presents some validation scenarios about the usage of back-end and finally, the report concludes with a reference to the presented material in Section 5.

## 2 The INFINITECH Marketplace

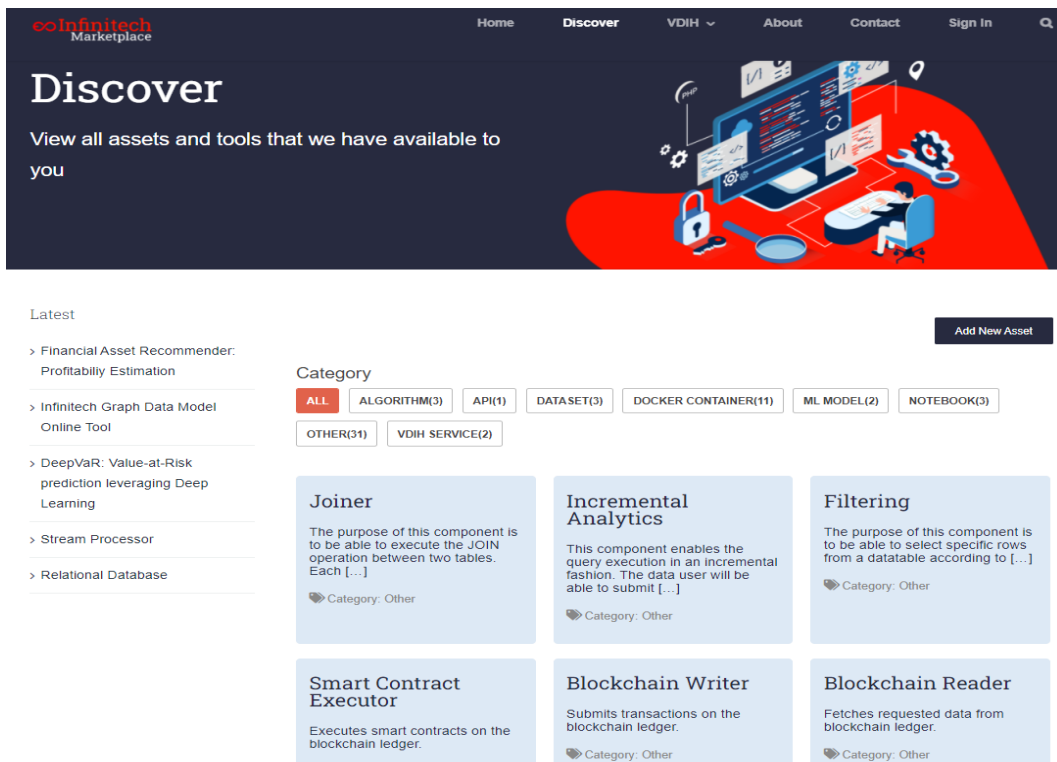
The INFINITECH multi-sided market platform offers BigData, IoT, Blockchain and VDIH solutions is a public web-based environment with various APIs, able to store several types of assets that may derive/result from the separate procedures and mechanisms that are either implemented in the scope of the project or not (e.g. third-party contributions through hackathons/webinars etc.).

### 2.1 Marketplace Overview

The INFINITECH marketplace is already deployed and available at <https://marketplace.infinitech-h2020.eu/>. It is currently populated with various assets and offerings which include:

- Algorithms
- Datasets
- Frameworks & mechanisms
- Scientific studies / tutorials
- Videos / webinars / lectures
- Experimentation results
- VDIH services
- End-to-end solutions
- Docker containers
- Combined solutions

The assets above have been provided by an extended audience, which includes INFINITECH Project members, developers, data scientists, VDIH services providers, as well as other authorized third-party users. The available offerings could be leveraged by a variety of end-users, with increased potential and emphasis being given to FinTech and InsuranceTech subsectors.



With the progress of INFINITECH Project and included pilots’ development, new scalable and adaptable assets are introduced on a regular basis and are made available to the end-users, which can then be deployed and executed based on their needs. Moreover, in conjunction with WP9 tasks and related activities, additional external stakeholders, organizations and service providers are also encouraged to accommodate their assets in the INFINITECH marketplace.



## 2.2 Architecture and main components

The marketplace platform provides several functionalities that are mapped to different layers. In more detail, 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). The four layers of the marketplace along the primary functionalities are depicted in the figure below:

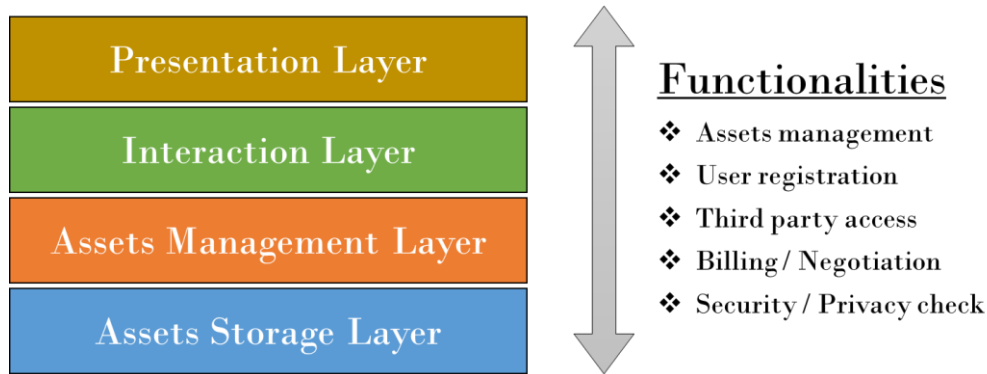


Figure 1 - Market platform's layers and main functionalities.

- 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.

The market platform is structured around two core components, the back-end and the front-end. This approach contributes towards the platform's enhancements in terms of functionality as well as provides additional information and capabilities. In that direction, the end-users are able to interact with the market platform through the front-end (through the presentation layer) that utilizes a user-friendly UI, while other additional services (e.g. from 3rd parties) can be implemented directly with the back-end (through the interaction layer).

The back-end, which contains structured information and the assets offered by the market platform, is considered the core of the marketplace and its functionalities. The front-end component is a user-friendly UI which presents to the users the offered content (the assets and their information), allowing them to interact with the platform in an easy and effective way. This section provides a short overview of the core components of the marketplace, describing some of their key features. It is noted that a more detailed description of the INFINITECH platform specifications and architecture has been provided in deliverable D8.2 -Market Platform and VDIH Specifications II.

### 2.2.1 Back-end

The back-end is the main component of the marketplace. It consists of three different layers and implements the main functionalities for the 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 market platform. 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 added to the platform through the APIs and then stored in the database (Assets Storage Layer), until they are to be deleted for any purpose 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 available 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 marketplace.

The last layer, the Assets Interaction Layer, is responsible for supporting the communication between the market platform and its end-users. It implements the interfaces (APIs) of the back-end (analyzed in section 3.2) 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.

### 2.2.2 Front-end

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

In short, the front-end allows users to register and log-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 deliverable D8.5, entitled "IoT and Blockchain Solutions Marketplace - I".

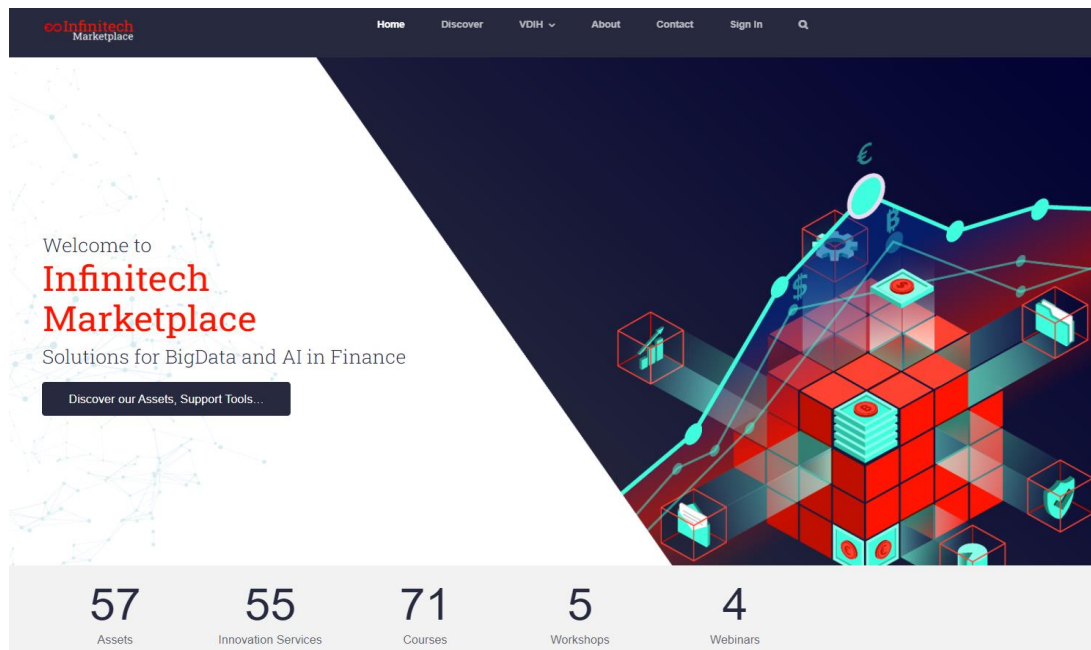


Figure 2 - Snapshot of the INFINITECH Marketplace Home web page

## 3 Marketplace Technical Overview

The following sub-sections are describing the baseline technologies that the INFINITECH marketplace exploits in order to implement its capabilities and its functionalities, and also enlists the core interfaces (APIs) of the back-end component.

### 3.1 Baseline technologies

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 that, among others, offer more efficient management and maintenance, enabling continuous updates and integration. Python is used as the programming language that along with the Flask framework, 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 that is used in combination with GridFS specification for storing and retrieving large files/objects, of any format. Moreover, Gunicorn, a Python WSGI HTTP Server for UNIX, is utilized with NGINX, 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.

In terms of the front-end, it has been implemented using various web technologies (HTML, CSS, etc.) and it is functional using PHP and JavaScript technologies. It also exploits WordPress and various plugins of it, in order to manage the content that is presented.

### 3.2 Interfaces (APIs)

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.

#### 3.2.1 APIs related to Users

This group of APIs offers functionalities intended for the management of marketplace users. The most important functionality is that of user registration, as it is necessary for the usage of the rest functionalities. 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 marketplace.

Table 1 – APIs of the back-end related to Users.

Action	HTTP Method	Endpoint
Register a new user (Sign in)	POST	{HOST}/accounts/users/registration
Check username availability	GET	{HOST}/accounts/username_availability
Authenticate a user (Login)	POST	{HOST}/accounts/users/authentication
Get user's information	GET	{HOST}/accounts/users/information
Update user's information	PUT	{HOST}/accounts/users/information
Change user's password	POST	{HOST}/accounts/users/change_password
Reset user's password	POST	{HOST}/accounts/users/password_reset
Delete user's account	DELETE	{HOST}/accounts/users/delete_account

{HOST} refers to the hosting domain name and the port that the back-end runs.

Most of these actions require additional fields in the headers or even in the body of the HTTP request. Example of a required field is the API key.

A more detailed description of all the actions in the table is presented below:

- Register a new user (Sign in) - {HOST}/accounts/users/registration

From this endpoint the registrations of the marketplace users are made. A POST request should be made and the next JSON schema must be in its body as raw data.

```
{
  "username": "...",
  "first_name": "...",
  "last_name": "...",
  "email": "...",
  "password": "..."
}
```

The following is an example of the request in cURL:

```
curl --request POST '{HOST}/accounts/users/registration' \
--header 'Content-Type: application/json' \
--data-raw '{
  "username": "...",
  "first_name": "...",
  "last_name": "...",
  "email": "...",
  "password": "..."
}'
```

- Check username availability - {HOST}/accounts/username\_availability

This endpoint is used in order to check the availability of a username during the users’ registration. A GET request should be made and the key “username” must be included in the headers of the request.

The following is an example of the request in cURL:

```
curl --request GET '{HOST}/accounts/username_availability' \
--header 'username: <value>'
```

- Authenticate a user (Login) - {HOST}/accounts/users/authentication

Through this endpoint, the users are authenticated in order to log in to their account. A POST request should be made and the next JSON schema, containing users’ credentials, must be in its body as raw data. It is noted that users can log in either with their email or with their username. Finally, single sign-on (SSO) schemes are supported (only through the front-end) for registration and login, using accounts from social media (i.e. Google, LinkedIn, Github).

```
{ "username": "...", "email": "...", "password": "..." }
```

The following is an example of the request in cURL:

```
curl --request POST '{HOST}/accounts/users/authentication' \
--header 'Content-Type: application/json' \
--data-raw '{ "username": "...", "email": "...", "password": "..." }'
```

- Get user's information - {HOST}/accounts/users/information

This endpoint is used in order to retrieve information about a user. Some of the information that is retrieved, are first and last name, about section, email, etc. A GET request should be made and the key "username" must be included in the headers of the request. Also, this endpoint is restricted and thus, users' API keys must be included too in the headers of the request.

The following is an example of the request in cURL:

```
curl --request GET '{HOST}/accounts/users/information' \
--header 'username: <value>' --header 'API_KEY: <value>'
```

- Update user's information - {HOST}/accounts/users/information

This endpoint handles requests for updating users' information. A PUT request should be made and the next JSON schema, containing users' information, must be in its body as raw data. Also, this endpoint is restricted and thus, users' API keys must be included in the headers of the request.

```
{ "first_name": "...", "last_name": "...", "about": "...", "email": "...", ... }
```

The following is an example of the request in cURL:

```
curl --request PUT '{HOST}/accounts/users/information' \
--header 'API_KEY: <value>' --header 'Content-Type: application/json' \
--data-raw '{ "first_name": "...", "last_name": "...", "about": "...", "email": "...", ... }'
```

- Change user's password - {HOST}/accounts/users/change\_password

This endpoint is used when the users want to change their account password. A POST request should be made and the next JSON schema, containing users' new and old credentials, must be in its body as raw data. Also, this endpoint is restricted and thus, users' API keys must be included in the headers of the request.

```
{ "username ": "...", "old_password": "...", "new_password": "..." }
```

The following is an example of the request in cURL:

```
curl --request POST '{HOST}/accounts/users/change_password' \
--header 'API_KEY: <value>' --header 'Content-Type: application/json' \
--data-raw '{ "username ": "...", "old_password": "...", "new_password": "..." }'
```

- Reset user's password - {HOST}/accounts/users/password\_reset

This endpoint handles the process of changing users' passwords after a password reset request. It works in combination with the front-end's features which, at first, sends an email to the users with a password reset link that redirects to a form from which the users can set their new password. After this process, the front-end sends a POST request to the back-end with the new credentials of the user (username and password).

```
{ "username ": "...", "password": "...", "password_reset_code": "..." }
```

The following is an example of the request in cURL:

```
curl --request POST '{HOST}/accounts/users/change_password' \
--header 'API_KEY: <value>' --header 'Content-Type: application/json' \
--data-raw '{ "username ": "...", "password": "...", "password_reset_code": "..." }'
```

- Delete user's account - {HOST}/accounts/users/delete\_account

In order to delete an account, this endpoint should be used, making a DELETE request and providing users' credentials in its body, as raw data (JSON format). As in the previous endpoints, the endpoint is restricted and thus, users' API keys must be included in the headers of the request.

```
{ "username ": "...", "password": "..." }
```

The following is an example of the request in cURL:

```
curl --request DELETE '{HOST}/accounts/users/delete_account' \
--header 'API_KEY: <value>' --header 'Content-Type: application/json' \
--data-raw '{ "username ": "...", "password": "..." }'
```

### 3.2.2 APIs related to Descriptions

This group of APIs offers functionalities intended for the management of the descriptions. They support all CRUD operations and 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 collections at once, using a keyword named "all". The collections of the database, as described in deliverable D8.2 "Market Platform and VDIH Specifications - II", vary as well as the offered marketplace's types of assets. The current list of the collections can be found at the end of the following Table 2, which presents the endpoints related to Descriptions as they are in the first version of the marketplace.

Table 2 – APIs of the back-end related to Descriptions.

Action	HTTP Method	Endpoint
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 "collection")	GET	{HOST}/descriptions/{collection}/{description_id}
Get latest descriptions from all collections	GET	{HOST}/descriptions/all/latest
Get latest descriptions from a specific 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
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 (“collection”)	PUT	{HOST}/descriptions/{collection}/{description_id}
Delete a specific description (using keyword “all”)	DELETE	{HOST}/descriptions/all/{description_id}
Delete a specific description (“collection”)	DELETE	{HOST}/descriptions/{collection}/{description_id}
Delete all descriptions	DELETE	{HOST}/descriptions/all/all
Delete all descriptions from specific collection	DELETE	{HOST}/descriptions/{collection}/all

{HOST} refers to the hosting domain name and the port that the back-end runs.

As a {collection} can be one of the following values derived from the current types of offered assets:

*“algorithms”, “notebooks”, “ml\_models”, “datasets”, “api”, “webinars”, “workshops”, “courses”, “vdih\_services”, “containers”, “uncategorized”, “other”*

Some of these actions require additional fields in the headers of the HTTP request. Example of a required field is the API key.

- Get a list with all descriptions - {HOST}/descriptions/all

A GET request to this endpoint will result in the retrieval of the stored descriptions from all collections. It uses the keyword “all” instead of a collection and that makes the system retrieve descriptions from all collections at once.

The following is an example of the request in cURL:

```
curl --request GET '{HOST}/descriptions/all'
```

- Get a list with all descriptions from a specific collection - {HOST}/descriptions/{collection}

This GET request is similar to the above request. The only difference between these two actions is that this request retrieves descriptions from a single – specific collections (instead of using keyword “all”). The values for the collection can be found below the Table 2.

The following is an example of the request in cURL:

```
curl --request GET '{HOST}/descriptions/{collection}'
```

- Get a specific description (using keyword “all”) - {HOST}/descriptions/all/{description\_id}

With this GET request, it is possible to retrieve a single description. The retrieval of a specific description is done using its unique identification code (ID) that is known when uploading it. Also, the retrieval of a specific description can be done both using keyword “all” and using the name of the collection that the description has been stored. This is feasible because the back-end ensures that the IDs are unique regardless of in which collection a description has been stored.

The following is an example of the request in cURL:

```
curl --request GET '{HOST}/descriptions/all/{description_id}'
```



- Get a specific description (using “collection”) - {HOST}/descriptions/{collection}/{description\_id}

This GET request is similar to the above request, with the difference that it uses a specific collection for the retrieval of the description (instead of using keyword “all”). The values for the collection can be found below the Table 2.

The following is an example of the request in cURL:

```
curl --request GET '{HOST}/descriptions/{collection}/{description_id}'
```

- Get latest descriptions from all collections - {HOST}/descriptions/all/latest

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 is a GET request, using the keyword “all” and has as result the twenty latest descriptions form all descriptions.

The following is an example of the request in cURL:

```
curl --request GET '{HOST}/descriptions/all/latest'
```

- Get latest descriptions from a specific collection - {HOST}/descriptions/{collection}/latest

This request is similar to the above GET request. It uses the value of a specific collections and not the keyword “all” and this results to return sorted the most recent descriptions of a specific collection. The values for the collection can be found below the Table 2.

The following is an example of the request in cURL:

```
curl --request GET '{HOST}/descriptions/{collection}/latest'
```

- Get random descriptions from all collections - {HOST}/descriptions/all/random

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 INFINITECH market platform, where random descriptions are displayed.

The following is an example of the request in cURL:

```
curl --request GET '{HOST}/descriptions/all/random'
```

- Get random descriptions from a specific collection - {HOST}/descriptions/{collection}/random

This endpoint is similar to the above endpoint. Instead of keyword “all” it uses a specific collection and thus it returns a number of random descriptions of the provided specific collection. The values for the collection can be found below the Table 2.

The following is an example of the request in cURL:

```
curl --request GET '{HOST}/descriptions/{collection}/random'
```

- Upload / Create a new description with random ID - {HOST}/descriptions/{collection}

Through this POST request, the users can upload their descriptions. It requires to specify (at the end of the endpoint) the collection in which the description will be stored (the values for the collection can be found below the Table 2). Also, in the headers of the request, the API key of the provider is necessary to be included.

In the body of the request should be placed the contents of the description, as raw data in JSON format. The schema of the descriptions’ content varies, and it is flexible to be extended. The following JSON schema presents some required fields of a description.

```
{
  "owner ": "<organization / author / etc.>", "contact": "<provider's name and email>",
  "description ": "<description of the provided asset>", "name": "<name/title of the asset>",
  "type": "<type of the asset (same as collection values)>",
  "simpleComposite": "<simple (single file) / composite (set of files e.g. compressed file)>",
  "availability": "<reflects the users who can see the description: public / infinitech / specific
                  Work Packages or Pilots / Other>",
  "fieldOfUse": ["<field 1>", ... ], "comments": "<provider's comments>"
}
```

Except of these fields, there are also some optional fields, like “subtype”, “deliveryDate”, “link” and others.

The following is an example of the request in cURL:

```
curl --request POST '{HOST}/descriptions/{collection}' \
--header 'API_KEY: <value>' --header 'Content-Type: application/json' \
--data-raw '{
  "owner ": "<organization / author / etc.>", "contact": "<provider's name and email>",
  "description ": "<description of the provided asset>", "name": "<name/title of the asset>",
  "type": "<type of the asset (same as collection values)>",
  "simpleComposite": "<simple (single file) / composite (set of files e.g. compressed file)>",
  "availability": "<reflects the users who can see the description: public / infinitech / specific
                  Work Packages or Pilots / Other>",
  "fieldOfUse": ["<field 1>", ... ], "comments": "<provider's comments>"
}'
```

- Upload / Create a new description with given ID - {HOST}/descriptions/{collection}/{given\_id}

This endpoint is similar to the above POST request. The difference is that at the end of the endpoint there is a given ID for the new description. This endpoint is only available to administrators. The values for the collection can be found below the Table 2.

The following is an example of the request in cURL:

```
curl --request POST '{HOST}/descriptions/{collection}/{given_id}' \
--header 'API_KEY: <value>' --header 'Content-Type: application/json' \
--data-raw '{
  "owner": "<organization / author / etc.>", "contact": "<provider's name and email>",
  "description": "<description of the provided asset>", "name": "<name/title of the asset>",
  "type": "<type of the asset (same as collection values)>",
  "simpleComposite": "<simple (single file) / composite (set of files e.g. compressed file)>",
  "availability": "<reflects the users who can see the description: public / infinitech / specific
                  Work Packages or Pilots / Other>",
  "fieldOfUse": ["<field 1>", ... ], "comments": "<provider's comments>"
}'
```

- Update a specific description (using keyword “all”) - {HOST}/descriptions/all/{description\_id}

With this endpoint, the providers and the administrators, are able to update the content of the (/their) descriptions. It is possible to update the whole description or only some fields. It requires the ID of the description to be at the end of the endpoint and the users’ API key in the headers of the PUT request. As in the create action, the description should be provided as raw data in JSON format. Note that this endpoint uses the keyword “all”.

The following is an example of the request in cURL:

```
curl --request PUT '{HOST}/descriptions/all/{description_id}' \
--header 'API_KEY: <value>' --header 'Content-Type: application/json' \
--data-raw '{
  "existing_or_new_field": "<value>", ...
}'
```

- Update a specific description (using “collection”) - {HOST}/descriptions/{collection}/{description\_id}

This PUT request is similar to the previous request. The only difference is that instead of using keyword “all” it uses a specific collection. The values for the collection can be found below the Table 2.

The following is an example of the request in cURL:

```
curl --request PUT '{HOST}/descriptions/{collection}/{description_id}' \
--header 'API_KEY: <value>' --header 'Content-Type: application/json' \
--data-raw '{
  "existing_or_new_field": "<value>", ...
}'
```

- Delete a specific description (using keyword “all”) - {HOST}/descriptions/all/{description\_id}

A DELETE request to this endpoint has as a result the deletion of a specific description, using its ID. This endpoint is restricted and thus, users’ API keys must be included in the headers of the request. Note that a description can be deleted only by its provider and the administrators.

The following is an example of the request in cURL:

```
curl --request DELETE '{HOST}/descriptions/all/{description_id}' --header 'API_KEY: <value>'
```

- Delete a specific description (using “collection”) - {HOST}/descriptions/{collection}/{description\_id}

This DELETE request is similar to the above request, with the difference that it uses a specific collection (instead of keyword “all”). The values for the collection can be found below the Table 2.

The following is an example of the request in cURL:

```
curl --request DELETE '{HOST}/descriptions/{collection}/{description_id}' \
--header 'API_KEY: <value>'
```

- Delete all descriptions - {HOST}/descriptions/all/all

Through this DELETE request, the administrators (and only them) can delete all descriptions and from all collections (keyword “all”). It is a restricted endpoint and thus, users’ API keys must be included in the headers of the request.

The following is an example of the request in cURL:

```
curl --request DELETE '{HOST}/descriptions/all/all' \
--header 'API_KEY: <value>'
```

- Delete all descriptions from specific collection - {HOST}/descriptions/{collection}/all

This DELETE request is similar to the above request, and it is only available to administrators. With this request it is possible to delete all descriptions from a specific collection. It is a restricted endpoint and thus, users’ API keys must be included in the headers of the request. The values for the collection can be found below the Table 2.

The following is an example of the request in cURL:

```
curl --request DELETE '{HOST}/descriptions/{collection}/all' \
--header 'API_KEY: <value>'
```

### Search functionality

The search functionality is a vital requirement for the most services in order to reduce the number of objects returned by a query. Thus, relative query filters can be added to the endpoints of the back-end that retrieve multiple descriptions simultaneously. These filters enable the users of the marketplace to search for assets, based on various parameters from the content of the stored descriptions. Currently, the search functionality supports only equality queries and relevant equality queries (i.e. if contains a value).

In order to make a query, at the end of the endpoint should be used the following syntaxes:

```
Single attribute: '{HOST}/descriptions/all?<attribute_name>=<value>'
More attributes: '{HOST}/descriptions/all?<attribute_1>=<value>&<attribute_2>=<value>&...'
For attribute in lower hierarchical level:
    '{HOST}/descriptions/all?<attribute_level1>.<attribute_level2>.<...>=<value>'
```

The following search example request in cURL, returns the descriptions that in their title contain the value “machine learning” and their type is “algorithms”:

```
curl --request GET '{HOST}/descriptions/all?title=machine%20learning&type=algorithm'
```

The search functionality is available in all GET requests except for the request of getting a specific description.

### 3.2.3 APIs related to Assets

This group of APIs offers functionalities intended for the management of the assets. They support all CRUD operations for files which are stored in the GridFS specification of MongoDB database. Table 3 presents the endpoints related to Assets as they are in the first version of the marketplace.

Table 3 – APIs of the back-end related to Assets.

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 specific description	POST	{HOST}/assets/{description_id}
Upload a new asset with given ID, linked to 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	DELETE	{HOST}/assets/all

{HOST} refers to the hosting domain name and the port that the back-end runs.

{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 API key.

- Get a list with the stored assets - {HOST}/assets

A GET request to this endpoint will result in the retrieval of a list with the stored assets and some additional information of them.

The following is an example of the request in cURL:

```
curl --request GET '{HOST}/assets'
```

- Get a specific asset, using its ID - {HOST}/assets/{asset\_id}

This GET 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, users' API keys must be included in the headers of the request.

The following is an example of the request in cURL:

```
curl --request GET '{HOST}/assets/{asset_id}' --header 'API_KEY: <value>'
```

- Upload a new asset with random ID - {HOST}/assets/{description\_id}

Through this POST 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 API key of the provider and b) the asset's filename. The assets have to be uploaded as form-data with the key "asset".

The following is an example of the request in cURL:

```
curl --request POST '{HOST}/assets/{description_id}' \
--header 'API_KEY: <value>' --header 'filename: <value>' \
--form 'file=@"<full_path_to_asset>"'
```

- Upload a new asset with given ID - {HOST}/assets/{description\_id}/{given\_asset\_id}

This POST request is similar to the previous endpoint. 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 of the market platform.

The following is an example of the request in cURL:

```
curl --request POST '{HOST}/assets/{description_id}/{given_asset_id}' \
--header 'API_KEY: <value>' --header 'filename: <value>' \
--form 'file=@"<full_path_to_asset>"'
```

- Update a specific asset, using its ID - {HOST}/assets/{asset\_id}

With this PUT request, it is possible to update an already stored asset. The asset's ID which is at the end of the endpoint, determines which asset should be replaced by the provided asset. As in the uploading, the asset should be uploaded as form-data with the key "asset" and the headers of the request have to contain provider's API key. Note that the users can only update the assets provided by themselves (except for administrators).

The following is an example of the request in cURL:

```
curl --request PUT '{HOST}/assets/{asset_id}' \
--header 'API_KEY: <value>' \
--form 'file=@"<full_path_to_asset>"'
```

- Delete a specific asset, using its ID - {HOST}/assets/{asset\_id}

A DELETE request to this endpoint has as a result the deletion of a specific asset, using its ID in order to find it. This endpoint is restricted and thus, users' API keys must be included in the headers of the request. Note that an asset can be deleted only by its provider and the administrators.

The following is an example of the request in cURL:

```
curl --request DELETE '{HOST}/assets/{asset_id}' --header 'API_KEY: <value>'
```

- Delete all assets - {HOST}/assets/all

This DELETE 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 users' API keys and it is only available to administrators.

The following is an example of the request in cURL:

```
curl --request DELETE '{HOST}/assets/all' --header 'API_KEY: <value>'
```

## 4 Validation scenarios

This section presents some validation scenarios regarding the functionalities of the back-end component of the marketplace in combination with the endpoints described in the previous sections.

Specifically, the scenarios that will be presented are how users can be authenticated to the back-end, how they can upload a new description and how they can upload an asset for the description they created in the second scenario.

It is noted that all scenarios will be done exclusively using the back-end. Similar scenarios, but using the front-end, will be presented in deliverable D8.5 "IoT and Blockchain Solutions Marketplace - I".

### 4.1.1 Authentication Scenario

We consider a registered user in the INFINITECH Marketplace with username "test\_user" and password "test\_password". The user, as described in section 3.2.1, in order to be authenticated, needs to send a POST HTTP request to the endpoint "{HOST}/accounts/users/authentication". In the body of the request, the user should add its credentials in the following format:

```
{ "username": "test_user", "password": "test_password" }
```

Thus, according to the description of the endpoint, the user should send the following request (using cURL):

```
curl --request POST '{HOST}/accounts/users/authentication' \
--header 'Content-Type: application/json' \
--data-raw '{ "username": "test_user", "password": "test_password" }'
```

The response of this request, if everything is fine (e.g. correct credentials), will be a temporary code that acts as an API key. Through this API key, the user is able to use the rest functionalities of the marketplace.

### 4.1.2 Upload Description Scenario

In this scenario, we will use the user of the previous scenario and we consider that the user followed the above scenario in order to be authenticated to the marketplace (i.e. has an API key).

The same user wants to upload a new asset / solution to the Marketplace. In this case, the first step that need to do, is to upload / create a new description to the marketplace.

All descriptions have a specific schema and should be filled in accordingly. Using the front-end this is done automatically through the corresponding forms, but in case the request is made in the back-end, the description must be created by the user.

As described in section 3.2.2, the user must add this description in JSON format as raw data to the body of the request. We suppose that the user wants to upload the description for the already uploaded description for the Python Notebook "DeepVaR" ([https://marketplace.infinitech-h2020.eu/infi\\_assets/deepvar-value-at-risk-prediction-leveraging-deep-learning](https://marketplace.infinitech-h2020.eu/infi_assets/deepvar-value-at-risk-prediction-leveraging-deep-learning)). The user - provider prepares the corresponding description as follows:

```
{
  "owner": "INNOV-ACTS LTD", "contact": "...", "type": "Notebook", "simpleComposite": "simple",
  "description": "This component both explains and implements the DeepVaR, which is a Value-at-Risk model based on deep neural networks and Monte Carlo simulations.",
  "name": "DeepVaR: Value-at-Risk prediction leveraging Deep Learning",
  "availability": "INFINITECH", "fieldOfUse": [ "Finance", "Risk Management" ], "comments": null
}
```

In order to upload this description to the marketplace, the user should send a POST request to the endpoint “{HOST}/descriptions/{collection}” where the {collection} will be the value “notebooks”. Also, the user should include the API key that got as a response during the authentication (previous scenario).

The following is an example of the corresponding request in cURL:

```
curl --request POST '{HOST}/descriptions/notebooks' \
--header 'API_KEY: <value>' --header 'Content-Type: application/json' \
--data-raw '{
  "owner ": "INNOV-ACTS LTD", "contact": "...", "type": "Notebook", "simpleComposite": "simple",
  "description ": "This component both explains and implements the DeepVaR, which is a Value-at-
  Risk model based on deep neural networks and Monte Carlo simulations.",
  "name": "DeepVaR: Value-at-Risk prediction leveraging Deep Learning",
  "availability": "INFINITECH", "fieldOfUse": [ "Finance","Risk Management" ], "comments": null
}'
```

The response of this request will be the ID of the new description. This ID will be used in the next scenario, in order to upload the description’s asset.

### 4.1.3 Upload Asset Scenario

This scenario extends the previous scenarios. It uses the API key of the user that has been authenticated as per the first scenario, and it uses the ID of the description that uploaded in the second scenario. In this scenario, the same user wants to upload the asset for the description of DeepVaR Notebook.

The user has a compressed file (.zip) to upload which need to be linked with a specific description.

Thus, as described in section 3.2.3, the user should send a POST request to the next endpoint “{HOST}/assets/{description\_id}” where the “description\_id” is the ID that got as a response in the second scenario, during the uploading of the description to which the asset will be linked. Moreover, the user should add to the headers of the request a) the API key from the first scenario and b) the asset’s filename (e.g. deepvar.zip).

The following is an example of the scenario’s request in cURL:

```
curl --request POST '{HOST}/assets/{description_id}' \
--header 'API_KEY: <value>' --header 'filename: deepvar.zip' \
--form 'file=@"/deepvar.zip"'
```

The response of the request is the new asset’s ID and the result, except for the storing of the asset, is the connection between the asset and the description.



## 5 Conclusions

The marketplace is considered the main driver for assets utilization among members of the INFINITECH ecosystem, as well as the main artefact for exploitation and sustainability of the project. As the specifications and detailed architecture of the INFINITECH marketplace have been provided in past deliverables, and in specific D8.1 and D8.2, this document provides a brief overview of the marketplace's specifications and architecture, thus giving focus to the accommodated assets, developed REST API endpoints descriptions and exploitation potentials.

The marketplace offers ready-to-use solutions covering a wide range of modern business and technical needs, focusing, as most pilots of INFINITECH project, on the Finance and Insurance sectors. Of course, most assets are related to Big Data and AI techniques and algorithms, with a variety of datasets, experimentation results and models being already available at the marketplace. Other than ready-to-use algorithms, frameworks and combined solutions, some assets offer state-of-the-art IoT and Blockchain solutions.

Additionally, the INFINITECH marketplace aspires to become a prominent multi-sided market platform and VDIH and continue adding value to practitioners, organizations and communities of the Finance and Insurance sector long after the INFINITECH Project has been completed.

Finally, this document and the progress presented within, is well aligned with the project's goals and overall strategy, awaiting additional assets to be populated with the pilot's development progress while achieving related objectives and KPIs as describes in the tables below.