D4.5 Guidelines and Instructions for PLUGGY Apps

Deliverable number	**D4.5**
Deliverable title	**Guidelines and Instructions for PLUGGY Apps**
Nature[1]	**R**
Dissemination Level [2]	**PU**
Author (email) Institution	Luis Molina Tanco (lmtanco@uma.es) UMA Peter Smatana (peter.smatana@tuke.sk) TUK Daniel González Toledo (dgonzalezt@uma.es) UMA Luna Herruzo Torrico (mlunaht@uma.es) UMA
Editor (email) Institution	Luis Molina Tanco (lmtanco@uma.es) University of Málaga.
Leading partner	**University of Málaga**
Participating partners	**Technical Universisty Kosice**
Official submission date:	**30/11/2019**
Actual submission date:	**11/11/2019**

[1] **R**=Document, report; **DEM**=Demonstrator, pilot, prototype; **DEC**=website, patent fillings, videos, etc.; **OTHER**=other

[2] **PU**=Public, **CO**=Confidential, only for members of the consortium (including the Commission Services), **CI**=Classified, as referred to in Commission Decision 2001/844/EC

Modifications index	
08/11/2019	Initial Release

This work is a part of the PLUGGY project. PLUGGY has received funding from the European Union’s Horizon 2020 research & innovation programme under grant agreement no 726765. Content reflects only the authors’ view and European Commission is not responsible for any use that may be made of the information it contains.

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This document gives guidelines and instructions for future developers of the PLUGGY pluggable applications and plugins. It condenses not only the development experience gained by the PLUGGY consortium, but also that of the external developers who participated in the App Challenge during the project and greatly contributed to the improvement of the PLUGGY Application Programming Interface (API).

The PLUGGY architecture was devised to be modular and to provide an API for the future development of applications, which will provide novel ways to experience and curate cultural heritage content. The PLUGGY consortium provides developers with a set of resources that allow the unassisted development of such applications, including:

- The Guidelines and Instructions for PLUGGY apps (this document).
- A set of source code examples at https://isense-gitlab.iccs.gr/pluggy_public/pluggy-examples
- The online API documentation at https://pluggy.eu/api/doc/

This document provides an overview of the PLUGGY Architecture and Data Model. Practical help comes in the form of code snippets which illustrate basic functionality of the API –such as Search or Asset[3] Creation–, and more advanced example use cases which show how to create different plugins and applications. It also provides instructions on how to become a PLUGGY developer and how to register applications to the PLUGGY platform.

These resources should be sufficient to start unassisted development of pluggable applications. However, PLUGGY is evolving and so will its API. The online documentation may be more updated that this document, and developers who seek updated support can reach the consortium at platform@pluggy-project.eu.

[3] An Asset is the basic cultural content unit in PLUGGY. It is a container for a media file such as, for example, an image or an audio file, together with some metadata which describes what the file is about and how its owner licenses its use.

1 Introduction

1.1 WHO IS THIS DOCUMENT FOR?

This document is for software developers outside the PLUGGY consortium who want to contribute to the PLUGGY platform. It condenses not only the development experience gained by the members PLUGGY consortium, but also that of the external developers who participated in the App Challenge during the project, and greatly contributed to the improvement of the **PLUGGY REST API** (the API, in the remaining of this document).

PLUGGY is a virtual place for sharing and bringing culture closer to everyone. If you are not familiar with PLUGGY, there are two main entry points: the PLUGGY project webpage at https://pluggy-project.eu/ and the **PLUGGY Social Platform** at https://pluggy.eu.

All registered users on the PLUGGY social platform can apply for a developer account. As a developer, you can create enhanced and non-traditional experiences of content for all users of PLUGGY. This document is your starting point to creating your application and make it available via the PLUGGY social platform to a wide audience.

The remaining of this document introduces the API and gives example use cases to help you get started:

- Section 2 **THE PLUGGY API** describes the PLUGGY architecture, datamodel, and the step-by-step procedure to become a developer and start using the API.
- Section 3 **EXAMPLE USE CASES** gives further insight into the details of the API by looking at typical examples you may need for your application, like searching, creating content, or managing social interaction notifications.

There are two **Appendices**, meant as reference:

- **API Documentation** is the reference document for endpoints, methods and models of the PLUGGY REST API.
- **Style Guidelines for Plugins** is directed to developers that want to use the PLUGGY Social Platform design in their plugin applications. This includes a recommended Colour Palette, a set of Icons, a Font and other basic visual elements.

1.2 WHAT CAN I DEVELOP WITH PLUGGY?

As a developer, you can create two types of applications in PLUGGY:

- **Plugins** are integrated within the PLUGGY social platform.
- **Applications**: standalone desktop, mobile or web applications.

Both types of applications connect to the **PLUGGY Back-end** via the API to access and create content according to the **PLUGGY Data Model**, which is explained later in this document.

1.3 WHERE CAN I FIND MORE RESOURCES?

1.3.1 Online resources[4]

- As previously mentioned, this document assumes you are familiar with the PLUGGY project. If not, visit https://pluggy-project.eu.
- To register as a PLUGGY user and a PLUGGY developer, visit the PLUGGY Social Platform at https://pluggy.eu.
- If you want to jump directly into development please visit this page for more information about the API: https://pluggy.eu/api.
- The API specification can be found at https://pluggy.eu/api/doc/.
- The source code of the examples presented later in this document can be found at: https://isense-gitlab.iccs.gr/pluggy_public/pluggy-examples.
- The PLUGGY Developers Videos offer a visual overview of how to become a PLUGGY developer: https://vimeo.com/showcase/5872393
- Complex applications and plugins were developed during the PLUGGY project, which can serve as inspiration for your development. Part or all of their source code is open can be found at
  - Gaming Application: https://github.com/xteamsoftware/PLUGGY
  - 3D Sonic Application: https://github.com/lpicinali/PlugSonic-soundscape
  - Geolocation Application: https://isense-gitlab.iccs.gr/george.chatzigeorgakidis/pluggy-pins.
  - Augmented Reality Application: https://isense-gitlab.iccs.gr/dgonzalezt/pluggy-ar-app
- The source code of the PLUGGY social platform, the main front-end to the API, is also open, and can be found at https://isense-gitlab.iccs.gr/PLUGGY/pluggy-web.
- The source code of the back-end is also open, and can be found at https://isense-gitlab.iccs.gr/PLUGGY/pluggy-core.
- If you find a problem with the API or you have a suggestion please consider creating an issue at https://isense-gitlab.iccs.gr/PLUGGY/pluggy-core/issues

1.3.2 Relation with other PLUGGY Deliverables

The PLUGGY project has generated a series of documents which can help you gain insight into the whole PLUGGY project, creating applications which are in line with the project's vision of cultural heritage.

- **D3.2** and **D3.3** specify the main front-end to the PLUGGY API, composed by two sub-systems: the Social Platform and the Curatorial Tool. Both constitute the front-end of a single website and thus the frontiers between them are blurred. D3.2 describes the Social Platform, and D3.3 describes the Curatorial Tool. Curation of content lies outside the scope of this deliverable, but D3.3 is fully dedicated to it.

[4] All links are correct at the writing of this document, but please bear in mind that they might vary in the future.

- **D4.1, D4.2, D4.3 and D4.4** describe the PLUGGY Apps, which also create and access the digital content created and reused in the PLUGGY Website. The apps have been extended to become, in some cases, full creation & experience toolchains, implementing part of the PLUGGY website front-end, so after the Social Platform and the Curatorial Tool, they are the most complex examples of use of the API at the writing of this document.
- **D3.1** "Architecture Specification" establishes the System Architecture, which includes the organisation in logical components of the system developed in PLUGGY. It contains specifications for the main modules of the system, including the Social Platform and the Curatorial Tool.
- **D3.4** "Content Management System" is a brief overview of the back-end which provides the serviced implemented in the API.

2 The PLUGGY API

2.1 The PLUGGY Architecture

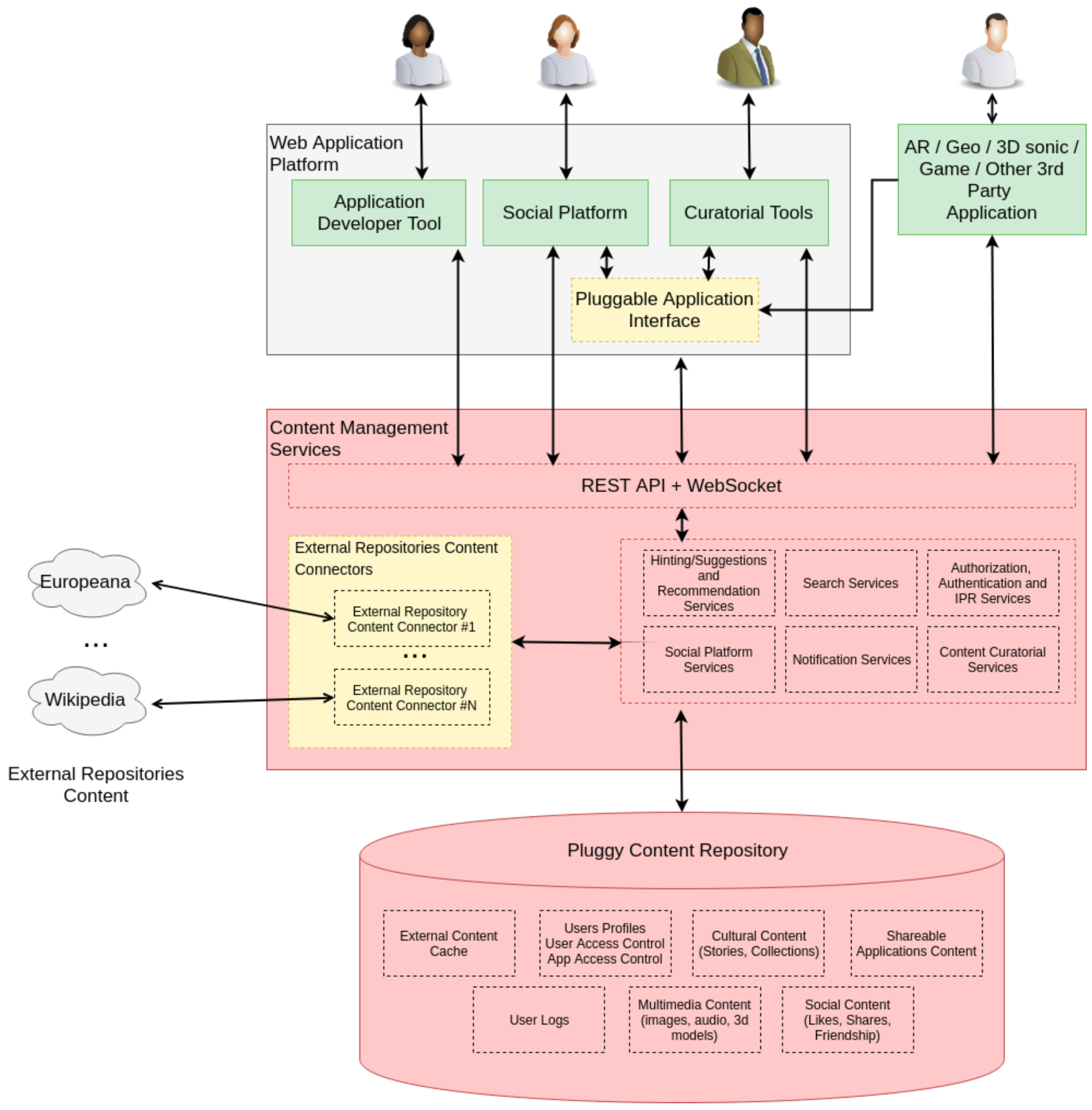

Web Application Platform
Application Developer Tool
Social Platform
Curatorial Tools
AR / Geo / 3D sonic / Game / Other 3rd Party Application
Pluggable Application Interface
Content Management Services
REST API + WebSocket
External Repositories Content Connectors
External Repository Content Connector #1
…
External Repository Content Connector #N
Hinting/Suggestions and Recommendation Services
Search Services
Authorization, Authentication and IPR Services
Social Platform Services
Notification Services
Content Curatorial Services
Europeana
…
Wikipedia
External Repositories Content
Pluggy Content Repository
External Content Cache
Users Profiles User Access Control App Access Control
Cultural Content (Stories, Collections)
Shareable Applications Content
User Logs
Multimedia Content (images, audio, 3d models)
Social Content (Likes, Shares, Friendship)

Figure 1. The PLUGGY Architecture

Figure 1 shows the PLUGGY back-end provides a REST API to access all functionality for the different front-ends and applications (green boxes in the figure): the Social Platform, the Curatorial Tools, the applications developed within the PLUGGY Project and the 3rd party applications that you might be planning to develop.

The common back-end allows to create content with one application which can be later experienced by the other applications and front-ends. The **Web Application Platform** is modular and provides common functionality to the Social Platform, Curatorial Tools and the Application Developer Tool, such as logging and security. This common functionality is also provided to 3rd party plugins (not shown in the figure), which you can also develop.

The back-end is actually composed of two modules (red boxes in the figure), the Core PLUGGY Components, which are the Content Management Services and the Content Repository.

The PLUGGY **Content Management Services** contain all the software components needed to process, store and serve information to external users via the REST API. The components are:

- External Repository Content Connectors: providing a unified interface to external libraries, including **Europeana**[5].
- Hinting/Suggestions and Recommendation Services
- Search Services, including basic (all fields text search) and advanced content search which can be filtered by several attributes such as geolocation, metadata, etc.
- Authorization, Authentication and Intellectual Property Services
- Social Platform Services: services used for social interaction between users and users' content
- Notification Services: user notification when new content is created, based on user preferences
- Content Services: full CRUD operation for PLUGGY content

All these components access the PLUGGY **Content Repositories** which store all the information, according to a data model which is described in the next Section.

Finally, the yellow boxes in Figure 1 represent the two basic **extensibility** points of the PLUGGY platform:

- The Pluggable Application Interface: allows users to explore and create content of different types from the ones supported by the Core PLUGGY applications and front-ends.
- External Repository Content Connectors: allow users to reuse content from existing external content repositories.

2.2 THE PLUGGY DATAMODEL

There is a data model that you need to understand as a developer to fully exploit the PLUGGY API. This section gives an overview of the main entities in the Data Model. For

[5] As this document is writing, more libraries are being integrated such as Wikipedia and the British Museum Library.

more information explore Figure 2, and the rest of this document, especially: **Annex: API Documentation**.

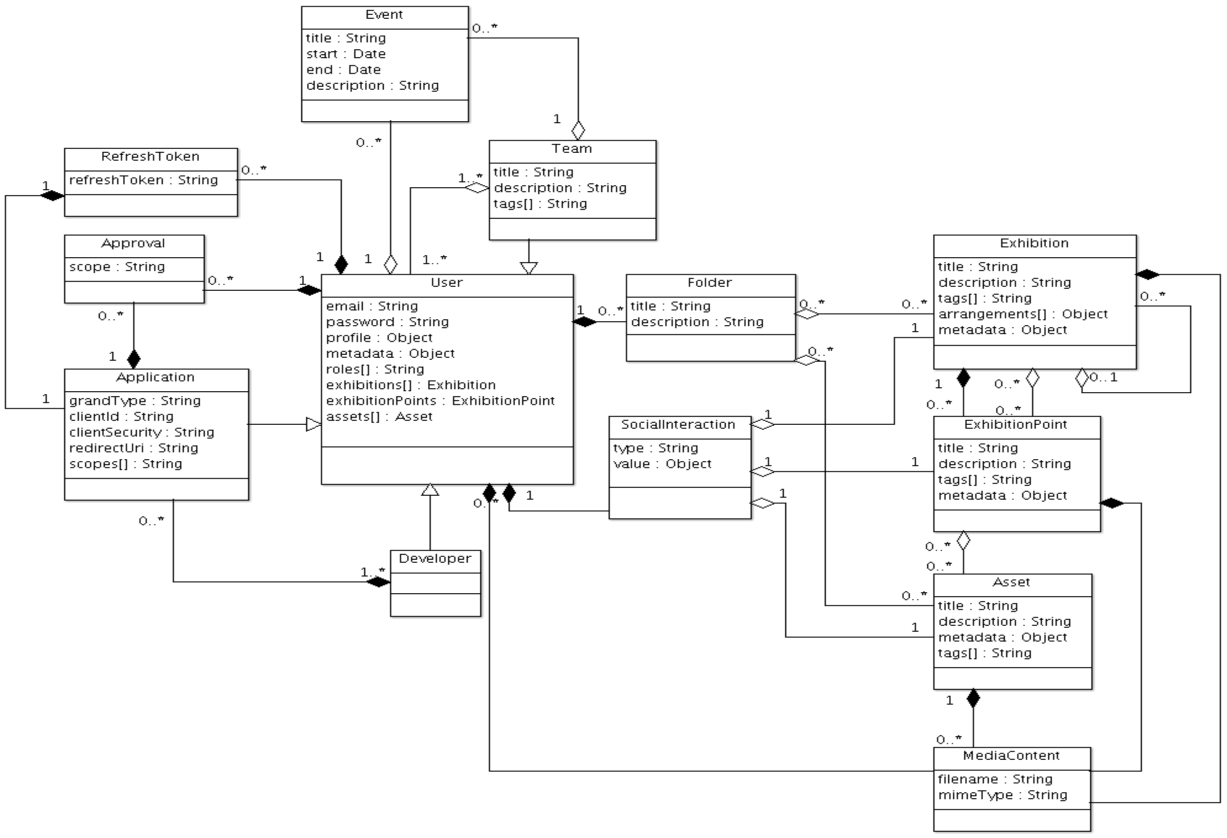


Figure 2. Overview of PLUGGY Data Model.

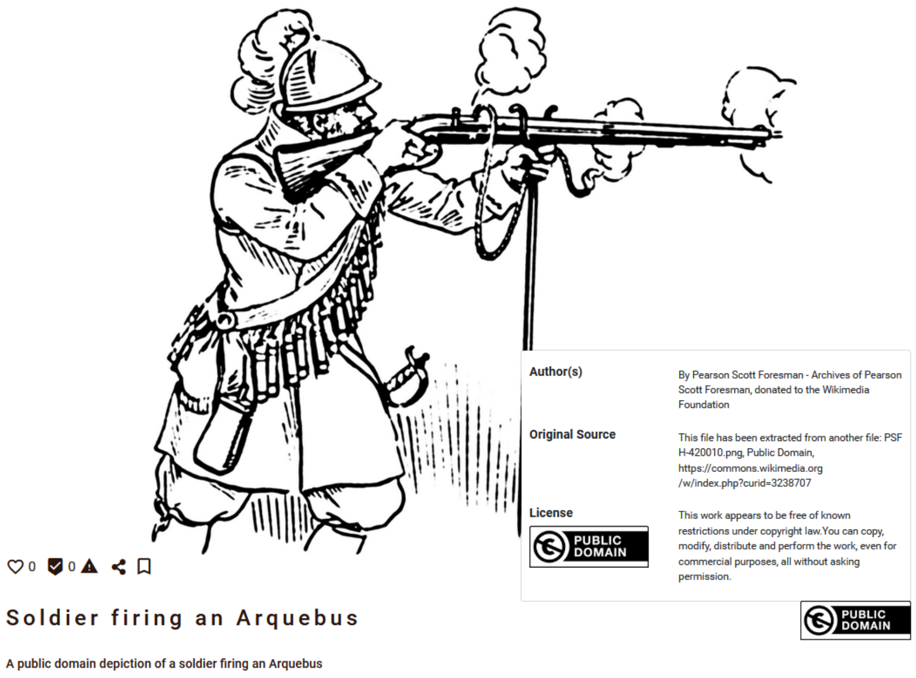
2.2.1 Content

Content is central in PLUGGY. Thus the main, basic concepts in the PLUGGY Data Model are the content entities Asset and Exhibition.

Asset (https://beta.pluggy.eu/api/doc/#/Asset_CRUD): is the elementary unit of content in PLUGGY. An Asset is a media file[6] with an identified owner, a title, a description, a set of tags and a license, which specfies how this file can be reused. The Asset can be used as a digital representation of cultural heritage artifact, tangible or intangible. The media file can be text, image audio, 3d model or any type of binary data.

The PLUGGY API provides basic CRUD methods to manipulate Assets, plus some advanced functionality for social interaction, importing from external sources, etc.

[6] Or set of files, in special cases such as 3D Model Assets, which necessarily span more than one file (mesh, material and texture files).



Soldier firing an Arquebus

A public domain depiction of a soldier firing an Arquebus

Author(s)	By Pearson Scott Foresman - Archives of Pearson Scott Foresman, donated to the Wikimedia Foundation
Original Source	This file has been extracted from another file: PSF H-420010.png, Public Domain, https://commons.wikimedia.org /w/index.php?curid=3238707
License	This work appears to be free of known restrictions under copyright law.You can copy, modify, distribute and perform the work, even for commercial purposes, all without asking permission.

Figure 3. An Image Asset in PLUGGY.

Exhibition (https://beta.pluggy.eu/api/doc/#/Exhibition_CRUD): cultural heritage stories curated by users of PLUGGY using one or more Assets. PLUGGY provides you with a set of curatorial tools for creation of exhibitions of several types: Media Stories, Timelines, Geolocated Tours, Augmented Reality Exhibitions and Games. You are welcome to create your own curatorial tool or viewer to engage users in ways not imagined by the PLUGGY project.

As with Asssets, the PLUGGY platform provides basic CRUD methods for manipulation with the Exhibitions and some advanced functionality for social interaction, following and notification functionalities.

Exhibition Point (https://beta.pluggy.eu/api/doc/#/Exhibition_Point_CRUD). Exhibition Points link Exhibitions and Assets. An Exhibition Point is the usage of an Asset in an Exhibition, and an Exhibition is composed of one or several Exhibition Points. For example, *Events* are the Exhibition Points of *Timeline*s, and *Chapters* are the Exhibition Points of *Media Stories*. Equally to Assets and Exhibitions, the PLUGGY provides basic CRUD methods for manipulation of Exhibition Points.

Folders (https://beta.pluggy.eu/api/doc/#/User_Folders): Users can organise their own content or content they have found in Pluggy using Folders. Folders are always private and serve both as folders per se and as bookmarks.

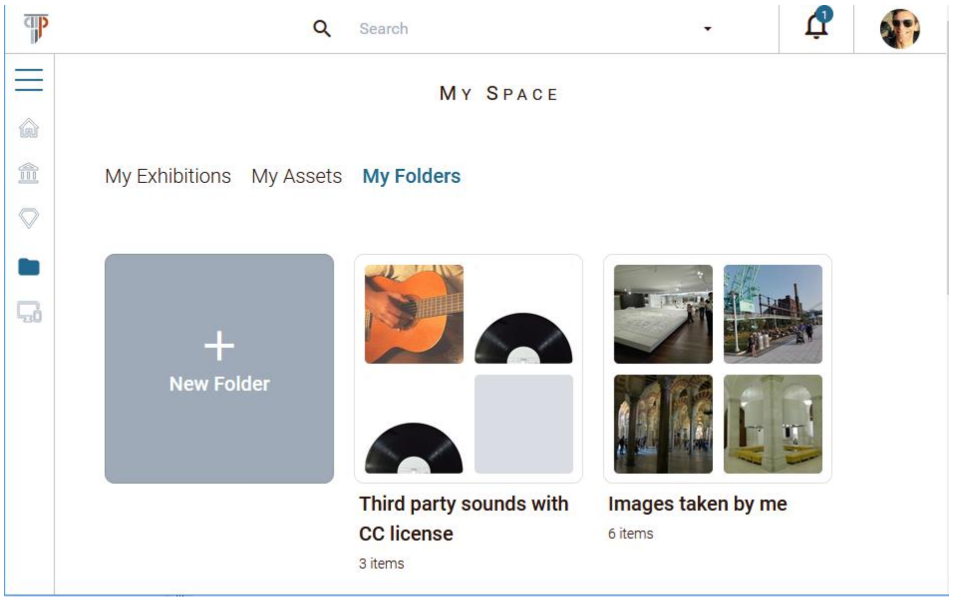


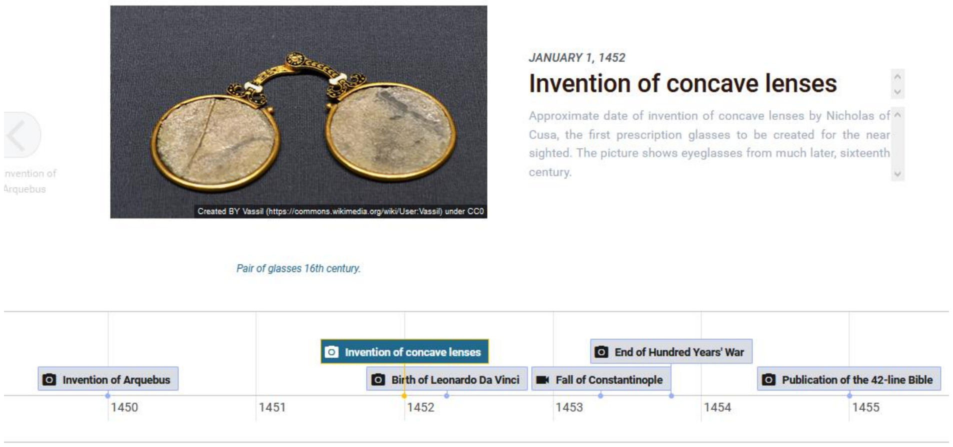
Figure 4. Folders, as shown privately to a user of the Social Platform in the "My Space" section.

2.2.2 Users and Social Interaction

Users and Teams (https://beta.pluggy.eu/api/doc/#/User_CRUD): Users of Pluggy own Assets and Exhibitions. There are three types of Users: Individual Users, Team Users and Applications. A Team (https://beta.pluggy.eu/api/doc/#/Team_CRUD) owns Assets and Exhibitions in the same way a normal user does. Within a Team, a user can have the role of Member or the role of Admin. (See Section 2.3 for Applications).

Social Interaction: Users can interact socially with other users by a one-way following system. (https://beta.pluggy.eu/api/doc/#/User_Social). Users can also interact socially with content, in terms of Assets and Exhibitions, which can be *liked*, commented or reported by other users. (https://beta.pluggy.eu/api/doc/#/Asset_Social), (https://beta.pluggy.eu/api/doc/#/Exhibition_Social).

Notifications (https://beta.pluggy.eu/api/doc/#/User_Notifications) are sent in real time via WebSockets (see Section 3.1.3) and linked to Users. If you are a user, the back-end creates notifications for you when a user followed by you creates new content, a User likes your content, or a Users starts following you.


JANUARY 1, 1452

Invention of concave lenses

Approximate date of invention of concave lenses by Nicholas of Cusa, the first prescription glasses to be created for the near sighted. The picture shows eyeglasses from much later, sixteenth century.

Created BY Vassil (https://commons.wikimedia.org/wiki/User:Vassil) under CC0

Pair of glasses 16th century.

nvention of Arquebus
Invention of concave lenses
End of Hundred Years' War
Invention of Arquebus
Birth of Leonardo Da Vinci
Fall of Constantinople
Publication of the 42-line Bible
1450
1451
1452
1453
1454
1455

Figure 5. A Timeline Exhibition in PLUGGY

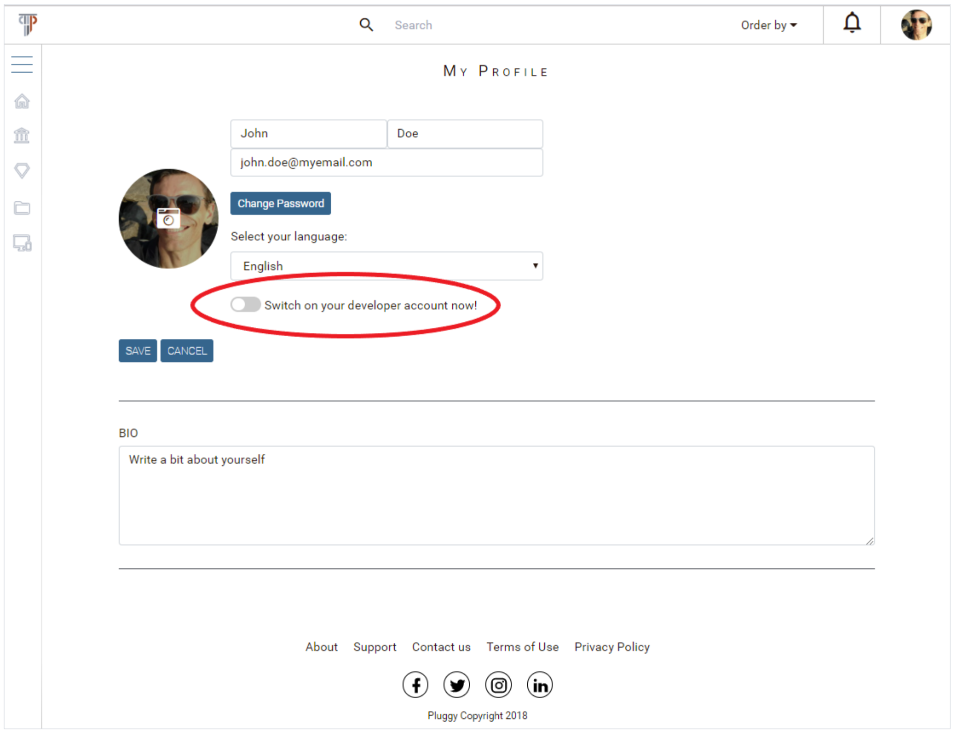
2.3 USING THE API STEP BY STEP

The following sections explain the first steps to become a developer of the PLUGGY applications, giving an overview of what is involved in creating applications and making them available to the PLUGGY users[7].

2.3.1 Becoming a developer

Start by registering yourself to the PLUGGY platform (http://pluggy.eu > *Register*). Click on your profile picture and then *My Profile*. In the Edit Section of your profile, choose *Advanced Settings*. This will let you apply for a developers' account (see Figure 6). Your request will be automatically accepted, but to apply changes you have to log out from the PLUGGY platform and log back in.

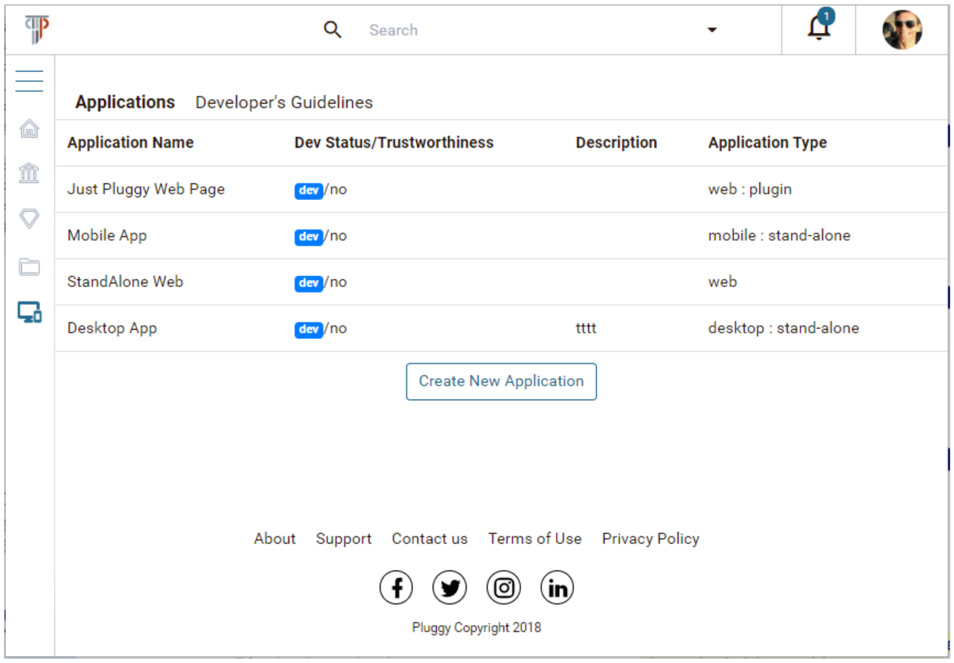
[7] For further step-by-step instructions for each type of application, whatch the videos at https://vimeo.com/showcase/5872393.



John | Doe
john.doe@myemail.com
Select your language: English
Switch on your developer account now!
BIO
Write a bit about yourself

Figure 6. Applying for a developer's account.

This will give you access to the list of the Applications which you own, or owned to a Team to which you belong (Figure 7). The next Section describes how to register your application and connect it to the PLUGGY Platform.



Application Name	Dev Status/Trustworthiness	Description	Application Type
Just Pluggy Web Page	dev/no		web : plugin
Mobile App	dev/no		mobile : stand-alone
StandAlone Web	dev/no		web
Desktop App	dev/no	tttt	desktop : stand-alone

Figure 7. List of your Applications.

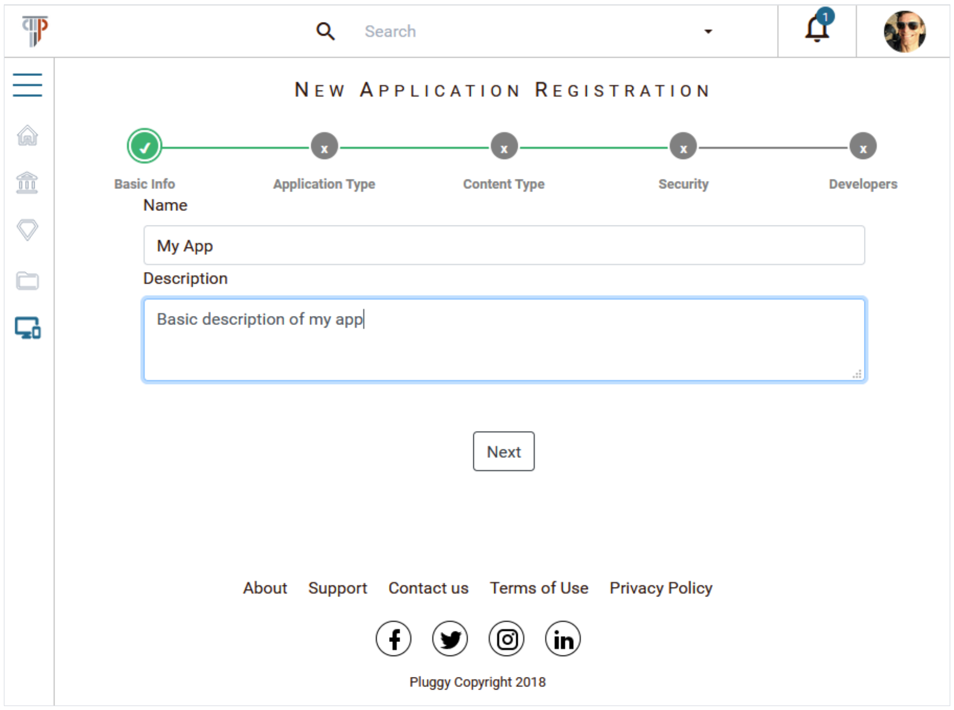
As a developer, you can create two types of applications:

- **Plugins** are integrated within the PLUGGY Social Platform front-end.
- **Applications** are stand-alone. They can be Desktop, Mobile or Web Applications, and they integrate with the PLUGGY back-end.

To create a Plugin or an Application, press the *Create New Application.*

2.3.2 Specifying Basic Information

To create a Plugin or an Application, press the *Create New Application* button (Figure 7) in your developer zone and provide the Basic Info required.



Name

My App

Description

Basic description of my app

Figure 8. Step 1 - basic info of your application.

2.3.3 Specifying your Application Type

The next screen will let you specify the type of application that you are registering.

A basic principle of PLUGGY plugins is that an externally hosted web application accessible via https can be displayed within the PLUGGY platform (see)

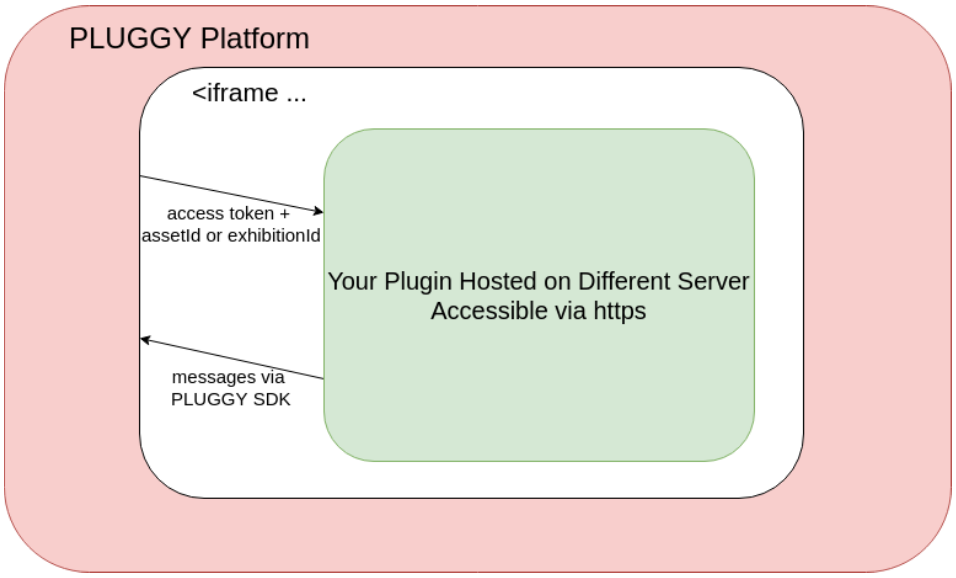
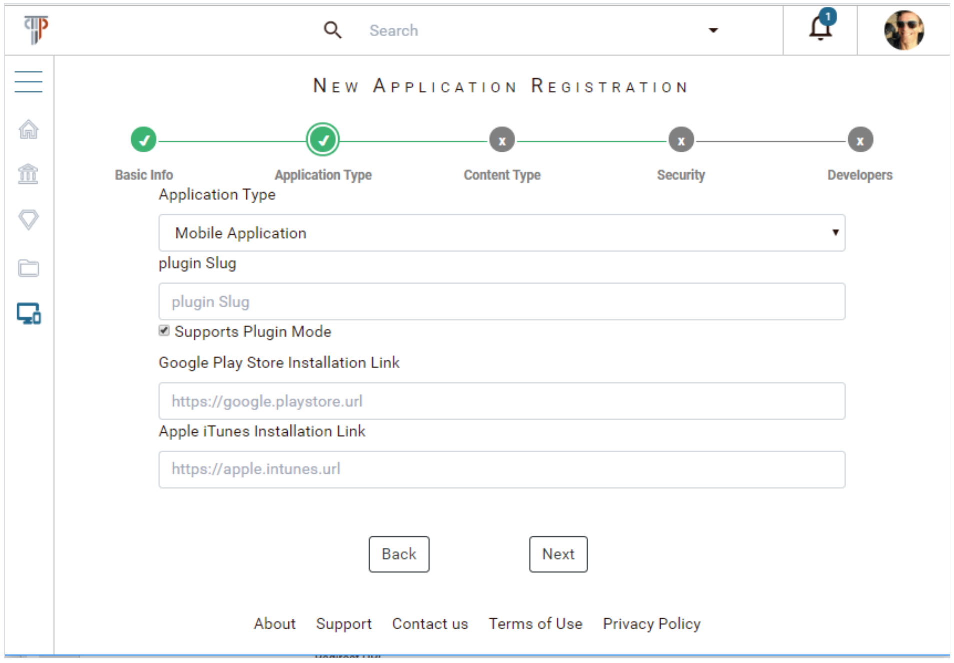

PLUGGY Platform
<iframe ...
access token + assetId or exhibitionId
messages via PLUGGY SDK
Your Plugin Hosted on Different Server Accessible via https

Figure 9. External plugin accessible via PLUGGY

If your application supports **Plugin Mode**, you can specify a Plugin Slug, a readable URL for your application. Then, for each type of application ou can specify a number of URLs which will allow you to **seamlessly plug your application** in PLUGGY :

- **Web Application:** You have to specify the URL of your Web Application.
- **Mobile Application:** You have to specify the URL of Google Play Store and/or Apple iTunes installation links.
- **Desktop Application:** You have to specify an installation URL.


NEW APPLICATION REGISTRATION

Basic Info — Application Type — Content Type — Security — Developers

Application Type: Mobile Application

plugin Slug: plugin Slug

☑ Supports Plugin Mode

Google Play Store Installation Link: https://google.playstore.url

Apple iTunes Installation Link: https://apple.intunes.url

Figure 10. Step 2 - Specifying the type of application and associated plug URLs. Figure shows the case of a mobile application.

2.3.1 Specifying Content Type and Reuse Policy.

The next screen allows you to specify exactly what your application does with content.

First you have to specify information regarding **intellectual property**. This will affect what assets you will get back by default when your application does searches on the back-end. Here, you have to specify if your application will:

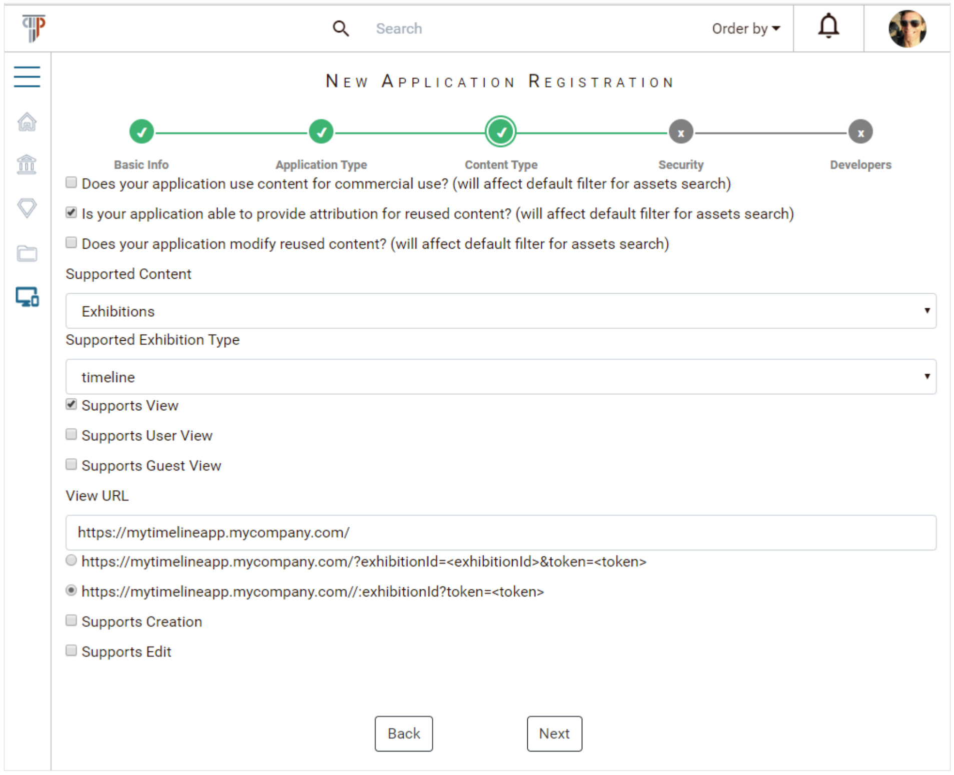
- use content for **commercially**,
- be able to provide **attribution** for reused content, and
- **creates derivative work** (i.e. modifies) or not the reused content.

For example, if you mark that your application uses content for commercial use, a default Asset search by your app will give back only Assets in the Public Domain, CC-BY or CC-BY-SA. It will not give back any of the non-commercial reuse Assets.

In the same screen, you have to specify the following attributes of the type of Content your Application will manage:

- Supported Content, which can be **Exhibitions**, **Assets** or Unspecific. Depending on this, you will have to specify which type of Exhibition or Asset you support.
- Supported actions on content, which can be **View**, **Creation** and/or **Edit**.

For each supported action, you will have to provide a URL which specifies how you will get the content ID passed on to your Application. Figure 11 shows an example of an Application that supports viewing of Timelines.



- [ ] Does your application use content for commercial use? (will affect default filter for assets search)
- [✓] Is your application able to provide attribution for reused content? (will affect default filter for assets search)
- [ ] Does your application modify reused content? (will affect default filter for assets search)

Supported Content: Exhibitions

Supported Exhibition Type: timeline

- [✓] Supports View
- [ ] Supports User View
- [ ] Supports Guest View

View URL: https://mytimelineapp.mycompany.com/

- ○ https://mytimelineapp.mycompany.com/?exhibitionId=<exhibitionId>&token=<token>
- ● https://mytimelineapp.mycompany.com//:exhibitionId?token=<token>

- [ ] Supports Creation
- [ ] Supports Edit

Figure 11. Step 3 - An example of an application that supports viewing of Timeline exhibitions.

2.3.2 Security

PLUGGY is adopting the **OAuth2**[8] scheme for secure access to users' credentials. Users can enter credentials only via trusted PLUGGY's authentication interface and every application which want to access user's content has to be approved.

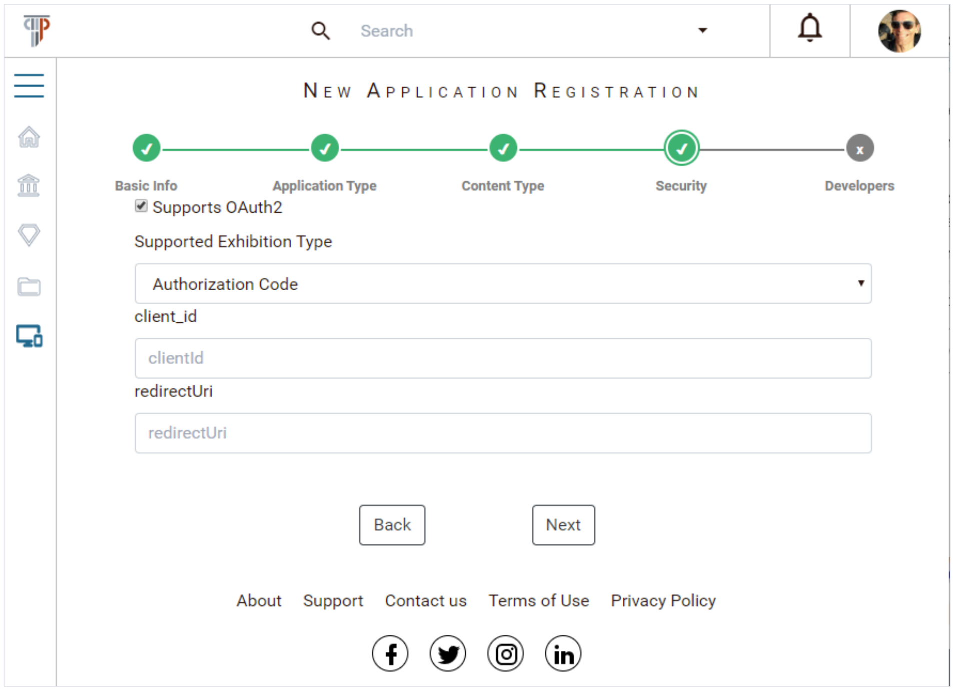
In this screen you must choose Supports OAuth2, and Authorization Code. You can then specify a client ID and a redirect Uri.

Finally, in the last step you can specify other Pluggy Users that will also be developers of your application, your **development team**.

This is all you need to register your application.

All this information will arrive to the PLUGGY administrators. The following section illustrates the different statuses that your application will go through before it is declared a trusted application.-

[8] https://tools.ietf.org/html/rfc6749. PLUGGY will be compatible with other platforms that support OAuth2 such as Google and Facebook. Users will be able to use their existing credentials for access to the PLUGGY platform.



☑ Supports OAuth2

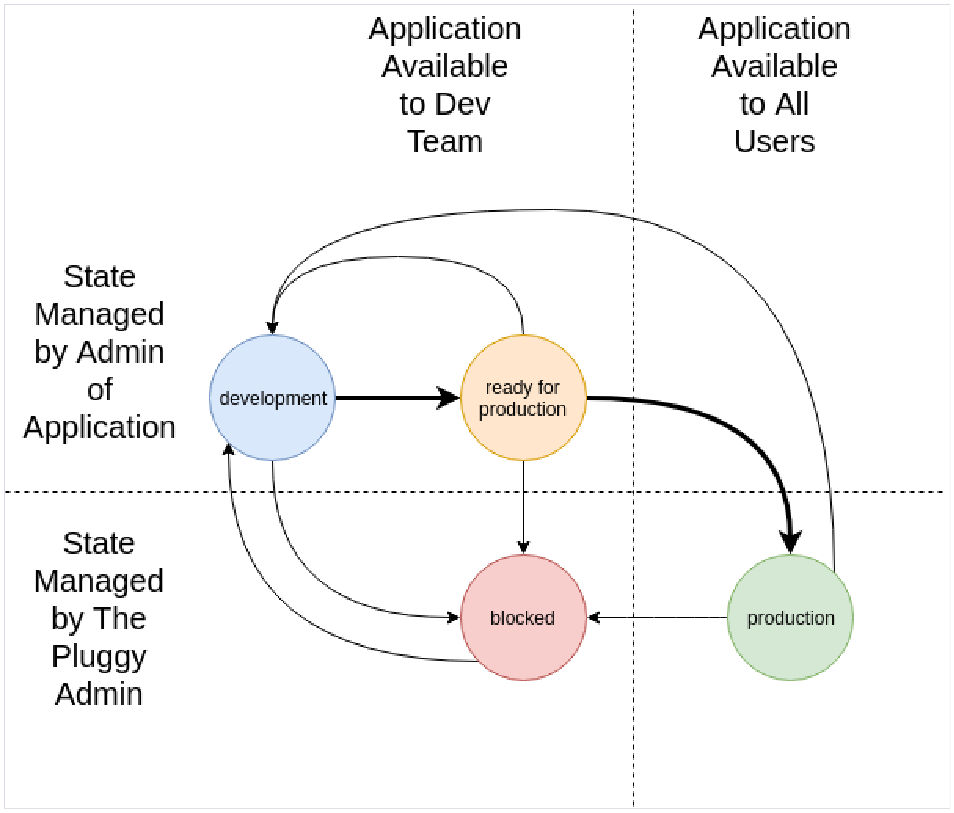
Supported Exhibition Type: Authorization Code

client_id: clientId

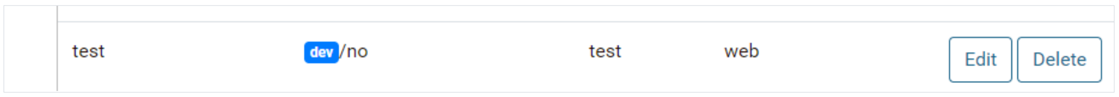
redirectUri: redirectUri

Figure 12 Step 5 - OAuth2 settings

2.3.3 Making your Application Available to PLUGGY Users

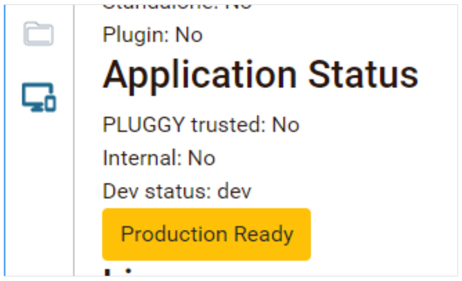

Application Available to Dev Team
Application Available to All Users
State Managed by Admin of Application
State Managed by The Pluggy Admin
development
ready for production
blocked
production

While the application is in development or ready for production, it will only be available to you and to the development team that Once you have developed your application, you can declare your application ready for production. For this, click on your Application in the Application Console (click on the line for your app):



test	dev/no	test	web	Edit Delete

This will take you to a summary page showing all the information about your application. Click on the *Production Ready* button under Application Status:


Plugin: No

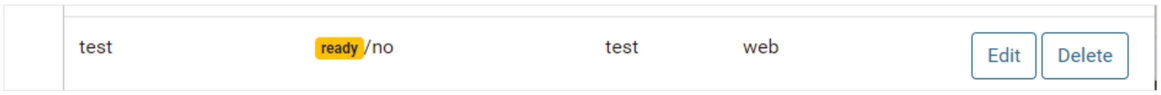
Application Status

PLUGGY trusted: No

Internal: No

Dev status: dev

This will advance your application status to the next stage.



test	ready/no	test	web	Edit Delete

The Pluggy Administrators will inspect your app and make it available for all users, or contact you if there are remaining issues to be solved.

3 Example Use Cases of the PLUGGY API

3.1 BASICS

Your application will be the **client** of PLUGGY REST API. All the API **endpoints** can be found at

https://pluggy.eu/api/v1/.

Even if future versions of the API are developed (e.g v2) this API will remain accessible. All the GET methods can be unauthenticated, but the POST, PUT and DELETE methods will require authenticated access.

3.1.1 Searching

GET methods avaliable via main endpoint for search, which is `/search`. Additional endpoints are `/assets`, `/exhibitions`, `/users`, `/teams` and `/applications`. Some examples follow (see https://pluggy.eu/api/doc for the full specification).

Assets.

Results are paginated. By default, 10 items will be returned. Use `&page=1&limit=10` to get the next page. Use `&sort=trending`, `&sort=recent` or `&sort=title` to get them in different orders.

```
/search?kind=ContentAsset
/search?kind=ContentAsset&sort=recent
```

If you just want all Assets you can just use:

```
/assets
```

The `/assets/{assetId}` endpoint allows you to get all the media and metadata in a given Asset

```
/assets/5dbace4242b0df9c2b1bae0e
/assets/5dbace4242b0df9c2b1bae0e/media/5dbace4242b0df9c2b1bae1b
```

Exhibitions.

Results are returned in the same fashion as Assets.

```
/search?kind=ContentExhibition
/search?kind=ContentExhibition&sort=recent
```

The endpoint can also return all exhibitions in which an Asset is used, given the Asset id, or all exhibitions win which an Asset is used, given the Exhibition id:

```
/search?exhibitionsForAsset=5c120d03fffb9430d682c277
/search?relatedExhibitions=5c120d03fffb9430d682c27f
```

If you just want all exhibitions you can use:

```
/exhibitions
```

The `/exhibitions/{exhibitionId}` endpoint allows you to get all the media and metadata in a given exhibition. See the Annex: API Documentation for further details.

Users & Teams:

Users can be searched by user id. All content created by a user can be retrieved by:

```
/search?user=5af19572ed0683578c1a6111
```

Users and Teams can also be searched by name using the general search `q=` and `&kind=UserPerson` or `&kind=UserTeam.`

```
/search?q=Joe&kind=UserPerson
```

Searches in External Libraries:

Searches in third party libraries are specified through `&repository=europeana`, `&repository=wikipedia`, etc. Results will not be paginated, so you need to use &page and `&limit` if you want to paginate them.

```
/search?q=mummy&page=0&limit=10&sort=trending&repository=europeana
```

3.1.2 Creating Content

PUT, POST and DELETE methods are available for authenticated users to create content at all levels, from media and metadata for Assets, to Exhibition Point content and metadata for Exhibitions.

The main endpoints to manage content in PLUGGY are `/assets`, `/exhibitions`, and `/exhibitions/{exhibitionId}/exhibition-points`. An example on how to create an Image Asset follows (see https://pluggy.eu/api/doc for the full specification).

Creating an Image Asset

A POST on the assets endpoint creates an asset. The best practice for PLUGGY is that one asset will contain one media file. A cover image can be added for every asset (not needed for Image Assets).

Thus the steps for creating an Image Asset would be:

1. Create asset with **POST: `/assets`**

   A series of optional metadata for the image Asset can be provided as payload:

```
{
    "location": {
        "geo": {
            "type": "Point",
            "coordinates": [36.76173250997905, -3.84477261453867],
            "zoom": 14
        }
    },
    "legal": {
        "licenseId": "",
        "license": "CC-BY",
        "isOwnWork": true,
        "source": "Pluggy",
        "author": "5c13924ca21c67804286af00",
        "authorKind": "UserPerson"
    },
    "public": true,
    "tags": ["Nerja","Nature","Cave","Málaga"],
    "mediaContent": [],
    "title": "Cueva de Nerja",
    "description": "Beautiful Natural cave in Nerja, Spain",
    "previewMedia": null,
    "coverImage": "",
    "creator": null,
    "type": "image"
}
```

2. Upload media content with **POST:** **`/assets/{assetId}/media`**

   When uploading media use "multipart/form-data". The name of the file input box should be "file".

3.1.3 Managing Notifications

Notifications are created per user automatically. For example, when any of the users followed by a given one creates new content. The back-end creates and stores this notifications, which can be retrieved in two ways: directly by the REST API or in real time, by connecting to a web socket.

Getting notifications by API

Getting the notifications can be done using the `/users/my` endpoint or the user id, but both require the user to be authenticated to the platform:

```
GET: /users/my/notifications
GET: /users/:userId/notifications
```

This will receive the 10 most recent notifications of the user:

```
{
    message: String,
    creator: GenericUser, // who created the notification
    owner: GenericUser, // receiver of the notification
    type: String, // ["warning", "info", "like", "follow", “unfollow”,
                  //  "comment", "contentUse",
                  //  "contentPublished","report"],
    referenceToContent: ContentGeneric, // content if it exists
  referenceToEvent: Event, // reference to event if exists
  internalLink: String,    // where the user will be redirected
  isRead: Boolean,
}
```

A client can mark notifications as read or unread:

```
PUT: /users/:userId/notifications/:notificationId/unread
PUT: /users/:userId/notifications/:notificationId/read
PUT: /users/:userId/notifications/allread
```

A client can receive notifications in real time. Please refer to https://isense-gitlab.iccs.gr/pluggy_public/pluggy-examples/blob/master/websocket/ for an example of how to connect to the API to listen websocket messages.

3.2 EXAMPLES

The repository https://isense-gitlab.iccs.gr/pluggy_public/pluggy-examples contains a series of simple examples using html, javascript, and the React framework in the more advanced examples. The examples can be used as an inspiration or guide for the integration with the PLUGGY Platform. All the examples have installation instructions and description in their root folder.

3.2.1 asset-plugin-collage

This is a web plugin that creates a Image Asset in the form of a collage created from 5 selected Image Assets. The example uses the React framework[9].

3.2.2 exhibition-plugin-gallery-creator

This is also a web plugin that creates a new virtual exhibition from 8 selected assets, which can be selected from a very simple image asset browser. This exhibition can be viewed by the next plugin. Uses the React framework.

3.2.3 exhibition-plugin-gallery-viewer

Web plugin that allows users to view virtual exhibitions created by the previous example, the exhibition-plugin-gallery-creator. Uses THREE.js[10].

3.2.4 pluggy-example-browser

Simple mobile application for browsing and "liking" the content of The PLUGGY Platform.

3.2.5 websocket

Simple example how to use websocket withing The PLUGGY Platform.

4 Conclusions

The modular architecture of PLUGGY and its API were designed to facilitate the development of innovative ways to create and experience cultural heritage.

This document provides guidelines and instructions for the development of applications connected to the PLUGGY platform. They are intended for software developers who want to extend PLUGGY in ways not yet imagined by the PLUGGY consortium. We have tried to facilitate the learning process of developers by introducing the functionality of the API, the steps to be a PLUGGY developer, the basics on how to create plugins and applications, and a set of example use cases which illustrate the potential of the platform and encourage developers to create their own applications.

PLUGGY is a live project and this document may become obsolete. The introductory section gives a comprehensive list of online resources for updated reference information.

[9] https://reactjs.org/
[10] https://threejs.org/

Annex: API Documentation

PLUGGY is evolving and so will its API. The online documentation at

https://pluggy.eu/api/doc/

may be more updated that this document, and developers who seek updated support can reach the consortium at platform@pluggy-project.eu.

License: *License - MIT*

MINIMAL REQUIRED DATA MODELS

This section clarifies the optional and required attributes for the main data models of the API.

Asset

```
{
    title: { //required: minimum length, 1 character
        type: String
    },
    description: { //required: can be empty string
        type: String
    },
    transcription: { // optional
        type: String
    },
    public: { // not used, always public
        type: Boolean,
        default: false,
    },
    creator: { //required - automatically generated by back-end
        type: Schema.Types.ObjectId,
        ref: 'UserGeneric'
    },
    owner: { //required - automatically generated by back-end
        type: Schema.Types.ObjectId,
        ref: 'UserGeneric'
    },
    tags: [{ // required, minimum one tag
        type: String
    }],
    mediaContent: [ // optional
        {type: Schema.Types.Object, ref: 'GFS' }
    ],
    coverImage: {// optional
        type: Schema.Types.Object, ref: 'GFS'
    },
    language: {// not used
        type: String
    },
    permalink: {// backend managed
        type: String
    },
    metadata: { // required if coverImage is used for coverLegal (same as legal)
        type: Schema.Types.Mixed,
    },
    social:  {  type:  SocialInteraction,  default:  ()  =>  ({})  },  //  required  -
automatically generated by backend
    location: { type: GeoLocation, default: () => ({}) },
    updatedAt:{ // automatically generated by backend
             type: Date,
             default: Date.now
    },
    createdAt:{ // automatically generated by backend
             type: Date,
             default: Date.now
    },
    legal: { // required
        license: { // required
            type: String
        },
        isOwnWork: { // required
            type: Boolean,
            default: false,
        },
        source: { // required
            type: String
        },
        author: { // required
            type: String
        },
        authorKind: { // required
            type: String,
            enum: ["UserTeam", "UserPerson", "External"],
        }
    },
    type: { // required
        type: String,
        enum: ["audio", "video", "text", "3Dmodel", "image", "mixed"],
        allowNull: true,
    },
    origin: { // not used, replaced by owner - i.e. team europeana
        type: String,
        enum: ['pluggy.eu', 'europeana.eu', 'wikipedia.org'],
        allowNull: true,
    },
}
```

Exhibition

```
{
  title: { //required: can be empty string
    type: String
  },
  description: { //required: can be empty string
    type: String
  },
  transcription: { // not used
    type: String
  },
  public: { //required
    type: Boolean,
    default: false,
  },
  creator: { //required - automatically generated by back-end
    type: Schema.Types.ObjectId,
    ref: 'UserGeneric'
  },
  owner: { //required - automatically generated by back-end
    type: Schema.Types.ObjectId,
    ref: 'UserGeneric'
  },
  tags: [{ // required: can be empty
    type: String
  }],
  mediaContent: [
    {type: Schema.Types.Object, ref: 'GFS' }
  ],
  coverImage: {
    type: Schema.Types.Object, ref: 'GFS'
  },
  language: { // not used
    type: String
  },
  permalink: { // not used
    type: String
  },
  metadata: { // should be array [{coverLegal: {//same as legal...}}, ...]
    type: Schema.Types.Mixed,
  },
  social: { type: SocialInteraction, default: () => ({}) }, // required -
automatically generated by backend
  location: { type: GeoLocation, default: () => ({}) },
  updatedAt:{ // automatically generated by backend
       type: Date,
       default: Date.now
  },
  createdAt:{ // automatically generated by backend
       type: Date,
       default: Date.now
  },
  legal: { // required, as for Asset
    license: {
      type: String
    },
    isOwnWork: {
      type: Boolean,
      default: false,
    },
    source: {
      type: String
    },
    author: {
      type: String
    },
    authorKind: {
      type: String,
      enum: ["UserTeam", "UserPerson", "External"],
    }
  },
  type: { //required
    type: String,
    // enum: ["tour", "ar", "sound", "game", "media", "timeline", "example"],
    allowNull: true,
  },
  exhibitionPoints: [{ // optional
    type: Schema.Types.ObjectId,
    ref: 'ContentExhibitionPoint'
  }],
  exhibitions: [{ // optional
    type: Schema.Types.ObjectId,
    ref: 'ContentExhibition'
  }],
  arrangements: [{ // optional
    type: {type: String},
    arrangement: Schema.Types.Mixed
  }],
}
```

Exhibition Point

```
{
  title: { //required: can be empty string
    type: String
  },
  description: { // optional
    type: String
  },
  transcription: { // not used
    type: String
  },
  public: { // not used    type: Boolean,
    default: false,
  },
  creator: { //required - automatically generated by back-end
    type: Schema.Types.ObjectId,
    ref: 'UserGeneric'
  },
  owner: { //required - automatically generated by back-end
    type: Schema.Types.ObjectId,
    ref: 'UserGeneric'
  },
  tags: [{ // not used
    type: String
  }],
  mediaContent: [ // not used
    {type: Schema.Types.Object, ref: 'GFS' }
  ],
  coverImage: { // not used
    type: Schema.Types.Object, ref: 'GFS'
  },
  language: { // not used
    type: String
  },
  permalink: { // managed by backend
```

```
    type: String
  },
  metadata: { // optional
    type: Schema.Types.Mixed,
  },
// not used for exhibition points
  social: { type: SocialInteraction, default: () => ({}) }, // required -
automatically generated by backend
  location: { type: GeoLocation, default: () => ({}) },
  updatedAt:{ // automatically generated by backend
       type: Date,
       default: Date.now
  },
  createdAt:{ // automatically generated by backend
       type: Date,
       default: Date.now
  },
  legal: { // not used on Exhibition Points – will be removed
    license: {
      type: String
    },
    isOwnWork: {
      type: Boolean,
      default: false,
    },
    source: {
      type: String
    },
    author: {
      type: String
    },
    authorKind: {
      type: String,
      enum: ["UserTeam", "UserPerson", "External"],
    }
  },
  assets: [{ //optional
    type: Schema.Types.ObjectId,
    ref: 'ContentAsset'
  }],
  content: {  // optional
    contentType: { // required in case of filled in content
      type: String,
      enum: ["html", "markdown", "json", "xml"]
    },
    content:{ // required in case of filled in content
      type: Schema.Types.Object, ref: 'GFS'
    }
  },
}
```

ENDPOINTS AND METHODS.

This section enumerates all methods available for each of the API endpoints.

Current version of the API: **v1**

URL prefix: https://pluggy.eu/api/v1/

/applications

GET

Summary:

customizable method for retrieving applications from the system

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	applications to be returned

Security

Security Schema	Scopes	
OAuth2	read	write

POST

Summary:

create new application

Responses

Code	Description
200	created application

Security

Security Schema	Scopes	
OAuth2	read	write

/applications/{applicationId}

GET

Summary:

get application by Id

Parameters

Name	Located in	Description	Required	Schema

			No	*applicationIdParam*

Responses

Code	Description
200	application to be returned

Security

Security Schema	Scopes	
OAuth2	read	write

PUT

Summary:

update application

Parameters

Name	Located in	Description	Required	Schema
			No	*applicationIdParam*

Responses

Code	Description
200	updated application

Security

Security Schema	Scopes	
OAuth2	read	write

DELETE

Summary:

delete application

Parameters

Name	Located in	Description	Required	Schema
			No	*applicationIdParam*

Responses

Code	Description
200	information about success or fail of the call

Security

Security Schema	Scopes	
OAuth2	read	write

/application/{applicationId}/status

PUT

Summary:

change development status of the application

Responses

Code	Description
200	status updated

Security

Security Schema	Scopes	
OAuth2	read	write

/application/{applicationId}/users

PUT

Summary:

update list of application members

Responses

Code	Description
200	application developer team updated

Security

Security Schema	Scopes	
OAuth2	read	write

/application/{applicationId}/users/{userId}/role

PUT

Summary:

WARNING - Not implemented yet! Change role of user in developer team.

Responses

Code	Description
501	

Security

Security Schema	Scopes	
OAuth2	read	write

/assets

GET

Summary:

customizable method for retrieving assets from the system

Parameters

Name	Located in	Description	Required	Schema
			No	*assetTypeParam*
			No	*qParam*
			No	*tagsParam*
			No	*lngParam*
			No	*latParam*
			No	*repositoryParam*
			No	*pageParam*
			No	*limitParam*
			No	*sortParam*

Responses

Code	Description
200	assets to be returned

Security

Security Schema	Scopes	
OAuth2	read	write

POST

Summary:

create new asset

Responses

Code	Description
200	created asset

Security

Security Schema	Scopes	
OAuth2	read	write

/assets/import

POST

Summary:

import asset from external library

Responses

Code	Description
200	created asset

Security

Security Schema	Scopes	
OAuth2	read	write

/assets/{assetId}

GET

Summary:

get asset by Id

Parameters

Name	Located in	Description	Required	Schema
			No	*assetIdParam*

Responses

Code	Description
200	asset to be returned

Security

Security Schema	Scopes	
OAuth2	read	write

PUT

Summary:

update asset

Parameters

Name	Located in	Description	Required	Schema
			No	*assetIdParam*

Responses

Code	Description
200	updated asset

Security

Security Schema	Scopes	
OAuth2	read	write

DELETE

Summary:

delete asset

Parameters

Name	Located in	Description	Required	Schema
			No	*assetIdParam*

Responses

Code	Description
200	information about success or fail of the call

Security

Security Schema	Scopes	
OAuth2	read	write

/assets/{assetId}/cover

GET

Summary:

get cover image of the asset

Parameters

Name	Located in	Description	Required	Schema
			No	*assetIdParam*

Responses

Code	Description
200	returned media file

Security

Security Schema	Scopes	
OAuth2	read	write

POST

Summary:

create cover image for the asset

Parameters

Name	Located in	Description	Required	Schema
			No	*assetIdParam*

Responses

Code	Description
200	success of the operation

Security

Security Schema	Scopes	
OAuth2	read	write

/assets/{assetId}/media

POST

Summary:

create new media for the asset

Parameters

Name	Located in	Description	Required	Schema
			No	*assetIdParam*

Responses

Code	Description
200	success or fail of the operation

Security

Security Schema	Scopes	
OAuth2	read	write

/assets/{assetId}/media/{mediaId}

GET

Summary:

get media of the asset by Id

Parameters

Name	Located in	Description	Required	Schema
			No	*assetIdParam*
			No	*mediaIdParam*

Responses

Code	Description

200	returned media file

Security

Security Schema	Scopes	
OAuth2	read	write

DELETE

Summary:

delete media of the asset by Id

Parameters

Name	Located in	Description	Required	Schema
			No	*assetIdParam*
			No	*mediaIdParam*

Responses

Code	Description
200	succes or fail of the operation

Security

Security Schema	Scopes	
OAuth2	read	write

/assets/{assetId}/media/{mediaId}/thumbnail

GET

Summary:

get thumbnail of the media of the asset by Id

Parameters

Name	Located in	Description	Required	Schema
			No	*assetIdParam*
			No	*mediaIdParam*

Responses

Code	Description
200	returned thumbnail of the image media file

Security

Security Schema	Scopes	
OAuth2	read	write

/assets/{assetId}/media/{mediaId}/original

GET

Summary:

get original sized image media file

Parameters

Name	Located in	Description	Required	Schema
			No	*assetIdParam*
			No	*mediaIdParam*

Responses

Code	Description
200	returned original media file

Security

Security Schema	Scopes	
OAuth2	read	write

/assets/{assetId}/media/{mediaId}/facebook

GET

Summary:

get optimized size of image media file for the facebook

Parameters

Name	Located in	Description	Required	Schema
			No	*assetIdParam*
			No	*mediaIdParam*

Responses

Code	Description
200	returned resized media file

Security

Security Schema	Scopes	
OAuth2	read	write

/assets/my

GET

Summary:

customizable method for retrieving assets of logged in user from the system

Parameters

Name	Located in	Description	Required	Schema
			No	*qParam*
			No	*tagsParam*
			No	*lngParam*
			No	*latParam*
			No	*repositoryParam*
			No	*pageParam*
			No	*limitParam*
			No	*sortParam*

Responses

Code	Description
200	assets to be returned

Security

Security Schema	Scopes	
OAuth2	read	write

/assets/{assetId}/like

POST

Summary:

update asset

Parameters

Name	Located in	Description	Required	Schema
			No	*assetIdParam*

Responses

Code	Description
200	success or fail of the operation

Security

Security Schema	Scopes	
OAuth2	read	write

/assets/{assetId}/unlike

POST

Summary:

unlike asset

Parameters

Name	Located in	Description	Required	Schema
			No	*assetIdParam*

Responses

Code	Description
200	success or fail of the operation

Security

Security Schema	Scopes	
OAuth2	read	write

/assets/{assetId}/comment

POST

Summary:

comment on asset

Parameters

Name	Located in	Description	Required	Schema
			No	*assetIdParam*

Responses

Code	Description
200	success or fail of the operation

Security

Security Schema	Scopes	
OAuth2	read	write

/assets/{assetId}/report

POST

Summary:

report asset

Parameters

Name	Located in	Description	Required	Schema
			No	*assetIdParam*

Responses

Code	Description
200	success or fail of the operation

Security

Security Schema	Scopes	
OAuth2	read	write

/exhibitions

GET

Summary:

customizable method for retrieving exhibitions from the system

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionTypeParam*
			No	*qParam*
			No	*tagsParam*
			No	*lngParam*
			No	*latParam*
			No	*repositoryParam*
			No	*pageParam*
			No	*limitParam*
			No	*sortParam*

Responses

Code	Description

Code	Description
200	exhibitions to be returned

Security

Security Schema	Scopes	
OAuth2	read	write

POST

Summary:

create new exhibition

Responses

Code	Description
200	created exhibition

Security

Security Schema	Scopes	
OAuth2	read	write

/exhibitions/{exhibitionId}

GET

Summary:

get exhibition by Id

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*

Responses

Code	Description
200	exhibition to be returned

Security

Security Schema	Scopes	
OAuth2	read	write

PUT

Summary:

update exhibition

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*

Responses

Code	Description
200	updated exhibition

Security

Security Schema	Scopes	
OAuth2	read	write

DELETE

Summary:

delete exhibition

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*

Responses

Code	Description
200	information about success or fail of the call

Security

Security Schema	Scopes	
OAuth2	read	write

/exhibitions/{exhibitionId}/cover

GET

Summary:

get cover image of the exhibition

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*

Responses

Code	Description
200	returned media file

Security

Security Schema	Scopes	
OAuth2	read	write

POST

Summary:

create cover image for the exhibition

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*

Responses

Code	Description
200	success of the operation

Security

Security Schema	Scopes	
OAuth2	read	write

/exhibitions/{exhibitionId}/media

POST

Summary:

create new media for the exhibition

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*

Responses

Code	Description
200	success or fail of the operation

Security

Security Schema	Scopes	
OAuth2	read	write

/exhibitions/{exhibitionId}/media/{mediaId}

GET

Summary:

get media of the exhibition by Id

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*

Name	Located in	Description	Required	Schema
			No	*mediaIdParam*

Responses

Code	Description
200	returned media file

Security

Security Schema	Scopes	
OAuth2	read	write

DELETE

Summary:

delete media of the exhibition by Id

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*
			No	*mediaIdParam*

Responses

Code	Description
200	succes or fail of the operation

Security

Security Schema	Scopes	
OAuth2	read	write

/exhibitions/{exhibitionId}/media/{mediaId}/thumbnail

GET

Summary:

get thumbnail of the media of the exhibition by Id

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*
			No	*mediaIdParam*

Responses

Code	Description
200	returned thumbnail of the media file

Security

Security Schema	Scopes	
OAuth2	read	write

/exhibitions/{exhibitionId}/media/{mediaId}/original

GET

Summary:

get original sized image media file

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*
			No	*mediaIdParam*

Responses

Code	Description
200	returned original media file

Security

Security Schema	Scopes	
OAuth2	read	write

/exhibitions/{exhibitionId}/media/{mediaId}/facebook

GET

Summary:

get optimized size of image media file for the facebook

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*
			No	*mediaIdParam*

Responses

Code	Description
200	returned resized media file

Security

Security Schema	Scopes	
OAuth2	read	write

/exhibitions/my

GET

Summary:

customizable method for retrieving exhibitions of logged in user from the system

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionTypeParam*
			No	*qParam*
			No	*tagsParam*
			No	*lngParam*
			No	*latParam*
			No	*repositoryParam*
			No	*pageParam*
			No	*limitParam*
			No	*sortParam*

Responses

Code	Description
200	exhibitions to be returned

Security

Security Schema	Scopes	
OAuth2	read	write

/exhibitions/{exhibitionId}/like

POST

Summary:

update exhibition

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*

Responses

Code	Description

200	success or fail of the operation

Security

Security Schema	Scopes	
OAuth2	read	write

/exhibitions/{exhibitionId}/unlike

POST

Summary:

unlike exhibition

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*

Responses

Code	Description
200	success or fail of the operation

Security

Security Schema	Scopes	
OAuth2	read	write

/exhibitions/{exhibitionId}/comment

POST

Summary:

comment on exhibition

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*

Responses

Code	Description
200	success or fail of the operation

Security

Security Schema	Scopes	
OAuth2	read	write

/exhibitions/{exhibitionId}/report

POST

Summary:

report exhibition

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*

Responses

Code	Description
200	success or fail of the operation

Security

Security Schema	Scopes	
OAuth2	read	write

/exhibitions/{exhibitionId}/exhibition-points

GET

Summary:

retrieving exhibitionPoints from the system

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*

Responses

Code	Description
200	exhibitionPoints to be returned

Security

Security Schema	Scopes	
OAuth2	read	write

POST

Summary:

create new exhibitionPoint

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*

Responses

Code	Description
200	created exhibitionPoint
400	

Security

Security Schema	Scopes	
OAuth2	read	write

/exhibitions/{exhibitionId}/exhibition-points/{exhibitionPointId}

GET

Summary:

get exhibitionPoint by Id

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*
			No	*exhibitionPointIdParam*

Responses

Code	Description
200	exhibitionPoint to be returned

Security

Security Schema	Scopes	
OAuth2	read	write

PUT

Summary:

update exhibitionPoint

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*
			No	*exhibitionPointIdParam*

Responses

Code	Description
200	updated exhibitionPoint

Security

Security Schema	Scopes	
OAuth2	read	write

DELETE

Summary:

delete exhibitionPoint

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*
			No	*exhibitionPointIdParam*

Responses

Code	Description
200	information about success or fail of the call

Security

Security Schema	Scopes	
OAuth2	read	write

/exhibitions/{exhibitionId}/exhibition-points/{exhibitionPointId}/content

POST

Summary:

create new content for the exhibitionPoint

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*
			No	*exhibitionPointIdParam*

Responses

Code	Description
200	success or fail of the operation

Security

Security Schema	Scopes	
OAuth2	read	write

GET

Summary:

get content of the exhibitionPoint

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*
			No	*exhibitionPointIdParam*

Responses

Code	Description
200	returned content file

Security

Security Schema	Scopes	
OAuth2	read	write

/exhibitions/{exhibitionId}/exhibition-points/{exhibitionPointId}/media

POST

Summary:

create new media for the exhibitionPoint

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*
			No	*exhibitionPointIdParam*

Responses

Code	Description
200	success or fail of the operation

Security

Security Schema	Scopes	
OAuth2	read	write

/exhibitions/{exhibitionId}/exhibition-points/{exhibitionPointId}/media/{mediaId}

GET

Summary:

get media of the exhibitionPoint by Id

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*
			No	*exhibitionPointIdParam*
			No	*mediaIdParam*

Responses

Code	Description
200	returned media file

Security

Security Schema	Scopes	
OAuth2	read	write

DELETE

Summary:

delete media of the exhibitionPoint by Id

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*
			No	*exhibitionPointIdParam*
			No	*mediaIdParam*

Responses

Code	Description
200	succes or fail of the operation

Security

Security Schema	Scopes	
OAuth2	read	write

/exhibitions/{exhibitionId}/exhibition-points/{exhibitionPointId}/media/{mediaId}/thumbnail

GET

Summary:

get thumbnail of the media of the exhibitionPoint by Id

Parameters

Name	Located in	Description	Required	Schema

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*
			No	*exhibitionPointIdParam*
			No	*mediaIdParam*

Responses

Code	Description
200	returned thumbnail of the media file

Security

Security Schema	Scopes	
OAuth2	read	write

/exhibitions/{exhibitionId}/exhibition-points/{exhibitionPointId}/media/{mediaId}/original

GET

Summary:

get original sized image media file

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*
			No	*exhibitionPointIdParam*
			No	*mediaIdParam*

Responses

Code	Description
200	returned original media file

Security

Security Schema	Scopes	
OAuth2	read	write

/exhibitions/{exhibitionId}/exhibition-points/{exhibitionPointId}/media/{mediaId}/facebook

GET

Summary:

get optimized size of image media file for the facebook

Parameters

Name	Located in	Description	Required	Schema
			No	*exhibitionIdParam*
			No	*exhibitionPointIdParam*
			No	*mediaIdParam*

Responses

Code	Description
200	returned resized media file

Security

Security Schema	Scopes	
OAuth2	read	write

/hinting/tags

GET

Summary:

hinting service for tags

Parameters

Name	Located in	Description	Required	Schema
			No	*qParam*

Responses

Code	Description
200	applications to be returned

Security

Security Schema	Scopes	
OAuth2	read	write

/hinting/places

GET

Summary:

hinting service for places - provided by google maps API

Parameters

Name	Located in	Description	Required	Schema
			No	*qParam*

Responses

Code	Description
200	applications to be returned

Security

Security Schema	Scopes	
OAuth2	read	write

/hinting/places/{placeId}

GET

Summary:

retrieve information for specific place - provided by google maps API

Parameters

Name	Located in	Description	Required	Schema
			No	*placeIdParam*

Responses

Code	Description
200	applications to be returned

Security

Security Schema	Scopes	
OAuth2	read	write

/hinting/title

GET

Summary:

hinting service for title

Parameters

Name	Located in	Description	Required	Schema
			No	*qParam*

Responses

Code	Description
200	applications to be returned

Security

Security Schema	Scopes	
OAuth2	read	write

/hinting/user

GET

Summary:

hinting service to find specific user

Parameters

Name	Located in	Description	Required	Schema
			No	*qParam*

Responses

Code	Description
200	users to be returned

Security

Security Schema	Scopes	
OAuth2	read	write

/search

GET

Summary:

full-text and geo location search with external repository search integration

Parameters

Name	Located in	Description	Required	Schema
			No	*qParam*
			No	*tagsParam*
			No	*lngParam*
			No	*latParam*
			No	*repositoryParam*
			No	*pageParam*
			No	*limitParam*
			No	*sortParam*

Responses

Code	Description
200	applications to be returned

Security

Security Schema	Scopes	
OAuth2	read	write

/users

GET

Summary:

retrieving users from the system

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	users to be returned

Security

Security Schema	Scopes	
OAuth2	read	write

/users/{userId}

GET

Summary:

get user by Id

Parameters

Name	Located in	Description	Required	Schema
			No	*userIdParam*

Responses

Code	Description
200	user to be returned

Security

Security Schema	Scopes	
OAuth2	read	write

PUT

Summary:

update user

Parameters

Name	Located in	Description	Required	Schema
			No	*userIdParam*

Responses

Code	Description
200	updated user

Security

Security Schema	Scopes	
OAuth2	read	write

DELETE

Summary:

delete user

Parameters

Name	Located in	Description	Required	Schema
			No	*userIdParam*

Responses

Code	Description
200	information about success or fail of the call

Security

Security Schema	Scopes	
OAuth2	read	write

/users/{userId}/media

POST

Summary:

create new media for the user

Parameters

Name	Located in	Description	Required	Schema
			No	*userIdParam*

Responses

Code	Description
200	success or fail of the operation

Security

Security Schema	Scopes

OAuth2	read	write

/users/{userId}/media/{mediaId}

GET

Summary:

get media of the user by Id

Parameters

Name	Located in	Description	Required	Schema
			No	*userIdParam*
			No	*mediaIdParam*

Responses

Code	Description
200	returned media file

Security

Security Schema	Scopes	
OAuth2	read	write

DELETE

Summary:

delete media of the user by Id

Parameters

Name	Located in	Description	Required	Schema
			No	*userIdParam*
			No	*mediaIdParam*

Responses

Code	Description
200	succes or fail of the operation

Security

Security Schema	Scopes	
OAuth2	read	write

/users/{userId}/media/{mediaId}/thumbnail

GET

Summary:

get thumbnail of the media of the user by Id

Parameters

Name	Located in	Description	Required	Schema
			No	*userIdParam*
			No	*mediaIdParam*

Responses

Code	Description
200	returned thumbnail of the media file

Security

Security Schema	Scopes	
OAuth2	read	write

/users/{userId}/folders

GET

Summary:

get user all user’s folders

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	user’s folders

Security

Security Schema	Scopes	
OAuth2	read	write

POST

Summary:

create new user’s folder

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	created folder returned

Security

Security Schema	Scopes	
OAuth2	read	write

/users/{userId}/folders/{folderId}

GET

Summary:

get specific folder

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	requested folder

Security

Security Schema	Scopes	
OAuth2	read	write

PUT

Summary:

update specific folder

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	successfully processed operation

Security

Security Schema	Scopes	
OAuth2	read	write

DELETE

Summary:

delete specific folder

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	successfully processed operation

Security

Security Schema	Scopes	
OAuth2	read	write

/users/{userId}/folders/{folderId}/add/{contentId}

PUT

Summary:

add content to specific folder

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	successfully processed operation

Security

Security Schema	Scopes	
OAuth2	read	write

/users/{userId}/folders/{folderId}/remove/{contentId}

PUT

Summary:

remove specific content from the folder

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	successfully processed operation

Security

Security Schema	Scopes	
OAuth2	read	write

/users/{userId}/following

GET

Summary:

get all users that userId following

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	retrieved users

Security

Security Schema	Scopes	
OAuth2	read	write

/users/{userId}/followers

GET

Summary:

get all user’s followers

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	retrieved users

Security

Security Schema	Scopes	
OAuth2	read	write

/users/{userId}/follow/{userIdToFollow}

PUT

Summary:

follow specific user

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	succesfully processed operation

Security

Security Schema	Scopes	
OAuth2	read	write

/users/{userId}/unfollow/{userIdToUnfollow}

PUT

Summary:

unfollow specific user

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	successfully processed operation

Security

Security Schema	Scopes	
OAuth2	read	write

/users/{userId}/notifications

GET

Summary:

Retrieve all user's notifications. Visit user guidelines for more information about using WebSocket for receiving real time notifications

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	retrieved notifications

Security

Security Schema	Scopes	
OAuth2	read	write

/users/{userId}/notifications/{notificationId}/unread

PUT

Summary:

mark specific notification as unread

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	successfully processed operation

Security

Security Schema	Scopes	
OAuth2	read	write

/users/{userId}/notifications/{notificationId}/read

PUT

Summary:

mark specific notification as read

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	successfully processed operation

Security

Security Schema	Scopes	
OAuth2	read	write

/users/{userId}/notifications/{notificationId}/allread

PUT

Summary:

mark all user’s notification as read

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	successfully processed operation

Security

Security Schema	Scopes	
OAuth2	read	write

/teams

GET

Summary:

retrieving teams from the system

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	teams to be returned

Security

Security Schema	Scopes	
OAuth2	read	write

POST

Summary:

create new team

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	created team

Security

Security Schema	Scopes	
OAuth2	read	write

/teams/{teamId}

GET

Summary:

retrieving specific team

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	requested team returned

Security

Security Schema	Scopes	
OAuth2	read	write

PUT

Summary:

update specific team

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	successful update

Security

Security Schema	Scopes	
OAuth2	read	write

DELETE

Summary:

delete specific team

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	team successfully deleted

Security

Security Schema	Scopes	
OAuth2	read	write

/events

GET

Summary:

retrieving events from the system

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description

200	events to be returned

Security

Security Schema	Scopes	
OAuth2	read	write

POST

Summary:

create new event

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	created event

Security

Security Schema	Scopes	
OAuth2	read	write

/events/{eventId}

GET

Summary:

retrieving specific event

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	requested event returned

Security

Security Schema	Scopes	
OAuth2	read	write

PUT

Summary:

update specific event

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	successful update

Security

Security Schema	Scopes	
OAuth2	read	write

DELETE

Summary:

delete specific event

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	event successfully deleted

Security

Security Schema	Scopes	
OAuth2	read	write

/licenses/filter

GET

Summary:

find appropriate license that match users criteria

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	list of licenses

Security

Security Schema	Scopes	
OAuth2	read	write

/licenses/resolve

GET

Summary:

find appropriate licenses for the derivative work which is compound of content with different licesnses

Parameters

Name	Located in	Description	Required	Schema

Responses

Code	Description
200	list of licenses

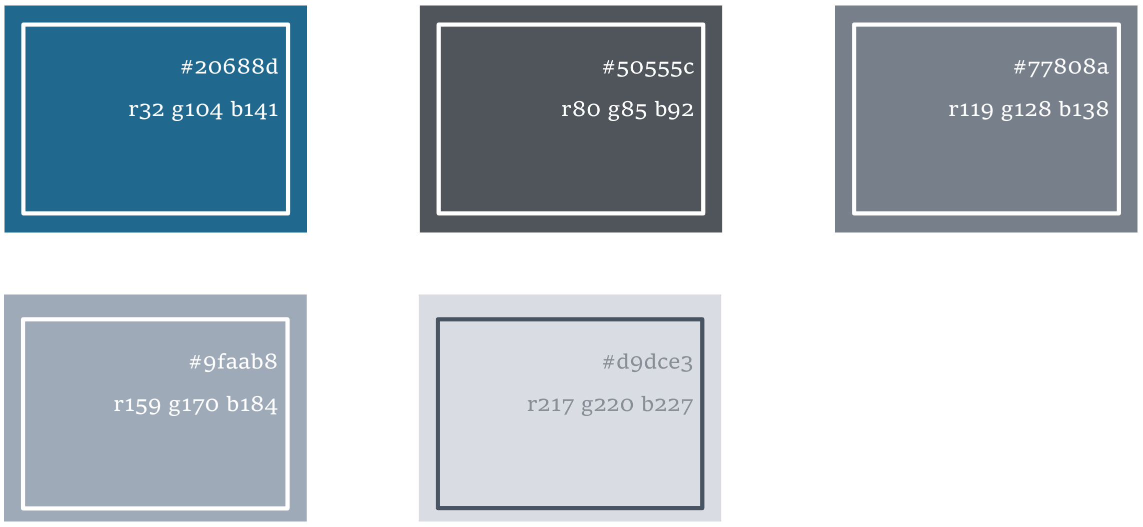
Security

Security Schema	Scopes	
OAuth2	read	write

Annex: Style Guidelines for Plugins

The aim of this visual guide is to summarise the design characteristics of the PLUGGY Social Platform. It is directed to developers that want to use the PLUGGY Social Platform design in their plugin applications. This includes a recommended **Colour Palette**, a set of **Icons**, a **Font** and other basic visual elements, described below.

The PLUGGY Social Platform uses a **Colour Palette**. The main colour is a dark blue, complemented by a selection of greys. These colours are defined in the *styles.css* file of the project, under the names like "--pluggyDarkBlue".



The following are the **icons** used throughout the Pluggy Social Platform.

ASSETS	Image	📷	camera_alt (Material Icons)
	Soundfile	🎵	audiotrack (Material Icons)
	3D object	3D	3d_rotation (Material Icons)
	Video	📹	videocam (Material Icons)

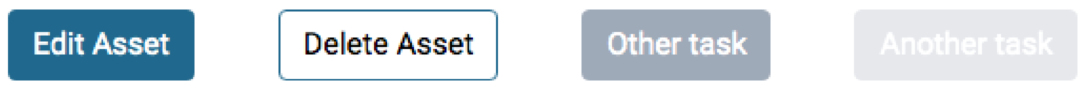
EXHIBITIONS	Blog Story	▦	developer_board (Material Icons)
	Soundscape	▚	SVG file
	AR/VR	▬	SVG file
	Game	✚	games (Material Icons)
	Timeline	■	SVG file
	Tour	📍	room (Material Icons)
MENU	Home	🏠	SVG file
	Exhibitions	🏛	SVG file
	Assets	💎	SVG file
	Folders	📁	SVG file
	Notifications	🔔	notifications_none (Material Icons)
	Search	🔍	search (Material Icons)
SOCIAL INTERACTION	Like	♥	favorite (Material Icons)
	Comment	💬	comment (Material Icons)

Been Here	✓	beenhere (Material Icons)
Report	⚠	warning (Material Icons)
Share	<	share (Material Icons)
Bookmark	🔖	bookmark_border (Material Icons)

The **Font** used by the Pluggy Social Platform is the Google Font Roboto. (https://fonts.google.com/specimen/Roboto)

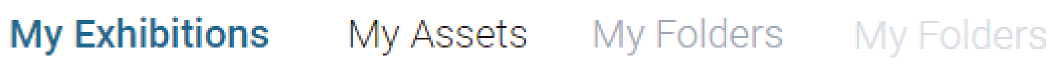
The user interface of the Pluggy Social Platform is built using the Bootstrap library (https://getbootstrap.com/). The components have been modified to suit the colour scheme of the project.

Bootstrap button classes are extended through CSS classes to have the following button states:



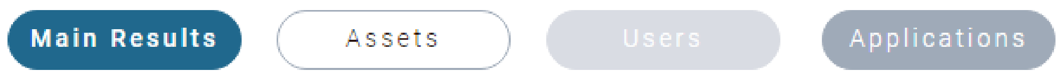
Normal state (two type of buttons) Hover state Disabled State

A text selector was created through CSS classes, and it has the following states:



Selected State Normal State Hover State Disabled State

A filter-badge CSS class was designed for the filters in the search page. It has the following states:



Selected State	Normal State	Disabled State	Hover State