Project Acronym:	HosmartAI
Grant Agreement number:	101016834 (H2020-DT-2020-1 – Innovation Action)
Project Full Title:	Hospital Smart development based on AI

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101016834

DELIVERABLE

D4.3– Platform Architecture Design and Open APIs – Final version

Dissemination level:	PU -Public
Type of deliverable:	R -Report
Contractual date of delivery:	31 January 2024
Deliverable leader:	ITCL
Status - version, date:	Final – v1.0, 2024-01-31
Keywords:	Platform architecture, Open APIs

Executive Summary

This document presents the final version of the HosmartAI Platform Architecture and Open APIs. The second version of the HosmartAI architecture from D4.2 are analysed, identifying all the possible changes on the implementation, communication between pilots or internal elements, and transforming the new requirements into elements to be added to the new version of the architecture.

Also, data inputs and outputs for each pilot are analysed and a common API is updated to standardize the flow of information between the pilots and the platform. This part of the project is focused on the identification of all the key components of the platform and how the interconnection is made.

In turn, the most important part of this document is the final specification of the OpenAPIs, where they have been implemented in the parts of the platform or pilots that were needed. In this way, it leaves a useful tool to be able to access the methods of use of a given tool.

Together, the OpenAPIs as standard communication elements, and the components described in previous deliverables, a complete, functional platform is achieved, with all the necessary elements to perform the proposed functionality within the framework of the project.

Deliverable leader:	Daniel Lozano (ITCL)
Contributors:	Daniel Lozano (ITCL)
Reviewers:	Angelo Consoli, Luca Gilardi (EXYS) Robert Hofsink (PHILIPS)
Approved by:	Athanasios Poulakidas, Anastasia Panitsa (INTRA)

Document History			
Version	**Date**	**Contributor(s)**	**Description**
0.1	2023-04-03	Daniel Lozano (ITCL)	Document creation
0.2	2023-06-15	Daniel Lozano (ITCL)	Include OpenAPI P2 & P6
0.3	2023-07-28	Daniel Lozano (ITCL)	Include OpenAPI Doc Registry
0.4	2023-10-02	Daniel Lozano (ITCL)	Include OpenAPI UM
0.5	2024-01-08	Daniel Lozano (ITCL)	Final details and completion
0.5.1	2024-01-22	Angelo Consoli (EXYS)	Added a section (4.2.2) about Security Events Logging API
0.6	2024-01-24	Daniel Lozano (ITCL)	Review fixes
1.0	2024-01-24	A. Poulakidas, A. Panitsa (INTRA)	Final version for submission after QA

Table of Contents

List of Figures

List of Tables

Definitions, Acronyms and Abbreviations

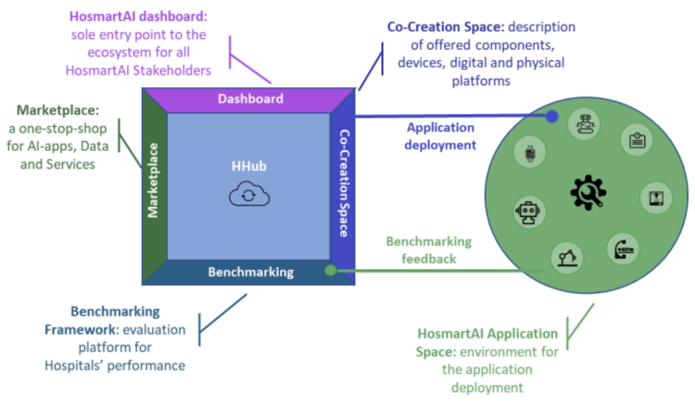
Acronym/ Abbreviation	Title
API	Application Programming Interface
DoA	Description of Action
EHR	Electronic Health Record
HHub	HosmartAI Hub
HL7	Health Level 7
HL-FHIR	Health Level 7 – Fast Healthcare Interoperability Resources
KPI	Key Performance Indicator
RAF	Reference Architecture Framework
SME	Subject Matter Expert
WP	Work Package

Term	Definition
Consortium	Group of beneficiaries that have signed the Consortium Agreement and the Grant Agreement (either directly as Coordinator or by accession through the Form A).
Consortium Agreement	Contractual document signed by all the beneficiaries (and not the EC), explaining how the Consortium is managed and works together.
Deliverable Leader	Responsible for ensuring that the content of the deliverable meets the required expectations, both from a contractual point of view and in terms of usage within the project. Is also responsible for ensuring that the deliverable follows the deliverable process and is delivered on time.
Description of Action	Annex 1 to the Grant Agreement. It contains information on the work packages, deliverables, milestones, resources and costs of the beneficiaries, as well as a text with a detailed description of the action. The DoA is made of Part A (structured data collected in web forms and workplan tables) and Part B (text document describing the action elements).
Dissemination	EC term for communication of information to a wide audience.
Grant Agreement	Contractual document which defines the contractual scope of the HosmartAI project. It is signed between the EC and the beneficiaries.

1 Introduction

1.1 Project Information

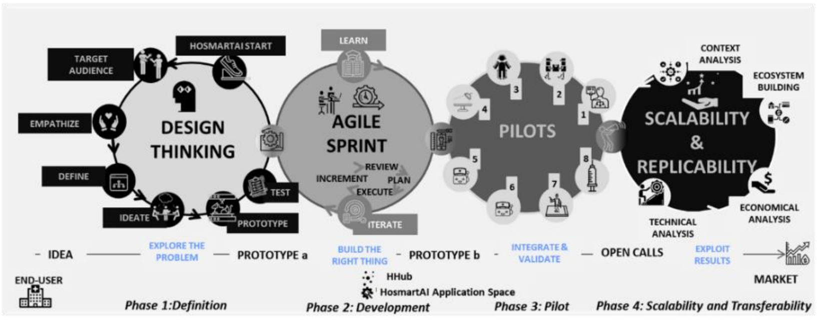
VISION	The HosmartAI vision is a strong, efficient, sustainable and resilient European **Healthcare system** benefiting from the capacities to generate impact of the technology European Stakeholders (SMEs, Research centres, Digital Hubs and Universities).
MISSION	The HosmartAI mission is to guarantee the **integration** of Digital and Robot technologies in new Healthcare environments and the possibility to analyse their benefits by providing an **environment** where digital health care tool providers will be able to design and develop AI solutions as well as a space for the instantiation and deployment of AI solutions.


HosmartAI dashboard: sole entry point to the ecosystem for all HosmartAI Stakeholders
Co-Creation Space: description of offered components, devices, digital and physical platforms
Marketplace: a one-stop-shop for AI-apps, Data and Services
Dashboard
Marketplace
HHub
Co-Creation Space
Benchmarking
Application deployment
Benchmarking feedback
Benchmarking Framework: evaluation platform for Hospitals' performance
HosmartAI Application Space: environment for the application deployment

HosmartAI will create a common open Integration **Platform** with the necessary tools to facilitate and measure the benefits of integrating digital technologies (robotics and AI) in the healthcare system.

A central **hub** will offer multifaceted lasting functionalities (Marketplace, Co-creation space, Benchmarking) to healthcare stakeholders, combined with a collection of methods, tools and solutions to integrate and deploy AI-enabled solutions. The **Benchmarking** tool will promote the adoption in new settings, while enabling a meeting place for technology providers and end-users.

Eight Large-Scale Pilots will implement and evaluate improvements in medical diagnosis, surgical interventions, prevention and treatment of diseases, and support for rehabilitation and long-term care in several Hospital and care settings. The project will target different **medical** aspects or manifestations such as Cancer (Pilot #1, #2 and #8); Gastrointestinal (GI) disorders (Pilot #1); Cardiovascular diseases (Pilot #1, #4, #5 and #7); Thoracic Disorders (Pilot #5); Neurological diseases (Pilot #3); Elderly Care and Neuropsychological Rehabilitation (Pilot #6); Fetal Growth Restriction (FGR) and Prematurity (Pilot #1).



To ensure a user-centred approach, harmonization in the process (e.g., regarding ethical aspects, standardization, and robustness both from a technical and social and healthcare perspective), the **living lab** methodology will be employed. HosmartAI will identify the appropriate instruments (**KPI**) that measure efficiency without undermining access or quality of care. Liaison and co-operation activities with relevant stakeholders and **open calls** will enable ecosystem building and industrial clustering.

HosmartAI brings together a **consortium** of leading organizations (3 large enterprises, 8 SMEs, 5 hospitals, 4 universities, 2 research centres and 2 associations – see Table 1) along with several more committed organizations (Letters of Support provided).

Table 1: The HosmartAI consortium.

Number[1]	Name	Short name
1 (CO)	INTRASOFT INTERNATIONAL SA	**INTRA**
1.1 (TP)	INTRASOFT INTERNATIONAL SA	**INTRA-LU**
2	PHILIPS MEDICAL SYSTEMS NEDERLAND BV	**PHILIPS**
3	VIMAR SPA	**VIMAR**
4	GREEN COMMUNICATIONS SAS	**GC**
5	TELEMATIC MEDICAL APPLICATIONS EMPORIA KAI ANAPTIXI PROIONTON TILIATRIKIS MONOPROSOPIKI ETAIRIA PERIORISMENIS EYTHINIS	**TMA**
6	ECLEXYS SAGL	**EXYS**
7	F6S NETWORK IRELAND LIMITED	**F6S**
7.1 (TP)	F6S NETWORK LIMITED	**F6S-UK**
8	PHARMECONS EASY ACCESS LTD	**PhE**
9	TERAGLOBUS LATVIA SIA	**TGLV**
10	NINETY ONE GMBH	**91**
11	EIT HEALTH GERMANY GMBH	**EIT**
12	UNIVERZITETNI KLINICNI CENTER MARIBOR	**UKCM**
13	SAN CAMILLO IRCCS SRL	**IRCCS**
14	SERVICIO MADRILENO DE SALUD	**SERMAS**
14.1 (TP)	FUNDACION PARA LA INVESTIGACION BIOMEDICA DEL HOSPITAL UNIVERSITARIO LA PAZ	**FIBHULP**
15	CENTRE HOSPITALIER UNIVERSITAIRE DE LIEGE	**CHUL**
16	PANEPISTIMIAKO GENIKO NOSOKOMEIO THESSALONIKIS AXEPA	**AHEPA**
17	VRIJE UNIVERSITEIT BRUSSEL	**VUB**
18	ARISTOTELIO PANEPISTIMIO THESSALONIKIS	**AUTH**
19	EIDGENOESSISCHE TECHNISCHE HOCHSCHULE ZUERICH	**ETHZ**
20	UNIVERZA V MARIBORU	**UM**

[1] CO: Coordinator. TP: linked third party.

Number[1]	Name	Short name
21	INSTITUTO TECNOLÓGICO DE CASTILLA Y LEON	**ITCL**
22	FUNDACION INTRAS	**INTRAS**
23	ASSOCIATION EUROPEAN FEDERATION FORMEDICAL INFORMATICS	**EFMI**
24	FEDERATION EUROPEENNE DES HOPITAUX ET DES SOINS DE SANTE	**HOPE**

1.2 Document Scope

This deliverable has as main objective to describe the last iteration of the architecture relative to the platform, as well as the changes that it has undergone regarding the second version described in the terrier deliverable.

All these changes will be described in detail in this document to understand why they have been made and what new needs they are covering.

In turn, other of the main elements that you want to detail in this deliverable, is the communication between the elements of the architecture through a specification OpenAPI public, that is known by all to be able to be used from any elements of the platform, either internal or external.

1.3 Document Structure

This document is comprised of the following chapters:

Chapter 1 presents an introduction to the project and the document.

Chapter 2 specifies the OpenAPI development and tools.

Chapter 3 explains and describes the final version of OpenAPI specification.

Chapter 4 explains and describes the final version of the HosmartAI architecture.

Chapter 5 provides some concluding remarks.

2 OpenAPIs development

To establish a stable and robust final specification, throughout the development of this task, different tools have been made available to the main partners involved in the development of parts of the architecture to generate a specification understandable to others.

As a result of this work, the main tool with which the whole specification has been elaborated is presented below, providing the clarity needed in a project as complex as the one we are dealing with.

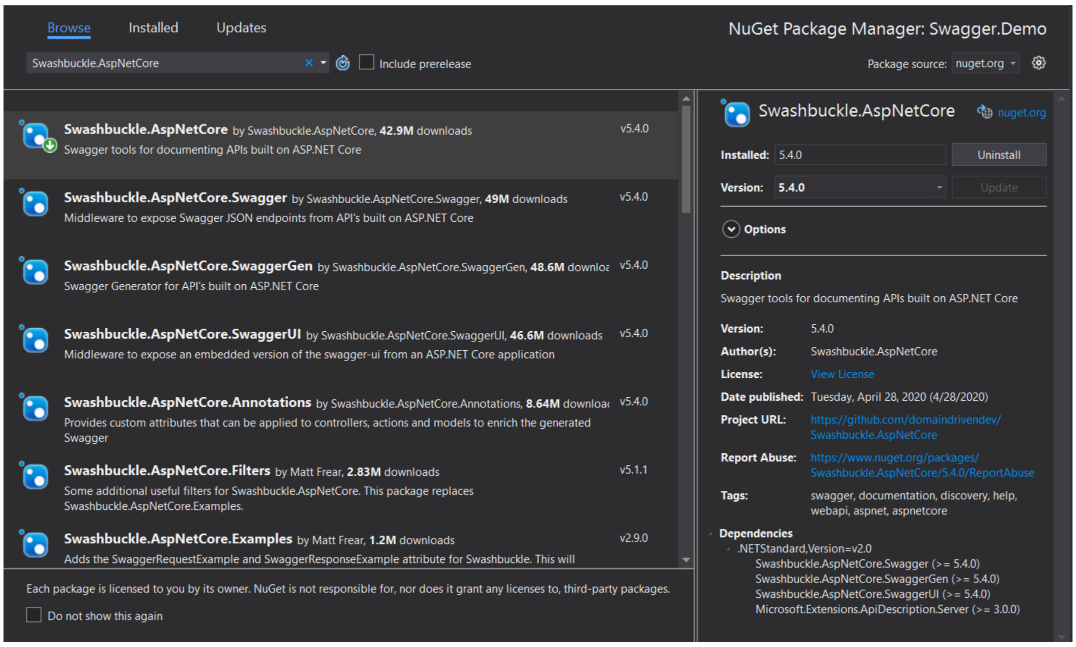
To choose the main tool, several factors have been considered, such as the ease of use to generate the information from the code, the accessibility of the tool at an economic level, as well as the results and efficiency when working with the tool.

2.1.1 How to generate an OpenAPI

2.1.1.1 Swashbuckle (NET Core)

This tool consists of a swagger tooling for APIs built with ASP.NET Core. Generates API documentation, including a UI to explore and test operations, directly from the routes, controllers, and models.

In addition to its Swagger 2.0 and OpenAPI 3.0 generator, Swashbuckle also provides an embedded version of the swagger-UI that's powered by the generated Swagger JSON. This means that can complement your API with living documentation that's always in sync with the latest code. It requires minimal coding and maintenance, allowing you to focus on building an awesome API.



Swashbuckle.AspNetCore by Swashbuckle.AspNetCore, **42.9M** downloads v5.4.0
Swagger tools for documenting APIs built on ASP.NET Core

Swashbuckle.AspNetCore.Swagger by Swashbuckle.AspNetCore.Swagger, **49M** downloads v5.4.0
Middleware to expose Swagger JSON endpoints from API's built on ASP.NET Core

Swashbuckle.AspNetCore.SwaggerGen by Swashbuckle.AspNetCore.SwaggerGen, **48.6M** downloa v5.4.0
Swagger Generator for API's built on ASP.NET Core

Swashbuckle.AspNetCore.SwaggerUI by Swashbuckle.AspNetCore.SwaggerUI, **46.6M** downloads v5.4.0
Middleware to expose an embedded version of the swagger-ui from an ASP.NET Core application

Swashbuckle.AspNetCore.Annotations by Swashbuckle.AspNetCore.Annotations, **8.64M** downloa v5.4.0
Provides custom attributes that can be applied to controllers, actions and models to enrich the generated Swagger

Swashbuckle.AspNetCore.Filters by Matt Frear, **2.83M** downloads v5.1.1
Some additional useful filters for Swashbuckle.AspNetCore. This package replaces Swashbuckle.AspNetCore.Examples.

Swashbuckle.AspNetCore.Examples by Matt Frear, **1.2M** downloads v2.9.0
Adds the SwaggerRequestExample and SwaggerResponseExample attribute for Swashbuckle. This will

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.

Do not show this again

Swashbuckle.AspNetCore nuget.org

Installed: 5.4.0 Uninstall

Version: 5.4.0 Update

Options

Description
Swagger tools for documenting APIs built on ASP.NET Core

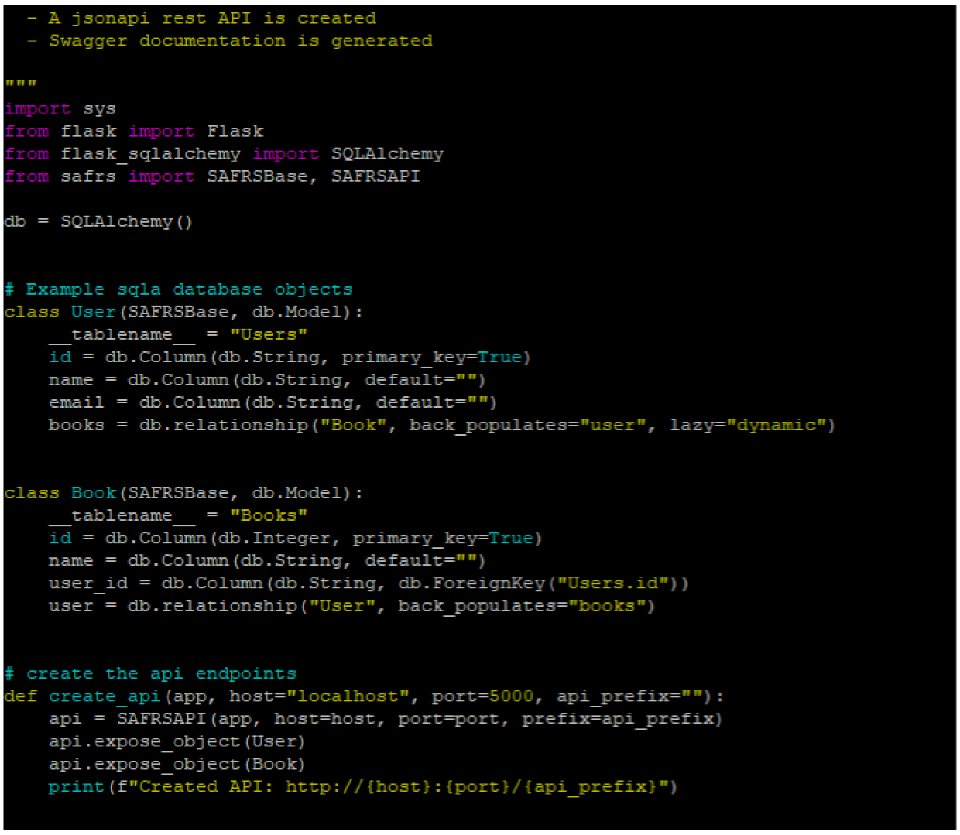
Version:	5.4.0
Author(s):	Swashbuckle.AspNetCore
License:	View License
Date published:	Tuesday, April 28, 2020 (4/28/2020)
Project URL:	https://github.com/domaindrivendev/Swashbuckle.AspNetCore
Report Abuse:	https://www.nuget.org/packages/Swashbuckle.AspNetCore/5.4.0/ReportAbuse
Tags:	swagger, documentation, discovery, help, webapi, aspnet, aspnetcore

Dependencies
.NETStandard,Version=v2.0
Swashbuckle.AspNetCore.Swagger (>= 5.4.0)
Swashbuckle.AspNetCore.SwaggerGen (>= 5.4.0)
Swashbuckle.AspNetCore.SwaggerUI (>= 5.4.0)
Microsoft.Extensions.ApiDescription.Server (>= 3.0.0)

Figure 1: Tool installation from IDE.

2.1.1.2 safrs (Python)

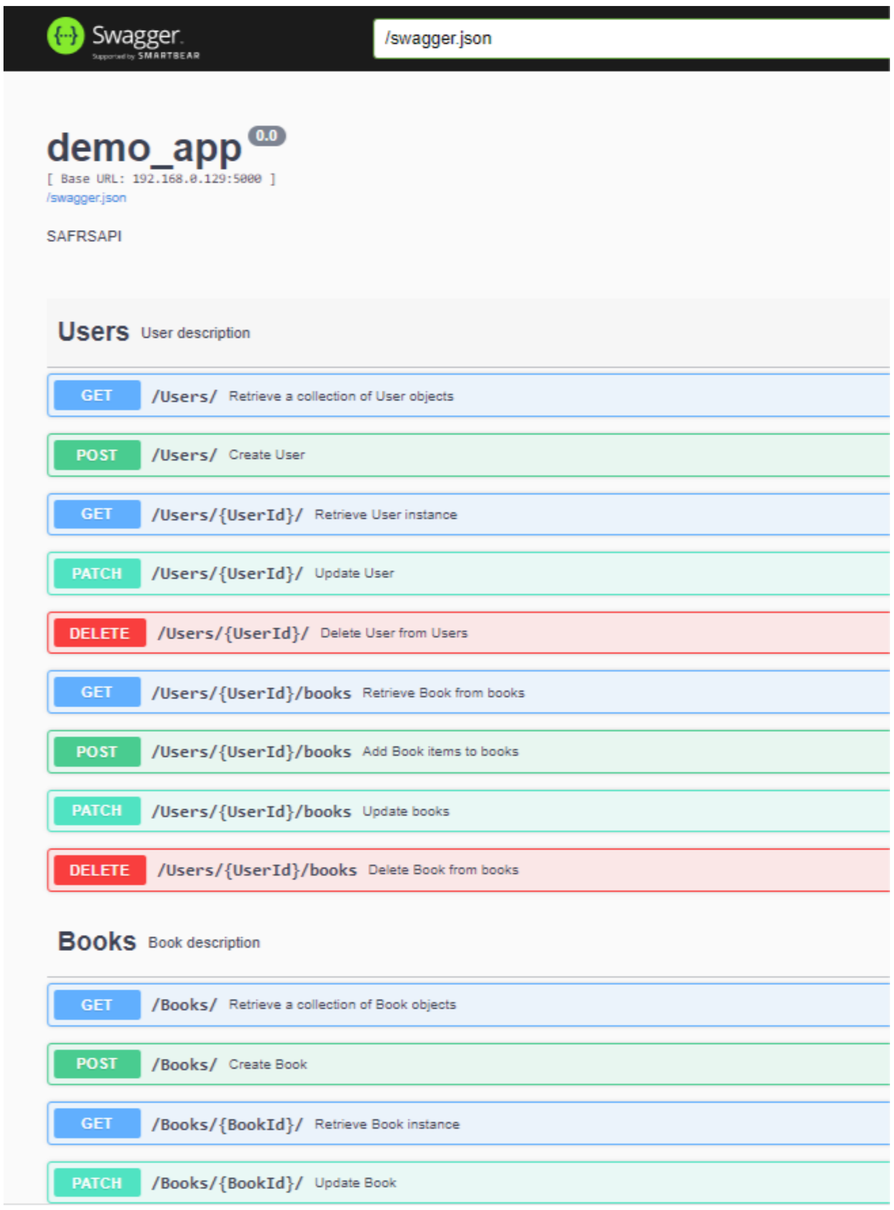
SAFRS is a Python library that exposes a database defined with the framework SQLAlchemy as a JSON:API web service and generates the corresponding OpenAPI specification for a swagger service.



```
  - A jsonapi rest API is created
  - Swagger documentation is generated

"""
import sys
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from safrs import SAFRSBase, SAFRSAPI

db = SQLAlchemy()


# Example sqla database objects
class User(SAFRSBase, db.Model):
    __tablename__ = "Users"
    id = db.Column(db.String, primary_key=True)
    name = db.Column(db.String, default="")
    email = db.Column(db.String, default="")
    books = db.relationship("Book", back_populates="user", lazy="dynamic")


class Book(SAFRSBase, db.Model):
    __tablename__ = "Books"
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String, default="")
    user_id = db.Column(db.String, db.ForeignKey("Users.id"))
    user = db.relationship("User", back_populates="books")


# create the api endpoints
def create_api(app, host="localhost", port=5000, api_prefix=""):
    api = SAFRSAPI(app, host=host, port=port, prefix=api_prefix)
    api.expose_object(User)
    api.expose_object(Book)
    print(f"Created API: http://{host}:{port}/{api_prefix}")
```

Figure 2: Example of safrs code.

By default, it generates GET (retrieve an object), POST (Create an object), DELETE (Remove an object) and PATCH (Update an object) methods for a selected object and the objects it's related to.



demo_app 0.0
[Base URL: 192.168.0.129:5000]
/swagger.json

SAFRSAPI

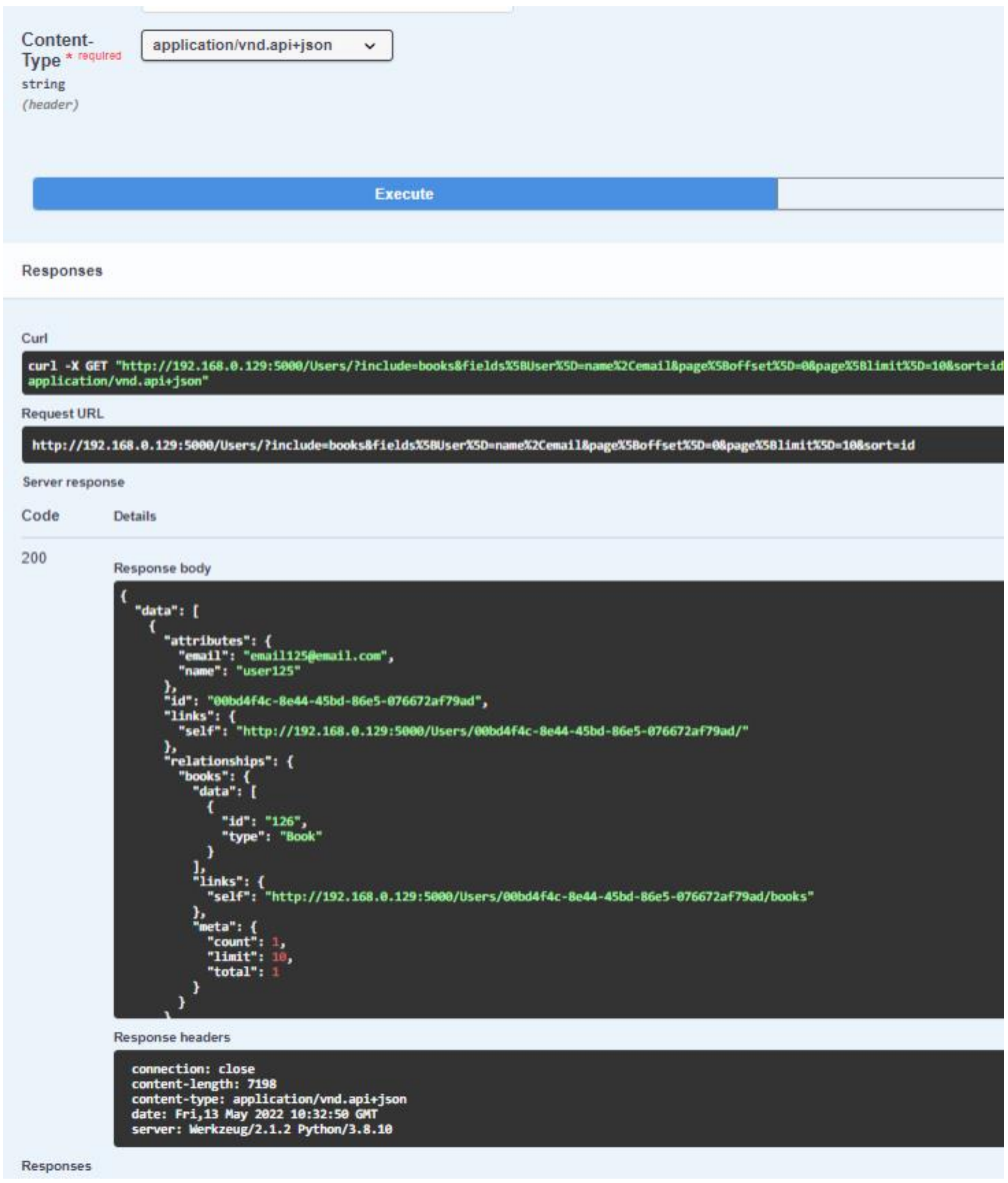
Users User description

GET /Users/ Retrieve a collection of User objects
POST /Users/ Create User
GET /Users/{UserId}/ Retrieve User instance
PATCH /Users/{UserId}/ Update User
DELETE /Users/{UserId}/ Delete User from Users
GET /Users/{UserId}/books Retrieve Book from books
POST /Users/{UserId}/books Add Book items to books
PATCH /Users/{UserId}/books Update books
DELETE /Users/{UserId}/books Delete Book from books

Books Book description

GET /Books/ Retrieve a collection of Book objects
POST /Books/ Create Book
GET /Books/{BookId}/ Retrieve Book instance
PATCH /Books/{BookId}/ Update Book

Figure 3: Safrs autogenerated Swagger.



```
curl -X GET "http://192.168.0.129:5000/Users/?include=books&fields%5BUser%5D=name%2Cemail&page%5Boffset%5D=0&page%5Blimit%5D=10&sort=id
application/vnd.api+json"
```

```
http://192.168.0.129:5000/Users/?include=books&fields%5BUser%5D=name%2Cemail&page%5Boffset%5D=0&page%5Blimit%5D=10&sort=id
```

```
{
  "data": [
    {
      "attributes": {
        "email": "email125@email.com",
        "name": "user125"
      },
      "id": "00bd4f4c-8e44-45bd-86e5-076672af79ad",
      "links": {
        "self": "http://192.168.0.129:5000/Users/00bd4f4c-8e44-45bd-86e5-076672af79ad/"
      },
      "relationships": {
        "books": {
          "data": [
            {
              "id": "126",
              "type": "Book"
            }
          ],
          "links": {
            "self": "http://192.168.0.129:5000/Users/00bd4f4c-8e44-45bd-86e5-076672af79ad/books"
          },
          "meta": {
            "count": 1,
            "limit": 10,
            "total": 1
          }
        }
```

```
connection: close
content-length: 7198
content-type: application/vnd.api+json
date: Fri,13 May 2022 10:32:50 GMT
server: Werkzeug/2.1.2 Python/3.8.10
```

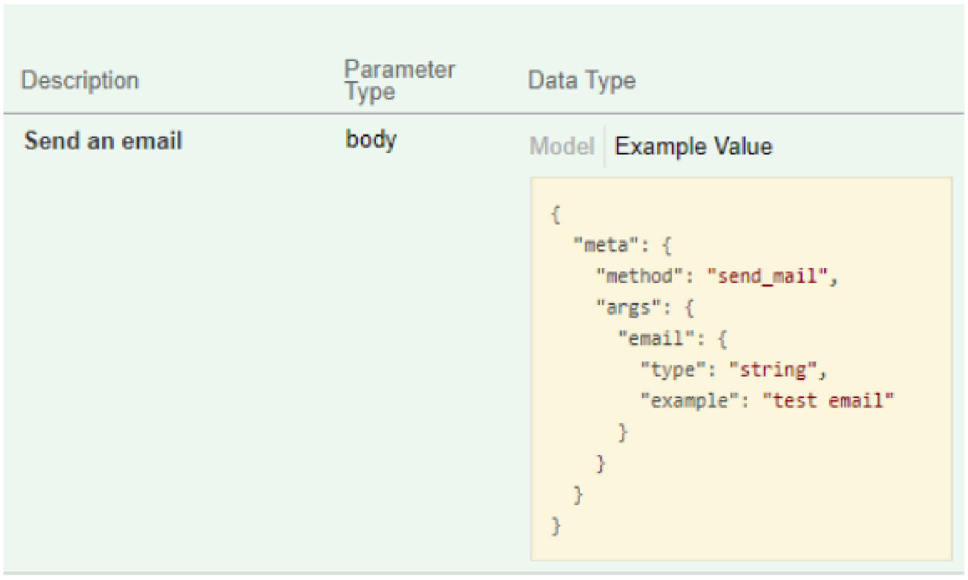
Figure 4: Safrs Swagger call example.

OpenAPI descriptions can be added by using comments to classes and functions, by adding a triple double-quote comment below the class or function definition.

The "description" comment describes the parameter, and the "args" comment can give more information about the field's names, types, and examples of use.

For example:

```
def send_mail(self, **args):
"""
    description: Send an email
    args:
      email:
        type: string
        example: test email
"""
```



Description	Parameter Type	Data Type
Send an email	body	Model Example Value

```
{
  "meta": {
    "method": "send_mail",
    "args": {
      "email": {
        "type": "string",
        "example": "test email"
      }
    }
  }
}
```

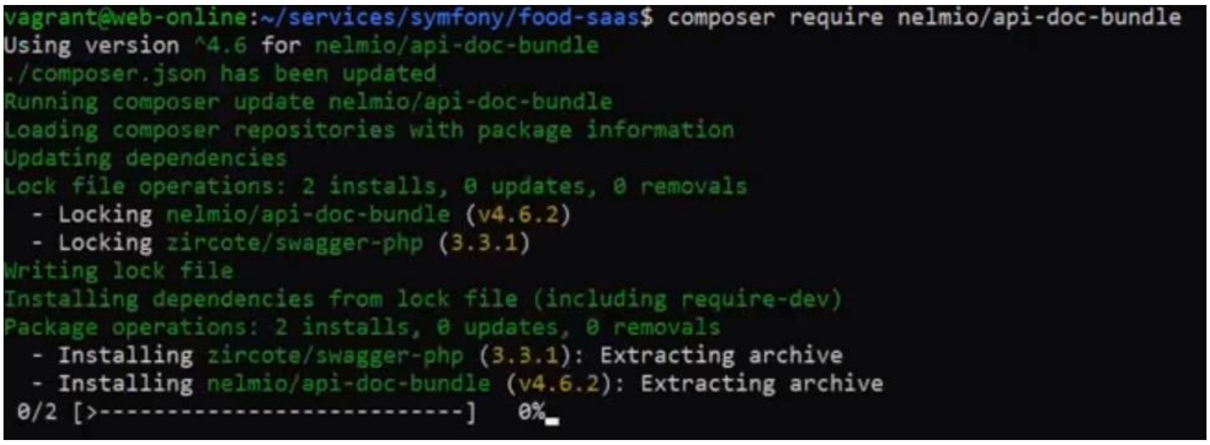
Figure 5: Safrs description swagger example.

2.1.1.3 NelmioApiDocBundle (PHP-symfony)

NelmioApiDocBundle is a framework that adds OpenAPI and swagger generation to PHP-Symfony.

To install it, Symfony and PHP must be installed and configured. Then, run the following command in the terminal:

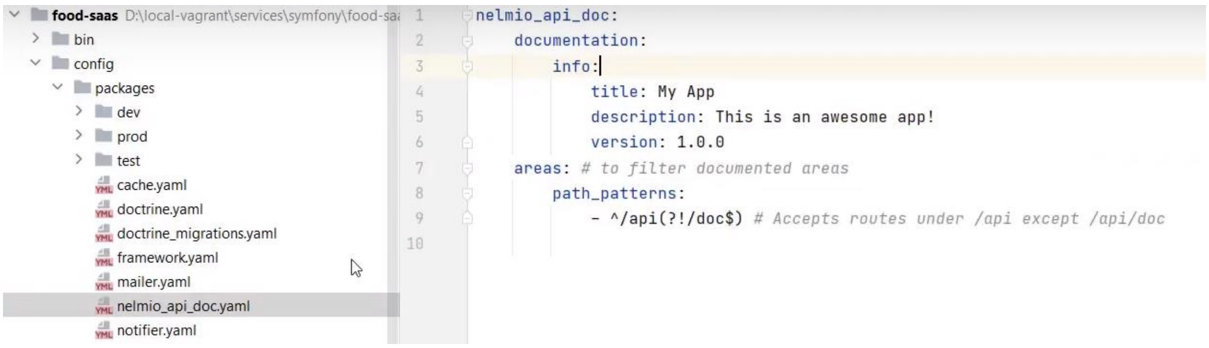
composer require nelmio/api-doc-bundle



```
vagrant@web-online:~/services/symfony/food-saas$ composer require nelmio/api-doc-bundle
Using version ^4.6 for nelmio/api-doc-bundle
./composer.json has been updated
Running composer update nelmio/api-doc-bundle
Loading composer repositories with package information
Updating dependencies
Lock file operations: 2 installs, 0 updates, 0 removals
  - Locking nelmio/api-doc-bundle (v4.6.2)
  - Locking zircote/swagger-php (3.3.1)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 2 installs, 0 updates, 0 removals
  - Installing zircote/swagger-php (3.3.1): Extracting archive
  - Installing nelmio/api-doc-bundle (v4.6.2): Extracting archive
 0/2 [>---------------------------]   0%
```

Figure 6: Nelmio api-doc-bundle installation.

The project needs the file: nelmio_api_doc.yaml, where the main documentation for the API can be added, and a regular expression for only showing paths to API URLs in the OpenAPI.

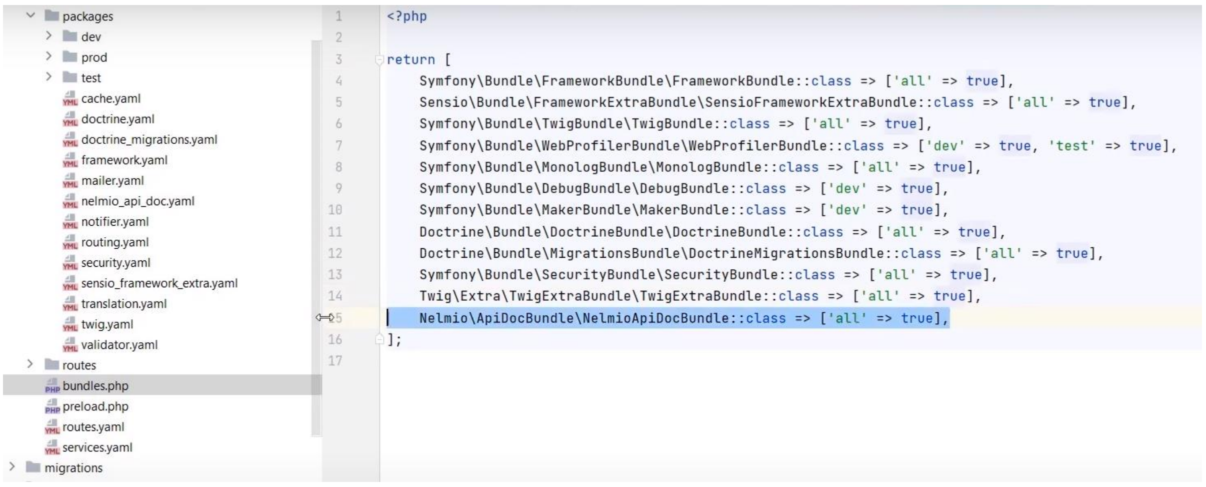


```
food-saas D:\local-vagrant\services\symfony\food-sa
  bin
  config
    packages
      dev
      prod
      test
      cache.yaml
      doctrine.yaml
      doctrine_migrations.yaml
      framework.yaml
      mailer.yaml
      nelmio_api_doc.yaml
      notifier.yaml
```

```
nelmio_api_doc:
    documentation:
        info:
            title: My App
            description: This is an awesome app!
            version: 1.0.0
    areas: # to filter documented areas
        path_patterns:
            - ^/api(?!/doc$) # Accepts routes under /api except /api/doc
```

Figure 7: Nelmio api doc yaml.

In the file bundles.php, there must be a line including nelmio \ apiDocBundle \ NelbioApiDocBundle.

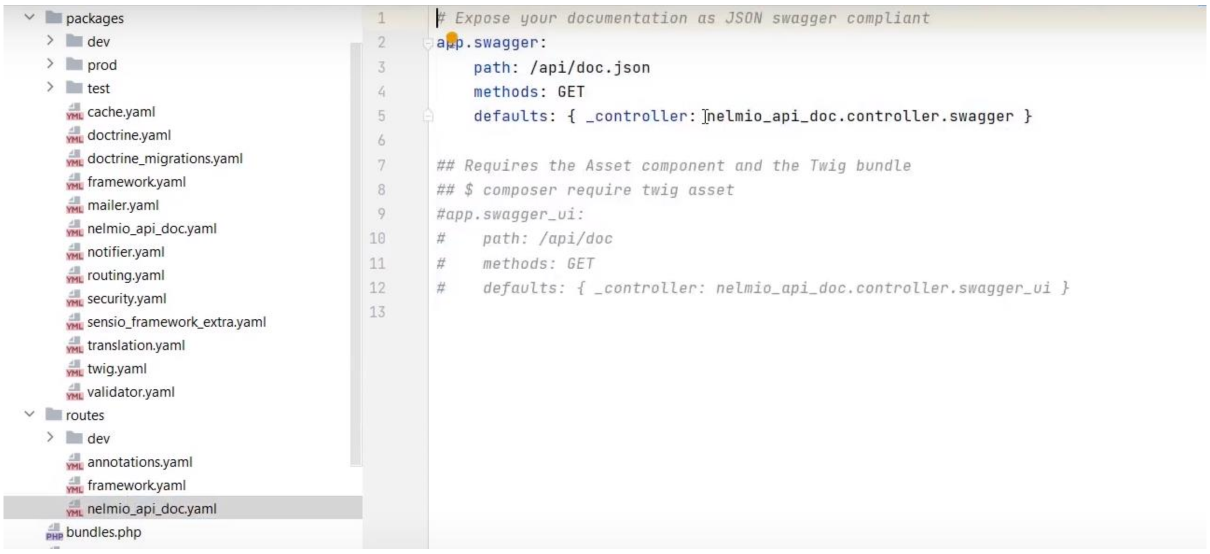


```
packages
  dev
  prod
  test
  cache.yaml
  doctrine.yaml
  doctrine_migrations.yaml
  framework.yaml
  mailer.yaml
  nelmio_api_doc.yaml
  notifier.yaml
  routing.yaml
  security.yaml
  sensio_framework_extra.yaml
  translation.yaml
  twig.yaml
  validator.yaml
routes
bundles.php
preload.php
routes.yaml
services.yaml
migrations
```

```php
<?php

return [
    Symfony\Bundle\FrameworkBundle\FrameworkBundle::class => ['all' => true],
    Sensio\Bundle\FrameworkExtraBundle\SensioFrameworkExtraBundle::class => ['all' => true],
    Symfony\Bundle\TwigBundle\TwigBundle::class => ['all' => true],
    Symfony\Bundle\WebProfilerBundle\WebProfilerBundle::class => ['dev' => true, 'test' => true],
    Symfony\Bundle\MonologBundle\MonologBundle::class => ['all' => true],
    Symfony\Bundle\DebugBundle\DebugBundle::class => ['dev' => true],
    Symfony\Bundle\MakerBundle\MakerBundle::class => ['dev' => true],
    Doctrine\Bundle\DoctrineBundle\DoctrineBundle::class => ['all' => true],
    Doctrine\Bundle\MigrationsBundle\DoctrineMigrationsBundle::class => ['all' => true],
    Symfony\Bundle\SecurityBundle\SecurityBundle::class => ['all' => true],
    Twig\Extra\TwigExtraBundle\TwigExtraBundle::class => ['all' => true],
    Nelmio\ApiDocBundle\NelmioApiDocBundle::class => ['all' => true],
];
```

Figure 8: bundles.php nelmioApiDocBundle.

And the last thing to configure the OpenAPI must be a file “nalmio_api_doc.yaml”, in the routes folder, with the path for showing the swagger interface.



```
packages
  dev
  prod
  test
  cache.yaml
  doctrine.yaml
  doctrine_migrations.yaml
  framework.yaml
  mailer.yaml
  nelmio_api_doc.yaml
  notifier.yaml
  routing.yaml
  security.yaml
  sensio_framework_extra.yaml
  translation.yaml
  twig.yaml
  validator.yaml
routes
  dev
  annotations.yaml
  framework.yaml
  nelmio_api_doc.yaml
bundles.php
```

```yaml
# Expose your documentation as JSON swagger compliant
app.swagger:
    path: /api/doc.json
    methods: GET
    defaults: { _controller: nelmio_api_doc.controller.swagger }

## Requires the Asset component and the Twig bundle
## $ composer require twig asset
#app.swagger_ui:
#    path: /api/doc
#    methods: GET
#    defaults: { _controller: nelmio_api_doc.controller.swagger_ui }
```

Figure 9: Example code.

By doing this, the /api/doc URL can be accessed, with the swagger interface, and the /api/doc.json can be accessed with the source code for the OpenAPI.

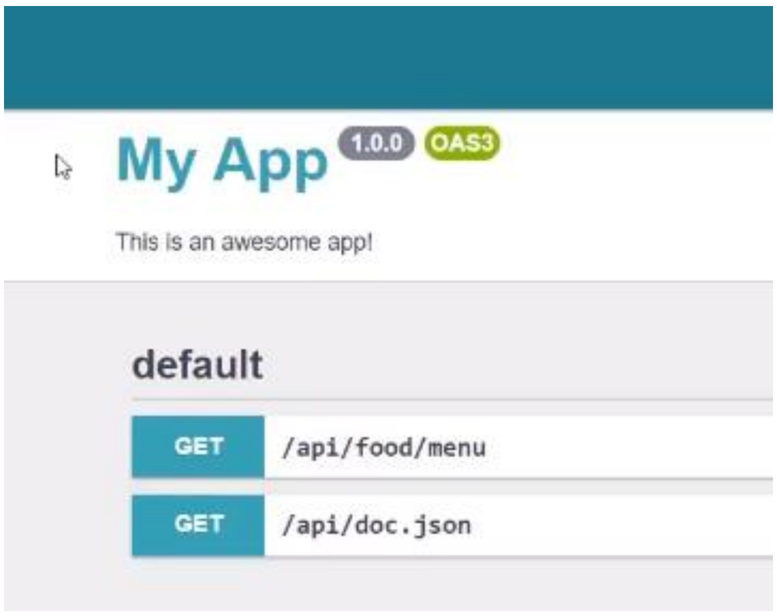
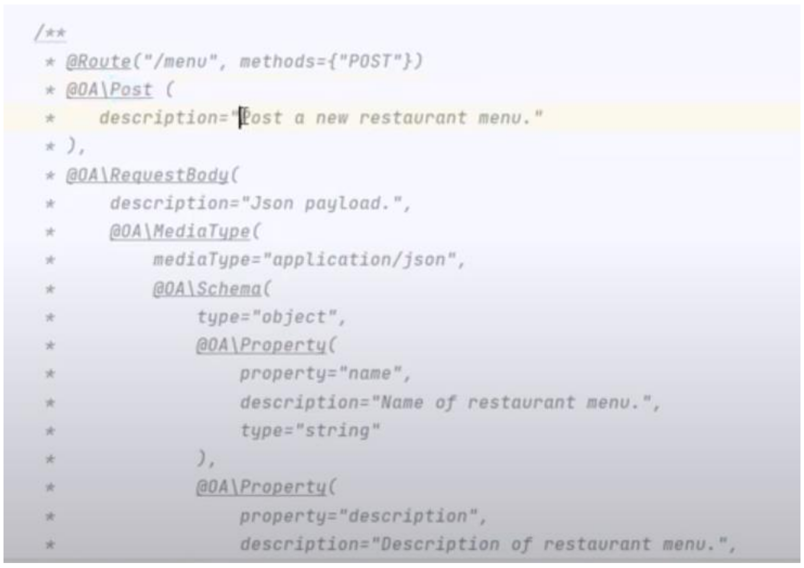


Figure 10: Swagger interface for NelmioApiDocBundle.



```
/**
 * @Route("/menu", methods={"POST"})
 * @OA\Post (
 *     description="Post a new restaurant menu."
 * ),
 * @OA\RequestBody(
 *      description="Json payload.",
 *      @OA\MediaType(
 *          mediaType="application/json",
 *          @OA\Schema(
 *              type="object",
 *              @OA\Property(
 *                  property="name",
 *                  description="Name of restaurant menu.",
 *                  type="string"
 *              ),
 *              @OA\Property(
 *                  property="description",
 *                  description="Description of restaurant menu.",
```

Figure 11: OpenAPI source code for NelmioApiDocBundle.

2.1.1.4 Swagger Maven Plugin

Swagger Maven Plugin is a plugin for java projects using maven to generate swagger API documentation while building with maven.

This plugin does not serve the online documentation after building but only generates the spec docs to be used later.

For using it, it's only necessary to add it in the plugins block by writing the plugin definition:

```
<plugin>
<groupId>com.github.kongchen</groupId>
<artifactId>swagger-maven-plugin</artifactId>
<version>${swagger-maven-plugin-version}</version>
<configuration>
<apiSources>
<apiSource>
...
</apiSource>
</apiSources>
</configuration>
<executions>
<execution>
<phase>compile</phase>
<goals>
<goal>generate</goal>
</goals>
</execution>
</executions>
</plugin>
```

And edit the pom.xml file to add the dependency:

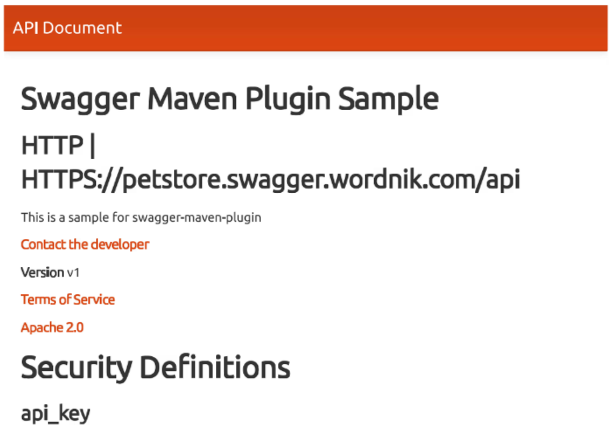
```
<dependency>
  <groupId>javax.xml.bind</groupId>
  <artifactId>jaxb-api</artifactId>
  <version>2.3.1</version>
</dependency>
```

Installation is complete at this point and just run the `mvn compile` command to generate the swagger.

As this plugin does not serve the web with generated documentation, it is necessary to expose it somehow. Nginx or Apache can be used, for example. The following command allows to get it up with docker:

docker run -it --rm -d -p 8080:80 --name web -v /home/adrian/swagger-maven-example/generated/:/usr/share/nginx/html nginx

Exposed documentation would be like this:


API Document

Swagger Maven Plugin Sample

HTTP |
HTTPS://petstore.swagger.wordnik.com/api

This is a sample for swagger-maven-plugin
Contact the developer
Version v1
Terms of Service
Apache 2.0

Security Definitions

api_key

Figure 12: Swagger maven plugin main documentation web.

2.1.1.5 drf-yasg (Yet another Swagger generator)

It's a real **swagger/openAPI** specifications generator for **Django REST APIs**. Provides the option to choose between swagger-UI and redoc or both for documentation generation but it's only working with OpenAPI 2.0 and has no support for OpenAPI 3.0 and it's unplanned to give support for 3.0 in short term.

This tool is installed using python-pip and requires python to be installed in one of the "3.6,3.7,3.8,3.9" versions. Once installed, only need to run the following command to install drf-yasg:

pip install -U drf-yasg

Once installed, we should add it to the requirements.txt file by adding the following two lines*:*

```
djangorestframework==3.11
drf-yasg==1.20
```

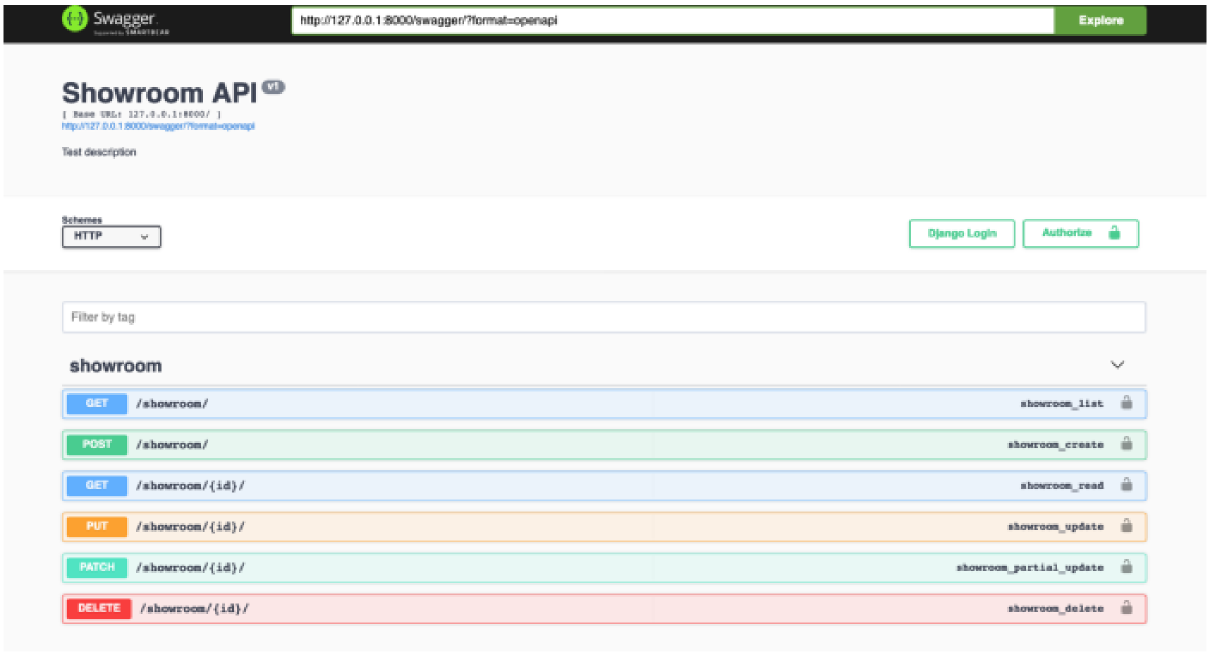
The next step is to add the following to the INSTALLED_APPS variable in settings.py:

```
INSTALLED_APPS=[
...
'rest_framework',
'django.contrib.staticfiles',
'drf_yasg',
...
]
```

In the **urls.py** file, the following lines must be added to enable the WebUI for swagger. In urlpatterns, we added two different views, redoc and swagger to make them available as examples. Two exposed endpoints show a *JSON* and a *YAML* representation of the API specification.

```
schema_view = get_schema_view(
   openapi.Info(
        title="Snippets API",
        default_version='v1',
        description="Test description",
        terms_of_service="https://www.google.com/policies/terms/",
        contact=openapi.Contact(email="contact@snippets.local"),
        license=openapi.License(name="BSD License"),
    ),
    public=True,
    permission_classes=[permissions.AllowAny],
)

urlpatterns = [
    re_path(r'^swagger(?P<format>\.json|\.yaml)$',
schema_view.without_ui(cache_timeout=0), name='schema-json'),
    re_path(r'^swagger/$', schema_view.with_ui('swagger', cache_timeout=0),
name='schema-swagger-ui'),
    re_path(r'^redoc/$', schema_view.with_ui('redoc', cache_timeout=0),
name='schema-redoc'),
    ...
]
```

With these changes, the API is exposed in /swagger/ as shown:


Swagger
http://127.0.0.1:8000/swagger/?format=openapi
Explore
Showroom API
v1
[Base URL: 127.0.0.1:8000/]
http://127.0.0.1:8000/swagger/?format=openapi
Test description
Schemes
HTTP
Django Login
Authorize
Filter by tag
showroom

```
GET     /showroom/              showroom_list
POST    /showroom/              showroom_create
GET     /showroom/{id}/         showroom_read
PUT     /showroom/{id}/         showroom_update
PATCH   /showroom/{id}/         showroom_partial_update
DELETE  /showroom/{id}/         showroom_delete
```

Figure 13: Swagger endpoint in drf-yasg.

2.1.1.6 HAPI swagger

OpenAPI generator plugin for HAPI to self-document the API interface for JavaScript objects.

nodeJS or npm must be installed to install this tool. Once installed, only need to run the following command to install hapi-swagger:

npm install hapi-swagger --save

npx install-peerdeps hapi-swagger

After installation, modules must be imported into the JavaScript code and endpoints tagged by ***tags: ['api'],*** in the *options* section of the code.

After running the application again, API documentation is generated in /documentation/ route.

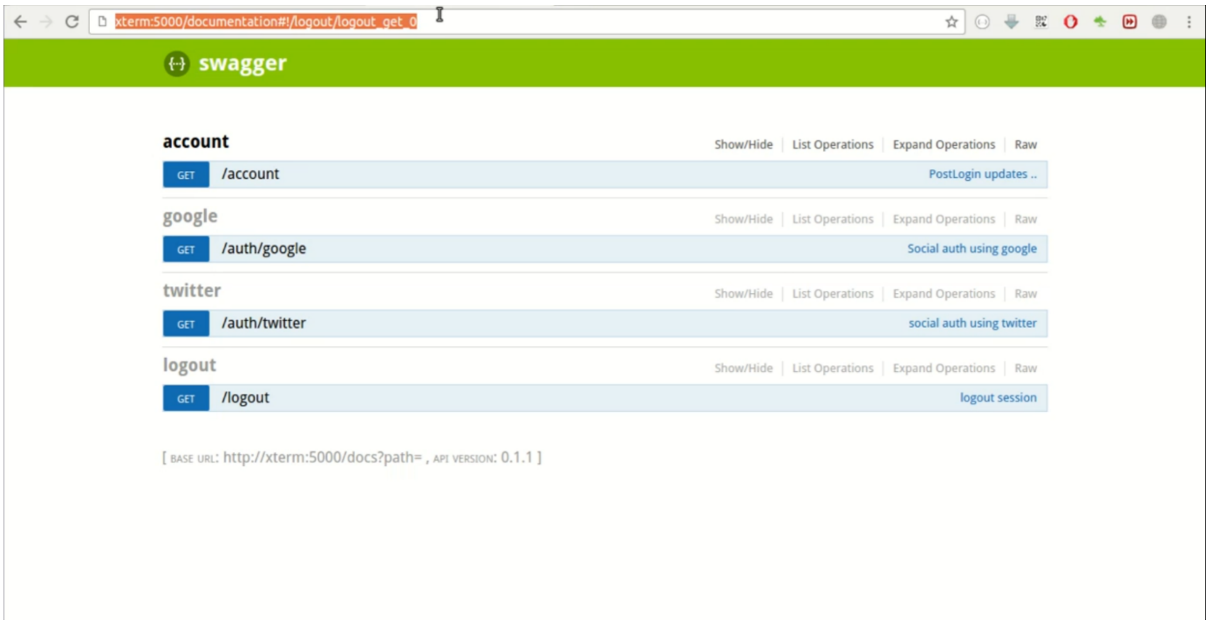
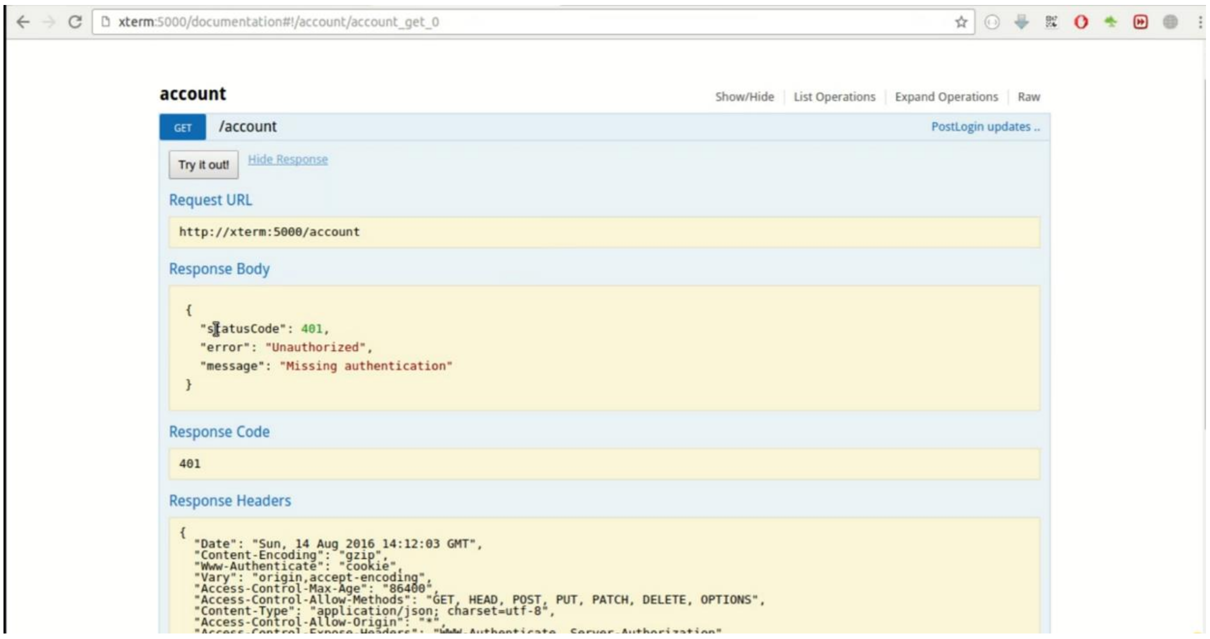

xterm:5000/documentation#!/logout/logout_get_0
swagger
account
Show/Hide | List Operations | Expand Operations | Raw
GET /account
PostLogin updates ..
google
GET /auth/google
Social auth using google
twitter
GET /auth/twitter
social auth using twitter
logout
GET /logout
logout session
[BASE URL: http://xterm:5000/docs?path= , API VERSION: 0.1.1]

Figure 14: HAPI swagger view.


xterm:5000/documentation#!/account/account_get_0
account
Show/Hide | List Operations | Expand Operations | Raw
GET /account
PostLogin updates ..
Try it out!
Hide Response

Request URL

```
http://xterm:5000/account
```

Response Body

```
{
  "statusCode": 401,
  "error": "Unauthorized",
  "message": "Missing authentication"
}
```

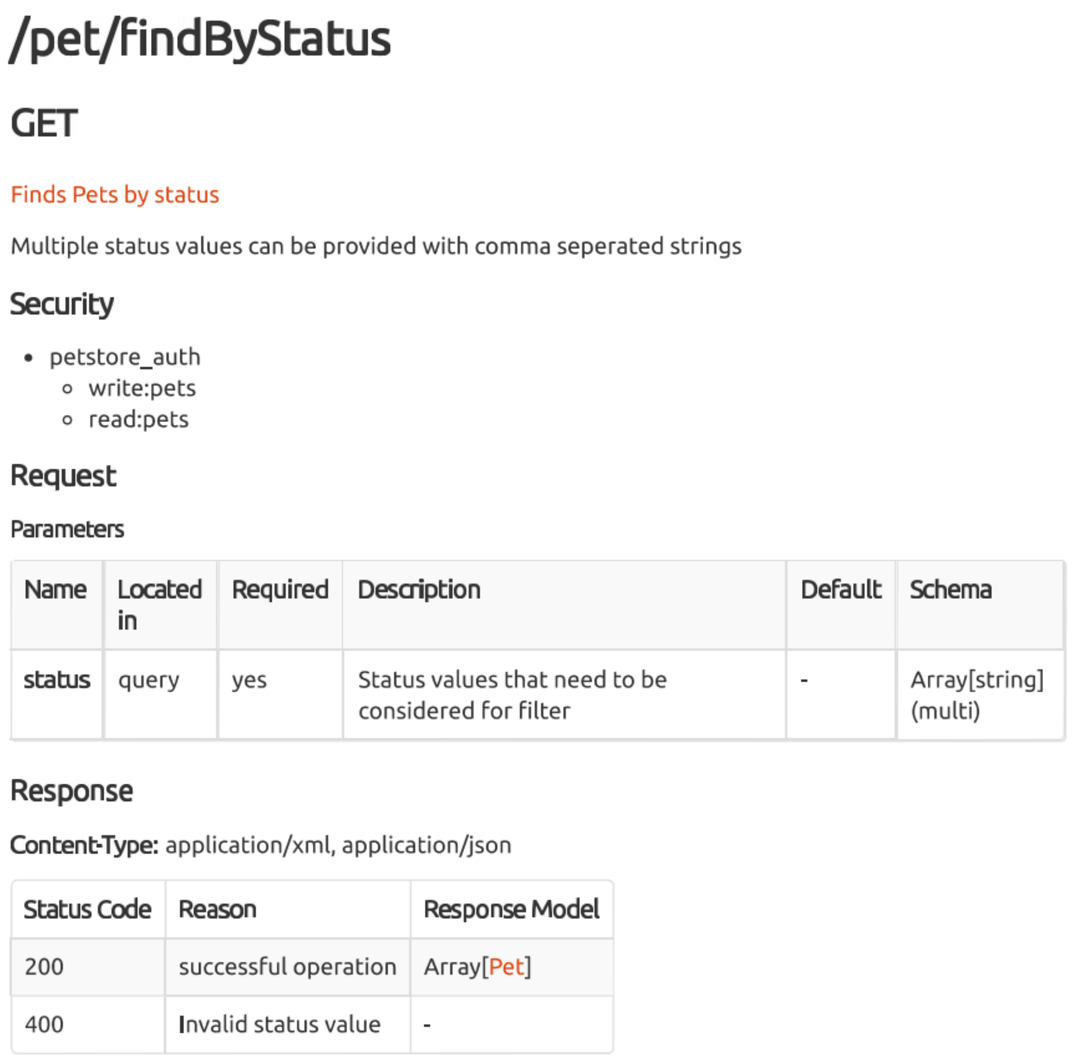
Response Code

```
401
```

Response Headers

```
{
  "Date": "Sun, 14 Aug 2016 14:12:03 GMT",
  "Content-Encoding": "gzip",
  "Www-Authenticate": "cookie",
  "Vary": "origin,accept-encoding",
  "Access-Control-Max-Age": "86400",
  "Access-Control-Allow-Methods": "GET, HEAD, POST, PUT, PATCH, DELETE, OPTIONS",
  "Content-Type": "application/json; charset=utf-8",
  "Access-Control-Allow-Origin": "*",
  "Access-Control-Expose-Headers": "WWW-Authenticate, Server-Authorization"
```

Figure 15: Swagger endpoint details for the ‘account’ object in hapi-swagger.



/pet/findByStatus

GET

Finds Pets by status

Multiple status values can be provided with comma seperated strings

Security

- petstore_auth
  - write:pets
  - read:pets

Request

Parameters

Name	Located in	Required	Description	Default	Schema
status	query	yes	Status values that need to be considered for filter	-	Array[string] (multi)

Response

Content-Type: application/xml, application/json

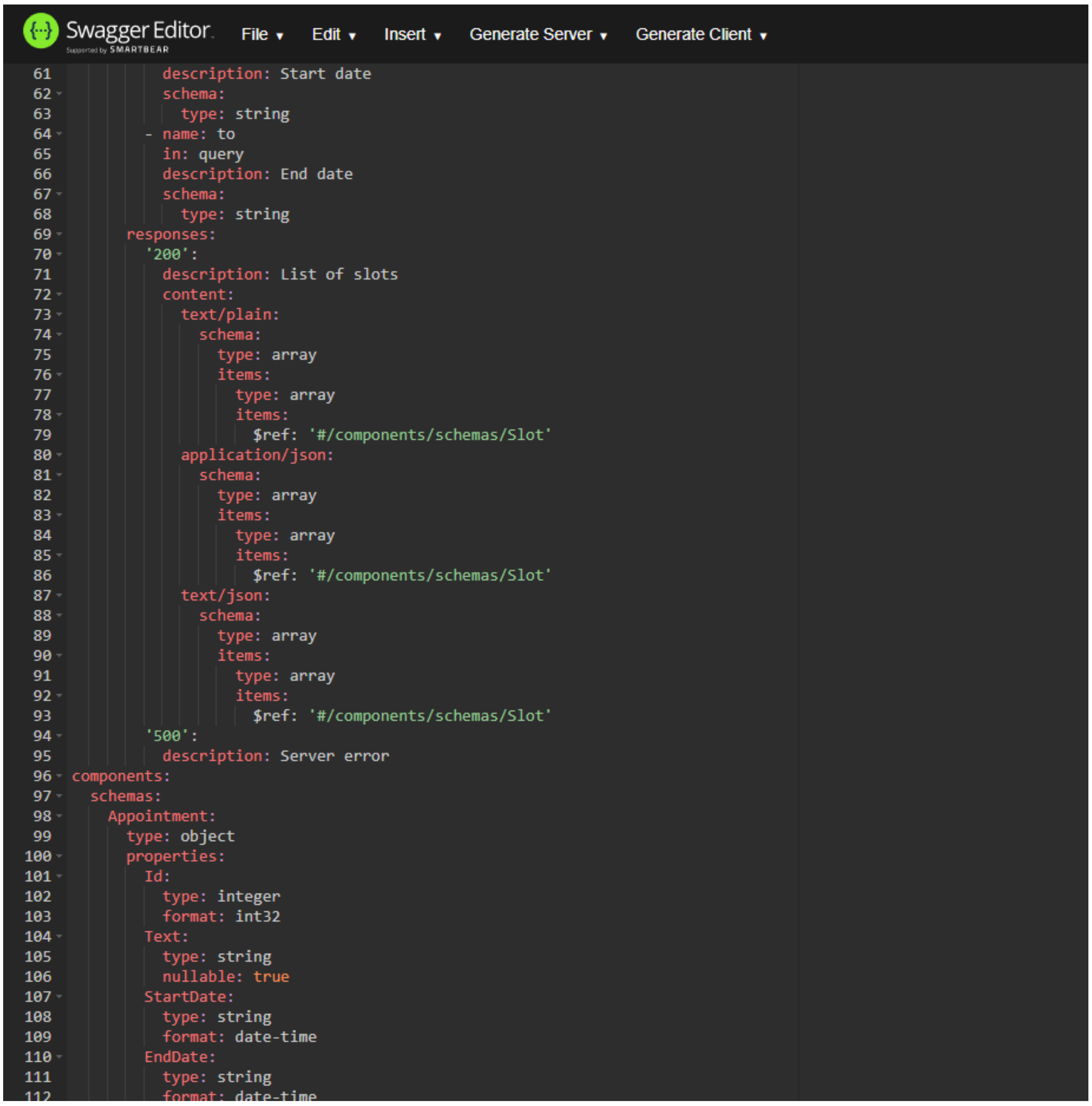
Status Code	Reason	Response Model
200	successful operation	Array[Pet]
400	Invalid status value	-

Figure 16: Swagger maven plugin API call example.

2.1.2 How to edit an OpenAPI

With Swagger editor you can perform a visual check of the elements that make up the API. At the same time, it allows you to identify code errors, indicating the line where the error is located. This editor requires specific knowledge of the OpenAPI standard.

This tool also has a wide range of possibilities in the field of testing. It allows the server part to be generated to make the calls defined in the API. At the same time, we can generate a client from a wide variety of languages to include it in the tool being developed.



```
              description: Start date
              schema:
                type: string
            - name: to
              in: query
              description: End date
              schema:
                type: string
          responses:
            '200':
              description: List of slots
              content:
                text/plain:
                  schema:
                    type: array
                    items:
                      type: array
                      items:
                        $ref: '#/components/schemas/Slot'
                application/json:
                  schema:
                    type: array
                    items:
                      type: array
                      items:
                        $ref: '#/components/schemas/Slot'
                text/json:
                  schema:
                    type: array
                    items:
                      type: array
                      items:
                        $ref: '#/components/schemas/Slot'
            '500':
              description: Server error
components:
  schemas:
    Appointment:
      type: object
      properties:
        Id:
          type: integer
          format: int32
        Text:
          type: string
          nullable: true
        StartDate:
          type: string
          format: date-time
        EndDate:
          type: string
          format: date-time
```

Figure 17: OpenAPI source code.

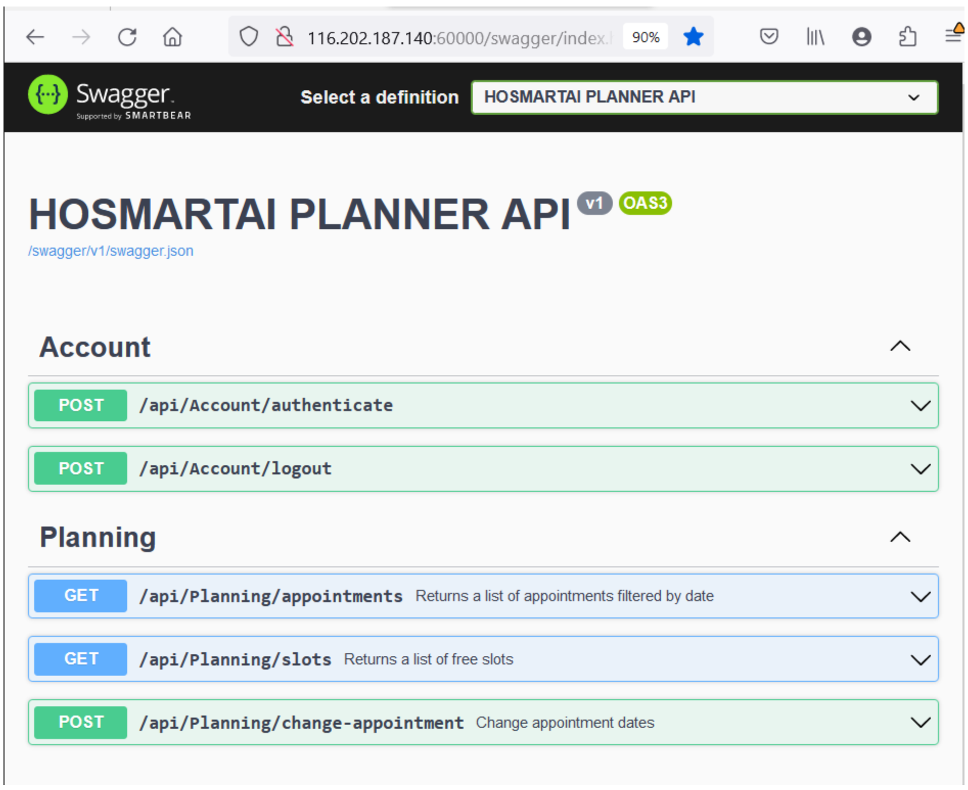
3 OpenAPI Specification

The following is the final OpenAPIs specification of the main elements that require the use of these tools to communicate.

After the work done in the previous versions analysing the tools, this deliverable shows in a clear way how they have been integrated and what is their final specification.

3.1.1 HosmartAI Planner

For this task, a Web Services (WS) system has been developed that allows authentication in the system and the execution of Get and POST requests, among others. The following image shows the Swagger system, which enables users and technical personnel to interact with the data exchange service securely http://116.202.187.140:60000. It is worth noting that the authentication system is based on token authentication, which is an HTTP authentication scheme where security relies on the use of encrypted text strings, typically generated by the server. These strings (tokens) identify the bearer of the message by including them in all resource requests made to the server.


116.202.187.140:60000/swagger/index

Select a definition: HOSMARTAI PLANNER API

HOSMARTAI PLANNER API v1 OAS3

/swagger/v1/swagger.json

Account

POST /api/Account/authenticate

POST /api/Account/logout

Planning

GET /api/Planning/appointments Returns a list of appointments filtered by date

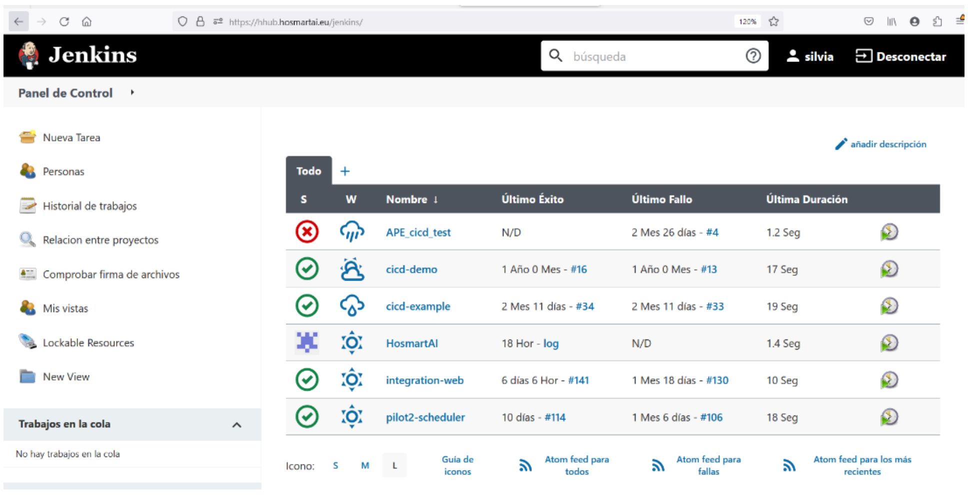
GET /api/Planning/slots Returns a list of free slots

POST /api/Planning/change-appointment Change appointment dates

Figure 18: Planner swagger.

In this case, the data that needs to be exchanged includes patient authentication and APIs with other elements of the pilot program, as well as information related to medical appointments and changes made by patients.

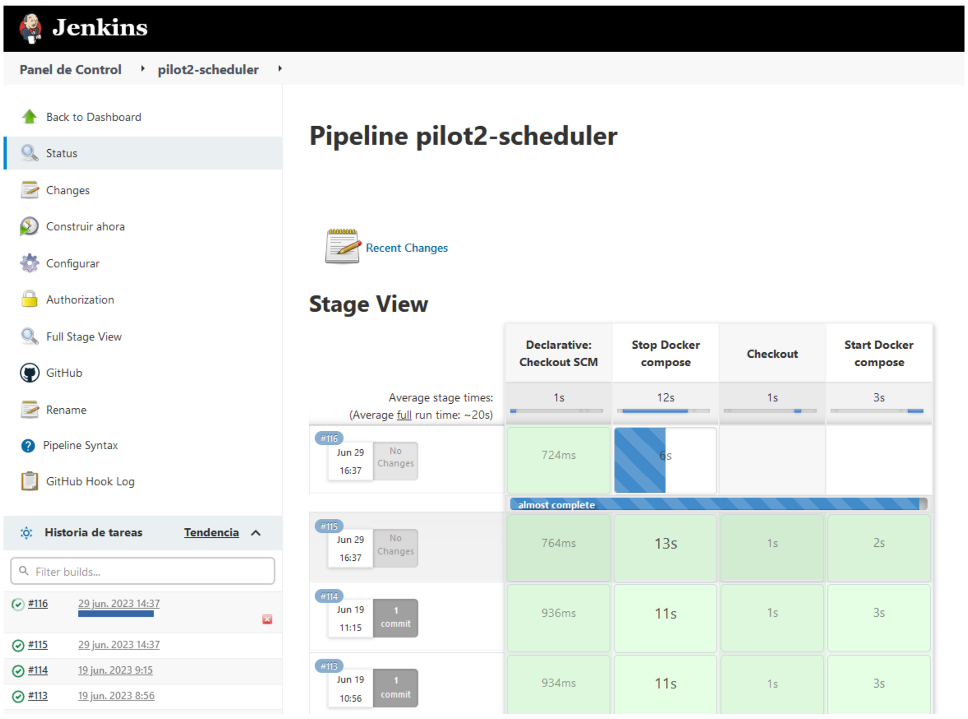
The system is deployed on the general project platform, making it available to all partners. The deployment process can be observed through Jenkins (https://hhub.hosmartai.eu/jenkins/).



S	W	Nombre ↓	Último Éxito	Último Fallo	Última Duración
✗		APE_cicd_test	N/D	2 Mes 26 días - #4	1.2 Seg
✓		cicd-demo	1 Año 0 Mes - #16	1 Año 0 Mes - #13	17 Seg
✓		cicd-example	2 Mes 11 días - #34	2 Mes 11 días - #33	19 Seg
		HosmartAI	18 Hor - log	N/D	1.4 Seg
✓		integration-web	6 días 6 Hor - #141	1 Mes 18 días - #130	10 Seg
✓		pilot2-scheduler	10 días - #114	1 Mes 6 días - #106	18 Seg

Figure 19: Jenkins.

Jenkins is a server for continuous integration. This tool is used to build and test software projects continuously, making it easier for developers to integrate changes into a project and deliver new versions to users. As shown in the following image, there is a history of deployed versions and changes, the time elapsed since the last deployed version, and various logs that help identify possible system failures.



#116	29 jun. 2023 14:37		
#115	29 jun. 2023 14:37		
#114	19 jun. 2023 9:15		
#113	19 jun. 2023 8:56		

	Declarative: Checkout SCM	Stop Docker compose	Checkout	Start Docker compose
Average stage times: (Average full run time: ~20s)	1s	12s	1s	3s
#116 Jun 29 16:37 No Changes	724ms	6s		
#115 Jun 29 16:37 No Changes	764ms	13s	1s	2s
#114 Jun 19 11:15 1 commit	936ms	11s	1s	3s
#113 Jun 19 10:56 1 commit	934ms	11s	1s	3s

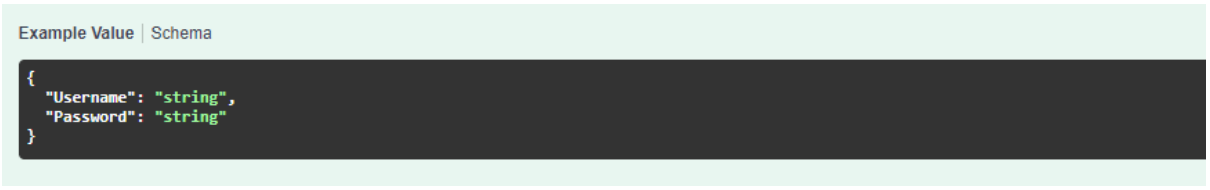
Figure 20: Jenkins pipeline.

3.1.2 Method Explanation

This section describes the breakdown of the full functionality of this pilot. All functionality is achieved through the methods specified in the OpenAPI. They are detailed below.

Account

1. **POST /api/Account/authenticate**
   - **Detailed Description:** Initiates the authentication process for a user. Users provide necessary credentials, and upon successful authentication, they gain access to protected resources or services.
   - **Example**



```
{
  "Username": "string",
  "Password": "string"
}
```

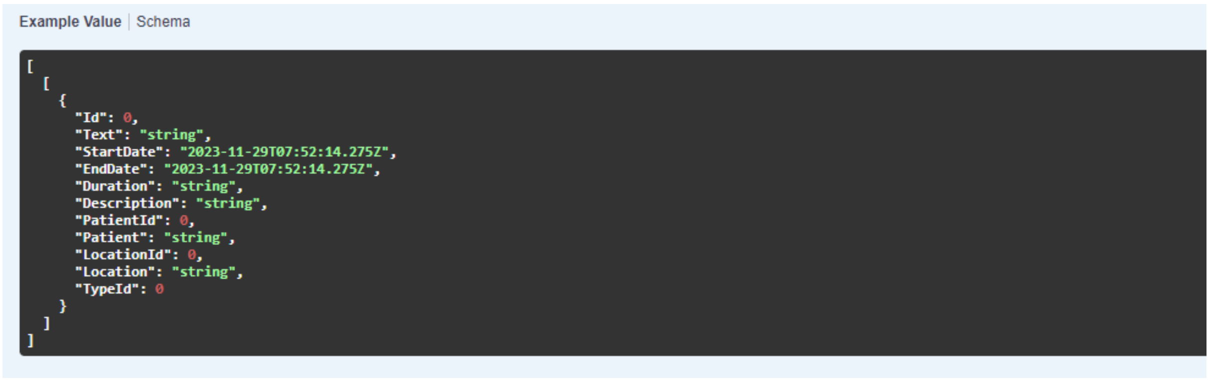
2. **POST /api/Account/logout**
   - **Detailed Description:** Logs out the currently authenticated user. This endpoint terminates the user's session, ensuring that subsequent requests require re-authentication for access.
   - **Example**

No parameters

Planning

3. **GET /api/Planning/appointments**

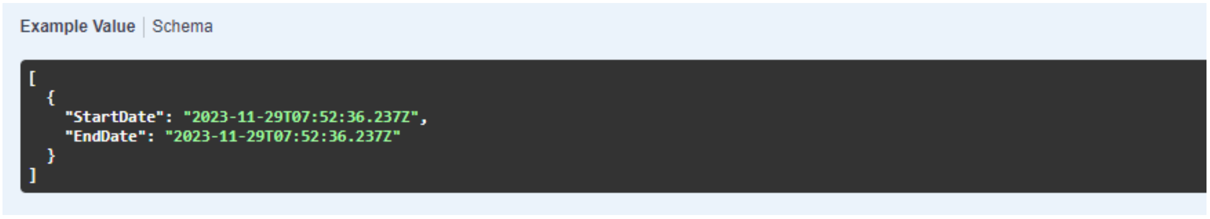
   - **Detailed Description:** Retrieves a list of appointments filtered by date. Users can query the API to obtain appointments scheduled for a specific date, facilitating efficient planning and organization.
   - **Example**


Example Value | Schema

```
[
  [
    {
      "Id": 0,
      "Text": "string",
      "StartDate": "2023-11-29T07:52:14.275Z",
      "EndDate": "2023-11-29T07:52:14.275Z",
      "Duration": "string",
      "Description": "string",
      "PatientId": 0,
      "Patient": "string",
      "LocationId": 0,
      "Location": "string",
      "TypeId": 0
    }
  ]
]
```

4. **GET /api/Planning/slots**

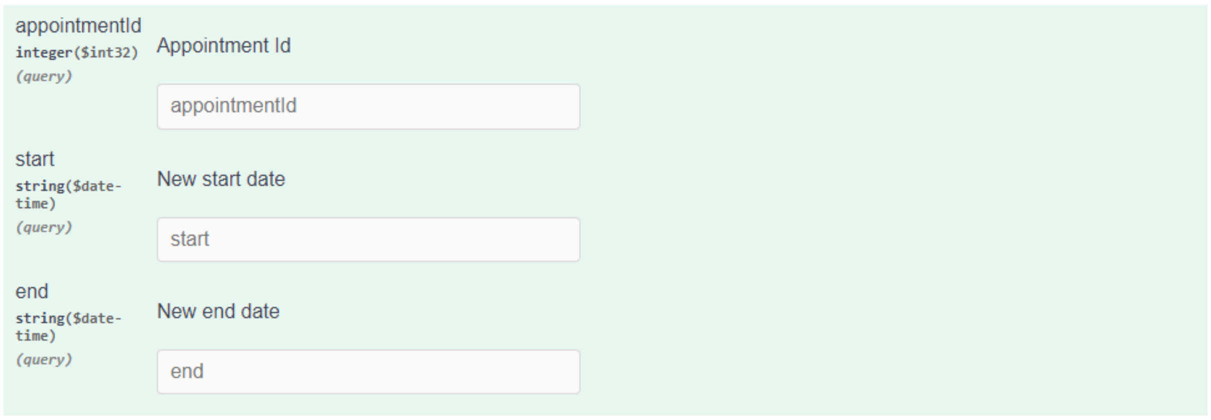
   - **Detailed Description:** Retrieves a list of available time slots for appointments. This endpoint allows users to identify and choose from open slots, streamlining the appointment scheduling process.
   - **Example**


Example Value | Schema

```
[
  {
    "StartDate": "2023-11-29T07:52:36.237Z",
    "EndDate": "2023-11-29T07:52:36.237Z"
  }
]
```

5. **POST /api/Planning/change-appointment**

   - **Detailed Description:** Facilitates the modification of appointment dates. Users can submit a request to change the date of an existing appointment through this endpoint, providing flexibility in managing scheduled events.
   - **Example**

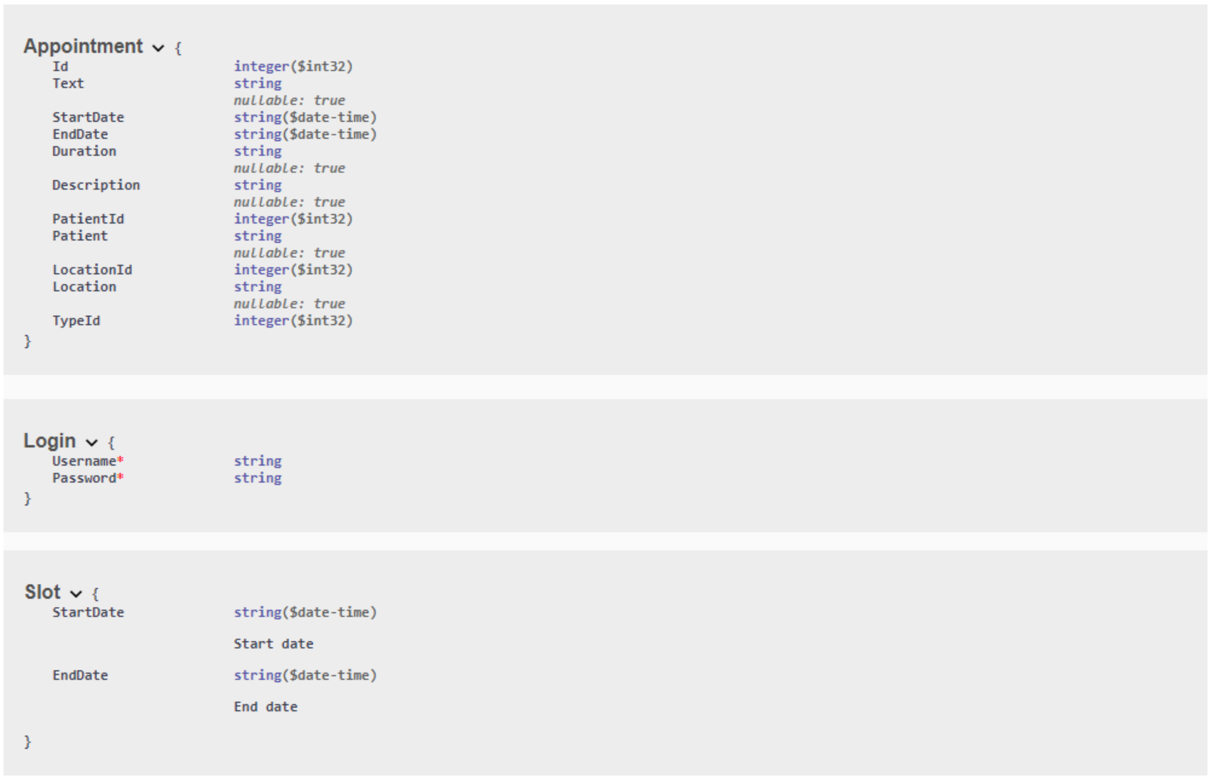


appointmentId integer($int32) *(query)*	Appointment Id	appointmentId
start string($date-time) *(query)*	New start date	start

end
string($date-time)
(query)
	New end date	end

3.1.3 Schemas

A schema defines how data should be organised and what type of data can be included. It provides a formal description of the data structure so that applications consuming the API can understand and process information consistently.



```
Appointment ⌄ {
    Id                     integer($int32)
    Text                   string
                           nullable: true
    StartDate              string($date-time)
    EndDate                string($date-time)
    Duration               string
                           nullable: true
    Description            string
                           nullable: true
    PatientId              integer($int32)
    Patient                string
                           nullable: true
    LocationId             integer($int32)
    Location               string
                           nullable: true
    TypeId                 integer($int32)
}

Login ⌄ {
    Username*              string
    Password*              string
}

Slot ⌄ {
    StartDate              string($date-time)

                           Start date

    EndDate                string($date-time)

                           End date

}
```

Figure 21: Schema for planner.

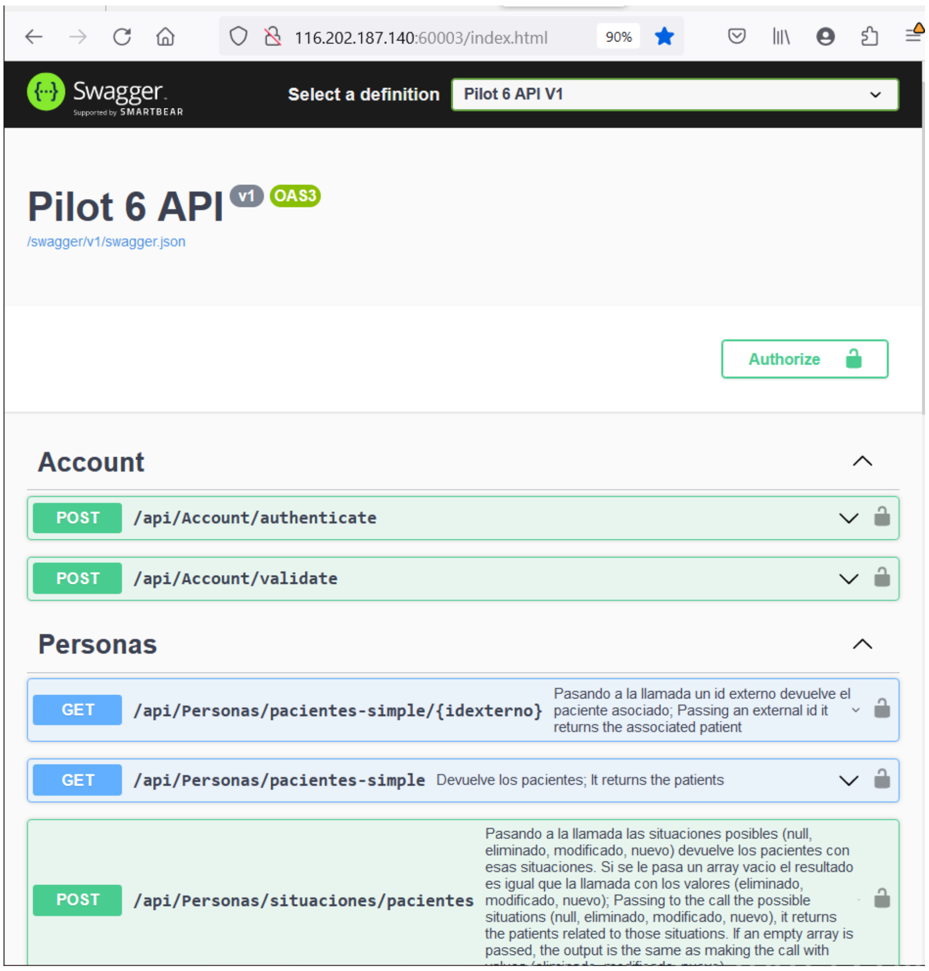
As can be seen in the schema, the data have all the necessary fields for their identification and organization of the system.

3.2 HosmartAI User Management for Pilot 6

For this task, a Web Services (WS) system has been developed that allows authentication in the system and the execution of GET, PUT and POST requests, among others. The following image shows the Swagger system, which enables users and technical personnel to interact with the data exchange service securely (http://116.202.187.140:60003).

In Swagger, you can see a description and objective of the different publicly deployed methods. However, there are other private methods between the Web interface and the Web Service that are not available on the Swagger platform.

It is worth noting that the authentication system is based on token authentication, which is an HTTP authentication scheme where security relies on the use of encrypted text strings, typically generated by the server. These strings (tokens) identify the bearer of the message by including them in all resource requests made to the server.


116.202.187.140:60003/index.html

Swagger — Select a definition: Pilot 6 API V1

Pilot 6 API v1 OAS3
/swagger/v1/swagger.json

Authorize

Account

POST /api/Account/authenticate

POST /api/Account/validate

Personas

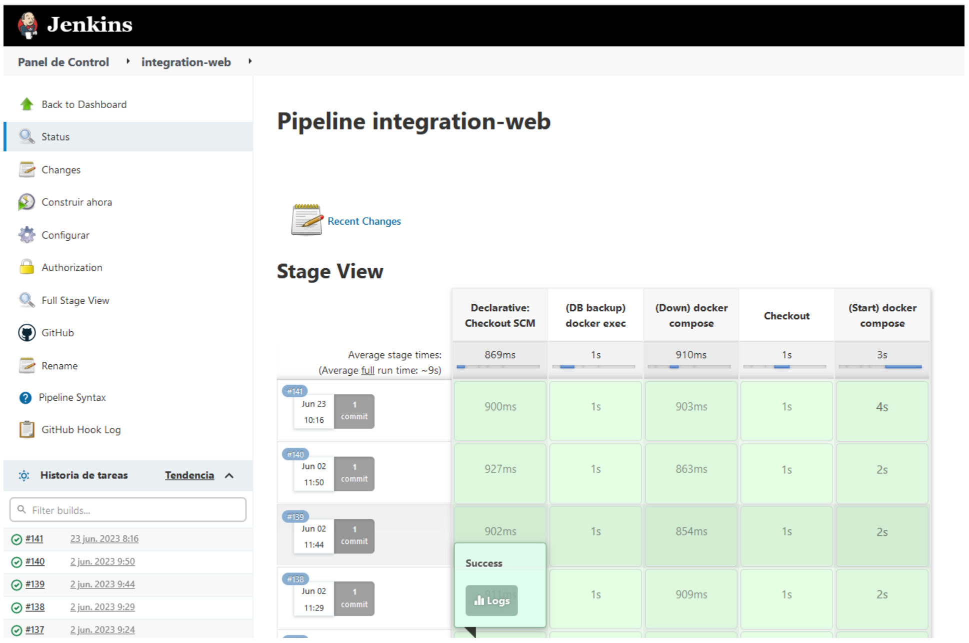
GET /api/Personas/pacientes-simple/{idexterno} Pasando a la llamada un id externo devuelve el paciente asociado; Passing an external id it returns the associated patient

GET /api/Personas/pacientes-simple Devuelve los pacientes; It returns the patients

POST /api/Personas/situaciones/pacientes Pasando a la llamada las situaciones posibles (null, eliminado, modificado, nuevo) devuelve los pacientes con esas situaciones. Si se le pasa un array vacio el resultado es igual que la llamada con los valores (eliminado, modificado, nuevo); Passing to the call the possible situations (null, eliminado, modificado, nuevo), it returns the patients related to those situations. If an empty array is passed, the output is the same as making the call with

Figure 22: Swagger for Pilot 6.

The deployment system used has been Jenkins, which allows us to view logs and maintain a historical record of the different executions performed. For this purpose, Jenkins is connected to the code repository where the various codes with functionalities have been updated.



Pipeline integration-web

Stage View

	Declarative: Checkout SCM	(DB backup) docker exec	(Down) docker compose	Checkout	(Start) docker compose
Average stage times: (Average full run time: ~9s)	869ms	1s	910ms	1s	3s
#141 Jun 23 10:16 1 commit	900ms	1s	903ms	1s	4s
#140 Jun 02 11:50 1 commit	927ms	1s	863ms	1s	2s
#139 Jun 02 11:44 1 commit	902ms	1s	854ms	1s	2s
#138 Jun 02 11:29 1 commit		1s	909ms	1s	2s

#141	23 jun. 2023 8:16
#140	2 jun. 2023 9:50
#139	2 jun. 2023 9:44
#138	2 jun. 2023 9:29
#137	2 jun. 2023 9:24

Figure 23: Jenkins pipeline in Pilot 6.

3.2.1 Method Explanation

All functionality is achieved through the methods specified in the OpenAPI. They are detailed below.

Account

1. **POST /api/Account/authenticate**

   - **Detailed Description:** Initiates the authentication process for a user, typically requiring credentials. Upon successful authentication, it provides access to protected resources or services.
   - **Example**

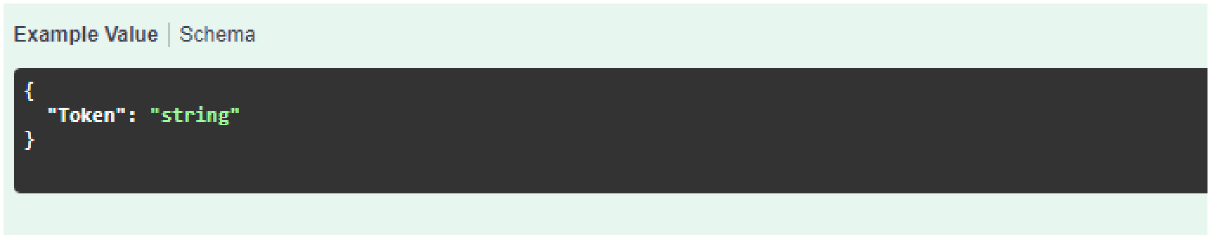
Example Value | Schema

```
{
  "Username": "string",
  "Password": "string"
}
```

2. **POST /api/Account/validate**

   - **Detailed Description:** Validates account information, ensuring the provided data meets the required criteria. This endpoint may be used for verifying user details or ensuring the integrity of account-related information.

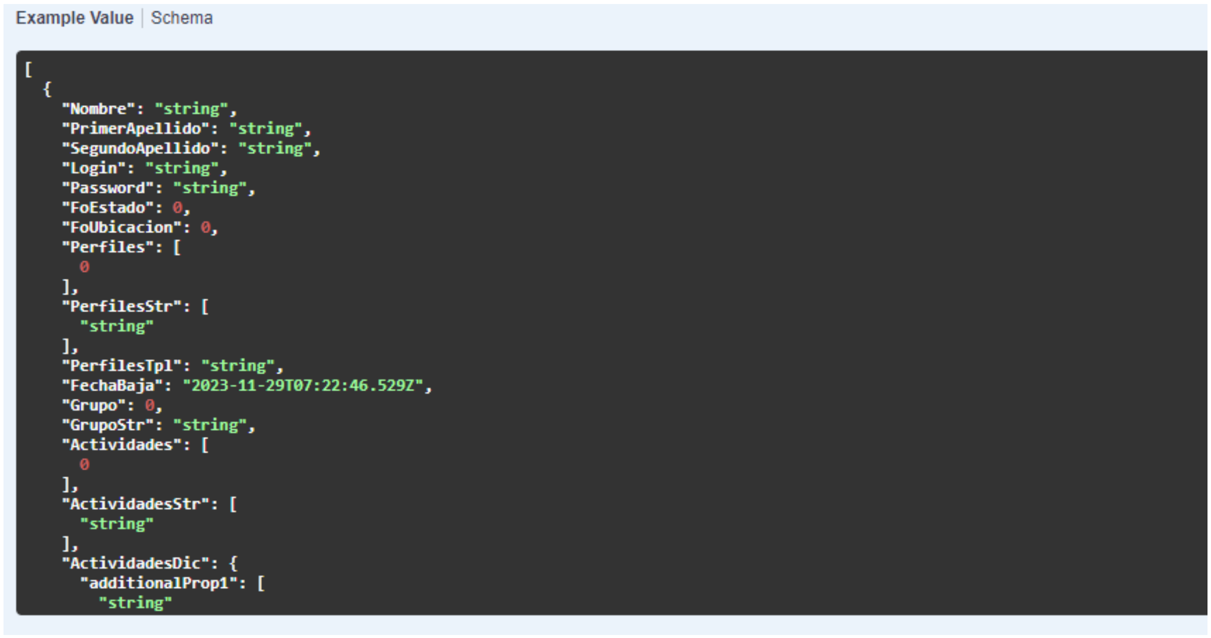
- **Example**


Example Value | Schema

```
{
  "Token": "string"
}
```

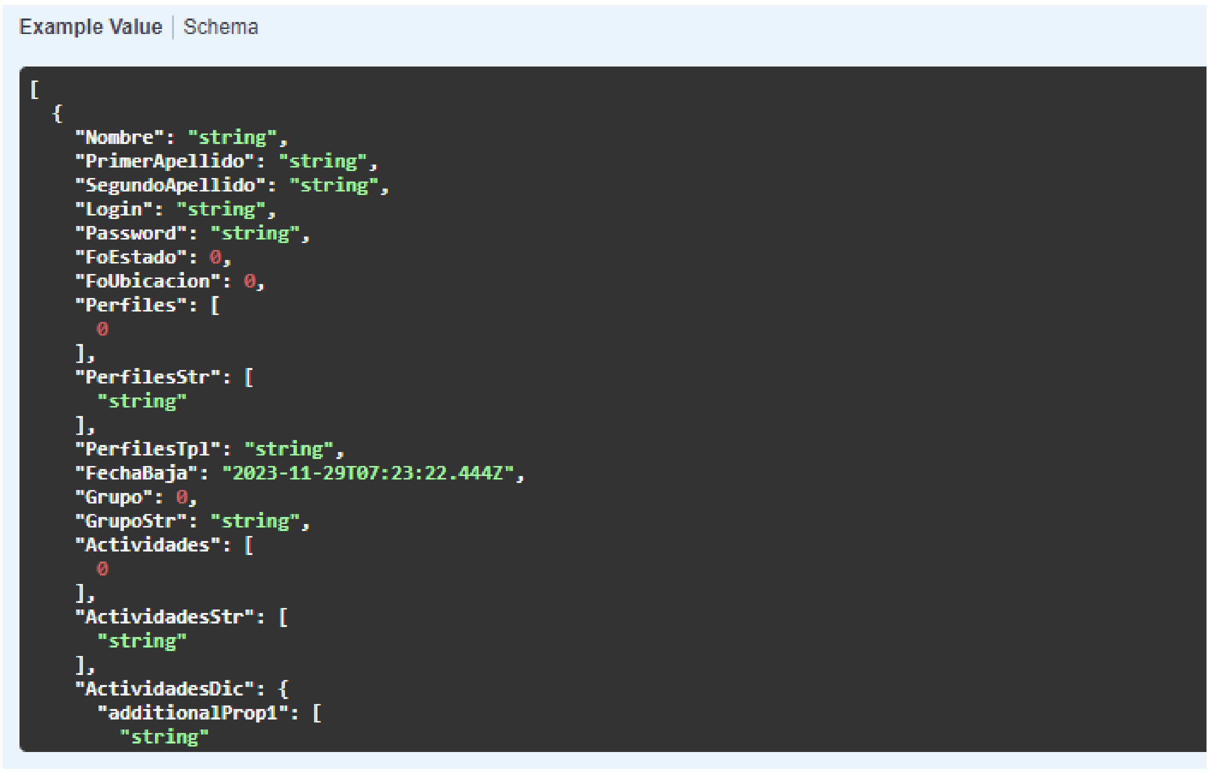
Personas

3. **GET /api/Personas/pacientes-simple/{idexterno}**
   - **Detailed Description:** Retrieves detailed information about a patient associated with the provided external ID. This can include patient demographics, medical history, and other relevant data.
   - **Example**


Example Value | Schema

```
[
  {
    "Nombre": "string",
    "PrimerApellido": "string",
    "SegundoApellido": "string",
    "Login": "string",
    "Password": "string",
    "FoEstado": 0,
    "FoUbicacion": 0,
    "Perfiles": [
      0
    ],
    "PerfilesStr": [
      "string"
    ],
    "PerfilesTpl": "string",
    "FechaBaja": "2023-11-29T07:22:46.529Z",
    "Grupo": 0,
    "GrupoStr": "string",
    "Actividades": [
      0
    ],
    "ActividadesStr": [
      "string"
    ],
    "ActividadesDic": {
      "additionalProp1": [
        "string"
```

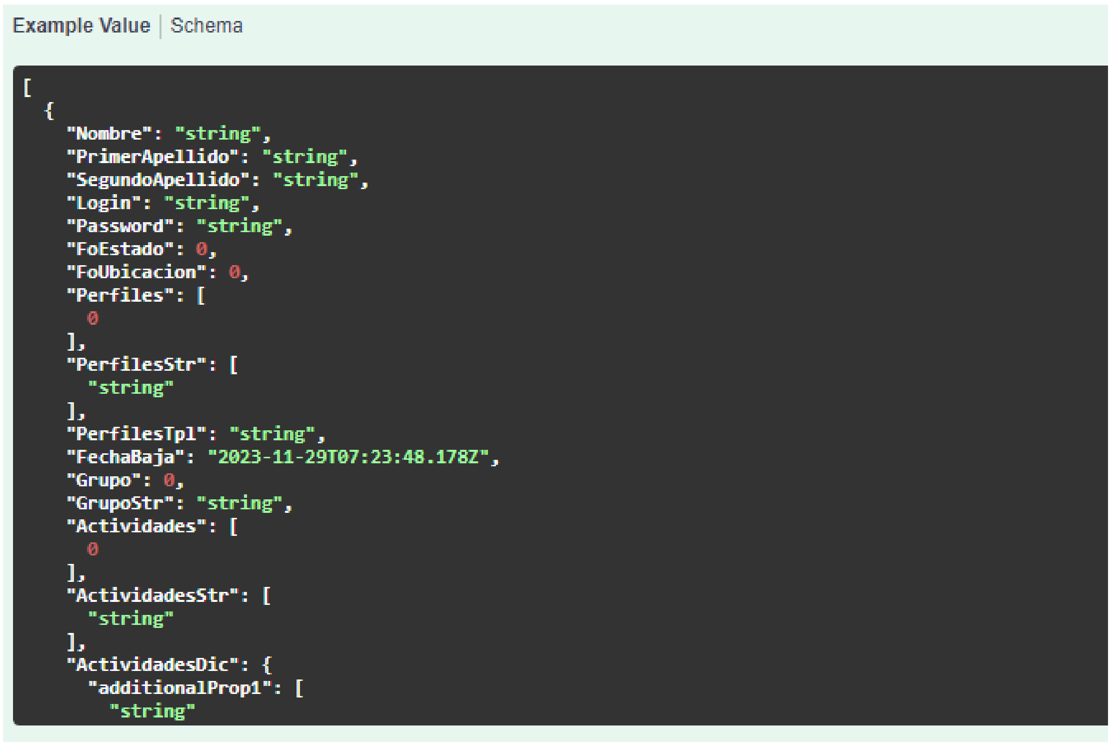
4. **GET /api/Personas/pacientes-simple**
   - **Detailed Description:** Retrieves a comprehensive list of all patients. The response would typically include details such as patient names, IDs, and other relevant information.
   - **Example**



Example Value | Schema

```
[
  {
    "Nombre": "string",
    "PrimerApellido": "string",
    "SegundoApellido": "string",
    "Login": "string",
    "Password": "string",
    "FoEstado": 0,
    "FoUbicacion": 0,
    "Perfiles": [
      0
    ],
    "PerfilesStr": [
      "string"
    ],
    "PerfilesTpl": "string",
    "FechaBaja": "2023-11-29T07:23:22.444Z",
    "Grupo": 0,
    "GrupoStr": "string",
    "Actividades": [
      0
    ],
    "ActividadesStr": [
      "string"
    ],
    "ActividadesDic": {
      "additionalProp1": [
        "string"
```

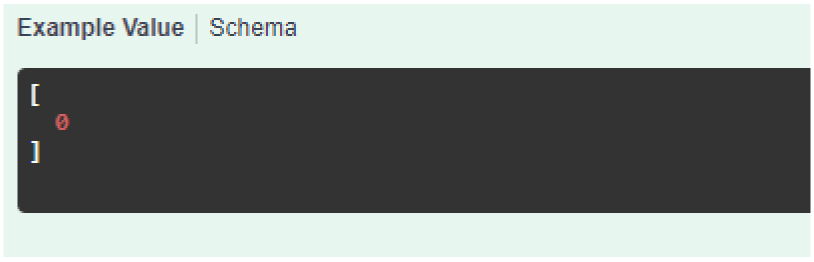
5. **POST /api/Personas/situaciones/pacientes**
    - **Detailed Description:** Retrieves a list of patients based on specified situations (null, eliminado, modificado, nuevo). If an empty array is passed, the result is equivalent to the call with values (eliminado, modificado, nuevo). This allows filtering patients based on their current status or situation.
    - **Example**



Example Value | Schema

```
[
  {
    "Nombre": "string",
    "PrimerApellido": "string",
    "SegundoApellido": "string",
    "Login": "string",
    "Password": "string",
    "FoEstado": 0,
    "FoUbicacion": 0,
    "Perfiles": [
      0
    ],
    "PerfilesStr": [
      "string"
    ],
    "PerfilesTpl": "string",
    "FechaBaja": "2023-11-29T07:23:48.178Z",
    "Grupo": 0,
    "GrupoStr": "string",
    "Actividades": [
      0
    ],
    "ActividadesStr": [
      "string"
    ],
    "ActividadesDic": {
      "additionalProp1": [
        "string"
```

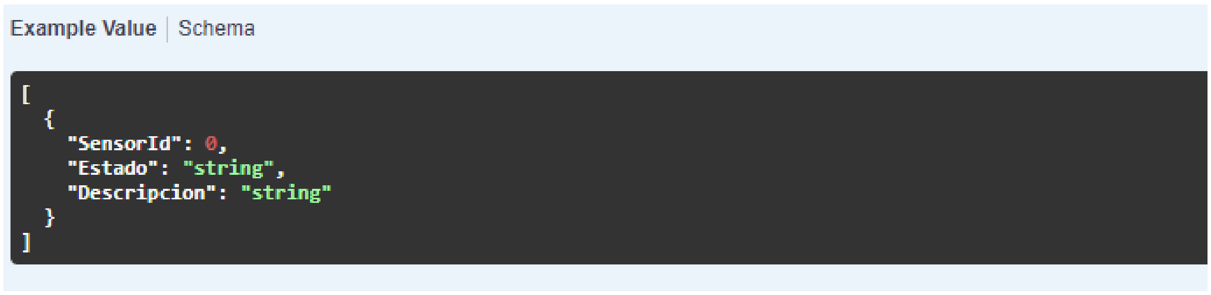
6. **POST /api/Personas/situaciones/registradas/pacientes**
   - **Detailed Description:** Modifies the situation of users related to the provided external IDs. By passing external IDs to the call, it sets the user situation value to the default (null) for users associated with those external IDs.
   - **Example**



```
[
  0
]
```

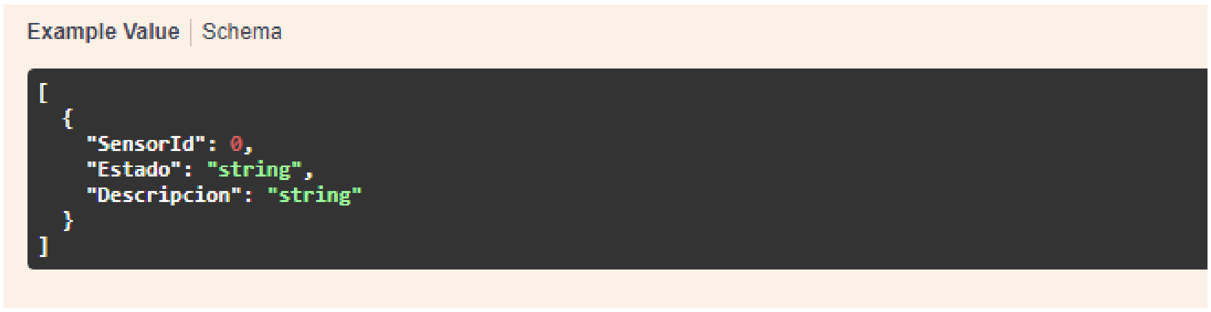
Sensors

7. **GET /api/Sensores**
   - **Detailed Description:** Retrieves a comprehensive list of all sensors. The response would typically include details such as sensor names, IDs, and other relevant information.
   - **Example**



```
[
  {
    "SensorId": 0,
    "Estado": "string",
    "Descripcion": "string"
  }
]
```

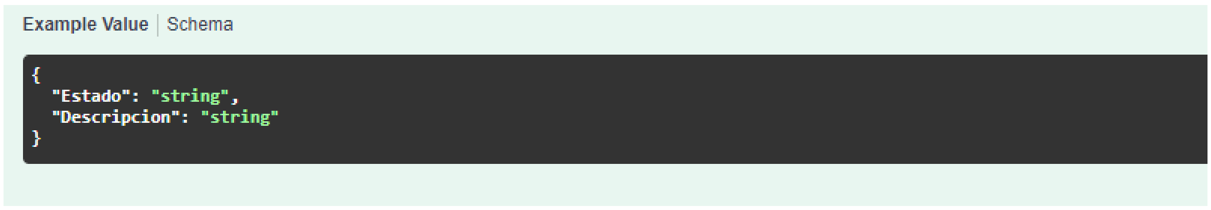
8. **PUT /api/Sensores/actualizar/{idexterno}**
   - **Detailed Description:** Updates the sensors of a patient based on the provided external ID and sensor data. This allows for real-time adjustments or additions to the sensor information associated with a specific patient.
   - **Example**



```
[
  {
    "SensorId": 0,
    "Estado": "string",
    "Descripcion": "string"
  }
]
```

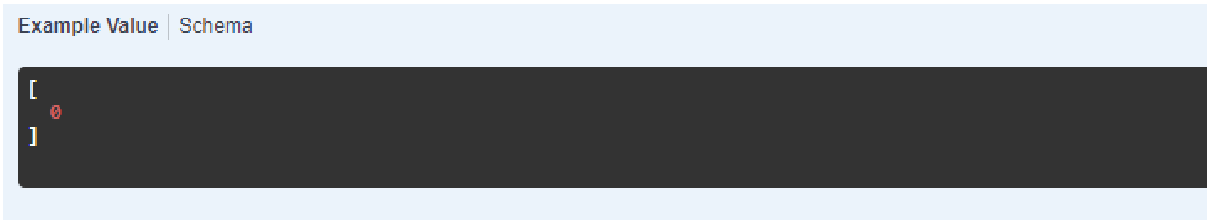
9. **POST /api/Sensores/actualizar/{idsensor}**

- **Detailed Description:** Modifies sensor values by passing the sensor ID and new values. This endpoint is useful for updating or calibrating sensor readings for a specific sensor.
- **Example**


Example Value | Schema

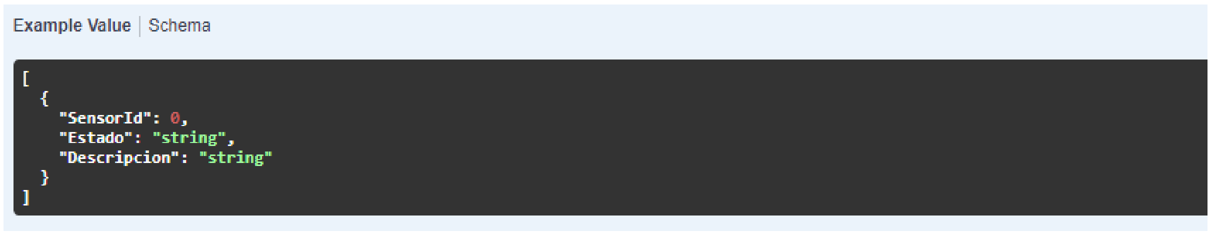
```
{
  "Estado": "string",
  "Descripcion": "string"
}
```

10. **GET /api/Sensores/sensor-id/{id}**
    - **Detailed Description:** Retrieves the external ID of the patient associated with the specified sensor. This can be valuable for cross-referencing sensor data with patient information.
    - **Example**


Example Value | Schema

```
[
  0
]
```

11. **GET /api/Sensores/persona/{idexterno}**
    - **Detailed Description:** Retrieves sensors associated with a user based on the provided external ID. This endpoint allows for obtaining a list of sensors related to a specific user, enabling a more personalized sensor data retrieval.
    - **Example**

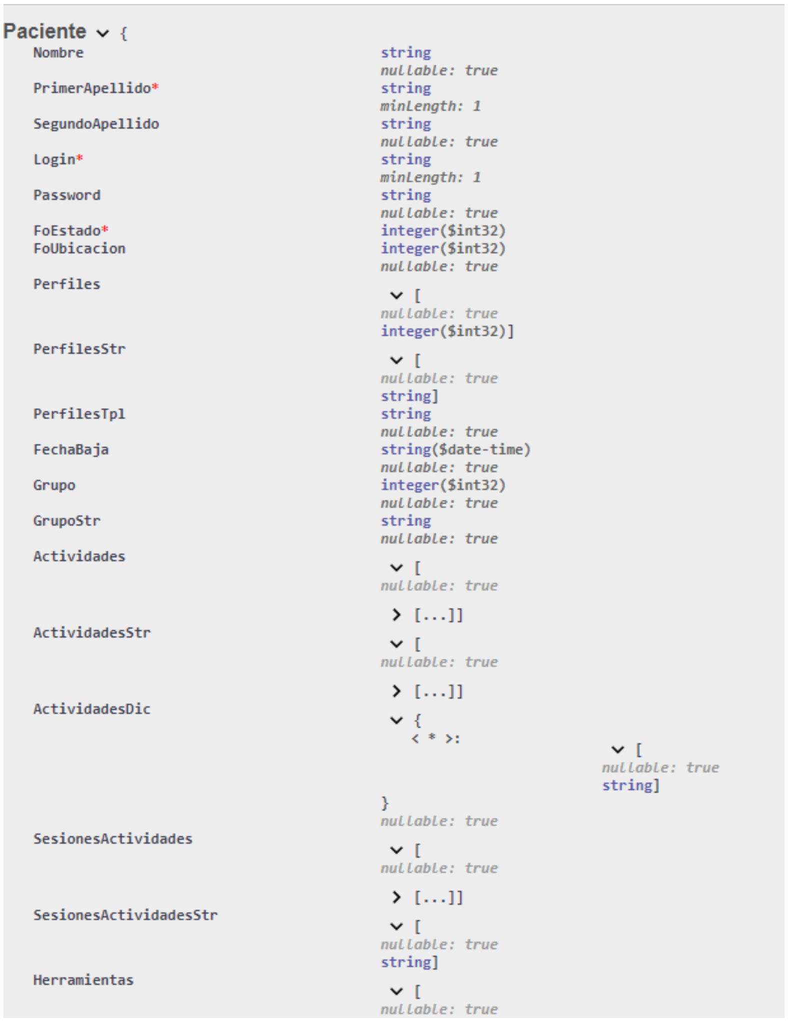

Example Value | Schema

```
[
  {
    "SensorId": 0,
    "Estado": "string",
    "Descripcion": "string"
  }
]
```

3.2.2 Schemas

A schema defines how data should be organized and what type of data can be included. It provides a formal description of the data structure so that applications consuming the API can understand and process information consistently.

In this case, we differentiated two important actors in this part of the system, which are the patients and the sensors associated with them, that collect information and will be associated with each patient.

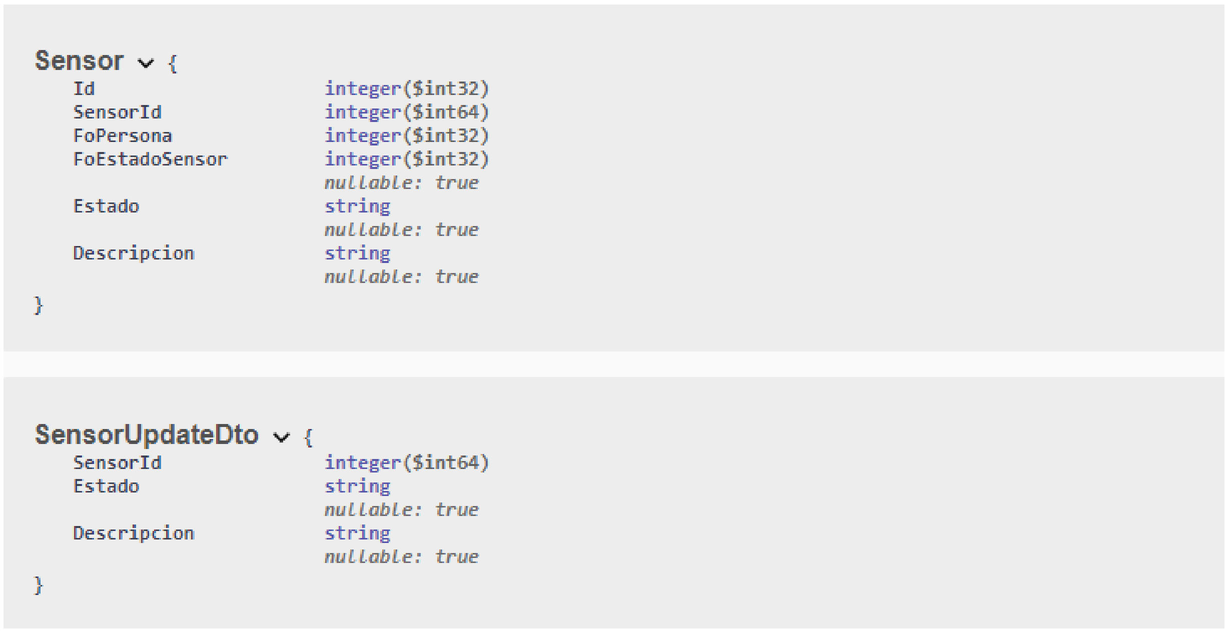


```
Paciente ∨ {
   Nombre                      string
                               nullable: true
   PrimerApellido*             string
                               minLength: 1
   SegundoApellido             string
                               nullable: true
   Login*                      string
                               minLength: 1
   Password                    string
                               nullable: true
   FoEstado*                   integer($int32)
   FoUbicacion                 integer($int32)
                               nullable: true
   Perfiles
                                ∨ [
                               nullable: true
                               integer($int32)]
   PerfilesStr
                                ∨ [
                               nullable: true
                               string]
   PerfilesTpl                 string
                               nullable: true
   FechaBaja                   string($date-time)
                               nullable: true
   Grupo                       integer($int32)
                               nullable: true
   GrupoStr                    string
                               nullable: true
   Actividades
                                ∨ [
                               nullable: true

                                > [...]]
   ActividadesStr
                                ∨ [
                               nullable: true

                                > [...]]
   ActividadesDic
                                ∨ {
                                  < * >:
                                                          ∨ [
                                                         nullable: true
                                                         string]
                               }
                               nullable: true
   SesionesActividades
                                ∨ [
                               nullable: true

                                > [...]]
   SesionesActividadesStr
                                ∨ [
                               nullable: true
                               string]
   Herramientas
                                ∨ [
                               nullable: true
```

Figure 24: Schema for Patient.

Here we can see all the patient related fields, where the necessary ones have been programmed to correctly identify all users of the pilot.

Here it is also seen that each patient has its associated sensors, and the information is robust and compact and with all the associations correctly performed.



```
Sensor ⌄ {
    Id                    integer($int32)
    SensorId              integer($int64)
    FoPersona             integer($int32)
    FoEstadoSensor        integer($int32)
                          nullable: true
    Estado                string
                          nullable: true
    Descripcion           string
                          nullable: true
}
```

```
SensorUpdateDto ⌄ {
    SensorId              integer($int64)
    Estado                string
                          nullable: true
    Descripcion           string
                          nullable: true
}
```

Figure 25: Schema for sensors.

And in the same way as with patients, sensors are also discharged, where they are seen to be associated with each patient so that the recorded information is saved automatically.

3.2.3 Complementary OpenAPIs

Complementary APIs that do not have a swagger but have been used for the communication of certain parts of the pilot are described below.

Content manager (APE)

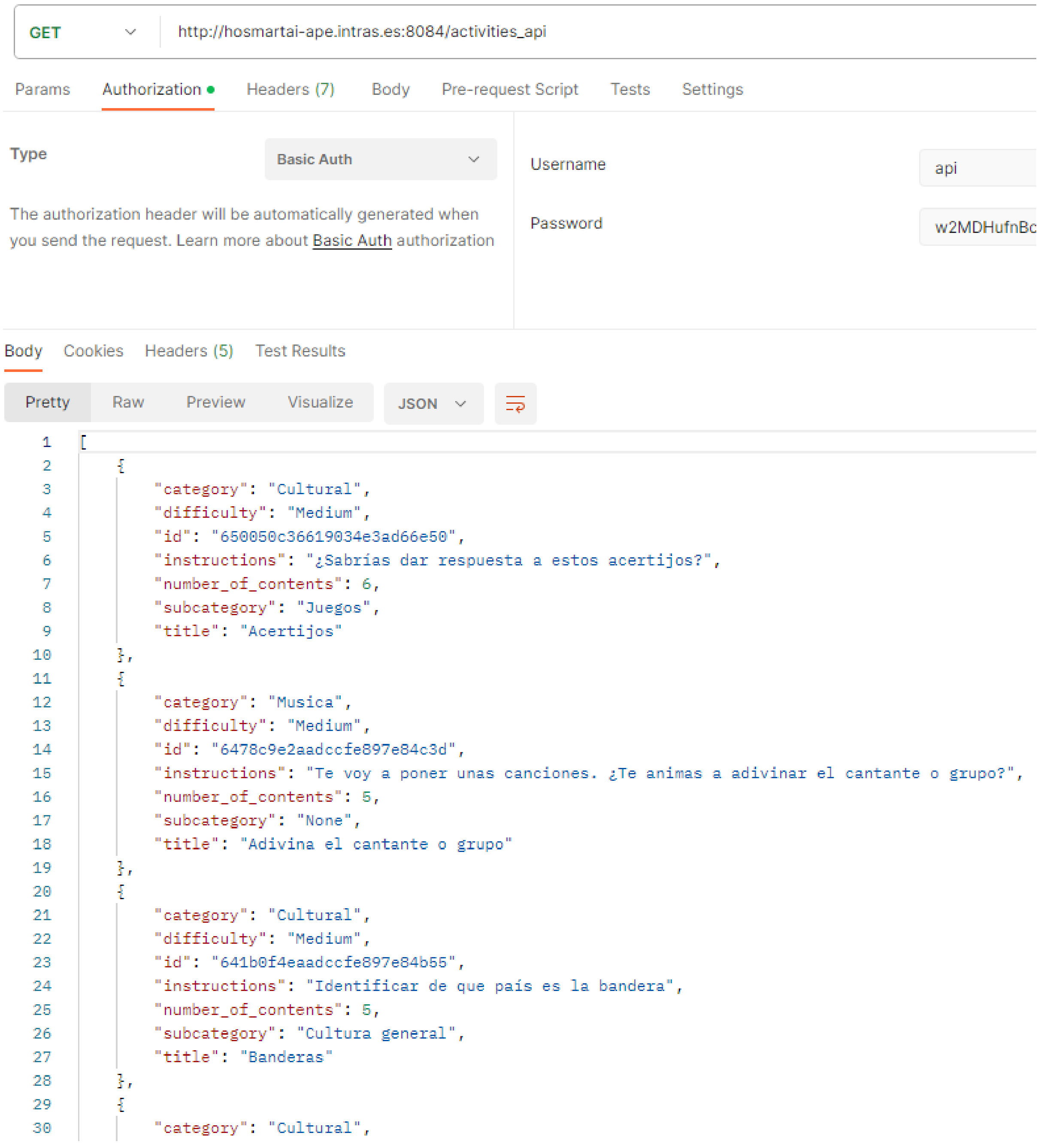
This API is mainly used to feed the application with the contents that the patients will see, as well as the workshops that the robot performs, the movements it must make and the activities available within the pilot's environment.

For this purpose, three endpoints have been developed that serve the necessary content to be displayed in the applications. Each of them displays different information, arranged in such a way that it is easy to handle in the code layer and can therefore be presented in the interface in a way that is easy for the user to handle.

The following are these 3 types of endpoints.

3.2.3.1 Activities

This endpoint has not been generated using swagger, due to its programming language and the needs of the project, but it is included in this section because of its importance for the final application of the pilot that has been developed.



http://hosmartai-ape.intras.es:8084/activities_api

Type: Basic Auth

The authorization header will be automatically generated when you send the request. Learn more about Basic Auth authorization

Username: api

Password: w2MDHufnBc

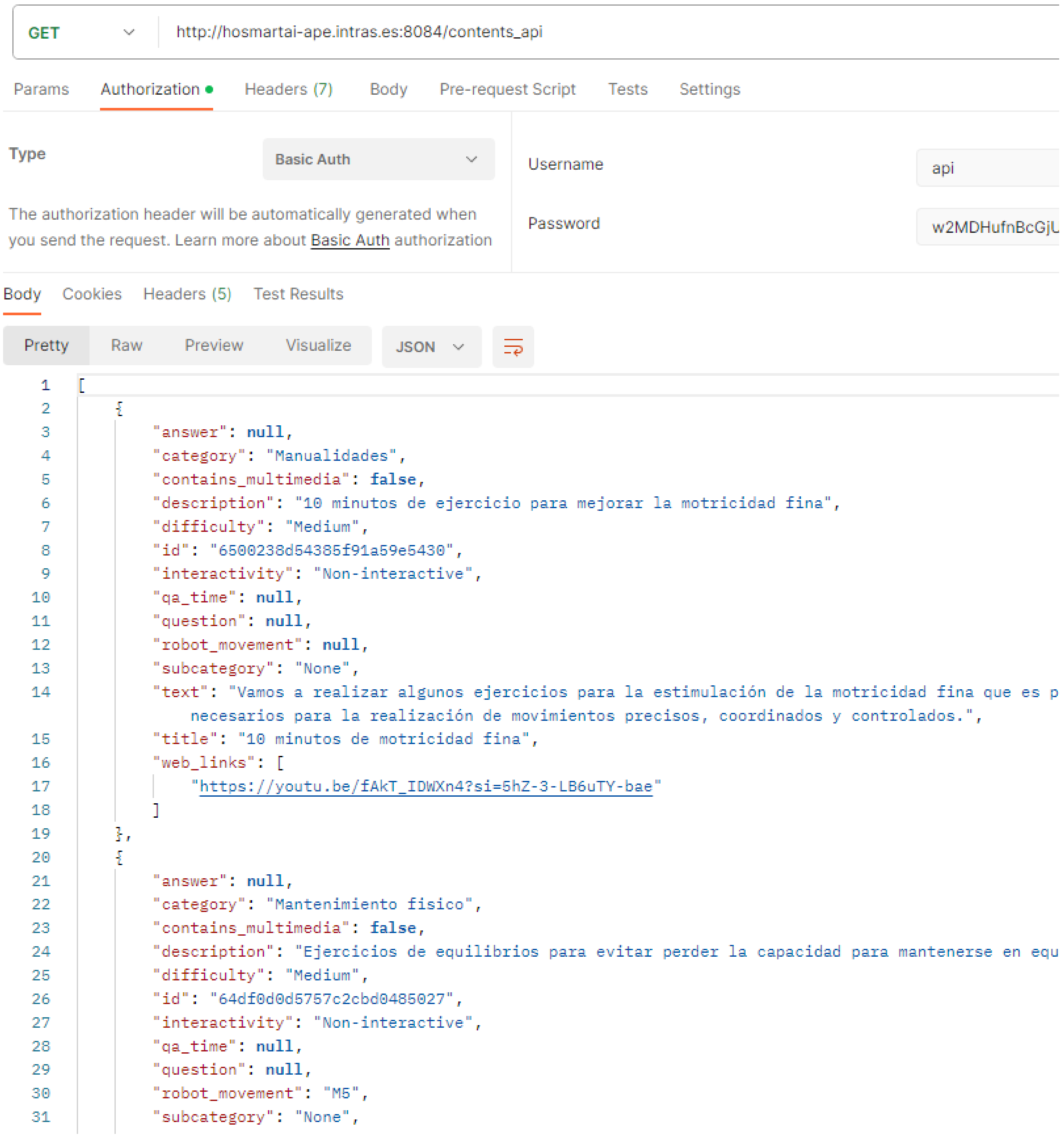
```
[
    {
        "category": "Cultural",
        "difficulty": "Medium",
        "id": "650050c36619034e3ad66e50",
        "instructions": "¿Sabrías dar respuesta a estos acertijos?",
        "number_of_contents": 6,
        "subcategory": "Juegos",
        "title": "Acertijos"
    },
    {
        "category": "Musica",
        "difficulty": "Medium",
        "id": "6478c9e2aadccfe897e84c3d",
        "instructions": "Te voy a poner unas canciones. ¿Te animas a adivinar el cantante o grupo?",
        "number_of_contents": 5,
        "subcategory": "None",
        "title": "Adivina el cantante o grupo"
    },
    {
        "category": "Cultural",
        "difficulty": "Medium",
        "id": "641b0f4eaadccfe897e84b55",
        "instructions": "Identificar de que país es la bandera",
        "number_of_contents": 5,
        "subcategory": "Cultura general",
        "title": "Banderas"
    },
    {
        "category": "Cultural",
```

Figure 26: Postman of activities.

As can be seen from the figure, this endpoint serves the information related to the activities. In this way, the category to which each activity belongs, difficulty, identifier, and title, among other properties, are indicated. In this way, the information is presented correctly in the applications that require it.

3.2.3.2 Contents

As with the previous endpoint, this one has not been generated using swagger either, due to its programming language and the needs of the project, but it is included in this section due to its importance for the final application of the pilot that has been developed.


GET http://hosmartai-ape.intras.es:8084/contents_api

Type: Basic Auth

The authorization header will be automatically generated when you send the request. Learn more about Basic Auth authorization

Username: api

Password: w2MDHufnBcGjL

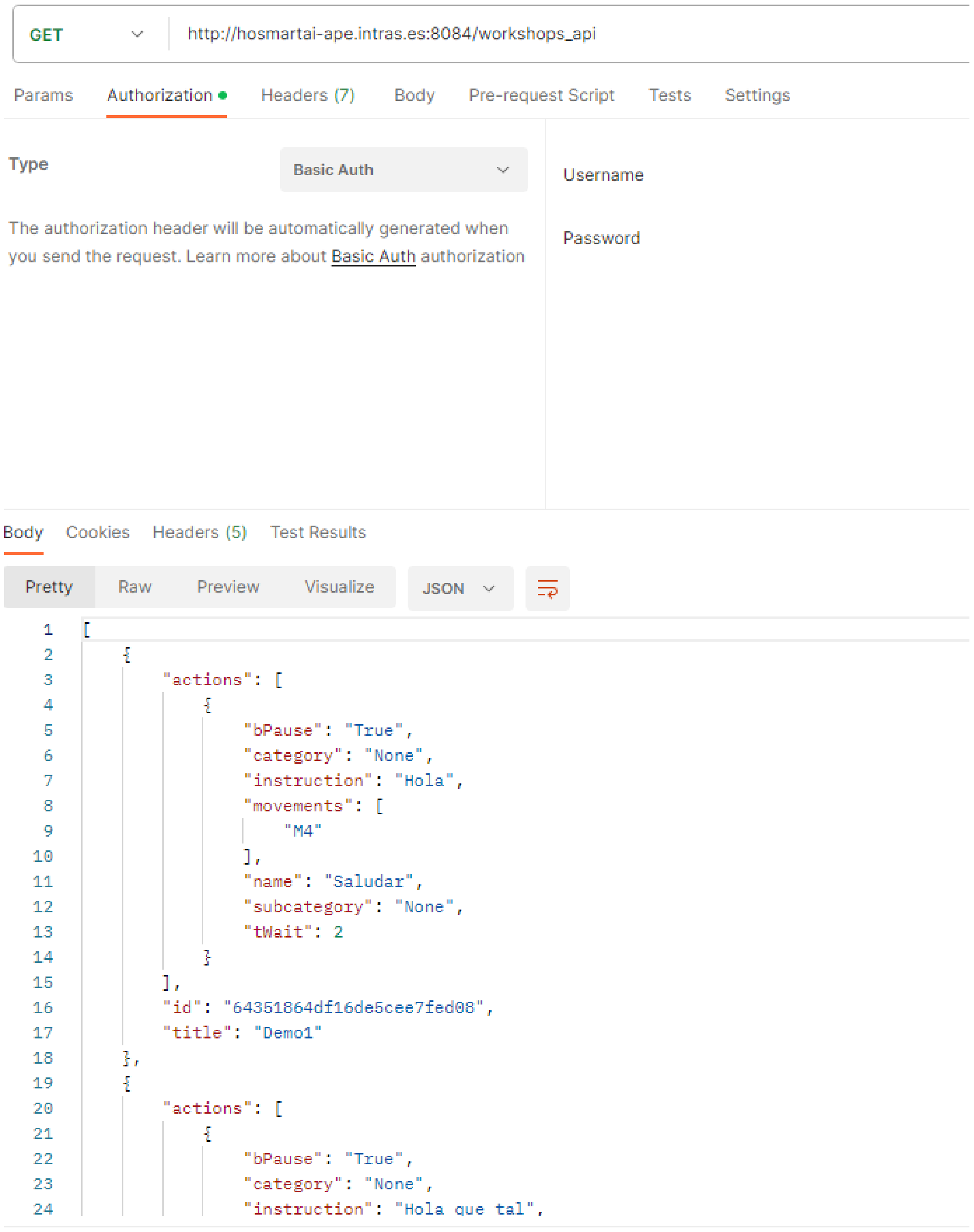
```
[
    {
        "answer": null,
        "category": "Manualidades",
        "contains_multimedia": false,
        "description": "10 minutos de ejercicio para mejorar la motricidad fina",
        "difficulty": "Medium",
        "id": "6500238d54385f91a59e5430",
        "interactivity": "Non-interactive",
        "qa_time": null,
        "question": null,
        "robot_movement": null,
        "subcategory": "None",
        "text": "Vamos a realizar algunos ejercicios para la estimulación de la motricidad fina que es p
            necesarios para la realización de movimientos precisos, coordinados y controlados.",
        "title": "10 minutos de motricidad fina",
        "web_links": [
            "https://youtu.be/fAkT_IDWXn4?si=5hZ-3-LB6uTY-bae"
        ]
    },
    {
        "answer": null,
        "category": "Mantenimiento fisico",
        "contains_multimedia": false,
        "description": "Ejercicios de equilibrios para evitar perder la capacidad para mantenerse en equ
        "difficulty": "Medium",
        "id": "64df0d0d5757c2cbd0485027",
        "interactivity": "Non-interactive",
        "qa_time": null,
        "question": null,
        "robot_movement": "M5",
        "subcategory": "None",
```

Figure 27: Postman of contents.

This endpoint focuses its functionality on serving activity-specific multimedia content to be shown to users. These are links to videos, images, as well as general information, news, and other content of interest to perform the exercises proposed for the user in this pilot.

3.2.3.3 Workshops

As with the previous endpoint, this one has not been generated using swagger either, due to its programming language and the needs of the project, but it is included in this section due to its importance for the final application of the pilot that has been developed.



GET http://hosmartai-ape.intras.es:8084/workshops_api

Type: Basic Auth

The authorization header will be automatically generated when you send the request. Learn more about Basic Auth authorization

Username

Password

```
[
    {
        "actions": [
            {
                "bPause": "True",
                "category": "None",
                "instruction": "Hola",
                "movements": [
                    "M4"
                ],
                "name": "Saludar",
                "subcategory": "None",
                "tWait": 2
            }
        ],
        "id": "64351864df16de5cee7fed08",
        "title": "Demo1"
    },
    {
        "actions": [
            {
                "bPause": "True",
                "category": "None",
                "instruction": "Hola que tal",
```

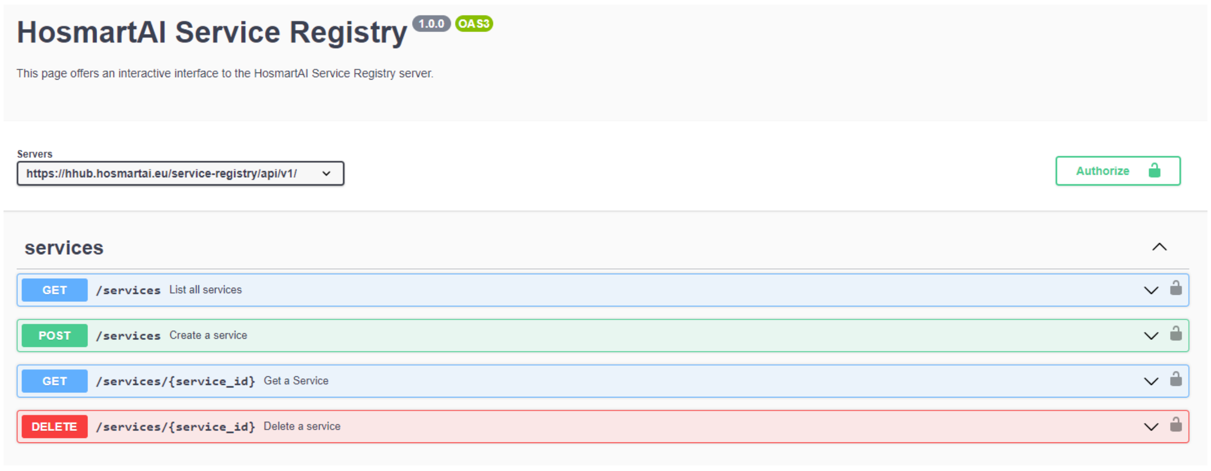
Figure 28: Postman of workshops.

Finally, this endpoint serves the workshops that will be executed by the Pepper robot through the application developed for this pilot. In this way, the sentences that the robot has to say are specified, as well as the movements that it has to perform at each moment.

3.3 HosmartAI Service Registry

Through this API, the necessary services are registered and included in the HosmartAI platform. Although it is a simple API with few methods, it fulfils its function and ensures that all services are registered correctly.

In the swagger shown below, the published methods are shown so that any other element of the architecture can make the corresponding calls.


HosmartAI Service Registry 1.0.0 OAS3

This page offers an interactive interface to the HosmartAI Service Registry server.

Servers
https://hhub.hosmartai.eu/service-registry/api/v1/

Authorize

services

GET /services List all services

POST /services Create a service

GET /services/{service_id} Get a Service

DELETE /services/{service_id} Delete a service

Figure 29: Swagger for service registry.

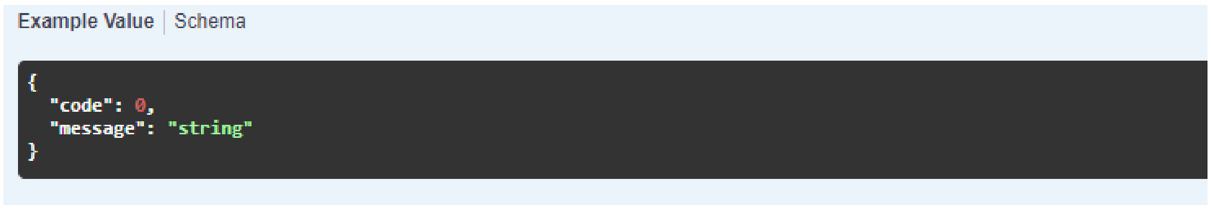
This swagger can be accessed via the following link: https://hhub.hosmartai.eu/service-registry/docs/

3.3.1 Method Explanation

It's time to break down the full functionality of this pilot. All functionality is achieved through the methods specified in the OpenAPI. They are detailed below.

1. **GET /services**

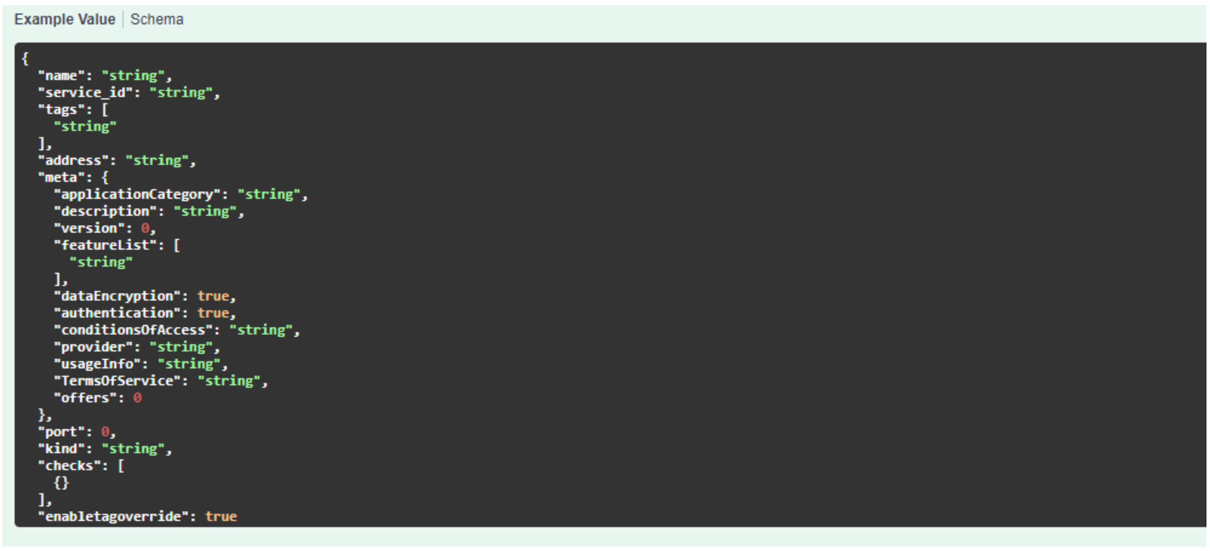
   - **Detailed Description:** This endpoint is used to retrieve the complete list of all registered services in the system. The expected response will be a collection of resources representing each service, with details such as name, description, and any other relevant information. This method is useful for displaying a summary of all available services.
   - **Example**


Example Value | Schema

```
{
  "code": 0,
  "message": "string"
}
```

2. **POST /services**

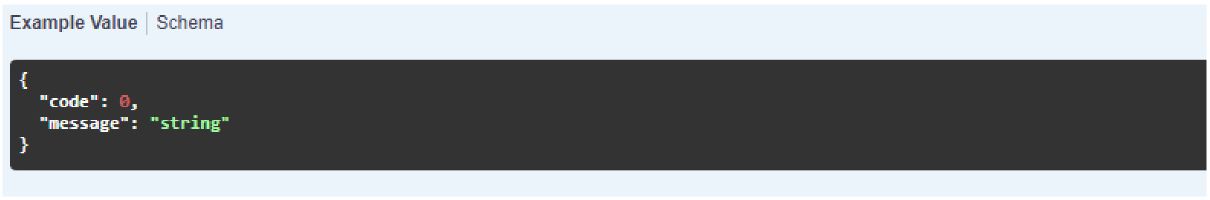
   - **Detailed Description:** By using this endpoint, you can create a new service in the system. The necessary data for creating the service is typically sent in the request body (payload). The response will provide information about the newly created service, such as its unique ID and any other relevant data.
   - **Example**


Example Value | Schema

```
{
  "name": "string",
  "service_id": "string",
  "tags": [
    "string"
  ],
  "address": "string",
  "meta": {
    "applicationCategory": "string",
    "description": "string",
    "version": 0,
    "featureList": [
      "string"
    ],
    "dataEncryption": true,
    "authentication": true,
    "conditionsOfAccess": "string",
    "provider": "string",
    "usageInfo": "string",
    "TermsOfService": "string",
    "offers": 0
  },
  "port": 0,
  "kind": "string",
  "checks": [
    {}
  ],
  "enabletagoverride": true
```

3. **GET /services/{service_id}**

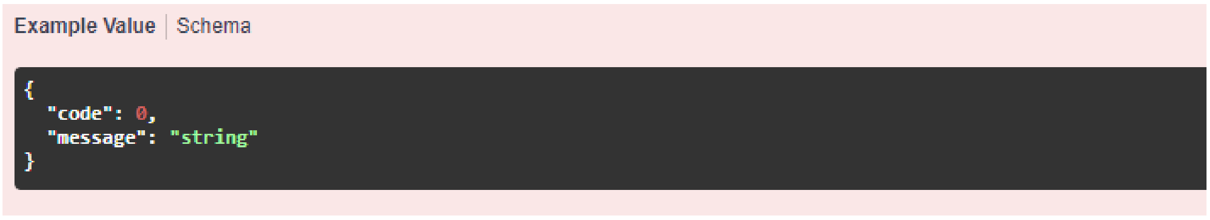
   - **Detailed Description:** This endpoint allows you to retrieve detailed information about a specific service identified by its unique ID (service_id). The response will contain all available details about that service, which may include its name, description, status, etc. This method is useful for retrieving specific information about a particular service.
   - **Example**


Example Value | Schema

```
{
  "code": 0,
  "message": "string"
}
```

4. **DELETE /services/{service_id}**

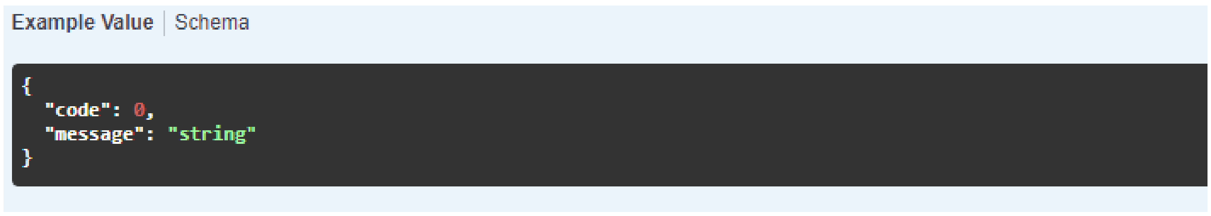
   - **Detailed Description:** By using this endpoint, you can delete a specific service from the system based on its unique ID (service_id). After performing this operation, the service will no longer be available in the system, and the response may include information about the success of the operation or relevant messages. This method is useful for removing services that are no longer needed or have been discontinued.
   - **Example**


Example Value | Schema

```
{
  "code": 0,
  "message": "string"
}
```

5. **GET /services/health/{service_id}**

- **Detailed Description**: This endpoint is designed to retrieve the health status of a specific service identified by its unique ID (service_id). The purpose is to monitor the operational health and status of a particular service within the system. The response typically includes information about the service's availability, performance, or any relevant metrics indicating its current health. This endpoint is crucial for systems to assess and ensure the well-being of individual services in real-time.
- **Example**

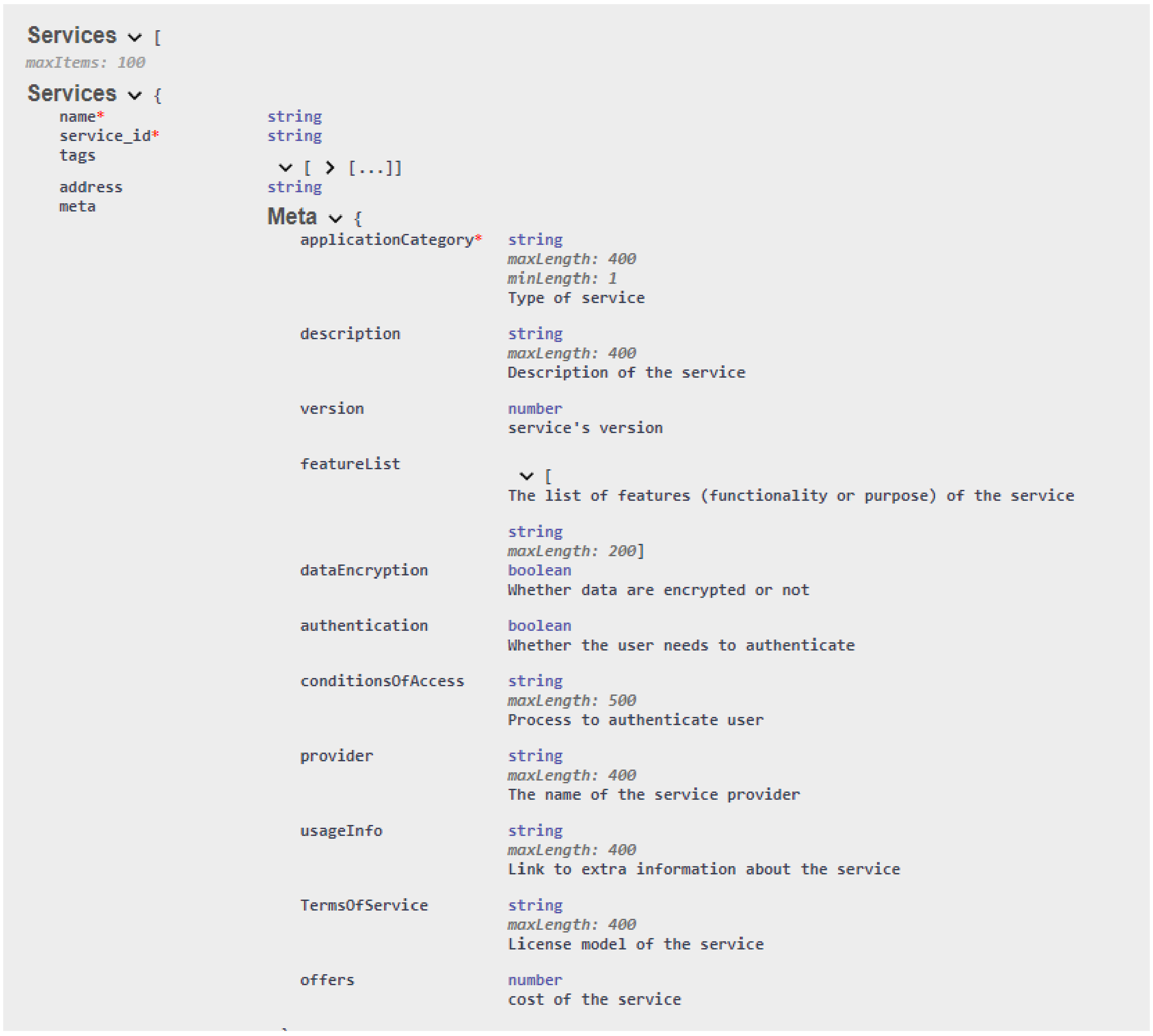


```
{
  "code": 0,
  "message": "string"
}
```

3.3.2 Schemas

A schema defines how data should be organized and what type of data can be included. It provides a formal description of the data structure so that applications consuming the API can understand and process information consistently.

The main object in this part of the API is the service being registered. This service must associate certain fields that have to be able to identify who registers it, for what it serves and the terms of use among other things.



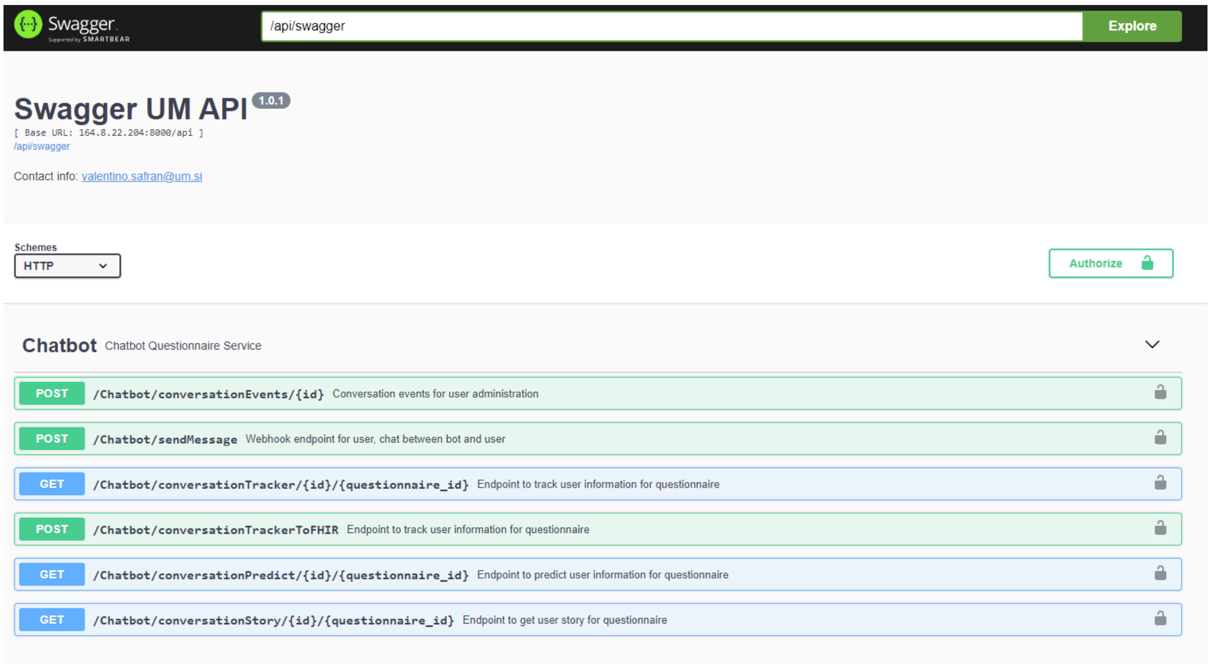
```
Services ⌄ [
maxItems: 100
Services ⌄ {
    name*               string
    service_id*         string
    tags
                          ⌄ [ › [...]]
    address             string
    meta
                        Meta ⌄ {
                            applicationCategory*  string
                                                  maxLength: 400
                                                  minLength: 1
                                                  Type of service

                            description           string
                                                  maxLength: 400
                                                  Description of the service

                            version               number
                                                  service's version

                            featureList
                                                    ⌄ [
                                                  The list of features (functionality or purpose) of the service

                                                  string
                                                  maxLength: 200]
                            dataEncryption        boolean
                                                  Whether data are encrypted or not

                            authentication        boolean
                                                  Whether the user needs to authenticate

                            conditionsOfAccess    string
                                                  maxLength: 500
                                                  Process to authenticate user

                            provider              string
                                                  maxLength: 400
                                                  The name of the service provider

                            usageInfo             string
                                                  maxLength: 400
                                                  Link to extra information about the service

                            TermsOfService        string
                                                  maxLength: 400
                                                  License model of the service

                            offers                number
                                                  cost of the service
```

Figure 30: Schema for services.

3.4 HosmartAI Chatbot

In this section, the operation of the open API that manages all communication with the Chatbot, developed by the University of Maribor, is detailed. It is mainly used in Pilot 2 and communicates with the Android application that schedules appointments for users. This Chatbot provides a more human-like communication interface to re-arrange a change of appointment if the patient is unable to attend.

To do so, the application asks through this OpenAPI for the phrases to be displayed on the interface (http://164.8.22.204:8000/webjars/swagger-ui/index.html?url=/api/swagger&validatorUrl=).



/api/swagger

Swagger UM API 1.0.1

[Base URL: 164.8.22.204:8000/api]
/api/swagger

Contact info: valentino.safran@um.si

Schemes
HTTP

Chatbot Chatbot Questionnaire Service

POST /Chatbot/conversationEvents/{id} Conversation events for user administration

POST /Chatbot/sendMessage Webhook endpoint for user, chat between bot and user

GET /Chatbot/conversationTracker/{id}/{questionnaire_id} Endpoint to track user information for questionnaire

POST /Chatbot/conversationTrackerToFHIR Endpoint to track user information for questionnaire

GET /Chatbot/conversationPredict/{id}/{questionnaire_id} Endpoint to predict user information for questionnaire

GET /Chatbot/conversationStory/{id}/{questionnaire_id} Endpoint to get user story for questionnaire

Figure 31: Swagger for chatbot.

3.4.1 Method Explanation

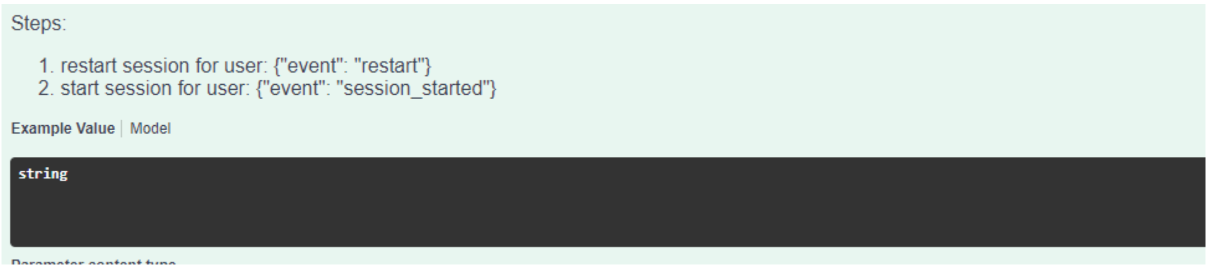
It's time to break down the full functionality of this pilot. All functionality is achieved through the methods specified in the OpenAPI. They are detailed below.

Chatbot Questionnaire Service

1. **POST /Chatbot/conversationEvents/{id}**

   - **Detailed Description:** Records conversation events for user administration. This endpoint is utilized to capture and manage events that occur during a conversation, facilitating user administration and allowing for the tracking of specific interactions.

   - **Example**



Steps:

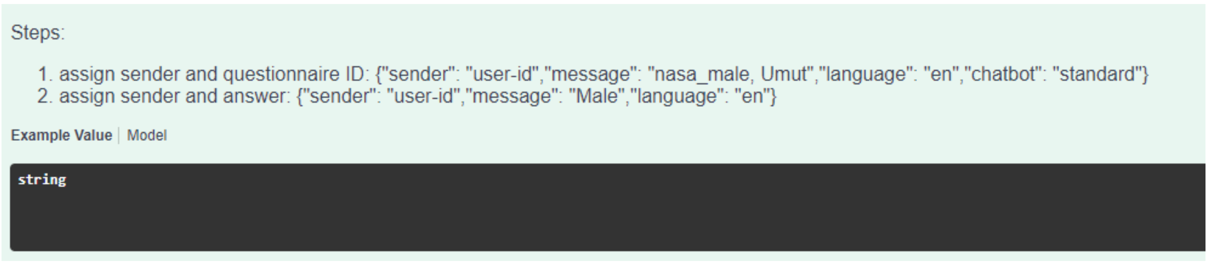
1. restart session for user: {"event": "restart"}
2. start session for user: {"event": "session_started"}

Example Value | Model

```
string
```

2. **POST /Chatbot/sendMessage**

    - **Detailed Description:** Webhook endpoint for user interactions, enabling communication between the bot and the user. This endpoint handles messages sent by users and triggers the appropriate actions or responses from the chatbot.
    - **Example**



Steps:

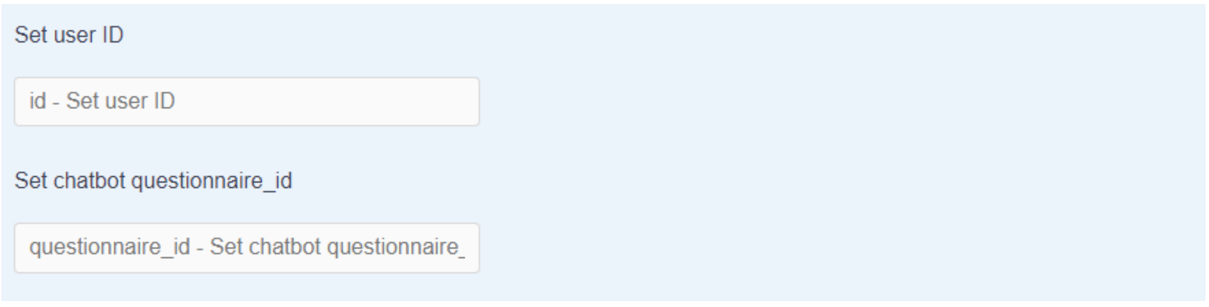
1. assign sender and questionnaire ID: {"sender": "user-id","message": "nasa_male, Umut","language": "en","chatbot": "standard"}
2. assign sender and answer: {"sender": "user-id","message": "Male","language": "en"}

Example Value | Model

```
string
```

3. **GET /Chatbot/conversationTracker/{id}/{questionnaire_id}**

    - **Detailed Description:** Endpoint designed to track user information specific to a questionnaire. By providing the user ID and questionnaire ID, this endpoint retrieves information related to the user's progress or responses within the specified questionnaire.
    - **Example**



Set user ID

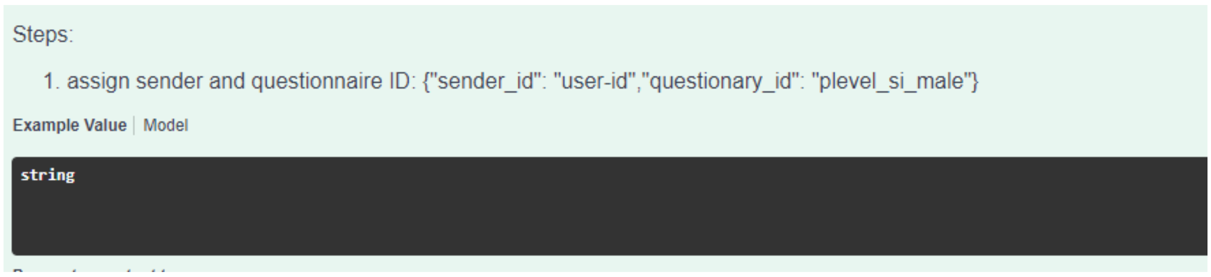
id - Set user ID

Set chatbot questionnaire_id

questionnaire_id - Set chatbot questionnaire_

4. **POST /Chatbot/conversationTrackerToFHIR**

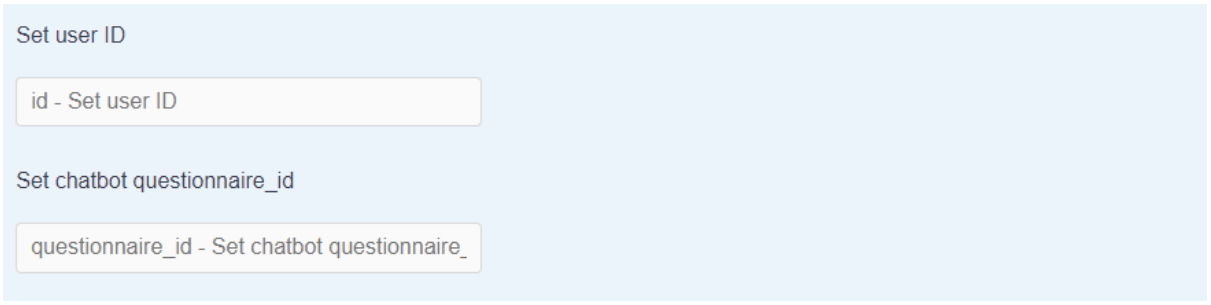
    - **Detailed Description:** Endpoint for tracking user information and converting it to FHIR (Fast Healthcare Interoperability Resources) format. This allows for standardized representation and exchange of healthcare-related information, enhancing interoperability.
    - **Example**


Steps:

1. assign sender and questionnaire ID: {"sender_id": "user-id","questionary_id": "plevel_si_male"}

```
string
```

5. **GET /Chatbot/conversationPredict/{id}/{questionnaire_id}**
   - **Detailed Description:** Endpoint for predicting user information based on the user's ID and questionnaire ID. This may involve leveraging machine learning or predictive algorithms to anticipate user responses or behaviour within the context of a specific questionnaire.
   - **Example**

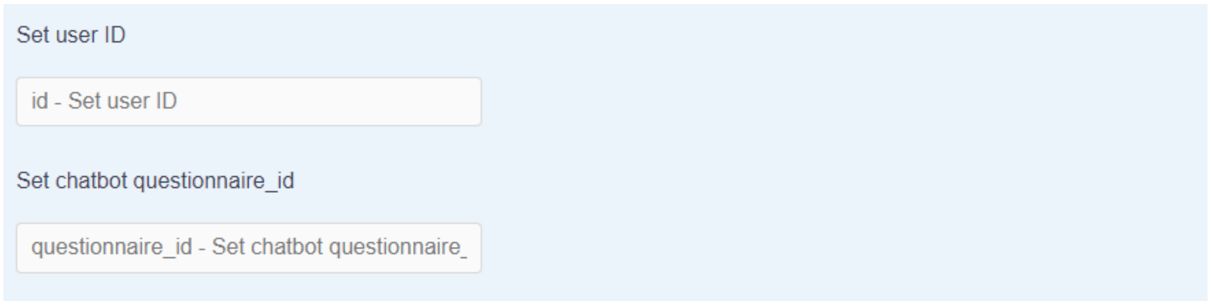

Set user ID

id - Set user ID

Set chatbot questionnaire_id

questionnaire_id - Set chatbot questionnaire_

6. **GET /Chatbot/conversationStory/{id}/{questionnaire_id}**
   - **Detailed Description:** Endpoint to retrieve the user story for a given user and questionnaire. This can provide a narrative or chronological account of the user's interactions and responses, offering insights into their journey through the questionnaire.
   - **Example**


Set user ID

id - Set user ID

Set chatbot questionnaire_id

questionnaire_id - Set chatbot questionnaire_

3.4.2 Schemas

A schema defines how data should be organized and what type of data can be included. It provides a formal description of the data structure so that applications consuming the API can understand and process information consistently.

In this OpenAPI no schemas are available as all information is sent via string type without any concrete format.

4 Updated HosmartAI Architecture Design

To understand the new version of the architecture, we present below the resulting version of D4.2 which is the previous version.

4.1 Previous version

We present the old version of the diagram, to give visibility to the changes made during this period.

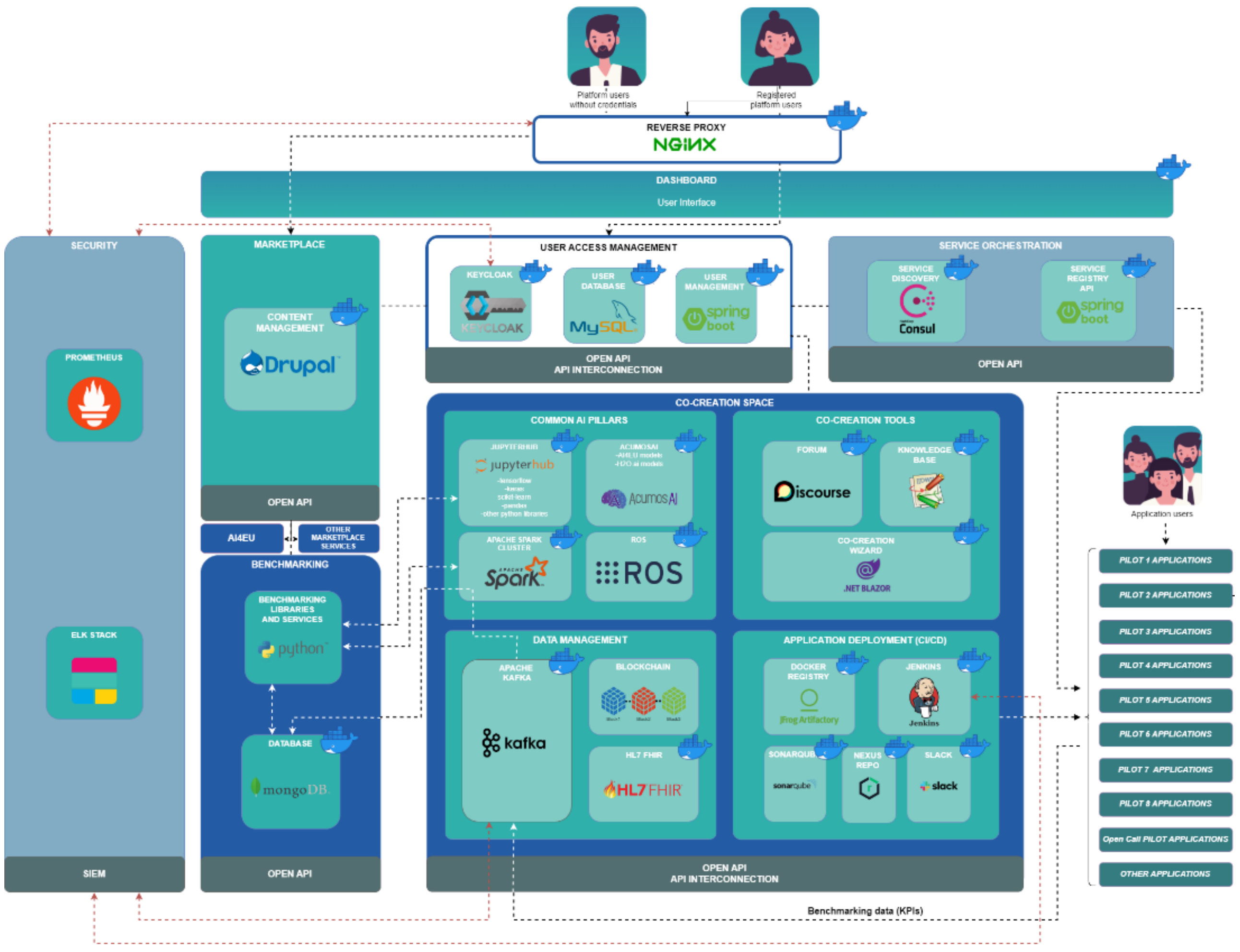


Figure 32: Old architecture diagram.

4.2 New elements

Below are the elements added to the platform, the result of the use and the needs detected throughout the development.

4.2.1 Graphene

The project, known as Eclipse Graphene™, emerges because of the European AI4EU initiative, funded by the European Commission between 2019 and 2021. Initially, Acumos AI was used for artificial intelligence (AI) experiments, but over time, it forked into a new development called AI4EU Experiments. This fork exhibits significant differences from Acumos, especially in the management of Docker images and gRPC communication, aiming for greater flexibility and interoperability for model providers. To enhance security and scalability, unfinished features of Acumos were deliberately removed. As the AI4EU project concludes, there is a plan to transform it into an open-source project with well-defined governance.

Eclipse Graphene covers an extensive market by offering reusable solutions for AI and machine learning. Its approach is not limited to machine learning experts but is also designed for ordinary developers to create applications with ease.

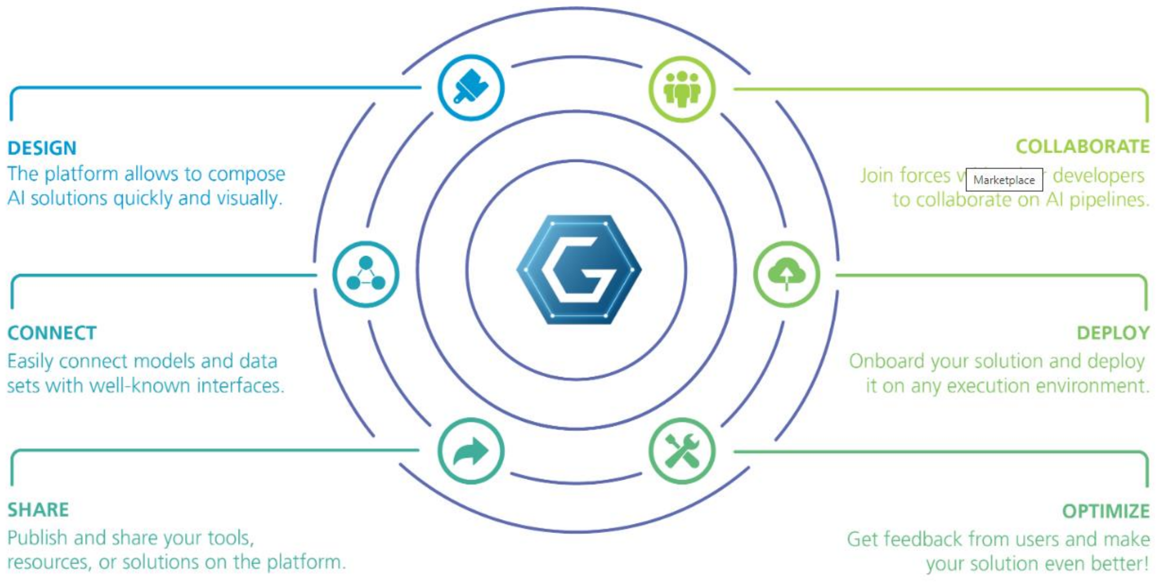

DESIGN
The platform allows to compose AI solutions quickly and visually.
CONNECT
Easily connect models and data sets with well-known interfaces.
SHARE
Publish and share your tools, resources, or solutions on the platform.
COLLABORATE
Join forces w Marketplace developers to collaborate on AI pipelines.
DEPLOY
Onboard your solution and deploy it on any execution environment.
OPTIMIZE
Get feedback from users and make your solution even better!

Figure 33: Graphene functionality.

In terms of its structure, Eclipse Graphene doesn't function as a centralized execution environment but rather as a comprehensive design and distribution framework. It acts as a launchpad for training and validating individual components and integrated solutions. Secure distribution of results takes place through an electronic catalogue and is compatible with various runtime environments, including those based on Kubernetes. The inclusion of AcuCompose, a graphical tool for linking models, data translation tools, filters, and adapters, enhances its utility. It also encourages collaboration in closed groups for specific projects and ensures solution portability by being compatible with various hardware infrastructures. Mechanisms for packaging, sharing, licensing, and deploying AI models as portable, containerized microservices further solidify its functionality.

The choice of Eclipse Graphene as an Eclipse project is based on the belief in its ability to support the adoption, innovation, and evolution of AI applications. Seeking a broader audience and aiming to attract more developers, the project anticipates active participation from individuals and teams involved in EU-funded projects.

In legal terms, it acknowledges that Acumos is a Linux Foundation project. Ongoing discussions with LF underscore the transition from the fork to Eclipse, emphasizing compliance with legal rules related to trademarks. This implies the removal of all instances of the Acumos trademark. The project prioritizes legal compliance in its evolution.

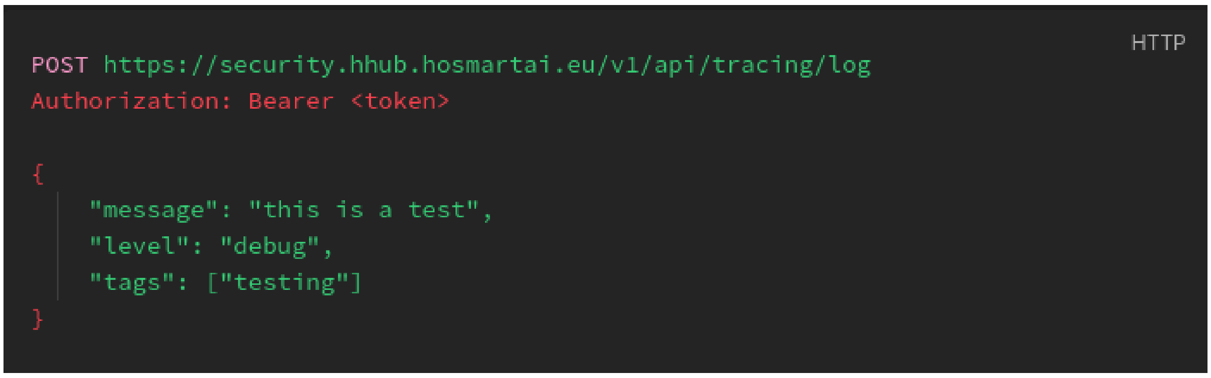
4.2.2 Security Events Logging API

For the purposes of monitoring and traceability inside each HosmartAI pilot's code base, a custom event logging API has been built and is currently deployed in the HHUB platform.
This makes it possible to trace important events on an external system, so that in case of application crashes or other critical situations, the history of events leading to such situation can still be analysed.

Custom alerts based on these event logs can also be written, as to be notified as soon as a specific combination of events happen.

The API requires the use of an API key that is provided to interested partners. The key is tied to an "application" the partner is building, so each partner can request multiple keys if needed (one for each application they are developing).

The following figure illustrates a sample request made to log some event:



```
POST https://security.hhub.hosmartai.eu/v1/api/tracing/log
Authorization: Bearer <token>

{
    "message": "this is a test",
    "level": "debug",
    "tags": ["testing"]
}
```

Figure 34: Sample Request.

4.3 Updated elements

In this section we will present the elements that are maintained from one iteration to the next, and have been updated, either in operation or implementation.

4.3.1 Service Registry API

In the dynamic and ever-evolving landscape of software development, choosing the right backend technology has been crucial for the success of a project. In some cases, the need to switch from an established framework like Spring Boot to Node.js has arisen. Below are explored some reasons that motivated this transition.

1. Performance and Scalability: Node.js has gained popularity for its ability to efficiently handle non-blocking input/output (I/O) operations. Its event-driven architecture and capacity to manage multiple simultaneous connections proved advantageous in environments where scalability and performance were critical.

2. Rapid Development: The use of JavaScript on both the client and server sides allowed developers to work more cohesively across the entire application stack. This language uniformity accelerated development and facilitated the transition between frontend and backend.

3. Community and Ecosystem: Node.js boasted an active community and an extensive package repository through npm. This facilitated collaboration, code reuse, and the integration of third-party libraries, providing developers access to a wide range of tools.

4. Microservices and Event-Driven Architecture: Node.js proved compatible with microservices and event-driven architectures. Its asynchronous nature adapted well to building distributed systems and efficiently implementing communication between services.

5. JavaScript on Both Sides: The ability to use the same language on the frontend and backend simplified development and reduced code complexity. This was particularly beneficial when employing popular frontend frameworks using JavaScript, such as React, Angular, or Vue.

6. Real-Time and Real-Time Applications: Node.js excelled in real-time applications, such as real-time chat, online games, and real-time collaboration. Its ability to handle simultaneous connections efficiently made it a natural choice for such projects.

7. Lower Overhead: Some developers argued that Node.js had less overhead compared to heavier frameworks like Spring Boot. This was beneficial for smaller or less complex projects, where agility and simplicity were valued.

8. Flexibility and Adaptability: Node.js was known for its flexibility. It allowed developers to choose and combine libraries based on the specific needs of the project, providing a more customized approach to software development.

9. Change in Project Requirements: As project requirements evolved, there arose the need to adopt a more event-driven or microservices-oriented approach. Node.js, with its ability to handle these architectures efficiently, became a more natural choice in such scenarios.

Ultimately, the choice between Spring Boot and Node.js was based on specific project factors, team skills, and long-term goals. Each technology had its strengths and weaknesses, and the decision was grounded in a careful evaluation of the project's requirements and objectives.

4.4 Deleted Elements

Removing an element from an architecture, whether it's in software, hardware design, or any complex system, may become necessary for various reasons. One primary reason is the evolving nature of project requirements. Over time, the needs of a project can change, rendering certain components obsolete or unnecessary.

Another key motivation for elimination is the pursuit of optimization and enhanced performance. Simplifying the architecture by eliminating redundancies or non-critical elements can significantly improve the overall efficiency of the system.

Adaptability and flexibility are critical aspects that may warrant the removal of certain elements. Creating an architecture that is more adaptable to future changes or improvements ensures that the system can efficiently incorporate new features or technologies.

Cost considerations also play a role in the decision to remove an element. Maintaining and updating certain components can be resource intensive. If a component no longer provides substantial value or its upkeep is economically prohibitive, removal becomes a viable option.

Design flaws or issues within a component can necessitate its removal to maintain the stability and reliability of the overall system. Technological advancements may also lead to the obsolescence of certain components, making it practical to remove them and adopt more modern technologies.

Scalability is another factor. Some architectures are designed to be scalable, and removing components may be necessary to facilitate the expansion or reduction of the system as needed.

Simplification is often a desired outcome. A simpler architecture is easier to understand, maintain, and debug. Removing unnecessary elements contributes to simplifying the architecture, making it more manageable.

In summary, the decision to eliminate an element from an architecture should be approached with caution, considering the potential cascading impacts on other parts of the system. Thorough analysis and a deep understanding of the system are crucial. Additionally, following development and design practices that facilitate adaptability and maintainability is essential as requirements and technologies evolve.

In this way all changes made to the architecture have been made by similar elements, which do not affect the functioning of the elements already included and are therefore transparent to the functioning of the rest of the platform.

4.4.1 Acumos

The decision to move from Acumos to Eclipse Graphene is significantly influenced by Graphene's distinct approach to handling AI models and its infrastructure. Unlike Acumos, Graphene does not require to build the docker images of models internally but instead stores the references to docker image URIs. By not storing docker images internally, Graphene minimizes the risk associated with data breaches or unauthorized access. The security responsibility is more distributed, with each data provider managing their own security. This methodology aligns with our goal of maintaining a streamlined, efficient infrastructure. Additionally, Graphene's requirement for models to support gRPC communication according to its container specification enhances interoperability between models, a capability that was limited under Acumos’s framework. This aspect of Graphene ensures a higher compatibility degree for model providers, facilitating smoother integration and communication between different components of HosmartAI.

Furthermore, Graphene's design studio supports a wider range of topologies and offers gRPC streaming capabilities, broadening the scope of our project's functionalities. By removing unfinished features from Acumos, like the Workbench and NIFI integration, Graphene presents a more streamlined and focused toolset. Notably, Graphene does not allow models to be executed internally for security and scalability reasons, a stark contrast to Acumos. This approach minimizes security risks and enhances our project's scalability by offloading execution responsibilities. These factors, combined, make Eclipse Graphene a more suitable and future-proof choice for our project, ensuring that we can effectively meet current needs while also being well-equipped to adapt to potential future developments and requirements.

4.5 New version

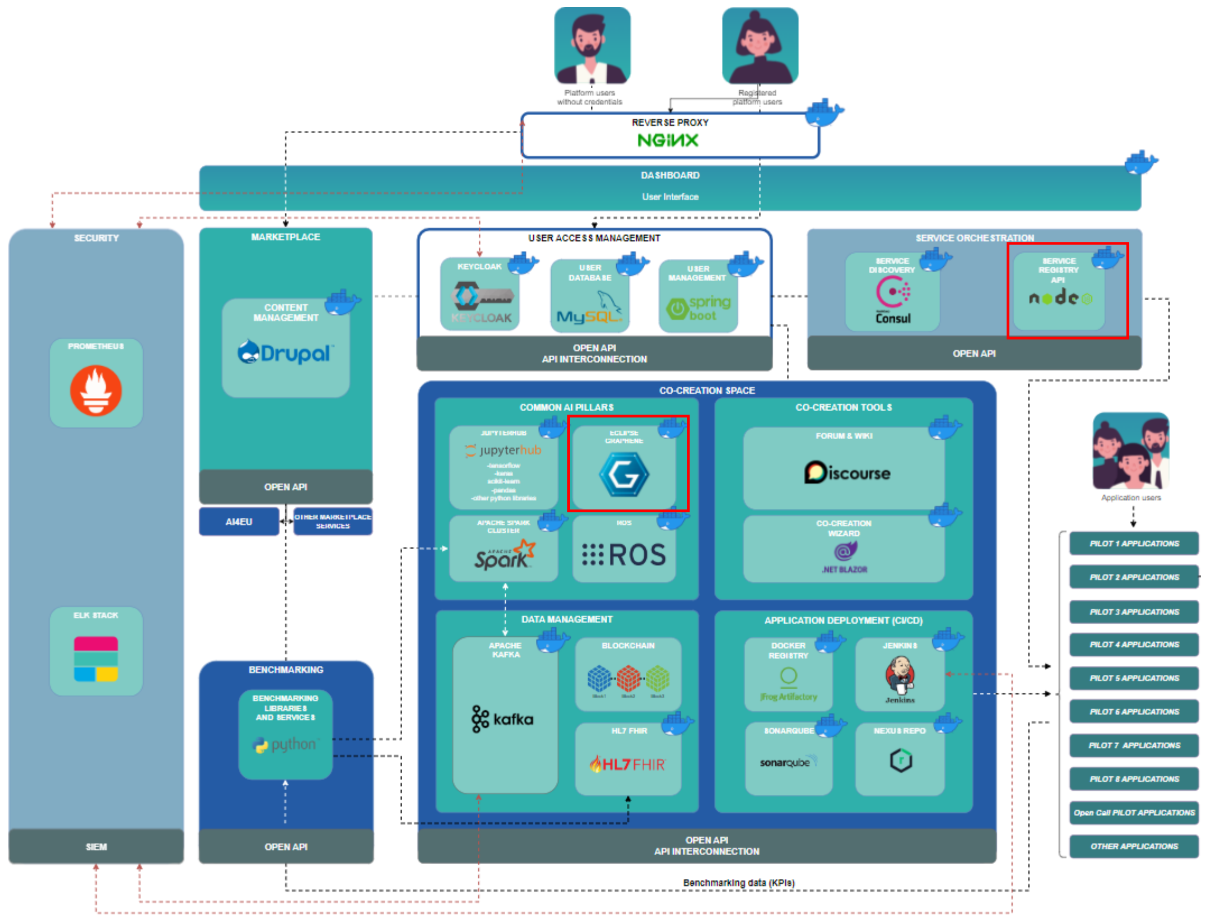


Figure 35: New architecture diagram.

The major changes of the architecture are highlighted through the red boxes.

The result of the great work that was carried out in an initial period of the project, identifying the needs of the projects, the partners and the functionality that was intended to be granted, the changes that the platform has undergone have been minor throughout the last two iterations. These changes, as mentioned above, are the result of the use and appearance of specific needs that have caused elements of the platform to change, be removed or appear new.

5 Conclusion

After the work done throughout Task 4.1 and through these three deliverables that compose it, a robust architecture has been developed that meets the established requirements of the project.

As already seen in the second installable, and repeated in the present, the changes that have been made have been few compared to the first version, where a significant effort was made to identify the needs and create a robust first version.

To this end, some elements that made no sense within the architecture have been eliminated. At the same time, others have been updated, changing the way in which they are integrated, or the version originally used. New elements have also been added, since it has been detected that they were missing to cover certain needs of the system.

Regarding OpenAPIs, it has had a later evolution than the platform since the development of each pilot or element has been done as the overall development time of the project progressed.

For this reason, the last deliverable details the way in which they have been generated as well as the final specification of the OpenAPIs used for the communication of elements within the project's own architecture.