Retrofitting Equipment for Efficient Use of Variable Feedstock in Metal Making Processes - REVaMP

H2020-NMBP-ST-IND-2018-2020 / H2020-NMBP-SPIRE-2019

Grant agreement no. 869882

Start Date:	July 1st, 2020
Duration:	48 months
Project Type:	Innovation Action

D 4.5. Software as a Service tool for Furnace Performance Management – Rev 1

Due Date:	June 30th, 2022
Submission Date:	February 24th, 2023 (1st revision)
Work Package:	**WP 4 – Process Monitoring, Control and Decision Support Systems**
Lead Beneficiary:	**AZTERLAN**

Authors:

Partner	Name
BFI	
NCBJ	
SYSKON	
POLON	
Fraunhofer ILT	
LSA	
AMB	
SIDENOR	
SIDENOR I+D	
EURECAT	
GRUPAL ART	
AZTERLAN	Javier Nieves, Ciro Sierra, Asier González
GHI HORNOS	
REFIAL	
INATEC	
RWTH AACHEN	
CARTIF	
EXIDE	

Dissemination level

PU	public	☒
CO	Confidential, only for members of the consortium (incl. the Commission Services)	☐

Table of contents

1. About REVaMP 3
2. Introduction and Summary 3
3. SaaS as Solution for REVaMP 4
3.1. What 4
3.2. How 5
3.2.1. .Net and Azure 5
3.2.2. .Net in REVaMP, .Net Core – Web API 6
3.2.3. Swagger as Documentation 7
3.3. Where 8
3.4. Why 10
3.5. Design 11
3.5.1. Technologies Description 11
3.5.2. Diagrams 24
3.5.2.1. Components Diagram 24
3.5.2.2. Package Diagrams 25
3.5.2.3. Class Diagram by Component 27
3.6. Storage 31
3.6.1. Data Base 31
3.6.2. File System 32
3.7. Server Entities Management 33
3.7.1. General 34
3.7.2. Third Party Integration 38
3.7.2.1. External DLLs 39
3.7.2.2. External Services in Python 40
3.7.2.3. Docker Integration 42
4. REVaMP DEMO 42
5. Conclusions 48

1. About REVaMP

The main objective of the project “Retrofitting Equipment for Efficient Use of Variable Feedstock in Metal Making Processes” (REVaMP) is to develop, adapt and apply novel retrofitting technologies to cope with the increasing variability and to ensure an efficient use of the feedstock in terms of materials and energy.

For this purpose, existing metal production plants shall be retrofitted with appropriate sensors for scrap analysis and furnace operation. Furthermore, the selection of the optimal feedstock in terms of material and energy efficiency shall be improved by application of appropriate process control and decision support tools. Also, a solid scrap preheating system operated with waste derived fuel shall increase the energy efficiency of the melting processes. To monitor and control the process behavior in an optimal way, model-based software tools will be developed and applied.

The retrofitting solutions will be exemplarily demonstrated within three different use cases from the metal making industry, namely electric and oxygen steelmaking, aluminium refining and lead recycling. The performance of the different technologies will be assessed, and the benefits will be evaluated in terms of economic and ecological effects, as well as cross-sectorial applicability in other process industries.

2. Introduction and Summary

This deliverable D4.5, “Software as a Service tool for Furnace Performance Management”, is included in the work package WP 4 “Process Monitoring, Control and Decision Support Systems” of the project. Specifically, this deliverable groups all work developed during WP4 and provides an access method that facilitates the use of the developed models and algorithms.

Deliverable D4.5. is of the type OTHER, consisting of a software solution to be deployed and adjusted during WP6, which is focused on making a demonstration on retrofitting solutions at aluminium use cases. Moreover, as these solutions will be deployed and tested during this WP, several adjustments could be carried out with the final aim to provide a better solution. Nevertheless, this deliverable is created as a small report to summarize the achieved results by developer teams in both use cases. Finally, as this software will introduce also third party models, it will be also tested in WP5 and WP7 which completes the aforementioned three diferente use cases.

3. SaaS as Solution for REVaMP

3.1. What

The idea of centralizing the business applications has been discussed since at least the 1960s. In the 1990s, the idea evolved with the rise of the Internet in application service providers (ASPs), that is, with the emergence of third-party applications managed and hosted by an ASP provider, but some software still required installation on users' computers.

SaaS is an evolution of the ASP model: in this solution, providers and distributors manage their own software and no installation is required, since the software is distributed instantly over the Internet, that is, through the cloud. Cloud computing allows businesses to consume computing resources over the Internet as a service (in the same way as electricity or water).

The SaaS-based cloud computing model now offers businesses significant efficiency and cost savings, but a number of events had to occur before SaaS could become a feasible option:

- Increased access to high-speed Internet connections.
- Standardization of digital technologies.
- Increased popularity and use of web-type interfaces
- Rapid and widespread adoption of mobile devices.

Companies list many reasons using cloud-based SaaS solutions, including increased efficiency and cost-effectiveness, along with scalability, remote access to services, and automation of upgrades. More accurately:

- *Low infrastructure and configuration costs.* There are no expenses to amortize in the balance sheet. Compared to traditional software packages, the SaaS model can be extremely cost-effective. Users always have the latest version of the software, usually without the need to pay for updates or maintenance.
- *Access from anywhere.* Services can be accessed from any other connected device, making it easy to work from home, reduce lost productivity from remote employees, and save office space.
- *Fast implementation.* Setup is quick and easy, compared to an in-house IT infrastructure.
- *Expandability.* Access is usually per license and monthly, so a company only takes what it needs. Adding more licenses and extending applications as requirements grow means convenience and scalability.
- *Automatic and frequent updates.* The provider takes care of all the updates, which can be weekly or monthly. There is no need to maintain or make backwards compatible with previous versions of the software.
- Security. The provider manages the security.wqeqwe

3.2. How

3.2.1. .Net and Azure

For the solution that has been carried out in the REVaMP project, the developments were made in .Net. Specifically, .NET is a Microsoft framework that emphasizes network transparency, regardless of hardware platform, and enables rapid application development. Until 2015, .NET only supported Windows, and its code was licensed under a proprietary license. This led to the creation of free implementations, such as Mono. However, Mono continued to have limitations compared to the original one, especially in relation to WinForms (a tool for Windows graphical interfaces), in addition to certain patent problems. For this reason, Microsoft decided to release part of the .NET framework under the name of .NET Core. Later, support for ASP .NET, ML .NET, and WinForms was added. .NET Core is expected to replace the .NET Framework in the future.

In this way, and trying to bring the results of the development closer to the world of the new cloud computing, the creations on this platform are easily extendable or deployable in Microsoft Azure. Microsoft Azure (formerly Windows Azure and Azure Services Platform) is a cloud computing service created by Microsoft to build, test, deploy, and manage applications and services using its data centers. It provides Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) and supports many different programming languages, tools and frameworks, including specific Microsoft and third-party software and systems. Azure was announced in October 2008, beginning with the codename "Project Red Dog"1 and released on February 1, 2010 as "Windows Azure" before being rebranded as "Microsoft Azure" on March 25, 2014.

The features of Microsoft Azure are as listed below:

- Compute: The Microsoft Azure runs Windows Server-based applications. These applications can be built using the .NET Framework in languages like C# and Visual Basic, or implemented without .NET in C++, Java, and other languages.
- Storage: Binary Large Objects (BLOBs) provide queues for communication between Windows Azure application components and provide a table class with a simple query language.
- Infrastructure services: possibility of easily deploying virtual machines with Windows Server or Linux distributions.
- Fabric controller: Microsoft Azure runs on a large number of machines. The fabric controller's job is to combine the machines in a single Microsoft Azure data center into a harmonious whole. Microsoft Azure compute and storage services are deployed on top of all this compute power.
- Content Delivery Network (CDN): Caching frequently accessed data close to your users speeds access to that data.
- Connect: Organizations interact with cloud applications as if they were inside the organization's own firewall.
- Identity and access management: The Active Directory solution enables centralized and easy management of identity and access control. This solution is perfect for account management and synchronization with local directories.

3.2.2. .Net in REVaMP, .Net Core – Web API

For this development a technological study was done. Inside .Net there are several technologies that are able to provide these kind of solutions. That was the reason to extract their way of working and compare and contrast the several features they provide. Hence, the first thing I would advise is to fully understand the problem we are facing in order to know how to solve it in the most efficient and correct way possible, using the tools we have at our disposal. Our options are WCF or Web API from the perspective of the problem to solve.

To know that WCF (Windows Communication Foundation) is a framework for creating service-oriented applications. It was originally conceived to create services based on SOAP and other related bindings. However, regarding the Web API technology, it is a lightweight alternative to WCF to build REST services that can be consumed by different clients. As a first approach, Web API should be the option for this project.

Some of the main characteristics of both technologies are compared below:

Table 1.- WCF and Web API comparison

WCF	Web API
Designed to build: • SOAP services that return data in XML • Secure, reliable and transactional services that could support messaging and be accessible through a variety of transports	It was intended to build: • REST services over HTTP • Services oriented to resources through HTTP, being able to take advantage of the complete characteristics of the protocol such as: • Version control • Cache control in browsers • Concurrency control through Etags
Duplex communication is feasible	It is light and adapts well to devices that have limited bandwidth such as smartphones, tablets...
It is more versatile than Web API, since it can use more protocols: TCP, HTTP, HTTPS, Named Pipes, MSMQ...	It allows exposing services in a wide range of clients: web browsers, tablets, mobiles...
It has a wide setup and therefore a higher learning curve	Easier and simpler to use than WCF

After taking into account the characteristics of each of them, as well as the final objective of the development, the selection of Web API has been made. This is because special scenarios such as one-way messaging, message queuing and duplex communication will not be used. Likewise, fast transport channels such as TCP, Named Pipes and UDP are not needed.

However, the creation of Services oriented to resources through HTTP (URIs, request/response headers, browser cache control, version control, concurrency using ETags, different content formats) is rewarded, in addition to allowing them to be cosumed by a wide range of customers.

3.2.3. Swagger as Documentation

Trying to be at the forefront of the generation of API Rest systems, Swagger is selected as documentation of the API itself. Swagger is used today as a de facto standard to achieve this goal. A Swagger document is the REST API equivalent of a WSDL document for a SOAP-based web service.

The Swagger document specifies the list of resources available in the REST API and the operations that can be called on these resources. The Swagger document also specifies the list of parameters for an operation, which includes the name and type of the parameters, whether the parameters are required or optional, and information about the acceptable values for these parameters. In addition, the Swagger document can include additional JSON schema that describes the structure of the request body that is sent to an operation in a REST API, and the JSON schema describes the structure of response bodies returned from an operation.

Swagger documents must be in JSON format with a .json file extension or in YAML format with a .yaml or .yml file extension.

A range of third-party tools are available that you can use with Swagger documents. We specifically want to highlight the Swagger User Interface. It has been included in the development because it allows to view and test a REST API defined with Swagger from any web browser. Built-in test functions allow to specify the inputs of an operation defined in that REST API, call that operation from the web browser, and inspect the results of calling that operation.

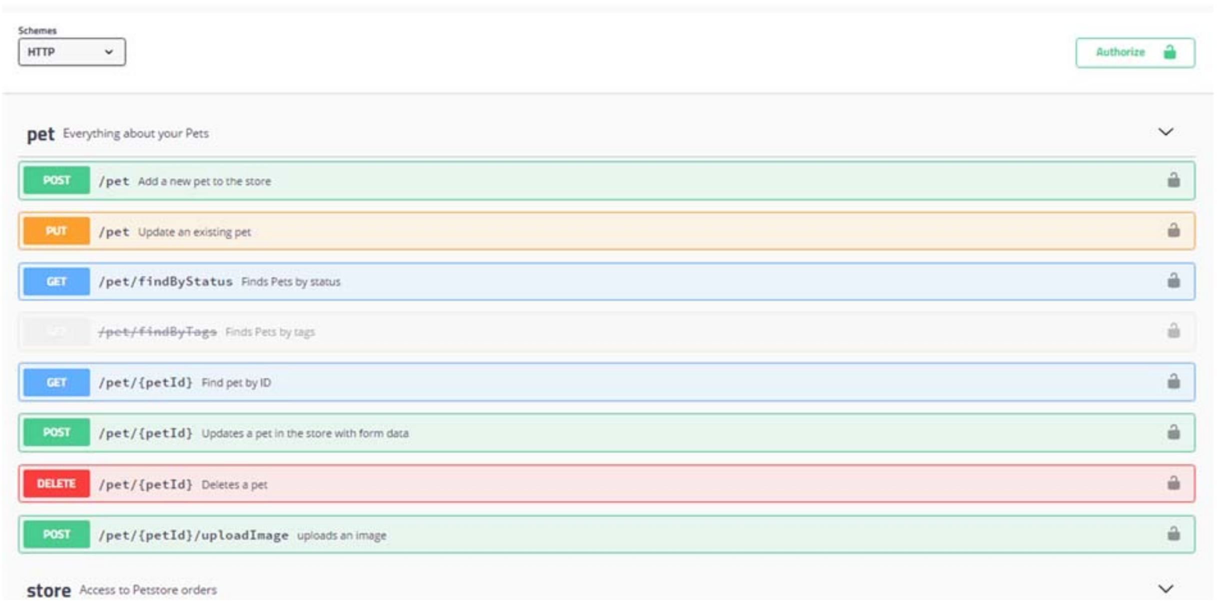

Schemes
HTTP
Authorize
pet Everything about your Pets
POST /pet Add a new pet to the store
PUT /pet Update an existing pet
GET /pet/findByStatus Finds Pets by status
GET ~~/pet/findByTags~~ Finds Pets by tags
GET /pet/{petId} Find pet by ID
POST /pet/{petId} Updates a pet in the store with form data
DELETE /pet/{petId} Deletes a pet
POST /pet/{petId}/uploadImage uploads an image
store Access to Petstore orders

Figure 1.- Swagger UI example

3.3. Where

The adopted solution is deployable in several ways. The best feature is that the development is the same and we do not need to change anything in the source code.

- Local Solutions:

1. Launching al service in a development IIS that Microsoft Visual Studio is able to create. Here you can see the local debug server created for this project. Moreover, please take into account that the communication is able using HTTP and HTTPs.

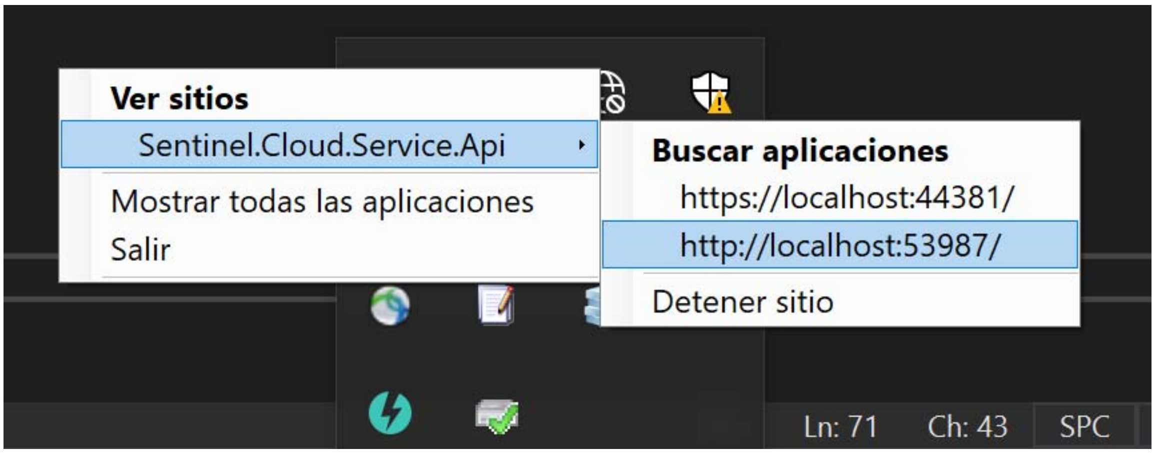


Figure 2.- Service up and running in a "Development IIS" that works as a Console Program with Network Connection.

2. Deploying the service in a real IIS located in a local PC or an imtranet PC. In this case the way of working is the same, but using the real Web Server where the application is deployed.

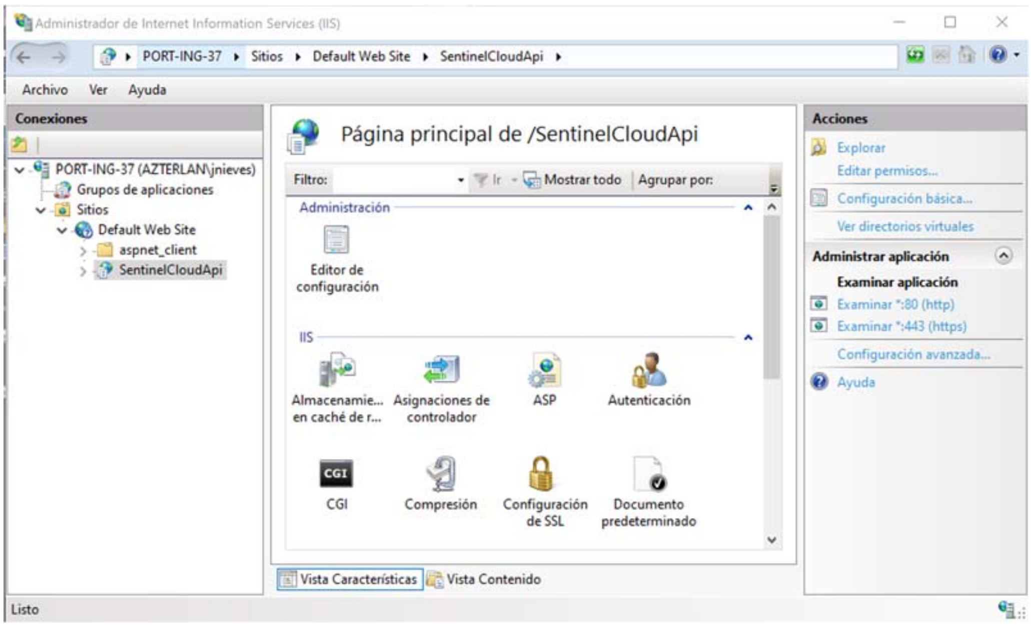


Figure 3.- Local IIS installed in the PC.

3. Generalization of storage creating several containers (DB and Service Containers) with Docker. This new way of deploying the solution is easy for scalability, remote deployment, givin the possibilities to be managed through Kubernetes (high scalability solution).

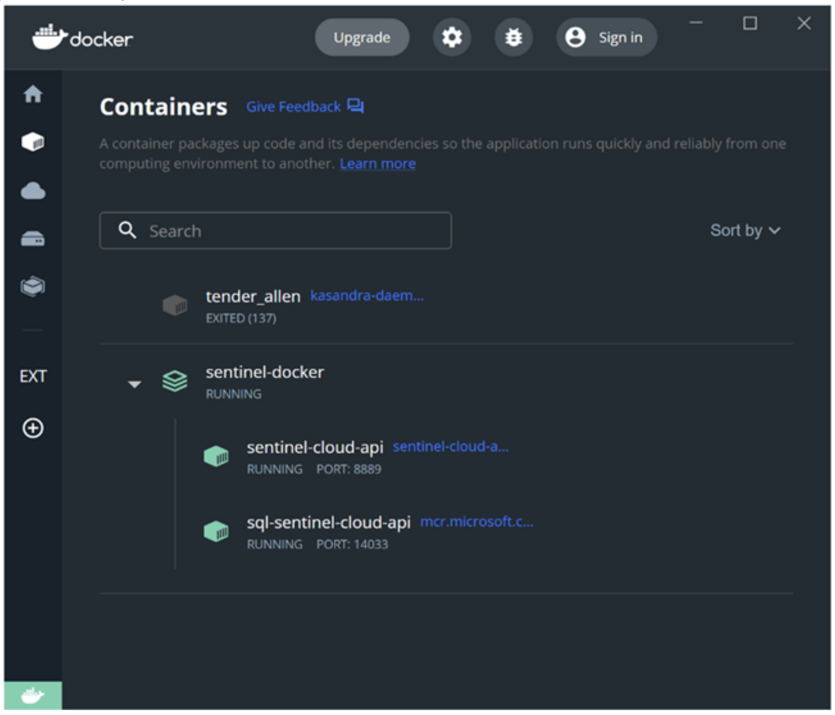

docker
Upgrade
Sign in
Containers
Give Feedback
A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more
Search
Sort by
tender_allen kasandra-daem...
EXITED (137)
sentinel-docker
RUNNING
sentinel-cloud-api sentinel-cloud-a...
RUNNING PORT: 8889
sql-sentinel-cloud-api mcr.microsoft.c...
RUNNING PORT: 14033
EXT

Figure 4.- Docker with .Net Core preinstalled and with Internet Connection.

- Remote Solutions:

  For our remote deployments we can manage the same solutions. First of all, a IIS Web server can be deployed wherever you want. If this server is able to be consumed from any place, the service will be accessible as a service. More over, the service can be deployed directly in Azure, as well as in a IIS server in Azure. Finally, Docker containers can also be deployed in Azure, giving access to the service.

3.4. Why

The selection of this type of solution has been simple. As can be seen in Figure 5, cloud computing is one of the basic technologies to achieve what is known today as *Industry 4.0*. In addition, our service brings together other features of the technologies since it has security and the management of cognitive computing and AI. That is why everything remains within the desired scope, providing an Industry 4.0 solution with the expected characteristics.

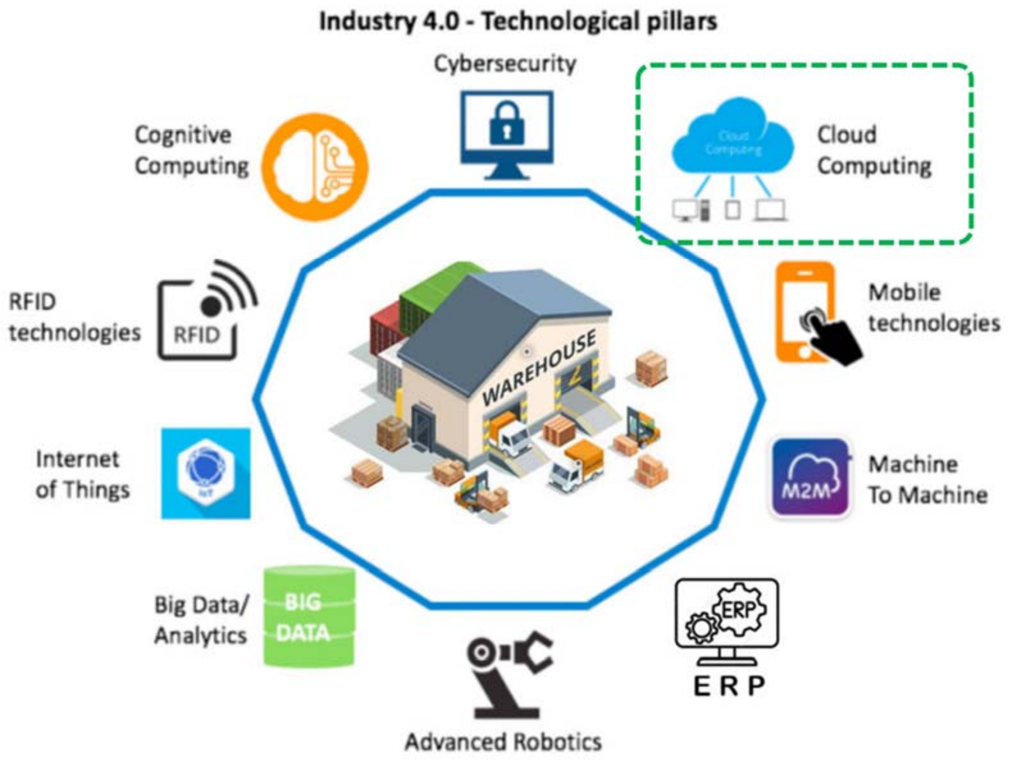


Figure 5.- Industry 4.0 Technolofical Pillars.

3.5. Design

3.5.1. Technologies Description

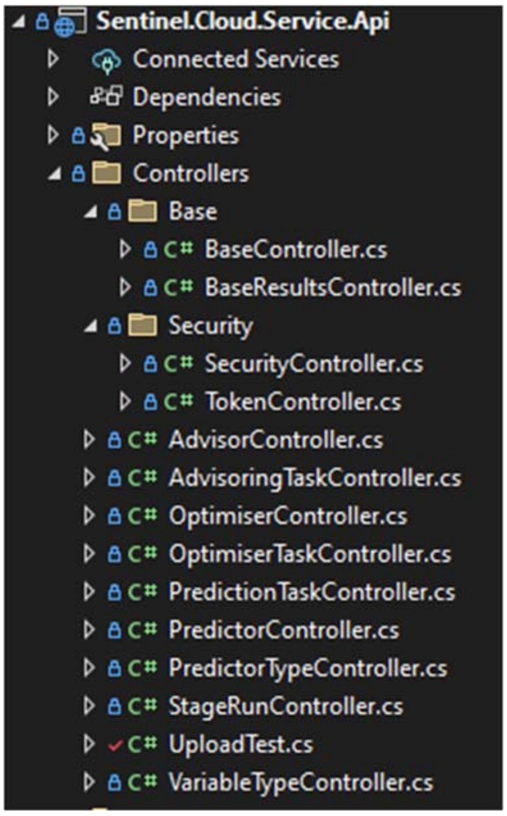
After the basic study of functional requirements that the system to be developed must have, as included in the Grant Agreement, different design proposals have been studied, selecting those listed below:

- *API Restful.*

The RESTful API is an interface that two computing systems use to securely exchange information over the Internet. Most enterprise applications must communicate with other internal or third-party applications to perform various tasks. For example, to generate a monthly payroll, an internal accounting system must share data with the customer's banking system to automate billing and communicate with an internal timesheet application. RESTful APIs support this information exchange because they follow secure, reliable, and efficient software communication standards.

The basic function of a RESTful API is the same as browsing the Internet. When it requires a resource, the client contacts the server through the API. API developers explain how the client should use the REST API in the server application API documentation.

The details of the REST API request and response vary a bit depending on how the API developers have designed the API.



```
Sentinel.Cloud.Service.Api
  Connected Services
  Dependencies
  Properties
  Controllers
    Base
      BaseController.cs
      BaseResultsController.cs
    Security
      SecurityController.cs
      TokenController.cs
    AdvisorController.cs
    AdvisoringTaskController.cs
    OptimiserController.cs
    OptimiserTaskController.cs
    PredictionTaskController.cs
    PredictorController.cs
    PredictorTypeController.cs
    StageRunController.cs
    UploadTest.cs
    VariableTypeController.cs
```

Figure 6.- Controllers for each resource, for RESTFul API.

- Richardson Maturity Model

The Richardson Maturity Model rates an API based on its adherence to the constraints of the REST architecture. The tighter your implementation, the higher up the ladder it will be. There are 4 levels: the first is 0, which designates the lowest level of implementation and the last is 3, which is the most suitable and, therefore, is RESTful. Thus, level 0 uses the HTTP protocol as a transport but not to indicate the state of the application. Level 1 distinguishes between different resources, but uses a single HTTP method. Level 2 makes full use of HTTP verbs combined with resource nouns. Finally, level 3 uses HATEOAS (Hypermedia As The Engine Of Application State) to manage the state of the application. In the REVaMP case we have developed a Leve 3 API REST.

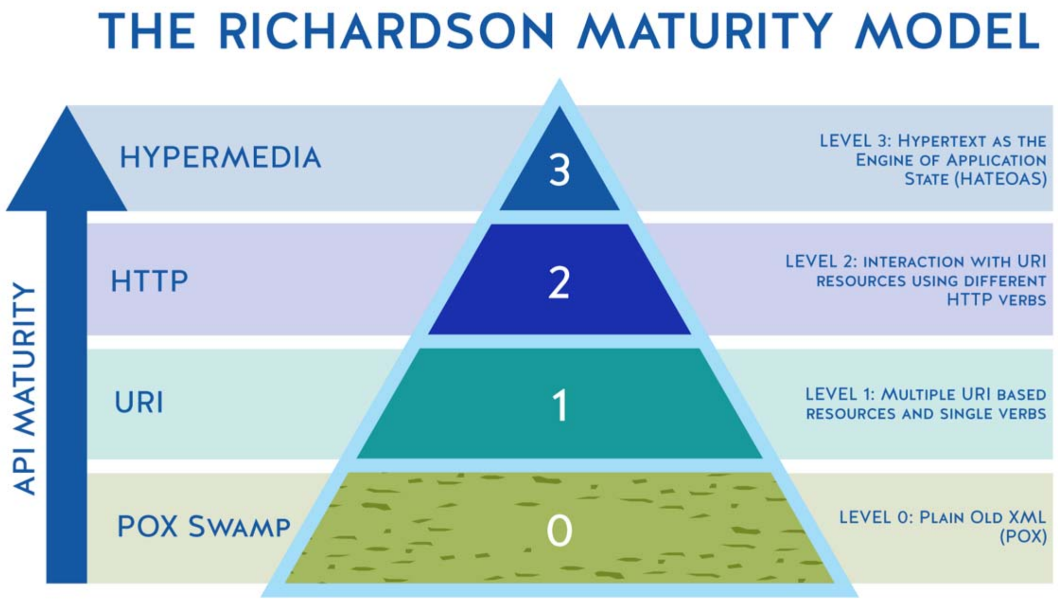

THE RICHARDSON MATURITY MODEL
API MATURITY
HYPERMEDIA
3
LEVEL 3: HYPERTEXT AS THE ENGINE OF APPLICATION STATE (HATEOAS)
HTTP
2
LEVEL 2: INTERACTION WITH URI RESOURCES USING DIFFERENT HTTP VERBS
URI
1
LEVEL 1: MULTIPLE URI BASED RESOURCES AND SINGLE VERBS
POX SWAMP
0
LEVEL 0: PLAIN OLD XML (POX)

Figure 7.- Richardson Maturity Model diagram.

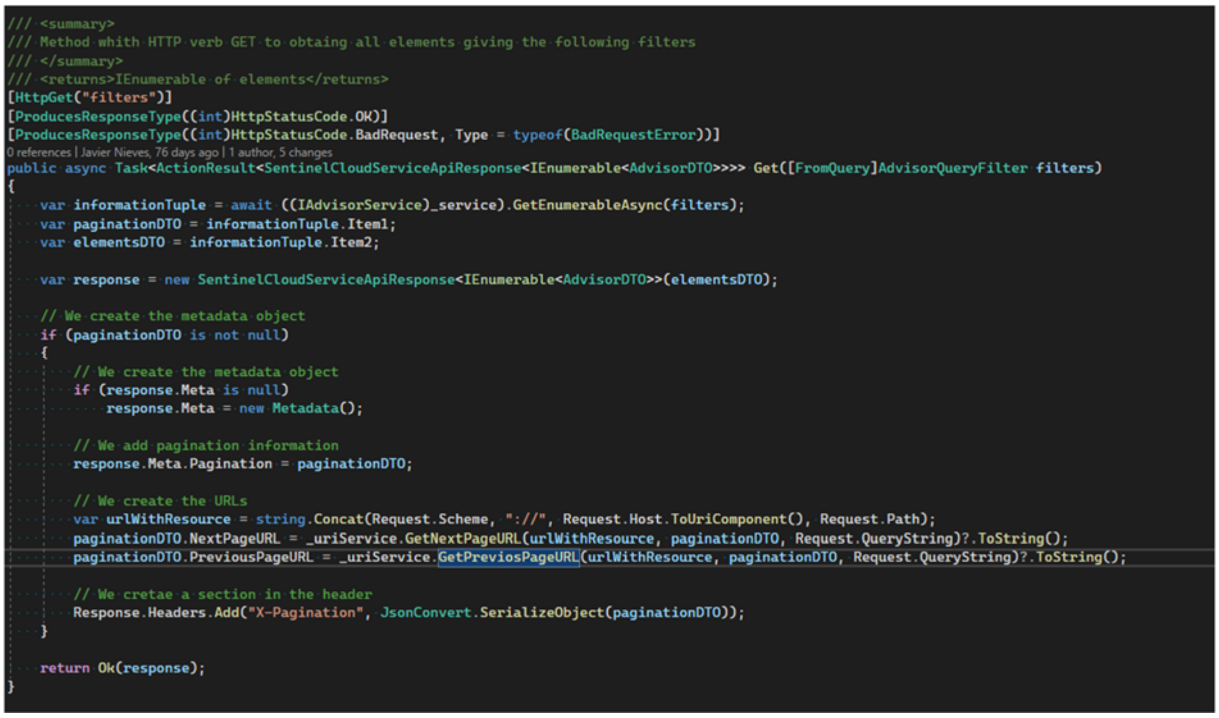
- HATEOAS

In the detailed development, this principle has been followed. Specifically, HATEOAS (Hypermedia as the Engine of Application State) is a constraint of the REST application architecture. HATEOAS keeps the REST style architecture unique from most other network application architectures.

The term “hypermedia” refers to any content that contains links to other forms of media such as images, movies and text.

REST architectural style lets us use the hypermedia links in the API response contents. It allows the client to dynamically navigate to the appropriate resources by traversing the hypermedia links.

Navigating hypermedia links is conceptually the same as browsing through web pages by clicking the relevant hyperlinks to achieve a final goal.



```
/// <summary>
/// Method whith HTTP verb GET to obtaing all elements giving the following filters
/// </summary>
/// <returns>IEnumerable of elements</returns>
[HttpGet("filters")]
[ProducesResponseType((int)HttpStatusCode.OK)]
[ProducesResponseType((int)HttpStatusCode.BadRequest, Type = typeof(BadRequestError))]
0 references | Javier Nieves, 76 days ago | 1 author, 5 changes
public async Task<ActionResult<SentinelCloudServiceApiResponse<IEnumerable<AdvisorDTO>>>> Get([FromQuery]AdvisorQueryFilter filters)
{
    var informationTuple = await ((IAdvisorService)_service).GetEnumerableAsync(filters);
    var paginationDTO = informationTuple.Item1;
    var elementsDTO = informationTuple.Item2;

    var response = new SentinelCloudServiceApiResponse<IEnumerable<AdvisorDTO>>(elementsDTO);

    // We create the metadata object
    if (paginationDTO is not null)
    {
        // We create the metadata object
        if (response.Meta is null)
            response.Meta = new Metadata();

        // We add pagination information
        response.Meta.Pagination = paginationDTO;

        // We create the URLs
        var urlWithResource = string.Concat(Request.Scheme, "://", Request.Host.ToUriComponent(), Request.Path);
        paginationDTO.NextPageURL = _uriService.GetNextPageURL(urlWithResource, paginationDTO, Request.QueryString)?.ToString();
        paginationDTO.PreviousPageURL = _uriService.GetPreviosPageURL(urlWithResource, paginationDTO, Request.QueryString)?.ToString();

        // We cretae a section in the header
        Response.Headers.Add("X-Pagination", JsonConvert.SerializeObject(paginationDTO));
    }

    return Ok(response);
}
```

Figure 8.- Example of method that includes, pagination, HATEOAS (generation of URLs), HTTP methods and resource access.

- Protecting the service with HTTPS

HTTPS (HyperText Transfer Protocol Secure) is an Internet communication protocol that protects the integrity and confidentiality of user data between their computers and the website. Thus, we have selected to complete the HTTP protocol in our service.

Sending data using the HTTPS protocol is protected by the Transport Layer Security (TLS) protocol, which provides these three main layers of security:

- Encryption: The data exchanged is encrypted to keep it safe from prying eyes. That means that when a user is browsing a website, no one can "eavesdrop" on their conversations, track their activities across different pages, or steal information from them.
- Data Integrity: Data cannot be changed or corrupted during transfers, intentionally or otherwise, without detection.
- Authentication: Prove that your users are communicating with the intended website. It provides protection against man-in-the-middle attacks and builds user trust, ultimately creating other business benefits.

- CORS

Cross-origin resource sharing, or CORS, is a mechanism that allows restricted resources to be requested. CORS defines a way in which the browser and server can interact to determine if it is safe to allow a cross-origin request. This allows more freedom and functionality than same-origin requests, but is additionally safer than simply allowing all cross-origin requests. Clean Architecture

- Onion Architecture

The Onion Architecture is an Architectural *Pattern* that enables maintainable and evolutionary enterprise systems. It is intended for use at a Solution/System level.

The primary proposition of this architecture is good coupling. Accurately, it is the level of dependency of one thing upon another. The higher the coupling, the lower the ability to change and evolve the system. Dependencies should be inward and never outward. Code should depend only on the same layer or layers more central to itself. Moreover, Business Logic behaviour is declared as contracts with the use of interfaces in a Object-Oriented context. Finally, each layer bounds together concepts that will have a similar rate of change.

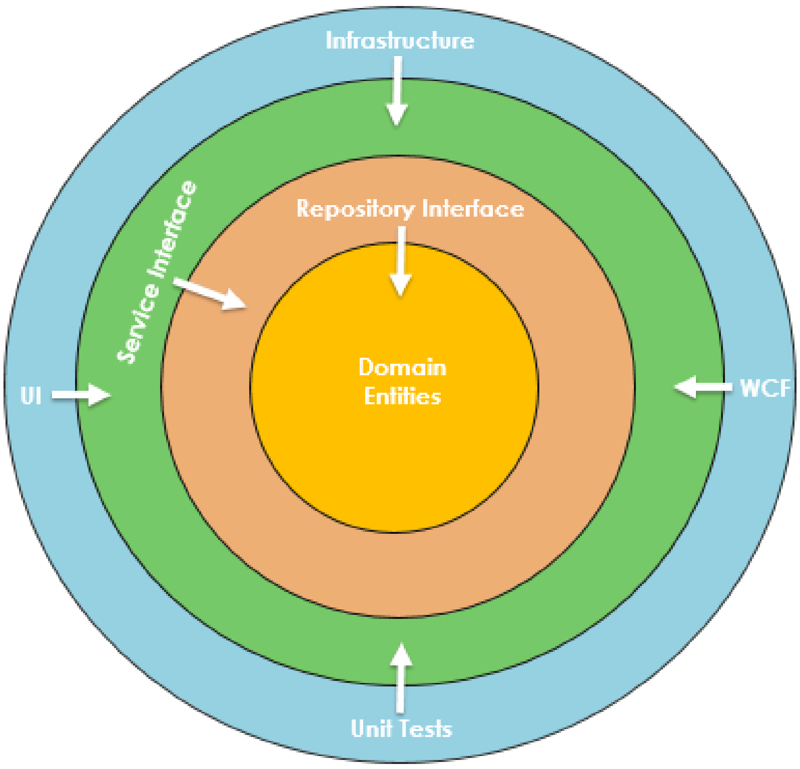


Figure 9.- Onion Architecture.

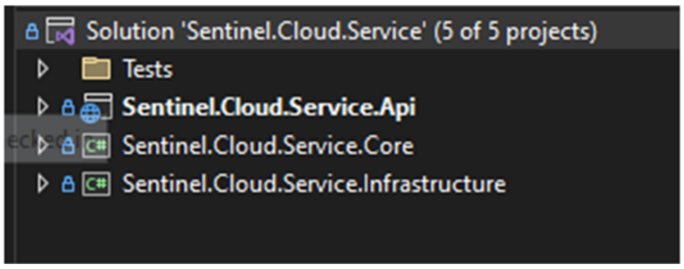

```
Solution 'Sentinel.Cloud.Service' (5 of 5 projects)
  Tests
  Sentinel.Cloud.Service.Api
  Sentinel.Cloud.Service.Core
  Sentinel.Cloud.Service.Infrastructure
```

Figure 10.- Our service implementing Onion Architecture.

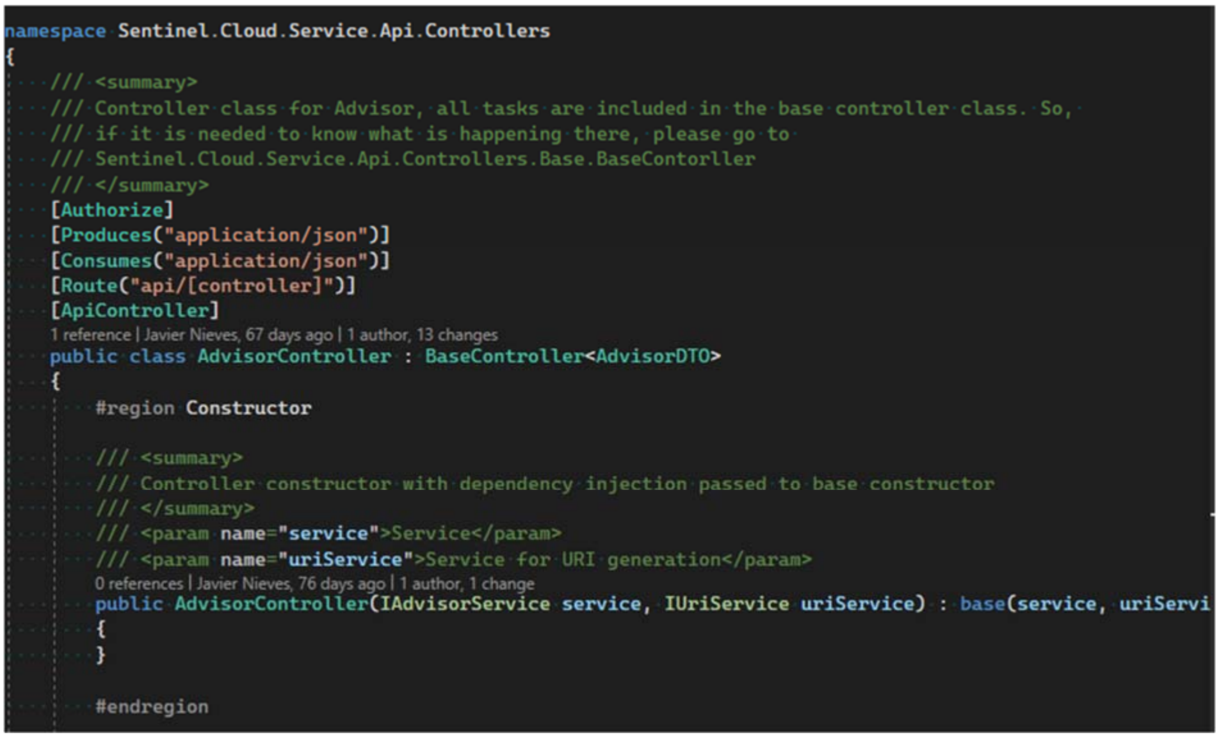
- Dependency Injection

Dependency inversion is a principle that describes a set of techniques intended to decrease the coupling between the components of an application. It is one of the most popular and used SOLID principles in the creation of applications, frameworks and components due to the advantages it brings to them. Therefore, we have decided to incorporate it into the solution created in the REVaMP project.

Very briefly, the Dependency Inversion Principle proposes to avoid rigid dependencies between components through the following techniques:

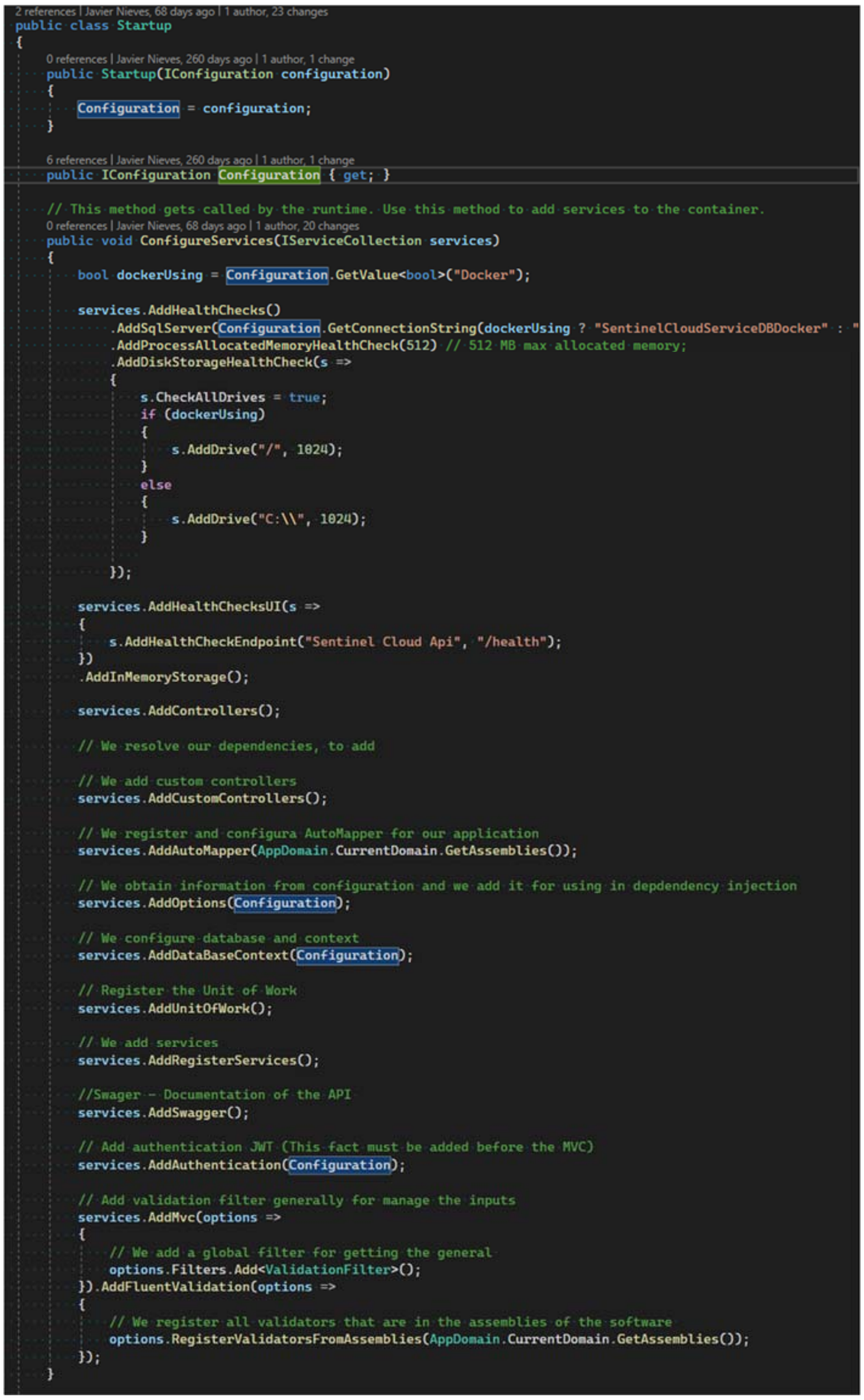
- Use abstractions (interfaces) instead of direct references between classes, which makes it easy for us to replace components.
- Make a class receive references to the components it needs to function, instead of allowing it to instantiate them directly or through factories.

A dependency injection system is in charge of instantiating the classes that we need and supplying ("injecting") the dependencies by sending the appropriate parameters to the constructor.



```
namespace Sentinel.Cloud.Service.Api.Controllers
{
    /// <summary>
    /// Controller class for Advisor, all tasks are included in the base controller class. So,
    /// if it is needed to know what is happening there, please go to
    /// Sentinel.Cloud.Service.Api.Controllers.Base.BaseContorller
    /// </summary>
    [Authorize]
    [Produces("application/json")]
    [Consumes("application/json")]
    [Route("api/[controller]")]
    [ApiController]
    1 reference | Javier Nieves, 67 days ago | 1 author, 13 changes
    public class AdvisorController : BaseController<AdvisorDTO>
    {
        #region Constructor

        /// <summary>
        /// Controller constructor with dependency injection passed to base constructor
        /// </summary>
        /// <param name="service">Service</param>
        /// <param name="uriService">Service for URI generation</param>
        0 references | Javier Nieves, 76 days ago | 1 author, 1 change
        public AdvisorController(IAdvisorService service, IUriService uriService) : base(service, uriServi
        {
        }

        #endregion
```

Figure 11.- Consturctor receiving the injection of the services.



```
2 references | Javier Nieves, 68 days ago | 1 author, 23 changes
public class Startup
{
    0 references | Javier Nieves, 260 days ago | 1 author, 1 change
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    6 references | Javier Nieves, 260 days ago | 1 author, 1 change
    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to the container.
    0 references | Javier Nieves, 68 days ago | 1 author, 20 changes
    public void ConfigureServices(IServiceCollection services)
    {
        bool dockerUsing = Configuration.GetValue<bool>("Docker");

        services.AddHealthChecks()
            .AddSqlServer(Configuration.GetConnectionString(dockerUsing ? "SentinelCloudServiceDBDocker" : "
            .AddProcessAllocatedMemoryHealthCheck(512) // 512 MB max allocated memory;
            .AddDiskStorageHealthCheck(s =>
            {
                s.CheckAllDrives = true;
                if (dockerUsing)
                {
                    s.AddDrive("/", 1024);
                }
                else
                {
                    s.AddDrive("C:\\", 1024);
                }

            });

        services.AddHealthChecksUI(s =>
        {
            s.AddHealthCheckEndpoint("Sentinel Cloud Api", "/health");
        })
        .AddInMemoryStorage();

        services.AddControllers();

        // We resolve our dependencies, to add

        // We add custom controllers
        services.AddCustomControllers();

        // We register and configura AutoMapper for our application
        services.AddAutoMapper(AppDomain.CurrentDomain.GetAssemblies());

        // We obtain information from configuration and we add it for using in depdendency injection
        services.AddOptions(Configuration);

        // We configure database and context
        services.AddDataBaseContext(Configuration);

        // Register the Unit of Work
        services.AddUnitOfWork();

        // We add services
        services.AddRegisterServices();

        //Swager - Documentation of the API
        services.AddSwagger();

        // Add authentication JWT (This fact must be added before the MVC)
        services.AddAuthentication(Configuration);

        // Add validation filter generally for manage the inputs
        services.AddMvc(options =>
        {
            // We add a global filter for getting the general
            options.Filters.Add<ValidationFilter>();
        }).AddFluentValidation(options =>
        {
            // We register all validators that are in the assemblies of the software
            options.RegisterValidatorsFromAssemblies(AppDomain.CurrentDomain.GetAssemblies());
        });
    }
```

Figure 12.- Startup method that register all services for being injected.

- Repository Pattern and Unity of Work

A typical software application will need to access some kind of data store to perform CRUD (Create, Read, Update, Delete) operations on the data, typically this could be some kind of database, file system or any type of storage mechanism used to persist information.

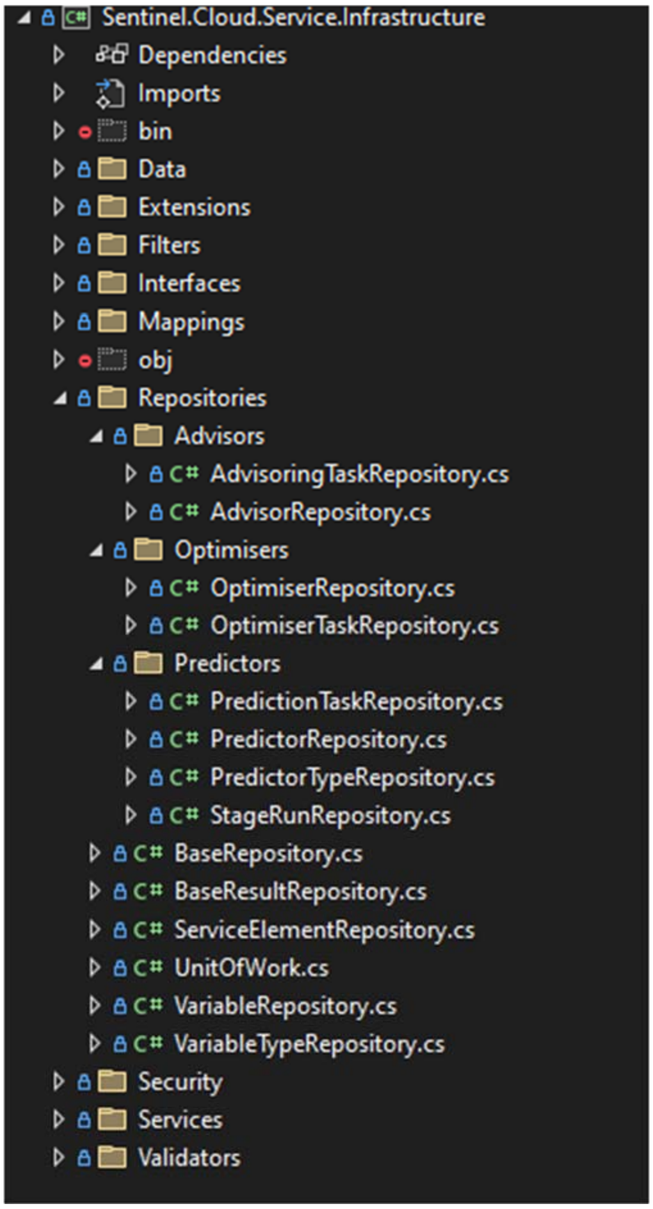
In some cases saving or creating data may require persistence on new files or creating a new row in a database table, in other cases saving new data may require multiple interactions with various web-based services or applications and other services that are not generally managed by ourselves.

For both cases we can make use of the Repository Pattern and a Unit of Work.

The Repository Pattern consists of separating the logic that retrieves the data and assigns it to an entity model from the business logic that acts on the model.This allows the business logic to be independent of the type of data that the source layer understands. In short, a repository mediates between the domain and data mapping layers, acting as a collection of domain objects in memory.

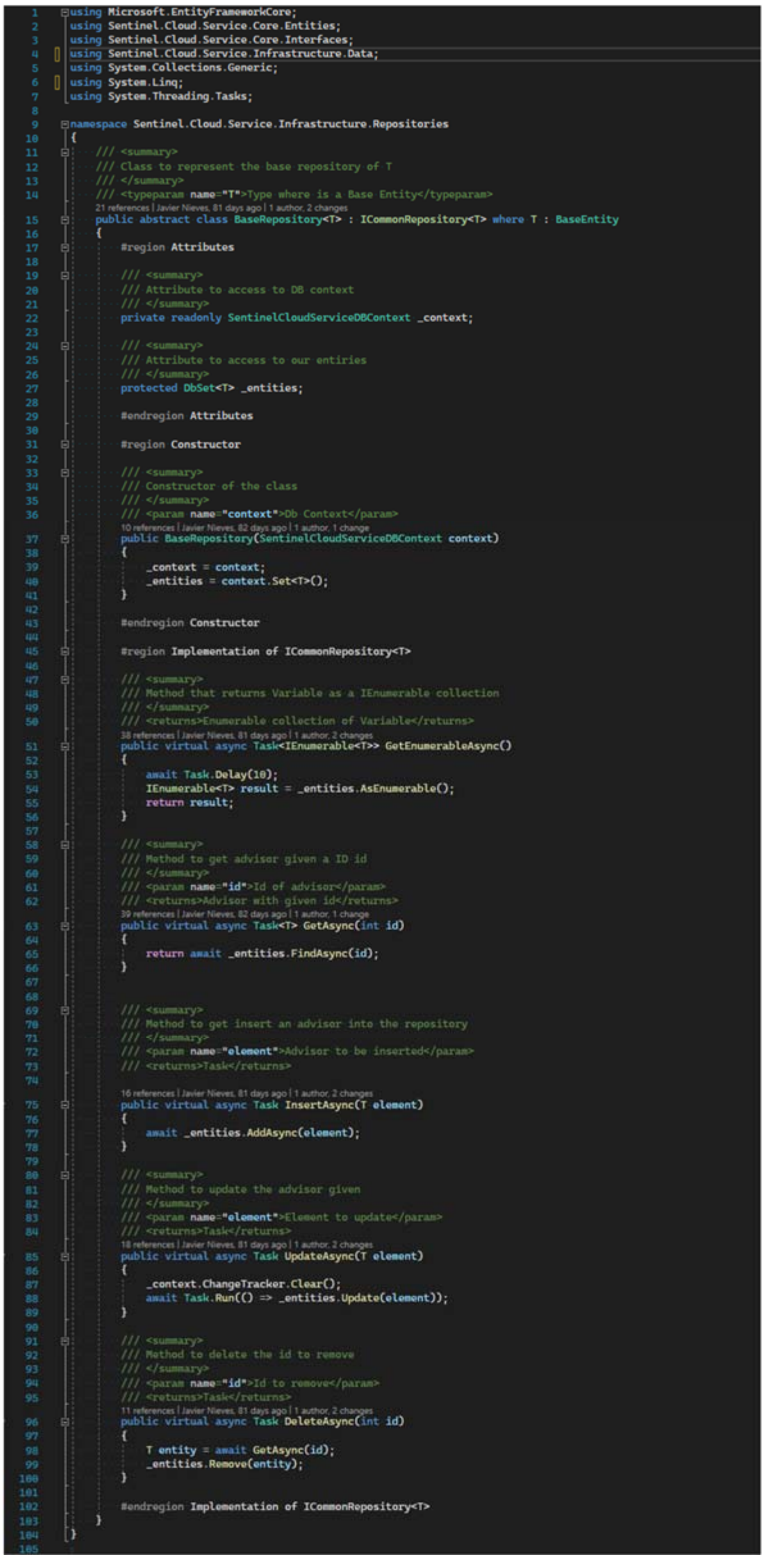
The Unit of Work is to centralize connections to the database by keeping track of everything that happens during a transaction when using data layers and will rollback the transaction if Commit() has not been called or there are logical inconsistencies.

Both patterns have been followed for the development of the REVaMP service.



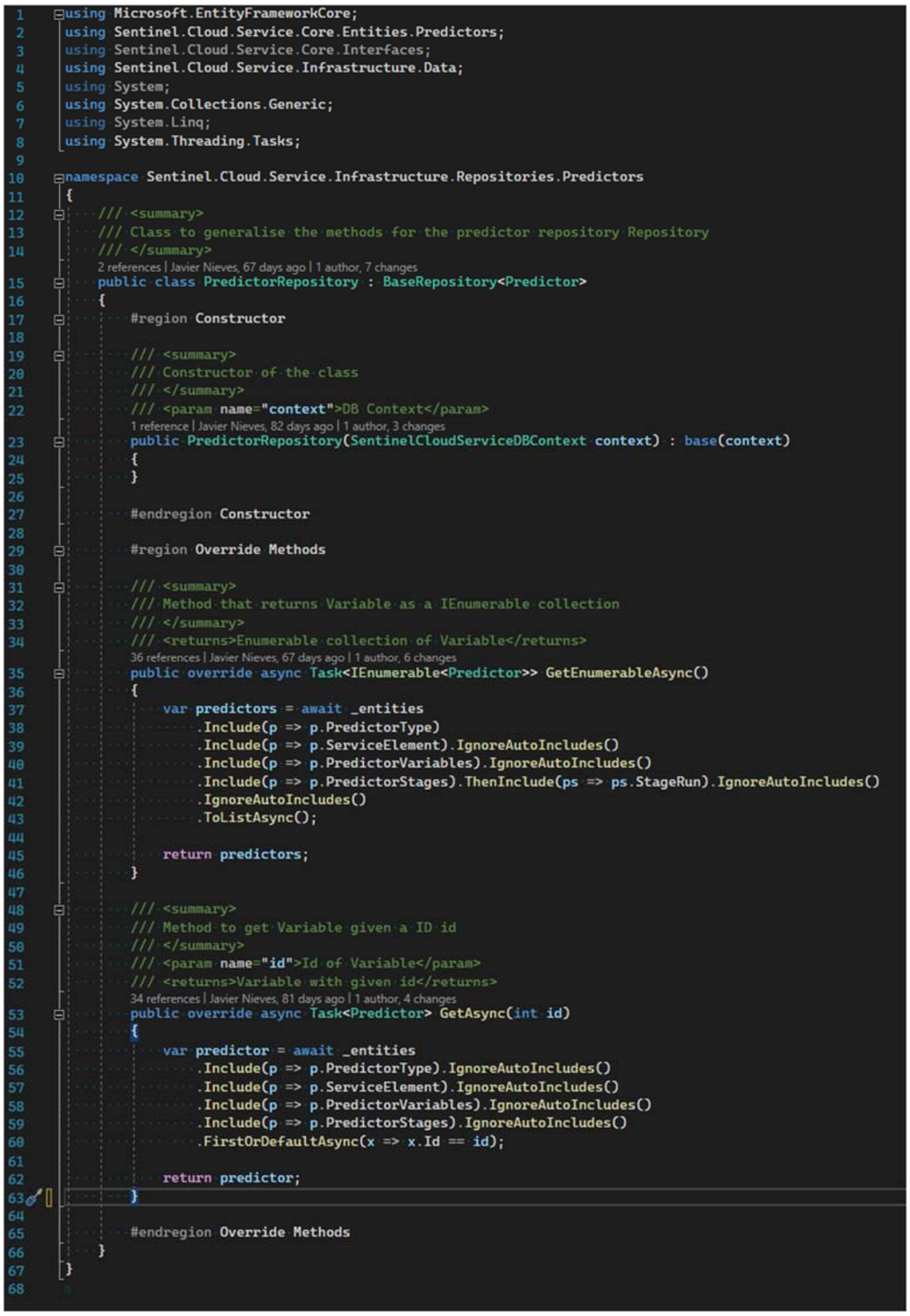
```
◢ Sentinel.Cloud.Service.Infrastructure
    ▷ Dependencies
    ▷ Imports
    ▷ bin
    ▷ Data
    ▷ Extensions
    ▷ Filters
    ▷ Interfaces
    ▷ Mappings
    ▷ obj
    ◢ Repositories
        ◢ Advisors
            ▷ AdvisoringTaskRepository.cs
            ▷ AdvisorRepository.cs
        ◢ Optimisers
            ▷ OptimiserRepository.cs
            ▷ OptimiserTaskRepository.cs
        ◢ Predictors
            ▷ PredictionTaskRepository.cs
            ▷ PredictorRepository.cs
            ▷ PredictorTypeRepository.cs
            ▷ StageRunRepository.cs
        ▷ BaseRepository.cs
        ▷ BaseResultRepository.cs
        ▷ ServiceElementRepository.cs
        ▷ UnitOfWork.cs
        ▷ VariableRepository.cs
        ▷ VariableTypeRepository.cs
    ▷ Security
    ▷ Services
    ▷ Validators
```

Figure 13.- Infrastructure project with all repositories generated for this solution.



```csharp
using Microsoft.EntityFrameworkCore;
using Sentinel.Cloud.Service.Core.Entities;
using Sentinel.Cloud.Service.Core.Interfaces;
using Sentinel.Cloud.Service.Infrastructure.Data;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Sentinel.Cloud.Service.Infrastructure.Repositories
{
    /// <summary>
    /// Class to represent the base repository of T
    /// </summary>
    /// <typeparam name="T">Type where is a Base Entity</typeparam>
    21 references | Javier Nieves, 81 days ago | 1 author, 2 changes
    public abstract class BaseRepository<T> : ICommonRepository<T> where T : BaseEntity
    {
        #region Attributes

        /// <summary>
        /// Attribute to access to DB context
        /// </summary>
        private readonly SentinelCloudServiceDBContext _context;

        /// <summary>
        /// Attribute to access to our entiries
        /// </summary>
        protected DbSet<T> _entities;

        #endregion Attributes

        #region Constructor

        /// <summary>
        /// Constructor of the class
        /// </summary>
        /// <param name="context">Db Context</param>
        10 references | Javier Nieves, 82 days ago | 1 author, 1 change
        public BaseRepository(SentinelCloudServiceDBContext context)
        {
            _context = context;
            _entities = context.Set<T>();
        }

        #endregion Constructor

        #region Implementation of ICommonRepository<T>

        /// <summary>
        /// Method that returns Variable as a IEnumerable collection
        /// </summary>
        /// <returns>Enumerable collection of Variable</returns>
        38 references | Javier Nieves, 81 days ago | 1 author, 2 changes
        public virtual async Task<IEnumerable<T>> GetEnumerableAsync()
        {
            await Task.Delay(10);
            IEnumerable<T> result = _entities.AsEnumerable();
            return result;
        }

        /// <summary>
        /// Method to get advisor given a ID id
        /// </summary>
        /// <param name="id">Id of advisor</param>
        /// <returns>Advisor with given id</returns>
        39 references | Javier Nieves, 82 days ago | 1 author, 1 change
        public virtual async Task<T> GetAsync(int id)
        {
            return await _entities.FindAsync(id);
        }


        /// <summary>
        /// Method to get insert an advisor into the repository
        /// </summary>
        /// <param name="element">Advisor to be inserted</param>
        /// <returns>Task</returns>

        16 references | Javier Nieves, 81 days ago | 1 author, 2 changes
        public virtual async Task InsertAsync(T element)
        {
            await _entities.AddAsync(element);
        }

        /// <summary>
        /// Method to update the advisor given
        /// </summary>
        /// <param name="element">Element to update</param>
        /// <returns>Task</returns>
        18 references | Javier Nieves, 81 days ago | 1 author, 2 changes
        public virtual async Task UpdateAsync(T element)
        {
            _context.ChangeTracker.Clear();
            await Task.Run(() => _entities.Update(element));
        }

        /// <summary>
        /// Method to delete the id to remove
        /// </summary>
        /// <param name="id">Id to remove</param>
        /// <returns>Task</returns>
        11 references | Javier Nieves, 81 days ago | 1 author, 2 changes
        public virtual async Task DeleteAsync(int id)
        {
            T entity = await GetAsync(id);
            _entities.Remove(entity);
        }

        #endregion Implementation of ICommonRepository<T>
    }
}
```

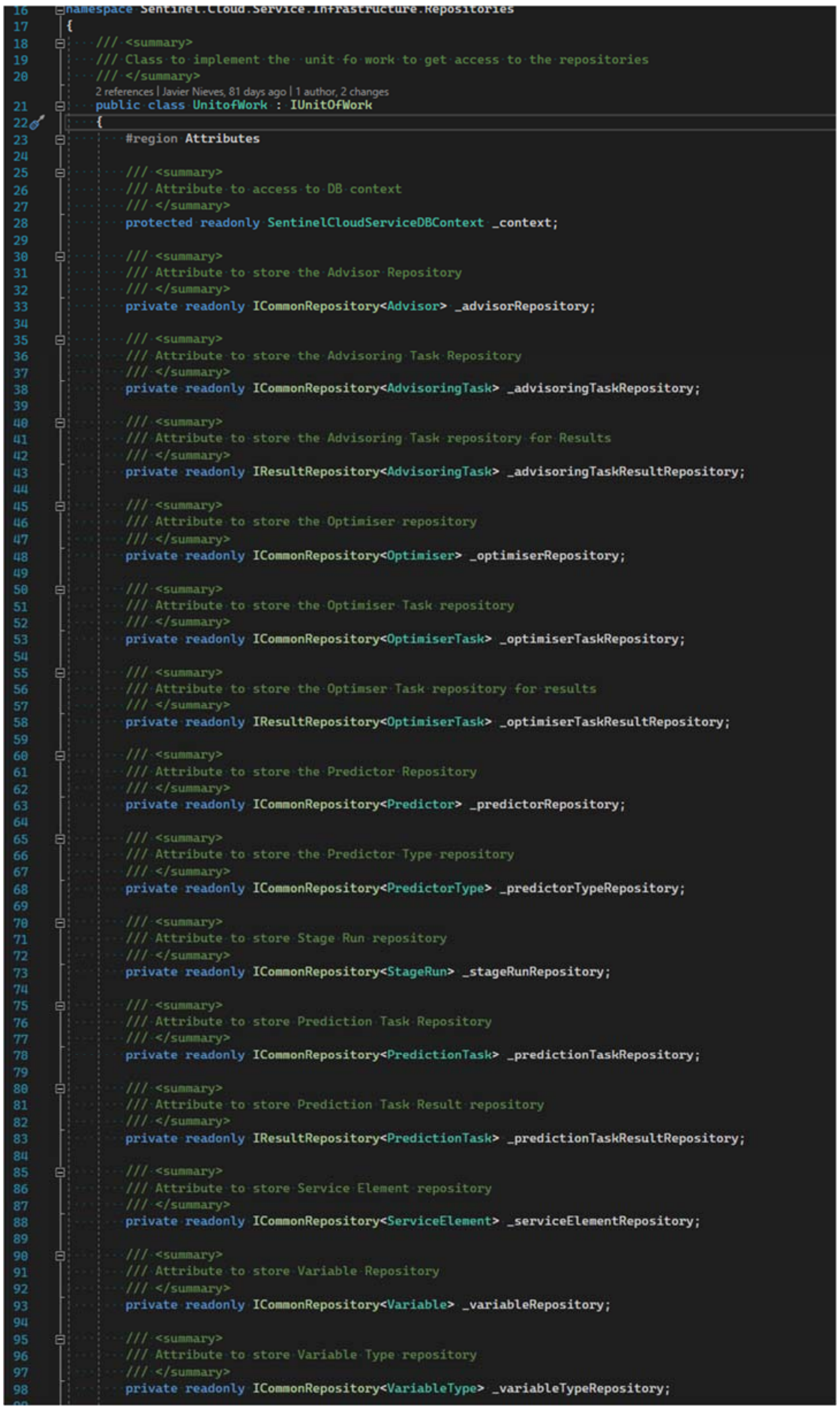
Figure 14.- Base repository created for the project.



```csharp
using Microsoft.EntityFrameworkCore;
using Sentinel.Cloud.Service.Core.Entities.Predictors;
using Sentinel.Cloud.Service.Core.Interfaces;
using Sentinel.Cloud.Service.Infrastructure.Data;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Sentinel.Cloud.Service.Infrastructure.Repositories.Predictors
{
    /// <summary>
    /// Class to generalise the methods for the predictor repository Repository
    /// </summary>
    2 references | Javier Nieves, 67 days ago | 1 author, 7 changes
    public class PredictorRepository : BaseRepository<Predictor>
    {
        #region Constructor

        /// <summary>
        /// Constructor of the class
        /// </summary>
        /// <param name="context">DB Context</param>
        1 reference | Javier Nieves, 82 days ago | 1 author, 3 changes
        public PredictorRepository(SentinelCloudServiceDBContext context) : base(context)
        {
        }

        #endregion Constructor

        #region Override Methods

        /// <summary>
        /// Method that returns Variable as a IEnumerable collection
        /// </summary>
        /// <returns>Enumerable collection of Variable</returns>
        36 references | Javier Nieves, 67 days ago | 1 author, 6 changes
        public override async Task<IEnumerable<Predictor>> GetEnumerableAsync()
        {
            var predictors = await _entities
                .Include(p => p.PredictorType)
                .Include(p => p.ServiceElement).IgnoreAutoIncludes()
                .Include(p => p.PredictorVariables).IgnoreAutoIncludes()
                .Include(p => p.PredictorStages).ThenInclude(ps => ps.StageRun).IgnoreAutoIncludes()
                .IgnoreAutoIncludes()
                .ToListAsync();

            return predictors;
        }

        /// <summary>
        /// Method to get Variable given a ID id
        /// </summary>
        /// <param name="id">Id of Variable</param>
        /// <returns>Variable with given id</returns>
        34 references | Javier Nieves, 81 days ago | 1 author, 4 changes
        public override async Task<Predictor> GetAsync(int id)
        {
            var predictor = await _entities
                .Include(p => p.PredictorType).IgnoreAutoIncludes()
                .Include(p => p.ServiceElement).IgnoreAutoIncludes()
                .Include(p => p.PredictorVariables).IgnoreAutoIncludes()
                .Include(p => p.PredictorStages).IgnoreAutoIncludes()
                .FirstOrDefaultAsync(x => x.Id == id);

            return predictor;
        }

        #endregion Override Methods
    }
}
```

Figure 15.- Predictor repository example.

Next image shows the Unit of Work of the project where all repositories are included. Moreover, then they are accessible thank to the employment of typical C# Properties.



```
namespace Sentinel.Cloud.Service.Infrastructure.Repositories
{
    /// <summary>
    /// Class to implement the  unit fo work to get access to the repositories
    /// </summary>
    2 references | Javier Nieves, 81 days ago | 1 author, 2 changes
    public class UnitofWork : IUnitOfWork
    {
        #region Attributes

        /// <summary>
        /// Attribute to access to DB context
        /// </summary>
        protected readonly SentinelCloudServiceDBContext _context;

        /// <summary>
        /// Attribute to store the Advisor Repository
        /// </summary>
        private readonly ICommonRepository<Advisor> _advisorRepository;

        /// <summary>
        /// Attribute to store the Advisoring Task Repository
        /// </summary>
        private readonly ICommonRepository<AdvisoringTask> _advisoringTaskRepository;

        /// <summary>
        /// Attribute to store the Advisoring Task repository for Results
        /// </summary>
        private readonly IResultRepository<AdvisoringTask> _advisoringTaskResultRepository;

        /// <summary>
        /// Attribute to store the Optimiser repository
        /// </summary>
        private readonly ICommonRepository<Optimiser> _optimiserRepository;

        /// <summary>
        /// Attribute to store the Optimiser Task repository
        /// </summary>
        private readonly ICommonRepository<OptimiserTask> _optimiserTaskRepository;

        /// <summary>
        /// Attribute to store the Optimser Task repository for results
        /// </summary>
        private readonly IResultRepository<OptimiserTask> _optimiserTaskResultRepository;

        /// <summary>
        /// Attribute to store the Predictor Repository
        /// </summary>
        private readonly ICommonRepository<Predictor> _predictorRepository;

        /// <summary>
        /// Attribute to store the Predictor Type repository
        /// </summary>
        private readonly ICommonRepository<PredictorType> _predictorTypeRepository;

        /// <summary>
        /// Attribute to store Stage Run repository
        /// </summary>
        private readonly ICommonRepository<StageRun> _stageRunRepository;

        /// <summary>
        /// Attribute to store Prediction Task Repository
        /// </summary>
        private readonly ICommonRepository<PredictionTask> _predictionTaskRepository;

        /// <summary>
        /// Attribute to store Prediction Task Result repository
        /// </summary>
        private readonly IResultRepository<PredictionTask> _predictionTaskResultRepository;

        /// <summary>
        /// Attribute to store Service Element repository
        /// </summary>
        private readonly ICommonRepository<ServiceElement> _serviceElementRepository;

        /// <summary>
        /// Attribute to store Variable Repository
        /// </summary>
        private readonly ICommonRepository<Variable> _variableRepository;

        /// <summary>
        /// Attribute to store Variable Type repository
        /// </summary>
        private readonly ICommonRepository<VariableType> _variableTypeRepository;
```

Figure 16.- Unit of work of the project.

- Token Based Authentictaion with JWT and Refresh Token

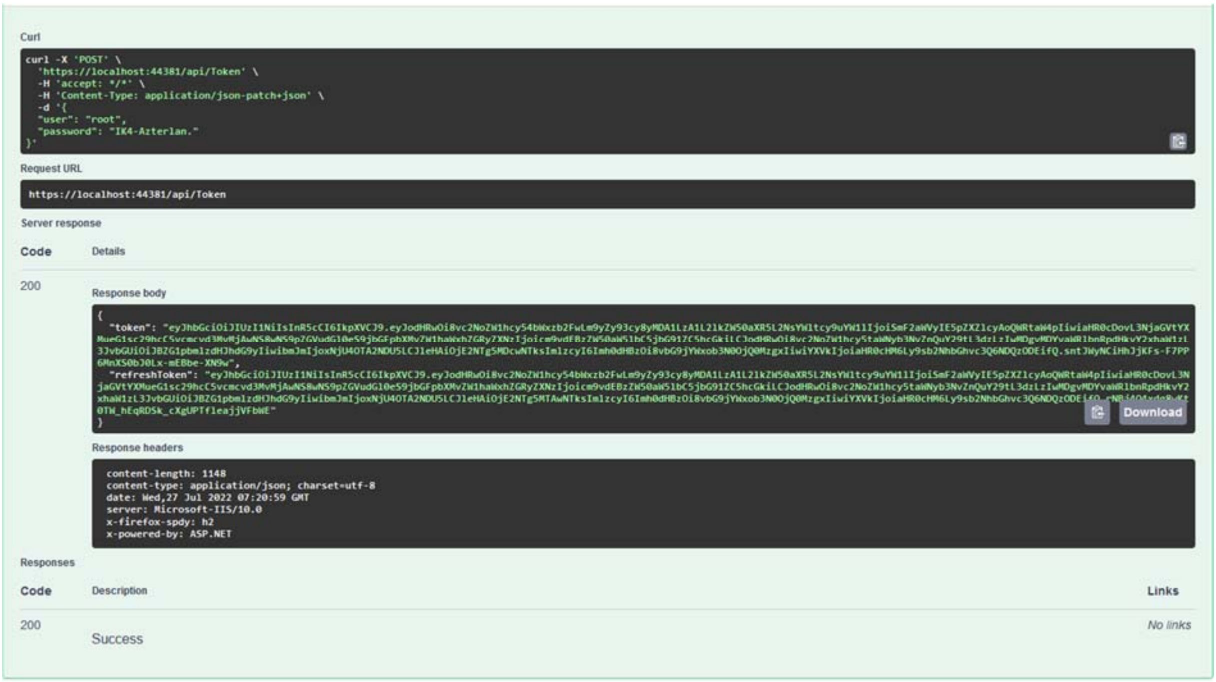
JSON Web Token (abbreviated JWT) is an open standard based on JSON for the creation of access tokens that allow the propagation of identity and privileges or claims. The token is signed by the server's key, so the client and server are both able to verify that the token is legitimate. JSON Web Tokens are designed to be compact, to be able to be sent in URLs - URL-safe- and to be used in Single Sign-On (SSO) scenarios. The privileges of the JSON Web Tokens can be used to propagate the identity of users as part of the authentication process between an identity provider and a service provider, or any other type of privileges required by business processes.

Authorization is achieved when the user successfully enters their credentials, then a JSON Web Token is generated and returned to the client, who has to save it locally, instead of the traditional model of creating a session on the server and returning a cookie.

Whenever the user wants to access a protected route or resource, the client has to send the JWT, usually in the Authorization header using the Bearer scheme.

This is a stateless authentication mechanism since the user's session is never saved in the identity provider or in the service provider. Protected resources will always check for a valid JWT on every access request. If the token is present and valid, the service provider grants access to the protected resources. Since JWTs contain all the necessary information in themselves, the need to query the database or other sources of information multiple times is reduced.

Also, since tokens have a lifetime, the referesh token option has been implemented. Specifically, with the refresh token, the step of re-authenticating can be skipped and with a request to the API, we can obtain a new access token that allows the user to continue accessing the application's resources.



Curl

```
curl -X 'POST' \
  'https://localhost:44381/api/Token' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json-patch+json' \
  -d '{
  "user": "root",
  "password": "IK4-Azterlan."
}'
```

Request URL

```
https://localhost:44381/api/Token
```

Server response

Code	Details
200	Response body

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L21kZW50aXR5L2NsYW1tcy9uYW1lIjoiSmF2aWVyIE5pZXZlcyAoQWRtaW4pIiwiaHR0cDovL3NjaGVtYX
MueG1sc29hcC5vcmcvd3MvMjAwNS8wNS9pZGVudGl0eS9jbGFpbXMvZW1haWxhZGRyZXNzIjoicm9vdEBzZW50aW5lbC5jbG91ZC5hcGkiLCJodHRwOi8vc2NoZW1hcy5taWNyb3NvZnQuY29tL3dzLzIwMDgvMDYvaWRlbnRpdHkvY2xhaW1zL
3JvbGUiOiJBZG1pbmlzdHJhdG9yIiwibmJmIjoxNjU4OTA2NDU5LCJleHAiOjE2NTg5MDcwNTksImlzcyI6Imh0dHBzOi8vbG9jYWxob3N0OjQ0MzgxIiwiYXVkIjoiaHR0cHM6Ly9sb2NhbGhvc3Q6NDQzODEifQ.smtJWyNCiHhJjKFs-F7PP
6MnXS0bJ0Lx-mEBbe-XN9w",
  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L21kZW50aXR5L2NsYW1tcy9uYW1lIjoiSmF2aWVyIE5pZXZlcyAoQWRtaW4pIiwiaHR0cDovL3N
jaGVtYXMueG1sc29hcC5vcmcvd3MvMjAwNS8wNS9pZGVudGl0eS9jbGFpbXMvZW1haWxhZGRyZXNzIjoicm9vdEBzZW50aW5lbC5jbG91ZC5hcGkiLCJodHRwOi8vc2NoZW1hcy5taWNyb3NvZnQuY29tL3dzLzIwMDgvMDYvaWRlbnRpdHkvY2
xhaW1zL3JvbGUiOiJBZG1pbmlzdHJhdG9yIiwibmJmIjoxNjU4OTA2NDU5LCJleHAiOjE2NTg5MTAwNTksImlzcyI6Imh0dHBzOi8vbG9jYWxob3N0OjQ0MzgxIiwiYXVkIjoiaHR0cHM6Ly9sb2NhbGhvc3Q6NDQzODEifQ.rNBi404xdeRvKt
0TW_hEqRDSk_cXgUPTfleajjVFbWE"
}
```

Response headers

```
content-length: 1148
content-type: application/json; charset=utf-8
date: Wed,27 Jul 2022 07:20:59 GMT
server: Microsoft-IIS/10.0
x-firefox-spdy: h2
x-powered-by: ASP.NET
```

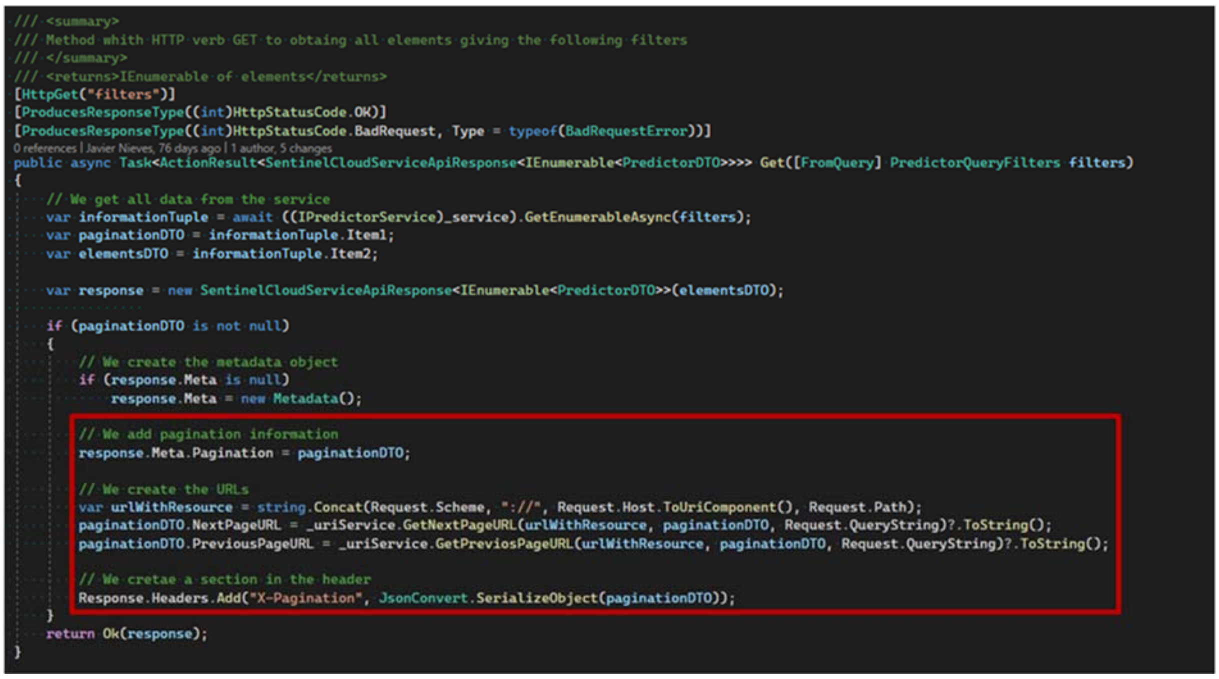
Responses

Code	Description	Links
200	Success	*No links*

Figure 17.- Login with token and refresh token for the user.

- Pagination

Some of the information returned from the service could be large. Thus, this feature has been implemented both in the response and in the meta-information data. Figure 18 illustrates an example of how it is done in REVaMP service.



```
/// <summary>
/// Method whith HTTP verb GET to obtaing all elements giving the following filters
/// </summary>
/// <returns>IEnumerable of elements</returns>
[HttpGet("filters")]
[ProducesResponseType((int)HttpStatusCode.OK)]
[ProducesResponseType((int)HttpStatusCode.BadRequest, Type = typeof(BadRequestError))]
0 references | Javier Nieves, 76 days ago | 1 author, 5 changes
public async Task<ActionResult<SentinelCloudServiceApiResponse<IEnumerable<PredictorDTO>>>> Get([FromQuery] PredictorQueryFilters filters)
{
    // We get all data from the service
    var informationTuple = await ((IPredictorService)_service).GetEnumerableAsync(filters);
    var paginationDTO = informationTuple.Item1;
    var elementsDTO = informationTuple.Item2;

    var response = new SentinelCloudServiceApiResponse<IEnumerable<PredictorDTO>>(elementsDTO);

    if (paginationDTO is not null)
    {
        // We create the metadata object
        if (response.Meta is null)
            response.Meta = new Metadata();

        // We add pagination information
        response.Meta.Pagination = paginationDTO;

        // We create the URLs
        var urlWithResource = string.Concat(Request.Scheme, "://", Request.Host.ToUriComponent(), Request.Path);
        paginationDTO.NextPageURL = _uriService.GetNextPageURL(urlWithResource, paginationDTO, Request.QueryString)?.ToString();
        paginationDTO.PreviousPageURL = _uriService.GetPreviosPageURL(urlWithResource, paginationDTO, Request.QueryString)?.ToString();

        // We cretae a section in the header
        Response.Headers.Add("X-Pagination", JsonConvert.SerializeObject(paginationDTO));
    }
    return Ok(response);
}
```

Figure 18.- Example of pagination creation and creating URLs for HATEOAS and Meta-information .

3.5.2. Diagrams

3.5.2.1. Components Diagram

This system is composed by 4 different components that work together.

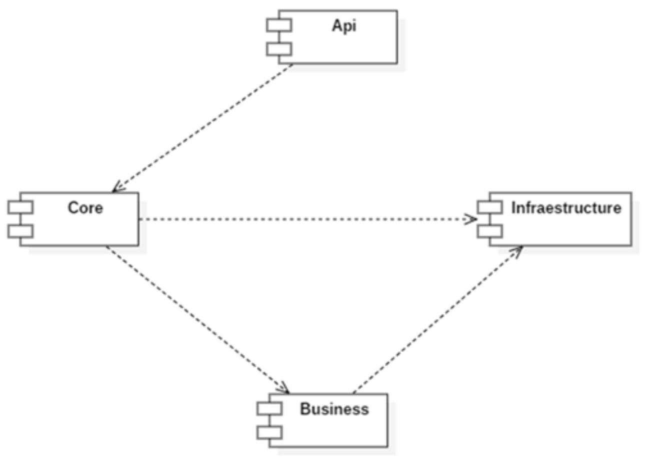


Figure 19.- General REVaMP service Components Diagram.

3.5.2.2. Package Diagrams

This project is complex and therefore a good distribution and organization of the code is necessary. Here is shown the package organization of it.

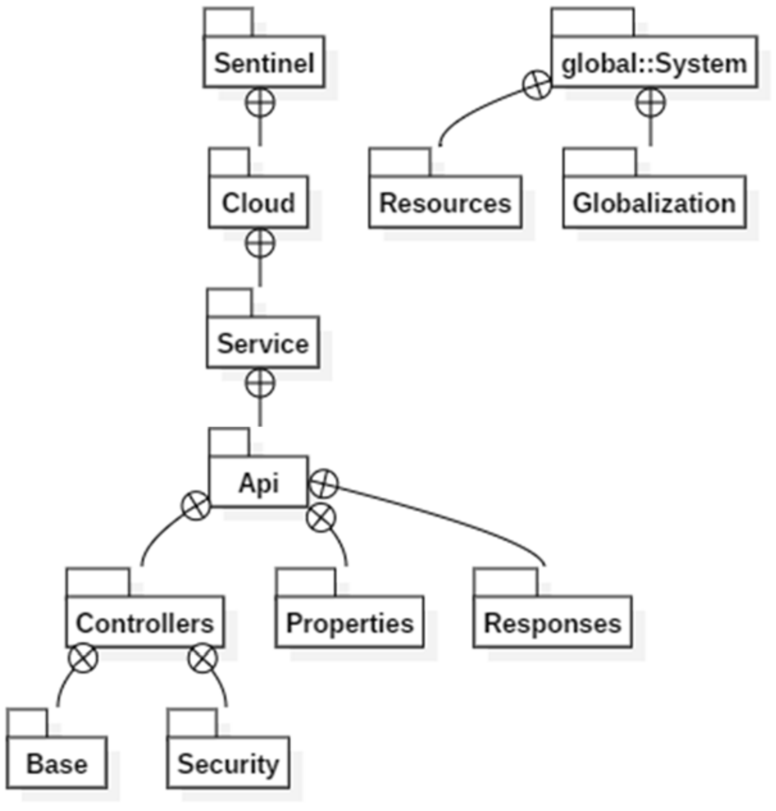


Figure 20.- API Package Diagram.

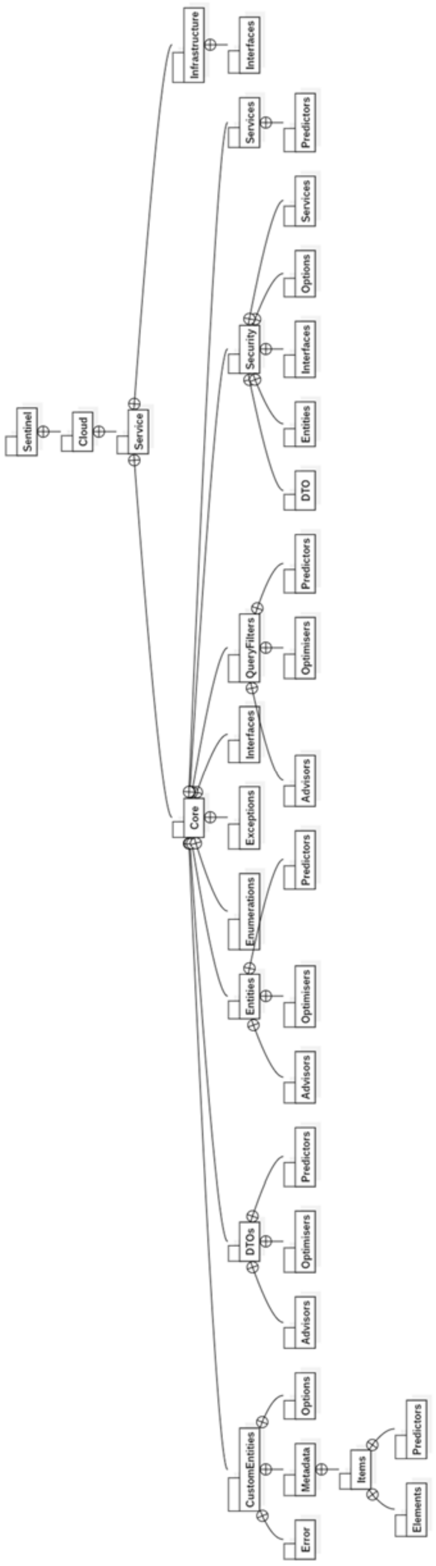


Figure 21.- CORE Package Diagram

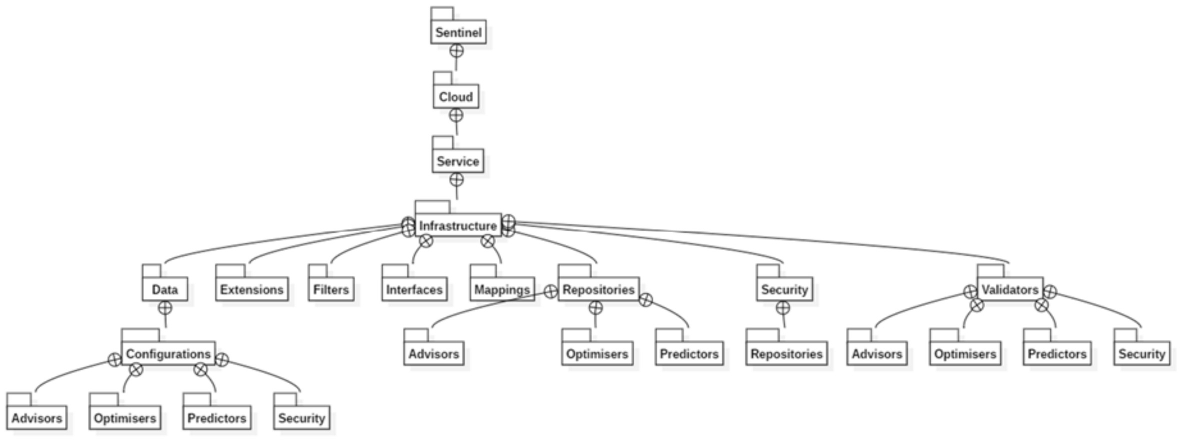


Figure 22.- INFRASTRUCTURE Package Diagram.

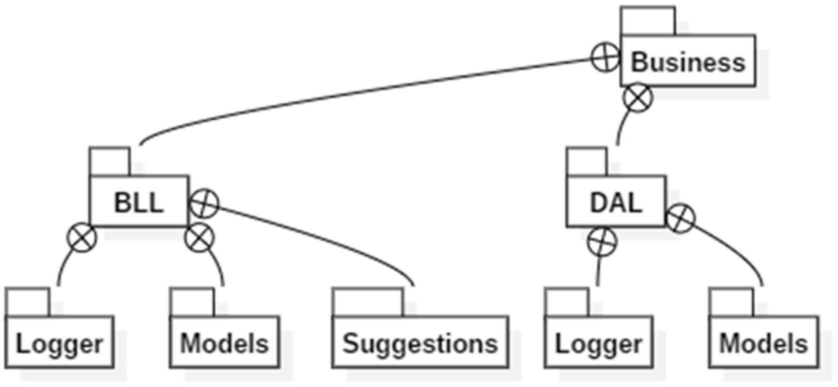


Figure 23.- BUSINESS Package Diagram.

3.5.2.3. Class Diagram by Component

Simplified class diagrams are shown below (does not include interfaces and other software artifacts) divided by components: Api, Core, Infrastructure and Business.

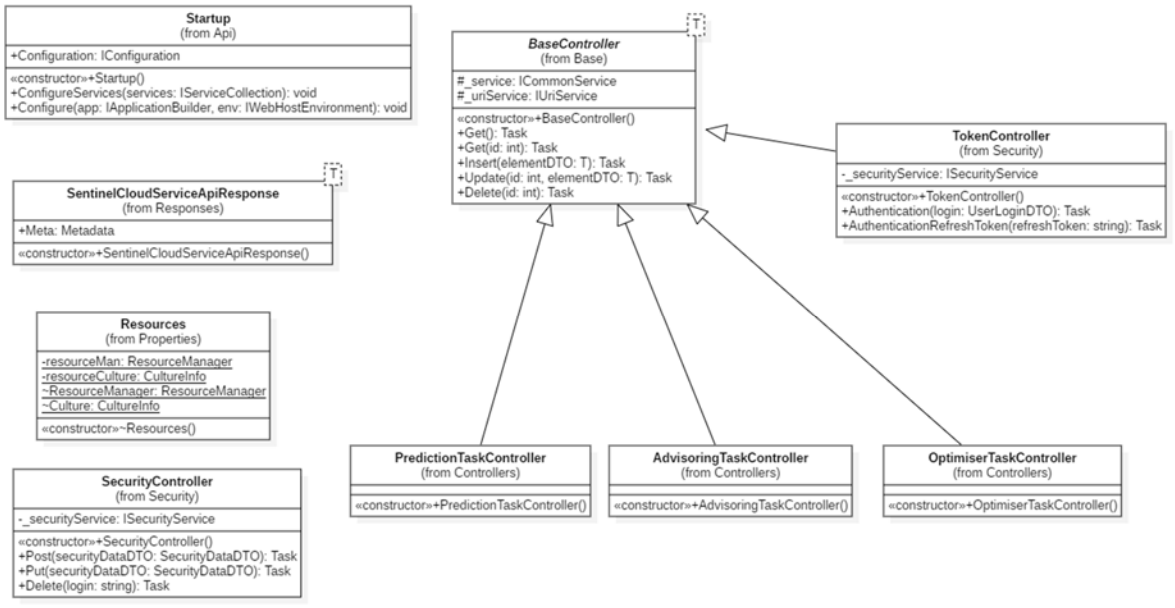

Startup
(from Api)
+Configuration: IConfiguration
«constructor»+Startup()
+ConfigureServices(services: IServiceCollection): void
+Configure(app: IApplicationBuilder, env: IWebHostEnvironment): void
T
SentinelCloudServiceApiResponse
(from Responses)
+Meta: Metadata
«constructor»+SentinelCloudServiceApiResponse()
Resources
(from Properties)
-resourceMan: ResourceManager
-resourceCulture: CultureInfo
~ResourceManager: ResourceManager
~Culture: CultureInfo
«constructor»~Resources()
SecurityController
(from Security)
-_securityService: ISecurityService
«constructor»+SecurityController()
+Post(securityDataDTO: SecurityDataDTO): Task
+Put(securityDataDTO: SecurityDataDTO): Task
+Delete(login: string): Task
T
BaseController
(from Base)
#_service: ICommonService
#_uriService: IUriService
«constructor»+BaseController()
+Get(): Task
+Get(id: int): Task
+Insert(elementDTO: T): Task
+Update(id: int, elementDTO: T): Task
+Delete(id: int): Task
TokenController
(from Security)
-_securityService: ISecurityService
«constructor»+TokenController()
+Authentication(login: UserLoginDTO): Task
+AuthenticationRefreshToken(refreshToken: string): Task
PredictionTaskController
(from Controllers)
«constructor»+PredictionTaskController()
AdvisoringTaskController
(from Controllers)
«constructor»+AdvisoringTaskController()
OptimiserTaskController
(from Controllers)
«constructor»+OptimiserTaskController()

Figure 24.- API Class Diagram Overview.

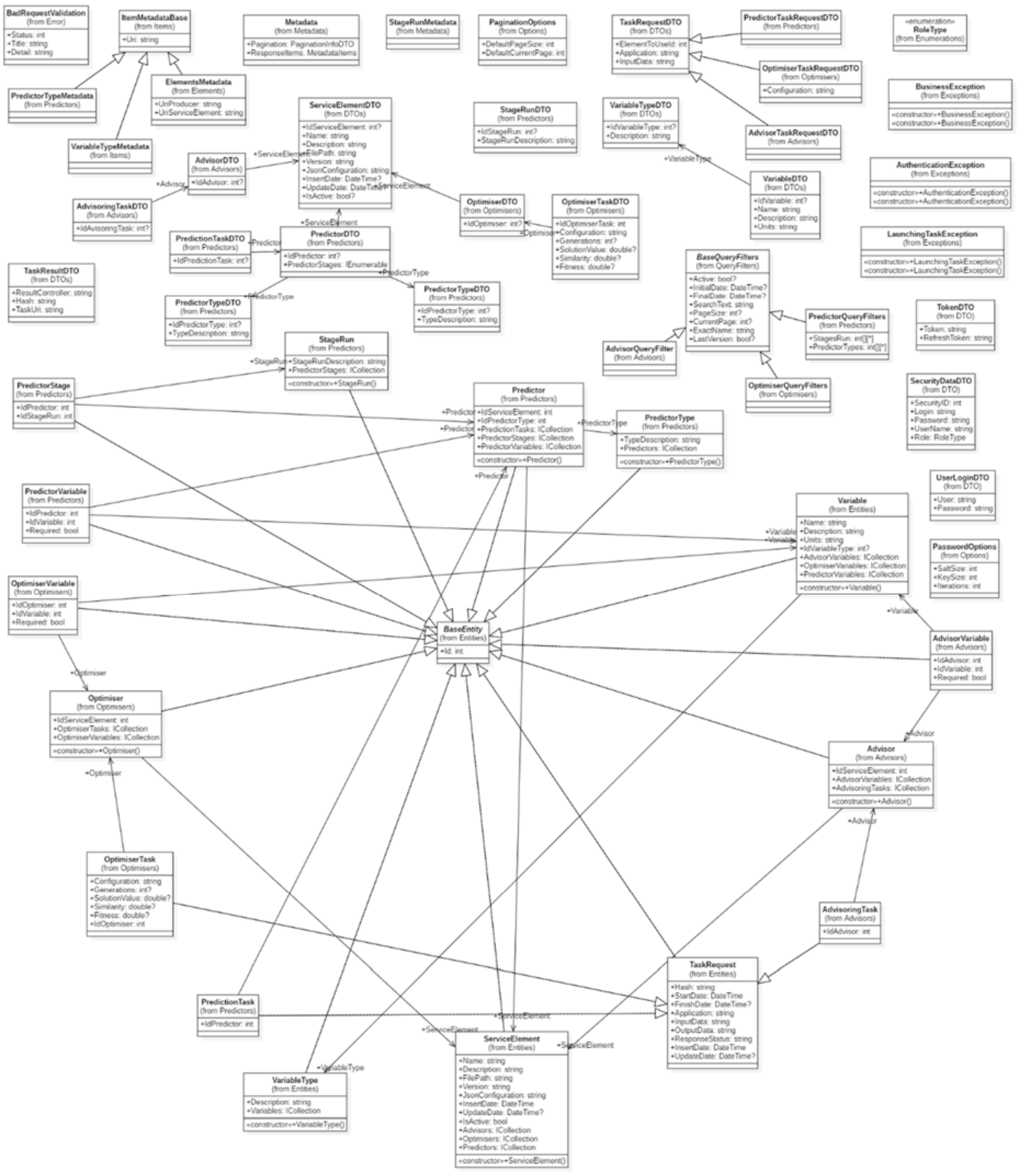


Figure 25.- CORE Class Diagram.

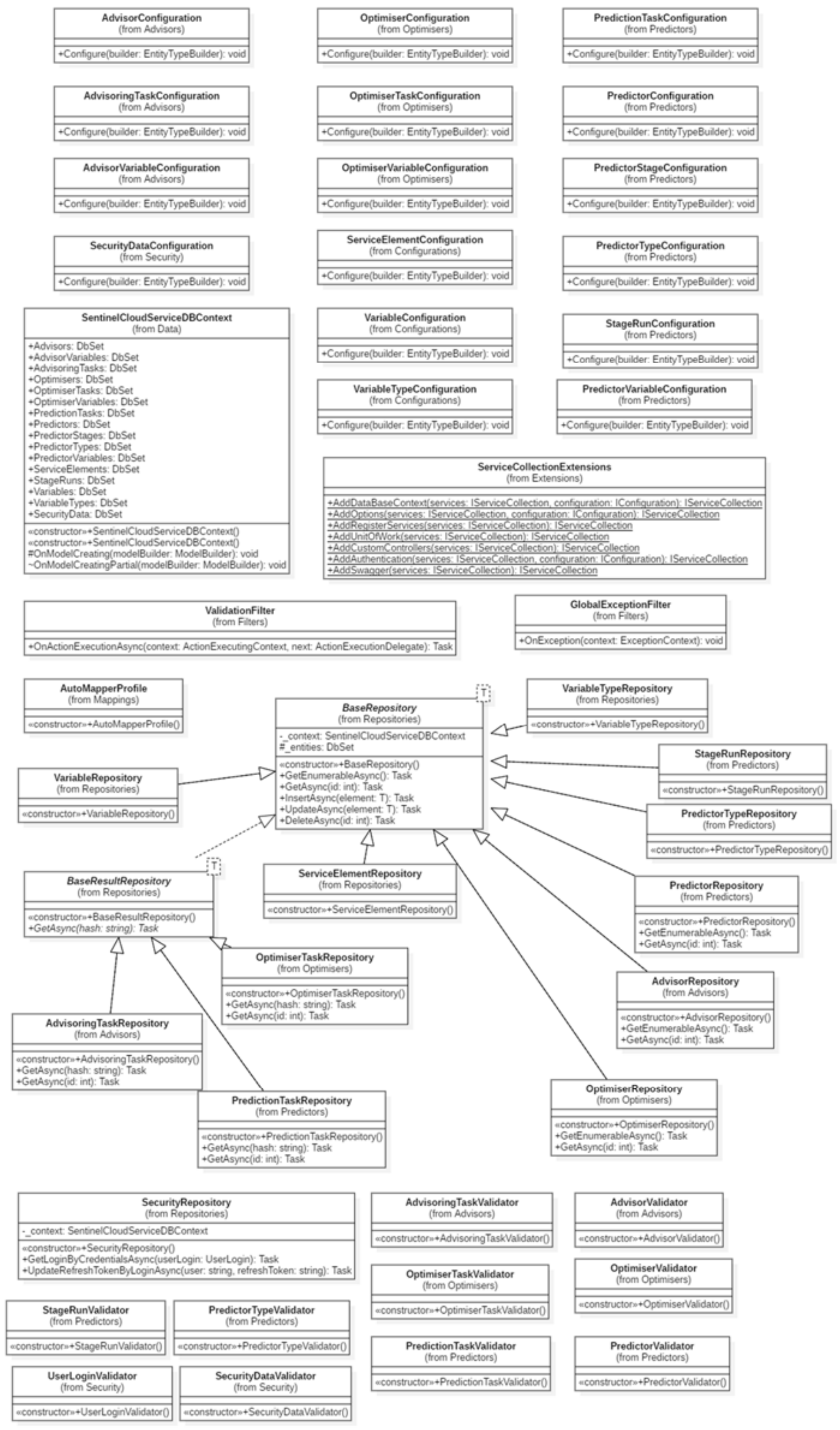

AdvisorConfiguration (from Advisors)
+Configure(builder: EntityTypeBuilder): void
OptimiserConfiguration (from Optimisers)
+Configure(builder: EntityTypeBuilder): void
PredictionTaskConfiguration (from Predictors)
+Configure(builder: EntityTypeBuilder): void
AdvisoringTaskConfiguration (from Advisors)
+Configure(builder: EntityTypeBuilder): void
OptimiserTaskConfiguration (from Optimisers)
+Configure(builder: EntityTypeBuilder): void
PredictorConfiguration (from Predictors)
+Configure(builder: EntityTypeBuilder): void
AdvisorVariableConfiguration (from Advisors)
+Configure(builder: EntityTypeBuilder): void
OptimiserVariableConfiguration (from Optimisers)
+Configure(builder: EntityTypeBuilder): void
PredictorStageConfiguration (from Predictors)
+Configure(builder: EntityTypeBuilder): void
SecurityDataConfiguration (from Security)
+Configure(builder: EntityTypeBuilder): void
ServiceElementConfiguration (from Configurations)
+Configure(builder: EntityTypeBuilder): void
PredictorTypeConfiguration (from Predictors)
+Configure(builder: EntityTypeBuilder): void
SentinelCloudServiceDBContext (from Data)
+Advisors: DbSet
+AdvisorVariables: DbSet
+AdvisoringTasks: DbSet
+Optimisers: DbSet
+OptimiserTasks: DbSet
+OptimiserVariables: DbSet
+PredictionTasks: DbSet
+Predictors: DbSet
+PredictorStages: DbSet
+PredictorTypes: DbSet
+PredictorVariables: DbSet
+ServiceElements: DbSet
+StageRuns: DbSet
+Variables: DbSet
+VariableTypes: DbSet
+SecurityData: DbSet
«constructor»+SentinelCloudServiceDBContext()
«constructor»+SentinelCloudServiceDBContext()
#OnModelCreating(modelBuilder: ModelBuilder): void
~OnModelCreatingPartial(modelBuilder: ModelBuilder): void
VariableConfiguration (from Configurations)
+Configure(builder: EntityTypeBuilder): void
StageRunConfiguration (from Predictors)
+Configure(builder: EntityTypeBuilder): void
VariableTypeConfiguration (from Configurations)
+Configure(builder: EntityTypeBuilder): void
PredictorVariableConfiguration (from Predictors)
+Configure(builder: EntityTypeBuilder): void
ServiceCollectionExtensions (from Extensions)
+AddDataBaseContext(services: IServiceCollection, configuration: IConfiguration): IServiceCollection
+AddOptions(services: IServiceCollection, configuration: IConfiguration): IServiceCollection
+AddRegisterServices(services: IServiceCollection): IServiceCollection
+AddUnitOfWork(services: IServiceCollection): IServiceCollection
+AddCustomControllers(services: IServiceCollection): IServiceCollection
+AddAuthentication(services: IServiceCollection, configuration: IConfiguration): IServiceCollection
+AddSwagger(services: IServiceCollection): IServiceCollection
ValidationFilter (from Filters)
+OnActionExecutionAsync(context: ActionExecutingContext, next: ActionExecutionDelegate): Task
GlobalExceptionFilter (from Filters)
+OnException(context: ExceptionContext): void
AutoMapperProfile (from Mappings)
«constructor»+AutoMapperProfile()
T
BaseRepository (from Repositories)
-_context: SentinelCloudServiceDBContext
#_entities: DbSet
«constructor»+BaseRepository()
+GetEnumerableAsync(): Task
+GetAsync(id: int): Task
+InsertAsync(element: T): Task
+UpdateAsync(element: T): Task
+DeleteAsync(id: int): Task
VariableTypeRepository (from Repositories)
«constructor»+VariableTypeRepository()
StageRunRepository (from Predictors)
«constructor»+StageRunRepository()
VariableRepository (from Repositories)
«constructor»+VariableRepository()
PredictorTypeRepository (from Predictors)
«constructor»+PredictorTypeRepository()
T
BaseResultRepository (from Repositories)
«constructor»+BaseResultRepository()
+GetAsync(hash: string): Task
ServiceElementRepository (from Repositories)
«constructor»+ServiceElementRepository()
PredictorRepository (from Predictors)
«constructor»+PredictorRepository()
+GetEnumerableAsync(): Task
+GetAsync(id: int): Task
OptimiserTaskRepository (from Optimisers)
«constructor»+OptimiserTaskRepository()
+GetAsync(hash: string): Task
+GetAsync(id: int): Task
AdvisorRepository (from Advisors)
«constructor»+AdvisorRepository()
+GetEnumerableAsync(): Task
+GetAsync(id: int): Task
AdvisoringTaskRepository (from Advisors)
«constructor»+AdvisoringTaskRepository()
+GetAsync(hash: string): Task
+GetAsync(id: int): Task
OptimiserRepository (from Optimisers)
«constructor»+OptimiserRepository()
+GetEnumerableAsync(): Task
+GetAsync(id: int): Task
PredictionTaskRepository (from Predictors)
«constructor»+PredictionTaskRepository()
+GetAsync(hash: string): Task
+GetAsync(id: int): Task
SecurityRepository (from Repositories)
-_context: SentinelCloudServiceDBContext
«constructor»+SecurityRepository()
+GetLoginByCredentialsAsync(userLogin: UserLogin): Task
+UpdateRefreshTokenByLoginAsync(user: string, refreshToken: string): Task
AdvisoringTaskValidator (from Advisors)
«constructor»+AdvisoringTaskValidator()
AdvisorValidator (from Advisors)
«constructor»+AdvisorValidator()
OptimiserTaskValidator (from Optimisers)
«constructor»+OptimiserTaskValidator()
OptimiserValidator (from Optimisers)
«constructor»+OptimiserValidator()
StageRunValidator (from Predictors)
«constructor»+StageRunValidator()
PredictorTypeValidator (from Predictors)
«constructor»+PredictorTypeValidator()
PredictionTaskValidator (from Predictors)
«constructor»+PredictionTaskValidator()
PredictorValidator (from Predictors)
«constructor»+PredictorValidator()
UserLoginValidator (from Security)
«constructor»+UserLoginValidator()
SecurityDataValidator (from Security)
«constructor»+SecurityDataValidator()

Figure 26.- INFRASTRUCTURE Class Diagram.

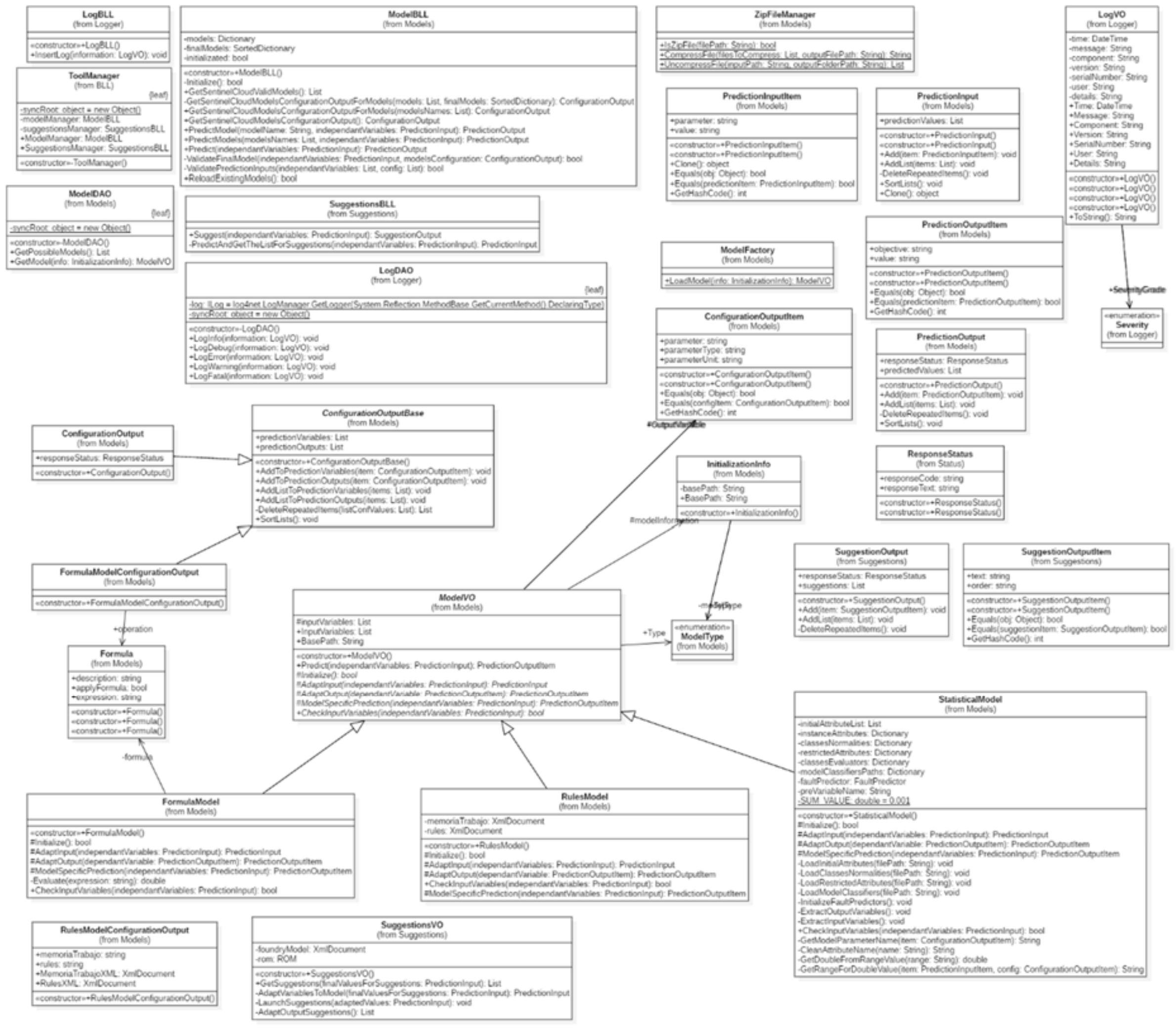


Figure 27.- Business Class Diagram.

3.6. Storage

The service has two different storage systems that will be briefly described below.

3.6.1. Data Base

- For the proper functioning of the service, the storage of different information is managed in its own database. Specifically, within this repository it will be registered:
  - All configurations of each of the models, predictors and optimizers.
  - All the history of tasks that have been generated.
  - Obtained results.
  - Versions of models, perdictors and optimizers, as well as their associated information.

The following figure shows the Entity-Relationship diagram obtained as a result of the design process for this service.

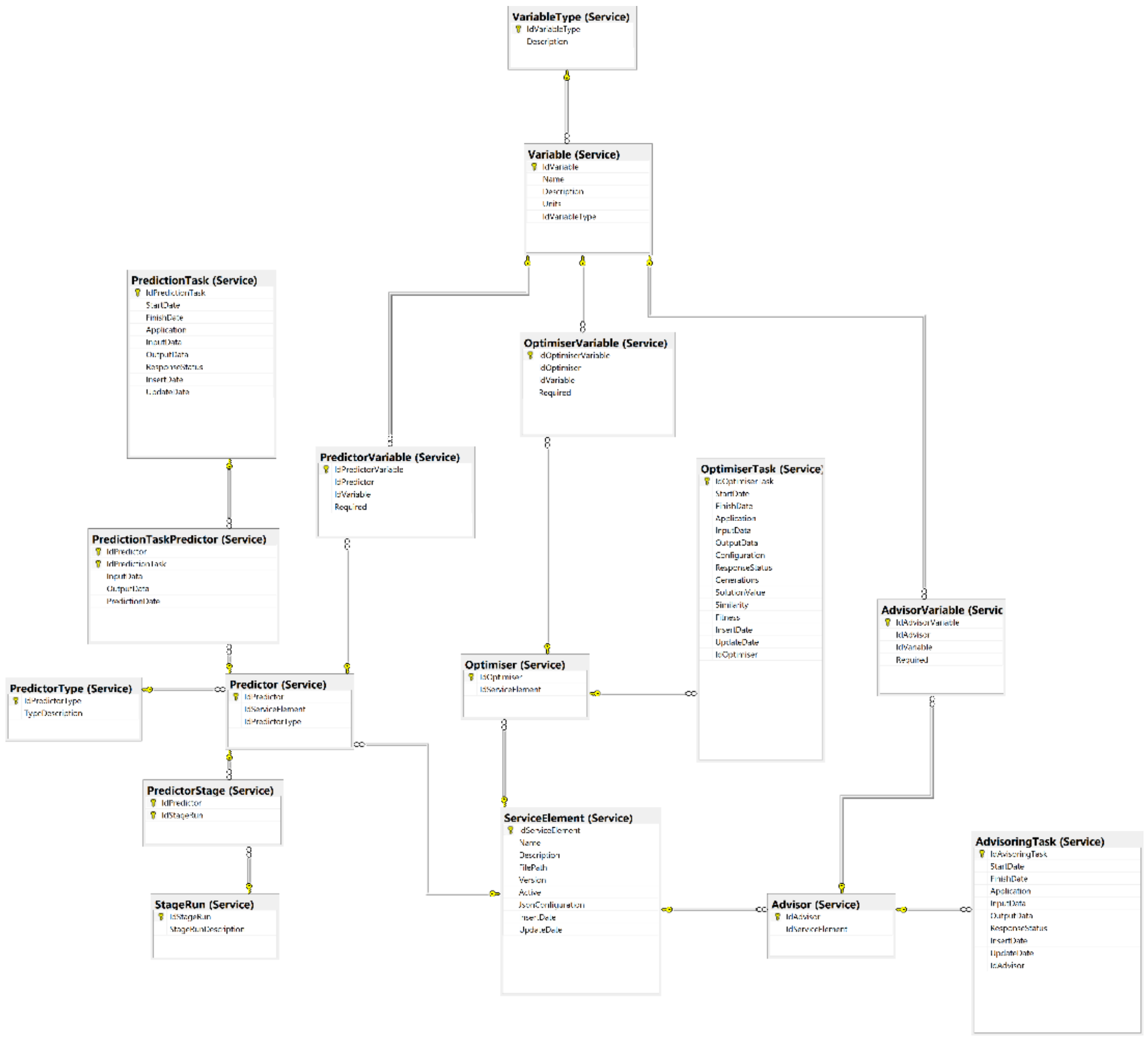


Figure 28.- Service Data Base.

3.6.2. File System

Our REVaMP service needs to keep a storage of each of the elements and their versions (called Service Elements). That is the reason why a file storage has been defined. Specifically, this storage divides the elements into different folders. Internally, each of them will have grouped all developments of the Service Element with the same name and, subsequently, its versions. They are all managed in the same way, thus, the storage structure will be the same. However, the last level will be dependent on each of them and their specific operation.

The following figure shows an overview of how this store could be generated for multiple elements.

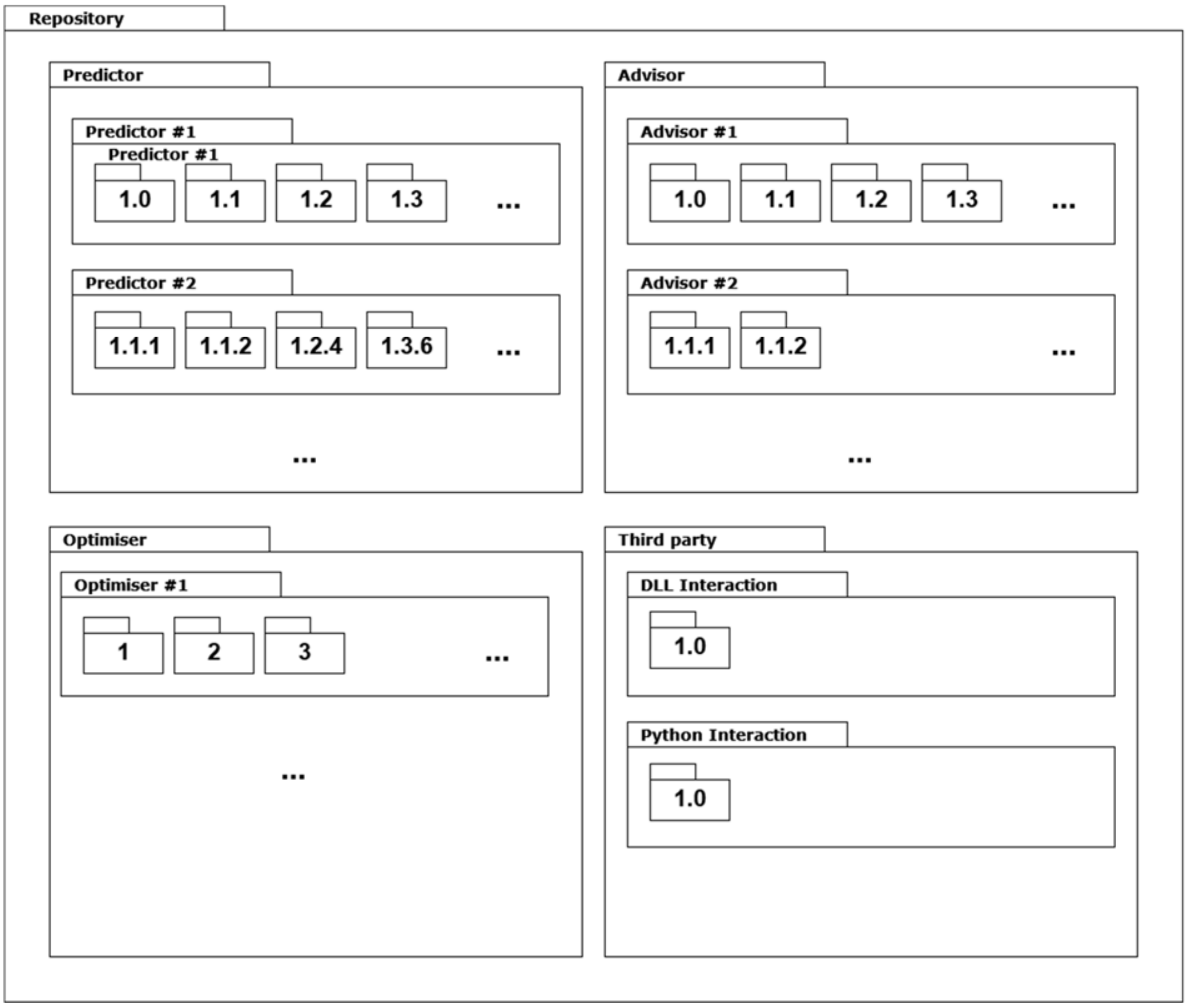


Figure 29.- Diagram simulating the storage for models into the server.

3.7. Server Entities Management

For models, optimizers and advisors a polymorphic operation is performed. Specifically, from a programmatic point of view, they are all managed in the same way. However, given the design with the class hierarchy and the factory pattern for creating them, each one applies its specific behavior.

Nevertheless, all of them are managed through compressed files that have their information inside. These files are the ones that are loaded into the service, validated, extracted and configured for all storage (displayed in section 3.6.2).

In the following sections we are going to show an example of each kind of models.

3.7.1. General

Every compressed file has a common part that carries its configuration. Specifically, it includes the type of element, the variables and its additional configuration.

```
{
  "type": "OPTIMISER",
  "name": "Chemical Composition Optimiser Model",
  "description": "Optimiser model to determine if the chemical composition (MG and Cu) is ok.",
  "version": "1.0.0",
  "predictionOutputs": [
    {
      "parameter": "Evaluated_Chemical_Composition",
      "parameterType": "String",
      "parameterUnit": "Label"
    }
  ],
  "predictionVariables": [
    {
      "parameter": "MG",
      "parameterType": "Double",
      "parameterUnit": "Percentage"
    },
        {
      "parameter": "MGMAX",
      "parameterType": "Double",
      "parameterUnit": "Percentage"
    },
        {
      "parameter": "MGMIN",
      "parameterType": "Double",
      "parameterUnit": "Percentage"
    },
    {
      "parameter": "CU",
      "parameterType": "Double",
      "parameterUnit": "Percentage"
    },
        {
      "parameter": "CUMAX",
      "parameterType": "Double",
      "parameterUnit": "Percentage"
    },
        {
      "parameter": "CUMIN",
      "parameterType": "Double",
      "parameterUnit": "Percentage"
    }
  ],
}
```

Figure 30.- Service Element configuration file

As we have mentioned, each of the elements have different files or information:

- **Rules and Advisors:** Both are based on Rule Based Expert System. In this way, both have same files.
  - **Working Memory.** An auxiliary temporary memory that stores the process data, the initial data of the problem and the intermediate results obtained throughout the consultation and resolution process. Through it we can know not only the current state of the system, but also how it was reached. Here, the

service enters the information it has about the current problem and the system matches this information with the available knowledge to deduce new facts.

```
<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<foundry>
        <input>
                <!-- Related to Metallurgical Quaility -->
                <MG value= \"\"/>
                <MGMAX value= \"\"/>
                <MGMIN value= \"\"/>
                <CU value= \"\"/>
                <CUMAX value= \"\"/>
                <CUMIN value= \"\"/>
        </input>
        <deduced>
                <!-- Related to Metallurgical Quality -->
                <MGMAX_ADJ value= \"-1\"/>
                <MGMIN_ADJ value= \"-1\"/>
                <CUMAX_ADJ value= \"-1\"/>
                <CUMIN_ADJ value= \"-1\"/>
                <CU_RESULT value= \" \"/>
                <MG_RESULT value= \" \"/>
        </deduced>
        <results>
                <Final_Result value=\" \"/>
        </results>
</foundry>
```

Figure 31.- Working Memory definition example for chemical composition.

- **Rules and facts.**

```
<?xml version=\"1.0\" encoding=\"utf-8\" ?>
<RuleEngine>
  <Rules>
    <!-- Begin utility rules -->
    <Rule id=\"AdjustMgMax\" desc=\"Utility rule to calculate the new Mg Max value\">
      <Condition><![CDATA[ FACT(mgMaxAdj) == FACT(EmptyNum) ]]></Condition>
      <Actions>
        <Action factId=\"mgMaxAdj\">
          <Expression><![CDATA[ FACT(mgMax) + FACT(factorMgLimSup) ]]></Expression>
        </Action>
      </Actions>
    </Rule>
    <Rule id=\"AdjustCuMax\" desc=\"Utility rule to calculate the new Cu Max value\">
      <Condition><![CDATA[ FACT(cuMaxAdj) == FACT(EmptyNum) ]]></Condition>
      <Actions>
        <Action factId=\"cuMaxAdj\">
          <Expression><![CDATA[ FACT(cuMax) + FACT(factorCuLimSup) ]]></Expression>
        </Action>
      </Actions>
    </Rule>
    <Rule id=\"CuOkCalculation\" desc=\"Cu bigger equal than CuMinAdj and less equal than
CuMaxAdj\">
      <Condition><![CDATA[ FACT(cuDeduced) == FACT(Empty) AND FACT(cu) >= FACT(cuMinAdj)
AND FACT(cu) <= FACT(cuMaxAdj) ]]></Condition>
      <Actions>
        <Action factId=\"cuDeduced\">
          <Expression><![CDATA[ FACT(okCu) ]]></Expression>
        </Action>
      </Actions>
    </Rule>
    ..
    <!-- End utility rules -->

    <!-- Begin calculation rules -->
    <Rule id=\"resultOK\" desc=\"Everything is OK in the Chemical Composition\">
      <Condition><![CDATA[  FACT(ResultValue)  ==  FACT(Empty)  AND  FACT(cuDeduced)  ==
FACT(okCu) AND FACT(mgDeduced) == FACT(okMG)]]></Condition>
      <Actions>
        <Action factId=\"ResultValue\">
```

```
          <Expression><![CDATA[ FACT(ok) ]]></Expression>
        </Action>
      </Actions>
    </Rule>
    <Rule  id=\"resultErrorCU\"  desc=\"CU  evaluation  was  wrong  in  the  Chemical
Composition\">
      <Condition><![CDATA[  FACT(ResultValue)  ==  FACT(Empty)  AND  FACT(cuDeduced)  ==
FACT(errCu) AND FACT(mgDeduced) == FACT(okMG)]]></Condition>
      <Actions>
        <Action factId=\"ResultValue\">
          <Expression><![CDATA[ FACT(errCu) ]]></Expression>
        </Action>
      </Actions>
    </Rule>
    …
    <Rule id=\"resultErrorCUMgAlto\" desc=\"CU evaluation was wrong and Mg was high in the
Chemical Composition\">
      <Condition><![CDATA[  FACT(ResultValue)  ==  FACT(Empty)  AND  FACT(cuDeduced)  ==
FACT(errCu) AND FACT(mgDeduced) == FACT(mgAlto)]]></Condition>
      <Actions>
        <Action factId=\"ResultValue\">
          <Expression><![CDATA[ FACT(errorCuMgAlto) ]]></Expression>
        </Action>
      </Actions>
    </Rule>
    <!-- End calculation rules -->

  </Rules>
  <Facts>
    <!-- Begin Generic facts -->
    <Fact id=\"Empty\" desc=\"Transaction type\" type=\"string\">
      <xpath><![CDATA[' ']]></xpath>
    </Fact>
    <Fact id=\"EmptyNum\" desc=\"Transaction type\" type=\"double\">
      <xpath><![CDATA[-1]]></xpath>
    </Fact>
    <Fact id=\"okCu\" desc=\"Transaction type\" type=\"string\">
      <xpath><![CDATA['OK_CU']]></xpath>
    </Fact>
    <Fact id=\"errCu\" desc=\"Transaction type\" type=\"string\">
      <xpath><![CDATA['ERR_CU']]></xpath>
    </Fact>
    <Fact id=\"okMG\" desc=\"Transaction type\" type=\"string\">
      <xpath><![CDATA['OK_MG']]></xpath>
    </Fact>
    …
    <!-- End Generic facts -->

    <!-- Begin Results facts -->
    <Fact id=\"ResultValue\" desc=\"Final result value\" type=\"string\">
      <xpath><![CDATA[ foundry/results/Final_Result/@value ]]></xpath>
    </Fact>
    <!-- End Results facts -->

    <!-- Begin Input facts -->
    <Fact id=\"mg\" desc=\"Mg perecentage\" type=\"double\">
      <xpath><![CDATA[foundry/input/MG/@value]]></xpath>
    </Fact>
    <Fact id=\"mgMax\" desc=\"Mg maximum precentaje allowed\" type=\"double\">
      <xpath><![CDATA[foundry/input/MGMAX/@value]]></xpath>
    </Fact>
    <Fact id=\"mgMin\" desc=\"Mg minimum precentaje allowed\" type=\"double\">
      <xpath><![CDATA[foundry/input/MGMIN/@value]]></xpath>
    </Fact>
    …
    <!-- End Input facts -->

    <!-- Begin Deduced facts -->

    <Fact id=\"mgMaxAdj\" desc=\"Mg maximum precentaje adjusted using the input factor\"
type=\"double\">
      <xpath><![CDATA[foundry/deduced/MGMAX_ADJ/@value]]></xpath>
    </Fact>
    <Fact id=\"mgMinAdj\" desc=\"Mg minimum precentaje adjusted using the input factor\"
type=\"double\">
```

```
      <xpath><![CDATA[foundry/deduced/MGMIN_ADJ/@value]]></xpath>
    </Fact>
    …
    <!-- End Deduced facts -->

  </Facts>
</RuleEngine>
```

Figure 32.- Rules and facts overview (reduced) for manageing chemical composition validation.

- **Formula:** In this model we have a file storaging the formula. The formula is in text format but, later, the service interprets it and calculates the values.

```
{
  "operation":
    {
        "applyFormula": true,
        "expression": "FLOOR( 0.1135 * DURATION_MINS - 11.7734 * MIN_ROTATING_SPEED - 16.4631 *
MAX_ROTATING_SPEED + 10.6025 * AVG_ROTATING_SPEED + 0.0391 * PRCT_CLOSED_DOOR + 0.1987 *
PRCT_BURNER_ON - 0.1600 * PRCT_ROTATION_RUNNING + 0.0829 * PRCT_VERTICAL_POSITION + 24.2863)"
    }
}
```

Figure 33.- File definition for formula, example of Gas Consumption.

- **Statístical models:** In both cases, classification and regresions, a binary file with the created moldes is included. Later, service will load the binary model. In this case, we can manage Weka and ML.Net models. As these models are im binary formats and are not readeable, we do not include an example.

- **Optimiser configuration:** This setting allows you to define how all optimization will work. All information is included in a file inside the generated package. The different elements that can be configured are the following:
    - *Genes.* Each of the genes can be configured one by one. They can be assigned with its maximum/minimum possible values, the mutation ratio that it will have and whether that gene will be able to be modified by the algorithm.
    - *Algorithm.* The execution will be modified according to the configuration. The algorithm can work based on the number of iterations or looking for a value higher than the fitness base value defined as limit for a solution. Both of them can also be combined and/or use a maximum value of iterations to ensure that the algorithm finishes. Likewise, the configuration allows us to activate elitism (in other words, keep the best of all solutions between generations) and the crossover factor to be used. Finally, you can also indicate the number of simultaneous solutions to be evaluated simultaneously by the algorithm.

```
{
  "UseGenerationNumber": true,
  "MaximunNumberOfGenerations": 3000,
  "UseEvaluationLimit": true,
  "EvaluationLimit": 0.9996,
  "UseElitism": true,
  "CrossoverRatio": 0.8,
  "Population": 40,
  "Genome": {
    "Genes": [
      {
        "Typology": " CONTINUOS ",
        "Name": "DURATION_MINS",
        "MutationRatio": 0.10,
        "CanBeModified": true,
        "MinValue": 5.0,
        "MaxValue": 30.0,
        "Labels": null
      },
      {
        "Typology": "CONTINUOS",
        "Name": "MIN_ROTATING_SPEED",
        "MutationRatio": 0.05,
        "CanBeModified": true,
        "MinValue": 5.00,
        "MaxValue": 10.00,
        "Labels": null
      },
      {
        "Typology": " CONTINUOS ",
        "Name": "MAX_ROTATING_SPEED",
        "MutationRatio": 0.05,
        "CanBeModified": false,
        "MinValue": 5.0,
        "MaxValue": 10.0,
        "Labels": null
        ]
      },{
        "Typology": "CONTINUOS",
        "Name": "AVG_ROTATING_SPEED",
        "MutationRatio": 0.05,
        "CanBeModified": false,
        "MinValue": 5.00,
        "MaxValue": 10.00,
        "Labels": null
      },
      …
    ]
  }
}
```

Figure 34.- Configuration overview (reduced) for improve gas consumption - Example.

3.7.2. Third Party Integration

The developed service tries to serve both the interests of the REFIAL use case and those of the rest of the project members (steel and lead use cases). That is why it is intended to provide the models developed by third parties through the same way of interaction. For this purpose, it is necessary to integrate models and generate the philosophy and basic development that allows it to be done. Taking into account the current third party developments, this service has been adapted in the following way:

3.7.2.1. External DLLs

Within the models generated by the partners of the REVaMP project we have the use case of steel (BFI). Specifically, the developed model has been generated as a DLL programmed in C++ that performs its tasks. In this way, to carry out the integration and be able to use it, the following starting information is needed (final definition pending that will be done during WP5):

- JSON about input/output variables. This will be included as the Service Element configuration data.
- Entry point to call the DLL. This entry point will be included in a JSON with the configuration of the model. This JSON is similar to the aforementioned JSON in the optimization process.

To be able to call this function, our service makes use of an advanced approach that allows us to load external developments. Thus, this kind of models will use Reflection in its specific way of working.

The specific code that is included in the models is the following one. First of all we need a mechanism that allows us to call the DLL. This is provided by kernel32.dll with LoadLibrary function. This source code prepare the call in C#.

```
        [DllImport("kernel32.dll", EntryPoint = "LoadLibrary")]
        static extern int LoadLibrary(
            [MarshalAs(UnmanagedType.LPStr)] string lpLibFileName);

        [DllImport("kernel32.dll", EntryPoint = "GetProcAddress")]
        static extern IntPtr GetProcAddress( int hModule,
            [MarshalAs(UnmanagedType.LPStr)] string lpProcName);

        [DllImport("kernel32.dll", EntryPoint = "FreeLibrary")]
        static extern bool FreeLibrary(int hModule);
```

Figure 35.- Source Code to use kernell32.dll to load dynamically external DLLs.

The *LoadLibrary* function loads an unmanaged file, the *GetProcAddress* function gets the function pointer of an unmanaged file and the *FreeLibrary* function frees the unmanaged file. Now I can get the function pointer in a.dll very easily:

```
int hModule = LoadLibrary($"{Path.Combine(repositoryPath, thirdPartyPath,
version, dllPath)}");
if (hModule == 0) return false;
IntPtr intPtr = GetProcAddress(hModule, entryPointFunction);
```

Figure 36.- Getting the function pointer for C++ dll.

And when we want to free the *dll* file:

```
FreeLibrary(hModule);
```

Figure 37.- Free loaded DLL.

For making the communication we need to define a delegate to translate the call to a function pointer.

```
public static Delegate GetDelegateForFunctionPointer (
      IntPtr ptr,
      Type t
)

delegate object ExternalFunction(object b);

ExternalFunction function =
(ExternalFunction)Marshal.GetDelegateForFunctionPointer(intPtr,
typeof(ExternalFunction));
```

Figure 38.- Source code to translate the Pointer to a Delegate Function in C#.

Finally, when all this work is done, we can call the function.

```
var results = function(new double[3.0, 4.8, 6.8, 3.3]);
```

Figure 39.- Calling the external function in C++ dll.

Please, note that here we have defined the parameter and return data as object, but can be specifically defined for each model.

3.7.2.2. External Services in Python

Regarding other third party models, such us aluminum use case (EURECAT), specifically, the developed model has been generated as a Python software (the calculation part). In this way, to carry out the integration and be able to use it, the following starting information is needed (final definition pending that will be done during WP6):

- JSON about input/output variables. This will be included as the Service Element configuration data.
- Entry point to call the Python function. This entry point will be included in a JSON with the configuration of the model. This JSON is similar to the aforementioned JSON in the optimization process. Here we will have the entry point call.

In this case, we are going to launch the external model as a process. So, we can launch the function, pass the needed parameters and get the results to be processed. This is the following source code:

```
ProcessStartInfo start = new ProcessStartInfo();
start.FileName = $"{Path.Combine(repositoryPath, thirdPartyPath, version,
pythonPath)}"
start.Arguments = string.Format("{0} {1}", var1, var2);
start.UseShellExecute = false
start.CreateNoWindow = true
start.RedirectStandardOutput = true
start.RedirectStandardError = true;
using (Process process = Process.Start(start))
{
    using (StreamReader reader = process.StandardOutput)
    {
        string stderr = process.StandardError.ReadToEnd();
        string result = reader.ReadToEnd();
        return result;
    }
}
```

Figure 40.- Launching Python module from C#.

This code can be adapted to launch an internal function of a Python module. In order to achieve this goal, the arguments must be changed to load modules.

In the REVaMP Aluminium use case we have the following models and the way of launching are listed below:

- Model Main Change Mix Model:
  - *Entry Point:* __main_optimization__.py
  - *Input Variables:* optimization_step == 'principalOptimization' and Weight (kg)-integer for each alloy, Security coef (min)-float, Security coef (max)-float of the chemical composition.
  - *Output Variables:* 3 CSV file to be transformed, one with the weight in spoonfuls of the materials, another with the weight in kg and another with the composition.
- Alloy Adjusting Model:
  - *Entry Point:* __main_optimization__.py
  - *Input Variables:* variable optimization_step == '_secondary_optimization' and alloy number (str), Security coef (min)-float, Security coef (max)-float of the chemical compostion, Waste(kg) (float), csv file with the composition of the furnace (path), Cost reheat (float) , Mg fading cost (float), Temperature (float), Time (list offloat values, max 6)
  - Output Variables: 3 CSV file to be transformed, one with the weight in spoonfuls of the materials, another with the weight in kg and another with the composition.
- Process Control Model:
  - Entry Point: __main_anomaly__.py
  - Input Variables: batch name (str)
  - Output Variables: 4 str with subprocess info.
- Mass Balance:
  - Entry Point: __main_composition.py
  - Input Variables: kg (int) for each raw material
  - Output Variables: Final chemical composition (str)

The developed service does not handle csv files or tables as results. Thus, we have created a mapping class that serialize all this information into a JSON format. This format is the standard de facto used in our communications and also to be stored in our database in our prediction, optimization and advisoring tasks.

3.7.2.3. Docker Integration

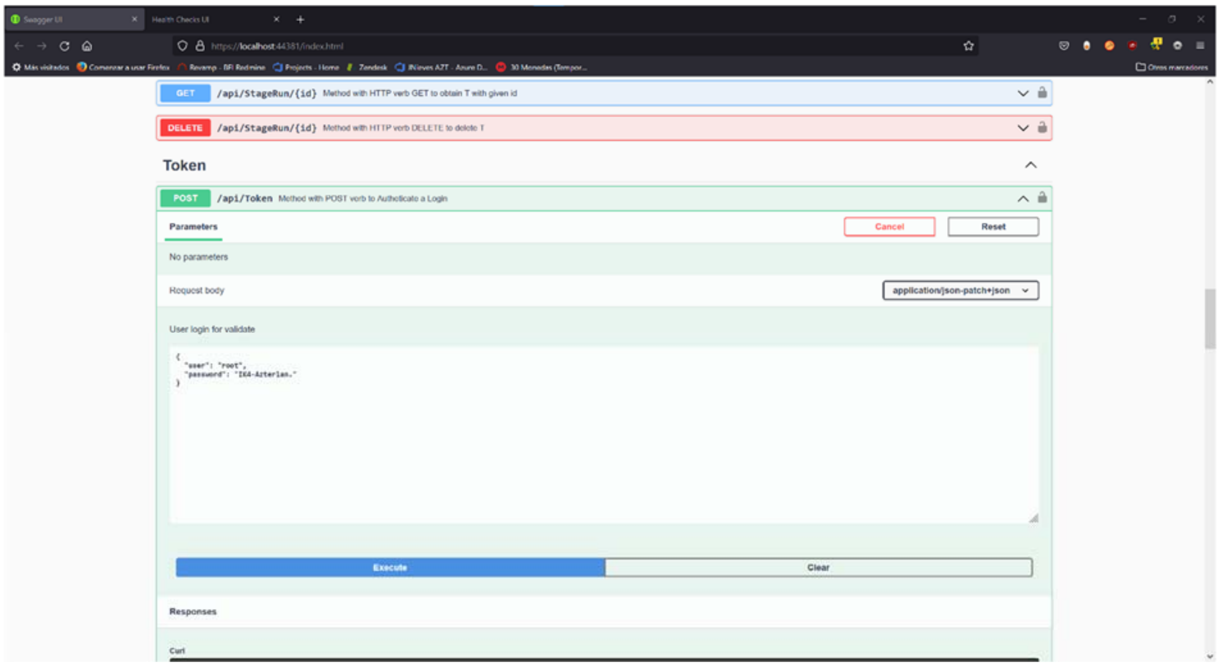
Like a possible further approach and without having developed it, the development team has concluded that the best integration option is the use of Docker containers. In this case, each of the developers (each use case or manufacturing process) could generate their image containing everything necessary for the specific model execution and would give us the configuration to launch it. Thus, there is no need to think about dependencies. Something that simplifies the growth of the system. As future work, the growth of the models should go this way.

4. REVaMP DEMO

On one hand, we must remember that the result of this deliverable is a processing service (creating software as a service), which does not have any type of graphical user interface. However, the documentation is included with Swagger, as already mentioned, and it can be launched from it, as we illustrate in the following demo.

On the other hand, the deliverable is a software and through the document an exhaustive explanation of how it has been developed has been made. In addition, the following explanations and images illustrate a demo of how a user can make a prediction and receive the results. One of the benefits of developed service is that all execution processes are launched in the same way. The only difference is in the values to pass to the models and the controller to call.

Firstly, to use the service it is needed to make the authentication and authorization. In order to achieve it, the user must call a method with the user and password to get the token (JWT token). Following image shows how to call the method. Each user has his own authorizations.



GET `/api/StageRun/{id}` Method with HTTP verb GET to obtain T with given id

DELETE `/api/StageRun/{id}` Method with HTTP verb DELETE to delete T

Token

POST `/api/Token` Method with POST verb to Authenticate a Login

Parameters

No parameters

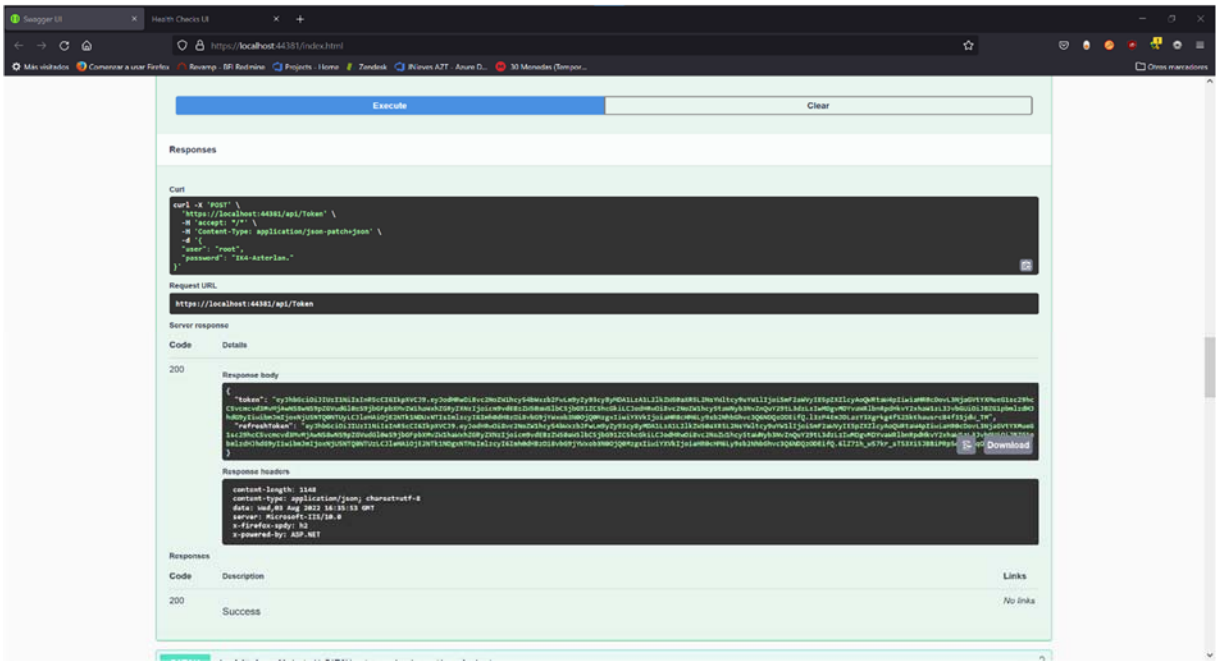
Request body: application/json-patch+json

User login for validate

```
{
  "user": "root",
  "password": "IK4-Azterlan."
}
```

Figure 41.- Calling the Token Controler POST method to login the user.

Secondly, if the user and password are right, our service will provide both, the authorization token and the refresh token to renew the authorization. This image shows the call done to get that information and the results with the tokens.



Responses

Curl

```
curl -X 'POST' \
  'https://localhost:44381/api/Token' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json-patch+json' \
  -d '{
  "user": "root",
  "password": "IK4-Azterlan."
}'
```

Request URL

```
https://localhost:44381/api/Token
```

Server response

Code 200

Response body

```
{
  "token": "[illegible]",
  "refreshToken": "[illegible]"
}
```

Response headers

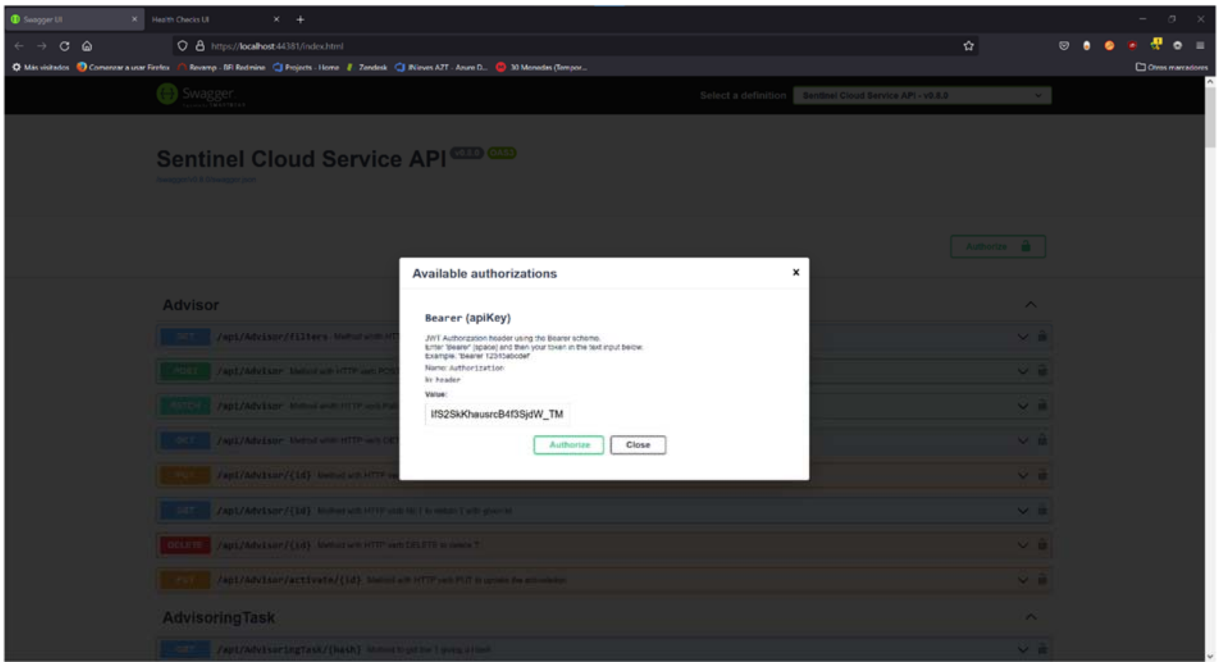
```
content-length: 1148
content-type: application/json; charset=utf-8
date: Wed,03 Aug 2022 16:15:53 GMT
server: Microsoft-IIS/10.0
x-firefox-spdy: h2
x-powered-by: ASP.NET
```

Responses

Code	Description	Links
200	Success	No links

Figure 42.- Getting the Authoriztaion Token and the Refresh Token.

Thirdly, if the user wants to call other methods, he must add the token to the communication headers. Here, with swagger, there is a button to add it. Next figure show how to do it.



Available authorizations

Bearer (apiKey)

JWT Authorization header using the Bearer scheme.
Enter 'Bearer' [space] and then your token in the text input below.
Example: 'Bearer 12345abcdef'

Name: Authorization

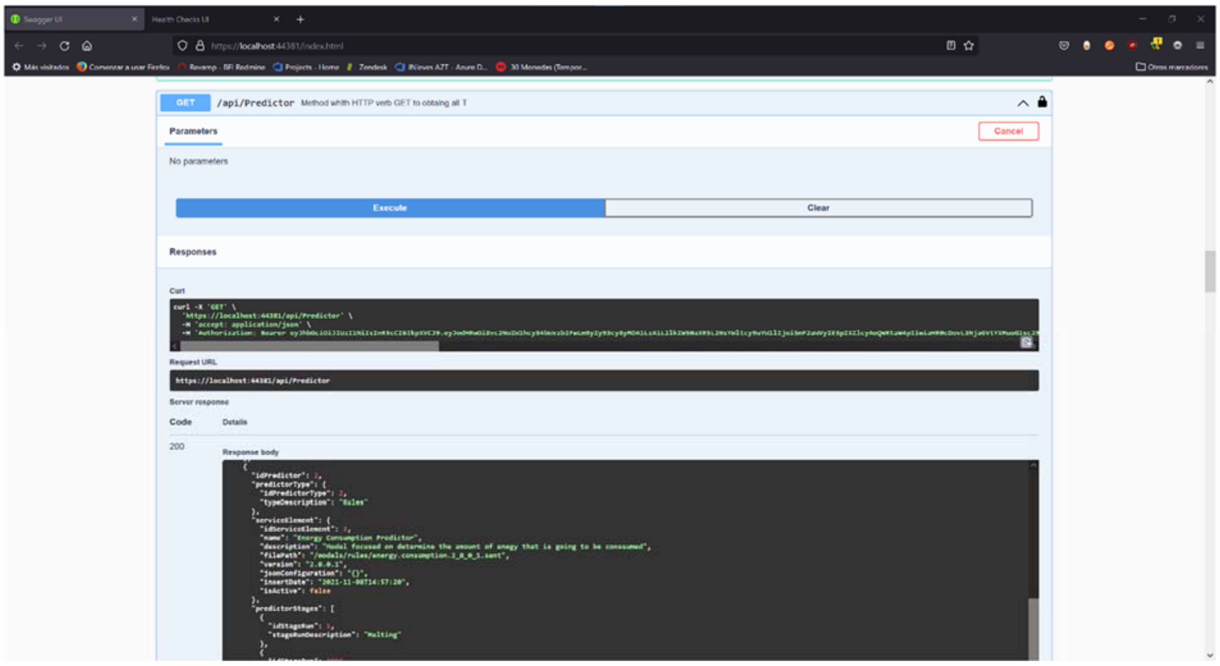
In: header

Value:

lfS2SkKhausrcB4f3SjdW_TM

Figure 43.- Adding the authorization token to our calls.

Before launching the predictions we need to know which model we want to use. Hence, the best option is to make a query to get the list of all available elements. In the following example, we made that call and we recieve the list of all available predictors.



GET /api/Predictor Method whith HTTP verb GET to obtaing all T

Parameters

No parameters

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:44381/api/Predictor' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer [illegible]
```

Request URL

```
https://localhost:44381/api/Predictor
```

Server response

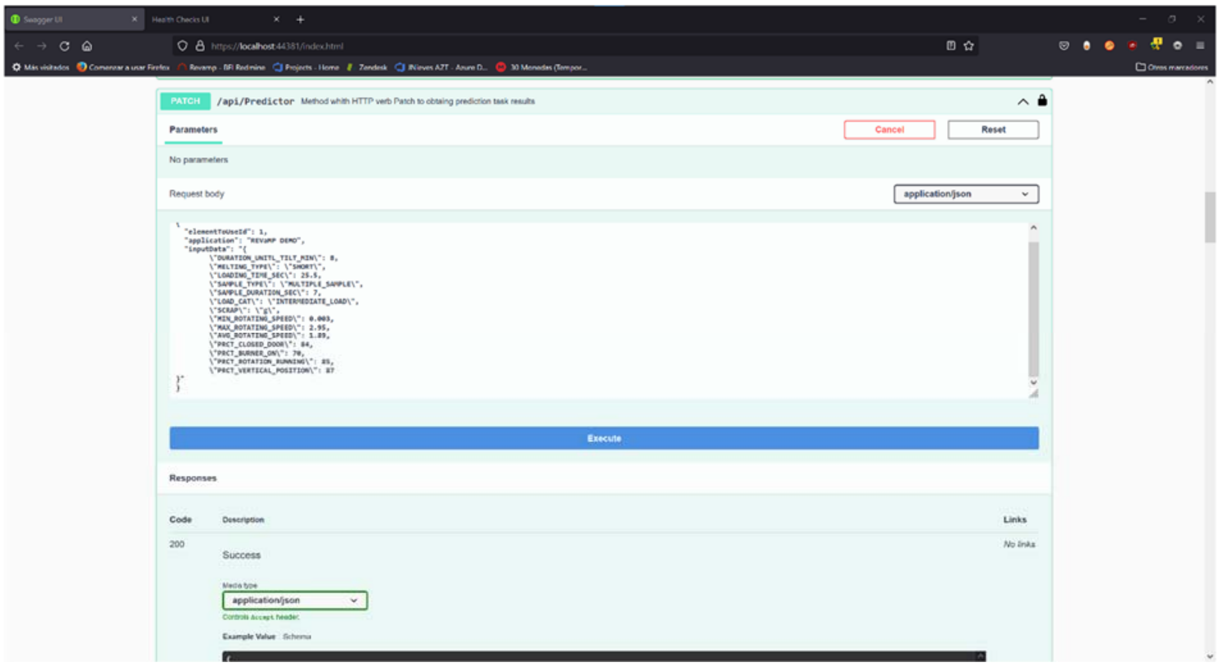
Code 200

Response body

```
  {
    "idPredictor": 2,
    "predictorType": {
      "idPredictorType": 2,
      "typeDescription": "Rules"
    },
    "serviceElement": {
      "idServiceElement": 2,
      "name": "Energy Consumption Predictor",
      "description": "Model focused on determine the amount of enegy that is going to be consumed",
      "filePath": "/models/rules/energy.consumption.1_0_0_1.sent",
      "version": "1.0.0.1",
      "jsonConfiguration": "{}",
      "insertDate": "2021-11-08T14:57:20",
      "isActive": false
    },
    "predictorStages": [
      {
        "idStageRun": 1,
        "stageRunDescription": "Malting"
      },
      {
```

Figure 44.- Getting the list of all predictors to select the one that we want to use.

Once we already know which model we want to lunch, we have to prepare the PATCH call with the ID of the model, the application name to register it and the JSON information to be used in the model. Here, the following figure illustrates this task.



PATCH /api/Predictor Method whith HTTP verb Patch to obtaing prediction task results

Parameters

No parameters

Request body: application/json

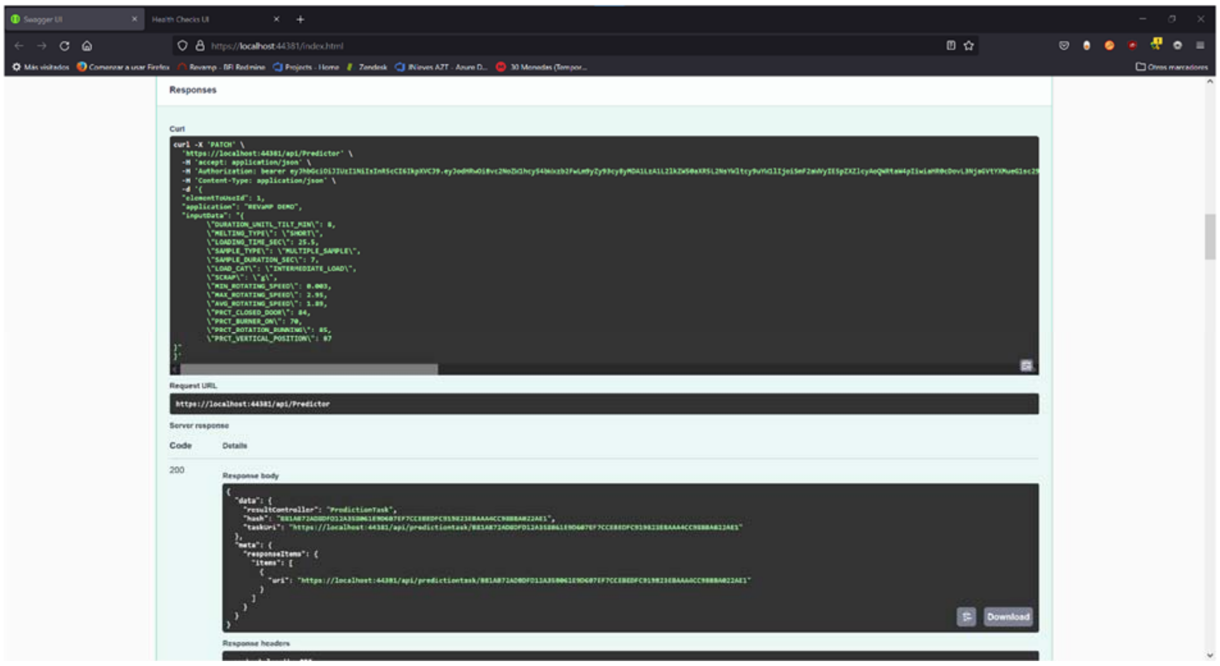
```
{
  "elementToUseId": 1,
  "application": "REVaMP DEMO",
  "inputData": "{
        \"DURATION_UNITL_TILT_MIN\": 8,
        \"MELTING_TYPE\": \"SHORT\",
        \"LOADING_TIME_SEC\": 25.5,
        \"SAMPLE_TYPE\": \"MULTIPLE_SAMPLE\",
        \"SAMPLE_DURATION_SEC\": 7,
        \"LOAD_CAT\": \"INTERMEDIATE_LOAD\",
        \"SCRAP\": \"g\",
        \"MIN_ROTATING_SPEED\": 0.003,
        \"MAX_ROTATING_SPEED\": 2.95,
        \"AVG_ROTATING_SPEED\": 1.89,
        \"PRCT_CLOSED_DOOR\": 84,
        \"PRCT_BURNER_ON\": 70,
        \"PRCT_ROTATION_RUNNING\": 85,
        \"PRCT_VERTICAL_POSITION\": 87
}"
}
```

Responses

Code	Description	Links
200	Success Media type: application/json Controls Accept header. Example Value | Schema	No links

Figure 45.- Preparing the message to call the prediction process, in tihs case, gas consumption prediction.

Due to the way this service was developed, after we ask for a work, we will receive a HASH value to extract the results. Moreover, the service also gives us the URL to the result to navigate to it. This next figure shows how we receive the HASH information.



Responses

Curl

```
curl -X 'PATCH' \
  'https://localhost:44381/api/Predictor' \
  -H 'accept: application/json' \
  -H 'Authorization: bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.[illegible]
  -H 'Content-Type: application/json' \
  -d '{
  "elementToUseId": 1,
  "application": "REVaMP DEMO",
  "inputData": "{
        \"DURATION_UNITL_TILT_MIN\": 8,
        \"MELTING_TYPE\": \"SHORT\",
        \"LOADING_TIME_SEC\": 25.5,
        \"SAMPLE_TYPE\": \"MULTIPLE_SAMPLE\",
        \"SAMPLE_DURATION_SEC\": 7,
        \"LOAD_CAT\": \"INTERMEDIATE_LOAD\",
        \"SCRAP\": \"g\",
        \"MIN_ROTATING_SPEED\": 0.003,
        \"MAX_ROTATING_SPEED\": 2.95,
        \"AVG_ROTATING_SPEED\": 1.89,
        \"PRCT_CLOSED_DOOR\": 84,
        \"PRCT_BURNER_ON\": 70,
        \"PRCT_ROTATION_RUNNING\": 85,
        \"PRCT_VERTICAL_POSITION\": 87
}"
}'
```

Request URL

```
https://localhost:44381/api/Predictor
```

Server response

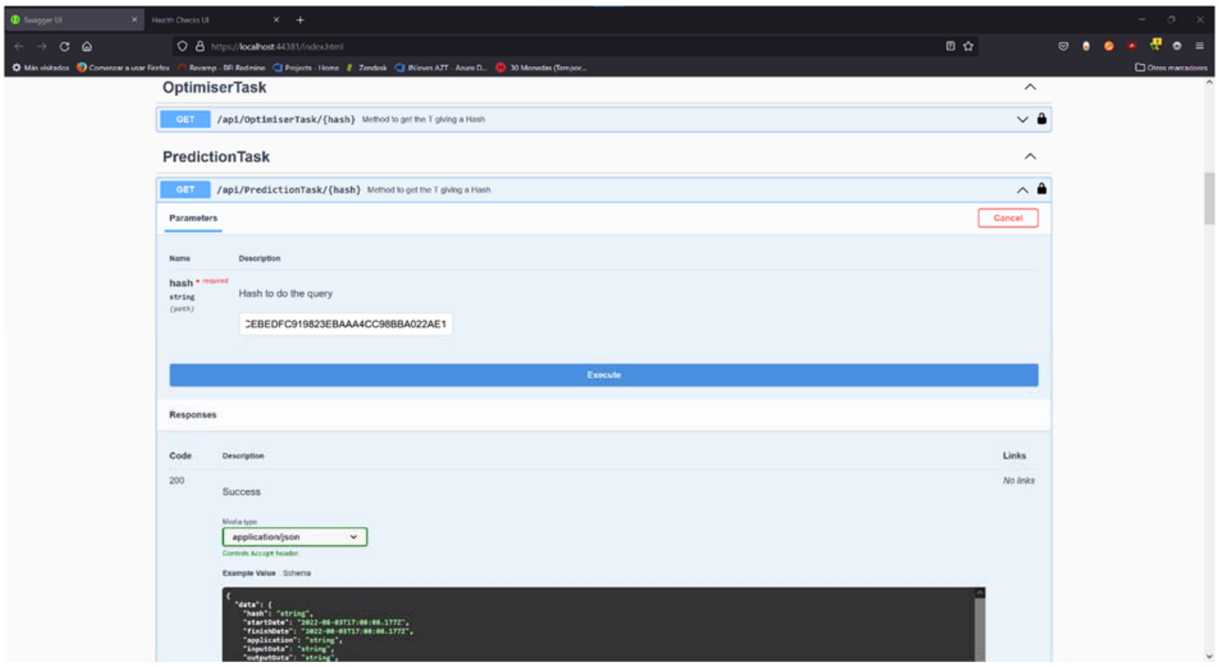
Code 200 — Response body

```
{
  "data": {
    "resultController": "PredictionTask",
    "hash": "881AB72AD8DFD12A35B061E9D607EF7CCEBEDFC9198236BAAA4CC98BBA022AE1",
    "taskUri": "https://localhost:44381/api/predictiontask/881AB72AD8DFD12A35B061E9D607EF7CCEBEDFC9198236BAAA4CC98BBA022AE1"
  },
  "meta": {
    "responseItems": {
      "items": [
        {
          "uri": "https://localhost:44381/api/predictiontask/881AB72AD8DFD12A35B061E9D607EF7CCEBEDFC9198236BAAA4CC98BBA022AE1"
        }
      ]
    }
  }
}
```

Response headers

Figure 46.- Getting the HASH information and the URL when we ask about a prediction.

Then, when we visit the link or when we call the controller to get the information of this HASH (see Figure 47), two situations can occur.



GET /api/OptimiserTask/{hash} Method to get the T giving a Hash

GET /api/PredictionTask/{hash} Method to get the T giving a Hash

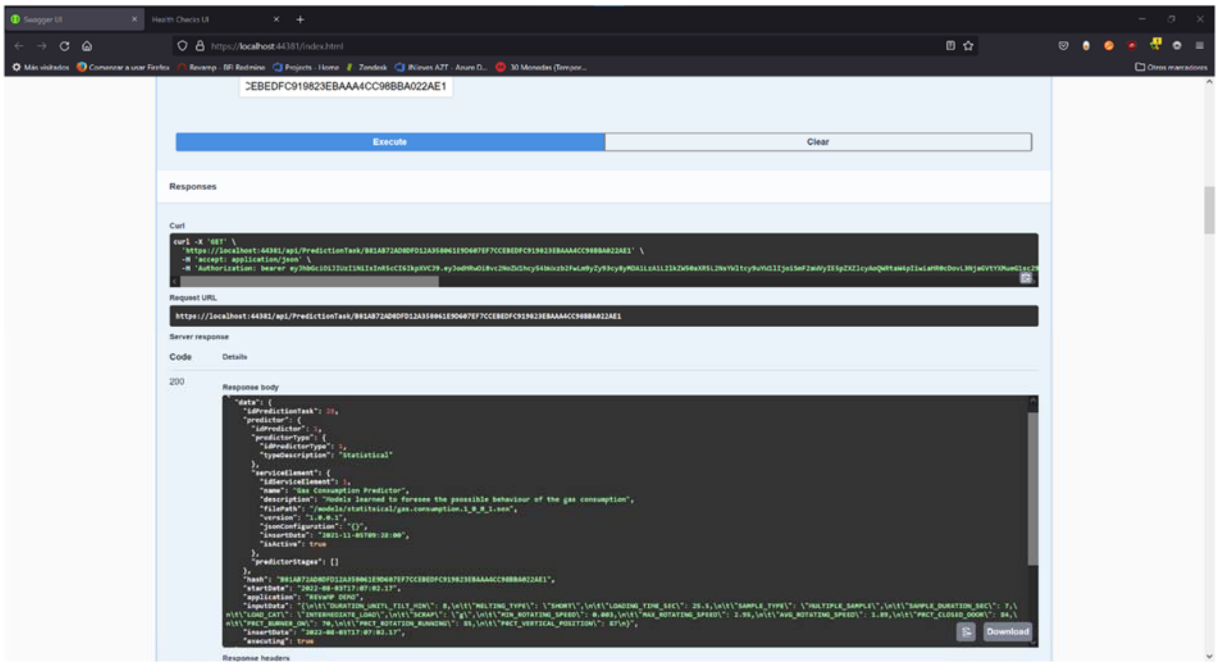
Name	Description
hash * required string (path)	Hash to do the query ↄEBEDFC919823EBAAA4CC98BBA022AE1

Code	Description	Links
200	Success Media type: application/json Controls Accept header. Example Value · Schema	No links

```
{
  "data": {
    "hash": "string",
    "startDate": "2022-08-03T17:08:08.177Z",
    "finishDate": "2022-08-03T17:08:08.177Z",
    "application": "string",
    "inputData": "string",
    "outputData": "string",
```

Figure 47.- Using the HASH value to get the results.

The first one is that the service is sill working and we do not have the result (see Fig. 48). And the last one is when we already have all work done (see Fig. 49). After that, we already have the result of our prediction to be used by the client.



CEBEDFC919823EBAAA4CC98BBA022AE1

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:44381/api/PredictionTask/B81AB72AD8DFD12A358061E9D607EF7CCEBEDFC919823EBAAA4CC98BBA022AE1' \
  -H 'accept: application/json' \
  -H 'Authorization: bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9uYW1lIjoiSmF2aWVyIESpZXZlcyAoQWRtaW4pIiwiaHR0cDovL3NjaGVtYXMueG1sc29
```

Request URL

```
https://localhost:44381/api/PredictionTask/B81AB72AD8DFD12A358061E9D607EF7CCEBEDFC919823EBAAA4CC98BBA022AE1
```

Server response

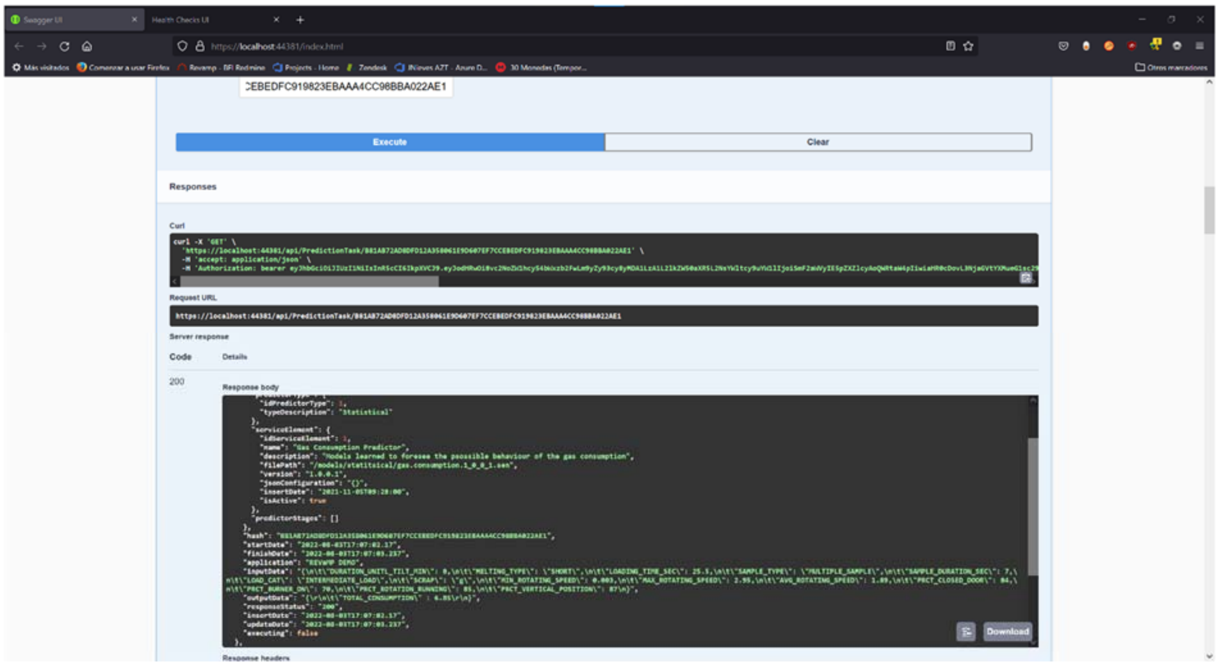
Code Details

200 Response body

```
  "data": {
    "idPredictionTask": 28,
    "predictor": {
      "idPredictor": 1,
      "predictorType": {
        "idPredictorType": 1,
        "typeDescription": "Statistical"
      },
      "serviceElement": {
        "idServiceElement": 1,
        "name": "Gas Consumption Predictor",
        "description": "Models learned to foresee the possible behaviour of the gas consumption",
        "filePath": "/models/statitsical/gas.consumption.1_0_0_1.sav",
        "version": "1.0.0.1",
        "jsonConfiguration": "{}",
        "insertDate": "2021-11-05T09:28:00",
        "isActive": true
      },
      "predictorStages": []
    },
    "hash": "B81AB72AD8DFD12A358061E9D607EF7CCEBEDFC919823EBAAA4CC98BBA022AE1",
    "startDate": "2022-08-03T17:07:02.17",
    "application": "REVaMP DEMO",
    "inputData": "{\n\t\"DURATION_UNITL_TILT_MIN\": 8,\n\t\"MELTING_TYPE\": \"SHORT\",\n\t\"LOADING_TIME_SEC\": 25.5,\n\t\"SAMPLE_TYPE\": \"MULTIPLE_SAMPLE\",\n\t\"SAMPLE_DURATION_SEC\": 7,\
n\t\"LOAD_CAT\": \"INTERMEDIATE_LOAD\",\n\t\"SCRAP\": \"g\",\n\t\"MIN_ROTATING_SPEED\": 0.003,\n\t\"MAX_ROTATING_SPEED\": 2.95,\n\t\"AVG_ROTATING_SPEED\": 1.89,\n\t\"PRCT_CLOSED_DOOR\": 84,\
n\t\"PRCT_BURNER_ON\": 70,\n\t\"PRCT_ROTATION_RUNNING\": 85,\n\t\"PRCT_VERTICAL_POSITION\": 87\n}",
    "insertDate": "2022-08-03T17:07:02.17",
    "executing": true
```

Response headers

Figure 48.- We have all information (for instance, inputs, model, dates) but it is still working.



CEBEDFC919823EBAAA4CC98BBA022AE1

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:44381/api/PredictionTask/B81AB72AD8DFD12A358061E9D607EF7CCEBEDFC919823EBAAA4CC98BBA022AE1' \
  -H 'accept: application/json' \
  -H 'Authorization: bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9uYW1lIjoiSmF2aWVyIESpZXZlcyAoQWRtaW4pIiwiaHR0cDovL3NjaGVtYXMueG1sc29
```

Request URL

```
https://localhost:44381/api/PredictionTask/B81AB72AD8DFD12A358061E9D607EF7CCEBEDFC919823EBAAA4CC98BBA022AE1
```

Server response

Code Details

200 Response body

```
        "idPredictorType": 1,
        "typeDescription": "Statistical"
      },
      "serviceElement": {
        "idServiceElement": 1,
        "name": "Gas Consumption Predictor",
        "description": "Models learned to foresee the possible behaviour of the gas consumption",
        "filePath": "/models/statitsical/gas.consumption.1_0_0_1.sav",
        "version": "1.0.0.1",
        "jsonConfiguration": "{}",
        "insertDate": "2021-11-05T09:28:00",
        "isActive": true
      },
      "predictorStages": []
    },
    "hash": "B81AB72AD8DFD12A358061E9D607EF7CCEBEDFC919823EBAAA4CC98BBA022AE1",
    "startDate": "2022-08-03T17:07:02.17",
    "finishDate": "2022-08-03T17:07:03.237",
    "application": "REVaMP DEMO",
    "inputData": "{\n\t\"DURATION_UNITL_TILT_MIN\": 8,\n\t\"MELTING_TYPE\": \"SHORT\",\n\t\"LOADING_TIME_SEC\": 25.5,\n\t\"SAMPLE_TYPE\": \"MULTIPLE_SAMPLE\",\n\t\"SAMPLE_DURATION_SEC\": 7,\
n\t\"LOAD_CAT\": \"INTERMEDIATE_LOAD\",\n\t\"SCRAP\": \"g\",\n\t\"MIN_ROTATING_SPEED\": 0.003,\n\t\"MAX_ROTATING_SPEED\": 2.95,\n\t\"AVG_ROTATING_SPEED\": 1.89,\n\t\"PRCT_CLOSED_DOOR\": 84,\
n\t\"PRCT_BURNER_ON\": 70,\n\t\"PRCT_ROTATION_RUNNING\": 85,\n\t\"PRCT_VERTICAL_POSITION\": 87\n}",
    "outputData": "{\r\n\t\"TOTAL_CONSUMPTION\" : 6.85\r\n}",
    "responseStatus": "200",
    "insertDate": "2022-08-03T17:07:02.17",
    "updateDate": "2022-08-03T17:07:03.237",
    "executing": false
  },
```

Response headers

Figure 49.- Prediction done, result and end date added.

5. Conclusions

The main result obtained in this task and described in this deliverable is the creation of a service for the use of the different predictive models and optimizers, as well as decision support tools associated with the REVaMP project. In fact, the development of this service is considered as a transversal tool that could be used by the project partners.

More accuratelly, we have created a high level REST API service with the following features:

- A full 3-layer architecture to solve problems related to the furnace optimization, including a high level design and giving an advanced method to achieve the calculation.
- Generation of a clean architecture solution, using several design patterns applied.
- Securization of the use of the service thanks to a token based authentication (JWT)
- Providing a Level 3 of Richardson maturity model (POX, URI, HTTP and Hypermedia) service.
- Creating its own storage.
- Documenting with Swagger, giving also the way of testing the service without using Third Party software.
- Give several deployment possibilities, making easy the use in the plants. For instance, using a Local Internet Information Server Service, Azure Internet Information Server Deployed Service or with Docker (local or remote)

Finally, we have provided the integration of other models developed in WP4. This integration can be done in different ways. Firstly, models developed in Python will be integrated, and, later, models developed in C++ .