Policy Cloud
Cloud for Data-Driven Policy Management

CLOUD FOR DATA-DRIVEN POLICY MANAGEMENT

Project Number: 870675 Start Date of Project: 01/01/2020 Duration: 36 months

D3.5 CLOUD INFRASTRUCTURE INCENTIVES MANAGEMENT AND DATA GOVERNANCE SOFTWARE PROTOTYPE 2

Dissemination Level	PU
Due Date of Deliverable	31/10/2021, Month 22
Actual Submission Date	27/10/2021
Work Package	WP3 Cloud Infrastructures Utilization & Data Governance
Task	T3.1, T3.3, T3.4, T3.6
Type	Demonstrator
Approval Status	
Version	1.0
Number of Pages	p.1 – p.54

Abstract: This document is an accompanying report providing information about the demonstrator of the second version of the Cloud Infrastructure, Incentives Management and Data Governance software prototype. It describes the cloud gateways and APIs, the cloud provisioning mechanisms, the implemented version of the algorithms as well as the data governance tools according to the D3.1 and D3.4 specifications.

PolicyCLOUD has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 870675.

Versioning and Contribution History

Version	Date	Reason	Author
0.1	14/09/2021	ToC	Konstantinos Oikonomou
0.2	18/09/2021	EGI Contributions	Valeria Ardizzone
0.3	20/09/2021	Ubitech contributions for Section 4	Konstantinos Oikonomou
0.4	05/10/2021	ATOS contributions for Section 3	Maria Angeles Sanguino, Miquel Milá, Ana Luiza Pontual
0.5	13/10/2021	EGI Check-In Addition	Valeria Ardizzone, Konstantinos Oikonomou
0.6	15/10/2021	UPRC contributions for Section 2	George Manias
0.7	18/10/2021	Merging and Final Internal Review	Konstantinos Oikonomou, Giannis Ledakis
0.8	25/10/2021	Changes after OKYS Review	Konstantinos Oikonomou, Giannis Ledakis
0.9	26/10/2021	Quality Check	Argyro Mavrogiorgou
1.0	27/10/2021	Submitted version	Konstantinos Oikonomou, Giannis Ledakis

Author List

Organisation	Name
ATOS	Maria Angeles Sanguino, Miquel Milá, Ana Luiza Pontual
EGI	Valeria Ardizzone
UBITECH	Giannis Ledakis, Konstantinos Theodosiou, Konstantinos Oikonomou
UPRC	Ilias Maglogiannis, George Manias, Thanos Kiourtis, Argyro Mavrogiorgou, Konstantinos Koutsoukos

Abbreviations and Acronyms

Abbreviation/Acronym	Definition
ABAC	Attribute Based Access Control
API	Application Programming Interface
EC	European Commission
EOSC	European Open Science Cloud
GUI	Graphical Unit Interface
OIDC	OpenId Connect
PAP	Policy Administration Point
PDP	Policy Decision Point
PDT	Policy Development Toolkit
PEP	Policy Enforcement Point
PIP	Policy Information Point
XACML	eXtensible Access Control Markup Language

Contents

List of Tables

List of Figures

Executive Summary

The second version of the Cloud Infrastructure, Incentives Management and Data Governance software prototype includes the cloud gateways and APIs, the cloud provisioning mechanisms, the implemented version of the algorithms as well as the data governance tools according to the D3.1[1]and D3.4[2] specifications and is built upon the first version of the prototype described in D3.2 [3]. The prototype's cloud infrastructure is supported by RECAS-BARI and is utilized by EGI through cloud gateways. These gateways allow the prototype to gather data from heterogenous data sources, such as Twitter and the global terrorism database.

This second version of the prototype also includes an ABAC based access control mechanism suitable for PolicyCLOUD. This is broken down to 4 key components that can be combined, along with a test client, in order to demonstrate the access control capabilities to the use cases and to proceed with the definition of an accurate data model, based on the provided feedback. Furthermore, an alternative way to authenticate to the prototype with academic and social credential is provided in order to increase the ease of access and the pool of potential adaptors of the whole platform. An updated version of this prototype will be delivered and documented at a later stage for deliverable D3.8 Cloud Infrastructure Incentives Management and Data Governance: Software Prototype 3, due in October 2022.

1 Introduction

This document is an accompanying report for the second iteration of "Cloud Infrastructure Incentives Management and Data Governance: Software Prototype", and is the fifth deliverable of WP3, covering tasks T3.1, T3.3, T3.4, and T3.6. Based on the design and the specifications provided in D3.1 [1] and D3.4[2], all task participants continued their collaboration towards the implementation of their corresponding outcomes to be utilized or integrated into the platform in later stages, and this second prototype is a snapshot of this effort. In the scope of *T3.1 - Cloud Provisioning of the PolicyCLOUD Infrastructure*, INDIGO-DataCloud PaaS Orchestrator continues to be the tool of choice, as reported in D3.2 [3], and thus no further mention or update is required. In the scope of *T3.3 - Cloud Gateways & APIs for Efficient Data Utilization* and *T3.4 - Incentives Management*, the second prototypes of cloud getaways and Incentive management have been described, respectively. Finally, for *T3.6 - Data Governance Model, Protection and Privacy Enforcement* the second prototype of the mechanism that is used for privacy enforcement and the protection of data is described, along with a new alternate way of gaining access to the system. For all these tasks, this document provides the work performed until October 2021 (M22).

1.1 Structure of the document

The rest of the document is structured as follows. Section 2 presents the Cloud Gateway components of the second prototype, while Section 3 describes the Incentives Management prototype for the efficient utilization of citizen and policy maker data. Section 4 presents the Data Governance model and the Privacy enforcement mechanism for the security of the prototype. Finally, Section 5 provides the conclusion of the document, while Section 6 adds the document references.

1.2 Summary of Changes

Section "Cloud Provisioning of the PolicyCLOUD" has been removed for this deliverable when compared to its predecessor (D3.2). Back in D3.2 [3] EGI presented the INDIGO-DataCloud PaaS Orchestrator but there are no updates to be reported in this deliverable. EGI is operating the cloud-based infrastructure for the project and monitoring its performance on a regular basis. The infrastructure is complemented by the INDIGO-DataCloud PaaS orchestrator which allows members of the project to facilitate the access and the deployment of virtual clusters on top of the IaaS resources.

Moreover, concerning the project's Cloud Gateways and APIs component several core updates and changes have been occurred in contrary to D3.2. More specifically, one major update is that from the Cloud Gateways side at least one scenario of each Use Case has been addressed and completed. The latter has been achieved through the implementation and utilization of several components and microservices as they will introduced and analysed in Section 2. To this end, the initial given and introduced in D3.2 API has been extended, and more API endpoints have been added, where each one triggers the parsing of respective datasets. Moreover, the integration with the project's Interim Repository, which was first introduced in the context of D6.6 [4], has been established. What is more, monitoring service has been implemented on top of the API microservices in order to track all requests, trace possible errors, find requests that are causing long response time and generally have troubleshooting solutions for problems regarding the whole Gateway services. On top of this, Traefik reverse proxy has been utilized in the PolicyCLOUD Cloud Gateways in order to receive all incoming requests and map them to the correct microservice and also to work as a load balancer. This specific service can facilitate any future utilization and deployment of the Cloud Gateways in a more distributed environment. What is more, in terms of providing an integrated and end-to-end solution and ingestion pipeline the Cloud Gateways component has integrated with project's serverless platform, the Apache OpenWhisk. One of the major and core changes is that the implementation of this component was based on the utilization of the MoleculerJS instead of having different implementations based on Laravel and Flask as in the 1st Prototype that was introduced in the scoped of D3.2. More specifically, the MoleculerJS is a framework on top of NodeJS that is specialized in providing microservices and has built-in tools and patterns that are utilized to build a robust and scalable Gateway component. Hence, instead of having different ways of implementation and different design approaches, a unified implementation has been followed, which facilitates the overall scalability and extendibility of the component. Finally, in contrary to D3.2 the integration with the Keycloak, project's User Authentication mechanism provided by Ubitech partner, has been established.

2 Cloud Gateways & APIs for Efficient Data Utilization

The Cloud Gateway and API component will enhance the abilities and services offered by a unified Gateway to move streaming and batch data from data owners into PolicyCLOUD data stores layers. Hence, it seeks to offer a unified framework for various microservices that are responsible for obtaining data from external data sources e.g., 3rd party APIs, files, streams etc. Based on the specifications provided in D3.4 [2] of the PolicyCLOUD project, the effort related to Cloud Gateways & APIs component will be focused on providing a complete and "smart" entryway into PolicyCLOUD project, allowing multiple APIs or microservices to act cohesively and thus provide a uniform, gratifying experience to each stakeholder. To this end, single API endpoints will be provided to its stakeholder in order to access all the data obtained from the external data sources in JSON format without having to care about complex integrations.

2.1 Prototype Overview

In the context of this deliverable, the 2nd Software Prototype of the Cloud Gateways & APIs component consists of five main microservices. The initial design of these five specific sub-components, that can be used either for fetching data from an external file or from a social media platform like Twitter, includes complete workflows and pipelines for pushing data into the PolicyCLOUD platform. These five sub-components base their functionality on all four different Use Cases according to D6.10 [5] and are being described into the below subsections. In contrary to the previous deliverable these sub-components integrate with PolicyCLOUD's authentication mechanism, the Keycloak, which is further introduced and analysed in Section 4 of this specific deliverable. The integration between Cloud Gateways & APIs component and the User Authorization mechanism ensures that all the required security standards are being met and controls the access to the. More specifically, by using the *KeycloakConnect* npm package and a custom middleware for MoleculerJS, the integration with the Keycloak has been established in order to access and authenticate each microservice with an access token that should be obtained by the Keycloak service and must be used in every future request as Authorization header. To this end, only authorized users can have access on Cloud Gateway's resources. Moreover, as introduced in Section 2 the Cloud Gateways and APIs component integrates with the project's Interim Repository and will act as project's repository where each stakeholder and data provider has the ability to upload its own datasets. This integration has a two-fold advantage. In one hand it offers the ability of not having to set up a potentially high number of adaptors in order to fetch data from various kinds of data sources, while on the other hand use case partners are able to upload fast and easy their respective datasets, which are in various file formats (e.g. xlsx and csv) and which can further be fetched from the Cloud Gateways component. Furthermore, in terms of providing an end-to-end ingestion pipeline the Cloud Gateways component integrates directly with the PolicyCLOUD's identified message bus, the Kafka message broker, and the PolicyCLOUD's identified serverless platform, the OpenWhisk, where functions are activated on demand. In the below figure, the overall architecture of the Cloud Gateways component as also the integrations with other project's tools and components (e.g. OpenWhisk, Kafka, and Interim Repository) are being depicted.

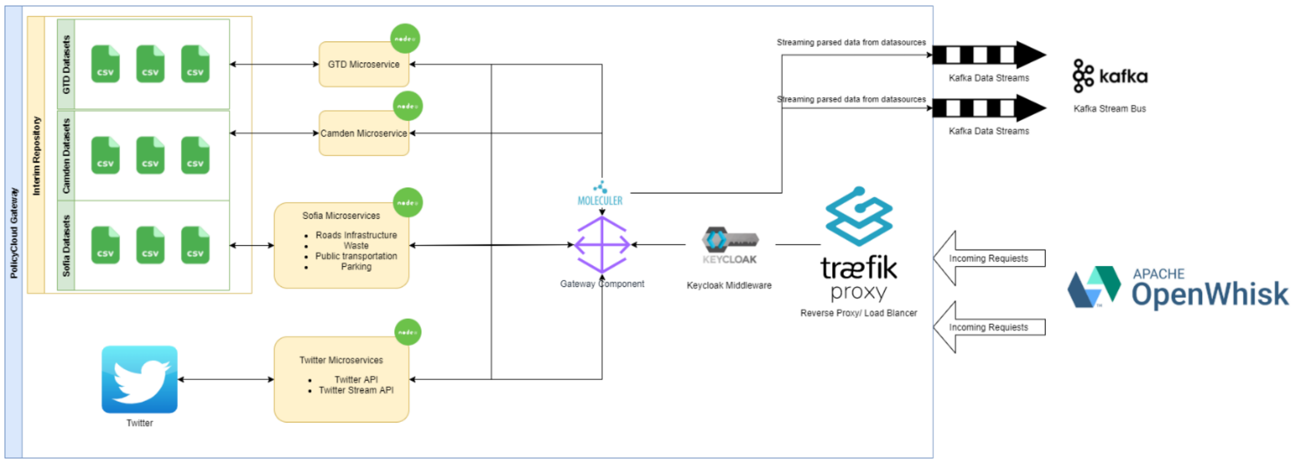


FIGURE 1 - CLOUD GATEWAYS AND APIS ARCHITECTURE AND INTEGRATIONS

2.2 Main Components of the Prototype

2.2.1 Microservices

In a microservices environment the running instances of services dynamically change location inside networks. In order for client or services to be able to make requests to a service it must use a service-discovery mechanism. As introduced in D3.4, the Cloud Gateways and APIs component will rely on the utilization of a unified set of microservices. To this end, MoleculerJS has been utilized as it provides a bult-in module to handle service discovery and registry. The underlying concept of service discovery is the exchange of heartbeats packets between the registry and the available nodes, to list the working services. If a node fails to broadcast a heartbeat is not used to serve requests made for this particular service.

2.2.2 Monitoring – Metrics

Monitoring is a very important aspect in developing microservices as it ensures that all exposed APIs are working as expected without errors. Moreover, it offers also the ability to get metrics for API calls to ensure that the provided infrastructures can handle the incoming traffic load or even to detect attacks. For the Cloud Gateways monitoring the swagger-stats [6] tool have been utilized, which is a Node.js library that collects metrics and traces API requests. Metrics are in Prometheus [7] format, so that enables possible future integration with other widely used metrics and alerting systems like Grafana [8].

Some very interesting and important metrics that are provided include:

- CPU and memory utilization
- Error logging and latest error that occurred
- Request tracing and long request tracing with details request details like headers, parameters, and times
- Statistics and summaries, overall payload measurement during periods of time
- Timelines to help you analyze trends of each request and peak periods

- Built-in Telemetry UI with minimum configuration and many settings that provides good user experience.

For utilizing swagger-stats for Cloud Gateways, the npm package @slanatech/swagger-stats 6 was installed, and in order to collect metrics it was configured as middleware on every request in the gateway microservice.

2.2.3 Global Terrorism Database component (GTD Component)

The GTD sub-component provides a REST application interface following the OpenAPI specification in order to be easier for the end user to discover the capabilities of the component and to provide well-structured documentation for each of the component's services. Furthermore, the component includes an API documentation page, by using Swagger UI, so that a graphical interface for interacting with the API can be provided. This makes it easier for the developer to explore all available requests and responses are listed including the required parameters, without the need of setting up a client on his own.

2.2.4 Twitter component

Twitter Microservice is a component of the Cloud Gateway, providing access to Twitter data to the Gateway's clients without the need to directly connect to Twitter API. It has been implemented in NodeJS utilizing the twitter-v2 npm package7. Furthermore, Twitter is launching the API v2, a new improved version which promises to provide a better developer experience by giving access to a wide variety of data sources and tools. Twitter assures the quality of its data by applying spam filters, access to all results of a query and not only to a partition of results, user-friendly and simplified JSON objects, shorter URLs and OpenAPI specification to test endpoints and watch for any change.

This specific microservice has two (2) basic functionalities:

- **Searching and filtering tweets:** By utilizing the Search Tweets endpoints [9] of the Twitter API v2, Cloud Gateway's users have access to the most recent tweets. The Twitter Connector sub-component provides a REST application interface following the OpenAPI specification in order to be easier for the end user to discover the capabilities of the component and to provide well-structured documentation for each of the component's services. The filters that can be applied include:
    - Keyword search
    - The start and end time parameters to limit tweet results to a specific period of time
    - Max number of tweets in order to limit the results returned
- **Stream tweets for a specific period of time:** By utilizing the Filtered Stream endpoints [10], Cloud Gateways component allow users to capture tweets in real-time related to a specific topic for a pre-selected time window. The available parameters for the endpoint are including:
    - Keyword
    - Duration (milliseconds): The duration parameter specifies the duration of the stream capturing process. There is a max-duration limit, in order to ensure gateway's users are not abusing the service and also Twitter's rate limits policy [11].

2.2.5 London

The London sub-component provides a REST application interface following the OpenAPI specification in order to be easier for the end user to discover the capabilities of the component and to provide well-structured documentation for each of the component's services. Furthermore, the component includes an API documentation page, by using Swagger UI, so that a graphical interface for interacting with the API can be provided. This makes it easier for the developer to explore all available requests and responses are listed including the required parameters, without the need of setting up a client on his own.

2.2.6 Sofia Components

The Sofia sub-component, such as London and GTD sub-components, provides also a REST application interface following the OpenAPI specification in order to be easier for the end user to discover the capabilities of the component and to provide well-structured documentation for each of the component's services.

In contrary to the two previously introduced sub-components this particular component incorporates four sub-basic functionalities that are being offered and its one providing a corresponding microservice for its specific scenario of the Sofia use case.

- Roads
- Transport
- Waste
- Parking

2.3 Interfaces

2.3.1 Monitoring

Through the utilization of the swagger-stats REST API requests and responses in Node.js Microservices and collects statistics per API Operation. This monitoring tool and its interface can be accessed via the below URL http://90.147.75.215:3000/api/swagger-stats/

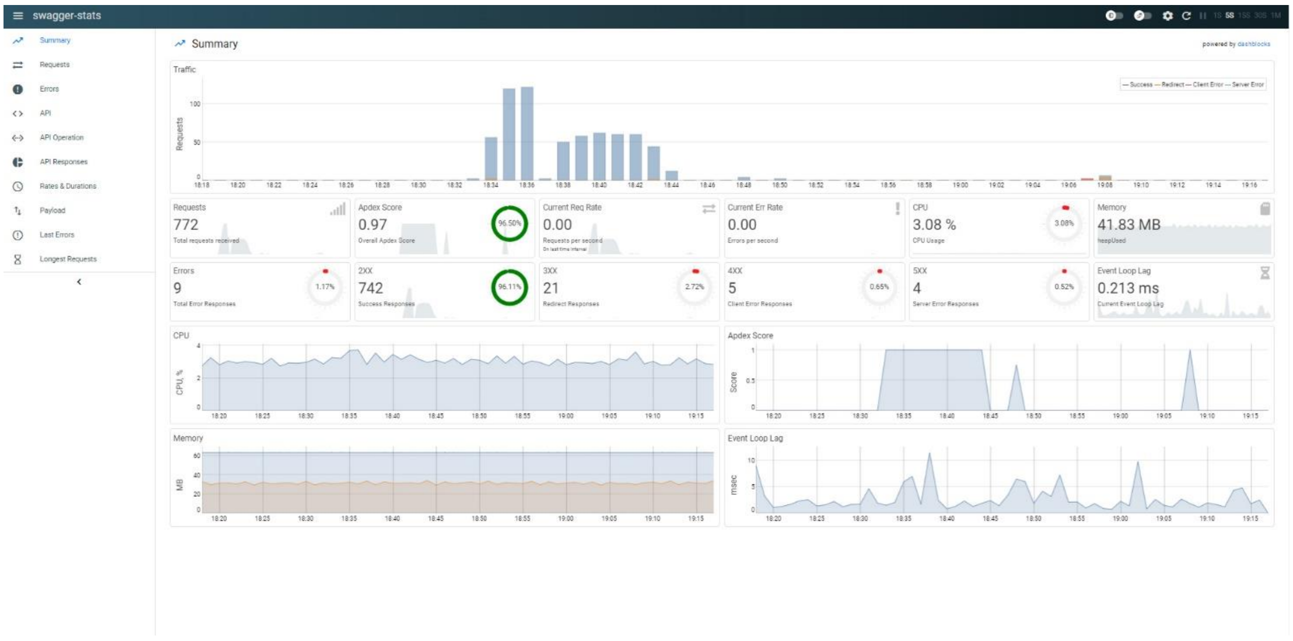
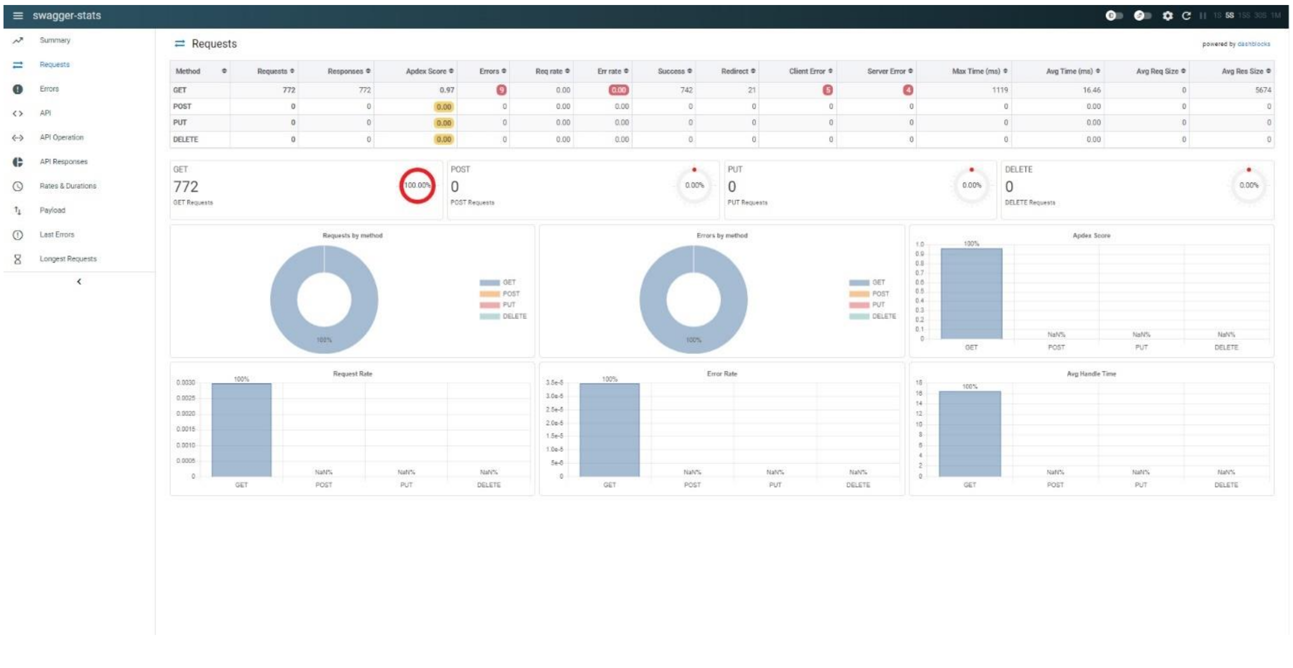


FIGURE 2 – SWAGGER STATS SUMMARY UI



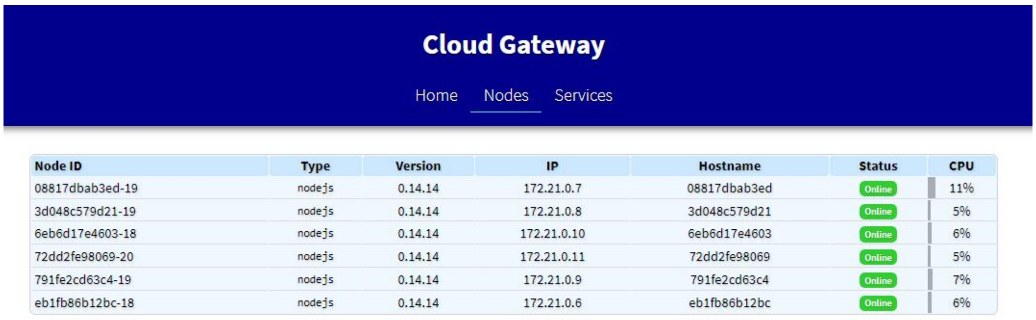
Method	Requests	Responses	Apdex Score	Errors	Req rate	Err rate	Success	Redirect	Client Error	Server Error	Max Time (ms)	Avg Time (ms)	Avg Req Size	Avg Res Size
GET	772	772	0.97	9	0.00	0.00	742	21	5	4	1119	16.46	0	5674
POST	0	0	0.00	0	0.00	0.00	0	0	0	0	0	0.00	0	0
PUT	0	0	0.00	0	0.00	0.00	0	0	0	0	0	0.00	0	0
DELETE	0	0	0.00	0	0.00	0.00	0	0	0	0	0	0.00	0	0

FIGURE 3 – SWAGGER STATS REQUEST PAGE

2.3.2 Microservices

2.3.2.1 MOLECULERJS API

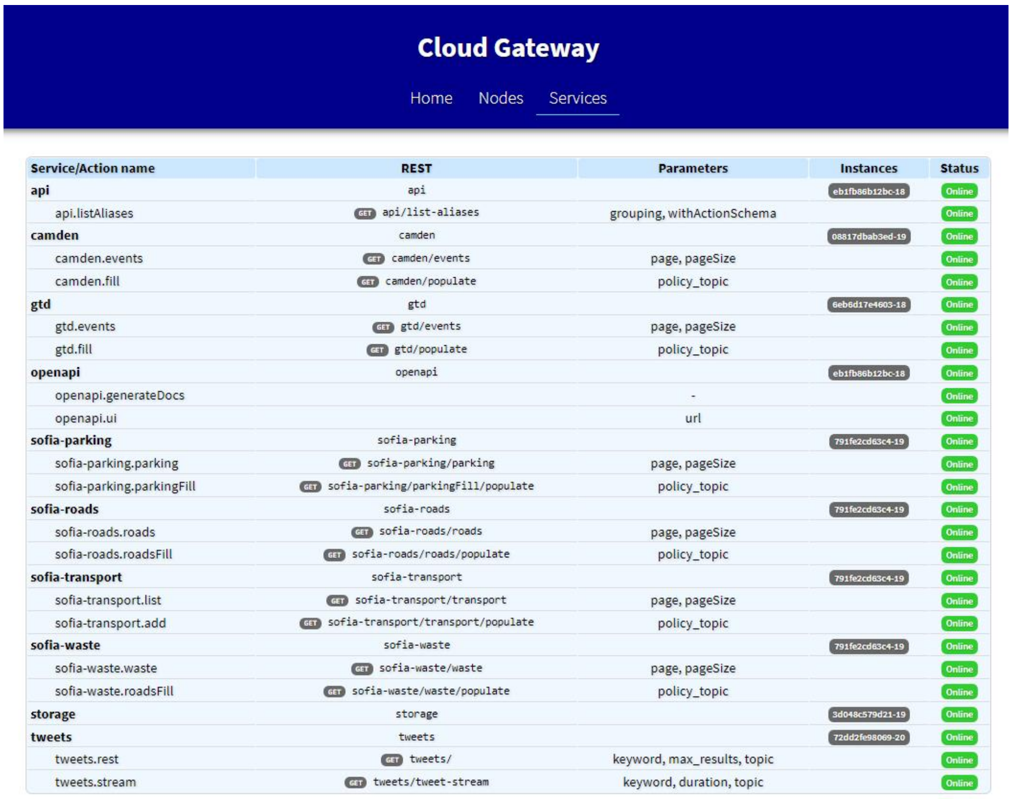
MolecularJS API microservice (http://90.147.75.215:3000/api/) is responsible for load balancing, service registry, monitoring and also serves the OpenAPI specification and any other needed page. In the Service Directory of the MolecularsJS are being depicted the nodes that host each microservice. Moreover, this interface offers information about the CPU utilization for each microservice as long as some other information about the nodes that are running, such as the type of service (nodejs), the ID node and the status of each corresponding microservice/node.



Node ID	Type	Version	IP	Hostname	Status	CPU
08817dbab3ed-19	nodejs	0.14.14	172.21.0.7	08817dbab3ed	Online	11%
3d048c579d21-19	nodejs	0.14.14	172.21.0.8	3d048c579d21	Online	5%
6eb6d17e4603-18	nodejs	0.14.14	172.21.0.10	6eb6d17e4603	Online	6%
72dd2fe98069-20	nodejs	0.14.14	172.21.0.11	72dd2fe98069	Online	5%
791fe2cd63c4-19	nodejs	0.14.14	172.21.0.9	791fe2cd63c4	Online	7%
eb1fb86b12bc-18	nodejs	0.14.14	172.21.0.6	eb1fb86b12bc	Online	6%

FIGURE 4 – MOLECULERJS – SERVICE DISCOVERY

Furthermore, in the "Services" tab it is being depicted the Service Registry visual graphical interface and the full list of the running microservices along with their endpoints, the node instance that are depending on, the exposed REST API and Status for each one of the available microservices.

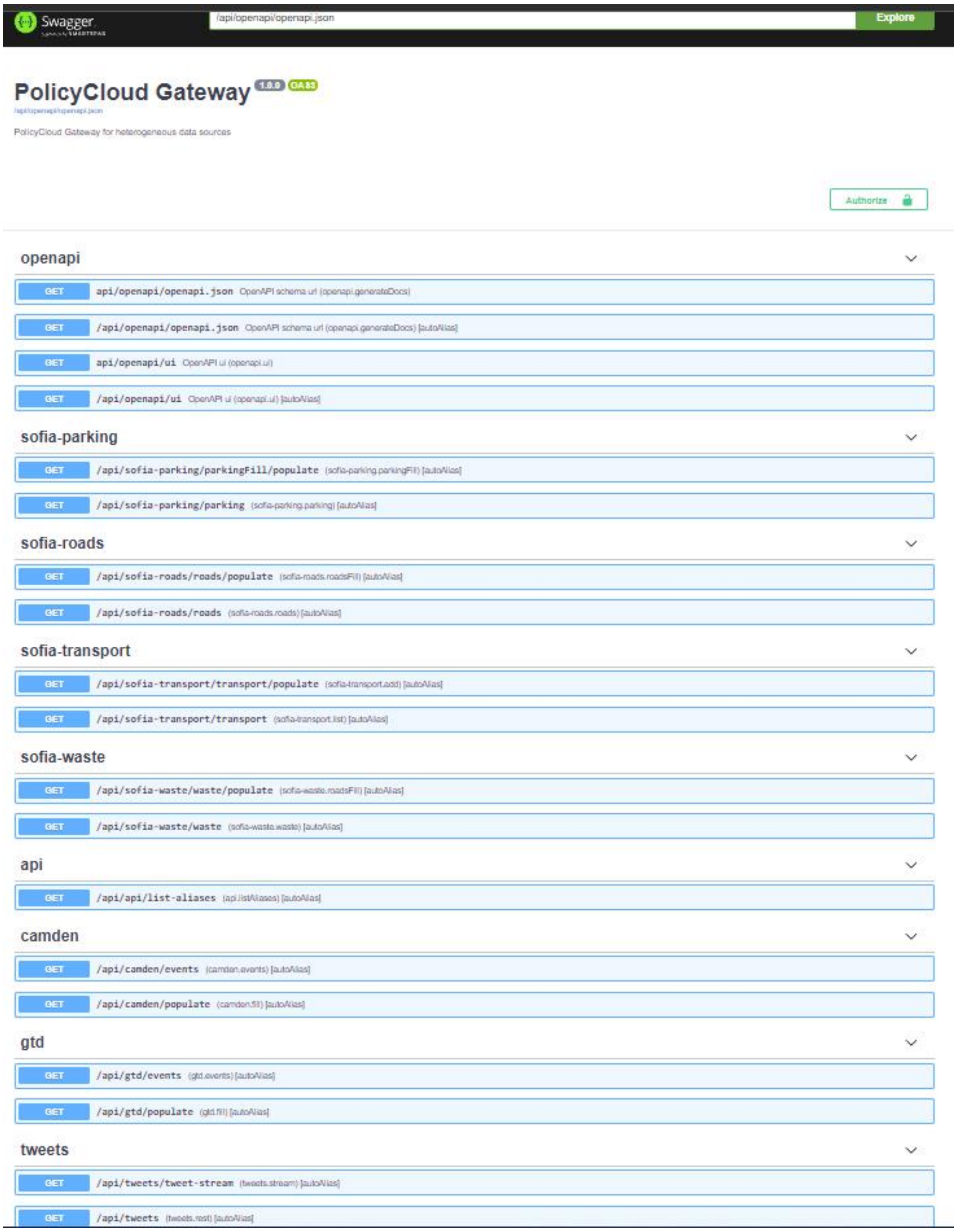


Service/Action name	REST	Parameters	Instances	Status
api	api		eb1fb86b12bc-18	Online
api.listAliases	GET api/list-aliases	grouping, withActionSchema		Online
camden	camden		08817dbab3ed-19	Online
camden.events	GET camden/events	page, pageSize		Online
camden.fill	GET camden/populate	policy_topic		Online
gtd	gtd		6eb6d17e4603-18	Online
gtd.events	GET gtd/events	page, pageSize		Online
gtd.fill	GET gtd/populate	policy_topic		Online
openapi	openapi		eb1fb86b12bc-18	Online
openapi.generateDocs		-		Online
openapi.ui		url		Online
sofia-parking	sofia-parking		791fe2cd63c4-19	Online
sofia-parking.parking	GET sofia-parking/parking	page, pageSize		Online
sofia-parking.parkingFill	GET sofia-parking/parkingFill/populate	policy_topic		Online
sofia-roads	sofia-roads		791fe2cd63c4-19	Online
sofia-roads.roads	GET sofia-roads/roads	page, pageSize		Online
sofia-roads.roadsFill	GET sofia-roads/roads/populate	policy_topic		Online
sofia-transport	sofia-transport		791fe2cd63c4-19	Online
sofia-transport.list	GET sofia-transport/transport	page, pageSize		Online
sofia-transport.add	GET sofia-transport/transport/populate	policy_topic		Online
sofia-waste	sofia-waste		791fe2cd63c4-19	Online
sofia-waste.waste	GET sofia-waste/waste	page, pageSize		Online
sofia-waste.roadsFill	GET sofia-waste/waste/populate	policy_topic		Online
storage	storage		3d048c579d21-19	Online
tweets	tweets		72dd2fe98069-20	Online
tweets.rest	GET tweets/	keyword, max_results, topic		Online
tweets.stream	GET tweets/tweet-stream	keyword, duration, topic		Online

FIGURE 5 – MOLECULERJS – SERVICE REGISTRY

2.3.2.2 OpenAPI

An OpenAPI specification page (http://90.147.75.215:3000/api/openapi/ui#) is also being provided for describing and documenting all API endpoints. This page contains all the required parameters for the request and gives the ability to the user to perform requests directly to the data sources and to obtain real data in the exact data schema and format.



PolicyCloud Gateway 1.0.0 OAS3
/api/openapi/openapi.json
PolicyCloud Gateway for heterogeneous data sources

openapi
GET `api/openapi/openapi.json` OpenAPI schema url (openapi.generateDocs)
GET `/api/openapi/openapi.json` OpenAPI schema url (openapi.generateDocs) [autoAlias]
GET `api/openapi/ui` OpenAPI ui (openapi.ui)
GET `/api/openapi/ui` OpenAPI ui (openapi.ui) [autoAlias]

sofia-parking
GET `/api/sofia-parking/parkingFill/populate` (sofia-parking.parkingFill) [autoAlias]
GET `/api/sofia-parking/parking` (sofia-parking.parking) [autoAlias]

sofia-roads
GET `/api/sofia-roads/roads/populate` (sofia-roads.roadsFill) [autoAlias]
GET `/api/sofia-roads/roads` (sofia-roads.roads) [autoAlias]

sofia-transport
GET `/api/sofia-transport/transport/populate` (sofia-transport.add) [autoAlias]
GET `/api/sofia-transport/transport` (sofia-transport.list) [autoAlias]

sofia-waste
GET `/api/sofia-waste/waste/populate` (sofia-waste.roadsFill) [autoAlias]
GET `/api/sofia-waste/waste` (sofia-waste.waste) [autoAlias]

api
GET `/api/api/list-aliases` (api.listAliases) [autoAlias]

camden
GET `/api/camden/events` (camden.events) [autoAlias]
GET `/api/camden/populate` (camden.fill) [autoAlias]

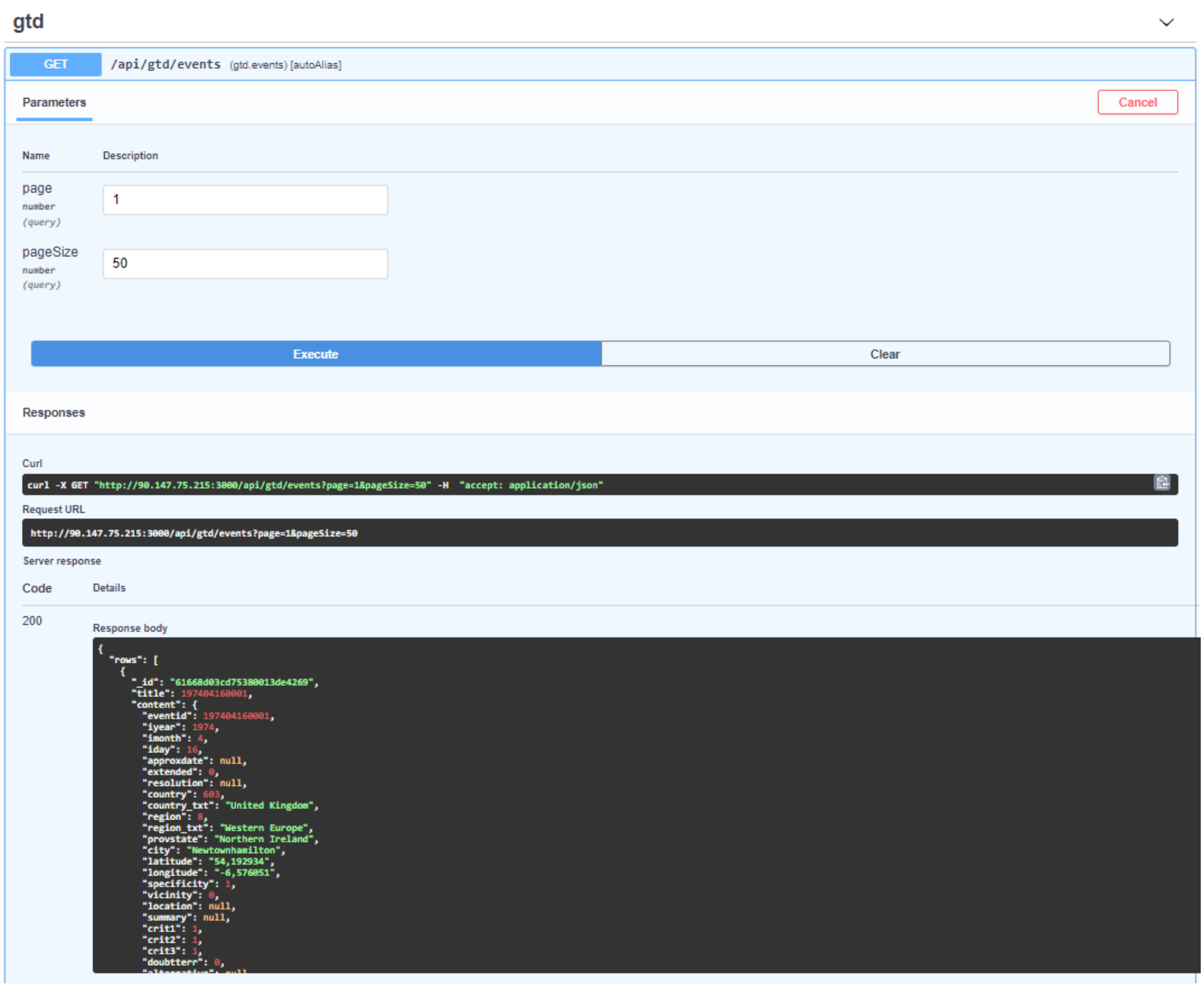
gtd
GET `/api/gtd/events` (gtd.events) [autoAlias]
GET `/api/gtd/populate` (gtd.fill) [autoAlias]

tweets
GET `/api/tweets/tweet-stream` (tweets.stream) [autoAlias]
GET `/api/tweets` (tweets.rest) [autoAlias]

FIGURE 6 – OPENAPI SPECIFICATION PAGE

2.3.2.3 Maggioli

For the Maggioli use case, the Global Terrorism Database (GTD) Service is utilized.



gtd

GET /api/gtd/events (gtd.events) [autoAlias]

Parameters — Cancel

Name	Description
page number *(query)*	1
pageSize number *(query)*	50

Execute — Clear

Responses

Curl

```
curl -X GET "http://90.147.75.215:3000/api/gtd/events?page=1&pageSize=50" -H  "accept: application/json"
```

Request URL

```
http://90.147.75.215:3000/api/gtd/events?page=1&pageSize=50
```

Server response

Code	Details
200	Response body

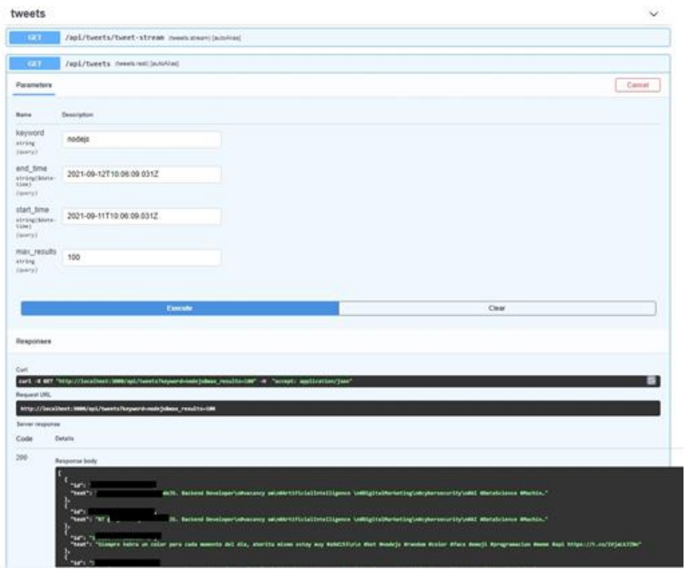
```
{
  "rows": [
    {
      "_id": "61668d03cd75380013de4269",
      "title": 197404160001,
      "content": {
        "eventid": 197404160001,
        "iyear": 1974,
        "imonth": 4,
        "iday": 16,
        "approxdate": null,
        "extended": 0,
        "resolution": null,
        "country": 603,
        "country_txt": "United Kingdom",
        "region": 8,
        "region_txt": "Western Europe",
        "provstate": "Northern Ireland",
        "city": "Newtownhamilton",
        "latitude": "54,192934",
        "longitude": "-6,576051",
        "specificity": 1,
        "vicinity": 0,
        "location": null,
        "summary": null,
        "crit1": 1,
        "crit2": 1,
        "crit3": 1,
        "doubtterr": 0,
```

FIGURE 7 - GTD'S SWAGGER OPENAPI INTERFACE

2.3.2.4 SARGA

For the SARGA use case, the Twitter API and Twitter Streaming API Services are utilized.

- Utilizing Twitter API


tweets

GET `/api/tweets/tweet-stream` (tweets.stream) [autoAlias]

GET `/api/tweets` (tweets.rest) [autoAlias]

Parameters

Name	Description
keyword string (query)	nodejs
end_time string(date-time) (query)	2021-09-12T10:06:09.031Z
start_time string(date-time) (query)	2021-09-11T10:06:09.031Z
max_results string (query)	100

Responses

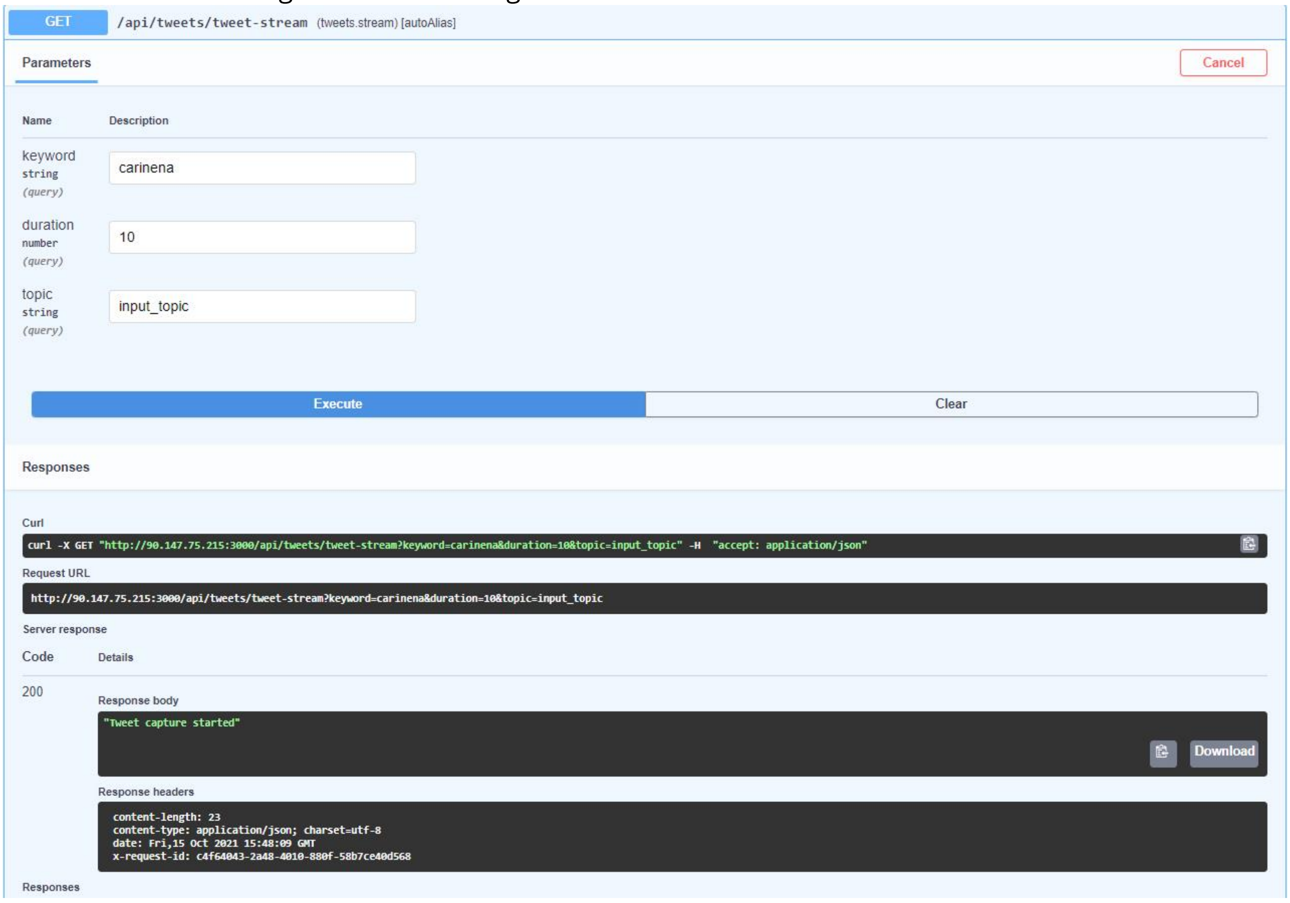
Curl
```
curl -X GET "http://localhost:3000/api/tweets?keyword=nodejs&max_results=100" -H "accept: application/json"
```
Request URL
```
http://localhost:3000/api/tweets?keyword=nodejs&max_results=100
```
Server response

Code: 200

Response body
```
[
  {
    "id": [redacted]
    "text": [redacted]4k35. Backend Developer\nVacancy [illegible]\nArtificialIntelligence \n#DigitalMarketing\n#cybersecurity\n#AI #DataScience #Machin."
  },
  {
    "id": [redacted]
    "text": "RT [redacted]35. Backend Developer\nVacancy [illegible]\nArtificialIntelligence \n#DigitalMarketing\n#cybersecurity\n#AI #DataScience #Machin."
  },
  {
    "id": "[redacted]
    "text": "Siempre habrá un color para cada momento del día, ahorita mismo estoy muy [illegible] #bot #nodejs #random #color #face #emoji #programacion #meme #api https://t.co/[illegible]"
  },
  {
    "id": "[redacted]
```

FIGURE 8 – TWEETS FILTERED SWAGGERUI

- Utilizing Twitter Streaming API


GET `/api/tweets/tweet-stream` (tweets.stream) [autoAlias]

Parameters

Name	Description
keyword string (query)	carinena
duration number (query)	10
topic string (query)	input_topic

Responses

Curl
```
curl -X GET "http://90.147.75.215:3000/api/tweets/tweet-stream?keyword=carinena&duration=10&topic=input_topic" -H  "accept: application/json"
```
Request URL
```
http://90.147.75.215:3000/api/tweets/tweet-stream?keyword=carinena&duration=10&topic=input_topic
```
Server response

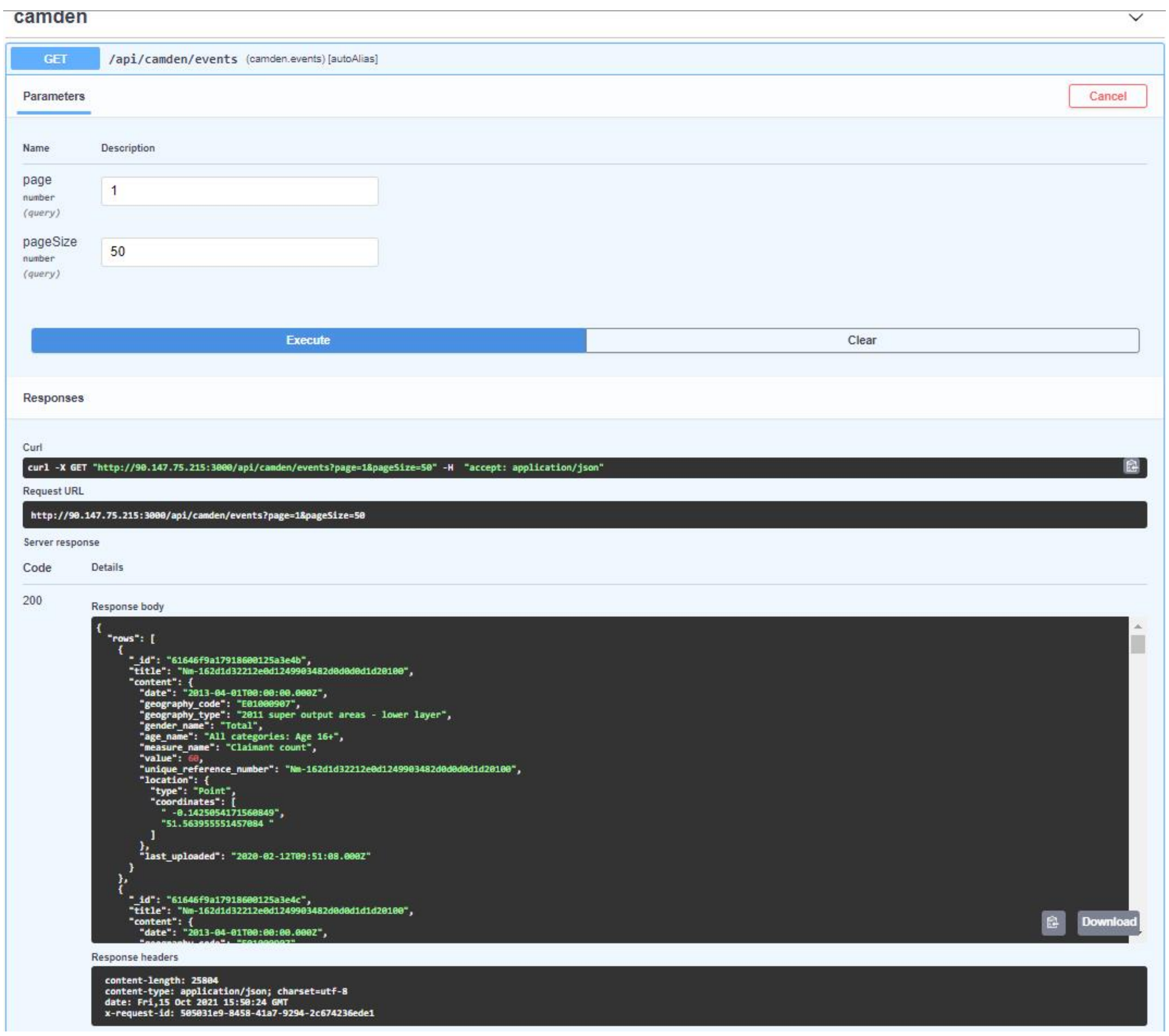
Code: 200

Response body
```
"Tweet capture started"
```
Response headers
```
content-length: 23
content-type: application/json; charset=utf-8
date: Fri,15 Oct 2021 15:48:09 GMT
x-request-id: c4f64043-2a48-4010-880f-58b7ce40d568
```
Responses

FIGURE 9 – TWITTER STREAMING PROCESS

2.3.2.5 LONDON

For the London use case, the Camden Service is utilized.



```
curl -X GET "http://90.147.75.215:3000/api/camden/events?page=1&pageSize=50" -H  "accept: application/json"
```

```
http://90.147.75.215:3000/api/camden/events?page=1&pageSize=50
```

```
{
  "rows": [
    {
      "_id": "61646f9a17918600125a3e4b",
      "title": "Nm-162d1d32212e0d1249903482d0d0d0d1d20100",
      "content": {
        "date": "2013-04-01T00:00:00.000Z",
        "geography_code": "E01000907",
        "geography_type": "2011 super output areas - lower layer",
        "gender_name": "Total",
        "age_name": "All categories: Age 16+",
        "measure_name": "Claimant count",
        "value": 60,
        "unique_reference_number": "Nm-162d1d32212e0d1249903482d0d0d0d1d20100",
        "location": {
          "type": "Point",
          "coordinates": [
            " -0.1425054171560849",
            "51.563955551457084 "
          ]
        },
        "last_uploaded": "2020-02-12T09:51:08.000Z"
      }
    },
    {
      "_id": "61646f9a17918600125a3e4c",
      "title": "Nm-162d1d32212e0d1249903482d0d0d1d1d20100",
      "content": {
        "date": "2013-04-01T00:00:00.000Z",
```

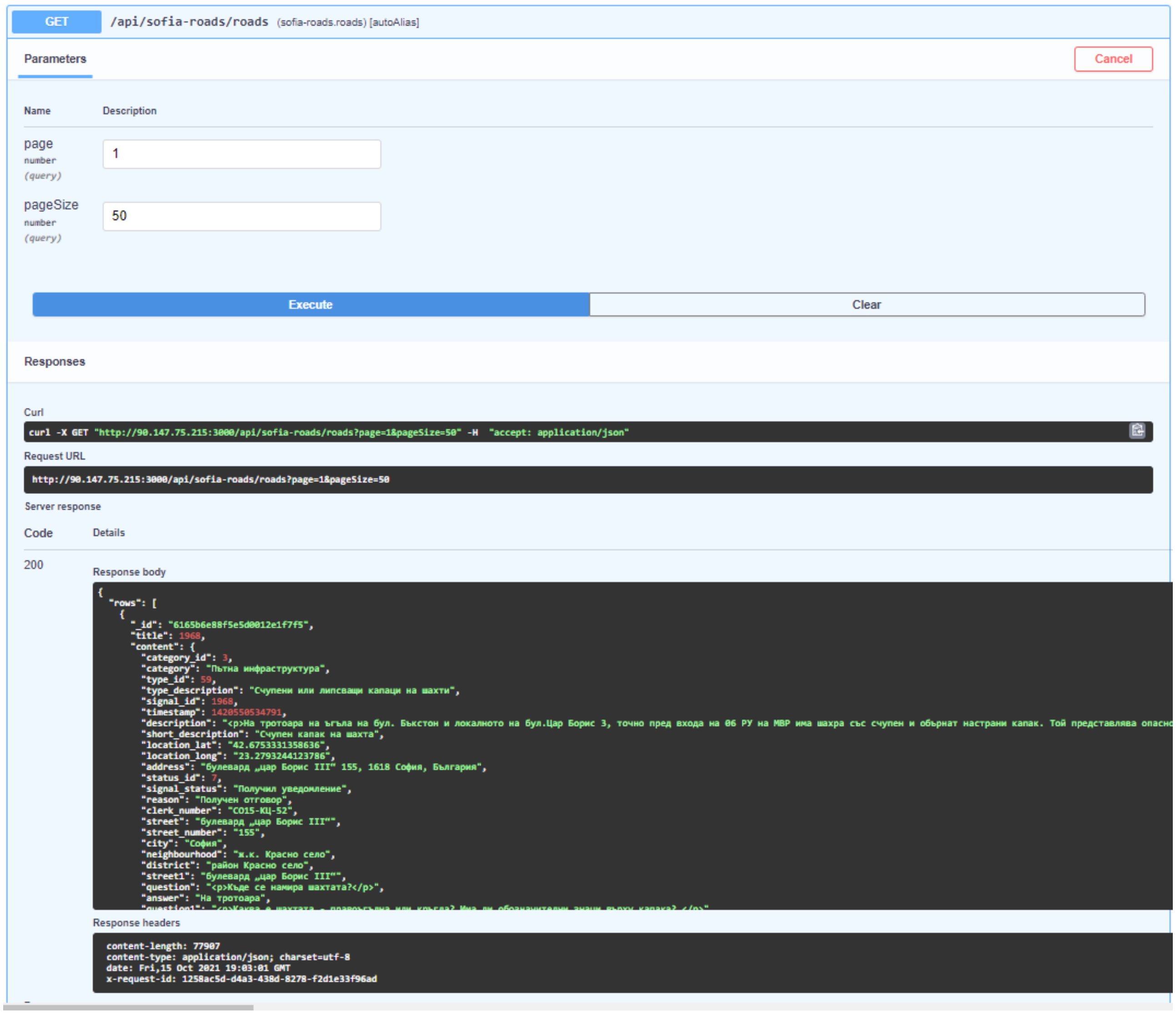
```
content-length: 25804
content-type: application/json; charset=utf-8
date: Fri,15 Oct 2021 15:50:24 GMT
x-request-id: 505031e9-8458-41a7-9294-2c674236ede1
```

FIGURE 10 - LONDON SWAGGER UI

2.3.2.6 Sofia

For the Sofia use case, the Roads, Transport, Waste and Parking Services are utilized.

- Roads



```
curl -X GET "http://90.147.75.215:3000/api/sofia-roads/roads?page=1&pageSize=50" -H  "accept: application/json"
```

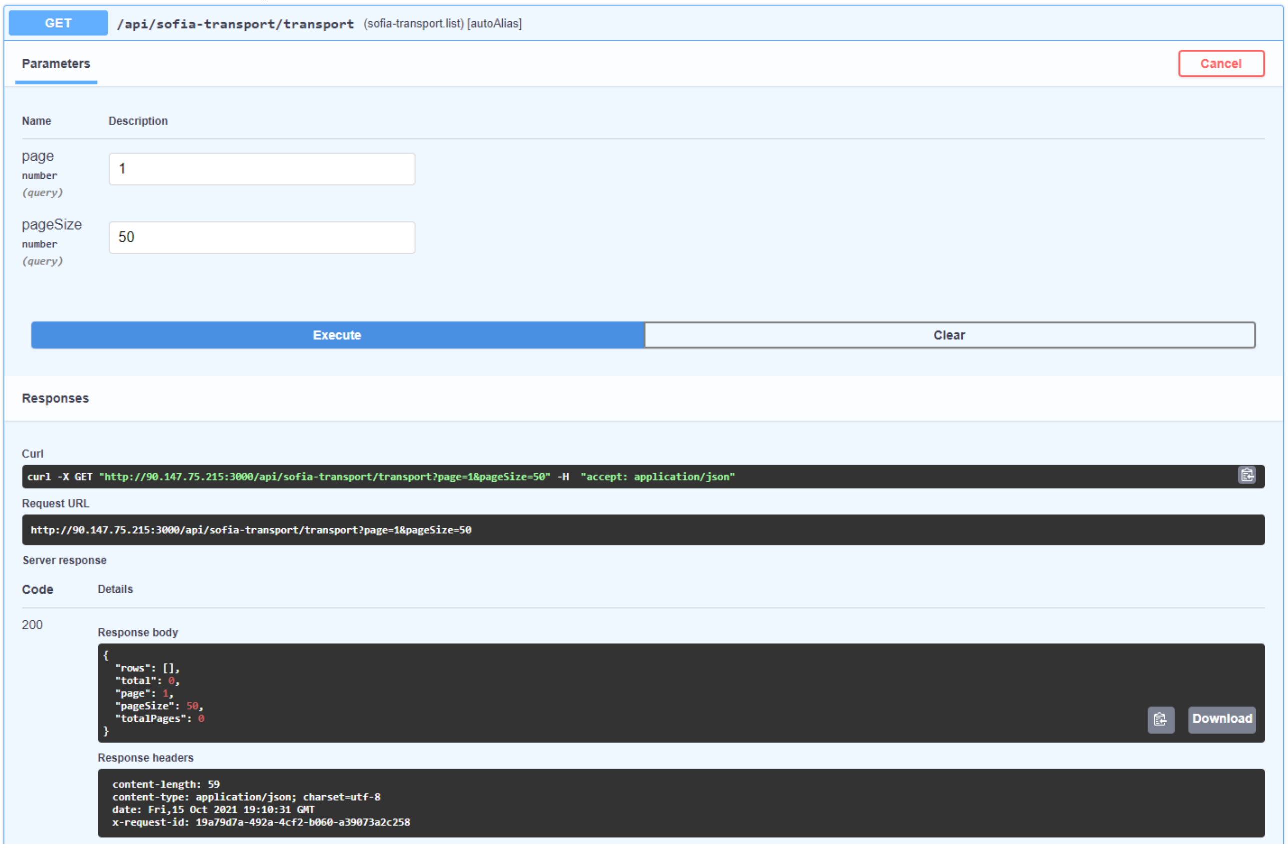
```
http://90.147.75.215:3000/api/sofia-roads/roads?page=1&pageSize=50
```

```
{
  "rows": [
    {
      "_id": "6165b6e88f5e5d0012e1f7f5",
      "title": 1968,
      "content": {
        "category_id": 3,
        "category": "Пътна инфраструктура",
        "type_id": 59,
        "type_description": "Счупени или липсващи капаци на шахти",
        "signal_id": 1968,
        "timestamp": 1420550534791,
        "description": "<p>На тротоара на ъгъла на бул. Бъкстон и локалното на бул.Цар Борис 3, точно пред входа на 06 РУ на МВР има шахра със счупен и обърнат настрани капак. Той представлява опасно
        "short_description": "Счупен капак на шахта",
        "location_lat": "42.6753331358636",
        "location_long": "23.2793244123786",
        "address": "булевард „цар Борис III" 155, 1618 София, България",
        "status_id": 7,
        "signal_status": "Получил уведомление",
        "reason": "Получен отговор",
        "clerk_number": "СО15-КЦ-52",
        "street": "булевард „цар Борис III"",
        "street_number": "155",
        "city": "София",
        "neighbourhood": "ж.к. Красно село",
        "district": "район Красно село",
        "street1": "булевард „цар Борис III"",
        "question": "<p>Къде се намира шахтата?</p>",
        "answer": "На тротоара",
        "question1": "<p>Каква е шахтата - правоъгълна или кръгла? Има ли обозначителни знаци върху капака? </p>"
```

```
content-length: 77907
content-type: application/json; charset=utf-8
date: Fri,15 Oct 2021 19:03:01 GMT
x-request-id: 1258ac5d-d4a3-438d-8278-f2d1e33f96ad
```

FIGURE 11 - SOFIA ROAD SWAGGER UI

- Transport



GET `/api/sofia-transport/transport` (sofia-transport.list) [autoAlias]

Parameters

Name	Description
page number *(query)*	1
pageSize number *(query)*	50

Responses

Curl

```
curl -X GET "http://90.147.75.215:3000/api/sofia-transport/transport?page=1&pageSize=50" -H  "accept: application/json"
```

Request URL

```
http://90.147.75.215:3000/api/sofia-transport/transport?page=1&pageSize=50
```

Server response

Code: 200

Response body

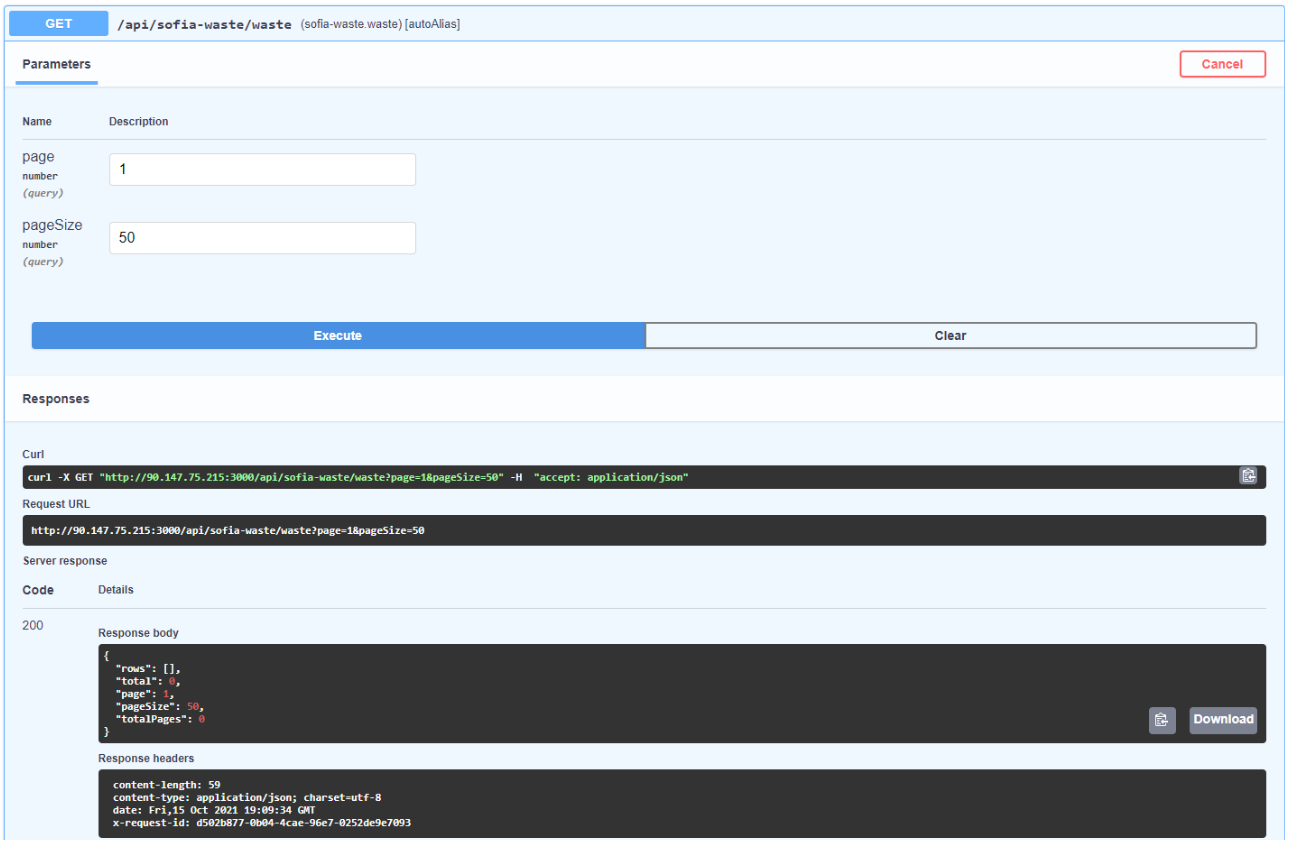
```
{
  "rows": [],
  "total": 0,
  "page": 1,
  "pageSize": 50,
  "totalPages": 0
}
```

Response headers

```
content-length: 59
content-type: application/json; charset=utf-8
date: Fri,15 Oct 2021 19:10:31 GMT
x-request-id: 19a79d7a-492a-4cf2-b060-a39073a2c258
```

FIGURE 12 - SOFIA TRANSPORT SWAGGER UI

- Waste



`/api/sofia-waste/waste` (sofia-waste.waste) [autoAlias]

page number (query): 1

pageSize number (query): 50

Curl

```
curl -X GET "http://90.147.75.215:3000/api/sofia-waste/waste?page=1&pageSize=50" -H  "accept: application/json"
```

Request URL

```
http://90.147.75.215:3000/api/sofia-waste/waste?page=1&pageSize=50
```

Server response

Code: 200

Details

Response body

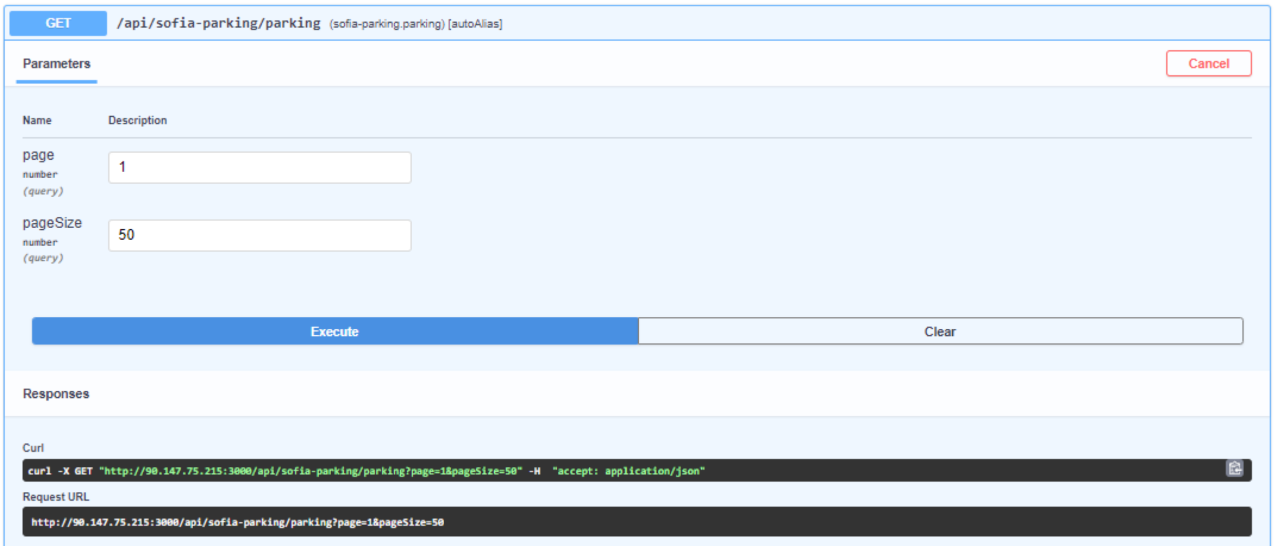
```
{
  "rows": [],
  "total": 0,
  "page": 1,
  "pageSize": 50,
  "totalPages": 0
}
```

Response headers

```
 content-length: 59
 content-type: application/json; charset=utf-8
 date: Fri,15 Oct 2021 19:09:34 GMT
 x-request-id: d502b877-0b04-4cae-96e7-0252de9e7093
```

FIGURE 13 - SOFIA WASTE SWAGGER UI

- Parking



`/api/sofia-parking/parking` (sofia-parking.parking) [autoAlias]

page number (query): 1

pageSize number (query): 50

Curl

```
curl -X GET "http://90.147.75.215:3000/api/sofia-parking/parking?page=1&pageSize=50" -H  "accept: application/json"
```

Request URL

```
http://90.147.75.215:3000/api/sofia-parking/parking?page=1&pageSize=50
```

FIGURE 14 – SOFIA PARKING SWAGGER UI

2.4 Baseline Technologies and Tools

2.4.1 MoleculerJS

MoleculerJS is a lightweight Node.js framework oriented in building and managing microservices. It provides many out-of-the-box features that make the development process faster easier. Moreover, this framework facilitated the focus on the business logic and the providing of the required data from external sources to the PolicyCLOUD in order to be further processed and analyzed by other components. Finally, MoleculerJS is a NodeJS framework meaning that most business logic parts can be transferred and mitigated to other frameworks and distributed systems without major changes and time drawbacks.

2.4.2 Traefik

The Cloud Gateways component utilizes Traefik, a popular HTTP reverse proxy and a load balancer software. The latter requires minimum effort for integration with infrastructure components like Docker and Kubernetes. Traefik instead of requiring manual route configuration for each service component, bundles to the registry service or orchestrator API and generates all routes automatically so this services to be available for public and ready to use. Continuously updating its configurations can be really helpful feature especially in a microservices environment that changes in each microservice are very common issue and component restarts are required [54]. Traefik also provides a UI to monitor metrics and toggle configurations.

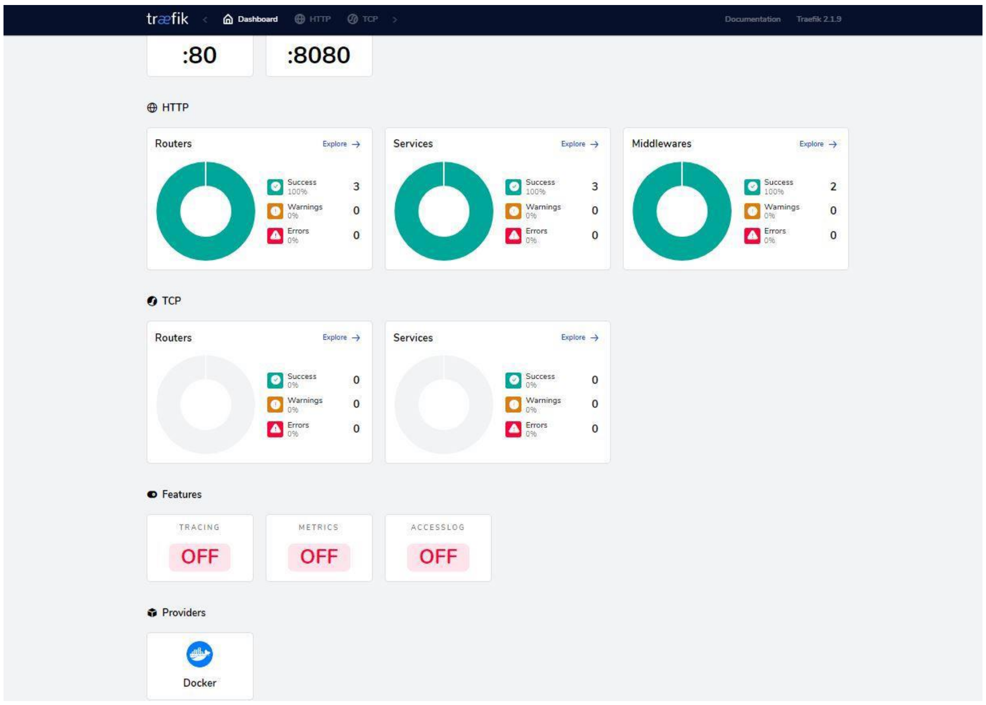


FIGURE 15 – TRAEFIK WEB UI

2.4.3 Docker

Docker is a virtualization platform that that gives the ability to run applications in isolated environments called containers. Each container contains all necessary software to run and application and a be very size efficient. Multiple containers can be running simultaneously on the same host. Docker has been selected in order to provide the Cloud Gateways component and its subcomponents and microservices as docker images that can be initialized as a virtualized container inside a Kubernetes pod. To this end, the development progress can dramatically speed up and the unified component can be deployed faster and safer in production. One other advantage of using containers is the ability to share between developers exactly the same developer environment, thus Cloud Gateways component can be offered to every stakeholder in the same environment.

2.4.4 File Parsers

Parsing data from files of different format has always been a challenge for data science. Different formats, different delimiters and compressions systems can lead add to the complexity of the task. Another serious problem is the parsing of really large data files. In this case common parsing techniques are not working because it is not possible to fit the entire file in the heap memory and the resources are limited.

The Cloud Gateway provides two (2) different file reading microservices. This scenario also includes an integration mechanism with the Interim Repository, in which data providers are responsible for uploading the files that are going to be processed by the Cloud Gateway’s microservices.

2.4.4.1 CSV PAPA.PARSE

For parsing CSV file the papa.parse JS package is being utilized since it provides parsing capabilities over huge files without loading the whole file into memory and thus causing memory leaks and timeouts in the requests. The parsing process and the streaming to the Kafka message broker is an asynchronous process and does not affect the performance or causes any timeout.

2.4.4.2 XLSX FILE PARSER

Along with the CSV datasets, also XLSX datasets are able to be parsed by the microservices provided in the Cloud Gateways and APIs component. These microservices are responsible for parsing the dataset files in XLSX format in order to extract data. Similarly, to the GTD microservice, the parsing procedure is a scheduled job that run in the background and while parsing the data are formatted in Kafka suitable format and sent to the Kafka’s water-topic in order to be later sent and stored in the storage component. The service is implemented in NodeJS and for the CSV parsing the @SheetJS/sheetjs npm package is utilized [12].

2.5 Source Code

2.5.1 Code Overview and Availability

The code and instructions to build the project locally are available at the repository located at https://registry.grid.ece.ntua.gr/george/cloudgatewayv2. Access is restricted to members of the project consortium.

Usage

Start the project with npm run dev command.

After starting, open the http://localhost:3000/ URL in your browser.

On the welcome page you can test the generated services via API Gateway and check the nodes & services.

In the terminal, try the following commands:

2.5.2 NPM scripts

- `npm run dev`: Start development mode (load all services locally with hot-reload & REPL)
- `npm run start`: Start production mode (set SERVICES env variable to load certain services)
- `npm run cli`: Start a CLI and connect to production. Don't forget to set production namespace with --ns argument in script
- `npm run lint`: Run ESLint
- `npm run ci`: Run continuous test mode with watching
- `npm test`: Run tests and generate coverage report
- `npm run dc:up`: Start the stack with Docker Compose
- `npm run dc:down`: Stop the stack with Docker Compose

Commands to setup VPS sudo apt update sudo apt install apt-transport-https ca-certificates curl software-properties-common curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add - sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable" sudo apt update apt-cache policy docker-ce sudo apt install docker-ce sudo systemctl status dokcer docker sudo curl -L "https://github.com/docker/compose/releases/download/1.27.4/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose sudo chmod +x /usr/local/bin/docker-compose docker-compose --version sudo apt install git git clone docker-compose up

2.6 Deployment Status

Currently the Gateway has been deployed on a public VPS server in project's cloud infrastructure that has been provided and is supported by RECAS-BARI and EGI. However, in the project's Gitlab repository are being provided detailed instructions and guidelines that facilitate the deployment of the Cloud Gateways and APIs component and its microservices in any environment.

3 Incentives Management

The Incentive Management software prototype presented here is one more step towards the realization of the final objective of the Incentive Management task, which is based, as we described in D3.1 [1] , on the idea of including citizens as participants in the policy development and management process, having the policy maker as main actor. In this way, we present below the prototype details of the Incentive Management Tool, that aims to help policy makers to achieve this objective.

3.1 Prototype Overview

The work started in D3.2 [3] resulted in a set of functionalities obtained from the use cases, that the Incentive Management Tool needs to provide. The functionalities corresponding to the "Incentives Management" capability were defined in D3.2:

- "Declaration of Incentives: The policy maker should be able to register new incentives with a set of attributes", where the Actions Type attribute takes special relevance.
- "Crowdsource Data Ingestion and Summary: a summary statistic from the crowdsourced data", where crowdsourced task reports, posted in the system, support the statistics which allow the policy maker the evaluation of applied incentives

The Incentive Management Tool has been designed to provide these functionalities to the policy maker. For this first prototype, the Incentive Management Tool component, focusing by now on the first functionality, declaration of incentives, provides:

- **Front End**: The gateway to the component for the policy maker. The Incentive Management Front End will be integrated into the PDT and will be accessed by policy makers through the PolcyCloud platform, although for this first prototype, this first integration is not done yet To make this integration possible, the Front End component has been developed in Angular, to be in line with the PDT, and is composed by:

    - ***incentives-m*anage*ment-home* component**: This component, composed by its "*incentives-management-home*.ts", "*incentives-management-home*.html", *incentives-management-home*.spec.ts and *incentives-management-home*.css files, provides a common "frame" for the following components.
    - ***incentive*-management.services.ts**: Is responsible for the communication with the Back End component.
    - ***incentives* component**: Represents the incentives given to citizens, to get their participation. At the Incentive Management Tool, this component is responsible for the management of the incentives.
    - **motives component**: Represents the motives that encourage citizens to participate. At the Incentive Management Tool, this component is responsible for the management of the motives.

- **actions component**: Represents the actions needed to launch one specific incentive. At the Incentive Management Tool, this component is responsible for the management of the actions.
- **producerstypology component**: Represents the best profile, or best group of people, to perform a selected action. At the Incentive Management Tool, this component is responsible for the management of the producerstypology.

For this first prototype, the communication between Front End and Back End is available, and policy makers can have an initial idea of how this management of Incentives, Motives, Actions and ProducersTypology (create, edit, and delete) can be done.

- **Back End**: This first prototype includes a first version of the Incentive Management Back End. This first version, developed in Python, is composed by five controllers, related to the five mains elements of the Front End:

    - **incentives_controller.py**: This controller contains all the functions related with the incentives management actions.
    - **actions_controller.py**:, This controller contains all the functions related with the actions management.
    - **motives_controller.py**: This controller contains all the functions related with the motives management.
    - **reports_controller.py**: This controller contains all the functions related with the reports management.
    - **producerstypology_controller.py**: This controller contains all the functions related with the producerstypology management.

3.2 Main Components of the Prototype

As commented in earlier sections, the main components of this first prototype are:

- **Front End component**: policy makers can access each of the Incentive Management Tool options from the component's initial page, or using the top left menu list:

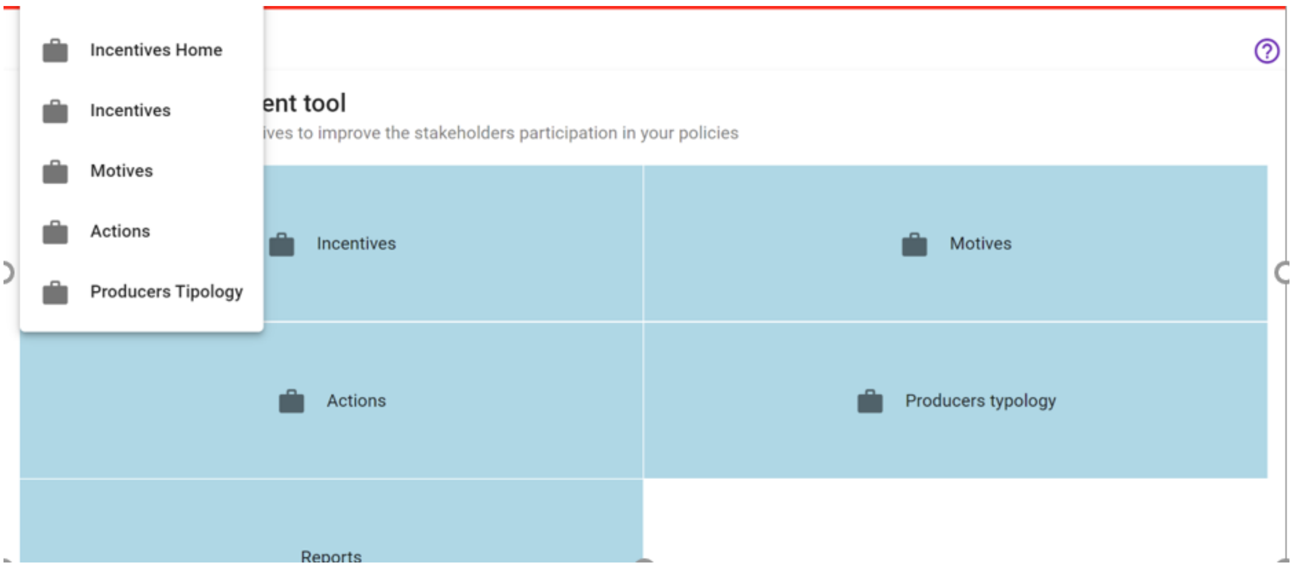
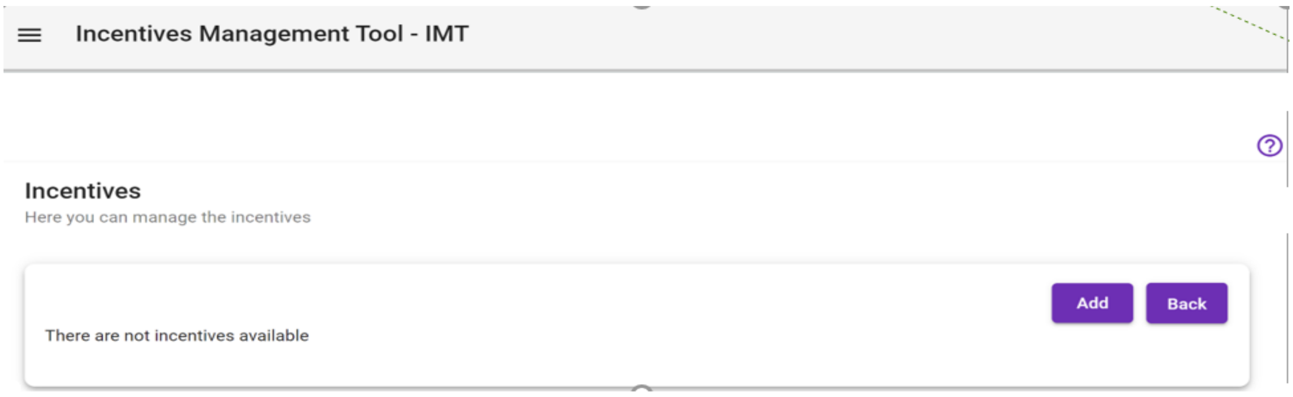


FIGURE 16 – ACCESS TO INCENTIVE MANAGEMENT TOOL OPTIONS

Once selected one option, policy makers will access the Incentives/Motives/Actions/Producers Typology page. For this first prototype, Report option is not yet allowed. Policy makers will then see, a list of the incentives they have created so far, if there's any. If no incentive has been created yet, as in the image bellow, two buttons will appear: Add, that allows the creation of a new Incentive, and Back, to return to the initial page:


Incentives Management Tool - IMT

Incentives

Here you can manage the incentives

There are not incentives available

FIGURE 17 - INITIAL PAGE FOR INCENTIVE'S OPTION

In the case of the previous figure, if there is any incentive created a list with all incentives created would appear, with a new button at the bottom right: Remove. This new button allows to remove/delete selected Incentives.

If the policy maker wants to create a new incentive, the button Add, must be pushed, and a pop-up will appear, where the data needed for this creation will appear. For this first approach, the name of the new incentive and if it is a suggested one, are the needed inputs. The policy maker can select a motive from a combo box:

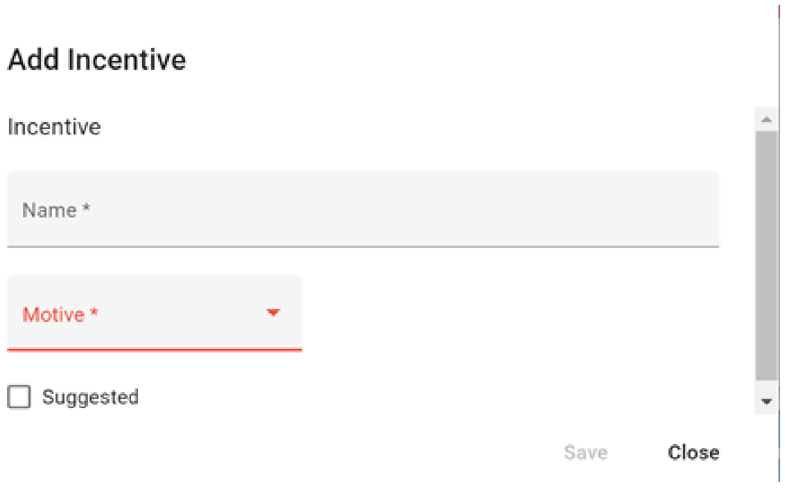


FIGURE 18 - FORM TO CREATE A NEW INCENTIVE

The same happens for Motives and Actions options, although for the creation of an action, the information needed, is larger than the ones for Incentives and Motives:

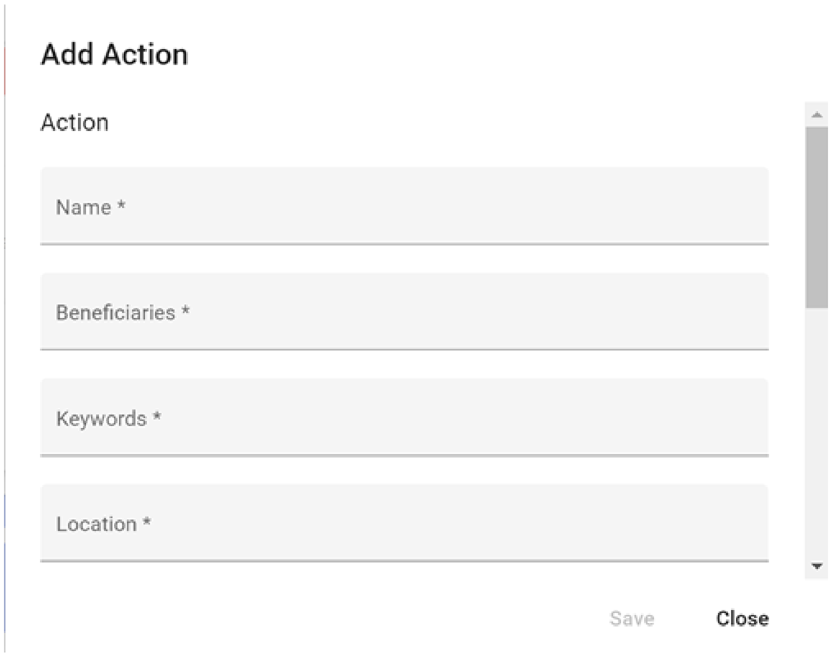


FIGURE 19 - FORM TO CREATE A NEW ACTION

But, as this is a first prototype, the fields needed for the creation of each component can change in future versions, if needed.

In all pages shown to the policy maker, except in the creation forms, it can be seen a help icon (?) in the top right of the page. Clicking it, policy makers can get some information about the actions they can perform at that page. Also, trying to help policy makers in the process of incentive creations, many fields have a tooltip that gives a small definition of them.

- **Back End component**: Aims to give all the support to the Front-End operations, including the communication with the Storage component. The Incentive Management Back End component is independent from the PDT and is responsible to process all the requests done by the Front End, to store information in the storage component, and to retrieve information from the Storage component. All these operations are done through the functions defined in previous section, provided to the Front End as APIs that can be called.
- **The storage component** for the Incentive Managed component will be integrated into the storage component of the PolicyCLOUD platform. For this first prototype the database has been structured as:

action incentive motive policy producertypology report

FIGURE 20 - TABLES CREATED FOR INCENTIVE MANAGEMENT TOOL STORAGE, FIRST PROTOTYPE

This first approach for the Incentive Management Storage component, considers the data model detailed in D3.4 [2], and has 6 tables: action, incentive, motive, policy, report and producertypology.

This is meant to be a first approach to the Incentive Management Storage component, and as such, it can and probably will likely undergo changes in the next release.

In the deliverable D3.4 [2] mentioned already, there was also a component initially thought as “Access to policies” component, which would “allows policy makers to see results from their policies, providing them with insights for the creation of new incentives”. This component finally has not turned out to be a component as such, since at this first prototype, the access to policies is done through a call to an PDT Back End API, done by the Front-End component to access policy makers policies; and insights will be provided by the reports stored in the Storage component, and recovered by the Incentive Management Tool Back End component.

3.3 Interfaces

For the Front-End component, the Interface, as explained in previous sections, will be integrated in the PDT, and the first page can be seen in Figure 16. As mentioned before, its main components are:

- *incentive*-management.services.ts: Composed by:
  - `getIncentivesList(creatorId:number = null)`:Gets list of Incentives from the server.
  - `getIncentiveDataById(incentiveId: number = null)`: Get Incentive by incentiveId from the server.
  - `updateIncentiveDataById(incentiveId: number = null, dataModel: any)`: Update Incentive by incentiveId from the server.
  - `addNewIncentive(dataModel: any)`: Create a new Incentive.
  - `deleteIncentive(incentiveId: number )`: Delete an incentive by incentiveId.
  - `getMotivesList(): Observable<Incentive[]>`:Get list of Motives from the server.
  - `getMotiveDataById(motiveId: number = null)`: Get Motive by motiveId from the server.
  - `updateMotiveDataById(motiveId: number = null, dataModel: any): Observable<any[]>`: Update Motive by motiveId from the server.
  - `addNewMotive(dataModel: any): Observable<any[]>`:Create a new Motive.
  - `deleteMotive(motiveId: number ): Observable<{}>`:Delete a motive by motiveId.
  - `getActionsList(): Observable<Action[]>`:Get list of Actions from the server.
  - `getActionDataById(actionId: number = null)`: Get Action by actionId from the server.
  - `updateActionDataById(actionId: number = null, dataModel: any)`: Observable< any[]>: Update Action by actionId from the server.
  - `addNewAction(dataModel: any): Observable<any[]>`: Create a new Action.
  - `deleteAction(actionId: number ): Observable<{}>`: Delete an action by actionId
  - `getProducerTyplogiesList(): Observable<Action[]>`:Get list of Producer typologies from the server.
  - `getProducerTypologyDataById(producerTypologyId: number = null): Observable<Action>`: Get Producer Typology by producerTypologyId from the server.
  - `updateProducerTypologyDataById(producerTypologyId: number = null, dataModel: any): Observable<any[]>`:Update Producer typology by producerTypologyId from the server.
  - `addNewProducerTypology(dataModel: any): Observable<any[]>`:Create a new Producer Typology.
  - `deleteProducerTypology(producerTypologyId: number ): Observable<{}>`: Delete a producer typology by producerTypologyId.

- *incentives* **component**: Composed by:
  - `addIncentive()`: Open a new window where policy makers will introduce the data needed to create an incentive.

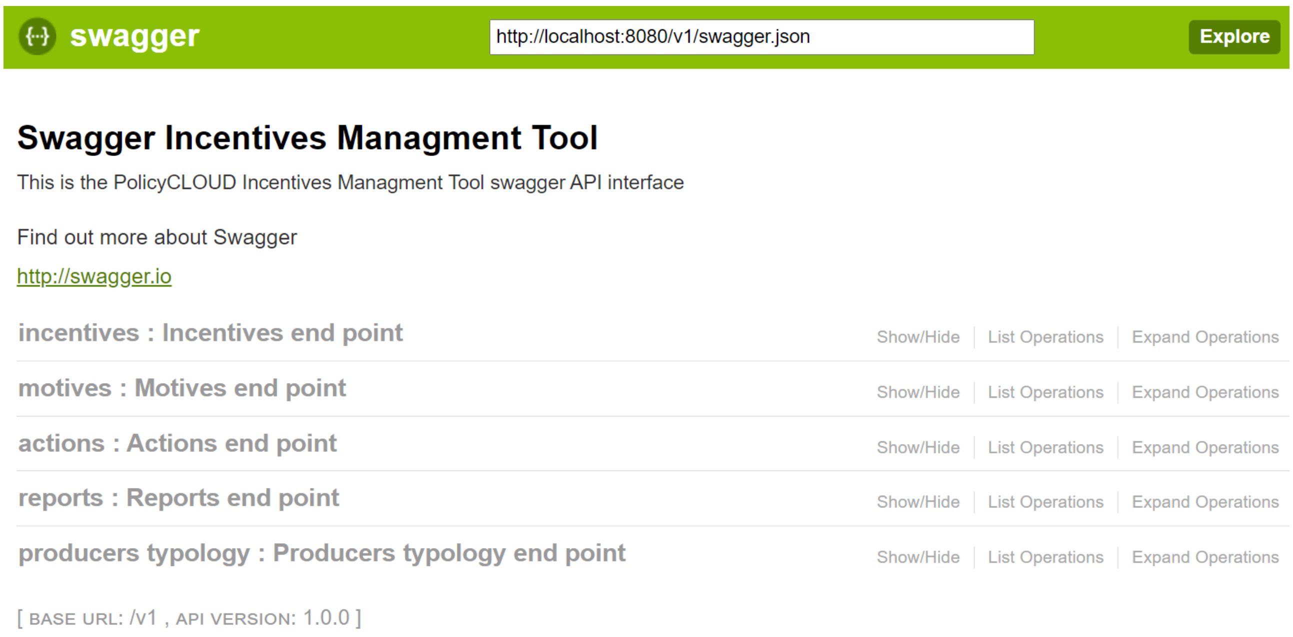
  - `editIncentive(id, element)`: Make changes in an already created incentive.
  - `deleteIncentive(id, element)`: Delete a created incentive.
  - `saveIncentiveForm()`: Creates a new incentive with the data introduced for the policy maker.

- **motives component**: Composed by:
  - `addMotive()`; `editMotive(id, element)`; `deleteMotive(id, element)`; and `saveMotiveForm()` with same functionalities as incentives, but with motives type.

- **actions component**: Composed by:
  - `addAction()`; `editAction(id, element)`; `deleteAction(id, element)`; and `saveActionForm()` with same functionalities, but with actions type.

- **producerstypology component**: Composed by:
  - `addProducerTypology();editPT(id, element);` `deletePT(id,element);` `savePTForm()`with same functionalities, but with actions type

The Back End interface that is provided by a Swagger GUI (https://swagger.io/) can be seen in the following image:



Swagger Incentives Managment Tool

This is the PolicyCLOUD Incentives Managment Tool swagger API interface

Find out more about Swagger

http://swagger.io

incentives : Incentives end point — Show/Hide | List Operations | Expand Operations

motives : Motives end point — Show/Hide | List Operations | Expand Operations

actions : Actions end point — Show/Hide | List Operations | Expand Operations

reports : Reports end point — Show/Hide | List Operations | Expand Operations

producers typology : Producers typology end point — Show/Hide | List Operations | Expand Operations

[BASE URL: /v1 , API VERSION: 1.0.0]

FIGURE 21 - SWAGGER FOR INCENTIVE MANAGEMENT BACK-END COMPONENT

Where all APIs provided by the component can be seen. This APIs provide all the functions described in previous section:

- **incentives_controller.py**: Composed by:
  - `def add_incentive(body):` Add a new incentive to the data base.
  - `def delete_incentive(incentiveId):` Deletes an incentive.
  - `def find_incentives(creatorId=None):` Get all incentives.
  - `def get_incentive_by_id(incentiveId):` Find incentive by ID.
  - `def update_incentive(incentiveId, body):` Updates an incentive by id.
- **actions_controller.py**:, Composed by:
  - `def add_action(body):` Add a new action to the data base.
  - `def delete_action(actionId):` Deletes an action.
  - `def find_actions():` Get all actions.
  - `def get_action_by_id(actionId):` Find action by id.
  - `def update_action(actionId, body):` Updates an action by id.
- **motives_controller.py**: Composed by:
  - `def add_motive(body):` Add a new motive to the data base.
  - `def delete_motive(motiveId):` Deletes a motive.
  - `def find_motives():` Get all motives.
  - `def get_motive_by_id(motiveId):` Find motives by id.
  - `def update_motive(motiveId, body):` Updates an motive by id.
- **reports_controller.py**: Composed by:
  - `def add_report(body):` Add a new report to the data base.
  - `def find_reports():` Get all reports.
  - `def get_report_by_id(motiveId):` Get reports by id.
- **producerstypology_controller.py**: Composed by:
  - `def add_producer_typology(body):` Add a new producer typology to the data base.
  - `def producers_typology():` Get all producer typologies.
  - `def get_producer_typology_by_id(producerTypologyId):` Get producer typology by id.
  - `def update_producer_typology(producerTypologyId, body):`Updates a producer typology in the database with form data.
  - `def delete_producer_typology(producerTypologyId):` Deletes a producer typology.

3.4 Baseline Technologies and Tools

The technologies used for the Incentive Management Tool can be divided in:

- For the Front-End component: the technology used is Angular (https://angular.io/), having in mind the integration with the PDT.
- For the Back End component: the technology chosen is Python (https://www.python.org/).
- For the Storage component: the technology used is MySQL (https://www.mysql.com/), having in mind the integration with the PolicyCLOUD storage, LXC, which is based in MySQL.

3.5 Source Code

3.5.1 Code Overview and Availability

The code for the two different Incentive Management Tool components can be found at:

Front End component: https://registry.grid.ece.ntua.gr/ana.pontual/incentivesmanagement_fe

Back End component: https://registry.grid.ece.ntua.gr/ana.pontual/incentivesmanagement_be

Access is restricted to members of the project consortium.

3.6 Deployment Status

The functionalities described for the Incentive Management Tool have been deployed restricted to the project environment, that is, private. However, in the git repository instructions and files are provided that allow components deployment whenever and wherever needed.

For this first prototype of the Incentive Management Tool, deployments have been done locally, for Incentive Management Front End, Back End and Storage components.

For the Front-End component, once its integration with the PDT has finish, deployments will take place together and in the same way as the PDT.

For the Back End component, although the definition of where it will be deployed is still a pending issue, for future releases it can be deployed automatically through Docker, whenever and wherever decided.

The Storage component will be deployed inside PolicyCLOUD storage, throw LXC. Works on this direction has already been started and will be completed in future releases.

4 Data Governance Model and Privacy Enforcement mechanism

4.1 Prototype Overview

The prototype's purpose is to demonstrate the ability of the ABAC Engine to intercept a request, obtain the attributes of the requestor and evaluate whether the request should be permitted, based on those attributes and the enforced Policies. The two main components responsible for this functionality are the ABAC Server and the ABAC Client Filter, while the prototype also contains a simple Web Client for testing purposes.

Keycloak [13] is selected as the Attribute Provider and User Authenticator for this version of the prototype. In the upcoming months, we will examine the integration of our solution as part of the integrated PolicyCLOUD platform and alternate authentication solutions that could be used.

The Data model is in continuous discussion with the pilots in order to fit their needs and requirements as they evolve through the maturity of the project. The current prototype that can be used to display the proper function of the mechanism and then integrate the refined Data Model with subsequent versions with all the necessary changes that might arise.

4.2 Main Components of the Prototype

4.2.1 ABAC Server

The ABAC server is the backbone of the ABAC Authorization Engine and is responsible for evaluating the access requests authorization. It communicates with the ABAC Client to retrieve these requests and responds with a Permit message based on the Policies currently applied and the attributes of the requestor. To set up the ABAC Server locally, Maven is required for an initial

- mvn clean install

For the first installation there would be no keystore and trustsore files present so they should be created for during the initialization by executing:

- keytool -genkey -alias pdp-server -keyalg RSA -keysize 2048 -storetype PKCS12 -storepass <YOUR_KEYSTORE_PASSWORD> -dname "CN=pdp-server,OU=Information Management Unit (IMU),O=Institute of Communication and Computer Systems (ICCS),L=Athens,ST=Attika,C=GR" -ext "SAN=dns:pdp-server,dns:localhost,ip:127.0.0.1" -keystore pdp-server-keystore.p12 -startdate -1d -validity 3650
- keytool -list -v -storetype pkcs12 -keystore pdp-server-keystore.p12

- keytool -export -storetype PKCS12 -keystore pdp-server-keystore.p12 -storepass <YOUR_KEYSTORE_PASSWORD> -alias pdp-server -file pdp-server.crt
- keytool -printcert -file pdp-server.crt
- keytool -import -alias pdp-server -file pdp-server.crt -storetype PKCS12 -keystore pdp-server-truststore.p12 -storepass<YOUR_TRUST_STORE_PASSWORD>
- keytool -list -v -storetype pkcs12 -keystore pdp-server-truststore.p12

When trying to run the server, the Configuration directory must be selected and exported via:

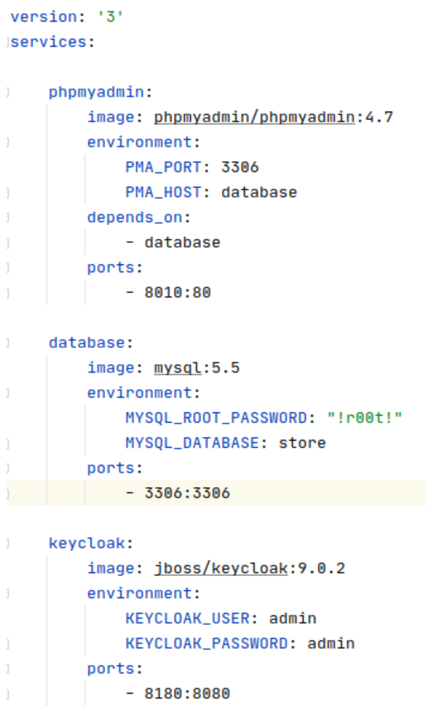
- export CONFIG_DIR=src/main/resources/config

Finally, the jar produced by the mvn clean install is executed via:

- java -jar target/abac-authorization-server.jar

4.2.2 KeyCloak

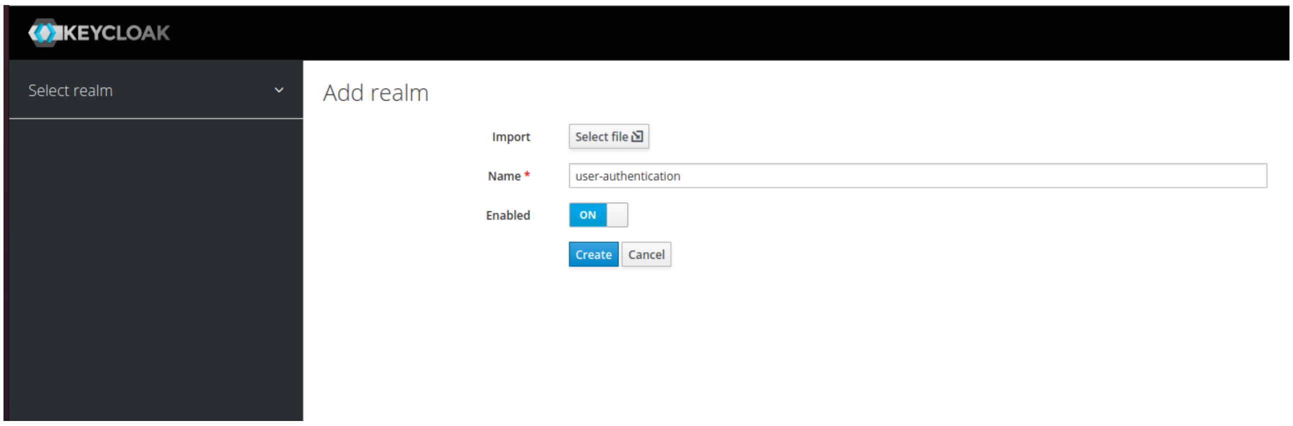
A dockerised version of Keycloak is used as the Attribute Provider and User Authenticator. The image selected is jboss/keycloak and the admin username and password, as well as the ports of the docker container are selected in the sample docker-compose.yaml provided below:



```yaml
version: '3'
services:

    phpmyadmin:
        image: phpmyadmin/phpmyadmin:4.7
        environment:
            PMA_PORT: 3306
            PMA_HOST: database
        depends_on:
            - database
        ports:
            - 8010:80

    database:
        image: mysql:5.5
        environment:
            MYSQL_ROOT_PASSWORD: "!r00t!"
            MYSQL_DATABASE: store
        ports:
            - 3306:3306

    keycloak:
        image: jboss/keycloak:9.0.2
        environment:
            KEYCLOAK_USER: admin
            KEYCLOAK_PASSWORD: admin
        ports:
            - 8180:8080
```

FIGURE 22 - DOCKER-COMPOSE.YAML FILE

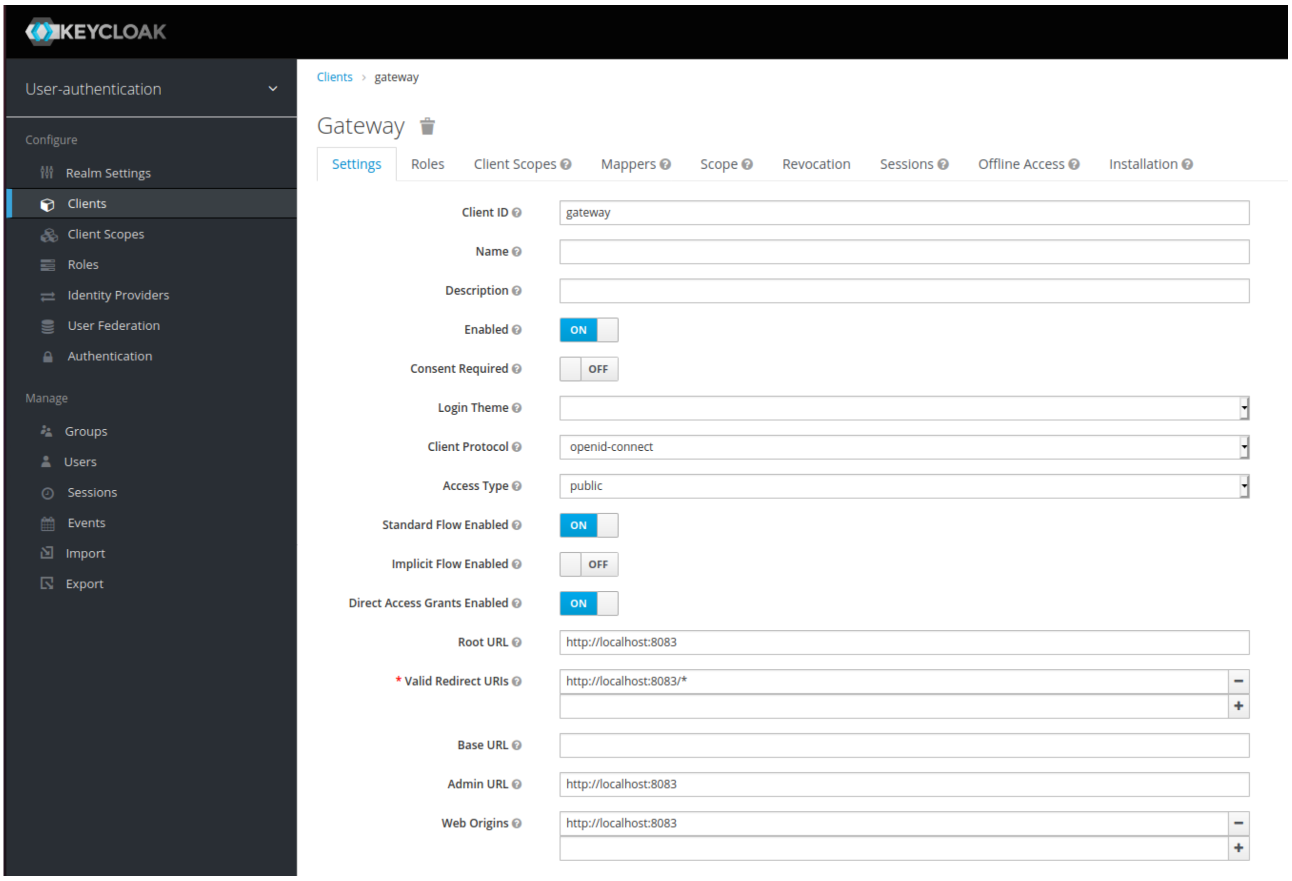
Along with Keycloak, a simple mysql database and mhpmyadmin are installed with basic settings. The Keycloak Admin Console can be accessed by visiting localhost:8180 and providing the login credentials specified in Figure 22. The necessary steps for setting up Keycloak to act as a User Authentication provider involve firstly creating the relevant realm. A Keycloak realm manages a set of users, credentials, roles, and groups. A user belongs to and logs into a realm and realms are isolated from one another and can only manage and authenticate the users that they control. It is important that the name of the realm matches the one used in the application.yml of the service we are trying to secure.



Import	Select file
Name *	user-authentication
Enabled	ON

FIGURE 23 - KEYCLOAK REALM CREATION

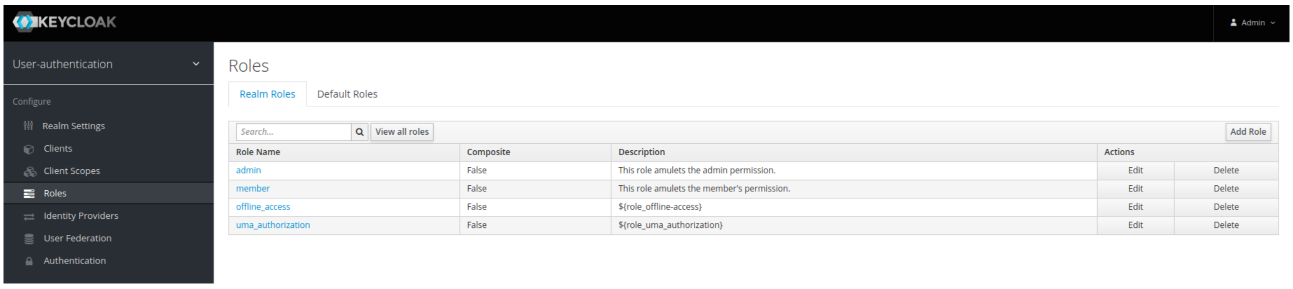
The next step is the creation of a Client inside our newly created Realm. In Keycloak, clients are entities that can request Keycloak to authenticate a user. As mentioned before, it is also important here that the new Client is named appropriately compared to the "resource" attribute in the application.yml of the service we are trying to secure.



Client ID	gateway
Name	
Description	
Enabled	ON
Consent Required	OFF
Login Theme	
Client Protocol	openid-connect
Access Type	public
Standard Flow Enabled	ON
Implicit Flow Enabled	OFF
Direct Access Grants Enabled	ON
Root URL	http://localhost:8083
* Valid Redirect URIs	http://localhost:8083/*
Base URL	
Admin URL	http://localhost:8083
Web Origins	http://localhost:8083

FIGURE 24 - KEYCLOAK CLIENT CREATION

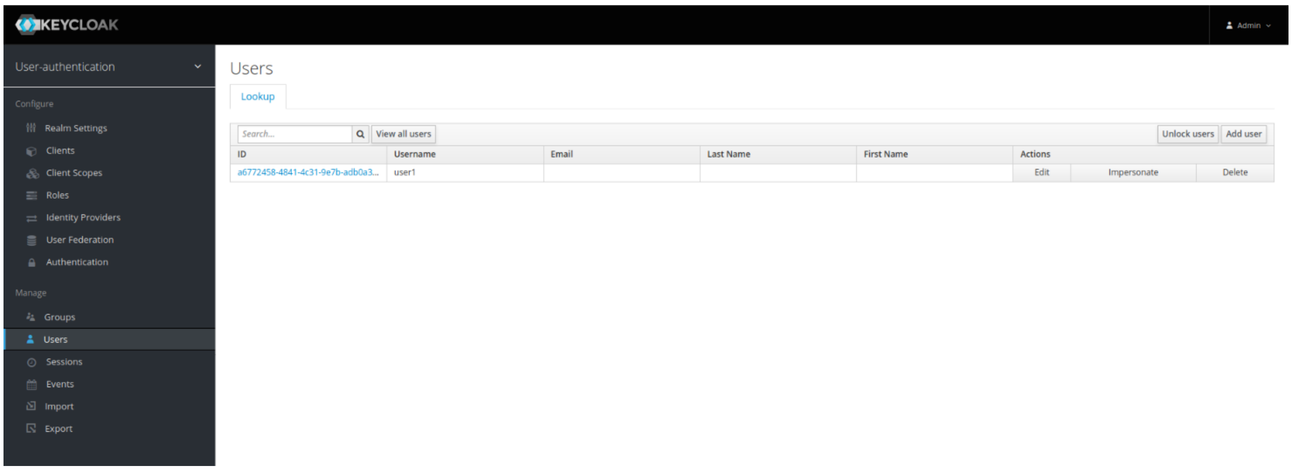
It is important to ensure that “Standard Flow Enabled” is selected as this enables standard OpenID Connect redirect-based authentication with authorization code. More specifically, in terms of OpenID Connect or OAuth2 specifications, this enables support of “Authorization Code Flow” [14] for this client. The “Valid Redirect URIs field must contain the location of the service we are trying to secure. Moving on, relevant roles for the created realm are defined from the Admin Console as shown in Figure 25.



Role Name	Composite	Description	Actions	
admin	False	This role amulets the admin permission.	Edit	Delete
member	False	This role amulets the member's permission.	Edit	Delete
offline_access	False	${role_offline-access}	Edit	Delete
uma_authorization	False	${role_uma_authorization}	Edit	Delete

FIGURE 25 - KEYCLOAK REALM ROLES

User creation can also be handled by the Admin Console and for the purpose of this prototype a sample user with username “user1” is created.



ID	Username	Email	Last Name	First Name	Actions		
a6772458-4841-4c31-9e7b-adb0a3...	user1				Edit	Impersonate	Delete

FIGURE 26 - KEYCLOAK USERS

At first the newly created user is not assigned the admin role but is given the member, offline_access and uma_authorization roles. This default behaviour can be changed through Keycloak settings accordingly for newly created or registered users.

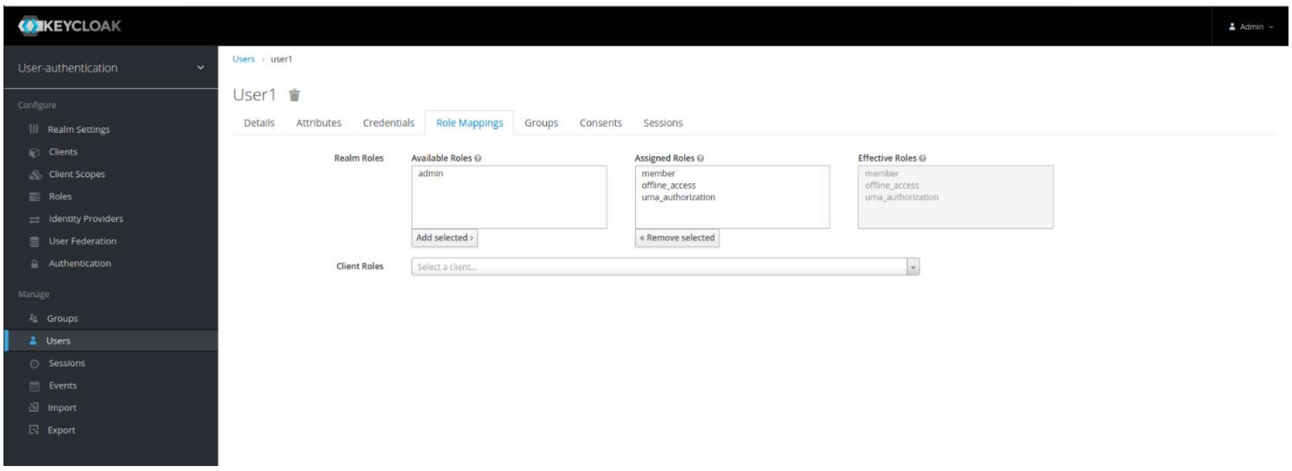


FIGURE 27 - INITIAL ROLE MAPPINGS

As shown in Figure 27, the Role Mappings Tab provides an overview of all the assigned Roles for this particular user. Roles can be added or removed accordingly and furthermore there is the option to assign client-specific roles via the relevant dropdown menu.

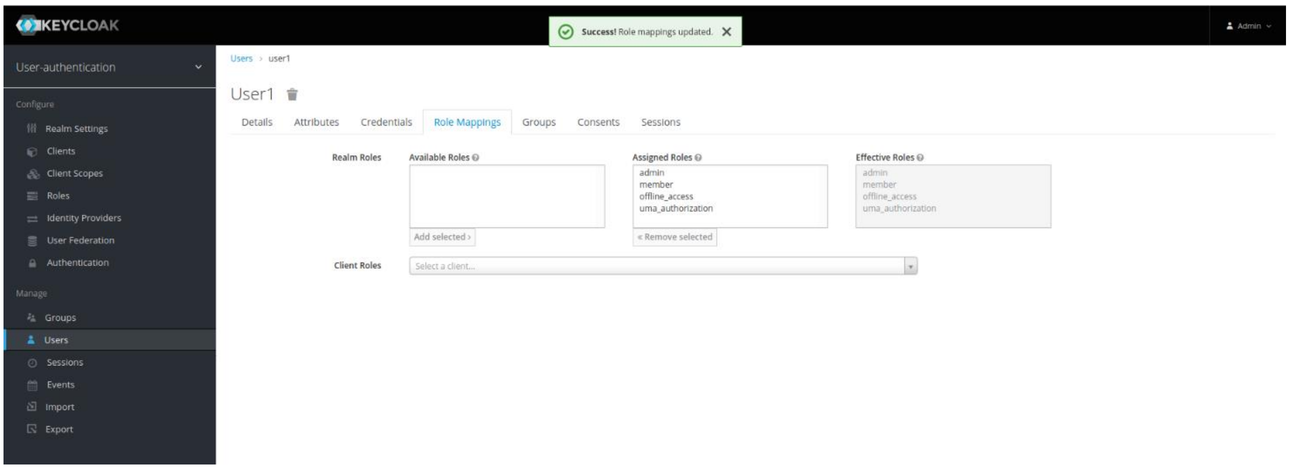
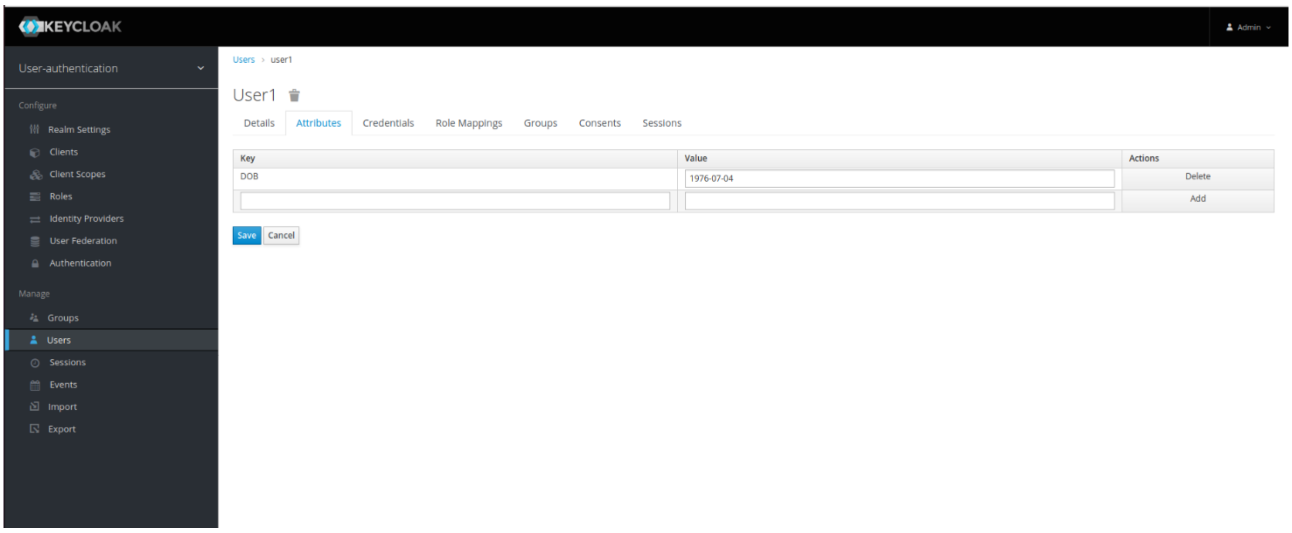


FIGURE 28 – ADMIN ROLE GRANT

Figure 28 presents an example of the admin Role assignment to our test user. Information about users in Keycloak is not restricted to Roles and Credentials, as each User can contain custom Attributes to further define their right in the PolicyCLOUD ecosystem. Figure 29 shows such an example with the Date of Birth of the user added as a custom Attribute from the relevant tab.

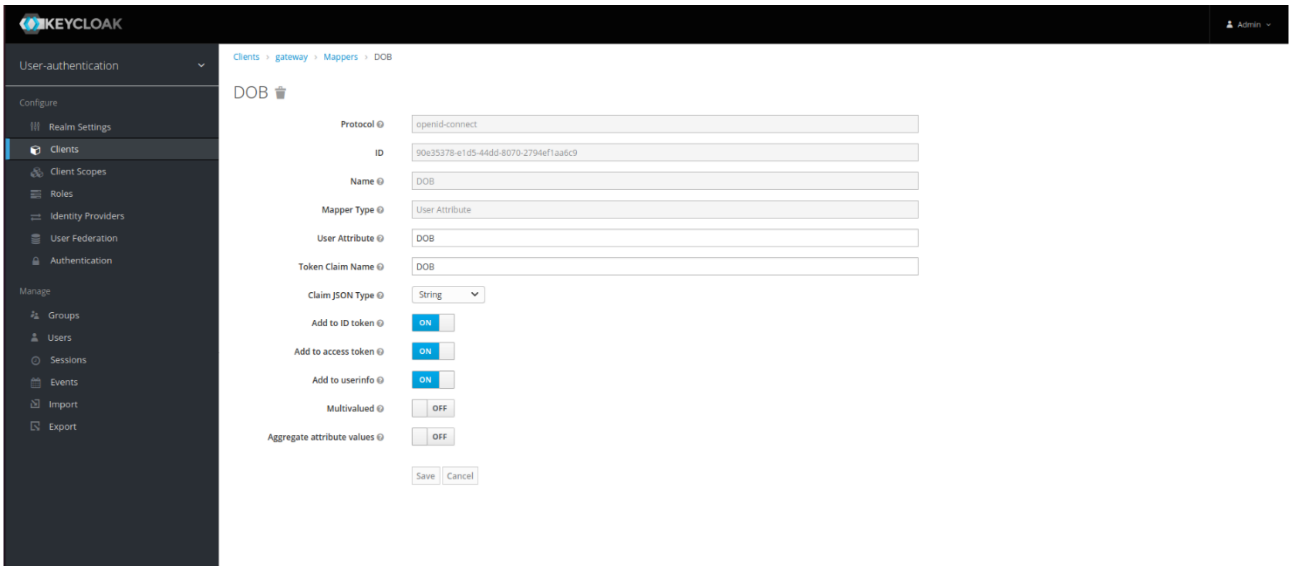


User1

Key	Value	Actions
DOB	1976-07-04	Delete
		Add

FIGURE 29 - CUSTOM USER ATTRIBUTE

These custom attributes are key for the PolicyCLOUD's ecosystem ability to create and manage complex Policies that contain multiple parameters for each User. It is essential therefore that Keycloak can communicate these attributes in a safe and secure way. This can be achieved via Client Mappers in the Keycloak Admin Console, as shown in Figure 30.



DOB

Protocol	openid-connect
ID	90e35378-e1d5-44dd-8070-2794ef1aa6c9
Name	DOB
Mapper Type	User Attribute
User Attribute	DOB
Token Claim Name	DOB
Claim JSON Type	String
Add to ID token	ON
Add to access token	ON
Add to userinfo	ON
Multivalued	OFF
Aggregate attribute values	OFF

FIGURE 30 - CLIENT ATTRIBUTE MAPPER

There are multiple options available for the Mapper Type but our Use Case dictates that we use the "User Attribute". The "Name" and "User Attribute" fields must match the name of the corresponding User Attribute from Figure 29, while the "Claim JSON Type" dictates the type of the mapped value. By selecting "Add to ID token", "Add to access token", "Add to userinfo" the attribute value is added as a claim to the relevant token. These can be used by the ABAC Client and Server in order to securely transmit the claims to the ABAC Engine.

4.2.3 ABAC Client Filter

The ABAC Client Filter can be included in a web client and trigger the necessary interception to the ABAC Authorization Engine. It depends on the ABAC Client and acts as an access filter to a specific API or website and restricts access until it receives authorization from the ABAC Engine. Furthermore, it handles the IDToken provided by Keycloak with the custom User Attributes needed for the Policy Evaluation and Enforcement. To include the Filter in a web client the relevant dependency must be added in the pom.xml

```
<dependencies>
  <dependency>
    <groupId>eu.policycloud.authorization.abac</groupId>
    <artifactId>abac-authorization-client</artifactId>
    <version>1.0.0-SNAPSHOT</version>
  </dependency>
</dependencies>
```

The next necessary step is to copy the truststore file of ABAC server into the application's resources directory. Take care not to expose this file publicly.

- Cp abac-authorization/server/src/main/resources/config/pdp-server-truststore.p12 <YOUR_APP_HOME> /src/main/resources/truststore-client.p12

The final step is to include the appropriate variables, either through a .env file or script

```
AZ_CLIENT_TRUST_STORE_FILE=truststore-client.p12
AZ_CLIENT_TRUST_STORE_TYPE=PKCS12
AZ_CLIENT_TRUST_STORE_PASSWORD=YOUR_PASSWORD_HERE
AZ_SERVER_ENDPOINTS=https://localhost:7071/checkJsonAccessRequest
AZ_SERVER_ACCESS_KEY=723568712658723154532175675265723123321765723
AZ_CALL_DISABLED=false
AZ_CALL_LOAD_BALANCE_METHOD=ORDER
AZ_CALL_RETRIES=1
```

```
export AZ_CLIENT_TRUST_STORE_FILE AZ_CLIENT_TRUST_STORE_TYPE
AZ_CLIENT_TRUST_STORE_PASSWORD AZ_SERVER_ENDPOINTS AZ_SERVER_ACCESS_KEY
AZ_CALL_DISABLED AZ_CALL_LOAD_BALANCE_METHOD AZ_CALL_RETRIES
```

If you want to change the API Key for both the server and client:

Create a long, difficult to guess (random) string. We suggest more than 32 characters long, including any combination of capital and plain letters, numbers and symbols. Avoid using phrases or values that can be guessed or extracted from context.

New API Key value must be set in both ABAC Server and ABAC Client configuration files.

- For ABAC Server, edit authorization-server.properties file and set property pdp.access-key.

- For ABAC Client, edit authorization-client.properties file and set property pdp.access-key or change the corresponding environment variable AZ_SERVER_ACCESS_KEY.

4.2.4 ABAC Proxy

There can be cases where applying the ABAC Client filter mentioned in Section 4.2.3 is not possible due to limitations of the programming language or the technology stack used. That is the case with programs utilizing PHP and in order to integrate ABAC in those cases, the ABAC Proxy has been developed. It is based on the Zuul Proxy [15] and ensures that traffic coming to a specified port are checked against the currently implemented policy and if the conditions are satisfied, allows for the redirection to the underlying php application. In order to configure the proxy for a specific php application, the following values of its application.yml file should be set:

```yaml
server:
  port: 80

proxy.rewrite.redirect_url: true

zuul:
  addHostHeader: true
  routes:
    phpapp:
      path: /php/**
      url: http://php-app:8080/
      sensitiveHeaders:
    all:
      path: /**
      url: http://pdp-server:7071/
      sensitiveHeaders:

ribbon:
  eureka:
    enabled: false
```

FIGURE 31 – ABAC PROXY APPLICATION.YML

The above file redirects all traffic to the php-app running locally, if the requests successfully pass through the ABAC PDP server located at http://pdp-server:7071. The routes section can be enhanced with more entries to fine-grain the needs of any application and allow for more specific routing to the php application.

4.2.5 Test Web Client

A simple Web Client can be used in order to test both the ABAC Server and Filter. The Web Client is a Spring-boot application that contains an ABAC Authorization Filter that is responsible for intercepting any access to the secured API's and instead redirects the user to login to Keycloak. Based on the User Attributes retrieved from the Keycloak login, as well as the implemented Policies evaluated at the ABAC Server, the request is either denied or permitted. An example of the aforementioned login page during the interception of a request to get the user's Date of Birth is shown on Figure 32.

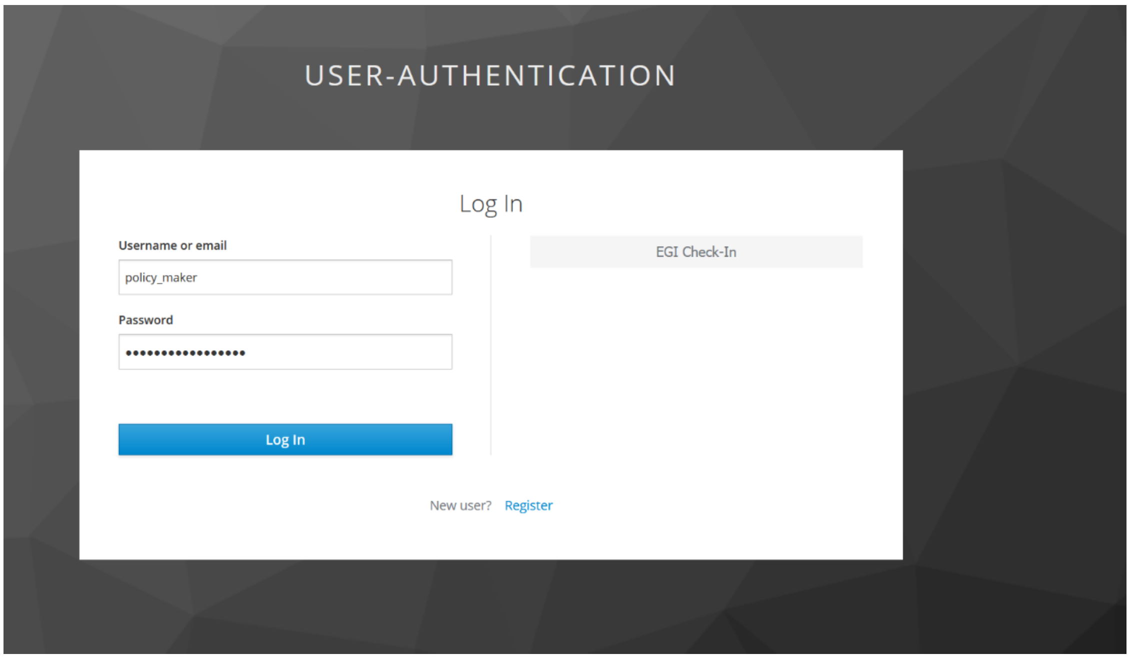
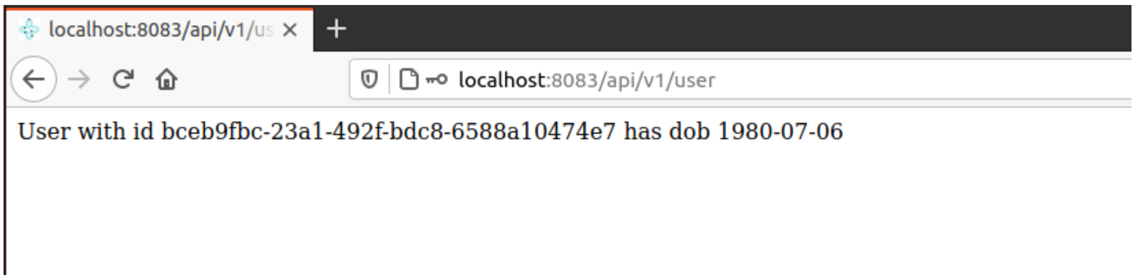


FIGURE 32 - INTERCEPT LOGIN

If the user login is successful and the ABAC Engine permits the request, based on the applied Policies, the Web Client is able to retrieve the userID and Date of Birth, as shown on Figure 33.

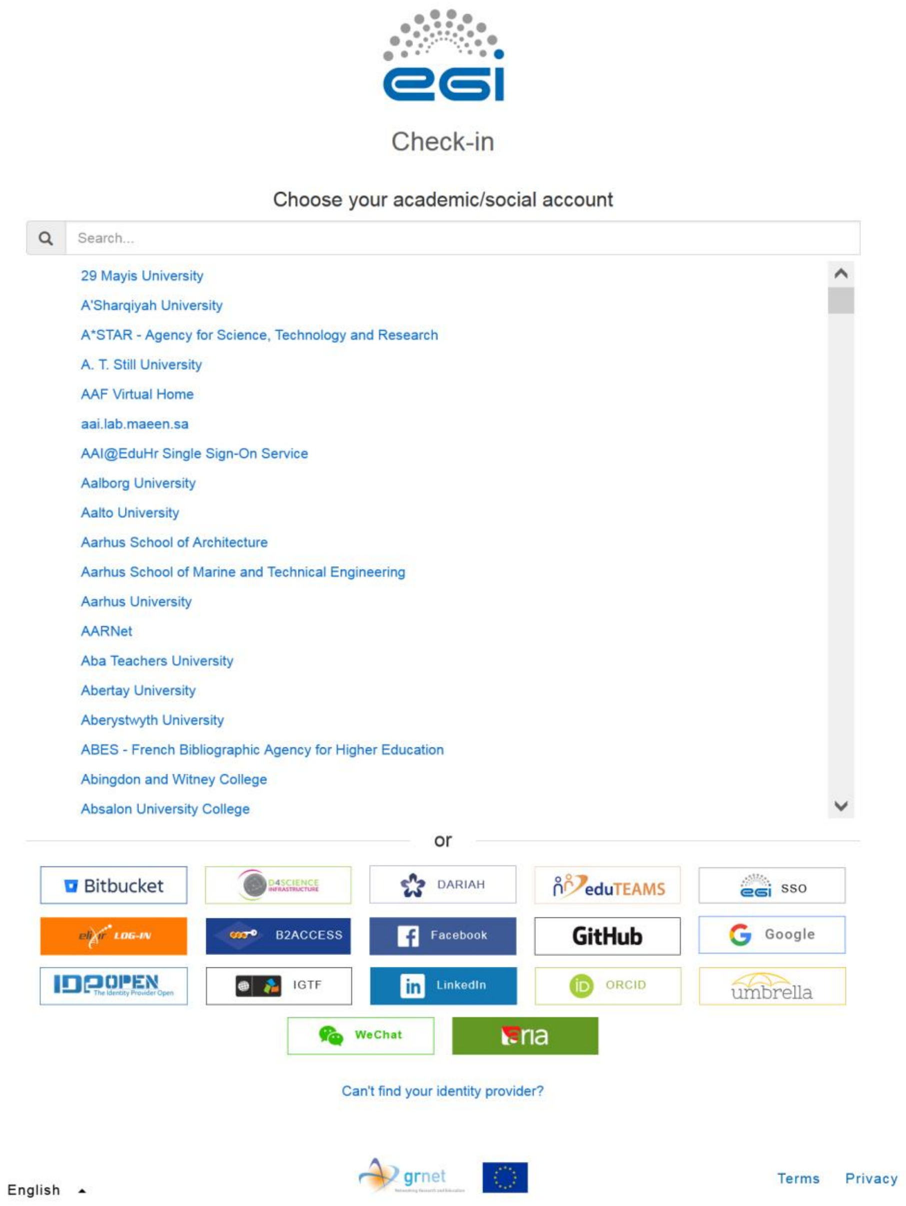


```
User with id bceb9fbc-23a1-492f-bdc8-6588a10474e7 has dob 1980-07-06
```

FIGURE 33 - SUCCESSFUL ATTRIBUTES RETRIEVAL

4.2.6 EGI Check-In integration

The need for managing user identity across borders between organizations, different domains and services, leads to the creation of federated identity environments. An Identity federation is a group of Identity and Service Providers that sign up to an agreed set of policies for exchanging information about users and resources to enable access to and use of the resources. Home organizations (e.g. a university, research institute, etc.), who operate an Identity Provider (IdP), register users by assigning a digital identity -- in this way, they are able to authenticate their users and provide a limited set of attributes that characterize the user in a given context. Service Providers (SPs) delegate the authentication to IdP, in order to control access to the provided resources. EGI Check-In [16] (a service provided by EGI Foundation) is a solution for federated identity management with the architecture of a proxy that operates as a central hub to connect federated Identity Providers and Service Providers.
In the PolicyCLOUD scenario, EGI Check-in proxy is provided like an alternative login button in the Keycloak service authentication page: through EGI Check-In users will be able to use their eduGAIN [17] credentials (institutional or academic accounts) as well as ORCID [18] credentials (researcher identifier that disambiguates researchers and their work) as well as most popular social media accounts (Google, LinkedIn, etc.) allowing PolicyCLOUD services to be easily accessible and adopted by a broader audience, while also maintaining the ability to have also internal users registered to the Keycloak service. The Keycloak Log In page shown on Figure 32 contains a link on the right side with the EGI Check-In label. This provides an alternative method of logging in and redirects the user to the integrated EGI Check-In page, as shown on Figure 34.


Check-in

Choose your academic/social account

Search...

29 Mayis University
A'Sharqiyah University
A*STAR - Agency for Science, Technology and Research
A. T. Still University
AAF Virtual Home
aai.lab.maeen.sa
AAI@EduHr Single Sign-On Service
Aalborg University
Aalto University
Aarhus School of Architecture
Aarhus School of Marine and Technical Engineering
Aarhus University
AARNet
Aba Teachers University
Abertay University
Aberystwyth University
ABES - French Bibliographic Agency for Higher Education
Abingdon and Witney College
Absalon University College

or

Can't find your identity provider?

English

Terms Privacy

FIGURE 34 - INTEGRATED EGI CHECK-IN

A user can select its Identity Provider from that page and input his/her credentials from the Identity Provider login page as usual. This allows users who are not already registered to the PolicyCLOUD ecosystem and Keycloak to use some of the provided tools, using their academic accounts or accounts from popular providers like Google, Facebook, LinkedIn, etc. This integration with EGI Check-In allows PolicyCLOUD to be easily accessible and adopted by a broader audience, while also maintaining the ability to have internal users registered to the Keycloak Server.

4.3 Interfaces

The following APIs are provided by Keycloak and are the most commonly used ones for requesting access from the component, retrieving user attributes and managing the server.

The following APIs are provided by the Cloud Backbone component.

Method	Path	Description
POST	{realm}/protocol/openid-connect/token	Token retrieval for user or service
PUT	/{realm}/users/{id}	User Update
DELETE	/{realm}/users/{id}	User Deletion
POST	/{realm}/users	Create User and set user attributes
GET	{realm}/users?username={username}	Retrieve user attributes by username
GET	{realm}/users/{id}	Retrieve user attributes by user-id
GET	/{realm}/users	Retrieve all realm users
PUT	/{realm}/users/{id}/reset-password	Reset the user password
GET	/{realm}/users/{id}/role-mappings/clients /{client}/available	Retrieve roles that can be mapped to the user
GET	/{realm}/clients/{id}/roles	Retrieve all roles of client
GET	/{realm}/roles	Retrieve all roles of realm
GET	/{realm}/users/{id}/role-mappings/realm	Get realm level role mapping

TABLE 1 - KEYCLOAK COMMON APIS

All the APIs provided by the Keycloak ADMIN REST API [19] are also available to be used, with the retrieval of a valid Keycloak Admin token.

4.4 Baseline Technologies and Tools

4.4.1 Balana

Balana [20] was the first open-source reference implementation of the XACML protocol and is a widely adopted solution. It supports the entire lifecycle of authorization processing. It is tightly integrated into the WSO2 Identity Server [21]. Balana, as XACML engine of the WSO2 Identity Server has two major components, the Policy Administration Point (PAP) and Policy Decision Point (PDP). Figure 34 presented the component architecture of the PDP that is our main interest.

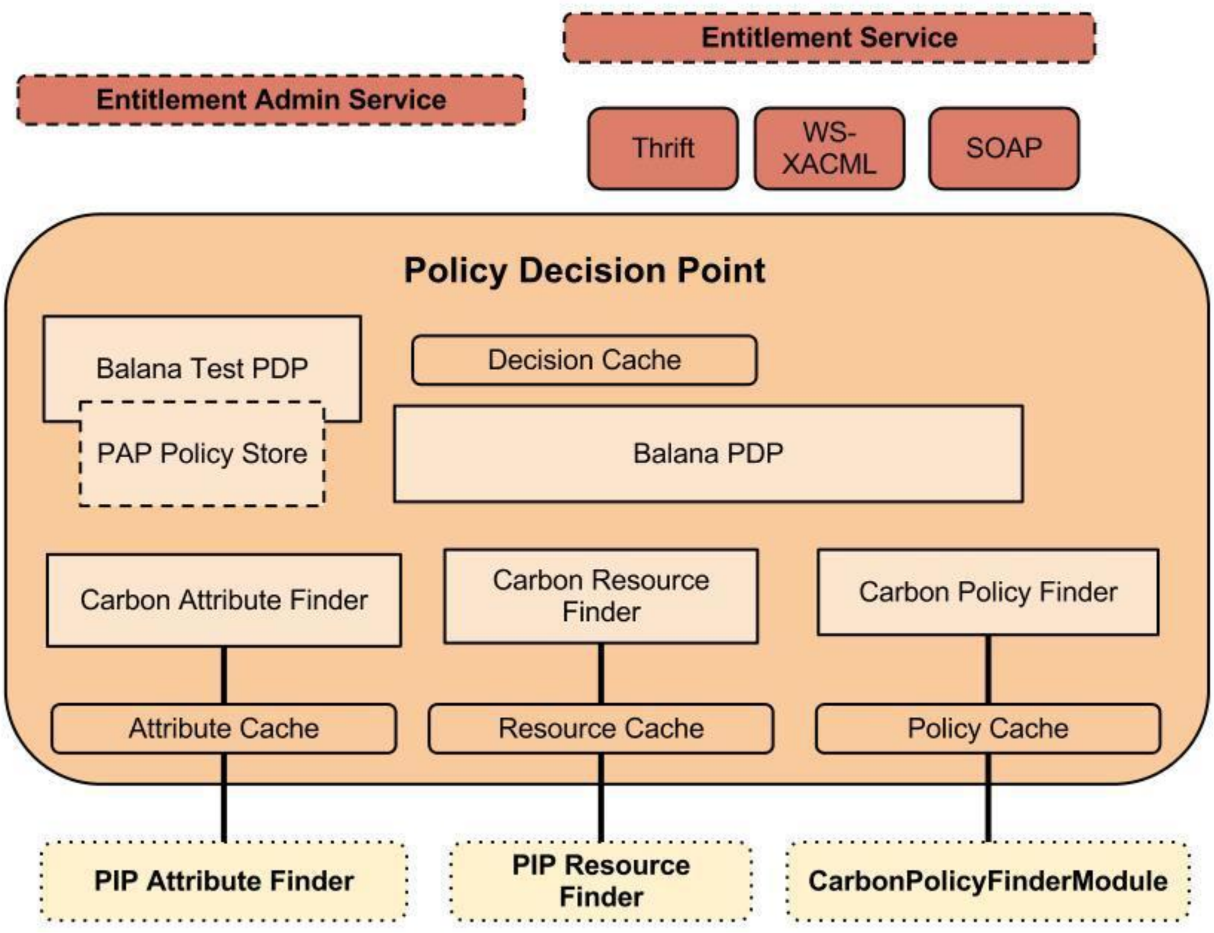


FIGURE 35 - BALANA PDP

More details on the components of in the PDP architecture are presented below.

Entitlement Admin Service provides an API that is used to expose all PDP configurations, such as:

- Invalidating caches
- Refreshing policy, attribute, resource finder modules
- Retrieving PDP configurations
- Testing the PDP

Entitlement Service provides XACML authorization API that supports the following three communication methods with PEP.

- SOAP-based Web service
- Apache Thrift binary protocol [22]
- WS-XACML

Balana PDP is the core of the engine of Balana

Balana Test PDP is a duplication of Balana PDP can be only used for testing policies.

Carbon Policy Finder is a module that finds policies from different policy stores to evaluate an XACML request. Figure 36 presents a high-level diagram of the usage of the carbon policy filter for the collection of the policies to be evaluated.

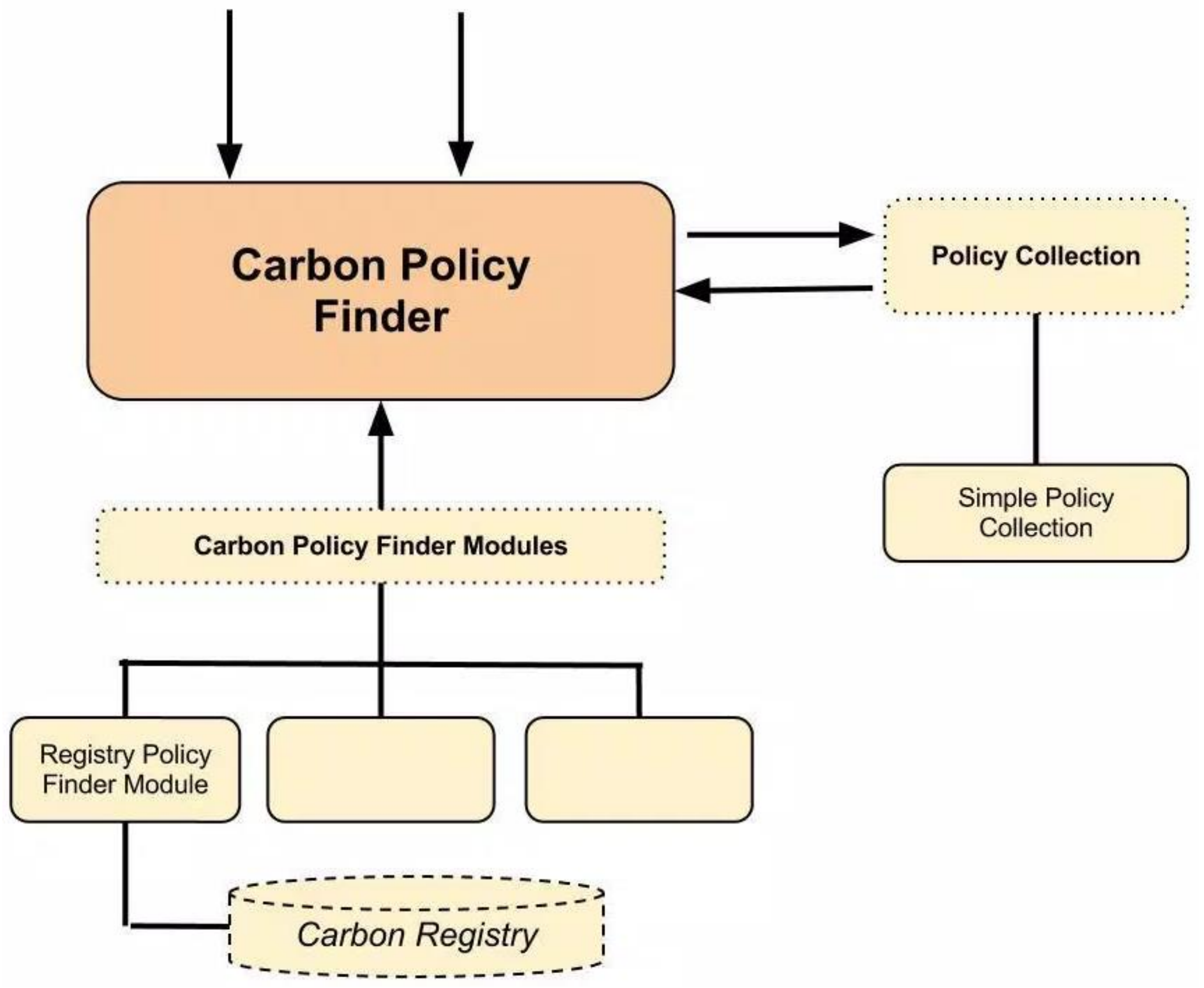


FIGURE 36 - CARBON POLICY FILTER

Policy finder modules implementing the CarbonPolicyFinderModule interface should be registered and plugged with the Carbon policy finder. WSO2 Identity Server provides by default a Carbon registry-based policy finder module that can retrieve policies from a registry collection. Carbon policy finder finds XACML requests and creates an effective policy. When an update in the policy store happens, Carbon policy finder can be re-initialized automatically by the module, or it can be re-initialized using the API of the Entitlement Admin Service.

Carbon Attribute Finder is a module that is responsible for finding missing attributes for a given XACML request, using the underlying PIP attribute finders. Figure 37 provides a high-level diagram for both the Carbon attribute finder and resource finders.

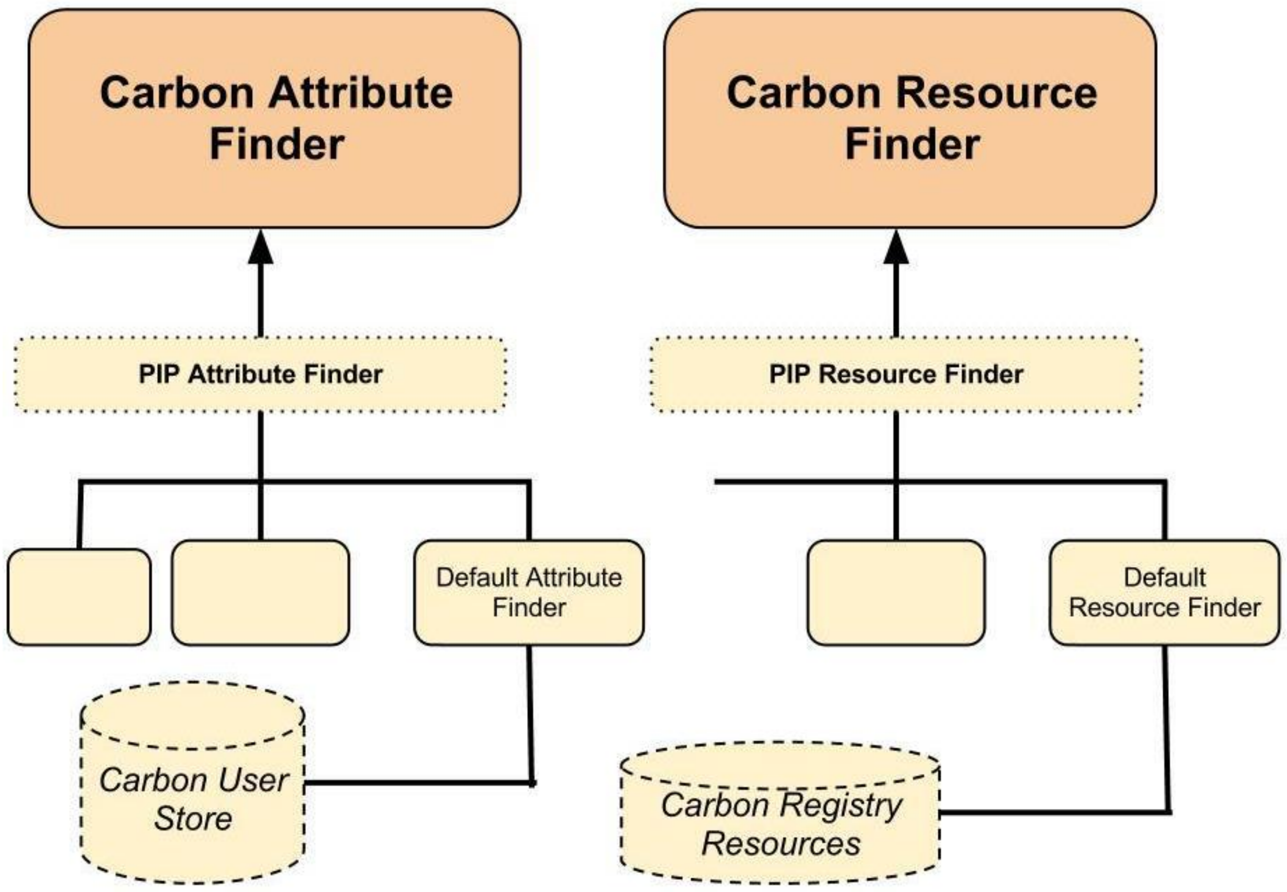


FIGURE 37 - CARBON ATTRIBUTE FINDER

A PIP attribute finder module should implement the PIPAttributeFinder interface, and register it using the entitlement properties configuration file to the Carbon attribute finder. WSO2 Identity Server by default communicates with the underlying user store of the Identity Server that is built with ApacheDS [23].

On runtime, Carbon attribute finder checks for the attribute Id and hands it over to the proper module to handle, while caching mechanism (provided by Carbon attribute finder) is used for caching the findings when possible.

Carbon Resource Finder is used to retrieve children or descendant resources of a given root level resource value, used to fulfil requirements for a multiple decision profile. Similarly to the PIP attribute finder module, it has to implement the PIPAttributeFinder interface.

In general, we consider Balana a highly extensive open-source solution, suitable for the needs of PolicyCLOUD.

4.4.2 Keycloak

Keycloak [13] is probably the most **powerful authentication proxy for micro-services** and legacy systems. As such, it **abstracts the functionality of identity extraction and identity verification** for different systems and for different protocols. In parallel, it is able to map users and roles from existing legacy systems in what it calls **authentication realms**. Through configured realms, Keycloak is able to centralize the login-process of various systems through the implementation of many protocols such as oAuth2.0 [24] and OpenIDConnect [25] (a.k.a. OIDC). The OpenIDConnect signalling is presented in Figure 38.

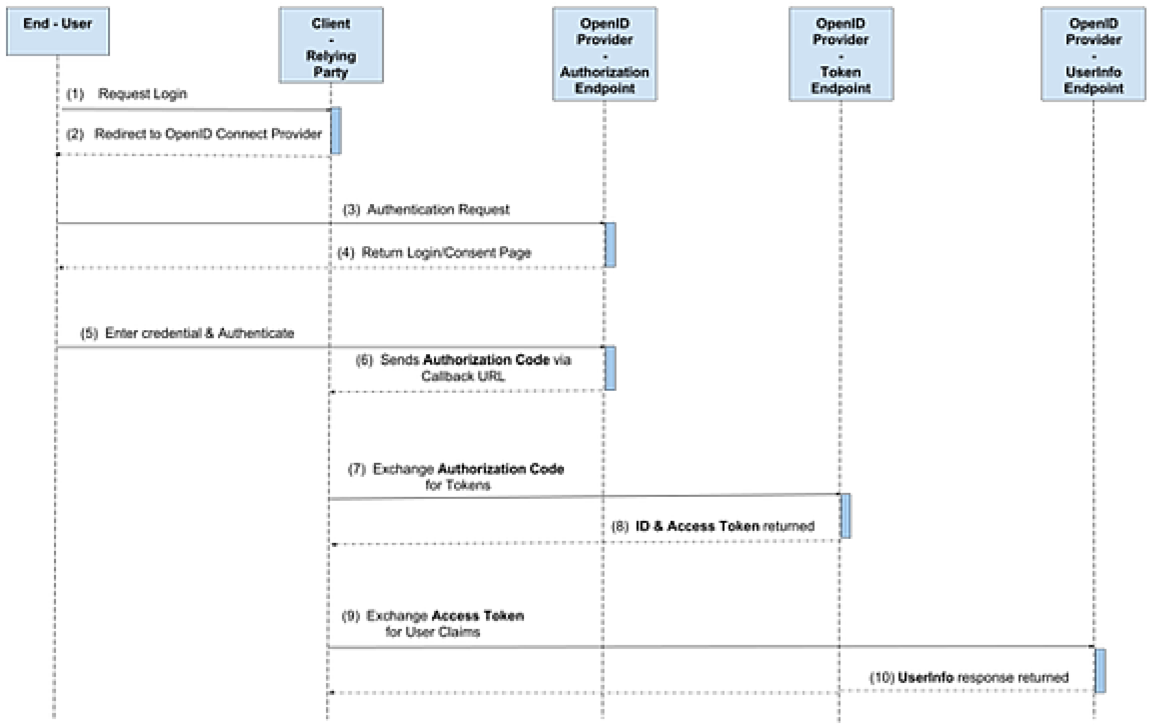

End - User
Client - Relying Party
OpenID Provider - Authorization Endpoint
OpenID Provider - Token Endpoint
OpenID Provider - UserInfo Endpoint
(1) Request Login
(2) Redirect to OpenID Connect Provider
(3) Authentication Request
(4) Return Login/Consent Page
(5) Enter credential & Authenticate
(6) Sends Authorization Code via Callback URL
(7) Exchange Authorization Code for Tokens
(8) ID & Access Token returned
(9) Exchange Access Token for User Claims
(10) UserInfo response returned

FIGURE 38 - OIDC SIGNALLING

According to the flow diagram, a user is attempting to connect to a service which supports OIDC. The health care service is redirecting the user to an OIDC provider that is configured to authenticate users based on **a service-id and a user-role mapping**. The **combination** of the service-id and the user-role-mapping is addressed as a **realm**.

The OIDC server is “challenging” the user to authenticate based on various methods (username/password, X509 certificate etc.). Our software prototype utilizes the username/password method and upon successful login a distinct set of claims are serialized as a token back to the user in order to use it in his/her interaction with the service. **These claims contain electronically signed attributes that can be used by the ABAC authorization engine**.

As a result, the OIDC signalling (and Keycloak in general) is extremely crucial for PolicyCLOUD ABAC even if it is not an ABAC engine per se. It acts as an **enabler of verifier attributes and attribute provider**.

4.5 Source Code

4.5.1 Code Overview and Availability

The docker image for the ABAC Server as well as the image and the code for the ABAC Proxy can be found at https://registry.grid.ece.ntua.gr/oikonomou/abac_proxy.

Access is restricted to members of the project consortium.

4.6 Deployment Status

The Keycloak Server containing the realms, clients and users of the PolicyCLOUD ecosystem is deployed at https://policycloud-auth.euprojects.net. The ABAC Server has been deployed together with the Gateways it protects on a public VPS server in project’s cloud infrastructure that has been provided and is supported by RECAS-BARI and EGI.

5 Conclusions

In this document the progress in the technical work of the tasks T3.1, T3.3, T3.4, and T3.6 until October 2021, M22 of the project was presented. At first, EGI is operating the cloud-based infrastructure for the project and monitoring its performance on a regular basis so no new mention is made to the INDIGO-DataCloud PaaS Orchestrator. Members of the project continue to have access and deployment capabilities of virtual clusters on top of the IaaS resources through this component.

Furthermore, in this document the status of the cloud gateways component has been reported. These gateways are responsible for obtaining data from heterogenous data sources; gateways for Maggioli, Aragon, London and Sofia use-cases have been provided through various services like the global terrorism database and Twitter. The integration between Cloud Gateways & APIs component has been implemented for several endpoints, while the integration with the user authorization mechanism has been completed, thus ensuring that all the required security standards are met.

Regarding Incentive Management, the software prototype presented here is one more step towards the realization of the final objective of the Incentive Management task. The prototype details of the Incentive Management Tool are presented and its functionality towards helping policy makers to achieve this objective is highlighted.

Finally, in section 4, the components and technologies that are used to provide an ABAC based access control mechanism suitable for PolicyCLOUD were described. This second prototype contains four key components that can be combined, along with a test client, in order to provide the required access control capabilities to the use cases, collect their feedback and proceed with the definition of a proper model to be used. Furthermore, to boost the number of potential adaptors of the prototype and increase the ease of access to it, an alternative way to authenticate to it through EGI Check-In has been implemented. The development of these mechanisms is also tightly connected to the actual integration of the platform and the authentication mechanism that will be used in it. These updates will be mostly reflected in the final version of the software prototype.

The current version of the deliverable was the second version of the software prototype; the prototypes will be further updated, and the updates will be documented in the upcoming deliverable D3.8 Cloud Infrastructure Incentives Management and Data Governance: Software Prototype 3, due in October 2022.

6 References

[1] PolicyCLOUD D3.1, Cloud Infrastructure Incentives Management and Data Governance: Design and Open Specification 1, August 2020

[2] PolicyCLOUD D3.4, Cloud Infrastructure Incentives Management and Data Governance: Design and Open Specification 2, August 2021

[3] PolicyCLOUD D3.2, Cloud Infrastructure Incentives Management and Data Governance: Software Prototype 1, November 2020

[4] PolicyCLOUD D6.6, Integration Plan, August 2021

[5] PolicyCLOUD D6.10, Use Case Scenarios Definition & Design M16, April 2020

[6] Swagger Stats, https://www.npmjs.com/package/swagger-stats

[7] Prometheus, https://prometheus.io/docs/introduction/overview/.

[8] Grafana, https://grafana.com/.

[9] Twitter Search Tweets, https://developer.twitter.com/en/docs/twitter-api/tweets/search/api-reference/get-tweets-search-recent

[10] Twitter Filtered Stream, https://developer.twitter.com/en/docs/twitter-api/tweets/filtered-stream/introduction

[11] Twitter Rate Limits, https://developer.twitter.com/en/docs/twitter-api/rate-limits

[12] SheetJS, https://docs.sheetjs.com/.

[13] KeyCloak, https://keycloak.org

[14] C. Flow, https://auth0.com/docs/flows/authorization-code-flow

[15] Netflix ZUUL Proxy, https://github.com/Netflix/zuul

[16] EGI Check-In, https://www.egi.eu/services/check-in/.

[17] eduGAIN, https://edugain.org

[18] ORCID, https://orcid.org/.

[19] Keycloak Admin REST API, https://www.keycloak.org/docs-api/5.0/rest-api/index.html.

[20] Balana, https://github.com/wso2/balana

[21] W. I. Server, https://wso2.com/identity-and-access-management/.

[22] Thrift, https://thrift.apache.org/.

[23] ApacheDS, https://directory.apache.org/apacheds/.

[24] O. 2.0, https://oauth.net/2/.

[25] OpenIDConnect, https://openid.net/connect/.