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

D5.3 CROSS-SECTOR POLICY LIFECYCLE MANAGEMENT: SOFTWARE PROTOTYPE 1

Dissemination Level	PU
Due Date of Deliverable	31/10/20, Month 10
Actual Submission Date	30/11/20
Work Package	WP5 Cross-sector Policy Lifecycle Management
Task	T5.2, T5.3 & T5.5
Type	Demonstrator
Approval Status	Final
Version	V1.0
Number of Pages	p.1 – p.36

Abstract: This document provides a description of the software components of the Policy Management Framework Layer, and Policy Development Toolkit Layer.

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	26/10/2020	ToC and assignments	Armend Duzha
0.2	10/11/2020	Contributions in Section 2 and 3	Ana Luiza Pontual Miquel Mila Prat
0.3	14/11/2020	Contributions in Section 2 and 3	Chris Maragkos Stelios Mpetas
0.4	16/11/2020	Contributions in Section 2 and 3	Kostas Moutselos, Ilias Maglogiannis
0.5	17/11/2020	Contributions in Section 2 and 3	Luis Miguel Garcia Jesús Manuel Gallego
0.6	21/11/2020	Consolidated draft ready for internal review	Armend Duzha
0.7	24/11/2020	Internal Review	Konstantino Oikonomou, Giannis Ledakis (UBI) / Enol Fernández (EGI)
0.8	27/11/2020	Addressing review comments	Armend Duzha
0.9	28/11/2020	Quality Check	Argyro Mavrogiorgou
1.0	30/11/2020	Final version ready for submission	Armend Duzha

Author List

Organisation	Name
MAG	Armend Duzha
UPRC	Ilias Maglogiannis
ICCS	Kostas Moutselos
LXS	Luis Miguel Garcia Jesús Manuel Gallego
OKS	Chris Maragkos Stelios Mpetas
ATOS	Ana Luiza Pontual Miquel Mila Prat

Abbreviations and Acronyms

Abbreviation/Acronym	Definition
API	Application Programming Interface
DSS	Decision Support System
EC	European Commission
KPI	Key Performance Indicator
PDT	Policy Development Toolkit
PM	Policy Model
PME	Policy Modelling Editor
PP	Public Policy
SOA	Service Oriented Architecture
UI	User Interface
UX	User eXperience
UML	Unified Modelling Language
WP	Work Package

Contents

List of Figures

Executive Summary

This document provides a description of the software components of the Policy Management Framework Layer and Policy Development Toolkit Layer, which are the frontend parts of the PolicyCLOUD platform and allow policy makers to create, update and evaluate policies. At this phase of the project, the prototypes are available at component level and not integrated in the PolicyCLOUD platform. For each component, a description of its APIs, specification of the main functionalities, and description of the source code are provided.

1 Introduction

1.1 Purpose and Scope

This document is the first iteration of "Cross-Sector Policy Lifecycle Management: Software prototype", and covers tasks T5.2, T5.3, and T5.5. Its main purpose is to provide a description of the prototype implementation of the Policy Modelling Editor (PME) and the Policy Development Toolkit (PDT) to support policy makers in the modelling, creation, update, and evaluation of the policies.

This document is consistent with deliverable D5.2 "Reusable Model & Analytical Tools: Design and Open Specification 1" [1], which provided the overall architecture and design specifications of the PME and PDT.

This initial report will be further updated during the project duration (at months M22 and M34) and will include additional advancements and contributions from other tasks.

1.2 Structure of the Deliverable

The rest of the document is structured as follows:

- Section 2 introduces the main components and sub-systems, their interfaces and technologies used for their development.
- Section 3 describes where the course code is made available.
- Finally, Section 4 provides the conclusion of the document.

2 Prototype overview

In this section we provide the description of the prototype implementation of the Policy Modelling Framework Layer and Policy Development Toolkit Layer, as illustrated in Figure 1.

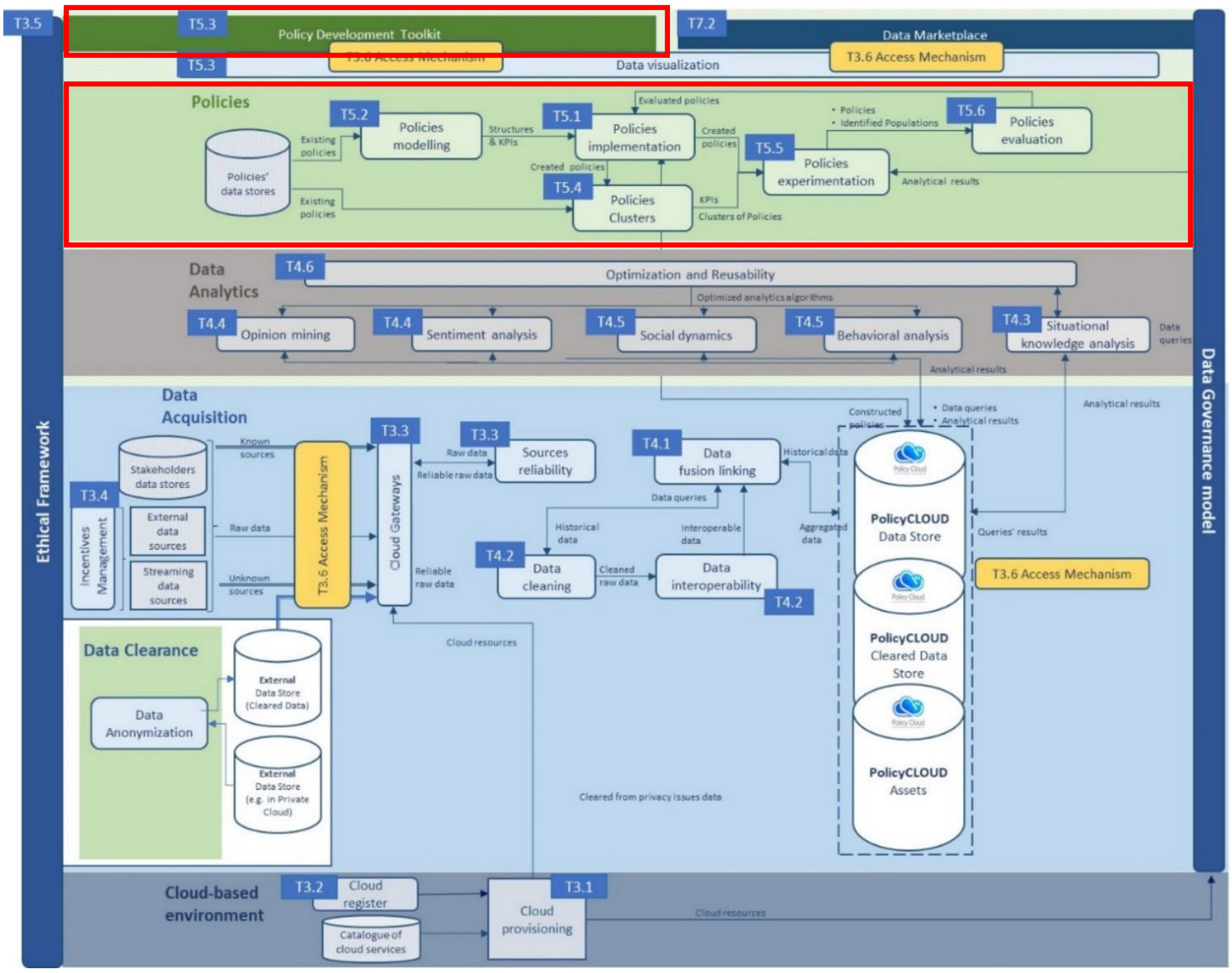


FIGURE 1: POLICYCLOUD SYSTEM ARCHITECTURE

2.1 Main components of the prototype

2.1.1 Policy Modelling Editor

The Policy Model Editor (PME) is the core component which supports and guides the end-user to effectively create a Policy Model safely. More specifically, the PME is a Single Page Application that relays on the PDT backend REST API for fetch or store related entities of the Policy Model.

2.1.2 Policy Development Toolkit

As described in the deliverable D2.2 "Conceptual Model and Reference Architecture"[2], PDT interacts with components from Layer 3 - Policies Management Framework (T5.1, T5.2), Analytical Tools (WP4) and the PDT Backend.

In this phase of the prototype, PDT provides the initial implementations of the following components as described in the deliverable D5.2 "Cross-sector Policy Lifecycle Management: Design and Open Specification 1" Section 4 [1]:

- Policy Model Viewer
- Policy Evaluations / Analytics
- A structural view of policy models owned by the user
- A User Transactions History (regarding Analytics Tools invocations by the policy maker)

Integration with the Backend has been implemented, utilising Docker images. This way, PDT can retrieve default policy models in the structured JSON format which represent the Policy Models. So, the functionalities included in this prototype are:

- The policy maker (PM) can obtain a list of available policies and select one for further exploration
- Once a policy is selected, the properties of the policy are shown, along with the available KPIs of the policy
- Once a KPI is selected, the relative KPI properties are also shown to the user
- The Analytical Tools (ATs) which can compute the selected KPI are presented. The PM can see the parameters related with each AT and invoke the respective AT. Finally, the PM can see a short overview of the current selected policy model.

The aforementioned functionalities are implemented as a policy wizard component, comprised by three steps.

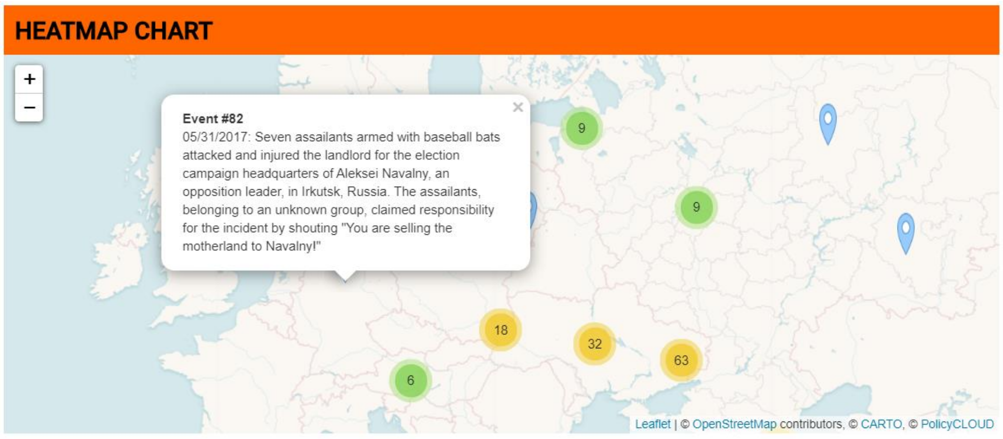
The relative UI components are described in Section 2.2.2.

2.1.3 Data Visualization

As is explained in deliverable D5.2 "Cross-sector Policy Lifecycle Management: Design and Open Specification 1" [1] section 4.1.2, the Data Visualization is an independent component that is part of the PDT, responsible for all its visualizations. However, for this first prototype, it is not be integrated yet in the PDT, and works as a standalone component. This integration will be done in upcoming iterations, and the Data Visualization, will be able to display all the visualizations required by the PDT inside of it.

For this first prototype the Data Visualization component can plot the initial set of charts agreed with two of the PolicyCLOUD Use Cases:

- A Heatmap as requested by the "Participatory Policies Against Radicalization" Use Case of Maggioli. This chart uses data extracted from the Data Analytics of WP4.


HEATMAP CHART
+
−
9
9
18
32
63
6
Leaflet | © OpenStreetMap contributors, © CARTO, © PolicyCLOUD

Event #82
05/31/2017: Seven assailants armed with baseball bats attacked and injured the landlord for the election campaign headquarters of Aleksei Navalny, an opposition leader, in Irkutsk, Russia. The assailants, belonging to an unknown group, claimed responsibility for the incident by shouting "You are selling the motherland to Navalny!"

FIGURE 2: EXAMPLE OF THE HEATMAP FOR THE MAGGIOLI USE CASE

This Heatmap is intended to show, in a glance, all incidents that occurred in a chosen region. For this first prototype, there are two possible regions: “North America” and “Eastern Europe”; but the list will be enlarged in next versions. Whenever a region is selected, the heatmap shows all incidents that took place in it. It can be filtered by year and is possible to see the incident details, when clicking on it. Users can change regions as desired.

The input data format to plot this chart has been agreed with WP4 and it is like this example:

```
[
  {
    "num_events": 1,
    "location": {"type": "Point","coordinates": [31.857623, 51.719870]},
    "location2": {"region": 9,"region_txt": "Eastern Europe"},
    "startDate": "341513723145",
    "endDate": "972662123145",
    "events":[
      {
        "eventid": 201611250026,
        "iyear": "2014",
        "latitude": 48.30696,
        "longitude": 38.029773,
        "esummary": "07/28/2014: Assailants launched rockets at the Horlivka Diocesan Administration in Horlivka city, Donetsk oblast, Ukraine. One person was killed in the attack. No group claimed responsibility for the incident."
      }
    ]
  }
]
```

- A Line and Gauge charts as requested by the “Intelligent Policies for Denomination of Origin” Use Case of Sarga, that plots the evaluation of the Sentimental Analysis considering some Keywords.

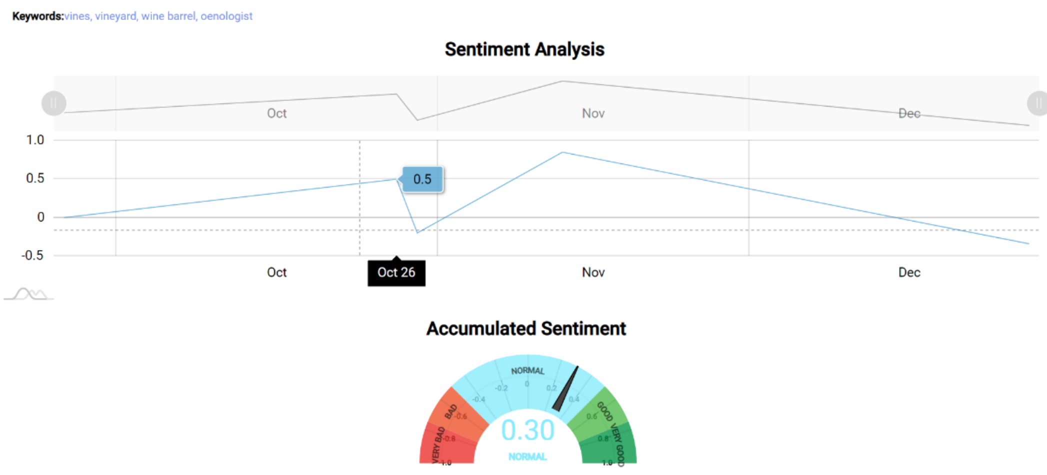


FIGURE 3: EXAMPLE OF THE LINE AND GAUGE CHARTS FOR THE SARGA USE CASE

Considering a set of keywords, the Sentiment Analysis Line chart shows the sentiments for these keywords during a given period. The Accumulated Sentiment Gauge chart shows the average of these sentiments, during the same period.

The input data used in this prototype has been provided by the Sentimental Analysis component from WP4, and the agreed format is:

```
{
  "keywords": "wine, grape, ecological",
  "acc_sentiment": 0.16,
  "list_sentiment": [["202010271000", 0.25], ["202010271100", -0.20],
["202010271200", 0.43]]
}
```

2.1.4 PDT Backend

The PDT backend provides support to the whole PDT, and it can be considered as the layer between the user interfaces that the policy maker can use to have access to the tool, the policy modelling editor, and the *Data Acquisition and Analytics* layer of the PolicyCLOUD platform, provided by the components of WP4 and WP3. It exposes its functionality via REST APIs and web sockets, to allow for the integration with the aforementioned components, while it also makes use of the REST API provided by the *Data Acquisition and Analytics* layer. As a result, its serves four main purposes:

- Allows storing, modifying and retrieving *policies,* along with their relevant corresponding entities (i.e. KPIs, stakeholders, etc.)
- Holds policy meta-information that can be used by the policy modelling editor, in order for the latter to propose new policies to the domain experts
- Invokes an analytical tool to perform an analysis through the *Data Acquisition and Analytics* layer, given the input parameters defined by the KPIs of the involved policy
- Allows for the retrieval of the results of the analytical tools, which can be later feed the visualization component of the tool. This involves again the integration of the backend with the *Data Acquisition and Analytics layer.*

More precisely, regarding the functionality implemented at this phase of the project for the PDT and the Policy Model Editor, the backend provides the ability to:

- Store, and retrieve *actors*, based on their name or the domain they are related to
- Retrieve information about the Analytical Tools that are available by the *Data Acquisition and Analytic* layer. Those tools can be retrieved either by name, by the domain they are related to or the datasets that they are compatible with.
- Retrieve information about the available datasets, stored in the central data repository of PolicyCLOUD and managed by the *Data Acquisition and Analytic* layer
- Add, remove, modify and retrieve domains that the entities of the *policies* can be relevant
- Add, remove, modify and retrieve goals of a *KPI*, and associate those tools with stakeholders
- Check the status of a *job*, which has been submitted as the result of the invocation of an analytical tool
- Get information about a *job,* submitted by the invocation of an analytical tool
- Get the result of an analysis, which is related with its *job*
- Add, remove, modify and retrieve *KPIs,* based on their name, their corresponding domain or their relevant actors.

- Add or remove actors, goals, domains and data models from a *KPI*
- Add, remove, modify and retrieve policies, according to their names, the user that stored the policy or their corresponding domain they belong to.
- Add or remove *KPIs* from *policies*

Regarding the interaction with the *Data Acquisition and Analytic* layer, the corresponding APIs have been defined in deliverable D4.2. The Backend of the PDT is integrated with the latter, in order to implement the higher-level functionality, as described above.

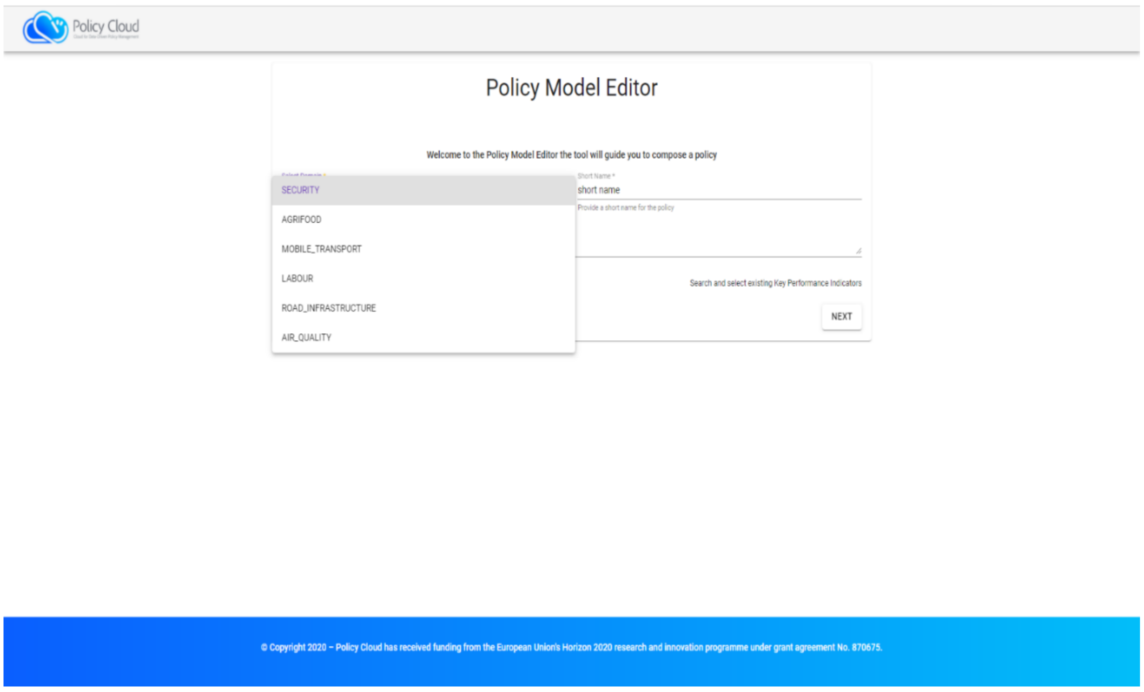
The backend of the PDT consists of a single process that runs on a servlet container. It relies on a relational datastore to store the relational model of the policy, which means, its attributes, its related entities, and the relations between them. It should be highlighted at this point, that the datastore that is being used by this component is not the central data repository of PolicyCLOUD, as the latter is used for Big Data and stores the raw data coming from the data providers, along with the results of the analytical tools. The purpose of the data storage element of this component is to store the policies and the metadata information that are related with those, while providing a standard way for communicating with the backend.

2.2 Interfaces

2.2.1 Policy Modelling Editor

Upon the end-user's entrance into the PME, the end-user has to both indicate the domain of its Policy Model and to provide a short description. Furthermore, the end-user has to provide to the PME the existing KPIs that might fit in the Policy Model under consideration.

Figure 4 illustrates the Home Page interface of the PME. It fetches from the PDT backend KPIs that belong in the same domain with the Policy Model; and then prompted by the component to continue in the next page to select existing KPIs.



Welcome to the Policy Model Editor the tool will guide you to compose a policy

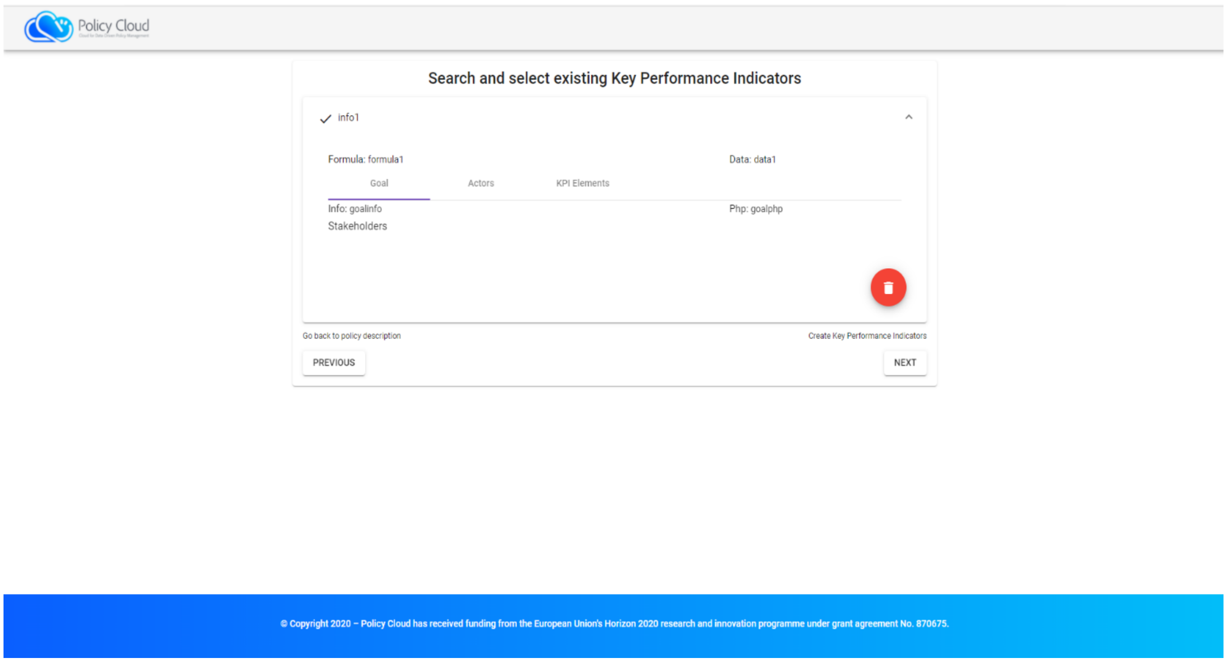
Short Name *
short name
Provide a short name for the policy

Search and select existing Key Performance Indicators

© Copyright 2020 – Policy Cloud has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 870675.

FIGURE 4: POLCIY MODEL EDITOR HOME PAGE

Figure 5 shows the Existing KPIs selection interface is illustrated, in which the end-user has the capability to add or to select existing Actors that are included in the Policy Model domain.


Search and select existing Key Performance Indicators

info1

Formula: formula1

Data: data1

Goal | Actors | KPI Elements

Info: goalinfo

Php: goalphp

Stakeholders

Go back to policy description

Create Key Performance Indicators

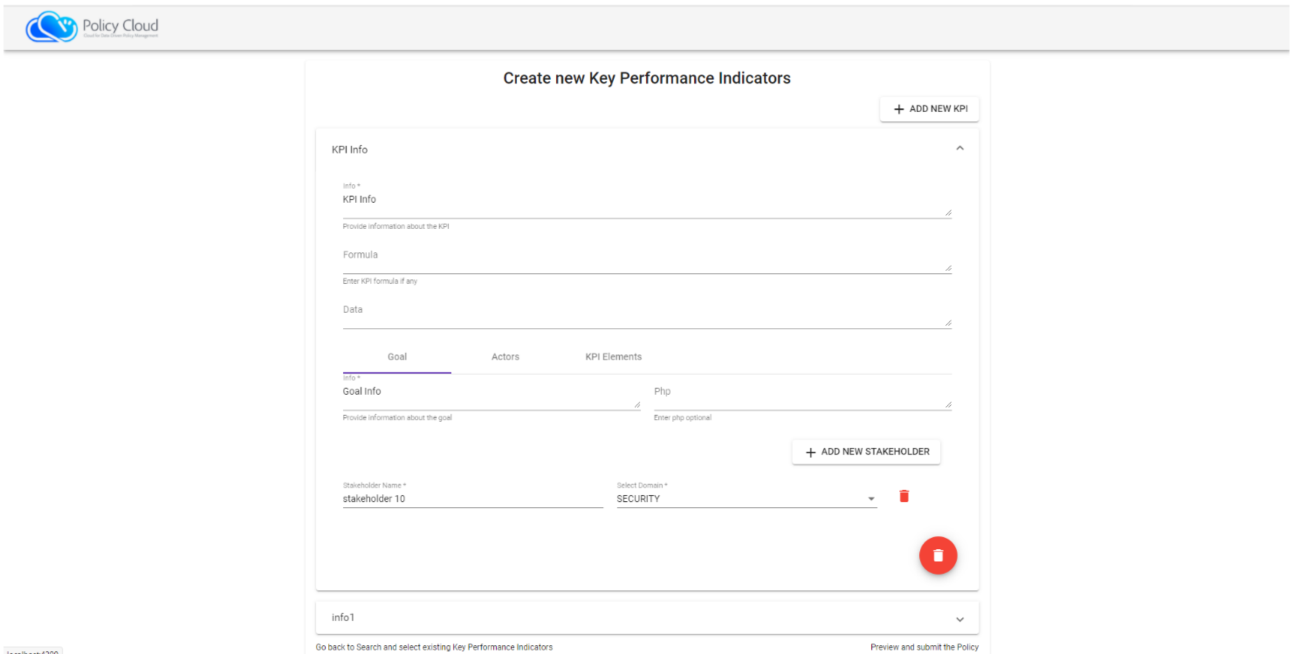
PREVIOUS

NEXT

© Copyright 2020 – Policy Cloud has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 870675.

FIGURE 5: EXISTING KPIS SELECTION

In the Add new KPI interface (see Figure 6) the end-user has the capability to create a new KPI; while he/she can enter the relevant info of the KPI, describe its Goals, and integrate the additional Stakeholders.


Create new Key Performance Indicators

ADD NEW KPI

KPI Info

Info *
KPI Info
Provide information about the KPI

Formula
Enter KPI formula if any

Data

Goal | Actors | KPI Elements

Info *
Goal Info
Provide information about the goal

Php
Enter php optional

ADD NEW STAKEHOLDER

Stakeholder Name *
stakeholder 10

Select Domain *
SECURITY

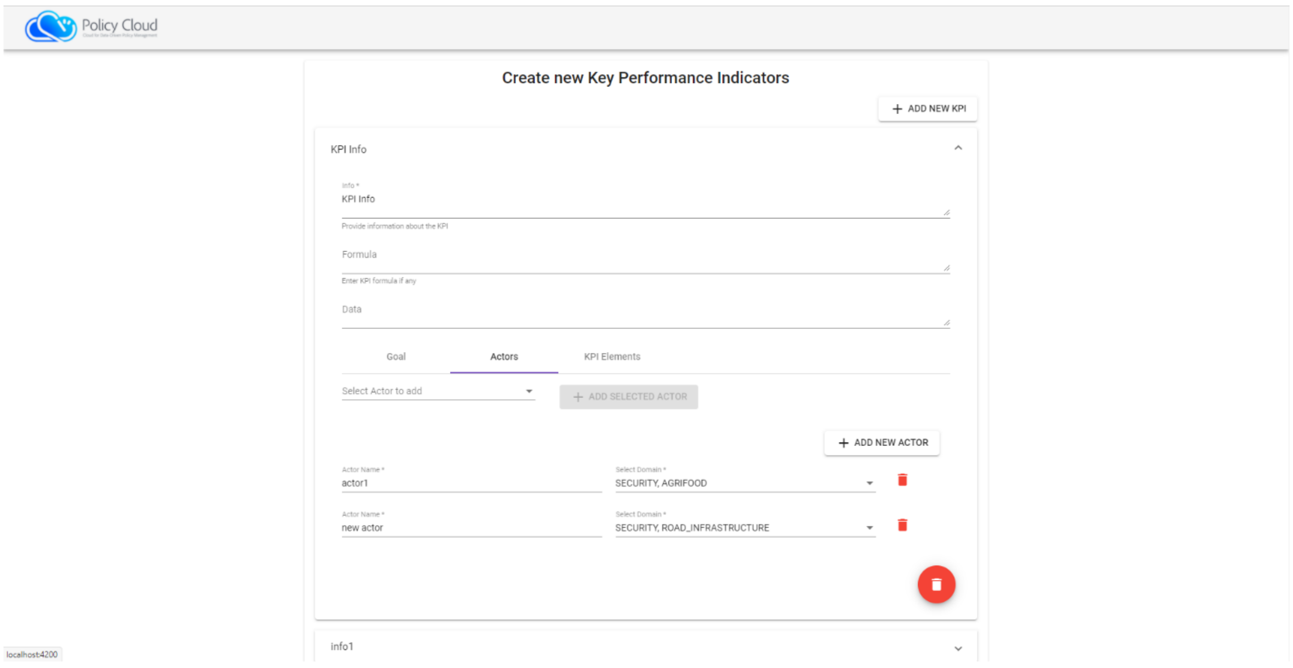
info1

Go back to Search and select existing Key Performance Indicators

Preview and submit the Policy

FIGURE 6: ADD NEW KPI

It should be noted that the user has the following capabilities: (i) to provide description of the Data Model and add or select Actors, (ii) to select the relevant Analytical Tools from a specific list (drop-down menu) (Figure 7), (iii) to provide the parameter values that comes for the Analytical Tools (Figure 8), (iv) to review and submit the Policy Model (Figure 9).


Create new Key Performance Indicators

KPI Info

Info *
KPI Info
Provide information about the KPI

Formula
Enter KPI formula if any

Data

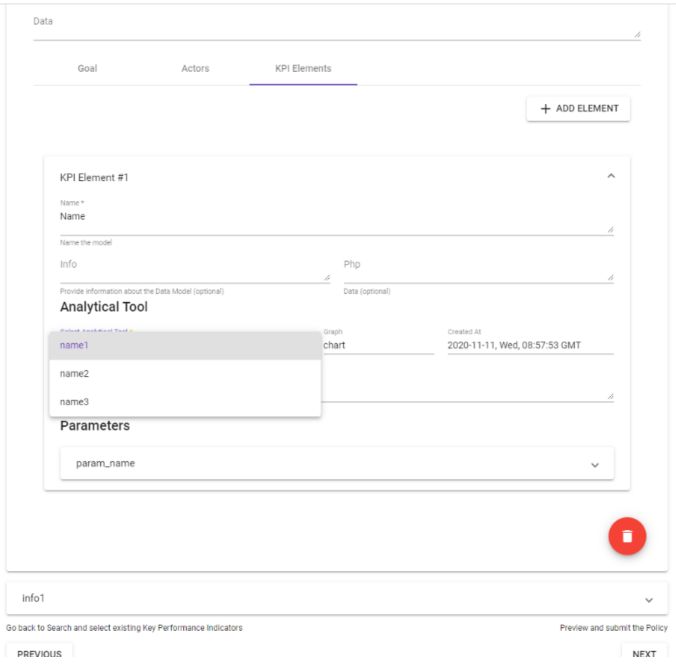
Actor Name *: actor1 — Select Domain *: SECURITY, AGRIFOOD

Actor Name *: new actor — Select Domain *: SECURITY, ROAD_INFRASTRUCTURE

info1

localhost:4200

FIGURE 7: ADD NEW ACTOR


Data

KPI Element #1

Name *
Name
Name the model

Info
Provide information about the Data Model (optional)

Php
Data (optional)

Analytical Tool

Graph: chart

Created At: 2020-11-11, Wed, 08:57:53 GMT

Parameters

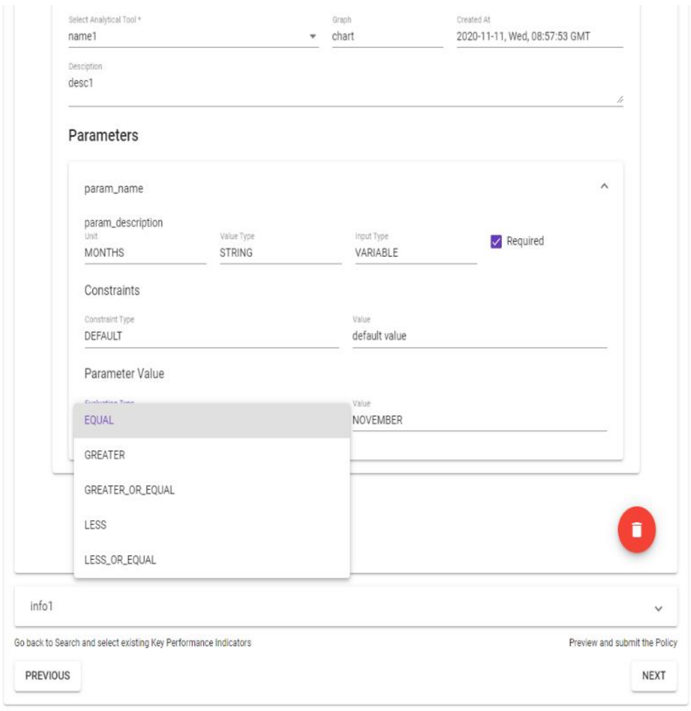
param_name

info1

Go back to Search and select existing Key Performance Indicators

Preview and submit the Policy

FIGURE 8: ANALYTICAL TOOL SELECTION



Select Analytical Tool *
name1
Graph
chart
Created At
2020-11-11, Wed, 08:57:53 GMT
Description
desc1

Parameters

param_name
param_description
Unit
MONTHS
Value Type
STRING
Input Type
VARIABLE
☑ Required

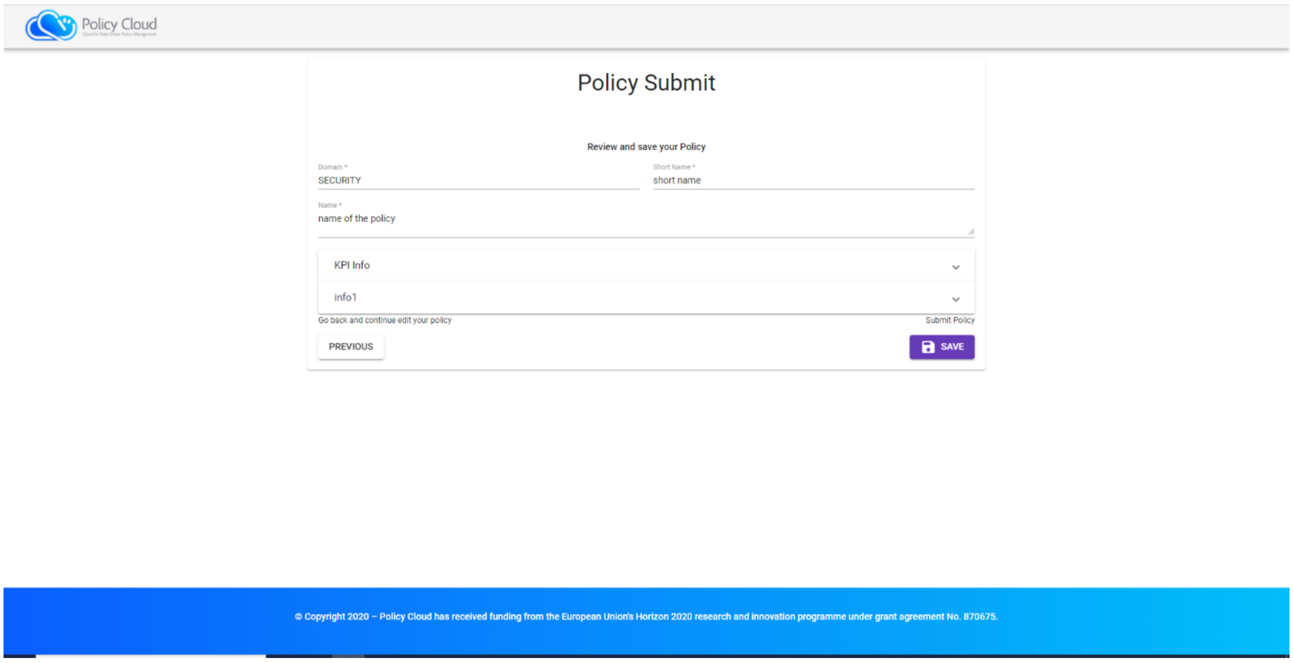
Constraints
Constraint Type
DEFAULT
Value
default value

Parameter Value
Value
NOVEMBER

EQUAL
GREATER
GREATER_OR_EQUAL
LESS
LESS_OR_EQUAL
info1
Go back to Search and select existing Key Performance Indicators
Preview and submit the Policy
PREVIOUS
NEXT

FIGURE 9: SET PARAMETER VALUE

Finally, if the process was successful the user is redirected to the last page that promotes the user to visit the PDT and evaluate the submitted Policy Model or compose a new one.



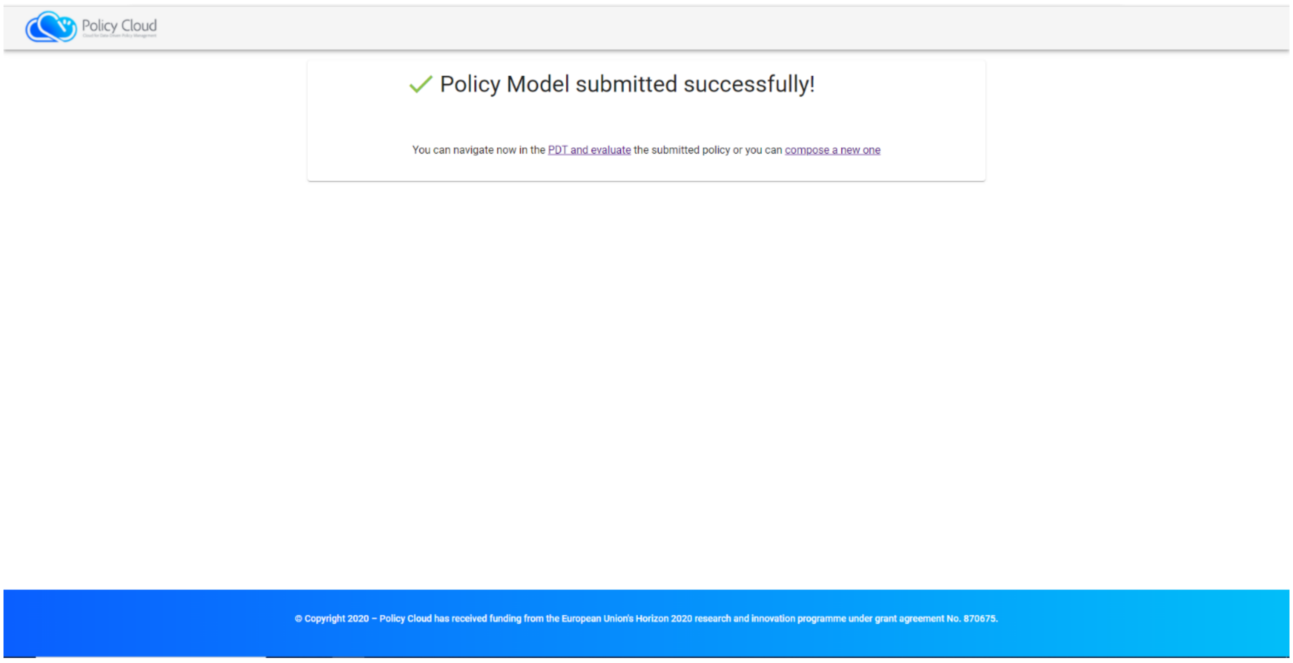
Policy Submit

Review and save your Policy

Domain *
SECURITY
Short Name *
short name
Name *
name of the policy

© Copyright 2020 – Policy Cloud has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 870675.

FIGURE 10: REVIEW AND SUBMIT


Policy Model submitted successfully!

You can navigate now in the PDT and evaluate the submitted policy or you can compose a new one

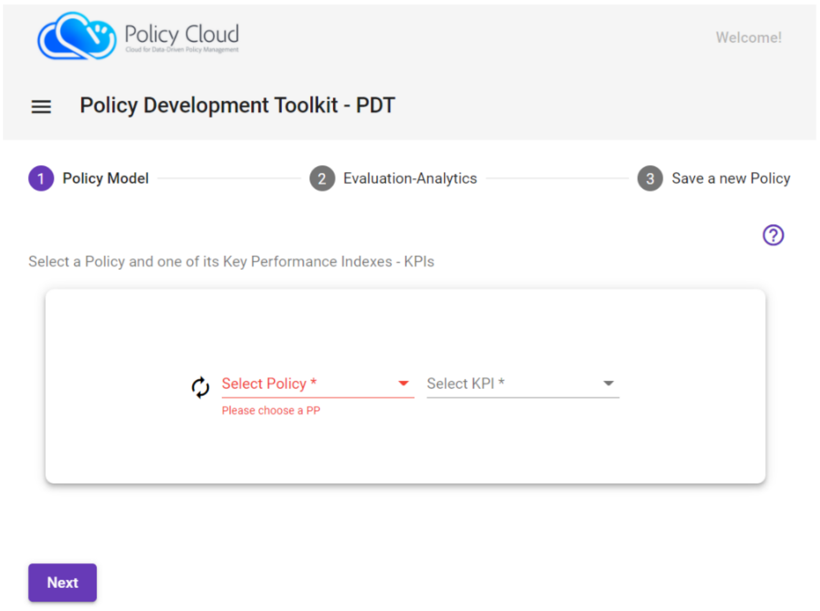
© Copyright 2020 – Policy Cloud has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 870675.

FIGURE 11: POLICY MODEL SUBMITTED

2.2.2 Policy Development Toolkit

The following figures present the UI implementations of the PDT functionalities, as listed in Section 2.1.2. The web pages are served locally communicating with the dockerised backend API.

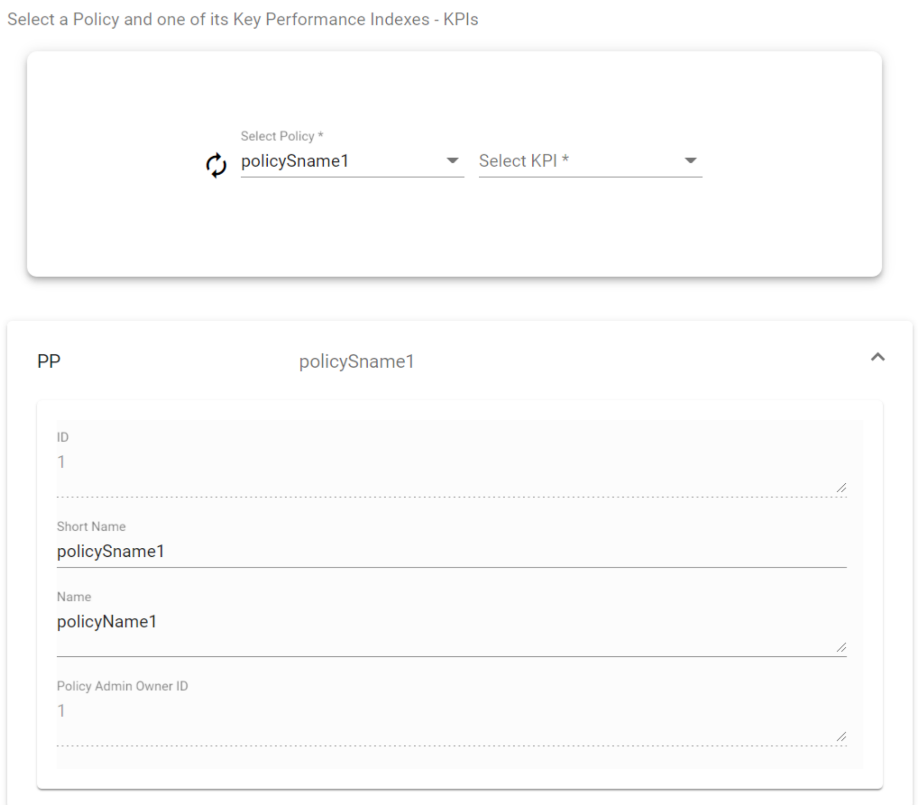
Figure 12 shows the landing web page of PDT (after the pending Authentication procedure). The page shows the three-step policy wizard, with the first step as the default location. Here the user selects a policy from a list of available policy models.


Select a Policy and one of its Key Performance Indexes - KPIs

Please choose a PP

FIGURE 12: POLICY SELECTION-EVALUATION WIZARD

Figure 13 shows the properties of the policy model selected by the user. After the selection of a policy, the related KPIs are displayed, and the PM can select one KPI for evaluation.


Select a Policy and one of its Key Performance Indexes - KPIs

Select Policy *
policySname1
Select KPI *

PP policySname1

ID
1

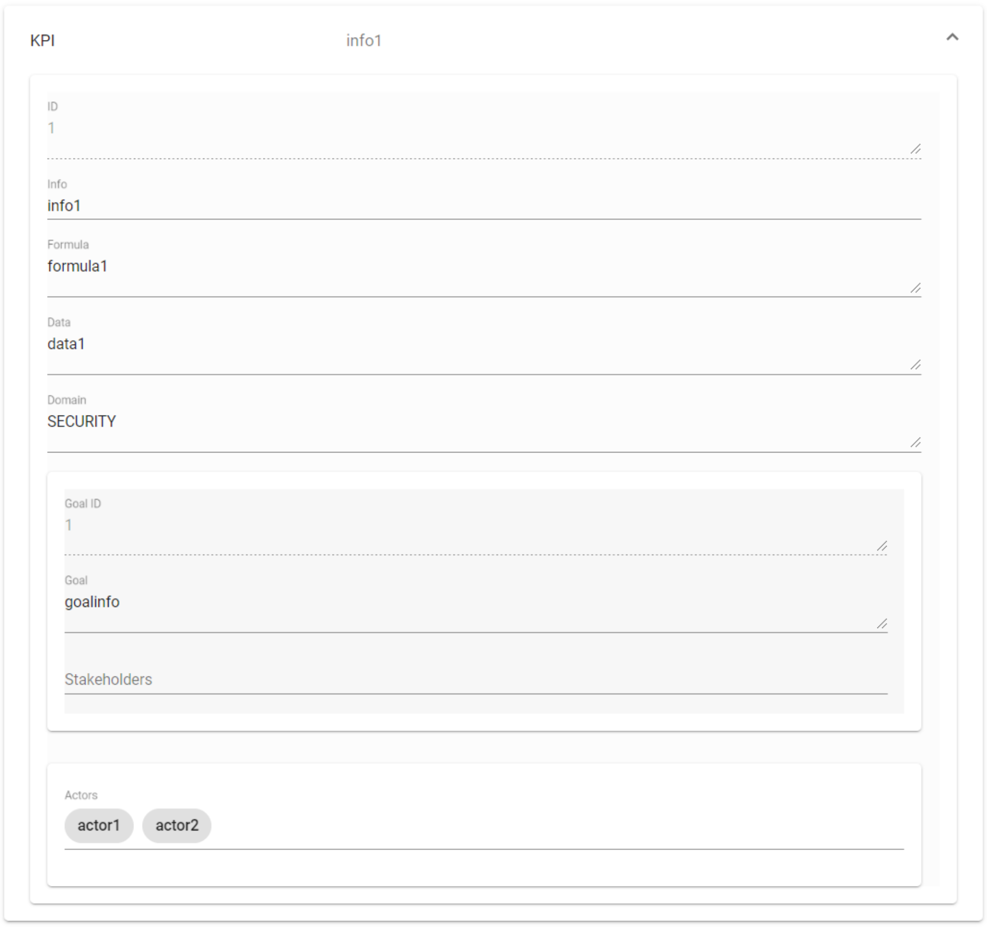
Short Name
policySname1

Name
policyName1

Policy Admin Owner ID
1

FIGURE 13: POLICY/KPI SELECTION. POLICY PROPERTIES.

Figure 14 shows the properties of the selected KPI, e.g. formula, data, domain, goal, actors and stakeholders.


KPI info1

ID
1

Info
info1

Formula
formula1

Data
data1

Domain
SECURITY

Goal ID
1

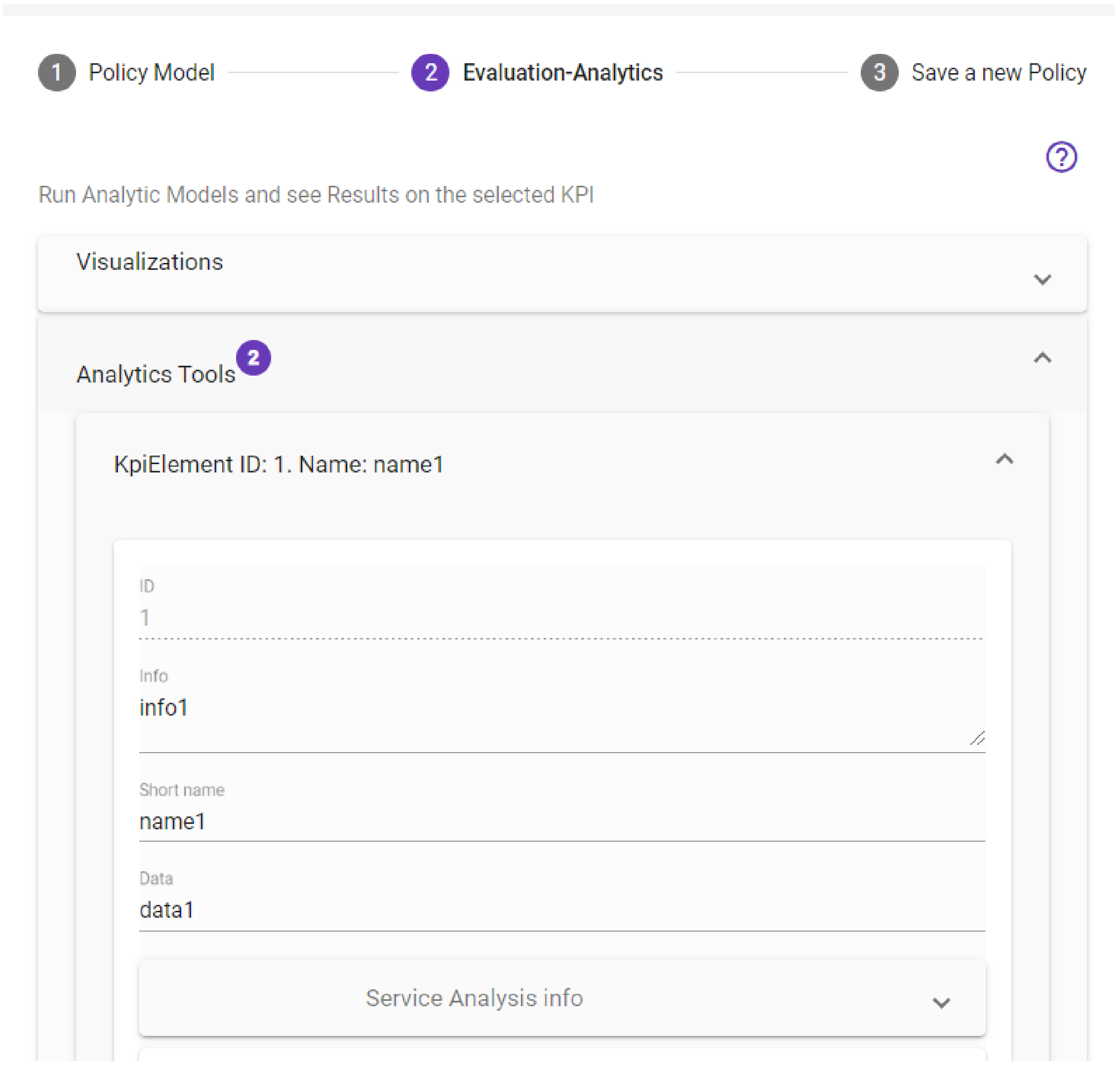
Goal
goalinfo

Stakeholders

Actors
actor1 actor2

FIGURE 14: KPI PROPERTIES

Figure 15 shows the second step of the wizard, which is the evaluation of the selected KPI, by using the proper Analytics Tool.



ID
1

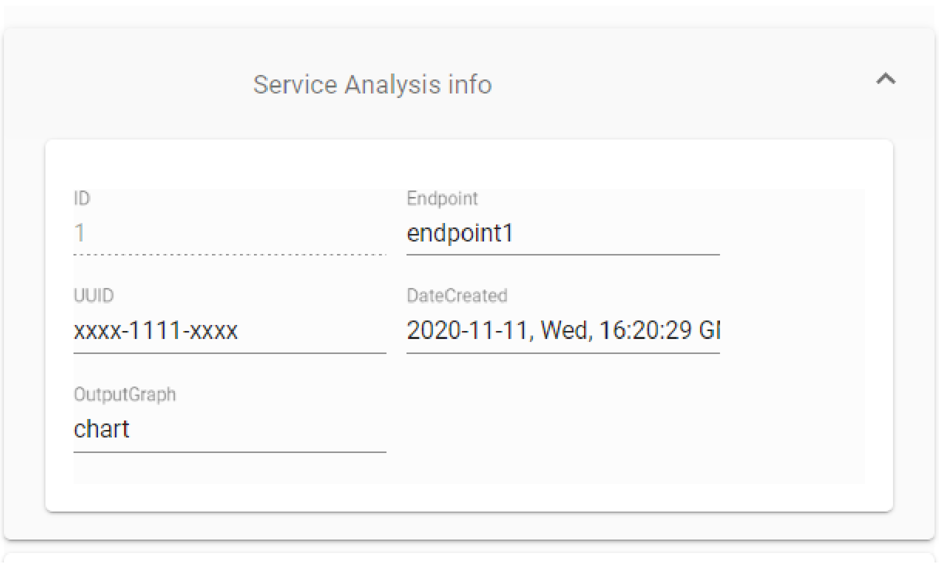
Info
info1

Short name
name1

Data
data1

FIGURE 15: EVALUATION - ANALYTICS TOOL SELECTION

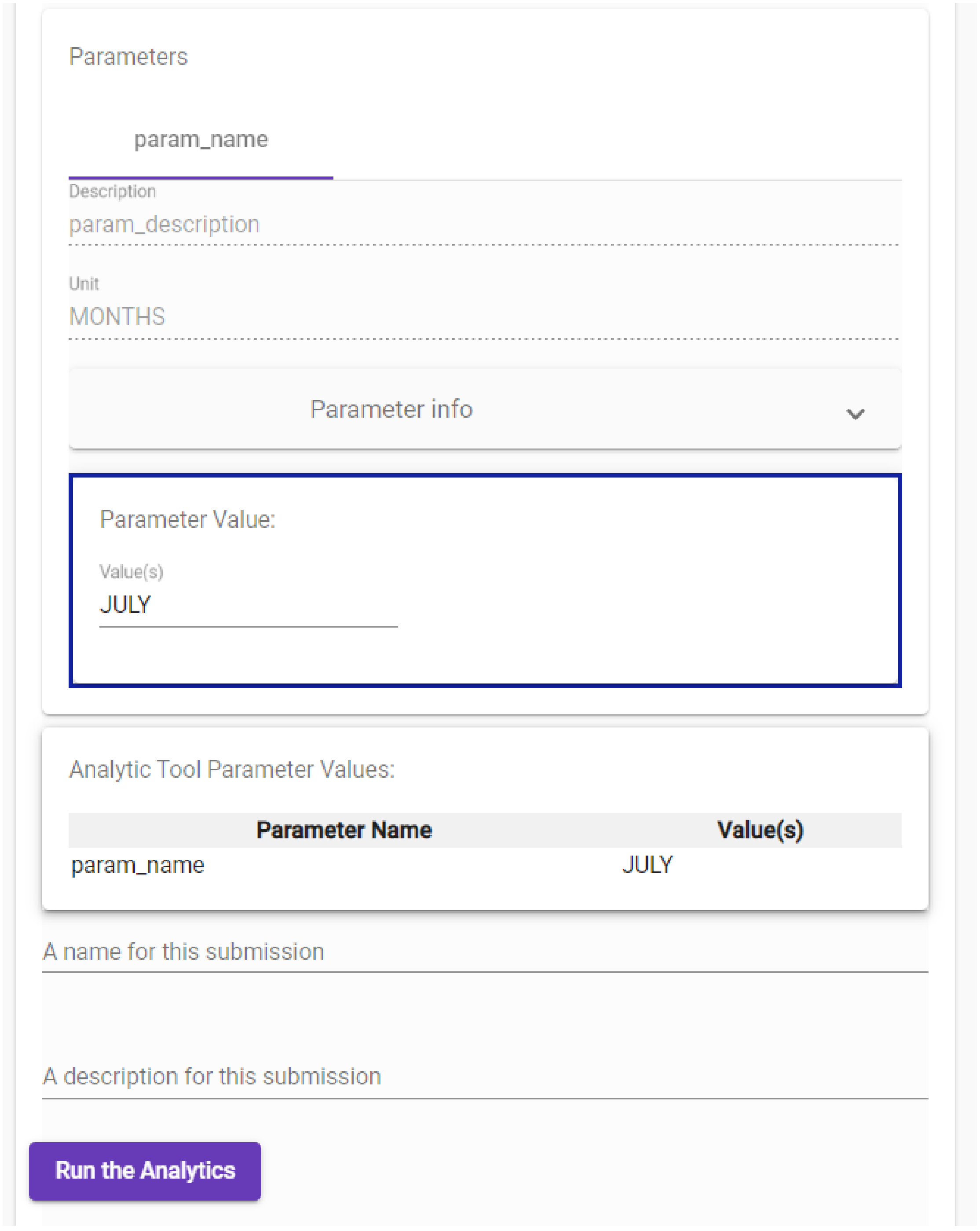
Figure 16 shows technical details of the Analytics Tool in an expandable card, as to not distract the PM.



ID: 1
Endpoint: endpoint1
UUID: xxxx-1111-xxxx
DateCreated: 2020-11-11, Wed, 16:20:29 GI
OutputGraph: chart

FIGURE 16: ANALYTICS TOOL TECHNICAL DETAILS

Figure 17 shows the parameter list that the selected Analytics Tool is expecting for invocation. A parameter's values overview is also presented, which includes the default values for the calculation of the KPI.


Parameters

param_name

Description
param_description

Unit
MONTHS

Parameter Value:

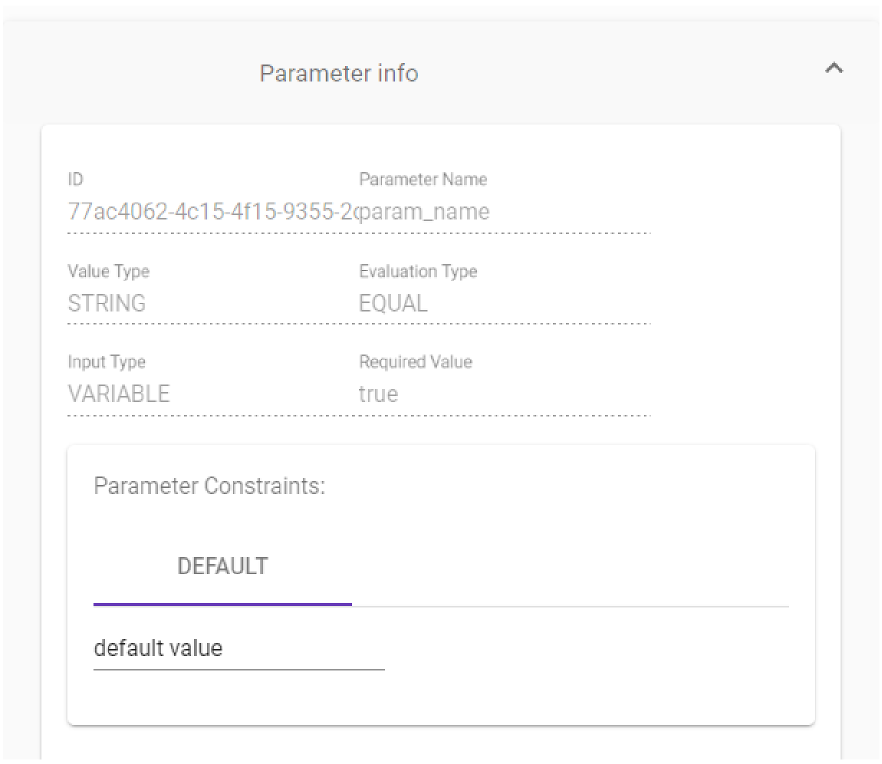
Value(s)
JULY

Analytic Tool Parameter Values:

Parameter Name	Value(s)
param_name	JULY

FIGURE 17: ANALYTICS TOOL PARAMETERS & SUBMISSION

Figure 18 shows the details for each parameter of the AT, also in the form of an expandable card.


Parameter info

ID	Parameter Name
77ac4062-4c15-4f15-9355-2	param_name
Value Type	Evaluation Type
STRING	EQUAL
Input Type	Required Value
VARIABLE	true

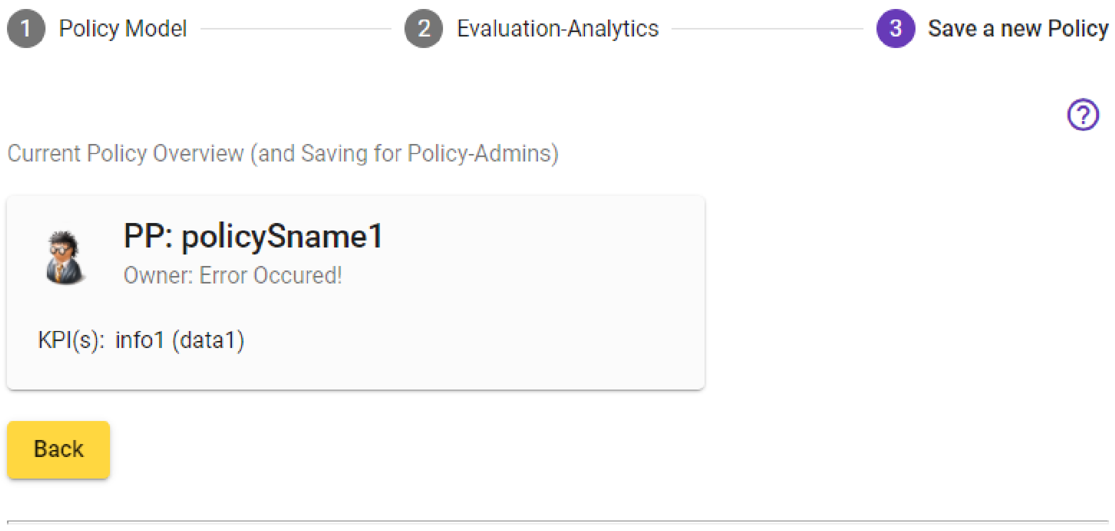
Parameter Constraints:

DEFAULT

default value

FIGURE 18: ANALYTICS TOOL PARAMETER DETAILS

Figure 19 shows the third step of the wizard, presenting a short overview of the selected policy. The option to save a new policy in case of parameter value changes is shown, although its functionality is under consideration.



Current Policy Overview (and Saving for Policy-Admins)

PP: policySname1

Owner: Error Occured!

KPI(s): info1 (data1)

FIGURE 19: POLICY OVERVIEW / SAVING

Figure 20 shows the UI Menu for selecting different PDT components.

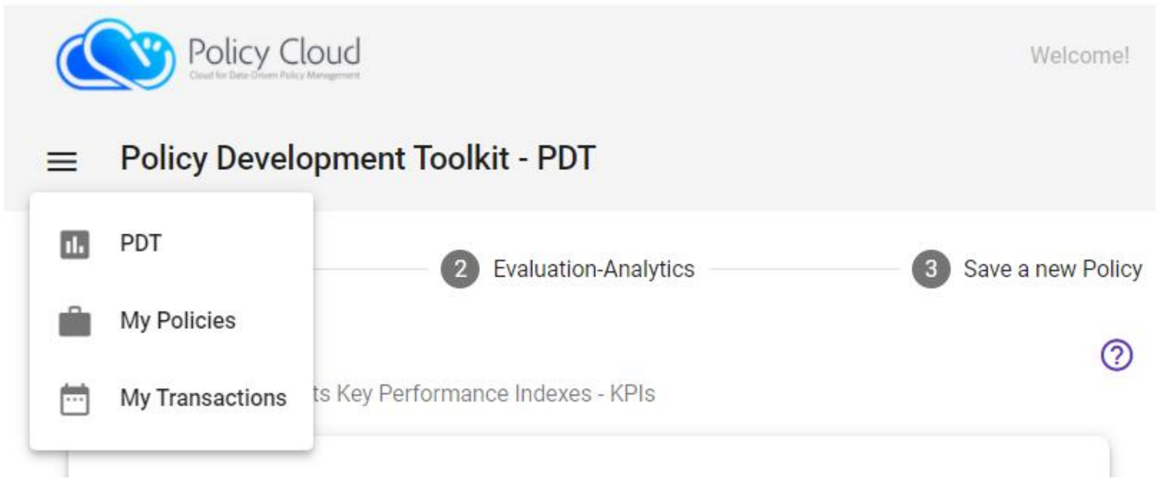


FIGURE 20: PDT COMPONENTS MENU

Figure 21 shows the 'My Policies' component, which will present the structure of the policy models owned by the PM.

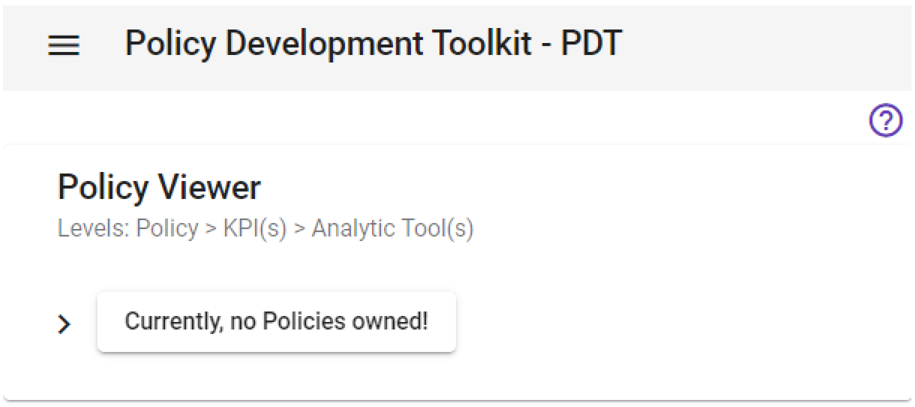
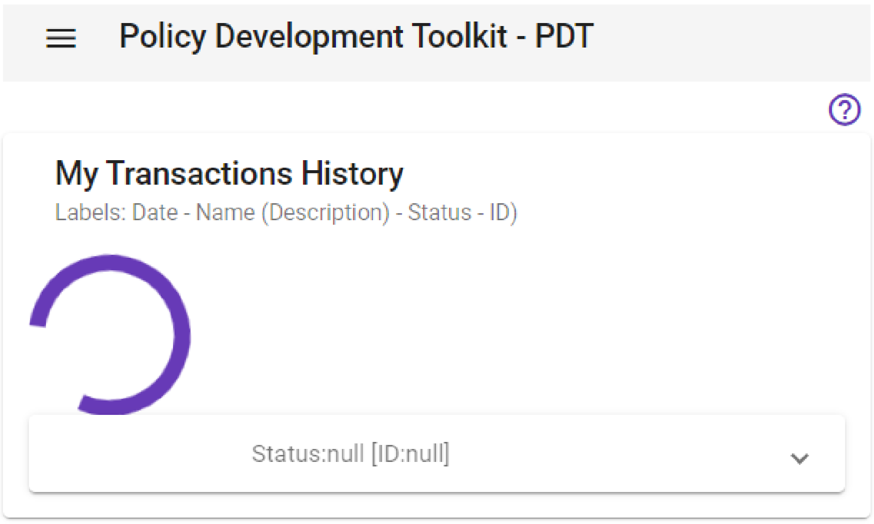

Policy Development Toolkit - PDT
Policy Viewer
Levels: Policy > KPI(s) > Analytic Tool(s)
Currently, no Policies owned!

FIGURE 21: MY POLICES VIEWER

Figure 22 shows the 'My Transactions History' component which will list all the analytics tools invocations by the PM, with the ability to filter depending on the policy.


Policy Development Toolkit - PDT

My Transactions History

Labels: Date - Name (Description) - Status - ID)

Status:null [ID:null]

FIGURE 22: MY TRANSACTIONS VIEWER / ANALYTICS HISTORY

2.2.3 Data Visualization

The user interface of the Visualization component provided in this prototype is an HTML page. Using the wizard available in this UI, the two initial chosen pilot Use Cases can be selected, as also some demo mock-up data, in order to plot the charts of this first prototype.

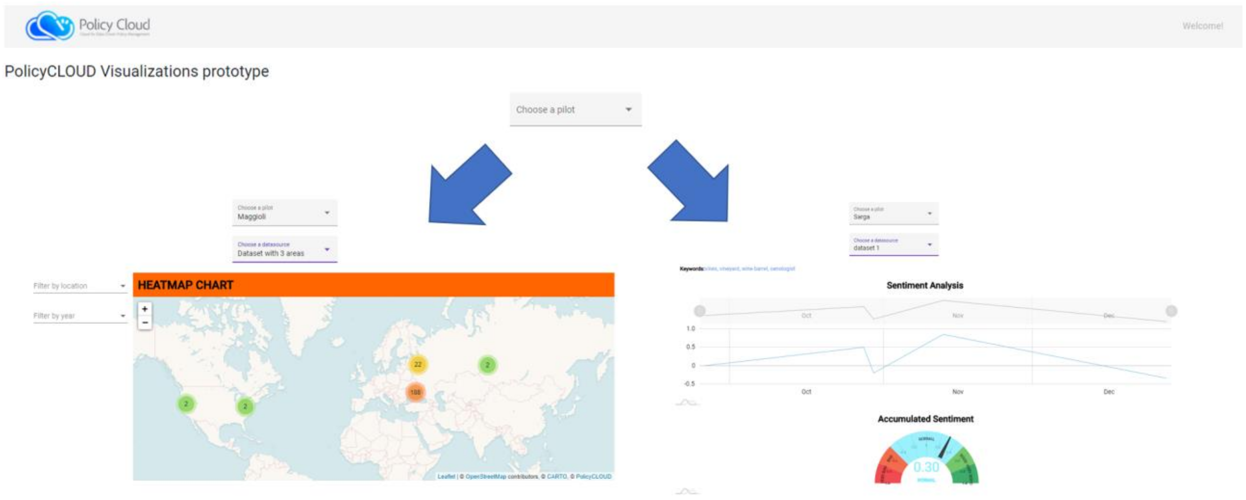


FIGURE 23: VISUALIZATION UI COMPONENT SCREENSHOT

2.2.4 PDT Backend

As described in the previous subsection, the backend of the PDT exposes its interfaces via REST API. In order to provide an always up-to-date documentation of the definition of the interface to the developers of other components, we automated this process by making use of the swagger [3] open source documentation UI. Swagger allows the developer to annotate her source code and provides tools and plugins that auto-generate the documentation during compilation time. It generates JSON files with all the relevant information. It also provides a web application that can be used as the UI tool for the consumer of the REST services to see their documentation. This web application makes use of those auto-generated JSON files to build the user interface. Moreover, it provides to the REST service consumer the ability to test the interface online, by invoking them in real-time and check the results.

The following code snippet indicates how the developer of a REST service can annotate the latter via the swagger framework.

```
@Path("domains")
@Api(basePath="http://localhost:54735/pdt", value = "/domains", description = "REST Service that
provides functionalities for domains")
public class DomainResource {
```

The class *DomainResource* implements the web methods related with the functionalities for the *domains,* and it has been annotated to use the REST path "/domains". As it can be noted, the developer can additionally annotate the class with the @Api that informs the swagger that this REST resource should be documented.

Moreover, at the level of web methods, the following code snippet shows how the developer can document the details of each invocation

```
@GET
@Path("/starts")
@ApiOperation(value="Returns the list of domains that starts with the given input",
               notes="Returns the list of domains that starts with the given input",
               responseContainer = "List", response = DomainDTOImpl.class)
@ApiResponses(value= {
        @ApiResponse(code=500, message="Exception's message")
})
@Produces(MediaType.APPLICATION_JSON)
@SuppressWarnings("UseSpecificCatch")
public Response getDomainsStartingWith(
               @ApiParam(value = "the start with parameter", required = true)
               @QueryParam("name") String name) {
```

This web method that returns all *domains* whose name starts with a given value. The swagger annotation indicates to the framework that this should be an operation that will return a *List* of serialized objects of the *DomainDTOImpl c*lass. Besides the default HTTP code 200 that indicates a successful invocation, this web method can also return an HTTP error code 500 and provides the parameter that is required. Swagger, uses Java Reflection to retrieve additional information like the type of the invocation (GET, POST, PUT etc.) the name and type of the input parameters, the media type used in the body of the request or/and the response of the HTTP call etc.

Finally, the DTOs that will be transferred via the REST calls, can also be annotated, as the following code snippet indicates.

```
@JsonIgnoreProperties(ignoreUnknown = true)
public class DomainDTOImpl implements DomainDTO, Serializable {
    private static final long serialVersionUID = 1L;

    @ApiModelProperty(value="domain id", required=true)
    private Long id;
    @ApiModelProperty(value="domain name", required=true)
    private String name;

    @JsonCreator
    public DomainDTOImpl(
            @JsonProperty("id") Long id,
            @JsonProperty("name") String name) {
        this.id = id;
        this.name = name;
    }
```

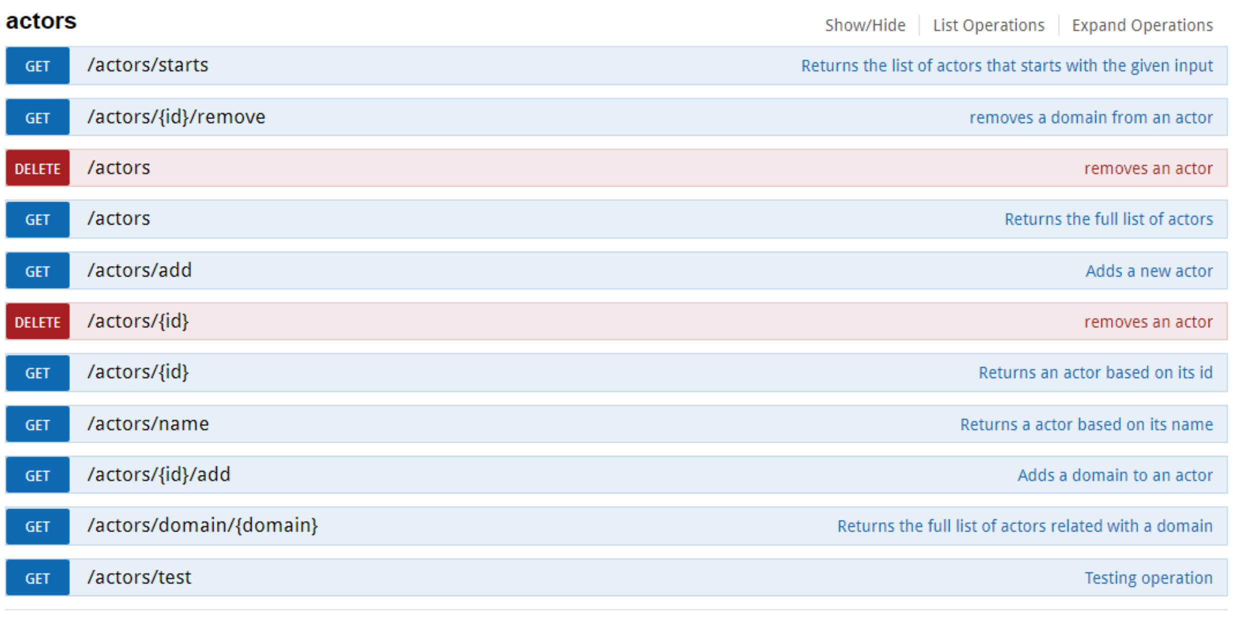
The aforementioned DomainDTOImpl used by the web method explained before, has two attributes, the id and the name of the domain. The *ApiModelProperty* can be used by swagger to provide this kind of information to the documentation.

After having annotated all our web services, web methods and DTOs, the swagger plugin for maven was used to auto-generate the stub information in JSON files. The following code snippet was used to do this work.

```
<plugin>
        <groupId>com.github.kongchen</groupId>
        <artifactId>swagger-maven-plugin</artifactId>
        <version>2.3.4</version>
        <configuration>
                <apiSources>
                        <apiSource>
                                <locations>eu.policycloud.rest.resources</locations>
                                <apiVersion>${project.version}</apiVersion>
                                <basePath>/resources</basePath>

        <swaggerDirectory>${project.basedir}/src/main/webapp/</swaggerDirectory>

        <outputTemplate>${project.basedir}/src/main/resources/markdown.mustache</outputTemplate>

        <mustacheFileRoot>${project.basedir}/src/main/resources/</mustacheFileRoot>

        <outputPath>${project.basedir}/src/main/webapp/document.html</outputPath>
                        </apiSource>
                </apiSources>
        </configuration>
        <executions>
                <execution>
                        <phase>compile</phase>
                        <goals>
                                <goal>generate</goal>
                        </goals>
                </execution>
        </executions>
</plugin>
```

It creates the JSON files during *compile phase,* and adds them to corresponding directory where the swagger app is expecting to retrieve this information, while it also modifies the markdown mustache template used by the web app.

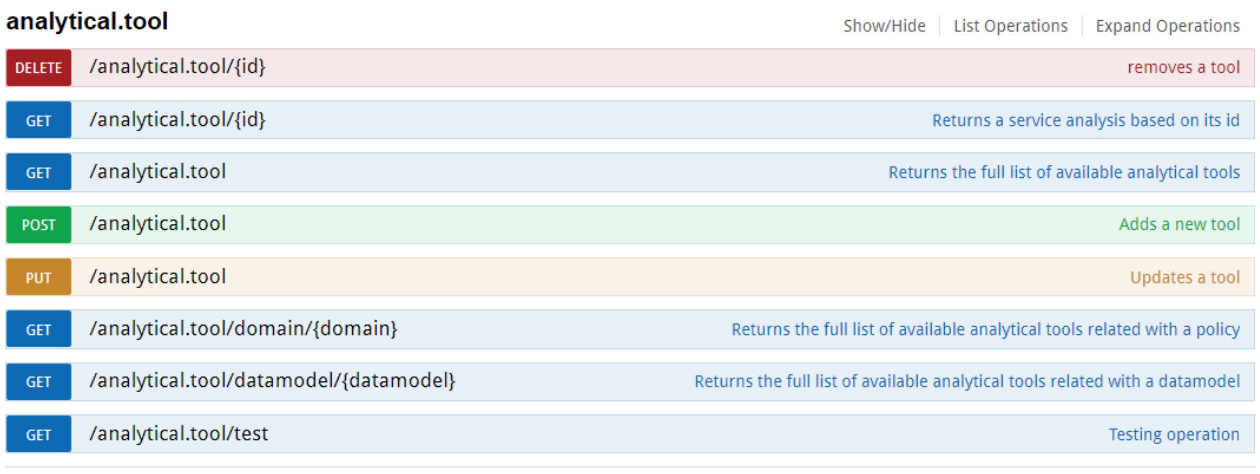
After setting up everything, the online documentation is now available. The following screenshots show this documentation per REST Web Service implemented at this phase of the project.



actors Show/Hide | List Operations | Expand Operations

GET	/actors/starts	Returns the list of actors that starts with the given input
GET	/actors/{id}/remove	removes a domain from an actor
DELETE	/actors	removes an actor
GET	/actors	Returns the full list of actors
GET	/actors/add	Adds a new actor
DELETE	/actors/{id}	removes an actor
GET	/actors/{id}	Returns an actor based on its id
GET	/actors/name	Returns a actor based on its name
GET	/actors/{id}/add	Adds a domain to an actor
GET	/actors/domain/{domain}	Returns the full list of actors related with a domain
GET	/actors/test	Testing operation

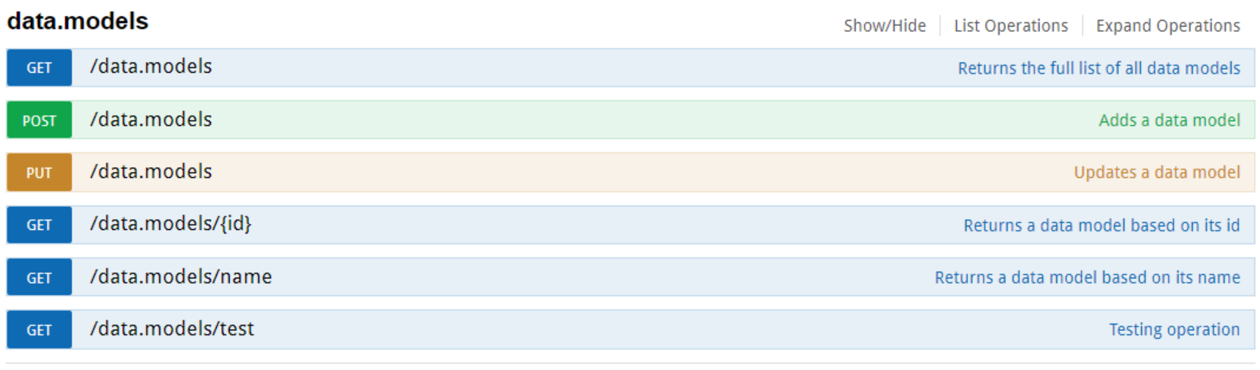
FIGURE 24: WEB METHODS RELATED WITH ACTORS



analytical.tool Show/Hide | List Operations | Expand Operations

DELETE	/analytical.tool/{id}	removes a tool
GET	/analytical.tool/{id}	Returns a service analysis based on its id
GET	/analytical.tool	Returns the full list of available analytical tools
POST	/analytical.tool	Adds a new tool
PUT	/analytical.tool	Updates a tool
GET	/analytical.tool/domain/{domain}	Returns the full list of available analytical tools related with a policy
GET	/analytical.tool/datamodel/{datamodel}	Returns the full list of available analytical tools related with a datamodel
GET	/analytical.tool/test	Testing operation

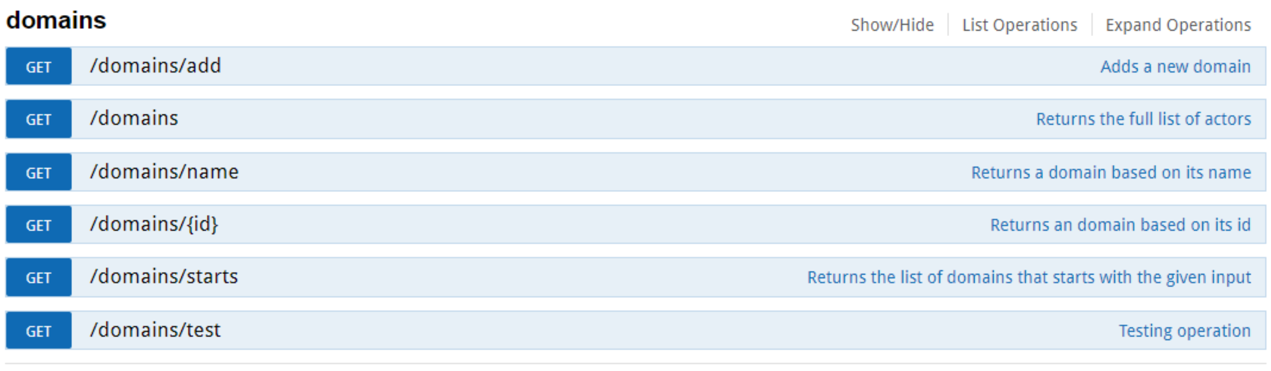
FIGURE 25: WEB METHODS RELATED WITH ANALYTICAL TOOLS



data.models — Show/Hide | List Operations | Expand Operations

GET	/data.models	Returns the full list of all data models
POST	/data.models	Adds a data model
PUT	/data.models	Updates a data model
GET	/data.models/{id}	Returns a data model based on its id
GET	/data.models/name	Returns a data model based on its name
GET	/data.models/test	Testing operation

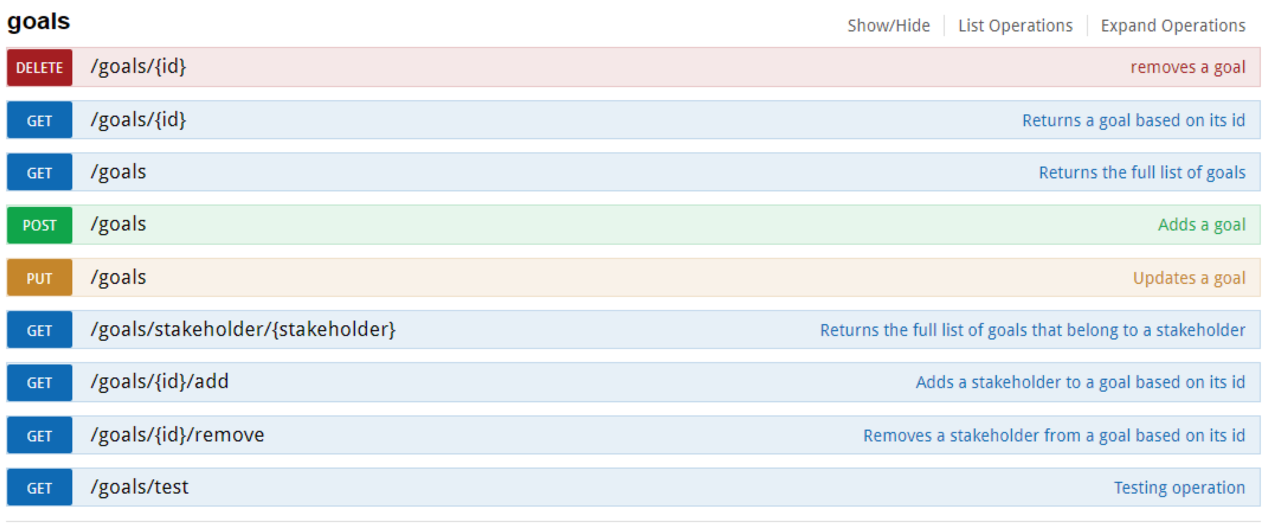
FIGURE 26: WEB METHODS RELATED WITH DATA MODELS



domains — Show/Hide | List Operations | Expand Operations

GET	/domains/add	Adds a new domain
GET	/domains	Returns the full list of actors
GET	/domains/name	Returns a domain based on its name
GET	/domains/{id}	Returns an domain based on its id
GET	/domains/starts	Returns the list of domains that starts with the given input
GET	/domains/test	Testing operation

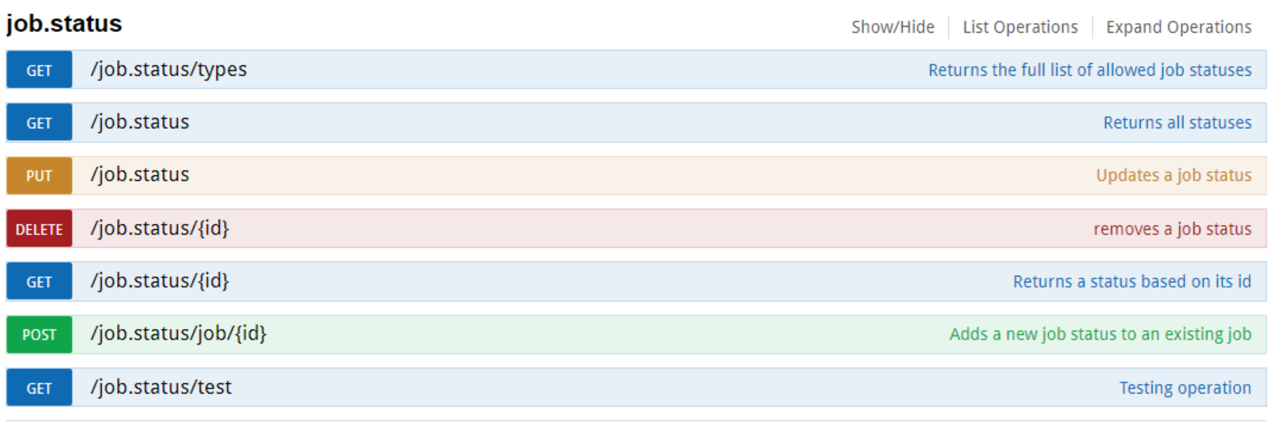
FIGURE 27: WEB METHODS RELATED WITH DOMAINS



goals — Show/Hide | List Operations | Expand Operations

DELETE	/goals/{id}	removes a goal
GET	/goals/{id}	Returns a goal based on its id
GET	/goals	Returns the full list of goals
POST	/goals	Adds a goal
PUT	/goals	Updates a goal
GET	/goals/stakeholder/{stakeholder}	Returns the full list of goals that belong to a stakeholder
GET	/goals/{id}/add	Adds a stakeholder to a goal based on its id
GET	/goals/{id}/remove	Removes a stakeholder from a goal based on its id
GET	/goals/test	Testing operation

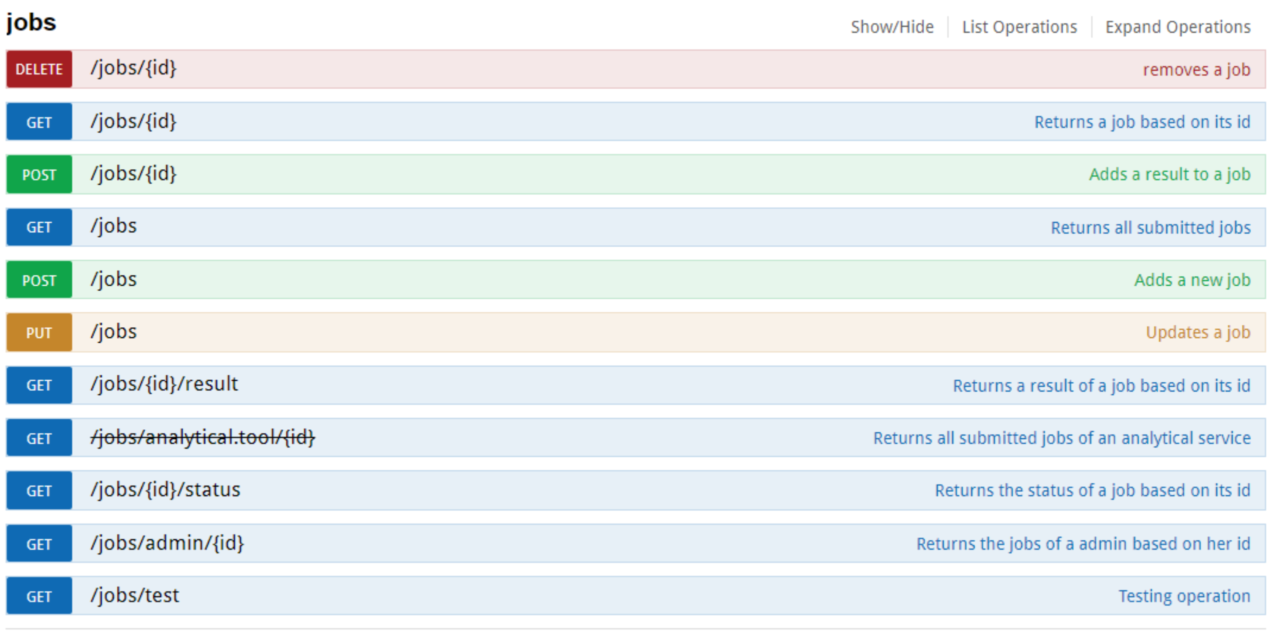
FIGURE 28: WEB METHODS RELATED WITH GOALS



job.status — Show/Hide | List Operations | Expand Operations

GET	/job.status/types	Returns the full list of allowed job statuses
GET	/job.status	Returns all statuses
PUT	/job.status	Updates a job status
DELETE	/job.status/{id}	removes a job status
GET	/job.status/{id}	Returns a status based on its id
POST	/job.status/job/{id}	Adds a new job status to an existing job
GET	/job.status/test	Testing operation

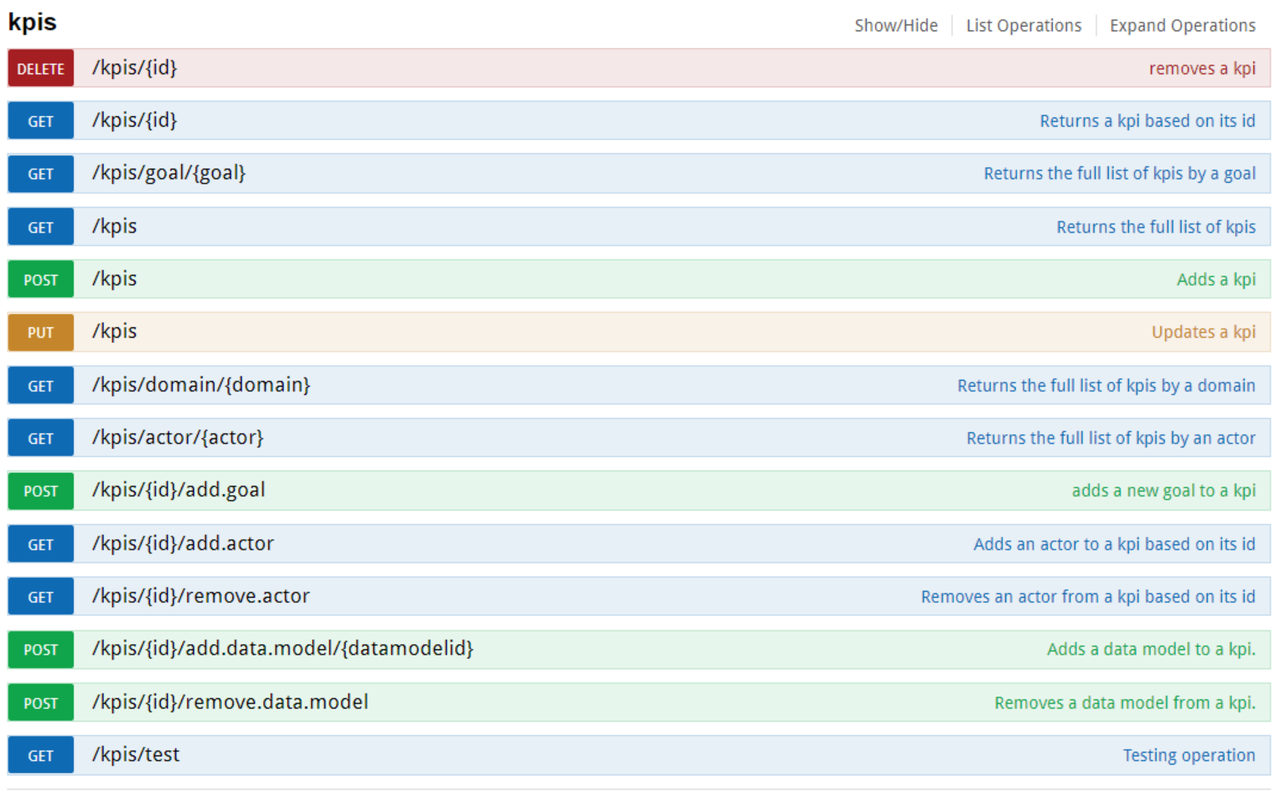
FIGURE 29: WEB METHODS RELATED WITH JOB STATUS



jobs — Show/Hide | List Operations | Expand Operations

DELETE	/jobs/{id}	removes a job
GET	/jobs/{id}	Returns a job based on its id
POST	/jobs/{id}	Adds a result to a job
GET	/jobs	Returns all submitted jobs
POST	/jobs	Adds a new job
PUT	/jobs	Updates a job
GET	/jobs/{id}/result	Returns a result of a job based on its id
GET	~~/jobs/analytical.tool/{id}~~	Returns all submitted jobs of an analytical service
GET	/jobs/{id}/status	Returns the status of a job based on its id
GET	/jobs/admin/{id}	Returns the jobs of a admin based on her id
GET	/jobs/test	Testing operation

FIGURE 30: WEB METHODS RELATED WITH JOBS

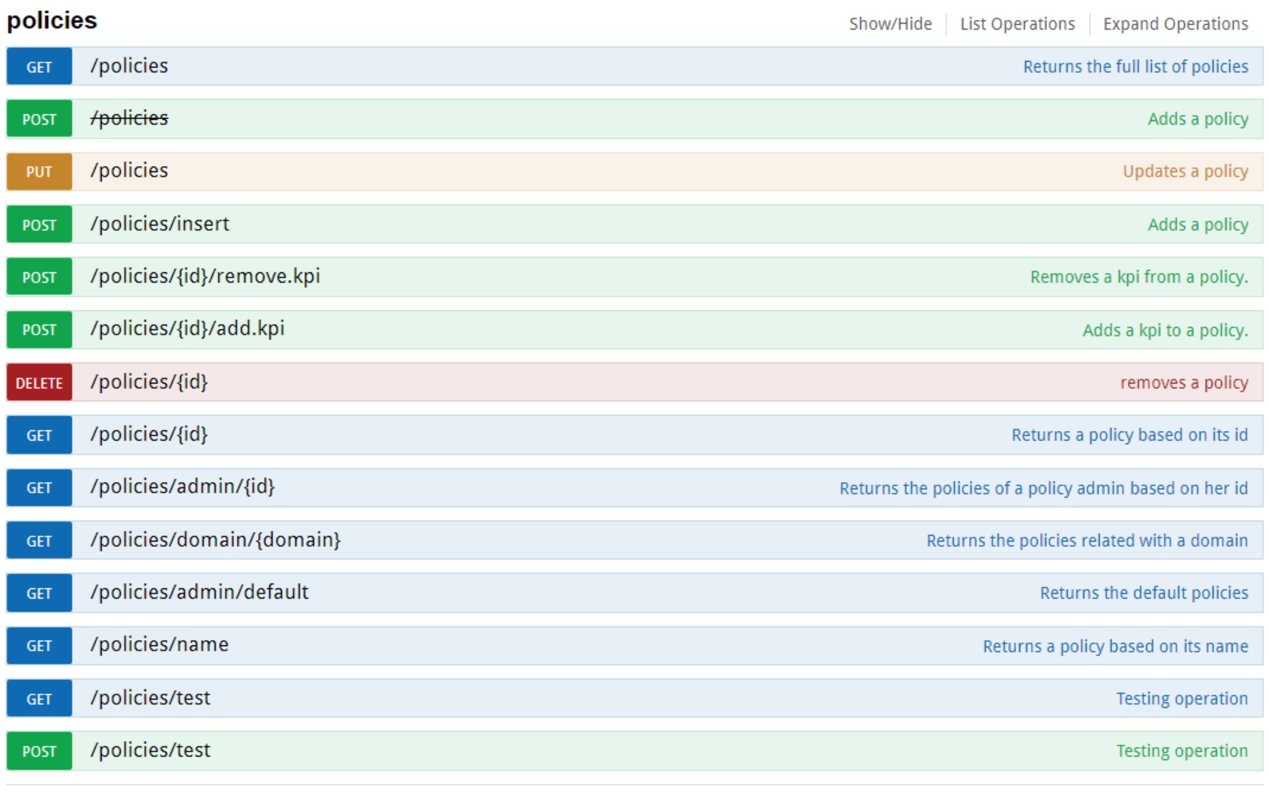


kpis

Show/Hide | List Operations | Expand Operations

DELETE	/kpis/{id}	removes a kpi
GET	/kpis/{id}	Returns a kpi based on its id
GET	/kpis/goal/{goal}	Returns the full list of kpis by a goal
GET	/kpis	Returns the full list of kpis
POST	/kpis	Adds a kpi
PUT	/kpis	Updates a kpi
GET	/kpis/domain/{domain}	Returns the full list of kpis by a domain
GET	/kpis/actor/{actor}	Returns the full list of kpis by an actor
POST	/kpis/{id}/add.goal	adds a new goal to a kpi
GET	/kpis/{id}/add.actor	Adds an actor to a kpi based on its id
GET	/kpis/{id}/remove.actor	Removes an actor from a kpi based on its id
POST	/kpis/{id}/add.data.model/{datamodelid}	Adds a data model to a kpi.
POST	/kpis/{id}/remove.data.model	Removes a data model from a kpi.
GET	/kpis/test	Testing operation

FIGURE 31: WEB METHODS RELATED WITH KPIS


policies

Show/Hide | List Operations | Expand Operations

GET	/policies	Returns the full list of policies
POST	~~/policies~~	Adds a policy
PUT	/policies	Updates a policy
POST	/policies/insert	Adds a policy
POST	/policies/{id}/remove.kpi	Removes a kpi from a policy.
POST	/policies/{id}/add.kpi	Adds a kpi to a policy.
DELETE	/policies/{id}	removes a policy
GET	/policies/{id}	Returns a policy based on its id
GET	/policies/admin/{id}	Returns the policies of a policy admin based on her id
GET	/policies/domain/{domain}	Returns the policies related with a domain
GET	/policies/admin/default	Returns the default policies
GET	/policies/name	Returns a policy based on its name
GET	/policies/test	Testing operation
POST	/policies/test	Testing operation

FIGURE 32: WEB METHODS RELATED WITH POLICIES

Finally, the Backend allows for asynchronous communication with the PDT via web sockets. This is crucial when other components perform updates that the Backend can be aware of, so that the PDT can modify the UI reflect to those updates, without having to continuously invoke the Backend to ask for potential pending updates. This helps to improve the overall UX of the end-user. For instance, when an analytical tool finishes its executions and produces some results, the *Data Acquisition and Analytic* API notifies the Backend, which will trigger sending a message via this web socket to the PDT, so that the latter can update the status of a pending job. However, the API and the DTOs that need to be exchanged are still under definition, therefore, this functionality will be documented in the second version of this deliverable, when it will have been implemented.

2.3 Baseline technologies and tools

2.3.1 Policy Modelling Editor

As described in the deliverable D5.2 “Cross-sector Policy Lifecycle Management: Design and Open Specification 1”, the PME is developed using the open source Angular framework [4]. For UI components and controls we use the Angular Material library [5].

2.3.2 Policy Development Toolkit

As described in the deliverable D5.2 “Cross-sector Policy Lifecycle Management: Design and Open Specification 1” Section 4.2, PDT is built as a Single Page Application developed using the open source Angular framework [4][3]. For UI components and controls we use the Angular Material library [5].

The PDT is using Docker to encapsulate the web serving functionality along with the NGINX [6] web server.

2.3.3 Data Visualization

As the Data Visualization will be part of the PDT, and to facilitate the integration with it, the data visualization component has been developed using the same base technology, AngularJS v10 [4].

The main JavaScript libraries used for this first prototype are:

- Leaflet [7] is used to plot the heatmap
- Amcharts [8] is used to plot the line and the gauge charts

The detailed instructions about how to start this component as standalone component are detailed into the readme file that accompanies the source code of the component, but as summary it can be started with a NodeJS server or using a Docker container.

2.3.4 PDT Backend

The backend of PDT has been implemented using Java SDK 8. It does not make use of any framework like Spring, but it relies on the standard JDK. For the implementation of the REST services, it makes use of the javax servlet API 3.1.0 while it also makes use of an embedded tomcat, as the servlet container that the web services are deployed. We rely on the Apache Embedded Tomcat 8 [9]. By doing this, there is no need to maintain an additional standalone container like Tomcat, Glassfish or similar, rather than the java virtual machine starts the embedded one inside its process. Moreover, maven 3 was used as the tool for the automation of the build process. For the persistent storage of this component, we relied on the LXS relational datastore. However, this is not mandatory and therefore, the Backend component of the PDT can switch to other relational datastore, with minimal changes in the source code.

3 Source code

3.1 Availability

3.1.1 Policy Modelling Editor

The source code of the first prototype of the Policy Modelling Editor component is available at:

- Project GitLab repository: http://snf-877903.vm.okeanos.grnet.gr/chris/policy-model-editor

3.1.2 Policy Development Toolkit

The source code of the first prototype of the Policy Development Toolkit component is available at:

- Project GitLab repository: http://snf-877903.vm.okeanos.grnet.gr/kostas/pdt

3.1.3 Data Visualization

The source code of the first prototype of the Data Visualization component is available at:

- Project GitLab repository: http://snf-877903.vm.okeanos.grnet.gr/miquel.mila/visualization

3.1.4 PDT Backend

The Backend of the PDT is currently released under open source license, and its source code has been uploaded and is publicly available at:

- Project GitLab repository: http://snf-877903.vm.okeanos.grnet.gr/pavlos/pdt-backend

It also provides the tools to compile the source code and also build a docker image that can deploy and run this component.

3.2 Deployment

3.2.1 Policy Modelling Editor

The Policy Modelling Editor will follow the Cloud provider setups and Kubernetes staging.

3.2.2 Policy Development Toolkit

The Policy Development Toolkit will follow the Cloud provider setups and Kubernetes staging.

3.2.3 Data Visualization

The Data Visualization will follow the Cloud provider setups and Kubernetes staging.

3.2.4 PDT Backend

In order to deploy and use this component, the administrator has firstly to download the source code from the project's repository.

```
git clone http://snf-877903.vm.okeanos.grnet.gr/pavlos/pdt-backend.git
```

And later compile the source code. Assuming maven 3 and Java 8 are pre-installed, it needs to execute the following:

```
mvn clean install
```

The component can be executed via a java JRE tools, but we also provide a Dockerfile to fully containerize the deployment and integrate it with the LXS datastore. Firstly, they would need to build a docker image locally. Assuming that docker has been already installed in the host machine, they should execute the following:

```
docker build –t pdt-backend .
```

This will create a docker image that can be found in the host machine's catalogue. To execute the component, a docker-compose file has also been provided in the GitLab, which makes use of the LXS datastore. The latter must have been already available in the host machine's docker catalogue. To start everything, the administrator should execute the following:

```
docker-compose up
```

The docker-compose file exposes the 54735 port to the host machine, where the servlet container listens to. Therefore, the administrator can now open their favorite browser and put the following URL, which will open the swagger documentation:

```
http://localhost:54735
```

Conclusions

This deliverable provided a description of the software components of the Policy Management Framework Layer and Policy Development Toolkit Layer, which consists of the frontend part of the PolicyCLOUD platform and allow policy makers to create, update and evaluate policies. At this phase of the project the prototypes are available at component level and not integrated in the PolicyCLOUD platform. For each component, a description of its APIs, specification of the main functionalities, and description of the source code is provided.

References

[1] PolicyCLOUD. D5.2 "Cross-sector Policy Lifecycle Management Design and Open Specification 1". Armend Duzha et al. 2020

[2] PolicyCLOUD. D2.2 "Conceptual Model and Reference Architecture", Panayiotis Michael et al. 2020

[3] Swagger [Online], Available at: https://swagger.io/tools/swagger-ui/.

[4] Angular [Online], Available at: https://angular.io.

[5] Angular Material [Online], Available at: https://material.angular.io/.

[6] NGINX [Online], Available at: https://www.nginx.com/

[7] Leaflet [Online], Available at: https://leafletjs.com/

[8] Amcharts [Online], Available at: https://www.amcharts.com/

[9] Apache Tomcat [Online], Available at: http://tomcat.apache.org/