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

D5.7 CROSS-SECTOR POLICY LIFECYCLE MANAGEMENT: SOFTWARE PROTOTYPE 3

Dissemination Level	PU
Due Date of Deliverable	31/10/2022, Month 34
Actual Submission Date	27/10/2022
Work Package	WP5 Cross-sector Policy Lifecycle Management
Task	T5.2, T5.3, T5.4, T5.5 & T5.6
Type	Demonstrator
Approval Status	
Version	v1.0
Number of Pages	p.1 – p.45

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	09/09/2022	Updated ToC and assignments	Samuele Baroni
0.2	09/12/2022	Contribution from ICCS	Kostas Moutselos
0.3	09/20/2022	Contribution from ATOS	Ana Luiza Pontual Miquel Mila Prat
0.4	09/27/2022	Contribution from LXS	Pavlos Kranas
0.5	09/27/2022	Contribution from OKS	Chris Maragkos
0.6	09/30/2022	Consolidated version ready for review	Samuele Baroni
0.7	10/03/2022	Internal review	Yosef Moatti
0.8	10/17/2022	Internal review	Giannis Ledakis
0.9	10/26/2022	Reviewers comments addressed	All contributors
1.0	10/27/2022	Quality check Submitted version	Argyro Mavrogiorgou

Author List

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

Abbreviations and Acronyms

Abbreviation/Acronym	Definition
API	Application Programming Interface
AT	Analytical Tool
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 is the third and final update of deliverable Cross-sector Policy Lifecycle Management: Software Prototype, whose first two versions were delivered respectively in November 2020 and October 2021. It describes in detail 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. 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 third and final iteration of "Cross-Sector Policy Lifecycle Management: Software prototype", and covers tasks T5.2, T5.3, T5.4, T5.5 and T5.6. 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 D5.6 "Cross-Sector Policy Lifecycle Management: Design and Open Specification 3" [1], delivered in August 2022, which provided the final overall architecture and design specifications of the PME and PDT.

1.2 Summary of changes

We have updated and revised the whole document (including figures) to reflect the current work done by M34 (October 2022). The main changes from the last version D5.5 [2] concerns the Policy Development Toolkit Layer and the Visualization Layer (sections 2.1.2, 2.1.3, 2.2.2 and 2.2.3), since the architecture of the PolicyCLOUD platform has just received minor modifications in this last period.

1.3 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 the technologies used for their development.
- Section 3 describes where the source code is available.
- Finally, Section 4 provides the conclusion and references of the document.

2 Prototype overview

In this section we provide the description of the prototype implementation of the Policy Management Framework Layer and Policy Development Toolkit Layer. In Figure 1 it is possible to appreciate the position and role of these layers in the overall architecture of the PolicyCLOUD platform.

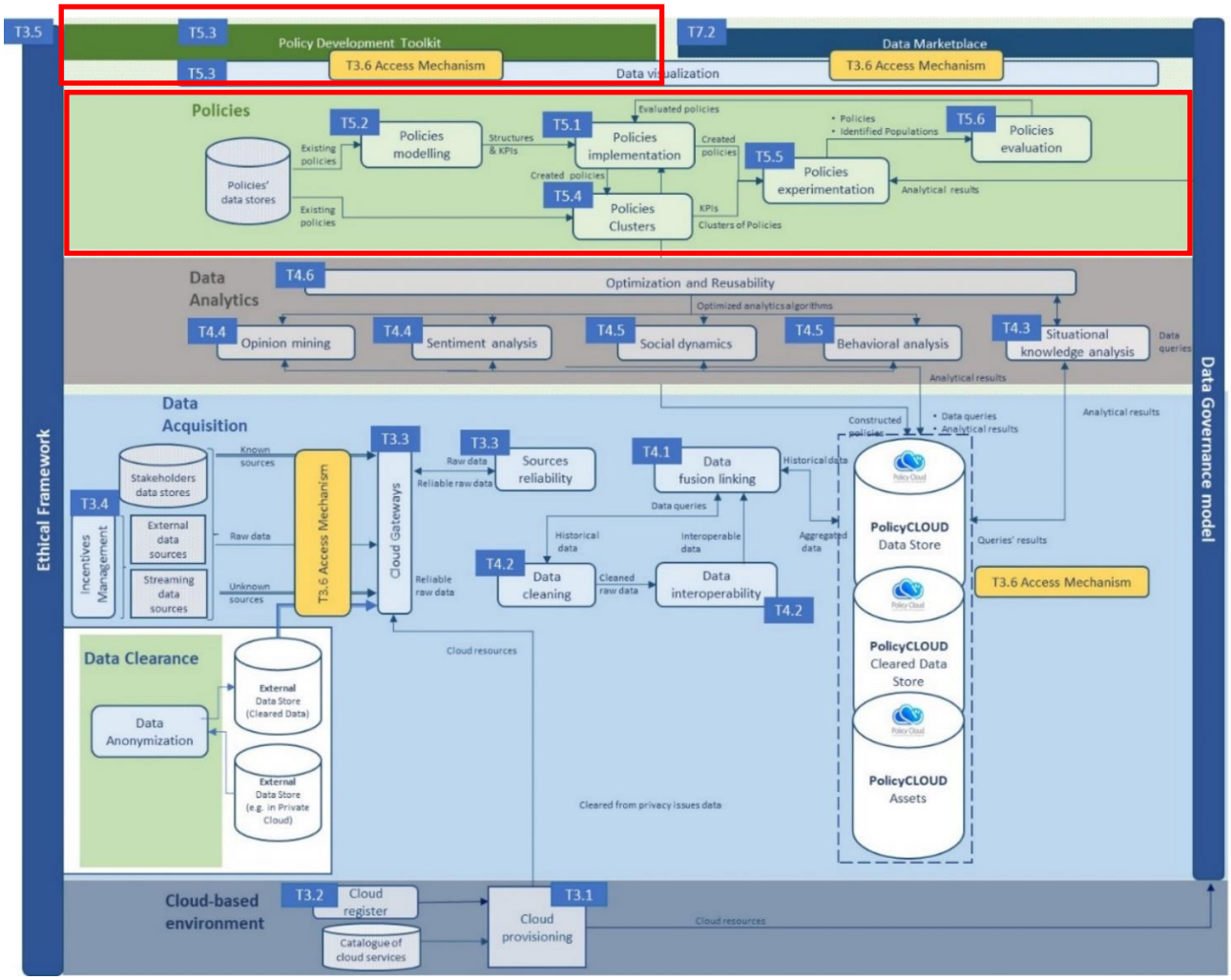

FIGURE 1 - POLICYCLOUD SYSTEM ARCHITECTURE

2.1 Main components of the prototype

2.1.1 Policy Modelling Editor

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

2.1.2 Policy Development Toolkit

As described in deliverable D2.7 "Conceptual Model and Reference Architecture 3"[3], the PDT interacts with components from Layer 3 - Policies Management Framework (T5.1, T5.2), Analytical Tools (WP4) and the PDT Backend.

PDT provides the integrated prototype of the following components as described in the deliverable D5.6 "Cross-sector Policy Lifecycle Management: Design and Open Specification 3" [1]:

- Policy Model Viewer
- Policy Evaluations / Analytics

Integration with the Backend has been implemented, utilising Docker images. Specifically, after a new GitLab commit, Docker-Compose orchestrates the procedure of temporarily stopping the provision of the web application, creating intermediate docker images to construct the proper compiling environment, compiling the application, transferring the executable app into the appropriate Nginx web server image which in co-operation with the SSL-keys provision/renewal container, provide the service for the new PDT web application.

Thereafter, PDT can retrieve default policy models in the structured JSON format which represent the Policy Models (PMs). So, the functionalities included in this prototype are:

- The policy maker 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 policy maker can see the parameters and their values related with each AT and invoke the execution of the respective AT.
- The policy maker can retrieve a list of the submitted ATs jobs along with the current status of their execution. Once an analytics job is in finished state, he/she can retrieve the results to be displayed in the Visualisation Dashboard.
- Finally, the policy maker can see a short overview of the current selected PM.

The aforementioned functionalities are implemented as a policy wizard component, comprised by four steps. The relative UI components are described in Section 2.2.2.

2.1.3 Data Visualization

The Data Visualization is a key component responsible of showing, in a more visual and easily interpretable way, the results of the different data analysis carried out on PolicyCloud platform over the different data sources provided. It is composed of several chart templates where the data harvested by the different analytical tools registered in the platform can be displayed.

Taking into account that the input of the data visualization component are the JSONs files that the different analytical functions developed uses in the platform has as output, a close work between the developers of these analytical functions has been done in order to create a valid flow of data that enable the visualisation component to reuse the chart templates to display data from different scenarios.

More information about the visualization component development is detailed into the deliverable D5.6 "Cross-sector Policy Lifecycle Management: Design and Open Specification 3" [1].

It is now fully integrated into the PDT, and can be accessed through it, as a final step of the chosen analysis. The final version of this component includes all graphs defined in the scope of the project. It is possible to see it in D5.6 "Cross-sector Policy Lifecycle Management: Design and Open Specification 3" [1].

The main chart templates used in real scenarios are:

- A Heatmap, requested by the "Participatory Policies Against Radicalization" use case of Maggioli and for the "Facilitating urban policy making and monitoring through crowdsourcing data analysis" use case of Sofia:

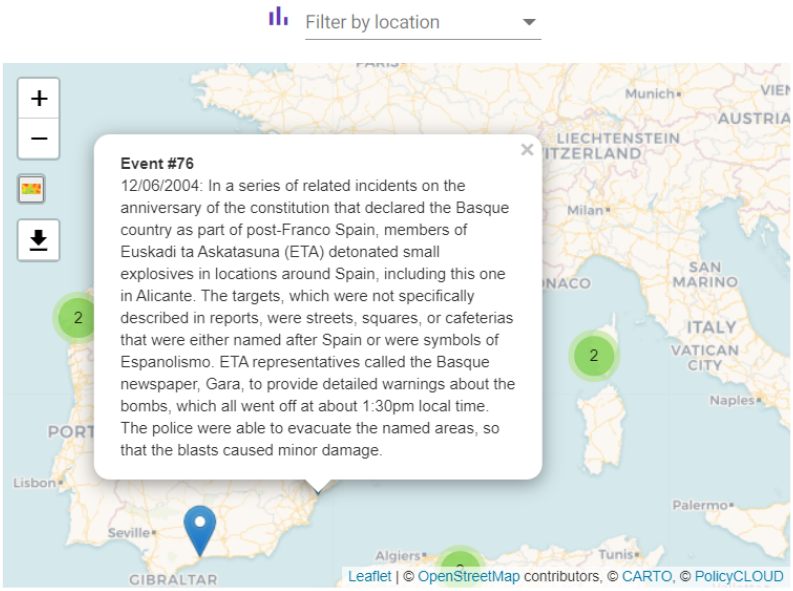

FIGURE 2 - EXAMPLE OF THE HEATMAP FOR MAGGIOLI AND SOFIA USE CASE (EVENT LOCATOR)

This Heatmap is intended to show, immediately, all incidents that occurred in a chosen region. Whenever a region is selected, the heatmap shows all incidents that took place in it. It can be filtered by location and by year, and it is possible to see the incident details, when clicking on it. Users can switch regions as they wish. In Sofia's case, the adapted Heatmap is focused on Sofia city events and can be filtered by Sofia's districts, by year.

The heat map chart template can also show "hot" zones (zones with a bigger number of incidents) using colours, instead of numbers:

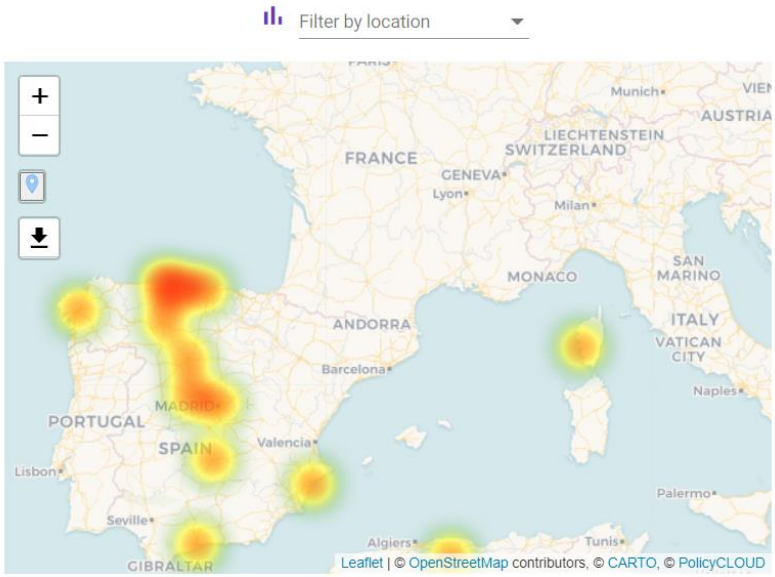

FIGURE 3 – EXAMPLE OF HEATMAP FOR MAGGIOLI USE CASE ("HOT" ZONES)

Among this chart template functionalities, there are some options to show the same data using other chart templates, like horizontal bars, accumulated charts, pie chart and tables.

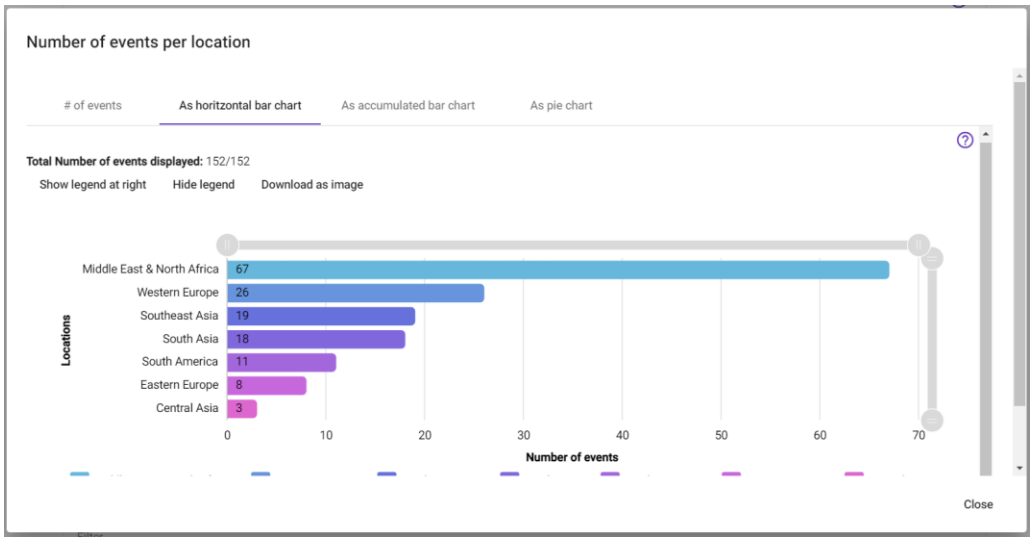

FIGURE 4 - DATA DISPLAYED IN THE HEATMAP IN DIFFERENT WAYS

- Another template is the Heat table chart, requested by Sarga for the "Politika price points plus multiples prices" scenario, where three data dimensions recovered from the Politika external tool, can be plotted accordingly to the pilot needs.

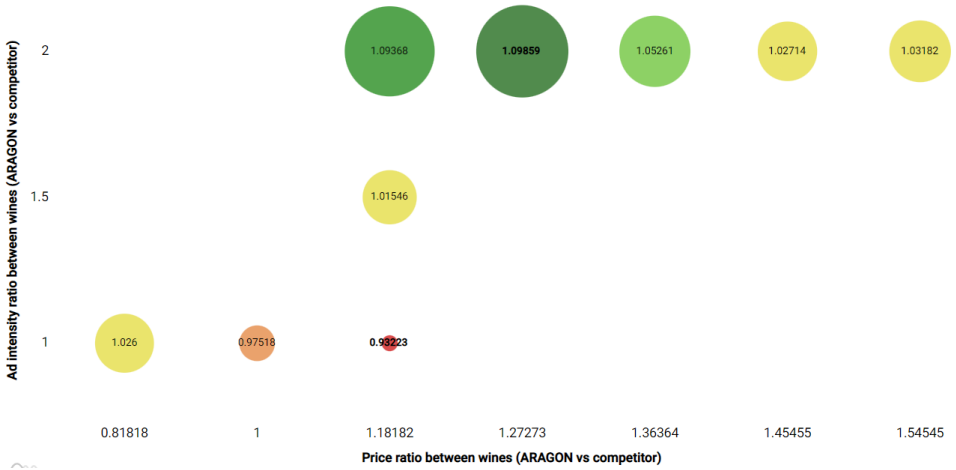

FIGURE 5 - HEATMAP TABLE CHART

This graph has emerged after an in-depth analysis of the best way to visually express the comparison between two brands of wine, considering the price and advertising intensity ratio between both, and the consequently purchase motivation for the one used as the used as a comparison reference; and to be able to quickly arrive at the best parameters obtained.
It is a 3-parameter graph, in which the values obtained for the variable shown in the centre of each circle determine both its size and its colour, thus making it much more visual and consequently more useful for decision-making.

- A Line chart used in different scenarios like the "Price evolution" of Sarga and the "Time series forecasting" of Sofia:
    - In the case of Sarga, the chart template can show the evolution of the price of a wine for different vendors providers. This chart templates enable users to find values that are upper or down a specific value, and focusing this functionality in this specific example this makes easier to check when the price of a wine is over or down a specific price threshold, in order to discover if a vendor is aligned with the producer or not. When prices threshold is crossed, in addition to the points being shown in red, which helps visual perception of the alarm, a WARNING window also appears, where the values that have caused this alarm can be seen, which greatly increases the visual impact. It is also interesting to comment that the threshold values can be dynamically modified by the user, resulting in a more iterative graph.

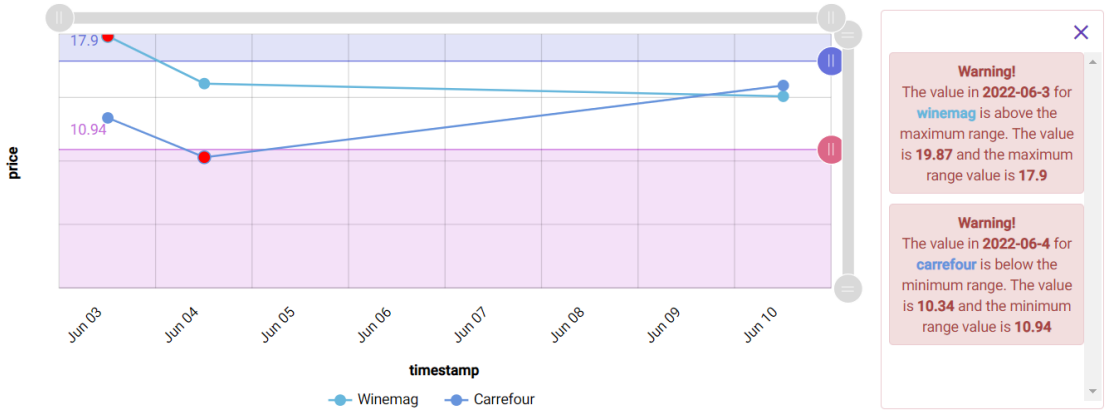

FIGURE 6 - LINE CHART FOR THE PRICE EVOLUTION SCENARIO

- In the case of Sofia Time series forecasting, the chart plots the evolution of the incidents and the prediction in time of the evolution of these incidents. In this case, the chart shows a line, for the historical values of incidents, and a line wrapped in a "shadow" with different colour, for predicted values of incidents. This "shadow" is defined by the margin of error of the values.

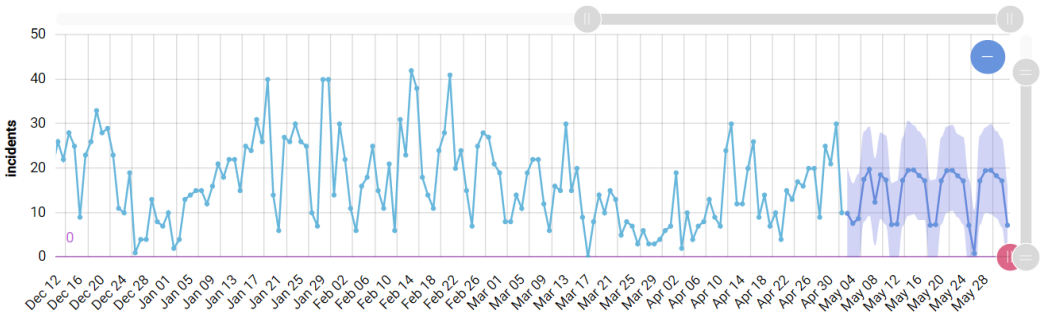

FIGURE 7 - LINE CHART FOR THE TIME SERIES FORECASTING

- A Tag cloud chart used in the "Trend Analysis" scenario of SARGA. This chart displays the output of the related analytical function, showing the most trend words related to the selected element: in this case, between some Wines, Wineries or Regions.

FIGURE 8 - TAG CLOUD FOR TREND ANALYSIS

- A custom dashboard used in the "Opinion on social networks" scenario of Sarga, where different chart templates like a heatmap, a gauge and a line chart are used all together, in order to plot different information at same time, harvested by the platform for this type of scenario. This kind of dashboards are a powerful tool for decision makers, since it unifies in a single visualization, much of the information necessary in decision making, allowing its comparison, and a better vision of the whole.

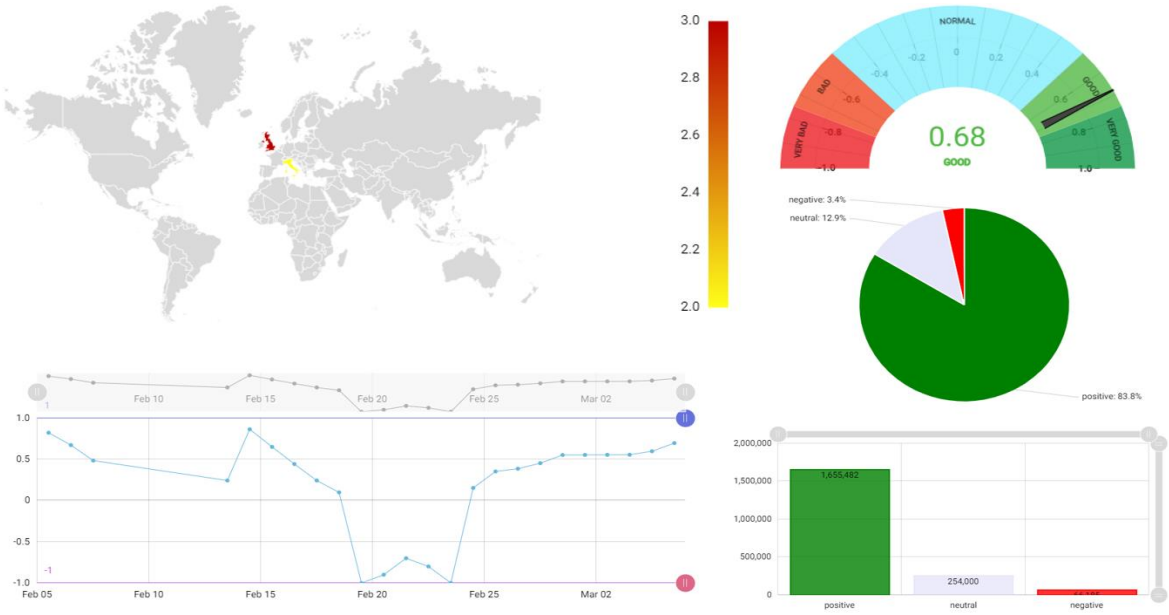

FIGURE 9 - CHARTS USED IN THE OPINION ON SOCIAL NETWORKS DASHBOARD

- Different types of bar chart templates have been designed and implemented, taking into account the needs of the Sofia scenarios, like the road infrastructures. In these charts the data related with the road infrastructure can be plotted in different bar charts like clustered or stacked columns, horizontal bar charts and in a line and/or a radar chart.

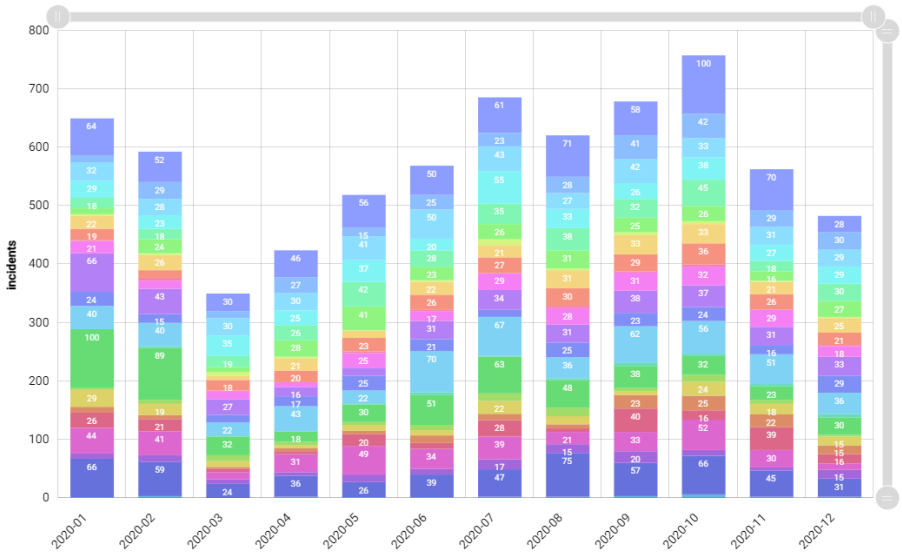

FIGURE 10 - BAR CHART TEMPLATE FOR THE ROAD INFRASTRUCTURE SCENARIO

2.1.4 PDT Backend

The PDT backend provides support to the whole PDT, and can be considered as the layer between the user interfaces that the policy maker can use to have access 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 [4]. As a result, it serves four main purposes:

- Allows storing, modifying and retrieving *policies,* along with their relevant corresponding entities (i.e., KPIs, stakeholders, etc.)
- Hold 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
- Invoke 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
- Allow for the retrieval of the results of the analytical tools, which can be later fed to 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 in the *Data Acquisition and Analytic* layer. Those tools can be retrieved either by name, by the domain they are related to or according to 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 can be relevant to the entities of the *policies*
- 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, related to its *job*
- Add, remove, modify, and retrieve *KPIs,* based on their name, their corresponding domain or 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*
- Store combined types of results of analytical tools, thus giving the opportunity to the end-user to switch between different views of the same results
- Correlate policies with the end-users that have defined them
- Retrieve policies depending on the user that has issued the request
- Correlate *jobs* with the users that initially submitted a request to execute an analysis
- Retrieve the results of the *jobs* owned by a given end-users
- Cancel the execution of an analysis if it exceeds the defined maximum response time allowed
- Provides a warning when the result of an analysis exceeds the maximum number of characters that can be efficiently processed by the visualization tools at a later phase

Regarding the interaction with the *Data Acquisition and Analytic* layer, the corresponding APIs have been defined in deliverable D4.6. 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, its attributes, related entities, and relations between them. It should be highlighted that the datastore which is being used by this component is not the central data repository of PolicyCLOUD, as it is used for storing 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, along with the results of the analytical tools, while at the same time it aims at 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 both to 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 11 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 guided by the platform to continue in the next page to select existing KPIs.

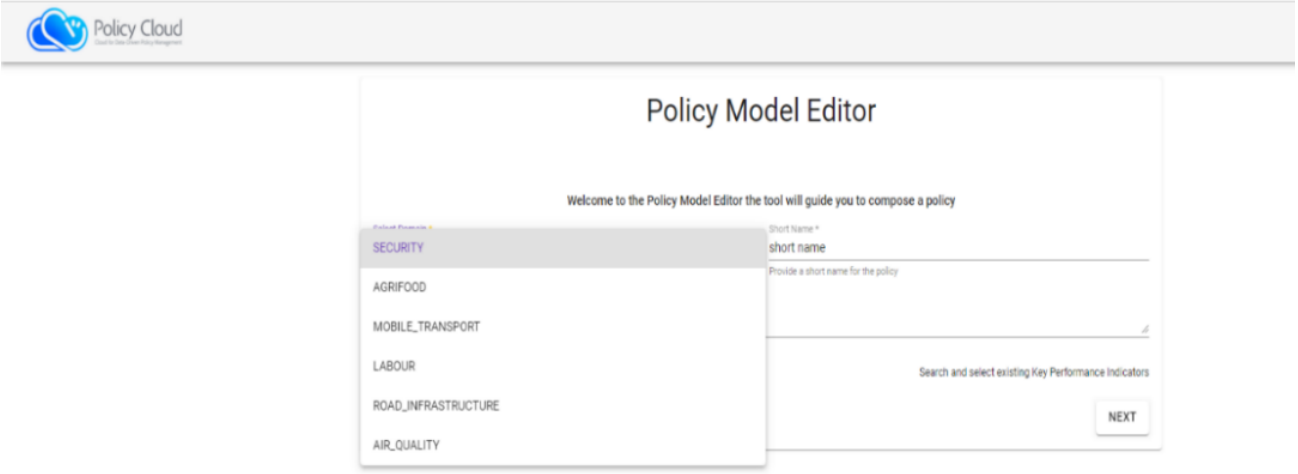

FIGURE 11 - POLICY MODEL EDITOR HOME PAGE

Figure 12 shows the Existing KPIs selection interface, in which the end-user has the capability to add or to select existing Actors that are included in the Policy Model domain.

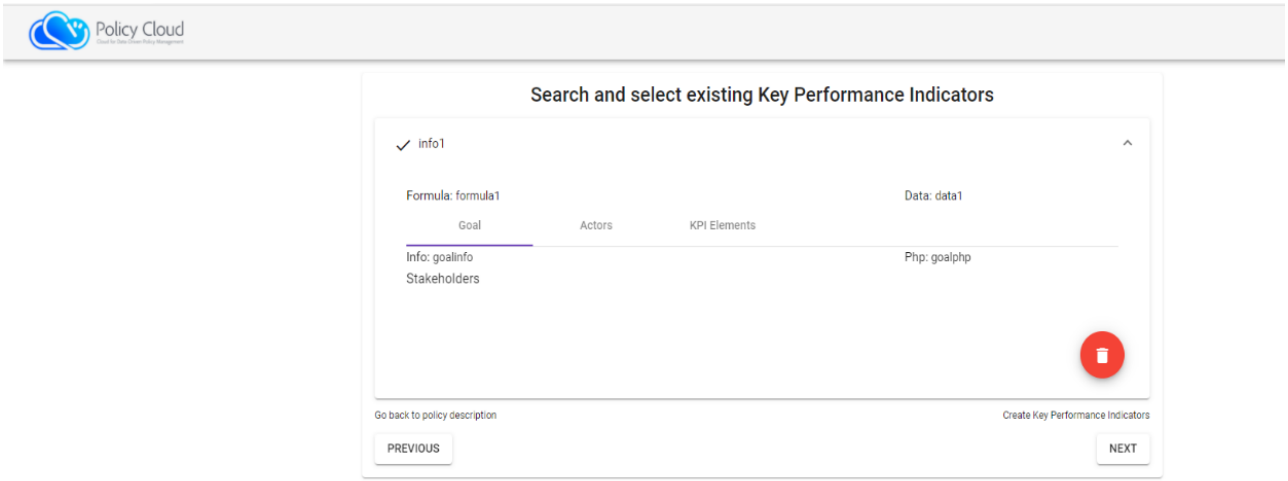

FIGURE 12 - EXISTING KPIS SELECTION

In the Add new KPI interface (see Figure 13) 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.

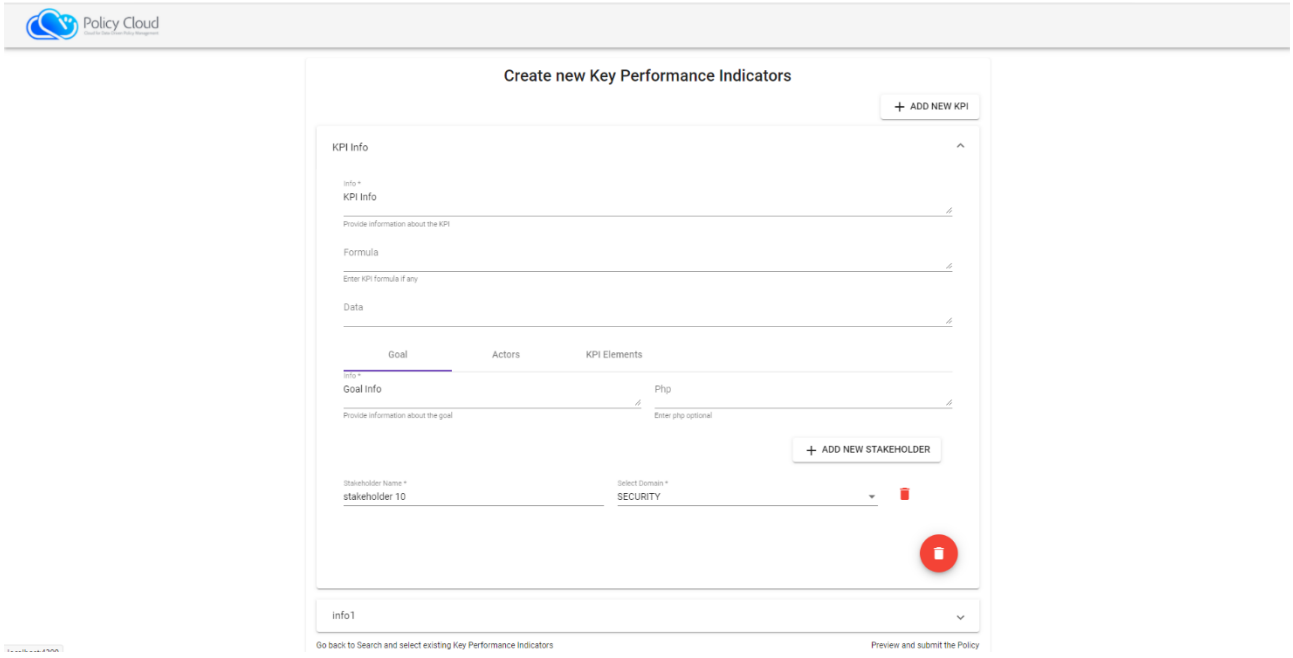

FIGURE 13 - ADD NEW KPI

It should be noted that the user has the following capabilities: (i) to provide description of the Data Model and add or select Actors (Figure 14 below), (ii) to select the relevant Analytical Tools from a specific list (drop-down menu) (Figure 15), (iii) to provide the parameter values that comes for the Analytical Tools (Figure 16), (iv) to review and submit the Policy Model (Figure 17).

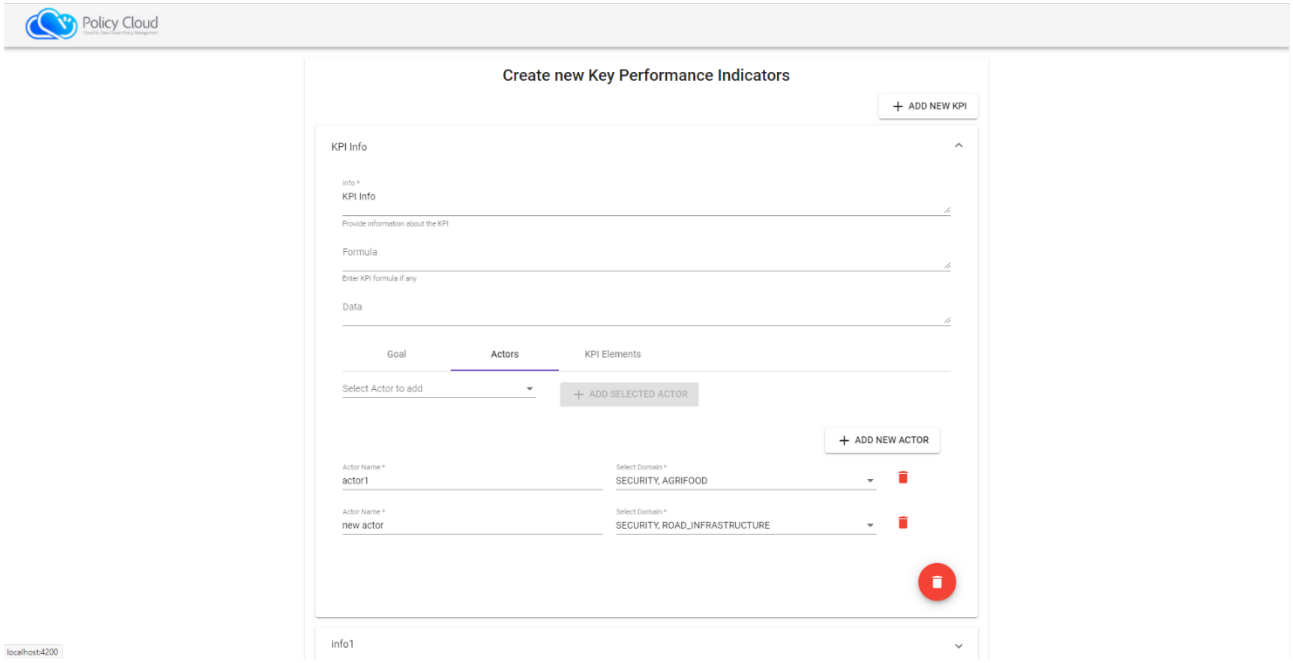

FIGURE 14 - ADD NEW ACTOR

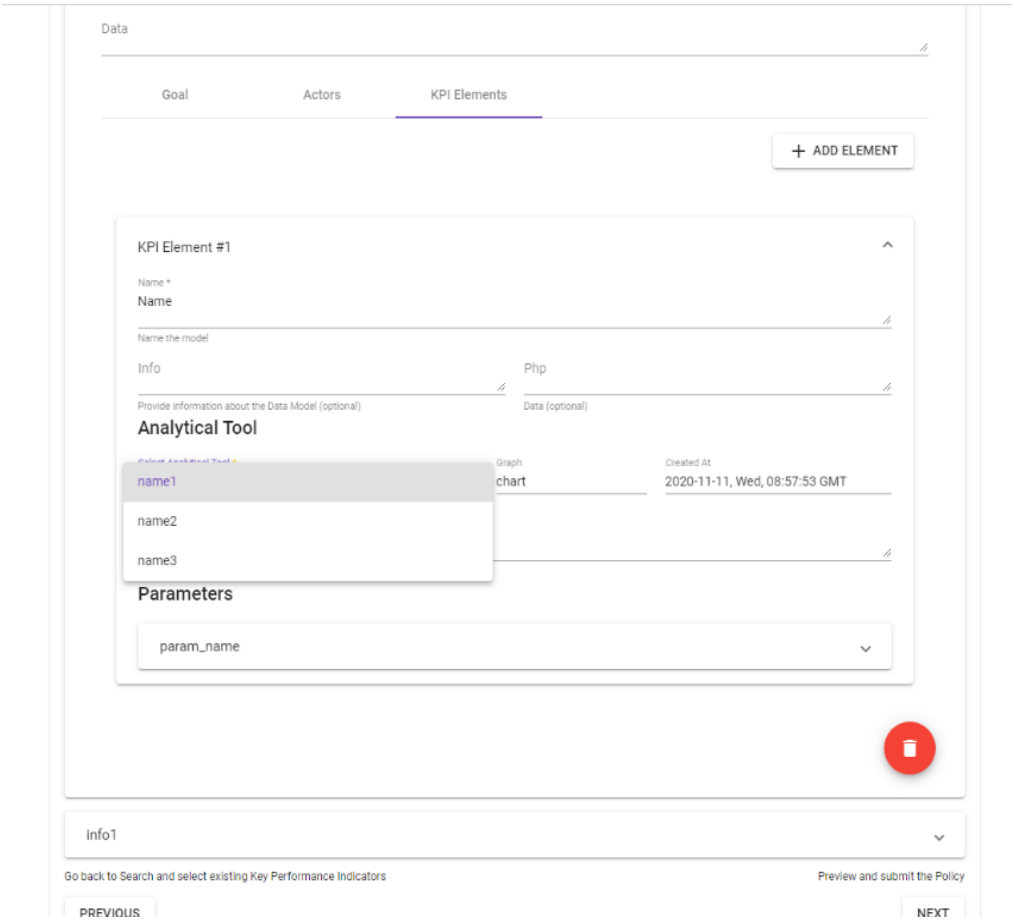

FIGURE 15 - ANALYTICAL TOOL SELECTION

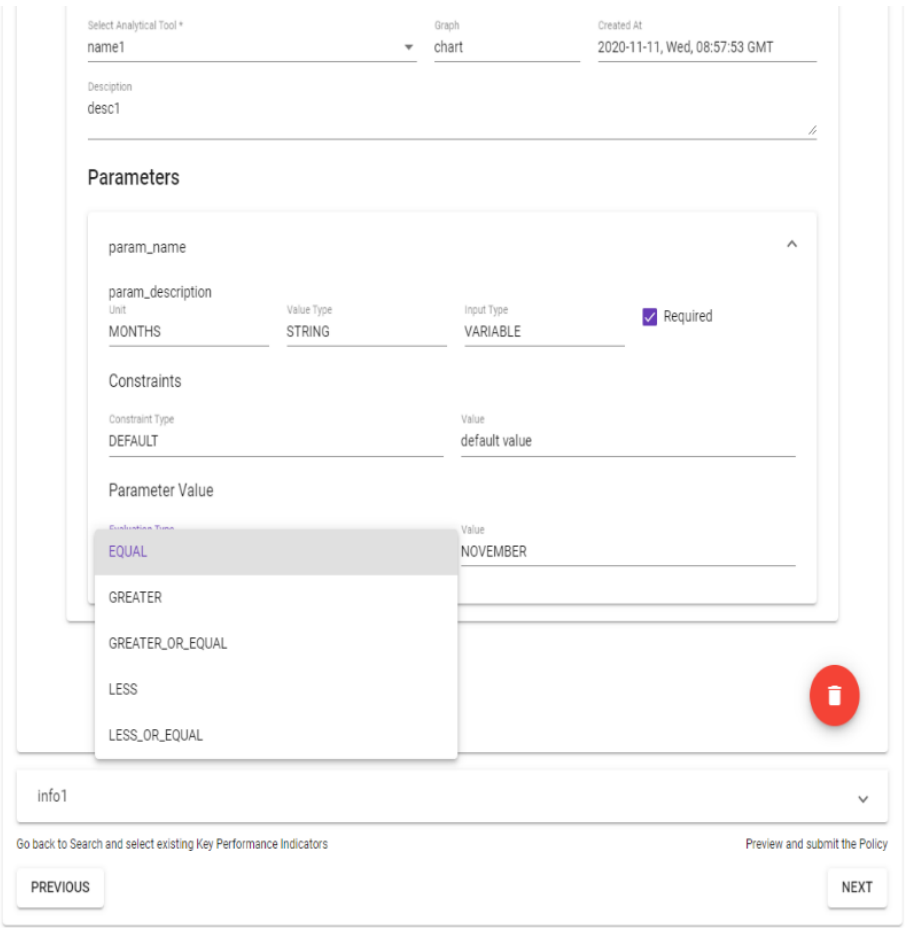

FIGURE 16 - SET PARAMETER VALUE

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.

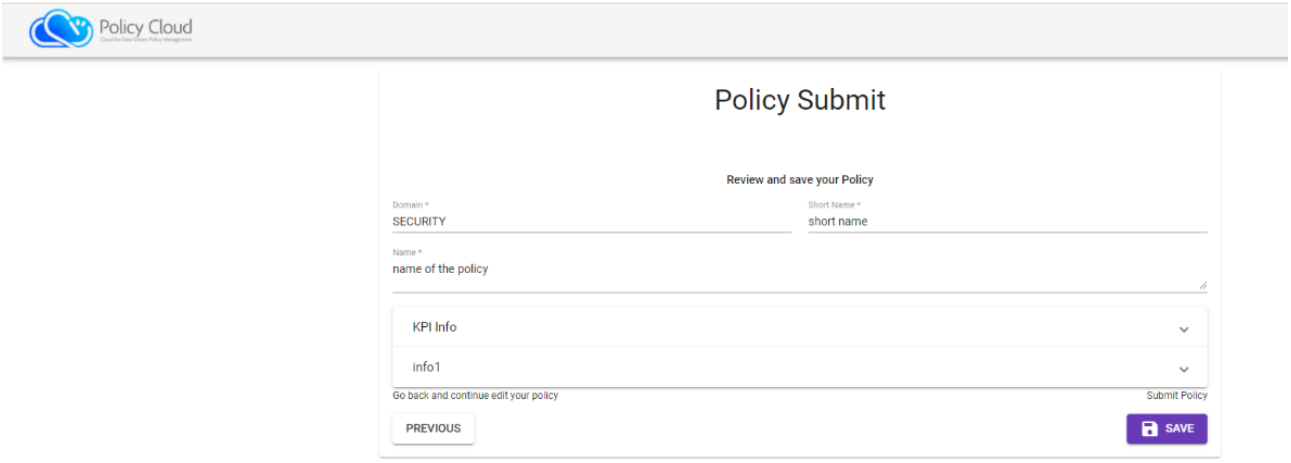

FIGURE 17 - REVIEW AND SUBMIT

FIGURE 18 - 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.

Figure 19 below shows the landing web page of PDT (after the Authentication procedure). The page shows the four-step policy wizard, with the first step as the default location. Here the user selects a policy from a list of available policy models.

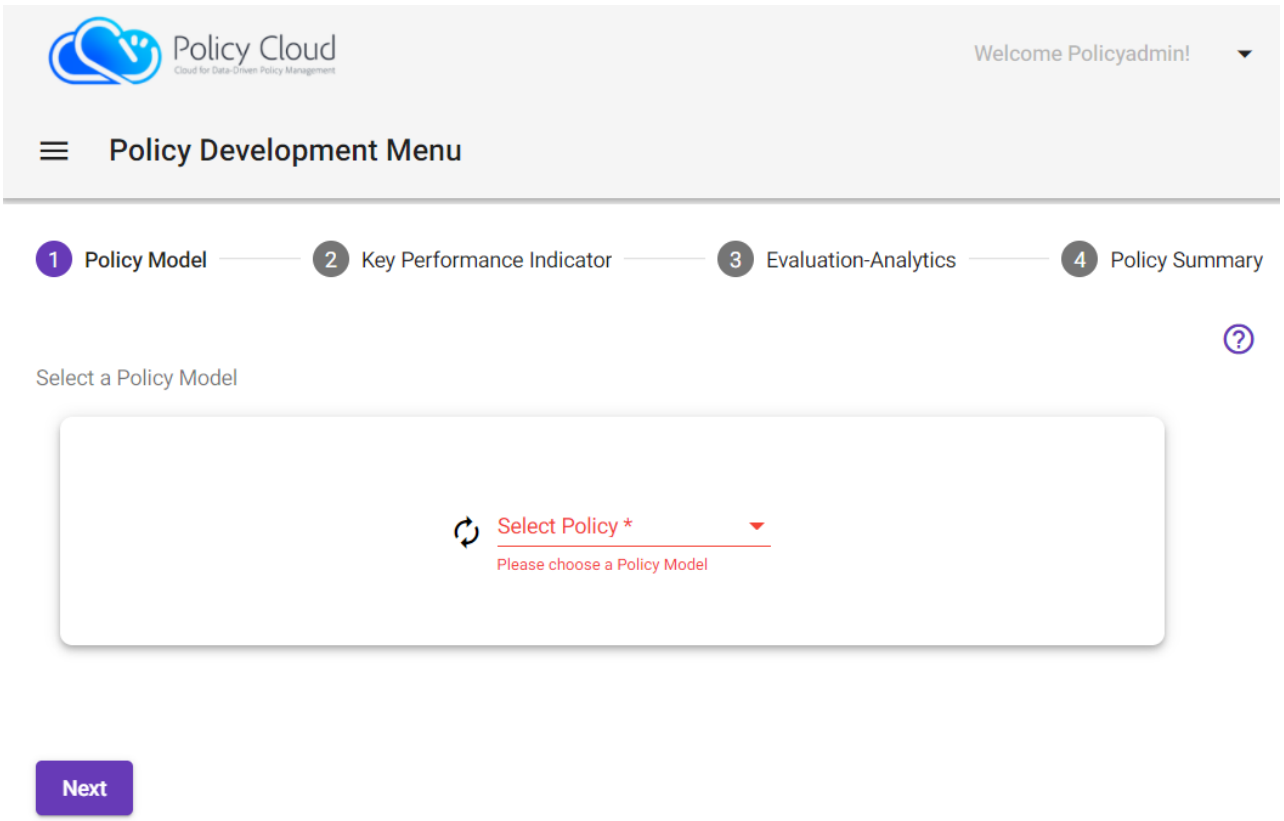

FIGURE 19 - POLICY SELECTION EVALUATION WIZARD

Figure 20 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.

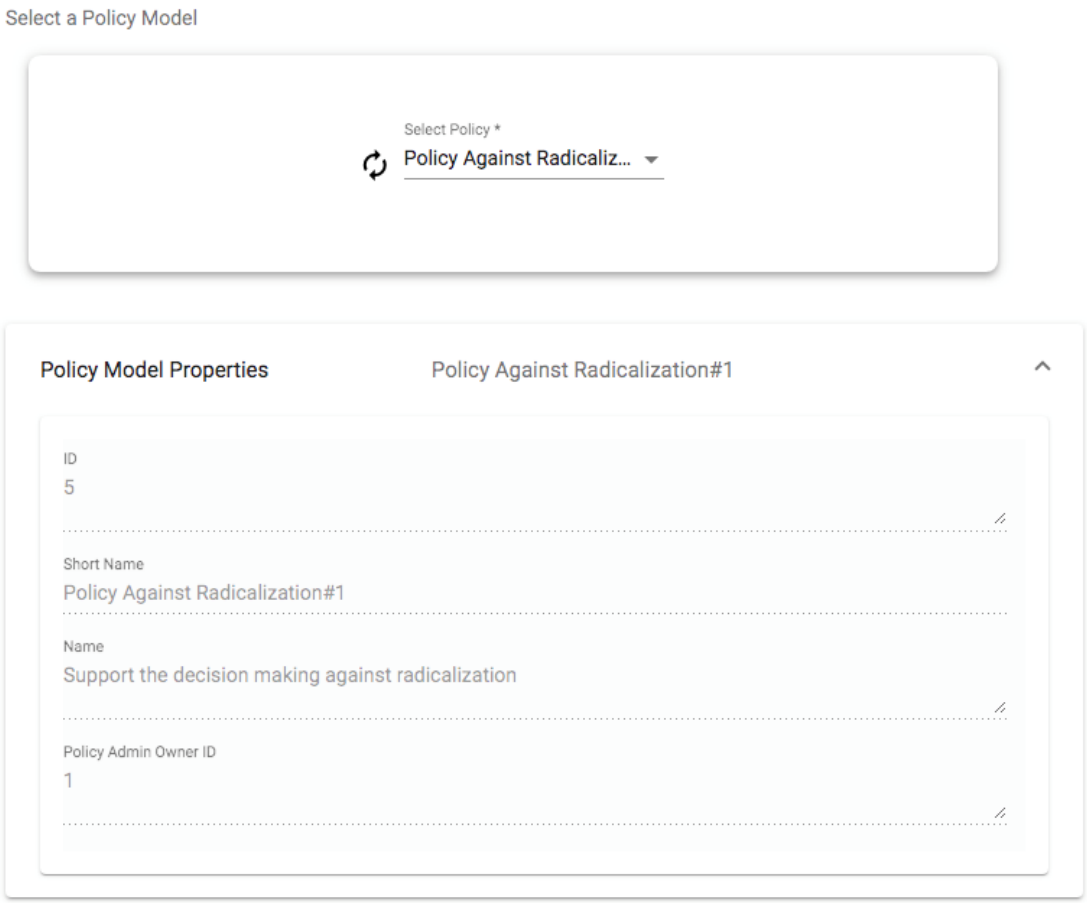

FIGURE 20 - POLICY SELECTION AND POLICY PROPERTIES

Figure 21 below displays the step for the KPI selection and Figure 22 shows the properties of the selected KPI, e.g., formula, data, domain, goal, actors and stakeholders.

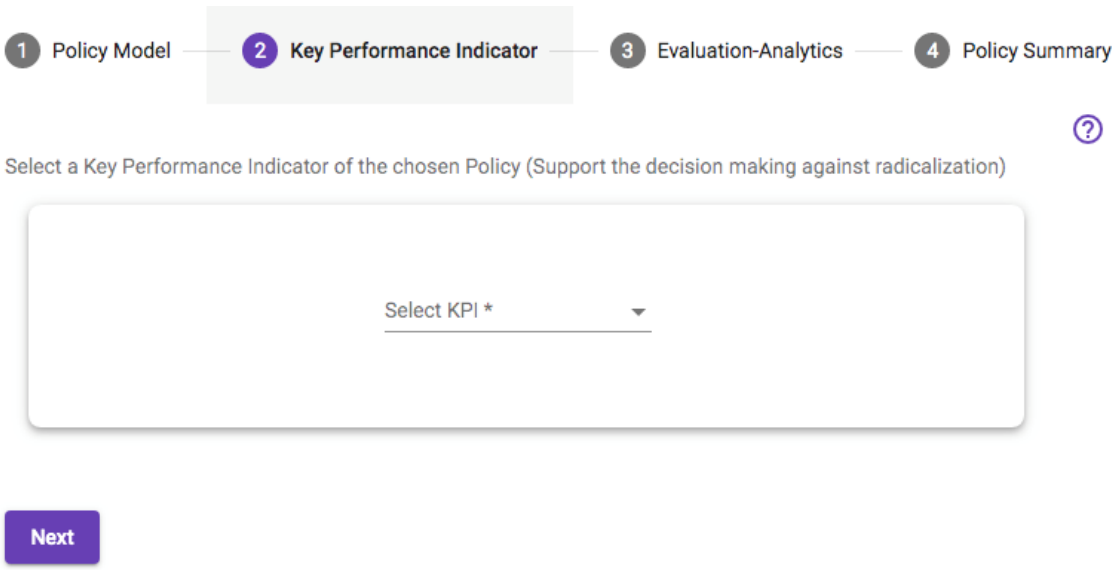

FIGURE 21 - KPI SELECTION

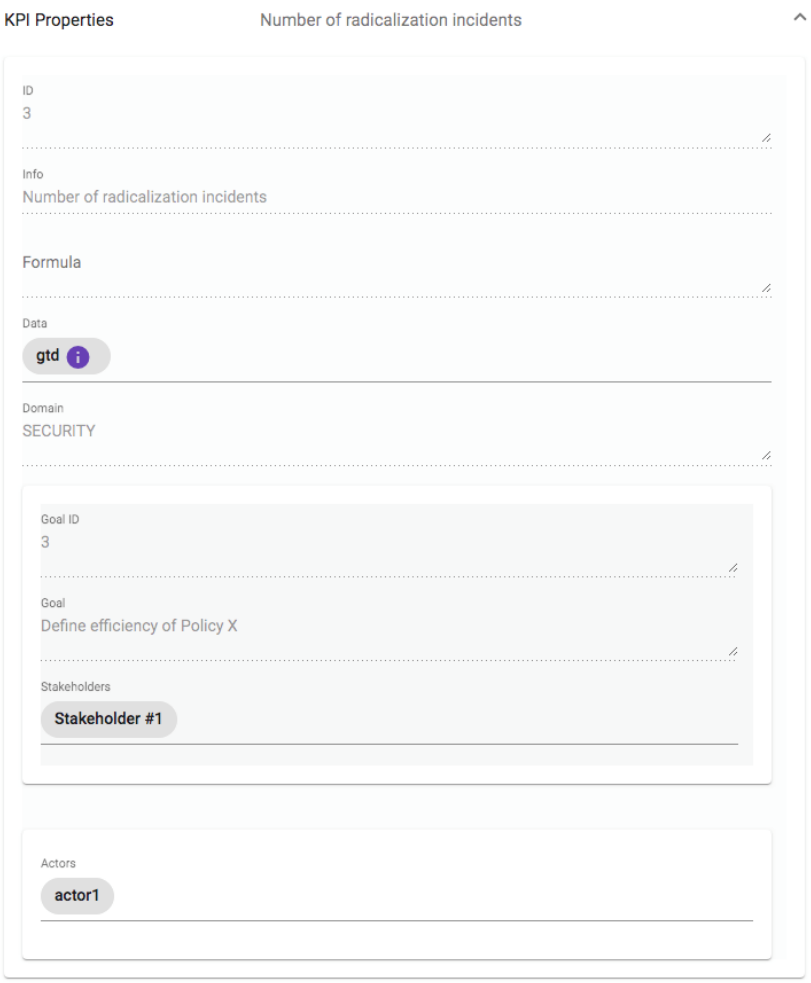

FIGURE 22 - KPI PROPERTIES

Figure 23 shows the third step of the wizard, which is the evaluation of the selected KPI, by using the proper Analytics Tool.

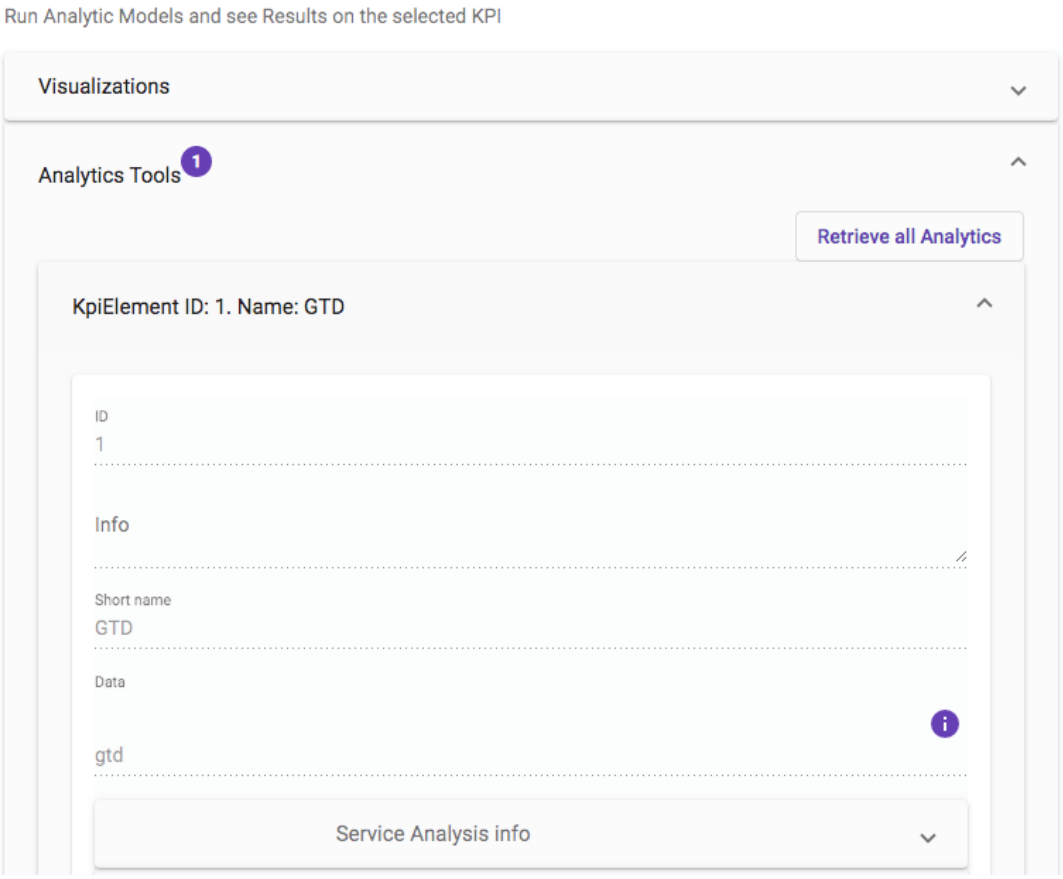

FIGURE 23 - EVALUATION - ANALYTICAL TOOL SELECTION

Figure 24 shows technical details of the Analytics Tool in an expandable card, as to not distract the PM.

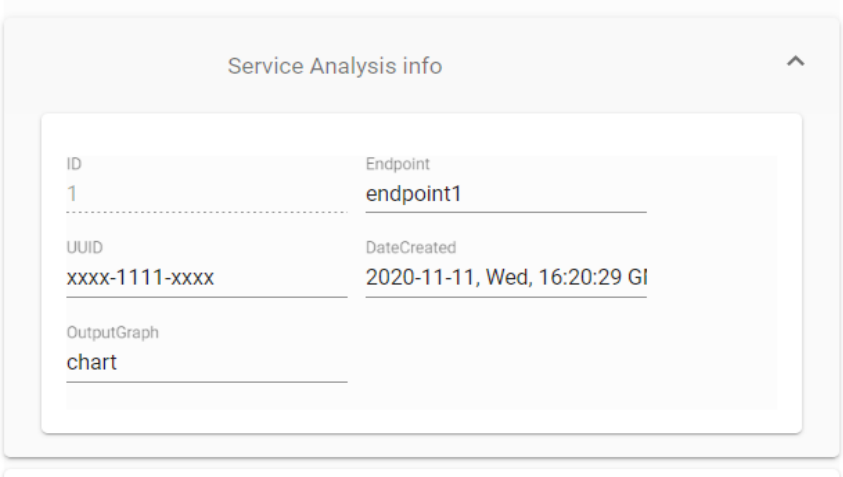

FIGURE 24 - ANALYTICAL TOOL TECHNICAL DETAILS

Figure 25 shows the parameter list that the selected Analytics Tool expects for invocation. A parameter's values overview is also presented, which includes the default values for the calculation of the KPI.

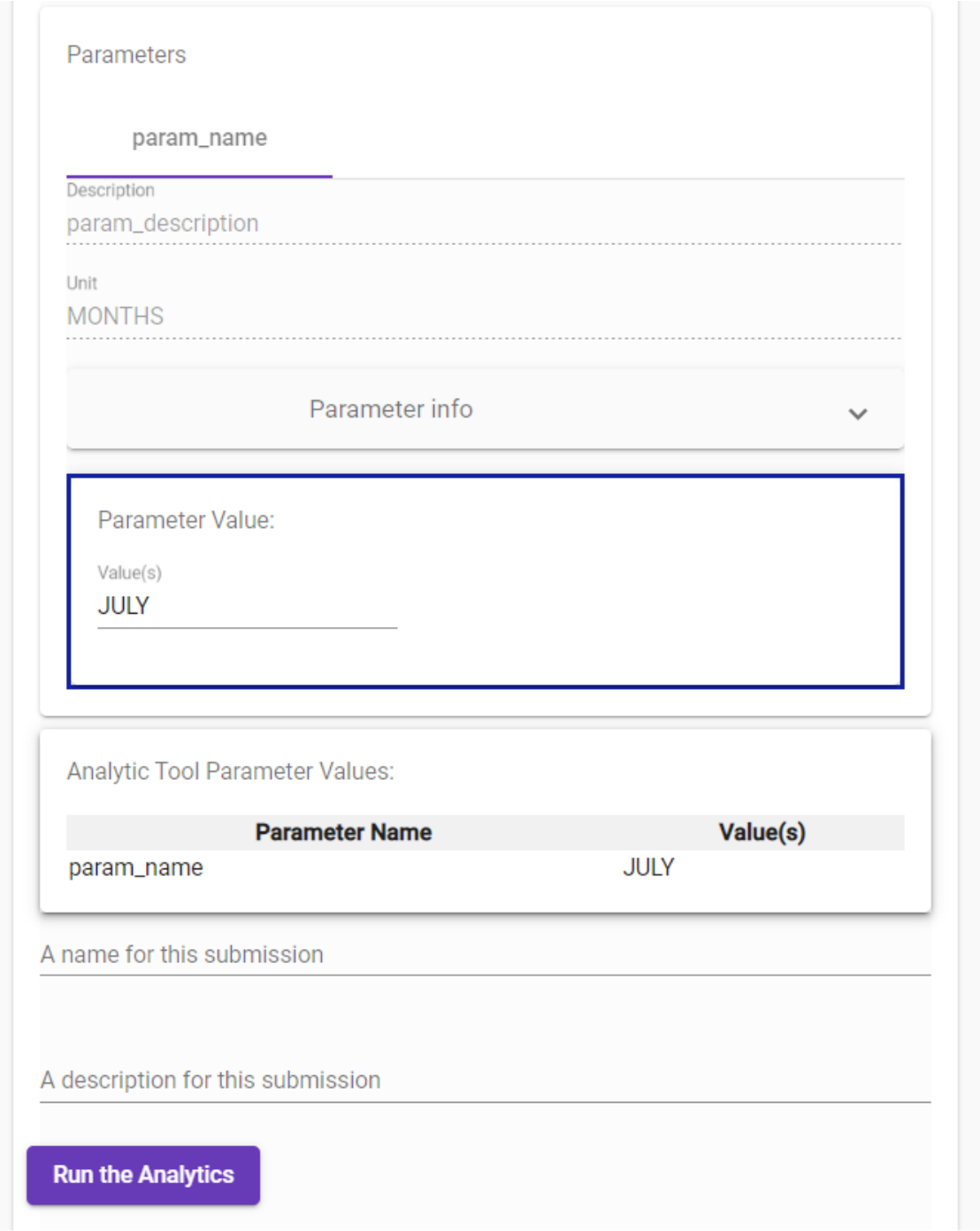

FIGURE 25 - ANALYTICAL TOOL PARAMETERS AND SUBMISSION

Figure 26 shows the details for each parameter of the Analytical tool, also in the form of an expandable card.

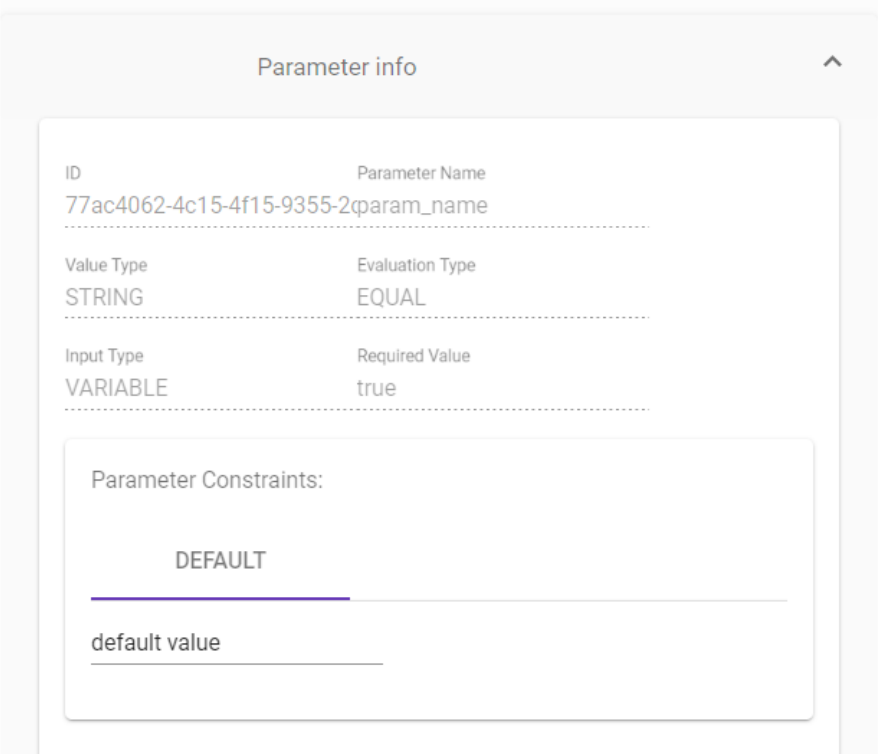

FIGURE 26 - ANALYTICAL TOOL PARAMETERS DETAILS

Figure 27 shows the fourth step of the wizard, presenting a short overview of the selected policy.

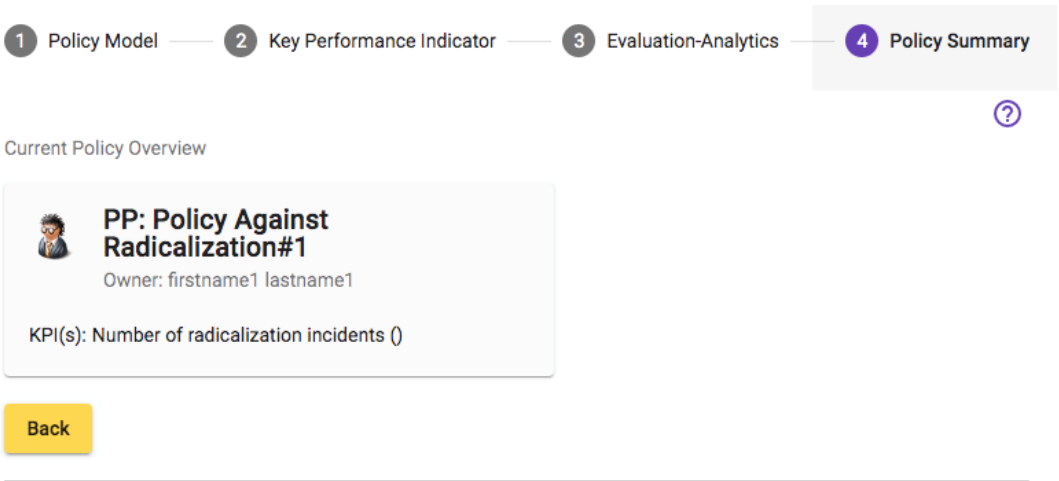

FIGURE 27 - POLICY OVERVIEW

Through the Analytics of the policy, the User can see the status of the submitted analytics job, as well as a list of past submissions. Figure 28 shows a list of Analytics results. For each analytics job, meta information is shown: submission date, the ID of the job, the given description by the user, and the current job status. Also, a popup with the executed parameter values is shown to help the User in the results selection procedure.

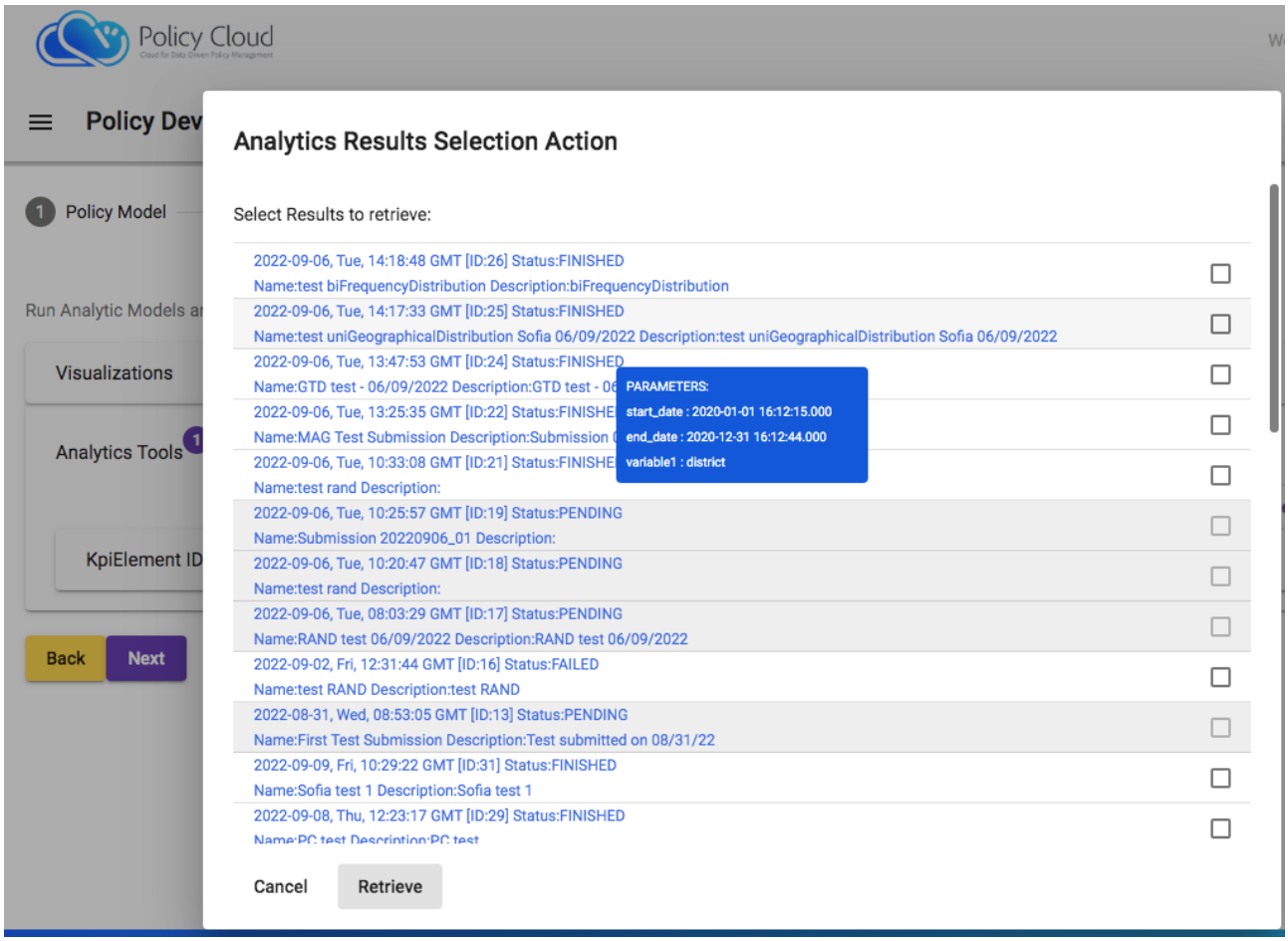

FIGURE 28 - LIST OF ANALYTICS RESULTS

Figure 29 shows the UI Menu for selecting different PDT components.

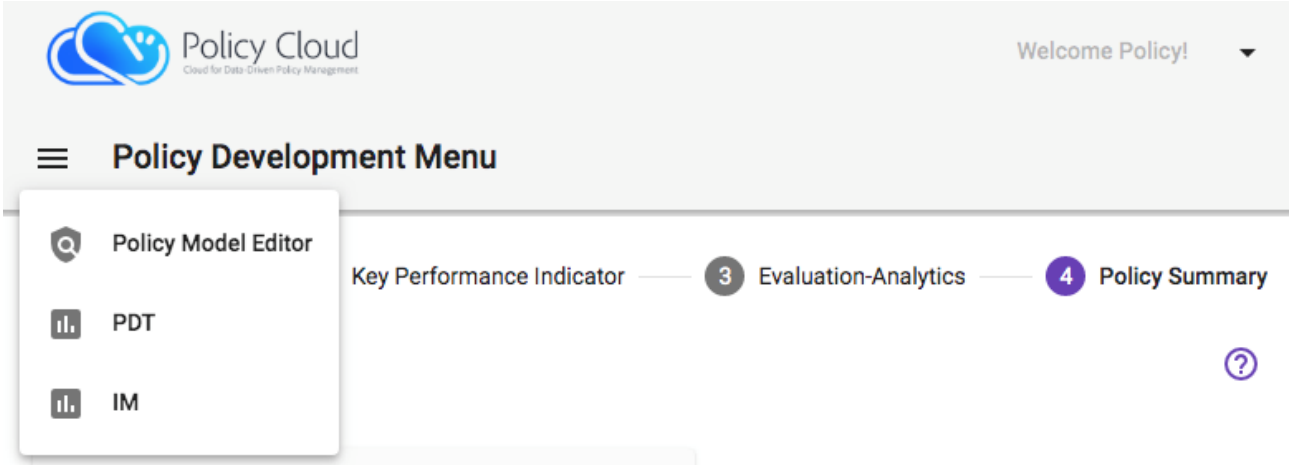

FIGURE 29 - PDT COMPONENTS MENU

For each step of the wizard help menu is available. Figure 30 shows the online help for the 1st step of the wizard.

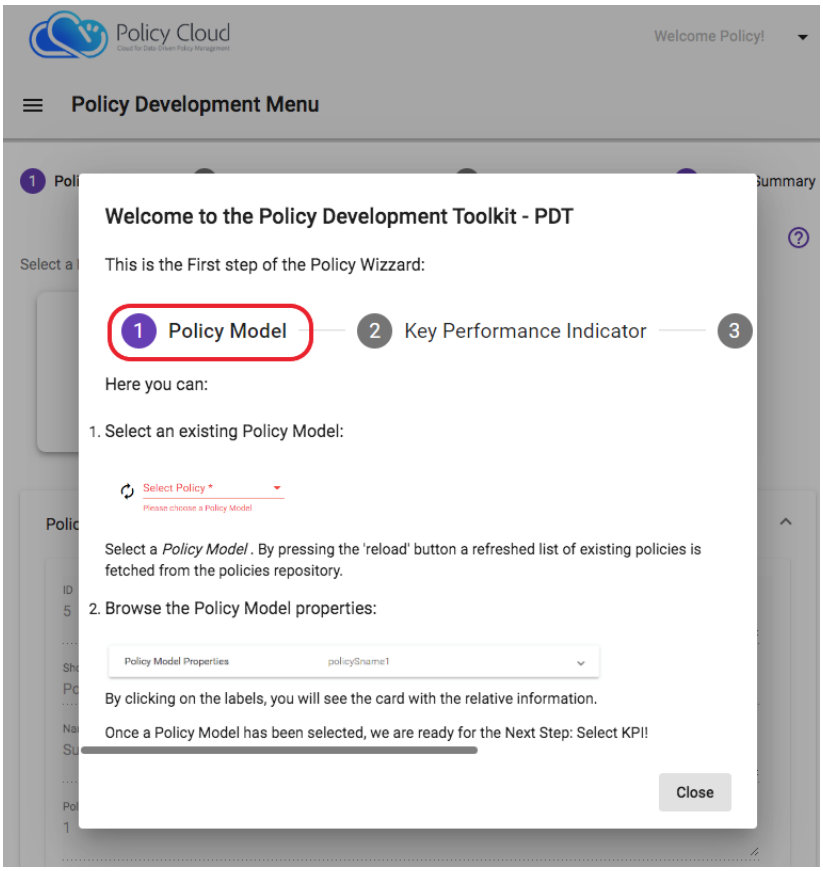

FIGURE 30 - ONLINE HELP

At the footnote of each PDT page, the link for the End Users Data Protection Information has been added, as shown in Figure 31 and Figure 32.

FIGURE 31 - USER DATA PROTECTION INFORMATION NOTICE

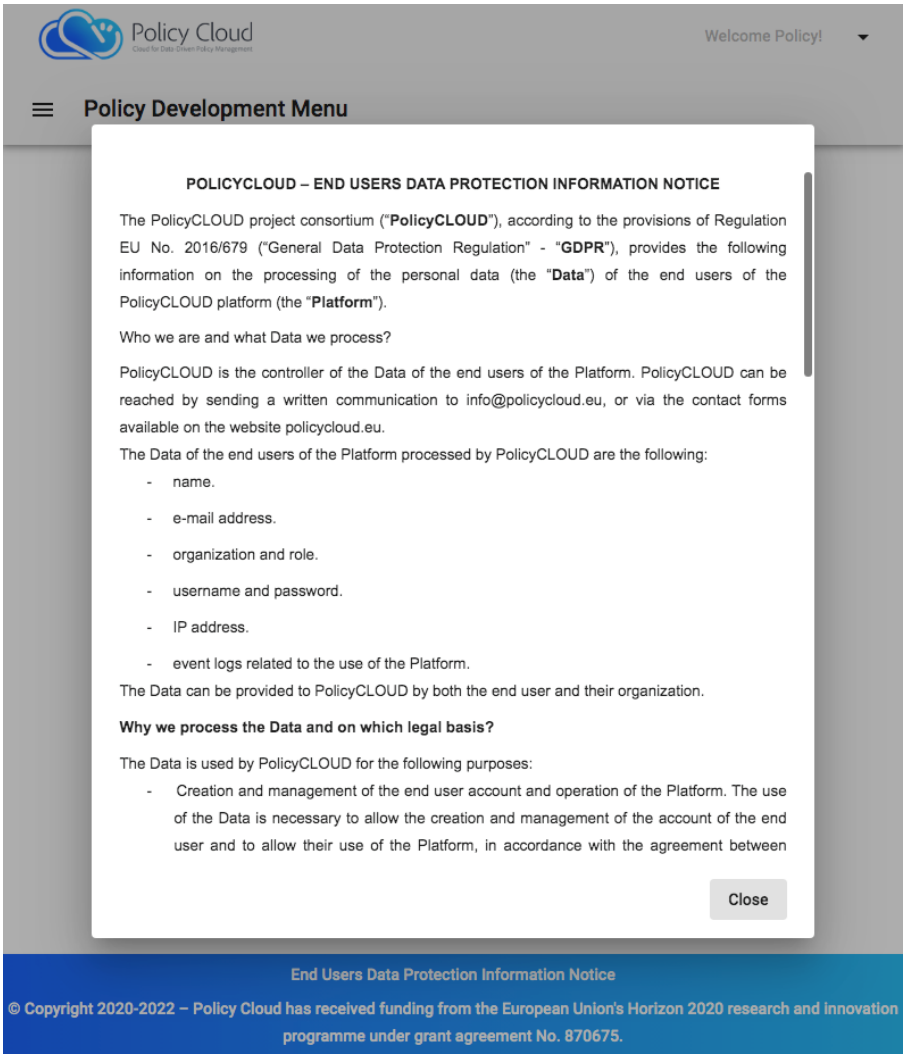

FIGURE 32 - USER DATA PROTECTION TEXT

A user-friendly guide / FAQ on how PolicyCLOUD users should address each parameter required from them for the registration (in particular, those specified in order to ensure legal/ethical compliance) is being developed as part of the implementation of the WP4 Legal/Ethical Checklist.

After the authentication of the User by providing his/her credentials to the Authentication server, the User Profile is available to the PDT page. Figure 33 shows the User Profile, as well as the Logout action button.

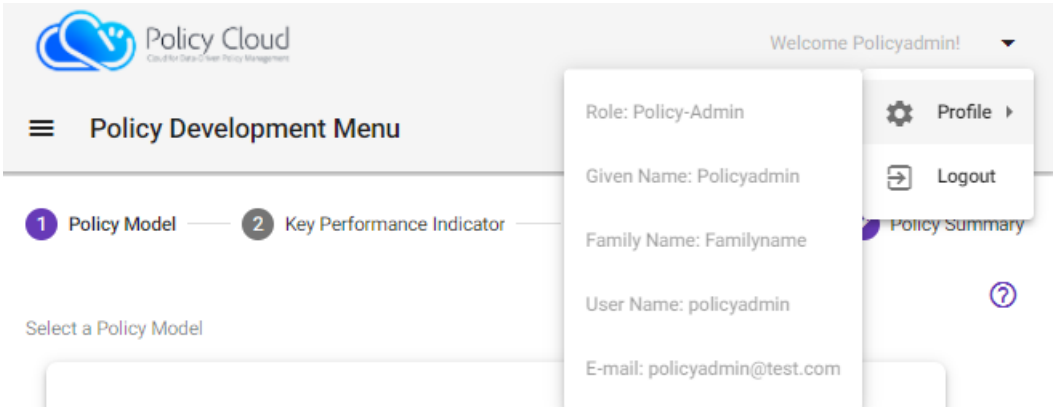

FIGURE 33 - USER PROFILE DETAILS/LOGOUT

The communication with the PDT Backend is conducted through the HTTPS secure protocol. The User's browser info regarding the PDT Certificate is shown in Figure 34.

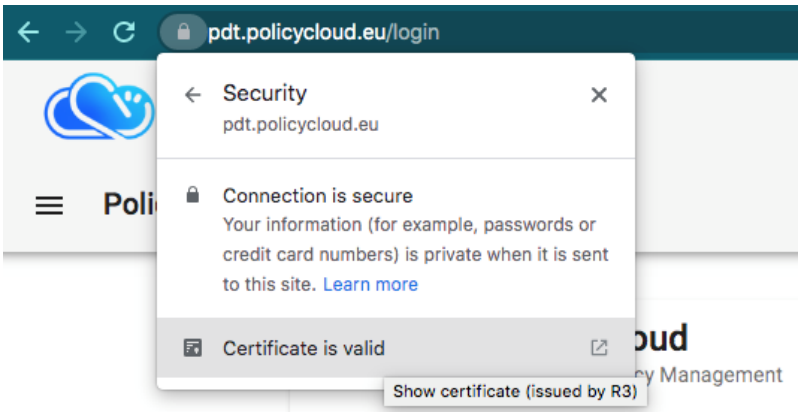

FIGURE 34 - HTTS CERTIFICATE

2.2.3 Data Visualization

All the templates that the visualization component provides can be consulted through a specific demo page in order to provide an online demo to the final users. This demo is available here: *https://pdt.policycloud.eu/viz_demo*

The data used to stablish the communication between the PDT and the Data Visualization must be in a JSON format. The specific format of these data for the different types of charts has been agreed between the WP4 and the WP5 teams throughout the duration of the project. The format and the content of these JSONs can be checked and download for all the charts using the "View Json" button that appears at the bottom of each chart.

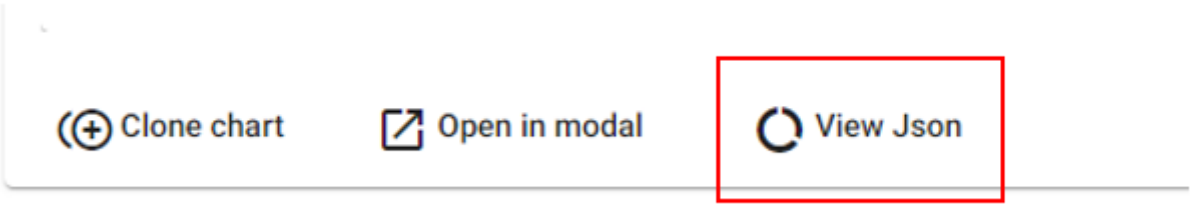

FIGURE 35 - VIEW THE DATA PLOTTED IN THE CHART IN JSON FORMAT

The following image depicts one of the JSONs used to plot a chart just as an example. The rest of the JSONs can be checked online using the visualization demo page


```
{
  "data": [
    {"events": [{"category": "2022-06-04", "winemag": 8.52}],"name_ontology": "Coto de Hayas Garnacha Centenaria","winery": "Bodegas Aletta"},
    {"events": [{"category": "2022-06-04","winemag": 11.36},{"category": "2022-06-10","winemag": 10.41}],"name_ontology": "Coto de Hayas Garnacha Centenaria","winery": "
    {"events": [{"category": "2022-06-04","winemag": 16.14,"carrefour": 10.34},{"category": "2022-06-03","winemag": 19.87,"carrefour": 13.44},{"category": "2022-06-10","
    {"events": [{"category": "2022-06-03","winemag": 13.25}],"name_ontology": "Coto de Hayas Garnacha Centenaria", "winery": "La Vida"},
    {"events": [{"category": "2022-06-04","winemag": 12.3},{"category": "2022-06-10","winemag": 14.19},{"category": "2022-06-09","winemag": 10.41}], "name_ontology": "Co
    {"events": [{"category": "2022-06-03","winemag": 11.36},{"category": "2022-06-10","winemag": 23.66}], "name_ontology": "Coto de Hayas Garnacha Centenaria", "winery":
    {"events": [{"category": "2022-06-03","winemag": 11.36}],"name_ontology": "Coto de Hayas Garnacha Centenaria","winery": "Penelope Sanchez"},
    {"events": [{"category": "2022-06-03","winemag": 23.66}],"name_ontology": "Coto de Hayas Garnacha Centenaria","winery": "Quo"},
    {"events": [{"category": "2022-06-10","winemag": 14.19}],"name_ontology": "Coto de Hayas Garnacha Centenaria","winery": "Santo Cristo"},
    {"events": [{"category": "2022-06-10","winemag": 15.14}],"name_ontology": "Coto de Hayas Garnacha Centenaria","winery": "Vinos del Viento"},
    {"events": [{"category": "2022-05-22","winemag": 14.19}],"name_ontology": "Viñas de Antillon Blanco","winery": "Viñas del Vero"},
    {"events": [{"category": "2022-06-03","winemag": 7.57}],"name_ontology": "Coto de Hayas Garnacha Centenaria","winery": "Zaza"}
  ],
  "description": "temporal evolution of price per name_ontology",
  "endDate": "2022-06-14 09:53:53.000",
  "name": "Temporal Evolution of price by winery_name_ontology",
  "startDate": "2022-03-24 09:53:39.000",
  "x-axes": "timestamp",
  "y-axes": "price"
}
```


FIGURE 36 - EXAMPLE OF THE JSON USED TO PLOT THE HEATMAP TABLE FOR THE POLITIKA TOOL

2.2.1 PDT Backend

As described in the previous subsection, the backend of the PDT exposes its interfaces via a 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 [5] open source documentation UI. Swagger allows the developer to annotate her source code and provides tools and plugins that auto-generate the documentation at 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.

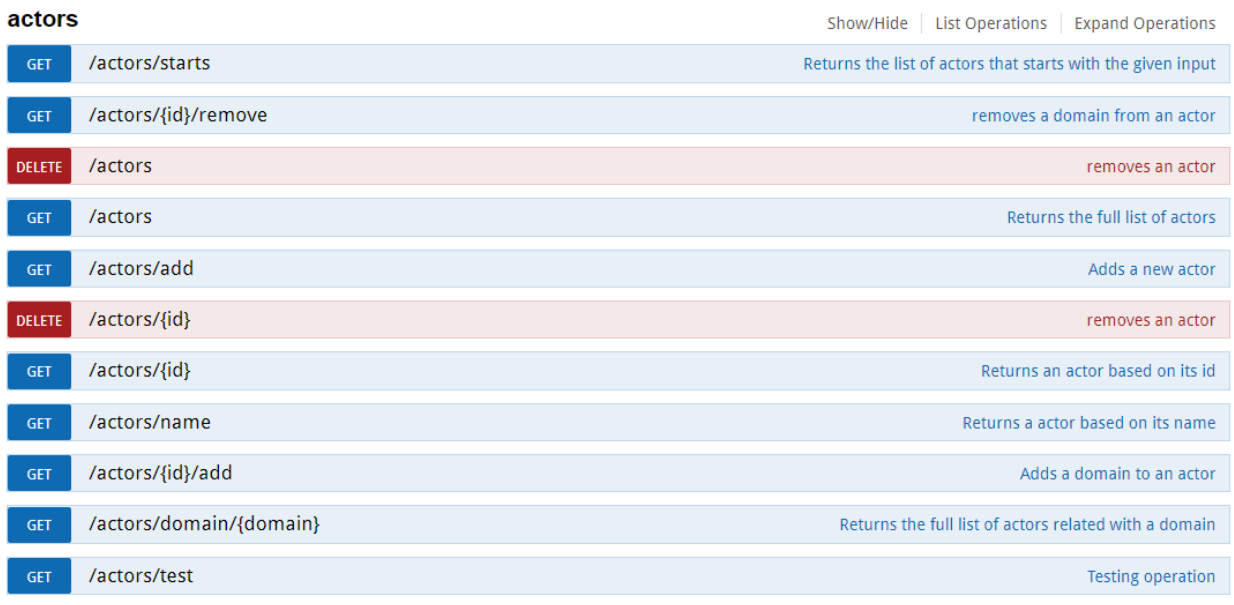

FIGURE 37 - WEB METHODS RELATED WITH ACTORS

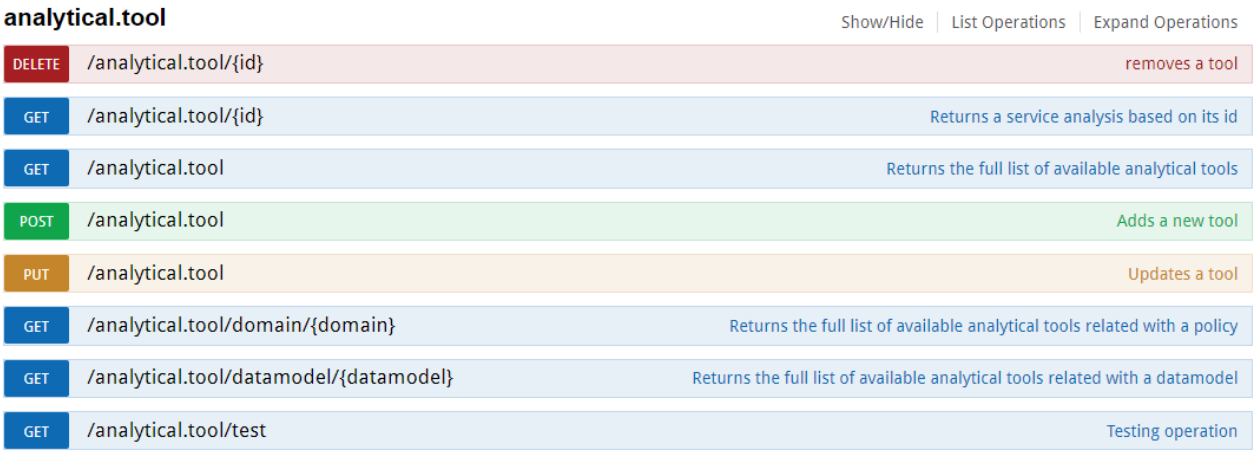

FIGURE 38 - WEB METHODS RELATED WITH ANALYTICAL TOOLS

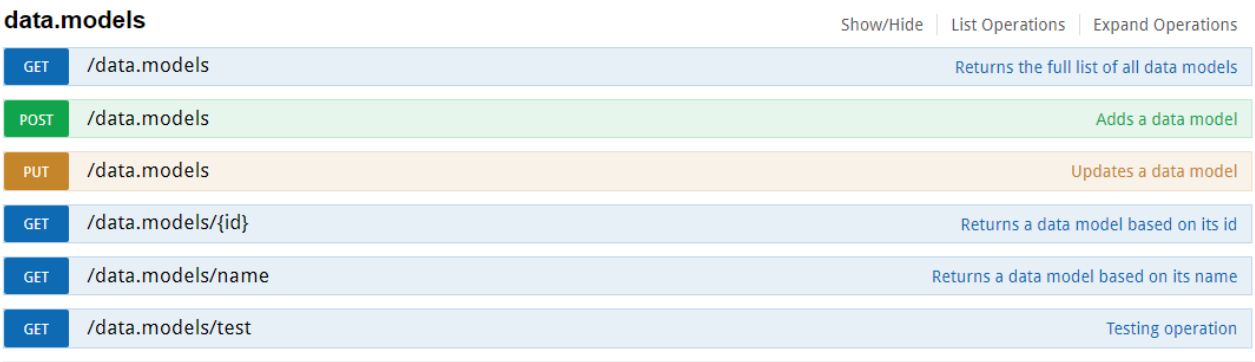

FIGURE 39 - WEB METHODS RELATED WITH DATA MODELS

FIGURE 40 - WEB METHODS RELATED WITH DOMAINS

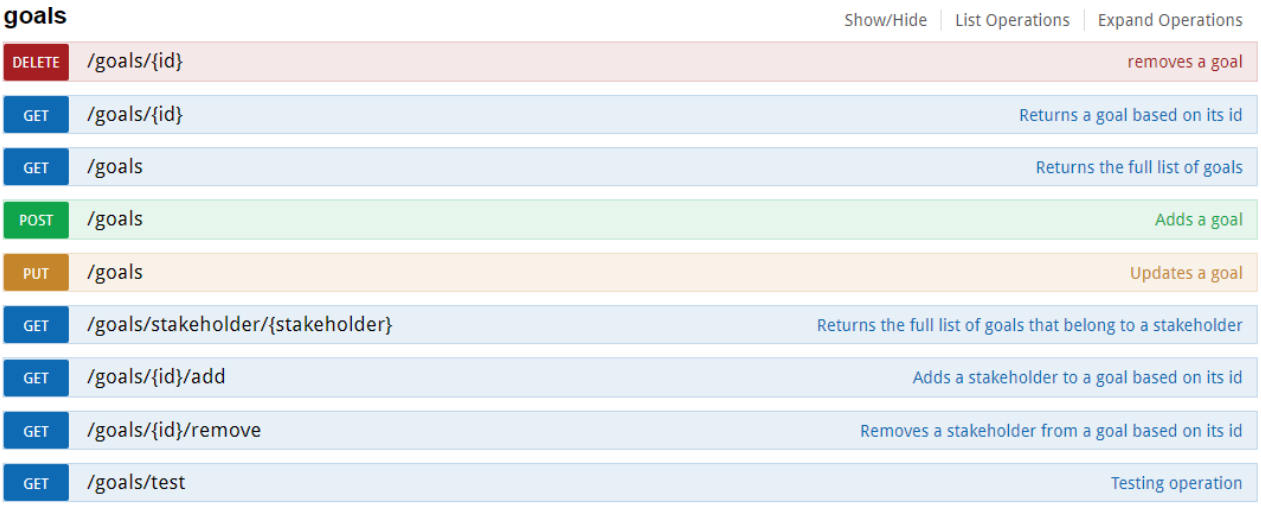

FIGURE 41 - WEB METHODS RELATED WITH GOALS

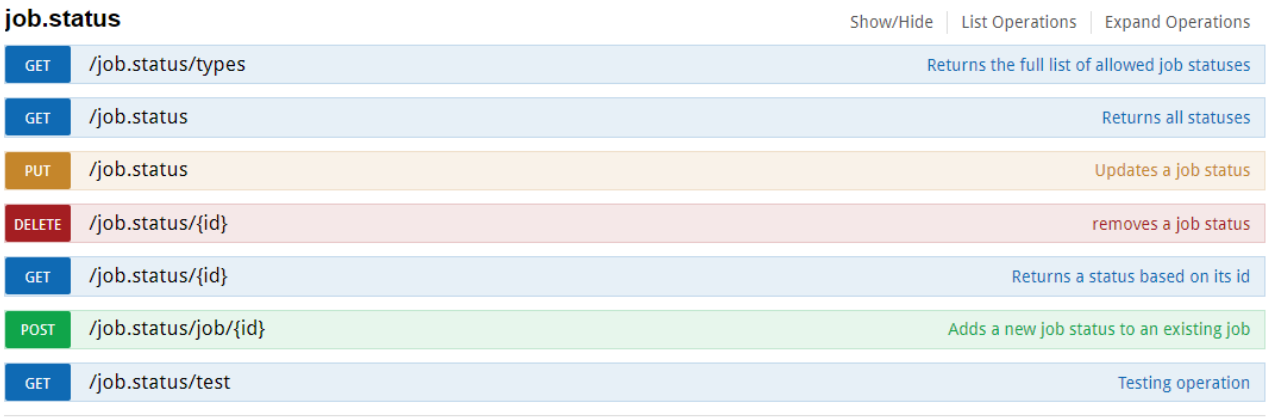

FIGURE 42 - WEB METHODS RELATED WITH JOB STATUS

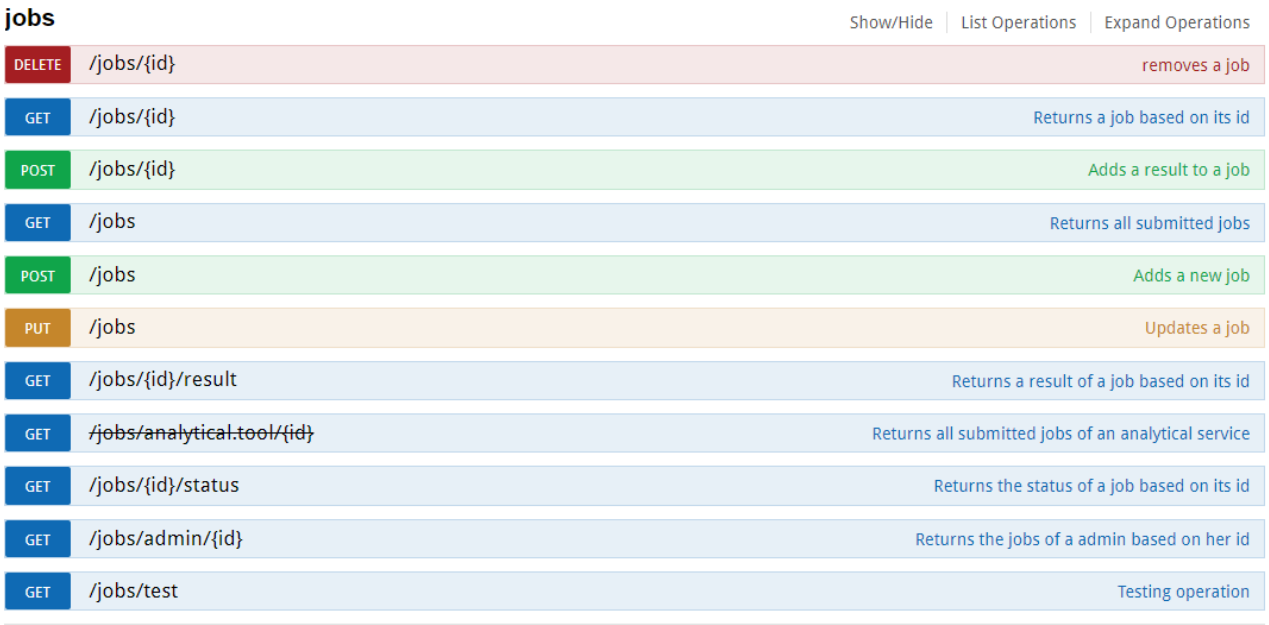

FIGURE 43 - WEB METHODS RELATED WITH JOBS

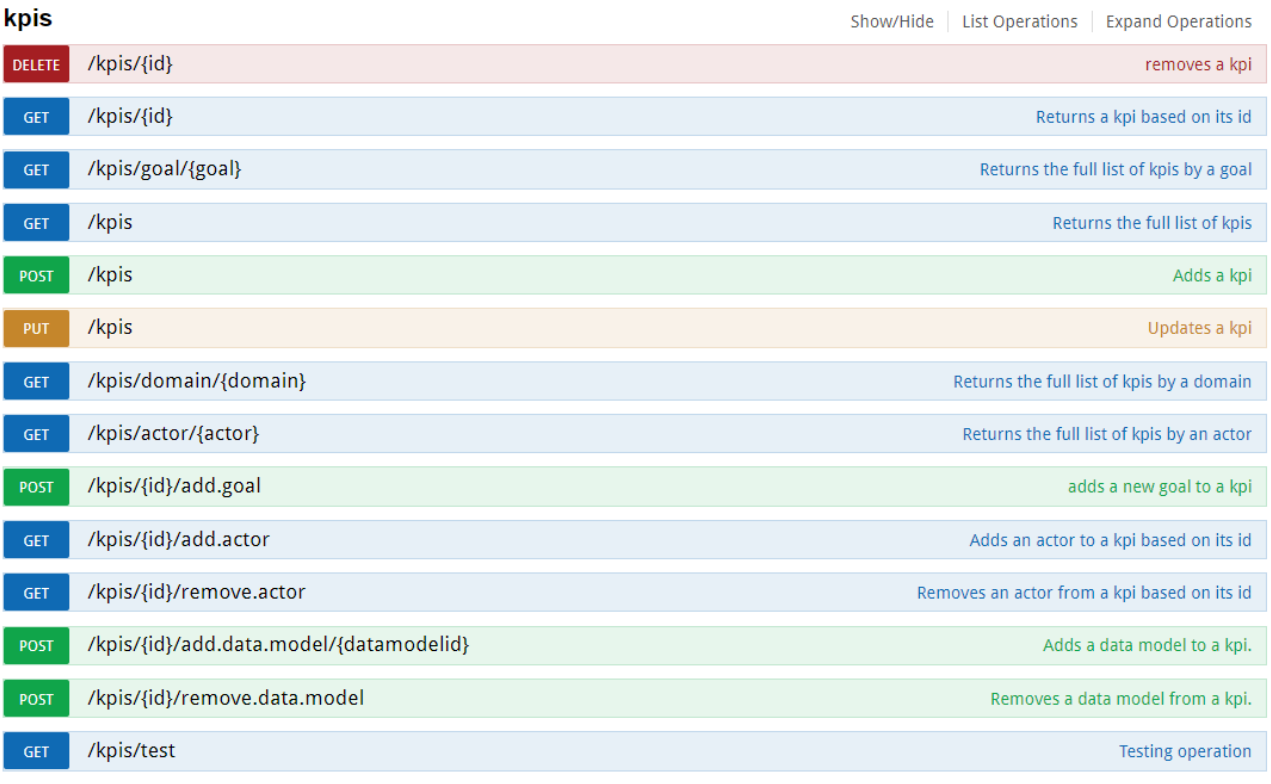

FIGURE 44 - WEB METHODS RELATED WITH KPIS

FIGURE 45 - 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 should be aware of, so that the PDT can modify the UI in order to reflect 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 be triggered in order to send a message via this web socket to the PDT. By doing this, the PDT can be in position to update the status of a pending job, without periodically ask this status from the backend. The DTOs that are exchanged have the following format:

```
{
 "id": 3242,
 "notification": "UPDATED",
 "job": {
  "id": 3242,
  "name": "job name",
  "description": "the description of the job",
  "status": "FINISHED",
  "dateCreated": "2021-10-01T09:23:43"
   .
   .
   .
   .
 }
}
```

In this DTO, the *id* is the unique identifier of the *job* its status has been changed and the *notification* can accept one of the following values: ADDED, UPDATED or DELETED. Finally, the *job* contains all information about the corresponding *job*, like its *id, name, description, status,* etc.

2.3 Baseline technologies and tools

2.3.1 Policy Modelling Editor

As described in the deliverable D5.6 "Cross-sector Policy Lifecycle Management: Design and Open Specification 3" [1], the PME is developed using the open-source Angular framework [6]. For UI components and controls we use the Angular Material library [7].

2.3.2 Policy Development Toolkit

As described in the deliverable D5.6 "Cross-sector Policy Lifecycle Management: Design and Open Specification 3" Section 4.2, PDT is built as a Single Page Application developed using the open-source Angular framework [4][6]. For UI components and controls we use the Angular Material library [7]

The PDT is using Docker to encapsulate the web serving functionality along with NGINX [8] web server. The web server enforces the HTTPS protocol for secure communication using its proxy configuration. The server certificate has been issued by the non-profit Certificate Authority Let's Encrypt [9]. The procedure for the renewal of the certification has also been included in the dockerised deployment process.

2.3.3 Data Visualization

As the Data Visualization is part of the PDT, it has been developed using the same base technology, AngularJS v10 [6]. The main JavaScript libraries used for this first prototype are:

- Leaflet [10] is used to plot the heatmap
- Amcharts [11][11] is used to plot the line and the gauge charts

Although the component is integrated into the PDT, a standalone version can also be used, and the detailed invocation instructions are detailed into the readme file that accompanies the source code of the component. Still, the service can be started either with a NodeJS server or using a Docker container (see Section 3.1.3).

To visualize it as part of the PDT, this component has to be started and all previous steps must be taken.

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 [12]. 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 prototype of the Policy Modelling Editor component is available at:

- Project GitLab repository:https://registry.grid.ece.ntua.gr/chris/policy-model-editor
- Project GitLab repository: https://registry.grid.ece.ntua.gr/kostas/pdt

3.1.2 Policy Development Toolkit

The source code of the 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 Data Visualization component is available for registered users inside the PDT repository at:

- Project GitLab repository: https://registry.grid.ece.ntua.gr/kostas/pdt

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: https://registry.grid.ece.ntua.gr/pavlos_LXS/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.

As already mentioned, the PDT Backend requires the use of a relational data store to persistently store its policies, policy related meta-information and results of the analytical tools. We relied on the datastore provided by LXS, which can be found at:

- Project GitLab repository: https://registry.grid.ece.ntua.gr/pavlos_LXS/policy-store

As the datastore is published with LXS IPR licence, its source code cannot be available. However, the GitLab repositories for the datastore contains all binaries and scripting files necessary for packaging the distribution and creating a docker container that the datastore can start within.

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. Will be deployed in conjunction with the PDT.

3.2.4 PDT Backend

In order to deploy and use this component, the administrator has first 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 locally build a docker image. 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 administrators can now open their favorite browser and put the following URL, which will open the swagger documentation:

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

An alternative option for starting the PDT backend is to manually start firstly the LXS relational datastore, and then the PDT backend. With the assumption that the docker images for both the relational datastore and the PDT backend has been already created and available, then the user needs to first start the datastore by executing the following:

```
docker run -d -p 2181:2181 -p 1529:1529 --name datastore --env KVPEXTERNALIP='datastore!9800' policy-store
```

Once the datastore is running, then the user can start the PDT backend with:

```
docker run -d -p 54735:54735 --name pdt_backend --env DATASTORE_HOST='172.17.0.2' --env swagger_path=/tmp/pdt-backend
```

Once the backend has been started, then the administration can still open a browser and check that everything is working by invoking the swagger documentation, as shown before at:

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

4 Conclusions

This deliverable provides a description of the software components of the Policy Modelling Editor Layer and Policy Development Toolkit Layer, which make up the frontend part of the PolicyCLOUD platform and allow policy makers to create, update and evaluate policies. For each component, a description of its APIs, specification of the main functionalities, and description of the source code is provided.

The most important modifications from the last version of the deliverable concern the Policy Development Toolkit Layer and the Data Visualization Layer. The changes in question have been performed mainly to improve the usability of the platform for the end user, going in the direction indicated by the co-creation workshops that the Use Cases periodically organize.

References

[1] PolicyCLOUD. D5.6 “Cross-sector Policy Lifecycle Management Design and Open Specification 3”. Samuele Baroni et al 2022.

[2] PolicyCLOUD. D5.5 “Cross-sector Policy Lifecycle Management Software Prototype 2”. Armend Duzha et al 2021.

[3] PolicyCLOUD. D2.7 “Conceptual Model and Reference Architecture 3”, Panayiotis Michael et al 2022.

[4] PolicyCLOUD, D4.5 “Reusable Model and Analytical Tools, Design and Open Specification 3”, Yosef Moatt et al, 2022.

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

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

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

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

[9] Let’s Encrypt Certificate Authority, at https://letsencrypt.org/

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

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

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