



CAMPUS  
DE EXCELENCIA  
INTERNACIONAL



# **Máster Universitario en Ingeniería Informática**

Universidad Politécnica de Madrid

Escuela Técnica Superior de  
Ingenieros Informáticos

## **TRABAJO FIN DE MÁSTER**

Visualización y monitorización de datos por los  
usuarios finales de un marketplace digital mediante  
mashup

Autor: Alejandro Rodríguez Fraga

Director: Miguel Jiménez Gañán

MADRID, JUNIO 2018



## Resumen

Con la cada vez mayor cantidad de datos disponibles para los usuarios y las limitaciones de los catálogos actuales de datos para poder mostrarlos, compararlos y elegir de forma ágil y eficaz, comienza a ser imprescindible disponer de herramientas adecuadas para generar visualizaciones para dichos datos de forma que los usuarios puedan fácilmente extraer la información contenida en los mismos. Dentro del proyecto europeo FI-NEXT, se ofrecen herramientas para la gestión de los datos (usando CKAN), así como para la creación de un gran mercado de datos. Sin embargo, no se dispone de una herramienta que permita a los usuarios visualizar adecuadamente dichos datos.

Este trabajo propone como solución el uso de la herramienta de mashup WireCloud, mediante la cual se pueden crear dashboards genéricos y adaptables a cualquier tipo de dato, proporcionando así una solución universal para la representación de los datos disponibles. Para poder usar WireCloud de esta forma, es necesario primero desarrollar un extenso catálogo de componentes de WireCloud que permita a sus usuarios generar estos dashboards, permitiendo mediante la combinación de múltiples de estos componentes lograr obtener los datos necesarios para generar las gráficas y visualizaciones que el usuario final necesita para consumir la información contenida en los datos. Este catálogo de componentes se ha desarrollado partiendo de ciertos componentes ya existentes, que se han actualizado y mejorado para adaptarlos a las necesidades de este catálogo, así como creado nuevos componentes para lograr un catálogo completo y coherente que permita la creación de dashboards de visualización para cualquier tipo de datos.

Debido a la gran cantidad de componentes necesarios para proporcionar una solución genérica y debido a que el usuario que quiere ver los datos puede no tener los conocimientos necesarios para crear el dashboard que necesita para ello, se propone y se ofrece además una herramienta que permita automatizar y ayudar al usuario en la creación de este tipo de dashboards. Esta herramienta guiará al usuario en la creación del dashboard mediante recomendaciones sobre qué tipos de gráficas usar y qué variables se pueden o deben usar con ellas. Sin embargo, ya que los datos en CKAN carecen de semántica que permita a la herramienta saber cuales son importantes e interesantes, esta recomendación está limitada. Gracias a los FIWARE DataModels, que normalizan los datos y ofrecen este valor semántico, esta limitación se puede evitar y así generar recomendaciones más efectivas. Una vez configurado el dashboard a crear usando la herramienta, esta se encargaría de crearlo, añadir los componentes y configurarlo, dejando libre de esta carga de trabajo al usuario.

Se propone también integrar CKAN con WireCloud, mediante una extensión de CKAN que permita usar WireCloud para mostrar visualizaciones de los datos,

así como crear nuevas visualizaciones directamente desde CKAN mediante la herramienta de creación de dashboards. Esta extensión que permite la visualización de los datos ya existe, siendo por tanto necesario ampliarla para poder crear dashboards de visualización completos.

## Abstract

With the increasing amount of data available to users and the limitations of current data catalogs for displaying, comparing and choosing data in an agile and efficient manner, it is becoming essential to have adequate tools to generate visualizations for such data so that users can easily extract the information contained therein. Within the FI-NEXT European project, tools are offered for data management (using CKAN), as well as for the creation of a large data marketplace. However, there is no tool available to allow users to properly visualize such data.

This work proposes as a solution the usage of the mashup tool WireCloud, through which generic dashboards can be created. These dashboards are adaptable to any kind of data, thus providing an universal solution for the representation of the available data. In order to use WireCloud in this way, it is necessary to first develop an extensive catalog of WireCloud components that allows its users to generate these visualization dashboards, allowing them to combine multiple of these components in order to obtain the data needed to generate the graphs and visualizations that the end user needs for him to be able consume efficiently the information contained in the data. This catalog of components has been developed from certain existing components, which have been updated and improved to adapt them to the needs of this catalog, as well as creating new components to achieve a complete and consistent catalog that allows the creation of visualization dashboards for any type of data.

Due to the large number of components needed in order to provide a generic solution, and because the user who wants to see the data visualization may lack the necessary knowledge and skills required to create the dashboard they need, a tool is proposed and offered to automate and assist the user in the creation of this kind of dashboards. This tool will guide the user while creating and configuring the dashboard by providing recommendations on what types of graphs to use and which variables to use with each of them. However, since the data in CKAN lacks the semantics to allow the tool to know which ones are important and interesting, this recommendation is limited. Thanks to the FIWARE DataModels, which standardize the data and offer this semantic value, this limitation can be avoided and thus generate more effective recommendations. Once the dashboard to be created is configured using the tool, it will be in charge of creating it, adding the components and configuring it, leaving the user free of this workload.

It is also proposed to integrate CKAN with WireCloud, through an extension of CKAN that allows to use WireCloud to show visualizations of the data, as well as to create new visualizations directly from CKAN through the dashboard creation tool. This extension that allows the visualization of the data already exists, so it is only needed to extend it in order to support the creation of complete visualization

dashboards.

# Index

<b>1</b>	<b>Introduction and objectives</b>	<b>1</b>
1.1	Project Context . . . . .	2
1.1.1	CoNWeT lab . . . . .	2
1.1.2	FIWARE . . . . .	2
1.1.3	Internet of things . . . . .	3
1.2	Motivation . . . . .	4
1.3	Objectives . . . . .	5
<b>2</b>	<b>State of the art</b>	<b>7</b>
2.1	Smart cities . . . . .	7
2.2	CKAN . . . . .	9
2.3	NGSI . . . . .	10
2.4	Orion Context Broker . . . . .	11
2.5	FIWARE DataModels . . . . .	11
2.5.1	Alerts . . . . .	12
2.5.2	Parks and gardens . . . . .	12
2.5.3	Environment . . . . .	13
2.5.4	Points of interest . . . . .	13
2.5.5	Civic issue tracking . . . . .	14
2.5.6	Street lighting . . . . .	14
2.5.7	Device . . . . .	14
2.5.8	Transportation . . . . .	15
2.5.9	Indicators . . . . .	15
2.5.10	Waste management . . . . .	15
2.5.11	Parking . . . . .	16
2.5.12	WeatherObserved . . . . .	16
2.6	WireCloud . . . . .	16
2.7	Data visualization tools . . . . .	18
2.7.1	Tableau . . . . .	19
2.7.2	Qlikview . . . . .	19

2.7.3	Power BI . . . . .	21
2.7.4	Knowage . . . . .	21
2.7.5	Datadog . . . . .	22
2.8	Technologies used . . . . .	22
2.8.1	Python . . . . .	23
2.8.2	JavaScript . . . . .	23
2.8.3	HTML and CSS . . . . .	23
2.8.4	Highcharts . . . . .	24
2.8.5	GoogleCharts . . . . .	24
2.8.6	Google maps . . . . .	24
2.8.7	OpenLayers map . . . . .	25
2.9	Leaflet map . . . . .	25
<b>3</b>	<b>Chart types</b>	<b>27</b>
3.1	Column chart . . . . .	27
3.1.1	Stacked column chart . . . . .	29
3.1.2	Column histogram . . . . .	30
3.2	Bar chart . . . . .	30
3.3	Line chart . . . . .	32
3.4	Area chart . . . . .	32
3.5	Pie chart . . . . .	33
3.6	Scatter chart . . . . .	33
3.6.1	Bubble chart . . . . .	35
3.7	Map chart . . . . .	35
3.7.1	Heatmap . . . . .	37
3.8	Gauge Chart . . . . .	37
<b>4</b>	<b>WireCloud visualization catalogue</b>	<b>39</b>
4.1	Data harvester components . . . . .	40
4.1.1	CKAN source operator . . . . .	40
4.1.2	NGSI source operator . . . . .	41
4.1.3	GitHub harvester operator . . . . .	42
4.1.4	GitLab harvester operator . . . . .	43
4.1.5	Jira harvester operator . . . . .	43
4.1.6	Jenkins harvester operator . . . . .	44
4.2	Data transformation and preparation components . . . . .	44
4.2.1	NGSI entity to PoI operator . . . . .	45
4.2.2	NGSI datamodel to PoI operator . . . . .	45
4.2.3	NGSI datamodel to heatmap operator . . . . .	46
4.2.4	Value list filter operator . . . . .	46



---

4.2.5	Calculate tendency operator . . . . .	47
4.2.6	Count list operator . . . . .	47
4.2.7	Basic list arithmetic operator . . . . .	48
4.2.8	Union list and intersect list operators . . . . .	48
4.2.9	Packlist operator . . . . .	49
4.2.10	Labels to dataserie operator . . . . .	49
4.2.11	Chart generator operators . . . . .	49
4.2.12	Data filter components . . . . .	50
4.3	Data representation components . . . . .	53
4.3.1	Highcharts widget . . . . .	53
4.3.2	Panel widget . . . . .	53
4.3.3	Data table viewer . . . . .	54
4.3.4	Leaflet map viewer . . . . .	54
4.3.5	OpenLayers 3 map viewer . . . . .	55
4.3.6	History player widget . . . . .	55
<b>5</b>	<b>WireCloud integration with CKAN</b>	<b>57</b>
5.1	Reference dashboards . . . . .	59
5.2	WireCloud CKAN extension . . . . .	61
5.3	CKAN dashboard creator . . . . .	62
5.4	WireCloud REST API for dashboard management . . . . .	66
<b>6</b>	<b>WireCloud integration with APIInf</b>	<b>71</b>
<b>7</b>	<b>Conclusions</b>	<b>73</b>
<b>8</b>	<b>Future work</b>	<b>75</b>
	Bibliografía . . . . .	78



# List of Figures

1.1	Internet of things . . . . .	4
2.1	What is MiNT . . . . .	8
2.2	WireCloud dashboard view . . . . .	18
2.3	WireCloud wiring view . . . . .	19
2.4	Tableau example view . . . . .	20
2.5	Qlikview example view . . . . .	20
2.6	Power BI example view . . . . .	21
3.1	Chart selection diagram . . . . .	28
3.2	Column chart displaying populations each year . . . . .	29
3.3	Stacked column displaying products by profit . . . . .	30
3.4	Example column histogram . . . . .	31
3.5	Bar chart displaying primary sector in 2010 . . . . .	31
3.6	Line chart displaying the evolution of multiple populations . . . . .	32
3.7	Area chart displaying the evolution of multiple populations . . . . .	33
3.8	Pie chart displaying pet percentages . . . . .	34
3.9	Example scatter chart . . . . .	34
3.10	Example bubble chart . . . . .	35
3.11	Map chart displaying US population by state . . . . .	36
3.12	Earthquake southeast Asia and Oceania . . . . .	37
3.13	Example gauge chart showing speed . . . . .	38
4.1	Basic filter wiring structure . . . . .	51
4.2	Filter view . . . . .	52
5.1	CKAN view creation workflow . . . . .	58
5.2	AirQuality example dashboard . . . . .	59
5.3	Parking example dashboard . . . . .	60
5.4	PoI example dashboard . . . . .	60
5.5	AirQuality example dashboard wiring . . . . .	61

5.6	CKAN dashboard creator dashboard view . . . . .	65
5.7	CKAN dashboard creator wiring view . . . . .	65
5.8	CKAN integration with WireCloud . . . . .	67
5.9	Basic component wiring . . . . .	70

# List of Tables

4.1 Basic list arithmetic operator behavior . . . . . 48



# List of Listings

1	NGSI JSON example . . . . .	11
2	Create workspace POST body . . . . .	66
3	Create widget POST body example . . . . .	68
4	Create operator PATCH body . . . . .	69
5	Create connection PATCH body . . . . .	70





# Chapter 1

## Introduction and objectives

The FI-NEXT European project, developed within the framework of the European free software community FIWARE supported by the FIWARE foundation under the European Commission, offers multiple tools aimed at open data management (mainly CKAN with extensions). This tools hold data that is either static data, or real time data, being the later based on FIWARE NGSI through the Orion Context Broker platform. FIWARE also provides tools aimed at the commercialization of open data and managing the access to such data, in order to allow the creation of a data economy ecosystem.

The combination of all this tools creates a data marketplace where anyone can become a data provider (either using a free to use model or using a paid model), while also being able to use the data other users provide, allowing users to generate value from data by combining, processing or creating applications and services that consume the data.

In a data ecosystem like the one FI-NEXT aims to archive, it becomes a must to have strong data visualization and monitorization capabilities that help extract information and value from the data for the end users. Therefore, it is essential to provide marketplace users with tools that allow end users to consume data by providing the most useful visualization available for the current dataset.

Due to the large size and disparity of the data available to the users, specially due to the increasingly large amount of data available thanks to Internet of Things, it is required to develop powerful tools that make it easier to generate the data visualizations needed by the end user, that are capable of displaying the correct information and avoiding inducing errors on what the user understands from the data, while also removing complexity and workload from the user generating the data visualizations.

This project intends to use and develop the WireCloud mashup tool so it becomes a powerful data visualization and monitoring tool, integrating it on the FI-NEXT

data marketplace and on the CKAN data catalogue.

## 1.1 Project Context

This section will define the context of the project.

- The CoNWeT laboratory is the developer of the WireCloud mashup tool, which is the reference implementation of FIWARE Mashup Application Generic Enabler, being the UPM on charge of its development, and CoNWeT is the laboratory where this project will be developed.
- FIWARE, whose data visualization capabilities this project aims to improve, in order to consume the NGSI data provided by the Publish/Subscribe Context Broker Generic Enabler (Orion Context broker), and the static data stored at CKAN.
- Internet of things generates lots of data, whose visualization this project aims to improve.

### 1.1.1 CoNWeT lab

CoNWeT (Computer Networks and Web Technologies Laboratory) is part of the CETTICO research group from the Escuela Técnica Superior de Ingenieros Informáticos de la Universidad Politécnica de Madrid. CoNWeT focuses on the research around Internet of Things (IoT) and web technologies, leading the development of the FIWARE chapters Apps and Data Delivery, as well as the development of the mashup platform WireCloud.

The CETTICO research group has been ranked thirteenth position among the 169 official research groups recognized by the UPM.

The CoNWeT lab is participating on the FIWARE project, being the owners of both the WireCloud platform and BAE (Business API Ecosystem), while also being responsible of developing the FIWARE CKAN extensions.

### 1.1.2 FIWARE

FIWARE [10] is an European project whose mission is to create an open sustainable ecosystem, built around public, royalty-free and implementation-driven software standards aimed at creating an open source standards that will ease the development of new Smart Applications in multiple sectors.

Any smart application collects relevant information from different sources about what is happening at any given time. This is known as “context information”. To get

an intelligent behavior the current and historical context information is processed, visualized and analyzed on a large scale.

FIWARE wants to promote a set of standards that describe how to collect, manage and publish context information and additionally provides elements that allow this information to be exploited once it is collected. FIWARE also provides free open source tools that can be used in order to achieve this. This set of standards and tools are key to building a unique digital market for smart applications where apps and solutions can be ported from one client to another without major changes. This also solves, in a simple way, how to capture and normalize information from sensor networks, although they communicate using different protocols and IoT languages, being able to solve the complexity of treating the information collected by the sensors and translating them into a common language.

These standards in FIWARE are called Generic Enablers(GE), all of which support interoperability with the other Generic Enablers. All Generic Enabler specifications are public and royalty free. A FIWARE Generic Enabler Implementation (GEi) is a reference implementation of one or more GE provided by FIWARE as a fully functional example. These GEis are intended to be used as examples while developing custom implementations of the Generic Enablers, while also being fully functional and usable when a custom implementation is not needed.

FI-NEXT is the European project leading the sustainability and evolution of FIWARE Platform and FIWARE Community onwards from 2016. The FIWARE Community is an independent open community whose members are committed to materialize the FIWARE mission. This community is not only formed by contributors to the technology (the FIWARE platform) but also those who contribute in building the FIWARE ecosystem and making it sustainable over time.

### **1.1.3 Internet of things**

Thanks to increasingly efficient power management and advances in communication and microprocessors, it has become possible to digitalize functions of modern products, monitoring the status of such products thanks to sensors and then using that information to generate responses. A basic example would be a light sensor used to regulate the power status and intensity of a street lamp, so that it would only be powered when it is needed to.

The impact which IoT technologies can achieve is however not limited to the value a single product has. Instead, multiple products could be used together to further improve its functions. For instance, traffic sensors and pedestrian sensors could be used alongside traffic lights in order to optimize the traffic flow and reduce the amount of traffic jams. Giving everyday objects the ability to connect to a data network would have a range of benefits: making it easier for homeowners to

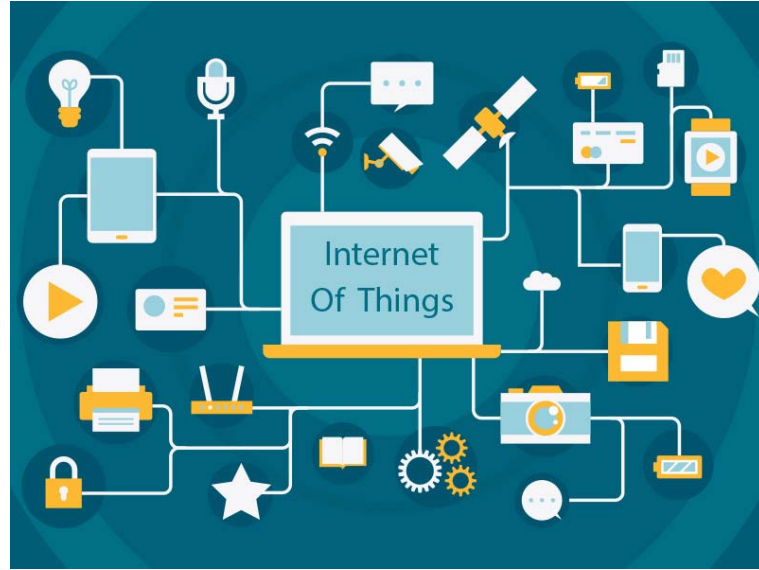


Figure 1.1: Internet of things

configure their lights and switches, reducing the cost and complexity of building construction, assisting with home health care[16].

The number fields of application of IoT technologies is quite large and diverse, being smart cities, smart industry and smart homes the most common ones. Smart cities aim to improve the efficiency of cities, some examples would be monitoring the air quality, status of parking spots or waste management. Smart homes look to improve the efficiency of homes, for instance by handling automatically the temperature of the house, notifying when food is low in the fridge or even buying and restocking it automatically. Smart industries look towards optimizing the production, for example connecting multiple production sites and managing stock of the resources needed for production.

## 1.2 Motivation

In a data ecosystem like the one of the FI-NEXT project, it is crucial to have data visualization and data monitoring capabilities in order to help its end users to extract value from the data. Therefore, it is a must to provide the data marketplace with tools that allow its end users to consume the data with the best visualizations available for the dataset, alongside the usage of the data through third party applications and services that would consume the data using the available APIs.

FIWARE's mashup platform, WireCloud, provides a great tool to generate dashboards that can be customized in order to fit the needs of the end users, but it lacks

the components needed to display great data visualizations that fit the needs said users and are able to handle the large data disparity available. Also, this components need to be correctly configured and connected through wiring in order to achieve the desired views, a task that, due to the large amount of different datasets and data sources, would be best to be automated, so that the user doesn't need to provide as much input in the generation of datasets nor to configure the wiring manually. WireCloud provides great tools to fully customize the dashboards, and by using generic WireCloud components, a wide range of views and data operations can be supported, this does, however, complicate the task of developing said components and also makes it harder to create and to configure the dashboard.

Thanks to the FIWARE data-models we can develop components aimed at those data-models, to provide a more in depth data visualization instead of a more generic one, making it easier to create powerful visualizations based on the FIWARE data-models by using homogenized data and allowing data to be categorized, so that data visualizations can be reused or partially reused among different datasets. This is important since a visualization could not be useful or lead to a mistaken data interpretation if used in another dataset carelessly.

## 1.3 Objectives

This project aims at developing data visualization and data monitorization dashboards for the FIWARE data marketplace so that users can easily generate new visualizations aimed at certain datasets to help them extract information from the data available. In order to achieve this, it is needed to develop new WireCloud components aimed at displaying data to allow the creation of said dashboards, this components would be of three types: Data harvesting components, whose rol is to harvest the data from a data source (such as CKAN); Data transformation components, whose rol is to transform the harvested data in order to get the data needed for the visualizations; and Data visualization components, whose rol is to display graphs and maps that will allow the user to extract useful information from the data. Due to the large amount of WireCloud components and disparity of the data available, creating dashboards with all the desired views can become a difficult task.

To tackle the hard generation of these data visualization dashboards and make it more feasible for all kinds of users, a system to generate them automatically based on the data available is needed. To help with this automatization, data models such as FIWARE's data models are quite useful, as they provide a way to normalize data structure and create dashboards based on the data available and on what other users with the same data model found useful. This would be achieved by homogenizing and giving semantic to the data, allowing the system to find properties and recognize

patterns on it, so that it can create data visualizations using generic components, that could be reused among many different types of datasets while still providing a useful data visualization and not leading the user into a mistaken data interpretation.

# Chapter 2

## State of the art

### 2.1 Smart cities

A smart city is an urban area where networks of electronic data collection sensors are used (Internet of Things), in addition to human input such as citizen feedback, to supply information to users or other devices in order to manage assets and resources efficiently, optimizing the efficiency of city operations and services and to connect to the citizens, providing them useful information about their surroundings. Examples of this would be traffic management, power plants management, waste management or air quality monitorization. In Smart Cities, digital technologies translate into better public services for citizens, better use of resources and less impact on the environment[5].

Madrid is an example of city trying become a smart city, where project MiNT (Madrid iNTeligente) is being implemented. This project scope is around environment, being its goal to improve the management of public services and to improve the quality of the services provided to its citizens. This project works by evaluating the quality of these services by using city sensors, inspectors and citizen reviews, and pay the service provider accordingly to the quality perceived. This works by opening more communication channels between the administration and the citizens, allowing them to open new issues when they see city issues and checking the status of those issues, alongside the reviewing the quality of the services received.

The city of Santander in Cantabria, Spain, has one of the largest sensor networks, having over 20,000 sensors connecting buildings, infrastructure, transport networks and utilities, monitoring pollution, noise, traffic and parking.

In the city of London, the SCOOT traffic management system optimizes green light time at traffic intersections by sending sensor data to a supercomputer, which then coordinates traffic lights across the city to improve traffic throughput and reduce traffic jams.



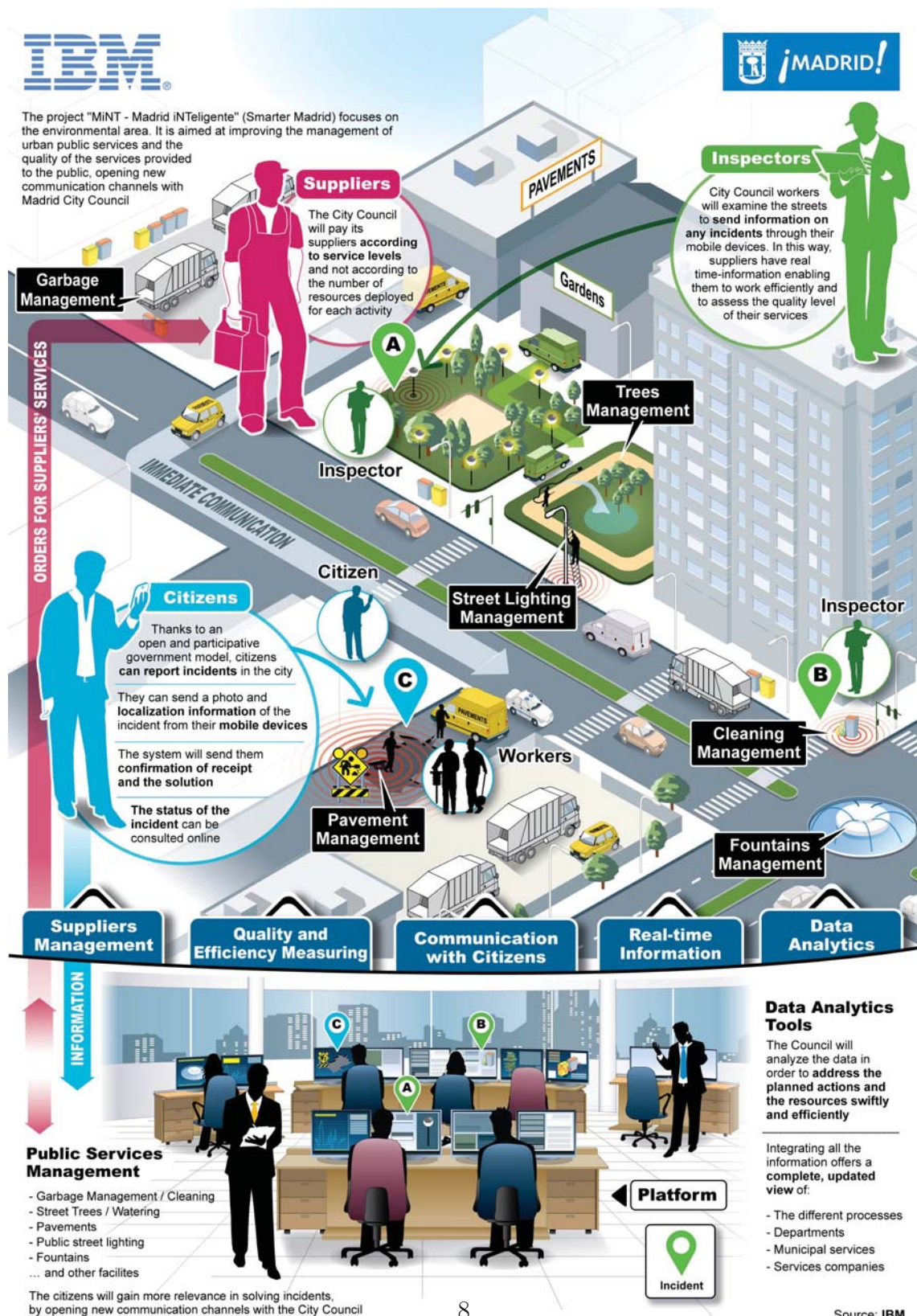


Figure 2.1: What is MiNT



## 2.2 CKAN

CKAN [8](Comprehensive Knowledge Archive Network) is a web based, open-source, free software management system used for the storage and distribution of open data aimed at data publishers wanting to make their data public. CKAN is mainly used is mainly used by public institutions to share their data with the general public.

CKAN is designed for publishing and managing data either through the web user interface or through the API, and it has a modular architecture, allowing plugins to add custom features.

The FIWARE CKAN Extensions<sup>1</sup> GE is a set of CKAN extensions, developed within FIWARE, that integrate the CKAN data portal with the FIWARE Generic Enablers, enhancing the default CKAN behavior with improved access control, data publication and visualization features.

- OAuth2: This extension enables the integration of CKAN with an external OAuth2 identity manager. This is needed in order to use CKAN private datasets from outside with the DataStore extension. It is also needed to perform OAuth2 request from CKAN, which is needed by the WireCloud View extension.
- NGSI View: This extension enables the integration of CKAN with NGSI, allowing the creation of datasets as queries to a NGSI Context Broker(such as Orion Context Broker).
- Private Datasets: This extension improve the default dataset permission by allowing the creation of protected datasets only accessible by a set of users. This allows monetization of datasets, creating this way a data marketplace.
- BAE Publisher: This extension enables the integration of CKAN with the Bussiness API Ecosystem(BAE)<sup>2</sup>, allowing the monetization of private datasets creating product offerings in the BAE.
- WireCloud View: This extension allows datasets to be easily visualized in WireCloud, by embedding a dashboard into CKAN and setting there a basic chart to display the data.
- Data Requests: This extension allows CKAN users to ask for data that is not yet available, allowing data publishers to know the data users want to access.

---

<sup>1</sup>FIWARE CKAN Extensions <https://catalogue.fiware.org/enablers/fiware-ckan-extensions>

<sup>2</sup>BAE <http://business-api-ecosystem.readthedocs.io/en/latest/>

## 2.3 NGSI

The FIWARE NGSI v2 (Next Generation Services Interface)[11] is a the Generic Enabler REST API intended to manage the entire life cycle of context information, such as updates, queries and subscriptions.

The NGSI API defines a data model for context information, a context data interface used for exchanging information using queries, subscriptions and update operations, and a context availability interface, used to obtain information about the context data interface.

The main elements of the NGSI data model are entities, attributes and meta-data. Entities are the center of the NGSI data model, and each entity represents a thing (physical or logical object). Each entity has an ID field, which identifies the entity, and a type field, used to describe the type of thing represented by the entity; the combination of both fields defines the entity. Each entity has a set of attributes, that have a name that describes the kind of property the attribute value defines, a type that defines the value type of the attribute, and a value, which contains the actual data of the attribute. Attributes can have meta-data, which provides additional information about the attribute, such as the type of the attribute, or the last modification date. Entities can have any attributes, being structure-less, even when being part of the same NGSI data model.

The three main interaction types with the NGSI API are one-time queries for context information, which retrieve data based on a set of filters, subscriptions to receive asynchronous context information updates based on the conditions set on the subscription, for instance, a subscription can be set so that only updates are received when a target attribute is modified, and lastly unsolicited updates that get invoked by the context providers.

The JSON representation of NGSI entities uses the following syntax: the object id property is the entity id, the object type property is the entity type, and each of the entity attributes has a object property with the name of the attribute with the value of the entity attribute. See example JSON 1.

There are some built in metadata properties that all entities have, which can be useful as they provide extra information about the entity. These metadata properties are: the date the entity was created, the date of the last modification of the entity, the previous value of an entity attribute which has been modified (for instance, after modifying the attribute location, stores the previous location), and the action type that triggered an update on a subscription (such as update if the entity was updated or the entity was removed).

NGSI v2 API is an improved version of the NGSI v1 API, which provides simplified payloads, reducing the size of the JSON payload, a RESTful approach instead of NGSI v1 approach, which was based on HTTP POST requests for all operations,

```
{  
  "id": "entityID",  
  "type": "entityType",  
  "attr_1": "val_1",  
  "attr_2": "val_2",  
  "attr_N": "val_N"  
}
```

Listing 1: NGSI JSON example

and a powerful query language that allows the user to filter and sort data by different criteria, including geographical relationships such as coverage.

## 2.4 Orion Context Broker

The Orion Context Broker [14] is the Generic Enabler implementation of FIWARE's Publish/Subscribe Context Broker GE, which implements both FIWARE NGSI v1 and FIWARE NGSI v2 APIs, that allow the creation of context elements and to manage context elements through updates, queries and subscriptions.

Orion Context Broker allows you to manage the entire life cycle of context information including updates, queries, registrations and subscriptions. Orion is a NGSIv2 server implementation to manage context information and its availability. Using the Orion Context Broker, you are able to create context elements and manage them through updates and queries. In addition, you can subscribe to context information so when some condition occurs (e.g. the context elements have changed) you receive a notification. These usage scenarios and the Orion Context Broker features are described in this documentation.

## 2.5 FIWARE DataModels

The FIWARE DataModels [7] are data schemas that add semantics and standardized syntax for data among different sectors, mainly aimed at smart cities, that are harmonized to enable data portability for different applications, such as Smart Cities. They make it possible to use these models when working with the defined types, which allow the reutilization of the components and applications used when working with the data. These datamodels are intended to be used together with the

FIWARE NGSI, by using a subscription to the Orion Context Broker and receive real time updates on the data provided by the sensors.

Most of these dataset are intended to get information through sensors or other means where the location of the data source is important, as such, it could be useful to be able to display most of the information they provide keeping in mind the geographical information, for example, in a map.

The following are the existing FIWARE DataModels:

### **2.5.1 Alerts**

The Alert DataModel is used to model alert entities of many types, and its intended to generate notification for a user or to trigger other actions based on the alerts. Currently there are five alert categories:

- **Traffic:** These alerts notify problems around traffic, such as traffic jams, accidents or road works.
- **Weather:** These alerts notify problems and hazards of the weather, such as rainfall, snow or ice, heat waves and tornadoes.
- **Environment:** These alerts notify about environmental problems, such as air or water pollution.
- **Health:** These alerts notify about health issues of citizens and patients, such as asthma attacks, heart attacks or fallen patients.
- **Security:** These alerts notify about security problems, and are most useful for keeping people secure or coordinating police movements. These alerts include robberies, assaults, and suspicious actions.

### **2.5.2 Parks and gardens**

These datamodels are intended to model different green spaces of a city

- **Park:** A park is an open space provided for recreational use A park is an area of open space provided for recreational use, usually designed and in semi-natural state with grassy areas, trees and bushes. Parks are typically open to the public, but may be fenced off, and may be temporarily closed for instance, at night time.

This datamodel contains information about the park, which is mainly useful for touristic porpoises.

- **Garden:** Gardens are distinguishable planned spaces, usually outdoors, set aside for the display, cultivation, and enjoyment of plants and other forms of nature. Gardens can also be a part of a park and be open to the public. Gardens are divided in smaller parts, named flower beds.

This datamodel contains information about the park, such as its closing hours, and last watering dates, which can be useful from both a management point of view and a touristic point of view.

- **Flower bed:** Flower beds are garden plots in which flowers or other plants are grown.

This datamodel describes the properties of the flower bed, such as dimensions, shape and soil properties.

- **Green space record:** This entity contains data about the conditions recorded on a particular area or point inside a garden or related green space.

### 2.5.3 Environment

These datamodels describe entities involving environmental issues

- **AirQualityObserved:** These entities represent the air quality at a certain place and time, having properties such as quantitative and qualitative pollutant levels (for instance CO<sub>2</sub>), precipitation and humidity.
- **WaterQualityObserved:** These entities represent water quality at a certain time and water mass, such as rivers, lakes, seas and swimming pools. These entities have properties like water temperature, salinity and pH among others.
- **NoiseLevelObserved:** These entities represent the noise levels at a certain place and time, in order to try and estimate the noise pressure levels.

### 2.5.4 Points of interest

These datamodels describe entities as Points of Interest (PoIs). These datamodels have a description property which can be used to provide extra information and are mainly interesting from a touristic point of view.

- **Point of Interest:** These are generic Point of Interest, used to define any kind of point of interest. These have a category property that can be used to define the type of point of interest.

- Beach: These datamodel is used to represent a beach, and provide some information about it, such as beach type (for instance sand or stones), and information about its occupation level and the facilities available.
- TouristInformationCenter: These datamodel is used to represent tourist information centers, and provide extra information such as its opening hours and its contact phone.
- Museum: These datamodel is used to represent a museum, providing information about it, such as the type of things displayed on it, if there is any kind of special display, and its opening hours.

### 2.5.5 Civic issue tracking

These datamodel describes entities intended to track civic issues, such as holes in a road pavement and broken streetlights, and are designed to enable interoperability between FIWARE NGSI and the Open311 standard. These entities have properties that define the type of issue, its status (closed, open or in progress), notes about the issue and a description of the issue.

### 2.5.6 Street lighting

These datamodels are used to describe the elements used to control street lamps.

- StreetLight: These datamodel is used to define street lamp entities, and has properties such as the status of the street lamp (on, off or broken), the height and intensity of the lamp, and its group and control cabinet among others.
- StreetLightGroup: These datamodel is used to define a group of street lamps that are coordinated together as a group by a control cabinet.
- StreetlightControlCabinet: These datamodel is used to represent the a street lamp control cabinet, used to manage multiple streetlights individually or in groups. This entities have properties such as power usage, intensity and status which help visualize and manage the status of the street lights of a city.GF

### 2.5.7 Device

These datamodel is used to represent any kind of device, such as a sensor or an actuator in a smart city environment. A device is a tangible object which contains some logic and produces or consumes data. This model properties identify the category of the device, its controlled properties and value, and information about the device itself such as its IP and MAC addresses.

### 2.5.8 Transportation

These set of datamodels are used to represent data related to traffic and transportation.

- **TrafficFlowObserved:** This datamodel represents information about the flow of traffic in a certain place at a certain time. Its properties define data about the road, the lane where the flow is being analyzed, and data about the flow itself, such as the congestion, average speed and the number of vehicles detected.
- **Vehicle:** This datamodel represents information about a vehicle, providing information about its current location, data about the vehicle itself such as the vehicle type, its plate and its owner, and data about where is it heading or the weight of its cargo
- **Road:** This datamodel represents a road, and provides information about it such as length, number of lanes and speed limit.
- **RoadSegment:** This datamodel is like the Road datamodel, but it only provides information about a segment of a road.

### 2.5.9 Indicators

These datamodel is used to represent data about key performance indicators (KPI). KPI are used to identify and evaluate an activity, and can be used to easily check the status of said activity. An example where this could be used is a sensor checking the temperature of a garden plot, where you wouldn't want the temperature to rise from a certain value These datamodel provide properties that help define what activity is the indicator monitoring, how was the indicator calculated and the value it has.

### 2.5.10 Waste management

These datamodels are used to represent information from a waste management perspective, defining the properties of the waster containers of a city.

- **WasteContainer:** This datamodel helps define a waster container, providing attributes about its location and properties, such as its filling level, its current weight and its temperature.
- **WasteContainerIsle:** This datamodel is used to group multiple waste containers in a single entity, so that they can be managed together.

This datamodel can be used alongside the transport datamodel in order to manage the waste levels of the containers and send the garbage trucks to collect it.

### **2.5.11 Parking**

This set of datamodels are used to define different kinds of parkings and is useful to help manage and monitorize attributes such as their occupancy levels.

- **ParkingSpot:** These datamodel represents a single parking monitored spot, and provides information about it such as its location, its status, such occupied, free or closed, and information about the parking spot dimensions and category.
- **ParkingGroup:** These datamodel represents a group of parking spots that are normally close together or grouped up, and contains information like the active hours of the parking spots, maximum parking time, charge type and its fees. It also contains information about its occupancy levels.
- **OnStreetParking:** These datamodel represent a type of parking group in which all parking spots are on-street with free entry (although the parking might be metered), and as such has the same properties as a the **ParkingGroup** datamodel.
- **OffStreetParking:** These datamodel represent a type of parking group in which all parking spots are off-street with defined entries and exits to the parking, and as such has the same properties as a the **ParkingGroup** datamodel.

### **2.5.12 WeatherObserved**

This datamodel is used to define the properties of the weather observed at a certain place and time. This datamodel has properties such as the weather type(rain, snow, ice, ...), visibility, temperature and humidity among others.

## **2.6 WireCloud**

WireCloud[15], the Generic Enabler implementation of FIWARE's Application Mashup GE, is a mashup platform aimed to allow the creation of composite applications with ease, so that the end user does not need programming skills in order to create the mashups that display information from multiple sources (mashup).

Using WireCloud users can create dashboards (also known as workspaces), where the logic of the mashup is stored. Users can configure workspaces privacy, allowing



this way workspace sharing with other users which are allowed to see the dashboard or even modify the dashboard and its components, if the guest user is granted enough permissions by the workspace owner.

Most modern mashup applications only allow the addition of widgets, with basic interactions between each other, while WireCloud's main advantage over other mashup tools is that it allows to fully customize the behavior of the dashboard by editing the 'wiring' of the dashboard/workspace, which handles the inner logic of the dashboard, and modifying it changes the way different components behave and interact with each other, allowing a really powerful customization of the dashboard produced. WireCloud dashboards can be set to be private or public, allowing access to only authorized members and the creation of public dashboards that can be used to share views with anyone. Dashboards can be also embedded into other websites while keeping all its functionality.

WireCloud uses three types of components:

- **Operator:** WireCloud operators are the components used to harvest and prepare the data for the widgets, therefore, they are not rendered on the dashboard view and can only be accessed from the wiring view. They are used to access data from other sources and transform in order to generate and change the information displayed by the widgets, so that it fits the needs of the user.
- **Widget:** WireCloud widgets are the components used to display information to the user, therefore, widgets are the only components displayed on the dashboard view.
- **Mashup:** WireCloud mashups are used to pack a dashboard configuration into a single file, so it can be duplicated easily by creating a new workspace using the mashup as a template. Mashups can have parameters, that need to be filled when creating the new workspace in order to customize the preferences of the WireCloud components, changing this way the functionality of the resulting dashboard so it fits the needs of the end user.

While WireCloud operators are intended to perform all data gathering and data transformation operations while leaving data representation and data visualization to widgets, component developers can choose otherwise and develop widgets that also perform this kind of operations. This can be useful when a widget and operator always go together, removing the need to add them separately. However, this could end up limiting the reusability of those components, for example, by not allowing the operator part to feed into other WireCloud components.

Every WireCloud workspace has two views:

- **Dashboard/workspace view:** This is the usage view of the mashup, where users interacts with the interface of the widgets and the data is displayed. Since this

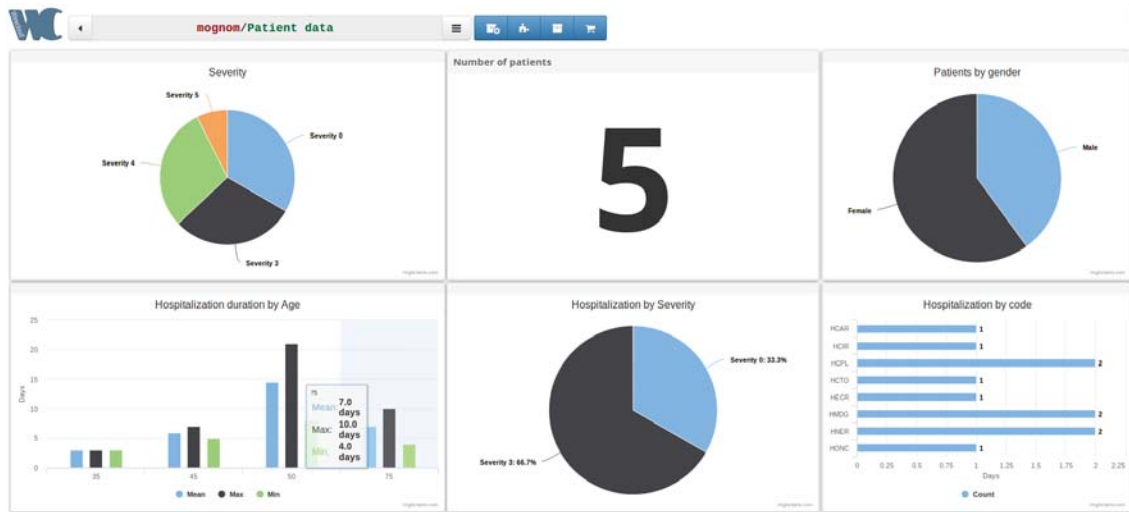


Figure 2.2: WireCloud dashboard view

is the main view of WireCloud, its where most of the time will be spent by end users, and the only view where data can be used by an end user. It can be divided in multiple tabs, allowing multiple views to be merged into a single dashboard; this is useful to help reduce processing time by not reprocessing data when changing views.

- **Wiring view:** This is the configuration view, where the dashboards are created and configured by managing the settings and connections between the WireCloud components in order to create and customize mashups so they fit the needs of the user. By managing the connections of the components, user can fully edit the inputs and outputs of them, allowing full control over the flow of the dashboard's logic and results. This view is only used while creating and configuring the dashboard, and therefore is not intended to be used nor even seen by end users. Only the owner of the dashboard has access to the wiring view.

## 2.7 Data visualization tools

Never before in history has data been generated at such high volumes as it is today. Exploring and analyzing the vast volumes of data is becoming increasingly difficult. Information visualization and visual data mining can help to deal with the flood of information[22]. This has caused a need for such visualization tools, making a market for it that is quickly rising.

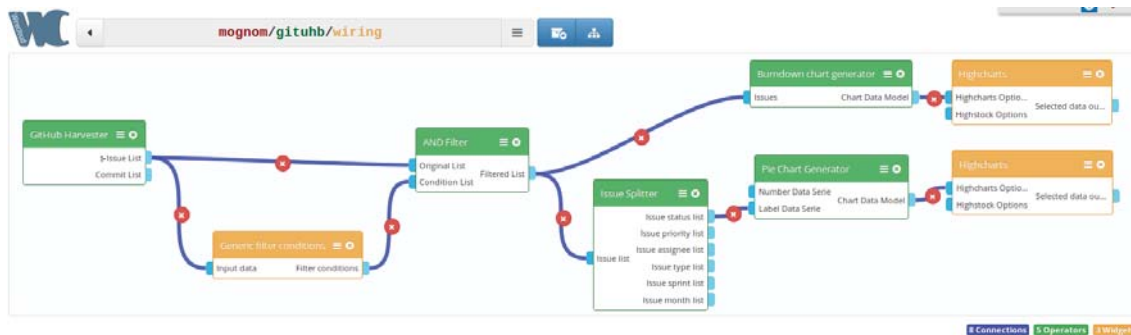


Figure 2.3: WireCloud wiring view

Currently, the biggest data visualization tools available tend to be easy to use, most of them having simple drag and drop interfaces used to generate the data visualizations. The main downside of this is that it increases the complexity when the end user needs something that is not common, being the drag and drop interface unable to satisfy its needs easily, often needing to customize things or directly being unable to make those adjustments.

### 2.7.1 Tableau

Tableau<sup>3</sup> is one of the most used data analysis and visualization tools, allowing its users to easily create and share interactive visualization dashboards, combining data exploration and data visualization into a single tool, and allowing users to setup connections to remote data stores such as cloud databases. One of its selling points is its simplicity of use and the ability to create powerful interactive data visualizations.

Tableau can be run from a desktop application or using a smart phone or a web browser, furthermore, Tableau can be setup to run on a scalable server, allowing multiple users to manage the same data and share its views, while using the processing power of the server.

### 2.7.2 Qlikview

Qlikview<sup>4</sup> is the main competitor of Tableau, providing a data visualization tool to create data visualization dashboards by using guided analytics to help the user discover data insights in a flexible manner and allowing users to combine multiple data sources into a single visualization.

<sup>3</sup>Tableau website <https://www.tableau.com/>

<sup>4</sup>Qlikview website <https://www.qlik.com/us>

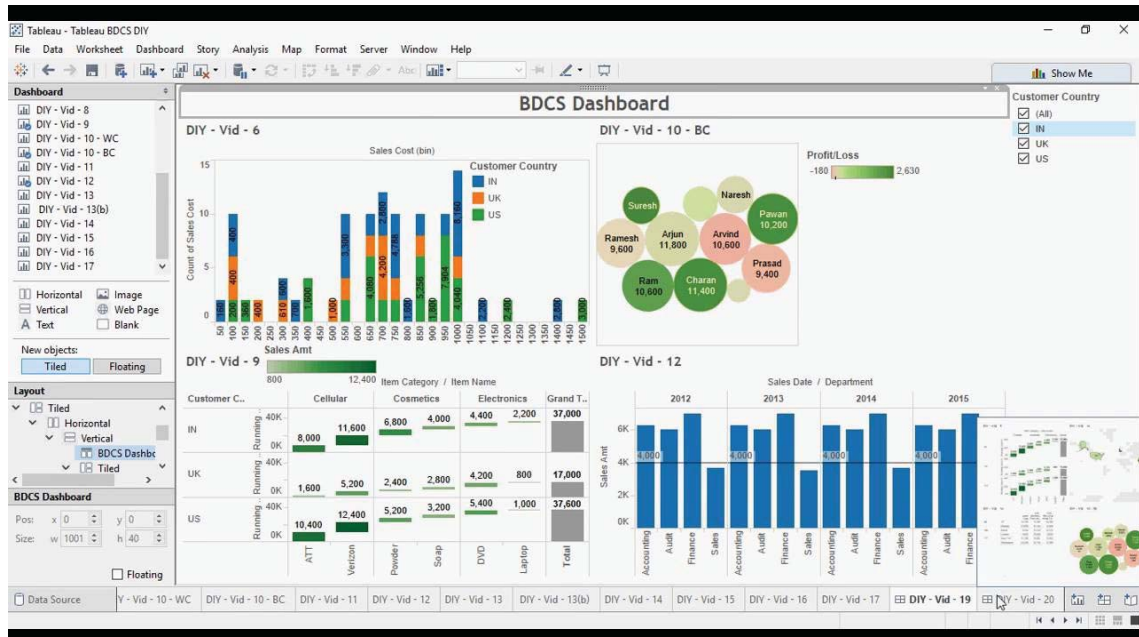


Figure 2.4: Tableau example view

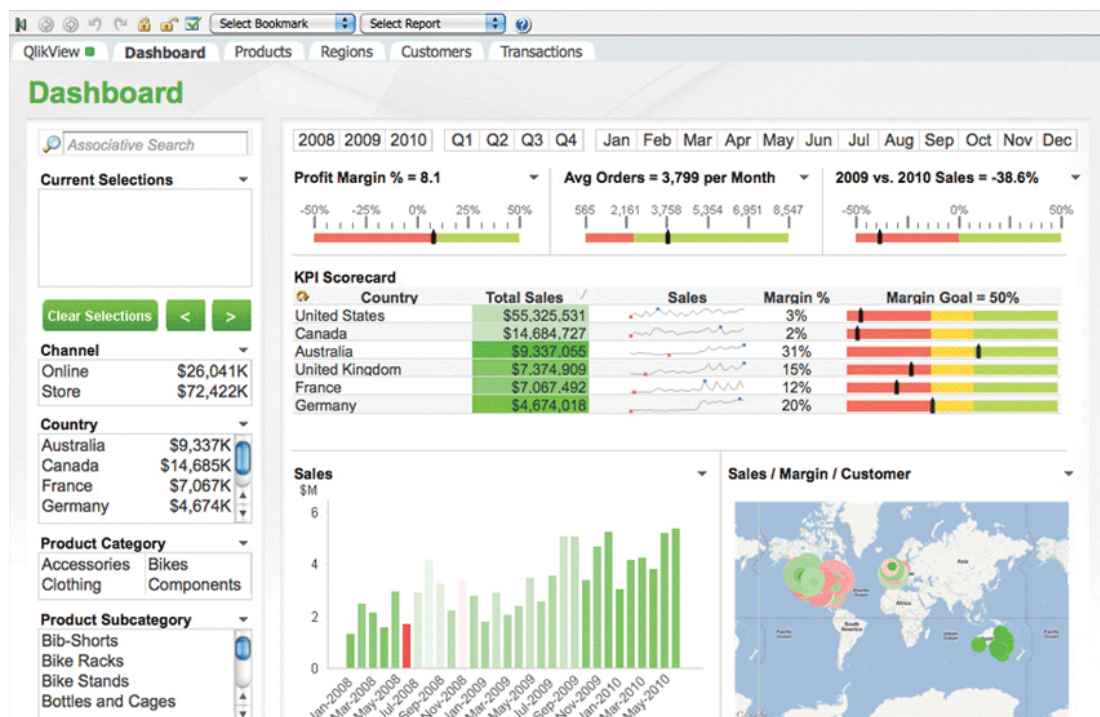


Figure 2.5: Qlikview example view

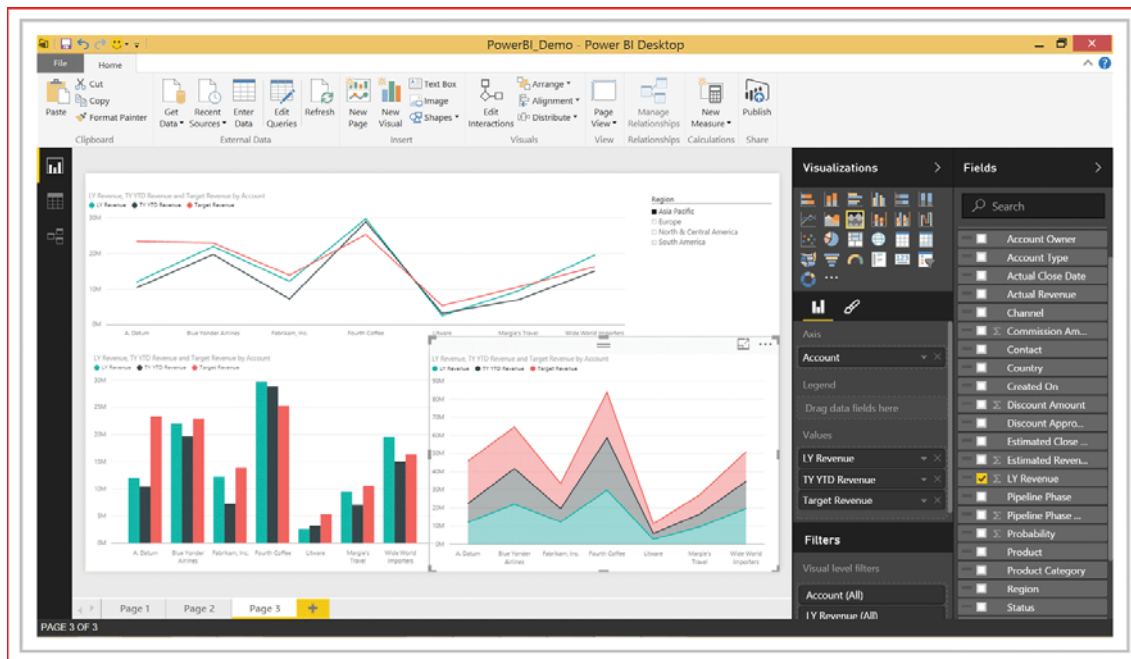


Figure 2.6: Power BI example view

Qlikview is a cloud solution, allowing users to share data and visualization among other members of the company in a secure manner.

### 2.7.3 Power BI

Power BI<sup>5</sup> is a suite of business analytics tools that deliver insights throughout your organization. Connect to hundreds of data sources, simplify data preparation, and drive ad-hoc analysis. Produce beautiful reports, then publish them for your organization to consume on the web and across mobile devices. Everyone can create personalized dashboards with a unique, 360-degree view of their business. And scale across the enterprise, with governance and security built-in.

### 2.7.4 Knowage

Knowage<sup>6</sup>, the Generic Enabler implementation of FIWARE's Data Visualization GE, is a open source suite for modern business analytics, being able to use traditional sources as well as big data systems. Knowage is the evolution of the a previous tool named SpagoBI.

<sup>5</sup>Power BI website <https://powerbi.microsoft.com/en-us/>

<sup>6</sup>Knowage website <https://www.knowage-suite.com/site/home/>

Knowage provides a modular offering, with each of the sub-products addressing an specific domain. These modules can be combined with each other.

- BD: The big data module is used to analyse data stored on big data systems, giving users the opportunity to work with big data sources and traditional ones. It also provides users with a drag & drop query builder so users can explore their data.
- SI: The smart intelligence module provides users with easy access to enterprise structured data by using pre-built analysis.
- ER: The enterprise reporting module delivers punctual information to a diversified set of users via static reports.
- LI: The location intelligence module allows users to plot their data over a map or a picture.
- PM: The performance management module allows users to measure and evaluate their bussiness performances and react to critical events with the use of alerts and notifications.
- PA: The predictive analysis module provides its users the ability to perform data mining techniques for forecasting and prescriptive purposes.

### 2.7.5 Datadog

Datadog<sup>7</sup> is a data visualization tool oriented to real-time performance monitoring to create dashboards where users can see metrics about the applications running or the performance of the hosts or virtual machines.

Datadog dashboards are created in a drag and drop manner so users can customize their dashboards to fit their needs. Users can collaborate to create dashboards and comment on different events of the metrics.

## 2.8 Technologies used

This section will describe the technologies used in the development of this project.

---

<sup>7</sup>Datadog website <https://www.datadoghq.com>



### 2.8.1 Python

Python <sup>8</sup> is an interpreted high-level programming language for general-purpose programming. Python has a design philosophy that promotes code readability, most notably by requiring the use of whitespace instead of brackets.

Python is a multi-paradigm language, supporting object oriented, imperative and functional programming. It also has a dynamic type system and automatic memory management.

Both WireCloud and CKAN extensions are developed using Python, so it will be used when extending WireCloud functionality and when developing CKAN extensions like the WireCloud View CKAN extension.

### 2.8.2 JavaScript

JavaScript (JS) is a high level interpreted programming language that is usually used on the client side by web browsers in order to create dynamic websites. JavaScript is a multi-paradigm language, supporting object oriented, functional, and event-driven programming. Nowadays JavaScript is also being used on the server side, with technologies like Node.js.

WireCloud components are developed on JavaScript, so it will be the most used programming language during the development of this project.

Since JavaScript code is run on the client, it does not have the data stored at the backend server from the get go, so the JavaScript application needs to query for said data, targeting the backend server, or other servers. The chart and map technologies used in these project and defined below are built on JavaScript.

### 2.8.3 HTML and CSS

HTML (HyperText Markup Language) is a markup language mainly used to create web pages and web applications. HTML documents are a set of HTML elements delimited by tags; these tags define the behavior of the elements, therefore defining the structure of the web page.

CSS (Cascading Style Sheets) is a style sheet language used to define the presentation of an HTML document, customizing this way the looks of the web page defined by HTML, including layout, colors and fonts. CSS provides web pages powerful ways to adapt its looks to the browser and its size, improving accessibility.

HTML alongside CSS and JavaScript are the three main technologies used in the the World Wide Web.

---

<sup>8</sup>Python <https://www.python.org/>

### 2.8.4 Highcharts

Highcharts<sup>9</sup> is a SVG-based library, written in pure JavaScript, aimed at creating dynamic and interactive charts with ease, created and developed by a Norway based company named Highsoft. Highcharts is open source and free for non commercial use, making it one of the most used charts libraries alongside GoogleCharts. Since Highcharts is designed with mobile browsers in mind, everything from multi-touch zooming to touch-friendly tooltips responds greatly on mobile platforms. Users can use the Highcharts library to prepare interactive and professional-quality charts and graphs for their applications quickly and easily [4]. Highcharts has both free and paid options.

Highcharts charts are created by passing a set of options, which define the characteristics of the chart, such as type, format of the chart legend, and the data series to be used. There are addons that add more chart types, such as gauge charts. Other product based on HighCharts code is Highstock, a charting solution optimized for financial data and large datasets, allowing users to plot stock charts and general timeline charts easily.

### 2.8.5 GoogleCharts

Google Charts is an interactive Web service that creates graphical charts from user-supplied information [18]. GoogleCharts is a powerful, simple and completely free chart library developed by Google. This library is aimed at embedding charts on HTML using JavaScript.

GoogleCharts charts are easy to make and are highly customizable by providing a set of options that can be modified to change the looks of the chart in order to fit the needs of the users, this, in addition to it being free, makes it one of the most used charting tools alongside Highcharts.

Since the input needed to configure GoogleCharts is different from the one needed by HighCharts, WireCloud components intended to work with one of these technologies would need changes in order to also support the other technology.

### 2.8.6 Google maps

Google maps[21] is a free web mapping service developed by Google, offering satellite images, street maps, panoramic views of streets, traffic conditions and route planning. Google maps is a powerful mapping tool that can be used easily to display maps and add PoI markers and other icons to it. Google maps require a API key to use its services; the free license only allows to 25,000 uses. In order to unlock

---

<sup>9</sup>Highcharts <https://www.highcharts.com/>



more uses, a fee for using the API has to be paid. This can make it harder to create a WireCloud widget using Google maps intended to be used by a large amount of people, since the rate limits might be exceeded.

### 2.8.7 OpenLayers map

Openlayers<sup>10</sup> is a free, open source, JavaScript library aimed at creating dynamic maps to be embed on a web page. Openlayers can display map tiles, vector data and markers loaded from any source. Unlike Google maps, OpenLayers doesn't have its own map layer, needing to use other layers such as OpenStreetMap(OSM)<sup>11</sup>, which is a free, open source, collaborative map of the world. The main advantages Openlayers has over Google maps is that it gives the user more flexibility, being able to choose any map provider (such as OpenStreetMap), has vector support and since OpenLayers is open source, users can add their own features to satisfy their needs.

## 2.9 Leaflet map

Leaflet<sup>12</sup> is a lightweight open source JavaScript library used to create mobile-friendly interactive maps designed with simplicity, performance and usability in mind. Leaflet can be extended with plugins in order to bring new functionalities. Leaflet, like OpenLayers, lacks its own map layer, therefore making use of other layers such as the OpenStreetMap layer. The main advantages of Leaflet over the other map libraries is its simplicity, making it very easy to implement the needed functionalities when working with it.

---

<sup>10</sup>Openlayers <https://openlayers.org/>

<sup>11</sup>OpenStreetMap <https://www.openstreetmap.org>

<sup>12</sup>Leaflet <https://leafletjs.com/>



# Chapter 3

## Chart types

Choosing the right chart for the right situation is an important task, since otherwise its possible to confuse the user or make him reach a mistaken data interpretation, therefore, designing quality charts is extremely important[3].

Charts can be classified in four basic presentation types, based on the type of information they are better at displaying:

- Comparison: This charts are used to compare data, making it easy to view the extrema of the analyzed data, such as the maximum and the minimum. This charts are ideally used when the goal is to compare different values. An example of this kind of charts would be the column chart.
- Distribution: This charts are used to view the distribution data, helping the user spot anomalies and outliers on the data. An example of this kind of charts would be then scatter chart.
- Composition: This charts are used to analyze the data composition, allowing the user to see which attributes have more weight on the data. An example of this kind of charts would be the pie chart.
- Relationship: This charts are used to view the relationship between multiple attributes of the data. This can be used to check the correlation, or lack of ,between multiple data attributes. An example of this kind of charts would be the bubble chart.

### 3.1 Column chart

The column chart is a kind of chart where data is displayed in columns, being the X axis (horizontal) the items displayed, and the Y axis (vertical) the value of

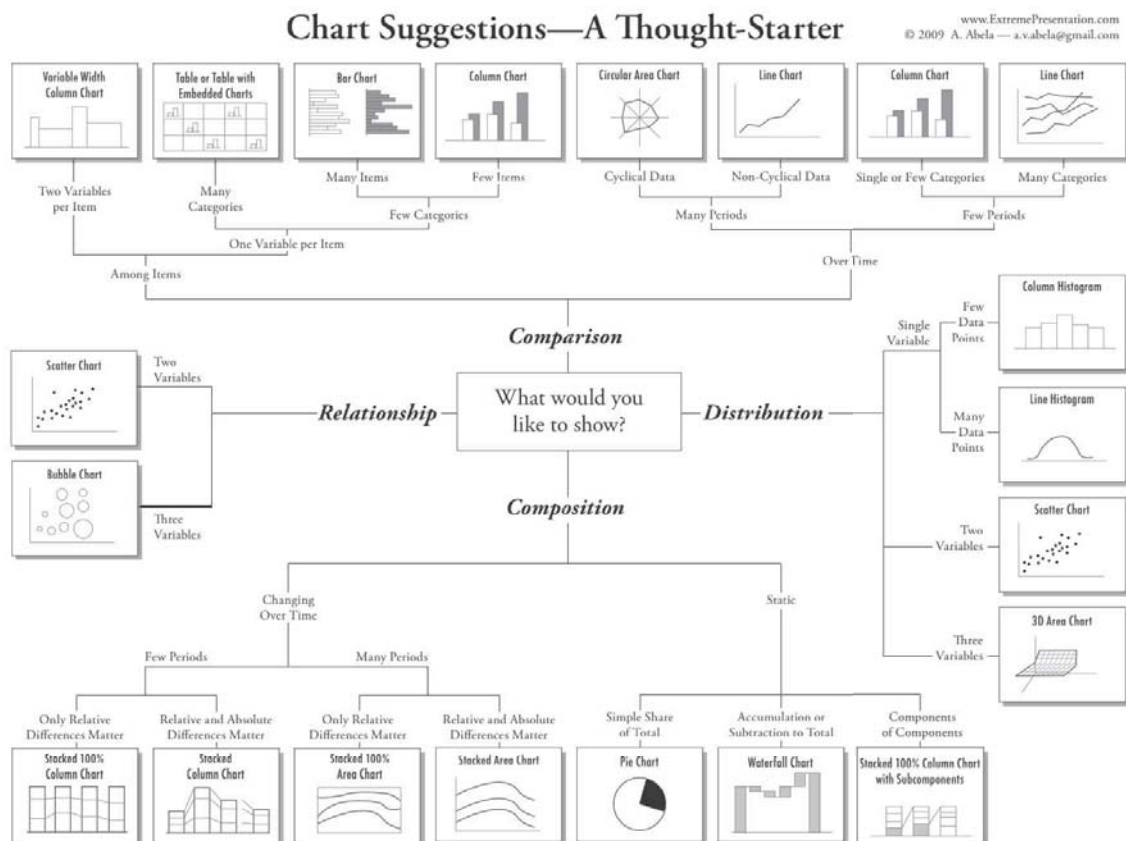


Figure 3.1: Chart selection diagram

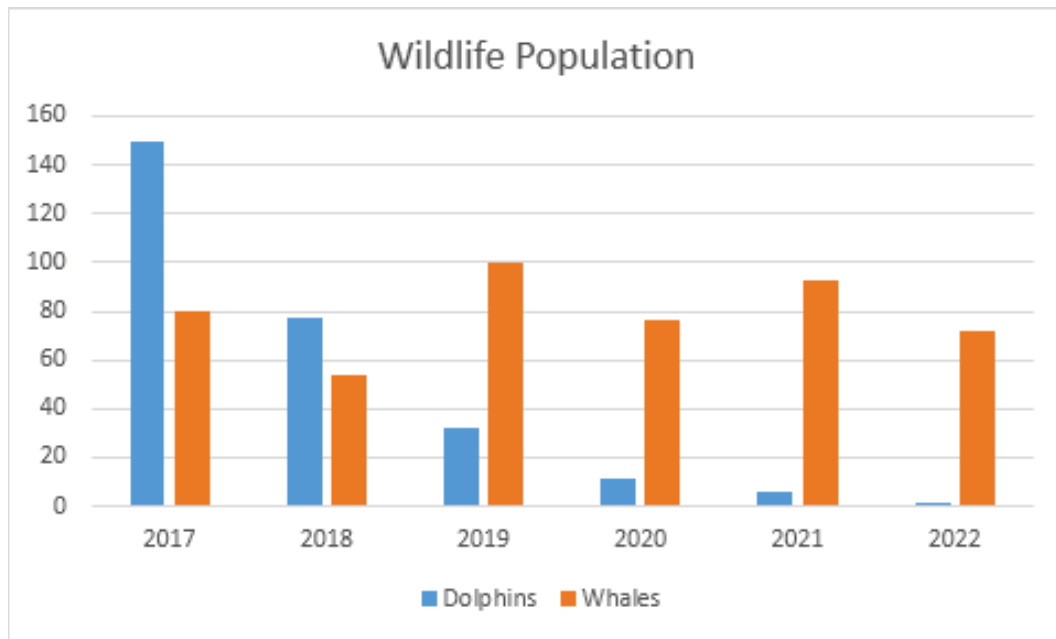


Figure 3.2: Column chart displaying populations each year

the data attribute. For example, the Y axis representing number of orders and the X axis being the month, showing this way the accumulate number of orders done each month as columns, allowing for easy comparison.

Multiple categories can be represented for each item using colors. In the previous example we could differentiate orders by country, splitting each month column into countries by using different colors for each country.

This kind of charts are useful to compare values for different categories (not many items nor categories), as well as comparing values over a period of time (not many categories nor periods). In this last case, the X axis should be always ordered to run from left to right.

Column charts become less effective as the amount of different items and categories rises, making it more likely to confuse the user and have him reach a mistaken data interpretation. Up to five-seven different items and up to 4 categories are recommended.

### 3.1.1 Stacked column chart

Stacked column charts are a kind of column chart, where instead of plotting the various categories in multiple columns with different colors, they are displayed on the same column, as sections of it.

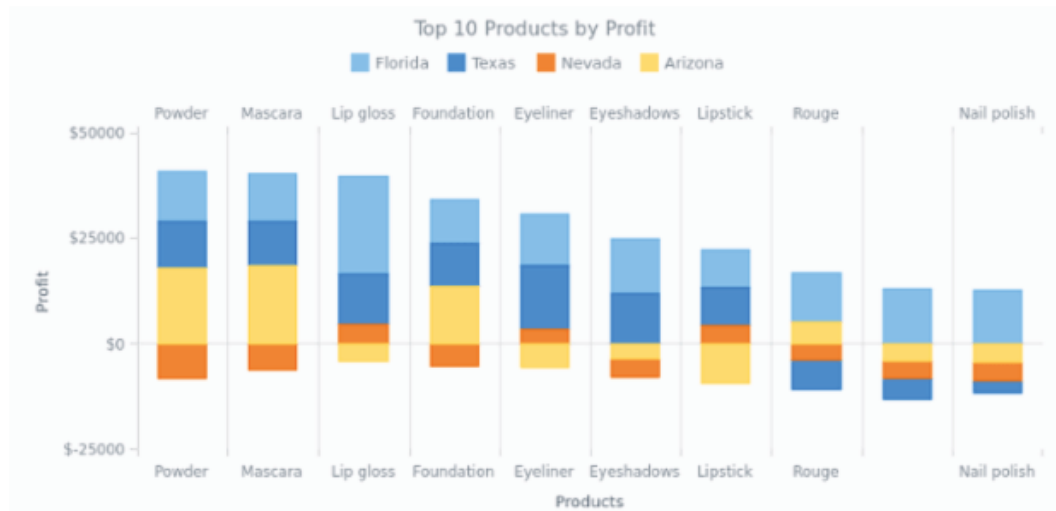


Figure 3.3: Stacked column displaying products by profit

This charts are used to show the composition of the data set and should not try to display too many composition items or it can get cluttered and confusing.

### 3.1.2 Column histogram

Column histogram, also known as “histogram”, is a variation of column charts, where the X axis contains bins of value ranges from a category. For example, X axis being time spent on a shop and Y axis being money spent.

This charts are used to show the distribution of the categories chosen, when there are few bins. Otherwise a line histogram should be used.

## 3.2 Bar chart

The bar chart is quite similar to the column chart, but displayed horizontally.

The main advantage over column charts is that, since tags are displayed horizontally, bar charts allow for bigger item tags. They also allow the display of negative values and are less limited in the amount of items that can be displayed without cluttering and confusion. On the other hand, this charts are not recommended when trying to display multiple data categories.

This kind of charts are useful to compare among items with few categories and many items.

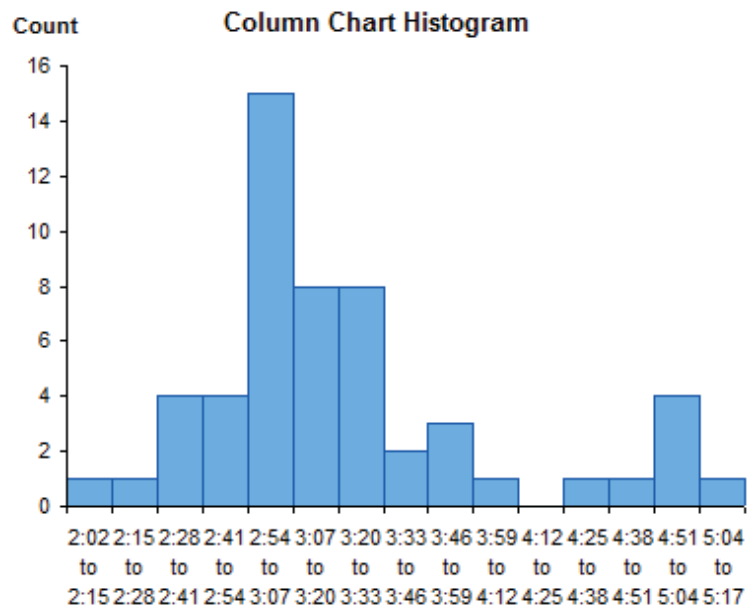


Figure 3.4: Example column histogram

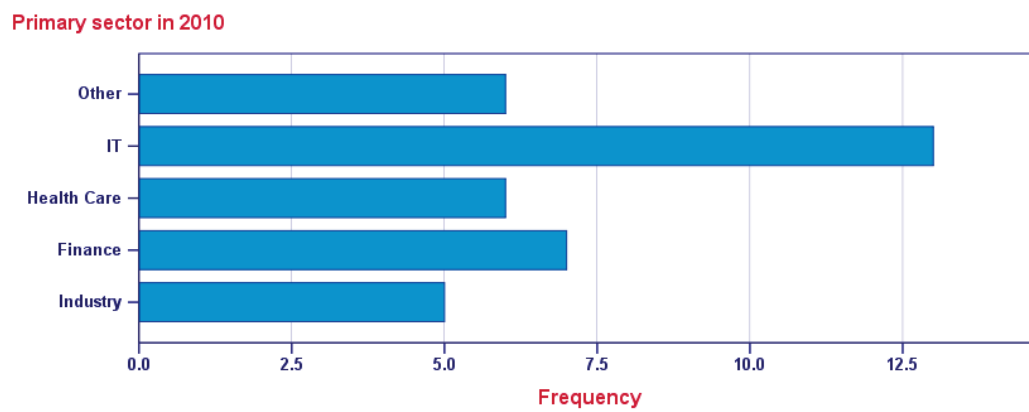


Figure 3.5: Bar chart displaying primary sector in 2010

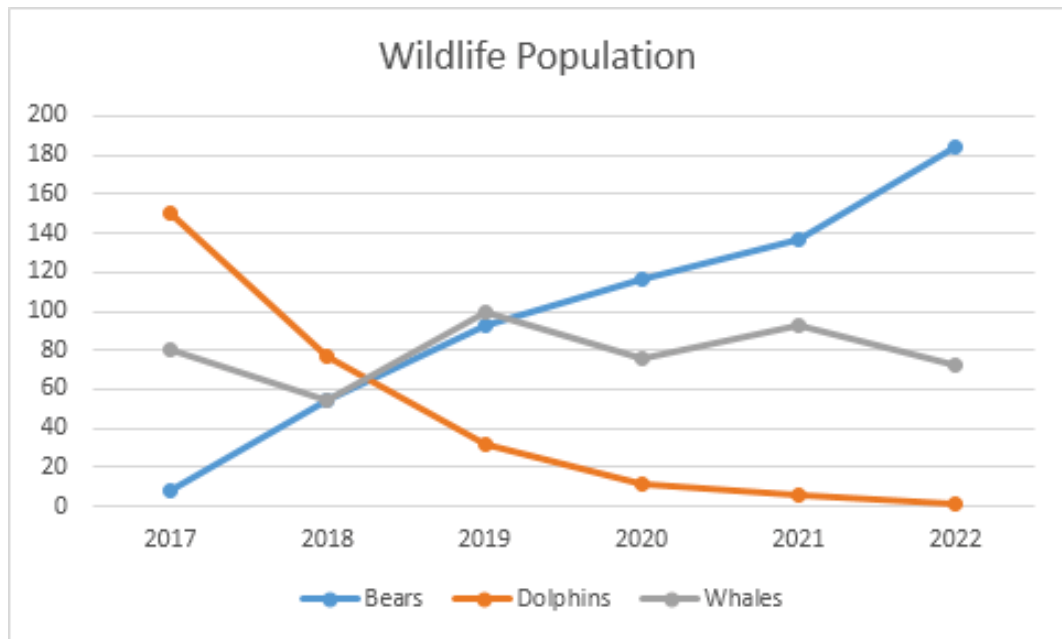


Figure 3.6: Line chart displaying the evolution of multiple populations

### 3.3 Line chart

Line charts are used to display a continuous data set, being best suited for trend based visualizations over time and giving emphasis to the flow of the values. Multiple items can be displayed by using multiple lines with different colors or shapes.

This kind of charts are useful to compare such flow of values over time when the data is not cyclical and has many periods(different tags of X axis) or the data has few periods with many categories.

When using line charts it is recommended to start the Y axis on 0 to avoid confusion. It is possible to start the Y axis on other values to show the evolution.

### 3.4 Area chart

Area charts are similar to line charts, the main difference being that the area below the lines is now filled. This charts are used to show cumulative values and compare the value composition over different periods of time.



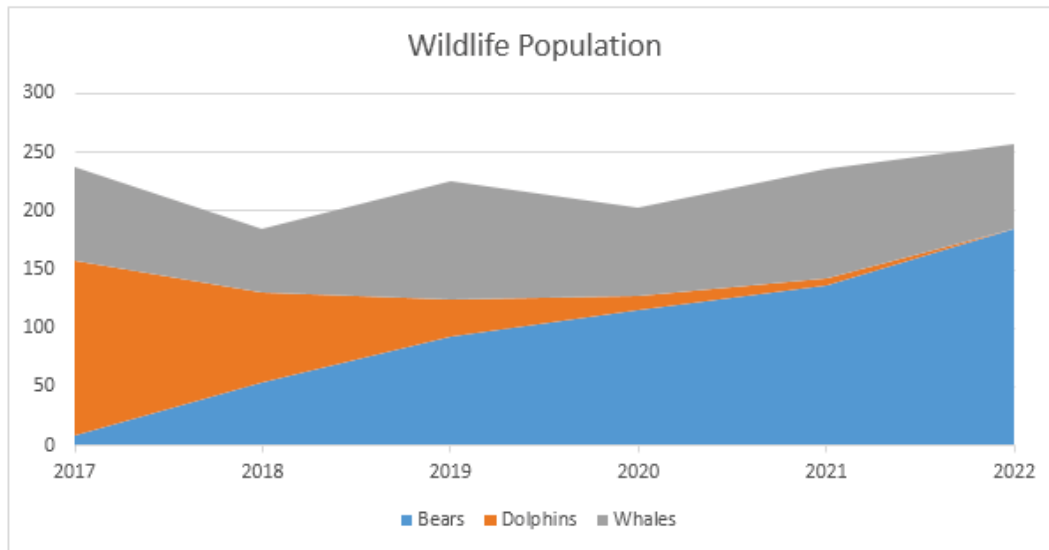


Figure 3.7: Area chart displaying the evolution of multiple populations

### 3.5 Pie chart

The pie chart is, alongside the column chart one of the most used charts. In this charts, categories are displayed as pie slices of a circle, the size being dependent on the value of the items in that category.

This charts are usually not useful and are not recommended, since its very difficult to compare the angles of the pies to understand properly this charts. Column charts should be used instead when possible, as those do allow proper comparison of values. Pie charts can be useful when one of the categories is much bigger than the others, where they can be used to show such difference clearly. Sizes are usually sorted.

### 3.6 Scatter chart

Scatter charts display data in a 2D grid, displaying two categories, one for each axis, and a dot for each item of the data set based on its value on both categories.

This charts are useful to display the relationship between multiple categories, making it easy to see the correlation or lack off correlation between two categories. They can also be used to show the data distribution and find data clusters and data anomalies.

The main advantage of this charts over the others is that it allows to represent large amounts of data without cluttering or generating confusion on the user side.

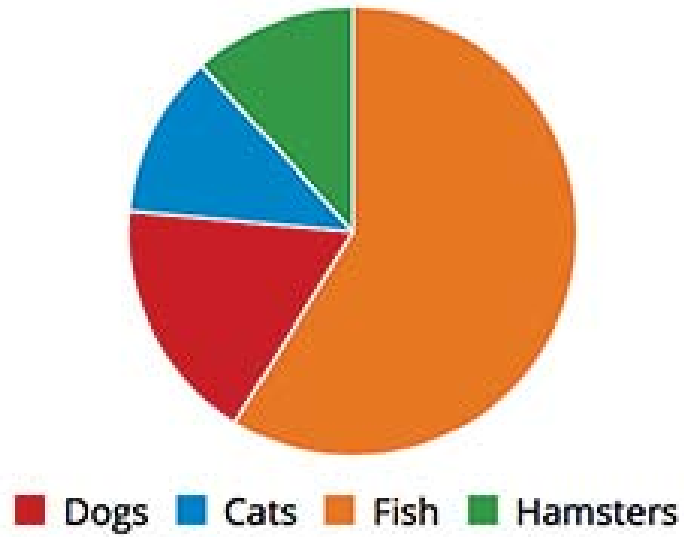


Figure 3.8: Pie chart displaying pet percentages

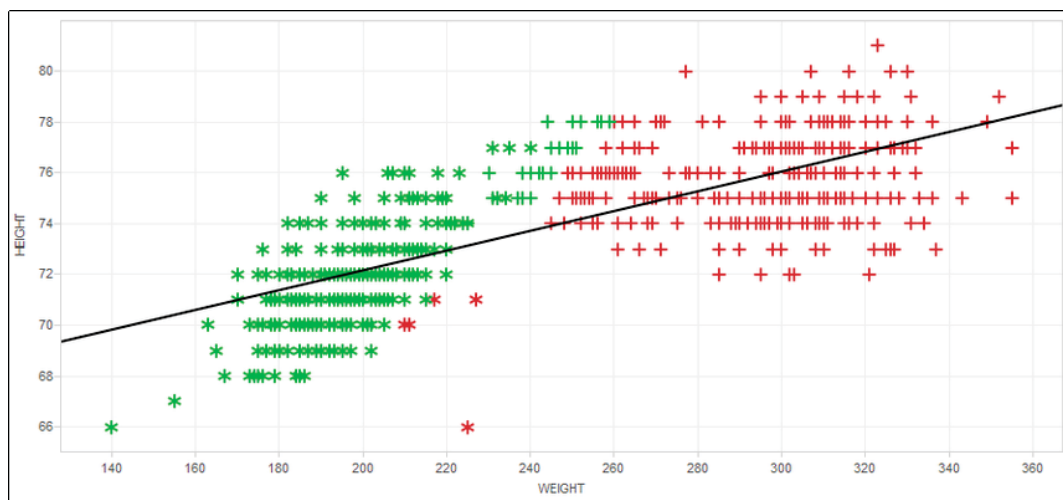


Figure 3.9: Example scatter chart

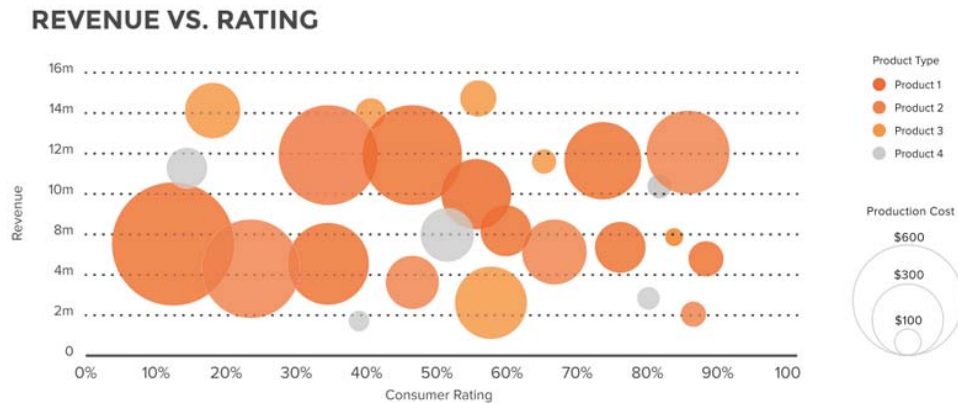


Figure 3.10: Example bubble chart

### 3.6.1 Bubble chart

Bubble charts are a type of scatter chart where a third category is added, encoded in the size of the dots (bubbles). More categories could also be added by encoding them into the color of the bubbles or their shape.

Due to the larger size of each data point, it becomes rapidly cluttered when using large datasets, allowing the representation of much fewer values than the scatter chart.

## 3.7 Map chart

Map charts represent data with geographic information. There's lots of ways of representing data on this charts, such as:

- Filling different geographic areas with color depending on a variable, such as a country's population.
- Using PoIs (points of interest) to display information about locations, such as restaurants.
- Plotting lines connecting locations, to display relationships between locations or movement, such as plane flights.

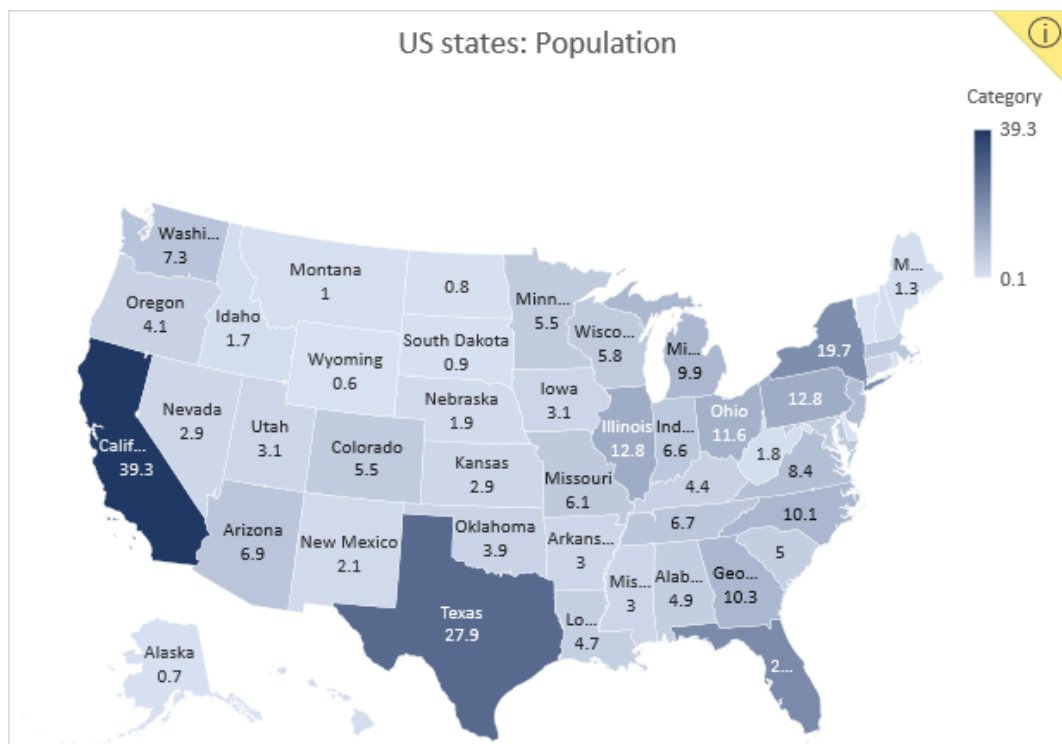


Figure 3.11: Map chart displaying US population by state

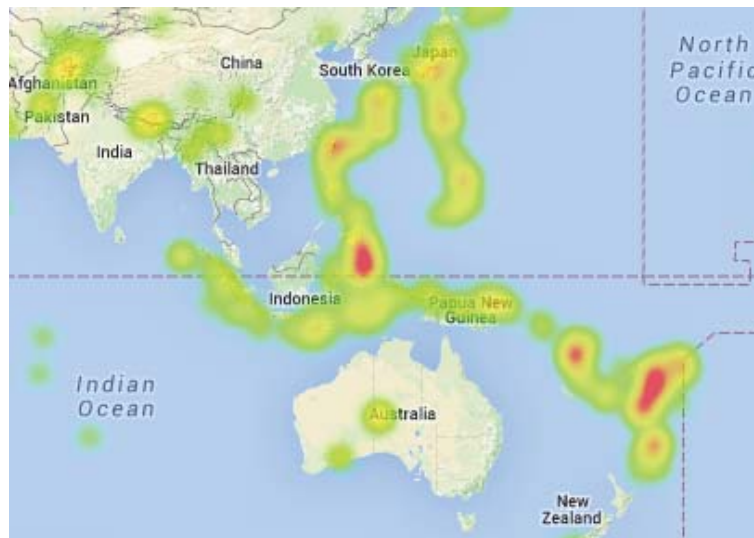


Figure 3.12: Earthquake southeast Asia and Oceania

### 3.7.1 Heatmap

A heatmap is a graphical representation of data, normally plotted on top of a map, where the values of the data are represented as colors. This kind of graphs are useful to see and locate high and low concentrations of data, based on the value of the data itself and based on the quantity of values concentrated in the same area. An example of this could be a heatmap showing the location and strength of earthquakes, which would show red on places where earthquakes are common.

## 3.8 Gauge Chart

Gauge charts show values in a gauge, normally with an upper and lower limit. Gauge charts are used to display key performance indicators in an easy and fast way, allowing the creation of dashboards to check the status of a project or company, or showing progress towards a target value.

Using colors is important since it allows users to know if the value is “good” or “bad” with a quick glance; A basic example of a gauge chart would be the velocimeter of most cars.

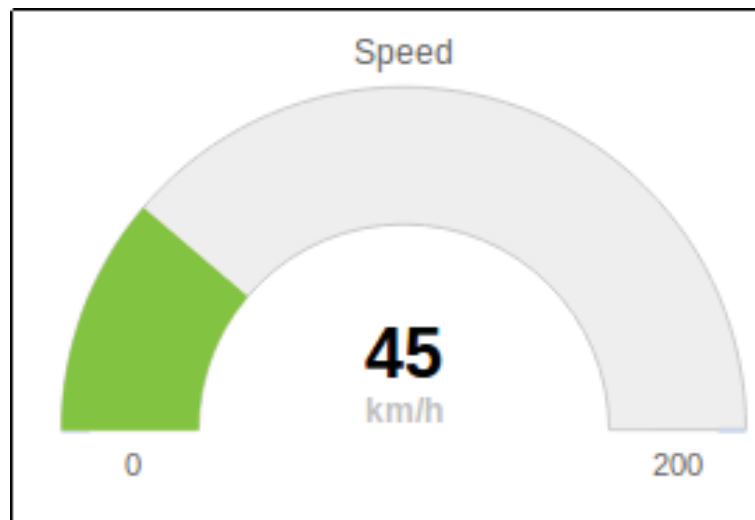


Figure 3.13: Example gauge chart showing speed

## Chapter 4

# WireCloud visualization catalogue

In WireCloud, all the logic of the workspace is performed by the WireCloud components and their connections between each other. The flow defined by these connections between data sources, through different WireCloud operators and widgets and the input and output endpoint chosen, define the final behavior of the dashboard. This allows users to generate a large amount of dashboards by combining the components used and how they interact with each other, customizing the dashboards this way so they fit the needs of the end users. This can be further extended if those components are generic, and can perform operations with any kind of data, making it possible to connect them to most WireCloud components.

In order to provide powerful visualization dashboards, there was a need to develop a set of generic WireCloud components aimed at data visualization, specifically around the scopes of this project: FIWARE DataModels harvested from CKAN and NGSI.

These components follow a MVC(Model-view-controller) pattern, since it adapts pretty well to the WireCloud architecture and it allows for easy component reutilization.

Some of these components are totally new as of the development of this project goes (for instance, the DataModel to heatmap operator and the Leaflet map viewer among others), although, some of them were developed before this project and were extended and improved while developing the project in order to adapt to the needs of the project goals and to adapt them to the newly created generic components(for instance, all the heatmap layer logic needed by the OpenLayers map widget in order to support the new heatmap layers added during the development of this project).

WireCloud visualization components can be categorized in three types:

- Data harvesters
- Data transformation and preparation

- Data representation

## 4.1 Data harvester components

These components are intended to gather the data used in the application logic by requesting it to a data provider like CKAN. These components are the first logical step in most dashboards, since they provide the data the rest of components need in order to perform their functions.

Data harvesters get data via HTTP requests upon APIs of different data sources. These requests, depending on the nature of the data might need authentication; in this case, authentication can be provided by using the WireCloud user credentials or by setting credentials through the component preferences.

Since these components do not require any kind of visual representation, they are developed as WireCloud operators. Data harvester components fulfill the model part of the MVC pattern.

### 4.1.1 CKAN source operator

The CKAN source operator is the data harvester used to gather data from a CKAN instance by performing HTTP GET requests to the DataStore extension of a CKAN instance (A CKAN regular instance does not provide a tabular gathering API, so the DataStore extension is needed in order to have such an API available).

This operator provides two output endpoints, one used to provide the complete CKAN resource, including its structure, and the other used to only send the resource data as an array, which can be useful in order to allow more generic components to use its data.

This operator has a preference used to provide a OAuth2 token used to verify the role and permissions of the user trying to access the CKAN resource. This token is only required when trying to access private datasets, and is not needed when accessing public datasets.

In order to use this operator, two preferences must be filled:

- CKAN server URL: Used to provide the operator with the URL of the CKAN server where the target data is stored.
- Resource ID: Used to provide the operator with the ID of the resource to be harvested. CKAN data is stored as datasets, each dataset having a set of resources where the actual data is stored; these resources is what the operator targets.



A row limit and sort parameters can be also provided in order to limit the number of rows or to sort the results, for example using the creation date to get the latest data first.

There is a version of this operator, called “CKAN source from URL operator” that instead of using preferences, gets those values from the parameters passed on the URL of the dashboard. The purpose of this operator is to be used on dashboards that are intended to be used with multiple CKAN resources.

### 4.1.2 NGSI source operator

The NGSI source operator is the data harvester operator used to gather data from a Context Broker instance using the NGSI format (Orion Context Broker). This data is gathered by requesting the initial data and creating a subscription to the NGSI server, which will then send real-time data updates to the operator.

This operator only provides a single data output, which contains the list of harvested entities.

The NGSI source operator requires the following preferences in order to work

- NGSI Server URL: This preference indicates the URL of the NGSI server the operator will attempt to gather data from.
- NGSI proxy URL: This preference indicates the URL of the NGSI proxy required in order to receive notifications about data updates.
- FIWARE Service: The tenant/service to use when connecting to the NGSI server.
- Use FIWARE credentials of user: This tells the operator to use the token of the current user when connecting to the Context Broker instance.
- Use FIWARE credentials of owner: This tells the operator to use the token of the user that owns the workspace when connecting to the Context Broker instance. This is useful in order to allow any user using the dashboard to retrieve the data.
- FIWARE ServicePath: The scope/path to use when connecting to the NGSI server.
- NGSI entity types: The NGSI entities to be retrieved, for example, a FIWARE DataModel entity.
- Monitored NGSI attributes: The entity attributes that will trigger a data update when modified on the server. The creation of a new entity will always

trigger an update if the entity has any of the target attributes. If this field is left blank, the NGSi source operator will not create a subscription to the context broker.

### 4.1.3 GitHub harvester operator

GitHub is a web based version control platform using Git. It extends the features of Git by adding code review, project management and integration with other tools such as continuous integration tools like Jenkins. GitHub offers free public repositories, which make it very popular for open source projects, and paid private repositories.

An issue is the unit of work to accomplish an improvement in a system. Issue can represent bugs, requested features and tasks and are used for project management. In GitHub, any user can create issues on public repositories, which are usually used in order to communicate bugs or desired features to the repository contributors. Issues can be given an assignee, which would be the person tasked to solve the issue, and a milestone, a point in the development timeline of the project where the issue should be completed.

The GitHub harvester operator is the data harvester operator used to gather data from the issues and commits of GitHub repositories. The GitHub harvester operator has two data output endpoints, one for commits and one for issues.

Authentication is only required when accessing private repositories, although there are rate limits in place for non-authenticated queries, so authentication is always recommended when using this operator.

This operator has the following preferences used to configure how it authenticates and to define the target repository:

- Repository owner: This preference indicates the owner of the target repository.
- Repository name: This preference indicates the name of the target repository.
- Max results: The maximum number of issues and commits to be harvested from GitHub.
- Oauth2 token: The Oauth token to be used when authenticating. This is the recommended authentication type.
- Username: The username to be used when authenticating using a password. This method of authentication is not recommended, and Oauth should be used when possible.
- Password: The password to be used when authenticating using an username and a password.

#### 4.1.4 GitLab harvester operator

GitLab is an open source software that allows the creation of services to host git repositories. Since a GitLab instance can be hosted on a private server, allowing this way the creation of private repositories, GitLab is very popular among private companies. Like GitHub, it extends git with issues among other things.

The GitLab harvester operator is the data harvester used to gather data from GitLab issues and commits. Alike the GitHub harvester, this operator has two data endpoints, providing issues and commits.

This operator has the following preferences used to identify the target repository and to configure how it authenticates:

- GitLab URL: This preference defines the URL of the GitLab instance from which the data will be harvested.
- Project name: The project name used to identify the target repository.
- Max results: The maximum number of issues and commits to be harvested from GitLab.
- Oauth2 token: The Oauth token to be used when authenticating.

#### 4.1.5 Jira harvester operator

Jira is an application for bug tracking and project management through the use of issues. As far as issue management is concerned, it is much more complex than GitLab and GitHub, although it does not have any version control capabilities for code management.

In Jira you can perform complex functions on the issues how to assign them a flow to control the status of the issue, forcing the issues to have to be transition between the states as defined in the flow, allowing to this way establish quality control mechanisms over the issues and the code associated with them.

The Jira harvester operator is used to harvest issues from Jira.

- Jira URL: This preference defines the URL of the target Jira instance.
- Jira project key: This preference defines the project from which data will be harvested.
- Jira component: This preference is used to define the target Jira component from a Jira project.

- Jira username: The username that will be used for password authentication against Jira
- Jira password: The password that will be used for password authentication against Jira.
- Max results: The maximum number of issues to be harvested from Jira.

#### 4.1.6 Jenkins harvester operator

Jenkins is a tool used for continuous integration, passing test sets automatically when certain trigger conditions happen (like a push on GitHub). A Jenkins build is the results associated to a test run.

The Jenkins harvester operator is used to harvest build data about the tests performed on a Jenkins instance. Jenkins is organized in jobs, each of one has a list of builds passed for said job; This operator gathers the list of builds for a job, obtaining data about the time spent passing tests for each build and data about the result of said tests.

This operator has two preferences:

- Jenkins server: This preference is used to define the URL of the Jenkins server from which data will be gathered.
- Job ID: The ID of the target job from which builds will be harvested

## 4.2 Data transformation and preparation components

These components are intended to use the data gathered by the previous components and transform it so it can be used by the data representation components, being the middle step on most WireCloud dashboards.

These components should be able to handle input from as many sources as possible to promote their reusability and make changing data sources trivial, since the dashboard logic would not need to be changed at all, only needing to change the data harvester operator.

Data transformation and preparation component have very limited functionality, encouraging users to use multiple of them in order to achieve their goals and promoting their reusability by being able to achieve the user needs by connecting multiple of them to each other through wiring.

Since most of these components do not require any kind of visual representation, the majority are developed as WireCloud operators. Data transformation and preparation components fulfill the model part of the MVC pattern.

### 4.2.1 NGSI entity to PoI operator

The NGSI entity to PoI operator is used to transform NGSI entities into Points of Interest; these PoIs can be used on map widgets like the OpenLayers map widget.

Since this operator has no information about the entities received and their structure, the PoI icon displayed will be generic, and the popup shown when interacting with the PoI on the map widget will just contain the key value pairs of the entity attributes.

This operator has two preferences: one used to define the attribute that contains the location information and the other one used to define the icon that will be used for the PoIs generated.

These operator can be used with the output of the NGSI source operator and the CKAN source operator if the later contains NGSI entities.

### 4.2.2 NGSI datamodel to PoI operator

This operator is used to transform NGSI entities that follow a FIWARE Data-Model into Points of Interest that can be used on a map widget like the Leaflet map widget.

The main advantage of this operator over the “NGSI entity to PoI operator” is that, since it has available the structure and semantics of the FIWARE DataModels, it can provide better information and data displays for the entities received as input, allowing the usage of custom icons for each of the entity types, and providing useful information on the pop-up of the PoI, instead of just showing every attribute of the entity as a key value pair. This way, only useful information is displayed, differentiating properly the different types of entities received by the operator and providing a more easy to read display that allows users to find the information they need faster.

This operator handles entities one by one, generating PoI information for each entity such as an icon, the position of the PoI, and a description that will be used as a pop-up when a user interacts with it.

This operator could be used alongside the NGSI source operator and the CKAN source operator, being the only requirement that the data follows a FIWARE Data-Model.

This operator is pretty straight-forward and does not require any kind of configuration in order to work.

### 4.2.3 NGSI datamodel to heatmap operator

This operator works alike the NGSI datamodel to PoI operator, being used to transform NGSI entities that follow a FIWARE DataModel into heatmap data that can be used to display a heatmap graphs on map widgets.

The NGSI datamodel to heatmap operator handles an array of entities, creating the configuration for a heatmap layer. The configuration of each point of the heatmap is based on the attributes of the entity and the type of the entity (this is, the DataModel the entity follows, such as AirQualityObserved). It uses the Heatmap.js library <sup>1</sup> to generate the heatmaps for the Leaflet map widget, and the heatmap layer settings when generating heatmaps for OpenLayers map widget.

This operator could be used alongside the NGSI source operator and the CKAN source operator, being the only requirement for its input that the data follows a FIWARE DataModel. Since the heatmap layer for the OpenLayers map widget and the heatmap layer for the Leaflet map widget are different, it has a different output endpoint for each of the map widgets.

A completely generic heatmap operator, on the line of the “NGSI entity to PoI operator” could be implemented, although unlike the former, this one would need more complex input from the user, defining manually parameters like the attribute used as the input value for each point of the heatmap (which defines its intensity), and values like the blur and radius for them. All of this makes it hard for users to configure and generate generic heatmaps, specially when trying to create a mashup were different types of entities are merged together.

This operator is pretty straight-forward and does not require any kind of configuration in order to work.

### 4.2.4 Value list filter operator

The Value list filter operator is used to filter input objects, sending through its output endpoint a list with the values of those objects for a single property. This target property is user defined in a WireCloud preference; this preference allows filtering nested properties using a dot notation (e.g. parentProperty.targetProperty in order to get the targetProperty which is nested inside the parent property).

This operator can be useful when trying to perform operations with the values of a single variable, since most data transformation operators that work with values expect a list of values, instead of a list of entities containing multiple properties which should be ignored since they are not the target one.

The Value filter operator works just like the “Value list filter operator”, but instead of filtering lists, it filters single entities. Although the “Value list filter”

---

<sup>1</sup>Heatmap.js <http://www.heatmap.js/>

operator is also able to filter single entities, its output is always a list, making this operator useful when a single value is needed as an output. The “Value filter operator” also defines the target property using a WireCloud preference.

### 4.2.5 Calculate tendency operator

This operator is used to calculate different trends from a value list. It provides one output for each kind of calculation it can perform:

- Minimum: The minimum value of the list.
- Maximum: The maximum value of the list.
- Arithmetic mean: The arithmetic mean of the list.
- Median: The median of the list.
- Mode: The mode of the list.
- Standard deviation: The standard deviation of the list.
- Count: The number of elements of the list.
- Sum: The sum of all the elements of the list.

This operator has no preferences and accepts as input any list of numeric values.

The Calculate tendency operator can be used alongside the Value List Filter operator so that it prepares the input required by this operator, extracting it from a list of objects, generating the list of values needed for the Calculate tendency operator in order to work.

### 4.2.6 Count list operator

This operator is used to calculate the count of a list of objects. It provides two kinds of counts: A simple length of the list, which returns how many items the list has, and a count of unique occurrences of each item of the list, which would return each unique item of the list and the amount of times it is contained in the list. This last endpoint could be used, for instance, when trying to build a pie chart displaying the composition of a variable, since this endpoint would provide the required values to build such chart.

This operator has no preferences and accepts as input any list of numeric values.

Input type	Single value	List of values
Single value	Normal operation	Operate single value with each value of the list
List of values	Operate single value with each value of the list	Operate each value of the list with the same value of the other list

Table 4.1: Basic list arithmetic operator behavior

### 4.2.7 Basic list arithmetic operator

The “Basic list arithmetic operator” is used to perform arithmetic operations between two inputs. Its behavior depends on the type of inputs received on each of its two input endpoints, that can both receive either a single value or a list of values. See table 4.1.

This operator can perform four kinds of arithmetic operations, providing an output endpoint for each of them:

- Addition: Add both inputs.
- Subtraction: Result is the first input minus the second input.
- Multiplication: Multiply both inputs
- Division: Result is the first input divided by the second input

This operator has no preferences and accepts as input any list of numeric values.

### 4.2.8 Union list and intersect list operators

The Union List operator is used to perform a union operation between two lists, returning a list that contains a single instance of each object of both input lists. This can be useful, for example, when trying to merge data from multiple sources, that therefore would use multiple data harvester operators; by using this operator those outputs can be merged into a single one that can be passed to other WireCloud components.

The Intersect list operator works alike the Union List operator, but instead of performing the union of two lists it calculates the intersection. This, although less applicable to real use cases, can prove to be useful when trying to only display data that two data sources have in common, for instance.

Both of this operators have no preferences and accept as input any list of values.



### 4.2.9 Packlist operator

The Packlist operator is used to merge multiple inputs into a single list of data. If multiple list are received in creates a list of lists, while if it receives two items it creates a list containing both items, and if it receives a list and an item, it appends the item to the list.

This operator is useful, for instance, when trying to plot multiple line charts; since the chart generator operators have a single input endpoint for their input dataserie, this operator can be used to create a list of dataseries which can be passed to this operators in order to plot multiple dataseries. This way, multiple dataseries can be merged together into a list of them and plot a line chart for each of the input dataseries.

### 4.2.10 Labels to dataserie operator

This operator generates a dataserie from a serie of labels, which allows other components that expect a list of values to work with lists of categorical values.

This is done by counting the occurrences of each value of the label list, generating this way a numeric value for each label.

This operator has two outputs, one being the dataserie, and the other a list of labels corresponding to each value of the dataserie. This output is useful to keep track of what each value of the dataserie output means.

This operator has no preferences and accepts as input any list of numeric values.

### 4.2.11 Chart generator operators

Chart generators are the operators intended to generate the configuration required by the Highcharts widget in order to work. Each chart generator generates the configuration required to display their type of chart. These work by processing the input dataserie received through the WireCloud endpoint and generating the Highcharts configuration for the target chart. They should always be connected to an instance of the Highcharts widget.

Although they all share the same output endpoint used to connect to the Highcharts widget, they have different input endpoints depending on the chart they generate.

These operators provide a callback function that allows users to retrieve the original data associated with the part of the chart the user is interacting with.

Currently, there are these chart generators:

- Column chart generator: This operator generates column charts from a dataserie received as input. It has two endpoints, one for the input data, and the other

for the labels associated to each of the columns generated, the latter being optional.

- **Line chart generator:** This operator generates line charts from a dataset received as input. It has two endpoints, one for the input data, which supports multiple datasets packed in an array as one (for instance by using the Packlist operator), and the other endpoint being for the timestamps used for the X axis of the chart, this one being optional.
- **Pie chart generator:** This operator generates pie charts from the input data received. It has two working modes, depending on the input endpoint used: it can either use an object with a property for each key and value, or a list of tags, which are counted to get the repetitions for each one and then build the pie chart with that data.
- **Scatter chart generator:** This operator generates a scatter chart using the input data received through both its input endpoints, one being for the X axis and the other being for the Y axis.
- **Gauge chart generator:** This operator displays a gauge with the value received through its input endpoint. By configuring the operator preferences, the minimum and maximum value used for the chart can be configured, as well as the units to be displayed. This maximum and minimum values can also be set through their respective input endpoints, which would override the value configured using the preferences of the operator.

#### 4.2.12 Data filter components

Data filters are the controller part of the MVC pattern, being the only place user input is used on data visualizations. Using data filters users can choose a set of conditions and limit the data that goes through the flow of the dashboard's wiring, modifying this way the behavior of the dashboard and the final data displayed.

The main way to filter data is using the “AND filter operator” alongside a widget that provides filter conditions for it, such as the “Set generic filter conditions widget”. See image 4.1 for an example of how this filter is done on wiring.

There are, however, other ways of filtering data using data representation components, for instance map widgets, which have an output endpoint that sends only the data associated to the POIs that are currently on the map boundaries, or the table viewer widget, which has an output that sends through wiring the selected rows.

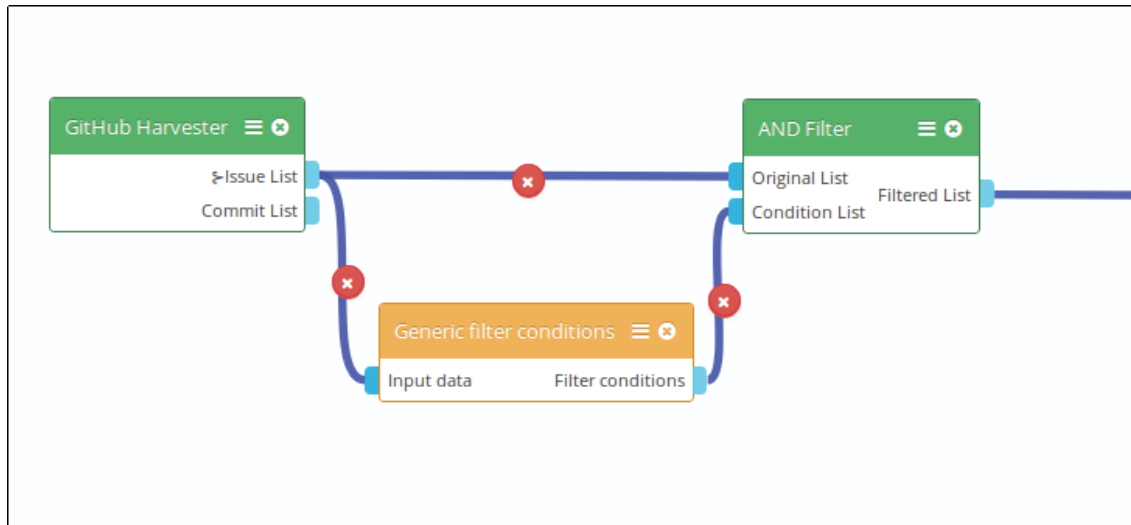


Figure 4.1: Basic filter wiring structure

The “AND filter operator” can apply a list of filters to a dataset, performing AND operations between the filters, so the resulting data passes all filters. These operator supports multiple types of filters

- Equal: This filter compares the input value with a set value, and only passes if they are equal.
- Not: This filter compares the input value with a set value, and only passes if they are different.
- In: This filter compares the input value with a list of values, and passes the filter if its equal to any of the values provided in the list.
- Range: This filter checks if the input value is in the range of values provided to the filter.
- Some: This filter checks if any of the input values is equal to a set value, failing to pass the filter otherwise.

This operator has two input endpoints, one used to get the dataset to be filtered, and the other used to receive the filter conditions to apply to the data. After filtering the data, it is send through the output endpoint of the operator.

The “Set generic filter conditions widget” uses the metadata associated to the dataset received to generate a set of filters, that the user then uses to select the filtering conditions they want to apply to the data. This widget does require that

**Generic filter conditions**

**Sprints**  

All

**Assignee**  

Mognom

**Status**  

closed

**Label**  

All

All

User Story

Bug

Confirmed

v1.0.1

WorkItem

Blocker

v1.2.0

v1.1.0

Figure 4.2: Filter view

the input data has the required metadata in order to generate its filters. This metadata should be added by the harvester operator that gathered the data in first place. Figure 4.2 has an instance of this widget running.

The output of this widget is intended to be connected to the “AND filter operator”.

## 4.3 Data representation components

Data representation components are responsible of displaying data to the end user, therefore, all of them are widgets, since those are the only WireCloud components displayed on the dashboard view.

These components are the View part of the MVC model. In some cases, some of these widgets do also act as the controller part of the MVC, by sending output data based on how the user interacted with their display, for instance, sending the entity the user selected on a chart.

### 4.3.1 Highcharts widget

The Highcharts widget is used to generate and display Highcharts charts from the input configuration received. This widget allows the user to select which variables it wants displayed on the chart interactively, being able to remove variables from the chart on real-time, allowing the users to filter the data they want to see on the charts.

The input data received by this widget must contain a valid Highcharts configuration, defining one of the chart types that are supported by Highcharts. The easiest way to generate such configuration is using the chart generator operators defined in section 4.2.11.

This widget has an output endpoint used to send through wiring the entities associated with the part of the chart selected by the user.

### 4.3.2 Panel widget

This widget is used to display data to the user in a generic way. It can display any kind of text or number, alongside its units, and can be used to display the value of a single variable easily. Therefore, this widget is the most simple data display available.

The user cannot make any interactions with this widget as it only displays information and it does not send any kind of output data.

### 4.3.3 Data table viewer

This widget handles the creation and display of tables in WireCloud using the platform style, providing this way an interface to use the WireCloud table API easily.

The widget generates a table with the data received, where users can select filters for each column, removing this way unwanted entries from the table, as well as sorting the data in ascending or descending order by any column. Users can also select multiple rows, which will then be sent through wiring using an output endpoint so that it can be used by other WireCloud components.

### 4.3.4 Leaflet map viewer

The leaflet map viewer widget provides the ability to display map data by using the Leaflet library. It supports both PoI icons, which display a popup with information when clicked, and a heatmap layer, which can be set to adapt its size and blur according to the map zoom.

This widget has the following input endpoints:

- **PoI Input:** This input is used to add new PoIs to the map, while keeping all the previous ones.
- **Replace PoIs:** This input is used to delete all the previous PoIs before adding new ones.
- **Delete PoI:** This endpoint is used to delete the PoI received if it is already added to the map.
- **Heatmap:** This endpoint is used to add a heatmap layer to the leaflet map viewer.

This widget has the following output endpoints:

- **PoI Output:** This endpoint provides the PoI selected by the user.
- **PoI List Output:** This endpoint sends a list with all the PoIs that are currently visible in the map segment displayed.

The main reason to use this map viewer over the “OpenLayers 3 map viewer widget” is that it works better with the heatmap layers generated by WireCloud components like the “NGSI Datamodel to heatmap operator”, allowing the heatmap to change its settings depending on the level of zoom applied to the map.

### 4.3.5 OpenLayers 3 map viewer

The OpenLayers 3 map viewer widget provides the ability to display map data by using the OpenLayers 3 library. This map widget supports PoIs, displaying an icon for them as well as a popup with information when the user interacts with them, as well as supporting most of the OpenLayers3 layers, such as heatmap layers.

This widget has the following input endpoints:

- **PoI Input:** This input is used to add new PoIs to the map, while keeping all the previous ones.
- **PoI Input Center:** This endpoint is used to add a PoI to the map and center the map view on the newly added PoI.
- **Replace PoIs:** This input is used to delete all the previous PoIs before adding new ones.
- **Delete PoI:** This endpoint is used to delete the PoI received if it is already added to the map.
- **Layer Info:** This input is used to add a OpenLayers3 layer to the map. The input must have the settings of the layer to add.

This widget has the following output endpoints:

- **PoI Output:** This endpoint provides the PoI selected by the user.
- **PoI List Output:** This endpoint sends a list with all the PoIs that are currently visible in the map segment displayed.

The main reason to use this map viewer over the “Leaflet map viewer widget” is that it supports more layer types instead of just the heatmap layer (although leaflet heatmap layers work better, since they can adapt their size to the zoom level, which can be useful in some scenarios), and that it has more settings and input endpoints, allowing for more complex dashboards to be developed.

### 4.3.6 History player widget

The history player widget is used to handle datasets that contain data over time, by displaying a line chart with the average values of the attributes of all entities in each time frame. The attributes displayed are based on the FIWARE DataModel the chart follows.

Users can interact with the chart in order to select it and send its data through wiring. This widget also has an auto-play feature, used to send the data of each

time frame step by step through its output endpoint, with a little delay between each of these steps, so that the data displays of the rest of WireCloud components of the dashboard are updated and the user can see the graphs update on real time as the time frame selected on the widget moves forward.

Currently, this widget only works if the input data follows a FIWARE Data-Model, since it is needed in order to select the attributes to be displayed. However, the functionality of this widget can be replicated by using other WireCloud components; for instance, by using the “Value list filter operator” in order to select the desired attributes, and the “Packlist operator” to create a list holding them all together in order to get the input data the “Column chart generator operator” needs in order to generate the same chart the “History player widget” generates.



## Chapter 5

# WireCloud integration with CKAN

Integrating CKAN with WireCloud would allow CKAN users to display powerful views for their datasets that can be easily customized to fit the needs of the end user by using WireCloud dashboards.

This is not supported by default on CKAN, which has poor built-in data visualization capabilities, and requires the use of extensions to have it at all.

In order to harvest data from CKAN, the CKAN instance must to have the DataStore extension, which provides an ad-hoc database for storage of structured data from CKAN resources. Data can be pulled out of resource files and stored in the DataStore. The DataPusher extension eases this process by doing it automatically on dataset creation.

The DataStore extension provides endpoints that can be used to query its database and retrieve the data stored on CKAN datasets. This queries are used by CKAN view extension to get the data needed to create the view.

In order to generate views on CKAN using WireCloud, a new extension is needed. This extension needs to be able to embed existing WireCloud dashboards. Also, WireCloud needs to be able to retrieve data from the DataStore extension so it can generate the views; this data harvesting functionality is already implemented by the CKAN source operator, which retrieves data from the DataStore.

Since creating new visualization dashboards might prove to be a difficult task for new users or users with little to no programming knowledge, users need to be able to create this dashboards automatically. In order to do this, the extension should provide tools to ease the process of dashboard creation.

Although it would be ideal to configure and create those new visualization dashboards entirely from the CKAN extension, the limitations the CKAN extension API has make this not viable. Therefore, the developed solution makes uses of an extra

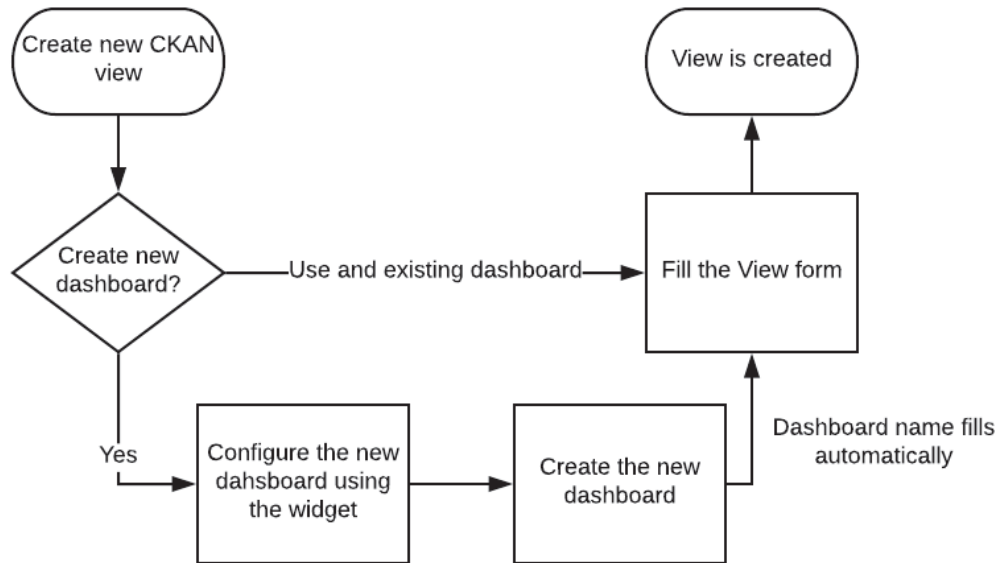


Figure 5.1: CKAN view creation workflow

WireCloud dashboard designed to help users create new visualization dashboards. This dashboard is embedded on the CKAN view creation so users can do it without leaving the CKAN site. The main downside to this approach is that most of the dashboard creation logic ends on the client side, where the computational power available might be scarce. On the other side, this allows users to modify the dashboard creation dashboard directly from WireCloud, making it easier to customize it to adapt to the needs of the end users.

The view creation workflow when creating a new dataset would be:

- 1- Add a new dataset to CKAN.
- 2- Add the new dataset to the DataStore(Automatic if using DataPusher)
- 3- Create a new view using the WireCloud view extension.
- 4- Create a new dashboard using the tools provided by the extension. Skip if using an existing dashboard.
- 5- Fill the view form.
- 6- View is created and configured.

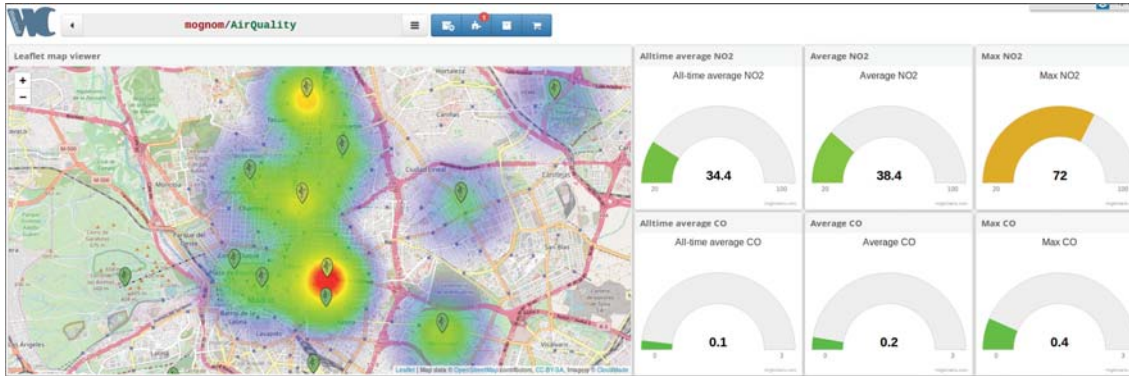


Figure 5.2: AirQuality example dashboard

## 5.1 Reference dashboards

In order to prove the capabilities of WireCloud’s data visualization, a set of example dashboards were developed for the main FIWARE DataModels. These dashboards make use of the widgets and operators developed for the WireCloud visualization catalogue. See figures 5.2, 5.3 and 5.4 for examples.

These dashboards aim at exploring the particularities of the different DataModels, while showing the flexibility provided by the WireCloud visualization catalogue, allowing the creation of very different dashboards using the same set of WireCloud components

This dashboards can also be used as example views for a DataModel, helping users get a basic understanding of WireCloud capabilities and how it works, while also inspiring them on what the dashboards they want to achieve could look like, allowing the users to customize them afterwards in order to fit their end user needs.

This reference dashboards wiring can become rapidly complex when adding more and more data views to them, due to the high quantity of WireCloud components (specially operators) used on said dashboards, to the point where they can become hard to create for regular users. As seen in image 5.5, when adding multiple charts to a single dashboard the wiring starts to become quite large and can get confusing for beginner WireCloud users.

While this complexity is the main downside of using generic components instead of specific components developed ad-hoc for each of the FIWARE DataModels, this approach allows users that know how to create and configure visualization dashboards for a DataModel to create them for any other DataModel, as well as helping the desired dashboard creation tool to work with any kind of input dataset. Also, a less generic approach would end in the need of developing new components as new datasets appear, which would make it harder to maintain the WireCloud visualiza-

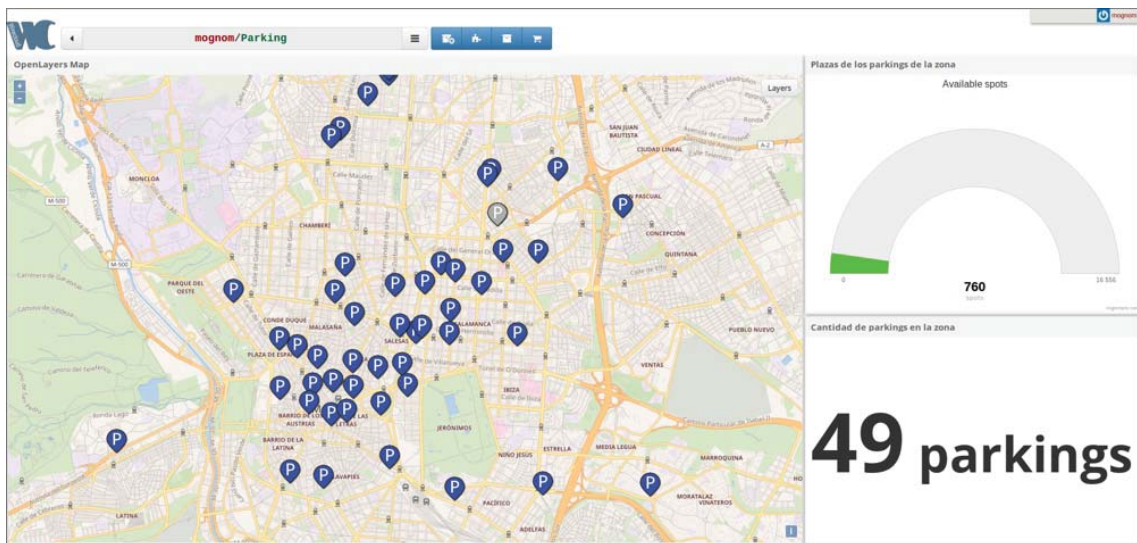


Figure 5.3: Parking example dashboard

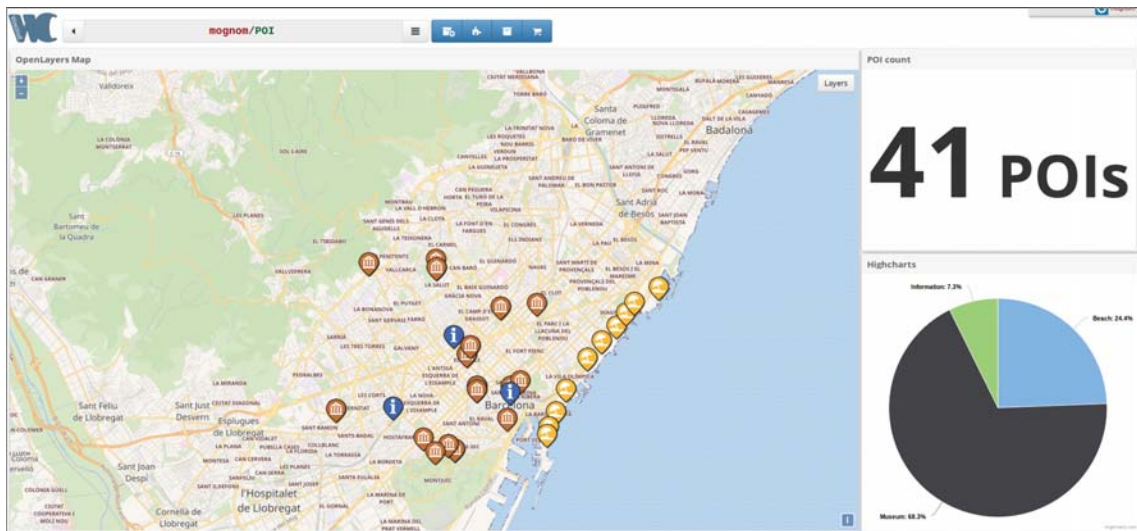


Figure 5.4: PoI example dashboard

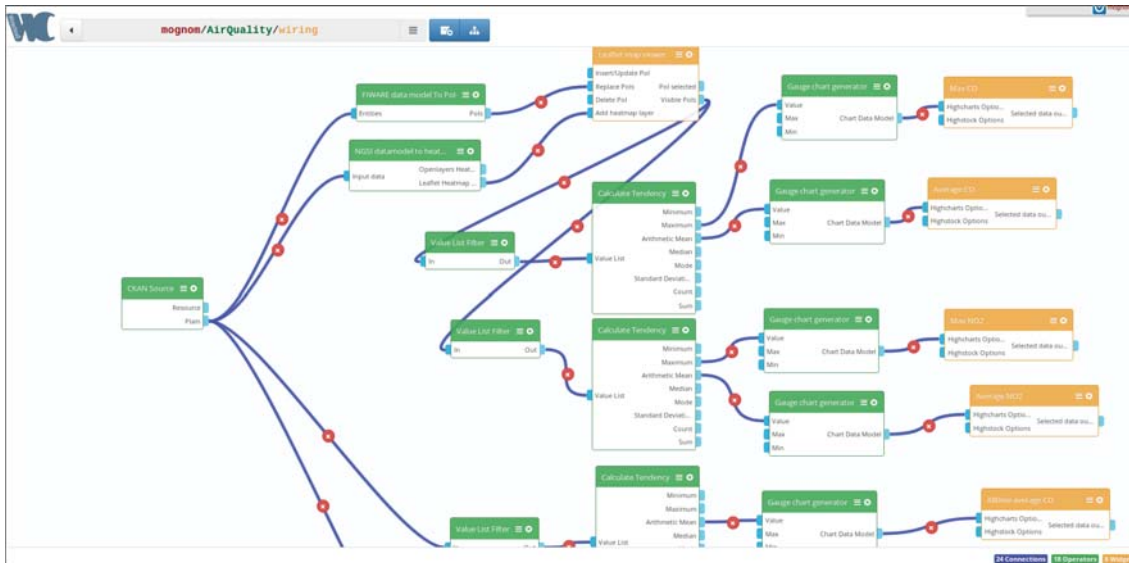


Figure 5.5: AirQuality example dashboard wiring

tion catalogue and the dashboard creation tool.

In order to help users create this dashboards, a dashboard creation tool must be developed, so users can use a simpler interface when creating the dashboards without losing too much of the dashboard creation freedom that WireCloud offers to its users. After generating the dashboard with the dashboard creation tool, users would be able to modify and customize it to their liking, even adding new WireCloud components in order to extend the dashboard capabilities.

## 5.2 WireCloud CKAN extension

The existing “Ckanext WireCloud view” extension<sup>1</sup> provides a way for CKAN users to visualize datasets using an embedded WireCloud dashboard. When creating a view, users can either choose an already existing WireCloud dashboard or create a new one by using a dashboard setup with that intent.

To create a view, a view form has to be filled, indicating the name of the view and an optional description. This is common to all CKAN views. In the case of WireCloud, an extra field for the dashboard name is added.

When creating a new dashboard to use for the dataset view, the extension waits until the dashboard creator sends a notification with the name of the new dashboard, which gets automatically filled into the view creation form.

<sup>1</sup>WireCloud view extension [https://github.com/conwetlab/ckanext-wirecloud\\_view](https://github.com/conwetlab/ckanext-wirecloud_view)



When configuring and installing the WireCloud CKAN extension, the URL of the WireCloud instance to be used must be filled. Also, the dashboard used when to help users create new view dashboards can be configured there.

This extension makes use of HTML Iframes to embed the WireCloud dashboard, while allowing all its functionality to work properly.

In order to create a new dashboard on WireCloud the user needs to be authenticated on the WireCloud instance; Users also need to be authenticated when trying to access from a WireCloud a CKAN private dataset, where the user needs to have the required permissions in order to retrieve the data successfully. Therefore, in order for this extension to work, it requires the CKAN OAuth2 extension to support user authentication and authorization using OAuth2. Both the CKAN instance and the WireCloud instance would need to share the same identity manager in order to authenticate the same users. An example of this would could be the FIWARE identity manager, which is already supported by both CKAN (with the CKAN OAuth2 extension) and by WireCloud.

## 5.3 CKAN dashboard creator

Previously, new dashboard created from CKAN used the “CKAN graph creator widget”<sup>2</sup>, which allowed users to create a new chart for a CKAN dataset. Using this widget, users could choose the type of chart and the variables from the data used for said chart. When the new chart is configured, user set the name of the new dashboard and the widget handles its creation, which is done using a widget that, through it’s preferences, generated the input data required by the chart widget.

This however, has a very limited functionality, only allowing its users to create a single chart for their visualization, and was also limited by how the widget did not modify the new dashboard at all, and configuring it on creation by using a mashup template with a parameters in order to tell the new mashup which CKAN resource to connect to and which one is the chart selected.

In order to generate the desired data visualization, a new widget able to create dynamic dashboards, with any amount of charts on them is needed. This requires the widget to be able to, instead of using a mashup as a template and not modifying it, create a dashboard from scratch and then add to it the WireCloud components and connections needed in order to get the desired final result.

Therefore, this dashboard creation is handled by a brand new widget, the “CKAN dashboard creator widget”<sup>3</sup>, which is an improved version of the “CKAN graph cre-

---

<sup>2</sup>CKAN graph creator widget <https://github.com/wirecloud-fiware/ckan-graph-creator-widget>

<sup>3</sup>CKAN dashboard creator widget <https://github.com/Mognom/>

ator widget”, that is able to create multiple charts for the visualization dashboard and have more freedom in the created dashboard customization by doing it dynamically, thanks to, after creating a base dashboard, making REST requests to the WireCloud API in order to add the needed widgets and operators, and create the connections between them as needed.

In order to use this new widget from CKAN for its dashboard creation, it uses the “CKAN source from URL” operator, that takes parameters from the URL instead of using the operator preferences, allowing its use without changing any preference, which is useful when using the dashboard from CKAN, by passing the CKAN resource parameters on the URL.

This operator then sends a part of the target CKAN dataset to the “CKAN dashboard creator widget”, which it uses to analyze the data contents and structure. Using this information, it can then help the user create the dashboard by providing recommendations on what options and variables to select.

Using the “CKAN dashboard creator widget”, users are able to create “components”, being a “component” a set of WireCloud operators and widgets that produce as a result a single data visualization, like a chart or a data display. The final dashboard created by this widget would, therefore, be a combination multiple of these “components”. Each of these components is responsible for the creation of a single data view or representation, such as a chart or a data display in the “Panel widget”. Since this components are completely modular, any amount of them can be added to the final dashboard, although it is not recommended to add many of them, since the dashboard can become cluttered or too large, requiring too much time and scrolling in order to find the desired data. When creating and adding this components to the final dashboard, the widget handles the selection and creation of the needed WireCloud components that provide the functionality required by the component, this way, the user doesn’t have to provide any input on which WireCloud components to use, how to configure their preferences, not how to connect them with each other in order to get the final result, being all this done automatically by the “CKAN dashboard creator widget”. This makes it quite easy and fast to create complex dashboards using the WireCloud visualization catalogue, and thanks to it being based on this modular components, if a new “component” is needed in order to generate a new visualization type, it can be easily added to the widget by adding the options needed by the new component.

When configuring each component, the information gathered through the data input endpoint comes into play, giving users recommendations in the form of visual clues about which variables are recommended and available with each kind of chart and which kind of displays to use. In this recommendation is where the analysis of

the different types of charts and when to use them comes into play, helping users generate visualizations that help users understand their data without leading them to a miss interpretation and filtering out data that can't be used with the desired visualization due to its type being incompatible. Unfortunately, the recommendations the widget can provide are limited, since it doesn't have any information of the data received and how to use it, it can't know which of the available variables are interesting for the user or have relations with other variables. Therefore, this recommendation works best when using data that is compliant to a FIWARE Data-Model, so that the widget can use the semantics provided by the DataModels and know which variables are most likely relevant to the final user, and which have relations with other variables, such as correlation, providing this way more accurate and useful recommendations that remove the designing weight from the user. Thanks to this recommendations, users can know which variables are most likely interesting to show with the components, while also filtering those variables that can not be used with selected component as they are not compatible at all.

After having the user finish configuring the new dashboard, the widget created it by using the REST API provided by WireCloud. Although WireCloud components have an API that makes uses of the REST API aimed at dashboard management and creation, this API is very limited, providing only support for the creation of the workspace and the creation of widgets in the current dashboard, being missing the creation widgets and operators on other workspaces, modifying components preferences and creating connections. Therefore, the REST API is used by this component as it provides all the functionality required by this widget.

In order to generate the new dashboard, the widget first create a dashboard using a basic template that has the basic WireCloud components needed and retrieves the new workspace and tab IDs, required for the next of the API calls. Then, it proceeds to create each "component", keeping track of the IDs assigned by WireCloud to the widgets and operators so it can then establish the connections. The IDs must be kept and used to build the URIs for the request since WireCloud's REST API does not return the URI for the newly created objects, leaving the duty of building it to its client applications.

If the WireCloud instance is using the SQLite engine as the database engine, database gets locked while making transactions, which results in errors if the database is used while its locked. In order to avoid this and allow the widget to work on instances using SQLite, all REST queries are done sequentially, waiting for the previous one to finish before starting the next query. Although this does increase the time it needs to create the whole dashboard, it is not noticeable for the end user, since there was a need anyway to wait for some queries, in order to get the IDs of the new components, which are required when creating connections between them.

Although it can be used from WireCloud itself for any kind of data source, the



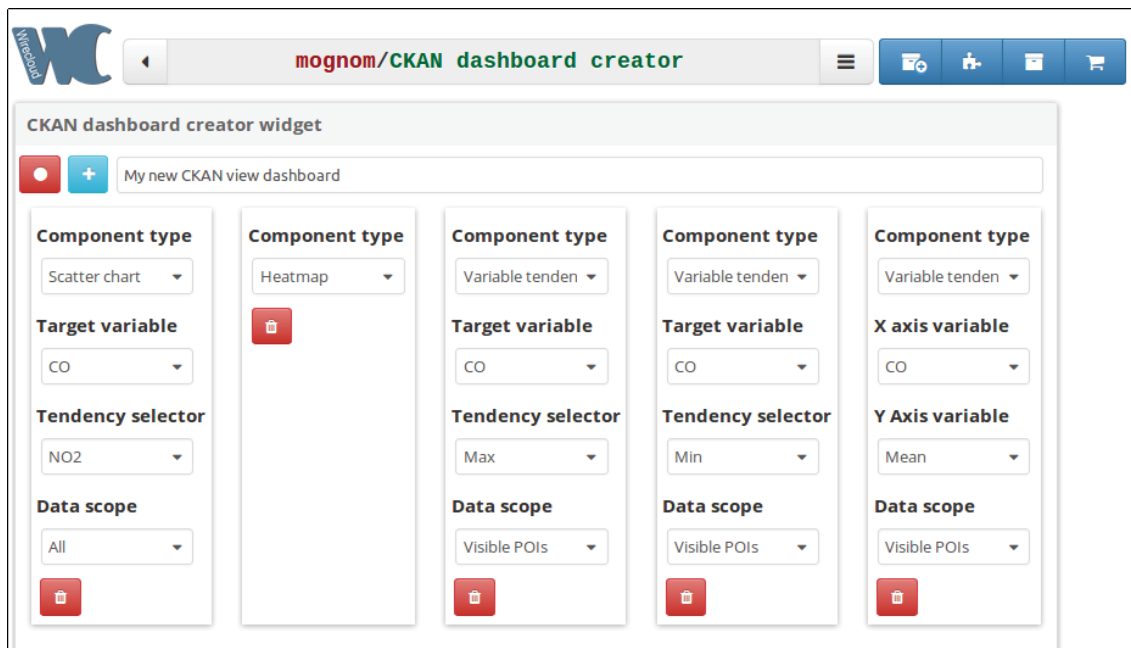


Figure 5.6: CKAN dashboard creator dashboard view

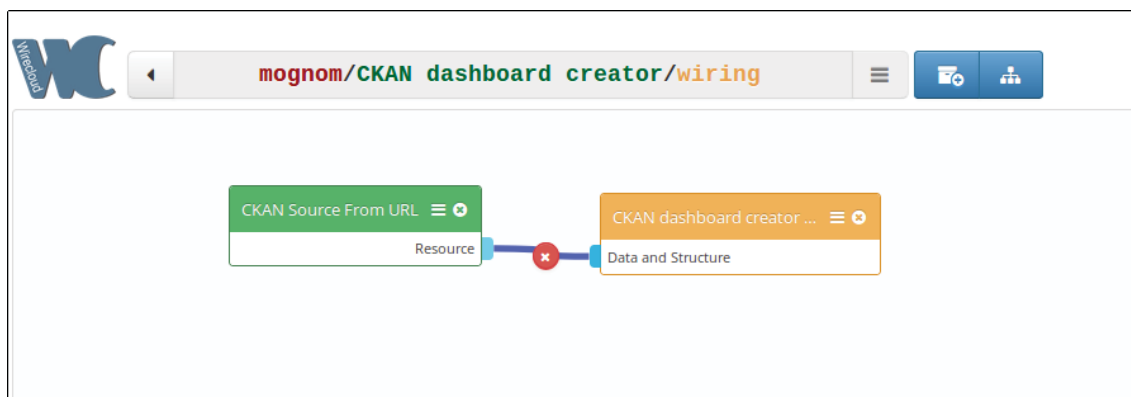


Figure 5.7: CKAN dashboard creator wiring view

main goal of this widget is to be used from CKAN, by embedding a dashboard with both the “CKAN dashboard creator widget” and the “CKAN source from URL” in order to allow the creation of data visualization for the CKAN datasets without needing to configure the source on WireCloud. With this embed dashboard, users make use of the widget to create and configure a new visualization dashboard, that after being created is captured and filled onto the view creation form on CKAN. See figure 5.8 to see an example of the WireCloud CKAN extension using the “CKAN dashboard creator widget” when creating a new dashboard.

## 5.4 WireCloud REST API for dashboard management

In this section I will explain and detail the API calls used by the “CKAN dashboard creator widget” when creating new visualization dashboards. These widget only needs to make use of 3 API endpoints: one for creating workspaces, another one for creating widgets, and an endpoint for handling the wiring of the dashboards, which includes creating operators, modifying operator preferences and properties, and creating connections between WireCloud components.

- `/api/workspaces`

By doing POST requests on this endpoint, new workspaces can be created. In this case, it will be used together with a mashup template. It returns the object of the new workspace created, which is used to get the ID of the workspace and the ID of the workspace tab. See example POST body 2


```
{
  "allow_renaming": true,
  "name": "Example name",
  "mashup": "CoNWeT/baseCKANmashup/0.1.0",
  "preferences": {
    "preference": 5,
    "example_preference": true
  }
}
```



Listing 2: Create workspace POST body

🏠 / Organizations / CoNWeT / AirQuality / Air Lite / Edit / **Add view**

**What's a view?**

A view is a representation of the data held against a resource

 Add view

 All views  View view

**\* Title:**



**Description:**   
You can use [Markdown](#) formatting here

**Filters:** [Add Filter](#)

**Dashboard:**

Create a new dashboard

**CKAN dashboard creator widget**




<b>Component type</b> <input type="text" value="Scatter chart"/>	<b>Component type</b> <input type="text" value="Variable tenden"/>	<b>Component type</b> <input type="text" value="Variable tenden"/>
<b>Target variable</b> <input type="text" value="CO"/>	<b>Target variable</b> <input type="text" value="NO"/>	<b>Target variable</b> <input type="text" value="NOx"/>
<b>Tendency selector</b> <input type="text" value="SO2"/>	<b>Tendency selector</b> <input type="text" value="Max"/>	<b>Tendency selector</b> <input type="text" value="Count"/>
<b>Data scope</b> <input type="text" value="Visible POIs"/>	<b>Data scope</b> <input type="text" value="Visible POIs"/>	<b>Data scope</b> <input type="text" value="All"/>
		

Figure 5.8: CKAN integration with WireCloud

- **/api/workspace/{workspace\_ID}/tab/{tab\_ID}/iwidgets**

By doing POST request on this endpoint, new widgets can be created. When creating a new widget, its preferences can be set. Also, its position on the workspace view can be set when creating the widget. The widget object is returned when as response to the POST request. See example POST body 3

```
{
  "widget": "vendor/MyWidgetName/version",
  "height": 180,
  "width": 180,
  "commit": true,
  "left": 2,
  "top": 4,
  "layout": 0
}
```

Listing 3: Create widget POST body example

- **/api/workspace/{workspace\_ID}/wiring**

This endpoint handles updates on wiring. It supports PUT requests that overwrite the whole wiring configuration and PATCH requests that only update a piece of it. Since WireCloud stores all wiring settings of a dashboard as a single JSON, both connections and operators share the same endpoint. By using the PATCH endpoint, new operators and connections can be created easily from the REST API without keeping track of all the wiring status, which would be needed if using the PUT endpoint. See example PATCH body 4 for operator creation and 5 for connection creation examples.

Using this API calls, the “CKAN dashboard creator widget” creates a template dashboard that only has the “CKAN source operator”, and then proceeds to create, for each component the WireCloud components it needs with their preferences, connecting them afterwards in order to get the desired result. For instance, when creating a tendency component with a panel showing the maximum of a variable, it creates the “Value list filter operator”, used to get the values of the target variable, the “Calculate tendency operator”, used for getting the maximum of said variable, and the “Panel widget”, used to display the value, by using

```
{
  "op": "add",
  "path": "/operators/1",
  "value": {
    "name": "vendorName/MyOperatorName/version",
    "preferences": {
      "preferenceName1": "value",
      "preference2": 5
    },
    "properties": {
      "property1": true
    }
  }
}
```

Listing 4: Create operator PATCH body

calls to `/api/workspace/workspace_ID/wiring` to create both operators and to `/api/workspace/{workspace_ID}/tab/{tab_ID}/iwidgets` to create the widget; then, it connects the value filter operator to the data input, the tendency input to the value list operator, and the max output of it to the panel widget, using a call to `/api/workspace/workspace_ID/wiring` for each of these connections. Therefore, creating and adding this component to the dashboard requires six API calls, three for creating the WireCloud components required, and three to create the connections between them. See figure 5.9 for the resulting wiring of these calls. Note that the CKAN source operator was created before adding these component, as it is added by the widget when creating the base dashboard.

```
{
  "op": "add",
  "path": "/connections/1",
  "value": {
    "source": {
      "id": 1,
      "type": "operator",
      "endpoint": "outputEndpointName"
    }
    "target": {
      "id": 2,
      "type": "operator",
      "endpoint": "inputEndpointName"
    }
  }
}
```

Listing 5: Create connection PATCH body

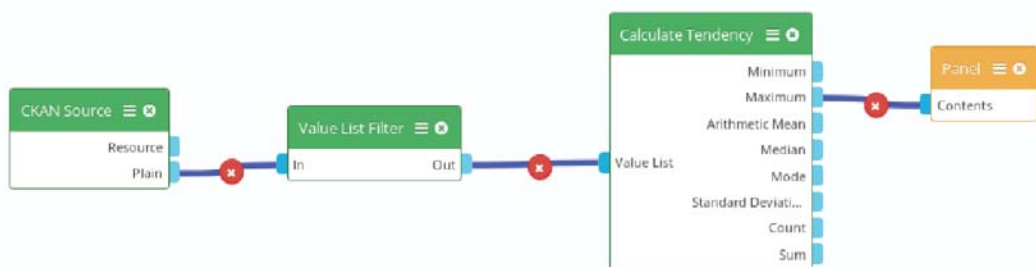


Figure 5.9: Basic component wiring

## Chapter 6

# WireCloud integration with APInf

APInf platform offers a comprehensive, yet easy to use API management tool.

It provides simplified workflow for API owners for API management tasks and also serves as a self-service tool for API consumers for discovering APIs.[1]

In order to use the utilities provided by APInf, users must first add their API to the catalogue of the APInf instance. By doing this the API is now published and API consumers can go check its details on the website, giving better visualization to the API. A proxy, like API Umbrella, can be set for the APIs; by setting this proxy allows using all the APInf features, like authorization, logging, and traffic management. After being connected to a proxy, now different API KPI can be seen on the API management dashboard provided by APInf.

APInf builds on API Umbrella, an open source API management proxy intended to be used in front of API calls, redirecting the calls to their respective APIs so that it can provide functionality like API keys, analytics by keeping track of the calls made, rate limiting and caching. API Umbrella can also be used to provide a single entry point for multiple APIs, routing the request to the proper servers. [2]

The FIWARE API Management Framework incorporates significant contributions from the APInf platform, since this allows to easily monetize datasets published in NGSI, by managing and monetizing the API calls done to this platform through APInf. Also, since all access to NGSI is done through API calls, APInf can be used as a catalogue for the NGSI data available to the users, serving both as a catalogue and to gather data for API call statistics.

In order to extend the data visualization capabilities of the WireCloud platform, and provide more complex and customizable dashboards for the APIs managed by APInf, aligned with the interests of the FIWARE foundation, WireCloud needs to have components that support harvesting, transforming and displaying the data harvested from an APInf instance.

In order to use APInf data and visualize it using WireCloud, the WireCloud

visualization catalogue can be used to generate the desired data displays, since most of those components are generic, and would accept most kinds of input and data sources. Therefore, only a new operator that is capable of harvesting APInf data is needed in order to support APInf data visualizations using WireCloud.

The `/api-umbrella/v1/analytics/drilldown` endpoint provides API hit data, which would be the core of an APInf data visualization. This endpoint provides data about the hits done to the API and to each of the API endpoints, the source IPs of the users connecting to the API and latency of the responses among others.

Using this data users can , for instance, easily generate charts with the hits done to the API over time, latency charts and keep track of the unique IPs connecting to the API each day, which can provide a lot of value for the owners of those APIs, helping them see which endpoints of their APIs are the most used, and helping them notice bottlenecks in their APIs and their response time.



# Chapter 7

## Conclusions

With the rapidly increasing data available to users nowadays, specially on data ecosystems like the one FI-NEXT aims to archieve, having the tools required to extract useful information from such data in a way that is easy to use and understand for the end users. Data portals like CKAN have large amounts of data ready to be consumed, but has no visualization capabilities built in.

Thanks to the WireCloud visualization catalogue users now have the tools required to generate powerful visualizations using the WireCloud platform. This catalogue allows users to generate complex mashups that provide the ability to generate a wide range of dashboards that can be fully customized easily in order to satisfy the needs of each end user. Even though this mashups can be difficult to generate manually for users with limited knowledge on the platform, thanks to the “CKAN dashboard creator widget” (which, although intended for CKAN, could be used from outside CKAN as well) users can generate this dashboard through a simple and easy to use interface that requires no previous knowledge from the user creating the dashboard. While this is limited compared to what a skilled WireCloud user could create manually, it provides most of the options users will need and use, while still being easily customizable afterwards so users can customize it to fit their needs.

Since the visualization dashboards are built in WireCloud, users can easily duplicate and share the visualization dashboards, allowing users to customize the dashboard with little effort, so the data displayed is more aligned to their needs, this way, multiple users working on the same dataset could modify the charts displayed on an existing visualization for that data, customizing them so that the information they need is displayed, while not needing to create the data visualization.

Also, since the components are mostly generic, specially the data transformation operators, they can be alongside with other WireCloud widgets and operators from outside the WireCloud visualization catalogue, allowing users to easily plug in their own data sources and generate visualizations for them.

By both the WireCloud platform and the WireCloud visualization catalogue being open source, users can customize the components and even WireCloud itself to adapt it to the needs and looks desired by the users.

# Chapter 8

## Future work

While developing this project, lots of new ideas and possibilities for improvement showed up, that, due to time and effort constraints could not be tackled; Despite this, these could provide a lot of value to the goals of the project, improving WireCloud data visualization capabilities as well as improving other FIWARE technologies. Below are detailed those improvement lines that I consider most interesting and useful.

In order to make the CKAN dashboard creation easier, the “CKAN dashboard creator widget” user interface could be improved, by for example providing visual clues on what each component looks like, and even adding a final dashboard preview on real time by using WireCloud’s temporary components, which allow the creation of temporary widgets and operators that are not stored on the server, therefore being removed when the user refreshes the site. This could be used to allow users to fully customize the dashboard looks by dragging around this temporary WireCloud components in order to set up the final dashboard, allowing this way user input in the final look of the dashboard instead of depending on the sizes and location the widget chooses for the dashboard widgets.

For the dashboard creation itself, better widget placement could be added, using sizes depending on the amount and order of the new dashboard chosen components, instead of using a standard size for every widget and placing them in the order they were created. Along these lines, component placement in the wiring view should also be improved, since currently all components are placed in top of each other in the wiring view. A better wiring placement would allow users to modify easily the dashboard to tweak when new needs arise, as well as allowing users to use the dashboard’s wiring as a point of reference in extension of the dashboard or even the creation of their own new dashboards.

More “components” could be added to the widget too, allowing for more chart types to be generated, or supporting more configurations for each component, for

instance, instead of choosing a single variable, selecting the result of an arithmetic operation between two variables.

Better recommendations could be provided to the users by storing on the WireCloud server data about the choices made by previous users using the same or similar CKAN datasets. This could be done by storing this values using a widget property, which behave like preferences, but cannot be edited manually by the user, only being changed by the component's code. The downside of this is that, since this values are stored on the WireCloud server, a CKAN instance using multiple WireCloud servers for data visualization would not have all of this data, although multiple CKAN servers using the same WireCloud server could share this data and make it easier to make recommendations for the users.

Also, the widget could have an option to add the components and settings required to build a reference dashboard for the dataset (specially when working with the FIWARE DataModels), so that users could have a more advanced starting point when creating new dashboards to visualize CKAN data.

The “CKAN dashboard creator widget” makes use of the REST API provided by WireCloud instead of using the built-in dashboard creation API WireCloud has for components since it is limited. This API should be improved so that this widget and future others can use it instead of having to implement their own REST API calls.

The WireCloud visualization catalogue, even though large enough to fulfill the needs of most users, could be extended to support more chart types and more data transformation operations. Furthermore, ad-hoc filters could be added for each of the FIWARE DataModels, providing an easy to use filter made specially for the kind of data the model has. On these line, more data sources could be supported by adding a data harvester operator for them, adding support this way for the data source, since the generic WireCloud visualization catalogue could be used to generate visualizations for them without the need of developing new components for this task. Alongside this catalogue extension, the “CKAN dashboard creator widget” should be extended in order to add the new operations and data visualization available to the user.

The dashboard creation widget could support the use of multiple data sources as well, allowing users to use multiple “CKAN source operators” aimed at different resources and merging their inputs. This can already be done using the WireCloud visualization catalogue, for example, by using the “Union list operator”. Also, support for other data sources apart from CKAN could be added, allowing users to generate visualization for any kind of data instead of being restricted to CKAN. For instance, support for GitHub and GitLab could be added in order to generate agile

data visualization dashboards using the existing components aimed at these types of data.

Regarding APIInf visualization dashboards, monetization charts could also be added, keeping track of the API calls and the fee each user has to pay for them. This way, users can generate dashboards to easily keep track of the revenue their APIs generate.



# Bibliography

- [1] APIInf (2018). APIInf Webpage.<http://www.apinf.com/> Last accesed on 11/05/2018.
- [2] APIUmbrella (2018). API Umbrella webpage.<https://apiumbrella.io/> Last accesed on 11/05/2018.
- [3] Carter, M. (2013). Charts. In Designing Science Presentations, pages 95–116.
- [4] Kuan, J. (2012). Learning Highcharts.
- [5] Mahizhnan, A. (1999). Smart cities. Cities, 16(1):13–18.
- [6] FIWARE (2018). FIWARE and APIInf. <https://www.fiware.org/2017/07/28/apinf-contributes-top-class-api-management-technologies-to-fiware-platform/> Last accesed on 11/05/2018.
- [7] FIWARE (2018). FIWARE DataModel webpage. <https://www.fiware.org/developers/data-models/>. Last accesed on 15/04/2018.
- [8] Wainwright, M. (2012). Using CKAN : storing data for re-use. OR2012: Open Repositories.
- [9] Yu, J., Benatallah, B., Casati, F., and Daniel, F. (2008). Understanding mashup development. IEEE Internet Computing, 12(5):44–52.
- [10] FIWARE (2018). FIWARE webpage. <https://www.fiware.org/>. Last accesed on 24/04/2018.
- [11] FIWARE (2018). NGSI webpage. [http://fiware.github.io/context.Orion/api/v2/stable/](http://fiware.github.io/context/Orion/api/v2/stable/). Last accesed on 12/04/2018.
- [12] Usama M. Fayyad. Information Visualization in Data Mining and Knowledge Discovery

- [13] A. Zanella, N. Bui, A. Castellani, L. Vangelista and M. Zorzi, "Internet of Things for Smart Cities"
- [14] FIWARE (2018). ORION webpage. <https://catalogue.fiware.org/enablers/publishsubscribecontext-broker-orion-context-broker>. Last accessed on 19/04/2018.
- [15] FIWARE (2018). WireCloud webpage. <https://catalogue.fiware.org/enablers/applicationmashup-wirecloud>. Last accessed on 17/04/2018.
- [16] Gershenfeld, N., Krikorian, R., and Cohen, D. (2004). The internet of things. *Scientific American*, 291(4):76–81.
- [17] Giffinger, R. (2007). Smart cities Ranking of European medium-sized cities. *October*, 16(October):13–18.
- [18] ] Google Developers (2018). Google Charts webpage. <https://developers.google.com/chart/>. Last accessed on 19/04/2018.
- [19] Hermans, P. (2015). Smart Cities FIWARE platform backed by Telefonica, Orange, Engineering and Atos. *ISN Conference*. 79 Future work
- [20] Hierro, J. (2016). FIWARE, el estándar que necesita el IoT. In *The IoT World of Telefónica*.
- [21] Huang, P.-Y. and Jan, J.-F. (2011). Comparison of Google maps API and openlayers for WebGIS development. In *32nd Asian Conference on Remote Sensing 2011, ACRS 2011*, volume 4, pages 2379–2382.
- [22] Keim, D. (2002). Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):1 –8.



Este documento esta firmado por



<b>Firmante</b>	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
<b>Fecha/Hora</b>	Sat Jun 16 18:22:24 CEST 2018
<b>Emisor del Certificado</b>	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
<b>Numero de Serie</b>	630
<b>Metodo</b>	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)