Development of Decidim.Viz

Features, technical decisions and incidents found

For the development of Decidim.Viz, we tried not to start from scratch but to carry out an investigation of different previously existing proposals, requests and workshops on Decidim in order to identify potential users and the needs that could be covered from a tool display. In this first phase, we recap the minutes and recordings of different workshops on Decidim that have been held in recent years. From all these events, we identified requirements and proposals that were reflected by the people participating in said events. Subsequently, the potential users that the tool should serve were identified:

- **Community**: The data displayed in Decidim.Viz must be useful for the community of people involved in the decisions that are made in a specific instance.

- **data science**: You must provide information that facilitates massive data analysis without having to have technical knowledge about GrapHQL or Decidim.

A feasibility analysis of each of these proposals was carried out taking into account different factors such as:

- **Technical viability**: The proposal can be implemented at a technical level or if there are factors that prevent its development.
- **technical availability**: The proposal can be developed taking into account that the project is implemented by a programmer who works part-time and who must combine his functions with other Tecnopolítica.net and CNSC projects.
- **Cohesion**: The proposal can be grouped together with other functionalities and if it corresponds to a potential user profile of a tool like Decidim.Viz

Decidim.Viz Technical Development: First Decisions

Architecture

First, we proceeded to define the basic architecture of the viewer. For this first step, it is necessary to keep in mind that a visualizer is limited to displaying existing information from a data source. In the case of Decidim instances, this data comes from two different sources:

- **Decidim API**: It is an API that uses GrapHQL technology and that allows access to a large amount of information about a Decidim instance as well as various metrics.
- **Decidim data files**: In principle, Decidim instances should have the option of downloading a set of files in CSV (Comma-Separated Values) format with information on all the proposals and comments made.

The information available through the aforementioned sources must be processed and subsequently sent to a graphical interface. Therefore, we can schematically represent the architecture of Decidim.Viz as illustrated in Figure 1.

Going deeper into the analysis of the different data sources, we can identify advantages and disadvantages of each of them:

Fountain	Advantages	Drawbacks
CSV data files	● All the information is available from the first moment. ● Information processing is easier.	● You don't have information in time real, but depends on the download date data file. ● The information needs to be pre-processed prior to its use by the visualizer. ● Not all instances of Decidim provide the data file not all provide

		the same files of data. ● The format changes between instances, which that makes it difficult or even impossible to design of a visualizer common for all.
API	● The information obtained is always updated. ● The set of information available is greater than that of the CSV data files. ● In some cases, the data is easily obtainable and does not require a large processing job in the user interface.	● The waiting time between the data request and its availability depends on the Decidim API and, in some queries, it is too high. ● Some inquiries are complex and require a job of processing of very information high. ● The format of GrapHQL queries changes according to version of Decidim and GrapHQL used by the instance, which complicates the task of performing a common display for all.

Both data sources present a problem with the lack of standardization between instances, which complicates the creation of a "universal viewer". Faced with this problem, there are two possibilities:

1. **Construction of an ad-hoc viewer for each instance**: This option would allow further analysis of the data of a specific instance, but the viewer could not be used for other instances.
2. **Constructing a visualizer for a series of concrete instances**: A common viewer is made that works with a series of selected instances

previously. Some of the code developed in the visualizer will not be common and it will be necessary to “reinvent the wheel” for each specific detail of an instance. In exchange, we will obtain a visualizer that works in different instances previously selected and configured by the technical team. It will not be possible to view any instance. The visualizer will perform a general analysis of data that is applicable to all instances.

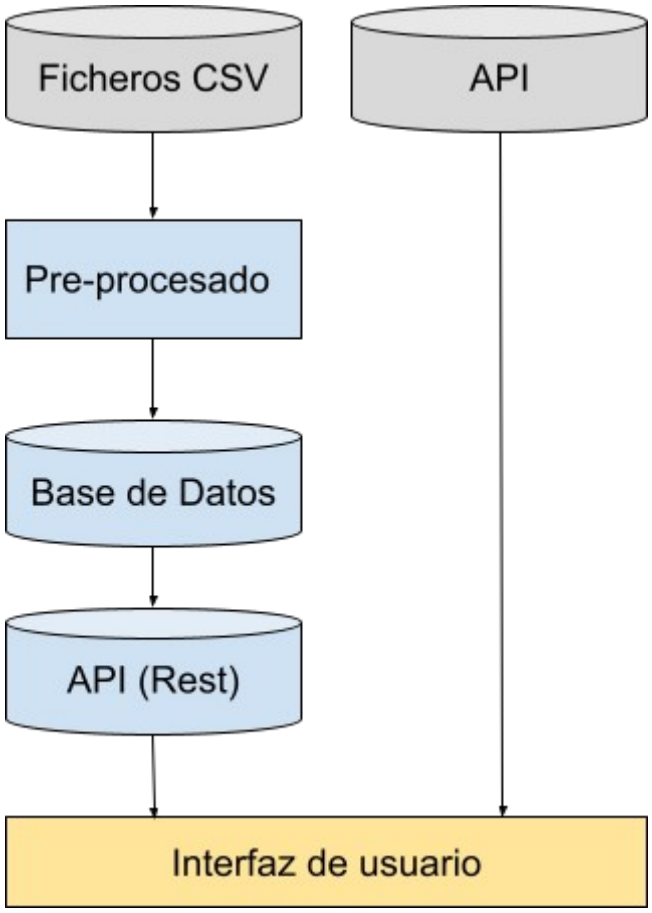


Figure 1: Schematic representation of the Decidim.Viz architecture

In principle, for the pilot test we have opted for the second solution, making a general design that allows working with several instances, but using as an example futureu . Futureu is a Decidim instance launched by the European Commission and whose purpose is to gather proposals from the citizens of the European Union. We chose this option because it has a large number of users, proposals and comments; which can allow us to identify the limits that a web viewer can have and make a more scalable design.

back end

Given the limitations of a technical, temporary and availability nature, it is necessary to resort to tools and support elements that greatly reduce the work to be done, avoiding complex and expensive developments. For the development of the pre-processing layer and the Rest API, we have chosen to use the Python programming language, which allows a very fast development process and is a language with a simpler learning curve than other alternatives, which that allows other people to also get involved in the development of the visualizer without having very high knowledge of programming or Software Engineering.

Within the Python ecosystem, we have opted fordjango for the implementation of the information pre-processing, the database and the Rest API. Django saves development time and effort, and has extensive community support, which can provide support for issues that occur during the development and maintenance of Decidim.Viz.

front end

While Django could also be used to implement the user interface, we've decided to move this responsibility to another web framework: Angular. The choice of Angular is due to the fact that it is a widely used technology, with a large number of libraries and plugins already implemented that save us effort and time as it is a technology known by the technical team, which saves time with respect to the learning other frameworks such as React or Vue.js.

development diary

This section details the Decidim.Viz implementation process as well as the different problems that have arisen, so that these problems can be addressed in future workshops with the community of people interested in building the visualizer.

back end

As mentioned above, for the implementation of the back-end, the Python programming language and the Django development framework have been chosen. not generally

there were important incidents that can be reviewed in this section. For the pilot test, a relational database has been created in the SQLite Database Management System . All of this work has been delegated to Django, who is in charge of creating, using, and maintaining the database. In a future production launch of the viewer, however, it must be taken into account that SQLite is not an ideal database for production and a more solid and robust alternative must be sought.

However, it is necessary to point out a very important technical limitation that affects the visualization of data that requires very important information processing, especially the visualization of user interaction networks. This problem has only been partially addressed and is explained in greater detail in the next section of this document.

In Django, a single application has been created, called "stats" where a series of endpoints that retrieve the data from the database are defined. The database is fed by a series of scripts that are responsible for reading the information from the .csv files that must be downloaded periodically to keep the System updated.

Processing of a large amount of information or application of highly complex algorithms

The visualization of interaction networks requires the handling of a large amount of data and RAM memory. On average, the information processing time has been about three days, which makes it completely unfeasible to collect the information in real time and display it to the user. On the other hand, in the case of instances such as Futureu, the large number of existing nodes makes the processing of information in web format unfeasible, since browsers and libraries are not capable of rendering this information.

On the other hand, not all the information can be obtained from the Decidim API, for reasons of efficiency and complexity. An example of this are the proposals, which in order to be downloaded from the API, it is necessary to know the id of the proposal and the id of the participatory space to which they belong, the user having to know this information previously. While obtaining the proposal from the data file is immediate, requiring only knowing the id of the desired proposal.

The same situation occurs with comments. From the data file, it is enough to know the id of the comment to be able to obtain it. If we have to download a comment by making a request to the Decidim API, we need to know the id of the participatory space to which the proposal belongs, the id of the proposal and the id of the comment.

front end

For the development of the Front-end, Angular has been used, as previously mentioned, together with several libraries that allow speeding up development. The most prominent are:

- **Apollo**: Allows making requests to the Decidim API.
- **PrimeNG**: Provides a series of Angular components that allow performing the most typical functions (tables, buttons, forms...)
- **plotly**: Visualization library, used to display graphs on the web.
- **AG Angular charts**: Allows the visualization of histograms and Treemaps, among others.
- **ngx-translate**: Allows the internationalization of the web, so that it can be displayed in several languages.
- **OpenLayers**: Set to display maps.

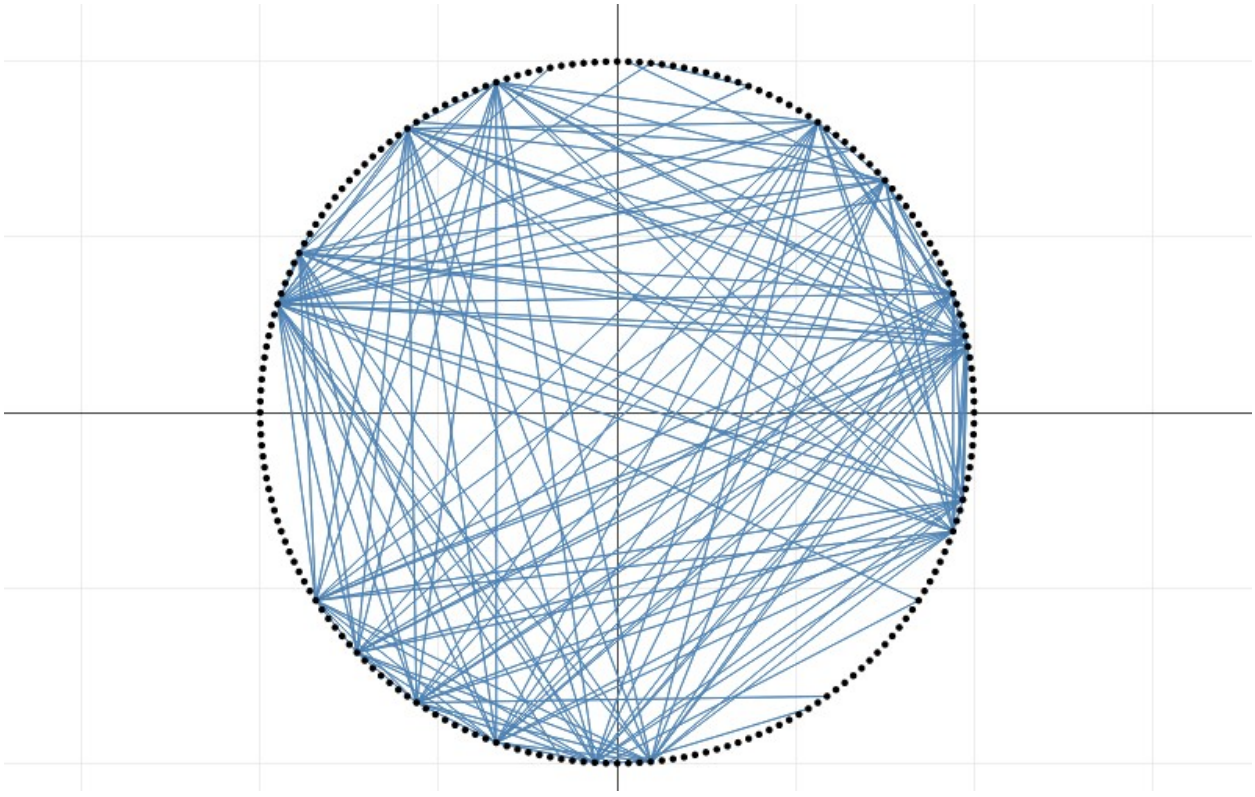
Types of visualization: advantages and disadvantages

Thanks to the use of different information visualization libraries in Angular (Plotly and AG angular charts) it has been possible to explore different information display options. The advantages and disadvantages of each of them are detailed below.

The TreeMap allows to visualize the proportion of different values. For example, in the previous image we can see the proportion of languages used in the comments made in Futureu. The problem with this type of display is the library's treatment of minority values, such as, in this case, the lesser-used languages in the instance. It generates its space but does not indicate its name, and the user must pass the mouse over each square without a name to find out what value it is. No way has been found to correct this behavior, which also seems designed on purpose since the library, in its documentation page, shows it like this in all the examples.

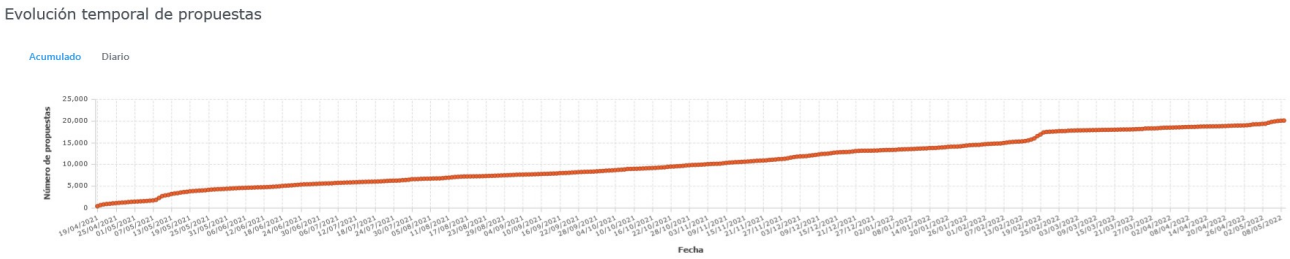
Título	Comentarios
Have your say on European Union policies	2106
European rights and values	1884
Protecting our democracies	1564
Restoring biodiversity and cutting pollution	1249
A more inclusive and fairer economy	1071
Education	986
Ensuring a fair and inclusive transition	757

Another alternative is to display information using tables, using the Table component provided by PrimeNG (which is widely used on the internet and has a high level of customization, as well as being very stable). This option is simple and facilitates the maintenance of the application, in addition to speeding up its development. However, it is the least “visual” element.

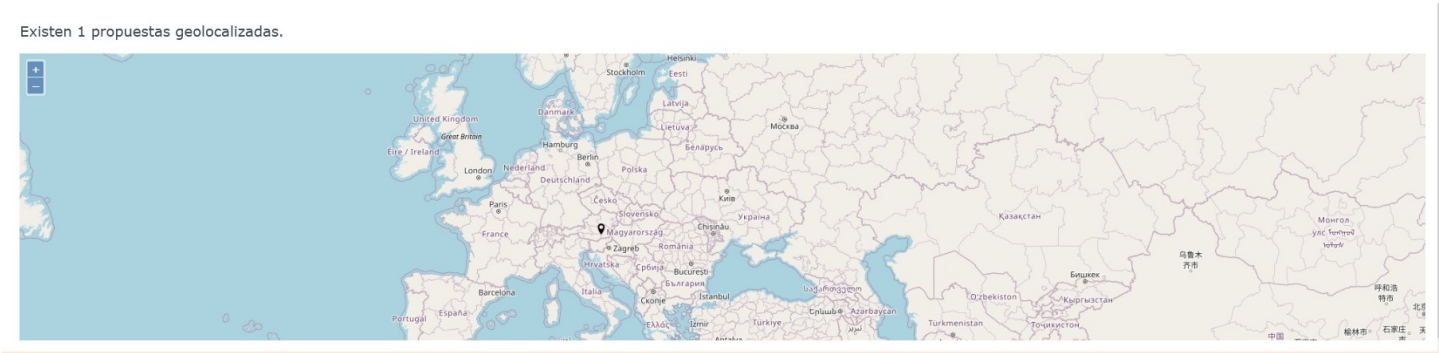


Another visualization alternative are graphs, which allow exploring the connections between different elements that form a network. Although it is a very visual option and that can

give very interesting information, the limitations of customization of the graph in terms of colors and visual designs is important. Also, it gives problems when generating graphs of many elements.



The Histogram is another resource that we can use to visualize information. In general, there have been no problems to highlight, except for its behavior when the data grows exponentially (and must be displayed on a logarithmic axis) or when there is little information, that the Y axis duplicates data and can generate some kind of confusion.



With the help of the OpenLayers library and the ProjectOpenStreetMaps , you can add maps in Decidim.Vis that show information about the geolocation of different proposals. Of all the possible visualizations, this is perhaps the most complex to create and maintain.

Summary of limitations encountered during the development phase

go	Description	Alternative
1	Not all information can be obtained from the Decidim API, for reasons of efficiency or complexity.	Use the downloaded .csv files. This prevents the possibility of obtaining information in real time.
2	Decidim's API requests vary by version, making it impossible to maintain a generic viewer that works for every instance.	
3	Not all instances provide the same data files, and their structure varies in each instance, making it impossible to maintain a generic viewer that works for each instance.	
4	The visualization of graphs requires a great complexity of information treatment.	Maintain a cache version of the generated graph, although it prevents getting information in real time.
5	The use of caches or systems that periodically download the .csv data files introduce complexity and make the viewer difficult to maintain.	
6	Although it has been possible to implement maps to visualize the geolocation of the proposals made. In Futureu, only one proposal has this information, which means that this functionality is not especially useful, despite the fact that it has required a significant effort in its implementation.	
7	Additional knowledge of User eXperience (UX) is required to complete the development of the Decidim.Vis user interface	
8	The visualization possibilities (tables, graphs, graphs...) are limited by the visualization libraries used: Not always any type of visualization and with any degree of customization is possible.	
9	It is necessary to find a server to host the viewer that is appropriate to the expected use.	

10	Currently, there is only one person developing Decidim.Vis, which means that development is not particularly fast or agile.	
eleven	The graph visualization library has not implemented functionality to display information about the links between nodes on hover.	