Desarrollo de Decidim.Viz

Funcionalidades, decisiones técnicas e incidencias encontradas

Para el desarrollo de Decidim.Viz, tratamos de no partir de cero sino de realizar un trabajo de investigación de diferentes propuestas, peticiones y talleres sobre Decidim previamente existentes con el fin de identificar los usuarios potenciales y las necesidades que se podían cubrir desde una herramienta de visualización. En esta primera fase, recapitulamos las actas y grabaciones de diferentes talleres sobre Decidim que se han realizado en los últimos años. De todos estos eventos, identificamos requisitos y propuestas que se plasmaron por las personas participantes en dichos eventos.Posteriormente, se identificaron los usuarios potenciales a los que debía servir la herramienta:

- **Comunidad**: Los datos visualizados en Decidim.Viz deben ser de utilidad para la comunidad de personas involucradas en las decisiones que se toman en una instancia concreta.
- **Ciencia de datos**: Se debe proporcionar información que facilite el análisis masivo de datos sin tener que disponer de conocimientos técnicos sobre GrapHQL o Decidim.

se realizó un análisis de viabilidad de cada una de estas propuestas teniendo en cuenta diferentes factores como:

- **Viabilidad técnica**: La propuesta se puede implementar a nivel técnico o si hay factores que impiden su desarrollo.
- **Disponibilidad técnica**: La propuesta puede desarrollarse teniendo en cuenta que el proyecto es implementado por un programador que trabaja a jornada parcial y que debe compaginar sus funciones con otros proyectos de Tecnopolítica.net y del CNSC.
- **Cohesión**: La propuesta puede ser agrupada conjuntamente con otras funcionalidades y si se corresponde a un perfil de usuario potencial de una herramienta como Decidim.Viz

Desarrollo técnico de Decidim.Viz: Primeras decisiones

Arquitectura

En primer lugar, se procedió a definir la arquitectura básica del visualizador. Para este primer paso, es necesario tener en cuenta que un visualizador se limita a mostrar información existente desde un origen de datos. En el caso de las instancias de Decidim, estos datos provienen de dos fuentes diferentes:

- **API de Decidim**: Es una API que utiliza la tecnología GrapHQL y que permite acceder a gran cantidad de información sobre una instancia de Decidim así como a diversas métricas.
- **Ficheros de datos de Decidim**: En principio, las instancias de Decidim deben disponer de la opción de descargar un conjunto de archivos en formato CSV (Comma-Separated Values) con información sobre todas las propuestas y comentarios realizados.

La información disponible por medio de las fuentes anteriormente mencionadas debe ser tratada y posteriormente enviada a una interfaz gráfica. Por ello, podemos representar esquemáticamente la arquitectura de Decidim.Viz tal y como queda ilustrado en la Figura 1.

Adentrándonos más en el análisis de las diferentes fuentes de datos, podemos identificar ventajas y desventajas de cada una de ellas:

Fuente	Ventajas	Inconvenientes
Ficheros de datos CSV	● Toda la información está disponible desde el primer momento. ● El tratamiento de la información es más sencillo.	● No se tiene información en tiempo real, sino que depende de la fecha de descarga del fichero de datos. ● La información necesita ser pre-procesada previamente a su uso por parte del visualizador. ● Ni todas las instancias de Decidim proporcionan el fichero de datos ni todas proporcionan

		los mismos ficheros de datos. ● El formato cambia entre instancias, lo que dificulta o incluso imposibilita el diseño de un visualizador común para todas.
API	● La información obtenida siempre está actualizada. ● El conjunto de información disponible es superior al de los ficheros de datos CSV. ● En algunos casos, los datos son fácilmente obtenibles y no requieren de un gran trabajo de procesado en la interfaz de usuario.	● El tiempo de espera entre la solicitud de los datos y su disponibilidad depende de la API de Decidim y, en algunas consultas, es demasiado elevado. ● Algunas consultas son complejas y requieren un trabajo de procesado de información muy elevado. ● El formato de las consultas GrapHQL cambia según la versión de Decidim y de GrapHQL utilizada por la instancia, lo que complica la tarea de realizar un visualizador común para todas.

Ambas fuentes de datos presentan como problema la falta de estandarización entre instancias, lo que complica realizar un “visualizador universal”. Ante este problema, se presentan dos posibilidades:

1. **Construcción de un visualizador ad-hoc para cada instancia**: Esta opción permitiría profundizar más aún en el análisis de los datos de una instancia en concreto, pero el visualizador no podría ser utilizado para otras instancias.
2. **Construcción de un visualizador para una serie de instancias concretas**: Se realiza un visualizador común que trabaja con una serie de instancias seleccionadas

previamente. Parte del código desarrollado en el visualizador no será común y será necesario “reinventar la rueda” para cada detalle específico de una instancia. A cambio, obtendremos un visualizador que funcione en diferentes instancias previamente seleccionadas y configuradas por el equipo técnico. No será posible visualizar cualquier instancia. El visualizador realizará un análisis general de datos que sean aplicables a todas las instancias.

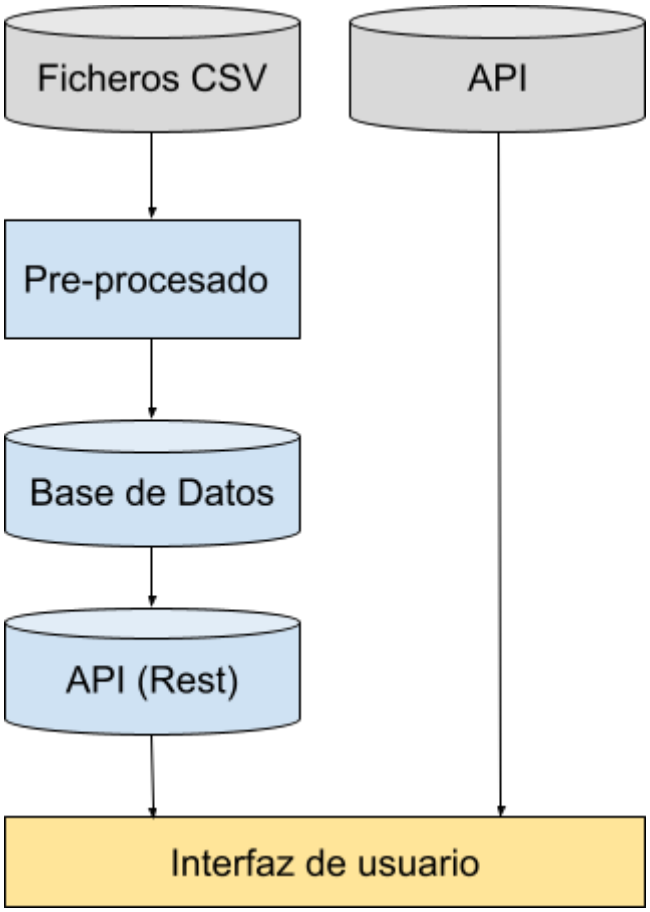


Figura 1: Representación esquemática de la arquitectura de Decidim.Viz

En principio, para la prueba piloto hemos optado por la segunda solución, realizando un diseño general que permita trabajar con varias instancias, pero utilizando como ejemplo Futureu. Futureu es una instancia de Decidim puesta en marcha por la Comisión Europea y que tiene como finalidad recabar propuestas de la ciudadanía de la Unión Europea. Elegimos esta opción porque tiene una gran cantidad de usuarios, propuestas y comentarios; lo que nos puede permitir identificar los límites que puede tener un visualizador web y realizar un diseño más escalable.

Back-end

Dadas las limitaciones de índole técnica, temporales y de disponibilidad, se hace necesario recurrir a herramientas y elementos de apoyo que reduzcan en gran medida el trabajo a realizar, huyendo de desarrollos complejos y costosos. Para el desarrollo de la capa de pre-procesado y de la API Rest se ha optado por usar el lenguaje de programación Python, que permite un proceso de desarrollo muy rápido y es un lenguaje con una curva de aprendizaje más sencilla que otras alternativas, lo que permite que otras personas también puedan involucrarse en el desarrollo del visualizador sin disponer de conocimientos muy elevados de programación o de Ingeniería del Software.

Dentro del ecosistema de Python, nos hemos decantado por Django para la implementación del pre-procesado de información, de la base de datos y de la API Rest. Django permite ahorrar tiempo y esfuerzo de desarrollo y tiene un amplio respaldo de la comunidad, que puede proporcionar soporte a problemas que ocurran durante el desarrollo y mantenimiento de Decidim.Viz.

Front-end

Si bien Django podría también utilizarse para implementar la interfaz de usuario, hemos decidido trasladar esta responsabilidad a otro framework web: Angular. La elección de Angular se debe al hecho de que es una tecnología ampliamente utilizada, con un gran número de bibliotecas y plugins ya implementados que nos ahorran esfuerzo y tiempo y a que es una tecnología conocida por el equipo técnico, lo que ahorra tiempo con respecto al aprendizaje de otros frameworks como React o Vue.js.

Diario de desarrollo

En esta sección se detalla el proceso de implementación de Decidim.Viz así como los diferentes problemas que han surgido, con el fin de que estos problemas puedan ser tratados en futuros talleres junto a la comunidad de personas interesadas en la construcción del visualizador.

Back-end

Como se mencionó anteriormente, para la implementación del back-end se ha optado por el lenguaje de programación Python y el framework de desarrollo Django. En general, no

hubo incidencias importantes que se puedan reseñar en esta sección. Para la prueba piloto, se ha creado una base de datos relacional en el Sistema Gestor de Base de Datos SQLite. Todo este trabajo se ha delegado a Django, quien se encarga de la creación, uso y mantenimiento de la base de datos. En una futura puesta a producción del visualizador habrá que tener en cuenta, no obstante, que SQLite no es una base de datos idónea para producción y se deberá buscar una alternativa más sólida y robusta.

Sin embargo, es necesario señalar una limitación técnica muy importante que afecta a la visualización de datos que requieren un procesamiento de la información muy importante, especialmente a la visualización de redes de interacción de usuarios. Este problema ha podido ser abordado solamente de manera parcial y se explica en la siguiente sección de este documento, en mayor detalle.

En Django, se ha creado una sola aplicación, llamada “stats” donde se definen una serie de endpoints que recuperan los datos de la base de datos. La base de datos es alimentada mediante una serie de scripts que se encargan de leer la información proveniente de los ficheros .csv que hay que descargar periódicamente para mantener el Sistema actualizado.

Procesamiento de gran cantidad de información o aplicación de algoritmos de gran complejidad

La visualización de redes de interacción requiere el manejo de un gran número de datos y de memoria RAM. En promedio, el tiempo de procesamiento de la información ha sido de unos tres días, lo que hace completamente inviable recoger la información en tiempo real y mostrarla al usuario. Por otro lado, en el caso de instancias como Futureu, la gran cantidad de nodos existentes hace inviable el procesamiento de la información en formato web, ya que los navegadores y las bibliotecas no son capaces de renderizar esta información.

Por otro lado, no toda la información puede ser obtenida des de la API de Decidim, por motivos de eficiencia y complejidad. Un ejemplo de ello son las propuestas, que para ser descargadas desde la API se requiere conocer el id de la propuesta y el id del espacio participativo al que pertenecen, debiendo el usuario conocer esta información previamente. Mientras que la obtención de la propuesta desde el fichero de datos es inmediata, requiriendo únicamente conocer el id de la propuesta deseada.

La misma situación ocurre con los comentarios. Desde el fichero de datos, basta con conocer el id del comentario para poder obtenerlo. Si tenemos que descargar un comentario realizando una petición a la API de Decidim, necesitamos conocer el id del espacio participativo al que pertenece la propuesta, el id de la propuesta y el id del comentario.

Front-end

Para el desarrollo del Front-end, se ha empleado Angular, como se había mencionado anteriormente, junto con varias bibliotecas que permiten agilizar el desarrollo. Las más destacadas son:

- **Apollo**: Permite realizar peticiones a la API de Decidim.
- **PrimeNG**: Proporciona una serie de componentes de Angular que permiten realizar las funciones más típicas (tablas, botones, formularios…)
- **Plotly**: Biblioteca de visualización, se utiliza para mostrar grafos en la web.
- **AG Angular charts**: Permite la visualización de los histogramas y los Treemaps, entre otros.
- **Ngx-translate**: Permite la internacionalización de la web, para que pueda ser mostrada en varios idiomas.
- **OpenLayers**: Permite mostrar mapas.

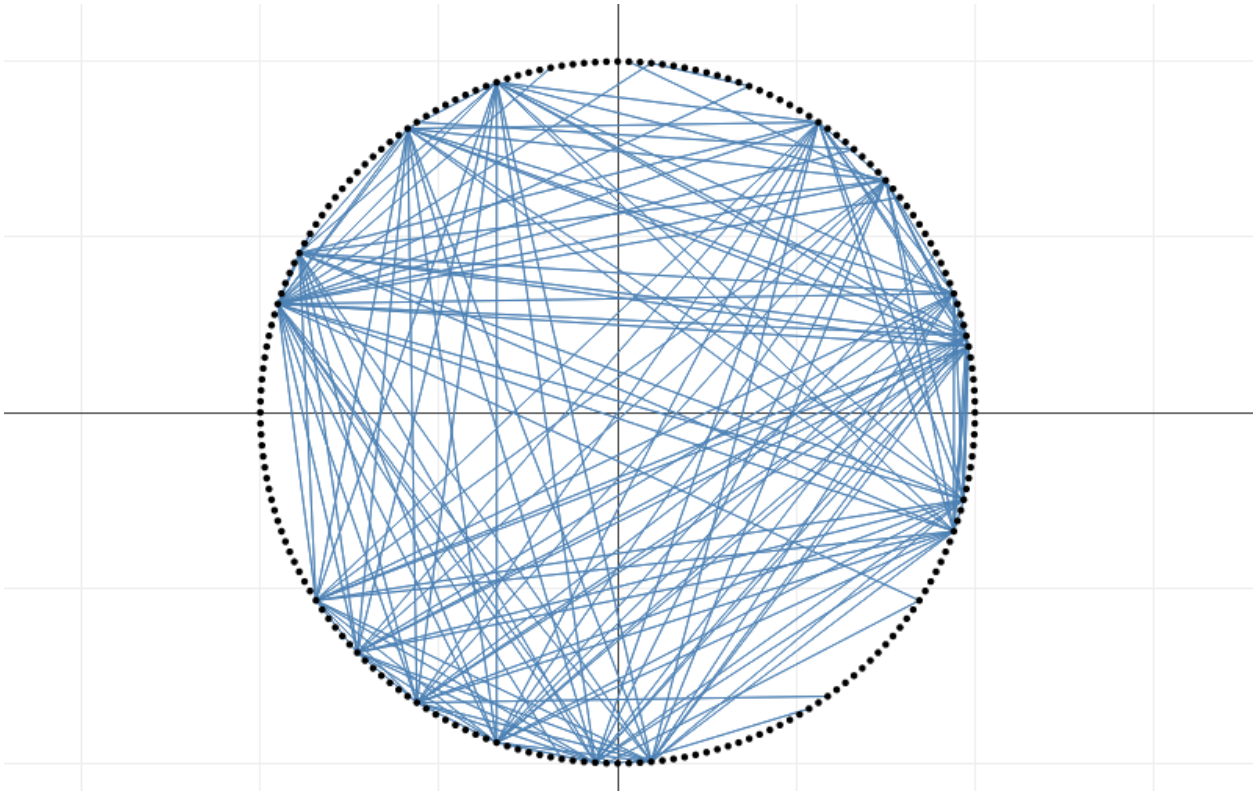
Tipos de visualización: ventajas e inconvenientes

Gracias al uso de diferentes bibliotecas de visualización de información en Angular (Plotly y AG angular charts) ha sido posible explorar diferentes opciones de visualización de información. A continuación se detallan ventajas e inconvenientes de cada una de ellas.

El TreeMap permite visualizar la proporción de diferentes valores. Por ejemplo, en la imagen anterior podemos ver la proporción de lenguajes utilizados en los comentarios realizados en Futureu. El problema de este tipo de visualización es el tratamiento que la biblioteca le da a los valores minoritarios como, en este caso, los idiomas menos utilizados en la instancia. Genera su espacio pero no indica su nombre, debiendo el usuario de pasar el ratón sobre cada cuadrado sin nombre para saber de qué valor se trata. No se ha encontrado manera de corregir este comportamiento, que además parece diseñado a propósito ya que la biblioteca, en su página de documentación, lo muestra así en todos los ejemplos.

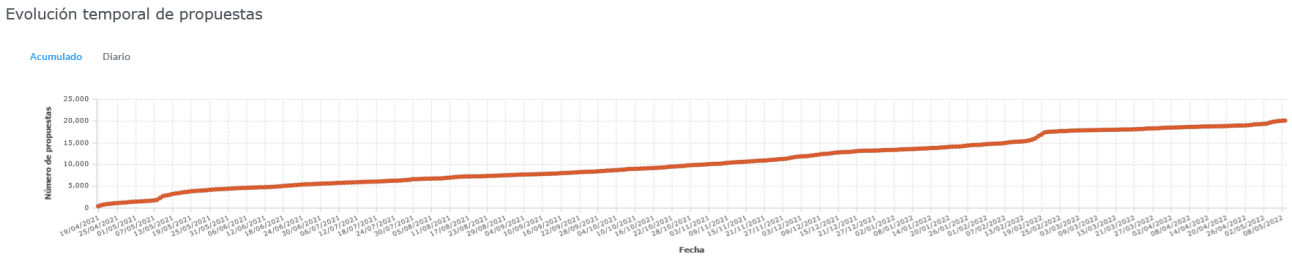
Título	Comentarios
Have your say on European Union policies	2106
European rights and values	1884
Protecting our democracies	1564
Restoring biodiversity and cutting pollution	1249
A more inclusive and fairer economy	1071
Education	986
Ensuring a fair and inclusive transition	757

Otra alternativa es mostrar información mediante tablas, utilizando el componente Table que proporciona PrimeNG (el cual está ampliamente utilizado en internet y tiene un alto nivel de personalización, además de ser muy estable). Esta opción es sencilla y facilita el mantenimiento de la aplicación, además de agilizar el desarrollo de la misma. No obstante, es el elemento menos “visual”.

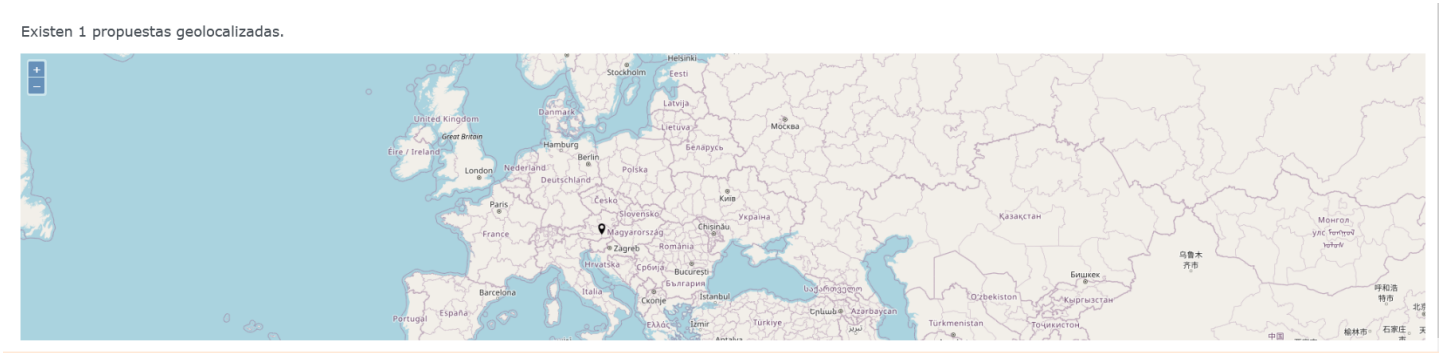


Otra alternativa de visualización son los grafos, que permiten explorar las conexiones entre diferentes elementos que formen una red. Si bien es una opción muy visual y que puede

dar información muy interesante, las limitaciones de personalización del grafo en cuanto a colores y diseños visuales es importante. Además, da problemas cuando genera grafos de muchos elementos.



El Histograma es otro recurso que podemos emplear para la visualización de información. En general no ha habido problemas a destacar, a excepción de su comportamiento cuando los datos crecen de forma exponencial (y deben mostrarse en un eje logarítmico) o cuando hay poca información, que el eje de la Y duplica datos y puede generar algún tipo de confusión.



Con la ayuda de la biblioteca de OpenLayers y del Proyecto OpenStreetMaps, se pueden agregar mapas en Decidim.Vis que muestren información sobre la geolocalización de diferentes propuestas. De todas las visualizaciones posibles, esta es, quizás, la más compleja de realizar y de mantener.

Resumen de limitaciones encontradas durante la fase de desarrollo

Id	Descripción	Alternativa
1	No toda la información puede ser obtenida de la API de Decidim, por motivos de eficiencia o complejidad.	Utilizar los archivos .csv descargados. Se impide así la posibilidad de conseguir información en tiempo real.
2	Las peticiones a la API de Decidim varían según la versión, lo que impide mantener un visor genérico que funcione para cada instancia.	
3	No todas las instancias proporcionan los mismos ficheros de datos, y la estructura de los mismos varía en cada instancia, impidiendo mantener un visor genérico que funcione para cada instancia.	
4	La visualización de grafos requiere una gran complejidad de tratamiento de información.	Mantener una versión caché del grafo generado, aunque se impide conseguir información en tiempo real.
5	El uso de cachés o de sistemas que descarguen periódicamente los ficheros de datos .csv introducen complejidad y dificultan el mantenimiento del visor.	
6	A pesar de que se ha podido implementar mapas para visualizar la geolocalización de las propuestas realizadas. En Futureu una única propuesta dispone de esta información, lo que hace que esta funcionalidad no sea especialmente útil, a pesar de que ha requerido un importante esfuerzo en su implementación.	
7	Se requieren conocimientos adicionales de User eXperience (UX) para completar el desarrollo de la interfaz de usuario de Decidim.Vis	
8	Las posibilidades de visualización (tablas, gráficos, grafos…) se encuentran limitadas por las bibliotecas de visualización empleadas: No siempre es posible cualquier tipo de visualización y con cualquier grado de personalización.	
9	Es necesario buscar un servidor donde alojar el visualizador que sea apropiado al uso esperado.	

10	Actualmente, solamente hay una persona desarrollando Decidim.Vis, lo que propicia que el desarrollo no sea especialmente rápido ni ágil.	
11	La biblioteca de visualización de grafos no ha implementado funcionalidad para mostrar información de los enlaces entre nodos al pasar el ratón por encima.	