UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

Gestor y visualizador de indicadores multidimensionales

Informe de Proyecto de Grado

Ingeniería en Computación

Facultad de Ingeniería

Universidad de la República

Tutora: Lorena Etcheverry

Leandro Burgos Líber Azambuya Matías Landeira

29 de agosto de 2017

Índice

Resumen

A partir de un conjunto de indicadores que modelan los efectos de los cultivos genéticamente modificados en Uruguay, se plantea como propósito de este proyecto la implementación de una plataforma *web* diseñada para la creación y almacenamiento de estos indicadores, que cuentan con estructuras multidimensionales, para que pudieran luego ser expuestos de forma pública tanto en un formato visual y gráfico como en la posibilidad de ser descargados.

En este contexto se implementa la plataforma que describe este informe, conformada principalmente por dos componentes independientes y complementarios, de *backend* y *frontend*, para llevar a cabo los requerimientos anteriormente mencionados. Dicha plataforma cuenta con la funcionalidad de permitir a un usuario crear esquemas de indicadores que serán almacenados luego en una base de datos en forma de grafos, y a los que luego les podrá asociar conjuntos de datos para poder visualizar en formato gráfico, y posteriormente analizar dicha información. Para esto la plataforma cuenta con una sección de visualización de indicadores en formato de gráficas de barras, que provee al usuario de distintos tipos de visualizaciones para un mismo conjunto de datos, desde distintos puntos de vista, así como realizar un cruzamiento de datos (una visualización simultánea) de dos indicadores con esquemas compatibles. Dentro de esta interfaz *web*, el usuario tiene la posibilidad de ingresar indicadores, así como también editar y eliminar los mismos. A su vez, para cada uno de estos indicadores tiene la posibilidad de insertar datos asociados a los mismos, con la posterior posibilidad de listar estos mismos datos y también eliminarlos si lo desea.

Por último, el usuario tiene la posibilidad de exportar la información almacenada en un formato que cumple los estándares de datos abiertos, y para usuarios ajenos a dichos formatos permitirá también la exportación en formato CSV.

1. Introducción

En este capítulo se enuncian los principales motivos que conllevaron a la realización de este proyecto, quién es la contraparte del mismo y qué necesidades fueron las que generaron la propuesta realizada. Luego se plantean los objetivos encontrados, luego de un análisis de requerimientos y diversas reuniones con la contraparte, que generaron la base de lo que resta del documento. Por último, se plantean los requerimientos, el alcance del proyecto y cuáles son las características con las que debe contar el mismo una vez entregado.

1.1. Contexto

Los organismos genéticamente modificados (OGM) pueden definirse como organismos en los cuales el material genético ha sido alterado de formas que no ocurren naturalmente por apareamiento o recombinación natural, sino por tecnologías de ADN recombinante. La adición de genes foráneos ha sido utilizada en plantas para producir proteínas que otorgan resistencia a pestes y enfermedades, y más recientemente, para mejorar el perfil químico del producto procesado[3].

Basándose en la idea de cualquier información genética, de cualquier fuente, puede ser expresada por cualquier organismo, la ingeniería genética ha posibilitado por ejemplo el mejoramiento de la protección de los cultivos. A la vez, ha permitido acortar los tiempos cuando se trata de obtener nuevas variedades, mejorar el rendimiento y la calidad de los cultivos, producir moléculas de alto valor agregado (fármacos, vitaminas, biopolímeros) y mejorar las cualidades nutricionales de las plantas.[1]

En los años posteriores a la introducción del primer organismo transgénico, el tomate Flavr Savr[10], el cultivo de otras varias especies de transgénicos aumentó rápidamente, alcanzando en el año 2016 más de 185,1 millones de hectáreas a nivel mundial[3]. En los últimos años, el número de eventos comercializados globalmente ha aumentado de manera constante. Para el año 2014, el Servicio Internacional para la Adquisición de Aplicaciones Agrobiotecnológicas (ISAAA) había contabilizado un total de 336 eventos de modificaciones genéticas en 27 especies de cultivos aprobados para su comercialización, plantado y/o su uso como alimento[7].

Según el Gabinete Nacional de Bioseguridad, en Uruguay se siembran en la actualidad 16 eventos transgénicos aprobados para uso comercial y consumo (11 de maíz y 5 de soja) y 9 eventos de soja destinados a la producción de semillas de exportación[13].En la última década, el crecimiento de las exportaciones uruguayas obedece principalmente

a la expansión de estos cultivos genéticamente modificados. Esta mayor participación de materias primas no procesadas (*agrocommodities*), ha sido acompañada de un incremento de agroinsumos importados, sujetos a derechos de propiedad intelectual. La fuerte dependencia del sector agrícola nacional del mercado externo, tanto para exportar sus productos como para importar insumos, es lo que pauta su ritmo de crecimiento. En los últimos tiempos se observa una tendencia al alza del precio de los agroinsumos, en contraposición a la baja en el precio de los granos. Este desequilibrio puede derivar en el aumento de externalidades ambientales; no siempre contempladas en el análisis de riesgo de las políticas agrícolas.

En el marco de un proyecto de la Facultad de Ciencias se ha desarrollado un conjunto de indicadores de los efectos ambientales de cultivos genéticamente modificados desde una perspectiva multidimensional, ecosistémica y socioeconómica.
El objetivo de dichos indicadores es ponderar los costos-beneficios de las externalidades ambientales o daños ambientales (institucionales, ecosistémicos, económicos y sociales) derivados de la producción y comercialización de cultivos genéticamente modificados.

En este contexto es que surge este proyecto, propuesto inicialmente por la Licenciada Liliana Terradás, quien plantea la necesidad de una solución donde centralizar los indicadores ya mencionados, que tenga la capacidad de cargar la información almacenada en el correr del tiempo con nuevos datos y nuevos indicadores, así como también la posibilidad de cruzar datos de los indicadores para poder generar contenido de valor.

Dadas las necesidades planteadas, los objetivos del proyecto consisten en la implementación de una plataforma *web* que permita acceso a las funcionalidades requeridas por la contraparte, de forma que cuente con un ingreso privado de los datos pero de público acceso a la visualización para darle la difusión necesaria. Asimismo, el sistema debe exportar la información como datos abiertos, de manera de generar datos de calidad que puedan ser utilizados por otras plataformas.

Como punto de partida al proyecto, contamos con un conjunto de indicadores proporcionado por la contraparte que servirán de parámetro para saber qué tipos de datos deberá soportar la plataforma, qué modelo siguen los indicadores mencionados y con qué cantidad de datos aproximados contará cada uno.

1.2. Objetivos

El principal objetivo de este proyecto es desarrollar una plataforma *web* que dé soporte a la creación y visualización de indicadores como los mencionados anteriormente. Además,

la plataforma debe contar con la capacidad de realizar cruzamiento entre los distintos indicadores que estén relacionados entre sí.

Uno de los requisitos del proyecto y que también es un objetivo es que dichos indicadores deberán estar modelados, y almacenados, en forma de cubos multidimensionales. Luego se cuenta con objetivos más puntuales como realizar un relevamiento de herramientas existentes en el mercado y que cumplan todas o algunas de las funcionalidades planteadas para esta plataforma, de manera de conocer cuáles son las falencias de las mismas y confirmar la necesidad de una nueva herramienta que supere dichas falencias. También forma parte de los objetivos la posibilidad de exportar toda la información almacenada en el sistema, tanto los esquemas de los indicadores como sus datos relacionados, en forma de datos abiertos en y siguiendo el paradigma de *Linked Data*.

Otro de los objetivos del proyecto es que la plataforma sea implementada en base a tecnologías y herramientas de código abierto.

1.3. Resultados esperados

Se espera como resultado contar con una plataforma *web* que permita el ingreso de usuarios al sistema. Dichos usuarios deben tener la capacidad de crear nuevos indicadores e ingresar datos para los mismos, y el usuario público debe poder visualizar cada uno de los indicadores ingresados así como también descargar la información visualizada. La plataforma debe contar con una sección donde el usuario podrá crear nuevos indicadores, ingresando todos los datos requeridos por el mismo (nombre, descripción, dimensiones, tipos de datos, etcétera) y también editar y borrar los indicadores existentes; otra sección también privada donde pueda ingresar y eliminar datos para cada uno de los indicadores ya ingresados.

A su vez, se debe contar con otra sección donde el usuario podrá elegir un indicador a graficar, poder cambiar el tipo de visualización que desee para dicho indicador y realizar operaciones de agregación sobre el mismo; y también debe poder seleccionar un segundo indicador compatible con el primero (en caso de que exista) para poder visualizar ambos en forma simultánea de forma que permita comparar la información de ambos. Luego, en la misma sección debe existir la posibilidad de que una vez seleccionado un indicador, sea posible descargar en formato de datos abiertos toda la información correspondiente al mismo, tanto su esquema como sus datos.

1.4. Estructura del documento

El siguiente documento cuenta con 7 capítulos o secciones, estructuradas de manera tal que se detalla cada aspecto del proyecto realizado yendo desde los aspectos de más alto nivel hacia los aspectos más detallados, como puede ser la implementación del mismo.

En el capítulo 2 se presenta una serie de conceptos, herramientas o lenguajes que fueron usados en durante la ejecución del proyecto, de manera que sus definiciones serán de ayuda para comprender este informe.

En el capítulo 3 se detalla el relevamiento de herramientas realizado previo al desarrollo del proyecto. En el mismo se especifican las herramientas que existen hoy en día junto con sus características, virtudes y defectos. A su vez, se discuten la razones por las que creemos que ninguna de ellas cumple con todas las características para satisfacer las necesidades de la contraparte, lo que motiva a la implementación de una nueva herramienta que las satisfaga.

En el capítulo 4 se presenta el diseño de la solución propuesta, explicando en detalle cada uno de los elementos que componen la arquitectura del proyecto, cuáles son las funciones y responsabilidades de los mismos y de qué manera se comunican entre sí.

En el capítulo 5 se describe detalladamente la implementación de la solución de cada uno de los componentes que conforman la plataforma, desde la interfaz de usuario hasta las operaciones del *backend* con la base de datos.

En el capítulo 6 se presentan un conjunto de pruebas realizadas con el objetivo de determinar ciertas limitantes que presenta la plataforma con distintas cargas de datos.

Por último, en el capítulo 7 se realiza una comparación entre los objetivos planteados al comienzo del proyecto y los resultados obtenidos al final del mismo, resaltando a su vez cuáles objetivos pudieron cumplirse y cuáles no, y cuál es el trabajo a futuro que debería realizarse sobre la plataforma para cumplir dichos objetivos.

2. Conceptos Preliminares

En este capítulo se detalla un conjunto de conceptos, relacionados tanto al modelado de tipos de datos particulares como son los cubos, así como también a distintos lenguajes de modelado y representación de estos datos, que servirán como conocimiento previo a lo que resta del documento, debido a que todos ellos son utilizados en la solución.

2.1. Modelos multidimensionales

Los modelos multidimensionales surgen a partir de una necesidad generada en las últimas décadas por parte de las organizaciones y empresas, que consiste en mejorar los procesos de tomas de decisiones en pos de aumentar su competitividad. Dicha necesidad no era solucionada de la mejor manera por los modelos convencionales de bases de datos (el modelo relacional), ya que están diseñados para soportar las operaciones diarias de las organizaciones y su principal tarea es proporcionar acceso rápido y concurrente a los datos, y no para realizar consultas analíticas.
Es así que surge un paradigma orientado al análisis de las bases de datos organizacionales con el objetivo de mejorar las decisiones tomadas, y dicho paradigma obtiene el nombre de *online analytical processing* (OLAP), y está centrado en las consultas analíticas de la información (*business intelligence*).

Dicho paradigma resultó en la creación de los llamados *data warehouses*, que en general consisten en grandes repositorios con información recolectada de distintas fuentes y que siguen el modelo de datos multidimensional.
Los *data warehouses* y sistemas OLAP están basados en el modelo multidimensional, que percibe los datos en un espacio de N dimensiones, normalmente llamado cubo o hipercubo.

La Figura 2.1 representa un cubo multidimensional que modela las ventas de determinados productos, y cuenta con tres dimensiones: Producto (*Product*), Tiempo (*Time*) y Cliente (*Customer*). Un nivel de dimensión representa la granularidad o nivel de detalle al que corresponden las medidas para cada dimensión del cubo. En el ejemplo, las ventas están agrupadas al nivel de Categoría (*Category*), Cuatrimestre (*Quarter*) y Ciudad (*City*), respectivamente. Las instancias de una dimensión son llamadas miembros. Por ejemplo, Comida de mar (*Seafood*) y Bebidas (*Beverages*) son miembros de la dimensión Producto al nivel Categoría.

Por otra parte, las celdas de un cubo también llamados hechos, tienen asociados valores numéricos llamados medidas. Por ejemplo, cada número de las celdas de la Figura

2.1 representan la medida Cantidad (*Quantity*), indicando la cantidad de unidades vendidas por categoría, tiempo y la ciudad del cliente.

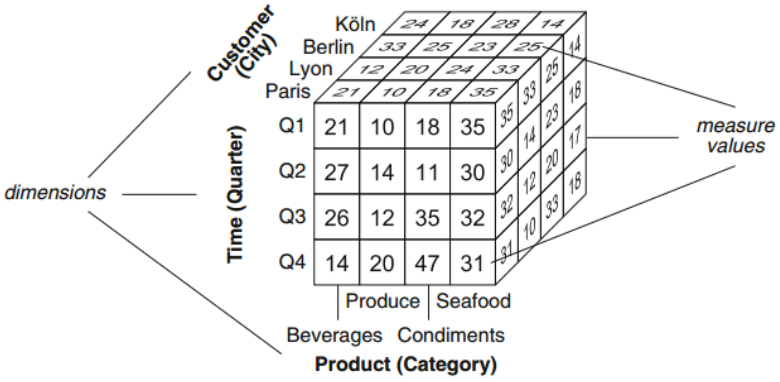


Figura 2.1: Cubo tridimensional para información de ventas, con dimensiones Producto (*Product*), Tiempo (*Time*) y Cliente (*Customer*), y su medida es Cantidad (*Quantity*). [17]

La granularidad de un cubo está determinada por la combinación de niveles correspondientes a cada eje del cubo. En la Figura 2.1 los niveles de dimensión están dispuestos entre parentescos: Categoría para la dimensión Producto, Cuarto para la dimensión Tiempo y Ciudad para la dimensión Cliente. Los cubos permiten analizar su información a distintos niveles de detalle, como en el ejemplo de la Figura 2.1, las ventas pueden verse a un nivel de granularidad fina en a nivel de meses, o a una granularidad más gruesa al nivel del país del Cliente.

Esto se logra a través de las **jerarquías**, que permiten definir secuencias de relaciones entre niveles de detalle más finos con niveles de detalle más gruesos; y dada una relación entre dos niveles en una jerarquía, el nivel más bajo es llamado hijo y el más alto es llamado padre. La Figura 2.2 detalla las jerarquías para el cubo de ejemplo de la Figura 2.1 en una forma simplificada.

En la dimensión Producto los productos se agregan en categirías, en la dimensión Tiempo la menor granularidad es Día (*Day*), que puede agregan en Mes (*Month*), que luego se agregan en Cuatrimestres, Semestres (*Semester*) y Años (*Year*). En la dimensión Cliente, la granularidad más baja es Cliente, que se agregan en Ciudad, Estado (*State*), País (*Country*) y Continente (*Continent*). Usualmente se representa el el nivel más alto de una jerarquía con un nivel llamado Todos (*All*).

A nivel de instancias, la Figura 2.3 muestra un ejemplo de la dimensión Producto, donde cada producto al nivel más bajo de la jerarquía puede ser mapeado a una categoría.

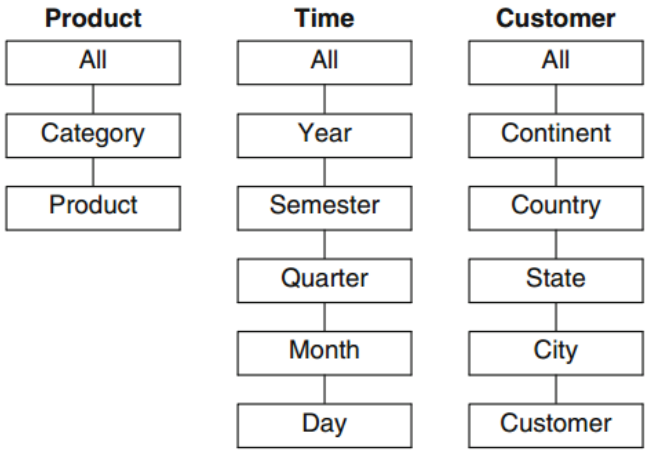


Figura 2.2: Jerarquías de las dimensiones Producto, Tiempo y Cliente.[17]

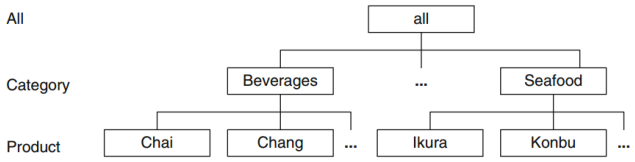


Figura 2.3: Miembros de la jerarquía Producto → Categoría.[17]

Cada **medida** de un cubo está asociada con una función de agregación que permite combinar múltiples valores de medidas en uno solo.
La agregación de medidas se realiza al cambiar el nivel de detalle en el que se quiere visualizar los datos de un cubo. Tomando como ejemplo la jerarquía Cliente de la Figura 2.2 para cambiar la granularidad del cubo de la Figura 2.1 desde Ciudad a País, las ventas de todos los clientes de un mismo país serán agregadas utilizando, por ejemplo, la función suma.

2.1.1. Operaciones OLAP

Las operaciones OLAP son las que permiten visualizar la información de un modelo multidimensional desde distintas perspectivas y a distintos niveles de detalle, mediante el uso de dimensiones y sus jerarquías. Algunas de estas operaciones son las llamadas *roll up*, *drill down*, *slice*, *dice*, *sort*, *pivot* y *drill across*. En el contexto de este proyecto se utilizarán para obtener distintos tipos de visualización de los indicadores las operaciones *roll up*, *drill down* (una es la función inversa de la otra) y *drill across*, que son detalladas a continuación

.

- *Roll up*: agrupa las medidas de los hechos de acuerdo a una jerarquía de dimensión mediante una función de agregación (la suma, por ejemplo), generando una nueva agrupación de las medidas a un nivel de granularidad más grueso. Esto significa que dada una jerarquía que está compuesta por dos o más niveles, donde estará establecida la relación padre-hijo entre ellos, los hechos de un indicador podrán estar representados en función tanto del padre (granularidad más gruesa) como del hijo (granularidad más fina). Para pasar de una agrupación respecto del hijo hacia una respecto del padre, deberá realizarse una función de *roll up* sobre las medidas de los hechos en base a la dimensión correspondiente, de manera que los hechos que coincidan en un mismo valor para el nivel padre, serán agrupados (sumados, por ejemplo) sus valores para el nivel hijo, obteniendo una nueva agrupación de los datos. La Figura 2.4 describe al cubo de la Figura 2.1 luego de realizada la operación de *roll up* sobre una de sus dimensiones.

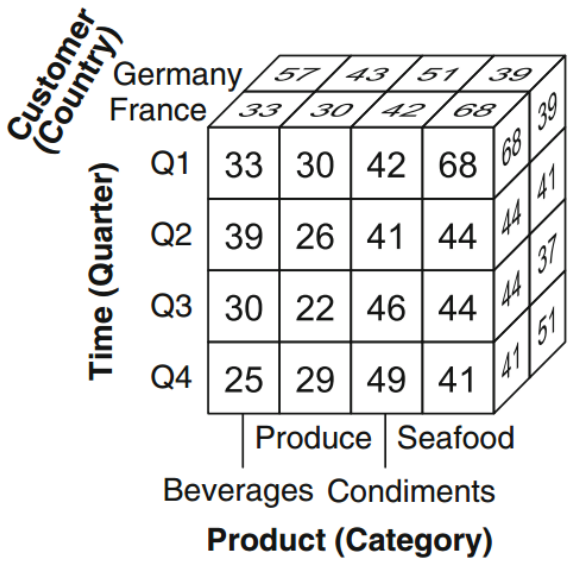


Figura 2.4: *Roll up* de la dimensión Cliente, del nivel Ciudad al nivel País.[17]

- *Drill down*: puede considerarse como la operación inversa al *roll up*, desagregando los datos previamente agregados.

- *Drill across*: operación que permite combinar los datos de dos cubos diferentes que compartan un mismo esquema (al menos una de sus dimensiones) e instancias. La Figura 2.5 muestra un ejemplo del cubo de la Figura 2.1, combinado con otro cubo de igual esquema y diferentes datos.

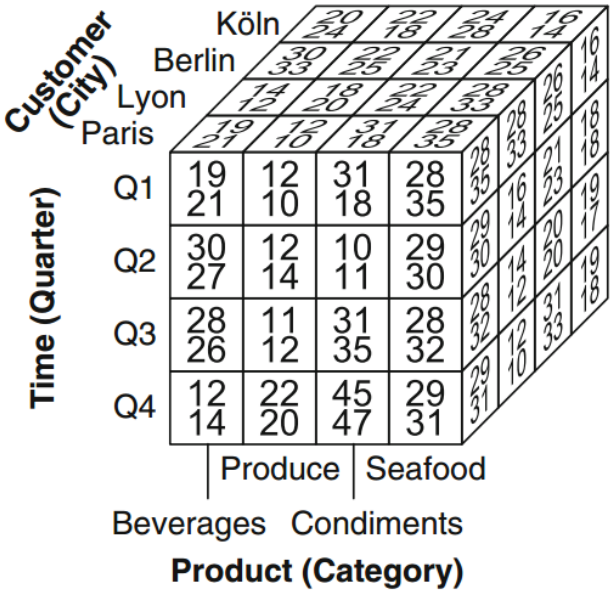


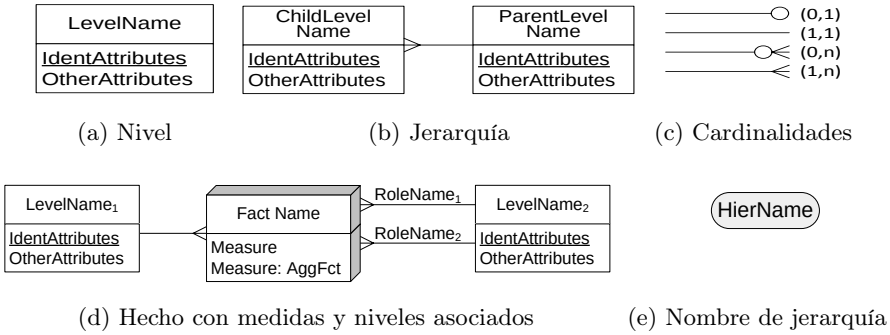
Figura 2.5: *Roll up* de la dimensión Cliente, del nivel Ciudad al nivel País.[17]

2.2. Modelado de cubos

Existen distintas formas de representar cubos multidimensionales, siendo la elegida para este proyecto una representación llamada MultiDim [17]. A continuación se detallan los distintos componentes en el modelado de un cubo y su representación asociada en el modelo MultiDum, como puede verse en la Figura 2.6. Asociaremos cada cubo con un esquema de datos, donde cada esquema se compone de un conjunto de dimensiones y un conjunto de hechos. Una dimensión se compone de un nivel o de una o más jerarquías, y las instancias de un nivel son llamadas Miembros. Un nivel tiene un conjunto de atributos que describe las características de sus miembros (Figura 2.6a), y uno o más identificadores, cada identificador siendo compuesto de uno o más atributos. Una jerarquía está compuesta por un conjunto de niveles (Figura 2.6b), y dados dos niveles relacionados en una jerarquía, el más bajo es llamado el hijo y el más alto el padre. Además, estas relaciones son llamadas relaciones padre-hijo, y pueden tener distintos tipos de cardinalidad (Figura 2.6c).

Una dimensión puede contener varias jerarquías identificadas por un nombre de jerarquía (Figura 2.6e). El nombre del nivel más bajo en una jerarquía define el nombre de la dimensión, salvo en el caso de que un mismo nivel participa varias veces en un hecho, y en dicho caso el nombre del rol definirá el nombre de la dimensión.

Un hecho relaciona varios niveles (Figura 2.6d). Las instancias de un hecho son llamadas Miembros de Hecho, y un hecho puede contener atributos llamados Medidas. La función de agregación asociada a una medida puede declararse junto al nombre de la medida (Figura



(a) Nivel (b) Jerarquía (c) Cardinalidades

(d) Hecho con medidas y niveles asociados (e) Nombre de jerarquía

Figura 2.6: Notación del modelo MultiDim

2.6d), donde la función suma será la utilizada por defecto.

A modo de ejemplo en la Figura 2.7 muestra la representación el MultiDim de uno de los indicadores, llamado Área sembrada, que representa la cantidad de miles de hectáreas sembradas para determinados cultivos en Uruguay, y cuenta con dos dimensiones, una dimensión con un solo nivel llamado Año, que representa el valor en años en los que se ubican los datos, y una jerarquía llamada Cultivos, compuesta por los niveles Cultivo (hijo) y Tipo de Cultivo (padre), que representa los distintos cultivos para los cuales se tiene información. Además, los hechos de este indicador contienen una medida llamada Hectáreas, que representa la cantidad de miles de hectáreas sembradas para un determinado año y un determinado cultivo.

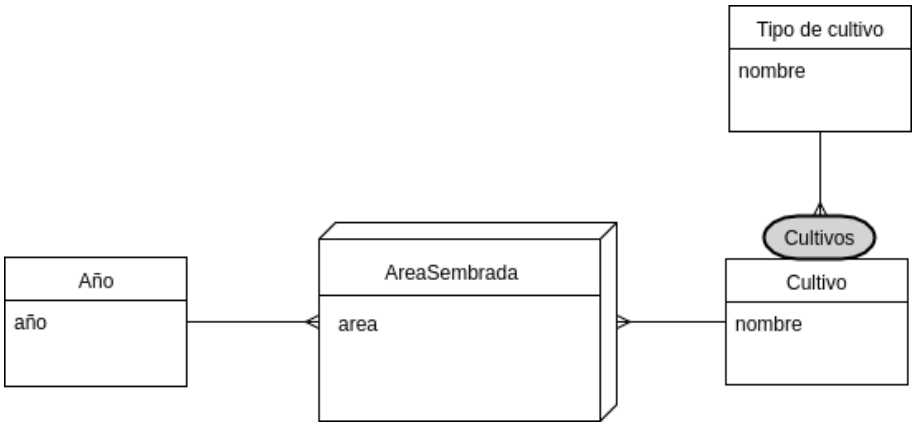


Figura 2.7: Representación del indicador Área sembrada en MultiDim

2.3. Datos abiertos

Datos abiertos [14] es una filosofía que tiene como objetivo poner a disposición de la sociedad los datos que gestionan tanto la administración pública como otros tipos de organizaciones, en formatos simples de manipular.

Para asegurar el cumplimiento de los estándares determinados para Datos Abiertos, es necesario que los datos ofrecidos cumplan los siguientes principios listados a continuación:

1. Se deben compartir todos los datos públicos, los cuales no están restrictos por privacidad o derechos de autor. De esta manera quedan de uso libre para los usuarios (los datos deben estar libres de derechos, patentes, copyright y no estar sujetos a derechos de privacidad).

2. Se deben publicar los datos completos, sin procesar ni resumir, dando así una mayor cantidad de detalles y compartiendo la mayor información posible.

3. Deben actualizarse para que estos no pierdan su valor y se mantengan precisos.

4. Estos datos tienen que poder procesarse automáticamente por maquinas, es decir que deben cumplir con estándares.

5. No tiene que ser necesaria una identificación de usuario para poder acceder a los mismos.

6. Los datos no tienen que publicarse en formatos los cuales precisen de una herramienta propietaria para su procesamiento. Ejemplos de formato abierto serían JSON, CSV o XML.

Tim Berners-Lee (creador de la Web y propulsor de *Linked Data*) propuso una categorización del grado de apertura de los datos basado en estrellas, respecto a qué tan abiertos y reutilizables son los datos que puede ofrecer una institución. A continuación se describe esta categorización [4]:

- 1 Estrella: Se refiere a datos disponible en la Web no siendo una limitante el formato en el que este se encuentre siempre que sea bajo licencia abierta. Por ejemplo datos publicados en formato PDF, TXT, ODT.

- 2 Estrellas: Los datos deben estar estructurados de forma tal que sea posible su procesamiento a través de máquinas. Por ejemplo en formato XLS.

- 3 Estrellas: Los datos deben ser 2 estrellas y también deben estar en un formato no propietario así como CSV en lugar de XLS.

- 4 Estrellas: Usa URIs para denotar cosas, para que sea posible que otras personas apunten a estas.

- 5 Estrellas: Enlaza los datos propios con datos de otras fuentes para proveer un contexto. Esta categoría se conoce también como *Linked Data* y es presentado en la Sección 2.4

El Estado Uruguayo cuenta con un Catálogo de Datos Abiertos [2] publicados por la Agencia de Gobierno Electrónico y Sociedad de la Información y del Conocimiento (AGESIC), provenientes de distintos organismos del mismo como pueden ser el Instituto Nacional de Estadística (INE), la Intendencia de Montevideo, el Ministerio de Salud Pública, entre otros. Algunos de los indicadores publicados en dicho catálogo son el Censo de Población, Hogares y Viviendas 2011, la Encuesta nacional de gastos e ingresos de los hogares 2005-2006, la Tasa de enfermedades y eventos de notificación obligatoria y la Red de monitoreo de la calidad del aire de Montevideo.

A su vez, cada indicador está compuesto por un conjunto de recursos. Por ejemplo, la Red de monitoreo de la calidad del aire en Montevideo cuenta con distintos recursos para las Medidas históricas de la calidad del aire, Lista de identificación de contaminantes, Métodos de medición, Estaciones de Medición, etcétera.

Por otra parte, cada uno de estos recursos está dividido en un conjunto de archivos CSV, que pueden ser visualizados o descargados. Como complemento, se provee también un archivo de texto que contiene una descripción para cada uno de los campos (columnas) que forman parte de los CSV de cada recurso.

2.4. Linked Data

El término *Linked Data* [6] refiere a un conjunto de buenas prácticas para publicación e intercomunicación estructurada de datos en la *web*, dichas prácticas son conocidas como *Linked Data Principles* y son los siguientes:

1. **Utilizar URIs (Uniform Resource Identifier) para identificar los recursos publicados en la *web***. Este principio promueve el uso de referencias URI para identificar no solo documentos Web y contenido digital sino también objetos de la vida real, y conceptos abstractos. Entre estos puede, incluirse conceptos tangibles como personas, lugares o autos, y también conceptos abstractos como el tipo de relación de conocer a una persona, o el conjunto de autos de color verde.

2. **Aprovechar el HTTP de la URI para que se puedan localizar y consultar (es decir, desreferenciar) estos recursos**. El segundo principio indica que para publicar datos en la *Web* aquellos recursos dentro de un dominio de interés primero

deben estar identificados. Como *Linked Data* está construido sobre la arquitectura Web, se utiliza sus mismos términos para referenciar estos recursos de interés, mediante URIs HTTP.

3. **Proporcionar información útil acerca del recurso cuando la URI haya sido desreferenciada, utilizando los estándares (RDF, SPARQL).** El tercer principio implica que los clientes HTTP puedan acceder a la URI utilizando el protocolo HTTP y obtener una descripción del recurso que se identifica mediante esa URI.

4. **Incluir enlaces a otras URI relacionadas con los datos contenidos en el recurso, de forma que se potencie el descubrimiento de información en la *web***. El último principio sugiere proveer enlaces RDF que apunten a otras fuentes de datos en la *Web*. Dichos enlaces externos son fundamentales para la *Web* de datos ya que son quienes conectan los distintos conjuntos de datos independientes para formar un espacio de datos global e interconectado.

La idea de *Linked Data* es alicar la arquitectura general de la *World Wide Web* [23] a la tarea de compartir datos estructurados a escala global. Para comprender estos principios es necesario comprender la arquitectura de la *web* documental clásica.
La *web* documental está construida bajo un conjunto de estándares: URIs como mecanismo de identificación global único, el *Hypertext Transfer Protocol* (HTTP) como mecanismo de acceso universal y el *Hypertext Markup Language* (HTML) como formato de contenido.

El desarrollo y uso de estándares permite a la Web trascender sobre distintas arquitecturas técnicas. Los *hyperlinks* son utilizados para navegar entre distintos servidores, y también permiten a los motores de búsqueda recorrer la *Web* y proveer capacidades de búsqueda sofisticados sobre el contenido recorrido. Por lo tanto los *hyperlinks* son cruciales para conectar contenido proveniente de distintos servidores en un único lugar de información global.

A su vez, *Linked Data* asegura que es importante establecer estándares sobre los formatos de contenido de la *Web*, de forma que sea posible procesarla por un conjunto amplio de aplicaciones. Al publicar *Linked Data* en la *Web*, dichos datos son representados utilizando RDF, ya que RDF provee un modelo de datos simple por un lado e inclinado hacia la arquitectura *Web* por otro.

2.4.1. Diseño de URIs para Linked Data

Las URIs proveen una manera simple de identificar recursos. Como se comentó en la Sección 2.4 uno de los principios de Linked Data es que los conceptos deben ser nombrados con URIs. A la hora de construir URIs hay un conjunto de recomendaciones a seguir [5] :

1. Simplicidad: la URI debe ser corta y nemotécnica, de manera que pueda ser fácil de recordar.

2. Estabilidad: Una vez que se especifica un identificador a cierto recurso, este debe identificar a dicho recurso el mayor tiempo posible. Para lograr se debe aislar la definición de la URI de la implementación. Por ejemplo, si el identificador de mi recurso contiene una extensión ".jsp", al tiempo de cambiar el servidor a php debemos reestructurar el mismo.

3. Manejabilidad: Las URIs construidas deben ser administrables. Una buena practica para esto es incluir el año en las URIs para que futuros cambios en el esquema no causen problemas en URIs antiguas.

2.5. RDF

La *World Wide Web Consortium* (W3C) ha establecido una serie de estándares también conocidos como estándares para la Web Semántica (WS), los cuales permiten compartir información en este formato de datos abiertos. Entre estos estándares se destaca el Resource Description Framework [24] (RDF), que permite describir la estructura y el significado de los datos por medio de ontologías o vocabularios, y el lenguaje de consultas SPARQL, y esos serán los elegidos para la publicación y consulta de datos de este proyecto.

El modelo de datos RDF es similar a los enfoques de modelado conceptual clásicos como entidad-relación o diagramas de clases, ya que se basa en la idea de hacer declaraciones sobre los recursos (en particular, recursos web) en forma de expresiones sujeto-predicado-objeto. Estas expresiones son conocidos como triplas. El sujeto indica el recurso y el predicado denota rasgos o aspectos del recurso y expresa una relación entre el sujeto y el objeto. Además, cada uno de estos recursos estará identificado con un identificador de recursos internacional (IRI). Además, un objeto también puede ser un valor, denominado Literal en RDF, o un nodo blanco, típicamente utilizado para representar recursos anónimos.

Un conjunto de triplas corresponde a un grafo dirigido cuyos nodos son objetos o sujetos y cuyas aristas representan predicados. Las IRIs son comúnmente escritas como una

etiqueta de prefijo y una parte local, separadas por dos puntos (':'). Estas se transforman en IRIs al concatenar la IRI asociada al prefijo y a la parte local. Una colección de palabras reservadas definidas en el RDF Schema [24] (llamadas el vocabulario RDF-s) son utilizadas para definir clases, propiedades y relaciones jerárquicas. Por ejemplo, la tripla (r, rdf:type, c) representa explícitamente que r es una instancia de c, y además implícitamente implica que el objeto c es una instancia de rdfs:Class.

La Figura 2.8 muestra una representación gráfica para grafos RDF, donde las elipses representan recursos identificados por IRIs, las aristas etiquetadas representan predicados y los rectángulos representan literales.

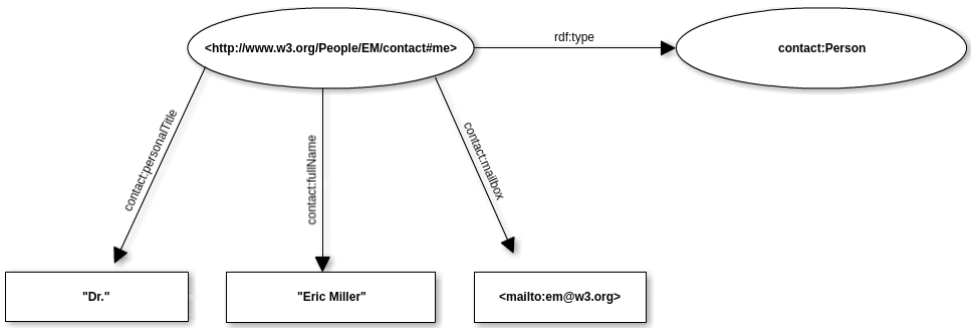


Figura 2.8: Ejemplo de grafo RDF.

Existen a su vez distintas maneras de representar estructuras en RDF en formato de texto, entre los que se destacan el lenguaje XML y la notación Turtle, siendo esta última la elegida para este proyecto.

2.5.1. Notación Turtle

La notación Turtle [25] define una sintaxis textual para representar modelos RDF, que permite escribir en texto natural y compacto cualquier grafo RDF, con abreviaciones para patrones y tipos de datos comunes. Turtle describe una forma de representar triplas (sujeto, predicado, objeto), separando cada término con un espacio en blanco y con un '.' al finalizada la tripla. Luego, para representar múltiples predicados y objetos para un mismo sujeto utiliza un ';', de manera que evita la repetición del predicado. El predicado especial rdf:type puede ser abreviado con 'a'.

A continuación se presenta un ejemplo de esta notación, y representa el mismo modelo que el grafo de la Figura 2.8, describiendo un contacto con nombre completo (*fullName*), dirección de correo (*mailbox*) y título personal (*personalTitle*).

```
<http://www.w3.org/People/EM/contact#me>
    rdf:type contact:Person;
  contact:fullName "Eric Miller";
  contact:mailbox <mailto:em@w3.org>;
  contact:personalTitle "Dr.".
```

Listado 1: Ejemplo de grafo RDF modelado en Turtle [20]

2.5.2. SPARQL

Dado que RDF es un formato de datos modelado en forma de grafo dirigido y etiquetado, SPARQL 1.1 [21] define una sintaxis para realizar consultas sobre dichos grafos RDF.

Su mecanismo de evaluación está basado en la correspondencia entre subgrafos, donde las triplas RDF se interpretan como nodos y aristas, de forma que el grafo utilizado para realizar la consulta es comparado con los grafos almacenados.
Los criterios de selección son expresados mediante la cláusula *WHERE* como patrones de grafos (Basic Graph Patterns [19]). SPARQL soporta funciones de agregación y agrupación, como el *GROUP BY*, que son necesarias para realizar las operaciones OLAP.

2.6. QB y QB4OLAP

La recomendación de la W3C para publicar datos estadísticos y *metadata* en RDF, siguiendo los principios de Linked Data, es el *RDF Data Cube Vocabulary* [22] (QB). Sin embargo, aunque es apropiado para representar y publicar datos estadísticos, éste no incluye algunos aspectos del modelo multidimensional que son imprescindibles para realizar análisis tipo OLAP como son las jerarquías de dimensión, los atributos en los niveles de dimensión, y las funciones de agregación para resumir valores de medidas. Para permitir este tipo de análisis sobre cubos en la Web Semántica fue creado el vocabulario QB4OLAP [18], que extiende al mencionado QB.

Las extensiones que presenta QB4OLAP en relación a QB son:

- Estructura de dimensión: la estructura de una dimensión está definida en términos de niveles, que componen jerarquías. La estructura de cada nivel está definida en términos de un conjunto de atributos.

- Instancias de dimensión: las instancias de dimensión son llamadas miembros de nivel, y existe una relación entre miembros de nivel de niveles consecutivos. En OLAP estas

relaciones son llamadas relaciones de *roll up*. En QB4OLAP estas relaciones entre miembros de nivel son modeladas utilizando propiedades de *roll up*. Además, los miembros de nivel están asociados a valores para cada atributo.

- Funciones de agregación: las funciones de agregación son utilizadas para computar agregaciones de valores de las medidas en el uso de operaciones OLAP (por ejemplo, *roll up*). QB4OLAP permite representar funciones de agregación y asociarlas con las medidas de un cubo.

Para representar un cubo en QB4OLAP se necesitan dos conjuntos de triplas RDF, un conjunto que represente el esquema del cubo y otro que represente las instancias del mismo. El primer conjunto es quien define la estructura del cubo, en términos de jerarquías, dimensiones, niveles y medidas; dicha información es necesaria para la construcción de consultas SPARQL que realizan las operaciones OLAP.
Por otro lado, las instancias de dicho cubo se almacenan en un otro conjunto de triplas que representan los miembros de los niveles, hechos y valores para las medidas definidas en el otro conjunto.

La Figura 2.9 describe el vocabulario utilizado en la ontología QB4OLAP, en los que se destacan los elementos qb:DimensionProperty (dimensiones), qb4o:LevelProperty (niveles), qb4o:LevelAttribute (atributos de nivel), qb4o:cardinality (cardinalidad entre entidades), qb4o:Hierarchy (Jerarquías), qb4o:aggregateFunction (funciones de agregación). Cabe destacar que los términos cuyos prefijos son 'qb:' son componentes definidos en el vocabulario QB, y aquellos con el prefijo 'qb4o:' son las extensiones presentadas por QB4OLAP. En mayúscula están representadas las clases RDF y en minúscula las propiedades RDF. Los términos en mayúscula y cursiva representan clases 'abstractas'.

Un *Data Structure Definition* (DSD) determina el esquema para un conjunto de datos de la clase qb:DataSet. Dicho DSD puede ser compartido entre distintos conjuntos de datos, y en él se definen propiedades (qb:componentProperty) para representar dimensiones (qb:dimension), medidas (qb:measure) y atributos (qb:attribute).
Los hechos representan puntos en el espacio multidimensional, y están vinculados a un valor en cada una de las dimensiones definidas en el DSD, utilizando instancias de la clase qb:DimensionProperty.

Como fue mencionado, QB4OLAP introduce la capacidad de estructurar dimensiones compuestas por jerarquías y niveles. La clase qb4o:LevelProperty representa los niveles de una dimensión, y los miembros de nivel son instancias de la clase qb4o:Member.
Las jerarquías se definen mediante la propiedad qb4o:Hierarchy, y las relaciones con las

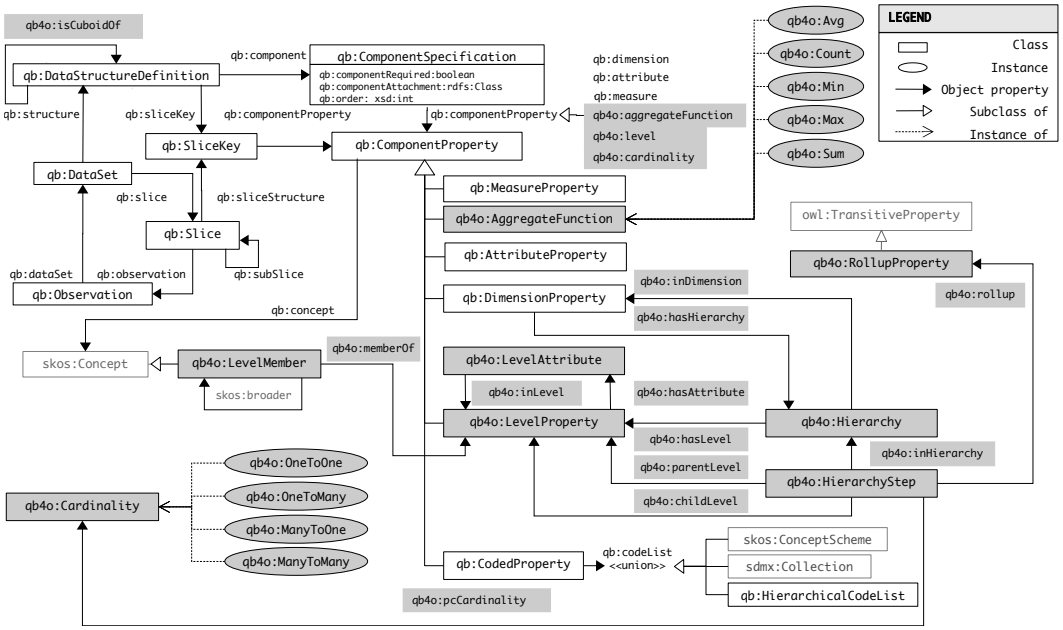


Figura 2.9: Vocabulario QB4OLAP [18]

dimensiones lo hacen mediante las propiedades qb4o:hasHierarchy y qb4o:inDimension, inversas entre sí.

La clase qb4o:LevelInHierarchy representa pares nivel-jerarquía, utilizando las propiedades qb4o:levelComponent para los niveles y qb4o:hierarchyComponent para las jerarquías. Además, la clase qb4o:HierarchyStep representa las relaciones padre-hijo entre pares de niveles de la jerarquía, mediante las propiedades qb4o:childLevel y qb4o:parentLevel; mientras que la cardinalidad de dichas relaciones se modelan mediante qb4o:cardinality, y esta propiedad utiliza miembros de la clase qb4o:Cardinality.

Luego, para modelar los atributos de niveles se utiliza la propiedad qb4o:hasAttribute, y las funciones de agregación son representadas mediante la clase qb4o:AggregateFunction; mientras que la relación entre medidas y funciones de agregación se representan con la propiedad qb4o:aggregateFunction.

En conclusión, QB4OLAP cuenta con los elementos necesarios para representar los cubos anteriormente mencionados y modelados en MultiDim como los mencionados en la Sección 2.2, y es la ontología que será utilizada para realizar la traducción de dichos modelos a un lenguaje interpretable por computadoras y más específicamente a RDF, lo cual será útil luego al momento de almacenar dichas estructuras en base de datos y a la hora de su exportación como datos abiertos.

3. Relevamiento de herramientas

A continuación se presentan un conjunto de herramientas que existen hoy en día y que están relacionadas con este proyecto en alguna de sus características. El objetivo de este capítulo es detallar cada una de estas herramientas, explicar qué funciones de las mismas son de nuestro interés y cómo se relacionan los requerimientos del presente proyecto.

3.1. Tableau

Tableau[16] es una empresa estadounidense especializada en la visualización de datos de forma gráfica para *business intelligence*. Cuenta con aplicaciones de escritorio, *web* y móvil, y permite en base a una conexión con un gran número tipos de bases de datos (relacionales, documentales, de grafos, cubos, etcétera) o mediante un archivo de entrada (CSV, JSON, por ejemplo), analizar datos multidimensionales dinámicamente mediante distintos tipos de gráficas. También permite al usuario modificar los distintos tipos de gráfica, los datos que serán parte de los ejes de la gráfica y la ubicación de los mismos (seleccionar medidas, filas y columnas de las gráficas), hacer agregación de datos sobre las distintas entradas y crear jerarquías entre ellos para luego poder realizar operaciones de cubos (roll up, drill across). Inclusive permite recibir más de un archivo de entrada y dinámicamente realizar operaciónes de unión o intersección sobre los mismos datos en tiempo real, o realizar visualizaciones de datos geolocalizados en forma de mapas.
También permite la exportación de la información graficada en formato de imagen o en archivo de excel.

Una de las funcionalidades que Tableau no permite es el agregado de información a una base de datos o un archivo de entrada dinámicamente, por lo que una vez que se obtiene la información de entrada se debe trabajar siempre sobre la misma o modificarla directamente desde la fuente. En ese orden, uno de los requisitos de la plataforma a desarrollar será permitir al usuario ingresar datos contínuamente para poder luego visualizarlos gráficamente y de manera dinámica.

Además, Tableau parte de una fuente de datos creada previamente y la utiliza como parámetro de entrada, pero no existe la posibilidad de crear nuevos cubos o indicadores dentro de la misma plataforma, con sus esquemas y estructuras, algo que también forma parte de los objetivos de la contraparte.

Por otra parte Tableau permite la exportación de las gráficas en formatos de imagen

y de los datos de entrada, por lo que si un usuario quisiera exportar los datos en RDF no sería posible. Además, la exportación es únicamente de los datos y no del esquema de los mismos, lo que implica que para poder entender la información exportada se requiere de conocimiento previo sobre la misma. Esto dificulta que la exportación de datos sea útil para ser compartida como datos abiertos en formato *linked data*, ya que dicha información no es interpretable automáticamente por una computadora.

A su vez, no es posible en Tableau graficar en forma simultánea (en una misma gráfica) datos obtenidos de bases de datos distintas. Es decir, no permite realizar cruzamiento de datos de distintas entradas, ya que la manera que tiene de combinar distintas entradas es seleccionando las dimensiones compartidas por ambas (en excel, por ejemplo, serían las columnas en común) y en base a eso generar una nueva base de datos única, que no permite distinguir de qué entrada proviene cada dato. Lo que sí permite es graficar por separado distintas bases de datos y mostrarlas en forma simultánea una al lado de la otra.

En conclusión, Tableau puede considerarse como una herramienta de visualización de indicadores, que soporta variedad de tipos de datos y tipos de visualización, de manera que coincide con varios de los objetivos de esta plataforma. Mientras que por otra parte, no cumple con otros de los objetivos propuestos como pueden ser el cruzamiento de indicadores. Y además, su uso está orientado al análisis de información proporcionada por el mismo usuario y no es posible realizar una publicación de la misma en formato de datos abiertos para el uso externo ella, otro de nuestros objetivos.

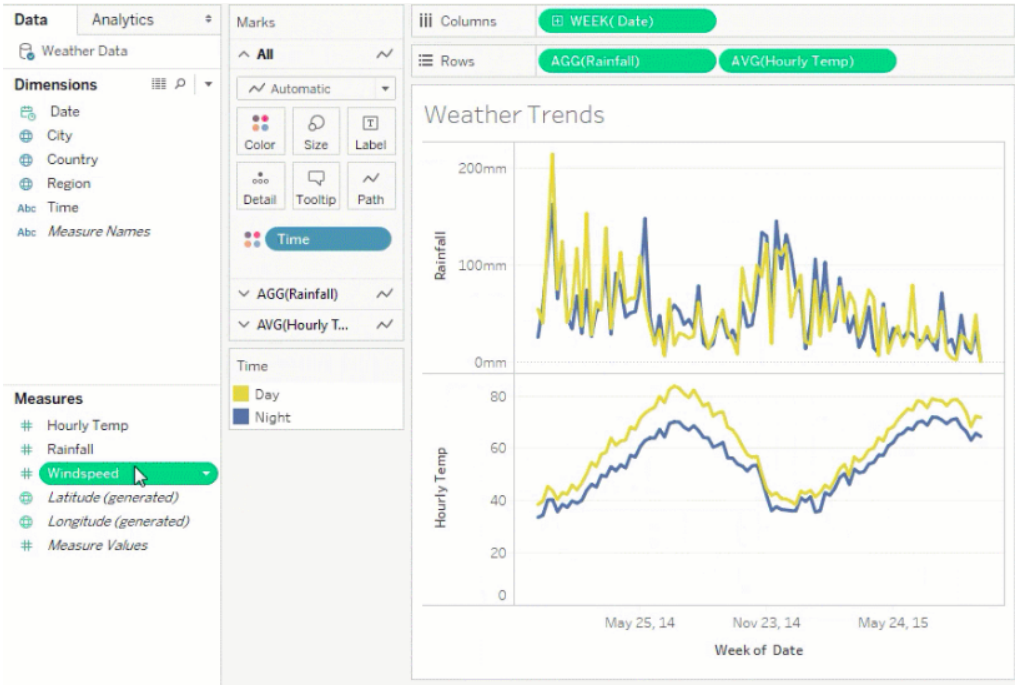


Figura 3.1: Ejemplo de aplicación Tableau [16]

3.2. Saiku

Saiku[12] es una herramienta *web* y *open source*, que tiene como principal propósito el proveer una interfaz gráfica para la lectura y análisis de complejos conjuntos de datos de forma *online* (OLAP). Además cuenta con distintos tipos de visualizaciones y gráficas, y provee al usuario la posibilidad de configurarse su propio panel de visualización de datos, por lo que su interfaz es personalizable.

Para esto permite la integración con distintas fuentes de datos, como pueden ser bases de datos relacionales y no-relacionales. Sin embargo, no es posible visualizar simultáneamente datos de dos fuentes distintas, al igual que en Tableau, por lo que no es posible realizar cruzamientos de indicadores provenientes de distintas fuentes de datos.

Además, la información almacenada en esta plataforma, a pesar de sus distintas formas de visualización, solo permite su exportación en formatos PDF o Excel, por lo que no cumple con los principios de datos abiertos para *Linked Data*.

Por último, al igual que Tableau, las fuentes de datos son requisitos previos al uso de la herramienta en cuestión, debido a que no cuentan con la funcionalidad de crear nuevos esquemas dentro de la plataforma, por lo que es necesario contar con las fuentes necesarias (esquemas y hechos, traducidos a cualquiera de los tipos de fuentes posibles) para poder hacer uso de la plataforma.

Figura 3.2: Ejemplo de aplicación Saiku [12]

3.3. CubesViewer

CubesViewer [8] es una aplicación *web* y biblioteca *open source*, para la visualización y exploración de distintos tipos de conjuntos de datos. En particular, se caracteriza por su capacidad para representar cubos multidimensionales en forma gráfica, contando con distintos tipos de gráfica y permitiendo al usuario realizar operaciones OLAP sobre los datos, en particular permite realizar *drilldowns* y *roll up*, y funciones de filtrado sobre cualquiera de sus dimensiones y en forma dinámica. Además, tiene la capacidad de que dado un hecho con múltiples medidas, se pueda intercambiar la visualización entre sus distintas medidas también de forma dinámica.

También provee un modo de visualización de los datos en formato de tabla, pudiendo además realizar las mismas funciones de agregación sobre los datos de la tabla obteniendo así distintas representaciones de los datos.

Por último tiene también la posibilidad de exportar los datos en el formato en que están siendo visualizados. Esto es, dado un cubo visualizado en forma de gráfica, permite exportar dicha gráfica en formato de imagen PNG, mientras que si un cubo está siendo visualizado en formato de tabla, permite exportar dicha tabla en formato CSV. Además, permite la exportación de los datos crudos (hechos), también en formato CSV. Sin embargo, dicha exportación de información no cumple con los principios de datos abiertos *Linked Data*, ya que tanto las imágenes como los CSV no cumplen con dichos principios. Por otra parte, tampoco es posible mediante *CubesViewer* visualizar dos cubos en forma simultánea, ya que no permite realizar gráficas desde distintas fuentes de datos, y por lo tanto, no es posible realizar el cruzamiento de indicadores.

Al igual que en las herramientas mencionadas previamente, *CubesViewer* tampoco provee la funcionalidad de crear nuevos esquemas e indicadores dentro de la plataforma, ya que cuenta solamente con funciones de visualización, pero no de creación de esquema e ingreso de datos. Por lo tanto, también es requisito previo para su uso el contar con las fuentes de datos necesarias su posterior visualización.

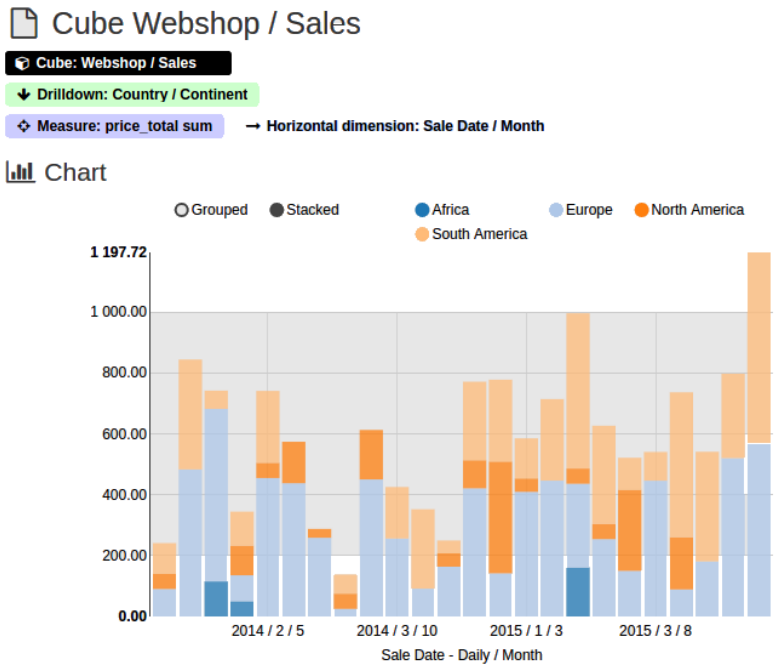


Figura 3.3: Ejemplo de aplicación CubesViewer [8]

3.4. Visualizador de cubos de Eurostat

Eurostat [9] es la oficina de estadísticas de la Unión Europea ubicada en Luxemburgo. Tiene como misión proveer estadísticas de alta calidad sobre Europa y accesible a todo público. Cuenta con una sección de publicaciones que presenta información sobre toda Europa en temas como economías y finanzas, condiciones sociales y población, agricultura y pesca, etcétera.

Para cada una de estas secciones cuenta con un conjunto de datos publicados como pueden ser migraciones por edad, sexo y país de nacimiento, o desempleo por sexo y edad, y para cada uno de estos sets de datos cuenta con la posibilidad de visualizarlos en formato de filas y columnas, permitiendo realizar filtros sobre cada una de las variables involucradas. Sin embargo, no permite la visualización en ningún formato gráfico.

Además, la información relacionada a cada uno de los indicadores está disponible como datos abiertos, provee un documento en formato XML que representa la *metadata* de cada uno de estos indicadores y dicho documento está disponible para ser descargado. Pero por otro lado, los datos correspondientes a dicha *metadata* solo está disponible a ser descargada en formato CSV.

En resumen, Eurostat provee una herramienta de visualización de indicadores, y aunque no es un producto abierto a la utilización de usuarios externos a la misma, fue útil el estudio de la misma ya que sirvió de ejemplo de visualización de indicadores con distintas

dimensiones y que permite la reagrupación de los datos de acuerdo a distintos tipos de granularidad. Además, también es un ejemplo de la exportación en formato de datos abiertos para el uso abierto de los mismos, aunque como pudimos ver no cumple con los mejores estándares de datos abiertos como lo son los *Linked Data*.

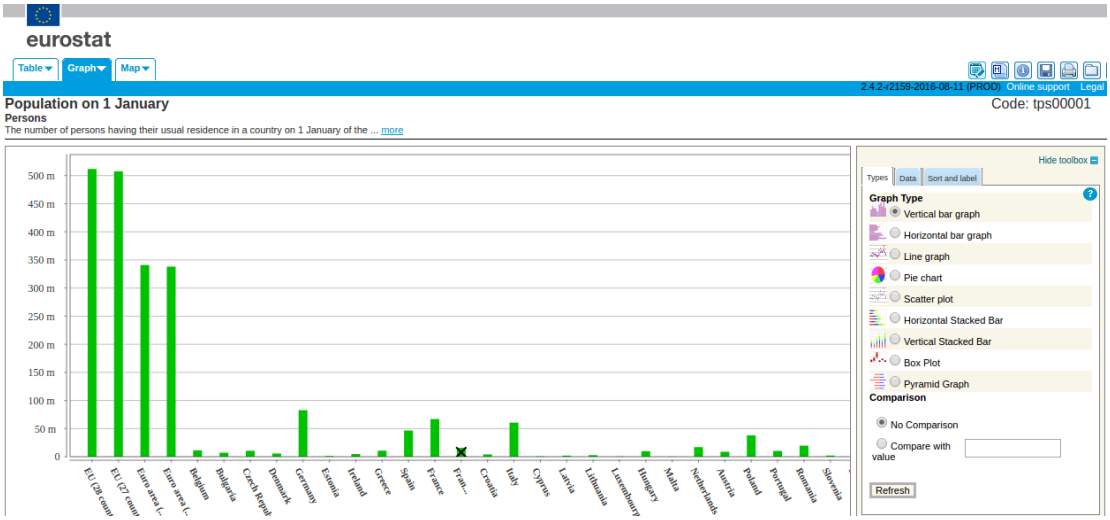


Figura 3.4: Ejemplo de aplicación Eurostat [9]

3.5. Conclusión

El Cuadro 1 ilustra la comparación entre cada una de las herramientas relevadas, haciendo hincapié en las características de cada una de ellas que nos interesan particularmente, de acuerdo a los objetivos planteados para este proyecto.

Producto / Característica	*Open Source*	Gratuito	Creación de indicadores	Operaciones OLAP	Cruzamiento de indicadores	*Linked Data*
Tableau	No	No	No	Si	No	No
Saiku	Sí	Sí	No	Si	No	No
CubesViewer	Sí	Sí	No	Si	No	No
Eurostat	No	Sí	No	Si	No	No

Cuadro 1: Comparación entre herramientas

Podemos concluir de acuerdo a la comparativa que se observa en el cuadro, que ninguna de las herramientas relevadas cumple con todos los requisitos planteados en la Sección 1.2 como necesarios para la contraparte.

Uno de las principales necesidades de la contraparte recae en la posibilidad de crear nuevos indicadores de forma contínua y dentro de la misma plataforma para poder luego ingresar datos relacionados a los mismos, y como pudimos ver, ninguna de las herramientas relevadas cuenta con funcionalidades para el ingreso tanto de esquemas como de datos, ya que todas permiten el ingreso de datos a partir de una fuente completa de los mismos, donde el esquema se deduzca a partir de los datos y donde tampoco es posible realizar modificaciones sobre dichos datos de forma dinámica, dentro de la misma herramienta.

Por otro lado, ninguna de las herramientas está orientada a la exportación de datos de calidad, ya sea como *Linked Data* o en algún formato adecuado. En general, se cuenta con funcionalidades de exportación de los datos crudos y generalmente interpretables por el ojo humano, ya sea mediante una imagen o un archivo CSV, pero no son formatos interpretables por computadoras, ya que no existe especificación ninguna sobre la *metadata* de la información exportada (cuáles son las dimensiones de un cubo, qué niveles hay dentro de dichas dimensiones y qué atributos tiene cada nivel, etcétera), y por lo tanto no hay manera de asegurar su correcta interpretación.

Además, también puede verse que ninguna cuenta con la funcionalidad de cruzamiento de cubos, es decir, todas proveen distintos tipos de visualización para un conjunto de datos de determinada fuente, pero no es posible realizar una relación y/o compatibilización entre distintas fuentes de datos, de manera de poder visualizar y analizar los datos de ambas fuentes en forma simultánea, tal y como fue presentado uno de los objetivos de este proyecto.

Cabe destacar que uno de los posibles abordajes de este proyecto pudo haber sido la utilización de alguna de estas herramientas *open source* (en particular, Saiku o CubesViewer) como base de la plataforma e implementar una solución que extienda la misma de acuerdo a las necesidades de la contraparte, pero fue una de las decisiones tomadas en etapas tempranas del proyecto que lo más conveniente era la implementación de una plataforma desde el inicio y en todos sus componentes, debido a que creímos que todas las funcionalidades requeridas eran factibles de implementar eligiendo las tecnologías correctas

y en un plazo razonable, mientras que por otro camino hubiera sido un riesgo grande el suponer que alguna de las herramientas mencionadas pudiera llegado el caso soportar las extensiones necesarias, algo que además llevaría un costo de investigación en profundidad de cada una de ellas y con un resultado final incierto.

4. Diseño de la solución

En esta sección se ve en detalle las distintas etapas de diseño de la solución. Primero, se presentan los indicadores y su modelado como cubos. Luego, y de acuerdo a los objetivos planteados, se presenta el diseño de la plataforma web que permite crear, gestionar, visualizar y exportar indicadores.

4.1. Modelado de indicadores

Se cuenta con los siguientes indicadores y sus correspondientes modelos en MultiDim:

- Solicitudes de eventos registradas según último año de evaluación

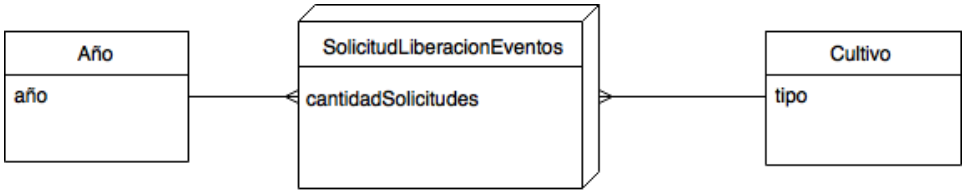


Figura 4.1: Solicitudes de liberación de eventos

- Patentes biotecnológicas registradas en Uruguay

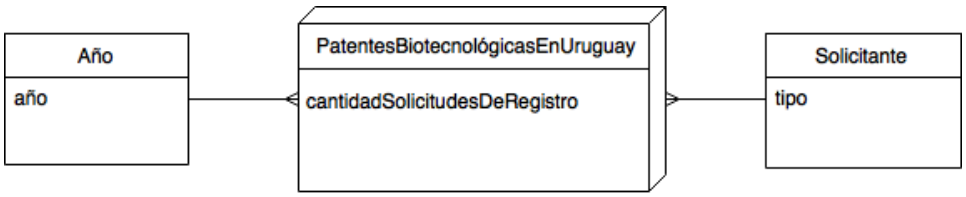


Figura 4.2: Patentes biotecnológicas registradas en Uruguay

- Gasto en I+D en Ciencias Naturales y Exactas y en Ciencias Agrícolas como porcentaje del gasto en I+D total

Figura 4.3: Gasto en I+D

- Área sembrada por cultivo en Uruguay

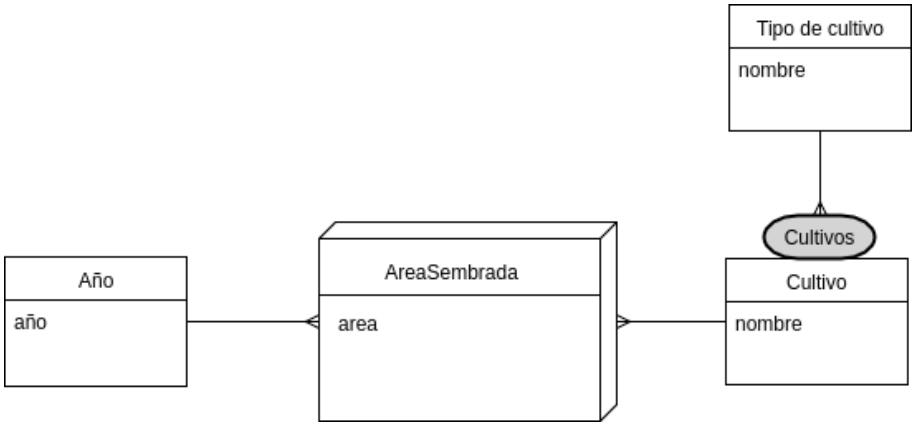


Figura 4.4: Área sembrada con maíz y soja

- Importaciones de fitosanitarios (kg de sustancia activa) por tipo de agroquímico (Herbicidas, Fungicidas, Insecticidas, Otros, Materias primas)

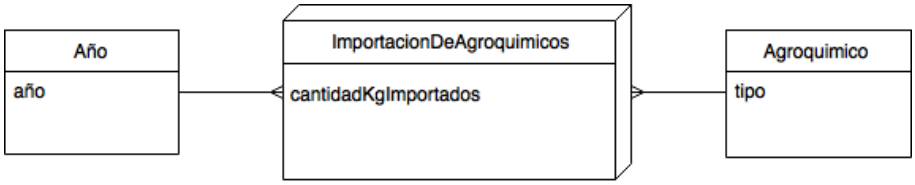


Figura 4.5: Importaciones de fitosanitarios por tipo de agroquímico

- Importaciones de fertilizantes en miles de toneladas

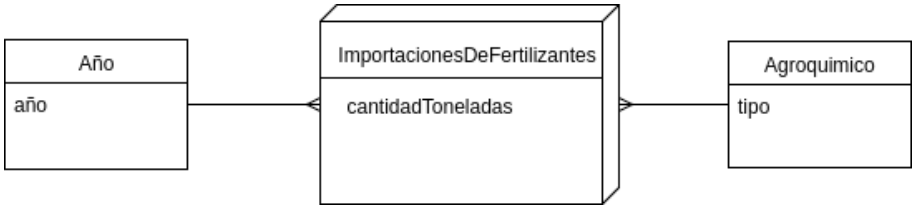


Figura 4.6: Importaciones de fertilizantes en miles de toneladas

- Margen de ganancia y rendimiento mínimo (rendimiento a partir del cual se empieza a obtener ganancia)

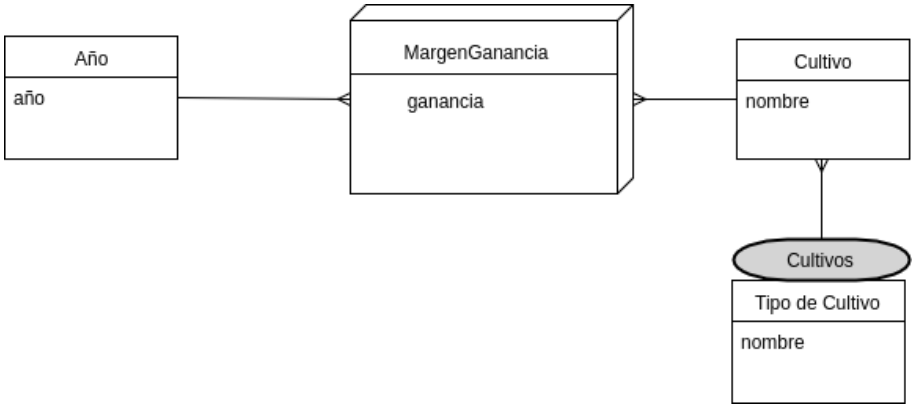


Figura 4.7: Márgen de ganancia

- Número de productores de cultivos cerealeros e industriales (trigo, cebada cervecera, maíz, sorgo, girasol y soja) y superficie de predio promedio según tamaño de predio (hectáreas).

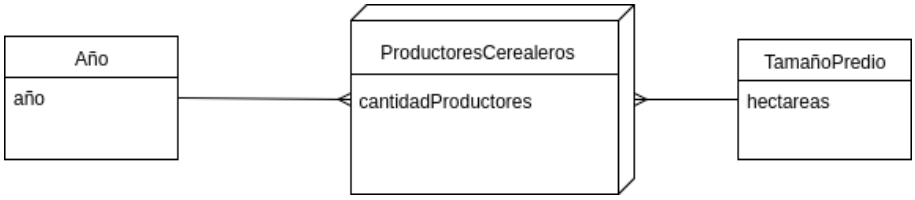


Figura 4.8: Número de productores de cultivos cerealeros e industriales

- Área sembrada con soja según tamaño de predio (en miles de hectáreas)

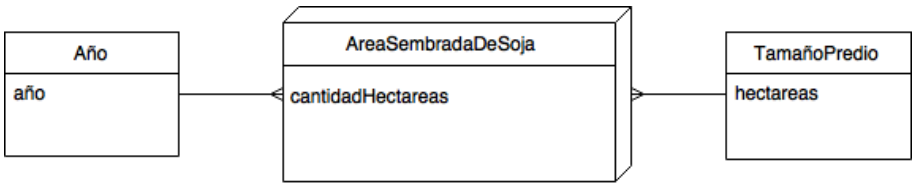


Figura 4.9: Área sembrada con soja según tamaño de predio

- Medidas no arancelarias técnicas iniciadas por los principales socios comerciales de Uruguay y Uruguay

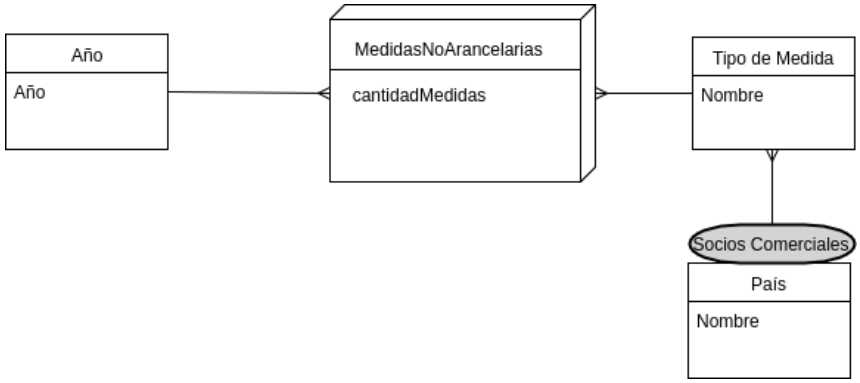


Figura 4.10: Medidas no arancelarias técnicas

4.2. Diseño de la plataforma

En esta sección se ve en detalle las distintas etapas de diseño de la solución. De acuerdo a los objetivos planteados, se diseña una plataforma en la cual los usuarios son capaces de:

- Crear, editar y eliminar esquemas de indicadores.
- Ingresar y eliminar conjuntos de datos para un indicador.
- Visualizar de forma gráfica los conjuntos de datos ingresados para un indicador dado.
- Elegir un par de indicadores para visualizarlos en forma simultánea y conjunta.
- Descargar como datos abiertos la información almacenada (esquema y datos) sobre cada indicador.

Para cumplir con las funcionalidades planteadas, la plataforma diseñada consiste en una aplicación *web*, accesible desde un navegador, en la cual los usuarios pueden ingresar esquemas de indicadores de su interés, siendo cada uno de ellos independientes entre sí, lo que implica que cada estructura junto con sus componentes deben ser ingresados por parte del usuario. Además, para cada uno de los indicadores ingresados debe existir la posibilidad de ingresarle datos, para que dichos datos puedan ser luego visualizados mediante gráficas también dentro de la aplicación.

Por otra parte, toda la información ingresada por los usuarios, tanto los esquemas de indicadores como sus datos, deben estar almacenados en una base de datos posteriormente. Para esto es necesario contar con un componente de *backend* que se encargue de recibir la información ingresada por los usuarios, interpretarla, traducirla y/o reestructurarla de tal manera que dicha información pueda luego ser insertada correctamente en la base de

datos.

Posteriormente, el componente de *backend* debe también realizar el proceso inverso a la inserción de datos, es decir, debe ser capaz de obtener la información requerida por el usuario desde la base de datos, y traducirla de tal manera que sea enviada al componente de *frontend* para que éste último la procese y realice las operaciones necesarias para mostrarla al usuario.

Por último, se debe contar también con la ya mencionada base de datos, que debe ser capaz de almacenar esquemas de indicadores con estructuras en principio diferentes y desconocidas de antemano. Estas estructuras heterogéneas deberán almacenarse de forma dinámica y a medida que el usuario lo requiera. Además de estos esquemas, también debe almacenar conjuntos de datos asociados a los mismos. Hay que destacar que además de ser capaz de almacenar los tipos de datos mencionados, es necesario contar con un tipo de almacenamiento que sea útil para la exportación de la información como datos abiertos. Es decir que se debe diseñar una base de datos de tal manera que permita realizar la exportación de la información almacenada, con el menor esfuerzo posible, en formato de datos abiertos como *Linked Data*, más precisamente utilizando QB4OLAP.

En la Figura 4.1 se presenta un diseño a alto nivel de la solución utilizada para cumplir con las funcionalidades del sistema. Se dedica una sección para detallar cada una de las capas del sistema presentadas en la figura 4.1 cumpliendo con la siguiente estructura, en la sección 4.3 se detalla la responsabilidad de cada uno de los grafos de la capa de almacenamiento, en la sección 4.4 los componentes del *backend* y en la sección 4.5 los componentes de *frontend*.

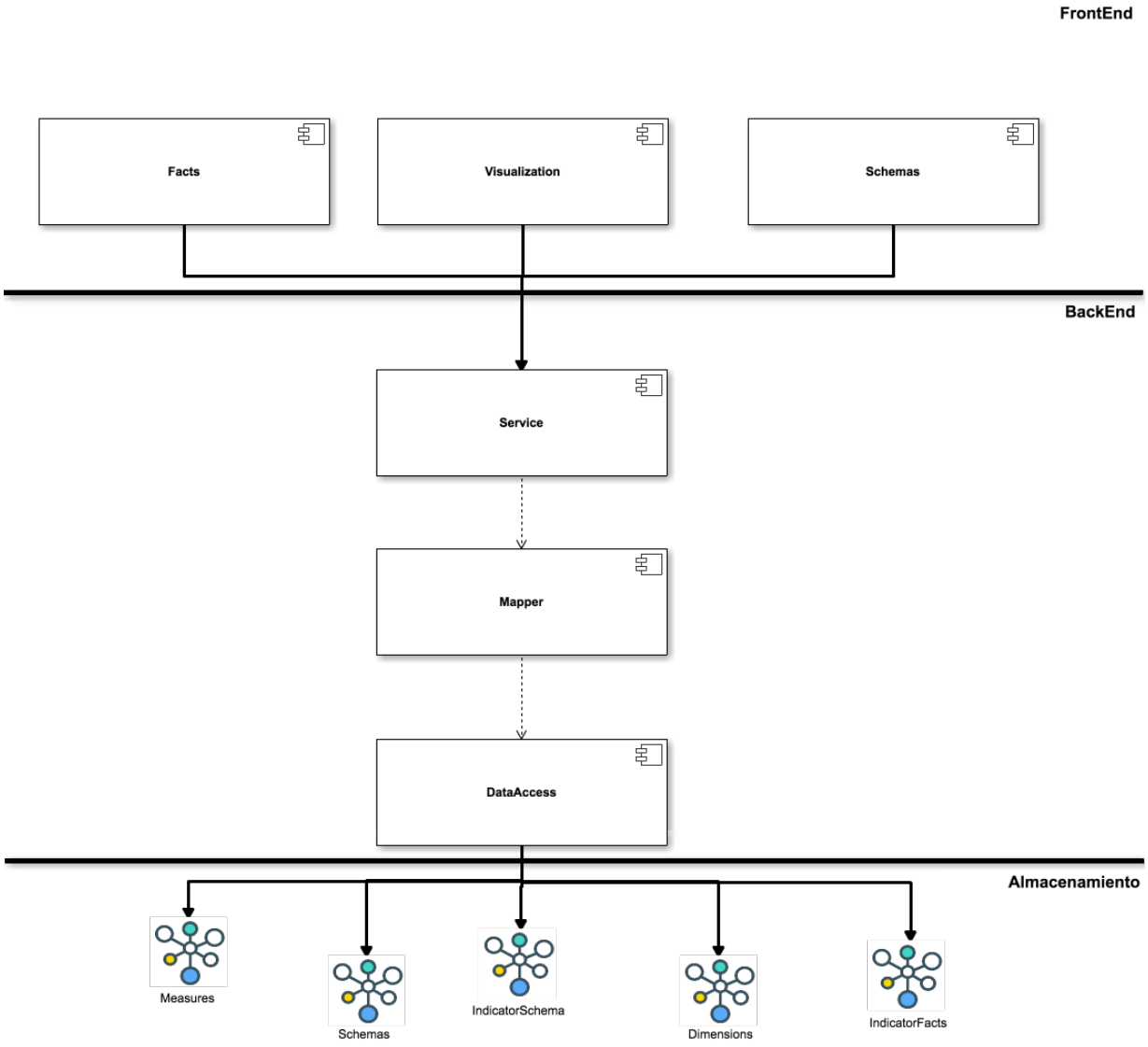


Figura 4.11: Componentes del sistema

4.3. Almacenamiento

En esta sección se comienza presentando los distintos factores que influyeron al momento de diseñar la base de datos multidimensional para luego adentrarnos en el modelo específico propuesto como solución en nuestro sistema. Uno de los factores a tener en cuenta es que se deben utilizar grafos para almacenar los cubos multidimensionales ya que es uno de los requisitos del proyecto. Los grafos están compuestos por aristas y nodos que son representadas a través de triplas <Sujeto, Predicado, Objeto>. Esto conlleva a tener que decidir en etapa de diseño la cantidad de grafos existentes en el sistema, sus responsabilidades y la forma en la cual se representan las triplas. A lo largo de esta sección se resuelven estos puntos.

Por otra parte, uno de los objetivos del proyecto consiste en publicar toda su información como datos abiertos integrando el nivel 5 de la clasificación de 5 estrellas propuesto por Tim Berners-Lee explicada en el Capítulo 3. Este es otro factor a tener en cuenta al momento de diseñar la base de datos debido a que existe un conjuntos de estándares planteados en *Linked Data* que deben estar presentes al momento de realizar el modelado.

El tercer factor que se hace presente son los requerimientos funcionales que debe cumplir el sistema. En este punto, al igual que con el resto de los modelos de bases de datos se debe tener en cuenta el tipo de sistema que se desea modelar, suelen plantearse distintas soluciones cuando se tiene un sistema cuyos requerimientos están fuertemente orientados al análisis de datos que cuando se tienen un sistema cuyo foco es realizar transacciones en la base.

Hasta el momento se nombraron los tres factores fundamentales que guiaron el proceso de diseñar la base de datos con el fin de brindar una introducción a las decisiones tomadas en la definición del modelo de datos. Con el propósito de cumplir con los principios de Linked Data y crear una base de datos acorde a los requerimientos del sistema, se realizan dos etapas de diseño, una enfocada al almacenamiento físico de los datos explicado en la sección 4.3.1 y otra enfocada en que la creación dinámica de URIs cumpla con los estándares explicada en la sección 4.3.2.

4.3.1. Diseño de base de datos

El diseño de base de datos se basa fuertemente en el tipo de sistema que se está construyendo, el cual tiene como propósito permitir a los usuarios crear indicadores dinámicamente y tener la posibilidad de visualizarlos de forma gráfica, así como también, permitir el cruzamiento entre distintos indicadores para tener una representación simultánea que aporte valor al usuario. Pensando a nivel de base de datos, dichas funcionalidades se traducen a la necesidad de crear cubos multidimensionales de forma dinámica para luego realizar consultas sobre los mismos. Los cubos multidimensionales se representan a través de grafos, esto implica analizar cuántos grafos van a existir en la base de datos y cómo se van a representar las triplas utilizadas para generarlos. Sobre este último punto no se debe tomar ninguna decisión debido a que uno de los requisitos del proyecto es utilizar la ontología QB4OLAP que define como tienen que estar compuestas las triplas para representar cubos multidimensionales. Para definir cuántos grafos son necesarios se tienen en cuenta distintos factores. Por un lado, se debe almacenar la metadata de los indicadores creados en el sistema para lo cual se consideran dos opciones, almacenarlos todos en un

mismo grafo o tener un grafo independiente para almacenar la metadata de cada uno de los indicadores. La primer opción se descarta considerando que el rendimiento del sistema se puede ver afectado al aumentar la cantidad de indicadores teniendo en cuenta que los usuarios solamente van a necesitar acceder a la metadata de un solo indicador a la vez o a lo sumo dos en el caso de cruzamiento. Por lo tanto, la opción elegida es tener un grafo independiente para almacenar la metadata de cada uno de los indicadores, se presenta un ejemplo en el Listado 2. Otra decisión a tomar es cómo almacenar los hechos de cada indicador, para lo cual, siguiendo el mismo razonamiento se decide almacenar los hechos de cada indicador en un grafo independiente en lugar de tenerlos todos en el mismo grafo, se puede ver un ejemplo en el Listado 3. Por otro lado, uno de los requerimientos del sistema es que se puedan elegir dos indicadores que compartan al menos una dimensión para realizar cruzamiento entre ellos. Para determinar de forma simple y eficiente si dos indicadores comparten una dimensión se decide tener un grafo que contenga información sobre la estructura de todos los indicadores del sistema, en la figura 4 se puede ver un ejemplo. Por último, el usuario desea que todas las dimensiones, medidas e instancias creadas en el sistema aparezcan disponibles para reutilizar entre distintos indicadores. Para contemplar este caso de forma eficiente se decide tener un grafo que contenga todas las medidas del sistema y otro grafo con todas las dimensiones e instancias, estas dos últimas se almacenan juntas ya que siempre se necesitan de forma conjunta. Dichos grafos se ejemplifican en los listados 5 y 6 respectivamente. Concluyendo, la base de datos está compuesta por cinco tipos de grafos presentados a continuación.

- *IndicatorSchema*: Contiene la metadata de un indicador, existen tantos grafos de este tipo como indicadores existan en el sistema. El Listado 2 presenta un ejemplo de este grafo en base al indicador de área sembrada por tipo de cultivo y año(IAS). Entre las líneas 1 y 5 se puede ver los prefijos asignados a algunas URIs. Entre las líneas 7 y 16 se define la estructura del indicador que contiene nombre, descripción, medida y los niveles base pertenecientes a la dimensiones del indicador. Entre las líneas 18 y 22 se puede ver información sobre la medida del indicador la cual en este ejemplo es 'Área Sembrada'. Se recuerda que una dimensión pertenece a una jerarquía y contiene niveles y atributos, entre las líneas 24 y 46 se puede apreciar la información correspondiente a la dimensión 'Año' del indicador y entre las líneas 48 y 89 se puede ver la información correspondiente a la dimensión 'Cultivos', la cual, al estar compuesta por dos niveles, contiene un paso de jerarquía definido entre las líneas 60 y 65 que indica cual es el nivel padre y cual es el hijo en la jerarquía.

```
@prefix qb:     <http://purl.org/linked-data/cube#> .
@prefix qb4o:   <http://purl.org/qb4olap/cubes#> .
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dc:     <http://purl.org/dc/terms/> .

<http://fing.edu.uy/sigmu/20170816/schemas#Areasembrada>
    a               qb:DataStructureDefinition ;
    rdfs:label      "Area sembrada" ;
    dc:description  "Area sembrada" ;
    qb:component    [ qb4o:cardinality  qb4o:ManyToOne ;
        qb4o:level <http://fing.edu.uy/sigmu/20170816/Cultivos/levels#Cultivo>] ;
    qb:component    [ qb4o:cardinality  qb4o:ManyToOne ;
        qb4o:level <http://fing.edu.uy/sigmu/20170816/Anio/levels#Anio>] ;
    qb:component    [ qb:measure  <http://fing.edu.uy/sigmu/20170816/measures#Areasembrada> ;
        qb4o:aggregateFunction  qb4o:Sum] .

<http://fing.edu.uy/sigmu/20170816/measures#Areasembrada>
    a               qb:MeasureProperty ;
    rdfs:label      "Area sembrada" ;
    rdfs:range      xsd:integer ;
    dc:description  "Area sembrada en miles de hectareas" .

<http://fing.edu.uy/sigmu/20170816/dimensions#Anio>
    a                   qb:DimensionProperty ;
    rdfs:label          "Anio" ;
    dc:description      "Anio o Zafra" ;
    qb4o:hasHierarchy   <http://fing.edu.uy/sigmu/20170816/Anio/hierarchy#Hierarchy_Anio> .


<http://fing.edu.uy/sigmu/20170816/Anio/hierarchy#Hierarchy_Anio>
    a                   qb4o:Hierarchy ;
    qb4o:hasLevel       <http://fing.edu.uy/sigmu/20170816/Anio/levels#Anio> ;
    qb4o:inDimension    <http://fing.edu.uy/sigmu/20170816/dimensions#Anio> .

<http://fing.edu.uy/sigmu/20170816/Anio/levels#Anio>
    a                   qb4o:LevelProperty ;
    rdfs:label          "Anio" ;
    dc:description      "Anio o Zafra" ;
    qb4o:hasAttribute   <http://fing.edu.uy/sigmu/20170816/Anio/attributes#Anio> .

<http://fing.edu.uy/sigmu/20170816/Anio/attributes#Anio>
    a               qb4o:LevelAttribute ;
    rdfs:label      "Anio" ;
    rdfs:range      xsd:integer ;
    dc:description  "Anio o Zafra" .

<http://fing.edu.uy/sigmu/20170816/dimensions#Cultivos>
    a                   qb:DimensionProperty ;
    rdfs:label          "Cultivos" ;
    dc:description      "Cultivos" ;
    qb4o:hasHierarchy   <http://fing.edu.uy/sigmu/20170816/Cultivos/hierarchy#Cultivos> .

<http://fing.edu.uy/sigmu/20170816/Cultivos/hierarchy#Hierarchy_Cultivos>
    a                   qb4o:Hierarchy ;
    qb4o:hasLevel       <http://fing.edu.uy/sigmu/20170816/Cultivos/levels#Cultivo>,
            <http://fing.edu.uy/sigmu/20170816/Cultivos/levels#Tipodecultivo> ;
    qb4o:inDimension <http://fing.edu.uy/sigmu/20170816/dimensions#Cultivos> .

<http://fing.edu.uy/sigmu/20170816/Cultivos/hierarchy#Step_1>
    a                   qb4o:HierarchyStep ;
    qb4o:cardinality    qb4o:ManyToOne ;
    qb4o:childLevel     <http://fing.edu.uy/sigmu/20170816/Cultivos/levels#Cultivo> ;
    qb4o:inHierarchy    <http://fing.edu.uy/sigmu/20170816/Cultivos/hierarchy#Cultivos> ;
    qb4o:parentLevel    <http://fing.edu.uy/sigmu/20170816/Cultivos/levels#TipoDeCultivo> .

```

```
<http://fing.edu.uy/sigmu/20170816/Cultivos/levels#Cultivo>
    a                   qb4o:LevelProperty ;
    rdfs:label          "Cultivo" ;
    dc:description      "Cultivo" ;
    qb4o:hasAttribute   <http://fing.edu.uy/sigmu/20170816/Cultivo/attributes#Nombrecultivo> .

<http://fing.edu.uy/sigmu/20170816/Cultivo/attributes#Nombrecultivo>
    a                qb4o:LevelAttribute ;
    rdfs:label       "Nombre cultivo" ;
    rdfs:range       xsd:string ;
    dc:description   "Nombre" .

<http://fing.edu.uy/sigmu/20170816/Cultivos/levels#Tipodecultivo>
    a                   qb4o:LevelProperty ;
    rdfs:label          "Tipo de cultivo" ;
    dc:description      "Tipo de cultivo" ;
    qb4o:hasAttribute   <http://fing.edu.uy/sigmu/20170816/Tipodecultivo/attributes#Nombre> .

<http://fing.edu.uy/sigmu/20170816/Tipodecultivo/attributes#Nombre>
    a                qb4o:LevelAttribute ;
    rdfs:label       "Nombre" ;
    rdfs:range       xsd:string ;
    dc:description   "Nombre" .
```

Listado 2: Grafo de esquema del indicador

- *IndicatorFacts*: Contiene los hechos de un indicador, existen tantos grafos de este tipo como indicadores existan en el sistema. El Listado 3 presenta un ejemplo de este grafo basado en el indicador IAS. El hecho representa que en el año 2010 existieron 81000 hectáreas sembradas con trigo.

```
<http://fing.edu.uy/sigmu/20170816/Areasembrada/facts#Anio_2010_Trigo>
        <http://fing.edu.uy/sigmu/20170816/Anio/levels#Anio>
             <http://fing.edu.uy/sigmu/20170816/Anio/instances#Anio_2010>;
        <http://fing.edu.uy/sigmu/20170816/Cultivos/levels#Cultivo>
             <http://fing.edu.uy/sigmu/20170816/Cultivos/instances#Soja1ra> ;
        <http://fing.edu.uy/sigmu/20170816/measures#Areasembrada>
                81000 .
```

Listado 3: Grafo de hechos de un indicador

- *Schemas*: Contiene información sobre la estructura de todos los indicadores del sistema, existe un único grafo de este tipo. El Listado 4 presenta un ejemplo que contiene la estructura de dos indicadores, entre las líneas 1 y 10 se puede ver el indicador IAS y entre las líneas 12 y 21 el indicador Tenencia de la Tierra (ITT) que representa la cantidad de productores por año en una determinada categoría. En este ejemplo se puede apreciar que ambos indicadores son candidatos a realizar cruzamientro entre sí debido a que comparten la dimensión Anio.

```
1  <http://fing.edu.uy/sigmu/20170816/schemas#Areasembrada>
2      a                qb:DataStructureDefinition ;
3      rdfs:label       "Area sembrada" ;
4      dc:description   "Area sembrada" ;
5      qb:component     [ qb4o:cardinality  qb4o:ManyToOne ;
6         qb4o:level <http://fing.edu.uy/sigmu/20170816/Cultivos/levels#Tipodecultivo>] ;
```

```
7     qb:component     [ qb4o:cardinality  qb4o:ManyToOne ;
8       qb4o:level <http://fing.edu.uy/sigmu/20170816/Anio/levels#Anio>] ;
9     qb:component     [ qb:measure  <http://fing.edu.uy/sigmu/20170816/measures#Areasembrada> ;
10      qb4o:aggregateFunction  qb4o:Sum] .
11
12  <http://fing.edu.uy/sigmu/20170816/schemas#TenenciaDeLaTierra>
13    a               qb:DataStructureDefinition ;
14    rdfs:label      "Tenencia de la tierra" ;
15    dc:description  "Tenencia de la tierra" ;
16    qb:component    [ qb4o:cardinality  qb4o:ManyToOne ;
17      qb4o:level <http://fing.edu.uy/sigmu/20170816/Clasificaciones/levels#Clasificacion>] ;
18    qb:component    [ qb4o:cardinality  qb4o:ManyToOne ;
19      qb4o:level <http://fing.edu.uy/sigmu/20170816/Anio/levels#Anio>] ;
20    qb:component    [ qb:measure <http://fing.edu.uy/sigmu/20170816/measures#Productores> ;
21      qb4o:aggregateFunction  qb4o:Sum] .
```

Listado 4: Grafo de listado de indicadores

- *Measures*: Contiene información de todas las medidas del sistema, existe un único grafo de este tipo. El listado 5 presenta un ejemplo que contiene la estructura de las medidas correspondientes a los indicadores IAS e ITT.

```
<http://fing.edu.uy/sigmu/20170816/measures#Areasembrada>
        a               qb:MeasureProperty ;
        rdfs:label      "Area sembrada" ;
        rdfs:range      xsd:integer ;
        dc:description  "Area sembrada en miles de hectareas" .

<http://fing.edu.uy/sigmu/20170816/measures#Productores>
        a               qb:MeasureProperty ;
        rdfs:label      "Productores" ;
        rdfs:range      xsd:integer ;
        dc:description  "Productores" .
```

Listado 5: Grafo de listado de medidas

- *Dimensions*: Contiene información de todas las dimensiones e instancias de las mismas, existe un único grafo de este tipo. El Listado 6 presenta un ejemplo donde se puede ver entre las líneas 1 y 11 información sobre una dimensión y una instancia correspondiente al indicador IAS y entre las líneas 13 y 22 se ve la misma información pero correspondiente al indicador ITT.

```
1   <http://fing.edu.uy/sigmu/20170816/dimensions#Cultivos>
2         a                   qb:DimensionProperty ;
3         rdfs:label          "Cultivos" ;
4         dc:description      "Cultivos" .
5
6   <http://fing.edu.uy/sigmu/20170816/Cultivos/instances#Cultivo_Trigo>
7         a        <http://purl.org/qb4olap/cubes#Instance> ;
8         <http://fing.edu.uy/sigmu/20170816/Cultivo/attributes#Nombre> "Trigo 1ra" ;
9         <http://fing.edu.uy/sigmu/20170816/Tipocultivo/attributes#Nombredeltipo> "Trigo" ;
10        <http://purl.org/qb4olap/cubes#memberOf>
11           <http://fing.edu.uy/sigmu/20170816/Cultivos/levels#Tipocultivo> .
12
13  <http://fing.edu.uy/sigmu/20170816/dimensions#Clasificaciones>
14        a                   qb:DimensionProperty ;
15        rdfs:label          "Clasificaciones" ;
```

```
16        dc:description      "Clasificaciones" .
17
18  <http://fing.edu.uy/sigmu/20170816/Clasificaciones/instances#Clasificacion_20a50>
19        a         <http://purl.org/qb4olap/cubes#Instance> ;
20        <http://fing.edu.uy/sigmu/20170816/Clasificacion/attributes#nombre> "20-50" ;
21        <http://purl.org/qb4olap/cubes#memberOf>
22           <http://fing.edu.uy/sigmu/20170816/Clasificaciones/levels#Clasificacion>
```

Listado 6: Grafo de listado de dimensiones

4.3.2. Diseño de URIs

Uno de los objetivos del proyecto consiste en publicar como datos abiertos todos los indicadores generados, para esta tarea Linked Data promueve un conjunto de estándares a considerar para generar información de calidad en su ecosistema. En esta sección se describe la etapa del diseño de URIs, la cual es una de las principales tareas a realizar para no salirse de dichos estándares. Cabe destacar que una URI se utiliza para identificar un recurso, por lo cual, cada URI generada en la plataforma debe ser única. Por otro lado, el sistema genera URIs de forma dinámica cada vez que se ingresa un nuevo indicador al mismo, por lo tanto, hay que realizar un diseño que asegure la unicidad al momento de generarlas y también cumpla con los estándares propuestos. Se recuerda que el impulsor de Linked Data Tim Berners-Lee plantea un conjunto de recomendaciones a la hora de diseñar URIs explicadas en detalle en la sección 2.4.1 sobre las cuales basaremos las decisiones tomadas.

1. Simplicidad: El universo de información que se publica está compuesto por esquemas, medidas, dimensiones, jerarquías, niveles, atributos, instancias y hechos. Todos estos recursos en conjunto representan un indicador. Una URI está compuesta por un prefijo y un nombre local concatenados(<prefijo + nombre local>). para cumplir con el criterio de simplicidad, el cual impone que una URI debe contener semántica sobre los datos que almacena se decide generar un prefijo para cada tipo de recurso con el objetivo que el mismo contenga el nombre del recurso que está definiendo. Para los casos donde la información pertenece solamente a un indicador como son los hechos y el esquema se agrega el nombre del indicador al cual pertenecen como parte del prefijo también con el propósito de agregar semántica. Con respecto al nombre local, se decide utilizar el nombre del recurso, el cual es ingresado por el usuario desde la plataforma y a través de una validación se asegura la unicidad de cada nombre en el sistema. De esta forma, el sistema va a generar URIs que contienen información sobre el tipo de recurso que definen, el indicador al cual pertenecen y el nombre ingresado por el usuario los cuales son datos que aportan semántica sobre el recurso

definido, cumpliendo de esta forma con el criterio de simplicidad.

2. Estabilidad: Ninguna de las URIs contiene detalles relacionados a la tecnología utilizada en el proyecto como tampoco referencias a cómo está implementada la misma. Con esto se busca cumplir el criterio de estabilidad que tiene como foco que las URIs una vez definidas perduren en el tiempo y sean transparentes a cambios tecnológicos del proyecto.

3. Manejabilidad: Todas las URIs contienen la fecha del momento en el que fueron creadas con el objetivo de cumplir con el criterio de manejabilidad que busca que las mismas sean administrables.

En el Listado 6 se presenta el diseño de URIs de cada recurso del sistema, las palabras contenidas entre los símbolos { y } son variables que se sustituyen por el valor correspondiente.

```
Esquemas:       <http://fing.edu.uy/sigmu/20170816/schemas#{schemaName}>
Medidas:        <http://fing.edu.uy/sigmu/20170816/measures#{measureName}>
Dimensiones:    <http://fing.edu.uy/sigmu/20170816/dimensions#{dimensionName}>
Jerarquias:     <http://fing.edu.uy/sigmu/20170816/{dimensionName}/hierarchy#{hierarchyName}>
Niveles:        <http://fing.edu.uy/sigmu/20170816/{dimensionName}/levels#{levelName}>
Atributos:      <http://fing.edu.uy/sigmu/20170816/{levelName}/attributes#{attributeName}>
Instancias:     <http://fing.edu.uy/sigmu/20170816/{dimensionName}/instances#{instanceName}>
Hechos:         <http://fing.edu.uy/sigmu/20170816/{schemaName}/facts#{factName}>
```

Listado 7: Diseño de URIs

4.4. Componente de *Backend*

La capa de *backend* tiene la responsabilidad de exponer en forma de *web services* todas las funcionalidades requeridas por el usuario funcionando como intermediario entre la capa de almacenamiento y el *frontend*. Para realizarlo, se divide en tres componentes con distintas responsabilidades con el objetivo de lograr una aplicación con bajo acoplamiento y de esta forma facilitar las tareas de mantenimiento y sustitución de módulos. A continuación se explican las responsabilidades de los componentes mencionados.

Componente *Service* Este componente tiene la responsabilidad de definir la interfaz de los *web services* y exponerlos hacia la capa de *frontend*. Se deciden utilizar servicios *RESTful* y publicarlos a través del framework *Spring*.

Componente *Mapper* Este componente tiene visibilidad sobre los componentes *DataAccess* y *Service*, su única responsabilidad es realizar el mapeo entre los objetos de base de datos y los objetos pertenecientes a la interfaz de los *web services*.

Componente *DataAccess* Este componente es el encargado de la comunicación entre la aplicación y los grafos existentes en la base de datos. Para realizarlo, se decide utilizar el *framework* Jena que provee un conjunto de funciones útiles para manipular datos almacenados en la plataforma virtuoso.

4.5. Componente de Frontend

Por último, hace falta una plataforma de *frontend* que sirva como punto de acceso global y remoto a los usuarios, que deberá comunicarse vía *web service* con el componente de *backend* para obtener y almacenar información, como ya fue mencionado. Este componente estará dividido en un conjunto de secciones relacionadas:

- Gestión de Indicadores: esta sección contará con las acciones relacionadas al ingreso de nuevos indicadores (junto con sus dimensiones, niveles y atributos) y al listado de los mismos, permitiendo a su vez la edición y eliminación de los mismos.
- Gestión de Datos: esta sección permitirá al usuario el ingreso de datos correspondientes a los indicadores almacenados, permitiendo también seleccionar un indicador en particular y visualizar (y eliminar) todos los datos ya ingresados para el mismo.
- Visualización de Indicadores: esta sección contará con todo lo relacionado a la visualización de los indicadores almacenados y el cruzamiento de datos.

Para esto fueron considerados dos de los frameworks de desarrollo de aplicaciones *frontend* más populares del mercado: ReactJs y Angular2, optando finalmente por ReactJs. A su vez, es necesario utilizar una librería que permita realizar gráfica sobre los datos ingresados, permitiendo distintos tipos de visualizaciones y tipos de gráficas. Para esto se consideraron Google Charts y D3js, optando finalmente por Google Charts. Las razones de dichas decisiones son detalladas en la Sección 5.2.1.

A continuación se detallan cada una de las secciones del componente de *frontend* mencionadas.

4.5.1. Gestión de Indicadores

Esta sección está a su vez dividida en dos subsecciones, 'Crear Nuevo' y 'Listado'. La Figura 4.12 representa sección 'Crear Nuevo' cuenta con un formulario para crear un nuevo esquema de indicador. Dicho formulario cuenta con los campos de texto Nombre y Descripción, a ingresar por el usuario; el campo Medida, donde deberá elegir una de las medidas existentes en el sistema (o crear una nueva, mediante el botón 'Crear Medida'). A

su vez, deberá elegir exactamente dos dimensiones entre las ya almacenadas en el sistema, pudiendo crear una nueva dimensión en el caso de ser necesario mediante el botón 'Crear Dimensión'.

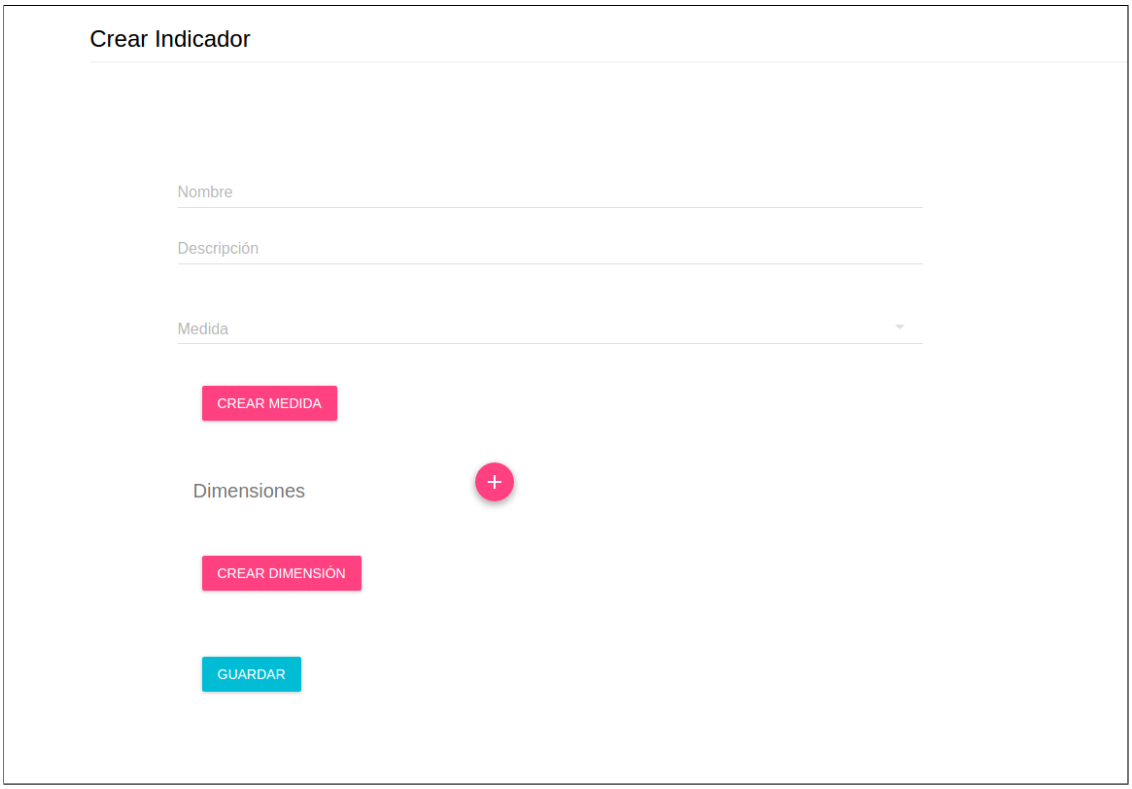


Figura 4.12: Formulario de creación de nuevo indicador

Para que el usuario pueda ingresar un nuevo esquema de indicador, el mismo debe ingresar cada una de los elementos que conforman dicho esquema de acuerdo a la ontología QB4OLAP. Como ya mencionamos, un esquema debe contar con un conjunto de dimensiones, donde cada dimensión está a su vez conformada por una jerarquía, que está relacionada con un conjunto de niveles y dichos niveles con un conjunto de atributos. Luego, si una jerarquía estuviera compuesta por dos niveles o más, deberá también contar con un paso de jerarquía, que es cuál indica qué nivel es el hijo y cuál el padre, qué cardinalidad manejan entre sí y cuál es su función de *roll up*.

Como el modelo multidimensional puede resultar complejo para los tipos de usuarios a los que se dirige la plataforma, se opta por simplificar lo más posible la información solicitada al usuario por cada esquema que desee crear. De esta forma se concluye que dada una dimensión, se pide al usuario que ingrese una lista de niveles con un atributo cada nivel, de forma que cada nivel esté anidado al nivel anterior. Dicha anidación crea un orden implícito de quién es el hijo y quién el padre en una jerarquía dada, y dicha jerarquía

es creada implícitamente y no se requiere que el usuario la cree. De esa forma, el usuario ingresa una lista de niveles y atributos y la plataforma crea automáticamente la jerarquía y los pasos de jerarquía correspondientes que los relacionan. Esto es posible debido a que la cardinalidad entre padre e hijo para los tipos de esquema que manejaremos será siempre de 1 a N (recordando que QB4OLAP permite modelar distintos tipos de cardinalidades).

A su vez, de acuerdo al conjunto de indicadores que fueron utilizados como referencia, las dimensiones que utilizan una operación OLAP para poder reagrupar datos entre niveles padre e hijo de una dimensión, utilizan siempre la función de agregación Suma (recordando que QB4OLAP permite el modelado de distintas funciones de agregación), es por esto que se tomó la decisión de crear las propiedades de *roll up* también de manera implícita para el usuario, declarando siempre a la función Suma como función de agregación del *roll up*.

La Figura 4.13 representa el formulario para la creación de nuevas dimensiones, desplegado mediante el botón 'Crear Dimensión'. Dicho formulario consta de los campos de texto 'Nombre' y 'Descripción', y una lista de niveles a ingresar por el usuario. Cada nivel tendrá a su vez dos nuevos campos de texto 'Nombre' y 'Descripción', y también un atributo asociado. Dicho atributo tendrá también asociados dos campos 'Nombre' y 'Descripción', como también un campo que indique de qué tipo será el atributo (cadena de caracteres o numérico). Una vez creada una dimensión o una medida, pasará a pertenecer a la lista de dimensiones y medidas disponibles para el usuario en el indicador que se encuentra creando.

La sección Listado cuenta con un listado de todos los indicadores disponibles en el sistema, indicando nombre y descripción de cada uno. A su vez, seleccionando cualquiera de ellos se podrá editar la descripción del mismo o eliminarlo por completo.

4.5.2. Gestión de Datos

Esta sección también está dividida en dos subsecciones, 'Insertar' y 'Listado'. La Figura 4.14 representa la sección de Insertar un nuevo dato y cuenta con un formulario donde se debe seleccionar un indicador de los disponibles en el sistema, para que luego se listen las dimensiones correspondientes a dicho indicador. Se debe elegir un valor para cada una de esas dimensiones (o insertar un valor nuevo mediante el botón 'Crear Instancia' asociado a cada dimensión), para luego ingresar un valor en el campo Medida. La combinación de estos campos forman lo que llamamos un hecho y estará asociado al indicador seleccionado. En el ejemplo de la figura mencionada se cuenta con dos dimensiones, Zafra y Cultivo,

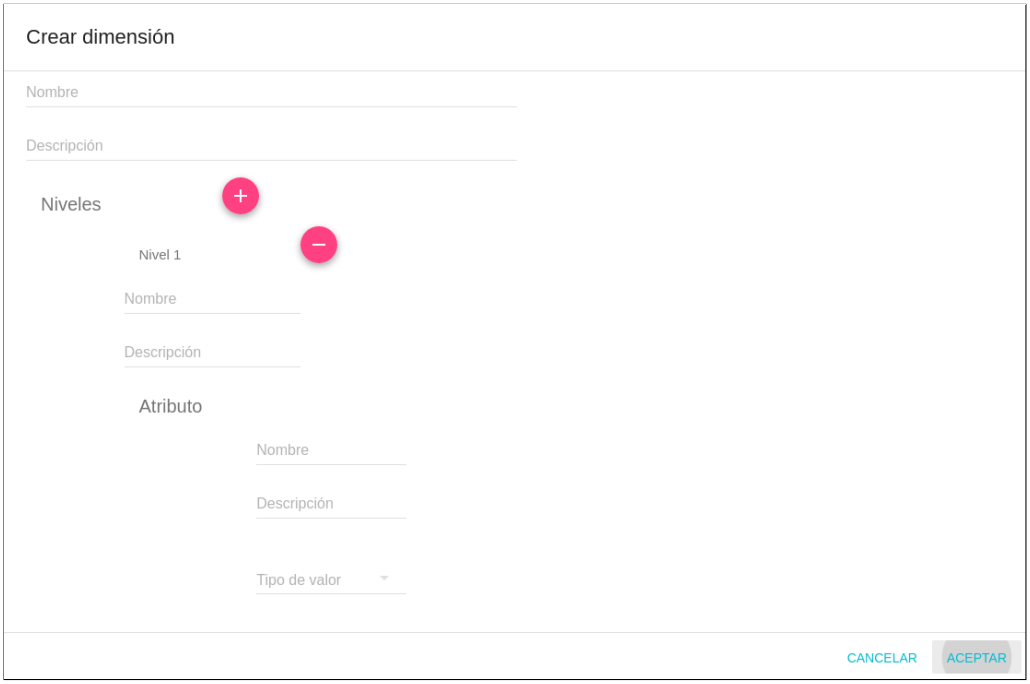


Figura 4.13: Formulario de creación de nueva dimensión

donde la dimensión Zafra cuenta con un nivel que representa al año de la zafra, mientras que la dimensión Cultivo cuenta con dos niveles asociados, Cultivo (hijo) y Tipo de cultivo (padre). En la figura, se muestra un ejemplo de un posible hecho a ingresad para la zafra de los años 2015/2016, el cultivo Soja de primera de tipo Soja, y un valor de medida de 15 solicitudes de eventos liberados. Al presionar el botón 'Crear Instancia' se despliega un nuevo formulario, donde se lista un campo de entrada por cada uno de los niveles que conforman a la dimensión. Para cada uno de estos niveles (puede ser un nivel solamente, o pueden ser nivel padre e hijo) se puede ingresar un dato nuevo o seleccionar uno de los datos ya ingresados previamente para ese mismo nivel. Una vez guardada la instancia, es posible seleccionarla desde el formulario de Insertar un nuevo dato.

Luego, la sección 'Listado' permite también seleccionar un indicador de los disponibles en el sistema, para luego listar todos los datos ingresados sobre el mismo en formato de tabla. Dicha sección está representada en la Figura 4.15. En el encabezado de la tabla se listan en forma de columnas cada uno de los niveles que conforman las distintas dimensiones junto con el nombre de la medida asociada al indicador, y para cada uno de los datos hay una fila correspondiente con el valor de ese dato para cada una de las columnas, permitiendo también la eliminación de los mismos mediante una selección múltiple. Además, se cuenta con un botón 'Descargar CSV' que permite al usuario descargar la información correspondiente a los datos que están siendo visualizados en un formato CSV, respetando

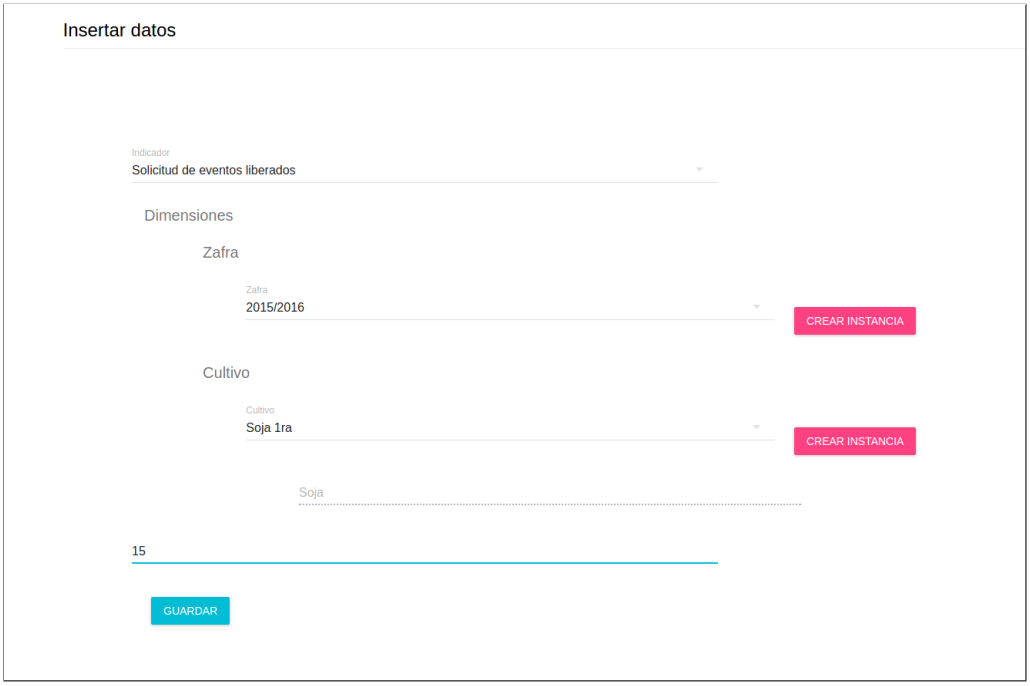


Figura 4.14: Formulario de creación de nueva dimensión

la misma estructura que la tabla mostrada en un principio.

En el ejemplo de la figura se puede ver que el indicador Solicitud de eventos liberados cuenta con dos hechos, uno para la zafra 2014/2015, el cultivo Soja de primera del tipo de cultivo Soja y un valor de medida de 9 solicitudes, y otro hecho para la zafra del año 2015/2016, el cultivo Soja de primera del tipo Soja y un valor de medida de 15 solicitudes.

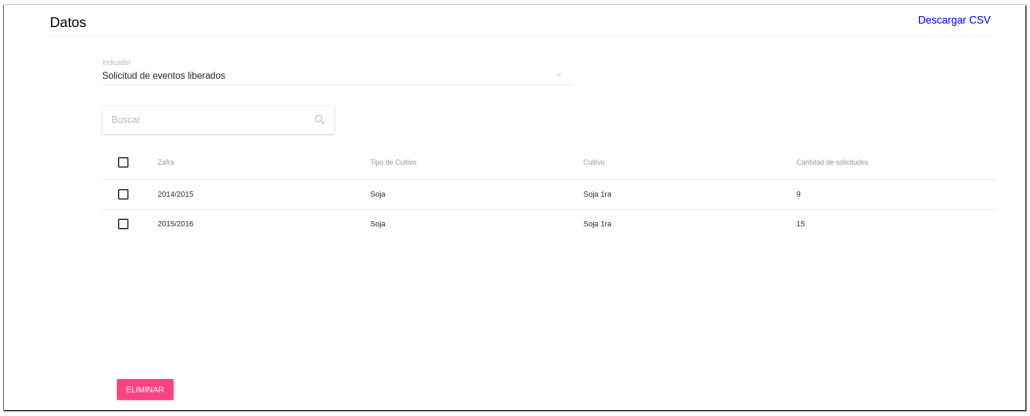


Figura 4.15: Listado de hechos de un indicador

4.5.3. Visualización de indicadores

En la sección de Visualización el usuario debe seleccionar uno de los indicadores disponibles para poder graficar sus datos en forma de gráfica de barras verticales. Dado un indicador, la metodología utilizada para graficarlo es la siguiente: una de las dimensiones

ocupa el eje horizontal, y para cada valor posible de esta dimensión hay un grupo de columnas correspondientes a él. Luego, la otra dimensión está representada mediante un color, y para cada valor posible de esta dimensión hay una barra con su correspondiente color, y habrá una barra para cada grupo de valores en el eje horizontal. Luego, para cada combinación de (valores del eje horizontal-cantidad de colores) hay un número asociado a ellos, que representa a la medida del indicador y tiene su representación en el eje vertical, como muestra la Figura 4.16.

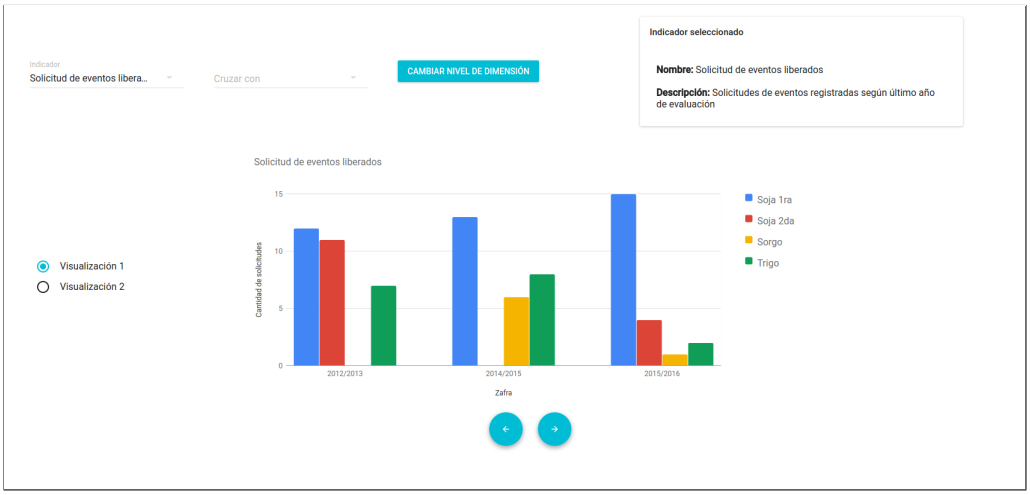


Figura 4.16: Visualización de un indicador

Además se cuenta con dos tipos de visualización; la gráfica de barras regular, donde cada color correspondiente a cada valor posible de la dimensión representada por colores está representado en una columna distinta, y otra visualización donde cada color es acumulado uno encima del otro en una misma columna, lo que se llama una gráfica de barras 'acumuladas'. El usuario puede cambiar de forma de visualización de una a otra en forma dinámica. En la Figura 4.16 puede verse la representación gráfica de un indicador en su forma de visualización por defecto, mientras que la Figura 4.17 representa al mismo indicador visualizado en su formato 'acumulado'.

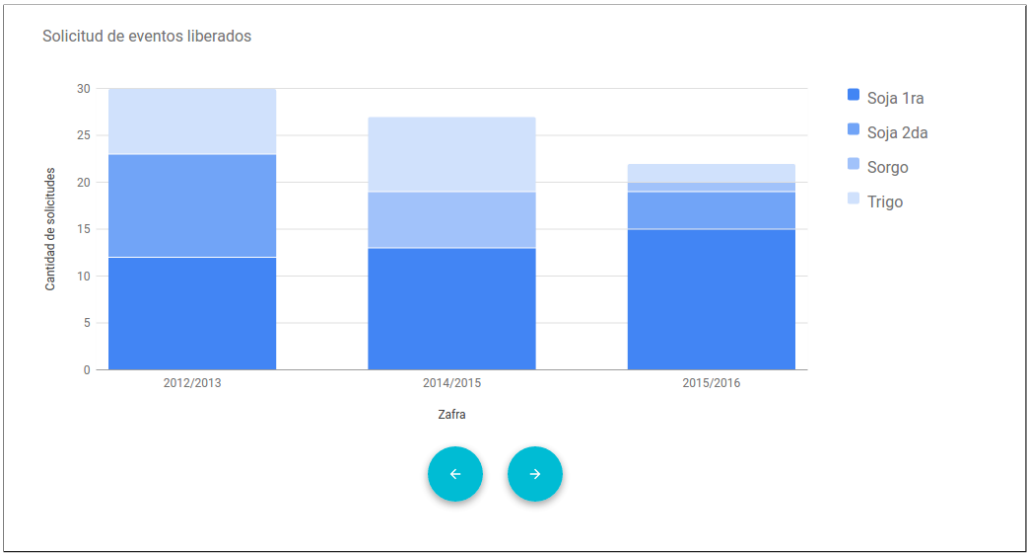


Figura 4.17: Visualización de un indicador en formato 'acumulado'

Dicha figura representa al indicador Solicitud de eventos liberados que modela la cantidad de solicitudes de autorización de vegetales genéticamente modificados, y que cuenta con las dimensiones Zafra y Tipo de cultivo, ya que refleja la cantidad de solicitudes realizadas para cada cultivo en cada año. En el ejemplo dado se cuenta con hechos correspondientes a las zafras 2013/2014, 2014/2015 y 2015/2016, y para los cultivos Soja de primera, Trigo y Sorgo, donde la altura de cada barra de la gráfica corresponde al valor de la medida para cada combinación zafra-cultivo.

Por otro lado, también se le permite al usuario intercambiar qué dimensión está representada de cuál de las dos maneras, es decir cuál está representada en el eje horizontal y cuál está representada mediante colores. Esto también puede realizarlo en forma dinámica mediante el par de botones 'Visualización 1' y 'Visualización 2'.
En la Figura 4.18 puede verse al mismo indicador que en las Figuras 4.16 , con sus dimensiones representadas en cada eje intercambiadas.

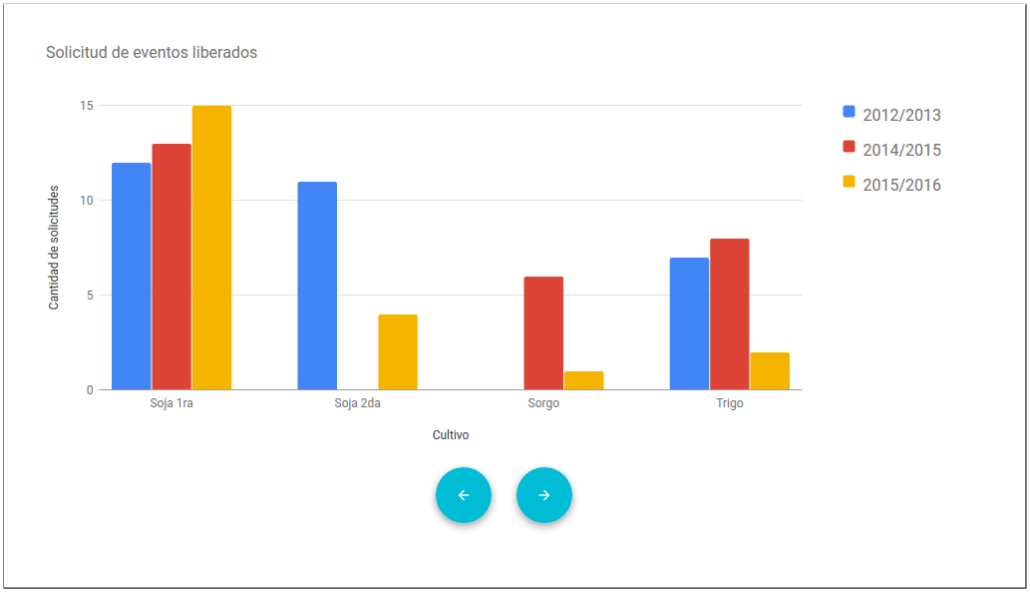


Figura 4.18: Visualización de un indicador con sus ejes 'intercambiados'

En caso de que el indicador seleccionado cuente con una dimensión conformada por dos o más niveles, puede también cambiar cuál de los niveles desea graficar. Para esto debe presionar el botón 'Cambiar nivel de dimensión' y elegir el nivel que desea graficar. Cuando se selecciona un nivel 'inferior' en una de las dimensiones es cuando deberán aplicarse las operación *roll up* sobre los datos, donde cada valor de los niveles 'hijos' son agrupados en un único valor para el nivel 'padre', mediante la suma cada uno de sus valores hijos.
En la Figura 4.19 se puede ver el formulario desplegado luego de presionar el botón 'Cambiar nivel de dimensión', donde se pueden ver las distintas dimensiones con sus distintos niveles a seleccionar.

Figura 4.19: Visualización del formulario para cambiar el nivel de dimensión de un indicador

En la Figura 4.20 se ve representado el indicador luego de aplicar la operación *roll up* sobre los valores de la dimensión Cultivo, por lo que pasan de estar los datos agrupados

en base a los cultivos individuales (Soja 1ra, Soja2 2da, etcétera) a estar agrupados de acuerdo a los tipos de cultivos de cada uno, por lo que los datos de cultivos que compartan el tipo de cultivo pasan a formar parte de un mismo dato, sumando sus valores.

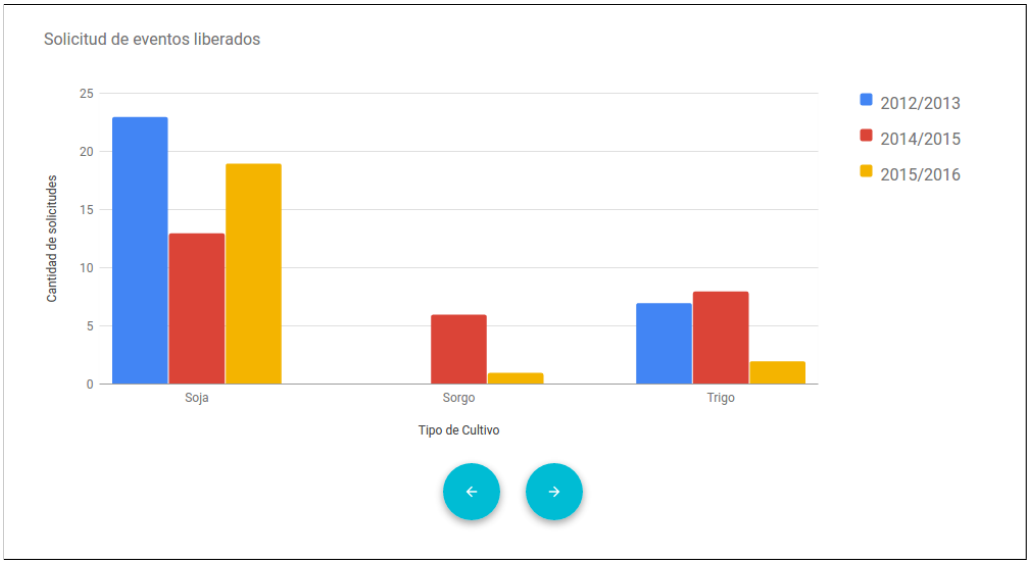


Figura 4.20: Visualización de un indicador luego de aplicar el *roll up* en una de sus dimensiones

Luego de seleccionado el primer indicador, se habilitan también una lista de indicadores compatibles a ser comparados con él y graficados en forma simultánea. Como ya fue mencionado, estos indicadores serán aquellos que compartan una o más dimensiones con el indicador seleccionado. Una vez seleccionado el indicador que se de sea comparar, se gráfica nuevamente ambos indicadores en una gráfica compartida, donde en el eje horizontal se ubica la dimensión que comparten (en caso de compartir ambas, podrá elegirse cuál será representada en el eje horizontal mediante los botones ya mencionados) y hay un color dedicado a cada una de las dimensiones restantes de cada indicador. Dicho cruzamiento implica realizar la operación OLAP de *drill across*, que dados dos esquemas compatibles arroja como resultado un nuevo resultado con los datos de ambos indicadores.

El formato de visualización elegido para estos cruzamientos es el de la gráfica de barras 'acumuladas', donde cada columna representará uno de los indicadores, y cada porción 'acumulada' de una columna es uno de los valores posibles de esa dimensión, y el ancho de dicha porción es el valor de la medida correspondiente a esa combinación de valores. A su vez, al haber dos indicadores hay también dos medidas distintas, por lo que hay dos ejes verticales uno correspondiente a cada indicador.

En el caso en que los indicadores compartan ambas dimensiones, como ya fue mencionado puede elegirse cuál de las dimensiones está representada en el eje horizontal y cuál

está representada mediante un color, y a su vez se habilitan también el botón 'Cambiar nivel de dimensión', para poder realizar las operaciones OLAP sobre cualquiera de sus dimensiones.

En la Figura 4.21 se puede ver cómo está representado el cruzamiento de dos indicadores en su forma gráfica, y será explicada en detalle en la Sección 5.2.4.

Figura 4.21: Visualización del cruzamiento de dos indicadores compatibles, con datos reales de Uruguay.

5. Implementación de la solución

En este capítulo se presentan detalles de implementación sobre los componentes del sistema, las tecnologías elegidas para realizar el desarrollo y por último las restricciones existentes en la plataforma. Para realizarlo, se estructura en tres secciones principales donde la primera tiene foco en el componente de *backend* en la cual se mencionan las tecnologías elegidas, se listan los *web services* definidos y se detallan alguna decisiones de implementación relacionadas al almacenamiento de indicadores en lenguaje RDF. La segunda sección esta enfocada al componente de *frontend* donde se discuten las alternativas tecnológicas existentes, se detallan los componentes principales y se explica el uso de las operaciones OLAP a través de un ejemplo. Para finalizar el capítulo, se presentan algunas restricciones existentes en el sistema.

5.1. Componente de *Backend*

En esta sección se detallan las diferentes tecnologías utilizadas para la implementación del componente de *frontend* y qué propósito cumple cada una de ellas.

Luego se presentan un conjunto de servicios implementados por dicho componente y que ofician de interfaz de comunicación entre el componente mencionado y el de *frontend*, detallando qué función cumple cada servicio. Se presentan también los dos tipos de consultas implementadas para el acceso a base de datos, destacando las características y presentando un ejemplo de cada una. Por último se describe cómo funciona la exportación de datos abiertos de los indicadores de la plataforma.

5.1.1. Elección de tecnologías

Para desarrollar el componente de *backend* se tomó la decisión de utilizar la tecnología Java ya que no requería la curva de aprendizaje por parte de ninguno de los integrantes y cuenta con basta documentación para el desarrollo de esta aplicación. A su vez existen distintos *frameworks* que permiten la integración con la base de datos multidimensionales que luego se detalla. A continuación se detallan los *frameworks* utilizados desde el *backend*.

Spring framework

Spring [15] es un *framework* para el desarrollo de aplicaciones y contenedor de inversión de control, de código abierto para la plataforma Java. *Spring* se ajusta a las aplicaciones Java sin tener que modificar el código para utilizar las funcionalidades y beneficios que

ofrece. Para usar este *framework* no es necesario implementar un interfaz propia de Spring o heredar de una clase propia. A lo sumo las clases tendrán anotaciones propias de *Spring*. Otro factor determinante en la elección de *Spring* fue la inyección de dependencias. Los objetos reciben sus dependencias en tiempo de creación, de manera que ellos no son responsables de la instanciación e inicialización de esas dependencias. Es decir que las dependencias se inyectan en el objeto que las necesite, porque de esta forma, el objeto no debe preocuparse de crear esas instancias.

Almacenamiento

La tecnología a utilzar para el almacenamiento de datos se decide en función de pruebas de *performance* realizadas entre Virtuoso y RDF4J que son dos bibliotecas *open source* utilizadas para almacenar RDF (*RDF Store*). Dichas pruebas se presentan en el anexo 9 y tienen como objetivo obtener los tiempos de respuesta de cada tecnología para insertar y obtener triplas variando la cantidad de triplas utilizadas. Las pruebas reflejan que Virtuoso es más eficiente que RDF4J cuando se desean insertar menos de 500 triplas simultaneamente, pero cuando la cantidad de triplas es mayor, los tiempos de virtuoso empiezan crecer por encima de los de RDF4J de forma considerable. Con respecto a los tiempos recolectados al obtener triplas, Virtuoso resulta más eficiente independientemente de la cantidad. Teniendo en cuenta dichos números y que un indicador promedio está compuesto por menos de 100 triplas se decide utilizar Virtuoso para realizar el almacenamiento de datos.

Jena

Apache Jena[11] es un *framework* Java para construir aplicaciones basadas en ontologías. Este *framework* de la web semántica es de código abierto para Java y proporciona un API para escribir y leer datos de grafos RDF. Su Arquitectura incluye:

1. API para leer, procesar, escribir ontologías RDF y OWL
2. Motor de inferencia para razonar sobre ontologías RDF y OWL
3. Estrategias de almacenamiento flexible para almacenar triplas RDF en memoria o fichero
4. Motor de consultas compatible con especificación SPARQL.

5.1.2. Interfaz definida

En el Cuadro 2 se presenta la interfaz definida para exponer las funcionalidades del sistema a través de servicios RESTful.

insertFact	Inserta un nuevo hecho dentro del esquema detallado en la url del servicio y los datos del hecho se incluyen en el cuerpo del servicio.	POST
insertSchema	Inserta un nuevo esquema y todos sus datos (menos el id, el cual se genera al insertar el esquema en la base de datos) se pasan a través del cuerpo del servicio. Dentro de los datos del esquema se tiene la medida y las dimensiones, las cuales van a ser insertadas en sus correspondientes grafos y también dentro del grafo de esquema.	POST
insertInstance	Agrega una nueva instancia junto con sus atributos, y además en caso de existir *roll up* también se insertan estas y se referencian como *roll up* desde las primeras.	POST
insertDimension	Inserta una nueva dimensión de un nivel, el cual también es insertado en su correspondiente grafo.	POST
insertMeasure	Agrega una nueva medida indicando nombre y descripción.	POST
insertUser	Se agrega un nuevo usuario con permisos de administrador a la aplicación.	POST
validateUser	Se valida que el usuario y contraseña tengan permisos para ingresar a la aplicación en modo administrador.	POST
getSchemas	Devuelve el listado de esquemas con su descripción y dimensiones.	GET
getSchemaFact	A partir de un nombre de esquemas se obtiene el esquema del mismo y sus hechos.	GET

getDimensions	Retorna la lista de todas las dimensiones de la aplicación y su respectivo nivel.	GET
getMeasures	Devuelve el listado de todas las medidas.	GET
getInstances	Retorna un listado de todas las instancias del sistema agrupadas por nivel.	GET
export	A partir del nombre de un *schema* se devuelve el RDF del grafo solicitado, cumpliendo con el estándar de '5 estrellas' de datos abiertos.	GET
dropSchemaFact	Este servicio a partir de un nombre de esquema borra todos los datos del grafo de estos y también los datos de hechos asociadas a este esquema.	DELETE
deleteFacts	A partir del nombre de un esquema y una lista de hechos se borran estos hechos del grafo correspondiente.	DELETE
updateDescription	Actualiza la descripción de un esquema.	PATCH

Cuadro 2: Interfaz de servicios

5.1.3. Consultas sobre grafos

Las consultas sobre los grafos se realizan utilizando el lenguaje SPARQL que permite entre otras cosas obtener las triplas de un grafo que cumplan ciertas condiciones. Dentro del proyecto existen dos tipos de consultas, por un lado las llamadas estáticas que son las que no contienen en su definición ninguna URI de las creadas dinámicamente por el sistema sino que por el contrario sólo utilizan ontologías conocidas como son QB4OLAP, QB, RDFS, RDF y DCT (Dublin Core Terms), dichas consultas se defiinen en etapa de desarrollo y se almacenan en un archivo XML. Por otro lado se tienen las consultas dinámicas que a diferencia de las anteriores dependen del indicador que se esté consultando y son generadas en tiempo de ejecución donde primero se busca información sobre la *metadata*

del indicador para luego generar la consulta a realizar. A continuación se presenta una consulta de cada tipo con el objetivo de ejemplificar lo dicho en la sección.

A continuación en el Listado 8 se listan los prefijos utilizados para realizar las consultas SPARQL, tanto en las estáticas como en las dinámicas.

```
PREFIX qb:    <http://purl.org/linked-data/cube#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dct:   <http://purl.org/dc/terms/>
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX qb4o:  <http://purl.org/qb4olap/cubes#>
```

Listado 8: Prefijos utilizados para consultas SPARQL

Consulta estática

A continuación en el Listado 9 se presenta una consulta utilizada para obtener la *metadata* completa de las dimensiones del sistema.

```
SELECT ?dim ?dimName ?dimDesc ?level ?levelName ?levelDesc
?attribute ?attributeName ?attributeDesc
?attributeRange ?childLevel
WHERE {
      ?hierarchy rdf:type qb4o:Hierarchy.
      ?hierarchy qb4o:hasLevel ?level.
      ?hierarchy qb4o:inDimension ?dim.
      ?dim rdf:type qb:DimensionProperty.
      ?dim rdfs:label ?dimName.
      ?dim dct:description ?dimDesc.
      ?level rdfs:label ?levelName.
      ?level dct:description ?levelDesc.
      ?level qb4o:hasAttribute ?attribute.
      ?attribute rdf:type qb4o:LevelAttribute.
      ?attribute rdfs:label ?attributeName.
      ?attribute dct:description ?attributeDesc.
      ?attribute rdfs:range ?attributeRange.
      OPTIONAL {
            ?hierStep rdf:type qb4o:HierarchyStep.
```

```
            ?hierStep qb4o:inHierarchy ?hierarchy.
            ?hierStep qb4o:childLevel ?childLevel.
            ?hierStep qb4o:parentLevel ?level.
        }
}
```

Listado 9: Consulta SPARQL estática

Consulta dinámica

El Listado 10 detalla Consulta utilizada para obtener todas las instancias de una dimensión perteneciente a un indicador. En este caso, los prefijos 'lvl' y 'attr' son generados dinámicamente debido a que la URI contiene información del indicador.

```
SELECT  ?instance ?value
WHERE {
       ?instance rdf:type qb4o:Instance .
       ?instance qb4o:memberOf lvl:CultivoLevel .
       ?instance attr:CultivoAttribute ?value
 }
```

Listado 10: Consulta SPARQL dinámica

5.1.4. Publicación de Indicadores como Datos Abiertos

Una de las funcionalidades del sistema es poder exportar toda la información de un indicador utilizando el lenguaje RDF. En este punto se refleja una de las ventajas de utilizar grafos RDF como modelo de datos debido a que a través del *framework* Jena existe una función que permite a partir de un grafo generar un archivo con la sintaxis deseada, en nuestro caso elegimos *Turtle* considerando que es la más sencilla de comprender. Aprovechando las facilidades provistas por Jena, la tarea que se debe resolver al momento de exportar un indicador es extraer solamente la información del mismo de los grafos que contienen todos los indicadores del sistema, recordando que un indicador está compuesto por la información de su esquema, instancias y hechos almacenada en tres grafos diferentes como se explica en la Sección 4.3.1. Tanto el esquema como los hechos de un indicador están en un grafo independiente que no contiene información de otros indicadores por lo que para realizar la exportación alcanza con pasar los grafos completos a la función de Jena que genera un archivo Turtle a partir de los mismos. Por el contrario, las instancias de un indicador están definidas en un grafo junto con las instancias del resto de los indicadores

debido a su necesidad de ser reutilizables por lo cual no es posible exportar el grafo completo. Como solución se realiza una consulta SPARQL que permite construir un sub-grafo a partir de un grafo para extraer solamente las instancias del indicador deseado y luego a través de la función de Jena generar el archivo Turtle a exportar.

5.1.5. *Roll up* y *Drill down*

Dado un cubo que contenga mas de un nivel en alguna de sus jerarquías se deben soportar las operaciones *Roll up* y *Drill down* sobre los *hechos* del mismo. Se plantea como solución que el servicio *getSchemaFact* encargado de devolver los *hechos* de un cubo no devuelva solamente los valores asociados al nivel raíz de cada jerarquía sino que también retorne los valores asociados al resto de los niveles con el objetivo de proveer toda la información necesaria para realizar dichas operaciones desde la capa de *frontend*. De esta forma, no es necesario invocar a un servicio cada vez que se necesite realizar *Roll up* o *Drill down* sobre los hechos de un cubo.

5.2. Componente de *frontend*

En esta sección se detallan las diferentes tecnologías que se tuvieron en cuenta para la implementación del componente de *frontend*, especificando las características estudiadas de cada una de ellas y los motivos para las decisiones tomadas.

Además, se presentan los principales elementos implementados y qué función cumple cada uno de ellos. También se describe cómo fueron implementadas las operaciones OLAP utilizadas para en análisis de los datos, las operaciones *roll up* y *drill down*, y por último se ven en detalle una serie de restricciones que presenta el componente en cuestión, por qué surgieron y en qué limitan a la aplicación.

5.2.1. Elección de Tecnologías

Para la implementación del *frontend* se consideran dos áreas principales sobre las cuales investigar la tecnología a ser utilizada. En primer lugar se debe decidir que biblioteca utilizar para realizar el desarrollo del mismo para luego investigar las diferentes bibliotecas existentes que permitan realizar gráficas sobre un conjunto de datos. En esta sección se discuten las diferentes alternativas fundamentando la elección de cada una.

Biblioteca de *frontend*

En el Cuadro 3 se presenta una comparación entre las bibliotecas Angular 2 y RectJS

poniendo foco en tres conceptos principales.

	Angular 2	**ReactJS**
Complejidad/Curva de aprendizaje	Angular es una biblioteca basada en el lenguaje Javascript, y promueve el desarrollo sobre la misma en lenguaje TypeScript, un lenguaje nuevo creado por Microsoft que es subconjunto de Javascript. Gran parte de la documentación de Angular y de información de la comunidad utiliza TypeScript, lo que implica la necesidad de investigar otro lenguaje además de Angular. A su vez, Angular se trata de insertar directivas propias (ngIf, ngFor, etc) sobre HTML, lo que implica también la necesidad de investigar dichas directivas y combinar parte del desarrollo en HTML y parte en TypeScript.	React es también una librería basada en Javascript, toda su documentación e información de la comunidad utiliza Javascript. A su vez, React a diferencia de Angular se trata de insertar HTML dentro del Javascript, por lo que el desarrollo está centralizado en Javascript y no implica investigación sobre otros lenguajes y directivas.
Documentación y comunidad	En cuanto a documentación tanto oficial como de la comunidad podemos decir que Angular tiene una tiene una buena documentación oficial con una gran cantidad de ejemplos, mientras que cuenta con más de 35 mil temas abiertos en *stackoverflow*. Por otra parte, gran parte de dicha documentación corresponde al lenguaje TypeScript.	En cuanto a documentación tanto oficial como de la comunidad podemos decir que React tiene una tiene una buena documentación oficial, no tan extensa como la de Angular. Mientras que cuenta con más de 30 mil temas abiertos en *stackoverflow*, con la ventaja de que toda su documentación corresponde al lenguaje Javascript.

Popularidad	Angular es en este momento cuenta en su cuenta de Github con más de 2 mil seguimientos, más de 20 mil favoritos y más de 5 mil bifurcaciones.	React es en este momento cuenta en su cuenta de Github con más de 4 mil seguimientos, más de 60 mil favoritos y más de 11 mil bifurcaciones.

Cuadro 3: Comparación entre Angular 2 y ReactJs

En función de los factores presentados en el Cuadro 3 se considera que React es la librería más adecuada a utilizar ya que dados nuestros conocimientos de Javascript implica una menor curva de aprendizaje. La diferencia de popularidad en la comunidad también es un factor importante para inclinarnos por dicha biblioteca.

Generación de gráficas

En el Cuadro 4 se presenta una comparación entre las bibliotecas Google Charts, D3js y D3js embebido poniendo foco en cuatro conceptos principales.

	Google Charts	**D3js**	**D3js embebido**
Complejidad/Curva de aprendizaje	Cuenta con una API pública propia de Google que permite graficar de forma simple y directa, con la posibilidad de tener que ingresar solamente los datos.	Cuenta con una API pública con la particularidad de que la gran mayoría de sus gráficas fueron desarrolladas por distintos desarrolladores, al ser una librería *open source*, y es por eso que su uso es complejo y permite una gran flexibilidad en cuanto a parámetros y configuración.	Existen distintos paquetes *Javascript* encargados de exponer las funcionalidades de D3 en funciones reutilizables de manera de simplificar a las APIs consumidoras. Dado que su funcionalidad es simplificar las funciones que implementa D3, su uso es simple y concreto.

Documentación	La documentación oficial de *Google* cuenta con ejemplos para cada una de sus gráficas así como una descripción de cada uno de los parámetros configurables.	La documentación de cada gráfica de D3 puede variar, dependiendo de si es un producto propio o una extensión de la librería hecha por desarrolladores externos, es por eso que no sigue una línea en común y en general no son tan extensas como las oficiales de *Google*.	Su documentación es por lo general muy concreta, hecha por los mismo desarrolladores y que cubren en general el caso básico para cada una de las gráficas.
Cantidad de gráficas/Posibilidad de extensión	Tiene una gran cantidad de gráficas a disposición de su API, pero dicha API no es extensible o modificable.	Cuenta con más de 200 tipos distintos de gráficas, de los cuales la gran mayoría escapan los propósitos de este proyecto. A su vez, al ser un producto *open source* tiene la posibilidad de ser extendido.	Dichos paquetes cuentan con un número mucho más reducido de gráficas que el D3 original, contando con las gráficas básicas de barras, líneas y tortas.
Facilidad de integración con *Reactjs*	*React* cuenta con múltiples extensiones de *Google Charts*, haciendo aún más simple su integración.	D3 también cuenta con múltiples extensiones hechas para *ReactJs*.	No cuentan con extensiones, lo que implica una implementación de *javascript* nativo dentro de la aplicación *React*

Cuadro 4: Comparación entre *Google Charts*, D3js y D3js embebido

En función de los factores presentados en el Cuadro 4 se considera que *Google Charts* es la opción que mejor se adapta a nuestras necesidades debido a que las gráficas que provee

abarcan todos los casos de uso necesarios para nuestra plataforma e implica una curva de aprendizaje relativamente sencilla teniendo en cuenta la buena documentación oficial y no oficial generada por los usuarios existentes. Por otro lado, por ser propia de Google nos aseguramos un soporte actualizado y completo y un menor margen de error en cuanto a posibles errores propios del *framework*.

5.2.2. Componentes *ReactJs*

En el Cuadro 5 se presentan los componentes implementados en *ReactJs* más relevantes del *frontend* indicando la responsabilidad de cada uno de ellos.

VisualizationComponent	Componente encargado de la sección de visualización de indicadores y cruzamiento de datos. Está conformado a su vez por los componentes IndicatorSelectComponent y ChartComponent.
IndicatorSelectComponent	Componente encargado de listar los indicadores para que puedan ser seleccionados y graficados. Para desplegar dicho listado, debe invocar al servicio de *backend* que retorna la lista de todos los indicadores del sistema y, una vez seleccionado el indicador a graficar, también invocará al servicio encargado de retornar todos los hechos relacionados a dicho indicador. Además contiene el formulario donde se podrá cambiar el nivel de dimensión de un indicador como se muestra en la Figura 14.
ChartComponent	Componente encargado de desplegar la visualización gráfica de los indicadores. Para esto, recibe de parte del componente IndicatorSelectComponent la información relacionada al indicador seleccionado a graficar, y realiza los llamados a la biblioteca de *Google Charts* necesarios para desplegar la gráfica.
CreateIndicatorsComponent	Componente encargado de la creación de nuevos esquemas de indicadores. Una vez llenado el formulario de creación, se encarga de invocar el servicio de *backend* encargado de almacenar los esquemas en base de datos. Está compuesto también por los componentes CreateMeasureComponent y CreateDimensionComponent.

CreateMeasureComponent	Componente encargado de la creación de nuevas medidas para que luego puedan formar parte de los esquemas de indicadores. Se encuentra dentro del componente CreateIndicatorsComponent, ya que las medidas pueden crearse desde la misma sección que los indicadores, mediante un formulario que aparece luego de presionar un botón con el texto 'Nueva Medida'. Luego de completado el formulario, se encarga de invocar al servicio de *backend* que luego almacenará la nueva medida en la base de datos.
CreateDimensionComponent	Al igual que las medidas, las dimensiones pueden ser creadas dentro de la misma sección que los indicadores, de forma que luego de presionar un botón con el texto 'Nueva Dimensión' aparece el formulario correspondiente, y luego de creada la dimensión el formulario desaparece y la nueva dimensión aparece disponible a ser utilizada en el indicador que está siendo creado en dicho momento. Además de agregar la nueva dimensión a la lista de dimensiones disponibles, previamente invoca al servicio de *backend* encargado del almacenamiento de nuevas dimensiones en base de datos.
IndicatorsListComponente	Componente encargado de invocar al servicio de *backend* que lista todos los indicadores disponibles en base de datos, para desplegarlos luego en un listado de indicadores y permitir la edición o eliminación de los mismos.
FactsListComponent	Componente encargado de listar los hechos relacionados a los distintos indicadores, tanto para visualización de los mismos como para proveer su posible eliminación. Para ello, se encarga de invocar al servicio de *backend* que lista los indicadores almacenados, y luego de seleccionado uno de estos indicadores por parte del usuario, invocar al servicio de *backend* encargado de listar todos los hechos relacionados a un indicador, para poder desplegar una lista de los mismos.

InsertFactComponent	Componente encargado de la inserción de datos para un indicador seleccionado. Para ello, se encarga de invocar al servicio de *backend* que lista los indicadores almacenados, y luego de seleccionado uno de estos indicadores por parte del usuario, este deberá llenar el formulario correspondiente a un nuevo hecho, y luego de finalizado se deberá invocar al servicio encargado de la inserción de nuevos hechos en base de datos.

Cuadro 5: Listado de los principales componentes implementados en ReactJs

5.2.3. *Roll up y Drill down*

Al momento de graficar un indicador que contiene al menos una de sus dimensiones compuesta por 2 o más niveles, será posible realizar operaciones OLAP sobre los datos de dichos niveles. Como ya fue mencionado anteriormente, esto será posible mediante el botón 'Cambiar niveles de dimensión'. Cabe destacar que para realizar las operaciones OLAP, es decir, seleccionar cuál nivel de cada dimensión se desea graficar, es necesario conocer el esquema del indicador en cuestión, ya que será necesario saber cuáles son sus dimensiones y qué niveles componen cada una de ellas, para poder luego seleccionar una de las opciones para cada dimensión. Dicha información proviene del servicio de *backend* encargado de listar los indicadores almacenados.

Dada una dimensión compuesta por ejemplo por dos niveles 'padre' e 'hijo', se obtiene desde el componente de *backend* todos los datos correspondientes a cada uno de esos niveles para cada uno de los hechos, y dichos valores estarán encapsulados dentro de un mismo objeto. Es decir, tomando como ejemplo una dimensión que compuesta de un nivel 'padre' llamado Tipo de Cultivo y nivel 'hijo' llamado Cultivo, cada dato ingresado para un indicador que utilice dicha dimensión deberá tener un valor para el nivel Tipo de Cultivo, un valor para el nivel Cultivo y un valor para la medida asociada. Al momento de realizar las operaciones *roll up* sobre los datos de dicho indicador, se deberán recorrer todos los datos uno por uno y deberán reagruparse de acuerdo a los valores del nivel 'padre'. Es decir, los datos que coincidan en el valor del nivel padre serán reagrupados a un nuevo y único dato, mediante la suma de todos los valores asociados a la medida. Siguiendo con el ejemplo, dado un tipo de cultivo (Soja) se sumarán todos los valores de medida de los

datos que contengan en el nivel Tipo de Cultivo al valor Soja, sin importar qué valor tenían en el nivel 'hijo'. A modo de ejemplo, si la otra dimensión fuera Tiempo compuesta por un nivel llamado Año, y los miembros existentes fueran 2014 y 2015, los datos reagrupados luego de realizar la operación de *roll up* sobre la dimensión Tipo de Cultivo deberán obtenerse a partir de la suma obtenida de todos los datos que compartan el valor en la dimensión Tipo de cultivo (agrupados por Año). Entonces, si el conjunto de datos tuviera los miembros Soja 1a (Soja de primera) y Soja 2a (Soja de segunda) cuyo padre es el tipo de cultivo Soja, y Trigo 1a (Trigo de primera) y Trigo 2a (Trigo de segunda), cuyo padre es el tipo de cultivo Trigo, y también contenga hechos que relacionen dichos cultivos con la dimensión Tiempo para los miembros 2014 y 2015, los datos reagrupados deberán contar con instancias solamente para Soja y Trigo (y 2014 y 2015), reagrupando (sumando) los datos que correspondían a cada tipo de cultivo según su año.

De esta forma, será posible transformar una gráfica donde en uno de sus ejes estaba representada una de sus dimensiones mediante el nivel Mes, a una nueva gráfica donde en uno de sus ejes estará representada la misma dimensión mediante el nivel Año.

Para lograr la funcionalidad recién descrita, se optó por realizar el procesamiento de los datos en el componente de *backend*, de forma que una vez solicitados los datos de un indicador mediante la invocación del servicio correspondiente, alcanzaría con la información retornada en dicho servicio para generar la visualización del indicador en cualquiera de sus formatos y agrupaciones de datos, sin necesidad de volver a invocar otro servicio. Esto se logra debido a que dado un hecho de un indicador, que puede tener valores asociados a distintos niveles (tanto padre como hijos) y a su medida, el servicio en cuestión retornará un objeto autocontenido por cada hecho, es decir, con los valores de su medida así como también con todos los valores de sus niveles asociados. Siguiendo con el ejemplo anterior, para un hecho del indicador cuyas dimensiones son Tiempo y Tipo de cultivo y cuya medida represente a las hectáreas sembradas para un cultivo en un año dado, el objeto retornado que representa a dicho hecho contendrá tanto los valores para los niveles de granularidad más fina (Cultivo y Año) como para los de granularidad más gruesa (Tipo de Cultivo). Entonces un objeto correspondiente a uno de los hechos podría contener los siguientes datos <Año: 2017, Tipo de cultivo: Trigo, Cultivo: Trigo 1a, Hectáreas cosechadas: 1500>. Esto implica que el valor de cada año estará replicado en todos los hechos que contengan valores para meses de ese mismo año, pero por otro lado, para realizar la función de *roll up* sobre la dimensión Año, y graficar los datos agrupados en base a los años y no a los meses, bastará con agrupar los hechos que coincidan en el valor Año y sumar sus respectivas valores de

medida. Luego, para realizar la función de *drill down* sobre dicha dimensión y volver a graficar con valores respecto a los meses, basta con realizar la misma agrupación de datos, pero esta vez los valores del hecho a tener en cuenta para agrupar serán los de los meses, pero cabe destacar que en ninguna de las operaciones es necesario realizar transformaciones sobre los datos.

Las Figuras 5.1 y 5.2 presentan un indicador con las características mencionadas, en su formato de visualización antes y después de realizar la operación *roll up*

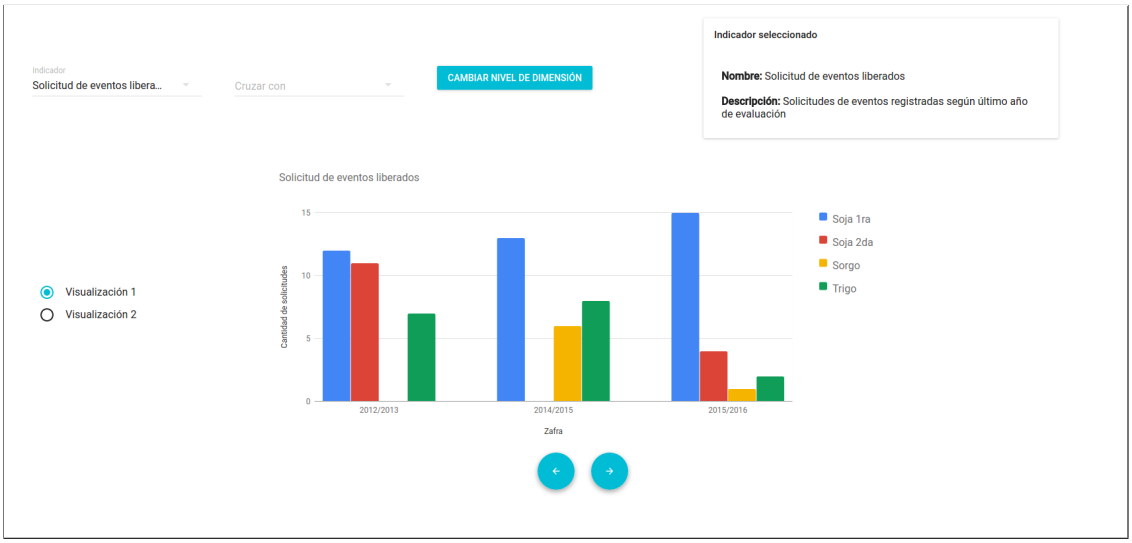


Figura 5.1: Visualización de un indicador de Área sembrada por cultivo según el año de cultivo

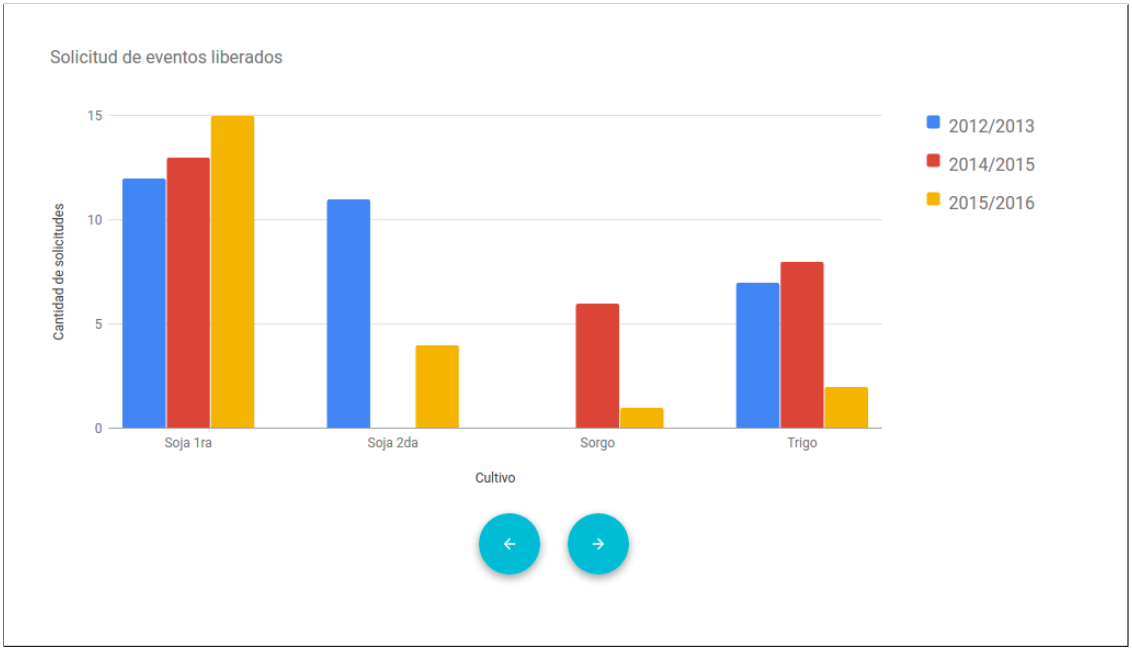


Figura 5.2: Visualización de un indicador de Área sembrada por tipo de cultivo según el año de cultivo, luego de realizar la operación *roll up* sobre la dimensión Tipo de Cultivo.

5.2.4. *Drill across*

Una vez que el usuario selecciona un indicador para visualizar, son listados también aquellos indicadores que son compatibles a ser graficados con él de forma simultánea. Como ya se dijo, dicha compatibilidad consiste en tener una o más dimensiones en común con el indicador graficado.

Luego, una vez seleccionado cualquiera de estos indicadores compatibles, debe realizarse la operación de *drill across* sobre los datos de ambos indicadores, para obtener una nueva agrupación de datos que cuente con toda la información.
Dados el indicador de Área sembrada para los cultivos maíz y soja según el año de sembrado, cuyas dimensiones son Tipo de cultivo y Tiempo respectivamente, y otro indicador llamado Solicitudes de eventos registrados, que modela la cantidad de solicitudes de autorización de vegetales genéticamente modificados, y que también cuenta con las dimensiones Tiempo y Tipo de cultivo, ya que refleja la cantidad de solicitudes realizadas para cada cultivo en cada año, por lo que podría ser cruzado con el indicador mencionado anteriormente.

Una vez seleccionados los dos indicadores a cruzar, se debe generar una nueva agrupación de datos que se corresponda con el formato de gráficas de dos ejes verticales, donde cada uno de ellos representa las medidas correspondientes a cada indicador; es decir, para cada valor representado en el eje horizontal deberá obtener dos valores asociados a él,

un valor para representar cada uno de los ejes verticales. Para obtener dicho conjunto, se deduce cuál de las dimensiones de ambos indicadores es la que tienen en común (en caso de ser ambas, se elije una de ellas), que en este caso será la dimensión Tiempo, y dicha dimensión estará reflejada en el eje horizontal (el eje compartido, de cierta forma), y luego las dos dimensiones restantes (una de cada indicador) corresponden a cada uno de los ejes verticales.

Entonces, para formar el conjunto de datos se debe obtener una lista de todos los miembros existentes en la dimensión común y que se graficarán en el eje horizontal, es decir, tanto los años para los cuales existen valores en al menos uno de los indicadores. Luego de obtenido dicho conjunto, se obtiene para cada valor del conjunto un valor para representar en cada eje vertical, correspondiente uno a cada indicador, y en caso de no existir valor para uno de los indicadores simplemente no se grafica nada.

De esta forma se obtiene una agrupación de datos donde para cada valor a graficar en el eje común, se cuenta con dos valores a graficar en los ejes verticales y que se representarán mediante dos barras acumuladas, y donde cada una de las acumulaciones corresponderá a un valor de la dimensión graficada y al valor de su medida.

En la Figura 5.3 puede verse un ejemplo de la gráfica del cruzamiento de los indicadores mencionados.

Figura 5.3: Visualización del cruzamiento de los indicadores de Área sembrada por tipo de cultivo según el año de cultivo y Solicitud de eventos registrados.

5.3. Restricciones establecidas

En la siguiente sección se detallarán un conjunto de restricciones que se encontraron necesarias para el correcto funcionamiento de la plataforma en cada uno de sus componentes. En particular, existen determinados requisitos que deben cumplir las entidades almacenadas (dimensiones, niveles, atributos, me didas) en base de datos para que el estado de la misma se mantenga consistente en todo momento, de manera que dichos requisitos son controlados por la plataforma tanto en el componente de *frontend* como en el de *backend*, y además se verán también qué características deben tener un par de indicadores para ser visualizados en simultáneo.

5.3.1. Restricciones de almacenamiento

La única restricción global en cuanto al almacenamiento de la información es que cada entidad debe contar con un nombre propio y único, en su contexto. Esto significa que cada nombre de indicador deberá ser único, así como también cada nombre de dimensión y cada nombre de nivel, y también cada nombre de atributo de nivel. Esto está controlado tanto por el *frontend* como por el *backend*.

Desde el *frontend* están restringidas a su vez las estructuras de los indicadores, ya que la implementación realizada para la visualización de indicadores soporta solamente un tipo reducido de estructuras posibles: Las restricciones son las siguientes:

1. Un indicador debe contar sí o sí con dos dimensiones exactamente, esto es necesario para que sea posible graficar sus datos en forma de columnas con un eje vertical y uno horizontal. Para graficar una sola dimensión deberíamos contar con una gráfica de líneas, y para graficar 3 o más dimensiones deberíamos contar con un filtrado que permita elegir qué dos dimensiones se desea graficar.

2. Para una dimensión dada, no pueden existir más de dos niveles de roll up. Es decir, una dimensión deberá tener como máximo 3 niveles.

3. Para un nivel, solo se podrá ingresar un atributo que luego será el graficado. Para agregar varios atributos, se debería contar con un filtro previo a graficar que permita elegir qué atributo del nivel se requiere graficar.

4. Para las operaciones de *roll up* sobre los valores de una dimensión, siempre se utiliza la función de agregación de la suma. Para agregar distintas funciones de agregación se

deberían listar a la hora de crear una dimensión, y luego diferencial según la función ingresada a la hora de realizar la operación.

5.3.2. Restricciones de cruzamiento

Para poder cruzar la información correspondiente a dos indicadores, es necesario que ambos compartan al menos una de sus dimensiones. De esta forma se puede graficar un eje compartido (el horizontal) y dos ejes distintos (dos ejes verticales) correspondientes a cada uno de los indicadores. Si dichos indicadores comparten ambas dimensiones, los ejes verticales serán idénticos.

6. Pruebas realizadas

En el siguiente capítulo se detallarán un conjunto de pruebas realizadas luego de finalizada la implementación de la plataforma, con el objetivo de medir la capacidad de la misma en relación a la carga de datos y los distintos tiempos de respuesta de la aplicación en el manejo de las distintas cargas, así como también a posibles limitantes en cuanto a la visualización gráfica de los indicadores. Para esto se verá en detalle cada una de las pruebas realizadas, cuál fue su objetivo y el resultado de las mismas. Las pruebas fueron realizadas en una computadora con sistema operativo Ubuntu 16.04, que cuenta con un procesador Intel Core i7 y 12 GB de memoria RAM.

6.1. Pruebas de carga

Se realizó una prueba de carga de datos para un indicador específico, monitoreando los tiempos de respuesta, con el objetivo de obtener parámetros que puedan provocar la sobrecarga de las distintas secciones de la aplicación. Para esto fue creado un indicador con dos dimensiones de un nivel cada una de ellas, y se ingresaron datos para dicho indicador mediante la funcionalidad de importar datos como CSV (es decir, se creó un archivo CSV de 1000 filas con los valores del nivel 1, nivel 2 y su medida correspondiente).
Fueron realizadas dos cargas de datos, de la siguiente manera:

Primera carga de datos

En esta primera instancia fueron ingresados 500 datos para el indicador mencionado anteriormente, y mediante el uso de la aplicación luego del ingreso fueron obtenidos los tiempos de respuesta, ilustrados en el Cuadro 6, realizando un promedio de los resultados arrojados luego de 10 pruebas para cada uno de los casos.

Operación	Tiempo de respuesta (segundos)
Inserción del conjunto de datos	7.3
Lectura del conjunto de datos	4.5

Lectura de datos y posterior graficado	2.2
Eliminación de datos	2.5

Cuadro 6: Primera carga de datos

Luego de esta carga de datos se puedo ver que la aplicación no tuvo mayores problemas para seguir funcionando de manera correcta, con tiempos de respuesta aceptables, siendo el mayor de ellos la inserción en base de datos, como era de esperar. Cabe destacar que los tiempos de respuesta listados son los mayores encontrados, por lo que fueron omitidos otros tiempos de respuesta como la navegación entre pestañas, el cruzamiento del indicador en cuestión con otro indicador, el cambio de tipo de visualización de la gráfica del indicador, etcétera.

Segunda carga de datos

En esta segunda instancia fueron ingresados 1000 datos para el indicador mencionado anteriormente, y mediante el uso de la aplicación luego del ingreso fueron obtenidos tiempos de respuesta representados en el Cuadro 7.

Operación	Tiempo de respuesta (segundos)
Inserción del conjunto de datos	14.8
Lectura del conjunto de datos	8.4
Lectura de datos y posterior graficado	6.4
Eliminación de datos	4.8

Cuadro 7: Segunda carga de datos

Se puede ver que los tiempos de respuesta son altos para el uso normal de una aplicación web, siendo el más alto de ellos el que está encargado de insertar dicha información en base de datos, que resultó de una demora considerable. Por otra parte, de acuerdo a como funciona React, que almacena toda la información en memoria para poder realizar una nueva renderización cada vez que se genera un cambio en la interfaz, el uso de la aplicación se vuelve poco práctico y con tiempos de respuesta no recomendables para una aplicación web, por ejemplo, al momento de navegar desde una pestaña a otra.

Entendemos que el tiempo de respuesta al insertar en base de datos no es algo que pueda ser mejorado desde la aplicación, por lo que creemos que la mejor manera de mitigar dicha demora es realizar la inserción de forma asincrónica.

Por otro lado, el inconveniente más visible sucede a raíz de la lectura de los datos y su posterior almacenamiento en la memoria del navegador, ya que el tiempo que toma realizar una gráfica es el menor de todos, y el posterior uso de la gráfica con todas sus funcionalidades no generaron ningún tipo de demora trascendente (al cambiar el tipo de visualización de la gráfica, o al cruzar dicha gráfica con otro indicador, por ejemplo). Para solucionar esto sería posible realizar un paginado sobre los datos, ya que en este momento todos los datos de un indicador son obtenidos en un solo llamado al *backend*. De esta forma, se produciría un uso más limitado de la memoria del navegador y eso conllevaría a un mejor tiempo de respuesta en todas las funcionalidades.
Otra posible mejora a esta dificultad sería el uso de una memoria *caché* externa, que almacene los datos del indicador y libere al navegador de esa carga de datos, ya que ese tipo de memorias se caracterizan por ser de acceso rápido y no impactaría en el tiempo de respuesta del resto de la aplicación.

Como conclusión de esta prueba de carga se puede decir que tanto la base de datos utilizada como el componente de *frontend* permiten una cantidad limitada de información para funcionar en forma correcta o aceptable, mientras que a partir de cierto punto (del orden de los mil datos, para un indicador dado) sería necesario realizar mejoras de infraestructura para que la aplicación se mantenga estable.

6.2. Pruebas de visualización

Para realizar pruebas relacionadas a la visualización de indicadores en forma de gráfica, fueron utilizados los datos obtenidos de la carga de datos de la Sección 6.1, que resultó en un indicador que cuenta con 1000 datos a graficar.
Dichos datos se organizan de la siguiente manera de acuerdo a sus dimensiones:

- Dimensión Año: se cuentan con datos desde 1960 hasta 2015, con datos para cada uno de los años sin espacios en blanco.
- Dimensión Cultivo: se cuentan con 15 tipos de cultivos diferentes, con un dato para cada cultivo para cada año.

Los resultados graficados fueron los siguientes:

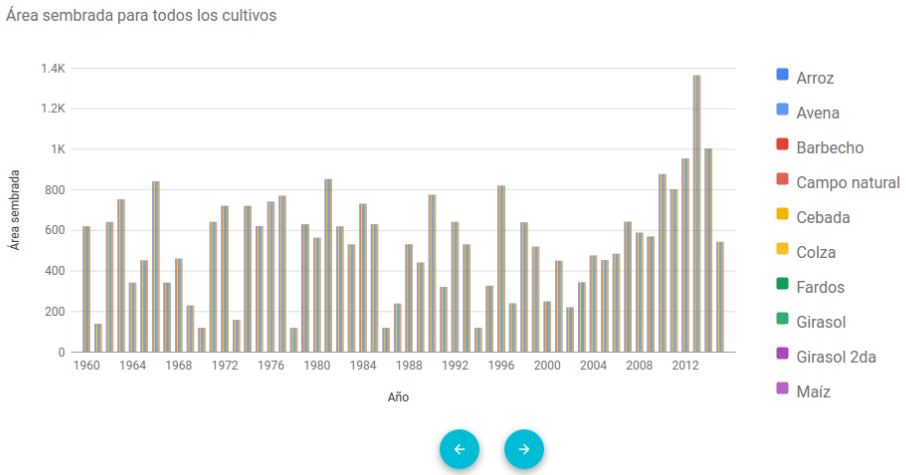


Figura 6.1: Visualización de un indicador con 1000 datos, con la dimensión Año en el eje horizontal y Cultivo en el eje vertical. Cada columna representa un cultivo diferente.

Observando la Figura 6.1, nos encontramos con una limitante en cuanto a la cantidad de columnas graficadas para un indicador. Mientras que en el eje horizontal la gráfica se encarga de generar columnas más finas y reducir la cantidad de leyendas (se muestran leyendas para 1960 y 1964, pero no para los años entre medio), en el eje vertical la forma de agrupar columnas para un determinado año no es otra que achicar el ancho de cada columna y reducir el tamaño entre ellas.
Para el indicador en cuestión, que cuenta con 15 o más tipos de cultivos distintos, resulta

imperceptible a simple vista ver el valor de cada cultivo, ya que las columnas son indistinguibles entre sí. Sin embargo, es posible conocer el valor para cada cultivo pasando con el puntero del ratón por sobre cada conjunto de columnas.

Además, se encontró una limitante en cuanto a las leyendas del eje vertical, ya que como puede verse en la figura solo 10 cultivos de los 15 totales. Esto se debe a la versión de Google Charts utilizada, que se encuentra aún en su versión *Beta* y todavía no cuenta con paginación de las leyendas, algo que sí se encuentra en sus anteriores versiones.

También puede verse que este tipo de gráfica cuenta solamente con 5 colores, azul, rojo, amarillo, verde y violeta, y que en caso de existir más valores que dicha cantidad simplemente se obtienen mediante otras tonalidades de los mismos colores ya mencionados.

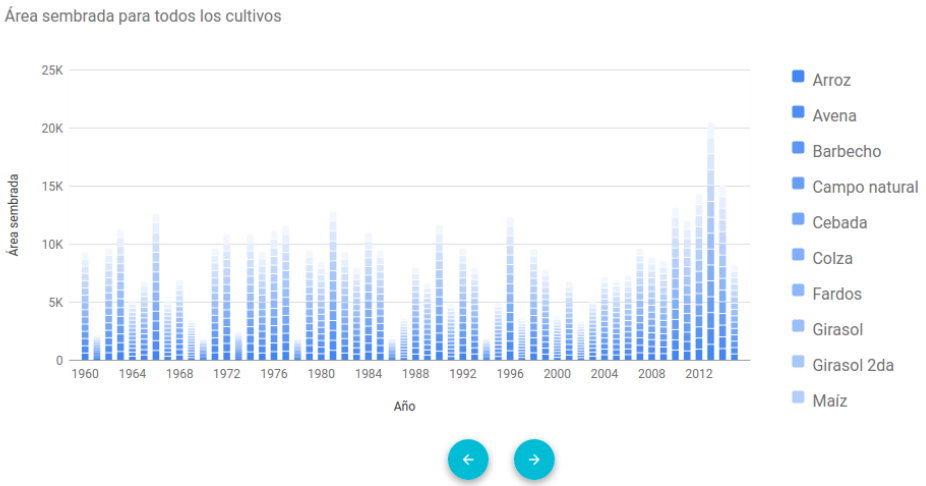


Figura 6.2: Visualización de un indicador con 1000 datos, con la dimensión Año en el eje horizontal y Cultivo en el eje vertical en formato *stack*. Cada *stack* representa un cultivo diferente y cada columna un año diferente.

Observando la Figura 6.2 pueden verse limitantes similares a la figura anterior en cuanto a las leyendas del eje vertical, pero además en este tipo de visualización, donde cada tipo de cultivo es un *stack*, ocurre que cada columna deberá representar los 15 distintos cultivos mediante distintas tonalidades del mismo color, en este caso el azul. Como puede verse en la gráfica, esta convención no es fácil de reconocer a simple vista y ocurre que hay cultivos cuya tonalidades es casi transparente y poco visible en la gráfica.

En la Figura 6.3 podemos ver graficados los mismos datos que en figuras anteriores pero cambiando el orden de los ejes, de manera que los años están representados en el eje vertical. Como se puede ver, de los 55 años posibles solo se muestra la leyenda de 10 de ellos, y además, para cada cultivo existen 55 tipos de tonalidad de azul diferentes, por lo

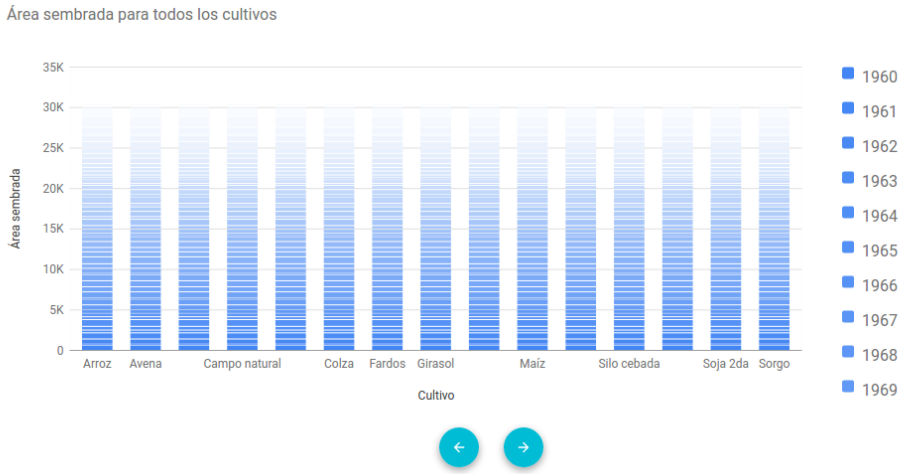


Figura 6.3: Visualización de un indicador con 1000 datos, con la dimensión Año en el eje vertical en formato *stack* y Cultivo en el eje horizontal. Cada *stack* representa un año diferente y cada columna un cultivo diferente.

que no es posible interpretar los datos a simple vista.

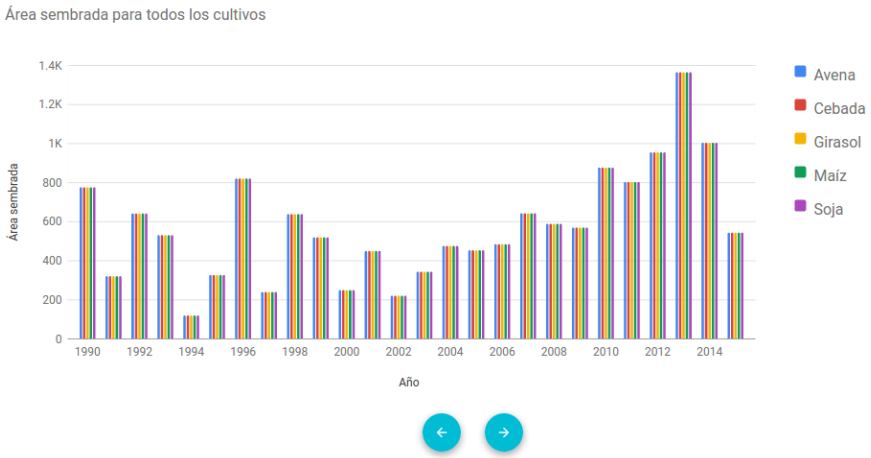


Figura 6.4: Visualización de un indicador con 150 datos, con la dimensión Año en el eje horizontal y Cultivo en el eje vertical.

La Figura 6.4 corresponde a una carga de 150 datos para el mismo indicador que las figuras anteriores, con la diferencia que los datos van desde el año 1990 hasta 2015 y para 5 cultivos diferentes.

A diferencia de las figuras anteriores, en esta sí pueden interpretarse los datos para cada una de las columnas a simple vista, pero con cierta dificultad. Si quisieran agregarse datos para más años o más cultivos, ya se haría cada vez más difícil para el usuario de interpretar.

Como puede observarse en las figuras 6.5 y 6.6, Google Charts provee la posibilidad de

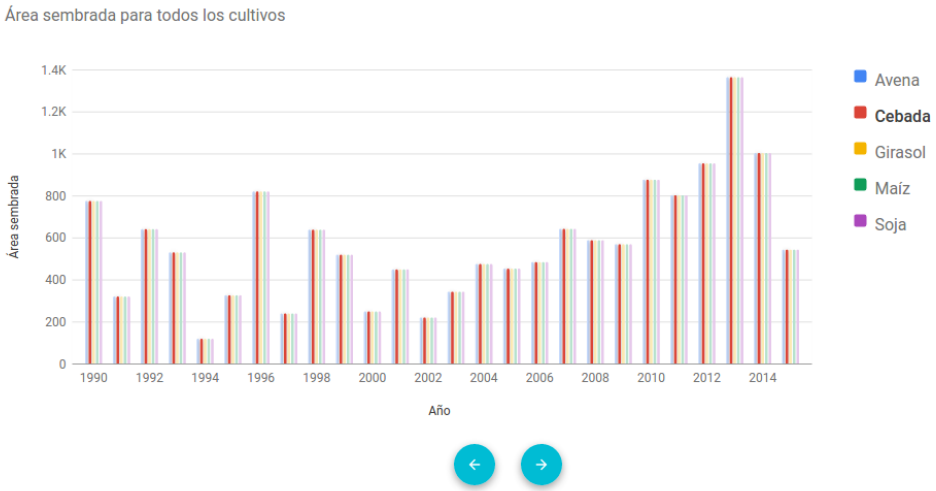


Figura 6.5: Visualización de un indicador con 157 datos, con la dimensión Año en el eje horizontal y Cultivo en el eje vertical y el Cultivo 'Cebada' seleccionado.

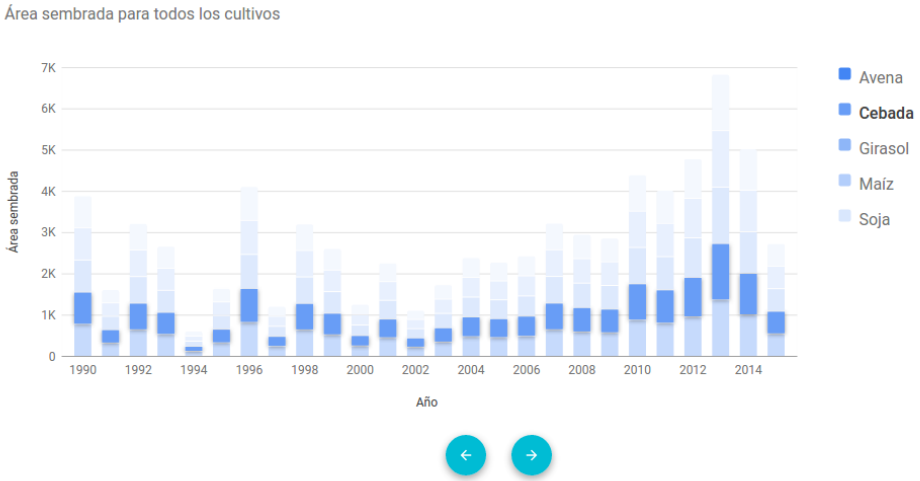


Figura 6.6: Visualización de un indicador con 150 datos, con la dimensión Año en el eje horizontal y Cultivo en el eje vertical en formato *stack*, y el Cultivo 'Cebada"seleccionado.

seleccionar una de las leyendas para el eje vertical, de manera que el resto de los valores son transparentados y el valor seleccionado es resaltado para mejor visualización. De esa manera sí resulta interpretable para el usuario la información correspondiente a dicha leyenda, sin importar el ancho que puedan tener las columnas o los *stacks* correspondientes a cada valor o el color de los mismos.

7. Conclusiones y Trabajo a futuro

En esta sección se realizará una evaluación del producto final implementado, detallando cuáles fueron los resultados alcanzados en comparación a los objetivos planteados en un principio, haciendo hincapié en cada una de las funcionalidades logradas. Además se detallarán las principales dificultades que debieron ser superadas a lo largo del proyecto, tanto en etapas de diseño como en la implementación de cada uno de los componentes. Por último, se describirá una lista de posibles extensiones a realizar sobre el producto entregado en un futuro, describiendo cuáles son consideradas como las mayores falencias de la plataforma y qué mejoras se podrían realizar para mitigar dichas falencias, así como también se describirán un conjunto de nuevas funcionalidades que fueron consideradas en etapas tempranas del proyecto pero que no entraron dentro del alcance del mismo.

7.1. Resultados alcanzados

El trabajo realizado tiene como resultado una plataforma *web* que alcanza a cumplir los objetivos planteados al inicio del proyecto, dentro de ciertas restricciones. Cumple con el objetivo principal de permitir que un usuario ingrese, de forma dinámica, modelos multidimensionales y los mismos sean almacenados en una base de datos acorde. Luego, para completar el ciclo de uso básico y principal, permite también ingresar información correspondiente a estos modelos para luego poder visualizarlos de forma gráfica y también, dinámica.

Luego, al momento de visualizar la información en forma gráfica, si bien originalmente no hubo requerimientos específicos en cuanto a cómo deberían verse los datos, se logró implementar una plataforma que permite al usuario observar el mismo conjunto de datos de diversas maneras, cambiando tanto el tipo de gráfica como la disposición de los ejes de la misma, como también realizando distintas agrupaciones de los mismos mediante las funciones de agregación, logrando resultados completamente distintos y complementarios, permitiendo realizar un análisis de la información desde distintos puntos de vista.

Por otro lado, uno de los grandes desafíos presentados por el cliente era también la posibilidad de poder comparar información correspondiente a distintos indicadores ingresados ('cruzar' la información), algo que también se alcanzó a lograr siempre y cuando se cumplan las condiciones necesarias para poder realizar la comparativa entre dos indicadores distintos, como ya fue mencionado anteriormente.

Con el objetivo de brindar una mejor experiencia de uso a los usuarios, también se

propuso como objetivos que tanto el ingreso de datos como la posterior disponibilidad de los mismos (exposición como datos abiertos) sea de la forma más amigable posible. Es por eso que se introdujeron las funcionalidades de ingresar datos masivos en un formato estándar como es el CSV, que le significa al usuario una mayor agilidad por no tener que ingresar los datos uno por uno; mientras que en cuanto a la disponibilidad de los mismos se logró tener tanto disponibilizar datos de calidad y en formato estándar (pudiendo descargar los modelos y sus datos en formato RDF), para poder ser tanto visualizados como reutilizados por cualquier agente externo, así como la posible descarga en formato CSV de los datos de cada modelo por separado, algo que permite realizar otro tipo de lecturas sobre los mismos por usuarios externos y ajenos a formatos más complejos como el RDF.

7.2. Dificultades encontradas y soluciones implementadas

7.2.1. *Backend*

Tiempos de respuesta El componente de *frontend* de la aplicación cuenta con distintos combos, los cuales se completan con datos que se obtienen desde la base de datos de grafos a la que accede el componente de *backend*. En estos combos se listan los esquemas y las medidas dependiendo de lo que requiera el usuario. Con la arquitectura actual del sistema, los tiempos de respuesta del *backend* se acercaban a los 10 segundos, lo que causaba un detrimento de la performance del mismo. Es por este motivo que se recurrió a implementar una memoria cache para estos grafos (de esquemas y medidas) con el fin de dinamizar la experiencia del usuario. Con estas modificaciones se logro disminuir significativamente los tiempos de respuestas, alcanzando valores menores al segundo.

Consultas dinámicas Generar consultas SPARQL de forma dinámica resultó ser una de las dificultades encontradas debido a que en algunas situaciones la información se encuentra separada en más de un grafo. En estos casos, la solución general utilizada es realizar primero una consulta sobre el grafo que contiene la metadata del indicador y luego con la información obtenida generar una nueva consulta sobre el grafo que contiene la información solicitada. Un ejemplo de este caso es cuando el usuario solicita ver todos los hechos de un indicador, desde la capa de *frontend* solamente nos envían el nombre del indicador y desde *backend* se debe realizar primero una consulta sobre el grafo que contiene la metadata del indicador con el propósito de obtener las URIs de los niveles base y la medida del indicador consultado, para luego, con las URIs obtenidas generar una nueva consulta SPARQL y

ejecutarla sobre el grafo que contiene los hechos del indicador.

7.2.2. *Frontend*

En el componente de *frontend* las dificultades encontradas pueden dividirse de acuerdo a los tres mayores desafíos del componente que son los presentados a continuación.

Diseño de la interfaz En primera instancia lograr diseñar una interfaz que permita modelar los cubos multidimensionales, y que a su vez dicha interfaz sea lo más intuitiva posible. Luego, poder también diseñar una interfaz de comunicación con el componente de *backend* que sea lo suficientemente genérica como para cumplir con todos los requisitos de dichos modelos.

Para el diseño de la interfaz de usuario lo primero que se hizo fue identificar todos los componentes que implican un modelo multidimensional definido mediante la ontología QB4OLAP, deteniéndonos sobre todo en aquellos indicadores que el cliente necesitaba modelar y que luego serían necesarios repicar.
Luego, tomando como base varios indicadores de los que fueron proporcionados por el cliente, y traduciéndolos a la ontología mencionada, se identificaron cuáles eran las entidades que necesariamente debían ser ingresadas por el usuario y cuáles podrían ser entidades autogeneradas sin la necesidad de que el usuario comprenda su complejidad.
En base a eso, se llegó a la conclusión de que un cubo multidimensional podía generarse a partir de una entrada formada por un conjunto de dimensiones, donde cada dimensión tendrá a su vez un conjunto de niveles y cada nivel un conjunto de atributos. Luego, por cuestiones de complejidad al momento de la visualización, se optó por limitar esa lista de atributos a un solo atributo.

A partir de dicha conclusión, fue fácil diseñar una interfaz de comunicación con el componente de *backend*, ya que bastó con identificar qué campos era necesarios para cada una de estas entidades (nombre, descripción, etc.), y traducirlo a un lenguaje estándar para intercambio de datos como lo es *JSON*. Ya que *JSON* soporta los tipos de lista o arreglos, y objetos de varios atributos, la traducción resultó ser lineal.

Diseño de la visualización Por otro lado lograr una interfaz de visualización de estos indicadores, que sea dinámica e intuitiva, y que además logre aportarle valor al usuario.
Para lograrlo, se utilizaron también como base una serie de los indicadores proporcionados por el cliente, y se estudió qué cantidad de dimensiones y niveles tenía cada uno y qué tipo de datos necesitaban representar. De esta forma se concluyó que los indicadores debían so-

portar un mínimo de dos dimensiones y un máximo de tres, mientras que cada dimensión contaba con uno o dos niveles. Por otra parte, todos los indicadores contaban con medidas numéricas.

A partir de dicha información fue que concluimos que en la mayoría de los casos lo mejor sería implementar un tipo de gráfica de barras, donde el eje horizontal fuera ocupado por una de las dimensiones, y luego cada una de las barras se correspondiera con un valor distinto de la segunda dimensión. Por último, el eje vertical sería utilizado para representar las medidas numéricas de cada indicador.

En el caso de los indicadores con tres dimensiones, no fue posible encontrar una herramienta que permita graficar tres dimensiones de forma simultánea, y si bien se buscaron alternativas que podían basarse en la posibilidad de elegir cuáles dos de entre las tres dimensiones se quería graficar, pudimos ver que en los casos presentados por el cliente en ningún caso se contaba con tres dimensiones independientes, sino que se contaba con casos donde una de las tres dimensiones se podía encapsular dentro de otra. De esta forma, optamos por modelar dichos indicadores de esta forma y permitir la visualización de cada dimensión mediante el uso de las funciones de agregación de estas dimensiones.

Cruzamiento de indicadores Otra de las dificultades encontradas al momento de visualizar los datos fue encontrar la mejor manera de graficar dos indicadores de forma simultánea, para permitir el cruzamiento de los datos que formaba parte de los requerimientos principales del cliente.

Para esto se estudiaron diversas formas de graficar dos conjuntos distintos de datos en forma simultánea, y qué requisitos deberían cumplir dichos datos. Dado que cada indicador cuenta con una medida numérica y que dicha medida se veía representada en el eje vertical de cada gráfica, encontramos que era una forma óptima de representar ambas medidas en forma simultánea mediante una gráfica que cuente con dos ejes verticales. De esta manera, se contaría con un eje para cada medida y cada eje contaría a su vez con su propia escala, de forma que no sería necesario transformar los datos.

Partiendo desde esa base, se determinó también que los indicadores que fueran a utilizarse en este cruzamiento debieran compartir como mínimo una de sus dimensiones. Esto es debido a que las gráficas cuentan con un eje horizontal que no puede duplicarse como el vertical, por lo que sería necesario contar con una dimensión de referencia a partir de la cuál se pudieran agrupar ambos conjuntos de datos de forma que pueda obtenerse la gráfica de doble eje vertical. Por otro lado, la dimensión restante a graficar no será necesariamente compartida, ya que la segunda dimensión es representada en las gráficas en forma

de columnas de distintos colores, de forma que para representar dos dimensiones distintas (una de cada indicador) en dicho formato, solamente es necesario contar con la cantidad de colores necesarios y generar una columna en base a cada valor posible de cada una de las dimensiones.

Además, debido a que se pudiera dar el caso en que ambas dimensiones (las no compartidas) contaran con un conjunto grande de valores posibles, y esto generaría una gráfica con demasiadas columnas de distintos colores, se logró también contar con otro tipo de representación de estas dimensiones 'secundarias'. Dicha representación constará de solamente un color para cada dimensión, es decir, siempre habrá dos columnas de colores diferentes, donde cada columna estará a su vez conformada por un conjunto de *stacks*, donde cada *stack* representará un valor posible para dicha dimensión y su altura representará el valor de la medida para dicho valor. Un ejemplo de este tipo de visualización puede verse en la Figura 4.17.

7.3. Posibles extensiones

Dado que el componente de *backend* y todo lo relacionado a la base de datos de los indicadores multidimensioanes fue diseñado de manera genérica y respetando la ontología QB4OLAP, todas las posibles extensiones de la plataforma estarían relacionadas a mejorar la infraestructura de la plataforma o al componente de *frontend*, debiendo contemplar luego las posibles modificaciones necesarias en los servicios expuestos por el *backend*.

En cuanto al componente de *frontend*, hay una conjunto de extensiones planteadas como una posibilidad a futuro:

- **Indicadores de N dimensiones:** Dado que hoy en día la plataforma está restringida para graficar solamente modelos de dos dimensiones, se podría extender a que pueda graficar modelos de N dimensiones. En principio, se plantea la posibilidad de graficar indicadores de tres dimensiones donde se podrá seleccionar dos de ellas para visualizar en las gráficas que ya existen, y la tercera dimensión se vería representada en un filtro por fuera de la gráfica, en el que se deberá elegir qué nivel de dimensión se desea visualizar, para luego elegir uno de los valores posibles para dicho nivel. De esa forma, se filtarán los datos de las otras dos dimensiones de acuerdo al valor de la tercera.

- **Niveles con N atributos:** Dado que la ontología QB4OLAP permite que cada nivel definido tenga un conjunto de atributos, y que esto está contemplado en base de datos, será posible extender la interfaz de usuario para modelar dicho conjunto, ya que hoy en día cada nivel cuenta solamente con un atributo. Esto implica no solo

cambios en el ingreso de un nuevo modelo sino que en la visualización de las gráficas, ya que también debería existir la posibilidad de elegir cuál de dichos atributos se desea graficar para cada uno de los niveles que conforman las gráficas.

- **Nuevas gráficas de cruzamientos:** Para el cruzamiento de indicadores, existe hoy en día solamente un tipo de visualización (aunque en dos modos distintos) y este es el tipo de gráficas de barras. Existe también la posibilidad de extender la interfaz de visualización a otros tipos de gráficas como podrían ser las gráficas de líneas, donde cada color representaría una línea que atraviesa la gráfica en vez de múltiples columnas.

- **Indicadores georeferenciados:** Dentro de los indicadores proporcionados por el cliente, existía un tipo de indicador que contenía datos georeferenciados y representados en forma de mapa. Esto implica un modelo completamente distinto de almacenamiento y una nueva interfaz de usuario, tanto para el ingreso de datos como para su visualización, pero también es una de las posibilidades de extensión a futuro.

- **ABM de usuarios:** La plataforma implementada fue pensada para que un usuario administrador sea el responsable de ingresar los indicadores y sus datos, dejando abierto a cualquier usuario externo el acceso a dicha información, tanto para visualizarla como descargarla. Sin embargo, podría a futuro extender al componente de *frontend* de manera que tenga un manejo de íntegro de los usuarios, es decir, pueda crear y eliminar usuarios, otorgando distintos tipos de permisos a cada uno de ellos de acuerdo se crea conveniente. De esta manera, podría extenderse a más de un usuario la responsabilidad de ingresar nuevos indicadores y datos a la aplicación.

- **Operaciones OLAP:** Dado que las operaciones OLAP soportadas por la plataforma hoy en día son las de *roll up*, *drill down* y *drill across*, dichas operaciones podrían ser extendidas también para usar el resto de las operaciones existentes, como puede ser *slice*, *dice*, *sort* o *pivot*.

8. Anexo: Máquina virtual

A continuación se detallan las dos opciones posibles para la inicialización del sistema:

8.1. Inicialización automática

Dado que el sistema está contenido dentro de una máquina virtual, la cual contiene la aplicación de backend y frontend, no será necesario ejecutar ningún comando para iniciar las aplicaciones. Por el contario, cada vez que la máquina virtual se inicia o se reinicia, correrá automáticamente un script que se encarga de ello.

8.2. Inicialización manual

8.2.1. Backend

En caso de querer iniciar manualmente el backend, se debe primeramente acceder desde la consola a la ruta /home/sigmu/Downloads/sigmu.10.06/sigmu-web". Una vez allí ejecutar el comando "java -jar target/sigmu-web-1.0.jar 59010", este comando dejará disponible el backend escuchando en el puerto 59010 por defecto. En caso de querer dejarlo disponible en otro puerto se deberá modificar el último parámetro del comando. De todas formas, cabe mencionar que es importante mantener este puerto para el correcto funcionamiento del sistema, ya que es a través de este puerto que el frontend realizará los pedidos.

8.2.2. Frontend

Para iniciar manualmente el frontend se debe acceder a la ruta /home/sigmu/Downloads/sigmu-frontend-master". Luego es necesario ejecutar el comando "npm start". De esta manera, el frontend queda disponible en el puerto 3000 por defecto.

Una vez completada la inicialización tanto del backend como del frontend se podrá acceder al sistema desde cualquier browser, accediendo a la ip asignada a la maquina virtual y el puerto 3000.

9. Anexo: Casos de prueba entre Virtuoso y RDF4J

En esta sección se presentan los cinco casos de prueba ejecutados para medir los tiempos de respuesta al insertar/obtener triplas utilizando Virtuoso y RDF4J. Para cada caso de prueba se realizaron cuatro ejecuciones con cada tecnología con el proposito de obtener un tiempo promedio de cada una.

9.1. Características del sistema

En la tabla 8 se muestran las características del sistema donde se realizaron las pruebas.

Sistema operativo	MAC OS Sierra Versión 10.12.6
Procesador	2.2 GHz Intel Core i7
Memoria RAM	16 GB 1600 MHz DDR3
Disco duro	256 GB SSD

Cuadro 8: Características del sistema

9.2. Caso 1

La tabla 9 refleja los tiempos de respuesta al insertar/obtener 1 tripla desde RDF4J y Virtuoso.

Caso 1	Virtuoso		RDF4J	
Nº Ejecución	Insertar(ms)	Obtener(ms)	Insertar(ms)	Obtener(ms)
1	357	468	491	494
2	347	464	506	490
3	352	465	494	483
4	352	472	491	481
Promedio	352	467.25	495.5	487

Cuadro 9: Caso de prueba 1

9.3. Caso 2

La tabla 10 refleja los tiempos de respuesta al insertar/obtener 5 triplas consecutivas desde RDF4J y Virtuoso.

Caso 2	Virtuoso		RDF4J	
Nº Ejecución	Insertar(ms)	Obtener(ms)	Insertar(ms)	Obtener(ms)
1	414	460	510	483
2	362	467	527	484
3	357	472	506	488
4	358	470	505	479
Promedio	372.75	467.25	512	483.5

Cuadro 10: Caso de prueba 2

9.4. Caso 3

La tabla 11 refleja los tiempos de respuesta al insertar/obtener 100 triplas consecutivas desde RDF4J y Virtuoso.

Caso 3	Virtuoso		RDF4J	
Nº Ejecución	Insertar(ms)	Obtener(ms)	Insertar(ms)	Obtener(ms)
1	482	481	606	520
2	468	493	607	497
3	466	476	610	490
4	470	490	617	492
Promedio	471.5	485	610	499.75

Cuadro 11: Caso de prueba 3

9.5. Caso 4

La tabla 12 refleja los tiempos de respuesta al insertar/obtener 500 triplas consecutivas desde RDF4J y Virtuoso.

Caso 4	Virtuoso		RDF4J	
Nº Ejecución	Insertar(ms)	Obtener(ms)	Insertar(ms)	Obtener(ms)
1	858	493	815	507
2	855	495	817	507

3	871	483	820	504
4	870	492	813	518
Promedio	863.5	490.75	816.25	509

Cuadro 12: Caso de prueba 4

9.6. Caso 5

La tabla 13 refleja los tiempos de respuesta al insertar/obtener 1000 triplas consecutivas desde RDF4J y Virtuoso.

Caso 5	Virtuoso		RDF4J	
Nº Ejecución	Insertar(ms)	Obtener(ms)	Insertar(ms)	Obtener(ms)
1	1422	517	1126	566
2	1432	500	1128	536
3	1402	500	1139	532
4	1384	503	1143	533
Promedio	1410	505	1134	541.75

Cuadro 13: Caso de prueba 5

Referencias

[1] Detection of genetically modified organisms (GMOs) by PCR: a brief review of methodologies available. *Trends in Food Science & Technology*, 9(11):380 – 388, 1998.

[2] AGESIC. Catálogo de Datos Abiertos del Estado Uruguayo. `https://catalogodatos.gub.uy/`. [Último acceso 09-08-2017].

[3] Elke Anklam, Ferruccio Gadani, Petra Heinze, Hans Pijnenburg, and Guy Van Den Eede. Analytical methods for detection and determination of genetically modified organisms in agricultural crops and plant-derived food products. *European Food Research and Technology*, 214(1):3–26, Jan 2002.

[4] Tim Berners-Lee. 5-star Open Data. `http://5stardata.info/en/`. [Último acceso 10-07-2017].

[5] Tim Berners-Lee. Cool URIs. `https://www.w3.org/Provider/Style/URI`. [Último acceso 10-07-2017].

[6] Tim Berners-Lee. Linked Data. `https://www.w3.org/DesignIssues/LinkedData.html/`. [Último acceso 10-07-2017].

[7] S. Broeders, I. Huber, L. Grohmann, G. Berben, I. Taverniers, M. Mazzara, N. Roosens, and D. Morisset. Guidelines for validation of qualitative real-time PCR methods. *Trends in Food Science & Technology*, 37(2):115 – 126, 2014.

[8] CubesViewer. CubesViewer. `http://www.cubesviewer.com/`. [Último acceso 20-08-2017].

[9] Eurostat. Eurostat. `http://ec.europa.eu/eurostat`. [Último acceso 19-07-2017].

[10] Arne Holst-Jensen. Testing for genetically modified organisms (GMOs): Past, present and future perspectives", journal = "Biotechnology Advances. 27(6):1071 – 1082, 2009. Biotechnology for the Sustainability of Human Society.

[11] Apache Jena. Jena. `https://jena.apache.org/`. [Último acceso 20-08-2017].

[12] Meteorite.Bi. Saiku software. `http://www.meteorite.bi/products/saiku`. [Último acceso 20-08-2017].

[13] MGAP. Dirección General de Control de Inocuidad Alimentaria. `http://www.mgap.gub.uy/unidad-ejecutora/`

direccion-general-de-control-de-la-inocuidad-alimentario. [Último acceso 09-08-2017].

[14] ODaF. Open Data Foundation. http://www.opendatafoundation.org/. [Último acceso 20-07-2017].

[15] Spring. Spring. http://projects.spring.io/spring-framework/. [Último acceso 13-08-2017].

[16] Tableau. Tableau Software. https://www.tableau.com/. [Último acceso 20-08-2017].

[17] Alejandro A. Vaisman and Esteban Zimányi. *Data Warehouse Systems - Design and Implementation*. Data-Centric Systems and Applications. Springer, 2014.

[18] Lorena Etcheverry Venturini. *QB4OLAP: Enabling Business Intelligence over Semant ic Web Data*. PhD thesis, Tesis de Doctorado en Informática, PEDECIBA Instituto de Computación, Facultad de Ingeniería Universidad de la República. Montevideo, Uruguay, ISSN 0797-6410 Reporte Técnico RT 16-10, 2016.

[19] W3C. Basic Graph Pattern. https://www.w3.org/TR/rdf-sparql-query/#GraphPattern. [Último acceso 13-08-2017].

[20] W3C. RDF-Turtle. https://www.w3.org/2007/02/turtle/primer/. [Último acceso 13-08-2017].

[21] W3C. SPARQL 1.1. https://www.w3.org/TR/sparql11-query/. [Último acceso 29-07-2017].

[22] Richard Cyganiak y Dave Reynolds. The RDF Data Cube Vocabulary. https://www.w3.org/TR/vocab-data-cube/. [Último acceso 10-07-2017].

[23] Ian Jacobs y Norman Walsh. Architecture of the World Wide Web. https://www.w3.org/TR/webarch/. [Último acceso 21-07-2017].

[24] Dan Brickley y Ramanathan Guha. RDF *Schema* 1.1. https://www.w3.org/TR/2014/REC-rdf-schema-20140225/. [Último acceso 10-07-2017].

[25] Dave Beckett y Tim Berners-Lee. RDF 1.1 *Turtle*. https://www.w3.org/TR/2014/REC-turtle-20140225/. [Último acceso 10-07-2017].