Hromek, Erik

Aporte para el área de procesos del área de operaciones

2019

Instituto: Ingeniería y Agronomía

Carrera: Ingeniería en Informática

PRÁCTICA PROFESIONAL SUPERVISADA (PPS)

Aportes para la mejora de procesos del área de Operaciones
Informe Final
Erik Hromek

DATOS DEL ESTUDIANTE
Apellido y Nombres: Hromek, Erik
DNI: 39812144
Nº de Legajo: 22843
Correo electrónico: erikhromek96@gmail.com
Cantidad de materias aprobadas al comienzo de la PPS: 43
PPS enmarcada en artículo 4 de la Resolución (CS) 103/16

DOCENTE SUPERVISOR
Apellido y Nombres: Prof. Dr. Morales, Martín Daniel
Correo electrónico: martin.unaj@gmail.com

DOCENTE TUTOR DEL TALLER DE APOYO A LA PRODUCCIÓN DE TEXTOS ACADÉMICOS DE LA UNAJ
Apellido y Nombres: Prof. Dra. Ferrari, Mariela Cristina
Correo electrónico: mariela_c_ferrari@hotmail.com

DATOS DE LA ORGANIZACIÓN DONDE SE REALIZA LA PPS
Nombre o Razón Social: JulaSoft S.A.
Dirección: Calle 9 N.º 811, esq. 524, B1906 Tolosa, Buenos Aires
Teléfono: (+54) 221 445.2344
Sector: Área de Operaciones

TUTOR DE LA ORGANIZACIONAL
Apellido y Nombres: Mg. Catucci, Miguel Adolfo
Correo electrónico: miguelcatucci@julasoft.com

FIRMA DEL COORDINADOR DE LA CARRERA

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Índice de figuras

1. Introducción

El siguiente trabajo es el informe final de la práctica profesional supervisada (PPS) desarrollada, dedicada a realizar diferentes tipos de tareas de mejora en el área de operaciones de la empresa Julasoft S.A.

Ante todo, resulta necesario explicar el ámbito en el cual se desarrolló la práctica, a fin de comprender las tareas efectuadas. Dentro del área de operaciones, se llevan a cabo procesos tales como el análisis funcional de nuevos requerimientos por parte de clientes; el soporte técnico a clientes con respecto a aplicaciones; la configuración y puesta en funcionamiento de productos de la empresa; la integración de los diferentes productos y servicios ofrecidos, el monitoreo de servidores que alojan los servicios y sistemas de la empresa y la ejecución y gestión de las copias de seguridad relevantes de los sistemas, entre otros.

Dentro de esta área, se encontraron oportunidades de mejora, en relación con la coordinación y los procedimientos llevados a cabo para el despliegue y mantenimiento de los productos de la empresa. Mediante las actividades planteadas para la PPS, se pretendió generar un aporte significativo en el trabajo del área, facilitando la automatización de varios de los procesos y actividades realizados dentro del área.

En primer lugar, se definirán los objetivos que se pretenden alcanzar dentro del plan de trabajo, ya que determinan el alcance de la práctica en relación a la cantidad de actividades y el tiempo definido para el presente trabajo. En segundo lugar, se explicarán y describirán las diferentes actividades y tareas a realizar para alcanzar tales objetivos, con su correspondiente desarrollo, mostrando el avance y los resultados obtenidos, así como también la justificación de cada decisión tomada que guía el desarrollo de la práctica. Debido a que cada tarea ejecutada debe ser descripta en detalle, se incluirán definiciones en anexos que permitan facilitar la comprensión de cada actividad realizada. Además, se comentarán las conclusiones de cada actividad, y se incluirá un cierre general de la práctica.

1.1 Objetivos de la práctica

Como se explicó anteriormente, el objetivo general del presente trabajo fue lograr mejoras para una serie de actividades llevadas a cabo dentro del área de operaciones. En este sentido, los objetivos iniciales fueron los siguientes:

- configurar y poner en funcionamiento productos de la empresa, tales como *DataView*, *LK-ERP*, *Proteo*, etc.;
- integrar los diferentes productos y servicios ofrecidos;
- monitorear los servidores que alojan los servicios y sistemas de la empresa;
- ejecutar y gestionar las copias de seguridad relevantes de los sistemas.

En relación con los objetivos generales, se definieron objetivos específicos agrupados de acuerdo con los generales. En las siguientes secciones, se detallarán los objetivos específicos.

1.1.1 Integración de dos productos de la empresa (LK-ERP y DataView)

Para los dos primeros objetivos descriptos previamente, se plantearon los siguientes objetivos específicos:

- integrar dos productos de la empresa (*LK-ERP* y *DataView*) para un cliente actual que tiene intención de adquirir uno de los sistemas;
- describir y documentar todo el procedimiento y actividades requeridas para realizar la integración entre los dos productos de la empresa, previamente mencionados;
- realizar un plan o proyecto de mejoras a aplicar dentro de la arquitectura de la solución de integración actual entre los dos productos anteriormente establecidos.

Estos objetivos se establecieron a partir de la necesidad de la organización de avanzar con la puesta en marcha de nuevas soluciones para varios clientes.

Hasta el momento, la empresa ofrece los productos *LK-ERP* y *DataView* por separado, pero, desde una perspectiva comercial, resulta una estrategia más eficaz ofrecer los dos

sistemas integrados. Para lograr esto, además de la implementación, es necesario documentar todo el proceso de integración, ya que uno de los principales inconvenientes actuales para poner en funcionamiento esto es la falta de documentación y especificación del procedimiento y todas las actividades relacionadas que permiten la integración entre los dos productos, lo cual obstaculiza la puesta en funcionamiento de la solución, y retarda el cumplimiento de compromisos con los clientes.
Además, resulta oportuno proponer y realizar mejoras que faciliten el trabajo de mantenimiento luego de la implementación e, incluso, que simplifiquen la solución actual.

1.1.2 Automatización y mejora de procedimientos de monitoreo y gestión de bases de datos y discos sobre servidores de la empresa

Para los dos últimos procesos listados en la sección 1.1, los objetivos específicos fueron los siguientes:

- automatizar el monitoreo de los discos duros de los servidores, permitiendo notificar al administrador (miembro del área de operaciones), en caso de que sea necesario;
- automatizar la gestión de copias de seguridad de las diferentes bases de datos utilizadas por las aplicaciones;
- generalizar estas soluciones a diferentes sistemas operativos, ya que, a futuro, se espera migrar varios de los servidores a otros sistemas operativos

Estos objetivos se plantearon debido a una problemática existente en el área de operaciones de la empresa. Dentro de esta, se llevan a cabo diferentes procedimientos de forma manual o automatizada, pero con inconvenientes, en relación con la gestión de copias de seguridad de las bases de datos de las diferentes aplicaciones y el control de discos, entre otras cuestiones, tales como la limpieza de registros de ciertas bases de datos. Ciertamente, estas tareas son, en gran medida, meramente rutinarias, ya que se llevan a cabo cada cierto intervalo de tiempo. Por lo tanto, se consideró la posibilidad de

automatizarlas y no tener que asignar una cantidad de tiempo al control o seguimiento de que cada rutina realizada en los servidores se haya completado, efectivamente, de forma exitosa.

1.2 Tareas a realizar

En función de los objetivos específicos planteados previamente, se determinaron un conjunto de actividades a realizar, que apuntaron a alcanzar los objetivos de la práctica. En lo que sigue, se explicará cada una de ellas.

1.2.1 Puesta en marcha de la integración de dos productos de la empresa (*LK-ERP* y *DataView*) y generación de documentación del proceso de integración

Estas tareas apuntaron a alcanzar el primer objetivo específico planteado (sección 1.1.1), que apunta al proceso de integración de *LK-EPR* y *DataView*. Si bien dicho proceso se desarrollará en detalle en la sección 2, en términos generales, esta tarea incluyó los siguientes aspectos:

- profundización en el estudio del uso de las diferentes herramientas que se encuentran actualmente disponibles para la integración entre *LK-ERP* y *DataView*;
- configuración y puesta en marcha de la integración entre los sistemas *LK-ERP* y *DataView* para un cliente en particular, a fin de comprender el proceso íntegramente y, además, llevar a cabo la actividad requerida por el cliente;
- documentación de todo el procedimiento y la arquitectura de integración entre las dos aplicaciones anteriores.

1.2.2 Propuesta de un conjunto de mejoras a aplicar sobre la solución de integración entre *LK-ERP* y *DataView*

Esta tarea apuntó a una mejora del proceso de integración de *LK-ERP* y *DataView*, definida por la tarea previa. Para ello, se realizó el análisis y delimitación de un plan o conjunto de pasos para dicha integración. Como se encuentra muy ligado a todo el desarrollo de las actividades de integración entre las dos aplicaciones nombradas anteriormente, su desarrollo se volcará en la misma sección.

1.2.3 Automatización y mejora de procedimientos de monitoreo y gestión de bases de datos y discos sobre servidores de la empresa

Esta última tarea a ejecutar corresponde al segundo objetivo planteado, ligado a la optimización de los procedimientos operativos dentro del área de la empresa. En relación con la estructura de la tarea, esta puede resumirse de la siguiente forma:

- relevamiento e inventario de los servidores (máquinas virtuales en la nube), junto con sus discos y bases de datos utilizadas por las diferentes aplicaciones;
- desarrollo de una aplicación configurable, que permita monitorear el estado de los discos, y notificar al administrador por correo electrónico en caso de detectar algún problema como la falta de espacio libre;
- realización de un programa que permita administrar las diferentes copias de seguridad de las bases de datos relevantes sin intervención de un administrador, una vez implementado en cada servidor;
- determinación de indicadores críticos relevantes (estado de discos, *backups* recientes, etc.), para graficar mediante la aplicación *DataView*, cuyos datos serán obtenidos de las dos aplicaciones anteriores.

2. Actividades realizadas

En esta sección, se detallarán las actividades de la práctica profesional supervisada. Toda decisión puede ser explicada en términos de las causas que la originaron y la estructura del plan de trabajo en ejecución no es la excepción. Durante la primera actividad, se decidió determinar cuál sería el plan de trabajo, qué objetivos se pretendían alcanzar, y qué actividades se llevarían a cabo en pos de tales objetivos, en base a las necesidades de la organización y las oportunidades disponibles. Durante esta actividad, se comenzó el plan inicial, ya que se decidió atender dos requerimientos fundamentales: los correspondientes a las actividades mencionadas: integración de *LK-ERP* y *DataView* y mejora de procesos de gestión de servidores. En las actividades dos y tres, se llevó a cabo el desarrollo correspondiente a los dos requerimientos fundamentales determinados en el plan inicial. Por último, se llevaron a cabo las actividades correspondientes al cierre de la práctica, como la evaluación de resultados y la generación de los informes correspondientes.

En relación con la asignación del tiempo, se consideró el uso de 20 horas por semana y se eligieron las dos semanas iniciales para la primera etapa de introducción a la organización (que incluye la primera actividad). Para las actividades siguientes, de implementación y desarrollo dentro del área de operaciones, se asignaron tres semanas para cada una (actividades descriptas en las secciones 2.2 y 2.3). Por último, se delegaron las últimas dos semanas para el cierre y la conclusión de la práctica. A continuación, se expondrán en detalle las actividades realizadas, vinculadas a sus objetivos específicos.

2.1 Inducción y análisis de productos y procesos de la empresa

2.1.1 Desarrollo

La actividad a realizar consistió principalmente en lo siguiente:

1. Investigación y análisis de los sistemas existentes: *AlertaTel*, *LK-ERP*, *Proteo* y *DataView.*

En la práctica, es decir, durante la ejecución de lo planificado, se han realizado otras actividades, como participación de reuniones con clientes, para comprender mejor cuáles eran las necesidades y requerimientos exigidos hacia la organización. Esto coincide en buena medida, con una de las ideas fundamentales de la práctica profesional supervisada, que es contribuir a integrar al estudiante practicante al ambiente de trabajo de una empresa que ofrece soluciones informáticas.

Para la profundización de los diferentes productos ofrecidos por la empresa, se trabajó con varios documentos sobre los productos, tanto internos (de la compañía), como con material de capacitación para usuarios y clientes.

Como ya se estableció, el mayor énfasis se situó en los productos *LK-ERP* y *DataView,* y, en menor medida, en *Proteo*. Debido a que esta práctica, en una buena medida, es sobre los dos primeros, en los anexos se encuentran, de forma resumida, características de las aplicaciones más relevantes.

2.1.2 Cierre

Como se mencionó previamente, se analizaron los sistemas o aplicaciones existentes, tanto en términos de funcionalidad ofrecida como en relación a varias características técnicas, desde la infraestructura (los sistemas operativos, las bases de datos que dan soporte a las aplicaciones ofrecidas como productos), hasta los inconvenientes actuales. Además, el involucramiento en las actividades de la compañía brindó un panorama más claro para conocer mejor el ambiente de trabajo. Gracias a esta inclusión, se terminaron de delimitar rápidamente las tareas a alcanzar.

Resulta difícil presentar los resultados, ya que no son tangibles, como un bien físico, o una obra intelectual concreta y delimitable; pero sus consecuencias son los fundamentos que permiten establecer las actividades a realizar.

2.2 Proceso de integración de *LK-ERP* y *DataView* y documentación de dicho proceso

Esta actividad consistió en la puesta en marcha de la integración de los dos productos mencionados previamente: *LK-ERP* y *DataView*, incluyendo la generación de la documentación que describe el proceso, por ser un objetivo a alcanzar para la empresa y, también, como parte de este informe.

El *LK-ERP*, el sistema de gestión de recursos, puede integrarse a la aplicación *DataView* para la visualización de información relevante en indicadores. Cada aplicación por separado puede utilizarse sin ningún inconveniente, pero su integración requiere de varios procedimientos y pasos intermedios.

A continuación, se presenta un desglose detallado de la integración entre estos, una presentación de varios componentes involucrados en toda la arquitectura y el desarrollo de todas las actividades que fueron necesarias para integrar las aplicaciones.

2.2.1 Desarrollo

Las actividades definidas y planificadas fueron las siguientes:

1. investigación sobre la arquitectura de integración entre *LK-ERP* y *DataView*;
2. configuración y puesta en marcha de la integración entre los sistemas *LK-ERP* y *DataView*;
3. documentación de todo el procedimiento y la arquitectura de integración entre las dos aplicaciones.

Como se dijo, estas actividades surgen para dar respuesta a la necesidad de integrar los dos productos de la organización. En tal sentido, puede dividirse este desarrollo en las tareas ejecutadas que engloban las actividades planteadas inicialmente y dan lugar al cumplimiento de los objetivos específicos establecidos. Se trata de las siguientes tareas:

a. toma de requerimientos funcionales solicitados por el cliente;
b. análisis de la estructura de los datos a tomar del sistema ERP;

c. integración entre el ERP y *DataView*;
d. configuración de un tablero en *DataView;*
e. presentación del tablero al cliente para su evaluación y aceptación.

Además, todo lo que hace a la implementación de esta integración (prácticamente todas las actividades listadas) fueron documentadas para uso interno de la organización. Un inconveniente que se encontró en la organización fue la falta de documentación con respecto a cómo llevar a cabo este proceso de forma ordenada, lo cual genera problemas a la hora de responder este tipo de solicitudes por parte de los clientes y, en algunos casos, limita la posibilidad de ofrecer mejores soluciones. A continuación, se describirán de manera pormenorizada las tareas presentadas previamente.

2.2.1.a) Toma de requerimientos funcionales solicitados por el cliente

El cliente, una organización usuaria del ERP, tenía la necesidad de extraer y representar gráficamente los datos de producción, por unidades y pesos; esto significa, en este caso específico, la producción de artículos, dividida en cinco rubros (clasificación que surge de la praxis propia del cliente):

I. papeles higiénicos (PH)
II. toallas intercaladas (TI)
III. bobinas industriales (BI)
IV. toallas en rollo (TR)
V. *outlet* (son artículos de los rubros anteriores que se gestionan de forma diferenciada, por decisiones de negocio).

El formato de la información a mostrar, coordinado con el cliente, fue el siguiente:

I. producción de cada día del mes actual, total y dividida por rubros;
II. producción de los meses del año, total y dividida por rubros;
III. producción acumulada y meta del mes actual.

Si bien inicialmente no resultaba claro cuál sería la mejor representación de la información, tanto por parte del autor como por parte del cliente, a medida que se avanzaba en la implementación, se fue delimitando de manera más nítida la forma de mostrar los datos, es decir, si resulta apropiado mostrar la producción de la semana actual, de la semana anterior; si es mejor mostrar los datos de producción en forma de gráficos con barras o torta, etc. Se omitirá parte de los cambios realizados a la representación de datos a lo largo de esta actividad, ya que no corresponden al trabajo de integración, aunque sí fueron parte de una discusión externa al presente desarrollo.

Independientemente de la forma de representar los datos, en primer lugar, lo importante fue extraer la información de producción de cada día del último año. Ya a partir de esa información, se pudo construir toda visualización o presentación de los datos requeridos.

Los usuarios de la aplicación, en cada día de fabricación, cargan la producción correspondiente a ese mismo día. Por lo tanto, la **primera variable** a tener en cuenta para el análisis fue la del **tiempo** (día-mes-año).

2.2.1.b) Análisis de la estructura de los datos a tomar del sistema ERP

Para poder realizar un análisis de las tablas existentes en la base de datos del ERP, en primer lugar, mediante *TeamViewer*[1], se accedió de forma remota al servidor de la empresa, para, luego, realizar una copia de seguridad de la base de datos existente, mediante *SQL Server Enterprise Manager*[2]. A continuación, esta copia se colocó en un entorno de pruebas propio de la empresa, utilizado específicamente para tales aplicaciones, que cuenta con una máquina virtual con el sistema ERP y sus bases de datos, preparadas para ser

[1] *TeamViewer* es una aplicación que facilita el acceso remoto a otras computadoras, y es el medio común mediante el cual se da soporte a los clientes. (TeamViewer, s.f.)

[2] *SQL Server Enterprise Manager* es una aplicación para la gestión de *SQL Server 2000* (base de datos de *LK-ERP*). Posteriormente *Microsoft* (empresa desarrolladora de *SQL Server*) reemplazó esta aplicación por otra, *SQL Server Management Studio*. (Alison Balter, 2006)

revisadas. Se muestra en la siguiente figura el uso de *Enterprise Manager* para generar la copia de la base de datos.

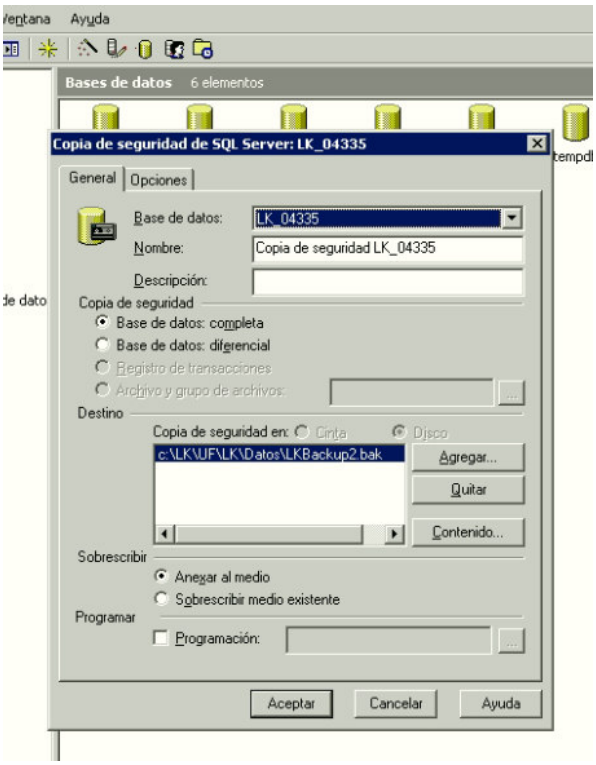


Figura 3: Copia de la base de datos del cliente
(imagen propia)

La base de datos del ERP cuenta con más de 450 tablas, cientos de vistas y procedimientos almacenados (es decir, es un sistema que tiene una complejidad considerable), entre otras características que obligaron a realizar una revisión extensa y minuciosa de la información. Para la conexión a la base de datos de prueba (accesible por la red interna de la empresa) se utilizó una aplicación que permite manipular diferentes bases de datos: *DBeaver*[3].

[3] *DBeaver* es un programa que tiene la capacidad de manipular una cantidad muy variada de servidores de bases de datos: *SQL Server, MySQL, Oracle, PostreSQL*, etc., ya que dispone de diversos *drivers* o controladores específicos para cada tipo de base de datos. Para el caso de *SQL Server 2000*, el *driver* utilizado es jTDS (un controlador específico para versiones antiguas de *SQL Server*).

En base a la revisión de los datos, se encontró la existencia de **órdenes de compra** con artículos a nombre de un proveedor con un nombre del tipo 'PRODUCCION'. Esto disparó una pregunta inicial en relación al uso que le da el cliente al sistema, ¿cuál es la relación entre la **producción** y las **órdenes de compra**?

La figura que se muestra a continuación contiene diagramas realizados con *SQL Server Enterprise Manager*. Estos diagramas sirven para mostrar una pequeña porción de la complejidad de la estructura de los datos que fue necesario analizar.

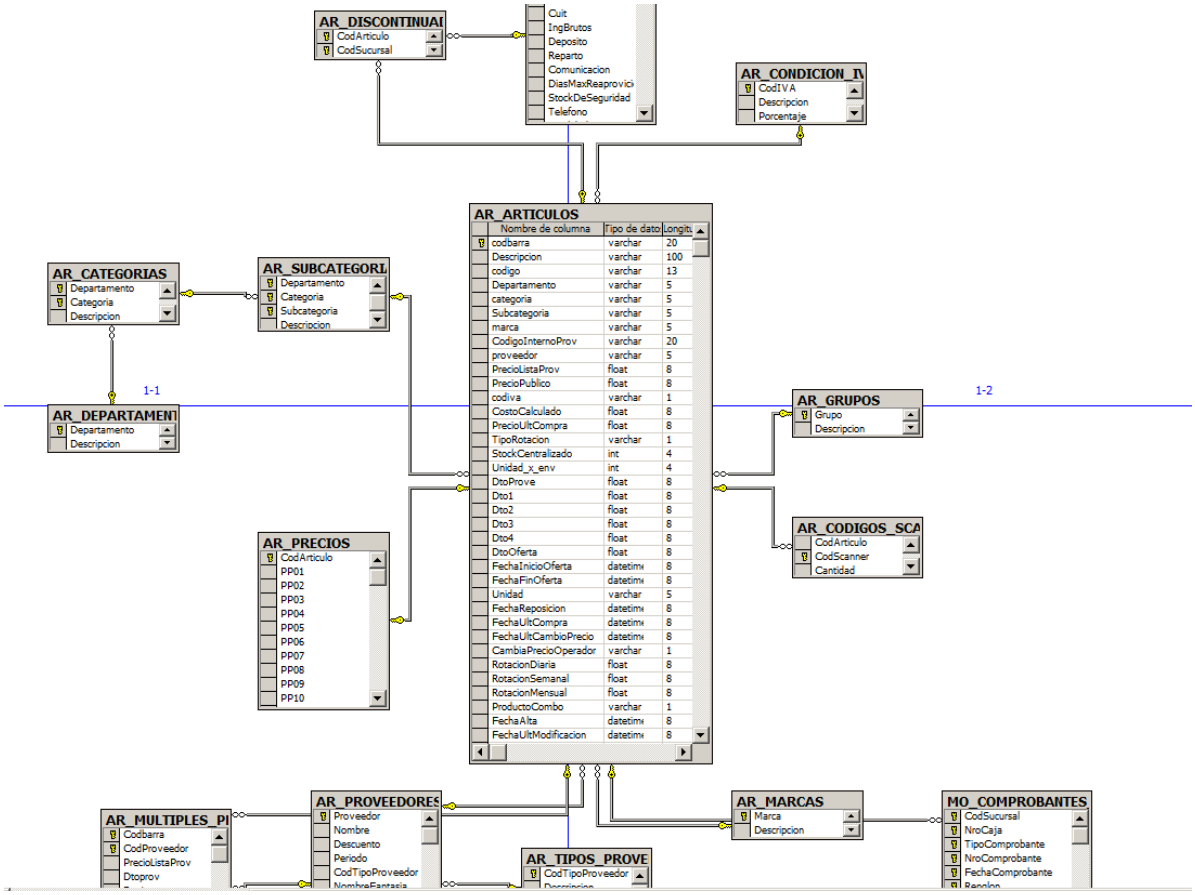


Figura 4: Diagrama de entidades y relaciones (imagen propia)

La figura previa muestra las entidades y relaciones correspondientes a la representación de artículos dentro del sistema.

A continuación, se muestra un diagrama que ilustra las relaciones de las tablas de proveedores y órdenes de compras del sistema.

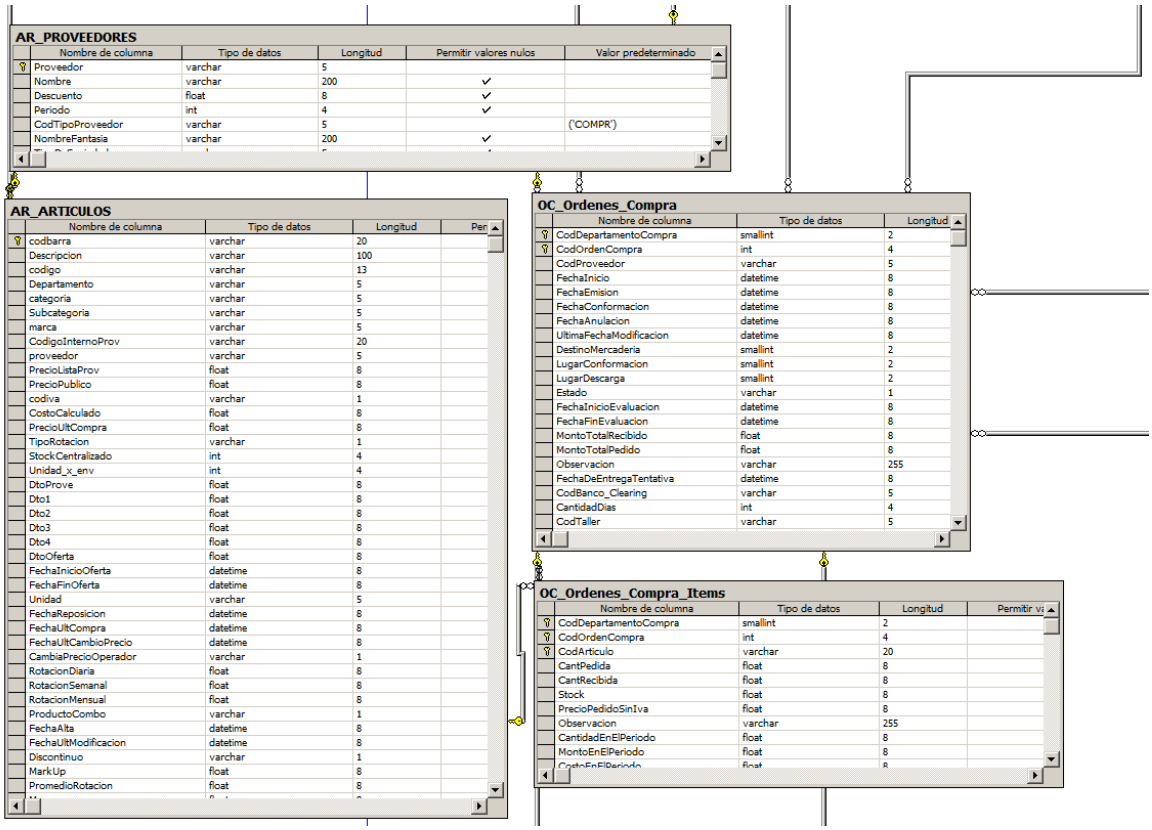


Figura 5: Tabla de órdenes de compra y sus relaciones (imagen propia)

Originalmente, consideramos que la información de producción se cargaba de otra forma, en un módulo específico[4], por ejemplo. Sin embargo, habiendo observado que el foco debía situarse en las órdenes de compra cargadas por los usuarios, se decidió continuar trabajando sobre las tablas que se enumeran a continuación:

- artículos (junto con las tablas de categorías y subcategorías);
- proveedores;

[4] Esto se debe a que el sistema ERP cuenta con una cantidad considerable de módulos: contabilidad, cobros, vendedores, artículos, administración, etc.

- órdenes de compra (con la tabla **OC_ORDENES_COMPRA_ITEMS**, que tiene los ítems por cada orden de compra).

Más adelante, se llegó a las siguientes conclusiones:

- la carga se realiza a nombre del proveedor 533 (proveedor de 'producción'); la siguiente figura muestra una consulta realizada contra la base de datos (mediante la aplicación *DBeaver*).

OC_ORDENES_COMPRA | CodProveedor = 533

	CodOrdenCompra	CodProveedor	FechaInicio	FechaEmision	FechaConformacion	FechaAnulacion
1	2.534	533	2018-12-07 14:37:00	2018-12-07 14:38:56	2018-12-07 14:39:04	[NULL]
2	2.571	533	2018-12-21 09:00:50	2018-12-21 09:02:48	2018-12-21 09:02:57	[NULL]
3	2.573	533	2018-12-21 11:29:13	[NULL]	[NULL]	2018-12-21 11:57:24
4	2.636	533	2019-01-15 15:29:21	2019-01-15 15:32:54	2019-01-15 15:34:08	[NULL]
5	2.686	533	2019-01-29 11:42:00	2019-01-29 11:43:19	2019-01-29 11:43:24	[NULL]
6	2.687	533	2019-01-29 11:45:23	2019-01-29 11:46:31	2019-01-29 11:46:37	[NULL]
7	2.697	533	2019-01-31 15:18:10	2019-01-31 15:20:12	2019-01-31 15:20:22	[NULL]
8	2.755	533	2019-02-22 13:40:56	2019-02-22 13:53:43	2019-02-22 13:54:08	[NULL]
9	2.759	533	2019-02-25 16:21:36	[NULL]	[NULL]	[NULL]
10	2.766	533	2019-02-27 12:55:23	2019-02-27 16:19:11	2019-02-27 16:19:37	[NULL]
11	2.599	533	2019-01-03 14:31:11	2019-01-03 14:31:53	2019-01-03 14:32:00	[NULL]
12	2.630	533	2019-01-14 14:07:31	2019-01-14 14:11:36	2019-01-14 14:12:01	[NULL]
13	2.676	533	2019-01-25 16:09:50	2019-01-25 16:14:53	2019-01-25 16:15:04	[NULL]
14	2.769	533	2019-02-28 09:05:17	2019-02-28 16:00:47	2019-02-28 16:01:15	[NULL]
15	2.777	533	2019-03-06 10:04:11	2019-03-06 10:05:41	2019-03-06 10:06:17	[NULL]
16	2.799	533	2019-03-11 10:39:38	2019-03-11 10:42:10	2019-03-11 10:42:26	[NULL]
17	2.809	533	2019-03-15 15:54:27	2019-03-15 15:56:53	2019-03-15 15:56:58	[NULL]
18	2.816	533	2019-03-18 11:54:27	2019-03-18 11:55:32	2019-03-18 11:55:38	[NULL]

Figura 6: Órdenes de compra con proveedor 533 (imagen propia)

La siguiente imagen presenta información del proveedor 533, un proveedor virtual que representa la carga de la producción

R_PROVEEDORES | proveedor = 533

Proveedor	Nombre	Descuento	Periodo	CodTipoProveedor	NombreFantasia
533	PRODUCCION	[NULL]	[NULL]	0001	PRODUCCION

Figura 7: Proveedor virtual (imagen propia)

- los artículos pertenecen a una subcategoría específica (artículos de producción papelera, ya que también se carga producción de otros rubros), que son copias de los artículos de venta (se gestionan de forma separada respecto de los artículos comunes, por el manejo de inventario disponible); la siguiente imagen muestra algunos artículos de producción cargados en el sistema:

AR_ARTICULOS | Subcategoría = PRODU

	codbarra	Descripcion	codigo	Subcategoria	CodigoInternoProv	proveedor
1	190000811	TOALLA INTERCALADA BEIGE SM/	BE-510_PROD	PRODU	BE-510_PROD	533
2	190000812	TOALLA INTERCALADA BEIGE - 20(	BE-550_PROD	PRODU	BE-550_PROD	533
3	190000813	TOALLA INTERCALADA BEIGE SM/	BE-590_PROD	PRODU	BE-590_PROD	533
4	190000814	TOALLA INTERCALADA BEIGE PRE	BE-910_PROD	PRODU	BE-910_PROD	533
5	190000815	TOALLA INTERCALADA BEIGE PRE	BE-950_PROD	PRODU	BE-950_PROD	533
6	190000816	TOALLA INTERCALADA BEIGE PRE	BE-990_PROD	PRODU	BE-990_PROD	533
7	190000817	TOALLA INTERCALADA BLANCA S	BL-510_PROD	PRODU	BL-510_PROD	533
8	190000818	TOALLA INTERCALADA BLANCA -	BL-550_PROD	PRODU	BL-550_PROD	533
9	190000819	TOALLA INTERCALADA BLANCA S	BL-590_PROD	PRODU	BL-590_PROD	533
10	190000820	TOALLA INTERCALADA BLANCA P	BL-910_PROD	PRODU	BL-910_PROD	533
11	190000821	TOALLA INTERCALADA BLANCA P	BL-950_PROD	PRODU	BL-950_PROD	533
12	190000822	TOALLA INTERCALADA BLANCA P	BL-990_PROD	PRODU	BL-990_PROD	533
13	190000823	N* TOALLA INTERCALADA BEIGE E(	P.NU18_PROD	PRODU	P.NU18_PROD	533
14	190000824	N* TOALLA INTERCALADA BEIGE E(	P.NU22_PROD	PRODU	P.NU22_PROD	533
15	190000825	TOALLA INTERCALADA BLANCA E	P.NU20_PROD	PRODU	P.NU20_PROD	533

Figura 8: Artículos de producción (imagen propia)

- como se explicó antes, el primer valor a observar es el del tiempo, por lo que resulta fundamental establecer que la producción del día se carga al finalizar la jornada, mediante una orden de compra;
- es posible que la producción se cargue en varias órdenes de compra en un mismo día, y la suma de todas será el total del día;
- debido a errores de los usuarios, hay órdenes de compra anuladas que no deben tenerse en cuenta; por lo tanto, las órdenes que interesan son aquellas que tienen una fecha de confirmación (que es un atributo de las órdenes de compra): más arriba, se observó que, en el caso de que se haya anulado, no puede tener fecha de confirmación (momento en el que se hace efectiva la carga).

Ya que con esto se puede obtener la carga de producción, solamente queda determinar un criterio para discriminarla por los rubros solicitados inicialmente. En base a lo relevado con el cliente, se desprendió que la forma de representar cada rubro es mediante los siguientes criterios:

- **papeles higiénicos**: el código interno comienza con ‘PH’ o ‘COD-PH’
- **toallas intercaladas**: el código interno comienza con ‘TI’, ‘BE’, ‘BL’, ‘COD-BE’ o ‘COD-BL’
- **bobinas industriales**: el código interno comienza con ‘BI’ o ‘COD-BI’

- **toallas en rollo**: el código interno comienza con con ‘TR’ o ‘COD-TR’
- **productos *outlet***: el código interno termina con ‘OUT_PR’

Para el caso de las primeras cuatro categorías, debe agregarse una condición adicional, ya que los productos de *outlet* son exactamente iguales a los artículos de otros rubros, pero con una diferencia en el código interno. Se muestra a continuación lo que ocurre al filtrar artículos según el criterio de papeles higiénicos (código interno comienza con ‘PH’ o ‘COD-PH’).

AR_ARTICULOS | subcategoria = 'PRODU' and (CodigoInternoProv like 'PH%' or CodigoInternoProv like 'COD-PH%')

	codBar	Descripcion	Subcategoria	CodigoInternoProv
1	190000910	* PAPEL HIGIENICO FAMILY BLANCO PREMIUM S/H - 24 ROLLOS X 100 MTS	PRODU	PH-990_PROD
2	190001688	* PAPEL HIGIENICO FAMILY BLANCO PREMIUM S/H X 24 ROLLOS GRANDES - OUTLET	PRODU	PH-990-OUT_PR
3	190000905	* PAPEL HIGIENICO FAMILY BLANCO PREMIUM S/H - 30 ROLLOS X 80 MTS	PRODU	PH-985_PROD
4	190000908	* PAPEL HIGIENICO FAMILY BLANCO PREMIUM S/H - 48 ROLLOS X 50 MTS	PRODU	PH-980_PROD
5	190000907	* PAPEL HIGIENICO FAMILY BLANCO PREMIUM S/H - 24 ROLLOS X 50 MTS	PRODU	PH-975_PROD
6	190001687	* PAPEL HIGIENICO FAMILY BLANCO PREMIUM S/H X 24 ROLLOS CHICOS - OUTLET	PRODU	PH-975-OUT_PR
7	190000906	* PAPEL HIGIENICO FAMILY BLANCO PREMIUM S/H - 48 ROLLOS X 30 MTS	PRODU	PH-970_PROD
8	190001228	* PAPEL HIGIENICO JUMBO BLANCO PREMIUM S/H C.GRANDE - 4 ROLLOS X 500 MTS	PRODU	PH-950_PROD
9	190000895	* PAPEL HIGIENICO JUMBO GOFRADO BLANCO PREMIUM S/H CC - 8 ROLLOS X 300 MT	PRODU	PH-910_PROD
10	190000898	* PAPEL HIGIENICO JUMBO GOFRADO BLANCO PREMIUM S/H C.GRANDE - 8 ROLLOS X	PRODU	PH-900_PROD
11	190000905	* PAPEL HIGIENICO FAMILY BLANCO S/H - 24 ROLLOS X 100 MTS	PRODU	PH-590_PROD
12	190001686	* PAPEL HIGIENICO FAMILY BLANCO S/H X 24 ROLLOS GRANDES - OUTLET	PRODU	PH-590-OUT_PR
13	190000904	* PAPEL HIGIENICO FAMILY BLANCO S/H - 30 ROLLOS X 80 MTS	PRODU	PH-585_PROD
14	190000903	* PAPEL HIGIENICO FAMILY BLANCO S/H - 48 ROLLOS X 50 MTS	PRODU	PH-580_PROD
15	190000902	* PAPEL HIGIENICO FAMILY BLANCO S/H - 24 ROLLOS X 50 MTS	PRODU	PH-575_PROD
16	190001685	* PAPEL HIGIENICO FAMILY BLANCO S/H X 24 ROLLOS CHICOS - OUTLET	PRODU	PH-575-OUT_PR

Figura 9: Rubro PH mezclado con rubro outlet (imagen propia)

Si se observa la última columna de la tabla, en la imagen, se percibe que el filtro aplicado no es suficiente, porque al filtrar utilizando el criterio de papel higiénico (‘PH’ o ‘COD-PH’) uno se encuentra con que los productos de *outlet* cumplen con ese criterio (ya que un producto de *outlet* de papel higiénico también comienza con ‘PH’ o ‘COD-PH’).
La solución a esto es agregar las condiciones de filtrado para los primeros cuatro rubros, asegurándose de que el artículo no termine en ‘OUT_PR’. Además, todos los artículos a filtrar deben pertenecer a la subcategoría ‘PRODU’. Con esta información, se tiene, en principio, todo lo necesario para consultar la base de datos y extraer los datos requeridos.

Para continuar con las tareas (extraer y adaptar la información a un formato adecuado), se trabajó sobre las siguientes tablas específicas :

- **AR_ARTICULOS**: para tomar los artículos.
- **OC_ORDENES_COMPRA**: para tomar las órdenes de compra (carga de producción).
- **OC_ORDENES_COMPRA_ITEMS**: para tomar los artículos incluidos en cada orden de compra, junto a las unidades de cada artículo.

Se muestra a continuación un ejemplo de extracción del rubro de papel higiénico por separado, a partir del mes de agosto, agrupado por el día de carga:

	Fecha	kilos_papel_higienico	unidades_papel_higienico
1	02/08/2019	1.635,39	618
2	02/09/2019	343,8	180
3	03/10/2019	482	290
4	05/08/2019	528,12	274
5	06/09/2019	545,1	345
6	07/08/2019	365,52	152
7	08/08/2019	384,04	185
8	09/08/2019	360,24	228
9	09/09/2019	658	175
10	09/10/2019	285,69	107
11	10/09/2019	619,09	201
12	11/09/2019	1.103,63	379
13	11/10/2019	730,52	323

Figura 10: Producción de artículos del rubro PH (imagen propia)

La siguiente figura muestra una consulta en lenguaje SQL ejecutada en la base de datos, que establece los resultados de la figura previa (producción de papel higiénico separada por días):

```
Declare @FechaInicio DATETIME;
SET @FechaInicio = '20190801';
select
       CONVERT (VARCHAR(10), dbo.oc_ordenes_compra.fechaconformacion,103) as Fecha,
       cast (sum(dbo.oc_ordenes_compra_items.cantpedida * ISNULL( dbo.ar_articulos.peso, 0)) as decimal(10, 2)) as kilos_papel_higienico,
       sum(dbo.oc_ordenes_compra_items.cantpedida)  as unidades_papel_higienico
from
        dbo.oc_ordenes_compra_items
inner join dbo.ar_articulos on
        dbo.oc_ordenes_compra_items.codarticulo = dbo.ar_articulos.codbarra
join dbo.oc_ordenes_compra on
        dbo.oc_ordenes_compra_items.codordencompra = dbo.oc_ordenes_compra.codordencompra
where
        dbo.oc_ordenes_compra.fechaconformacion >= @FechaInicio
        and dbo.oc_ordenes_compra.codproveedor = 533
        and dbo.ar_articulos.subcategoria = 'produ'
        and dbo.ar_articulos.codigointernoprov not like '%out_pr'
        and (dbo.ar_articulos.codigointernoprov like 'ph%' or dbo.ar_articulos.codigointernoprov like 'cod-ph%')
group by
        CONVERT (VARCHAR(10), dbo.oc_ordenes_compra.fechaconformacion,103)
```

Figura 11: Consulta ejecutada (imagen propia)

Puede resumirse esta consulta de datos mediante las siguientes acciones:

- tomar órdenes de compra a partir de la fecha de inicio (agosto);
- tomar órdenes de compra del proveedor de producción (533);
- tomar todos los artículos con subcategoría de producción papelera (‘produ’) y que sean papeles higiénicos de las órdenes de compra (en base a los filtros determinados previamente);
- multiplicar para cada artículo las unidades cargadas por su peso (para obtener el peso total);
- sumar todas las unidades de los artículos (para obtener todas las unidades).

Aplicando la misma metodología, pero variando los filtros de artículos (primero, para PH; luego, para TI, etc.) se obtiene la producción de cada día, discriminada por rubro.
Lo siguiente fue agrupar todas estas consultas (cinco consultas separadas, una para cada rubro) en una única consulta. Esto se realizó combinando las cinco consultas y realizando algunas modificaciones (ver anexo), como buscar también todas las fechas donde hubo cargas de producción, para utilizar esas fechas como elemento que permita cruzar y agrupar todas las consultas separadas (es decir, por ejemplo, agrupar toda la carga de PH

del día 02/08 con toda la carga de TI del 02/08).[5] A continuación, se muestran los resultados de ejemplo de la consulta desarrollada:

	Fecha	UnidadesPH	UnidadesTR	UnidadesBI	UnidadesTI	UnidadesOutlet	KilosPH	KilosTR	KilosBI	KilosTI	KilosOutlet
99	02/08/2019	618	229	78	810	0	1.635,39	822,36	304,98	2.802,6	0
100	05/08/2019	274	0	0	159	0	528,12	0	0	550,14	0
101	06/08/2019	0	258	0	160	0	0	1.099,08	0	553,6	0
102	07/08/2019	152	0	0	144	0	365,52	0	0	498,24	0
103	08/08/2019	185	0	0	28	50	384,04	0	0	96,88	148,54
104	09/08/2019	228	124	0	247	1	360,24	349,68	0	845,88	2,83
105	12/08/2019	111	260	179	0	0	417,36	890,18	658,72	0	0
106	14/08/2019	231	0	0	433	0	671,22	0	0	1.267,66	0
107	15/08/2019	115	0	0	300	0	244,95	0	0	967,48	0
108	16/08/2019	626	0	0	0	0	989,08	0	0	0	0

Figura 12: Consulta que agrupa datos de rubros (imagen propia)

Con esta consulta funcionando, ya se tiene la información que es necesario exteriorizar para reflejar en *DataView*, mediante tableros e indicadores gráficos.

2.2.1.c) Integración entre *ERP* y *DataView* y configuración del tablero de *DataView*

Teniendo la información que se debe extraer, se procedió a configurar los diferentes componentes que permiten integrar el sistema *ERP* y *DataView*. Resulta propicio mostrar un diagrama de cómo está compuesta la integración:

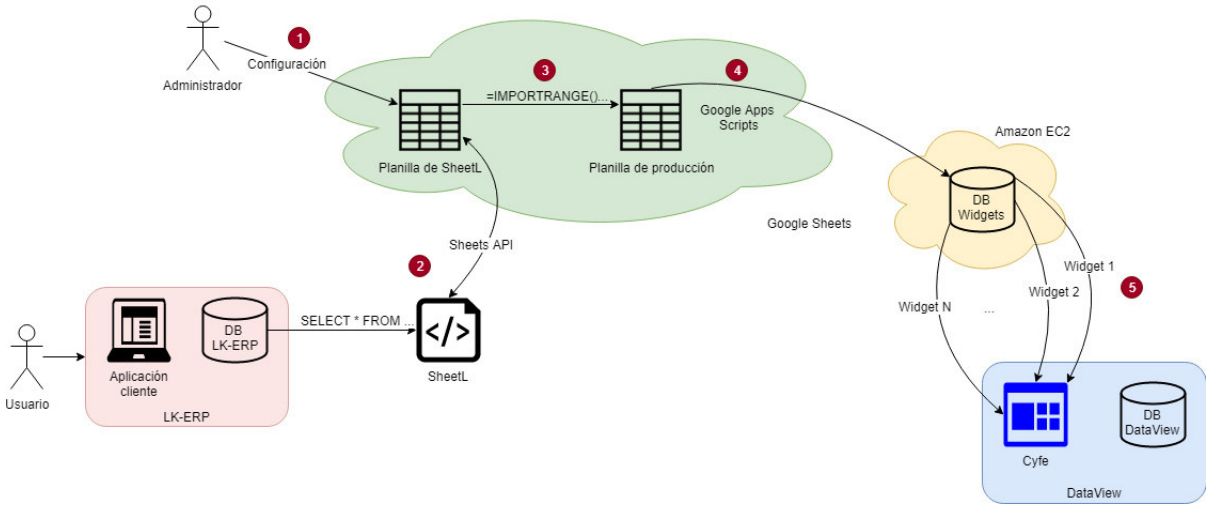


Figura 13: Arquitectura de integración entre LK-ERP y DataView (imagen propia)

[5] La consulta SQL terminada ocupó más de 150 líneas, en un formato ordenado y legible.

En el diagrama, se pueden observar los diferentes componentes que hacen a la integración entre las dos aplicaciones. Lo primero que se presenta y que debe tenerse en cuenta es el requerimiento del usuario/cliente (*LK-ERP*). El *ERP* se encuentra instalado en el ambiente o servidor del cliente y solamente es accesible de forma local. Es decir, quien quisiese tomar datos debería pensar en varios aspectos:

- tomar datos directamente de la base de datos, como se realizó previamente mediante las consultas ejecutadas contra el servidor de bases de datos;
- crear una solución configurada de forma local;
- enviar datos al exterior y no permitir que aplicaciones externas extraigan datos (es decir, que el flujo de información sea hacia el exterior del servidor del cliente, pero que se imposibilite cualquier intento de que una aplicación de afuera ejecute consultas contra el servidor interno);
- establecer que la cantidad de accesos remotos para la configuración y modificación del sistema sea la menor posible, más allá de la instalación inicial.

La solución existente para esto es un programa llamado *SheetL*[6], que permite, de forma simplificada, leer una configuración de una planilla de *Google Sheets*[7], extraer datos de una base de datos local y volcar los resultados en la planilla. Esto significa que permite visibilizar los datos deseados. Originalmente, se desarrolló para que fuese posible llevar a cabo esta integración. Esta actividad constó de varios pasos, que se explicarán en las siguientes secciones.

[6] *SheetL* es un programa que permite acceder a una base de datos de una red interna (que no es accesible por fuera), leer y ejecutar consultas de *SQL* para extraer datos y que estén disponibles para el exterior, para su posterior procesamiento, mediante *Google Sheets*. Utiliza la API (interfaz de comunicación y de acceso a funcionalidades entre dos aplicaciones) de *Google Sheets*, dispuesta por *Google*. (Google, s.f.)

[7] *Google Sheets* es una herramienta colaborativa en la nube de *Google*, que permite crear, modificar y compartir planillas de cálculo. Cuenta con la posibilidad de crear y desarrollar funciones que realizan acciones (como enviar un email, notificar, validar cambios, procesados datos ingresados en la planilla, etc.) así como también programar disparadores de las acciones. (Google, s.f.)

2.2.1.c) 1. Instalación y configuración de SheetL

El paso inicial fue colocar *SheetL* en el servidor del cliente; de la misma forma que se trajo la copia de seguridad de la base de datos del cliente al ambiente de pruebas, se colocó la aplicación *SheetL* en el servidor del cliente mediante *TeamViewer*. Además, se instaló la última versión de Java, lenguaje y entorno que *SheetL* necesita para funcionar. Se observan en la siguiente imagen los archivos que conforman *SheetL*:

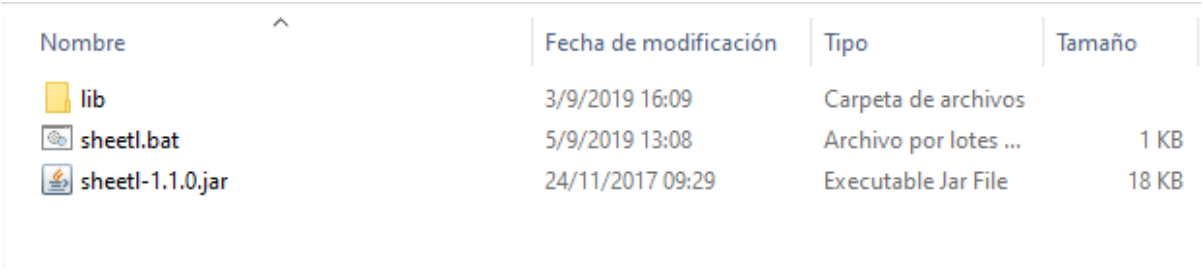


Figura 14: Carpeta de SheetL (imagen propia)

Luego de haber instalado *SheetL* en la misma red del servidor de base de datos del *ERP*, se creó una planilla de *Google Sheets* para utilizarla como configuración y volcado de datos de *SheetL*. Toda planilla tiene un identificador único que es observable en la URL. Esta planilla es accedida únicamente por un usuario de *Google* creado específicamente para el uso de SheetL dentro de este cliente.

Este ID se indicó como parámetro de inicio (-sid) de SheetL en el archivo sheetl.bat; a continuación, se muestra el archivo. Las imágenes presentadas son a modo de muestra, ya que no es el servidor del cliente, para preservar su confidencialidad:

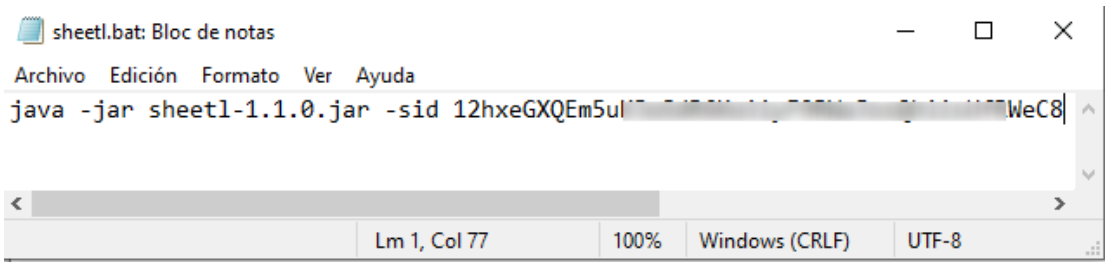


Figura 15: Parámetros para arranque de SheetL (imagen propia)

Luego de abrir el programa, se inicia automáticamente el navegador y solicita credenciales de acceso de *Google* para la planilla; luego, las almacena localmente, para que la próxima vez que se inicie *SheetL*, no requiera autorización. Esta característica es propia del uso de OAuth[8] por parte de *SheetL*.

Dentro de la planilla a usar por *SheetL*, se configuraron tres solapas, más una solapa adicional donde se vuelcan los resultados, que forman parte de la configuración de *SheetL*, como se puede verificar en la figura a continuación:

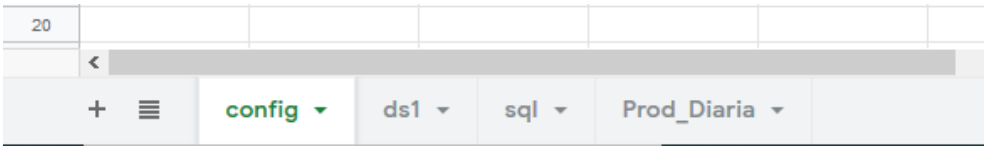


Figura 16: Solapas de la planilla de SheetL (imagen propia)

1. **config**: permite configurar los mensajes de error en el programa y la frecuencia de reconfiguración de este (no es necesario modificarlo, generalmente). La siguiente imagen muestra las variables de configuración:

[8] OAuth es un protocolo estandarizado para permitir la autenticación de diferentes aplicaciones (*Google* permite utilizar OAuth para permitir el acceso a sus servicios por parte de aplicaciones externas como *SheetL*).

	A	B	C
1	debug	S	
2	delay-reconfigura	10000	
3			
4			

Figura 17: Hoja config (imagen propia)

2. **sql**: en cada fila, se colocaron las consultas a ejecutar por *SheetL* contra la base de datos (en principio, la consulta definida en la actividad de análisis funcional), junto a una descripción opcional, el nombre de la hoja que tiene la conexión a la BD (**ds1**), la frecuencia, y dónde se colocarán los resultados (otra hoja, **Prod_Diaria**). Además, en cada ejecución, *SheetL* vuelca la fecha de la ejecución y un mensaje de error, en caso de que la consulta no se haya ejecutado correctamente. En este caso, la frecuencia es 5 minutos, pero, en la práctica, se colocó una frecuencia de 60 minutos. A continuación, la imagen muestra todos los parámetros de las consultas a colocar en la hoja **sql**:

	A	B	C	D	E	F	G
1	Query	Descripción	Origen De Datos	Frecuencia	Resultado	Ultima Ejecución	Último Error
2	Declare @Fechalnicio DATETIME; SET @Fechalnicio = '20190701'; select * from (select DiasDeCarga.Fecha, cast(ISNULL(unidades_p	Reporte de producción diaria, discriminada por 5 rubros en unidades y kilos desde el 01/08/2019 (es configurable)	ds1	5	Prod_DiariaA2	09/09/2019 22:20:49	
3							

Figura 18: Hoja sql (imagen propia)

3. **origen de datos** (**ds1**): contiene la configuración de conexión a la BD; como se utiliza un driver JDBC antiguo para conectarse, tiene una URL de conexión particular; se muestra una posible conexión a un servidor local. En el caso de la implementación en el cliente, se colocó la URL correspondiente junto a un usuario del servidor de bases de datos específico, creado con permisos de solo lectura sobre la base de

datos del ERP, LK_04335; en la figura siguiente, se muestra gráficamente la configuración:

	A	B	C	D
1	connection.driver	net.sourceforge.jtds.jdbc.Driver		
2	connection.url	jdbc:jtds:sqlserver://localhost:1433/LK_04335		
3	connection.userr	sheetl		
4	connection.pass\	[redacted]		

Figura 19: Configuración en la solapa ds1 (imagen propia)

A continuación, se muestra la creación del usuario utilizado por *SheetL*:

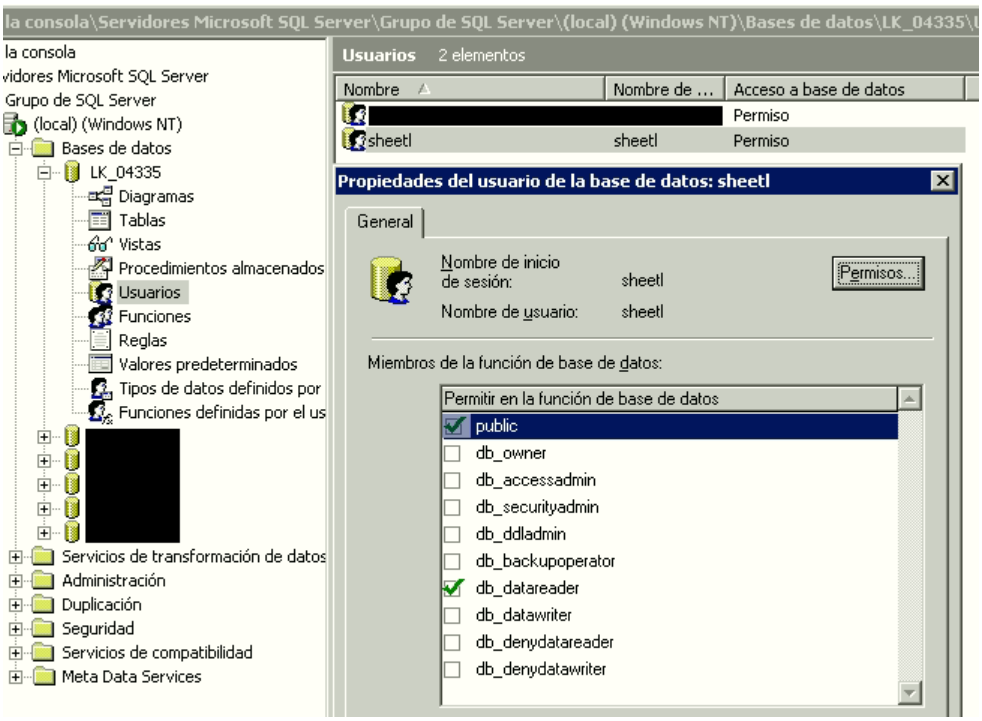


Figura 20: Usuario 'sheetl' específico para SheetL (imagen propia)

4. **hoja de resultados:** toda consulta debe tener una celda de una hoja, creada específicamente para ese uso (en este caso, tiene el nombre de **Prod_Diaria**); a continuación, se muestra la hoja de resultados correspondiente a los datos de producción:

Fecha	UnidadesPH	UnidadesTR	UnidadesBI	UnidadesTI	UnidadesOutlet	KilosPH	KilosTR	KilosBI	KilosTI	KilosOutlet
02/08/2019	618.0	157.0	78.0	810.0	0.0	1717.06	639.15	315.90	3142.80	0.00
05/08/2019	274.0	0.0	0.0	159.0	0.0	591.01	0.00	0.00	616.92	0.00
06/08/2019	0.0	258.0	0.0	160.0	0.0	0.00	1130.04	0.00	620.80	0.00
07/08/2019	152.0	0.0	0.0	144.0	0.0	395.27	0.00	0.00	558.72	0.00
08/08/2019	224.0	0.0	11.0	28.0	50.0	414.37	0.00	38.50	108.64	38.50
09/08/2019	229.0	0.0	0.0	247.0	1.0	386.23	0.00	0.00	946.77	0.00
12/08/2019	111.0	167.0	179.0	0.0	0.0	426.24	649.63	681.99	0.00	0.00
14/08/2019	231.0	0.0	0.0	433.0	0.0	721.61	0.00	0.00	1431.44	0.00
15/08/2019	115.0	0.0	0.0	300.0	0.0	254.15	0.00	0.00	1087.48	0.00
16/08/2019	626.0	0.0	0.0	0.0	0.0	1060.44	0.00	0.00	0.00	0.00
20/08/2019	1363.0	226.0	212.0	0.0	172.0	3730.93	885.96	836.60	0.00	571.80
23/08/2019	306.0	0.0	0.0	1483.0	0.0	691.56	0.00	0.00	5141.85	0.00
26/08/2019	387.0	483.0	212.0	0.0	0.0	859.07	2115.54	1021.84	0.00	0.00

Figura 21: Hoja de resultados (imagen propia)

Con esto, se logró la primera parte de la arquitectura integrada (exteriorizar información de la base de datos local). Además, ya que la aplicación debe estar funcionando en segundo plano en el servidor del cliente, se configuró para que, al momento del inicio del servidor (en el caso de que se apague el servidor por mantenimiento u otros problemas), también se inicie *SheetL*. Esto se realizó mediante el planificador de tareas de Windows que permite ejecutar acciones según el tiempo o eventos; aquí, el evento es el inicio del servidor. En la siguiente imagen, se observa *SheetL* funcionando:

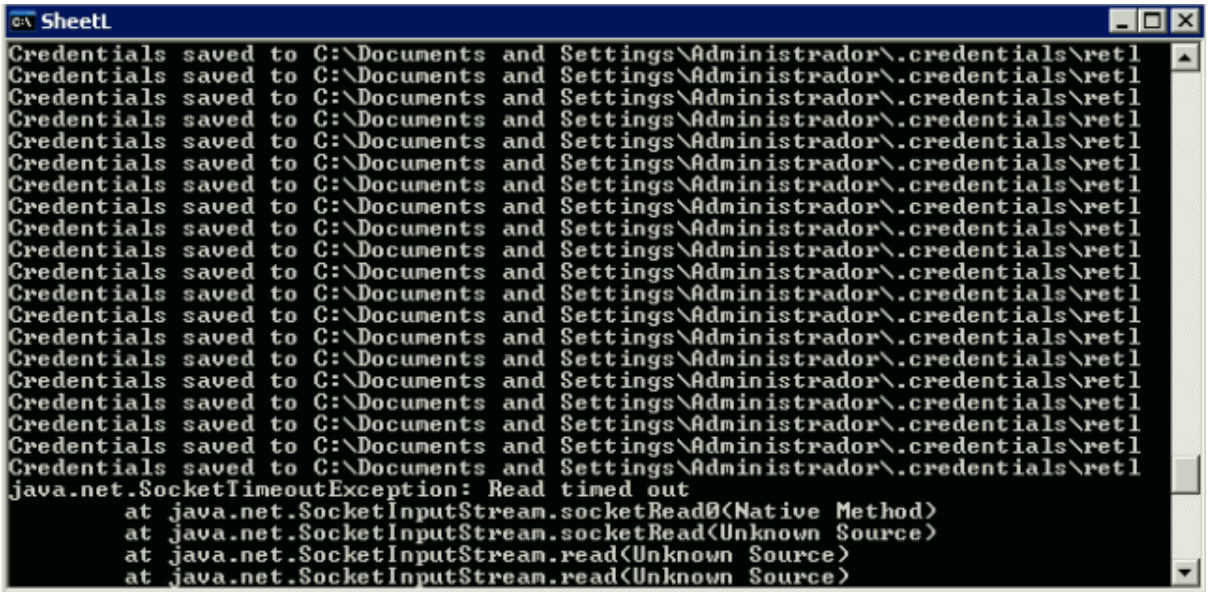


Figura 22: SheetL en ejecución (imagen propia)

2.2.1.c) 2. Carga de datos a planilla de *SheetL*

SheetL se encargó de ejecutar las consultas (*queries) SQL* y volcar los resultados en la hoja configurada. Se encontraron discrepancias entre los resultados obtenidos por el programa y los volcados en la planilla de datos. Al respecto, puede consultarse el anexo 4.1.b, pero aquí conviene aclarar que esto obligó a convertir los números en otra solapa u hoja, que tomara de la original los datos. Se puede observar en la siguiente figura el formato incorrecto de los datos:

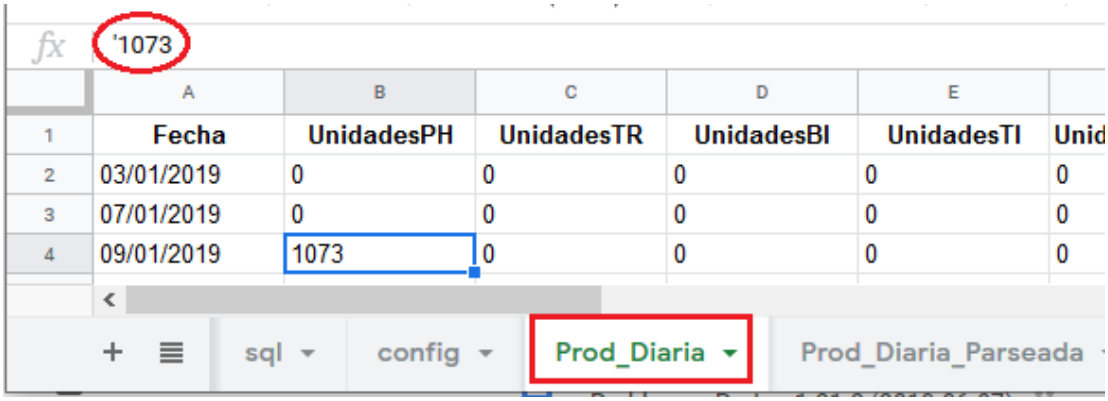


Figura 23: Números como cadenas de texto (imagen propia)

Como dijimos, la solución fue transformar los datos de la hoja en otra hoja, como se muestra en la siguiente figura:

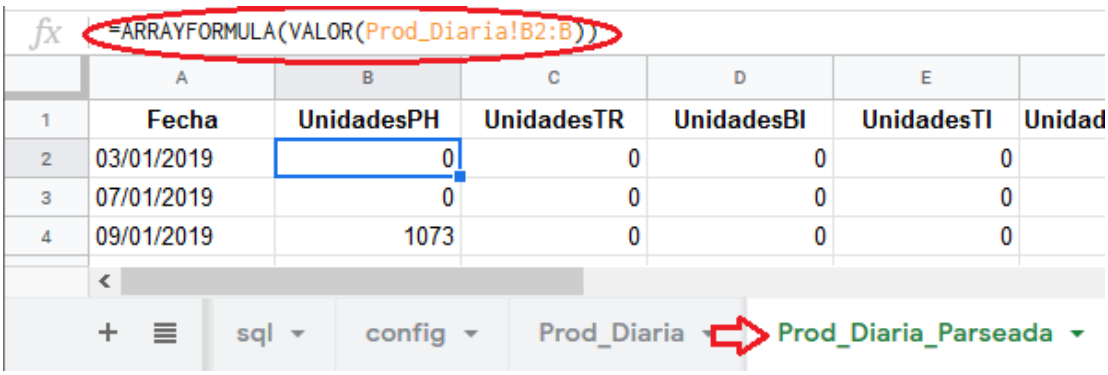


Figura 24: Valores corregidos en otra hoja (imagen propia)

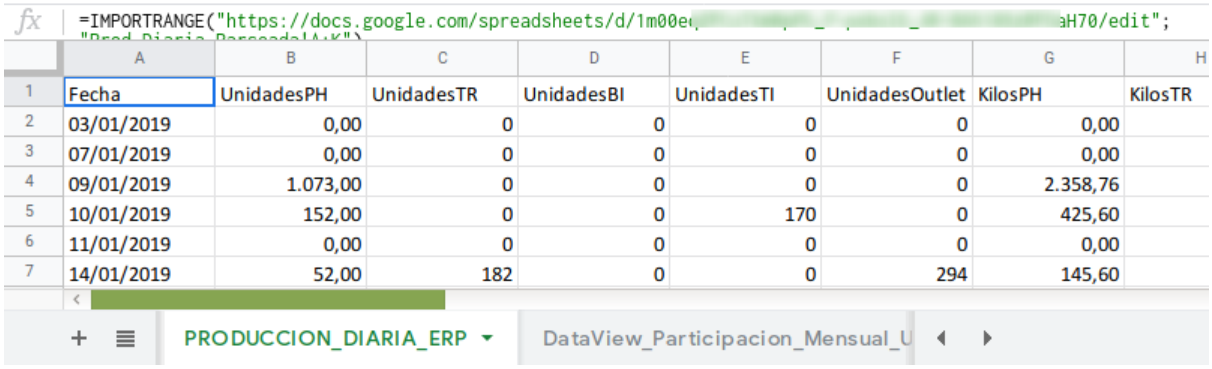
2.2.1.c) 3. Planilla de producción

El paso siguiente de la integración fue la conexión entre las dos planillas: por una parte, la que tiene los datos ya extraídos y procesados y, por otro, la utilizada en producción por parte de la organización cliente.

Dentro de la planilla de producción, se encuentra el siguiente listado de información:

- información cargada por el usuario (en muchos casos, ya que muchos clientes hacen uso de planillas para uso interno, además de sistemas de gestión);

- información cargada automáticamente mediante la importación de la planilla de *SheetL* a la de producción: esto se realiza con la función de *Sheets* llamada IMPORTRANGE, que permite a un usuario que tenga acceso a la planilla de *SheetL* disponer estos datos para uso dentro de la planilla de producción; se colocó la URL entera de la planilla, junto a la hoja o solapa a importar; se dedicó una solapa especial para colocar estos datos. En la siguiente imagen, se pueden observar los datos obtenidos del ERP y volcados en la planilla utilizada del cliente:



	A	B	C	D	E	F	G	H
1	Fecha	UnidadesPH	UnidadesTR	UnidadesBI	UnidadesTI	UnidadesOutlet	KilosPH	KilosTR
2	03/01/2019	0,00	0	0	0	0	0,00	
3	07/01/2019	0,00	0	0	0	0	0,00	
4	09/01/2019	1.073,00	0	0	0	0	2.358,76	
5	10/01/2019	152,00	0	0	170	0	425,60	
6	11/01/2019	0,00	0	0	0	0	0,00	
7	14/01/2019	52,00	182	0	0	294	145,60	

Figura 25: Datos del ERP en la planilla de producción (imagen propia)

Además, para la integración con *DataView*, se generaron unas hojas específicas que contienen la configuración de cada indicador gráfico (*widget*) de los tableros. Por una convención propia, delimitada para su mejor identificación, cada hoja lleva el prefijo ***Dataview_*** a fin de diferenciarlas de las planillas convencionales. A continuación, se muestran las hojas generadas para *DataView*:

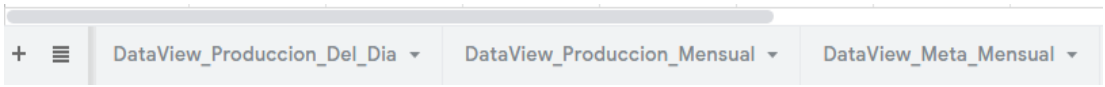


*Figura 26: Hojas con el prefijo DataView_* (imagen propia)

En términos de información y presentación, se crearon varias solapas u hojas y cada hoja describe un *widget*, tomando y procesando datos.

El procesamiento de los datos de la solapa de producción diaria para cada *widget* se realizó mediante el uso de funciones propias de *Sheet*. Como se observa en esta imagen, cada campo tiene una fórmula que devuelve ese valor:

```
=REDONDEAR(SUMAR.SI(ARRAYFORMULA(SI(ESBLANCO(PRODUCCION_DIARIA_ERP_MP!A:A);0;
MES(PRODUCCION_DIARIA_ERP_MP!A:A) )); "="&A2; PRODUCCION_DIARIA_ERP_MP!G:G) / 1000; 1)
```

	A	B	C	D	E	F
1	Mes	Papel higiénico	Toallas en rollos	Bobinas industri	Toallas intercalad	Outlet
2	1	5	0,6	0,8	3	1,2
3	2	4,7	0,7	0,2	5,5	0,1
4	3	3,9	1,6	2	8,5	0
5	4	7,2	5,4	1,1	9,5	0
6	5	10,9	5,2	1,3	13,3	1,2
7	6	8,4	4,4	1,7	8,2	0,1
8	7	10,7	4,2	2,6	6	0,1
9	8	11,6	9	3,2	16,4	0,7
10	9	11,5	1,3	1,7	8,9	2,7
11	10	5,6	2,3	2,4	6,2	0,5
12	11	0	0	0	0	0
13	12	0	0	0	0	0
14	Type	column	column	column	column	column
15	Color	#00FF00	#FFFF00	#0000FF	#FFA500	#808080
16	Total	79,50	34,70	17,00	85,50	6,60
17	YAxis	0,00	0,00	0,00	0,00	0
18	LabelShow	0,00	0,00	0,00	0,00	0
19						

Figura 27: Hoja con formato de DataView (imagen propia)

La hoja mostrada en la imagen previa procesa los datos del *ERP* y los adapta a un formato legible por *DataView*. En este caso, se representa la producción de cada mes del año dividida en los diferentes rubros.

Si bien están dentro de las requeridas para llevar a cabo esta integración planeada, varias de las actividades que se describen a continuación ya están bien constituidas por parte de la organización para dar respuestas a requerimientos a clientes. Entre estas, se cuenta la configuración del tablero dentro de la aplicación *Cyfe*. Esto se debe a que previamente se proveía la opción de utilizar *Google Sheets*. Mediante la implementación y configuración de diferentes componentes, se logró volcar información del sistema de gestión (ERP). De todas formas, para cerrar el circuito, se explicarán y desarrollarán estas actividades, ya que hacen

al cierre del proceso de integración (es decir, llevar la información volcada en las planillas hacia *DataView*).

2.2.1.c) 4. Configuración y uso de *Google Apps Scripts*

Una vez preparadas las hojas de los *widgets*, el paso siguiente fue llevar los datos a una base de datos intermedia. Esta base de datos (*DB Widgets*, que es un servidor de bases de datos *MySQL*) contiene el registro de todos los *widgets* presentes en todos los tableros utilizados de las diferentes planillas.

Mediante la programación de funciones, *Google Apps Scripts*[9] permite comunicarse con esa base de datos y colocar nuevos datos cada cierto tiempo programado. Estas funciones son porciones de código que se pueden ejecutar (en base a condiciones de tiempo o libremente) para realizar acciones sobre los datos.

La base de datos para el uso de los *widgets* de los tableros (de nombre *Dataview*), contiene una única tabla. Como esta tabla debe ser capaz de almacenar todos los datos de las diferentes columnas de todas las hojas **dataview_**, la tabla de nombre “data” tiene un formato particular, que se denomina “desnormalizado”, ya que debe tener la capacidad de

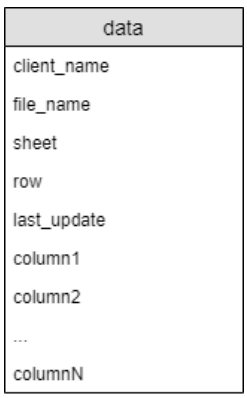


Figura 28: Tabla data (imagen propia)

[9] *Google Apps Scripts* es un lenguaje y entorno que permite programar *scripts* para la ejecución de funciones; como se explicó antes, pertenece a *Google Sheets* y a otros servicios similares de Google. (Google, s.f.)

almacenar la información de todas las planillas, con la flexibilidad necesaria. Se muestra a continuación el formato de la tabla:

Actualmente, la tabla tiene catorce columnas para las hojas (en el caso de la hoja anterior, solo se necesitan cinco). Para cada fila, de cada hoja ***dataview_***, de cada planilla ofrecida a un cliente, se genera una entrada en la tabla. Se identifica cada fila perteneciente a un *widget* mediante la siguiente combinación:

- cliente (client_name)
- nombre de la planilla (file_name)
- nombre de la hoja **dataview_** (sheet)
- número de fila (row)

Esta base de datos ya se utilizaba previamente, pero, mediante este trabajo, se le sumó el hecho de pertenecer a la arquitectura que permite integrar *LK-ERP* y *DataView*.
Lo que se buscó lograr fue insertar cada fila de cada hoja como una fila de la tabla de la base de datos intermedia. Luego de entender cómo es la estructura de la tabla para insertar los datos, se generaron los *scripts* para enviarlos.
Para cada *widget* (junto a su hoja dataview_), se creó una función o *script* que debía encargarse de las siguientes tareas:

- conectarse a una base de datos intermedia (DB Widgets, en *Amazon*) y enviar los datos, ya que tiene incorporados controladores y bibliotecas para diferentes bases de datos;
- refrescar los datos cada cierto tiempo (es decir, configurar 'disparadores' por tiempo).

A continuación, se muestra un ejemplo de una hoja con el formato apropiado. En la imagen, se marcan las columnas con diferentes colores para focalizar la información diferenciada:

	A	B	C	D	E	F	G
1	Semana	PH	TR	BI	TI	Outlet	Total
2	16/09 - 21/09	0	0	0	0,47	0	0,47
3	Color	#00FF00	#FFFF00	#0000FF	#FFA500	#808080	

Figura 29: Hoja con formato para DataView (imagen propia)

La imagen siguiente contiene el nombre de la hoja mostrada previamente:

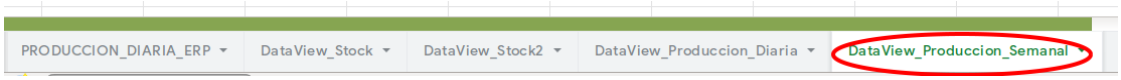


Figura 30: Nombre de la hoja (imagen propia)

Para generar los *scripts* o funciones asociadas a los indicadores, se ingresó al menú de *Google Sheets*, mostrado a continuación:

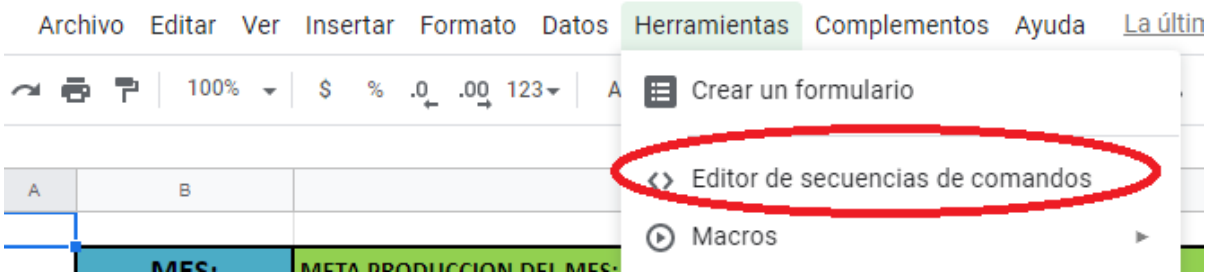


Figura 31: Menú de Google Sheets (imagen propia)

La siguiente imagen muestra la función asociada a la hoja presentada anteriormente:

DataView_Produccion_Semanal

Code.gs

```
function DataView_Produccion_Mensual()
{
  try {
    var conn = Jdbc.getConnection(dbUrl, user, userPwd);
    var stmt = conn.createStatement();
    var start = new Date(); // Get script starting time
    var sheet_name = 'DataView_Produccion_Mensual';
    // delete data
    var sql="DELETE FROM `dataview`.`data` "+
      "WHERE client_name = '" + client_name + "' and file_name = '" + file_name + "' and sheet= '" + sheet_name + "'"
    var count = stmt.executeUpdate(sql);

    // insert data
    var ss = SpreadsheetApp.getActiveSpreadsheet();
    ss.setActiveSheet(ss.getSheetByName(sheet_name));
    var data = ss.getDataRange().getValues();

      for (var i = 0; i < data.length; i++) {
        // insert data
        var sql="INSERT INTO `dataview`.`data` (`client_name`,`file_name`,`sheet`, `row`,`column1`,`column2`,`column3
          "('" + client_name + "', '" + file_name + "','" + sheet_name + "', " +
            String(i) + ",'" + data[i][0] + "', '" + data[i][1] + "', '" + data[i][2] + "', '" + data[i][3] + "', '"
        //Logger.log(sql);
        var count = stmt.executeUpdate(sql);
      };

    conn.close();
    var end = new Date(); // Get script ending time
    Logger.log('Time elapsed: ' + (end.getTime() - start.getTime()));
  } catch(err) {
    errorReport(err);
  }
}
```

Figura 32: Función asociada a la hoja previa (imagen propia)

La función mostrada borra los datos anteriores correspondientes a esa hoja en la base de datos y los sobreescribe con los nuevos datos que se traducen en un *widget* o indicador.
Las credenciales de acceso a la BD (usuario, contraseña, URL de conexión) se encuentran en el mismo archivo con las funciones. No se muestran, pero son variables globales: *user* y *userPwd*. Este usuario solo tiene permisos de acceso, lectura y escritura sobre la base de datos intermedia específica, por cuestiones de seguridad.
Se observa en la esquina superior izquierda de la imagen previa un ícono de reloj; es para configurar los disparadores temporales. Al ingresar allí, se accede al menú de disparadores configurados para las hojas, como se muestra en la siguiente figura:

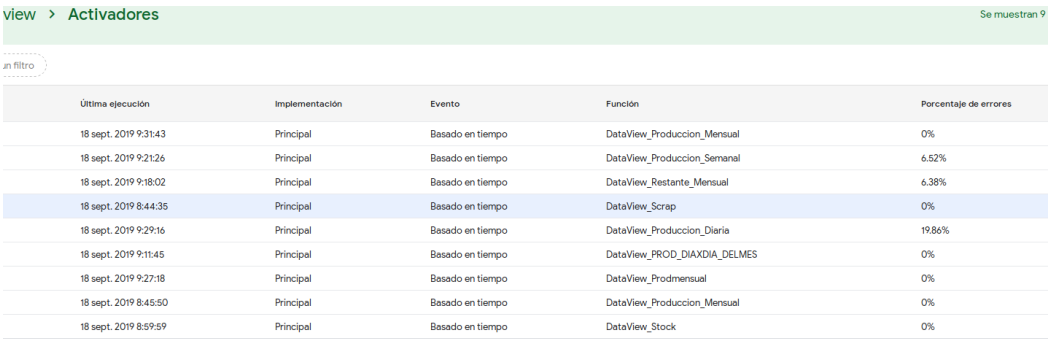


Figura 33: Menú de disparadores (imagen propia)

Luego de crear un disparador y ejecutarlo, se insertan datos en la base de datos, como se muestra a continuación:

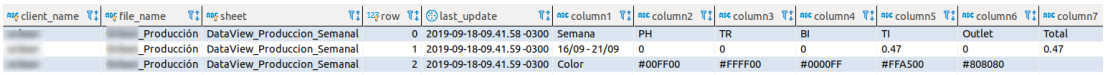


Figura 34: Datos insertados en la BD (imagen propia)

2.2.1.c) 5. Conexión de *Cyfe* con la base de datos de los *widgets*

DataView tiene la capacidad de leer directamente de *Google Sheets* (ya que *DataView* hace uso interno de otra herramienta, *Cyfe*), pero, esta integración no se utiliza (por diferentes inconvenientes)[10]. Debido a esto, se utiliza la base de datos (*DB Widgets*) mostrada previamente.

Cyfe, como se describe en los anexos, permite realizar consultas *SQL* para realizar los gráficos de los tableros (las consultas se realizan sobre la base de datos intermedia). Por la forma en la que se almacenan los datos en la tabla “data”, es muy fácil obtenerlos, ya que es posible filtrarlos por planilla, cliente y están ordenados por fila. Además, se puede configurar la frecuencia de actualización en la configuración del *widget*. Cada *widget* realiza consultas *SQL*, mediante un usuario de la base de configurado. La configuración de cada tablero, a ser asociado previamente a *DataView*, se realizó mediante *Cyfe*, ya que *DataView*

[10] El uso de la base de datos intermedia no es algo que se hizo dentro del trabajo, sino que ya estaba previamente, porque el servicio externo *Cyfe* nunca funcionó bien. La solución actual (el uso de la base de datos intermedia) es utilizada debido al requerimiento urgente de mantener operativa la integración de *DataView* y *Google Sheets*.

presenta a los usuarios una vista final, sin posibilidad de edición de los tableros generados y configurados y solamente se asocia el tablero a los usuarios de *DataView* correspondientes. Se muestra a continuación un ejemplo de tablero de *Cyfe*:

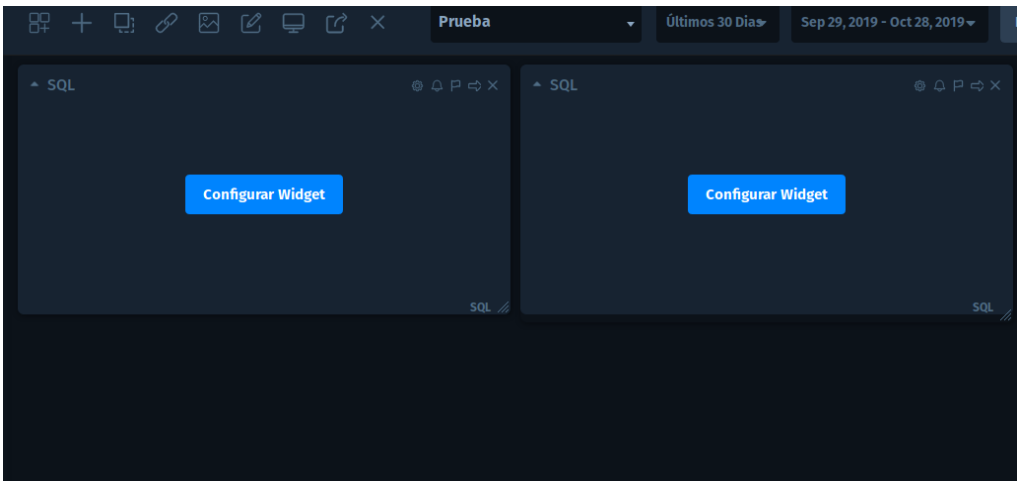


Figura 35: Menú de un tablero en Cyfe (imagen propia)

El paso siguiente es configurar un usuario en *Cyfe* para leer datos de la base de datos, como se muestra en la siguiente imagen:

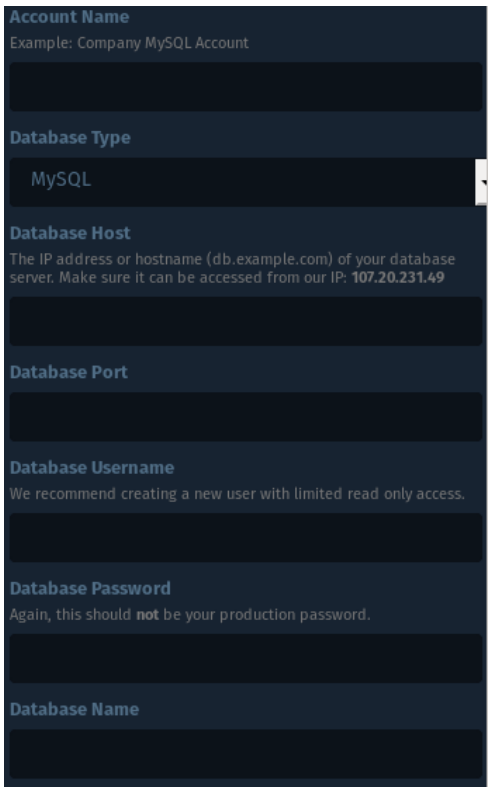


Figura 36: Configuración de usuario de DB en Cyfe (imagen propia)

Luego de haber configurado el usuario en *Cyfe* para acceder a los datos, se configura el *widget*, como muestra la imagen:

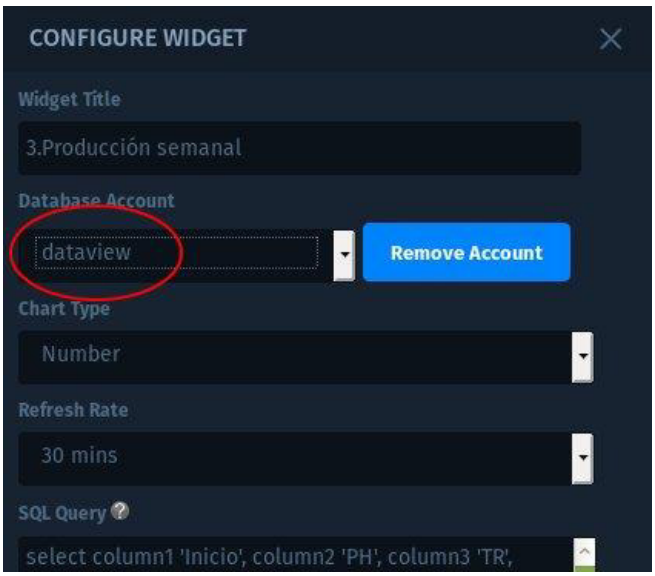


Figura 37: Configuración de un widget (imagen propia)

Se muestra, asimismo, la consulta *SQL* del *widget* configurado:

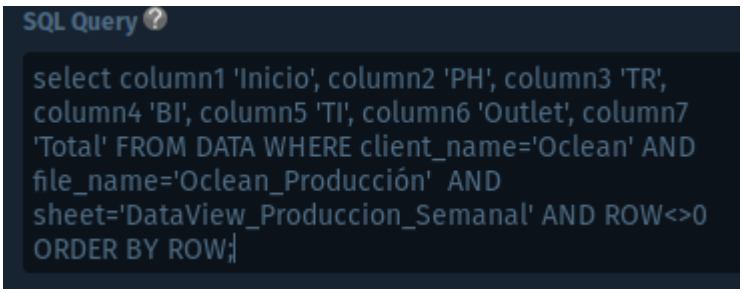


Figura 38: Consulta SQL del widget configurado (imagen propia)

Nótese que se filtró la primera fila (que lleva los nombres de cada columna), ya que no sirve para construir el gráfico del *widget*. Se asignó a cada columna el nombre deseado, característica que puede cambiarse en caso de que se considere necesario más adelante. Se muestra a continuación el *widget*, luego de haber configurado el usuario para leer los datos y el propio *widget*:

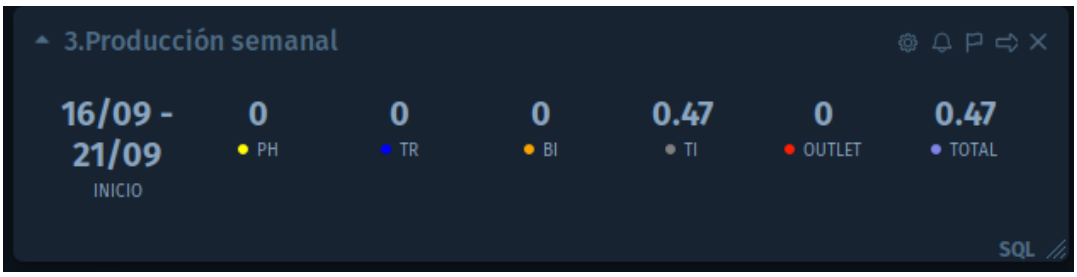


Figura 39: Widget configurado (imagen propia)

Una vez terminado un tablero con un *widget*, se integró a *DataView*. *Cyfe* permite hacer accesible un tablero mediante un identificador único, como se muestra a continuación:

Figura 40: Menú de Cyfe para hacer accesible al tablero (imagen propia)

Se observa en la siguiente imagen el link generado para permitir hacer accesible el tablero:

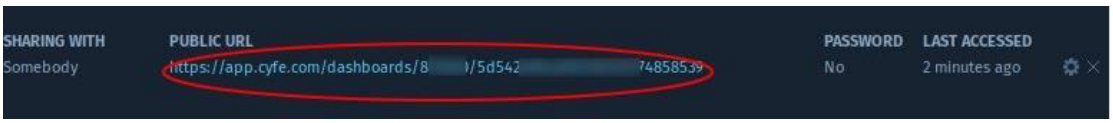


Figura 41: Link único asociado al tablero (imagen propia)

Este link único es embebido dentro de *DataView* al colocarlo dentro de la lista de tableros de una organización (que está compuesta por usuarios) por parte de un administrador. Cada usuario que tenga permisos de acceso en *DataView* podrá ver el tablero. Se omitirá la configuración del tablero en *DataView*, ya que no es de gran complejidad, comparado a los pasos anteriores; la aplicación ofrece un menú de configuración para esto, mostrado a continuación:

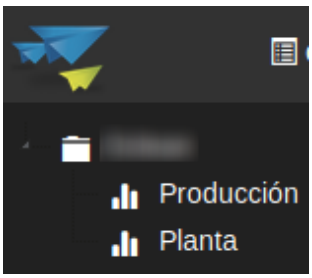


Figura 42: Tableros en DataView (imagen propia)

En la siguiente imagen, se presenta el tablero en *DataView*:

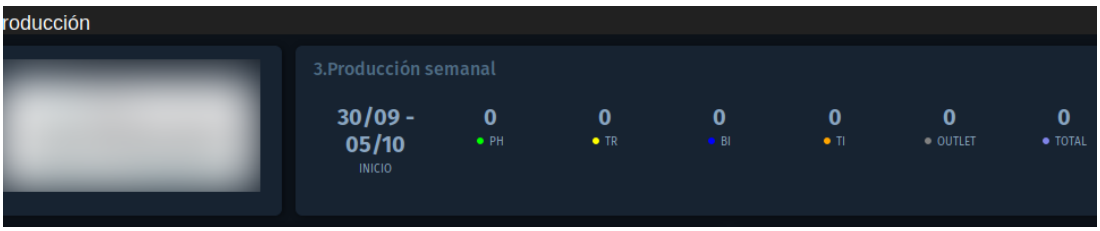


Figura 43: Tablero visualizado dentro de DataView (imagen propia)

Solamente se muestra el indicador configurado previamente, pero, en la práctica, son varios los *widgets* visualizados (en este caso, representa la producción semanal). La siguiente imagen contiene el tablero presentado, con varias modificaciones realizadas a lo largo del tiempo, mientras ocurrió el desarrollo de esta actividad. Se incluyeron otros indicadores solicitados por el usuario, pero, de todas formas, esto no afectó al desarrollo de la actividad, ya que lo principal era lograr la integración con el *ERP* y la extracción de datos.

Figura 44: Tablero configurado (imagen propia)

Como se observa, posteriormente, se reemplazó la producción de la semana por la producción de la semana anterior, por retroalimentación solicitada por el cliente.

2.2.2 Cierre

El cierre de esta actividad concluyó con la presentación del tablero al cliente. El desarrollo llevado a cabo deja varias conclusiones y consideraciones que ameritan ser comentadas y desarrolladas con mayor detalle dentro de las conclusiones generales de la práctica, que muestran el resumen de lo realizado, y además dan la opinión del autor sobre las actividades efectuadas (cf. sección “Conclusiones”). Luego de haber realizado la entrega y puesta en marcha, la actividad pasó a un estado de mantenimiento o soporte y modificaciones o mejoras al tablero, dadas por el uso de este por parte del cliente y la retroalimentación o *feedback* recibidos en base a su experiencia como usuario.[11]

[11] En las bases de datos relacionales (en general, corresponde a las que utilizan SQL como lenguaje), existe el concepto de procedimientos almacenados y funciones que son porciones de código que permiten realizar acciones sobre dichas bases, y, en muchos casos, se utilizan para agregar o modificar funcionalidades, sin tener que modificar la aplicación que interactúa con la base de datos.

Con respecto a la implementación, se puede destacar la arquitectura montada para dar solución a necesidades manifestadas por los clientes. Como se observa, la integración es un grupo de diferentes herramientas, tecnologías y lenguajes combinados mediante interfaces muy variadas en sus características (desde bases de datos, hasta planillas similares a Excel y aplicaciones externas, por nombrar algunas de estas herramientas), de las cuales se utilizan solamente una pequeña porción de sus funcionalidades. Sin embargo, esto proveyó de una gran flexibilidad a la hora de realizar la implementación buscada, ya que se notó que no solamente existe una forma de lograr la integración, sino que, además, se aligera el trabajo de realizar cambios o modificaciones solicitadas, como extraer más información.

2.3 Mejoras aplicadas al proceso de integración entre *LK-ERP* y *DataView*

En relación con esta actividad, originalmente, se pensó en modificar y realizar una mejora a la arquitectura de integración existente entre las dos aplicaciones. En la medida en que se avanzaba con el proceso, se encontraron algunos inconvenientes que afectaron la viabilidad de dicha actividad:

- con respecto a la primera mitad de la integración (exteriorización de datos del *ERP*), la arquitectura de integración actual se comenzó a utilizar en otros proyectos, que involucran la integración del *ERP* con otros sistemas de la organización (*Proteo* y otro producto); modificar esta estructura perjudicaría o complicaría este proyecto, ya que implicaría rehacer la integración realizada, por lo que no resultaba viable en este sentido.
- en cuanto al uso de *DataView*, debido a la utilización de una aplicación externa como *Cyfe*, aparecen varios pasos intermedios que podrían ser eliminados o incorporados a *DataView*, en caso de dejar de depender de esta herramienta; es decir, podría trabajarse sobre el diseño e implementación de un reemplazo de *Cyfe*, pero esto conllevaría un trabajo que escapa al alcance de esta actividad

2.3.1 Desarrollo

Se implementaron algunas mejoras desde el punto de vista de la seguridad y confiabilidad, que surgieron, en parte, de forma espontánea, según las necesidades que se fueron generando durante el desarrollo, implementación y uso de la arquitectura. En este sentido, se implementaron las siguientes mejoras:

1. aislamiento del acceso de información de *DataView* a diferentes clientes;
2. notificación en caso de que no se esté extrayendo información del *ERP* (en el caso de que *SheetL* no esté funcionando o el servidor esté apagado);

En el caso del aislamiento del acceso a la información de *DataView* por parte de los clientes, tiene su origen en el uso de la base de datos intermedia (*DB Widgets*), explicado en la sección correspondiente a la actividad anterior. A continuación, se muestra la porción de todo el proceso de integración involucrada entre *LK-ERP* y *DataView*:

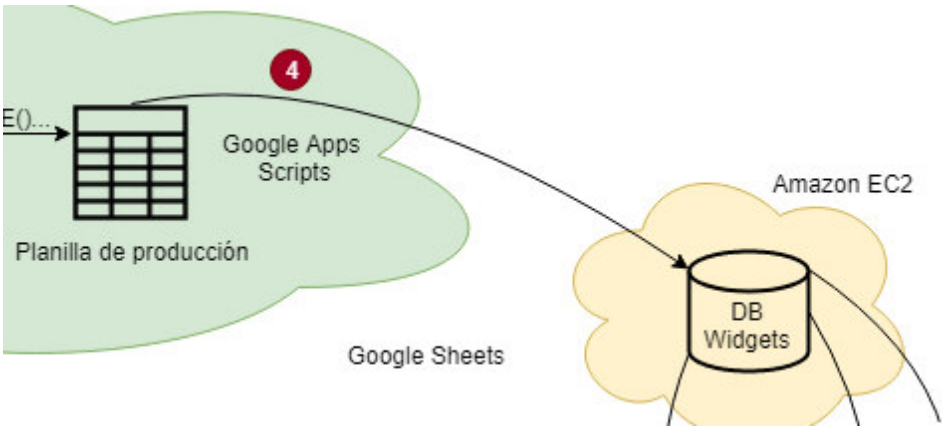


Figura 45: Conexión entre Google Apps Scripts y DB Widgets (imagen propia)

Debe observarse con detalle el hecho de que todas las planillas de producción de los clientes, mediante el uso de *Google Apps Scripts*, tienen una conexión directa con la tabla *data* de la base de datos. En todo *script* configurado, se debe colocar un usuario y contraseña, que es visible por toda la organización “cliente”, si el usuario tiene conocimiento técnico. Como la tabla almacena toda la información de los clientes, cada usuario que tenga acceso a la tabla podrá ver información que no le corresponde. En la siguiente figura, se observa la presencia de variables sensibles (es decir, datos de acceso), expuestos a los usuarios:

```
// database credentials
var dbUrl = 'URL_DATABASE';
var user = 'DB_USER';
var userPwd = 'DB_USER_PW'

//data
var client_name = 'CLIENT_NAME';
var file_name = 'CLIENT_SHEET';
```

Figura 46: Variables sensibles dentro de los scripts (imagen propia)

Las variables mostradas se utilizan para realizar la conexión a la base de datos intermedia, como se ilustra a continuación:

```
try {
 var conn = Jdbc.getConnection(dbUrl, user, userPwd);

 var stmt = conn.createStatement();
 var start = new Date(); // Get script starting time
```

Figura 47: Variables sensibles utilizadas para la conexión (imagen propia)

La solución generada frente a este problema fue aplicar el concepto de “vistas” o tablas virtuales, que son tablas que permiten al usuario trabajar sobre un conjunto parcial de datos. En las siguientes dos tablas, se muestra una representación simplificada de la diferencia entre una tabla que incluye todos los datos, y una tabla virtual, que aisla los datos de tal forma que cada cliente solo acceda a su información:

cliente	planilla	hoja	columna1	columna2	...	columnan
cliente1	planilla1	hoja1	DATO1	DATO2	...	DATOn
cliente2	planilla1	hoja1	DATO1	DATO2	...	DATOn
cliente1	planilla1	hoja2	DATO1	DATO2	...	DATOn

Figura 48: Representación simplificada de la tabla data (imagen propia)

cliente	planilla	hoja	columna1	columna2	...	columnan
cliente1	planilla1	hoja1	DATO1	DATO2	...	DATOn
cliente1	planilla1	hoja2	DATO1	DATO2	...	DATOn

Figura 49: Representación simplificada de una tabla virtual (figura propia)

En el caso de la imagen anterior, esta tabla virtual es aplicada al cliente 1, para que solamente vea sus datos.

Se aplicó esta metodología para aislar a cada cliente, logrando que cada uno solamente pueda acceder a sus datos. Se creó una vista y un usuario específico solamente con permisos para acceder a la vista o tabla virtual.

Se asignó según el cliente su usuario y contraseña correspondientes, dentro de los *scripts* que funcionan junto a las planillas de *Google Sheets*. De esta forma, se logró brindar una pequeña cuota de seguridad adicional, ya que, previamente, se encontraba un riesgo presente.

Por otro lado, se añadió una mejora al uso de *SheetL* para la extracción de datos del *ERP* en su planilla[12]. Con respecto a la extracción de información del *ERP*, han ocurrido algunos inconvenientes, luego de haber puesto en marcha la práctica. Por ejemplo, en algún momento, el tablero generado para el cliente dejó de mostrar información reciente, y se especularon dos orígenes para este problema:

- los usuarios del *ERP* no cargaron información. Suele haber saltos en el hábito de volcar la información al sistema por parte de los usuarios.
- *SheetL* no está funcionando o funciona incorrectamente, por lo que la información no llega a volcarse en la planilla.

[12] Mencionada como “Planilla de *SheetL*” en el diagrama representado en la figura 14 que esquematiza la integración entre el *ERP* y *DataView*.

Accediendo de forma remota al servidor, se encontró que había sido apagado y reiniciado (voluntariamente o por una causa externa) y *SheetL* no se había iniciado correctamente, lo cual implicó que cesara la carga de datos a la planilla.
Esto implicó un retraso en la carga de datos a los tableros de más de 5 días, por lo que se ideó una solución para notificar de forma rápida en caso de haber una falla en la carga de información. Se muestra a continuación una imagen que representa los últimos resultados de una consulta:

Query	Descripción	Origen De Datos	Frecuencia	Resultado	Jltima Ejecució	Último Error
Declare @FechaInicio DATET SET @FechaInicio = '201901(select * from (select DiasDeCarga.Fec cast(ISNULL(unidad	Reporte de producción diaria, discriminada por 5 rubros en unidades y kilos desde la fecha	ds1	60	Prod_Diaria!A2	03/11/2019 20:33:01	

Figura 50: Últimos resultados volcados en la planilla (imagen propia)

En el caso de que la última ejecución supere un límite especificado, debería enviarse una notificación por email al administrador. Para esto, se hizo uso de *Google Apps Scripts*, programando una función periódica que, para cada consulta ejecutada por *SheetL*, compruebe la fecha de la última ejecución. Se muestra en la siguiente imagen la función programada que permite implementar la mejora:

```
function checkQueriesLastResults() {
  try {
    var today = new Date();
    var ss = SpreadsheetApp.getActiveSpreadsheet();
    ss.setActiveSheet(ss.getSheetByName("sql"));
    var data = ss.getDataRange().getValues();
     for (var i = 1; i < data.length; i++) {
      var dateParts = data[i][5].split("/");
      var timeParts = dateParts[2].split(" ")[1].split(":");
      dateParts[2] = dateParts[2].split(" ")[0];
      query_date = new Date(+dateParts[2], dateParts[1] - 1, +dateParts[0], timeParts[0], timeParts[1], timeParts[2]);
      var diffInDays = Math.floor((today.getTime() - query_date.getTime())/(1000*60*60*24));
      if (diffInDays >= 2)
        mailSend(i+1); // Número de query
    }
  } catch(err) {
    errorReport(err);
  }
}
```

Figura 51: Función que implementa la funcionalidad (imagen propia)

En la siguiente imagen, se muestra el email enviado por la función, en caso de detectar que se encuentre algún problema en la planilla de *SheetL*:

Julasoft - [redacted] SheetL - Problema cargando datos de SheetL

para mí

Hace mucho tiempo que no se cargan resultados de la query de la fila 2,
¿tal vez SheetL no esté corriendo o el servidor esté apagado?

Figura 52: Correo de notificación (imagen propia)

Esta mejora ayudó a automatizar el monitoreo sobre *SheetL* en el servidor del cliente[13], ya que permitió minimizar el tiempo dedicado a revisar el estado del servidor por lapsos o períodos determinados o preestablecidos.

[13] Por ejemplo, originariamente, *SheetL* se configuró incorrectamente, y, en caso de reinicios en el servidor del cliente, la aplicación no se iniciaba correctamente como servicio del sistema; gracias a esta corrección, se pudo detectar y corregir este problema.

2.3.2 Cierre

Esta actividad se planteó inicialmente con la idea de generar un mayor cambio dentro de la arquitectura de la integración entre *LK-ERP* y *DataView*. Debido a las razones explicadas al inicio de la sección, se encontraron diferentes cuestiones que limitaron la posibilidad de proponer cambios más profundos o con un impacto mayor, pese a que la integración se realizó eficazmente y se realizaron otro tipo de mejoras explicadas en el desarrollo previo. De todas formas, esto posibilitó el poder brindarle una mayor cantidad de tiempo al desarrollo de las demás actividades.

2.4 Mejora de procedimientos de monitoreo y gestión de servidores

Dentro de estas estas tareas, se realizó el conjunto de actividades mostradas a continuación, que motivaron el desarrollo de un conjunto de aplicaciones para agilizar tareas operativas que se llevan a cabo dentro del área de operaciones.

El desarrollo implicado en estas actividades está concluido y las aplicaciones y resultados se encuentran operativos, pero la incorporación de mejoras incrementales (es decir, pequeños cambios que proporcionan nuevas funcionalidades o solucionan errores) sobre las aplicaciones desarrolladas continúa hasta la actualidad. Lo realizado en este período ha servido como base para ampliar el futuro alcance de la funcionalidad de las aplicaciones implementadas, a medida que surjan nuevas necesidades dentro de la organización, en relación a la gestión operativa.

2.4.1 Desarrollo

Las principales actividades efectuadas fueron las siguientes:

1. Realización del relevamiento e inventario de los servidores.
2. Programación de una aplicación para monitorear los discos de los servidores.
3. Desarrollo de un programa para administrar las copias de seguridad de las bases de datos.
4. Determinación de indicadores críticos relevantes para graficar mediante la aplicación *DataView.*

A pesar de que previamente ya se ha presentado el origen de este planteo, en la sección 1.2.1, aquí debe describirse con mayor detalle, en primer lugar, la problemática y situación existente en la organización. Dentro del área de operaciones, se dedican aproximadamente entre diez y doce horas mensuales a las siguientes tareas:

- controlar que las copias de seguridad de bases de datos de diferentes servidores se hayan realizado y se hayan sincronizado con un alojamiento externo en la nube (*Dropbox*);

- revisar el estado de los discos de los servidores, ya que cada servidor suele tener varios discos virtuales asignados;
- monitorear algunos servidores particulares, ya que, por diversas razones, pueden dejar de funcionar temporalmente o su rendimiento está deteriorado.

Previamente se realizaron algunos intentos por automatizar estas tareas, ya que existen dos *scripts* que realizan algunas funciones y se ejecutan dentro de los servidores, pero se han encontrado varios inconvenientes. Algunos impedimentos que presenta esta solución provisoria son los que se listan a continuación:

- la solución actual no es portable a nivel sistema operativo, es decir, solamente funciona con Windows;
- la solución no es configurable; todo parámetro (datos del servidor, bases de datos presentes, etc.) está colocado en el código fuente;
- debido a su programación, la solución actual no es reutilizable ni extensible;
- las copias de seguridad generadas por la solución actual no tienen garantía de ser guardadas en la nube;
- no existe un aseguramiento sobre la ejecución de la solución presente, puesto que no se generan registros;
- cada cierto tiempo, es necesario revisar en qué estado se hallan estas aplicaciones (copias no realizadas, errores durante ejecuciones previas, etc.)

En relación a la no portabilidad, se debe a que uno de los *scripts* es un archivo Batch (archivo de procesamiento por lotes), con una serie de instrucciones del tipo DOS. Si bien la empresa actualmente utiliza mayoritariamente *Windows Server*, tiene planes de migrar al sistema GNU/Linux en el futuro. Sobre la programación, por la naturaleza de la programación procedural, esta resulta poco legible o comprensible a quien desee realizar modificaciones, por lo que no son extensibles. En el caso del *script Batch*, que es el encargado de realizar los *backups*, hace uso de prácticas de *hardcoding* (es decir, coloca datos dentro del código fuente que, en realidad, deberían ser configurables). Así, por

ejemplo, funciona para la ejecución de algunas aplicaciones externas[14], como se muestra a continuación. En la figura, puede observarse el uso de las rutas absolutas:

```
set mysqlHome=C:\Program Files\MySQL\MySQL Server 5.5\bin
cd %mysqlHome%
mysqldump -h%MysqlHost% -u%MysqlUser% -p%MysqlPass% --database %DBName% --routines > "%DirForBackups%\%backupDate%\%DBName%-%bac
mysqldump -hlocalhost -u%MysqlUser% -p%MysqlPass% --database %DBName% --routines > "%DirForBackups%\%backupDate%\%DBName%-%backu

@echo Zipping ...   >> dbBakpAndFtpLog.log
"C:\Program Files\7-Zip\7zG.exe" a -tzip "%DirForBackups%\%backupDate%\%DBName%-%backupDate%_%serverId%.zip" "%DirForBackups%\%b
```

Figura 53: Ejemplo del script actual (imagen propia)

Se manifiesta otro problema en relación a la subida de las copias de seguridad en la nube. Las copias de seguridad se sincronizan con una cuenta de Dropbox mediante una aplicación de escritorio. La aplicación cliente funciona en segundo plano, y, cada cierto tiempo, sube los archivos volcados en la nube. El problema es que por razones que se desconocen (y no se ha podido encontrar la causa) deja de funcionar azarosamente y obliga a que el administrador, cada cierto tiempo, tenga que revisar de forma remota el estado de la aplicación en los diferentes servidores, reiniciándola, en caso de que sea necesario. De allí, el uso de una parte de las doce horas mensuales dedicadas a estas tareas. A causa de la frecuencia elevada de los *backups* (diariamente), resulta desfavorable la ocurrencia de estos problemas.

Estos problemas se visibilizaron durante el análisis de la situación. De todas maneras, resultó útil tomar nota sobre el estado actual de las herramientas, ya que, además, permitió sumar consideraciones para las actividades a realizar. La posibilidad de diseñar una solución a esto generó una discusión y planteo con respecto a qué es lo que debería y podría realizarse.

Se montó un repositorio externo *Git* (*Bitbucket*) para la gestión del código fuente y posterior despliegue y uso en los demás servidores. Al trabajar mediante *Git*, se utilizaron dos ramas

[14] Para ser más específico, la solución actual inicia una herramienta (*mysqldump*) propia del servidor de base de datos, utilizada para realizar una copia de seguridad y coloca la ruta completa (o ruta absoluta) del archivo, en lugar de delegar ese trabajo al sistema operativo, cuando lo ideal es no generar este tipo de dependencias, ya que esto produce el problema de que, en caso de que cambie la versión de la base de datos, lo más común actualmente, este se instale en otro lugar o no sea el mismo SO, el programa no funcionará.

de desarrollo para gestionar la incorporación de mejoras y nuevas funcionalidades (ya que inicialmente no se tuvieron en cuenta todas las características nombradas). Las dos ramas son las siguientes:

- master: rama donde se encuentra el código estable y listo para ser colocado en los servidores;
- develop: rama donde se vuelcan los cambios, previo a las pruebas.

Además, para el desarrollo, se trabajó con una herramienta de *Python* que permitió simplificar el flujo de trabajo de programación, permitiendo generar un entorno virtual (versión de *Python* a utilizar, gestión de dependencias, etc.) llamado *Pipenv*.

2.4.1.a) Desarrollo de *Julatools*

Esta actividad se dedicó al diseño y desarrollo de un conjunto de herramientas que permitieran resolver las cuestiones problemáticas vinculadas a las rutinas operativas y, además, permitieran atender nuevas solicitudes a futuro. En principio, se plantearon las siguientes características que debería tener este conjunto de herramientas, llamado *Julatools*:

- capacidad para configurar discos para comprobar sus estados (por ej. porcentaje de uso)
- posibilidad de realizar *backups* de varias bases de datos;
- uso de compresión para el ahorro de espacio;
- registro de eventos (tanto de errores como de acciones ejecutadas satisfactoriamente);
- portabilidad a nivel de sistemas operativos;
- integración con Dropbox;
- capacidad de envío de emails a diferentes receptores, en caso de que sea necesario notificar ciertas acciones o resultados;

- centralización de la información generada por estas herramientas, mediante el uso de *DataView*.

Desde el punto de vista de las especificaciones no funcionales, se consideró que sería apropiado que el sistema tuviese las siguientes características:

2. programación en *Python* 3, ya que es un lenguaje muy popular y utilizado, lo cual facilita su extensión a futuro por parte de otras personas;
3. uso de estándares PEP8 (estándar de estilo de código en *Python*);
4. orientado a objetos, para favorecer el diseño funcional;
5. documentación *inline* (documentación dentro del código fuente, especialmente en la definición de ciertos módulos y sus funciones), para facilitar su legibilidad; en la siguiente imagen, se muestra un ejemplo de esto:

```
def log_error(self, caller, message):
    """
    Escribe un mensaje (message) de ERROR en el log utilizando el nombre de la clase que llamó al método (caller)
    :param caller: object
    :param message: str
    :return: None
    """
```

Figura 54: Documentación inline (imagen propia)

Una cuestión discutida con el cliente fue cómo debería ser la ejecución de estas herramientas, ya que la idea es que, cada cierto tiempo (es decir, con una determinada frecuencia), se realicen ciertas tareas. Se contrapusieron dos alternativas:

- desarrollar un planificador de tareas, que, en base a una configuración, realice las rutinas descritas previamente y que trabaje en segundo plano, como un *daemon*[15];
- construir pequeños módulos que sean independientes y cuya ejecución sea planificada mediante herramientas existentes en los sistemas operativos, tales como *Task Scheduler* en *Windows* y cron en GNU/Linux.

[15] Un *daemon* o demonio es la denominación dentro del ámbito informático de un proceso o servicio que trabaja en segundo plano. (The Linux Information Project, 2006)

Se decidió por la segunda alternativa, siguiendo con la filosofía de *UNIX*, predecesor de *GNU/Linux*, y continuador de sus principios: software mínimo, que realice una única tarea y coopere con otras herramientas pequeñas para realizar acciones más extensas. Así, se espera que *Julatools* funcione como un conjunto de herramientas pequeñas, que hagan uso compartido de diversos módulos que encapsulan comportamiento común, como, por ejemplo, las notificaciones por email o el registro de eventos. Dichos módulos se denominan "herramientas auxiliares".

Para esta aplicación, se consideró que tiene más ventajas la ejecución cada cierto tiempo de tareas pequeñas, en lugar de desarrollar o planificar un servicio en segundo plano. Una aplicación que corre dentro del servidor de forma ininterrumpida, en algún momento, podría tener fugas de memoria (o *memory leaks*), errores no descubiertos, apropiación recursos, etc. En cambio, con una aplicación que tiene un plazo de ejecución breve, se evitan estos problemas.

Además de las herramientas auxiliares que se describirán en segundo lugar, para esta actividad, se desarrollaron las siguientes tres herramientas principales, que tienen puntos de entrada, es decir, se ejecutan directamente:

- jula_ckup: para la gestión de las copias de seguridad;
- jula_dsk: para la gestión de discos;
- jula_logger: para el envío de logs de la aplicación.

La configuración de todas las herramientas (principales y auxiliares) es compartida por los diferentes puntos de entrada, para una simplificación de configuración; si no, el administrador de la aplicación debería configurar varias veces las diferentes herramientas auxiliares. El siguiente diagrama esquematiza la arquitectura completa de *Julatools*:

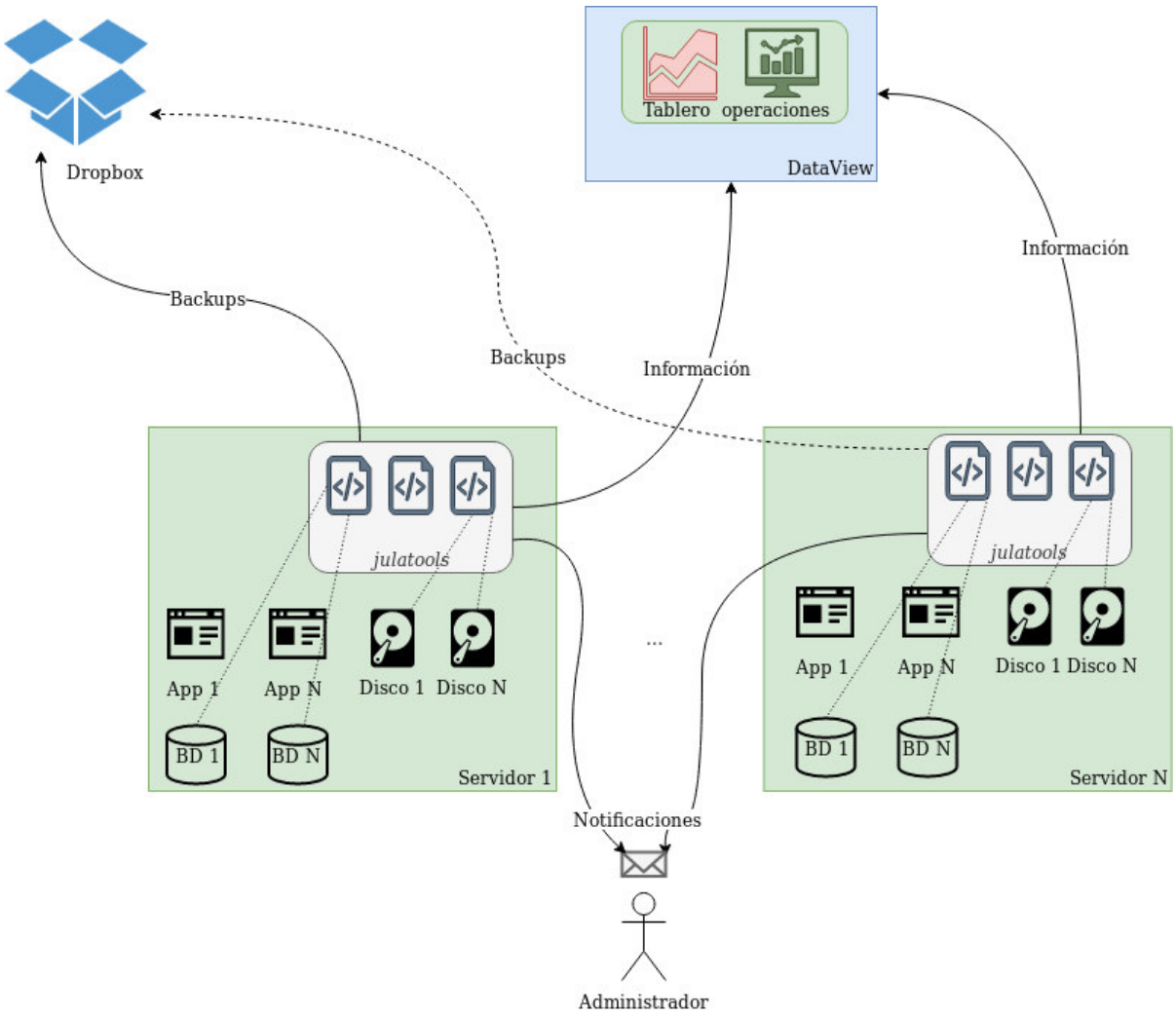


Figura 55: Diagrama que condensa la arquitectura de julatools (imagen propia)

2.4.1.a) 1. Herramientas auxiliares

Como se explicó antes, se focalizó el desarrollo orientado a objetos. Para esto, se encapsularon diversos comportamientos comunes a cada una de las herramientas realizadas en objetos, que funcionan como herramientas auxiliares para las herramientas principales. Estos comportamientos comunes son los siguientes:

- abstraer la lectura y carga de configuración (lo que, en inglés, se denomina *parse*);

- encapsular la conexión a un servicio SMTP (correo electrónico) y el envío de correos;
- administrar el *logging* de eventos en archivos locales y evitar problemas como controlar el crecimiento de tamaño de los archivos de logs;
- aislar la conexión a bases de datos y el envío de información con el formato adecuado para *DataView,* a fin de centralizar la información;
- esconder y abstraer la interacción con *Dropbox* (que se realiza mediante una API[16] propia de *Dropbox*);

Para esto, se crearon las siguientes clases (objetos), que se explicarán más adelante:

- ConfigReader
- MailSender
- Logger
- DataViewPusher
- DropboxManager

ConfigReader

Es una clase que permite leer el archivo de configuración. Para la configuración *Julatools* se decidió utilizar un formato de texto llamado *TOML* (en inglés, la sigla corresponde a la designación “Tom's Obvious, Minimal Language”), debido a que, como lo indica su nombre, es minimalista y su sintaxis se asemeja a la de *Python*. Como es necesario realizar un *parse,* es decir, un análisis sintáctico del archivo, se utilizó una biblioteca específica para *Python* (**toml**) dentro la clase ConfigReader. A continuación, se muestra un diagrama UML de clases de ConfigReader:

[16] *Dropbox* tiene una API para Python cuya documentación se encuentra disponible en la web Dropbox for Python Documentation.

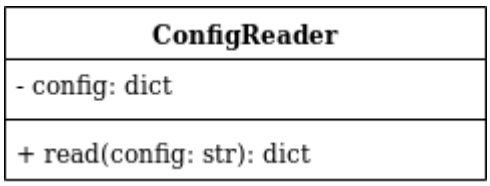


Figura 56: Diagrama UML de ConfigReader (imagen propia)

El diagrama solamente posee una variable, que es la configuración entera, y un método para leer la configuración requerida en cada clase (ya que *TOML* permite dividir un archivo según claves, de manera similar a un diccionario[17]). Se muestra a continuación la lectura (*parsing*) de un archivo *template* de la configuración que se asigna en cada servidor y el archivo que genera en sí una variable del tipo diccionario:

Figura 57: Archivo .toml leído (imagen propia)

A continuación, en la siguiente imagen, se muestra cómo es la estructura parcial del archivo de configuración. Se muestran solo variables de configuración de algunas herramientas, no de todas:

[17] Como tipo de dato se entiende por “diccionario”, el tipo de variable presente en *Python* que se caracteriza por dividir la información en clave-valor; mediante una clave, accede al valor que esta almacena (Python Software Foundation, 2001).

```ini
[server]
name = 'test-server'
connection_url = 'xxxx'  # URL de conexión
user = 'username'
password = '...'

[mail_sender]
enable = true
smtp_server = 'smtp.gmail.com' # Servidor STMP
smtp_server_port = 587
enable_tls = true
sender_username = 'emailtest@gmail.com'
sender_password = 'password_test'
receivers = ['admin1@test.com', 'admin2@test.com'] # Receivers

[logger]
enable = true
filename = '/julatools_files/julatools.log'

```

Figura 58: Archivo plantilla de configuración (imagen propia)

Al leer la configuración, se genera un diccionario donde cada sección utilizada por las diferentes clases es accesible por su clave (server, mail_sender, db_saver, etc.).

MailSender

Es una clase que se encarga de gestionar la conexión a servidores de email y el envío de estos, ya que, en muchos casos, es necesario notificar al administrador. En la imagen siguiente, se ilustra un diagrama UML de clases de MailSender:

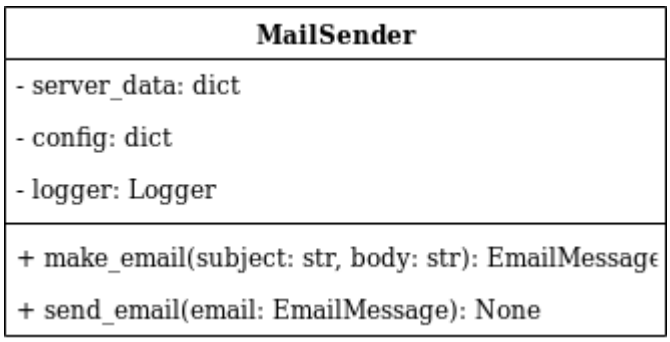


Figura 59: Diagrama UML de MailSender (imagen propia)

En relación a sus variables, el diagrama tiene información de su configuración (y datos del servidor donde se encuentra colocado) y una variable del tipo *Logger*, ya que hace uso de la herramienta *Logger,* que se desarrollará más adelante, a fin de registrar sus acciones. Tiene los métodos para generar un objeto del tipo EmailMessage (abstracción presente entre las bibliotecas de *Python* para representar un correo electrónico) y para enviar ese email en base a la configuración provista.

Logger

Esta clase provee a *Julatools* de una simplificación para dejar registro de eventos y sucesos por parte de las demás herramientas. Con “simplificación”, se hace referencia a la abstracción para realizar las siguientes acciones:

- dejar registros con un formato uniforme, respetado por todos los componentes;
- no superar el límite de tamaño del archivo de un log;
- limpiar el archivo de registro, en caso de que haya excedido el límite.

Se muestra a continuación un ejemplo de registros dejados por diferentes herramientas:

```
51 2019-10-11 16:24:01,324 - Logger - INFO - DiskChecker: Disco / con tamaño 219 GiB; 42.0% de uso (límite 80.0%)
52 2019-10-11 16:24:01,393 - Logger - INFO - DataViewPusher: Datos del tipo DISK insertados en DataView
53 2019-10-19 14:48:38,365 - Logger - ERROR - LogUploader: Error leyendo información del log de DataView: time data '2019-10-11 16:24:01,324
54 2019-10-19 14:49:44,689 - Logger - ERROR - LogUploader: Error leyendo información del log de DataView: time data '2019-10-11 16:24:01,393
55 2019-10-19 14:49:56,286 - Logger - ERROR - LogUploader: Error leyendo información del log de DataView: unconverted data remains:
```

Figura 60: Registros dejados por varias herramientas (imagen propia)

En la siguiente imagen, se observa un diagrama UML de clases de *Logger*:

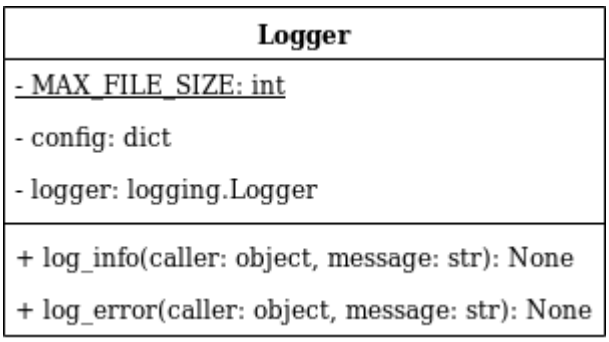


Figura 61: Diagrama UML de Logger (imagen propia)

Entre las variables de la clase *Logger*, se define un límite en bytes para el tamaño de archivo (se trata de un límite constante, no se planteó como modificable), su propia configuración y, además, se estableció que utilizara internamente una biblioteca de *Python* para el *logging*, lo cual implica que esta clase construye una abstracción sobre las abstracciones brindadas por la biblioteca de *Python*. Esta clase ofrece a las demás herramientas dos métodos para registrar eventos exitosos y dar cuenta de eventos con salida errónea.

DataViewPusher

Esta clase permite la centralización de información. Su desarrollo estuvo relacionado con la determinación de indicadores de relevancia para el área de operaciones. Si bien, en principio, son pocos los datos importantes, ya que la actividad se orientó en esta instancia a solventar lo más crítico, o sea, las bases de datos y los discos, se generalizó la solución para, en el futuro, poder agregar más categorías de información. *DataViewPusher* es una clase que permite, de la misma forma que en la actividad anterior se volcó información en *DataView* (mediante *Google Sheets* y otros componentes), colocar información de relevancia sobre rutinas operativas, en un formato legible por *DataView*. La figura siguiente muestra la información volcada por *DataViewPusher* en la base de datos:

server	type	last_updated	column1	column2	column3	column4	column5
alt-db-m1	DB	2019-10-31 22:58:58	Servidor	BD	Última copia	Archivo	Tamaño
alt-db-m1	DB	2019-10-31 22:58:59	alt-db-m1	alertatel2	2019-10-31 23:00:02.929746	31-10-2019_23-00-02_alt-db-m1_alertatel2_E	206.24 MiB
dataview	DB	2019-10-31 23:25:37	Servidor	BD	Última copia	Archivo	Tamaño
dataview	DB	2019-10-31 23:25:37	dataview	dataview	2019-10-31 23:30:01.477724	31-10-2019_23-30-01_dataview_dataview_BA	0.02 MiB
julasoft-testing	DB	2019-10-31 23:00:02	Servidor	BD	Última copia	Archivo	Tamaño
julasoft-testing	DB	2019-10-31 23:00:02	julasoft-testing	dataview	2019-10-31 23:00:01.502484	31-10-2019_23-00-01_julasoft-testing_datavi	0.03 MiB
proteo	DB	2019-10-31 15:59:34	Servidor	BD	Última copia	Archivo	Tamaño
proteo	DB	2019-10-31 15:59:34	proteo	proteo	2019-10-31 16:00:01.814552	31-10-2019_16-00-01_proteo_proteo_BACKU	6.2 MiB
scada-db	DB	2019-10-31 23:01:03	Servidor	BD	Última copia	Archivo	Tamaño
scada-db	DB	2019-10-31 23:01:03	scada-db	scada	2019-10-31 23:00:03.938572	31-10-2019_23-00-03_scada-db_scada_BACK	11.98 MiB
scada-db	DB	2019-10-31 23:01:03	scada-db	scada_hist	2019-10-31 23:00:27.576360	31-10-2019_23-00-27_scada-db_scada_hist_E	50.81 MiB

Figura 62: Información de las herramientas funcionando en varios servidores (imagen propia)

En este caso, se muestra información de bases de datos. La herramienta para gestionar bases de datos es jula_ckup y se desarrollará más adelante.

Se muestra a continuación el diagrama UML de clases de DataViewPusher:

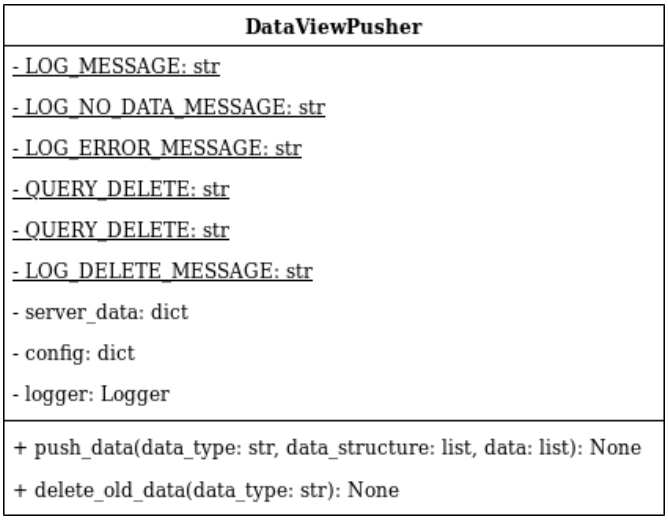


Figura 63: Diagrama UML de clase de DataViewPusher (imagen propia)

Las variables de clase (en mayúsculas y subrayadas) son constantes con mensajes predefinidos (para registros y consultas SQL, entre otros, ya que interactúan con la base de datos de *DataView*). Las variables de instancia repiten el esquema de los casos anteriores (configuración propia, información del servidor donde se despliega, etc.).

En relación con los métodos ofrecidos, esta herramienta permite el envío de datos y el borrado de datos antiguos, con respecto a un servidor y a un tipo de dato. Hace uso de una biblioteca de *Python* para la interacción con bases de datos *MySQL* (**mysql-connector**). Para el envío de datos, se debe especificar la siguiente información:

- **tipo de dato** (disco, base de datos, logs, etc.);
- **estructura de datos**: lista cuyo contenido es similar al encabezado de una tabla;
- **filas de datos**: lista de filas, donde cada fila contiene las celdas de una columna.

Se muestra a continuación una ejemplificación gráfica de los puntos anteriores; el tipo se resalta en verde; la estructura, en rojo y las filas, en azul:

DISK	Servidor	Disco	Tamaño	Uso	Ultimo Check	Estado
DISK	julasoft-testing	C:	29 GiB	88.1	2019-11-01 00:00:00.419169	OK
DISK	julasoft-testing	D:	79 GiB	93.42	2019-11-01 00:00:00.419169	OK

Figura 64: Ejemplo de la estructura de datos (imagen propia)

A la información recibida para enviar a *DataView*, se suman metadatos propios del servidor (como el nombre del servidor, la fecha de envío, etc.)[18]. Al momento de volcar información, se borra la información enviada previamente por ese servidor con respecto al tipo de dato. Así como actualmente se utilizan tres tipos de datos, se pueden crear más, de forma dinámica, en base a la necesidad, ya que el esquema es genérico.

DropboxManager

La clase DropboxManager se encarga de encapsular la comunicación con la API de *Dropbox* (biblioteca **dropbox** en *Python*). Para esto, *Dropbox* permite generar un identificador único asociado a una cuenta (*token*), para ser utilizado con una aplicación. Dentro de la configuración del identificador, es posible limitar el acceso de la aplicación a los archivos almacenados en la nube, por lo que se generó una carpeta específica para *Julatools*. Toda interacción con *Dropbox* se encuentra aislada dentro de esa carpeta, es

[18] A cada servidor se lo identifica con la información presente en la primera sección del archivo de configuración colocado en cada uno (ver figura 56, en la sección 2.4.1.a).

decir, no se puede acceder, modificar o eliminar elementos que no estén dentro del alcance de la carpeta generada específicamente para la aplicación. A continuación, se muestra la generación del token en *Dropbox* para *Julatools* desde la página web de *Dropbox*:

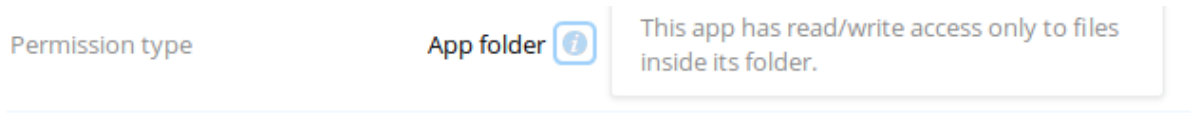


Figura 65: Configuración del token en Dropbox (imagen propia)

La siguiente figura contiene el diagrama UML de clases de DropboxManager:

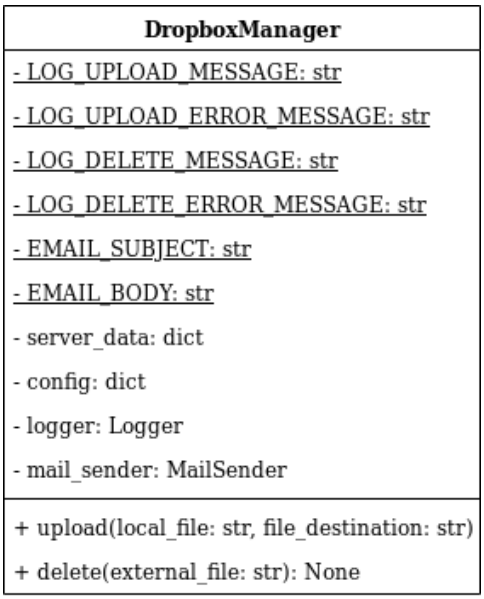


Figura 66: Diagrama UML de DropboxManager (imagen propia)

Se muestran varias variables de clase, que se utilizan como constantes y que definen mensajes y variables de instancia (server_data, config, etc.), cuya función es similar a la de las demás clases. La herramienta provee dos métodos para subir un archivo y para eliminar un archivo, respectivamente. En el caso del método *upload*, se gestiona la subida mediante transacciones, si el tamaño del archivo es elevado (arriba de varios MiB).

2.4.1.a) 2. Jula_ckup

Se denominó *jula_ckup* a una de las principales herramientas, compuesta por un archivo *Python* que contiene el punto de entrada (función *main*) que se encarga de instanciar los diferentes objetos (herramientas auxiliares) y coordinarlos para que lleven a cabo las tareas especificadas. En este caso, se trata de las tareas de realizar copias de seguridad sobre el servidor, subirlas a la nube, dejar registro, borrar viejas copias de seguridad y notificar por email, en caso de haber problemas. La clase que encapsula la funcionalidad de estas tareas es *DBSaver*. Una vez que *jula_ckup* inicializa los objetos, delega la ejecución a la función *make_backups de DBSaver.* Se muestra en la siguiente imagen el punto de entrada de jula_ckup:

```
def main(args):
    config_file = None
    if len(args) != 2:
        config_file = os.path.join(os.path.dirname(os.path.realpath(__file__)), "config_local.toml")
    else:
        config_file = args[1]
    cfg = ConfigReader(config_file)
    logger = Logger(cfg.read("server"), cfg.read("logger"))
    dataview_pusher = DataViewPusher(cfg.read("server"), cfg.read("dataview_pusher"), logger)
    mail_sender = MailSender(cfg.read("server"), cfg.read("mail_sender"), logger)
    dropbox_manager = DropboxManager(cfg.read("server"), cfg.read("dropbox_manager"), logger, mail_sender)
    db_saver = DBSaver(cfg.read("server"), cfg.read("db_saver"), logger, mail_sender, dataview_pusher, dropbox_manager)
    db_saver.make_backups()
    return 0
```

Figura 67: Punto de entrada de jula_ckup (imagen propia)

La figura próxima ilustra el diagrama UML de clases de DBSaver:

Figura 68: Diagrama UML de DBSaver (imagen propia)

Se puede observar cómo esta clase hace uso de las demás clases auxiliares para su ejecución. Las variables tienen la misma funcionalidad que en los casos previos, aunque se suma el uso de definiciones para el envío de datos a *DataView*:

```
DATAVIEW_DATA_STRUCTURE = ['Servidor', 'BD', 'Última copia', 'Archivo', 'Tamaño']
DATAVIEW_TYPE = 'DB'
```

Figura 69: Estructura de los datos a enviar a DataView (imagen propia)

El método make_backups realiza las siguientes acciones para cada base de datos configurada:

1. realizar un backup con la herramienta mysqldump (genera archivo .sql); en relación al uso de mysqldump (ver sección 4.4.b del anexo);
2. comprobar que no haya *backups* viejos -se puede configurar un límite de *backups* almacenados;
3. eliminar *backups* antiguos en caso de que se supere el límite configurado;
4. comprimir el *backup* realizado (genera archivo .zip), utilizando una biblioteca de *Python*;

5. eliminar el archivo backup sin comprimir (archivo .sql);
6. registrar en log (delega la tarea a *Logger*);
7. generar información con el formato adecuado para volcar en *DataView* (delega la tarea a *DataViewPusher*);
8. enviar información a *DataView* (delega tarea a *DataViewPusher*);
9. eliminar backups viejos en *Dropbox* (delega la tarea a DropboxManager);
10. sube backups nuevos a *Dropbox* (delega la tarea a DropboxManager).

En el caso de que durante la ejecución de cualquiera de estos pasos se genere un error, se envía un email notificando el problema (*MailSender*) y se registra el error (*Logger*). Existen otros métodos en la clase, pero son internos.

Se muestra a continuación un diagrama que representa el árbol de carpetas y archivos generados por DBSaver en *Dropbox*:

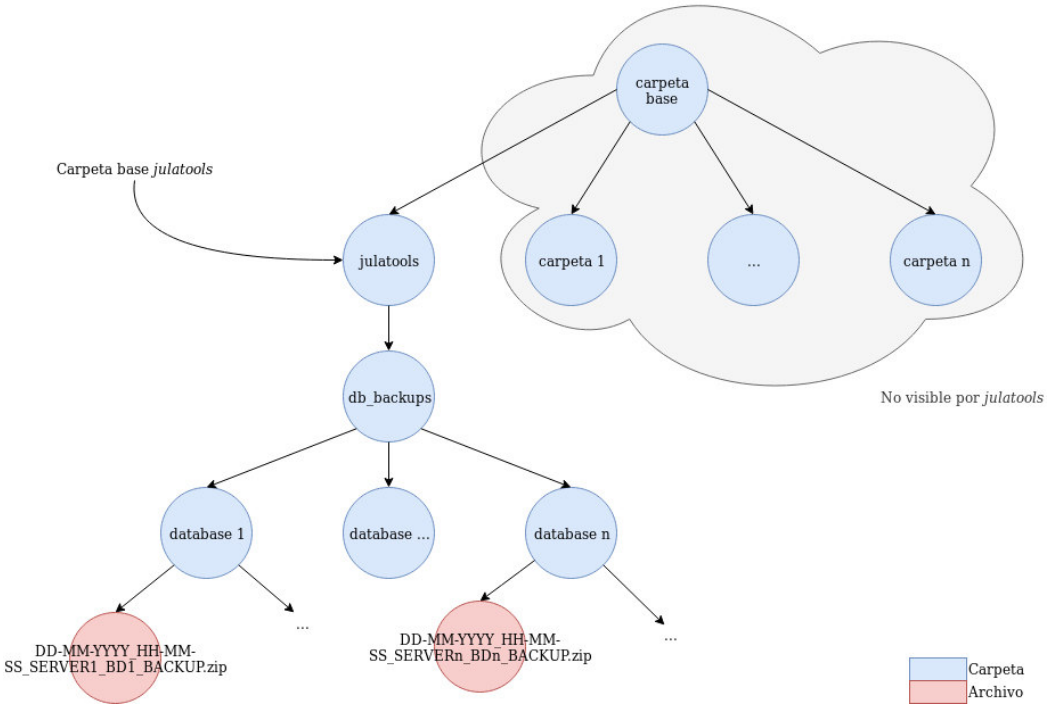


Figura 70: Estructura de archivos generada en Dropbox por DBSaver (imagen propia)

La siguiente figura presenta un diagrama de la interacción entre las diferentes clases que hacen jula_ckup:

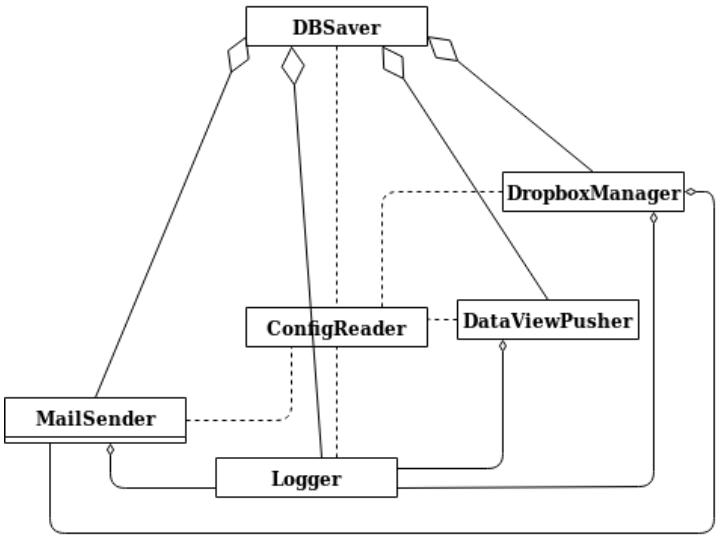


Figura 71: Interacción entre las diferentes clases en jula_ckup (imagen propia)

En caso de que ocurra un problema al realizar un *backup* de una base de datos, se envía un mail, como se muestra a continuación:

para mí

Ocurrió un problema al realizar un backup de la base de datos test_db_2.
Servidor: Prueba local
Ingreso: xxxx
Usuario: username
Contraseña: pw...

Figura 72: Ejemplo de email enviado (imagen propia)

2.4.1.a) 3. Jula_dsk

Jula_dsk es la herramienta que permite monitorear el estado de los discos (particiones, en realidad, pero se las denomina “discos” por fines prácticos). Se ejecuta y, así como

jula_ckup inicializa todos los objetos necesarios y delega el trabajo, jula_dsk hace lo mismo, ya que también posee una función *main*, y delega el trabajo a un método (check_disks), de una clase que encapsula el comportamiento de la aplicación (*DiskChecker*). Se continuó el desarrollo basándose en los mismos conceptos de la herramienta anterior. Se muestra a continuación el punto de entrada o función principal de jula_dsk:

```
def main(args):
    config_file = None
    if len(args) != 2:
        config_file = os.path.join(os.path.dirname(os.path.realpath(__file__)), "config_local.toml")
    else:
        config_file = args[1]
    cfg = ConfigReader(config_file)
    logger = Logger(cfg.read("server"), cfg.read("logger"))
    dataview_pusher = DataViewPusher(cfg.read("server"), cfg.read("dataview_pusher"), logger)
    mail_sender = MailSender(cfg.read("server"), cfg.read("mail_sender"), logger)
    disk_checker = DiskChecker(cfg.read("server"), cfg.read("disk_checker"), logger, mail_sender, dataview_pusher)
    disk_checker.check_disks()
    return 0
```

Figura 73: Punto de entrada de jula_dsk (imagen propia)

La siguiente figura muestra el diagrama UML de clases de DiskChecker:

Figura 74: Diagrama UML de DiskChecker (imagen propia)

Esta clase hace uso de una biblioteca de *Python* para la manipulación de los sistemas de archivos (*shutil*).

El método principal (check_disks) lleva a cabo las siguientes operaciones para cada disco o partición configurada:

1. medir el espacio total, espacio usado y porcentaje máximo permitido de uso;
2. determinar si el uso está por encima del límite;
3. formatear información del disco para *DataView*;
4. registrar estado actual (*Logger*);
5. en caso de ser necesario, notificar por email (MailSender);
6. enviar información a *DataView* (*DataViewPusher*).

En caso de cualquier error, se registra el problema (*Logger*).

Se muestra en la siguiente imagen la estructura de datos enviados a *DataView* por parte de jula_dsk:

```
DATAVIEW_DATA_STRUCTURE = ['Servidor', 'Disco', 'Tamaño', 'Uso', 'Último Check', 'Estado']
DATAVIEW_TYPE = 'DISK'
```

Figura 75: Datos enviados a DataView (imagen propia)

Este diagrama muestra la interacción de las clases que hacen a jula_dsk:

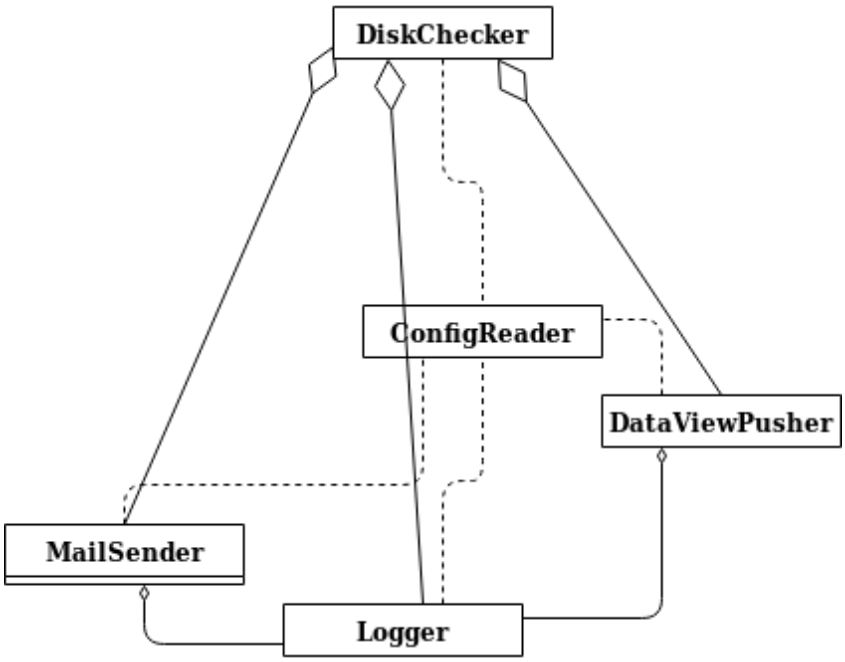


Figura 76: Interacción entre las diferentes clases en jula_dsk (imagen propia)

Se muestra un ejemplo del correo enviado por jula_dsk, ya que, en caso de haber algún problema, como se indicó, se envía un email:

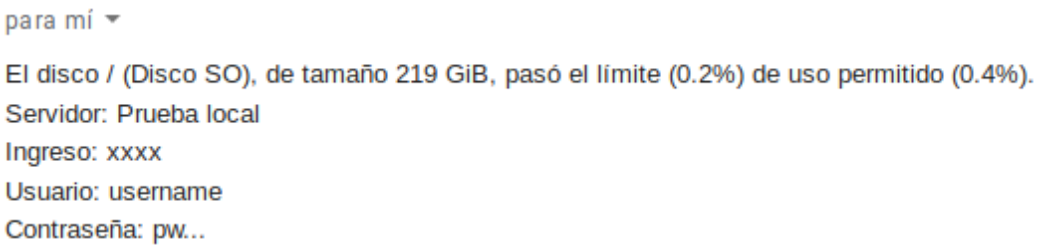


Figura 77: Ejemplo de email enviado (imagen propia)

2.4.1.a) 4. jula_logger

Luego de haber trabajado sobre las herramientas anteriores, al momento de comenzar a probar el sistema, se estableció que sería de gran utilidad poder ver en tiempo real qué es lo que ocurría con las herramientas en funcionamiento. Además, debido a que en los servidores corren varias aplicaciones en producción y es frecuente el acceso remoto a estos para revisar *logs* de estas aplicaciones, se pensó en poder llevar estos registros a la nube para su revisión, en caso de que, en alguna aplicación, ocurrieran fallas. Esto último fue descartado más adelante, y solamente se trabajó el log de julatools, por diversas razones.[19] Similar a las herramientas previas, jula_logger contiene el punto de entrada (función *main*) que inicializa los objetos y deja el flujo de ejecución al método upload_log de la clase *LogUploader*. Se omite la muestra del punto de entrada ya que es similar al de las demás herramientas. Se presenta el diagrama UML de clases de LogUploader (clase principal que encapsula la funcionalidad principal de jula_logger):

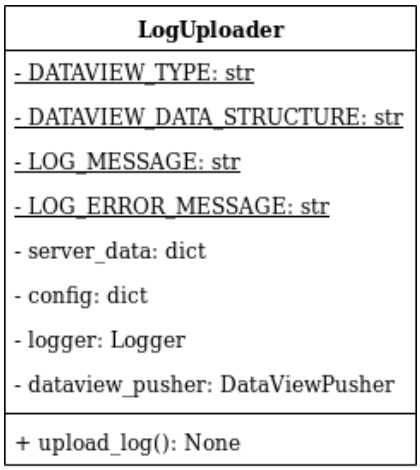


Figura 78: Diagrama UML de LogUploader (imagen propia)

[19] La principal razón es la existencia de diferentes formatos de los registros de aplicaciones, como, por ejemplo, *Apache* y *Tomcat*, entre otras. Cada archivo de *log* tiene un formato único y es extremadamente dificultoso procesarlos para determinar qué información contienen (fecha, si es un error o no, etc.), por lo que la pragmaticidad y eficacia del desarrollo global obligaron a dejar de lado la consideración de esta idea.

El método **log_upload** tiene un funcionamiento sencillo, ya que, en base al archivo de log de *julatools*, procesa línea por línea (hasta un máximo definido en su configuración), con la siguiente metodología:

1. abre y lee el archivo de configuración línea por línea;
2. en caso de leer una línea cuya entrada sea del tipo ERROR y su fecha sea posterior a la última semana, la marca para subir a *DataView*;
3. registra las acciones (*Logger*);
4. en caso de tener varias entradas marcadas para enviar, las envía a *DataView* (*DataViewPusher*);
5. en caso de no encontrar errores recientes, elimina de *DataView* los errores antiguos que estén subidos del servidor (*DataViewPusher*, método delete_old_data).

En cualquier caso de error durante su ejecución, el sistema registra el error (*Logger*). La imagen siguiente muestra la estructura de datos para el envío de información a *DataView*:

```
DATAVIEW_TYPE = 'LOG'
DATAVIEW_DATA_STRUCTURE = ['Servidor', 'Linea', 'Fecha de linea']
```

Figura 79: Estructura de datos para DataView (imagen propia)

A continuación, se muestra un diagrama que establece la interacción entre las diferentes clases de jula_logger:

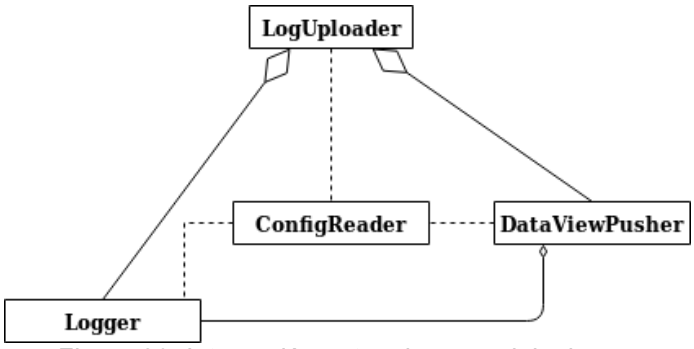


Figura 80: Interacción entre clases en jula_logger (imagen propia)

Por último, se muestra un ejemplo de un log, generado por las diferentes herramientas, y que puede ser procesado por jula_logger para el envío de errores a DataView:

```
2019-10-11 16:24:01,324 - Logger - INFO - DiskChecker: Disco / con tamaño 219 GiB; 42.0% de uso (limite 80.0%)
2019-10-11 16:24:01,393 - Logger - INFO - DataViewPusher: Datos del tipo DISK insertados en DataView
2019-10-19 15:10:37,504 - Logger - INFO - LogUploader: Se leyó información del log de DataView para subir errores
2019-10-19 15:10:37,511 - Logger - ERROR - DataViewPusher: Error insertando datos del tipo LOG en DataView: 1406 (22001): Data too long for column 'column2' at row 1
2019-10-19 15:11:48,747 - Logger - INFO - LogUploader: Se leyó información del log de DataView para subir errores
2019-10-19 19:40:12,259 - Logger - INFO - DBSaver: Backup de test_db realizado
2019-10-19 19:40:12,349 - Logger - INFO - DataViewPusher: Datos del tipo DB insertados en DataView
2019-10-19 19:40:14,199 - Logger - INFO - DropboxManager: Se subió el archivo /home/erik/julatools_files/dbs_backups/db_backups/test_db/19-10-2019_19-40-12_test_db_B
```

Figura 81: Ejemplo de log para jula_logger (imagen propia)

2.4.1.b) Creación de tablero e indicadores de relevancia en *DataView*

La actividad de delimitación de indicadores importantes para el área de operaciones se realizó de forma simultánea al diseño y desarrollo de las aplicaciones anteriores. Como se explicó previamente, se definieron tres tipos de datos:

- información referida a discos (‘DISK’);
- información correspondiente a copias de seguridad de bases de datos (‘DB’);
- información relacionada a logs de julatools (‘LOG’).

Se creó una tabla (denominada “jula_operaciones”) en la misma base de datos de *DataView* (la base de datos intermedia), para centralizar esta información, cuya estructura, mostrada a continuación, es similar a la existente:

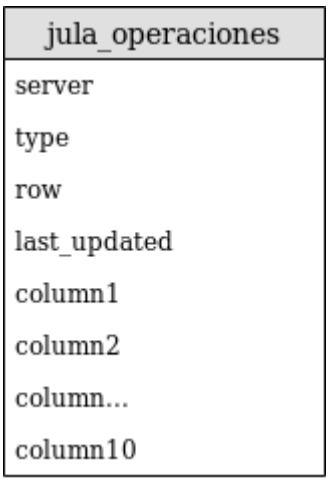


Figura 82: Tabla jula_operaciones (imagen propia)

Las primeras cuatro columnas son “metadatos” de cada línea insertada, para identificar el servidor y el tipo de dato de donde proviene el dato. En cada inserción, estos metadatos son brindados por la clase *DataViewPusher,* previamente desarrollada. Las demás columnas son dinámicas y permiten que se ajuste la información de forma dinámica en base a las diferentes aplicaciones que podrían enviar datos. En el caso de las herramientas desarrolladas, es posible ver cómo ordenan la información viendo su variable de clase *DATAVIEW_DATA_STRUCTURE*. Esto provee flexibilidad para ampliar el repertorio de usos a futuro.

Una vez que se diseñaron los indicadores y se colocó *Julatools* en algunos servidores, se crearon los *widgets* en el tablero de *DataView*, con una frecuencia de actualización de treinta o sesenta minutos; se muestra a continuación el *widget* para la visibilidad de los *backups* generados (jula_ckup):

1. Reporte de backups

SERVIDOR	BD	ARCHIVO	TAMAÑO	ÚLTIMA COPIA
proteo	proteo	02-11-2019_16-00-01_proteo_proteo_BACKUP.zip	6.25 MiB	1 día/s
dataview	dataview	02-11-2019_23-30-02_dataview_dataview_BACKUP.zip	0.02 MiB	1 día/s
scada-db	scada	02-11-2019_23-01-55_scada-db_scada_BACKUP.zip	12.17 MiB	1 día/s
alt-db-m1	alertatel2	02-11-2019_23-00-01_alt-db-m1_alertatel2_BACKUP.zip	206.63 MiB	1 día/s
julasoft-testing	dataview	02-11-2019_23-00-00_julasoft-testing_dataview_BACKUP.zip	0.03 MiB	1 día/s
scada-db	scada_hist	02-11-2019_23-05-51_scada-db_scada_hist_BACKUP.zip	50.81 MiB	1 día/s

Figura 83: Últimos backups realizados (imagen propia)

El *widget* mostrado en la imagen previa tiene asociado la siguiente consulta SQL:

```
select
    column1 'Servidor', column2 'BD', column4 'Archivo', column5 'Tamaño',
    concat(DATEDIFF(current_timestamp, column3), ' día/s') as 'Última copia'
from dataview.jula_operaciones
where type = 'DB'    and row <> 0
order by row
```

Figura 84: Consulta SQL de widget (imagen propia)

Se presenta a continuación cómo se encuentra almacenada la información en la base de datos recuperada por el *widget*, mediante la consulta configurada:

	server	type	row	last_updated	column1	column2	column3	column4	column5
1	alt-db-m1	DB	0	2019-11-01 22:58:22	Servidor	BD	Última copia	Archivo	Tamaño
2	alt-db-m1	DB	1	2019-11-01 22:58:23	alt-db-m1	alertatel2	2019-11-01 23:00:02.007328	01-11-2019_23-00-02_	206.44 MiB
3	dataview	DB	0	2019-11-02 13:41:28	Servidor	BD	Última copia	Archivo	Tamaño
4	dataview	DB	1	2019-11-02 13:41:28	dataview	dataview	2019-11-02 13:45:45.298894	02-11-2019_13-45-45_	0.02 MiB
5	julasoft-testing	DB	0	2019-11-01 23:00:01	Servidor	BD	Última copia	Archivo	Tamaño
6	julasoft-testing	DB	1	2019-11-01 23:00:01	julasoft-testing	dataview	2019-11-01 23:00:00.761697	01-11-2019_23-00-00_j	0.03 MiB
7	proteo	DB	0	2019-11-01 15:55:47	Servidor	BD	Última copia	Archivo	Tamaño
8	proteo	DB	1	2019-11-01 15:55:47	proteo	proteo	2019-11-01 16:00:01.879033	01-11-2019_16-00-01_j	6.25 MiB
9	scada-db	DB	0	2019-11-01 23:48:29	Servidor	BD	Última copia	Archivo	Tamaño
10	scada-db	DB	1	2019-11-01 23:48:29	scada-db	scada	2019-11-01 23:00:49.833124	01-11-2019_23-00-49_s	12.07 MiB
11	scada-db	DB	2	2019-11-01 23:48:29	scada-db	scada_hist	2019-11-01 23:05:32.732137	01-11-2019_23-05-32_s	50.81 MiB

Figura 85: Datos de backups (imagen propia)

La siguiente figura muestra el *widget* correspondiente a la visualización de los últimos errores generados por las herramientas en los servidores (jula_logger):

3. Últimos errores

SERVIDOR	FECHA DE ERROR	LINEA
dataview	01/11/2019 23:30	2019-11-01 23:30:02,624 - Logger - ERROR - DropboxManager: Error eliminando archivo /db_backups/dataview/30-10-2019_23-30-02_dataview_dataview_BACKUP.zip de Dropbox: HTTPSConnectionPool(host='api.drop
alt-db-m1	01/11/2019 23:02	2019-11-01 23:02:42,973 - Logger - ERROR - DropboxManager: Error eliminando archivo /db_backups/alertatel2/29-10-2019_23-00-02_alt-db-m1_alertatel2_BACKUP.zip de Dropbox: HTTPSConnectionPool(host='api
julasoft-testing	01/11/2019 23:00	2019-11-01 23:00:01,323 - Logger - ERROR - DropboxManager: Error eliminando archivo /db_backups/dataview/30-10-2019_23-00-01_julasoft-testing_dataview_BACKUP.zip de Dropbox: HTTPSConnectionPool(host='

Figura 86: Últimos errores (imagen propia)

A continuación, se puede observar el *widget* correspondiente a los reportes de discos (jula_dsk):

2. Reporte de discos

SERVIDOR	DISCO	TAMAÑO	ESPACIO LIBRE	ÚLTIMO CHECK	ESTADO
alt-db-m1	C:	99 GiB	20.8 GiB	0 día/s	OK
dataview	/	7 GiB	2.9 GiB	0 día/s	OK
julasoft-testing	C:	29 GiB	3.5 GiB	0 día/s	OK
julasoft-testing	D:	79 GiB	5.5 GiB	0 día/s	OK

Figura 87: Reporte de discos (imagen propia)

Se pueden observar algunas diferencias con respecto a la estructura visualizada en el *widget* de la imagen anterior y la estructura de los datos enviados a *DataView* por parte de jula_dsk.[20]

En la figura anterior, las columnas de espacio libre y de último *check* fueron generadas del lado de la consulta de *DataView,* ya que la información que se envía no tiene este formato. Este fue uno de los puntos que se fueron modificando a medida que se implementó y se comenzó a utilizar el tablero.

Estos son los indicadores que actualmente se encuentran en funcionamiento, es decir, el tablero del área de operaciones posee tres *widgets*.

2.4.1.c) Despliegue en servidores

En la medida en que se fue progresando con el desarrollo de las herramientas, no en el estado que se muestra actualmente, se comenzaron a colocar las herramientas en algunos servidores, por un lado, de pruebas y, por otro, productivas. Como se explicó al principio, fue de nuestro interés plantear una solución generalizada tanto para los servidores con *Windows* Server, como para aquellos con GNU/Linux, a pesar de que, actualmente, prima el primero. Para cada servidor, se configuró *Julatools* según las características del sistema:

- En *Windows Server*, se trabajó con la aplicación de sistema *Task Scheduler*.

[20] Las columnas de espacio libre y de último *check* fueron generadas del lado de la consulta de *DataView*, ya que la información que se envía no tiene este formato. Este fue uno de los puntos que se fueron modificando a medida que se implementó y comenzó a utilizarse el tablero.

- En GNU/Linux, se trabajó con *cron*, mediante la edición del archivo crontab, con el comando crontab -e.

Ya que mediante *Git* se gestionan los códigos fuentes, se lo utilizó, además, para colocar las versiones funcionales de *Julatools* en los diferentes servidores.

Asimismo, en algunos casos, se configuraron las variables de sistema (para el inicio de la creación de tablero e indicadores de relevancia en DataViewPython, mysqldump, etc.) y se instalaron las dependencias requeridas para el uso de *Julatools*.

Se pueden resumir las consideraciones tomadas y acciones realizadas en la siguiente lista:

- comprobar que *Python* 3 esté instalado;
- comprobar que *Git* esté instalado;
- comprobar que las variables de sistema para *mysqldump* y *python* estén configuradas;
- clonar el repositorio de código fuente de *julatools* mediante *Git*;
- instalar dependencias de *Python* (que se encuentran en un archivo de los códigos fuentes) mediante el uso de pip (herramienta por defecto para gestionar dependencias de *Python*);
- crear un archivo de configuración para *julatools* dentro del servidor (tomando como plantilla un archivo que se encuentra junto al código fuente);
- configurar los permisos del usuario que accede a la base de datos local para realizar copias de seguridad, tanto para la base de datos como la tabla mysql.proc que contiene procedimientos y funciones de cada base de datos;
- ejecutar las herramientas para probar su funcionamiento.

Una vez que se aseguró en cada servidor que se hayan tomado todas las acciones requeridas, se configuraron las herramientas para dejarlas en funcionamiento.

Para iniciar cada herramienta, se debe colocar el intérprete de *Python* e indicar por parámetro tanto la ubicación de la herramienta (sea jula_ckup, jula_logger o jula_dsk), como la ubicación del archivo de configuración generado, por ejemplo:

```
python /home/user/julatools/jula_ckup.py /home/user/config_local.toml
```

Se muestra a continuación la configuración de las herramientas en un servidor con GNU/Linux, mediante crontab.

```
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command

30 23 * * * /usr/bin/python3 /home/ubuntu/julatools/jula_ckup.py /home/ubuntu/julatools/config_local.toml
00 * * * * /usr/bin/python3 /home/ubuntu/julatools/jula_dsk.py /home/ubuntu/julatools/config_local.toml
0,30 * * * * /usr/bin/python3 /home/ubuntu/julatools/jula_logger.py /home/ubuntu/julatools/config_local.toml
```

Figura 88: Configuración de julatools en GNU/Linux (imagen propia)

En el caso de cron, solo admite rutas absolutas, por lo que no puede indicarse solamente el nombre del intérprete de *Python* (sea python, python3, py, etc.).

En la siguiente imagen, se puede observar la configuración de las herramientas en un servidor con *Windows Server* mediante *Task Scheduler* (el planificador de tareas):

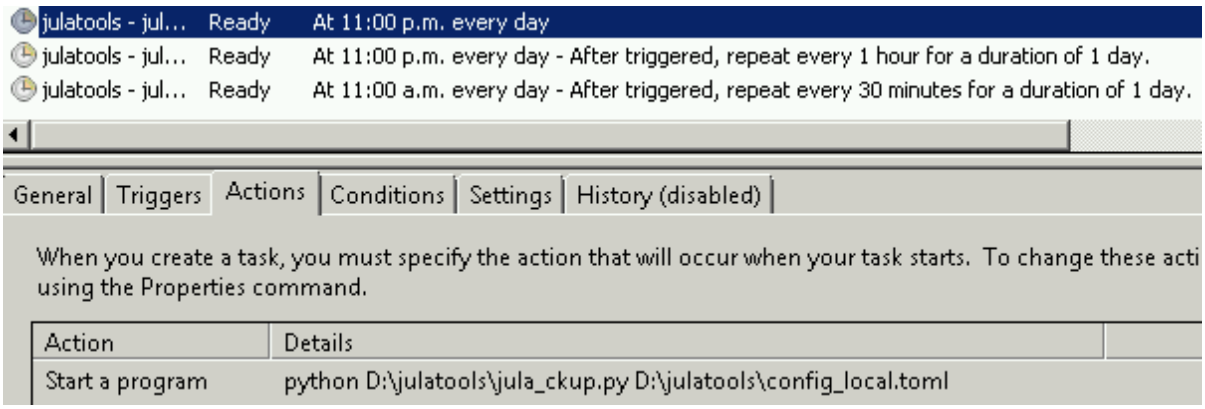


Figura 89: Configuración de julatools en Windows Server (imagen propia)

En relación con la frecuencia de ejecución, para el caso de las copias de seguridad, se asignó la realización de esta tarea una vez el día; en el caso de los discos, cada hora (pero, puesto que el crecimiento es lento, alcanzaría con asignar una frecuencia diaria también); en el caso de los registros, se aumentó la frecuencia en comparación con los anteriores, para que los *logs* se suban más rápido al tablero, en caso de haber errores recientes.
Esto se realizó para algunos servidores y, si bien, en algunos, la ejecución ocurrió sin ningún inconveniente, en otros, hubo que dedicar más tiempo a esta tarea, por problemas de configuración (falta de bibliotecas, falta de permisos de usuarios de sistema para el acceso a recursos, versiones antiguas de *Python*, etc.). De todas maneras, mediante la visualización del tablero, se logró determinar de forma rápida los problemas presentes en los servidores donde la aplicación fallaba, o funcionaba a veces y fallaba, en otras, además de que tener información relevante en tiempo real permitió realizar un seguimiento constante sobre el despliegue en los servidores. Los resultados pueden verse en la sección anterior, donde se presentaron datos de los tableros que son de servidores en producción.

2.4.2 Cierre

Esta actividad permitió involucrarse de cerca en las tareas operacionales llevadas a cabo dentro del área de operaciones de la organización, específicamente, en relación al trabajo

que implica el mantenimiento de los servidores y las diferentes aplicaciones que funcionan en estos.

La implementación efectiva de esto ayudó a detectar algunos problemas en servidores y permitió tomar acciones. Por ejemplo, en el caso del único servidor de GNU/Linux (que no estaba siendo revisado ni monitoreado previamente), se notaron tres cuestiones: en primer lugar, la falta de espacio libre del disco principal (100 MiB, tema que fue resuelto mediante la limpieza del sistema -actualizaciones antiguas y *logs* de aplicaciones-) y, en segundo lugar, la existencia de un disco sin uso[21]; finalmente, también se observó y solucionó en otros servidores una asignación de espacio poco adecuada.

Otro resultado que se logró con esto fue el poder determinar un conjunto de elementos que pueden ser reutilizables para conseguir nuevas funcionalidades. Por ejemplo, podría automatizarse una tarea llevada a cabo por el área de operaciones en relación a la gestión de códigos fuentes de proyectos. Es viable diseñar una herramienta que, cada cierto tiempo, realice copias de los códigos fuentes de varios proyectos en progreso (gestionados mediante *Git*) y, así, resguardar copias de seguridad; lo único que debería ser desarrollado es un módulo que pueda interactuar con repositorios *Git*, ya que todo lo demás se puede reutilizar.[22]

2.4.3 Mejoras posteriores a julatools

A medida que se avanzó con las actividades finales de la práctica, se adicionaron cambios al conjunto de herramientas desarrolladas. Estos cambios y mejoras se aplicaron por necesidades surgidas por el uso de la aplicación y por apreciaciones propias (es decir, por

[21] Cada recurso no aprovechado dentro de los servidores impacta en los costos de mantenimiento de la organización; por eso, se buscó optimizar estas cuestiones o buscar puntos de desperdicio de recursos de infraestructura.

[22] Si bien los repositorios de *Git* son confiables en relación con la perduración de datos, podría realizarse un módulo/herramienta que permita almacenar versiones o hitos de ciertos proyectos.

observaciones realizadas sobre las herramientas y su funcionamiento). Entre estas incorporaciones, pueden nombrarse:

- refactorización y simplificación de algunas herramientas: esto se refiere a la revisión de varias partes de las aplicaciones, junto a la inclusión de pequeños cambios que permiten hacerlas más útiles (por ejemplo, configurar el envío de notificaciones por email o información a DataView por herramienta, no de forma global, algo que inicialmente se consideró innecesario);
- 'empaquetización' de las herramientas: gracias a la incorporación de una biblioteca de *Python* llamada *setuptools*, se encapsularon todas las herramientas en un único paquete que puede ser instalado en un servidor y utilizado como una aplicación de consola, como se observa en la siguiente imagen:

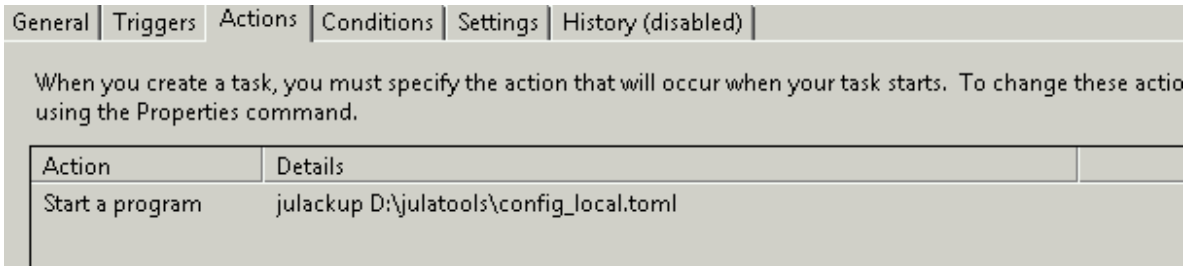


Figura 90: Nueva forma de utilizar las herramientas (imagen propia)

Se verifica que se inicia la herramienta de la misma forma que se inician aplicaciones de consola. Esto, además de facilitar el despliegue y uso de las herramientas, simplifica el procedimiento para instalarla, ya que, en lugar de utilizar el código fuente distribuido mediante el repositorio de *Git*, se distribuye un paquete que se instala mediante el manejador de paquetes de Python (*pip*)

Además, se incorporó una herramienta, llamada **jula_dbx**, siguiendo los mismos lineamientos de las demás herramientas, para poder visualizar el espacio disponible en *Dropbox*, que es utilizado para el resguardo de información.

Por otra parte, se agregaron cambios al tablero, para permitir reflejar mejor la información que este posee:

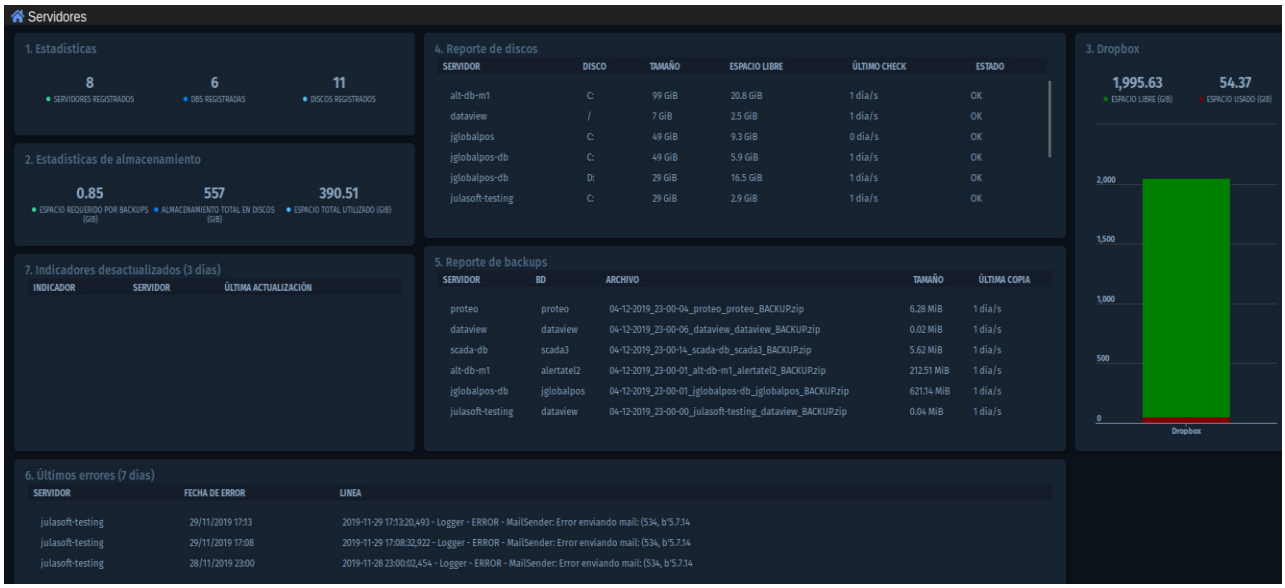


Figura 91: Nuevos cambios del tablero de operaciones/servidores (figura propia)

Se observa en la figura anterior un mejor ordenamiento de la información; todos estos cambios y mejoras permiten observar la evolución incremental del conjunto de herramientas desarrolladas. Uno de los objetivos iniciales del diseño llevado a cabo era poder contemplar la posibilidad de generar e incorporar mejoras y cambios de forma iterativa.

2.5 Actividades asociadas al cierre de la práctica

A continuación, se explican las actividades que incluyen el cierre de la práctica y el desarrollo de informes.

2.5.1 Desarrollo

El desarrollo del cierre de la práctica puede desglosarse de la siguiente manera:

- cierre, revisión y evaluación de resultados;
- escritura de informes de la práctica y sus resultados.

Dentro de esta actividad se realizó la escritura de los informes de avance y final asociados a la práctica. Durante el desarrollo, se desplegaron las conclusiones generales que abordan toda la práctica, haciendo hincapié en la experiencia del ejercicio laboral y en reflexiones más globales, junto a mejoras y modificaciones que surgieron del uso continuo del desarrollo realizado (como en la sección 2.4.3, donde se muestran cambios incorporados a las herramientas desarrolladas).
No se incluye el cierre de esta actividad porque corresponde al cierre de la práctica en su totalidad, y esto se encuentra en las conclusiones y reflexiones sobre la práctica.

3 Conclusiones

3.1. Conclusiones generales

En primer lugar, se debe hacer algunos comentarios sobre las diferencias entre los planes originales y su ejecución. Debido a la naturaleza del ejercicio, se han encontrado diferentes obstáculos, impedimentos y cuestiones a resolver que generaron el análisis y la toma de decisiones con respecto a cómo proceder, y han tenido como resultado el cambio o desvío de algunos objetivos y actividades realizadas. De todas formas, se mantuvo el plan original que incluía los objetivos principales de la práctica. Estos inconvenientes pueden notarse especialmente durante la actividad 2.3 (dedicada a las mejoras aplicadas al proceso de integración entre *LK-ERP* y *DataView*), ya que, como se explicó, se encontraron diferentes inconvenientes que diluyeron la viabilidad de los planes originales. De forma similar ocurrió con otras actividades, por ejemplo, en el caso del desarrollo de **julatools**, ya que, a medida que se avanzaba, se realizaron algunas modificaciones o cambios, algunos de los cuales fueron descriptos dentro de la sección de desarrollo de la actividad.

Por el lado de la primera actividad principal, el proceso de integración entre *LK-ERP* y *DataView*, se pueden desprender varias conclusiones, tanto en relación a aspectos técnicos como a cuestiones no técnicas.

En primer lugar, se debe dar un lugar de ponderación a la labor que implicó el análisis de la estructura de los datos del sistema ERP. Esto es, en parte, debido al ERP y por las costumbres y modo de uso del sistema por parte del cliente (carga de datos del cliente, con sus errores). En relación a las características sistema, debe partirse de la base de que es un sistema que, debido a que a lo largo de los años ha sido sometido a diferentes modificaciones (en varios casos no planificadas, especialmente, dentro de la base de datos mediante el uso de procedimientos almacenados y funciones) y ha provocado un crecimiento en su complejidad, es difícil de medir y, claramente, provoca una carga adicional a la hora de trabajar con él, ya que el usuario debe tener mucho cuidado con lo que altera, puesto que hay muchos factores no observables a simple vista que pueden causar resultados no esperados. De todas maneras, esto es entendible, ya que es propio de la naturaleza de los sistemas de gestión organizacionales, que deben ser adaptables y extensibles a las necesidades que van surgiendo a lo largo del tiempo (y acompañan a la evolución y transformaciones que ocurren en todas las empresas). Como mencionamos, otra cuestión que incidió adicionalmente en relación al análisis de los datos, es la forma en la que el cliente hace uso del sistema. Todo sistema implementado condiciona la forma en la que la organización genera, comunica y procesa información. En varias oportunidades, a la hora de analizar los datos, se encontraron varios problemas, como valores no cargados (pesos de los artículos de producción, artículos asignados a una categoría incorrecta, entre otras cuestiones), que motivaron a comunicar a la empresa la necesidad de depurar y refinar los datos manejados, ya que esto es necesario para que la información extraída sea confiable.

En consonancia con esto último, se demuestran las ventajas de contar con un sistema que permite visualizar información resumida y consistente en tiempo real. En el caso de este

cliente en particular, al momento de su utilización, se encontraron diferencias entre la información reflejada por el sistema y los reportes generados manualmente para la toma de decisiones (es decir, aquellos utilizados por la dirección de la compañía). Esto provocó discusiones por parte de los miembros de la gerencia en torno a cómo se llegaba a obtener tales números y valores, lo cual obligó a realizar modificaciones en la forma de carga de datos, para unificar criterios sobre la interpretación de información, entre otras medidas que ayudan a la organización en la mejora de los procesos que hacen a la gestión de la información. Esto resulta muy positivo, ya que la empresa modifica en cierta medida su cultura organizacional sin la implantación de un nuevo sistema o de nuevos procesos de trabajo, de manera que se continúa trabajando sobre lo que se estaba utilizando previamente (por ejemplo, el mismo sistema ERP o planillas de cálculo).

Por otra parte, deben retomarse conclusiones iniciadas en la sección correspondiente al cierre de la segunda actividad principal, el aporte en la mejora de monitoreo y gestión de servidores.

Uno de los aspectos que más tiempo ocupó fue el trabajo que implica el despliegue de las herramientas generadas en los diferentes servidores y su posterior configuración. Debido a la heterogeneidad de los servidores (en relación a sus prestaciones, versiones de *Windows Server* y aplicaciones instaladas previamente), se encontraron diferentes inconvenientes para lograr implementar **julatools** en cinco servidores diferentes. Entre los problemas generados, surgieron los siguientes:

- manejo de diferentes versiones de *Python* e instalar dependencias;
- configuración de variables de entorno del sistema;
- imposibilidad de acceso a recursos externos para usuarios (según la versión de *Windows Server*);
- limitaciones para configurar tareas a determinados usuarios dentro de *Task Scheduler* (especialmente en el caso de usar una cuenta de administrador).

Además, luego de la implantación, se tardó bastante tiempo en lograr reducir la cantidad de fallas generadas por la ejecución de *julatools* en los servidores, ya que se encontraron errores como problemas para el envío de emails, problemas recurrentes para subir archivos a *Dropbox*, problemas con *Task Scheduler*, problemas con el horario de los servidores, entre otras asperezas que se fueron puliendo a medida que se avanzó con la revisión constante de la puesta en marcha de las herramientas. En la actualidad, la cantidad de errores o inconvenientes es muy reducida.

Este pasaje de la fase de pruebas hasta la fase de producción (es decir, del uso operativo de las herramientas sobre los servidores), que ocupó una porción considerable de esta actividad, quedó en un segundo plano frente a los resultados mostrados, pero, de todas maneras, amerita ser mencionado ya que ha formado parte del desarrollo durante dicha actividad.

La gran cantidad de tiempo asignado a esto último puede tener relación con la dificultad para realizar el *testing* de integración. Se considera como “buenas prácticas” el uso de pruebas unitarias (sobre módulos aislados de *software*) y de pruebas de integración (sobre varios módulos o componentes integrados). Debido a que es propio de la naturaleza de las herramientas desarrolladas (su función es, principalmente, interactuar con sistemas de archivos, bases de datos, otras aplicaciones, APIs web, etc.), prácticamente, no hay funcionalidad que se pueda probar de forma aislada, y las pruebas serían esencialmente de integración. Con respecto a las pruebas de integración, se tienen dos enfoques:

- probar sobre componentes reales (contra bases de datos, contra las *APIs* web, etc.);
- probar sobre *mockups* de los componentes (maquetas que emulan las interfaces definidas por los componentes).

Probar sobre componentes reales resulta una tarea tediosa puesto que se deben construir los *tests* sobre los componentes externos, y probar sobre *mockups* tiene dos desventajas: deben utilizarse bibliotecas y herramientas que ayuden a modelar los componentes (añadiendo una complejidad adicional), y, además, no aseguran que la interacción entre el

módulo desarrollado y los componentes externos sean correctas, ya que los resultados dependen del grado de precisión del mockup diseñado; por estas razones, el *testing* no fue abordado de forma amplia desde el inicio del desarrollo.

3.2 Reflexión sobre la práctica profesional supervisada como espacio de formación

En la sección anterior, se desarrollaron las conclusiones generales de las actividades realizadas, haciendo foco en los aspectos técnicos y, adicionalmente, en algunas otras problemáticas que surgieron a lo largo de la PPS (problemas encontrados, interacción con el cliente, etc.). En esta sección, se pretende destacar consideraciones propias sobre el aporte de la práctica profesional supervisada dentro de la formación como estudiante.

Dentro de la práctica, se han podido llevar a cabo diferentes tareas de naturaleza variada (desde configuración, integración de aplicaciones, análisis funcional, interacción con clientes y usuarios, etc.). Se considera enriquecedor el involucramiento dentro de estas actividades en lo que respecta al aprendizaje dentro de la carrera. Por ejemplo, algunas de las cuestiones a destacar son el haber puesto en producción desarrollos propios, el haber logrado entregar de forma satisfactoria resultados a usuarios finales, la participación en procesos de forma transversal y la incorporación e integración a un espacio de trabajo. Otro resultado muy positivo es el haber tenido habilitado un alto grado de autonomía al momento de llevar a cabo los lineamientos de las actividades, el diseño de los desarrollos, así como también la posibilidad de proponer y aportar mejoras y cambios dentro de toda la práctica.

En cuanto al uso de los conocimientos técnicos, se considera que se han aprovechado gran parte de los adquiridos dentro de las diferentes materias, ya que se trabajó sobre bases de datos, programación, integración con sistemas web, análisis de datos, entre otros; lo realizado dentro de esta práctica complementa la formación académica y, además de ayudar a consolidarla, ha permitido observar un panorama sobre la industria del software y la informática y ser parte del proceso en dicha industria.

4. Anexos

4.1 Anexo 1

4.1.a *LK-ERP*

El sistema *ERP* es un sistema de gestión de recursos organizacionales. Generalmente, un sistema *ERP* provee módulos variados como compras, ventas, finanzas, logística, gestión de recursos humanos, herramientas de análisis y estadísticas de datos, etc.

En el caso de la empresa, su sistema *ERP* está compuesto por los siguientes módulos:

- una aplicación de escritorio, utilizada por el cliente;
- un servidor de base de datos, específicamente, *SQL Server 2000*, accesible localmente.

Es un sistema orientado a dar soluciones generalizadas; se ofrece como un paquete de software *off-the-shelf* (es decir, enlatado, lo cual implica que, por defecto, no permite su extensión y personalización más allá de las previstas originalmente). Para su funcionamiento, se instala localmente en el ambiente de la organización cliente.

SQL Server 2000 actualmente no tiene soporte, y para trabajar sobre él, deben tenerse en cuenta algunas cuestiones:

- tiene tipos de datos limitados comparados con sistemas de bases de datos modernos; esta limitación puede llegar a dificultar la expresividad de la información (por ejemplo, la representación de fechas);
- el dialecto de *SQL* ha sido modificado y extendido en versiones posteriores;
- trabaja con una única configuración regional (EE.UU.), y no es alterable, lo cual puede generar conflictos con los datos intercambiados con aplicaciones externas;
- no es instalable en versiones relativamente nuevas de *Windows* (solo hasta Windows 7).

Las siguientes imágenes muestran el menú principal del *ERP* y el menú de gestión de artículos, respectivamente, para ejemplificar parte de la funcionalidad del sistema:

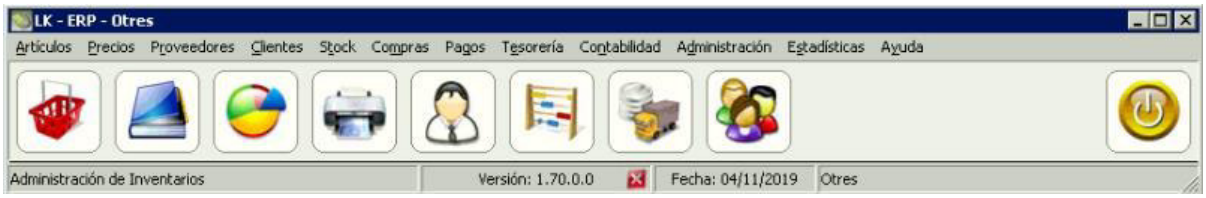


Figura 92: Menú principal del ERP (imagen propia)

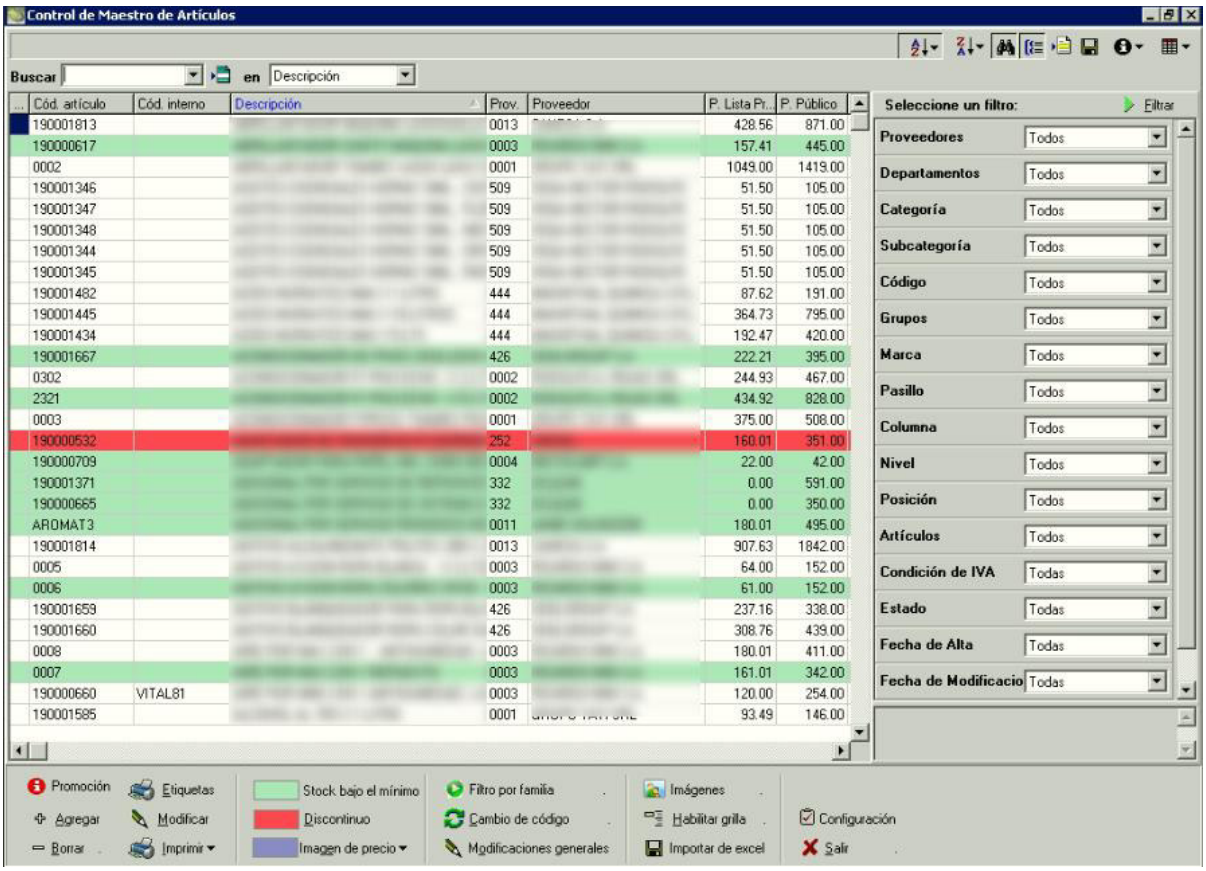


Figura 93: Gestión de artículos (imagen propia)

4.1.b Inconvenientes durante la extracción de datos de LK-ERP

Como se mencionó previamente, un problema fundamental de *SQL Server 2000* es que tiene un único tipo de dato para representar fechas (por ejemplo, el instante de tiempo '22/12/2019 12:00:00' no es lo mismo que '22/12/2019', pero su representación es la misma

en *SQL Server 2000*). Es decir, no se puede transformar una fecha (día y hora, tipo de dato *DATETIME*) a un tipo de dato del tipo día (ej. tipo de dato *DATE*). Esto sí es una posibilidad existente en versiones más nuevas. Ya que en un mismo día se realizan varias órdenes de compra para cargar, y se realizan, asimismo, en diferentes momentos del día, es necesario filtrar la parte de la hora para agrupar las órdenes de compra y generar una solución *ad-hoc*, ya que *SQL Server* no provee un tipo de dato para representar día. Para superar este inconveniente, se realizó lo siguiente:

- para cada orden de compra, se convierte la fecha y hora (DD/MM/YYYY HH:MM:SS) a una cadena de caracteres que solo contiene la parte del día (DD/MM/YYYY), es decir, el nuevo tipo de dato es VARCHAR;
- para ordenar los datos (ya que no se puede ordenar datos del tipo VARCHAR), se volvió a convertir la cadena de caracteres al tipo DATETIME; de esta forma, se pudieron ordenar los datos de cada día al momento de realizar la extracción.

Otro problema encontrado fue el tratamiento de los campos con datos del tipo números con coma flotante y la forma en la que *SheetL* vuelca los datos en la planilla de *Google Sheets*. Todo dato volcado en estas planillas no se vuelca como número, sino como una cadena de textos, por lo cual, no puede ser utilizado para operaciones matemáticas, es decir, no se pueden sumar o multiplicar estas celdas insertadas; por lo tanto, primero deben ser convertidas en un dato matemático. Si bien no fue un problema mayor, implicó un inconveniente adicional al momento de ejecutar estas actividades. De todas formas, la solución es simple, ya que *Sheets* permite realizar una conversión.

Un problema adicional de *SQL Server 2000* es, como se describe en su apartado, el problema con la configuración regional. Dependiendo de la región (Estados Unidos o Argentina, por ejemplo), el separador decimal de un número puede ser una coma (,) o un punto (.). Esta versión de *SQL Server* no permite cambiar la configuración regional, por lo que puede traer problemas para interpretar los datos obtenidos. Se acentuó el problema al

momento de extraer los datos en herramientas externas, como *Google Sheets*. La solución implementada fue convertir los datos antes de enviarlos, y pasarlos a cadenas de texto, además de modificar el punto por la coma. Esto no afectó al proceso de conversión de los datos dentro de *Google Sheets,* ya que, de todas formas, la herramienta *SheetL* los coloca como cadenas de texto.

4.2 Anexo 2

4.2.a *DataView*

DataView es una aplicación web (incluso, accesible por dispositivos móviles), que permite la visualización de indicadores, métricas, gráficos, etc. de información mediante *widgets* (elementos gráficos), distribuidos por tableros. Esta aplicación brinda la posibilidad de representar diferentes tipos de información a organizaciones, y agruparlas por áreas, además de incluir restricciones en la vista de información, según el nivel de acceso de los usuarios. Todos los gráficos e indicadores son personalizables y adaptables a los requerimientos y naturaleza de la información a representar. La siguiente imagen muestra el menú principal de *DataView*:

Figura 94: Menú principal de DataView (imagen propia)

El sistema permite extraer información de varias fuentes de información, tales como los siguientes, entre otros:

3. bases de datos relacionales (mediante *SQL*);
4. planillas de *Excel*;
5. planillas de *Google Sheets*;
6. archivos con formatos específicos;
7. recursos web.

Tiene una base de datos propia, que no contiene información que se utiliza en los gráficos, sino que se utiliza para cuestiones de autorización, registros, etc.

Para la extracción de datos y elección, configuración y visualización de gráficos o indicadores, utiliza una herramienta externa, llamada *Cyfe*, que se integra de forma embebida en la aplicación principal.

4.3 Anexo 3

4.3.a *Cyfe*

Cyfe es una aplicación integrada a *DataView*, que se encarga de los datos y los gráficos, su configuración y 'renderizado'. Ofrece, en su modelo de negocio, la posibilidad de re-utilizar su funcionalidad de forma embebida en aplicaciones externas, como es el caso de *DataView*. *Cyfe* es la aplicación que posee la lógica y funcionalidad de ejecutar consultas *SQL*. Se muestra a continuación un tablero de *Cyfe*:

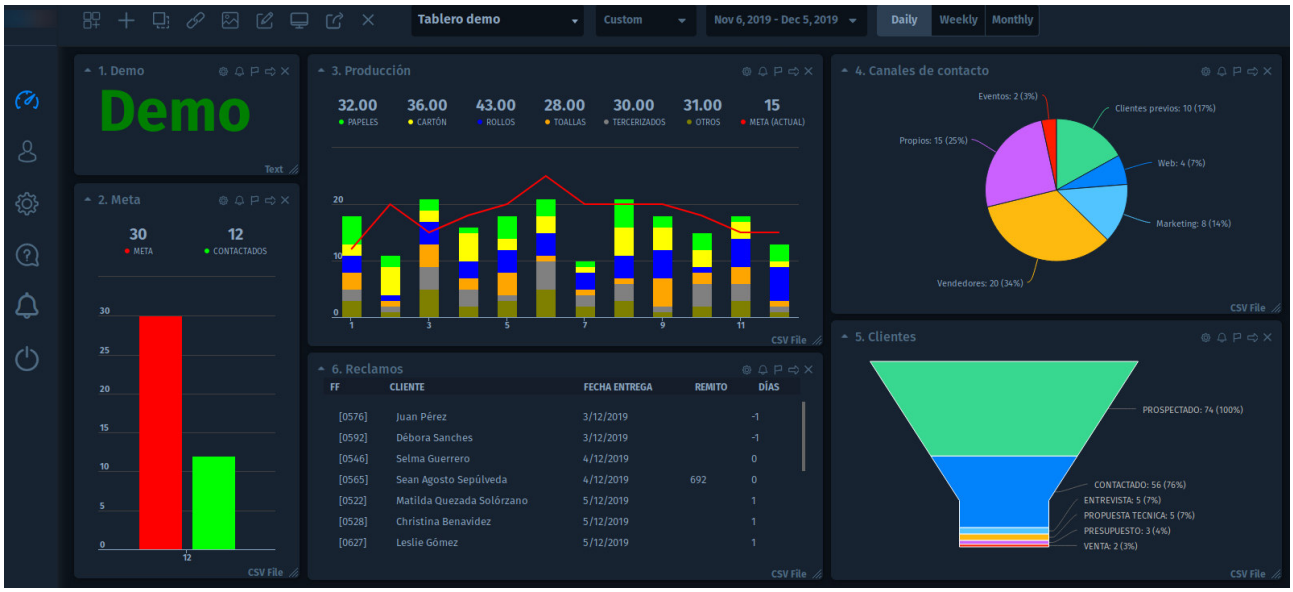


Figura 95: Tablero de Cyfe (imagen propia)

4.4 Anexo 4

4.4.a *Backups* e incovenientes asociados al diseño de jula_ckup

En relación con la forma en la que se realizan los *backups* de cada base de datos, se encontraron algunos impedimentos. Originalmente, se planteó no utilizar herramientas externas existentes en el sistema operativo, pero se investigó y no se encontró un método estandarizado, por ejemplo, mediante lenguaje *SQL*, para bases de datos *MySQL*, el sistema de bases de datos utilizado por todas las aplicaciones. *mysqldump* viene por defecto en todas las instalaciones de *MySQL*, por lo que es posible utilizarla. Para evitar el problema de no tener que especificar la ruta absoluta de la herramienta, que varía según servidor, sistema operativo y versión de *MySQL*, se dio uso a las variables de sistema, lo cual permite iniciar *mysqldump* solamente indicando su nombre. Se muestra a continuación el método interno de jula_ckup, contenido por la clase DBSaver, que encapsula el uso *mysqldump* (la herramienta externa):

```
def _make_backup(self, backup_full_path, database_name):
    with open(backup_full_path, 'wb+') as backup_file:
        mysql_command = subprocess.Popen(
            ['mysqldump',
             '-h', self.config["host"],
             '--port', self.config["port"],
             '-u', self.config["user"],
             '-p{}'.format(self.config["password"]),
             '{}'.format(database_name),
             '--routines',
             '--events'], stdout=backup_file, shell=False)
        stream_data = mysql_command.communicate()[0]
        return_code = mysql_command.returncode
        if return_code != 0:
            raise Exception("mysqldump finalizó con error")
    return os.path.getsize(backup_full_path)
```

Figura 96: Método interno de jula_ckup (imagen propia)

Se considera que la información relevante de la base de datos se compone del siguiente listado:

- estructura de tablas

- datos
- vistas (tablas virtuales)
- eventos
- funciones y procedimientos almacenados
- disparadores

En el caso de las funciones y procedimientos almacenados, estos no se encuentran definidos dentro de la base de datos en sí, sino en una base de datos propia del servidor de base de datos. Esto es una característica propia de MySQL.

Para el uso de *mysqldump* sobre una base de datos en particular, se tienen que tener los siguientes permisos sobre cada base de datos a resguardar:

- SELECT
- LOCK TABLES
- SHOW VIEW
- TRIGGER
- EVENT

Además, en el caso de que se quieran resguardar procedimientos y funciones de esa base de datos, se necesita el permiso SELECT sobre la tabla **proc** de la base de datos **mysql** (base de datos del sistema de bases de datos).

Bibliografía

TeamViewer (s.f.). “What is TeamViewer?”. Disponible en: https://community.teamviewer.com/t5/Knowledge-Base/What-is-TeamViewer-incl-video/ta-p/33184. [Consulta: 23/10/2019].

Alison Balter (2006). “Introduction to SQL Server”. En: *Sams Teach Yourself SQL Server 2005 Express in 24 Hours.* New Riders, 2006, pp. 5-59.

Google (s.f.). “Introduction to Google Sheets API”. Disponible en: https://developers.google.com/sheets/api/guides/concepts. [Consulta: 23/10/2019].

Google (s.f.) “Google Sheets – Docs Editors Help”. Disponible en: https://support.google.com/docs/topic/9054603?hl=en&ref_topic=1382883. [Consulta: 23/10/2019].

Google (s.f.). “Overview of Google Apps Script”. Disponible en: https://developers.google.com/apps-script/overview. [Consulta: 23/10/2019].

Google (s.f.). “JDBC”. Disponible en: https://developers.google.com/apps-script/guides/jdbc. [Consulta: 23/10/2019].

The Linux Information Project (2005). “Daemon Definition”. Disponible en: http://www.linfo.org/daemon.html. [Consulta: 30/10/2019]

AA.VV. (s.f.). "Tom's Obvious, Minimal Language". Disponible en: https://github.com/toml-lang/toml. [Consulta: 30/10/2019].

Raymond, Eric S. (2003). “Basics of the Unix Philosophy”. En: *The Art of Unix Programming*. Disponible en: https://homepage.cs.uri.edu/~thenry/resources/unix_art/. [Consulta: 30/10/2019].

van Rossum, Guido (2001). “PEP 8 -- Style Guide for Python Code“. Disponible en: https://www.python.org/dev/peps/pep-0008/. [Consulta: 30/10/2019].

Python Software Foundation (2001). “The Python Tutorial”. Disponible en: https://docs.python.org/3/tutorial/index.html. [Consulta: 30/10/2019].

Oracle (2012). "MySQL Connector/Python Developer Guide". Disponible en: https://downloads.mysql.com/docs/connector-python-en.a4.pdf. [Consulta: 30/10/2019].

Dropbox (s.f.). "Dropbox for Python Documentation". Disponible en: https://dropbox-sdk-python.readthedocs.io/en/latest/index.html. [Consulta: 30/10/2019].