

Frontend de métricas

Maldonado Alcocer, Elizabeth Jhenny

Curs 2015-2016

Director: JUAN SOLER COMPANY

GRAU EN ENGINYERIA TELEMÀTICA



Universitat
Pompeu Fabra
Barcelona

Escola
Superior Politècnica

Treball de Fi de Grau

Frontend de métricas

Elizabeth Jhenny Maldonado Alcocer

TRABAJO DE FIN DE CARRERA

GRADO EN INGENIERÍA TELEMÁTICA

ESCOLA SUPERIOR POLITÈCNICA UPF

AÑO 2015/2016

JUAN SOLER COMPANY

BARCELONA 8 DE JULIO DE 2016

A mi familia.

AGRADECIMIENTOS

Primero agradecer a Carlos Moratalla por confiar en mí y permitirme realizar este proyecto. Sin su aprobación no habría sido posible.

A Daniel Cufi por toda la paciencia y soporte durante el desarrollo del proyecto. Sin su ayuda día tras día, todo habría sido más difícil.

Y dar gracias también al tutor Juan Soler, por el voto de confianza y por los mensajes de apoyo en momentos críticos.

RESUMEN

Aunque la aparición de las Web 2.0 y la evolución de las aplicaciones son la clave de las comunicaciones de hoy en día, no se deben olvidar las antiguas pero imprescindibles Webs informativas.

A raíz del descuido de esas Webs, surge este proyecto realizado en colaboración con Everis, donde el objetivo principal es crear una aplicación web que se diseñará e implementará como una herramienta informativa.

La aplicación está al alcance de cualquier tipo de usuario, es capaz de mostrar el estado de una máquina indicando, por ejemplo, el rendimiento de la CPU en un momento determinado. Es capaz también de mostrar el estado de los mensajes enviados a través de una aplicación en un tiempo concreto. El tiempo es tratado como un user-input ya que, se trata de un parámetro que puede ser configurado por el usuario.

Durante el desarrollo de la parte del cliente, las tecnologías de programación implicadas son Java, HTML, CSS o Javascript y en la del servidor Oracle SQL Developer y Oracle WebLogic.

Palabras clave: frontend, visualización de la información, data mining, AOD, aplicación web, Java, Oracle, WebLogic, servlet, d3, CSS, HTML, Javascript, SQLDeveloper, ARQDES, datasource.

RESUM

Tot i que l'aparició de les Web 2.0 i l'evolució de les aplicacions són la clau de les comunicacions d'avui dia, no es deuen oblidar les antigues però encara imprescindibles Webs informatives.

Arrel la manca de cura d'aquestes webs, sorgeix la proposta d'aquest projecte realitzat en col·laboració amb Everis. L'objectiu principal d'aquest treball consisteix en crear una aplicació web que es dissenyarà e implementarà com una eina informativa.

Aquesta aplicació està a l'abast de qualsevol tipus d'usuari o entitat, és capaç de mostrar l'estat d'una màquina indicant, per exemple, la memòria disponible o el rendiment de la CPU en un moment determinat. És capaç també de mostrar l'estat dels missatges enviats correctament a través d'una aplicació en un temps concret. El temps és tractat com un user-input ja que, es tracta d'un paràmetre que pot ser configurat per l'usuari.

Al desenvolupament de la part del client, les tecnologies de programació implicades són Java, HTML, CSS o Javascript i per la del servidor, són Oracle SQL Developer y Oracle WebLogic.

Paraules clau: frontend, visualización de la información, data mining, AOD, aplicación web, Java, Oracle, WebLogic, servlet, d3, CSS, HML, Javascript, SQLDeveloper, ARQDES, datasource.

ABSTRACT

Although the advent of Web 2.0 and the evolution of applications, these are the key of communications today, we must not forget the old but essential informative websites.

Following the neglect of these sites, there is the proposal of this project in collaboration with Everis. The main objective of this work is to create a web application that is designed and implemented as an information tool.

This application is available to any user or entity, is able to show the status of a machine indicating, for example, available memory or CPU performance at a given time. It is also able to show the status of messages sent correctly through an application in a specific time. The time is treated like a user-input because is a parameter that can be configured by the user.

For the development in the client-side, the technologies of programming involved are Java, HTML, CSS or Javascript and in the server side are Oracle SQL Developer and Oracle WebLogic.

Key words: frontend, visualización de la información, data mining, AOD, aplicación web, Java, Oracle, WebLogic, servlet, d3, CSS, HTML, Javascript, SQLDeveloper, ARQDES, datasource.

PREFACIO

Este es un proyecto que se lleva a cabo junto con la empresa Everis. Everis an NTT DATA Company es una consultora multinacional que ofrece soluciones de negocio, estrategia, desarrollo y mantenimiento de aplicaciones tecnológicas, y outsourcing. La compañía, desarrolla su actividad en los sectores de telecomunicaciones, entidades financieras, industria, utilities, energía, administración pública y sanidad [1].

El proyecto consiste en la combinación de diferentes procesos para la creación de una única aplicación capaz de mostrar información sobre otros proyectos, aplicaciones o tareas de negocio.

Se trata de crear una página web informativa, semejante a las estándar en la era de las Web 1.0, aquellas informativas y disponibles para cualquier tipo de usuario.

En esta aplicación web se muestran diferentes gráficos que indican distintos valores o índices de rendimiento, como el número total de mensajes enviados correctamente durante la semana.

La aplicación es capaz de recoger toda la información necesaria, almacenarla para después mostrarla por pantalla y además, ofrecer funcionalidades a los usuarios.

ÍNDICE

AGRADECIMIENTOS	v
RESUMEN	vii
RESUM	viii
ABSTRACT	ix
PREFACIO	xi
ÍNDICE	xiii
ÍNDICE FIGURAS	xv
ÍNDICE TABLAS	xvii

PARTE 1 DESCRIPCIÓN DEL PROYECTO	1
1. Introducción	1
1.1 Motivaciones	1
1.2 Objetivos	2
1.3 Metodología y estructura del trabajo	3
1.4 Planificación	6
2. Contexto temporal y circunstancial.....	9
2.1 Línea temporal de las aplicaciones web	9
2.2 Introducción a la Arquitectura del Software	12
2.3 Introducción al Frontend de métricas	16
3. La propuesta y el uso de las tecnologías.....	21
3.1 Modelo de trabajo actual.....	21
3.1.1 Generación de estadísticas	22
3.1.2 Aspectos a mejorar	23
3.2 Creando la librería	23
3.2.1 Elección del lenguaje de programación (JAVA).....	24
3.2.2 Funcionalidades	25
3.2.3 Justificación de la elección	27
3.3 Creando la aplicación web	29
3.3.1 Elección de las tecnologías para la programación.....	30
3.3.2 Funcionalidades	32
3.3.3 Justificación de la elección	33
3.4 Creando el modelo de base de datos	34
3.4.1 Elección de las tecnologías para la programación.....	35

3.4.2	Funcionalidades	35
3.4.3	Justificación de la elección	36
3.5	Otras características	38
3.5.1	Sistema de logs	39
4.	Conocimiento	41
PARTE 2 DESARROLLO DE LA APLICACIÓN		43
1.	Introducción	43
2.	Análisis y diseño	45
2.1	Requisitos.....	45
2.2	Diagramas de casos de uso	47
2.3	Diseño y Maquetación: pantallas y transiciones	51
3.	Desarrollo	55
3.1	Descripción de etapas del desarrollo	55
4.	Modelo de base de datos	57
4.1	Diagrama de base de datos.....	57
4.2	Tablas.....	58
4.2.1	Información sobre las tablas	58
5.	Desarrollo de la librería	63
5.1	Funcionalidades más relevantes del proyecto.....	63
5.2	Modelo de la creación de la librería.....	65
5.2.1	Adaptador para la recuperación de datos.....	68
5.2.2	Adaptador para la visualización de gráficos.....	73
5.3	Otras características	80
5.3.1	Fichero logs	81
6.	Desarrollo página web.....	83
6.1	Diseño	83
6.1.1	HTML.....	83
6.1.2	D3, Javascript y CSS	84
6.1.3	Bootstrap.....	85
7.	Inversión y ahorro.....	87
7.1	Inversión financiera	87
7.2	Ahorro.....	88
8.	Conclusiones	89
8.1	Objetivos alcanzados	89

8.2	Objetivos de mejora	90
8.3	Resultados y contribución a Everis.....	91
8.4	Valoraciones personales	91
9.	Anexos.....	93
10.	Bibliografía	95
11.	Glosario	1011

ÍNDICE FIGURAS

Figura 1:	Etapas de la metodología UML Based Web unificada.	3
Figura 2:	Diagrama de bloques donde se pueden ver las tres capas de desarrollo.	5
Figura 3:	Diagrama de Gantt con la planificación del proyecto.....	7
Figura 4:	Una pantalla de un esquema ENQUIRE.....	10
Figura 5:	Actual página web https://www.w3.org/History.html constituida en las primeras características del hipertexto y la Web.	11
Figura 6:	Arquitectura Cliente-Servidor.....	13
Figura 7:	Arquitectura Cliente-Servidor de tres capas.	14
Figura 8:	Diagrama de bloques patrón MVC del proyecto.	15
Figura 9:	Ejemplo de gráfico generado por el QC_mensual de la aplicación de mensajes enviados.....	18
Figura 10:	Diagrama de bloques relación tecnologías y partes del MVC.....	21
Figura 11:	Esquema funcionalidades de la librería.	27
Figura 12:	Rating uso lenguajes de programación Mayo 2016.....	29
Figura 13:	Esquema servidor WebLogic y aplicación web.....	30
Figura 14:	Tecnologías implicadas en el desarrollo de la aplicación.	41
Figura 15:	Conexión de servidor con la base de datos y el usuario.	43
Figura 16:	Diagrama de casos de uso para los usuarios.	48
Figura 17:	Diagrama de casos de uso para el administrador.	50
Figura 18:	Prototipo de la página principal de la aplicación.	51
Figura 19:	Prototipo del detalle de uno de los gráficos de la pantalla principal.	52
Figura 20:	Prototipo de la página “Añadir nuevo valor”.....	53
Figura 21:	Prototipo de la página “Información nuevo valor”.....	53
Figura 22:	Diagrama del modelo de base de datos.....	57
Figura 23:	Proceso que se sigue a la hora insertar datos.	64
Figura 24:	Comando con el que se insertarán tantos valores como se quieran a la base de datos. Con esto se evita hacerlo manualmente por el formulario de la aplicación. ...	64
Figura 25:	Proceso que se sigue cuando se pasan el objeto que los gráficos necesitan para mostrarse en la pantalla.	65
Figura 26:	Consola Weblogic donde se puede ver el nombre del JNDI del origen de datos creado.	66

Figura 27: Fichero web.xml del proyecto Java donde se puede ver el nombre de la aplicación, front4, subrayado.....	67
Figura 28: Fichero frontend-servlet.xml del proyecto Java donde se puede ver la invocación al JNDI creado en el servidor Weblogic.....	67
Figura 29: Diagrama de clases para la recuperación de datos.....	68
Figura 30: Se define la clase AplicacionController.java como controlador y /aplicacion como la dirección url con la que el usuario llamará, desde el servidor, a cada una de las funciones que hayan en esta clase, si no tienen otra dirección adicional.....	69
Figura 31: Método nuevaAplicacion que se encuentra en el controlador AplicacionController.....	69
Figura 32: La variable aplicacionesDAO es de tipo AplicacionDAO.....	70
Figura 33: JdbcTemplate es la forma como Spring permite acceder al origen de datos y permitir la conexión a la base de datos para ejecutar queries, por ejemplo. Como se puede ver datasourceARQDESONline coincide con el nombre del datasource que contiene el JNDI del proyecto, como se vio en el fichero frontend-servlet.xml.....	71
Figura 34: Query para insertar las tres variables en la tabla FRONTEND_METRICAS de la base de datos.....	71
Figura 35: Update es un método del jdbcTemplate que se usa para ejecutar la query insert.....	72
Figura 36: Método insertaAplicación de la clase AplicacionController donde se devuelve el JSP resultado con los valores insertados por el usuario.....	72
Figura 37: Diagrama de clases para la visualización de los gráficos.....	73
Figura 38: El usuario puede acceder a los métodos del controlador desde el servidor, con la dirección /index si no se habilitan otras.....	73
Figura 39: Método pintamosPantallaPrincipal al que se accede desde el servidor con la dirección /index.....	74
Figura 40: Método getCaracterisClaGrafs que ejecuta una query para extraer la clave y el tipo de operación que tiene un determinado id_grafico, que se pasa como parámetro.....	75
Figura 41: ToolClaveGraficoMapper donde se ve el mapeo de 3 columnas.....	76
Figura 42: En el método devolverListaDias los parámetros fechaIni y fechaFin se pasan al formato de fecha que interesa.....	76
Figura 43: Código del método devolverListaDias donde se puede ver cómo se van añadiendo las fechas en la listaDias, hasta llegar a la fechaFin.....	76
Figura 44: Método getCaracterisGrafos de la clase EstadisticasDAOImpl y que devuelve la listaCarGrafs por cada id_grafico que se pase en la invocación.....	77
Figura 45: Método getListaAplicaciones con la que se obtienen todos los valores de las claves que contienen cada uno de los gráficos.....	78
Figura 46: Parte del código que sirve para rellenar el objeto con el que se pintarán los gráficos de la aplicación.....	79
Figura 47: Se pasa el objeto listaApps al JSP que toque dependiendo del id_grafico que sea. Este JSP es el tipoGrafico que se extrae de la lista.....	79

Figura 48: Método del controlador que devuelve el JSP que contiene el detalle del gráfico con cada id_grafico que se pasa. En este ejemplo, el JSP que se devuelve ha de ser el del id_grafico=totalSMSEnviadosPorOperadora.....	80
Figura 49: Método de la clase Init donde se declara initLogs.....	81
Figura 50: Se avisa que initLogs de la clase Init será el que escuchará si hay qué logs hay en cada ejecución.....	81
Figura 51: Se direcciona al método con url ./insertaAplicacion.....	83
Figura 52: Iframe que muestra el gráfico con identificador PlantillasPorMaquina.....	84
Figura 53: Array es el objeto que se recupera de la librería.....	85
Figura 54: Modelo de base de datos que se ha diseñado como mejora del proyecto.....	90

ÍNDICE TABLAS

Tabla 1: Tabla con la planificación del proyecto.....	7
Tabla 2: Una de las pestañas del QC_mensual generado para la aplicación de mensajes enviados.....	17
Tabla 3: Una de las pestañas del QC_mensual generado para la aplicación de subida de plantillas.....	19
Tabla 4: Comparación tecnologías similares a d3.js.....	34
Tabla 5: Comparación diferentes plataformas para la gestión de datos.....	37
Tabla 6: Casos de uso de la aplicación.....	47
Tabla 7: Tabla FRONTEND_CLAVES de la base de datos.....	59
Tabla 8: Tabla FRONTEND_METRICAS de la base de datos.....	59
Tabla 9: Tabla FRONTEND_GRAFICOS de la base de datos.....	60
Tabla 10: Tabla FRONTEND_CLAVEGRAFICO de la base de datos.....	60

PARTE 1 DESCRIPCIÓN DEL PROYECTO

1. Introducción

En estos últimos meses, el número de aplicaciones gestionadas por el departamento de Arquitectura de Everis, ha incrementado notablemente. Esto ha hecho pensar al grupo, desarrollar alguna herramienta para mantener informado al cliente al que pertenece cada aplicación, el estado en el que se encuentra cada una.

Por esta razón, desde el Grupo de Proyectos Arquitectura de Objetos Digitales (en adelante AOD, dadas sus siglas en castellano) nace la iniciativa de presentar este proyecto como propuesta.

Con esta aplicación sacarán beneficios los usuarios o clientes interesados y también los propios departamentos de Everis, el caso más próximo y seguro AOD. Desde el lado del usuario o cliente, servirá para informar sobre el estado de sus aplicaciones o procesos, y desde el de Everis, para ver la eficacia de cada departamento, la carga de trabajo que hay en esta o para el mismo seguimiento de rendimiento de las aplicaciones del cliente.

Como se verá más adelante, estos índices de rendimiento son varios y dependen de la aplicación que se esté tratando. Puede ser el número de plantillas subidas a una máquina, el número de mensajes enviados en un día o el tiempo medio que tardan en procesarse ciertos documentos.

Respecto al desarrollo del proyecto y como se comentará más adelante, la aplicación está diseñada para ser ejecutada en un servidor de la empresa y así, ser utilizado por los demás compañeros de Everis.

En este capítulo de la memoria, se explicarán las diferentes motivaciones que impulsaron a su realización, se expondrán los objetivos de éste, la metodología de trabajo utilizada y finalmente, se mostrará la planificación temporal del trabajo realizado.

1.1 Motivaciones

AOD da soporte y está presente en diversos proyectos para la digitalización de documentos. En ésta se realizan tareas como la resolución de incidencias, el testeo de aplicaciones, el desarrollo de aplicaciones o la creación de estadísticas para la comprobación del estado de algunas aplicaciones en todo momento. Esta última tarea mencionada, fue el motivo principal por el que se propuso la realización de una plataforma web que recogiera dichas estadísticas. De esta forma, clientes interesados en saber cómo se encuentra el funcionamiento o rendimiento de sus aplicaciones pudiese, de manera más visual y cómoda, verlo a través de esta herramienta.

Aunque la tarea de generar estadísticas ya es realizada por un trabajador de Everis cada mes, la forma de visualización no es en una plataforma web como lo será con este proyecto. Cabe decir que, antes la forma en la que se generaban estos cálculos no era automática por lo que, implicaban muchas horas dedicadas. Se trataba de un proceso manual con los que se generaban diferentes estadísticas mensuales calculando una variedad de tasas de rendimiento dependiendo del proyecto.

Al haber varios proyectos, eran muchos procesos y por tanto, muchas horas perdidas. En los últimos meses parte de estos procesos han cambiado y ahora muchas de estas estadísticas son automáticas, y con esto una de las motivaciones principales de este proyecto es mostrar de manera visual parte de estos procesos ya automatizados. De esta forma, tanto los clientes como cada departamento podrán ver números de manera más entendedora y fácil.

Por otro lado, otra de las motivaciones son los buenos resultados esperados ya que, desde el grupo se busca una aplicación sencilla, práctica y dinámica. Con estas características se espera que, en el momento que se decidiese ampliar funcionalidades o introducir un mayor número de estadísticas no hubiera problemas y fuese rápido hacerlo. Por esto y tal y como se verá en el apartado de planificación del proyecto se dedica un tiempo bastante amplio a la toma de requerimientos.

Finalmente, cabe decir que, en el mundo de la empresa de consultoría hay mucha competencia por lo que es importante mantener un cliente ofreciendo calidad en los servicios que ya se ofrecen y es por esto, otra motivación para el proyecto.

1.2 Objetivos

Los objetivos del proyecto se han ido introduciendo en el apartado anterior pero, de manera resumida, lo principal es crear una aplicación web que recoja y relacione diferentes aplicaciones. De esta forma en común, las estadísticas de cada aplicación pueden mostrarse, usando una misma plataforma y durante un determinado rango de tiempo.

Entrando en más detalle, desde AOD se pide una única aplicación web con la que poder visualizar datos y valores de diferentes aplicaciones en una sola plataforma. Así, no habría la necesidad de desarrollar una para cada una sino que, se reaprovechará la misma con pequeñas modificaciones. Siendo así, esta será capaz de permitir añadir alguna nueva funcionalidad o ampliar el número de gráficos sin que esto suponga mucho tiempo y dificultades para un desarrollo futuro.

Al tener poca experiencia en el ámbito de las aplicaciones web, el desarrollo de este proyecto ha supuesto una serie de objetivos formativos desde el primer día. Con este proyecto, el aprendizaje y ampliación de las competencias a adquirir están presentes durante todas las etapas. Cabe decir que, el soporte aportado por el grupo de AOD y la

profundización a través de, libros y/o Internet han sido la mejor fuente de adquisición de nuevos conceptos y conocimientos en programación y diseño durante toda la realización del proyecto.

En la parte de diseño nos encontramos con el reto de realizar una aplicación web atractiva sin tener apenas conocimiento. Por lo tanto, se ha tenido en cuenta unos mínimos criterios de usabilidad para que los usuarios se sientan cómodos utilizando la herramienta. Y por otro lado, este diseño genera otro objetivo que es el de aprender las tecnologías útiles en este entorno.

Por la parte de la programación, el reto también ha estado en aprender a utilizar las tecnologías. Tecnologías que desde el inicio se tenían claras debido al entorno de trabajo determinado en AOD. Aunque ya tengo algún conocimiento de estas, por tareas realizadas hasta ahora como becaria, no cuento con la suficiente experiencia como para manejarlas igual a otro de los integrantes del grupo.

1.3 Metodología y estructura del trabajo

La metodología que sigue el proyecto está basado en el modelo unificado UML Based Web, una herramienta para modelar aplicaciones web utilizadas en la ingeniería web o también conocida como ingeniería del software. Esta metodología diferencia cuatro fases o etapas durante el desarrollo de una aplicación donde la principal atención se encuentra en la primera y última etapa ya que, en ambas se busca la máxima eficiencia [2].

Es importante mencionar que esta metodología tampoco limitará la forma de desarrollar la aplicación ya que, en diferentes fases o etapas se harán modificaciones a medida que se vaya avanzando. Podemos ver gráficamente estas cuatro etapas a continuación:

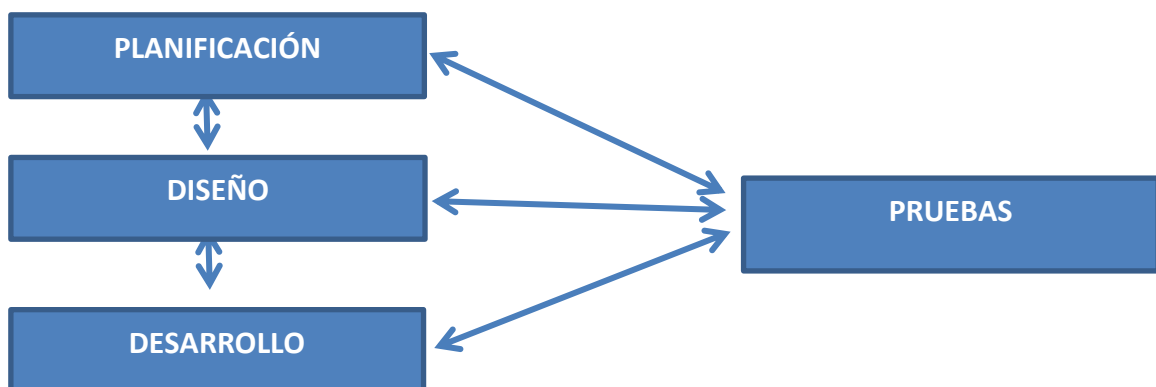


Figura 1: Etapas de la metodología UML Based Web unificada.

Durante la etapa de **planificación** se realiza la recogida de requisitos, se especifican las características funcionales y no funcionales que ofrece esta aplicación web. En esta etapa también está incluido el diseño de software, que engloba toda la parte

correspondiente a la descripción de los componentes, tecnologías y características con las que se estará organizado el proyecto. Para que las siguientes etapas sean más sencillas, se va a tomar buen tiempo antes de finalizar la parte de toma de requerimientos. Tal y como veremos en la planificación temporal del proyecto, una vez se tenga claro todo lo que se necesita y lo que se quiere, se procederá a la siguiente etapa, la del **diseño**.

En esta etapa de **planificación** o más bien toma de requerimientos se han centrado todas las fuerzas en buscar los diferentes gráficos que se van a utilizar para visualizar los índices que interesan. Para poder buscar los gráficos apropiados, antes se ha dedicado mucho tiempo en buscar la tecnología que se iba a utilizar para la visualización de los gráficos. No se trata de una simple visualización sino que es en cierta manera dinámico ya que, estos gráficos se van actualizando cada vez que haya valores nuevos. Por esto, la tecnología a utilizar debía ser bastante flexible y además no debía ser complicada de integrar. Después de varias pruebas y como se explicará más adelante, con un poco más de detalle, se ha decidido utilizar la librería Data Driven Documents (en adelante d3, dadas sus siglas en inglés) disponible para Javascript. Después de esta medida tomada, de los gráficos decididos, tocaba pensar en las funcionalidades que los usuarios podían tener al acceder a la aplicación. En este proyecto estas funcionalidades están limitadas como es de imaginar ya que, como ya se ha dicho se trata de una web informativa. Por este motivo estas funcionalidades de forma resumida (se explicarán en detalle más adelante) son filtrar los datos que se muestran, cambiar el tipo de gráfico y ver el detalle de los gráficos.

Aunque en la etapa de **diseño** no se ha perdido demasiado tiempo (ya que, el trabajo más importante era el de antes) se ha tratado que sea lo más simple y atractiva posible. Se cuenta con una librería que es llamativa visualmente y ofrece una amplia variedad de gráficos así que, la tarea es fácil. Otra de las cosas que se decide en esta etapa es la estructura que se le va a dar a la aplicación.

La fase de **desarrollo** es la más importante del proceso. El lenguaje de programación elegido ha sido Java. Al tratarse de una arquitectura cliente-servidor bastante sencilla, lo más eficiente era dividir el desarrollo en tres capas o niveles tal y como describe la ingeniería del software. Esto consiste en separar la capa de datos de la capa de presentación al usuario. La principal ventaja de hacerlo de esta forma es la facilidad para cambiar o añadir código. Estas tres capas en las que se dividen son más conocidas como la **capa de presentación**, la **capa de negocio** y la **capa de datos** [3].

En este proyecto la **capa de presentación** se corresponde a la visualización de los gráficos o estadísticas, es decir, la parte que ve el usuario. Esta parte de desarrollo es básicamente lenguaje HyperText Markup Language (en adelante HTML, dadas sus siglas en inglés), Javascript y Cascading Style Sheets (en adelante CSS, dadas sus siglas en inglés).

La **capa de negocio** es donde encontramos todos los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. En esta capa de negocio es donde se establecen las reglas que deben cumplirse durante el desarrollo del proyecto y comunica con capa de presentación para recibir las peticiones de los usuarios y con la capa de datos para almacenar o recuperar datos de él [3]. En este proyecto el encargado de esta parte es el Enterprise Archive (en adelante EAR, dadas sus siglas en inglés) generado para el desarrollo de la aplicación y que está conectado a nuestro servidor de aplicaciones Oracle WebLogic. Dicho EAR se encarga de empaquetar la aplicación y de describir su configuración de despliegue [4]. Cuando el servidor lee esta configuración de despliegue sabe de qué aplicación se trata (porque conoce la ruta y el resto de detalles de configuración) y la ejecuta.

Para acabar con la división de la etapa de desarrollo, la capa de datos es la parte donde se almacenan los datos, reciben peticiones de los usuarios o se recupera la información.

La base de datos al que está conectado es uno de la empresa llamado ARQDES o Arquitectura de Desarrollo. En cuanto a tecnología, el implicado en esta parte es Oracle SQLDeveloper.

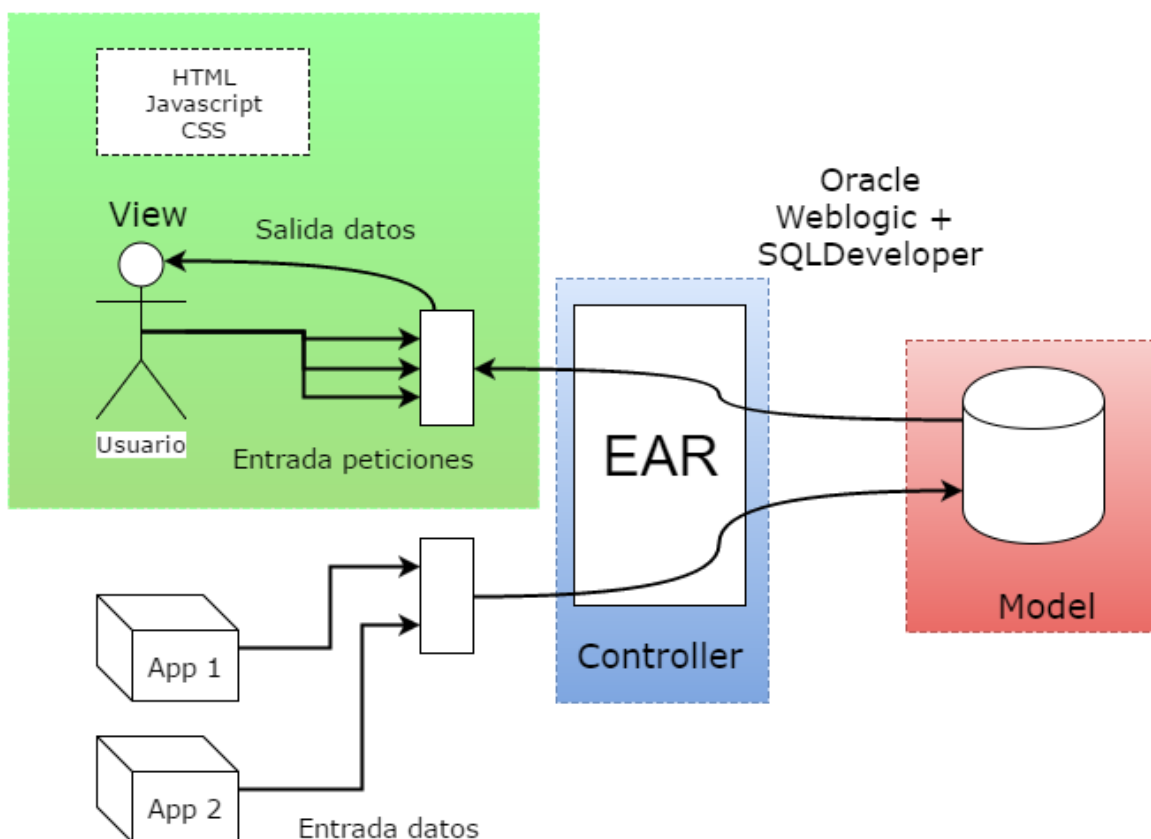


Figura 2: Diagrama de bloques donde se pueden ver las tres capas de desarrollo.

Aprovechando el concepto de desarrollo en capas se ha elegido (se explicará con más detalle más adelante) el patrón de arquitectura de software conocido como **Modelo Vista Controlador** (en adelante MVC, dadas sus siglas en castellano). Para entender mejor la etapa de desarrollo de este proyecto, se puede ver en **forma de esquema** el diagrama de la Figura 2, donde hay una relación entre las capas de desarrollo, las tecnologías implicadas y el patrón de arquitectura MVC escogido.

Tal y como se ha explicado y cómo podemos ver en la Figura 2, la capa de presentación, implica el uso de tecnologías de la programación como html, javascript o css.

La capa de negocio usa tecnologías como Java y Oracle WebLogic y la capa de datos, Java y SQLDeveloper.

Finalmente, la etapa de **pruebas** se va realizando durante todas las fases para asegurar el correcto funcionamiento de las diferentes secciones tanto de código, como de diseño o requerimientos.

Y por lo que es la **estructura del trabajo** tal y como se puede ver en el índice del proyecto se divide en dos partes diferenciadas: la parte más **teórica** y la parte del **desarrollo**, donde se aplica lo introducido en la parte teórica. Estas dos partes a su vez se organizan siguiendo las partes del MVC.

1.4 Planificación

En función de las necesidades identificadas y de la previsión inicial realizada desde AOD, se ha estimado que el proyecto tiene una duración de 78 días es decir, cerca de 3 meses: desde la segunda semana de Marzo de 2016 hasta la última semana del mes de Mayo, contando con la redacción de la memoria hasta el fin del proyecto.

La toma de requerimientos tiene una duración de aproximadamente 3 semanas ya que, es el momento en el que se deciden qué tipo de visualizaciones se realizarán, qué tipo de funcionalidades se irán a implementar y siempre teniendo en cuenta las necesidades a cubrir.

Una vez confirmados los requisitos, el proceso de diseño, que viene ligado a la etapa anterior de la aplicación, es dónde se decide, mediante un diagrama de bloques arquitectura y la herramienta online Mockup, el diseño inicial de las pantallas. Esta etapa tiene una duración de una semana aproximadamente y en cuanto sean aceptadas por AOD, se pasará al desarrollo de la aplicación.

Una vez aceptado y claro el diseño, se empieza con el desarrollo de toda la interfaz de la aplicación o más conocida como el front-end, capa de presentación [5]. Esta fase tiene una duración de un mes y medio ya que, es la parte más importante de todo el proyecto.

Una vez finalizada la parte del desarrollo, se harán una serie de testeos para comprobar el correcto funcionamiento de la aplicación web.

Como se ha comentado antes, la memoria es redactada durante todo el proyecto, tal y como se puede ver en la siguiente tabla:

Actividades	Inicio	Duración (días)	Fin
Toma Requerimientos	14/03/2016	17	31/03/2016
Diseño Aplicativo	01/04/2016	6	08/04/2016
Diseño Front-end	01/04/2016	5	06/04/2016
Prototipo en Mockup	07/04/2016	1	08/04/2016
Desarrollo Front-end	09/04/2016	40	19/05/2016
Testing Front-end	20/05/2016	7	27/05/2016
Documentación Memoria	14/03/2016	78	31/05/2016

Tabla 1: Tabla con la planificación del proyecto.

De forma análoga pero, para que la planificación sea más visual podemos ver el siguiente diagrama de Gantt:

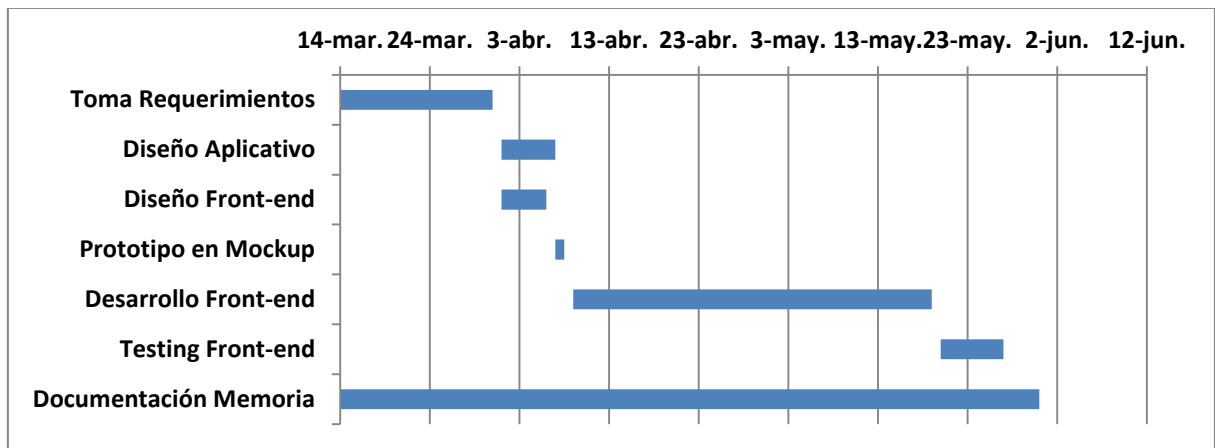


Figura 3: Diagrama de Gantt con la planificación del proyecto.

2. Contexto temporal y circunstancial

En este capítulo se situará el proyecto en contexto, para poder entender mejor en qué situación se ha realizado. Para hacerlo lo primero que se explicará, a grandes rasgos, son los cambios que han ido sufriendo las páginas web desde su inicio hasta este momento. Después entraremos un poco más en el desarrollo y veremos una introducción a la arquitectura del software y se acabará con una introducción al frontend de métricas.

2.1 Línea temporal de las aplicaciones web

Antes de poder empezar con este proyecto era necesario conocer un poco del origen de las páginas web para poder situar en el tiempo, el momento en que aparecieron las primeras aplicaciones web con una idea parecida.

Según diversas fuentes se sitúa en los años 80 la primera aparición de un proyecto software o más conocido en la actualidad como página web [6]. Este proyecto, que fue llamado ENQUIRE [7], fue desarrollado por Sir Timothy “Tim” John Berners-Lee, un científico de la computación británico.

Inicialmente **ENQUIRE** no estaba pensado para dirigirse al público. **No se trataba realmente de una web** como la conocemos en estos tiempos pero, si se tiene que hacer alguna comparación, ENQUIRE era o se parecía al actual concepto de una wiki. Contaba con una base de datos sobre la que se trabajaba, presentaba hipervínculos bidireccionales (enlaces entre páginas de doble sentido), permitía una edición directa del servidor tal y como pasa con las wikis o blogs y su manejo era sencillo [8].

En la Figura 3 se puede ver una pantalla de ENQUIRE que básicamente representa una lista de links. Esta lista de links incluye un tipo, como puede ser “includes”, antes de cada comentario que hay sobre cada link. Otra de las cosas que también se puede ver en la Figura 3 es la barra de herramientas que hay al final de la página dónde hay opciones o botones como Añadir, Atrás o Editar [7].

Pocos años después de ENQUIRE, sobre el año 1989, es el mismo Tim Berners-Lee quien regresa a antiguos proyectos donde realmente nace la World Wide Web (en adelante WWW, dadas sus siglas en inglés) y en consecuencia, las distintas páginas web que hoy en día conocemos.

Sus orígenes ocurrieron en el Centro Europeo de Investigación Nuclear (CERN), aquí Tim Berners-Lee se encargó de diseñar un método con el que hacer fácil el acceso a la información del centro de investigación. Lo presentó como proyecto, recibió aprobación para continuar y fue así como nació el primer navegador web, llamado **WorldWideWeb** [9]. Este sistema igual que el ENQUIRE utilizaba una red de enlaces

o links. El sitio web describía las características básicas de la propia web, indicaba cómo acceder a documentos de otras personas y cómo configurar su propio servidor. Era básicamente informativo basado en hipertexto [6].

```
Documentation of the RPC project (concept)

Most of the documentation is available on VMS, with the two
principle manuals being stored in the CERNDOC system.

1) includes: The VAX/NOTES conference VXCERN::RPC
2) includes: Test and Example suite
3) includes: RPC BUG LISTS
4) includes: RPC System: Implementation Guide
   Information for maintenance, porting, etc.
5) includes: Suggested Development Strategy for RPC Applications
6) includes: "Notes on RPC", Draft 1, 20 feb 86
7) includes: "Notes on Proposed RPC Development" 18 Feb 86
8) includes: RPC User Manual
   How to build and run a distributed system.
9) includes: Draft Specifications and Implementation Notes
10) includes: The RPC HELP facility
11) describes: THE REMOTE PROCEDURE CALL PROJECT in DD/OC

Help Display Select Back Quit Mark Goto_mark Link Add Edit
```

Figura 4: Una pantalla de un esquema ENQUIRE.

Durante un año Lee desarrolló todas las herramientas que se necesitaban para que la Web funcionara, desde el Protocolo de Transferencia de Hipertexto (en adelante HTTP, dadas sus siglas en inglés) hasta el servidor web (actualmente activo, <http://info.cern.ch>) [6]. El protocolo Hypertext Transfer Protocol o más conocido como HTTP es el protocolo base de la WWW, es decir, el usuario establece conexión con la WWW mediante este protocolo y es de esta manera cómo puede visualizar la página web y su contenido [9].

Durante ese año el crecimiento de la WWW se había extendido, aparecieron nuevos proyectos. Dos de los más importantes fueron el llamado Mosaic diseñado para la National Center for Supercomputing Applications (NCSA) de la Universidad University of Illinois at Urbana-Champaign (UIUC) y el primer navegador (Cello) del actual Microsoft Windows.

Todos estos proyectos creados seguían el mismo sistema, el del hipertexto y el protocolo de comunicaciones de Internet HTTP, saltando de un enlace a otro dentro de un texto, a través de enlaces [10]. La mayoría de estos navegadores tenían una página principal con un menú a diferentes enlaces. En cada uno de estos enlaces la mayor parte era texto, texto informativo sobre el funcionamiento de la propia página o la historia

sobre algún hecho. Además de texto también contaban con algunas imágenes informativas [11].

Aunque año tras año eran más las comunidades que se iban uniendo a la web, no fue hasta el año 1994 que el código se hizo disponible libremente sin patentes ni pago de derechos de autor [12].

En la siguiente imagen (Figura 5) podemos ver un ejemplo de uno de los primeros navegadores donde se reflejan las características de esos métodos de manera muy significativa:



Figura 5: Actual página web <https://www.w3.org/History.html> constituida en las primeras características del hipertexto y la Web.

Poco después, entre los años 1996 y 2001, los comercios ya veían en la web la posibilidad de hacer crecer la venta de sus productos a través de Internet. Por esto, una gran cantidad de compañías grandes y pequeñas tenía planes e ideas de construir una web propia mediante la que pudieran dar a conocer con mayor facilidad sus productos. La idea de esta etapa se conoce como comercio electrónico, forma o medio como una compañía o comercio se promueve a sí misma a través de Internet.

Aunque la mayoría de esos planes eran realistas y hábiles, pocas tuvieron éxito. El problema es que tenían tan poca experiencia en la web que, muchas de las características creadas no eran posibles o dirigían a errores graves. Por esta razón, algunas se quedaron en simples ideas, otras se intentaron y florecieron, muchas fracasaron y otras por el contrario vendieron la idea a inversores y tuvieron éxito. A esta etapa de la web se la conoce como el auge de las **empresas punto-com** [12].

No todo fue malo porque de este fracaso se aprovecharon ideas como las que actualmente conocemos y que son herramientas imprescindibles en la red. Dos de los

ejemplos claros son el buscador de Google o Amazon (uno de los comercios electrónicos que tuvo éxito).

Después del fin de las empresas punto-com hasta el presente, se conoce la web como **Web 2.0**. Los principales cambios los podemos ver por nosotros mismos, el éxito de las redes sociales es algo innegable e imparable (en enero de 2016 Facebook alcanzó 1590 millones de usuarios activos, activo al menos una vez al mes [13]). La gracia de estos, al igual del resto de navegadores nacidos en esta etapa de la web, está en la necesidad que crean en nosotros de estar siempre en contacto con nuestro entorno ya sea social, laboral o educativo.

Otro de los éxitos es el de las aplicaciones web dinámicas en las que, como eBay los encargados de mantener la página actualizada con los productos más recientes somos nosotros mismos o el de los blogs donde cualquiera puede compartir información, hobby o cualquier tema que se le ocurra con el resto de usuarios.

Siguiendo la misma filosofía, otro de los grandes éxitos igual que el de Google es el de YouTube, donde podemos encontrar la forma más fácil y rápida de ver, compartir (con cualquier otra persona o en cualquier otro sitio web mediante Internet) y subir (cargar un video y crearlo en tu cuenta) un video.

La clave de esta nueva etapa está en la facilidad que se nos ha dado para poder compartir, crear y obtener cualquier tipo de información o servicio.

Sin mencionar los cambios de diseño que como podemos imaginar han ido evolucionando a medida que han ido apareciendo más técnicas y lenguajes de programación, el método del hipertexto y el protocolo HTTP siguen coexistiendo. Cabe mencionar que, a raíz de los cambios en los navegadores y el avance del comercio electrónico o las redes sociales, existe una nueva forma de comunicación mediante el protocolo HTTP que es más seguro. Este nuevo protocolo es conocido como HTTPS y la diferencia con el anterior es que utiliza un canal cifrado a la hora de intercambiar datos con el usuario (por ejemplo cuando se hace login en alguna red social o cuando se paga con tarjeta de crédito) [14].

Después de ver un poco como ha sido el paso de las páginas web en el tiempo hasta la actualidad, y como se ha comentado en apartados anteriores, este proyecto se asemeja al concepto de Web 1.0 o mejor dicho a la idea de las empresas punto-com. Aunque no tiene nada que ver con un comercio electrónico o algo parecido, adopta la idea de web informativa y sencilla.

2.2 Introducción a la Arquitectura del Software

El siguiente paso después de saber el origen de las aplicaciones web es conocer un poco la estructura o diseño del proyecto.

Se conoce como arquitectura la forma en cómo se resolverá un proyecto, en cómo estará estructurada (tanto la parte del diseño como la parte del desarrollo), las funcionalidades e interacciones que habrán entre las partes que diferencie o las restricciones que se determinen [15].

Existen diferentes métodos y dependiendo del proyecto unos son más convenientes que otros.

Cada arquitectura de software determina diversos aspectos del proyecto a su manera y utilizando modelos o vistas. Estos modelos o vistas son las partes en los que está organizado un sistema. Cada modelo o vista está enfocado en una característica en concreto, un ejemplo, es la base de datos (encargado de enviar y/o recibir información y tiene relación con el resto que componen el sistema) [16].

De todas las arquitecturas de software que existen las más utilizadas en entornos como lo son las aplicaciones web son: las aplicaciones monolíticas y la arquitectura Cliente-Servidor.

- › **Aplicaciones monolíticas:** se trata de una estructura sencilla que está compuesta únicamente por una máquina. Describe una aplicación de software donde tanto la interfaz de usuario como el código de acceso a datos se combinan en un solo programa de una plataforma única. Lo que significa que hay una única máquina con único usuario accediendo al servicio [17].
- › **Arquitectura Cliente-Servidor:** es una de las arquitecturas más típicas de las aplicaciones web, están basadas en las peticiones que un cliente realiza a un programa, el servidor, y las respuestas (dando el servicio) que este le devuelve [16].

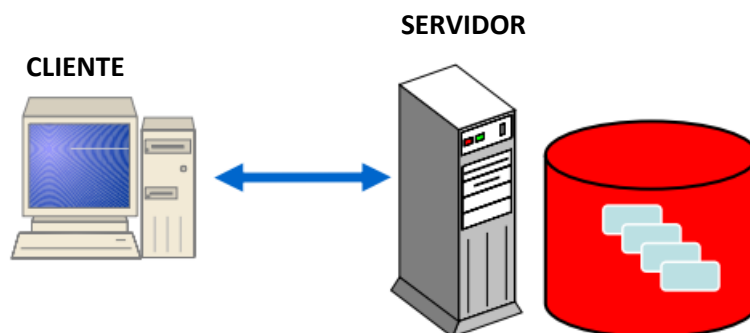


Figura 6: Arquitectura Cliente-Servidor.

Tal y como podemos ver en la Figura 5, la máquina representa el cliente accediendo al servidor mediante peticiones. Estas peticiones son procesadas por el servidor y cuando son resueltas se envían en forma de respuesta al cliente, por esto la relación

bidireccional de la imagen. Por tanto, la máquina del usuario se comunica con el servidor para usar la aplicación (acceder a la página web).

Al tratarse de una aplicación web, y como se ha comentado en la parte de metodología y estructura del trabajo, la arquitectura que se va a utilizar es la de tres niveles o capas, un tipo de arquitectura Cliente-Servidor que estructura la aplicación en tres capas. En concreto, se ha dicho que se va a usar el modelo MVC. El funcionamiento básico consiste en un ordenador (cliente) que se comunica, a través de peticiones a otro ordenador (el servidor) y este le envía mensajes de respuesta a sus peticiones mediante un protocolo de comunicación HTTP, tal y como podemos ver en la siguiente imagen (Figura 6). En medio de estas peticiones y respuestas hay un controlador que hace de negociador entre las otras dos capas.

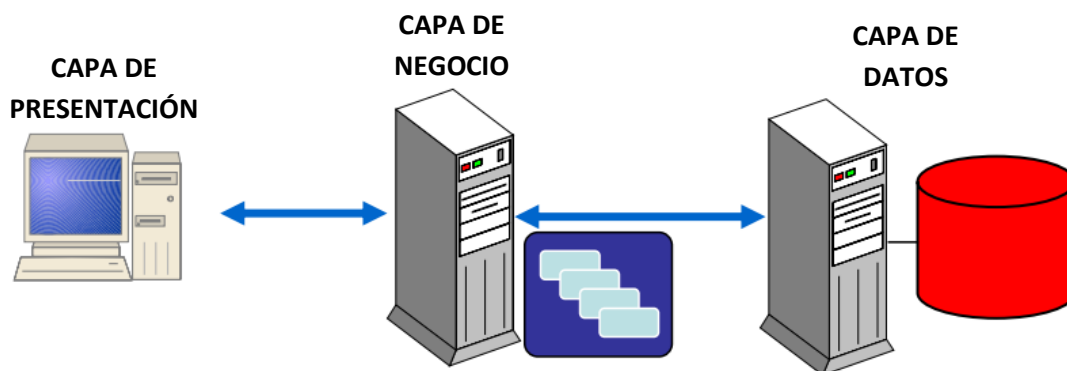


Figura 7: Arquitectura Cliente-Servidor de tres capas.

El MVC es muy frecuente en aplicaciones web, ya que permite desacoplar la programación del diseño gráfico, con lo que se puede tener dos equipos de personas especializadas trabajando en paralelo, además de simplificar el mantenimiento del código.

Antes de entrar en detalle explicaremos un poco conceptos importantes de este patrón cuando hablamos de modelo, vista y controlador.

Un **modelo** se encarga del tratamiento de la información del sistema, generalmente datos almacenados en una base de datos, ya sea devolviendo la información requerida por el usuario o **vista** como haciendo actualizaciones o añadiendo información [18]. A esta parte del patrón MVC se la asocia a la capa de datos de la arquitectura.

Una **vista** presenta los datos devueltos por el modelo de forma que el usuario pueda entenderlos e interactuar con ellos. Esta parte del patrón se denomina también capa de presentación, tal y como lo indica la arquitectura de tres capas [18].

Y finalmente, un **controlador** recibe las interacciones del usuario con la **vista**, las procesa y realiza las llamadas a los **modelos** para obtener la información que debe

devolver a la nueva vista que se va a mostrar [18]. Esta parte del patrón MVC se le conoce también como capa de negocio en la arquitectura.

Existen diferentes maneras de implementar un MVC, pero en este proyecto la aplicación web seguirá los siguientes pasos:

- › El usuario interactúa con la **vista**.
- › El **controlador** recibe las acciones solicitadas por los usuarios.
- › El **controlador** accede al **modelo** para realizar las acciones requeridas: recoger información, actualizar datos o insertar nuevas entradas.
- › El **controlador** envía nuevos datos a la vista como respuesta a las acciones solicitadas.
- › Finalmente, la **vista** muestra los datos recibidos y espera que el usuario vuelva a realizar otra acción.

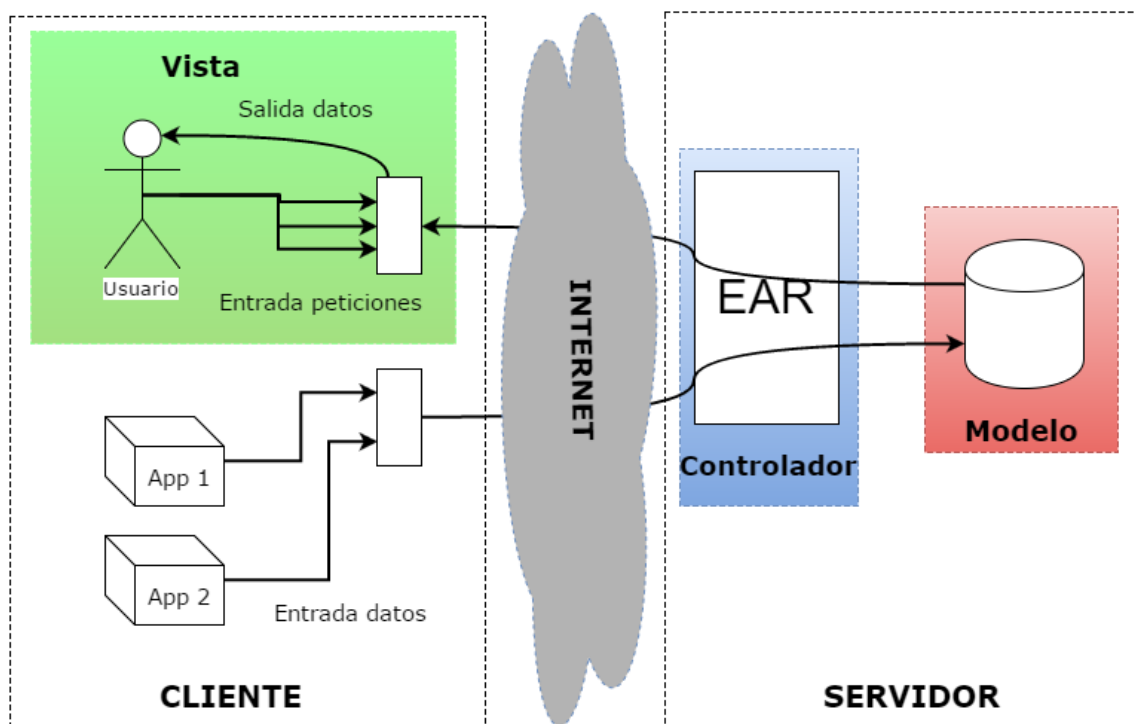


Figura 8: Diagrama de bloques patrón MVC del proyecto.

En la imagen que se presentó en el apartado de metodología del proyecto se podía ver el patrón MVC que se correspondía a este proyecto. Para explicarlo un poquito más en detalle volvemos a retomarlo, pero con algunas modificaciones como se puede ver.

Como podemos ver en la Figura 6 y como se entrará en detalle en la parte de desarrollo, este proyecto diferencia dos tipos de entrada de petición: el de los usuarios y el de las aplicaciones. Los usuarios son los que van a ser los interesados en utilizar la aplicación y las aplicaciones que son las que nos proporcionarán los datos que se muestran en pantalla cuando el usuario las solicita.

Por el lado del servidor sigue el mismo esquema que la mayoría de aplicaciones, tiene un controlador que es en este caso el EAR generado para la aplicación, el servidor de Oracle WebLogic y la base de datos ARQDES.

2.3 Introducción al Frontend de métricas

Ahora que ya sabemos cuándo aparecieron el tipo de aplicaciones web que nos interesan y cuál es la estructura del proyecto toca definir qué es, para qué sirve y cómo se calculará el “Frontend de métricas”.

En Everis lo más importante es buscar la máxima satisfacción de los clientes y para esto se le pone mucho énfasis en la extracción de datos numéricos, estadísticas diarias y/o indicadores mensuales que puedan mostrar el rendimiento en cualquier momento.

Existen numerosas aplicaciones de soporte para la generación de dichos cálculos, dos de los ejemplos son unos índices de calidad llamados Key Performance Indicators (en adelante KPI, dadas sus siglas en inglés) o el las estadísticas mensuales como uno llamado QC_mensual (lo llamo así para este proyecto).

La primera aplicación son métricas financieras o no financieras, que se usan para cuantificar el desempeño de una organización. Estos indicadores se usan, generalmente, para ver la situación actual de una empresa y marcan las acciones a realizar según lo que muestre la métrica. Cada una de estas acciones se conoce como tarea, que en Everis tienen un código diferente dependiendo del departamento en el que se trabaja. Cada tarea mide uno o varios KPIs.

La segunda aplicación y la que más nos interesa, porque de esa nació el proyecto, es la del QC_mensual. Como indica su nombre, los clientes piden a Everis mensualmente un informe con los números, cálculos y/o gráficos (en forma de estadística) que les interesa para ver el estado y rendimiento de sus aplicaciones. Estos cálculos dependen de la aplicación que se trate, pueden ser contabilizar el número total de mensajes enviados correctamente, o el número de mensajes fallidos o no recibidos hasta el número total de plantillas subidas en una máquina. Cuando se habla de plantillas se refiere a archivos PDF que se cargan en servidores o máquinas para hacer testear otras aplicaciones.

Entrando en el ejemplo de los mensajes enviados correctamente, hay un proyecto Java que se ejecuta cada mes para la generación de las estadísticas. Estas estadísticas están en formato Excel con diversas pestañas, en una de ellas hay sólo gráficos según los

intereses de los clientes. En el resto de pestañas están los datos que también sirven al cliente, estos datos se muestran diariamente por lo que siempre hay una columna que es **Fecha**. Además de la columna Fecha, otra de las columnas que también están siempre son el número total de mensajes enviados, el número mensajes enviados con error y el número de mensajes enviados correctamente, es decir, siempre hay un **Valor** que mostrar.

Después de estas columnas mencionadas, hay diversas columnas por las que se pueden ir filtrando los valores y según interés.

Estas columnas pueden ser la operadora por la cual se ha enviado los mensajes, el código de contrato que tiene el número al que se envía el mensaje o el tipo de mensaje que es (Entrada y que sólo recibe mensajes o Entrada-Salida, que también puede responder a los mensajes que recibe). A estos filtros y como veremos en el apartado de desarrollo, en este proyecto se les llamará **Clave**.

Además del proyecto Java existe una parte de base de datos, encargado de extraer los datos que nos interesan o que los Clientes han solicitado, y de realizar las sumas para la calcular los totales mencionados.

Un ejemplo de la extracción de este QC_mensual es el que podemos ver a continuación:

Fecha	Tipo	OperadorEnvio	msgOk	msgErr	TotalPeticiones
01/01/2016	Salida	Movistar	15	0	15
01/01/2016	Salida	Movistar	1	0	1
01/01/2016	Salida	Movistar	15	0	15
01/01/2016	Entrada-Salida	Movistar	1	0	1
01/01/2016	Salida	Netsize	0	12	12
01/01/2016	Entrada-Salida	Netsize	28	0	28
01/01/2016	Salida	Vodafone	0	12	12
01/01/2016	Entrada-Salida	Netsize	28	0	28

Tabla 2: Una de las pestañas del QC_mensual generado para la aplicación de mensajes enviados.

Como se puede ver en la Tabla 2, los filtros en este caso son el tipo de mensaje y el operador por el cual se envían. Los valores son el total de mensajes enviados correctamente, con error y la suma de los dos. Como estos filtros, la empresa de facturación también puede ser otro filtro, y se puede ver en la Figura 9.

Y los gráficos que se generan son del siguiente estilo:

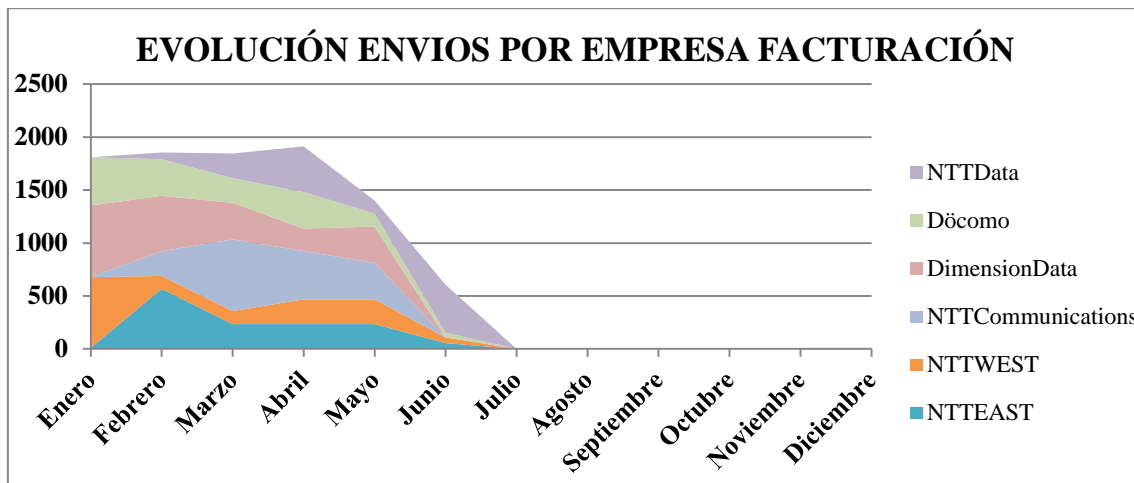


Figura 9: Ejemplo de gráfico generado por el QC_mensual de la aplicación de mensajes enviados.

Si ahora entramos en el ejemplo de las plantillas es el mismo procedimiento, pero como se trata de otra aplicación tiene otras características por las que se filtrará. En esta ocasión lo que se mira son las máquinas que hay en los diferentes entornos de desarrollo. En Everis y como normalmente funciona, hay tres entornos de desarrollo el de test (en adelante TST, dada la abreviación y como se conoce en Everis), el de preproducción (en adelante PRE, dada la abreviación y como se conoce en Everis) y finalmente, producción (en adelante PRO, dada la abreviación y como se conoce en Everis). Estos entornos como podemos imaginar según sugieren sus nombres se diferencian en función de la importancia de los datos con los que se estén trabajando. Los de PRO son los reales, los del Cliente directamente y los de TST son los de prueba. Para este proyecto, por ejemplo, se utilizarán los datos de TST y se harán todo tipo de pruebas con estas. Cada uno de estos entornos según la aplicación tiene unas máquinas u otras, la aplicación de las plantillas por ejemplo, tiene ocho entre los tres entornos.

Lo que se hace para extraer estadísticas de esta aplicación es calcular el tiempo medio que se tarda en subir una cantidad de plantillas en un determinado momento, el número total de plantillas subidas correctamente, el número de plantillas subidas con error y el número total de plantillas subidas. Como hay muchas máquinas lo más óptimo es ver estos cálculos por separado ya que, podría pasar que el error sólo estuviese en una máquina o que fuese un fallo general. En el Excel generado también hay una columna **Fecha** como pasaba en la anterior aplicación, además de las agrupaciones que hay por máquina. Para cada máquina hay un cálculo de tiempo medio, total plantillas enviadas, total plantillas enviadas correctamente y total plantillas enviadas con error. Todos estos datos representan el **Valor** y las **Claves** de este QC_mensual para esta aplicación.

Igual que pasaba con la anterior aplicación, también hay un trabajo en la base de datos para el cálculo de todos estos valores mencionados. Y un ejemplo de este lo podemos ver en la siguiente imagen:

Fecha	CENPRASR01			CENPRASR02		
	Total peticiones	NOK	Tiempo	Total peticiones	NOK	Tiempo
01/10/2015	7.786	0	69ms	8.196	8.196	28ms
02/10/2015	23.044	3	68ms	23.020	23.020	28ms
03/10/2015	23.049	7	70ms	23.512	23.512	29ms
04/10/2015	22.931	1	69ms	23.422	23.422	29ms
05/10/2015	22.933	0	70ms	22.404	752	104ms
06/10/2015	22.932	0	69ms	22.784	0	107ms
07/10/2015	22.920	0	70ms	22.811	1	105ms
08/10/2015	22.972	1	71ms	22.777	1	102ms

Tabla 3: Una de las pestañas del QC_mensual generado para la aplicación de subida de plantillas.

En este caso, como se puede ver en la tabla las máquinas son cenprsr01 y cenprsr02 y los valores que interesa mirar por cada uno son el total peticiones, que son las plantillas subidas, el número de plantillas subidas con error respecto el total de peticiones o NOK como en la tabla y el tiempo medio de subida o tiempo como en la Tabla 3.

Como estas estadísticas las hay de más tipos, todo depende de la aplicación de la cual se quiera generar los índices. Esta fue una de las razones más importantes por las que surgió la idea del proyecto. Desde AOD se buscaba la manera de mostrar estas estadísticas de manera más visual y sin la necesidad de tener que tener más de un proyecto para cada aplicación. De aquí **“Frontend de métricas”**.

Teóricamente en diseño de software se conoce como frontend la parte del software que interactúa con los usuarios. En la práctica y en este proyecto representa exactamente lo mismo, la representación gráfica de los índices o métricas de rendimiento. Esta estará en forma de aplicación web a disposición del usuario en cualquier momento.

3. La propuesta y el uso de las tecnologías

Para simplificar la propuesta del proyecto, se dividirá en tres partes. Estas partes siguen la metodología del trabajo explicada en apartados anteriores y son las que podemos ver a continuación:

- › La creación de la librería o capa de negocio.
- › La creación de la aplicación web o capa de presentación.
- › La creación de la base de datos o capa de datos.

Además de explicar estas tres partes también se explicarán otras características del proyecto y los conocimientos previos antes de entrar a la parte del desarrollo.

3.1 Modelo de trabajo actual

La situación actual es que hay diferentes aplicaciones que generan diferentes estadísticas. Todas estas estadísticas tienen características que sólo identifican a una aplicación en concreto. Pero también hay características que ya se han ido mencionando como **Fecha**, **Clave** y **Valor** que son comunes en todas.

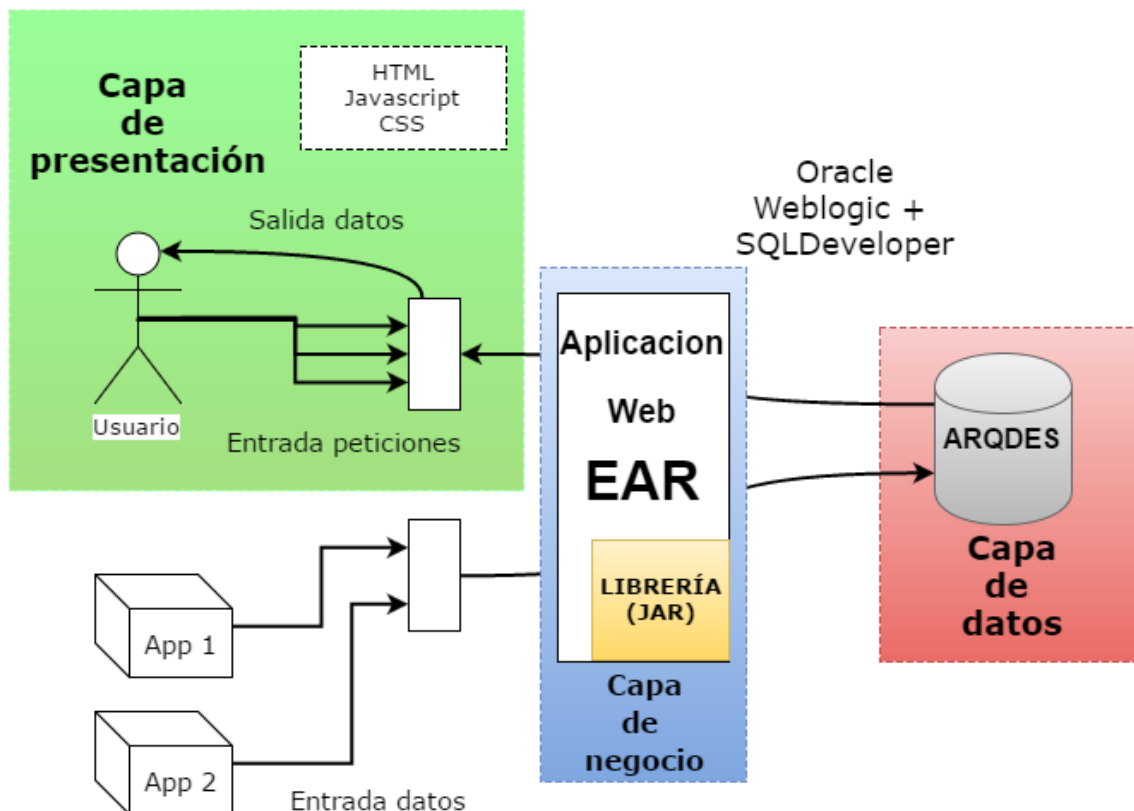


Figura 10: Diagrama de bloques relación tecnologías y partes del MVC.

Las características **Fecha**, **Clave** y **Valor** son las que han hecho pensar que este proyecto se puede presentar como un modelo unificado como el que se puede ver en la Figura 9.

Como se dijo en apartados anteriores, se centran todas las fuerzas en la fase de desarrollo del modelo unificado y para empezar a describir todas las partes implicadas en él, retomamos el diagrama del proyecto (Figura 9).

En los apartados de este punto de la memoria del proyecto, se hará un recorrido de este diagrama para identificar la parte que se está explicando y para justificar las tecnologías utilizadas en cada una.

Tal y como se puede ver en el diagrama (Figura 9), podemos identificar las tres capas que se tratarán en este apartado de la memoria. Con cada una de las secciones de este apartado y explicando las tres capas de nuestro modelo, se podrá tener una idea de la parte de desarrollo de la aplicación.

3.1.1 Generación de estadísticas

Ahora que ya se tiene idea de cómo se generan estas estadísticas, se puede intuir que el trabajo de este proyecto no consiste en generarlas sino en aprovechar de los proyectos que ya lo hacen y plasmarlos en una aplicación web.

Se sabe también que ya se cuenta con una base que son los gráficos generados a partir de los datos extraídos del Excel y por lo tanto, son los que servirán como muestra para el proyecto.

Como se ha dicho anteriormente, se ha optado por la idea de **Fecha**, **Clave** y **Valor** porque se buscaba la manera de poder relacionar cada una de las estadísticas extraídas de cada una de las aplicaciones. De esta forma, con la idea de estas tres características, en común para todas, se evitaba tener que desarrollar una aplicación web con lógicas diferentes para cada una (como pasa con la generación de estadísticas en Excel, actualmente).

La primera característica que se pudo identificar fue **Fecha**, porque obviamente es imprescindible para sacar cualquier estadística. La siguiente fue **Valor** porque para cada Fecha se observan ciertos números de los que luego se sacan conclusiones. Estos números en el caso de las aplicaciones en AOD, como se ha mencionado antes, pueden ser sumas o promedios, como el total de mensajes enviados o el tiempo medio al subir una plantilla.

Finalmente, la última en sacar fue **Clave**, cuando se vio que estos valores podían ser filtrados por criterios como el operador por el que se envían los mensajes. Un ejemplo aplicativo con estas tres características una vez encontradas, puede ser “¿Cuántos

mensajes (Valor) se han enviado en el **mes de Julio** (Fecha) por el operador **Vodafone** (Clave)”. Se ha de recordar que, como este ejemplo también puede servirnos otro más general como “¿Cuál ha sido la velocidad media (**Valor**) de la CPU de mi máquina (**Clave**) durante el último mes (**Fecha**)?”

Una vez se tuvo esta idea la propuesta del proyecto se puso en marcha.

3.1.2 Aspectos a mejorar

Como se ha dicho, lo que se trata ahora es de crear una aplicación web capaz de coger esas **Fecha**, **Clave** y **Valor** y mostrarlas en forma de gráficos. Si bien esto ya se hace pero en forma de Excel, ahora estos podrán estar a disposición de cualquier usuario y como una aplicación web.

Uno de los aspectos mejorados es que los datos que se van a mostrar, podrán ser filtrados en un rango de fechas que el mismo usuario podrá introducir. Por tanto, los datos son mucho más dinámicos, que los que generados hasta ahora.

Otro de los puntos que se mejora con la aplicación web de este proyecto es el dinamismo de los gráficos. Ahora se pueden visualizar de diferentes tipos en uno mismo y con alguna interacción en ellos.

3.2 Creando la librería

En este y en los siguientes apartados de este punto de la memoria de este proyecto, hablaremos de las tecnologías utilizadas en cada una de las partes del desarrollo del proyecto. Además se justificará la elección de estas tecnologías en lugar de otras.

Con lo que ya se sabe hasta ahora, ya se conocen las tecnologías con las que se desarrollan cada una de las capas del proyecto. En este apartado concretamente se hablará de la tecnología utilizada en la capa de negocio. Capa encargada de la comunicación entre las capas de presentación y datos.

Esta librería será capaz de proporcionar diferentes funcionalidades a la capa de presentación, para que esta muestre los resultados de la manera más atractiva posible. Para crear una librería con estas características y que realice estas funcionalidades de la manera más simple posible, se ha elegido Java.

A esta parte la llamamos librería ya que, de esta se encargará el Java Archive (JAR en adelante, dadas sus siglas en inglés) generado por nuestro proyecto, que ya explicaremos en su momento en la parte de desarrollo. Este JAR no es nada más que el proyecto Java donde, como se verá en la etapa de desarrollo, podemos encontrar todos los cálculos para el funcionamiento de las otras dos capas (la de presentación y la de datos). Aquí se encuentra la lógica del proyecto.

3.2.1 Elección del lenguaje de programación (JAVA)

La compañía ha decidido desarrollar esta aplicación web en Java. Java es el lenguaje de programación con el que cualquiera, que empiece en Everis, puede encontrarse, es parte del entorno de trabajo.

Java es un lenguaje de programación orientado a objetos (usa objetos en sus interacciones, para diseñar aplicaciones y programas informáticos [20]). Se basa en los conceptos de clases, herencia y polimorfismo. Una de las ventajas principales es que ofrece un lenguaje multiplataforma (desde portátiles hasta centros de datos, desde consolas para juegos hasta súper ordenadores, desde teléfonos móviles hasta Internet, Java está en todas partes) y permite la posibilidad de ejecutar una aplicación en diferentes sistemas operativos [19].

Se trata de un lenguaje que tiene que ser compilado e interpretado. Todo programa Java ha de compilarse y una vez compilado, es interpretado por una máquina virtual. De esta manera es como consigue la independencia de la arquitectura.

Java es la base para la mayoría de las aplicaciones, además es el estándar global para desarrollar y distribuir aplicaciones móviles, juegos, contenido basado en web y/o software de empresa. Con más de 9 millones de desarrolladores en todo el mundo, Java permite desarrollar, implementar y utilizar de forma eficaz interesantes aplicaciones y servicios [21].

Y si queremos ver, un poco más, cuánto está presente Java en nuestro día a día, sólo tenemos que mirar las siguientes estadísticas [21]:

- › Con una penetración del 97% Java se ejecuta en los entornos empresariales.
- › El 89% de los ordenadores en Estados Unidos ejecutan Java.
- › Java es la primera plataforma de desarrollo.
- › 3 mil millones de teléfonos móviles ejecutan Java.
- › Y 125 millones de dispositivos de televisión ejecutan Java.

Además de todas estas características y como remarcaremos en la parte de justificación de la elección, Java facilita el trabajo al programador, realiza comprobaciones durante la compilación, se encarga de liberar la memoria ocupada por objetos que no están referenciados y elimina el uso de puntero en la programación (una fuente muy común de errores, a través de referencias) [22].

3.2.2 Funcionalidades

La librería debe ser una capa invisible para el usuario. Debe realizar determinadas funciones y devolver información a la capa de web o presentación y así, se da abstracción a la capa física.

Antes de poder empezar con la lógica de la librería, el primer paso será conectarse a **ARQDES**, la base de datos en la que se almacenan todas las instancias que nuestra aplicación mostrará en los gráficos. Una vez conectados a la base de datos, lo siguiente será sacar todos los datos que hay hasta el momento. Para conectar con ARQDES, se ha decidido utilizar una conexión mediante un **servicio web**, en nuestro caso en la plataforma de **Oracle WebLogic** junto con **Spring Framework** y **Oracle SQLDeveloper** para la conexión a base de datos. Un servicio web (Web Service o Web Services) es un sistema de software que permite el intercambio de datos y funcionalidad entre aplicaciones sobre una red [23].

“Spring Framework es una plataforma Java que proporciona un amplio soporte de infraestructura para el desarrollo de aplicaciones Java” (Documentación Spring 2011 [24]).

“Una base de datos Oracle es una recopilación de datos tratados como una unidad. El propósito de estas es almacenar y recuperar información relacionada una con otra. Un servidor de base de datos es la clave para resolver los problemas de gestión de la información. En general, un servidor gestiona de forma fiable una gran cantidad de datos en un entorno multiusuario para que muchos usuarios puedan acceder simultáneamente a los mismos datos. Sin que muchos usuarios supongan una bajada del rendimiento” (Documentación Oracle 2016 [25]).

Entonces, cuando ya haya una conexión a ARQDES, lo que toca es recoger toda la información necesaria de las aplicaciones a las que les interesa nuestra aplicación web. Para hacerlo estas aplicaciones nos envían la información (recogemos cada una de las Fecha, Clave y Valor que tengan hasta el momento) mediante una entrada de datos o formulario que se ha diseñado, como una de las páginas de la aplicación web. Desde esta página todos los datos introducidos van directamente a base de datos para ser almacenados en tablas que previamente se han creado. En este proyecto hablaremos de los dos QC_mensuales que ya se han explicado antes por eso, nos centraremos en los datos de estas dos aplicaciones. Aunque se tiene que recordar que la aplicación web puede extenderse y aplicarse a otras del mismo estilo o como se dijo al principio, se puede mirar el rendimiento de la CPU del ordenador de casa.

Aunque ya se explicará en detalle más adelante, la entrada de datos que harán estas aplicaciones interesadas estará controlada por el desarrollador (en este caso mi persona).

Sólo será posible si estas aplicaciones están dadas de alta en base de datos. Dadas de alta significa que la Clave de estas se encuentra en una de las tablas que han sido creadas precisamente para que conste como parte de la aplicación web. Otra cosa a remarcar es que, estas aplicaciones están dadas de alta porque a algún usuario le interesa o le ha interesado ver algún gráfico de rendimiento en un momento determinado.

Como es de suponer antes se tiene que crear el modelo de base de datos, con todas las tablas y todas las relaciones que requiera la aplicación web para su funcionamiento.

El cálculo y las extracciones de Fecha, Clave y Valor se hacen mediante procedimientos de la base de datos que serán invocados por la librería. Estos cálculos dependerán de dos parámetros de entrada, que el usuario introducirá, la Fecha Inicio y Fecha Fin. Estos parámetros lo que determinarán es la cantidad de información (sólo un rango de fechas) que quieren ver plasmado en los gráficos.

Una vez todos los cálculos están efectuados como interesan, son pasados a una serie de objetos dentro de la misma librería que, luego nos servirán para mostrar cada uno de los gráficos que queremos. A estos objetos se los conocen como beans. Los beans son un modelo de componentes muy útiles en aplicaciones Java. Estos beans se usan para encapsular varios objetos (en el caso del proyecto todas las instancias que se tengan de Fecha, Clave y Valor) en un único objeto [26]. De esta forma podemos trabajar con Fecha, Clave y Valor como parte de un solo objeto en lugar de por separado y por ejemplo, no saber qué fecha se tenía datos sobre esa Clave. Estos beans junto con otros que ya veremos en la parte de desarrollo son paquetes base de Spring Framework que serán muy útiles en este proyecto.

Ahora que ya tenemos toda la información necesaria sólo queda pasar estos datos a cada uno de los gráficos y que se explicará en la siguiente parte de este punto de la memoria.

Entonces de manera resumida, se puede decir que la librería tiene tres funcionalidades principales:

- › **Actualizar los datos:** conectándose al servicio web de la aplicación y a ARQDES para ir almacenando la información nueva de las Claves de las aplicaciones que están dadas de alta en la base de datos.

- › **Recoger el rango de Fecha Inicio y Fecha Fin:** conectándose al servicio web, recogiendo el valor que los usuarios han introducido y pasando estos valores a los procedimientos de base de datos.

- › **Calcular cada uno de los valores necesarios:** conectándose a base de datos, ARQDES, y calculando los Valores para cada una de las Claves ya que, no todas se calculan de la misma forma. Una vez se tengan estos valores se recopilarán y

guardarán en los beans para que pueda ser leído por la capa de presentación (en nuestro caso por los gráficos).

Esto se puede resumir, retomando la imagen del MVC, de la siguiente manera:

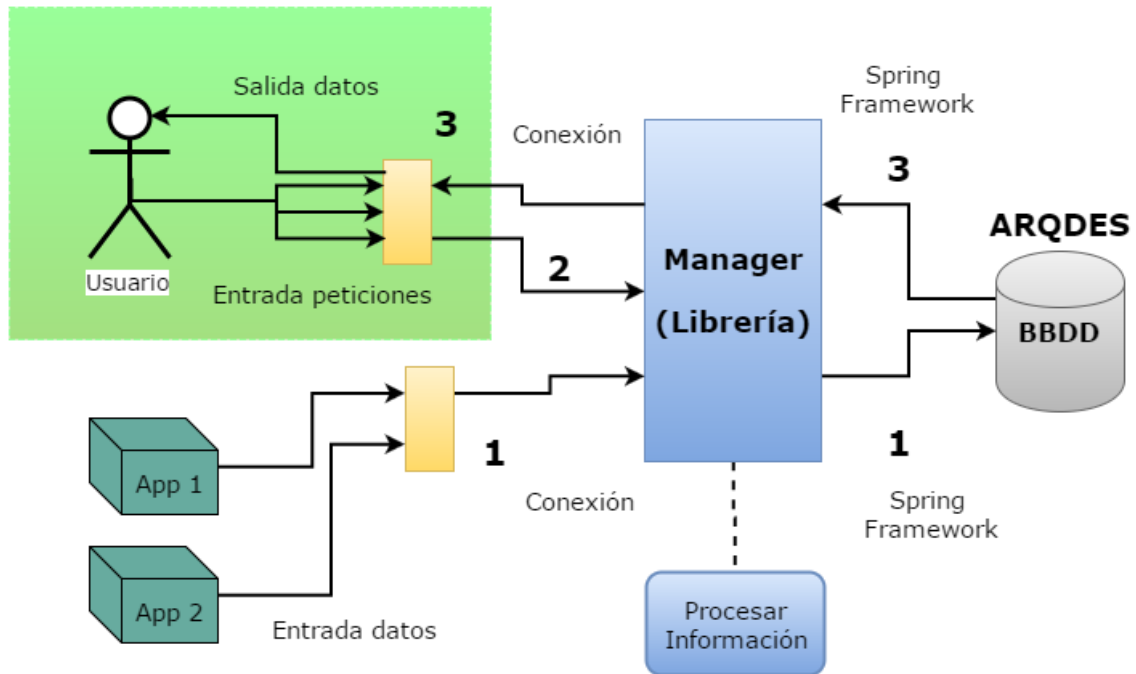


Figura 11: Esquema funcionalidades de la librería.

Tal y como se comentará en la parte de desarrollo, la librería es capaz de separar la recuperación de datos de la visualización o parte que ve el usuario. Por este motivo, los puntos 1 y 2-3 de la imagen superior (Figura 10) se entienden como dos **adaptadores** diferentes (una procesa la entrada de datos y la otra, los datos hacia los usuarios).

Estando en este punto ya sabemos cómo funcionará esta parte del proyecto y como bien se dijo aquí se encuentra la lógica de la aplicación web.

3.2.3 Justificación de la elección

Aunque la lógica del proyecto no sea tan compleja, las funcionalidades son varias y por esto, se necesitaba un lenguaje de programación sencillo y a la vez completo, uno como Java.

Además de ver la penetración que Java tiene en nuestro entorno diario, mirando en diversas fuentes de información se ha obtenido el rating que se puede ver en la imagen inferior (Figura 11). En este rating, obtenido por TIOBE, se puede ver que los lenguajes de programación más utilizados en Mayo de 2016 han sido: Java, C, C++, C# y Python. TIOBE es una aplicación web indicadora de la popularidad de los lenguajes de programación [27]. Cogiendo este rating se mirarán los puntos débiles de otros

lenguajes en comparación con Java y de estos sacaremos los motivos por los que se escogió Java y no otro para este proyecto.

- › **C:** este es un lenguaje de programación que pese a la popularidad que tiene, carece de soporte para la programación a objetos y por lo tanto, no es útil para el proyecto.

- › **C++:** esta es la versión combinada de la flexibilidad y simplicidad del lenguaje de programación C que permite la manipulación de objetos (no es del todo orientado a objetos). Una de las causas, aún más importantes por las que no se ha decidido por este lenguaje ha sido la escasa fuente para proveer librerías que faciliten al programador. Aunque es cierto que C++ es mucho más rápido que Java en tiempo de compilación. Esto es debido a que antes de que un proyecto Java pueda ser compilado primero, todas las clases de este proyecto tienen que ser cargados (rapidez que se vuelve insignificante cuando el proyecto es uno de gran magnitud como es el caso) [28].

- › **C#:** es un lenguaje derivado de C/C++, es similar a Java y también es orientado a objetos. Es la versión mejorada de los antecesores y con la productividad que posee Java. Es flexible como C++, tiene una librería de clases muy completa y bien diseñada y se trata de un lenguaje orientado a objetos puro igual que Java. El tratamiento de muchas características tales como las excepciones es similar y se consideran dos lenguajes de programación igual de seguros. C# no es un lenguaje usado en el entorno de trabajo y este fue el principal motivo por el que no se optó por este [29].

- › **Python:** es un lenguaje de programación que soporta, entre otros, la programación orientada a objetos. Se trata de un lenguaje sencillo, moderno, y potente [30]. Es cierto que Java y Python tienen un mismo propósito pero, se trata de dos lenguajes de programación diferentes son eficientes a niveles diferentes. El problema como pasaba antes, es que no está muy presente en proyectos realizados por AOD.

De estos lenguajes tenemos la siguiente imagen, donde se puede ver el porcentaje de uso que tienen los lenguajes de programación durante el mes de Mayo este 2016 [27].

Porcentaje Rating Lenguajes de Programación

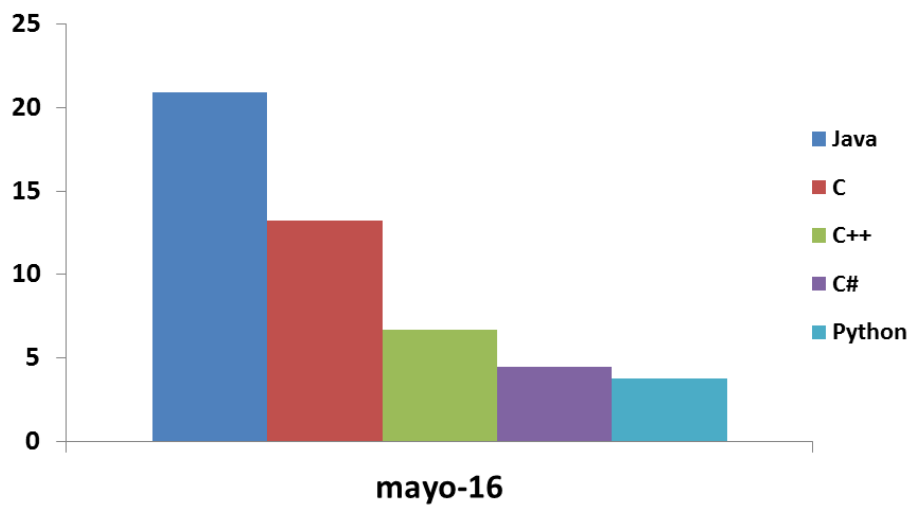


Figura 12: Rating uso lenguajes de programación Mayo 2016.

3.3 Creando la aplicación web

Con este apartado pasamos a la capa de presentación, capa responsable de proporcionar toda la información al usuario de manera que pueda ser entendida, una vez mostrada en la pantalla.

Para tal fin y como se verá en una imagen a final de este apartado (Figura 12), se desarrolló una aplicación web capaz de usar la librería explicada en el apartado anterior. Esta aplicación está diseñada para funcionar en un servidor dentro de la red de la empresa y podría adherirse a ella por una URL. Pero para este proyecto las pruebas y resultados se harán en localhost (se ejecutará en el host del propio equipo u ordenador). El siguiente paso después del proyecto será poner la aplicación en el servidor de la empresa porque de esa manera, podrá ser utilizado por todos los empleados. El servidor local será un servidor de Oracle WebLogic.

“Oracle WebLogic Server es un servidor de aplicaciones escalable, es una plataforma de Java lista para empresas, de la plataforma Enterprise Edition (Java EE) [31]. Java EE es el estándar en software empresarial más popular en la comunidad para el desarrollo de aplicaciones web y tiene varias especificaciones de Application Programming Interface (en adelante API, dadas sus siglas en inglés) tales como Servicios Web (la que nos interesa para este proyecto)” (Documentación Oracle 2016 [32]). “Una API es el conjunto de funciones, procedimientos o métodos que ofrece una librería que será usada por otro software (la API del Web Service en este caso, que será usada por Java)” (Wikipedia 2016 [33]).

Esta aplicación web, tal y como se introdujo en apartados anteriores de la memoria, se almacena en un archivo EAR que será ejecutado desde el servidor WebLogic.

“Un EAR es un formato de archivo JAR que se utiliza para empaquetar módulos como archivos de almacenamiento individuales, para asegurar el despliegue coherente de los diferentes módulos a servidores de aplicaciones. EAR utiliza ficheros Extensible Markup Language (en adelante XML, dadas sus siglas en inglés), o descriptores de despliegue, para describir la implementación del módulo” (Documentación Techopedia 2016 [34]).

La capa de presentación comprende toda la aplicación web y es la única capa del desarrollo que el usuario verá. De manera resumida, es la parte en la que definimos las representaciones de cada uno de los gráficos. Para el desarrollo de la capa de web los lenguajes que se han elegido han sido: Javascript, HTML, CSS y d3.

Retomando el diagrama del MVC, este apartado se puede representar de la siguiente manera:

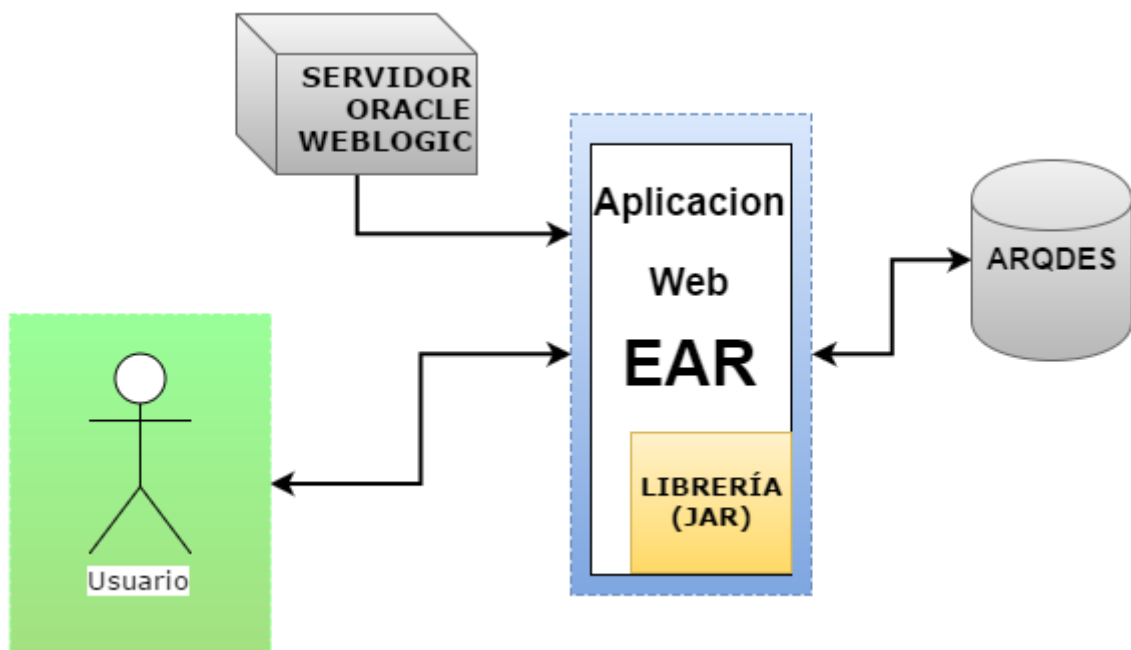


Figura 13: Esquema servidor WebLogic y aplicación web.

3.3.1 Elección de las tecnologías para la programación

En este lado del proyecto la elección de estas tecnologías no ha sido una tarea difícil ya que, la mayoría de las aplicaciones web están constituidas, de una manera o de otra, sobre estas.

Empecemos por el lenguaje principal y sobre el que utilizaremos el resto de tecnologías, este es el tan popular HTML. Lenguaje que se emplea para el desarrollo de páginas de Internet o páginas web, como las conocemos en estos días.

“HTML está basado en un lenguaje de marcado o maquetación, lenguaje que se caracteriza por códigos y etiquetas capaces de definir contenido en una página web. Contenido en forma de texto, imágenes, gráficos, videos, juegos y demás” (Enrique Gonzales, 2016 [35]).

Desde la aparición de la WWW, HTML se ha convertido en el estándar para la visualización de las páginas web y hoy en día está presente en las más de mil millones webs ya existentes en el año 2015 [36].

Es cierto que HTML no podía ser tan perfecto y por eso, con el paso del tiempo las tecnologías como Javascript fueron complementándolo para hacerlo más completo. De esta manera, funcionalidades que se veían limitadas, con Javascript eran posibles. Por tanto, Javascript se comporta como un lenguaje de programación auxiliar que puede ir dentro del HTML. Esta es una técnica conocida como la entremezcla de lenguajes entre sí, para resolver problemas o implementar funcionalidades que con una sola no eran posibles.

“Javascript es un lenguaje de programación interpretado, orientado a objetos, basado en prototipos y dinámico. Se utiliza principalmente como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas” (Documentación Mozilla 2015 [37]).

La sintaxis de Javascript es muy similar a C, C++ y Java, permite a los navegadores tomar decisiones y procesar información mediante diferentes cálculos o métodos lógicos. Estas características, son la clave para que los gráficos que se vayan a mostrar puedan ser los más interactivos y dinámicos posibles (que puedan cambiar la Fecha rápidamente, por ejemplo).

De la misma forma que Javascript surgió como complemento para ayudar a mejorar las páginas web HTML, el CSS es la respuesta a la necesidad que había de dar un poco de forma o diseño (por ejemplo, dándole color, fuente a las letras o medida a los objetos) a las páginas web.

“CSS es un lenguaje usado para definir y crear la presentación de un documento estructurado escrito en HTML o XML. Se trata de una información de estilo que se puede encontrar dentro del mismo HTML o en un documento separado. Si se encuentra dentro, se puede reconocer por el atributo <<style>> en la etiqueta donde se ha aplicado” (Documentación W3schools 2016 [38]).

De forma resumida y recopilando lo que se ha dicho hasta ahora sobre las tecnologías de la capa de web, tendremos páginas web donde el usuario podrá ver los gráficos y estas estarán escritas sobre HTML usando Javascript y CSS.

A pesar de que Javascript es el lenguaje más importante de la capa de web o mejor dicho, la tecnología realmente fundamental para que la representación de los gráficos sea posible es d3.js.

“D3.js es una librería de Javascript para producir, a partir de datos, gráficos dinámicos e interactivos en navegadores web y hace uso de tecnologías como HTML, SVG y CSS” (Bostock, M. 2015 [39]).

Aprovechando el mismo HTML, Javascript y CSS se utilizará el Framework bootstrap para el diseño visual de la página web. **Bootstrap** hace que el desarrollo sea más fácil y rápido porque proporciona diferentes plantillas que luego se acomodan a la estructura del proyecto.

Finalmente otra herramienta más que tecnología implicada en el desarrollo de esta parte del proyecto es **Balsamiq**. Una herramienta que como se verá en la segunda parte de la memoria sirve para una primera construcción de la aplicación.

3.3.2 Funcionalidades

Si bien es cierto que HTML no es un lenguaje como los que conocemos o como los que hemos mencionado (el mismo Java de la librería). La verdad es que gracias a HTML podemos utilizar otros como Javascript o CSS, para poder implementar una cantidad ilimitada (las que se le ocurra a cada programador) de funcionalidades.

En el caso de este proyecto utilizaremos el HTML para crear la estructura, base o esqueleto de cada una de las páginas con las que contará la aplicación web. Con este lenguaje únicamente daremos el “formato” a las páginas que utilizaremos para mostrar los gráficos. Además de dar estructura a las páginas, con HTML permitiremos al usuario poder filtrar el rango de las fechas (Fecha Inicio y Fecha Fin) en las que quiere ver los gráficos.

Javascript por su lado nos servirá para la recopilación de los datos extraídos en la librería, para mostrar los gráficos con la librería d3 y para dar estilo a las páginas con el CSS.

Estos datos recopilados que Javascript almacenará se pasarán a objetos del d3 para representarlos como lo pide cada gráfico. Por lo tanto, es la librería d3 la que se encarga de cada detalle que tenga que ver con los gráficos. Desde el tamaño de los ejes hasta el color de las líneas, el color del relleno de las barras del histograma, hasta la posición de cada uno de las Fecha y Valor.

Realmente el trabajo que la tecnología d3 ha proporcionado a esta parte del proyecto ha sido trascendental porque ha ahorrado mucho tiempo y ha hecho que los gráficos puedan ser lo más dinámicos posibles.

Como se ha mencionado antes, todo lo que tenga que ver con el estilo de los gráficos y que el d3 hace uso en sus funcionalidades, estarán a cargo del CSS.

3.3.3 Justificación de la elección

No hay mucho que justificar por el lado del HTML, Javascript y CSS porque como se ha dicho en el punto anterior, hay más de mil millones de páginas web constituidas sobre HTML y propiedades como Javascript y CSS.

Por lo que, queda justificar la elección del d3. No ha sido una elección muy difícil porque tampoco había muchas opciones. Se tenía que buscar una tecnología compatible con Javascript y HTML porque estas ya formaban parte del proyecto. Entonces se pensó en buscar una librería que pudiese ser lo más flexible posible.

Los resultados eran infinitos pero, desde el inicio del proyecto siempre se ha tenido una idea o pauta que se buscaba. Este modelo es una aplicación web que también sirve para la representación de gráficos y es de software libre: **dashing.io**. Se conoce **dashing.io** porque es utilizada por Everis para el seguimiento de sus máquinas. Por eso, de la misma manera que a la empresa le interesa saber el estado del rendimiento de su propio sistema le puede interesar al cliente. De ahí, la idea de este proyecto.

Como podemos ver en la Tabla 4, la clara diferencia se encontraba en la capacidad que tenían para ofrecernos variedad de gráficos y flexibilidad en los gráficos. Por eso, las tecnologías que ofrecían gráficos de tipo social (**Sigma.js**), esos que sirven para pintar relaciones entre nodos no eran útiles para el proyecto. Otra de las tecnologías que tampoco podían servirnos de mucha ayuda eran las tecnologías que sólo contaban con representaciones estáticas como eran el caso de las librerías **C3.js**, **Vis.js** o **xCharts**.

De los otros tres que quedaban por elegir:

- › **D3.js**: ofrece una amplia variedad de funcionalidades para la interactividad con los gráficos, son flexibles y dinámicos.
- › **Canvas.js**: aunque se tratan de gráficos no estáticos son simples y poco moldeables.
- › **Chartist.js**: es quizás la que más se parece al d3 porque está basada en animaciones.

Para poder hacer un pequeño resumen de las librerías que se han encontrado disponibles y hacer la diferencia respecto d3.js se ha construido la siguiente tabla:

	D3.js	C3.js	Vis.js	Sigma.js	Canvas.js	xCharts	Chartist.js
Propiedad 1	Gráficos de todo tipo.	Gráficos no dinámicos	Gráficos 3D	Gráficos sociales o lineales	Gráficos lineales	Gráficos estáticos o no dinámicos	Gráficos dinámicos
Propiedad 2	Gráficos dinámicos y no dinámicos	Gráficos sencillos pero sutiles	Gráficos no dinámicos	Gráficos poco comunes, no como barras	Gráficos dinámicos pero, muy típicos o clásicos	Gráficos que sólo informan	Donuts dinámicos que informan

Tabla 4: Comparación tecnologías similares a d3.js

Para el proyecto se prefirió d3.js por todo su contenido.

3.4 Creando el modelo de base de datos

Finalmente, toca hablar de la última parte, la capa de datos. Antes de explicarla es importante remarcar que esta separación se hace únicamente para una explicación un poco más profunda de las partes del desarrollo. No encontramos la capa de datos separada del resto porque está implementada dentro de la librería, tal y como se ha comentado. Simplemente se está siguiendo la estructura de la metodología del trabajo, explicada en uno de los apartados de la memoria de este proyecto.

Lo que se puede decir de la creación de esta capa es que, se ha optado por la tecnología más apropiada y completa para realizar las diferentes funciones de la manera más rápida posible. Esta tecnología de desarrollo, como a estas alturas ya se sabe, es el Oracle SQLDeveloper.

“Oracle SQLDeveloper es un entorno de desarrollo integrado libre que simplifica el desarrollo y gestión de base de datos Oracle en ambas implementaciones tradicionales y de nube. SQLDeveloper ofrece un desarrollo completo de extremo a extremo de las aplicaciones PL/SQL, una hoja de cálculo para ejecutar consultas y scripts, una consola de DBA (Database Administration) para la gestión de la base de datos, una interfaz de informes, una solución completa de modelado de datos, y una plataforma de migración para mover su base de datos de terceras partes a Oracle” (Documentación Oracle 2016 [40]).

3.4.1 Elección de las tecnologías para la programación

SQLDeveloper es la plataforma de trabajo más importante, incluso antes que Java, en el entorno de Everis. Se trata de una compañía donde se trabaja con todo tipo de datos y es necesario un entorno de desarrollo capaz de soportar la mayor cantidad de datos sin que esto afecte al rendimiento de sus funciones. Por esta misma razón, no era necesario buscar otra tecnología ya que, se contaba con esta que, ya da buenos resultados.

Tal y como lo definen las fuentes, SQLDeveloper ofrece muchísimas formas de poder trabajar con un número infinito de datos y como sugiere el nombre, está basado en el lenguaje de programación SQL. En este proyecto, se ha utilizado la forma más sencilla de manejar las diferentes instancias que debemos almacenar, actualizar o calcular. A este método se le conoce con el nombre de query.

Una query es una consulta que se hace en lenguaje SQL desde una consola del SQLDeveloper o desde otra librería como Java (en el caso del proyecto) sobre una base de datos. Esta base de datos está compuesta por las tablas con las columnas y filas que interesen o con las que se han decidido. Cada una de las consultas que se hacen sirve para recuperar datos, para hacer operaciones sobre alguno de los campos de la tabla o para añadir nuevas instancias en alguna de estas tablas.

3.4.2 Funcionalidades

En cuanto a las funcionalidades de esta capa, ya se ha ido explicando en la parte de la capa de librería pero, las retomaremos para entrar en detalle. De todas maneras, la forma en la que se han implementado ya se verán en la parte de desarrollo. Por lo tanto y de manera resumida, la capa de datos está presente en el proyecto de dos formas:

- › Por un lado, cuando se tienen que realizar las conexiones a la base de datos.
- › Y por el otro lado, cuando se tienen que realizar cada una de las consultas, con las que se extraerán los registros que se necesitan procesar en la librería. Una vez procesados, serán pasados a la capa de presentación para finalmente, mostrarlos en forma de gráficos.

Para poder realizar la conexión a la base de datos, ARQDES en el caso del proyecto, participa también el servidor de Oracle WebLogic. La razón por la que esto pasa es porque, tanto la aplicación web (con el EAR del proyecto) como la base de datos (ARQDES) tienen que estar alojados en el mismo servidor WebLogic para poder ejecutarse. La configuración que hay que hacer para la parte del despliegue de la aplicación web se hace desde Java pero, la configuración para la conexión a base de datos se hace de forma manual.

Esta configuración manual se hace directamente desde la consola que WebLogic tiene disponible en el localhost <http://localhost:7001/console> cuando se instala. Allí se busca en servicios la opción Java Database Connectivity (en adelante JDBC, dadas sus siglas en inglés). Una vez en esta opción, se selecciona Orígenes de Datos que, como informa la propia consola se trata un objeto enlazado a un árbol de Java Naming and Directory Interface (en adelante JNDI, dadas sus siglas en inglés) que proporciona conectividad a base de datos a través de un pool de conexiones JDBC. Dentro de Orígenes de Datos es donde con todas las propiedades de conexión, añadiremos la base de datos ARQDES.

“Un JDBC es el estándar de la industria para la conectividad de base de datos independiente entre el lenguaje de programación Java y la base de datos SQL propia. La tecnología JDBC permite utilizar el lenguaje de programación Java para explotar "escribir una vez, ejecutar en cualquier lugar" capacidades para aplicaciones que requieren acceso a los datos empresariales. Con un controlador compatible con la tecnología JDBC, puede conectar todos los datos corporativos, incluso en un entorno heterogéneo” (Documentación Oracle 2014 [41]).

Con esto sacado de las fuentes de Oracle sabemos que, este JDBC será independiente de la base de datos que hay en la plataforma SQLDeveloper y que el controlador compatible con el JDBC es el EAR de la aplicación web (como ya se había ido introduciendo anteriormente).

El árbol JNDI es nada más que la nomenclatura con la que se accederá a esta base de datos desde Java [42].

Finalmente, la otra funcionalidad de esta capa de datos ya es sencilla después de tener la conexión a base de datos. Para almacenar cada una de las entradas, actualizar o recopilar cada uno de los registros que hay en Oracle SQLDeveloper, se hace uso de las queries antes explicadas. Cada una de estas consultas a base de datos se encuentra en la librería y se ejecutan cada vez que se necesitan. Con estas por ejemplo, se recuperarán todas las Fecha, Clave y Valor para la Fecha Inicio y Fecha Fin que el usuario ha indicado desde la aplicación web.

3.4.3 Justificación de la elección

Igual que pasaba con Java, SQL era un lenguaje de programación cerrado para este proyecto. Sin duda la elección es obvia, porque de todas las páginas web que haya actualmente, si alguna hace uso de algún tipo de almacenamiento de datos, seguro que la usa. Y si no es exactamente SQL es una versión moderna de esta.

El lenguaje SQL tiene diferentes versiones y dependiendo del entorno de desarrollo que se elija, la sintaxis del lenguaje puede variar o no. Por esta razón, la comparación y en parte pequeña justificación que se puede hacer, es la del motivo por la que se elige SQLDeveloper y no otra.

Característica	Microsoft SQL Server	Oracle	PostgreSQL
Descripción	Es un sistema de gestión de base de datos relacional RDBMS basado en SQL [44].	Es un sistema de gestión de base de datos relacional RDBMS basado en SQL [40].	Es un sistema de gestión de base de datos relacional de objetos ORDBMS basados en POSTGRES [45].
Licencia	Licencia comercial con versión libre disponible.	Licencia comercial con versión libre disponible.	Open Source con licencia BSD (Berkeley Software Distribution).
Sistemas Operativos donde pueden ser utilizados	Windows.	AIX, HP-UX, Linux, OS X, Solaris, Windows , z/OS.	FreeBSD, HP-UX, Linux, NetBSD, OpenBSD, OS X, Solaris, Unix, Windows.
APIs y otros métodos de acceso	OLE DB, Tabular Data, Stream (TDS), ADO.NET, JDBC , ODBC.	ODP.NET, Oracle Call Interface (OCI), JDBC , ODBC.	Native C library streaming API for large objects ADO.NET, JDBC ODBC.
Lenguajes de programación soportados	.Net, Java , PHP, Python, Ruby, Visual Basic.	C, C#, C++, Clojure, Cobol, Eiffel, Erlang, Fortran, Groovy, Haskell, Java , JavaScript , Lisp, Objective C, OCaml, Perl, PHP, Python, R, Ruby, Scala, Tcl, Visual Basic.	.Net, C, C++, Java , Perl, Python, Tcl.

Tabla 5: Comparación de los diferentes sistemas para la gestión de datos.

Si buscamos plataformas que ofrecen las mismas funcionalidades, que tengan las mismas o características similares a SQLDeveloper de Oracle, son pocas las que realmente cumplen. Una de las que está casi a la par y que también está presente aunque, con menor frecuencia en Everis, es PostgreSQL. Otra aunque no está presente

en el entorno de trabajo y que yo como developer del proyecto desconozco es el servidor SQL de Microsoft.

Realmente no hay un motivo por el cual rechazar ninguna de las otras dos ya que, se ha elegido SQLDeveloper de Oracle por comodidad. De todas maneras, para conocer mejor el resto de tecnologías tenemos la Tabla 5 [43] donde, se puede ver que cualquiera podría haber servido para este proyecto.

Si se ve la Tabla 5 comparativa a modo resumen de las características de estas tres plataformas para la gestión de base de datos, se puede llegar a la conclusión que, las propiedades que interesan al proyecto pueden ser satisfechas por todas.

Sin más que decir, se eligió la base de datos Oracle, con la plataforma SQLDeveloper, por comodidad y porque yo como responsable de este proyecto tenía más conocimiento. Más conocimiento implica siempre, menos dificultad a la hora de resolver problemas.

3.5 Otras características

Hasta ahora, en las secciones anteriores se han ido definiendo los objetivos principales y las funcionalidades de cada una de las capas de este proyecto. Pero además de estos, desde AOD siempre se busca más allá de una aplicación web, por eso se ha pensado que se debe optar por otras funcionalidades que le proporcionarían valor añadido.

Es cierto que lo que interesa es que la aplicación web funcione y que el usuario y cliente estén satisfechos con los resultados pero, para que este trabajo sea fácil, sostenible y extensible en un futuro se tiene que mirar más allá.

Para conseguirlo, la aplicación tendrá una funcionalidad muy interesante y muy presente en cualquier desarrollo que se realiza en AOD, el sistema de logs.

Además de este sistema de logs que se implementará en este proyecto, existe otra funcionalidad también particular en la mayoría de las aplicaciones que se desarrollan en AOD. Esta última es la del fichero de properties. Para este proyecto no se usa porque su funcionalidad está integrada en la creación de la librería y ya no hace falta. Este fichero de properties se usa principalmente en proyectos Java para almacenar parámetros configurables de una aplicación. Lo más típico son las consultas o queries que se hacen a base de datos.

3.5.1 Sistema de logs

Un sistema de logs permitirá durante y después del desarrollo de la aplicación poder tener un registro oficial de los eventos durante un intervalo de tiempo determinado. Son información o mensajes claros sobre dónde, qué, cuándo y porqué se ha producido un evento dentro de la aplicación. Se entiende como evento, en este proyecto y como programadores, cualquier proceso dentro de la aplicación que está presente en el momento de la ejecución.

Este sistema es importante sobre todo durante el desarrollo de la aplicación ya que, se hace notar en el momento de la detección de errores entre procesos del código. El uso de este sistema de logs se hace dentro del proyecto Java con una invocación y con la nomenclatura que pide. Funciona de manera muy similar al mostrar por pantalla que nos proporciona la consola de Java porque, se usa donde se quiera inspeccionar y sólo se necesita ejecutar el proyecto Java. En este proyecto, como se puede imaginar, no nos sirve este mostrar por pantalla porque se trata de un entorno de desarrollo de aplicaciones y no de un simple proyecto Java.

De todas maneras, hay una diferencia entre ambas y es que con el sistema de logs, una vez ejecutado el proyecto se genera un fichero (si no está creado) con un nombre que se indica previamente. Este fichero se actualiza en cada ejecución con cada uno de los logs que se vayan añadiendo o cada vez que se quiera verificar resultados.

Con esto, evitamos tener que debuggear (ir punto por punto) cuando haya un error en el código y no se sepa exactamente dónde. Además, se puede ir verificando cada vez que haya un cambio si los resultados siguen siendo los que tienen que ser.

4. Conocimiento

Con este punto llegamos al final de esta primera parte de la memoria de este proyecto. En este momento, se espera que el que esté leyendo esta memoria, tenga alguna idea de cómo va a ser el desarrollo de la aplicación web. Se conocen las funcionalidades de cada una de las capas que componen el proyecto y de cada una las tecnologías que permiten su desarrollo.

Como el nombre del punto indica, realmente no se trata de explicar el conocimiento de terceros o de lo que se espera del conocimiento de estos sino, del conocimiento que yo, como responsable del proyecto, tengo sobre el entorno de la aplicación.

Se ha ido diciendo al principio de esta memoria que este proyecto es la aplicación de los conocimientos aprendidos durante la carrera, en cuanto a Java. Y por el resto de tecnologías, aprovechar todas las oportunidades que la empresa me ha ido dando en toda la temporada como becario con tareas que implicaban SQL o Javascript.



Figura 14: Tecnologías implicadas en el desarrollo de la aplicación.

PARTE 2 DESARROLLO DE LA APLICACIÓN

1. Introducción

En los siguientes apartados se procederá a explicar cómo todas las partes (la librería o capa de negocio, la capa de web o presentación y la capa de datos) interactúan entre sí.

Con el proyecto lo que se va a conseguir es una aplicación web capaz de mostrar diferentes gráficos en un período de tiempo. Este rango de tiempo puede ser modificado por el usuario según sus intereses.

Durante el desarrollo de esta aplicación, la capa de datos se encarga de la conexión a la base de datos ARQDES, para extraer toda la información necesaria que la librería después, va a procesar. Con la creación de la librería o capa de negocio se deja preparada toda esta información extraída y se la pasa a la capa de presentación. Finalmente, es la capa de presentación la encargada de mostrar toda esta información de la manera más atractiva y fácil de entender posible.

El modelo simplificado del proyecto, igual que en la primera parte, se puede ver en la siguiente imagen:

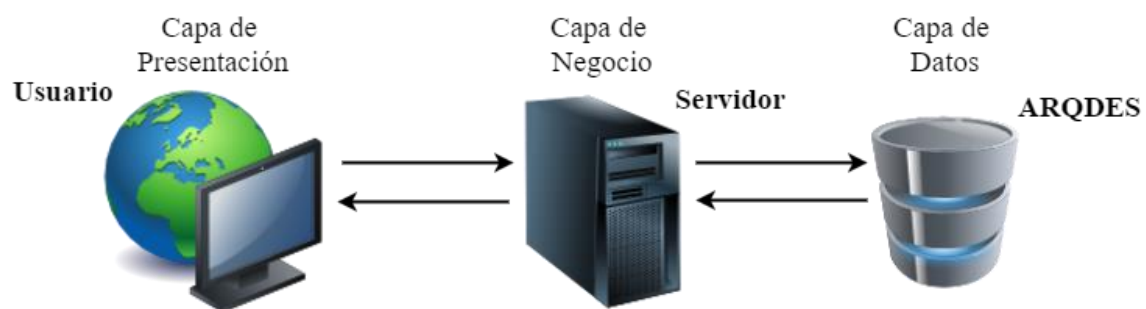


Figura 15: Conexión de servidor con la base de datos y el usuario.

Además de profundizar en la implementación de las tres capas del modelo, en esta parte de la memoria se van a explicar los requisitos y el diseño que se han tenido en cuenta antes de empezar el desarrollo de la aplicación. Otro de los puntos a tratar serán las aplicaciones similares a la del proyecto, la inversión que ha supuesto la realización del proyecto y las conclusiones que se extraídas después de los resultados.

2. Análisis y diseño

Antes de empezar con el desarrollo de la aplicación web, uno de los puntos donde se ha empleado un tiempo razonable ha sido, en pensar en los requisitos que se debían seguir durante el proyecto. Estos requisitos son por un lado, los que satisfacen a los usuarios y por el otro, los que serán útiles tanto a la propia aplicación como a AOD. Por esta razón, en las siguientes secciones se explicará cuales se han tenido en cuenta para obtener los resultados esperados y de qué manera han determinado la forma de la implementación.

Otro de los temas a tratar en este apartado de la memoria, teniendo en cuenta los requerimientos, era dibujar un primer esquema del diseño o aspecto de la aplicación. Este diseño, se corresponde a una combinación de la primera y segunda fase del modelo unificado UML explicado en la primera parte de la memoria. Está compuesta por tres procesos. Cada uno de estos procesos ayudará a realizar un análisis más completo de todas aquellas características que, determinarán como debe ser la aplicación web. El primero de los tres es la toma de requerimientos, el segundo el análisis del diseño con la construcción de diagramas de casos de uso y finalmente el del prototipo o primeras instancias de lo que se quiere en el resultado final del proyecto.

Cada uno de estos procesos sólo ha sido la primera concepción en papel hasta el resultado final en el navegador y como se verá en apartados de la implementación, se han ido refinando a lo largo del desarrollo.

2.1 Requisitos

Los requerimientos nos indican las necesidades y limitaciones que se tienen que tener en cuenta antes del desarrollo de la aplicación web. Existen diversas maneras de obtener requisitos o requerimientos, en el caso del proyecto se utilizaron básicamente dos métodos: estudiando lo que le interesaría al usuario con rol de usuarios y analizando aplicaciones similares.

Para esta fase de análisis de requerimientos del proyecto, Daniel Cufi, el tutor del proyecto en Everis, ha sido quien más los ha marcado porque conoce la parte cliente o parte interesada (en este caso). Durante varias sesiones se dedicó un tiempo razonable para dejar claro cada uno de los requisitos, ya que con ellos se decidía cómo empezar y de qué manera, iban a ser las interacciones entre usuario y aplicación.

Antes de empezar numerando cuales son los requerimientos que engloban la aplicación, primero se va a diferenciar los dos tipos de requerimientos con los que se puede encontrar en cualquier desarrollo software. Estos dos tipos son el **funcional** y el **no funcional**. La principal diferencia entre ambos tipos es que, el primero describe un comportamiento más bien técnico y concreto y el segundo, indica cualidades que ha de tener un sistema para llevarse a cabo correctamente.

Ahora que ya se conocen los diferentes tipos de requisitos con los que se puede definir el proyecto, empezaremos nombrando los del primer tipo, los **funcionales**:

- › **Visualización de los diferentes gráficos por defecto:** la aplicación debe estar diseñada de tal manera que, cuando se acceda por primera vez, se muestren los gráficos sin ningún problema y con un rango de fechas, definidas por defecto. Además de mostrar estos gráficos, la aplicación debe dar la opción al usuario, de poder filtrar las fechas o el rango de datos de estos gráficos. Requisito ligado al requisito **Filtrado por fechas** ya que, se trata de esta última especificación que se ha explicado.
- › **Acceder al detalle de cada gráfico:** una vez se ha accedido a la página principal de la aplicación, una de las opciones debe ser, poder acceder al detalle de cada uno de los gráficos que se muestran. Si se ha hecho un filtro, se debe poder acceder al detalle con este filtro ya aplicado. Como pasa con el anterior requisito, este también está ligado al requisito **Filtrado por fechas**, por el mismo motivo.
- › **Añadir/editar datos de un gráfico:** esta debe ser otra opción disponible en la aplicación, de manera similar al detalle de los gráficos. Con esta opción y si el administrador, previamente, ha dado permisos, un usuario podrá añadir o editar los datos de los gráficos que se muestran. Debe estar diseñada como una entrada de datos capaz de recibir un envío masivo de datos, mediante otros métodos no manuales, para guardar la nueva información. Esta última característica permitirá al administrador poder actualizar o insertar nuevos datos si quiere editar o añadir gráficos.
- › **Filtrado por fechas:** con este requisito se va a permitir al usuario poder filtrar el rango de fechas en el que quiere ver cada uno de los gráficos tanto de la página principal como cuando quiera ver el detalle de alguno de estos. Como se ha mencionado, está ligado a los requisitos: **Visualización de los diferentes gráficos por defecto** y **Acceder al detalle de cada gráfico**.
- › **Edición del tipo de gráfico:** no de la misma forma que filtrar las fechas de los gráficos pero, de una manera similar, los usuarios pueden cambiar el tipo de gráfico que quieran visualizar en cualquier momento. Para hacerlo, los usuarios deben tener conocimiento de las características de los gráficos, que sólo conoce el administrador. Por este motivo, esta opción es posible sólo si lo consultan antes.

De los requisitos no **funcionales** que determinan el desarrollo de la aplicación y que necesita para que los requisitos **funcionales** se lleven a cabo correctamente, la aplicación debe ser: **eficiente, escalable, multiplataforma** y sobretodo **reusable**.

2.2 Diagramas de casos de uso

Lo siguiente después de determinar los requisitos del proyecto, es la construcción del funcionamiento general de la interfaz, en el caso de este proyecto, la aplicación web. Para hacerlo, se tiene que definir quién va utilizar la aplicación y de qué manera interactuarán con la interfaz.

En cuanto a las personas que interactúan con el sistema, podemos encontrar de dos tipos:

- › **Usuario:** perfil de la persona visitante de la web, con intención de obtener la información que se expone en esta. En el caso de este proyecto, el usuario será el cliente al que se le quiere ofrecer el proyecto desde Everis.
- › **Administrador:** perfil de la persona con acceso completo a todas las funcionalidades de la web. En el caso de este proyecto, mi persona como desarrollador y el resto de trabajadores de Everis cuando esta aplicación se publique en un servidor de la empresa.

Ahora que ya se han definido los **actores** implicados en la aplicación, el siguiente paso es analizar qué pueden hacer, es decir, los **casos de uso**. Para que se entienda mejor y antes de mostrar los diagramas de casos de uso, en la siguiente tabla (Tabla 6) podemos ver un resumen con todos los casos de uso y los actores involucrados:

Actor	Caso de uso
Usuario	Visualizar los gráficos
Usuario	Filtrar el rango de fechas de los gráficos
Usuario	Ver el detalle de los gráficos
Usuario	Insertar valores en los gráficos
Usuario	Cambiar el tipo de gráfico
Usuario	Cargar el valor añadido en los gráficos
Administrador	Acceder al sistema de gestión de gráficos
Administrador	Añadir/Editar gráficos

Tabla 6: Casos de uso de la aplicación.

La forma gráfica de ver esta tabla de los casos de uso de la aplicación, son los diagramas. El primer diagrama de casos de uso que se va a tratar es el que protagonizan los usuarios de la interfaz y es la que se puede ver en la imagen inferior (Figura 18). Como se puede ver, existen dos tipos de relaciones entre casos de usos, presentes en el diagrama: el **include** y el **extends**.

La relación **include** indica que el caso de uso al que apunta la flecha o caso de uso incluido, está presente en el proceso de funcionamiento del caso de uso desde donde

sale esta flecha o más conocido como caso de uso base. Por lo tanto, sin el caso incluido, el caso de uso base no cumple su objetivo [46].

Aunque la relación **extends** es parecida, no es necesario que el segundo caso suceda para que el caso de uso base funcione. La flecha es inversa a la del **include**, apunta al caso de uso base que extiende el funcionamiento del segundo caso [46].

Ahora conociendo estas relaciones el diagrama de casos de uso para los usuarios es el que se puede ver a continuación:

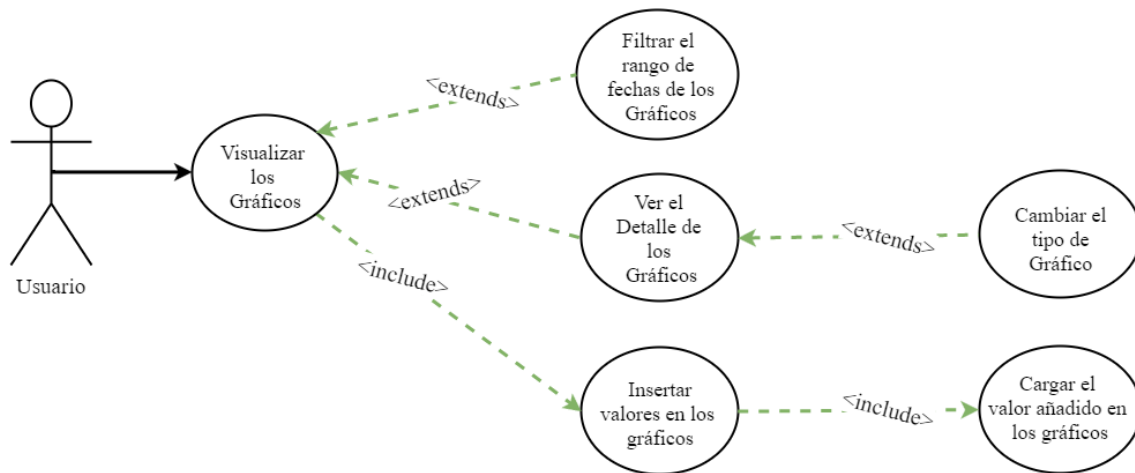


Figura 16: Diagrama de casos de uso para los usuarios.

- › **Visualizar los gráficos:** el usuario de la web quiere ver el estado de sus aplicaciones de forma gráfica. Para ello, hará uso del aplicativo a desarrollar. Antes de este caso de uso, el usuario se ha conectado a la aplicación introduciendo la dirección en el navegador. El sistema muestra todos los gráficos disponibles. El usuario observa la información que puede sacar de cada uno y si quiere cambiar el rango de fechas de los gráficos (relación de **extends**), ver el detalle de los gráficos (relación de **extends**) o insertar un valor en los gráficos (relación de **include**) puede hacerlo desde este caso de uso.

- › **Filtrar el rango de fechas de los gráficos:** el usuario ha decidido cambiar el rango en el que se ven los gráficos que previamente ha visualizado. El sistema pide al usuario una fecha inicio y una fecha fin para el nuevo rango de los gráficos. El usuario hace click en el botón de “filtrar” una vez ha insertado estas dos fechas que indican, el inicio y el final del nuevo rango. El sistema vuelve a visualizar los gráficos con el nuevo rango de fechas.

- › **Ver el detalle de los gráficos:** cuando el usuario ha visualizado los gráficos y quiere algunos de estos gráficos con un poco más de detalle. El usuario hace click en las opciones del menú lateral donde están todos los gráficos. El sistema carga el detalle del gráfico en cuestión. Desde este caso de uso, el usuario puede cambiar el tipo de gráfico que se está visualizando.

- › **Cambiar el tipo de gráfico:** el usuario decide cambiar el gráfico que está viendo sin tener que volver a la página principal. Para hacerlo, el usuario modifica la url que ve cuando entra en el detalle de alguno de los gráficos. Previamente el administrador le ha tenido que informar sobre cómo y a qué direcciones cambiar. El sistema carga el gráfico que se informa por la url.

- › **Insertar valores en los gráficos:** el usuario puede insertar valores nuevos en los gráficos. Para hacerlo y como pasa con el anterior caso de uso, el usuario debe consultar con el administrador el funcionamiento de la aplicación. El sistema pide tres campos: Fecha, Clave y Valor. La Clave y el Valor son dos campos que deben rellenarse pero, si Fecha está vacía el sistema se encarga de interpretarlo y poner el día actual como Fecha. La Clave debe ser una que ya ha estado dada de alta por el administrador y que ya esté siendo usada en alguno de los gráficos. Si no es utilizada en los gráficos pero, si dada de alta se pueden insertar los valores pero esto no cambiará nada los gráficos que ve. Por este motivo, el usuario debe conocer qué Claves se usan en cada gráfico y esto sólo es posible si el administrador le informa. Una vez los tres campos han sido rellenados, el usuario hace click en “Insertar”. El sistema interpreta los campos y los muestra en otra página que es el siguiente caso de uso.

- › **Cargar el valor añadido en los gráficos:** el usuario ve los tres campos que ha insertado previamente y hace click en “Cargar” para que estos se guarden en el sistema y para que ya se puedan cargar en los gráficos al volver a cargar la página principal. Una vez cargados, el sistema manda al usuario otra vez a la página de los tres campos, el del caso de uso anterior.

De la misma forma, el diagrama de casos de uso para el administrador es el que se puede ver a continuación:

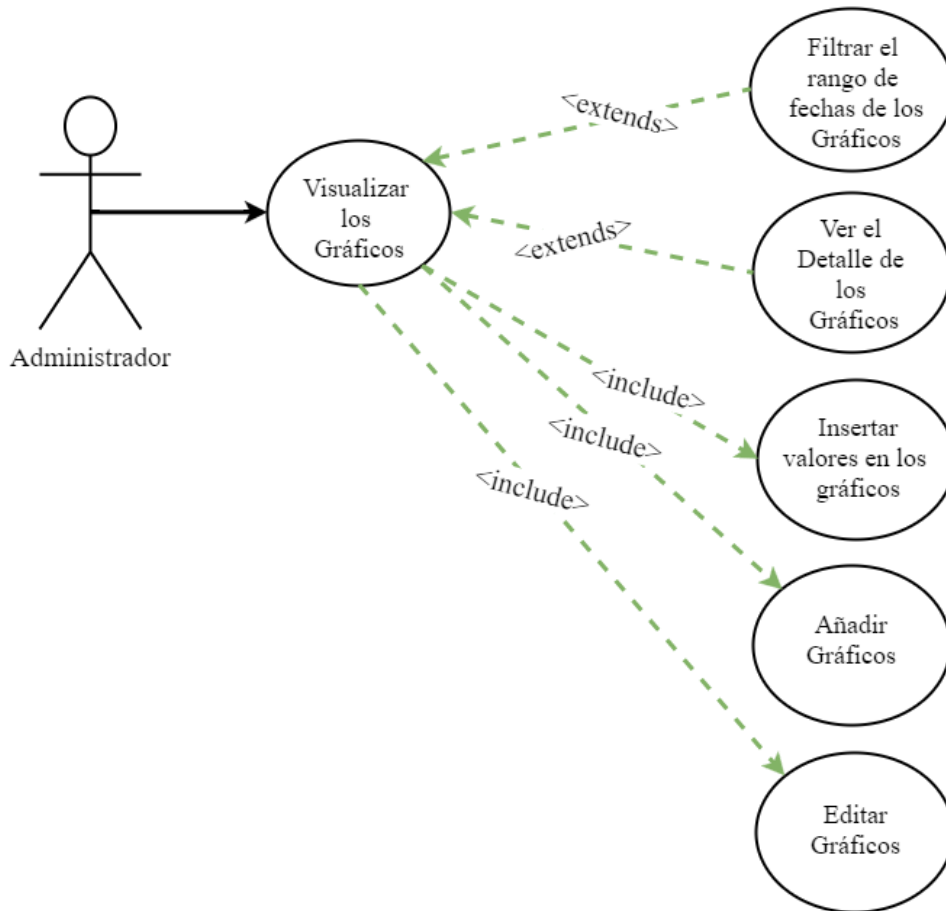


Figura 17: Diagrama de casos de uso para el administrador.

- › **Añadir gráficos:** el administrador es el único actor que puede añadir más gráficos. Aunque no haya un botón o una página específica para realizar este caso de uso, es un caso de uso presente en la aplicación.

- › **Editar gráficos:** es cierto que si el administrador proporciona toda la información mencionada en los casos de uso, que implican a los usuarios, estos pueden “editar” los gráficos. A pesar de eso, es el administrador quien realmente puede modificar los gráficos, tanto el diseño como los valores que se muestran en estos. El administrador tiene conocimiento de cómo funcionan todas las urls y campos de los formularios que hay. El sistema sólo responde a lo que éste va haciendo.

Este fue un primer planteamiento de los casos de uso, durante el desarrollo del proyecto se intenta mejorar la aplicación haciéndola más eficiente cada vez.

2.3 Diseño y Maquetación: pantallas y transiciones

En este apartado se tratará los aspectos básicos del diseño visual de la aplicación, sin tener en cuenta aspectos de programación. Al no tener conocimientos de diseño gráfico, la tarea de distribuir el contenido o la elección de colores se han puesto en manos de una librería disponible para Javascript. Por este motivo, la tarea de diseño que quedaba era la forma que se quería desde AOD.

Para poder hacer este primer prototipo se ha utilizado una herramienta software online diseñada con este fin y conocida con el nombre de Balsamiq. Este prototipo es una pequeña representación gráfica de lo que inicialmente se ha propuesto desde AOD, como el resultado de la aplicación web. Ha servido de guía durante la programación y se han llegado a los resultados que ha da tiempo, tal y como se verá en apartados siguientes.

Utilizar prototipos durante el diseño de la web es muy útil ya que, ahorra esfuerzos a la hora de implementar y evaluar el sistema. Cualquier cambio se puede realizar antes en el prototipo y así, tener una idea de la dificultad o posibilidad del cambio antes de empezar con el código.

A continuación se puede ver el prototipo de cada una de las pantallas con las que se puede encontrar el usuario al usar la aplicación. La primera es la página principal:



Figura 18: Prototipo de la página principal de la aplicación.

Una vez los usuarios están en la **página principal**, donde se visualizan todos los gráficos, ya pueden ir a ver el detalle de alguno de los gráficos y encontrarse con una pantalla, como el siguiente prototipo Balsamiq. Desde esta página se puede acceder al **resto** que se puede ver en el menú lateral.

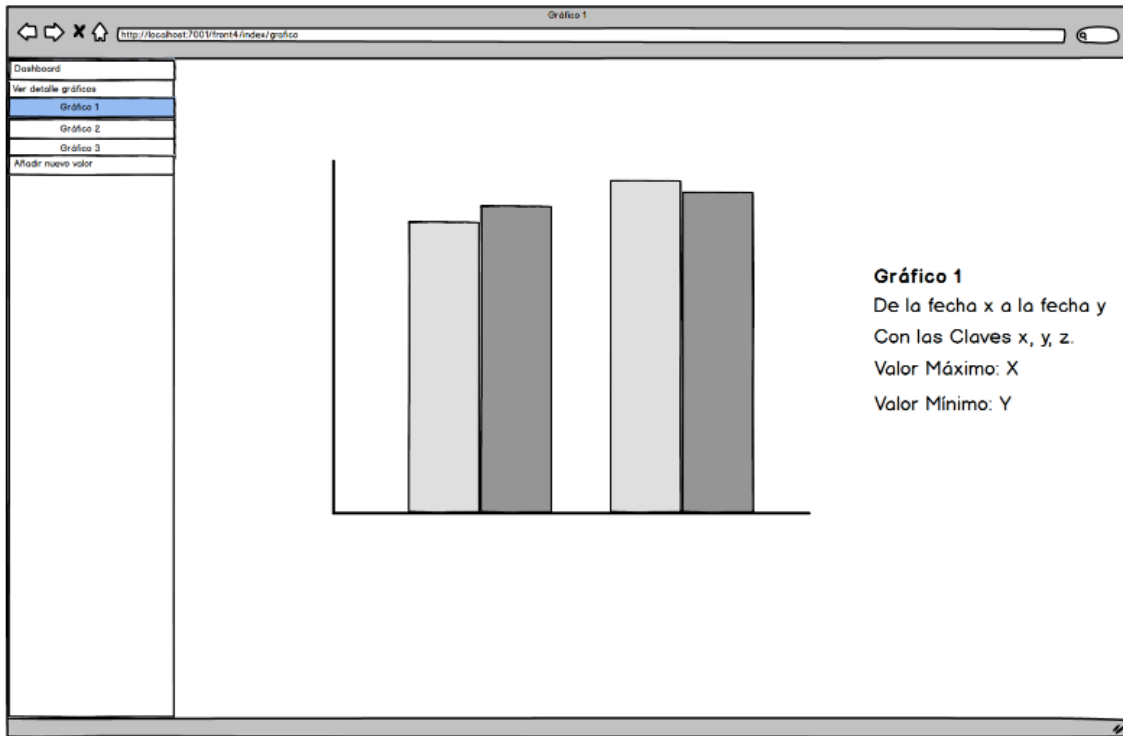


Figura 19: Prototipo del detalle de uno de los gráficos de la pantalla principal.

Desde la **página principal** otra de las páginas a las que se puede acceder es a “**Añadir nuevo valor**”. Como se ha explicado anteriormente, desde esta última página se redirige al usuario a otra página “**Información valor nuevo**” una vez se han rellenado todos los campos del formulario. Como pasa con el detalle de los gráficos también se puede acceder al resto de páginas de la barra lateral en cualquier momento.

“**Información valor nuevo**” muestra los datos insertados en la página anterior y el usuario decide si cargar la nueva instancia en los gráficos, volver a insertar otro valor o simplemente volver a la página principal sin hacer nada. O acceder al resto de páginas del menú lateral.

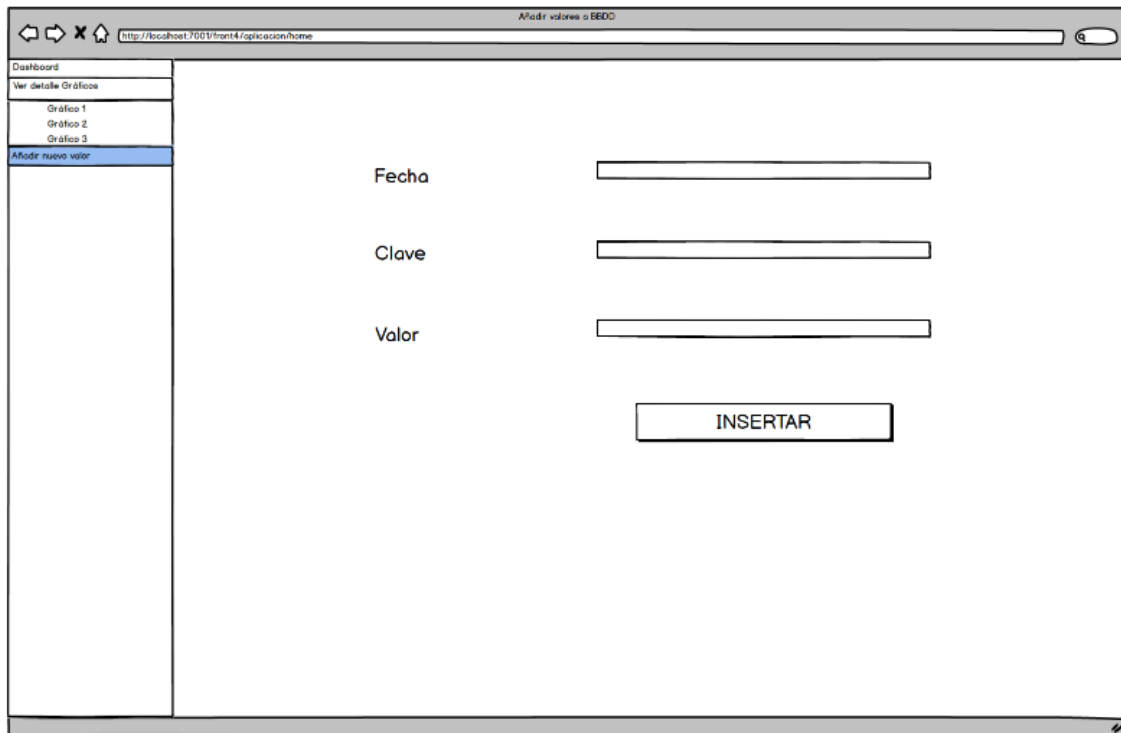


Figura 20: Prototipo de la página “Añadir nuevo valor”.

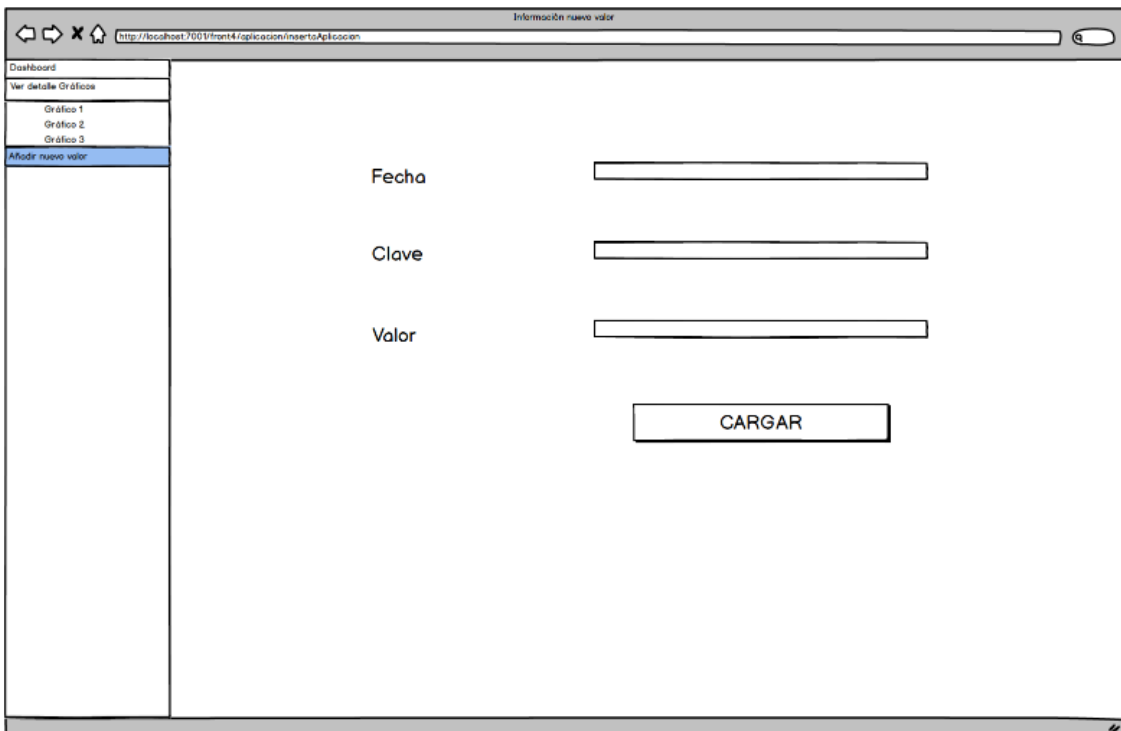


Figura 21: Prototipo de la página “Información nuevo valor”.

3. Desarrollo

Ahora que ya se sabe lo necesario, en los siguientes capítulos se explicarán cada una de las capas de desarrollo de la aplicación web. No se ha comentado en el capítulo de diseño pero, este proyecto se ha inspirado en una aplicación ya existente: **dashing.io**. El desarrollo de la aplicación habría sido tan simple como utilizar el código libre y aplicarlo con los datos necesarios. Uno de los problemas de hacerlo de esta manera era que, no se utilizaban los lenguajes de programación y tecnologías que para AOD son importantes. Otro de los problemas, y el más determinante, era el formato en el que se habían de rellenar los gráficos, no se podía aplicar la técnica pensada como Fecha, Clave y Valor. De aquí otra de las razones por la que este proyecto surgió con la idea de crear una aplicación implementada con herramientas conocidas y presentes en la mayoría de compañías como Everis.

El objetivo principal del desarrollo de esta aplicación es conseguir que clientes de Everis acepten el proyecto y la utilicen con fines complementarios a las estadísticas ya generadas desde AOD.

La esencia de la aplicación es poder ser tan general que no sólo sea útil para el tipo de clientes al que está acostumbrado Everis sino que también se pueda aplicar a cualquier usuario que no tenga nada que ver con empresas y sirva para mostrarles el estado de su PC o las temperaturas de su alrededor. Aunque durante esta memoria las imágenes y cada una de las capas de desarrollo tratan el primer caso, se puede aplicar a los otros dos casos cambiando la gestión de datos.

3.1 Descripción de etapas del desarrollo

Tal y como se ha explicado en la primera parte de la memoria, el desarrollo está dividido en tres etapas, que se corresponden con las tres capas del modelo de arquitectura de la aplicación:

- › La capa de presentación o web.
- › La capa de negocio o librería.
- › Y la capa de datos.

En la primera parte de la memoria se empezó explicando la creación de la librería y se vio que esta contiene la capa de datos por esta razón, en esta parte se empezará explicando el modelo de base de datos, después el desarrollo de la librería y finalmente el desarrollo de la página web o capa de presentación.

4. Modelo de base de datos

Se procede a crear la base de datos para almacenar y procesar información. Para ello se ha utilizado una base de datos Oracle 11. Esta base de datos sirve para mantener la información insertada a través de la entrada de datos de la aplicación web. Estos datos se almacenan en tablas. Todas estas tablas e información que contienen, luego serán utilizadas en la librería para realizar cálculos con estos.

4.1 Diagrama de base de datos

Para poder ver mejor el modelo de base de datos del proyecto se proporciona el siguiente diagrama de base de datos (Figura 24) que contiene cada una de las tablas utilizadas en el desarrollo y las relaciones que hay entre ellas:

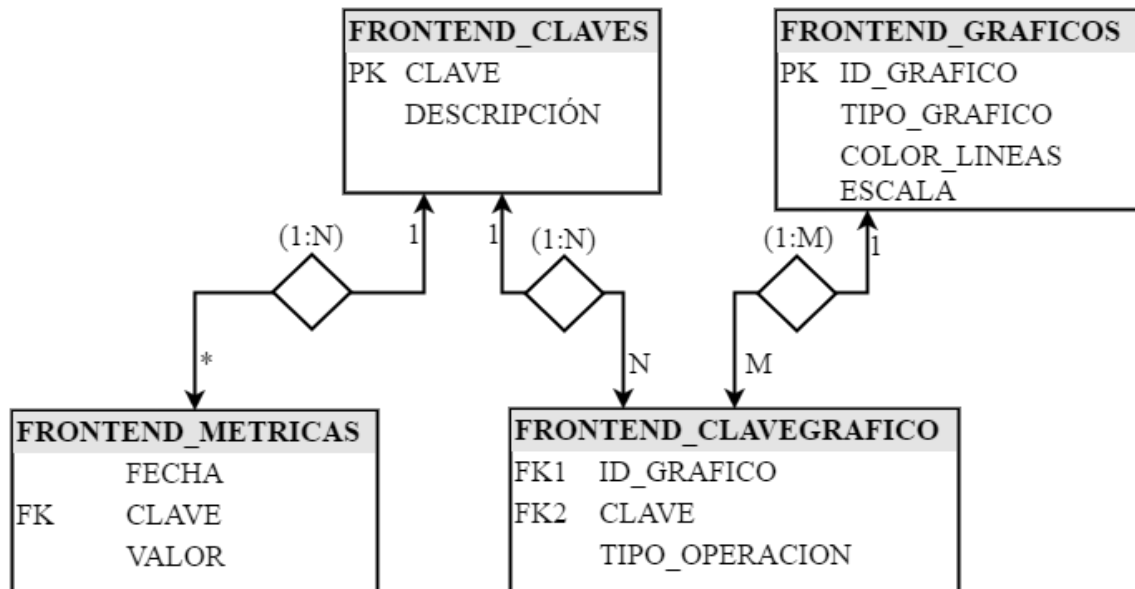


Figura 22: Diagrama del modelo de base de datos.

Como se puede ver en el diagrama (Figura 24), cada una de las tablas está relacionada con al menos otra. Se parte de la idea que la CLAVE y el ID_GRAFICO son únicos y para ser utilizados en el resto han de estar dados de alta previamente (por el administrador). Por lo tanto, para que haya métricas (FRONTEND_METRICAS) o para que haya información sobre qué claves están en cada uno de los gráficos (FRONTEND_CLAVEGRAFICO) primero, se deben insertar las respectivas CLAVE e ID_GRAFICO en las tablas FRONTEND_CLAVES y FRONTEND_GRAFICOS.

El resto de relaciones del diagrama se resume con los siguientes puntos:

- › Para una única CLAVE de FRONTEND_CLAVES pueden haber N registros en FRONTEND_METRICAS.

- › Para una única CLAVE de FRONTEND_CLAVES pueden haber N registros en FRONTEND_CLAVEGRAFICO.

- › Para un único ID_GRAFICO de FRONTEND_GRAFICOS pueden haber M registros en FRONTEND_CLAVEGRAFICO.

- › Con estas dos últimas relaciones FRONTEND_CLAVEGRAFICO, se consigue que un gráfico o ID_GRAFICO contenga más de una CLAVE. Lo que permitirá mostrar en un mismo gráfico más de una CLAVE.

4.2 Tablas

Considerando el modelo de base de datos (Figura 24), se procede a crear las tablas. En esta base de datos, se ha creado dos tipos de tablas.

La primera que se utiliza para almacenar información de los ID_GRAFICO y CLAVE. Estas tablas nos dan la información necesaria para luego crear generar instancias en las otras dos tablas FRONTEND_GRAFICOS y FRONTEND_CLAVES.

El segundo tipo de tablas son precisamente las instancias creadas a partir de las primeras, las que servirán para hacer cálculos en la librería. Por ejemplo, para saber el número de gráficos (con el ID_GRAFICO) que tendrá la aplicación web o la suma de valores para una CLAVE en un rango de fechas.

FRONTEND_METRICAS y FRONTEND_CLAVEGRAFICO almacenan los registros que haya de cada CLAVE y de cada ID_GRAFICO.

4.2.1 Información sobre las tablas

Lo que se puede decir de esta base de datos formada por 4 tablas es que, cada una de las tablas será utilizada en el proyecto realizando tareas diferentes dentro de la librería.

Cada una de las tablas será creada y estará gestionada por el administrador pero, la tabla FRONTEND_METRICAS es la que podrá ser editada por el usuario o cliente.

FRONTEND_CLAVES	
COLUMN_NAME	DATA_TYPE
CLAVE	VARCHAR2(200 BYTE)
DESCRIPCION	VARCHAR2(200 BYTE)

Tabla 7: Tabla FRONTEND_CLAVES de la base de datos.

En esta tabla el administrador da de “alta” las claves de las que se pueden insertar métricas o registros en la tabla FRONTEND_METRICAS. Si se quiere insertar un valor en FRONTEND_METRICAS de una CLAVE que no está en FRONTEND_CLAVES, la base de datos da error devuelve un error de integridad (clave primaria no encontrada).

Cuando el usuario conozca las claves que necesita entonces, puede añadir instancias de esta y editar la tabla FRONTEND_METRICAS.

Otra información que proporciona esta tabla además de la clave, es la DESCRIPCION de la misma. Con esta DESCRIPCION se pretende hacer más entendedora una CLAVE.

Por ejemplo, en el proyecto se puede encontrar con una CLAVE como Plantilla_Maquina2_Total y no es tan claro. Entonces, con su DESCRIPCION Total plantillas en la maquina 2, es comprensible.

FRONTEND_METRICAS	
COLUMN_NAME	DATA_TYPE
FECHA	DATE
CLAVE	VARCHAR2(200 BYTE)
VALOR	VARCHAR2(200 BYTE)

Tabla 8: Tabla FRONTEND_METRICAS de la base de datos.

Esta es una de las tablas más importantes de la base de datos ya que, a partir de esta, se hacen todos los cálculos necesarios para mostrar en los gráficos de la aplicación. Contiene la FECHA, CLAVE y VALOR.

Para una misma FECHA puede haber registros de más de una CLAVE y una misma CLAVE puede tener instancias en más de una FECHA.

En cada uno de esos registros hay un VALOR que es el que luego será utilizado en los cálculos de la librería.

FRONTEND_GRAFICOS	
COLUMN_NAME	DATA_TYPE
ID_GRAFICO	VARCHAR2(200 BYTE)
TIPO_GRAFICO	VARCHAR2(200 BYTE)
COLOR_LINEAS	VARCHAR2(200 BYTE)
ESCALA	VARCHAR2(200 BYTE)

Tabla 9: Tabla FRONTEND_GRAFICOS de la base de datos.

FRONTEND_GRAFICOS funciona de forma similar a la tabla FRONTEND_CLAVES ya que, tiene una clave primaria que en este caso no es CLAVE sino, el identificador ID_GRAFICO de cada gráfico.

En esta tabla se almacena información de los gráficos que luego la librería utilizará para hacer más dinámica la capa de presentación. Por ejemplo, con el TIPO_GRAFICO se informa el gráfico o JSP que se va a visualizar. Por lo tanto, como TIPO_GRAFICO contiene el nombre del JSP, en un mismo método de la librería, se pueden abrir todos los gráficos si previamente se indica de qué ID_GRAFICO se trata.

El COLOR_LINEAS y ESCALA son dos columnas de la tabla que se han creado con el mismo objetivo, se pretende que los gráficos sean lo más dinámicos posibles.

El COLOR_LINEAS sirve para determinar el color de los objetos del gráfico. Y ESCALA determina el tamaño de los gráficos ya que, como se verá más adelante, no todos tienen las mismas dimensiones.

FRONTEND_CLAVEGRAFICO	
COLUMN_NAME	DATA_TYPE
ID_GRAFICO	VARCHAR2(200 BYTE)
CLAVE	VARCHAR2(200 BYTE)
TIPO_OPERACION	VARCHAR2(20 BYTE)

Tabla 10: Tabla FRONTEND_CLAVEGRAFICO de la base de datos.

Esta tabla igual que la tabla FRONTEND_METRICAS es muy importante para el desarrollo de la aplicación. Con esta tabla se conocen las claves que se van a mostrar en cada uno de los gráficos. Presenta una relación N:M con FRONTEND_CLAVES y FRONTEND_GRAFICOS ya que, un gráfico puede mostrar más de una clave y una clave puede estar en más de un gráfico. Además ambas claves foráneas han de estar previamente en las dos tablas donde se encuentran las claves primarias.

La otra columna de esta tabla TIPO_OPERACION será importante también ya que, gracias a esta, la librería realizará un tipo de cálculo u otro. Por ejemplo, podemos encontrar una ID_GRAFICO PlantillasPorMaquina, con una CLAVE Plantilla_Maquina2_Total y un TIPO_OPERACION sum. Este “sum” indica a la librería que esta CLAVE agrupará todas sus instancias VALOR con una operación de suma en un rango de FECHA.

5. Desarrollo de la librería

La librería se encarga de obtener los datos necesarios para calcular los valores de rendimiento, que se van a mostrar después en los gráficos. Para hacerlo, se definen dos objetivos. El primer objetivo es crear métodos con los que recuperar los datos, desde la aplicación. Y el segundo, obteniendo la información de los datos recuperados, procesarlos y prepararlos para luego, enviarlos a la siguiente capa, la de web o presentación.

De esta forma, con la creación de la librería es posible separar la recuperación de datos de la visualización de los mismos. Y es por esta división que, la librería se puede entender como si utilizase dos adaptadores o dos canales de información diferentes:

- › Por un lado, **el adaptador para la recuperación de datos** que se consigue con la conexión y actualización de la base de datos ARQDES, a través de la entrada de datos de la aplicación.

- › Y por el otro, **el adaptador para la visualización de gráficos** que cumple sus funciones, calculando cada uno de los valores necesarios para generar los gráficos de rendimiento. Cálculos que son posibles gracias a las instancias recuperadas con el primer adaptador.

5.1 Funcionalidades más relevantes del proyecto

Después de conocer el funcionamiento de la capa de negocio de la aplicación, en esta sección se van a mostrar los procesos más relevantes del proyecto. Estos procesos son la representación gráfica, de las funcionalidades que se explicarán en cada apartado dedicado a los adaptadores de la librería.

1. La entrada de datos es un formulario con tres campos y como se ha visto cuando se ha explicado el controlador del primer adaptador, no es un proceso largo ni complejo.





Figura 23: Proceso que se sigue a la hora insertar datos.

Obviamente este proceso se sigue cada vez que el Administrador actualiza la base de datos o cada vez que el usuario utiliza la opción insertar valor.

La idea cuando la propuesta sea aceptada, es crear un proceso **fichero Shell (extensión .sh)** que ayudará a que esta inserción de información en la base de datos no sea manual, como se ha explicado en este proyecto.

Desde AOD ya se realizan este tipo de subidas y la opción más fácil, tratándose del diseño del formulario de la aplicación, es utilizar los **curls**.

El Shell es un lenguaje basado en comandos y en este proyecto servirá para ejecutar todos los comandos **curl**, que se utilizarán para transferir datos al formulario cargaAplicación. Un ejemplo de este comando lo podemos ver a continuación:

```
curl --data "fecha=17/06/2016&clave=PlantillasMaquina1&valor=1234" http://localhost:7001/front4/insertaAplicacion
```

Figura 24: Comando con el que se insertarán tantos valores como se quieran a la base de datos. Con esto se evita hacerlo manualmente por el formulario de la aplicación.

2. La segunda función más importante es cómo se pasan los datos a cada uno de los gráficos:





Figura 25: Proceso que se sigue cuando se pasan el objeto que los gráficos necesitan para mostrarse en la pantalla.

5.2 Modelo de la creación de la librería

En esta sección se va a explicar cómo se han definido los objetos Java del proyecto y como en el modelo de base de datos, se va a mostrar dos diagramas resumen para que, se entienda mejor.

Antes de mostrar ningún diagrama se debe mencionar que, para un buen orden y una buena estructura dentro del proyecto Java se han utilizado los **paquetes**. Los **paquetes** son una propiedad de los proyectos Java que sirven de contenedor de clases, agrupan distintas partes del programa según la funcionalidad de las clases. Este proyecto contiene 7 paquetes y cada uno constituye una parte importante del proyecto:

- › El **primer** paquete contiene dos controladores, uno para cada adaptador.
- › El **segundo** paquete contiene dos interfaces, uno para cada adaptador. Cada una de estas interfaces es la definición de otras dos clases útiles para la recuperación de datos.
- › El **tercer** paquete contiene la implementación de las dos interfaces anteriores. En estas se realizan todos los cálculos necesarios para la recuperación de datos.
- › El **cuarto** paquete contiene dos clases para las excepciones. Excepciones que se puedan producir tanto a nivel de base de datos, como a nivel de capa de presentación (cuando los usuarios insertan campos en los formularios de la aplicación y no cumplen el formato que deben o cuando los dejan vacíos).
- › El **quinto** paquete contiene una clase que servirá para escuchar a la aplicación durante la ejecución del código y después, generar los logs.
- › El **sexto** paquete contiene cinco objetos beans, que servirán para guardar todos los datos recuperados.

- › El **séptimo** paquete contiene tres clases Mapper necesarios para la recuperación de datos. Estas serán útiles durante este proceso, para que no hayan errores de identificación de tablas, columnas o similares. Habrán tantas como las del tercer paquete necesiten o hayan utilizado para extraer datos de las tablas.

Conociendo el orden que siguen las clases, antes de ver la relación entre ellas y ver el detalle de cada una, primero explicaré partes importantes del proyecto. La ejecución de estas partes es previa para el funcionamiento correcto del resto de clases, que se explicarán después de mostrar los diagramas de clases de ambos adaptadores.

Como se explicó en la primera parte de la memoria, para el desarrollo de esta aplicación se necesitaba por un lado la conexión a la base de datos ARQDES y por otro lado al servidor local de Weblogic.

Se obtiene una conexión al servidor local de Weblogic con la instalación de Oracle Weblogic y la configuración del proyecto Java en este servidor, una vez se ha instalado correctamente.

Cuando el proyecto Java y el servidor Weblogic están configurados sin problemas, desde cualquier navegador se puede acceder a la consola local Weblogic y comprobar si el despliegue del EAR del proyecto Java está en estado OK. Si es así entonces, ya se puede crear una conexión a base de datos.

La conexión a la base de datos **ARQDES** se crea desde la consola Weblogic y siguiendo los pasos que se ha explicado en la primera parte de la memoria.

Cuando el origen de datos **JDBC** se crea, la consola se encarga de avisar si se ha logrado la conexión. En caso afirmativo, lo que nos interesa y lo que se utiliza en el proyecto Java para acceder a las tablas de esta base de datos es el nombre de **JNDI**.

En el caso de ARQDES el JNDI que corresponde es el que podemos ver a continuación:

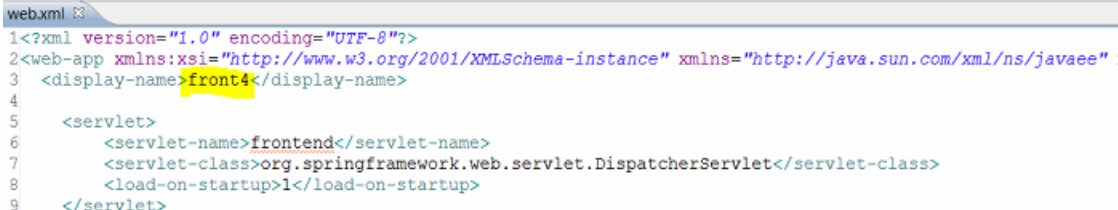
Orígenes de Datos (Filtrado: Existen Más Columnas)		
Nombre	Nombre de JNDI	Destinos
IODBTBL1	jdbc/IODBTBL1	AdminServer

Figura 26: Consola Weblogic donde se puede ver el nombre del JNDI del origen de datos creado.

Con este JNDI, desde el proyecto Java ya se puede invocar a la base de datos para actualizar la información de las tablas o para calcular los valores que interesan para luego, guardarlos en objetos que se pasarán a la capa de presentación.

Antes de mostrar la invocación en el proyecto Java primero es necesario conocer dos ficheros del proyecto que son fundamentales para el despliegue de la aplicación. Estos ficheros son dos **XML**.

El primer **XML** es el fichero por defecto, que se genera cuando se crea un proyecto web en plataformas como Eclipse, como ha sido en este caso, para el desarrollo de una aplicación. Este fichero está con el nombre de **web.xml** y aquí se define entre otras cosas el nombre de la aplicación, nombre con el que se accederá desde el navegador. En la siguiente imagen se puede ver cómo se hace esta definición:

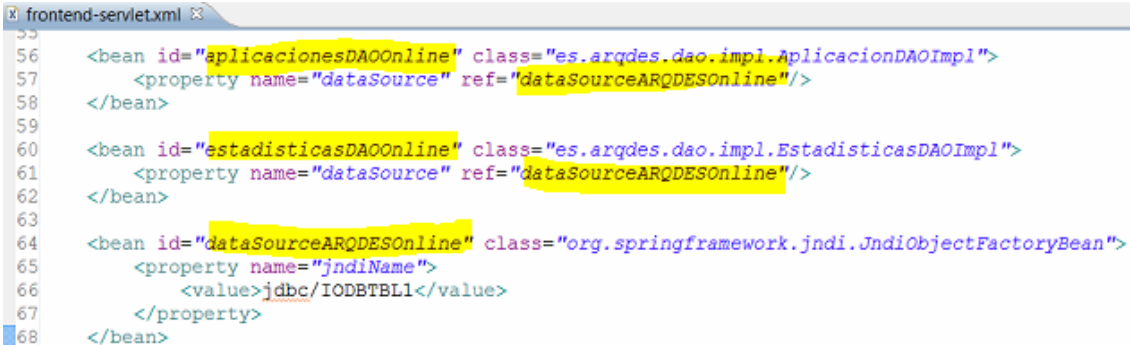


```
web.xml
1<?xml version="1.0" encoding="UTF-8"?>
2<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee"
3  <display-name>front4</display-name>
4
5  <servlet>
6    <servlet-name>frontend</servlet-name>
7    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
8    <load-on-startup>1</load-on-startup>
9  </servlet>
```

Figura 27: Fichero web.xml del proyecto Java donde se puede ver el nombre de la aplicación, front4, subrayado.

Como se puede ver en la Figura 26, se trata de **servlet** definido con el nombre de **frontend**. Un **servlet** es una clase de lenguaje de programación Java que facilita el tratamiento de las peticiones que llegan al servidor. Facilita peticiones como el proceso de entrada de datos mediante formularios o la generación de contenidos en formatos variables como lo son el de los gráficos [47].

Con el nombre de frontend seguido de **-servlet.xml** se crea el segundo fichero **XML** que se ha mencionado antes. Este fichero **frontend-servlet.xml** es el que contiene, entre otras cosas, la invocación al JNDI creado.



```
frontend-servlet.xml
56 <bean id="aplicacionesDAOOnline" class="es.arqdes.dao.impl.AplicacionDAOImpl">
57   <property name="dataSource" ref="dataSourceARQDESOnline"/>
58 </bean>
59
60 <bean id="estadisticasDAOOnline" class="es.arqdes.dao.impl.EstadisticasDAOImpl">
61   <property name="dataSource" ref="dataSourceARQDESOnline"/>
62 </bean>
63
64 <bean id="dataSourceARQDESOnline" class="org.springframework.jndi.JndiObjectFactoryBean">
65   <property name="jndiName">
66     <value>jdbc/IODBTBL1</value>
67   </property>
68 </bean>
```

Figura 28: Fichero frontend-servlet.xml del proyecto Java donde se puede ver la invocación al JNDI creado en el servidor Weblogic.

Tal y como se puede ver en la Figura 27, se define un **datasource** principal o acceso a la base de datos principal desde Java con el nombre de **datasourceARQDESOnline**.

Como se puede ver este datasource se define con el nombre del JNDI y a partir de esta se crean otras dos instancias **aplicacionesDAOOnline** y **estadisticasDAOOnline**, una para cada controlador del proyecto.

Con estas dos instancias ya se puede empezar a explicar el funcionamiento de ambos adaptadores.

5.2.1 Adaptador para la recuperación de datos

Para explicar de una forma más sencilla se va a utilizar un diagrama de clases. En este caso es sencillo de explicar, ya que para que este adaptador cumpla con sus funciones, se necesitan básicamente la relación entre las siguientes clases:

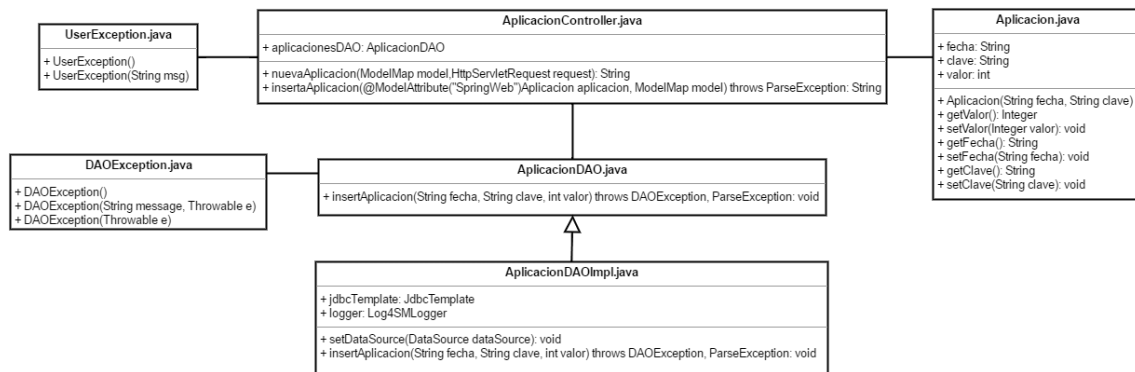


Figura 29: Diagrama de clases para la recuperación de datos.

Las funciones de este adaptador son:

1. Habilitar una url en la aplicación por la que cada vez que haya una petición en el servidor, el usuario pueda insertar tres variables: Fecha, Clave y Valor.
2. Y habilitar otra url por la que se cargan las variables, antes insertadas, en base de datos. Además, se muestran por pantalla para que el usuario las vea.

La clase principal de este adaptador es el controlador: **AplicacionController.java** y desde esta se llevan a cabo las dos funcionalidades anteriores.

Antes de explicar el detalle de cada funcionalidad primero se debe conocer las siguientes líneas de código que hay en este controlador:

```
@Controller
@RequestMapping(value="/aplicacion")
@EnableWebMvc
public class AplicacionController {
```

Figura 30: Se define la clase `AplicacionController.java` como controlador y `/aplicacion` como la dirección url con la que el usuario llamará, desde el servidor, a cada una de las funciones que hayan en esta clase, si no tienen otra dirección adicional.

`@Controller`, `@EnableMvc` y `@RequestMapping (value="/")` son anotaciones de Spring que ayudan a gestionar los controladores.

`@Controller` para etiquetar las clases que hacen de controladores en la aplicación.

El `@RequestMapping` se usa para conocer a que método del controlador tiene que direccionar cada llamada que hace el usuario desde la aplicación.

Para la primera funcionalidad:

- Se crea un método llamado **nuevaAplicacion**, donde se declaran tres variables, que sirven para guardar los valores que los usuarios insertan en el formulario, de la entrada de nuevas instancias en la base de datos. Estas variables son Fecha, Clave y Valor. Como se puede ver en la Figura 30, este método tiene una nueva dirección con la que el usuario la invocará desde el servidor. Y como ya se había definido una dirección para acceder al controlador, la dirección de este método es: `/aplicación/home`.

```
@RequestMapping(value="/home")
public String nuevaAplicacion(ModelMap model,HttpServletRequest request){

    logger.info("Se va a insertar una nueva instancia!");

    String fecha = (String)request.getParameter("fecha");
    String clave = (String)request.getParameter("clave");
    String valor = (String)request.getParameter("valor");

    return "cargaAplicacion";
}
```

Figura 31: Método `nuevaAplicacion` que se encuentra en el controlador `AplicacionController`.

- › Lo siguiente que se hace es llamar a la capa de presentación invocando al JSP diseñado para esta funcionalidad. En este caso se trata de un formulario **cargaAplicacion** y que ya se verá en detalle, en el siguiente apartado de esta memoria.

Para la segunda funcionalidad:

- › Se crea un método **insertaAplicacion** donde se pasan los valores de las tres variables, guardadas en el método **nuevaAplicacion**, a otra página JSP llamada **resultado**. Estas tres variables son atributos de la clase **Aplicación** ya que, es este el objeto diseñado con esa estructura.
- › Antes de pasar estas variables para que se muestren en el JSP, lo primero que se hace en este método es insertar estos valores en la base de datos. Para hacerlo primero se comprueba que la **Clave** y el **Valor** no sean nulos. Este chequeo se hace con la ayuda de la excepción **UserException**. Una vez se comprueba que no hay ningún problema cuando los usuarios insertan estos valores, se invoca al método **insertAplicacion**, con el que se cargan los valores en la base de datos.
- › El método **insertAplicacion** sólo se puede utilizar con la instancia de tipo **AplicacionDAO** declarada previamente. Esto se hace con la llamada al **datasource** creado en el fichero frontend-servlet.xml, para el primer adaptador tal y como se puede ver en la siguiente imagen:

```
@Qualifier("aplicacionesDAOOnline")  
private AplicacionDAO aplicacionesDAO;
```

Figura 32: La variable aplicacionesDAO es de tipo AplicacionDAO.

- › Si se ve la Figura 31, con la etiqueta **@Qualifier** (“aplicacionesDAOOnline”) se indica que la variable es de tipo **aplicacionesDAOOnline**. Mismo nombre que tiene el datasource de este adaptador, declarado en frontend-servlet.xml.
- › Que el método **insertAplicacion** sea de tipo **AplicacionDAO** significa que también lo es del tipo **AplicacionDAOImpl**. La razón, como se puede ver en el diagrama de clases (Figura 28), es que se trata de una clase que implementa la interfaz **AplicacionDAO**.

- › Antes de utilizar el método **insertAplicacion** de la clase **AplicacionDAOImpl**, primero se declara una variable que indica que se van a utilizar propiedades de base de datos. Para hacerlo igual que en el controlador, se indica que se trata del datasource. La siguiente imagen muestra cómo se hace:

```
private JdbcTemplate jdbcTemplate;

@Autowired
@Qualifier("dataSourceARQDESOnline")
public void setDataSource(DataSource dataSource) {
    this.jdbcTemplate = new JdbcTemplate(dataSource);
}
```

Figura 33: JdbcTemplate es la forma como Spring permite acceder al origen de datos y permitir la conexión a la base de datos para ejecutar queries, por ejemplo. Como se puede ver dataSourceARQDESOnline coincide con el nombre del datasource que contiene el JNDI del proyecto, como se vio en el fichero frontend-servlet.xml.

- › Cuando se han hecho las declaraciones necesarias, el método **insertAplicacion** contiene una query que se ejecuta cada vez que hay una **Fecha**, una **Clave** y un **Valor** diferentes. Como ya se ha visto en el controlador, no hay ningún problema si el usuario deja el campo **Fecha** vacío ya que si es así, se pone la fecha del día de hoy como valor.
- › Otra cosa que se puede ver en el método **insertAplicacion**, es el formato con el que se permite introducir valores en el campo de **Fecha**. Siendo exactos, se permite dos formatos diferentes y se controla que sean estos con la excepción **ParseException**. Una vez controlados estos tres campos, se ejecuta la query. En la Figura 33 se puede ver la query y en la Figura 34 cómo se ejecuta.

```
String insert = "INSERT INTO IODBTBL1.FRONTEND_METRICAS (FECHA, CLAVE, VALOR) " + "VALUES (?, ?, ?)";
```

Figura 34: Query para insertar las tres variables en la tabla FRONTEND_METRICAS de la base de datos.

```

try{
    jdbcTemplate.update(insert,new Object[]{date,clave,valor});
}
catch (DataAccessException e){
    throw new DAOException(e);
}

```

Figura 35: Update es un método del jdbcTemplate que se usa para ejecutar la query insert.

- › Como se puede ver en la Figura 33, no se realiza esta acción sin comprobar que la conexión a la base de datos es correcta o si la sentencia está bien escrita. Esto es gracias a la excepción DAOException.
- › Una vez ejecutada la query, desde el controlador se llama al JSP **resultado** para mostrar los valores que el usuario ha introducido. La dirección desde la que se invoca a este método es **/aplicación/insertaAplicacion** tal y como se puede ver en la siguiente figura:

```

@RequestMapping(value = "/insertaAplicacion", method = RequestMethod.POST)
public String insertaAplicacion(@ModelAttribute("SpringWeb")Aplicacion aplicacion, ModelMap model) throws ParseException{

    String page = "";

    model.addAttribute("fecha", aplicacion.getFecha());
    model.addAttribute("clave", aplicacion.getClave());
    model.addAttribute("valor", aplicacion.getValor());

    logger.info("Fecha: " +aplicacion.getFecha());
    logger.info("Clave: " +aplicacion.getClave());
    logger.info("Valor: " +aplicacion.getValor());

    try{
        if(aplicacion.getFecha()==null || aplicacion.getFecha()!=null) {
            if(aplicacion.getClave()==null || (aplicacion.getClave()!=null && aplicacion.getClave().length()==0)) throw new UserException("La clave no puede ser nula");
            if(aplicacion.getValor()==null || (aplicacion.getValor()!=null && aplicacion.getValor()<0)) throw new UserException("El valor no puede ser nulo o negativo");
            aplicacionesDAO.insertAplicacion(aplicacion.getFecha(), aplicacion.getClave(), aplicacion.getValor());
        }
    }
    catch (DAOException e){
        logger.error("Error cargando las apps",e);
    }
    catch (UserException e){
        logger.error("Error de parametro incorrecto",e);
    }
    page = "resultado";
    return page;
}

```

Figura 36: Método insertaAplicación de la clase AplicacionController donde se devuelve el JSP resultado con los valores insertados por el usuario.

5.2.2 Adaptador para la visualización de gráficos

De la misma manera, el diagrama de clases que puede resumir la estructura para el correcto funcionamiento del segundo adaptador de la librería es el siguiente:

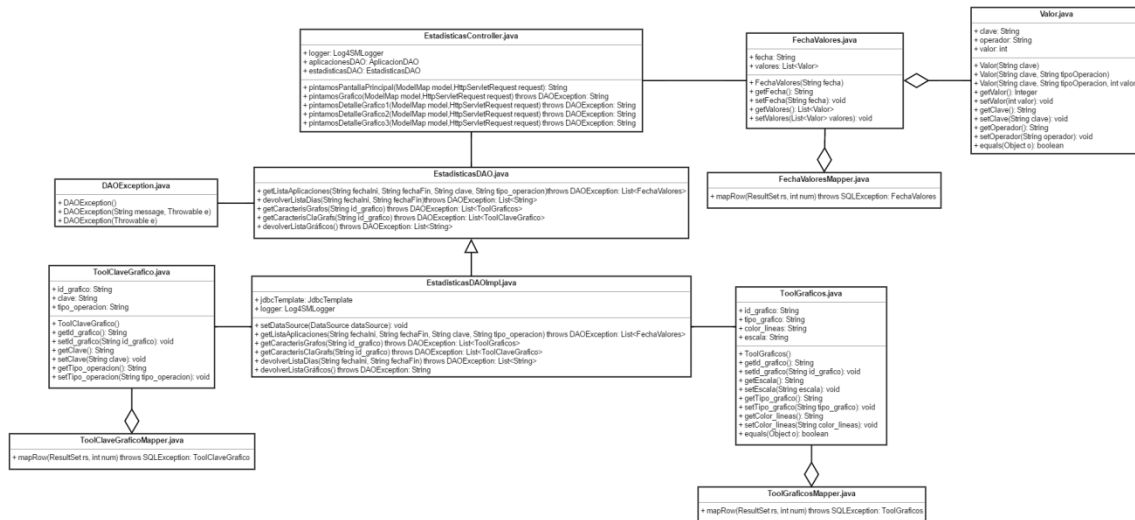


Figura 37: Diagrama de clases para la visualización de los gráficos.

En esta parte del proyecto, las funciones se describen en los siguientes puntos:

1. Habilitar la dirección para invocar a la pantalla principal de la aplicación.
2. Habilitar la dirección con la que se invocará a un método donde se hacen todos los cálculos con la información de la base de datos. Que luego se pasan a la capa de presentación para pintar cada uno de los gráficos.
3. Habilitar la dirección que invoca a un método con el que se llama al JSP que pinta el detalle de cada uno de los gráficos, que se ven en la página principal.

Como pasa en el primer adaptador, la clase principal de esta parte del proyecto es el controlador llamado **EstadisticasController**. De la misma forma que en **AplicacionController**, se define una dirección general con la que se invoca este controlador:

```

@Controller
@RequestMapping("/index")
@EnableWebMvc
public class EstadisticasController {

```

Figura 38: El usuario puede acceder a los métodos del controlador desde el servidor, con la dirección /index si no se habilitan otras.

Para la primera funcionalidad:

- › Se crea un método llamado **pintamosPantallaPrincipal** que utiliza la misma dirección general del controlador **/index** para llamar al JSP **dashboard**, que muestra la pantalla principal de la aplicación:

```
@RequestMapping(value="/index")
public String pintamosPantallaPrincipal(ModelMap model,HttpServletRequest request){

    return "dashboard";
}
```

*Figura 39: Método **pintamosPantallaPrincipal** al que se accede desde el servidor con la dirección **/index**.*

- › Se tiene que saber que para que esta primera funcionalidad se lleve a cabo, primero tiene que funcionar correctamente el método **pintamosGrafico** de este controlador.

Para la segunda funcionalidad:

- › Antes de explicar en detalle esta parte del proyecto, se tiene que mencionar que es aquí donde se encuentra la lógica más relevante de la aplicación y que se obvian las conexiones a base de datos.
- › Lo primero que se hace es crear un método llamado **pintamosGrafico** donde se recuperan tres valores desde la url que la invoca. Estos tres valores son **fechaInicio**, **fechaFin** e **idGrafico**. Cuando se obtienen estas variables primero se comprueba que no sean nulas. Una vez estas tres variables ya son conocidas, con estas se llaman a diferentes métodos, que ejecutan las diferentes queries que servirán para realizar cálculos necesarios. Como se verá también cuando se explique la capa de presentación **idGrafico** sirve para determinar qué gráfico se muestra y determinar sus características como la escala del gráfico. Y **fechaInicio** y **fechaFin** para determinar el rango de los gráficos.
- › Con el valor del **idGrafico**, que como el nombre lo sugiere representa el identificador del gráfico, se llama al método **getCaracterisClaGrafis**. Este método se encuentra en la clase **EstadisticasDAOImpl** y sirve para obtener una lista de tipo **ToolClaveGrafico**.

- › La clase **ToolClaveGrafico** es un bean como **Aplicación** del primer adaptador. Tiene tres variables que mostrarán, el **id_grafico**, la **clave** y el **tipo_operación** de la tabla **FRONTEND_CLAVEGRAFICO** de la base de datos.
- › El método **getCaracterisClaGrafts**, ejecuta la query cada vez que se pasa un **id_grafico** diferente, desde la invocación al método **pintamosGrafico** del controlador. Y como se puede ver en la Figura 39, se utiliza la clase **ToolClaveGraficoMapper** para el mapeo de las columnas que hay en la tabla y la query de **getCaracterisClaGrafts**.

```

public List<ToolClaveGrafico> getCaracterisClaGrafts(String id_grafico) throws DAOException{
    String query = "SELECT f.ID_GRAFICO, f.CLAVE, f.TIPO_OPERACION " +
                  "FROM IOBTBL1.FRONTEND_CLAVEGRAFICO f " +
                  "WHERE f.ID_GRAFICO=?" +
                  "ORDER BY f.ID_GRAFICO, f.CLAVE, f.TIPO_OPERACION";

    List<ToolClaveGrafico> listCarClaGraf = null;

    try{
        listCarClaGraf=jdbcTemplate.query(query, new ToolClaveGraficoMapper(), id_grafico);
    }
    catch (InvalidResultSetAccessException e){
        throw new DAOException(e);
    }
    catch (DataAccessException e){
        throw new DAOException(e);
    }
    return listCarClaGraf;
}

```

*Figura 40: Método **getCaracterisClaGrafts** que ejecuta una query para extraer la clave y el tipo de operación que tiene un determinado **id_grafico**, que se pasa como parámetro.*

- › Las clases Mapper siguen la misma estructura y tienen la misma funcionalidad. Sirven para recoger las filas que coincidan por las columnas que se indican. Cuando el nombre de las columnas no coincide salta la excepción de error en la sintaxis de la sentencia SQL. Como todas las clases Mapper tienen el mismo esquema, se va explicar una. En este caso, **ToolClaveGraficoMapper** que recoge las filas contenidas en las columnas **id_grafico**, **clave** y **tipo_operación**. Como se puede ver en la siguiente figura, aquí también se indica la relación con la clase **ToolClaveGrafico**, que guarda las tres columnas en tres variables del mismo tipo.
- › Con esta lista se podrá determinar qué tipo operación se debe realizar cuando se vaya a calcular el **Valor** de cada **Clave**.

```

public class ToolClaveGraficoMapper implements RowMapper{
    public ToolClaveGrafico mapRow(ResultSet rs, int num) throws SQLException{
        ToolClaveGrafico a = new ToolClaveGrafico();
        a.setId_grafico(rs.getString("id_grafico"));
        a.setClave(rs.getString("clave"));
        a.setTipo_operacion(rs.getString("tipo_operacion"));
        return a;
    }
}

```

Figura 41: ToolClaveGraficoMapper donde se ve el mapeo de 3 columnas.

- > Cuando ya se tiene la lista que contiene las características del gráfico con el **id_grafico** que se haya pasado al invocar al método **pintamosGrafico**, se extrae otra lista al llamar a otro método de **EstadisticasDAOImpl**. Esta vez no se trata de una query sino de una lista de tipo **String** que servirá para guardar el total de días que hay entre la **fechaInicio** y **fechaFin**. Ambos parámetros se conocen cuando el método lee la dirección url con la que se invoca al método.
- > El método con el que se obtiene esta lista de días es **devolverListaDias** y sirve para obtener el rango de fechas en el que se va a mostrar los gráficos:

```

startDate = (Date)formatter.parse(fechaIni);
endDate = (Date)formatter.parse(fechaFin);

```

Figura 42: En el método devolverListaDias los parámetros fechaIni y fechaFin se pasan al formato de fecha que interesa.

```

Calendar calendar = new GregorianCalendar();
calendar.setTime(startDate);

Date result = calendar.getTime();
listaDias.add(formatter.format(result));
do {
    calendar.add(Calendar.DATE, 1);
    result = calendar.getTime();
    listaDias.add(formatter.format(result));
} while (calendar.getTime().before(endDate));

```

Figura 43: Código del método devolverListaDias donde se puede ver cómo se van añadiendo las fechas en la listaDias, hasta llegar a la fechaFin.

- > Lo siguiente que se hace después de obtener la lista de días, es conseguir otra lista de tipo **ToolGraficos**. Esta lista se extrae con la llamada al método **getCaracterisGrafos** de la clase **EstadisticasDAOImpl** y pasando como parámetro el **id_grafico**.

- › En este método **getCaracterisGrafos** se ejecuta otra query con la que se extraen las filas de la tabla **FRONTEND_GRAFICOS**. Este método se ejecuta por cada **id_grafico** de cada invocación que se hace. La lista sirve para obtener características de un gráfico como el **tipo de gráfico** que es el que determina qué JSP corresponde a ese **id_grafico**.

```

public List<ToolGraficos> getCaracterisGrafos(String id_grafico) throws DAOException{
    String query = "SELECT f.ID_GRAFICO, f.TIPO_GRAFICO, f.COLOR_LINEAS, f.ESCALA " +
        "FROM IODBTBL1.FRONTEND_GRAFICOS f " +
        "WHERE f.ID_GRAFICO=?" +
        "ORDER BY f.ID_GRAFICO";

    List<ToolGraficos> listCarGraf = null;

    try{
        listCarGraf=jdbcTemplate.query(query, new ToolGraficosMapper(), id_grafico);
    }
    catch (InvalidResultSetAccessException e){
        throw new DAOException(e);
    }
    catch (DataAccessException e){
        throw new DAOException(e);
    }
    return listCarGraf;
}

```

*Figura 44: Método **getCaracterisGrafos** de la clase **EstadisticasDAOImpl** que devuelve la **listaCarGraf** por cada **id_grafico** que se pase en la invocación.*

- › Una vez esto, por cada una de las **Claves** que contiene la lista sacada en el método **getCaracterisClaGraf**, se extrae otra lista en el método **getListasAplicaciones**. Esta nueva lista es de tipo **FechaValores** y es con la que se rellena el objeto, que se usa para mostrar los gráficos.
- › **FechaValores** es otro bean como los anteriores y es sin duda el más importante ya que, con este se rellenan los datos que solicitan los gráficos. Está formado por dos atributos: **fecha** y una lista de tipo **Valor**. Esta última lista es la que contiene cada una de las instancias **Clave**, **Operador** y **Valor** que hay para cada **fecha**.
- › Como se puede ver en la Figura 44, la query del método **getListasAplicaciones** solicita una **fechaIni**, una **fechaFin**, una **clave** y un **tipo_operacion**. Las fechas son para seleccionar sólo los valores de las claves que hayan para ese rango de tiempo, y la clave y operación para realizar un tipo de cálculo u otro en función de la clave.

```

public List<FechaValores> getListaAplicaciones(String fechaIni, String fechaFin, String clave, String tipo_operacion) throws DAOException{
    String query = "";
    if(tipo_operacion=="avg"){
        query = "SELECT to_char(f.FECHA,'dd/mm/yyyy') as FECHA, f.CLAVE, " + "\" + \" + tipo_operacion + \"\ as OPERADOR, round(" + tipo_operacion + "(f.VALOR),0) as VALOR " +
        "FROM IOBFTBL1.FRONTEND_METRICAS f " +
        "WHERE f.FECHA BETWEEN to_date(?, 'dd/MM/yyyy') and (to_date(?, 'dd/MM/yyyy')+1) AND f.CLAVE=? " +
        "GROUP BY to_char(f.FECHA,'dd/mm/yyyy'), f.CLAVE " +
        "ORDER BY FECHA ASC, f.CLAVE";
    }
    else{
        query = "SELECT to_char(f.FECHA,'dd/mm/yyyy') as FECHA, f.CLAVE, " + "\" + \" + tipo_operacion + \"\ as OPERADOR, " + tipo_operacion + "(f.VALOR) as VALOR " +
        "FROM IOBFTBL1.FRONTEND_METRICAS f " +
        "WHERE f.FECHA BETWEEN to_date(?, 'dd/MM/yyyy') and (to_date(?, 'dd/MM/yyyy')+1) AND f.CLAVE=? " +
        "GROUP BY to_char(f.FECHA,'dd/mm/yyyy'), f.CLAVE " +
        "ORDER BY FECHA ASC, f.CLAVE";
    }
    List<FechaValores> list = null;
    try{
        list=jdbcTemplate.query(query, new FechaValoresMapper(), fechaIni, fechaFin, clave);
    }
    catch (InvalidResultSetAccessException e){
        throw new DAOException(e);
    }
    catch (DataAccessException e){
        throw new DAOException(e);
    }
    return list;
}

```

Figura 45: Método `getListaAplicaciones` con la que se obtienen todos los valores de las claves que contienen cada uno de los gráficos.

- > El siguiente paso después de obtener las cuatro listas necesarias es crear el objeto final, el que utilizan los gráficos para poder mostrarse en la aplicación. Este objeto es de tipo **FechaValores** y se rellena con la lista sacada en el método **getListaAplicaciones** pero, teniendo en cuenta todas las claves que hayan, dado un **id_grafico**.
- > Además de tener en cuenta todos los valores de todas las claves, también se rellena el objeto siguiendo el orden y el total de fechas que haya en la lista de días. Esta forma de completar el objeto se hace para evitar, que las fechas en las que no haya valores se ignoren y entonces, los gráficos no sean coherentes.
- > Además de que es la manera más correcta de cómo guardar la cantidad de claves con sus respectivos valores y tipo de operación que puede haber para una misma fecha.
- > Como se puede ver en la Figura 45, se recorren todas las listas extraídas para comparar cada una de las claves que hay guardadas en la **listaClavGraf** con cada una de las claves de la lista sacada de base de datos en **getListaAplicaciones**. De esta manera, se completa el objeto sin alterar ninguna información. Se comprueba, básicamente, que se trate de la misma **Fecha**. Y cuando sean iguales, se comprueba si se trata de la misma **Clave** y **Operador**.

- › Esta comparación que se hace, es posible gracias al método **equals** que hay en el bean **Valor**, donde se define la comparación de **Clave** y **Operador** entre dos objetos de tipo **Valor**.

```

List<FechaValores> listaApps = new ArrayList<FechaValores>();

//Relleno mi objeto
for(int i=0; i<listaDias.size(); i++){

    List<Valor> valores = new ArrayList<Valor>();
    FechaValores fval = new FechaValores(listaDias.get(i));

    for (int j=0; j<listaClavGraf.size(); j++){

        Valor vals = new Valor(listaClavGraf.get(j).getClave(), listaClavGraf.get(j).getTipo_operacion());

        //Se busca fecha, clave y operador en listaAplicaBBDD
        for(int k=0; k<listaAplicaBBDD.size(); k++){

            //Se hace una copia de la base de datos para recorrer cada uno de los días por cada uno de los pares de Valor
            FechaValores fvbd=listaAplicaBBDD.get(k);

            //Primero se comprueba que estamos buscando esa fecha
            if(fvbd.getFecha().equals(listaDias.get(i))){

                //Se hace una copia de cada uno de los pares de Valor que hay cada día
                List<Valor> lv = fvbd.getValores();

                for(int g=0; g<lv.size(); g++){

                    //Y si además es la misma clave y el mismo operador se añade a la lista listaApps
                    if(lv.get(g).equals(vals)){

                        vals.setValor(lv.get(g).getValor());

                    }

                }

            }

        }

        valores.add(vals);

    }

    fval.setValores(valores);
    listaApps.add(fval);
}

```

Figura 46: Parte del código que sirve para rellenar el objeto con el que se pintarán los gráficos de la aplicación.

- › Una vez rellenado el objeto, sólo queda pasárselo a la capa de presentación y eso se hace con las siguientes líneas:

```

model.addAttribute("estadísticas", listaApps);

return tipoGrafico;

```

Figura 47: Se pasa el objeto listaApps al JSP que toque dependiendo del id_grafico que sea. Este JSP es el tipoGrafico que se extrae de la lista

- › Este JSP es el **tipoGrafico** que contiene la lista extraída en el método **getCaracterisGrafos** para un determinado **id_grafico**.

Para la tercera funcionalidad:

- › Se crea un método al que se invoca accediendo a la dirección **/index/detalle**.
- › Se extrae una lista, de tipo **String**, con la query **devolverListaGráficos** que está en **EstadisticasDAOImpl**. Esta lista contiene todos los gráficos que hay en la base de datos y sirve para devolver un JSP con el detalle del **id_grafico** que se indique.

```
@RequestMapping(value="/detalle")
public String pintamosDetalleGrafico(ModelMap model,HttpServletRequest request) throws DAOException{

    String page = "";
    String width, heigth, align;
    List<String> listaGraficos = estadisticasDAO.devolverListaGráficos();

    int k = listaGraficos.indexOf("totalSMSEnviadosPorOperadora");

    width = "88%";
    heigth = "500";
    align = "left";

    model.addAttribute("idGrafico", listaGraficos.get(k));
    model.addAttribute("width", width);
    model.addAttribute("height", heigth);
    model.addAttribute("align", align);

    page = "detalle";

    return page;
}
```

*Figura 48: Método del controlador que devuelve el JSP que contiene el detalle del gráfico con cada **id_grafico** que se pasa. En este ejemplo, el JSP que se devuelve ha de ser el del **id_grafico=totalSMSEnviadosPorOperadora**.*

- › Como se puede ver en la Figura 47, se pasan parámetros como el ancho y alto del gráfico.

5.3 Otras características

Como se ha explicado en la primera parte de la memoria, se utiliza un método para el control de las ejecuciones, el de la extracción de **logs**. Cada uno de estos logs se pueden ver en un fichero llamado **frontend.log**.

5.3.1 Fichero logs

Para poder actualizar este fichero en cada ejecución o cada vez que se quieran revisar métodos del proyecto Java, se debe configurar las clases y el **web.xml** de la siguiente manera:

- › Primero se crea una clase, la única que no se ha comentado en los adaptadores. Esta clase se llama **Init.java** y sirve para escuchar durante la ejecución.

```
public void contextInitialized(ServletContextEvent event) {  
  
    ServletContext ctx= event.getServletContext();  
    Log4SMConfigurator.reloadLog(ctx.getInitParameter("initLogs"), null);  
  
}
```

Figura 49: Método de la clase Init donde se declara initLogs.

- › Otro de los ficheros del proyecto Java donde se debe avisar que se usará este método es el **web.xml**:

```
<listener>  
    <description>Listener de spring</description>  
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  
</listener>  
  
<listener>  
    <description>Listener de inicializacion</description>  
    <listener-class>es.arqdes.listener.Init</listener-class>  
</listener>  
  
<context-param>  
    <param-name>initLogs</param-name>  
    <param-value>/shared/apps/frontend/conf/log4sm_online.xml</param-value>  
</context-param>
```

Figura 50: Se avisa que initLogs de la clase Init será el que escuchará si hay qué logs hay en cada ejecución.

- › En el fichero log4sm_online.xml se indica la ubicación del fichero de logs **frontend.log**.

Para acabar este apartado, no se ha explicado el uso de las librerías pero, durante todo el proyecto Java ha habido invocaciones a tantas como se han necesitado. La mayoría de estas para son de tipo **org.springframework** debido la implementación del MVC con el Framework de Spring.

6. Desarrollo página web

Ahora ya sólo queda la última parte, la que realmente tiene valor para el usuario ya que, es la que puede ver. Por esto, esta parte ha de ser sencilla y atractiva para que su uso sea un éxito.

En el apartado de Anexos de la memoria se pueden ver las pantallas de la aplicación.

6.1 Diseño

Desde el principio se ha buscado que esta aplicación sea lo más flexible posible para que sea útil en diferentes ámbitos o proyectos dentro de AOD. Se ha intentado al máximo que esto sea así y por eso se ha elegido la librería para **Javascript d3.js**.

La aplicación está basada en **HTML** contenido en ficheros **JSP** que se llaman desde la librería.

Como no tenía la experiencia suficiente en diseño me he ayudado de la Framework **Bootstrap** que también está basada en el lenguaje **Javascript** y por tanto **CSS**.

6.1.1 HTML

La aplicación mucha interacción ya que, desde su origen siempre se ha dicho que el objetivo es que sea informativa. Por esta razón, tampoco hay un uso exhaustivo del lenguaje HTML.

En el proyecto a día de hoy, hay un total de 8 JSP. De todos estos, sólo 4 JSP son fijos, el resto son los JSP que contienen los gráficos que se van a mostrar. Se dice que son fijos, no porque no vayan a cambiar sino porque son lo que estarán siempre. De los otros JSP puede haber tantos como se quieran mostrar en la aplicación.

- › **cargaAplicacion.jsp**: es un formulario con tres campos. Cuando el usuario pulsa el botón de “**Insertar**” invoca al método donde los valores introducidos, se cargan en la base de datos.

```
<form name="formAddApp" method="post" action="./insertaAplicacion">
```

Figura 51: Se direcciona al método con url ./insertaAplicacion.

- › **resultado.jsp**: es otro formulario. Con este se muestran los valores que se han insertado en **cargaAplicacion.jsp**. También tiene un botón que al pulsar se vuelve al JSP **cargaAplicacion** para que se introduzcan nuevos datos.

- › **dashboard.jsp**: es la página principal de la aplicación. En esta se muestran todos los gráficos y un menú lateral con el que se puede volver a la página principal desde cualquier otra. En este menú se puede encontrar también la opción de ver el detalle de cada gráfico o la opción de insertar un nuevo valor.

Los gráficos de esta página están contenidos en objetos llamados **iframe**. Estos **iframe** no son nada más complicado que otras páginas y la ventaja de usarlas es que se pueden tener tantas como se quiera.

```
<iframe name="ifc1" id="ifc1" src="http://localhost:7001/front4/Indes/grafico?idOrafico=PlantillasPorMaquina" width=88% height=500 align="right" scrolling="no" frameBorder="0"></iframe>
```

Figura 52: Iframe que muestra el gráfico con identificador PlantillasPorMaquina.

Con el **src** o dirección que se indica en cada iframe, se invoca al método con url **./grafico**, que si recordamos es **pintamosGrafico**.

En esta página el usuario puede modificar las fechas en las que quiere ver los gráficos. Para hacerlo sólo tiene que rellenar un formulario que se prepara con dos campos: Fecha Inicio y Fecha Fin.

- › **detalle.jsp**: igual que la página principal, para mostrar el detalle de los gráficos se utiliza el **iframe** aunque, esta vez sólo habrá uno. En esta página también se puede modificar el rango de fechas.
- › **graficoLineal**, **graficoDinamico**, **graficoHistograma**, **graficoHistograma2**: aunque se han modificado según el interés, d3.js ya hace el mayor trabajo por los gráficos.

6.1.2 D3, Javascript y CSS

D3 permite:

- › Dibujar gráficos dinámicos con los que se pueden interactuar. Pueden cambiar de color al situar el ratón en una posición o mostrar el valor exacto al poner el ratón encima de la línea.
- › Insertar leyendas o botones con los que se puede cambiar el aspecto del gráfico.
- › Cambiar de un tipo de gráfico a otro de forma sencilla y sin tener que modificar mucho código.

- › El formato de los datos con los que se muestran los gráficos depende de si se trata de un gráfico lineal, de un histograma o de otro tipo de gráfico pero, para la mayoría se ha podido definir uno general.

Con Javascript:

- › Se construye el formato del objeto con el que trabajan los gráficos. Como ya sabemos estos datos los proporciona la capa de negocio y se recuperan desde el JSP de los gráficos de la siguiente manera:

```

<%
String array = "";
List estadist = (List) request.getAttribute("estadisticas");

for(int i=0; estadist!=null && i<estadist.size(); i++){

    FechaValores app = (FechaValores) estadist.get(i);
    array += "{fecha:\"" +app.getFecha() + "\", ";
    for(int j=0; app!=null && j<app.getValores().size(); j++){
        array += app.toGraficoGeneral();
    }
    array += "}, ";
}
%>
var data = [<%=array%>];

```

Figura 53: Array es el objeto que se recupera de la librería.

- › Como se puede ver en la Figura 55, la lista de tipo FechaValores se recupera con el `request.getAttribute("estadisticas")`. Si volvemos a la librería podemos ver que "estadisticas" es como se guarda la lista de **FechaValores**.
- › Otra de las cosas que permite Javascript es la actualización del **dashboard** y del **detalle** cada vez que se insertan las fechas en el formulario para volver a obtener los datos y mostrarlos en el nuevo rango.

Con **CSS** se definen los colores o los tamaños de los gráficos.

6.1.3 Bootstrap

Es un framework basado en Javascript y CSS que en este proyecto va a servir para dar todo el aspecto a cada una de las páginas de la aplicación. Se ha elegido el estilo **Admin & Dashboard**.

7. Inversión y ahorro

En esta parte de la memoria se va explicar la inversión y el ahorro que supone la realización de este proyecto para la empresa.

Para saber el valor de dicha inversión, se calculará el coste directo de la aplicación teniendo en cuenta dos puntos del acuerdo entre universidad y empresa:

- › Por un lado, el salario pactado en dicho convenio.
- › Y por otro lado, el período que marca el mismo convenio.

Finalmente, para mostrar el ahorro que implica el proyecto no sólo se tiene en cuenta la parte monetaria sino, también el ahorro de trabajo para Everis.

7.1 Inversión financiera

Al tratarse del desarrollo de una aplicación, los costes de un proyecto como este se resumen en la dedicación en horas, que realiza el programador hasta su fin.

En este caso, la aplicación ha sido desarrollo por mí a tiempo parcial (4 horas diarias) y de acuerdo al convenio con la universidad, esta jornada supone un coste directo para Everis de 517.44€ al mes.

Si el período del convenio para el desarrollo de la aplicación, ha sido por 3 meses y medio, el coste total se puede calcular de la siguiente manera:

$$(517.44 \cdot 3) + 210.81\text{€} = 1763.13\text{€}$$

Está claro que un programador senior o uno con más experiencia, podría haber hecho la misma aplicación en menos tiempo. En contraposición, el salario de este programador senior habría sido mayor.

Otra de las opciones por las que Everis también se habría podido decidir, habría sido dividir los roles dentro de la aplicación y llevar a cabo el proyecto entre diferentes personas. Y como pasa con el programador senior, los costes habrían sido mayores aunque, el período habría más corto.

En resumen, podemos decir que es una buena inversión para la empresa y para mí, como estudiante, ambos hemos alcanzado los resultados esperados.

7.2 Ahorro

Si hasta ahora desde AOD se pueden generar estadísticas de forma automática, con esta aplicación ya no será necesario que el usuario interesado (en este caso el cliente al que se quiere hacer la propuesta), tenga que esperar a fin de mes para acceder a estos. Podrá ver el rendimiento de su sistema en cualquier momento.

De esta forma, aunque no con el mismo detalle, la aplicación implica ahorro de tiempo en trabajadores ya que, hay meses en los que un cliente quiere la extracción de estas estadísticas más de dos veces. Con esto, se ahorra aproximadamente 4 horas por proyecto o estadística a generar. Actualmente, AOD tiene 5 proyectos de los que extraer métricas, lo que significa que se ahorra 20 horas al mes. Tiempo que se puede dedicar por completo al proyecto. Si se tiene en cuenta que el salario promedio de los trabajadores en Everis es de 9€ por hora. Entonces, el ahorro cada mes en Everis se puede calcular de la siguiente manera:

$$20 \cdot 9\text{€} = 180\text{€}$$

Si además, le sumamos el ahorro del salario extraído antes, se confirma que hemos ganado las dos partes.

8. Conclusiones

Pienso que es muy importante leer esta pequeña conclusión para entender realmente el proyecto. Es cierto que la parte fundamental se encuentra en el desarrollo de la aplicación y en la redacción de esta memoria. Pero detrás de todo esto, también está el esfuerzo y las ganas con las que se han realizado. Desde el comienzo de este proyecto, las percepciones han sido buenas y así ha acabado.

Se han conseguido los objetivos que se han propuesto tanto desde AOD, como los que personalmente me había definido al inicio del proyecto. He utilizado software y herramientas de programación que me han ayudado a mejorar mis habilidades y han aumentado mi valor en el mercado de trabajo.

Otros conocimientos que he aprendido con la implementación de este proyecto en la compañía han sido: la gestión del tiempo, la importancia de mi trabajo dentro de la empresa, el informe del estado del mismo y la comunicación con el responsable del proyecto en AOD (Daniel Cufi).

Durante el desarrollo se han superado diferentes problemas. El primero con el que se tropezó fue durante la conexión a la base de datos ARQDES, desde la consola Weblogic. Aunque fue uno de los problemas más pequeños, siendo la primera aplicación a desplegar había pasos intermedios que yo como programadora desconocía, y esto hacía el comienzo más difícil.

Cuando la conexión ya estaba controlada, recuerdo que hubo tres momentos más, que provocaron retrasos en el desarrollo. El primero fue cuando se preparaba la primera página de la aplicación, para habilitar la entrada de datos y después almacenarlos en las tablas. El segundo fue durante la creación del objeto que utilizan los gráficos para pintarse. Era tan complejo que inicialmente, sólo permitía mostrar una clave por cada gráfico. Daniel Cufi supo cómo ayudarme y se pudo solucionar y quizá fue este uno de los momentos más satisfactorios del proyecto. Y ya el otro retraso en el proyecto, fue averiguando cómo cambiar el rango de fechas desde la aplicación sin que esta afecte al resto de código.

8.1 Objetivos alcanzados

Siguiendo el modelo unificado se ha desarrollado una página web capaz de mostrar una serie de gráficos informativos con los que un usuario puede interactuar.

Los objetivos alcanzados han sido los que se han propuesto desde AOD:

- › La capa de negocio o librería es capaz de calcular a partir de la información, que hay en la base de datos, todos los valores que necesita la capa de presentación.

- › La capa de datos es capaz de procesar y gestionar una cantidad de información, lo suficientemente grande, sin que suponga una bajada de rendimiento.
- › La capa de presentación es capaz de mostrar los gráficos de forma dinámica y permite al usuario poder interactuar con ellos. Da la oportunidad también de filtrar las fechas en las que se muestran los gráficos y ver el detalle de estos.

8.2 Objetivos de mejora

A pesar de los resultados, cuando un proyecto va por buen camino siempre se pide más. Los siguientes puntos pueden resumir las partes que se van a mejorar:

- › Desde la capa de datos se quiere crear otra tabla que contenga N dashboards por usuario y cada uno de estos a su vez, N gráficos. Desde estos N gráficos ya se conoce la estructura porque es la que se tiene actualmente. De manera gráfica se quiere:

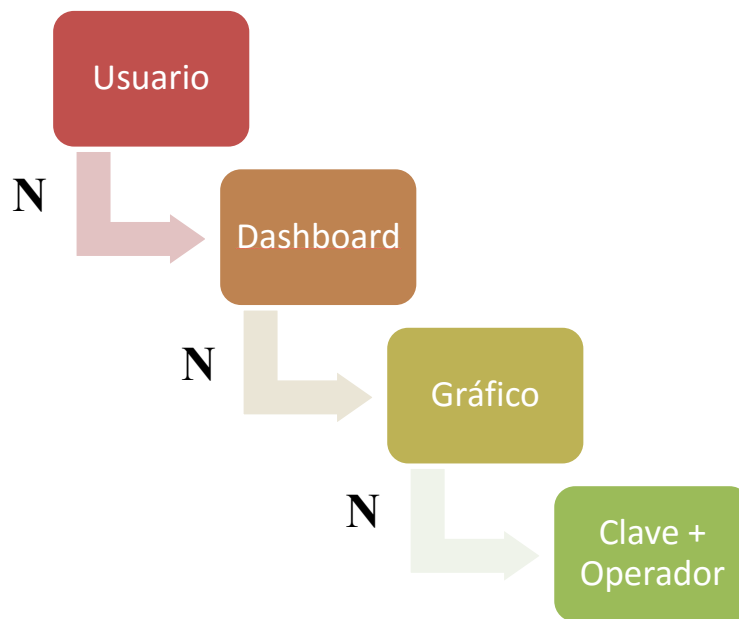


Figura 54: Modelo de base de datos que se ha diseñado como mejora del proyecto.

- › Desde la capa de negocio se tendría que aplicar esta nueva tabla para el cálculo de los datos a pasar a la capa de web.

- › La capa de presentación aplicaría los cambios de la base de datos. Además de esta modificación se quiere conseguir más dinamismo en los datos que aparecen en la aplicación. Por ejemplo, se quiere obtener el título de los gráficos, el color de las líneas o barras, la información de las leyendas o los nombres del menú lateral desde la base de datos. Hay partes, a día de hoy, que ya aplican pero, se quiere ampliar al resto.

8.3 Resultados y contribución a Everis

Everis puede presentar el proyecto a los clientes y si estos la aceptan, sería una buena contribución.

Por otro lado, la aplicación ha sido diseñada para poder utilizarla también desde AOD por ejemplo, para el seguimiento del rendimiento de las propias aplicaciones.

8.4 Valoraciones personales

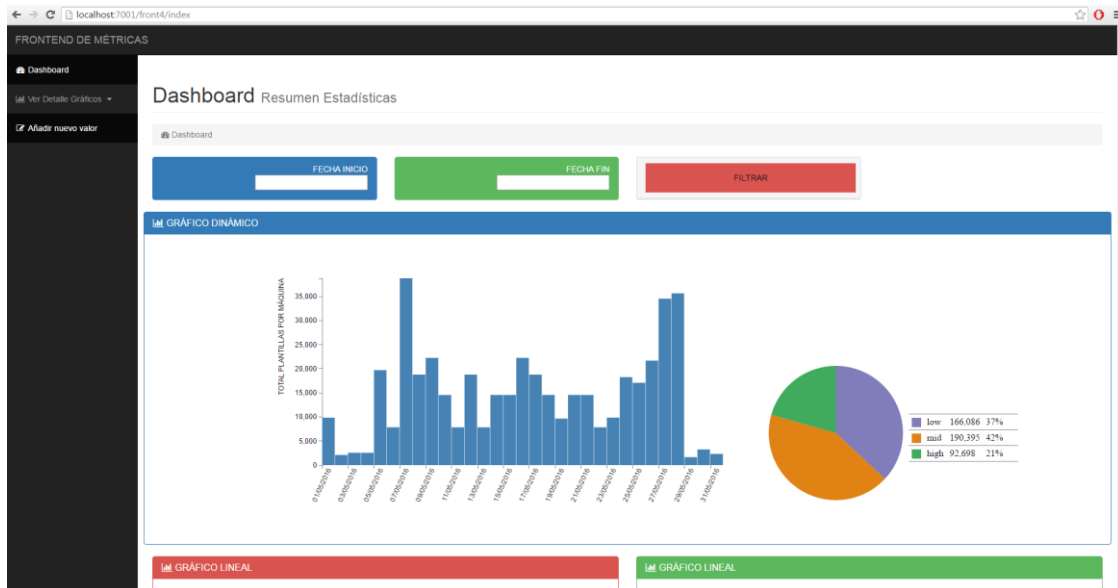
Aunque hay un par de cosas que mejorar, se han conseguido los objetivos marcados. El problema más grande para mí, ha sido que se ha empezado más tarde de como se había planificado. Los motivos han sido meramente a causa del retardo durante el trámite del convenio.

Por todo lo demás ha ido muy bien. Siento una gran satisfacción cuando recuerdo que el proyecto empezó de cero y no se sabía hasta dónde iba a poder llegar.

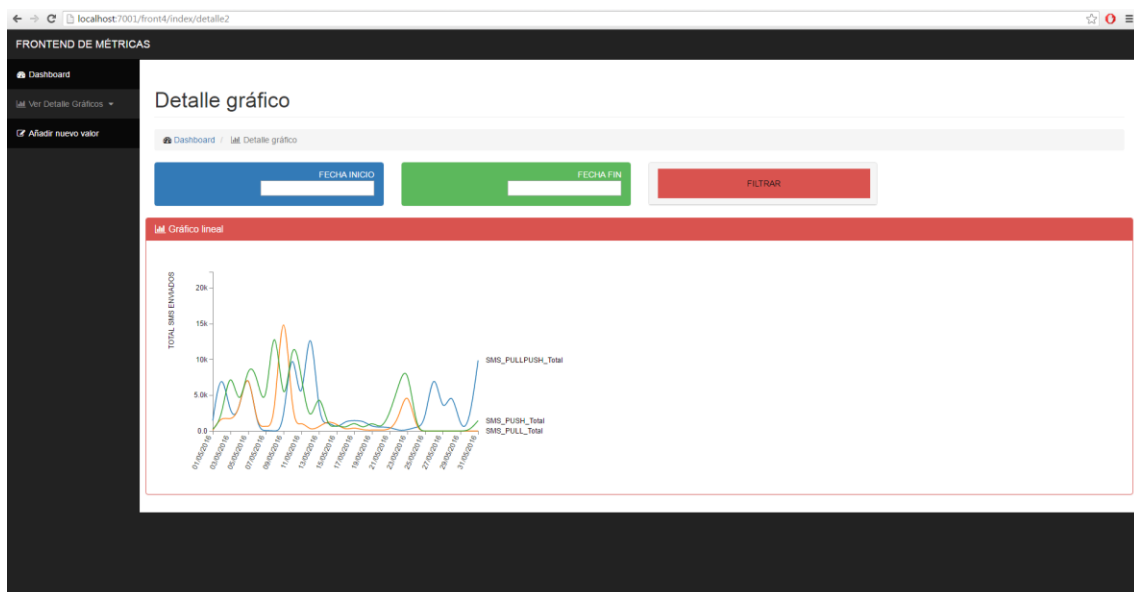
Puedo decir que aunque no se trate de un proyecto espectacular o un proyecto innovador, con el desarrollo de esta aplicación, he superado mis propias expectativas y he aprendido mucho más de lo que esperaba. También ha sido una oportunidad para demostrar a la compañía mis virtudes y por esto, estoy muy agradecida.

9. Anexos

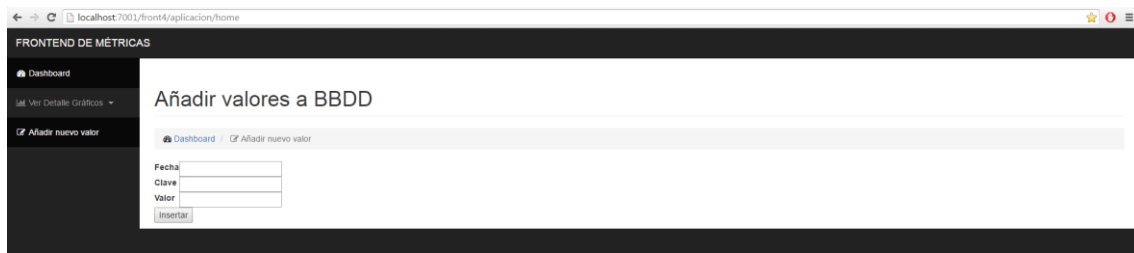
Página principal: Dashboard del Frontend de métricas



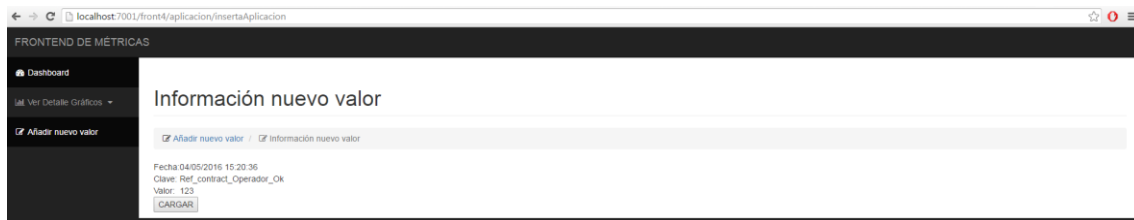
Pantalla con el **detalle** de uno de los gráficos:



Pantalla con el **formulario** desde donde se pueden **insertar valores nuevos**:



Pantalla con el **formulario** con la **información** de los **valores insertados** en la anterior pantalla:



Link a un video en Youtube con una pequeña demostración de las funcionalidades de la aplicación:

<https://youtu.be/TJ1x-a1AYy4>

10. Bibliografía

- [1] The Everis Group (2014). Consulting, IT & Outsourcing Professional Services [en línea] <<http://www.everis.com/spain/es-ES/inicio/Paginas/inicio.aspx>> [Consulta: 16 Mar 2016].
- [2] Milena Rincon, M., stalingrado, P, M. y Quiroga, A. (2015). Proyecto de Grado Ingeniería de Sistemas: UWE UML (UML-Based Web Engineering) [en línea] <<http://proyectogradoingenieriasistemas.blogspot.com.es/2015/03/metodologia-uwe-uml-uml-based-web.html>> [Consulta: 24 Mar, 2016].
- [3] Es.wikipedia.org, Elvisor (2015). Programación por capas [en línea] <https://es.wikipedia.org/wiki/Programaci%C3%B3n_por_capas> [Consulta: 22 Abr 2016].
- [4] Oracle and/or its affiliates (2010). Packaging Applications - The Java EE 5 Tutorial [en línea] <<http://docs.oracle.com/javaee/5/tutorial/doc/bnaby.html#indexterm-47>> [Consulta: 22 Abr 2016].
- [5] Es.wikipedia.org, Sabbut (2015). Front-end y back-end [en línea] <https://es.wikipedia.org/wiki/Front-end_y_back-end> [Consulta: 22 Abr 2016].
- [6] Home.cern (2014). The birth of the web [en línea]. <<http://home.cern/topics/birth-web>> [Consulta: 25 Abr 2016].
- [7] W3.org, Berners-Lee, T. (1990). The original proposal of the WWW, HTMLized [en línea] <<https://www.w3.org/History/1989/proposal.html>> [Consulta: 22 Abr, 2016].
- [8] Es.wikipedia.org, Shalbat (2016). Wiki [en línea] <<https://es.wikipedia.org/wiki/Wiki>> [Consulta: 25 Abr 2016].
- [9] Mateu, C., Megías Jiménez, D. and Mas, J. (2004). “Desarrollo de aplicaciones web”. Barcelona: UOC.
- [10] Lamarca Lapuente, M. J. (2013). Hipertexto, el nuevo concepto de documento en la cultura de la imagen [en línea] <<http://www.hipertexto.info/documentos/hipertexto.htm>> [Consulta: 25 Abr 2016].
- [11] W3.org. (2000). A Little History of the World Wide Web [en línea] <<https://www.w3.org/History.html>> [Consulta: 5 May 2016].
- [12] Es.wikipedia.org, Lemilio775 (2016). Historia de la World Wide Web [en línea] <https://es.wikipedia.org/wiki/Historia_de_la_World_Wide_Web> [Consulta: 5 May 2016].

- [13] Moreno, M. (2016). Facebook ya tiene 1.590 millones de usuarios [en línea] <<http://www.trecebits.com/2016/01/28/facebook-ya-tiene-1-590-millones-de-usuarios/>> [Consulta: 11 May 2016].
- [14] The Internet Society (1999). RFC 2660 - The Secure HyperText Transfer Protocol [en línea] <<https://tools.ietf.org/html/rfc2660#page-3>> [Consulta: 11 May 2016].
- [15] Es.wikipedia.org, Jaimes B. (2016). Arquitectura de software [en línea] <https://es.wikipedia.org/wiki/Arquitectura_de_software> [Consulta: 11 May 2016].
- [16] Universidad Simón Bolívar (2007). Arquitectura de Software [en línea] <<http://ldc.usb.ve/~mgoncalves/IS2/sd07/clase7.pdf>> [Consulta: 18 May 2016].
- [17] Lau7821 (2013). Desarrollo de aplicaciones para ambientes distribuidos - Aplicaciones Monolíticas [en línea] <<https://laurmolina7821.wordpress.com/1-1-1-aplicaciones-monoliticas/>> [Consulta: 11 May 2016].
- [18] Es.wikipedia.org, Jesuja (2016). Modelo–vista–controlador [en línea] <<https://es.wikipedia.org/wiki/Modelo%20%93vista%20%93controlador>> [Consulta: 11 May 2016].
- [19] Sun Microsystems (1997). The Java Language Environment [en línea] <<http://www.oracle.com/technetwork/java/intro-142807.html>> [Consulta: 11 May 2016].
- [20] Es.wikipedia.org, Victorciko2 (2016). Programación orientada a objetos [en línea] <https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos> [Consulta: 11 May 2016].
- [21] Oracle Corporation (2016). Conozca más sobre la tecnología Java [en línea] <<https://www.java.com/es/about/>> [Consulta: 13 May 2016].
- [22] Oscar Belmonte Fernández (2004). Introducción al lenguaje de programación Java: Una guía básica [en línea] <<http://www3.uji.es/~belfern/pdidoc/IX26/Documentos/introJava.pdf>> [Consulta: 13 May 2016].
- [23] Morales Machuca, C. (2013). Estado del Arte: Servicios Web [en línea] <<http://www.methesis.fcs.ucr.ac.cr/blogs/rumana/wp-content/uploads/2010/09/articulo2.pdf>> [Consulta: 13 May 2016].
- [24] Johnson, R., Hoeller, J., Donald, D., Sampaleanu, C., Harrop, R., Arendsen, A., Risberg, T., Davison, D., Kopylenko, D., Pollack, M., Templier, T., Vervaeet, E., Tung, P., Hale, B., Colyer, A., Lewis, J., Leau, C., Fisher, M., Brannen, S., Laddad, R.,

Poutsma, A., Beams, C., Abedrabbo, T., Clement, A., Syer, D., Gierke, O., Stoyanchev, R. (2011). Introduction to Spring Framework [en línea] <<http://docs.spring.io/spring/docs/3.1.0.M1/spring-framework-reference/html/overview.html>> [Consulta: 13 May 2016].

[25] Docs.oracle.com (2016). Introduction to the Oracle Database [en línea] <http://docs.oracle.com/cd/B19306_01/server.102/b14220/intro.htm> [Consulta: 13 May 2016].

[26] Docs.spring.io (2016). The IoC container – Part III. Core Technologies [en línea] <<http://docs.spring.io/autorepo/docs/spring/3.2.x/spring-framework-reference/html/beans.html>> [Consulta: 13 May 2016].

[27] TIOBE software BV (2016). TIOBE Index | Tiobe - The Software Quality Company [en línea] <http://www.tiobe.com/tiobe_index?page=index> [Consulta: 15 May 2016].

[28] Cplusplus.com (2016). A Brief Description - C++ Information [en línea] <<http://www.cplusplus.com/info/description/>> [Consulta: 15 May 2016].

[29] Kirk Radeck (2003). *C# and Java: Comparing Programming Languages* [en línea] <<https://msdn.microsoft.com/en-us/library/ms836794.aspx>> [Consulta: 19 May 2016].

[30] Python Software Foundation (2016). Overview – Python 3.5.1 documentation [en línea] <<https://docs.python.org/3/>> [Consulta: 19 May 2016].

[31] Oracle and/or its affiliates (2011). Fusion Middleware Introduction to Oracle WebLogic Server – Introduction to Oracle Weblogic Server [en línea] <http://docs.oracle.com/cd/E23943_01/web.1111/e13752/toc.htm#INTRO108> [Consulta: 19 May 2016].

[32] Oracle and/or its affiliates (2016). Java Platform, Enterprise Edition (Java EE) - Oracle Technology Network - Oracle [en línea] <<http://www.oracle.com/technetwork/java/javaee/overview/index.html>> [Consulta: 21 May 2016].

[33] Es.wikipedia.org, Correogsk (2016). Interfaz de programación de aplicaciones [en línea] <https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones> [Consulta: 21 May 2016].

[34] Techopedia.com (2016). What is an Enterprise Archive File (EAR)? - Definition from Techopedia [en línea] <<https://www.techopedia.com/definition/26370/enterprise-archive-file-ear>> [Consulta: 21 May 2016].

[35] Enrique Gonzales (2016). Orientación sobre el curso "Tutorial básico del programador web: HTML desde cero" (CU00702B) [en línea] <http://www.aprenderaprogramar.es/index.php?option=com_content&view=article&id=422:orientacion-sobre-el-curso-qtutorial-basico-del-programador-web-html-desde-ceroq-cu00702b&catid=69:tutorial-basico-programador-web-html-desde-cero&Itemid=192> [Consulta: 25 May 2016].

[36] Internetlivestats.com (2016). Internet Live Stats - Internet Usage & Social Media Statistics [en línea] <<http://www.internetlivestats.com/>> [Consulta: 25 May 2016].

[37] jsx, galambalazs, PierreNeter, Saliene, Sheppy, xfq, fscholz, claudepache, velvel53, ethertank, markg, Sonrisa, inma_610, Dhtmlkitchen@gmail.com, 99corps, Cnmahj, MookyMa, Pd, George3, Dan Smith, DavidCary, BenoitL, Hillman, Dria (2015). About JavaScript – What is JavaScript? [en línea] <https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript?redirectlocale=en-US&redirectslug=JavaScript%2FAbout_JavaScript> [Consulta: 25 May 2016].

[38] W3schools.com (2016). CSS Tutorial [en línea] <<http://www.w3schools.com/css/>> [Consulta: 25 May 2016].

[39] Bostock, M. (2015). D3.js - Data-Driven Documents [en línea] <<https://d3js.org/>> [Consulta: 25 May 2016].

[40] Oracle and/or its affiliates (2016). Oracle SQL Developer [en línea] <<http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index-097090.html>> [Consulta: 26 May 2016].

[41] Oracle and/or its affiliates (2014). Oracle JDBC Frequently Asked Questions [en línea] <http://www.oracle.com/technetwork/database/enterprise-edition/jdbc-faq-090281.html#00_01> [Consulta: 26 May 2016].

[42] Oracle and/or its affiliates (2014). JDK 5.0 Java Naming and Directory Interface-related APIs & Developer Guides from Sun Microsystems [en línea] <<http://docs.oracle.com/javase/1.5.0/docs/guide/jndi/index.html>> [Consulta: 26 May 2016].

[43] Db-engines.com (2016). System Properties Comparison Microsoft SQL Server vs. Oracle vs. PostgreSQL [en línea] <<http://db-engines.com/en/system/Microsoft+SQL+Server%3BOracle%3BPostgreSQL>> [Consulta: 29 May 2016].

[44] Microsoft.com (2016). SQL Server 2016 | Microsoft [en línea] <<https://www.microsoft.com/en-us/server-cloud/products/sql-server/>> [Consulta: 29 May 2016].

[45] The PostgreSQL Global Development Group (2016). PostgreSQL: Documentation: 9.5: What is PostgreSQL? [en línea] <<https://www.postgresql.org/docs/9.5/static/intro-what-is.html>> [Consulta: 3 Jun 2016].

[46] Es.wikipedia.org, Ks-M9 (2016). Diagrama de casos de uso [en línea] <https://es.wikipedia.org/wiki/Diagrama_de_casos_de_uso> [Consulta: 3 Jun 2016].

[47] Es.wikipedia.org, Technopat (2016). Java Servlet [en línea] <https://es.wikipedia.org/wiki/Java_Servlet> [Consulta: 3 Jun 2016].

11. Glosario

NTT DATA: Everis pertenece a una compañía Japonesa de Comunicaciones controlada por Nippon Telegraph and Telephone.

AOD: Grupo de Proyecto en Everis llamado Arquitectura de Objetos Digitales.

HTML: Lenguaje de marcado HyperText Markup Lenguaje.

MVC: Modelo de arquitectura llamado Modelo Vista Controlador.

EAR: Formato Java llamado Enterprise Archive y que tiene el papel de controlador.

JAR: Formato Java llamado Java Archive y que contiene la librería del proyecto.

CSS: Lenguaje Cascading Style Sheets utilizado para definir el estilo de la aplicación.

D3: Librería de Javascript llamada Data Driven Documents que se usa para producir, a partir de datos, gráficos dinámicos.

XML: Lenguaje de marcado llamado Extensible Markup Language.

API: Application Programming Interface es un conjunto de funciones, procedimientos o métodos que ofrece una librería.

WWW: World Wide Web es una web o una red de páginas.

CPU: Central Processing Unit es el hardware dentro de un ordenador.

SQL: Lenguaje de comunicación con base de datos llamada Structured Query Language.

ARQDES: Base de datos de la aplicación llamada Arquitectura de Desarrollo.

UML: Unified Model Lenguaje es el modelo o sistema en el que se basa la aplicación.

HTTP: HyperText Transfer Protocol es el protocolo de comunicación que permite la comunicación entre aplicación y usuario.

URL: Uniform Resource Locator es el identificador de la dirección con la que se puede acceder a la aplicación.

JDBC: Java Database Connectivity es la forma de conexión a base de datos.

JNDI: Java Naming and Directory Interface contiene el nombre con el que se puede acceder a una base de datos desde Java.

