UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de una herramienta para el análisis de consumos eléctricos que de soporte a una auditoría energética.

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Santiago Sánchez Llobregat

Tutor: Lenin Guillermo Lemus Zúñiga, Miguel Ángel Mateo Pla

2016/2017

Resumen

El coste de la generación de energía eléctrica por parte de las empresas de dicho sector está siendo muy elevado los últimos años y la previsión es que siga subiendo, afectando al precio de venta de la energía eléctrica al sector público. Este relevante aspecto afecta a particulares, empresarios y a los gobiernos, de manera que todo el mundo está interesado en incrementar la eficiencia energética con el fin de disminuir los costes generados por el consumo eléctrico.

En cualquier caso, para mejorar la eficiencia energética, es conveniente monitorizar los consumos eléctricos y realizar auditorías energéticas para detectar ineficiencias. A partir de dicha monitorización se pretende visualizar el patrón de consumo eléctrico para tener una referencia del mismo en cada uno de los casos y poder proponer ideas de ahorro energético, que favorecerán a ahorrar dinero, disminuir la exposición de gases de efecto invernadero y a hacer una mayor y mejor uso de las energías renovables.

En este proyecto se propone el desarrollo de una herramienta basada en las TIC que ayude a los auditores energéticos a tener acceso a datos históricos de los edificios que se van a auditar además de poder comparar mediante gráficas las cargas de consumo de dichos edificios en distintos periodos de tiempo para poder así ofrecer propuestas para obtener eficiencia energética.

Palabras clave: consumo eléctrico, auditoria energética, eficiencia energética, ahorro.

Abstract

The cost of electricity generation by companies in this sector has been very high in recent years and the forecast is that it continues to rise, affecting the sale price of electricity to the public sector. This important aspect affects individuals, entrepreneurs and governments, so that everyone is interested in increasing energy efficiency in order to reduce the costs generated by electricity consumption.

In any case, in order to improve energy efficiency, it is convenient to monitor electrical consumption and conduct energy audits to detect inefficiencies. From this monitoring is intended to visualize the pattern of electricity consumption to have a reference of the same in each case and to be able to propose ideas of energy saving, which will help to save money, reduce the exposure of greenhouse gases and make a greater and better use of renewable energies.

This project proposes the development of a tool based on ICT to help energy auditors to have access to historical data of buildings to be audited in addition to being able to compare graphically the consumption loads of these buildings in different periods of time to be able to offer proposals for energy efficiency.

Keywords : electrical consumption, energy audit, energy efficiency, saving.

Tabla de contenidos

1. Introducción

Hoy en día el concepto de eficiencia energética está muy en mente de los gobiernos e instituciones. Esto es debido a que sin darnos cuenta estamos destruyendo poco a poco nuestro medioambiente produciendo grandes cantidades de CO2 y gastando mucha más energía eléctrica de la que se necesita. Para mejorar estos efectos nocivos, existe el recurso de realizar auditorías energéticas, que ayudan, mediante la realización de un estudio intensivo, a detectar si se está haciendo un mal uso de la energía eléctrica, y si es así, proponer ideas de ahorro energético para concienciar de cómo se puede conseguir eficiencia energética.

El consumo de energía depende de las condiciones climáticas locales, las características de la envolvente del edificio, las condiciones del interior diseñadas, las características y la configuración de los sistemas técnicos del edificio, las actividades y/o procesos en el edificio y el comportamiento del ocupante y el régimen operacional.

Una auditoria energética constituye un paso importante para una organización de cualquier tipo o tamaño que desee mejorar su eficiencia energética, reducir el consumo de energía y obtener los beneficios medioambientales consiguientes. El proceso de una auditoria energética debe de ser adecuado, completo, representativo, trazable, útil y verificable.

En nuestro caso, este proyecto se va a basar en auditorías energéticas referentes a edificios públicos y viviendas, con lo que habrá que tener en cuenta la provisión de servicios de los mismos tales como la calefacción, refrigeración, ventilación, iluminación, agua caliente sanitaria y sistemas de transporte(ascensores).

El proceso para la realización de una auditoria energética consiste en primer lugar en un contacto preliminar donde el auditor energético debe identificar a todas las partes/organizaciones y sus funciones en la propiedad. En segundo lugar, una reunión inicial donde el auditor energético se pone de acuerdo con la organización para las visitas al emplazamiento, saber el nivel de participación del ocupante y obtener de la misma los puntos de consigna y limites operacionales de las condiciones ambientales interiores, los patrones de ocupación para el rango de actividades diferentes dentro del edificio y los certificados de energía preparados para el edificio.

En tercer lugar, se tiene que realizar una recopilación de los datos, donde dicha recopilación debe ser adecuada al alcance de la auditoria energética. El auditor energético debe recopilar con la organización los portadores de energía, actuales y disponibles, datos relacionados con la energía como por ejemplo la energía distribuida, producida y exportada para cada portador de energía, también datos del consumo de energía y de medición individual si están disponibles y la curva de demanda energética/de carga a intervalos cortos si está disponible.

Otros datos de gran importancia que deben ser recopilados son los factores de ajuste que afectan al consumo de energía, como pueden ser los datos climáticos y los patrones de ocupación. También la información sobre cambios importantes en los últimos 3 años o en el periodo cubierto por los datos operacionales disponibles como pueden ser la forma física del edificio y su envolvente (renovación de ventanas, aislamiento añadido).

También tenemos otros datos importantes como documentos e información existentes sobre el diseño, operación y mantenimiento tales como planos del edificio tal como está construido, cualquier factor externo que pueda influir en el desempeño energético del edificio, diagramas del sistema técnico del edificio, diagramas y configuración del control, el modelo de información de construcción y el equipo que utiliza energía en los espacios ocupados y otras cargas internas.

Una vez ya tenemos todos los datos necesarios recopilados se realiza una revisión de los mismos, donde el auditor energético debe revisar dicha información proporcionada por la organización. El auditor energético debe juzgar si la información proporcionada por la organización permite continuar el proceso de auditoria energética y alcanzar los objeticos acordados.

Una vez terminada la revisión de los datos, el auditor energético procede a realizar un análisis preliminar de los mismos. Este análisis realizado por el auditor energético consiste en emprender un análisis preliminar del balance energético del objeto auditado sobre la base de datos de la energía, establecer los factores de ajuste pertinentes, establecer los indicadores del desempeño energético pertinentes, evaluar la distribución del consumo de energía si es posible y si se dispone de suficiente información, establecer una referencia energética inicial a utilizar para cuantificar los impactos de las intervenciones de ahorro de energía. Con todo ello el auditor energético deberá desarrollar una lista preliminar de oportunidades de mejora de la eficiencia energética.

En cuarto lugar, se encuentra el trabajo de campo, donde el auditor energético debe inspeccionar el emplazamiento respecto a los datos recibidos, evaluar para cada servicio del edificio significativo el nivel de servicio real y futuro, verificar que los sistemas técnicos pueden proporcionar el servicio requerido, evaluar el desempeño de dichos sistemas, comprender los impulsores de los cambios en los sistemas técnicos y buscar oportunidades de mejora de la eficiencia energética.

En quinto lugar tenemos el apartado de análisis, donde dicho análisis debe proporcionar al menos para cada servicio del edificio, una comparación del nivel de servicio real frente al adecuado, además de una evaluación del desempeño real de los sistemas técnicos frente a una referencia adecuada, una evaluación del desempeño de la envolvente del edificio y una evaluación del desempeño energético de todo el edificio, teniendo en cuenta la interacción potencial entre los sistemas técnicos y la envolvente del edificio.

Dentro del análisis se debe hacer un desglose de energía, donde el auditor energético detalla el desglose de la energía distribuida por portador de energía en términos de consumo, coste y emisiones(gráficos), también un desglose del uso final de la energía por el servicio y otro uso en cifras absolutas o específicas y en unidades energéticas

coherentes(gráficos). En definitiva, dicho desglose de energía debe ser representativo de la entrada de energía y del uso de la misma, al mismo tiempo que debe de aclarar qué flujos de energía se basan en mediciones y cuales en estimaciones.

En sexto lugar tenemos la redacción del informe por parte del auditor energético, donde dicho informe debe garantizar que se hayan alcanzado los requisitos acordados con la organización, resumir las mediciones relevantes realizadas durante la auditoria, indicar si los resultados se basan en cálculos, simulaciones o estimaciones, indicar los límites de precisión de las estimaciones de ahorro y coste y finalmente indicar las oportunidades de mejora de la eficiencia energética.

Por último, tenemos la reunión final donde el auditor energético entrega el informe de la auditoria energética, presenta los resultados obtenidos en la misma y procede a su explicación.

En los siguientes apartados se desarrollarán los diferentes aspectos que se han tratado para llevar a cabo el proyecto. En primer lugar, se realiza una especificación de requisitos para establecer que elementos debe contener la aplicación y cuál debe ser su comportamiento. En segundo lugar, se realiza un análisis más extenso y concreto sobre cómo se comporta la aplicación. En tercer y cuarto lugar se lleva a cabo el diseño e implementación de la aplicación y, por último, con el proyecto ya finalizado se procede a la puesta en marcha de la aplicación web.

2. Especificación de requisitos

2.1 Introducción

En este apartado se tratan todos los puntos clave que definen el proyecto en su totalidad. Será utilizado de guía para seguir durante el desarrollo del mismo.

2.1.1 Propósito

La especificación de requisitos de software “ERS” de un proyecto, es el proceso mediante el cual se describe toda la funcionalidad y comportamiento que debe presentar el sistema a desarrollar. Siguiendo el estándar IEEE Std. 830-1998, se procede en los siguientes apartados al análisis de los diferentes aspectos que debe cumplir la aplicación. Destacar que se ha hablado personalmente con un auditor energético para tomar nota de que funcionalidades, comportamiento y apariencia debe de tener la herramienta a desarrollar.

2.1.2 Ámbito del sistema

El objetivo de este proyecto es el desarrollo de una herramienta de apoyo a las auditorías energéticas para que los auditores energéticos tengan la información que necesiten de una forma más automatizada y directa. La herramienta debe de dar servicio a los auditores de una manera sencilla y entendible.

Dicho servicio se lleva a cabo mediante servicios web *RestFul*, que consiste en solicitar directamente la información que necesiten los auditores en dicho momento de una base de datos creada y dedicada explícitamente para ello.

2.1.3 Definiciones, acrónimos y abreviaturas

-Netbeans: Entorno de desarrollo para llevar a cabo la creación de la aplicación.

-ARC: "Advanced Rest Client", es la herramienta que utilizarán los usuarios para consumir los servicios web "RestFul".

-Frontend: Parte frontal de la aplicación. Destinada a captar a usuarios visitantes.

-Backend: Parte interna de la aplicación.

-MySQL: Sistema de gestión de base de datos relacionales.

-Java: Lenguaje de programación utilizado para el desarrollo de la aplicación.

-phpMyAdmin: Herramienta para la administración de las bases de datos.

-SpagoBI: multiplataforma integrada para la inteligencia de negocios.

2.2 Descripción General

2.2.1 Perspectiva del proyecto

La finalidad de esta herramienta es la de dar un soporte a los auditores energéticos a la hora de realizar una auditoria energética. Para ello ideo una nueva forma de que tengan la información al instante, creando una herramienta y servicio web donde les dé la disponibilidad de elegir el edificio que quieran auditar al instante además de los demás datos significativos que les sean necesarios.

Para la consulta de los servicios web "*RestFul*" a los usuarios, principalmente serán los auditores energéticos, se les ofrece la plataforma ARC para la realización de las mismas, donde pueden acceder a la base de datos donde se encuentran todas las tablas y todos los datos mediante la "URL" correspondiente.

También tendrán la opción de poder visualizar mediante graficas la carga de consumo del edificio que estén auditando en el periodo y el formato de tiempo que deseen, dándoles una referencia respecto a los consumos actuales que obtengan y así poder obtener conclusiones respecto a si se está haciendo un buen uso de la energía eléctrica o hay una perdida a la hora de su utilización.

2.2.2 Funciones del proyecto

La aplicación debe de ofrecer las siguientes funciones:

Gestión de los edificios:

Se le tiene que dar la disponibilidad al auditor energético de poder elegir de una lista de edificios específicos, el edificio que él necesite en dicho momento para la realización de la auditoria energética.

Gestión de las fechas:

El auditor energético debe de tener la posibilidad de poder elegir la fecha correspondiente a sus necesidades, tanto si es para comparar resultados de un edificio en fechas concretas como si es para realizar un análisis de los datos que se obtuvieron en cierto día además de elegir la fecha de la auditoria energética que vayan a realizar.

Generación de gráficas:

Una vez seleccionado el edificio que el auditor necesita, procedemos a la selección de la fecha que se necesite y una vez tenemos estos dos datos recogidos, la aplicación procede a la generación de una gráfica de dicho edificio donde se observa el consumo eléctrico del edificio en la fecha seleccionada durante las veinticuatro horas del día en periodos de una hora.

2.3 Requisitos Específicos

En esta parte del documento se tratan en alto detalle los requisitos que se deben satisfacer a la hora de desarrollar la aplicación. Esto permitirá posteriormente realizar una serie de pruebas para demostrar que el sistema satisface los requisitos establecidos.

2.3.1 Requisitos Funcionales

En este apartado se va a describir de una forma más detallada la funcionalidad que debe de presentar la aplicación, donde se mostrarán principalmente la tablas donde está la información almacenada.

2.3.1.1 Clase Edificio

Esta clase representa la descripción y la información más importante relevante a los edificios.

Nombre	Tipo	Restricciones
Cod	Int(2)	-
EdificioMunicipal	Varchar (38)	Debe de ser único
Dirección	Varchar (40)	Valor no nulo
Cup	Varchar (25)	Debe de ser único
DirecciónIberdrola	Varchar (23)	-
Contador	Varchar (8)	-
Peaje	Varchar (5)	-
Pp	Varchar (9)	-
Pll	Varchar (8)	-
Pv	Varchar (8)	-
Contrato	Varchar (9)	-

2.3.1.2 Clase Factura

Esta clase almacena toda la información relacionada con las facturas de los edificios.

Nombre	Tipo	Restricciones
Id	Int (11)	Debe de ser único y auto-incrementable
Fecha	Date	-
Factura	Varchar (26)	-
Dirección	Varchar (40)	-
Importe	Decimal (6,2)	Máximo dos decimales

2.3.1.3 Clase "Dim_Tiempo"

En esta clase se almacenan las distintas dimensiones temporales en las que se puede disponer la obtención de los datos.

Nombre	Tipo	Restricciones
Fecha	Date	Debe de ser único
Anio	Smallint (6)	Valor no nulo
Trimestre	Smallint (6)	Valor no nulo
Mes	Smallint (6)	Valor no nulo
Semana	Smallint (6)	Valor no nulo
Día	Smallint (6)	Valor no nulo
DíaSemana	Smallint (6)	Valor no nulo
NTrimestre	Varchar (7)	Valor no nulo
NMes	Varchar (15)	Valor no nulo
NMes3L	Varchar (3)	Valor no nulo
NSemana	Varchar (11)	Valor no nulo
NDía	Varchar (15)	Valor no nulo
NDíaSemana	Varchar (15)	Valor no nulo

2.3.1.4 Clase "DetalleContrato"

En esta clase se definen los detalles de cada contrato que tenemos.

Nombre	Tipo	Restricciones
Id	Int (11)	Debe de ser único y auto-incrementable
Contrato	Int (11)	Valor no nulo
Fecha	Date	Valor no nulo
Hora	Time	Valor no nulo
ConsumoActiva	Int (11)	-
ConsumoReactiva	Int (11)	-

2.3.1.5 Clase Contrato

En esta clase se tiene cada contrato encontrado y su correspondiente dirección, además de su estado de tramitación, su contenido de datos y su CUPS (Código Universal de Punto de Suministro) y número de contrato de acceso.

Nombre	Tipo	Restricciones
Contrato	Int(9)	Debe de ser único
Dirección	Varchar (69)	-
Estado	Varchar (12)	-
Datos	Varchar (2)	-
Intervalo	Varchar (32)	-
Cup	Varchar (25)	-
Número contrato de acceso	Int (11)	-

2.3.2 Requisitos Tecnológicos

El proyecto se va a realizar con el entorno de desarrollo "*Netbeans* IDE" cuya versión será la 8.2 debido a que es la más completa y la última versión del mismo. Se utilizará también el paquete de programación "*Xampp*" para tener disponible al mismo tiempo y de una manera más sencilla un servidor Apache y una base de datos "MySQL". La gestión de la base de datos se va a llevar a cabo con "*phpMyAdmin*" y la consulta y gestión de los servicios web "*RestFul*" se llevará a cabo a través de la extensión ARC (Advanced Rest Client) que posee el navegador "Google Chrome" especialmente dedicada para servicios web.

2.3.3 Atributos del Sistema

Requisitos de fiabilidad

La aplicación y los servicios deben estar operativos durante todo el tiempo. Se deberá de disponer de una copia de seguridad por si hubiese una caída del servicio tener un reemplazo rápido para tener las menores consecuencias posibles.

Mantenibilidad

El proyecto debe de estar en constante mantenimiento, concretamente la actualización diaria de la base de datos, ya que cada día se descargan nuevos datos y hay que administrarlos de forma correcta para que cuando un usuario, mejor dicho, un auditor energético, los necesite los tenga disponibles.

Portabilidad

El proyecto a desarrollar será portable en todo momento, teniendo en un servidor dedicado la base de datos para tener acceso a ella desde cualquier dispositivo, además la extensión que se tiene que usar está disponible en los navegadores, y todos los PC's tienen instalados mínimo uno a día de hoy.

Seguridad

Se implantarán protocolos de seguridad para el acceso de la base de datos. Solo el administrador o administradores tendrán acceso a ella mediante unas credenciales especiales.

3. Análisis

3.1 Introducción

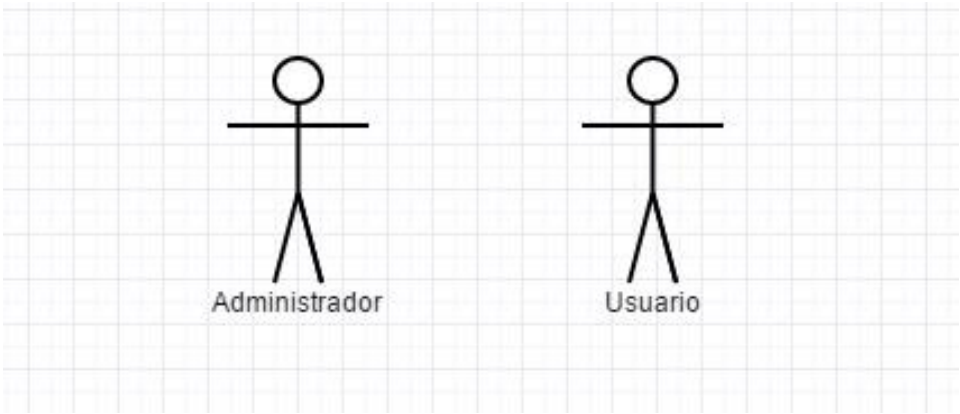
En este capítulo de realiza un análisis de la aplicación. Se especifica la estructura que debe presentar y cuál debe ser su comportamiento. El análisis se ha realizado utilizando el lenguaje UML, en inglés "*Unified Modeling Language*" mediante el cual se establecen tres tipos de diagrama: diagramas de comportamiento, de estructura y de interacción.

3.2 Diagramas de casos de uso

El diagrama de casos de uso es un diagrama de comportamiento UML. Identifica todas las acciones posibles que puede realizar el usuario en un escenario concreto de la aplicación. En primer lugar, se identificarán los actores del sistema y posteriormente se tratarán las diferentes acciones que pueden desempeñar.

3.2.1 Actores

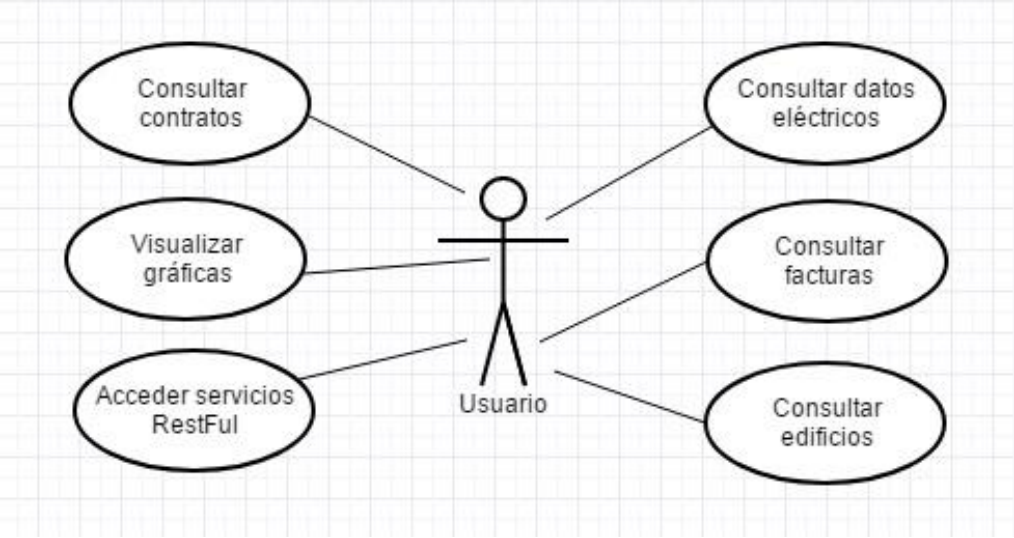
En este apartado vamos a describir a los actores que van a interactuar con el sistema. En la siguiente imagen vamos a ver a los usuarios que forman parte de la aplicación.



Se puede ver como existen dos tipos de actores tipo. El administrador y el usuario.

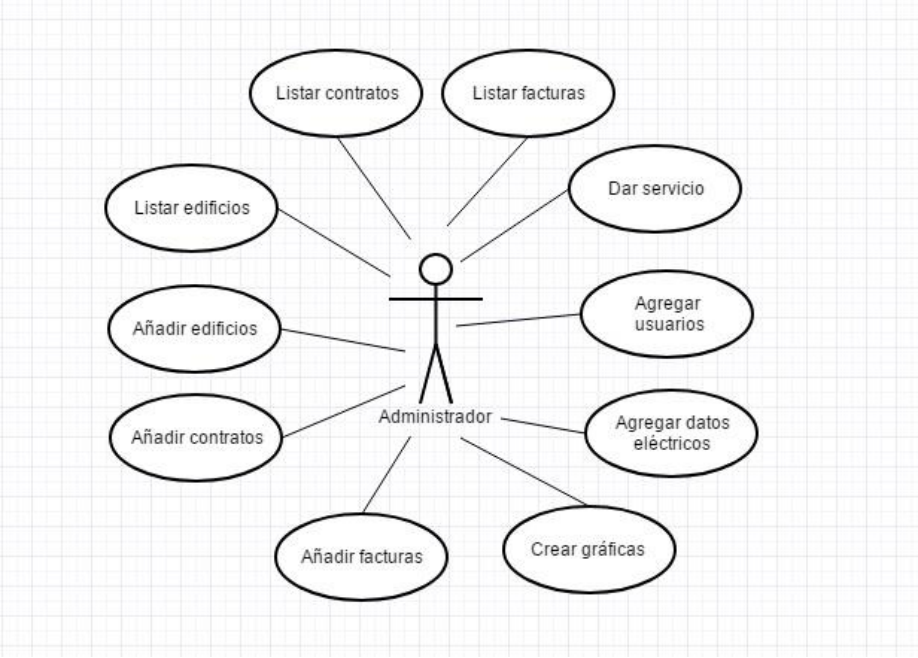
3.2.1.1 Usuario

Los casos de uso que desempeña este actor son los casos que puede desempeñar cualquier usuario que utilice la aplicación, que principalmente serán los auditores energéticos para la realización de las auditorías energéticas. En la siguiente imagen vemos las acciones que puede realizar este actor.



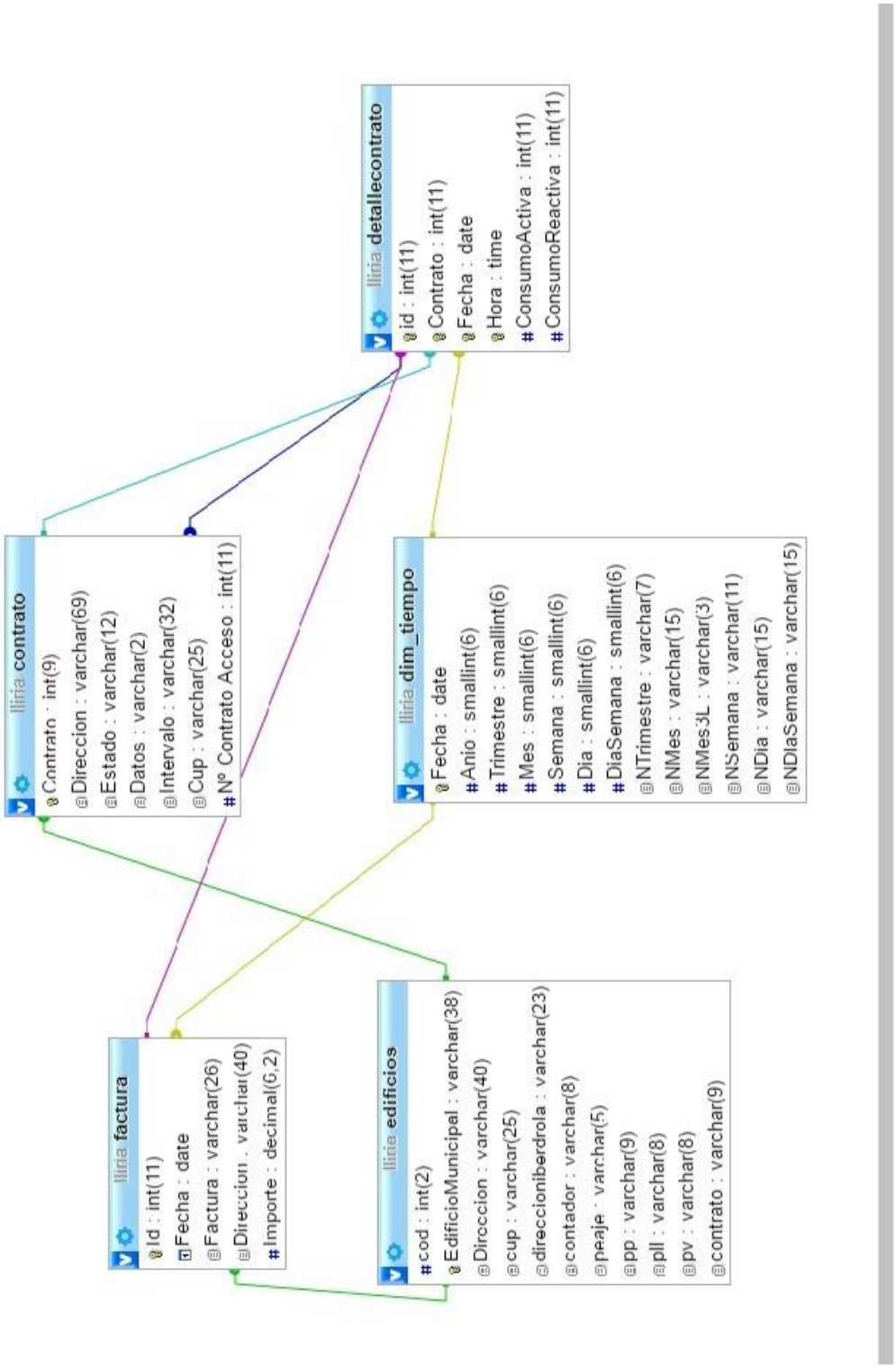
3.2.1.2 Administrador

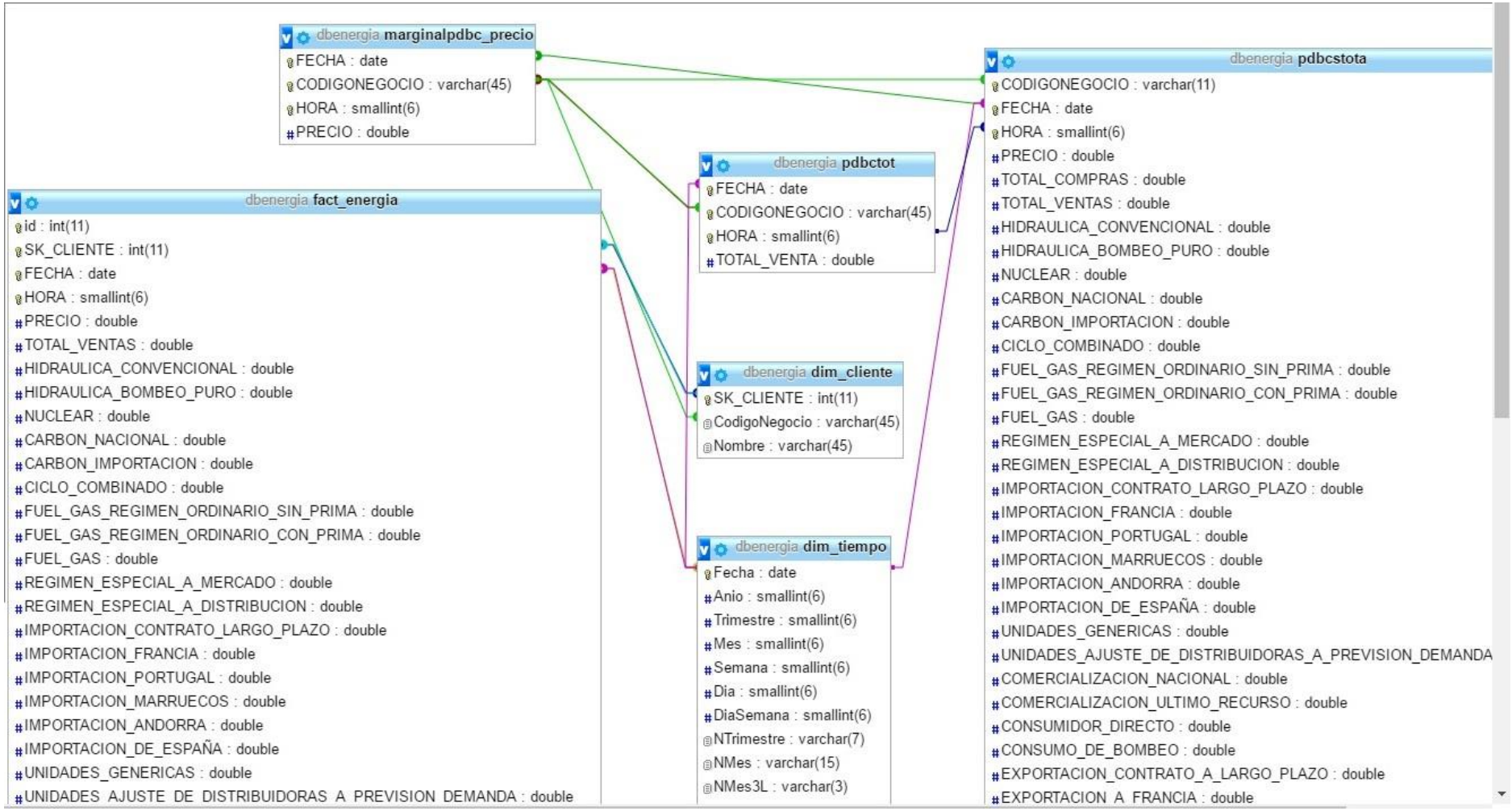
El administrador es el que tiene la obligación de mantener principalmente la base de datos a la orden del día para que no haya problemas a la hora de detectar datos no existentes o erróneos. En la siguiente imagen vemos las acciones que puede realizar este actor.



3.3 Diagrama base de datos

El diagrama de la base de datos es una representación gráfica que muestra las entidades y sus propiedades y las relaciones entre diferentes clases. En primer lugar, se tiene la base de datos llamada “lliria” donde tenemos alojados los edificios y su información correspondiente. En segundo lugar, está la base de datos “dbenergia” donde se aloja todo lo relacionado con los consumos eléctricos y demanda de energía.




dbenergia marginalpdbc_precio
FECHA : date
CODIGONEGOCIO : varchar(45)
HORA : smallint(6)
PRECIO : double
dbenergia pdbctot
FECHA : date
CODIGONEGOCIO : varchar(45)
HORA : smallint(6)
TOTAL_VENTA : double
dbenergia pdbcstota
CODIGONEGOCIO : varchar(11)
FECHA : date
HORA : smallint(6)
PRECIO : double
TOTAL_COMPRAS : double
TOTAL_VENTAS : double
HIDRAULICA_CONVENCIONAL : double
HIDRAULICA_BOMBEO_PURO : double
NUCLEAR : double
CARBON_NACIONAL : double
CARBON_IMPORTACION : double
CICLO_COMBINADO : double
FUEL_GAS_REGIMEN_ORDINARIO_SIN_PRIMA : double
FUEL_GAS_REGIMEN_ORDINARIO_CON_PRIMA : double
FUEL_GAS : double
REGIMEN_ESPECIAL_A_MERCADO : double
REGIMEN_ESPECIAL_A_DISTRIBUCION : double
IMPORTACION_CONTRATO_LARGO_PLAZO : double
IMPORTACION_FRANCIA : double
IMPORTACION_PORTUGAL : double
IMPORTACION_MARRUECOS : double
IMPORTACION_ANDORRA : double
IMPORTACION_DE_ESPAÑA : double
UNIDADES_GENERICAS : double
UNIDADES_AJUSTE_DE_DISTRIBUIDORAS_A_PREVISION_DEMANDA
COMERCIALIZACION_NACIONAL : double
COMERCIALIZACION_ULTIMO_RECURSO : double
CONSUMIDOR_DIRECTO : double
CONSUMO_DE_BOMBEO : double
EXPORTACION_CONTRATO_A_LARGO_PLAZO : double
EXPORTACION_A_FRANCIA : double
dbenergia fact_energia
id : int(11)
SK_CLIENTE : int(11)
FECHA : date
HORA : smallint(6)
PRECIO : double
TOTAL_VENTAS : double
HIDRAULICA_CONVENCIONAL : double
HIDRAULICA_BOMBEO_PURO : double
NUCLEAR : double
CARBON_NACIONAL : double
CARBON_IMPORTACION : double
CICLO_COMBINADO : double
FUEL_GAS_REGIMEN_ORDINARIO_SIN_PRIMA : double
FUEL_GAS_REGIMEN_ORDINARIO_CON_PRIMA : double
FUEL_GAS : double
REGIMEN_ESPECIAL_A_MERCADO : double
REGIMEN_ESPECIAL_A_DISTRIBUCION : double
IMPORTACION_CONTRATO_LARGO_PLAZO : double
IMPORTACION_FRANCIA : double
IMPORTACION_PORTUGAL : double
IMPORTACION_MARRUECOS : double
IMPORTACION_ANDORRA : double
IMPORTACION_DE_ESPAÑA : double
UNIDADES_GENERICAS : double
UNIDADES_AJUSTE_DE_DISTRIBUIDORAS_A_PREVISION_DEMANDA : double
dbenergia dim_cliente
SK_CLIENTE : int(11)
CodigoNegocio : varchar(45)
Nombre : varchar(45)
dbenergia dim_tiempo
Fecha : date
Anio : smallint(6)
Trimestre : smallint(6)
Mes : smallint(6)
Semana : smallint(6)
Dia : smallint(6)
DiaSemana : smallint(6)
NTrimestre : varchar(7)
NMes : varchar(15)
NMes3L : varchar(3)

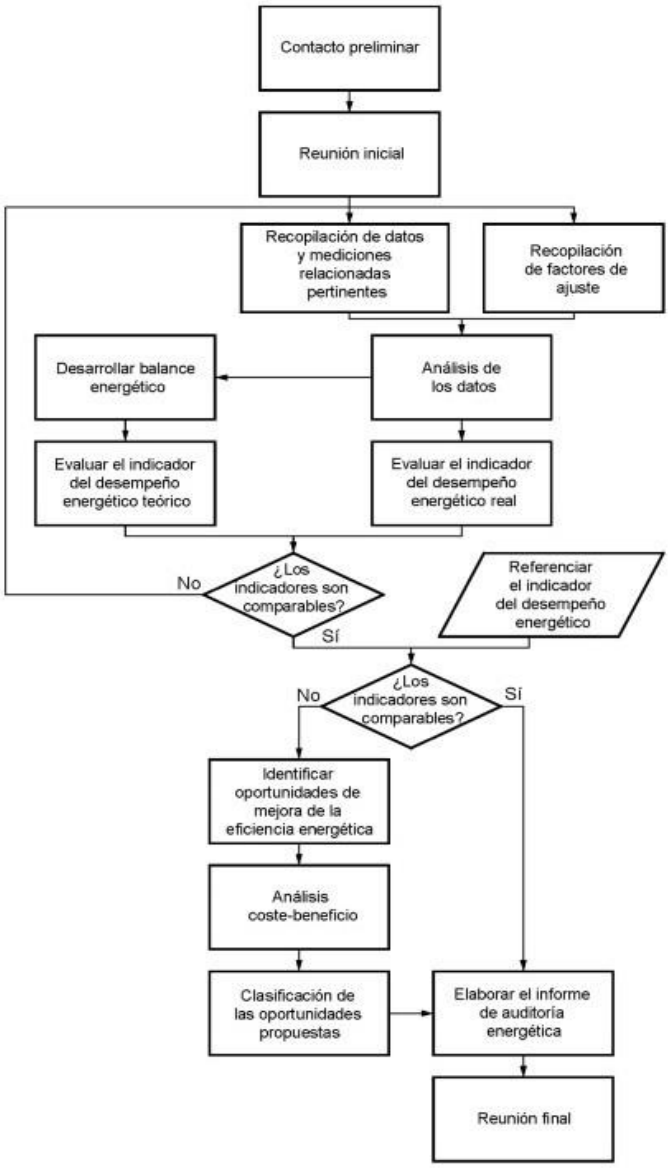
#CONSUMIDOR_DIRECTO : double
#CONSUMO_DE_BOMBEO : double
#EXPORTACION_CONTRATO_A_LARGO_PLAZO : double
#EXPORTACION_A_FRANCIA : double
#EXPORTACION_A_PORTUGAL : double
#EXPORTACION_A_MARRUECOS : double
#EXPORTACION_A_ANDORRA : double
#EXPORTACION_A_ESPAÑA : double
#UNIDADES_GENERICAS_DE_DISTRIBUCION : double
#UNIDADES_GENERICAS_SUBASTAS_DISTRIBUCION : double
#TOTAL_HIDRAULICA : double
#TOTAL_TERMICA : double
#TOTAL_REGIMEN_ESPECIAL : double
#TOTAL_IMPORTACION : double
#TOTAL_GENERICAS : double
#TOTAL_PRODUCCION : double
#TOTAL_DEMANDA_NACIONAL_CLIENTES : double
#TOTAL_CONSUMO_BOMBEO : double
#TOTAL_EXPORTACIONES : double
#TOTAL_GENERICAS_9301 : double
#TOTAL_DEMANDA : double
#TOTAL_VENTAS_NACIONALES_COMERCIALIZACION_A_MERCADO : double
#TOTAL_VENTAS_INTERNACIONALES_COMERCIALIZACION_A_MERCADO : double
#TOTAL_REGIMEN_ORDINARIO_CON_PRIMA : double
#TOTAL_POTENCIA_INDISPONIBLE : double
#UNIDADES_AJUSTE_DE_DISTRIBUIDORAS_A_PREVISION_DEMANDA : double
#COMERCIALIZACION_NACIONAL : double
#COMERCIALIZACION_ULTIMO_RECURSO : double
#CONSUMIDOR_DIRECTO : double
#CONSUMO_DE_BOMBEO : double
#EXPORTACION_CONTRATO_A_LARGO_PLAZO : double
#EXPORTACION_A_FRANCIA : double
#EXPORTACION_A_PORTUGAL : double
#EXPORTACION_A_MARRUECOS : double
#EXPORTACION_A_ANDORRA : double
#EXPORTACION_A_ESPAÑA : double
#UNIDADES_GENERICAS_DE_DISTRIBUCION : double
#UNIDADES_GENERICAS_SUBASTAS_DISTRIBUCION : double
#TOTAL_HIDRAULICA : double
#TOTAL_TERMICA : double
#TOTAL_REGIMEN_ESPECIAL : double
#TOTAL_IMPORTACION : double
#TOTAL_GENERICAS : double
#TOTAL_PRODUCCION : double
#TOTAL_DEMANDA_NACIONAL_CLIENTES : double
#TOTAL_CONSUMO_BOMBEO : double
#TOTAL_EXPORTACIONES : double
#TOTAL_GENERICAS_9301 : double
#TOTAL_DEMANDA : double
#TOTAL_VENTAS_NACIONALES_COMERCIALIZACION_A_MERCADO : double
#TOTAL_VENTAS_INTERNACIONALES_COMERCIALIZACION_A_MERCADO : double
#TOTAL_REGIMEN_ORDINARIO_CON_PRIMA : double
#TOTAL_POTENCIA_INDISPONIBLE : double

En la anterior captura se muestra la parte final de las tablas de "*fact_energia*" y de "*pdbcstota*" de la base de datos "*dbenergia*".

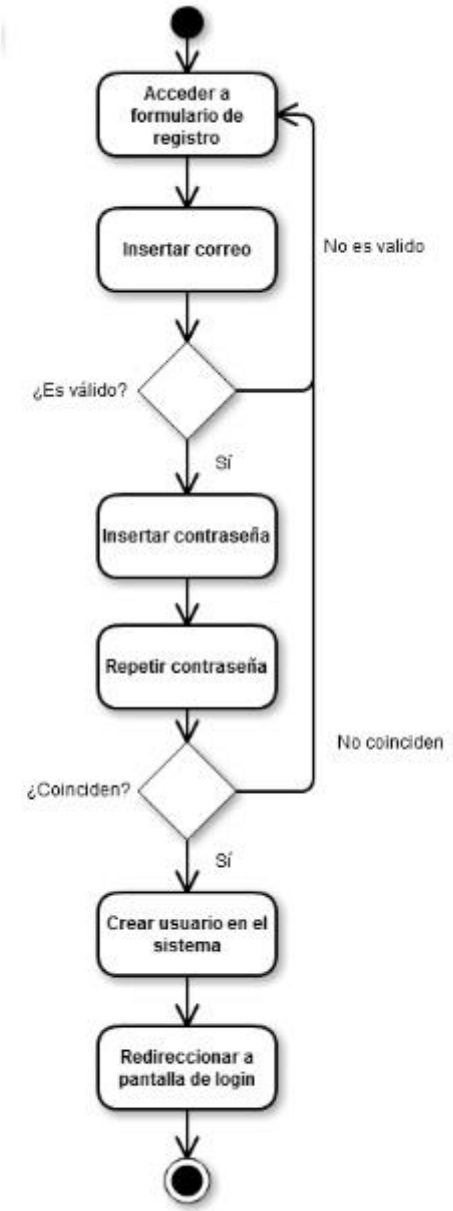
3.4 Diagramas de actividad

El diagrama de actividad es la representación gráfica del flujo o proceso que realiza el sistema para implementar una funcionalidad. Modelan el comportamiento que el usuario tendrá cuando interactúe con el sistema. En este apartado se mostrarán los principales flujos de trabajo que se realizan en el sistema.

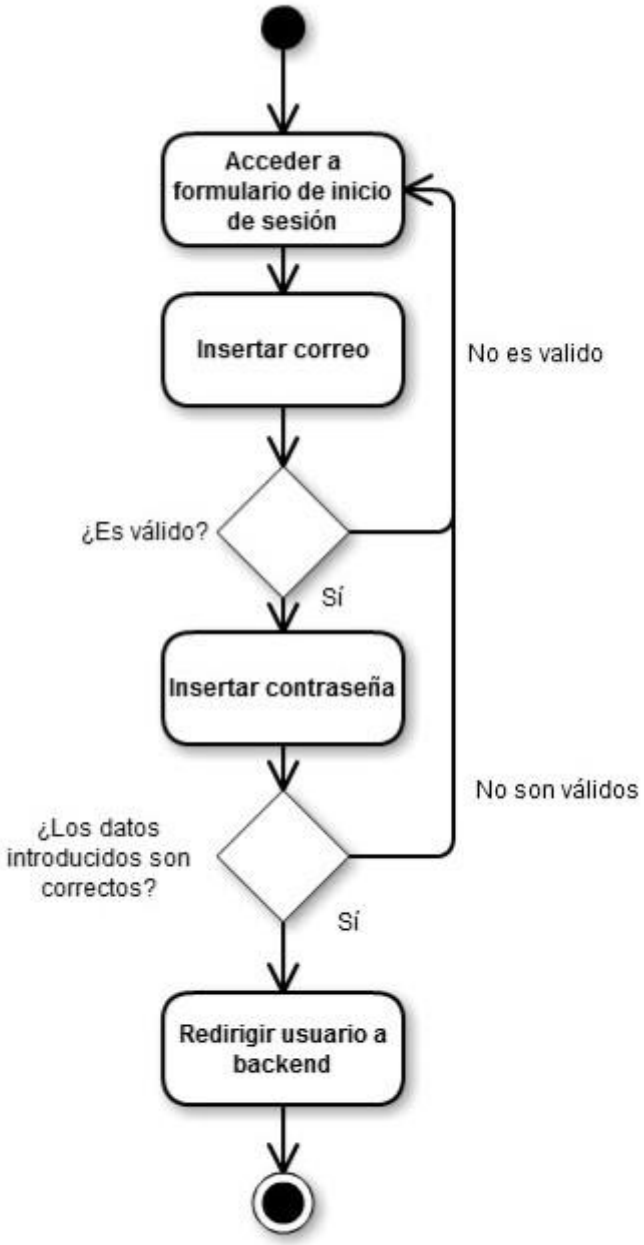
3.4.1 Proceso auditoría energética



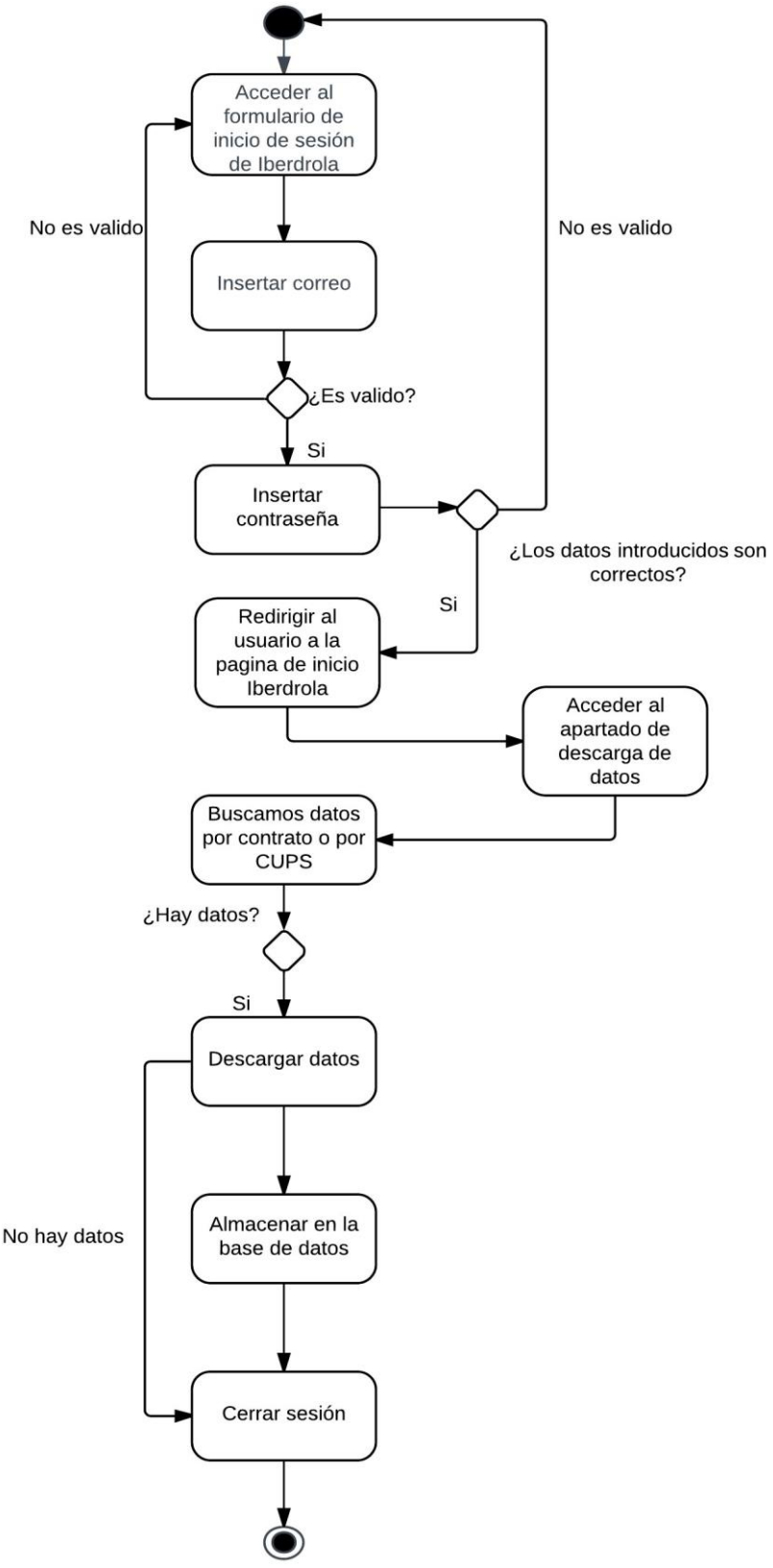
3.4.2 Registro Usuario Iberdrola



3.4.3 Inicio de sesión Iberdrola



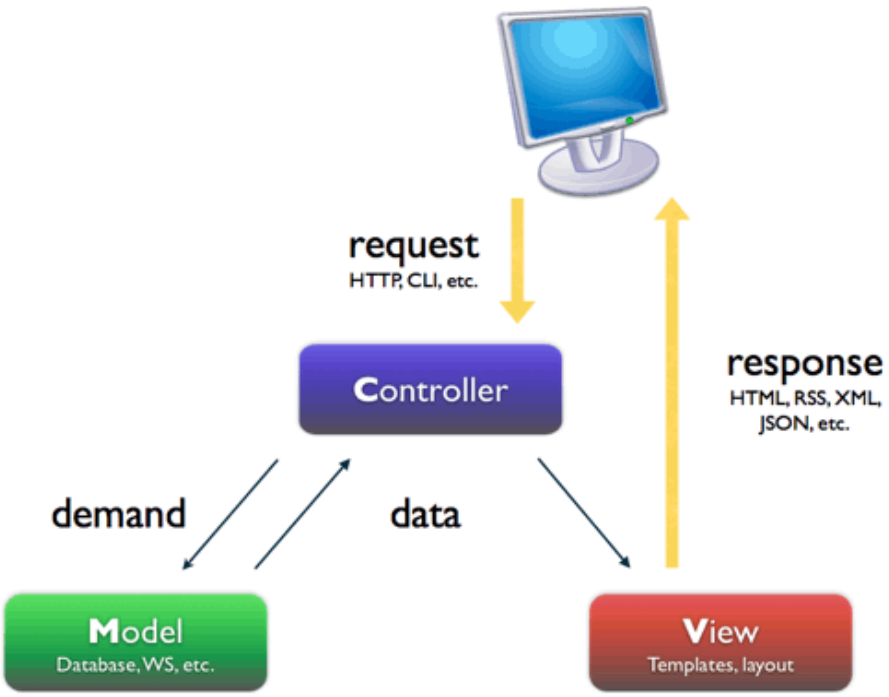
3.4.3 Descarga de Datos



4. Diseño

4.1 Arquitectura MVC

La aplicación a desarrollar en este proyecto se ha llevado a cabo con el entorno de desarrollo Netbeans IDE 8.2, donde en este entorno podemos llevar a cabo el uso de la arquitectura MVC(Modelo-Vista-Controlador). Este patrón de diseño, introducido en los años 70 por Trygve Reenskaug, se caracteriza por dividir el código de la aplicación en tres capas diferentes. Estas divisiones lógicas son el modelo, la vista y el controlador, ya mencionadas anteriormente. A continuación, se muestra una imagen donde se representa el esquema principal de esta arquitectura.



Por una parte, tenemos la **capa Modelo**, donde se puede ver como se encarga de realizar toda la conexión con la base de datos y de gestionar todas las operaciones que se realizan en ella. Luego tenemos la **capa de Vista**, que es la responsable de representar toda la información obtenida a través de la pantalla, dicho de otra forma, se encarga de la visualización de la información. Por último, tenemos la **capa Controlador**, que es la encargada del procesamiento de la información obtenida a través de la capa Modelo y la prepara para retornarla a la capa de Vista.

4.2 Flujo de Ejecución

El flujo de ejecución que se lleva a cabo en este modelo es el siguiente:

- En primer lugar, el usuario realiza una petición "*Request*" HTTP (Protocolo de Transferencia de Hipertexto) al sitio web que se ha implementado. Esta petición que acabamos de realizar es captada por el controlador de nuestra aplicación. Luego el Controlador se encarga de procesar la petición y de determinar las operaciones que se deben ejecutar para poder construir la respuesta a la petición realizada por el usuario. Si hay que realizar alguna modificación de los datos de la aplicación, se realizará una comunicación con la capa Modelo y le transmitirá la información que desea obtener.

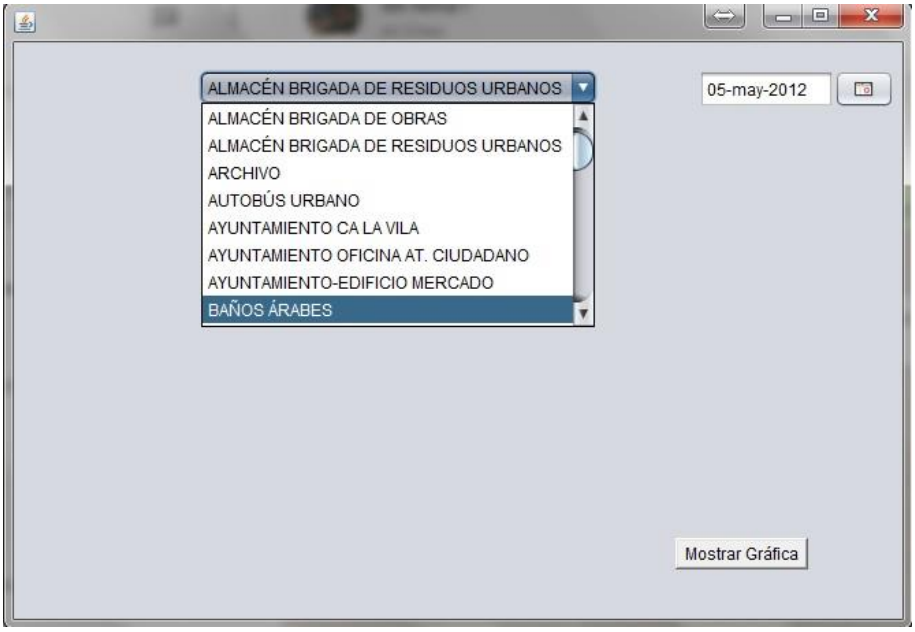
- En segundo lugar, la capa Modelo realiza el procesamiento de las órdenes recibidas por el Controlador y se comunica con la base de datos para obtener la información que se necesita, donde seguidamente se le devuelve los resultados obtenidos al Controlador.

- En tercer lugar, una vez el Controlador tenga toda la información que necesita y haya procedido a su procesamiento, le transmite a la capa Vista los resultados obtenidos para que realice la construcción de la respuesta para el usuario.

- En cuarto y último lugar, la capa Vista podrá generar una respuesta "Response" donde se represente la información mediante el lenguaje de marcado JSON ("JavaScript Object Notation").

4.3 Frontend

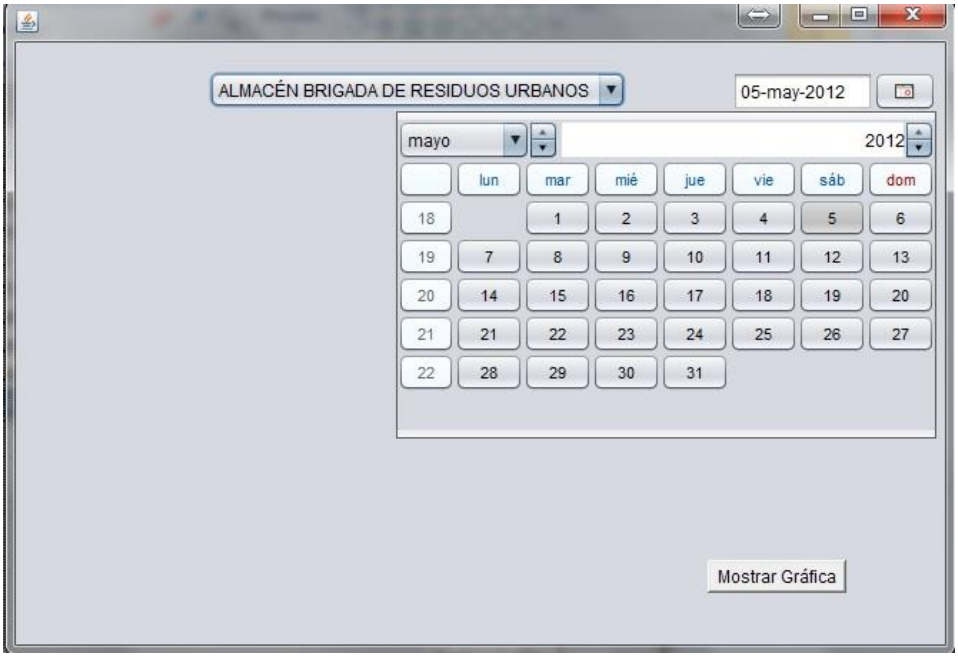
En este proyecto nos vamos a centrar en la parte "*Frontend*" de la aplicación, ya que es la que nos requiere mayor relevancia para el mismo. La parte frontal ("*Frontend*") es la encargada de recibir a los usuarios que visitan por primera vez la aplicación. En ella se muestra el listado de los edificios de los que tenemos sus datos de consumo eléctrico y por tanto son sobre los que se llevará a cabo una auditoria energética, además de poder elegir la fecha que se desee para la realización de la misma. A continuación, se muestran unas imágenes simples de lo que es la parte "*Frontend*" de la aplicación:



En esta imagen se puede ver la vista inicial que tendría cualquier usuario que accede a la aplicación, se ha tratado de hacer una aplicación sencilla y fácil de manejar para los usuarios, ya que lo que interesa principalmente y en lo que consiste este proyecto es en proporcionar una herramienta de apoyo para las auditorias, por tanto, cuanto más sencilla sea de entender y manejar la aplicación, más rápido y fácil se podrá realizar el trabajo deseado.

En esta siguiente imagen se va a mostrar la funcionalidad de la lista desplegable, donde se puede observar la lista de los edificios que tenemos en la base de datos, que son los edificios de los que disponemos de sus datos de consumo eléctrico.

En esta última imagen se puede ver la funcionalidad de la aplicación a la hora de la selección de la fecha, donde se observa de una forma entendible como funciona su funcionamiento de selección.



5. Implementación

5.1 Introducción

En este apartado se va a explicar de una forma muy detallada las distintas decisiones que se han tomado en el desarrollo de la aplicación al mismo tiempo que los aspectos técnicos que esta conlleva. En algunos casos voy a usar capturas de pantalla junto a la propia explicación para complementar dicha información y poder dejar más claro el proceso de desarrollo.

Esta sección se va a dividir en un solo apartado pero muy amplio, donde se incluyen en un mismo nivel todas las tecnologías utilizadas.

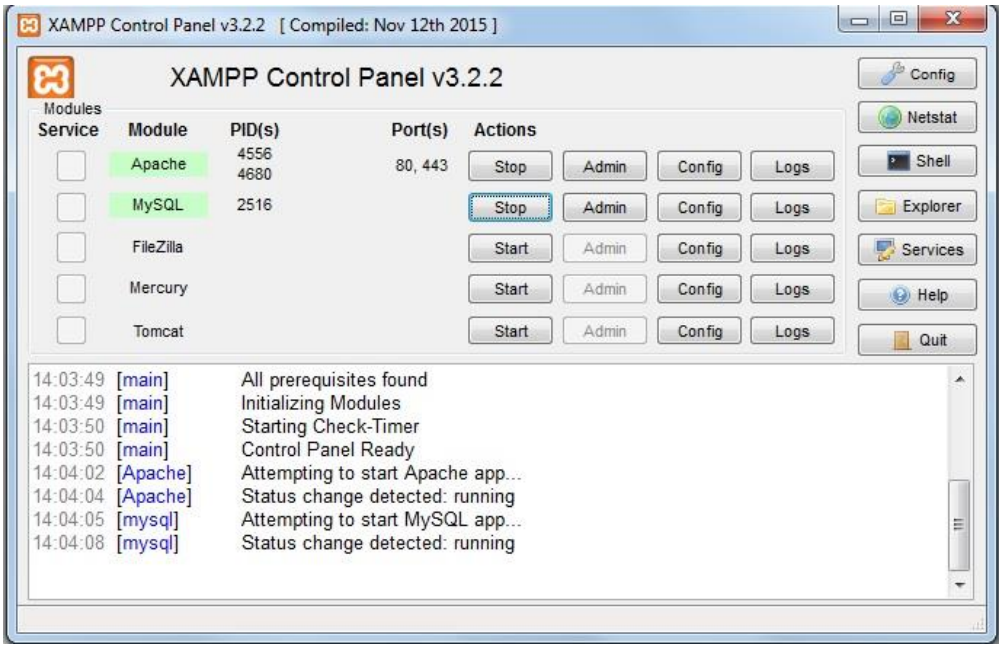
5.2 Principales Tecnologías

5.2.1 Xampp

Es una conocida herramienta de desarrollo utilizada por los desarrolladores de aplicaciones web. Esta herramienta da lugar a un entorno de desarrollo formado por las siguientes tecnologías:

- Servidor web Apache
- Base de datos MySQL
- PHP
- Perl
- Tomcat

Nos decantamos por su utilización debido a que es un software libre y además con la instalación de un solo paquete podemos disponer de todas las tecnologías que posteriormente necesitaremos en un entorno de producción. Se puede observar en la siguiente captura de pantalla la interfaz gráfica de la herramienta que estamos nombrando y analizando.



5.2.2 MariaDB

Es un sistema de gestión de bases de datos derivado de MySQL que posee licencia GPL (General Public License). Este sistema introduce dos motores de almacenamiento nuevos, uno llamado Aria y otro llamado XtraBD. El principal objetivo de MariaDB es el poder cambiar un servidor por otro directamente.

En este proyecto se ha utilizado dicho sistema para gestionar y administrar nuestra base de datos porque a diferencia de los demás sistemas de este tipo que existen, MariaDB es con el que más familiarizado estoy debido a las distintas asignaturas que he estudiado a lo largo de la carrera. A continuación, una captura de pantalla para la visualización de las características de dicho sistema de gestión.

5.2.3 SQL

El lenguaje SQL" *Structured Query Language*" (lenguaje de marcas estructurado) es un lenguaje de acceso a bases de datos relacionales. Permite entre otras cosas crear y modificar esquemas de bases de datos y especificar las operaciones sobre bases de datos. Aúna características de álgebra y el cálculo relacional. Se ha usado este lenguaje para realizar las consultas de acceso a la base de datos además de usarlo para realizar alguna modificación de la misma si fuese necesario.

5.2.4 Python

Es un lenguaje de programación interpretado cuya filosofía haca hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, es decir, soporta programación orientada a objetos, programación imperativa y de una menos medida programación funcional.

Destacar de este lenguaje también que usa tipado dinámico y que es multiplataforma. En este proyecto hemos usado "*Python*" para el desarrollo de los "*script*" de descarga masiva de datos históricos de los edificios debido al gran volumen de ficheros. Concretamente se ha usado para convertir los archivos que tienen el detalle de los contratos en dos archivos CSV (Comma-Separated Values), uno para los consumos de energía activa y otro para los consumos de energía reactiva.

5.2.5 PhpMyAdmin

PhpMyAdmin es una herramienta escrita en PHP ("*Hypertext Preprocessor*") que maneja la administración de MySQL a través de páginas web, usando Internet. Es "*Open Source*", es decir, es una fuente abierta y no hay que pagar para su uso. Debido al proyecto que se ha realizado, esta es una de las características por la que hemos elegido esta herramienta.

También destacar que podemos importar y exportar datos en formato CSV, que a la hora de cargar los ficheros de datos históricos de los edificios y de los consumos eléctricos será de gran ayuda.

5.2.6 Spoon

Spoon es el diseñador gráfico de transformaciones y trabajos del sistema de ETTLs de "*Pentaho Data Integration*" (PDI), también conocido como "*Kettle*". Está diseñado para ayudar en los procesos ETTLs, que incluyen la Extracción, Transformación, Transporte y Carga de datos. También decir que es una Interfaz Gráfica de Usuario(GUI), que permite diseñar transformaciones y trabajos que se pueden ejecutar con las herramientas de "*Kettle*" (Pan y "*Kitchen*").

Pan es un motor de transformación de datos que realiza muchas funciones tales como lectura, manipulación, y escritura de datos hacia y desde varias fuentes de datos. "*Kitchen*" es un programa que ejecuta los trabajos diseñados por "*Spoon*" en XML (Lenguaje de Marcado Extensible) o en un catálogo de base de datos.

5.2.7 Java

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Está diseñado para permitir el desarrollo de aplicaciones portátiles de elevado rendimiento para el más amplio rango de plataformas informáticas posible.

Se ha elegido este lenguaje de programación para el desarrollo del proyecto debido a que permite crear programas que se pueden ejecutar en un explorador y acceder a servicios Web disponibles, que en este caso serán servicios Web propios que pondremos a disposición de los usuarios que hagan uso de la herramienta que se está desarrollando.

5.2.8 XML

XML es un metalenguaje que nos permite definir lenguajes de marcado adecuado a usos determinados. Sirve para representar información estructurada en la web (todo documento), de modo que esta información pueda ser almacenada, transmitida, procesada, visualizada e impresa, por muy diversos tipos de aplicaciones y dispositivos.

Dentro de las aplicaciones generales de XML, en este proyecto resaltan dos de ellas, la primera es la de publicar e intercambiar contenidos de bases de datos; y la segunda es la de crear documentos específicos, concretamente los "*template*", para la visualización de las gráficas que necesitemos.

5.2.9 Servicios Web RestFul

Según el consorcio W3C, se definen los servicios Web como sistemas "*software*" diseñados para soportar una interacción interoperable máquina a máquina sobre una red. Estos servicios Web suelen ser APIs Web que pueden ser accedidas desde dentro de una red (principalmente Internet) y son ejecutados en el sistema que los aloja.

Dentro de los servicios Web tenemos dos tipos bien diferenciados, los servicios REST y los servicios SOAP. En este proyecto se ha elegido la primera opción, los servicios REST ("*Representational State Transfer*") debido a que es un estilo de arquitectura de "software" para sistemas hipermedias distribuidos como la Web.

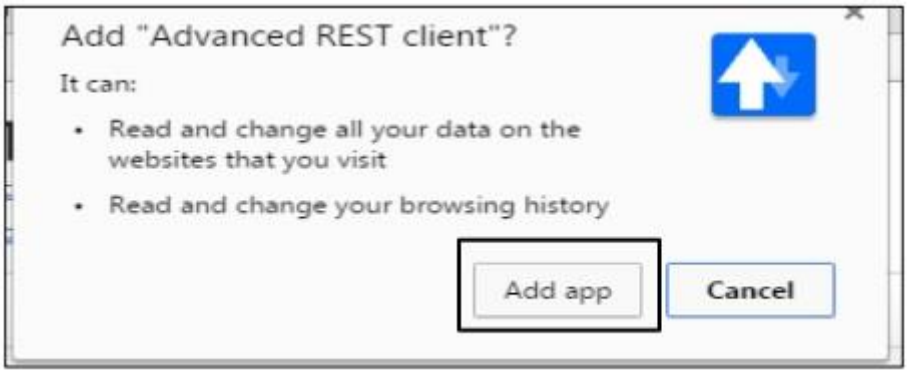
El termino fue introducido en la tesis doctoral de Roy Fielding en el año 2000, quien a su vez es uno de los principales autores de la especificación de HTTP ("Protocolo de Transferencia de Hipertexto"). Como principales características de REST hay que destacar que las operaciones que se realizan se definen en los mensajes, se tiene una dirección única para cada instancia del proceso y cada objeto soporta las operaciones estándares definidas.

Ahora procedo a definir las ventajas de este servicio que son muy interesantes bajo mi punto de vista a la hora de consumir servicios a través de Internet. Dichas ventajas son que posee un bajo consumo de recursos, las instancias del proceso son creadas explícitamente, el cliente no necesita información de enrutamiento a partir de la URI inicial, los clientes pueden tener una interfaz "*listener*" (escuchadora) genérica para las notificaciones y, por último, generalmente son fáciles de construir y de adaptar.

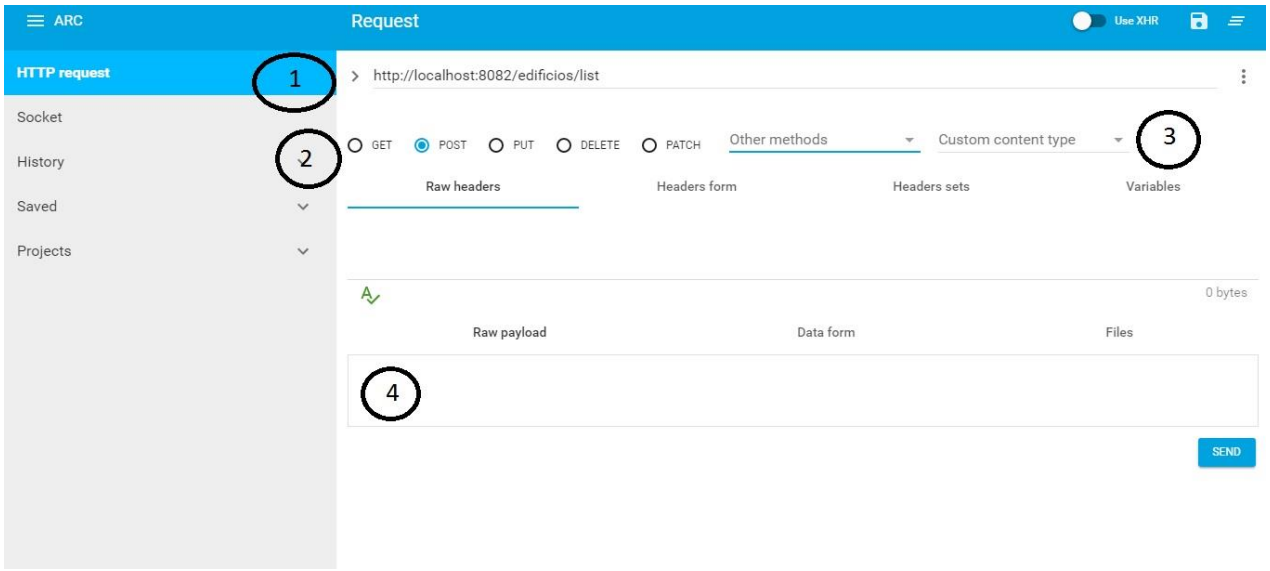
5.2.10 ARC

ARC o "*Advanced Rest Client*" es una extensión que posee el navegador "Google Chrome", navegador presente hoy en día en cualquier dispositivo tanto ordenadores como teléfonos móviles, que nos permite realizar servicios "*Rest*".

Dichos servicios se van a basar en operaciones CRUD, dicho de otro modo, en operaciones como Crear, Leer, Actualizar y Eliminar. Para tener acceso a esta extensión, primero tenemos que acceder a la tienda del propio navegador donde se encuentran todas las extensiones disponibles. Una vez se haya localizado solo hay que añadirla al mismo y ya la tenemos a nuestra disposición.

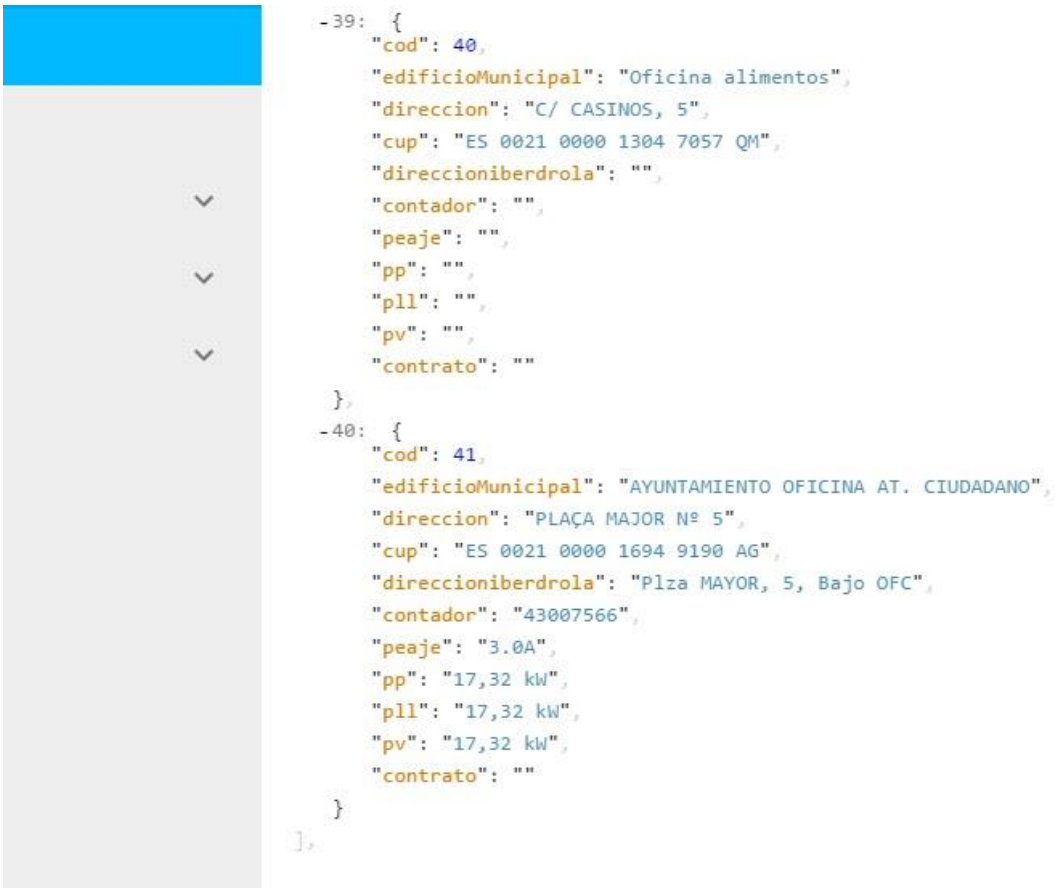


Ahora voy a proceder a enseñar la interfaz principal de ARC mediante una captura de pantalla y una breve explicación de los campos más importantes que tenemos que tener en cuenta, que es con la que se va a trabajar y solicitar en cada momento toda la información que nos sea necesaria.



He marcado con números las cuatro partes principales y las cuales voy a definir su función ahora. En primer lugar, tenemos el URL que tiene que especificar el usuario para poder consumir los servicios, en este caso es el URL http://localhost:8082/edificios/list que lo que va a hacer es a listar todos los edificios de nuestra base de datos, o mejor dicho va a listar la tabla edificios de nuestra base de datos. A continuación, muestro una serie de capturas para demostrarlo:


```
[Array[41]
  -0:  {
       "cod": 1,
       "edificioMunicipal": "AYUNTAMIENTO CA LA VILA",
       "direccion": "PLAÇA MAJOR Nº 64",
       "cup": "ES 0021 0000 0810 5330 RN",
       "direccioniberdrola": "Plza MAYOR, 64, Bajo",
       "contador": "3206205",
       "peaje": "3.0A",
       "pp": "30 kW",
       "pll": "30 kW",
       "pv": "60 kW",
       "contrato": "543502545"
   },
  -1:  {
       "cod": 2,
       "edificioMunicipal": "MUSEO ARQUEOLÓGICO",
       "direccion": "PL TRINQUET VELL Nº 10",
       "cup": "ES 0021 0000 0810 5495 PQ",
       "direccioniberdrola": "",
       "contador": "",
       "peaje": "",
       "pp": "",
       "pll": "",
       "pv": "",
       "contrato": ""
```



En segundo lugar, tenemos los métodos HTTP, que son el método GET (Obtener la representación de un recurso), POST (Crear un nuevo recurso), PUT (Actualizar un recurso) y DELETE (Eliminar un recurso). El usuario tiene que seleccionar uno de estos métodos, dependiendo de cuál sea su interés, para realizar una llamada a la API.

En tercer lugar, tenemos el encabezado, donde el usuario (auditor energético) tiene que especificar la clave de encabezado y el contenido del encabezado para autorizar la solicitud. En cuarto y último lugar, tenemos el órgano de solicitud, dado que hemos seleccionado el método POST, en este caso lo que se tiene que hacer es que el usuario escriba el cuerpo de la solicitud que se adjuntará a la anterior.

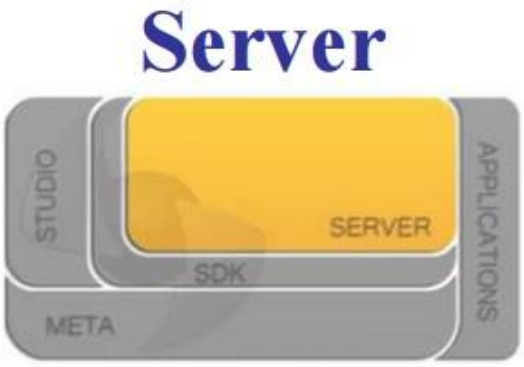
5.2.11 SpagoBI

"*SpagoBI*" es una multiplataforma integrada para la inteligencia de negocios desarrollada enteramente de acuerdo con la filosofía del "software" libre y de código abierto. Dentro de "*SpagoBI*", tenemos distintas arquitecturas o módulos (en la siguiente página hay una imagen gráfica de los mismos), como son "*SpagoBI Server*", "*SpagoBI Meta*" y "*SpagoBI Studio*". En este proyecto se va a usar el módulo de "*SpagoBI Server*" debido a que es el que necesitamos concretamente para las distintas operaciones que se tienen que realizar.

5.2.11.1 SpagoBI Server

"*SpagoBI Server*" es una aplicación web desarrollada en un servidor de aplicaciones J2EE (Tomcat) que se puede ejecutar en cualquier sistema operativo que soporte JMV 1.5. Funciona con un repositorio privado alojado en DBMS (MySQL) y se puede acceder a ella a través de casi todos los principales navegadores.

Se ha elegido este módulo porque ofrece funcionalidades tales como "*Charting*", "*Real-time dashboards and consoles*", "KPI", "*Data Mining*", "*Reporting*" y "OLAP" entre otras muchas más, pero se han recalcado las que en este proyecto tendrán más énfasis. Aquí se muestra una breve estructura del módulo "Server".

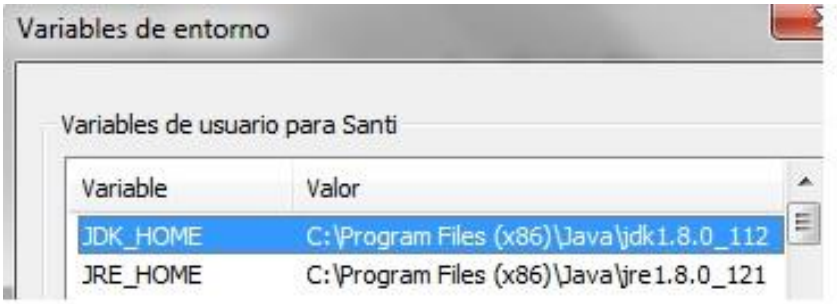


5.2.11.1.1 Instalación

Para la descarga del "software" accedemos a "OW2 Consortium Project Forge" (http://forge.ow2.org). Una vez se haya accedido a la página web, se tiene que descargar "All-In-One-SpagoBI-<etiqueta de la última versión>", este paquete contiene una instalación estándar de SpagoBI que corre en Tomcat, todos los motores ya configurados y una simple demo de funcionalidades de SpagoBI en un repositorio HSQL DB.

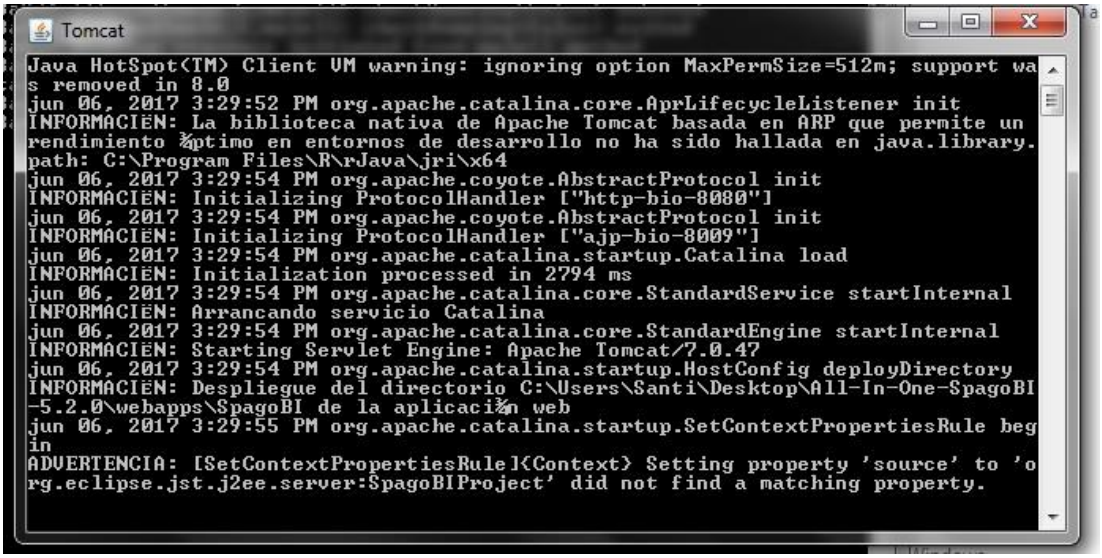
Una vez descargado, descomprimido e instalado SpagoBI, lo primero que se tiene que hacer es modificar el conector que trae por defecto de MySQL, debido a que nosotros vamos a utilizar la versión 10.1.10 de MariaDB, el conector no lo soporta y hay que poner la última versión del mismo. Para ello en "**C:\Users\Santi\Desktop\All-In-One-SpagoBI-5.2.0\lib**" tenemos que eliminar el conector de MySQL que viene por defecto y poner el conector "**mysql-connector-java-5.1.29.jar**".

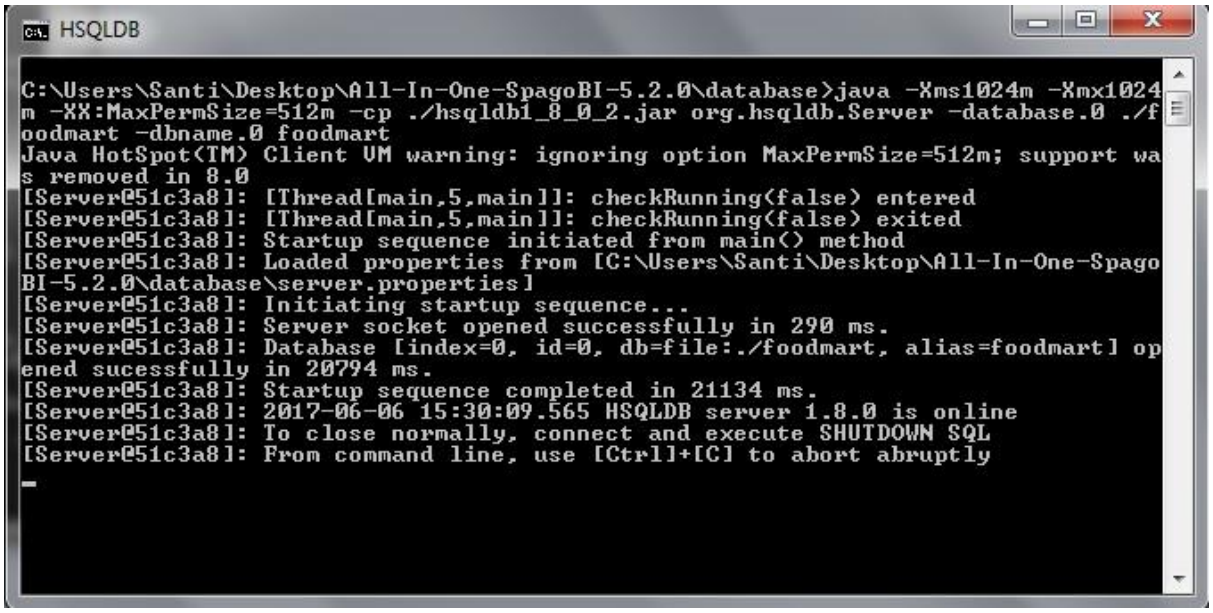
El siguiente paso es configurar las variables de entorno de la cuenta que se esté utilizando, hay que añadir la URL de localización tanto de nuestro Java JDK como de nuestro Java JRE para que el servidor Tomcat pueda funcionar correctamente.



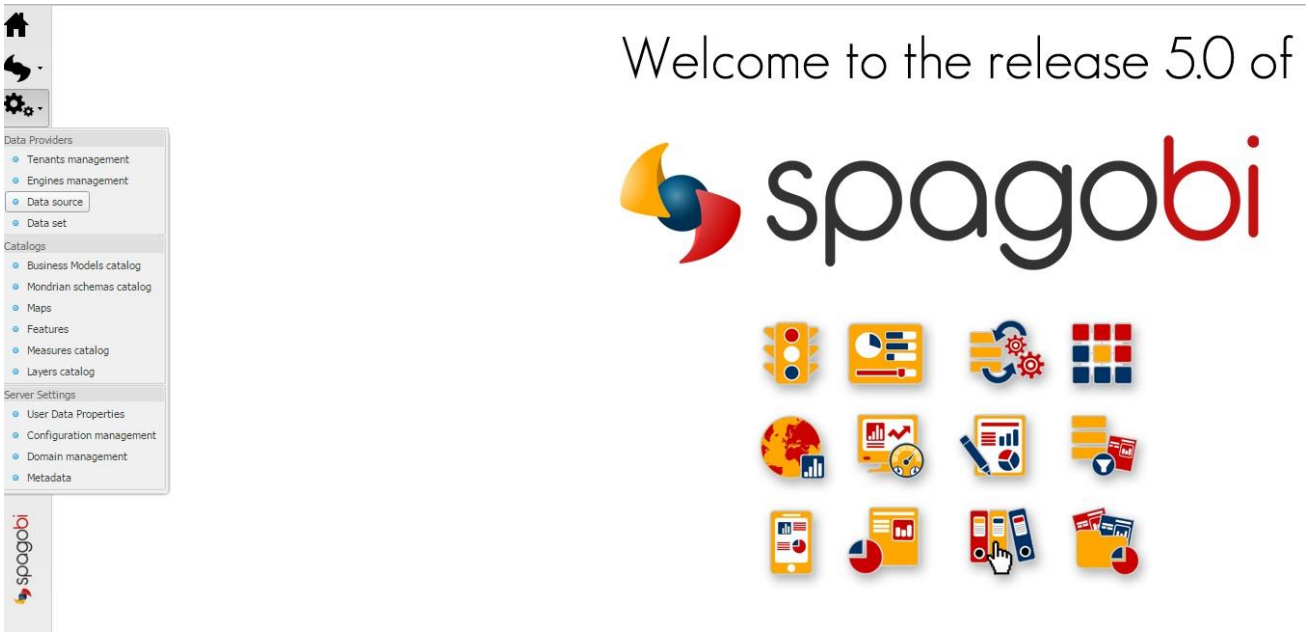
Una vez ya tenemos realizados los pasos anteriores solo nos queda acceder a "***C:\Users\Santi\Desktop\All-In-One-SpagoBI-5.2.0\bin***" y seleccionar el fichero "*SpagoBIStartup*"; recalcar que hay dos ficheros de arranque (uno para Windows y otro para Linux), en este caso como se está usando Windows hay que seleccionar el fichero de archivos por lotes de Windows.

Hacemos doble "*click*" sobre el fichero que he nombrado antes y empieza a cargar el servidor Tomcat al mismo tiempo que el HSQL DB. Esperamos unos pocos segundos a que se carguen bien y accedemos al navegador con la URL "***localhost:8080/SpagoBI***". Se añaden a continuación tres capturas de pantalla como ayuda visual.





En esta captura de pantalla se observa la página de inicio de sesión de SpagoBI, para su acceso pulsaremos sobre el tercer icono de debajo a la izquierda, quiero decir el icono de Administrador. Una vez pulsamos sobre él, se nos muestra la siguiente pantalla:

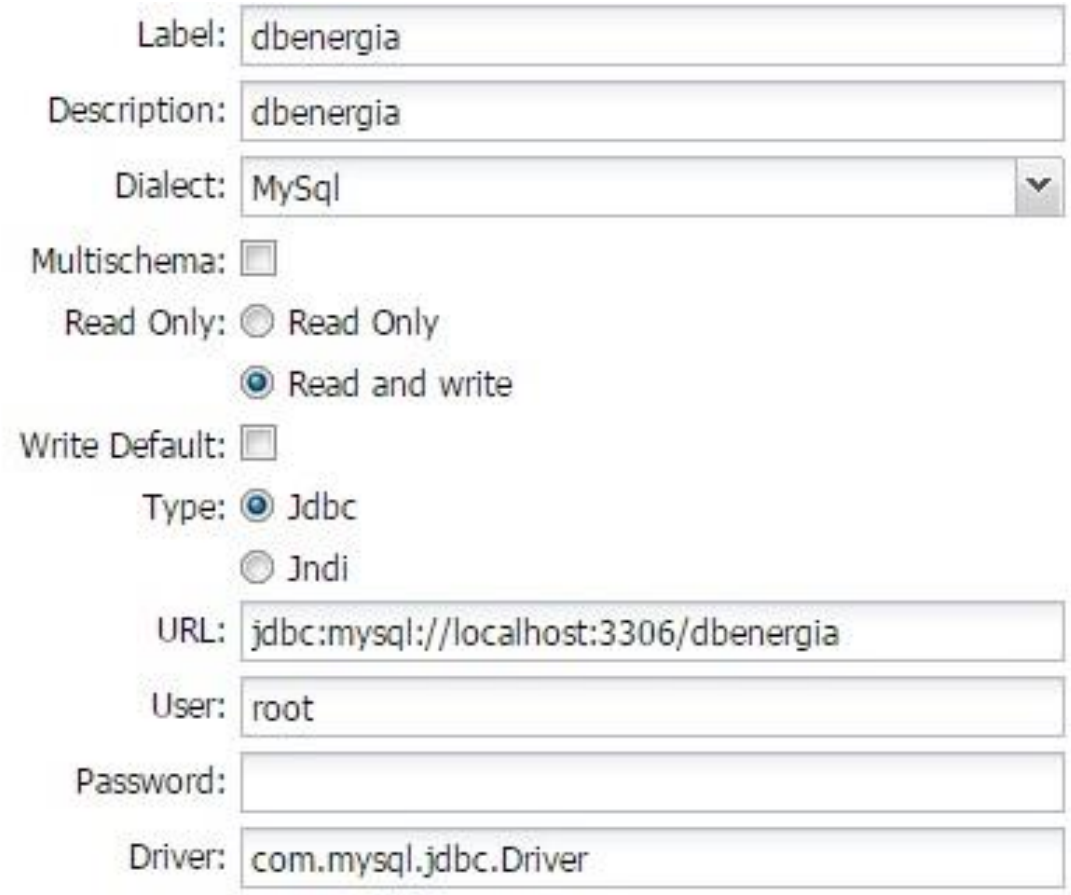


5.2.11.1.2 Configuración

Aquí ya es donde empezamos a realizar la creación de nuestro "*Data Source*". En primer lugar, como en la imagen puede apreciarse, vamos al apartado "*Data Providers*" y seleccionamos "*Data Source*". Una vez lo seleccionamos se nos muestra la siguiente pantalla:

En esta captura se muestran los "*Data Sources*" que tiene por defecto creados el programa en sí mismo, lo que hay que hacer ahora es hacer "*click*" sobre el botón "*Add*" para crear un nuevo "*Data Source*". Cuando pulsamos sobre dicho botón accedemos a la siguiente ventana:

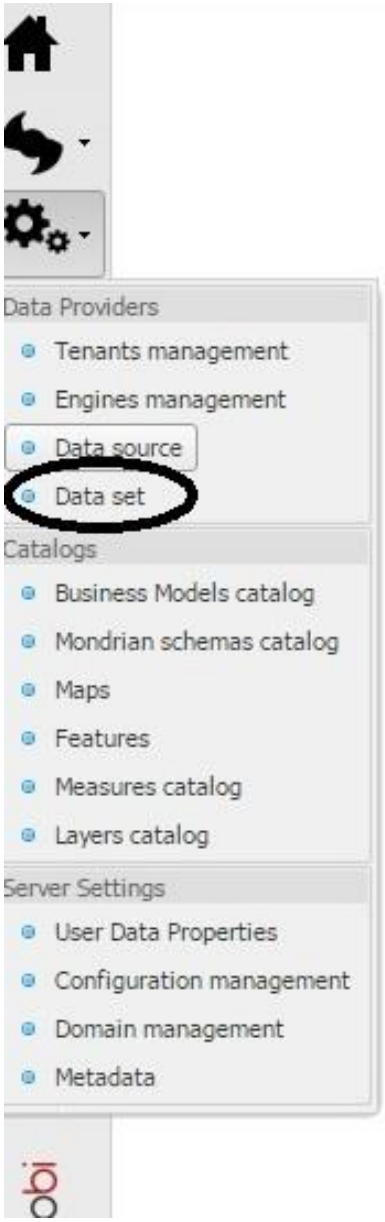
Se puede observar en esta captura que lo primero que se hace es crear un "*Label*", que en el ámbito informático significa dar nombre a una etiqueta concreta, que en nuestro caso es la base de datos. Luego tenemos el campo "*Description*" donde se hace una descripción del "*label*" que hemos creado, en mi opinión este campo es mejor rellenarlo con el mismo nombre que el "*label*", asique le asignamos el nombre de la base de datos.



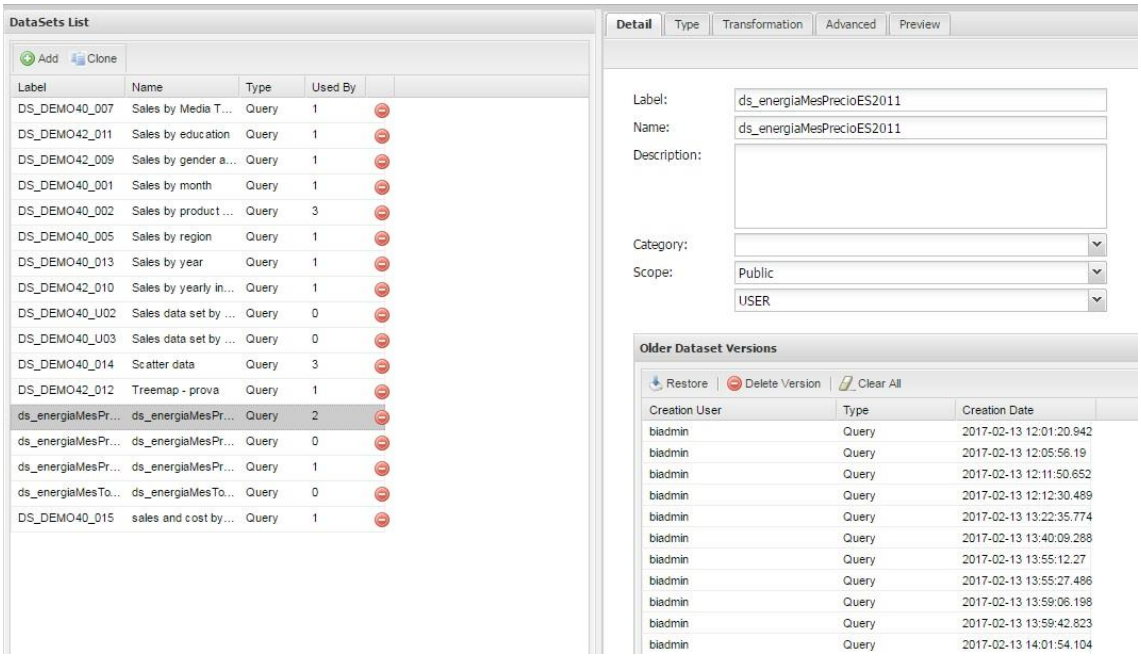
El siguiente paso es elegir el formato que vamos a querer utilizar en dicho "*Data Source*". Como nos interesa tener nuestra base de datos para poder generar gráficas de los consumos eléctricos de los edificios, ponemos MySQL que es con el que hemos trabajo la base de datos. Los siguientes dos apartados "*Multischema*" y "*Write Default*" vienen configurados por defecto, por lo que no se tocan.

Luego tenemos el URL, donde tenemos que poner el "*jdbc*", que es la ruta donde se encuentra nuestra base de datos alojada para así poder conectar SpagoBI con ella. Por último, tenemos el "*Driver*", parte importante para poder conectar la base de datos con SpagoBI, que es el que se muestra en la captura. Una vez tenemos todos estos campos rellenados, guardamos el "*Data Source*" y ya lo tenemos creado y accesible.

El siguiente paso a realizar es la creación del "*Data Set*". En la ventana de inicio hay que acceder al apartado "*Data Providers*" y seleccionar "*Data Set*". En la siguiente captura de pantalla lo vemos de forma más visual.

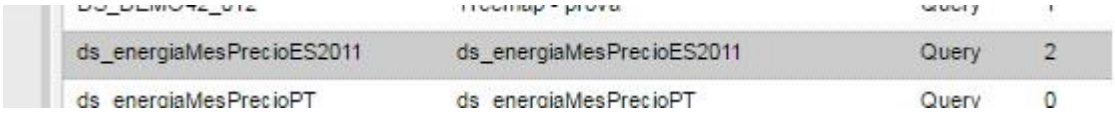


Una vez seleccionamos que se quiere acceder a dicho apartado, se muestra la siguiente pantalla de configuración, en donde vemos toda la lista de "*Data Set*" creados por defecto por el programa y la parte de creación de los nuestros propios.

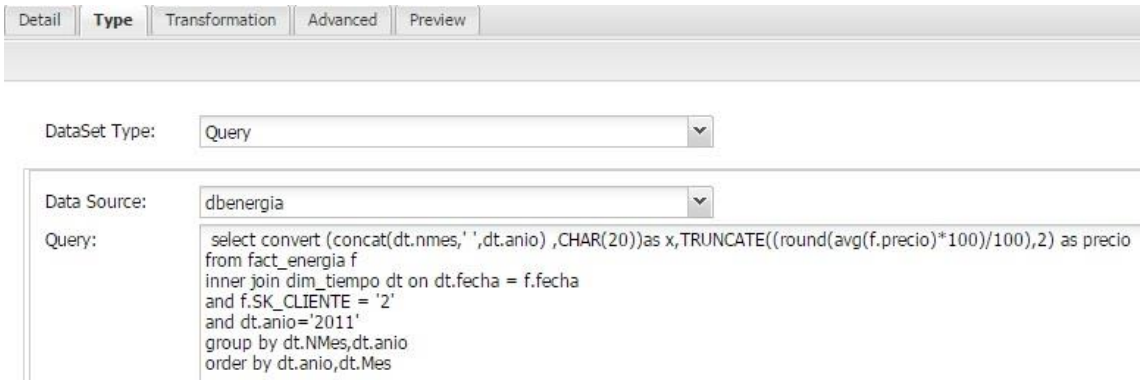


Del mismo modo que se hizo en el apartado anterior del "*Data Source*", en la creación de un nuevo "*Data Set*" tenemos que pulsar sobre el botón "*Add*" para configurar los campos que vemos en la captura. En primer lugar, tenemos el campo "*Label*", que como en el caso anterior, se trata de dar nombre a una etiqueta, en este caso se le da el nombre de "*ds_energiaMesPrecioES2011*".

En el campo "*Name*", para evitar confusiones, se ha utilizado el mismo nombre. Los dos siguientes campos de la configuración se dejan en blanco mientras que los dos últimos los tenemos que configurar de forma de que quede el apartado "*Scope*" tal y como se aprecia en la captura. En la siguiente captura de pantalla se puede apreciar cómo se ha creado correctamente el "*Data Set*".

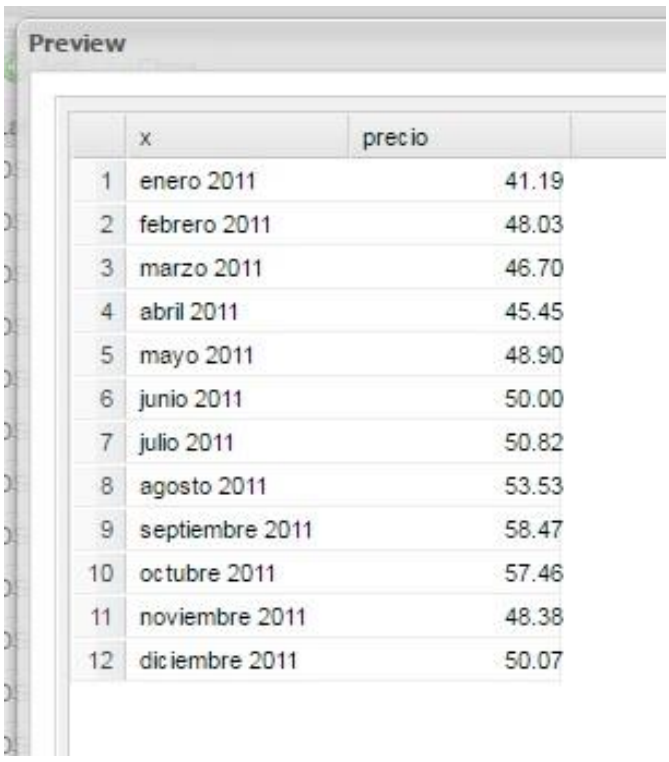


En la siguiente captura de pantalla se va a mostrar cómo realizar la creación de la consulta que deseamos realizar a nuestra base de datos. En primer lugar, lo que se tiene que hacer es acceder al "*tab*" "*Type*" para comenzar el proceso.



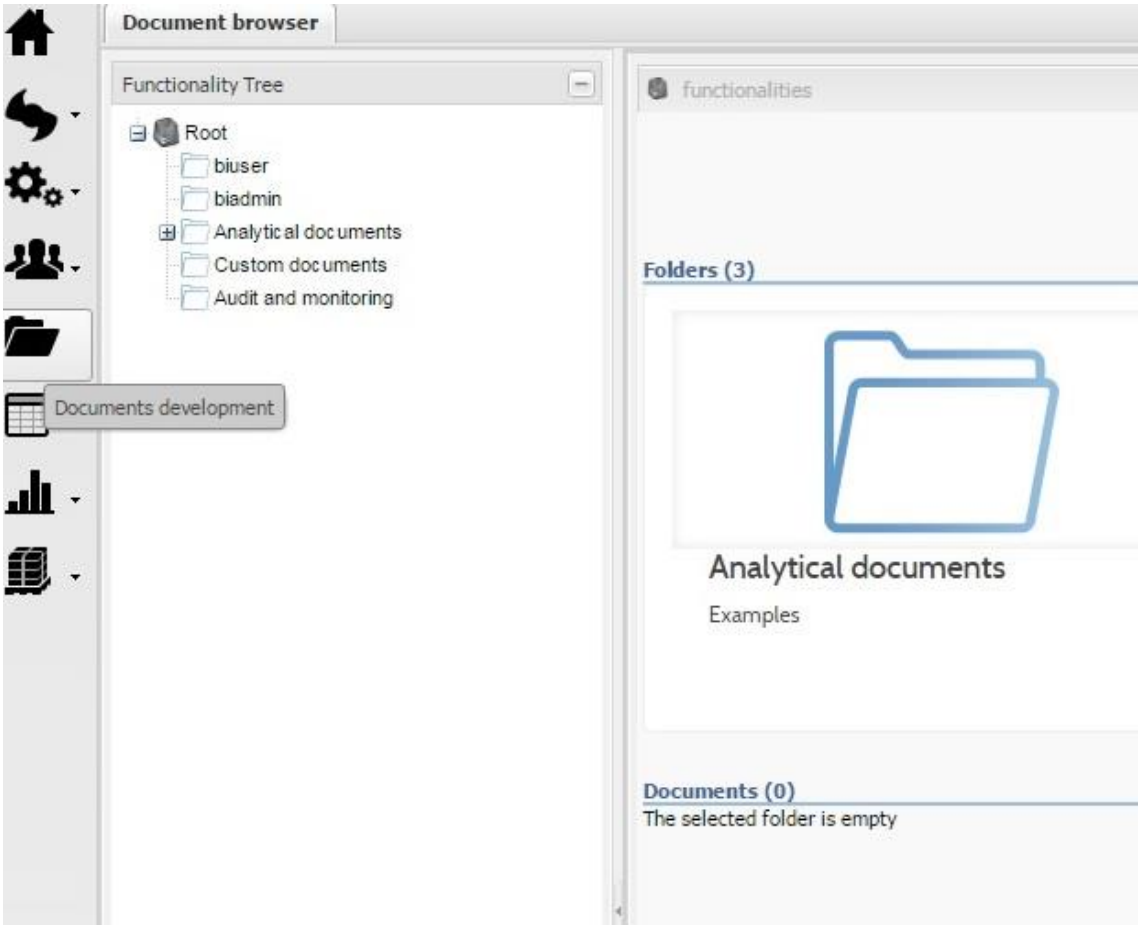
Aquí se observa como en el "*tab*" "*Type*" se tiene que seleccionar el tipo de "*dataset*" que se quiere utilizar, como se quiere realizar una consulta a una base de datos, el tipo de "*dataset*" que se elige es el "*Query*". El siguiente paso es seleccionar el "*data source*" sobre el cual se va a llevar a cabo la consulta, como hemos definido en el apartado anterior, se va a realizar sobre nuestra base de datos "*dbenergia*". Y por último en el campo "*Query*" observamos la consulta programada al completo para que nos de los resultados que queremos.

Recalcar que esta es una consulta de otras muchas que se han realizado en el proyecto. Para pre-visualizar el resultado de esta consulta y ver que la hemos programado correctamente, se accede al "*tab*" "*Preview*" y en la parte superior derecha se tiene el botón "*Preview*" para pre-visualizar el resultado final de la consulta, si el resultado obtenido es el deseado, hacemos "*click*" sobre el botón "*Save*" y se guarda el "*dataset*". En la siguiente captura de pantalla se muestra de manera más visual.

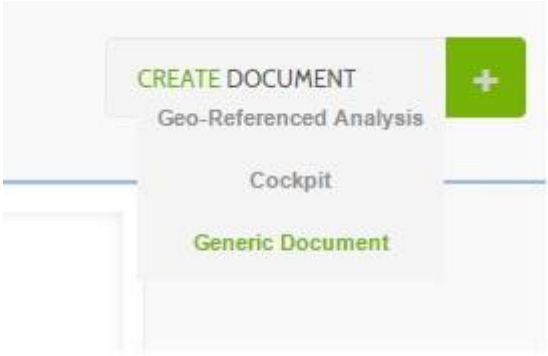


	x	precio
1	enero 2011	41.19
2	febrero 2011	48.03
3	marzo 2011	46.70
4	abril 2011	45.45
5	mayo 2011	48.90
6	junio 2011	50.00
7	julio 2011	50.82
8	agosto 2011	53.53
9	septiembre 2011	58.47
10	octubre 2011	57.46
11	noviembre 2011	48.38
12	diciembre 2011	50.07

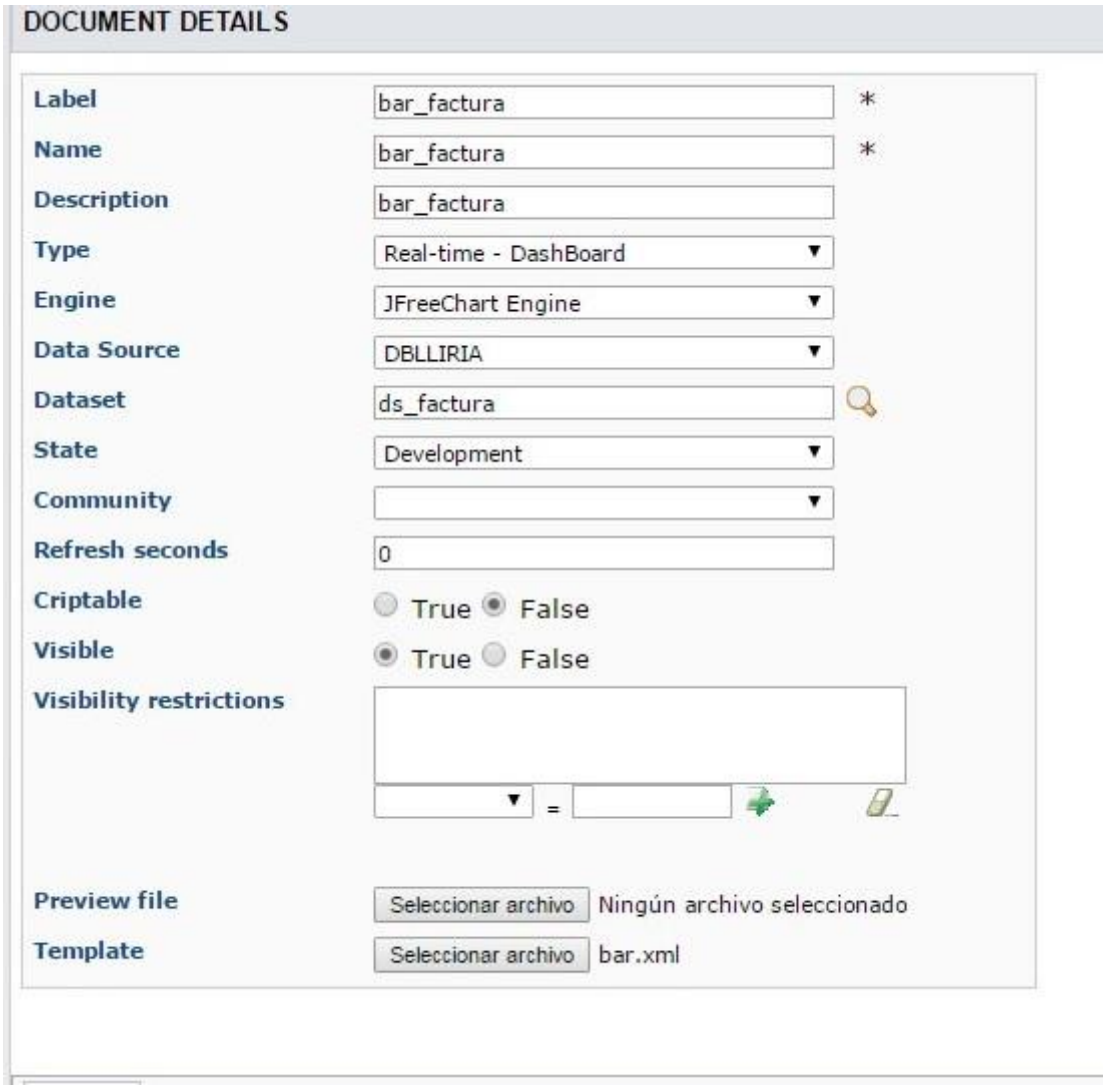
El siguiente paso a realizar una vez se tiene el "*dataset*" y el "*datasource*" creados es elegir qué tipo de "*Chart*" queremos para nuestra representación gráfica, ya que hay muchos tipos y formatos. Para empezar a crearlo, lo primero que se tiene que hacer es acceder a la ventana principal de SpagoBI y entrar dentro del apartado "*Documents developement*".



Una vez se ha accedido a este apartado, en la parte superior derecha, tenemos el desplegable *"Create Document +"* en el cual se nos muestran tres posibles opciones de selección, la opción que se tiene que seleccionar es la de "*Generic Document*".



Una vez se accede al apartado "*Generic Document*", se nos muestra la siguiente pantalla:



Los tres primeros campos son de nombramiento, con lo cual una manera fácil y sencilla de nombrar cuando estamos a ciertos niveles de dificultad es poner el mismo nombre sabiendo que no afecta para nada que lo tengan. Luego tenemos el tipo, que siempre será "*Real-time-Dashboard*", se seleccionará siempre ese tipo debido a que es el que se corresponde a nuestras necesidades.

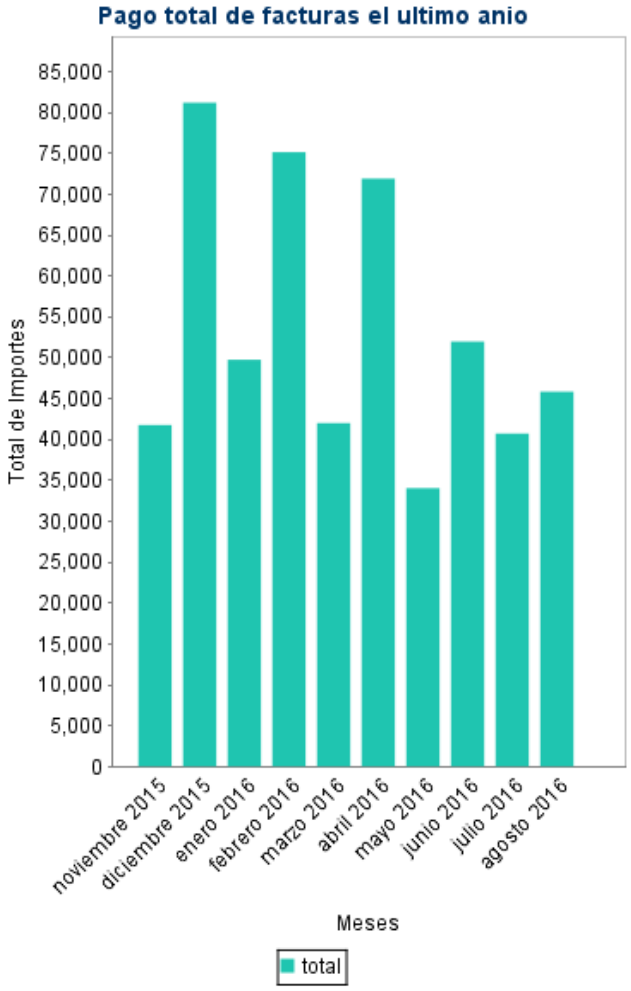
Luego tenemos que seleccionar en el campo "*Engine*" "*JFreeChart Engine*". Se selecciona este porque el "*template*" creado es para hacer uso de esa librería. El siguiente paso es seleccionar nuestro "*datasource*" creado anteriormente, en este proyecto se han tenido que crear dos distintos, uno llamado "*dbenergia*" y otro llamado "*dblliria*", en la imagen aparece la selección del "*datasource*" "*dblliria*".

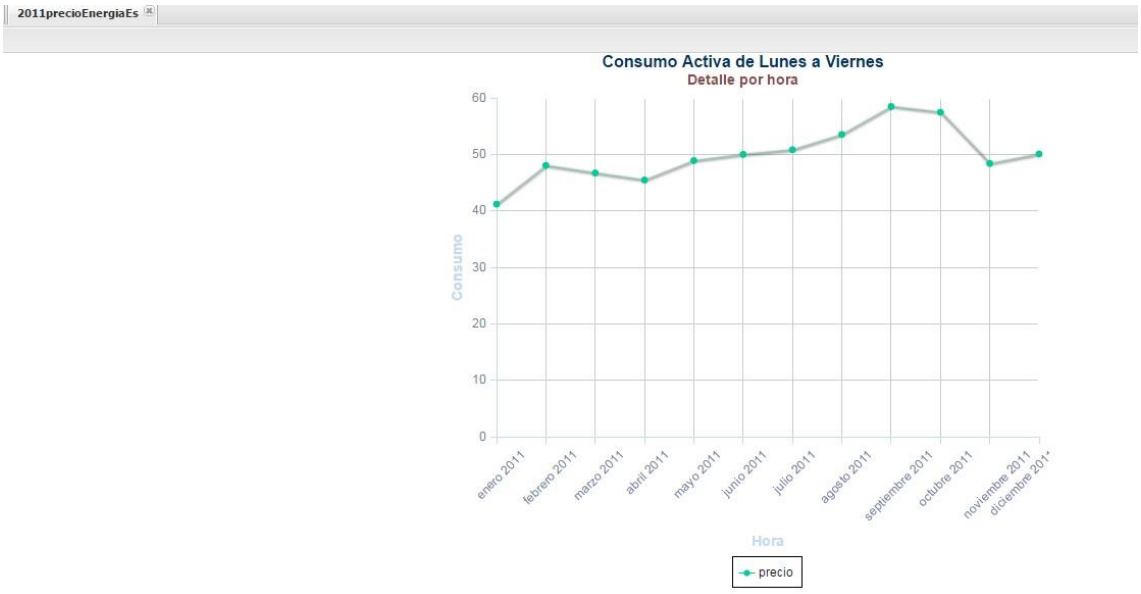
Luego seleccionamos el "*dataset*" que hayamos creado, en este caso se ha seleccionado el "*dataset*" "*ds_factura*" de la base de datos "*dblliria*". El resto de campos de dejan como están por defecto excepto el último, en donde tenemos que importar nuestro "*template*". En este caso se va a importar el "*template*" (bar.xml) que nos muestra los resultados en barras. Recalcar que se han tenido que modificar las líneas diez y once del fichero, donde se ha cambiado el valor por defecto del campo "*value*" por el valor **Meses** y **Total de importes** en cada una de las líneas respectivamente. Aquí observamos su estructura:

bar.xml

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <BARCHART type='overlaid_barline' name='Pago total de facturas el ultimo anio'>
3    <DIMENSION width='380' height='570'/>
4      <STYLE_TITLE font='Arial' size='14' color='#003366'/>
5      <STYLE_SUBTITLE font='Arial' size='12' color='#1F57C5' name=''/>
6      <STYLE_LABELS_DEFAULT font='Arial' size='12' color='#000000'/>
7      <STYLE_X_AXIS_LABELS font='Arial' size='12' color='#000000'/>
8      <STYLE_Y_AXIS_LABELS font='Arial' size='12' color='#000000'/>
9    <CONF>
10   <PARAMETER name='category_label' value='Meses'/>
11   <PARAMETER name='value_label' value='Total de Importes'/>
12   <PARAMETER name='n_visualization' value='10'/>
13   <PARAMETER name='view_filter' value='false'/>
14   <PARAMETER name='view_slider' value='false'/>
15   <SERIES_COLORS total='#1FC5B0'/>
16   <SERIES_DRAW total='bar'/>
17   </CONF>
18 </BARCHART>
```

Una vez ya se ha cargado el "*template*" que deseamos y tenemos los campos correctamente rellenados, lo guardamos. El siguiente paso es ir al apartado "*Analytical documents*" y allí está guardado nuestro documento creado. Hacemos "*click*" sobre él y se puede observar la siguiente imagen:


Pago total de facturas el ultimo anio
Total de Importes
85,000
80,000
75,000
70,000
65,000
60,000
55,000
50,000
45,000
40,000
35,000
30,000
25,000
20,000
15,000
10,000
5,000
0
noviembre 2015
diciembre 2015
enero 2016
febrero 2016
marzo 2016
abril 2016
mayo 2016
junio 2016
julio 2016
agosto 2016
Meses
total


2011precioEnergiaEs
Consumo Activa de Lunes a Viernes
Detalle por hora
Consumo
60
50
40
30
20
10
0
enero 2011
febrero 2011
marzo 2011
abril 2011
mayo 2011
junio 2011
julio 2011
agosto 2011
septiembre 2011
octubre 2011
noviembre 2011
diciembre 201
Hora
precio

La primera captura se puede apreciar claramente el diseño de las barras y como estas representan los datos de la base de datos de una manera más gráfica. La segunda captura se muestra para enseñar otra forma de "*template*". Este es de una forma lineal y hace referencia al consumo de energía activa de todo el año dos mil once de los días laborables. Se pueden apreciar los variantes cambios que hay entre los doce meses.

5.2.12 Netbeans IDE

NetBeans IDE es un entorno de desarrollo, una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas. Está escrito en Java, pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el NetBeans IDE.

NetBeans IDE es un producto libre y gratuito sin restricciones de uso. En este proyecto se ha usado la versión 8.2, debido a que era la última y más completa de todas. Se ha elegido este entorno de desarrollo porque facilita mucho el trabajo al programador, debido a sus cualidades de crear interfaces graficas de una forma sencilla y dar opciones de autocompletado de código a la hora de programar.

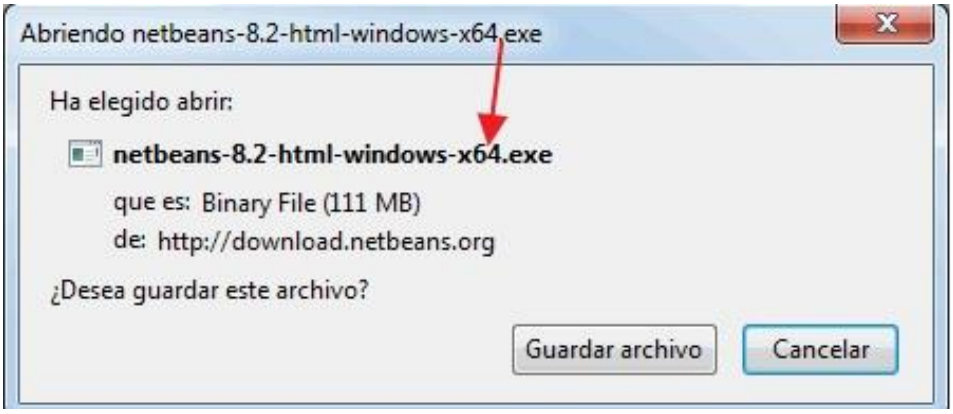
5.2.12.1 Instalación

Para descargar "Netbeans IDE" accederemos a su página oficial con el siguiente URL https://netbeans.org/downloads/index.html. Una vez se haya accedido, se nos muestra la siguiente pantalla en donde tenemos las opciones de seleccionar el idioma en el que queremos descargar el "*software*" y el entorno de desarrollo en el que se le va a dar uso.

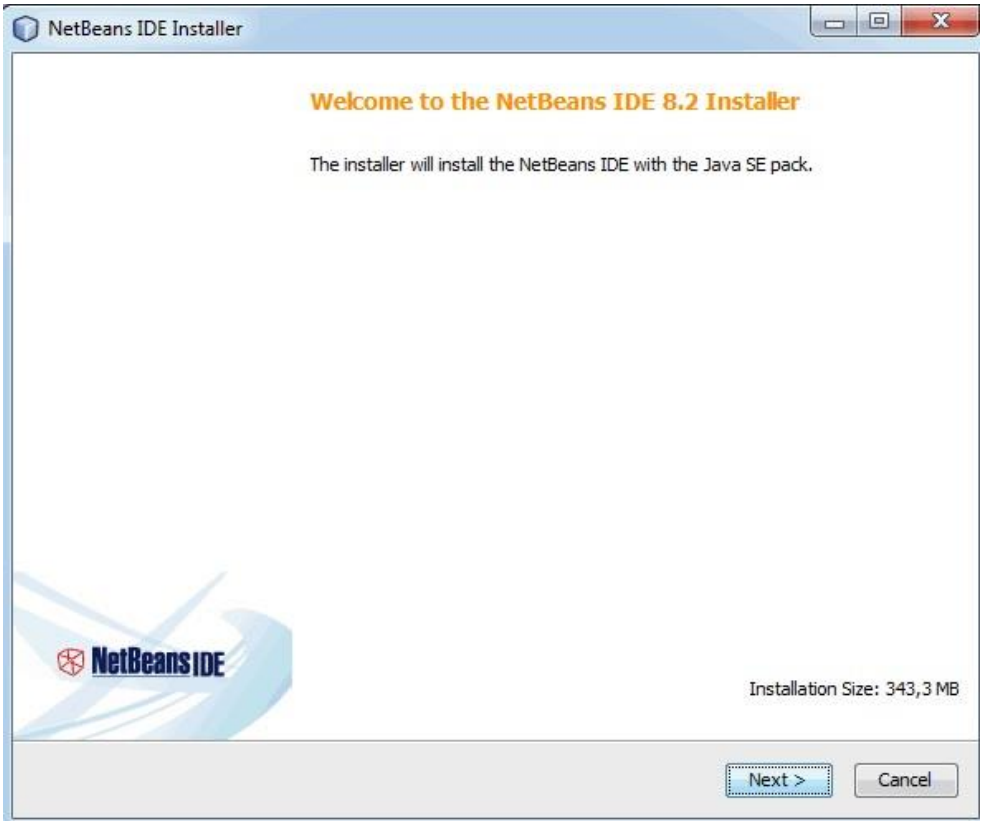
Además de las opciones que nombraba antes de selección de idioma y plataforma, se tiene la posibilidad de elegir qué tipo y que tecnologías se quiere que posea "Netbeans". En este caso seleccionaremos el último, que contiene todas las tecnologías y es el más completo.

Una vez hagamos "*click*" sobre el botón "*Download*" se procede a su descarga. Una vez se descarga el "software" en nuestro ordenador se puede empezar a realizar la instalación. Un dato muy importante a tener en cuenta en la instalación es que se tiene que tener instalada en el sistema la versión "JDK 8.112" de Java o superiores para que la versión que se ha descargado de "Netbeans" se pueda instalar y funcionar de forma correcta.

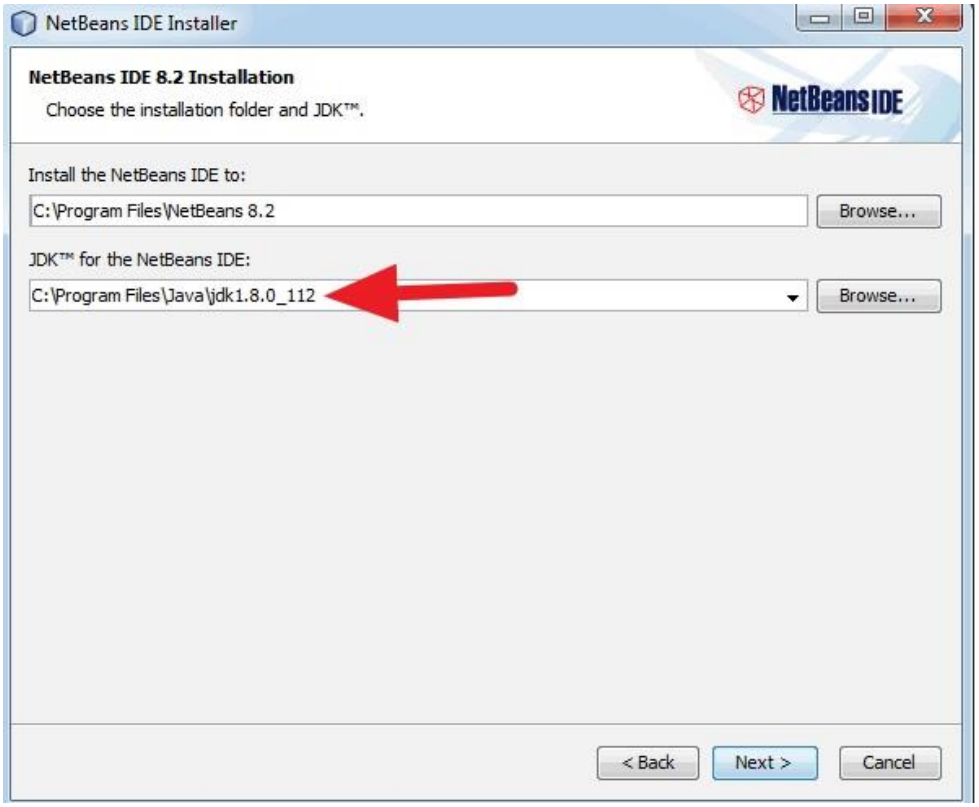
Si no se tuviese esa versión del JDK de Java, no pasa nada, ahora describiré mediante pasos gráficos como descargar dicha versión. Ahora voy a mostrar los pasos de instalación de "Netbeans IDE 8.2" mediante capturas de pantalla para que se tenga una forma más visual y entendible de su proceso.



En esta imagen se observa cómo se descarga el tipo "x64", que es el mismo en el que está basado el sistema donde se va a utilizar. Pulsamos "Guardar Archivo" y se procede, como he dicho antes, a su descarga en nuestro PC.

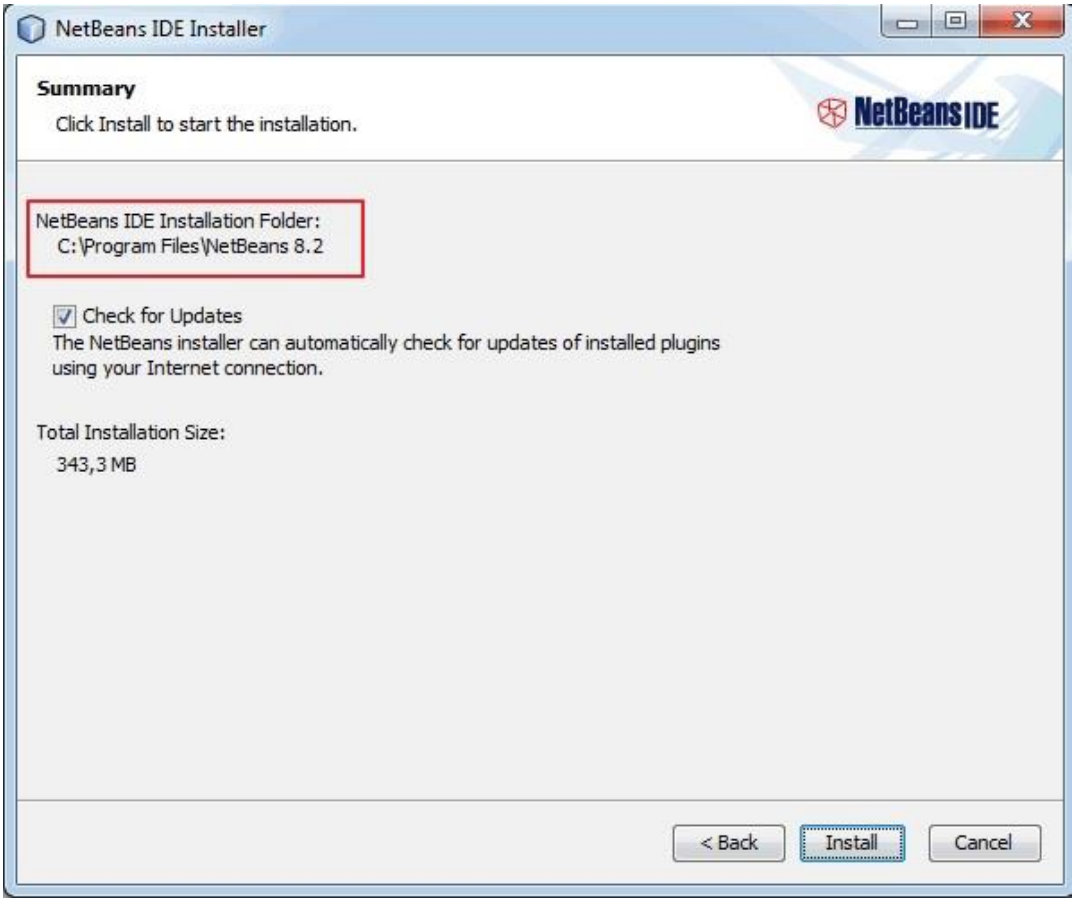


Esta es la pantalla del inicio de la instalación de “Netbeans IDE”. Solo hay que pulsar el botón “Next” para seguir con los siguientes pasos, donde el siguiente es este:

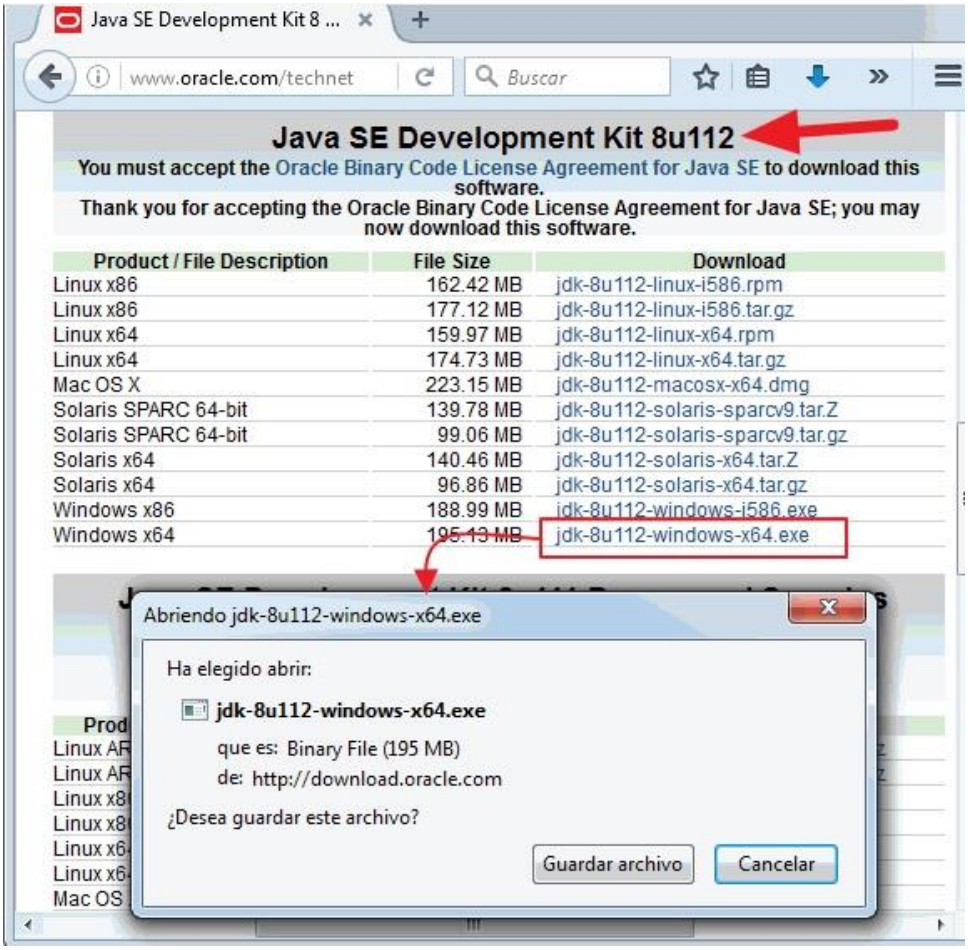


Como he comentado en los párrafos anteriores, para la correcta instalación se tiene que tener la versión "JDK 8.112" de Java o superiores por temas de compatibilidad, como es en este caso. En esta captura se observa cómo se nos muestran los directorios tanto de donde se va a proceder a la descarga y la guarda de "Netbeans" como donde se encuentra alojado el JDK de Java. Ambas rutas de almacenamiento se pueden cambiar por otras, pero no es lo recomendable, bajo mi punto de vista creo que es mejor dejar las rutas de almacenamiento que vienen por defecto.

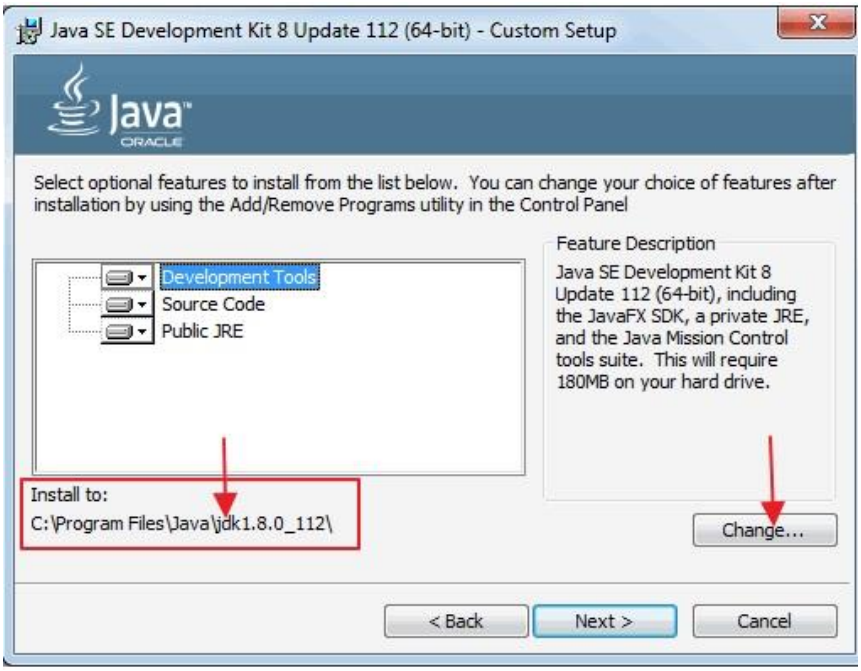
Una vez tenemos las rutas de almacenamiento definidas, pulsamos el botón "*Next*" y nos aparece la siguiente pantalla, en donde solo tenemos que pulsar el botón "*Install*" para realizar la instalación. Y con estos pasos se procede a la instalación de "Netbeans IDE".



Ahora bien, si se tuviese el problema de que no tenemos en nuestro sistema la versión de JDK de Java adecuada, los pasos que tendríamos que realizar para solucionarlo serían los siguientes. En primer lugar, se accede a la siguiente URL "http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html" y se muestra la siguiente captura de pantalla.



Aquí se selecciona la versión del sistema operativo y el tipo de sistema del que se dispone, y descargamos el fichero correspondiente. En la captura se be claro toda esa información. El siguiente paso que se realiza es el que se muestra en la captura a continuación:

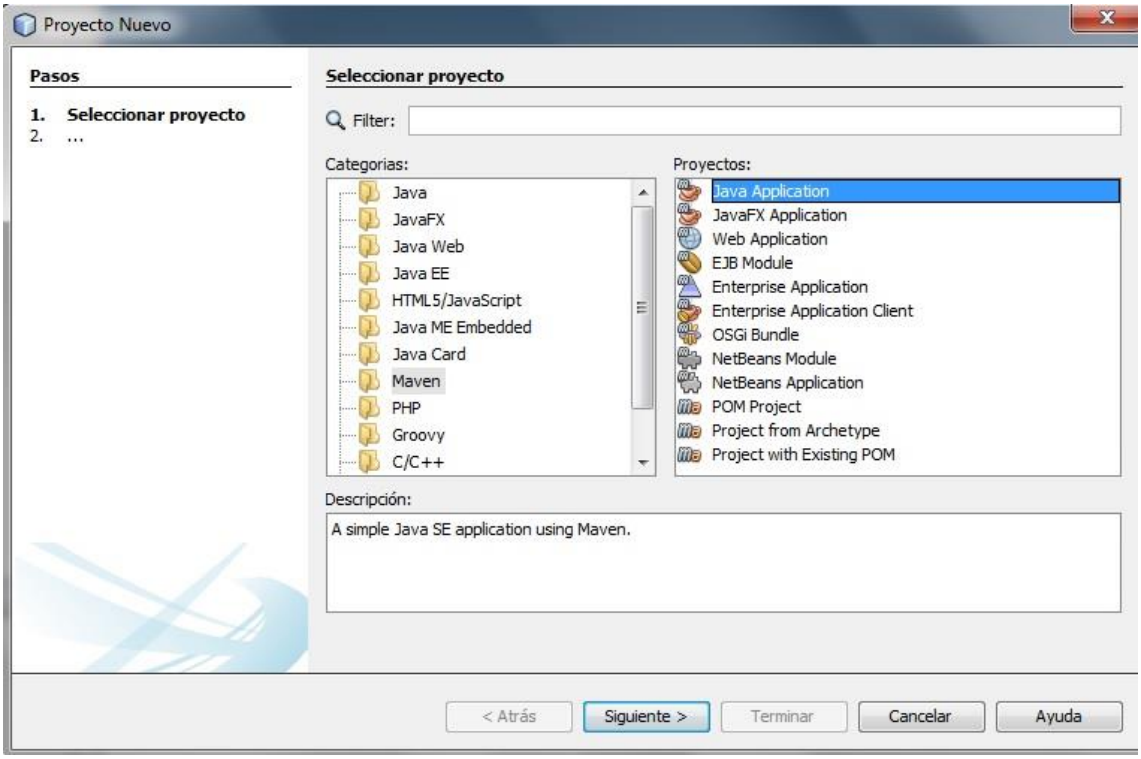


Se puede observar como en este paso se va a realizar la instalación del JDK en su correspondiente ruta de almacenamiento, como he dicho antes esa ruta viene por defecto, siempre se puede cambiar por otra que el usuario considere oportuna. Una vez se tiene ya la ruta definida, procedemos a pulsar el botón "*Next*" y se instala el JDK de Java.

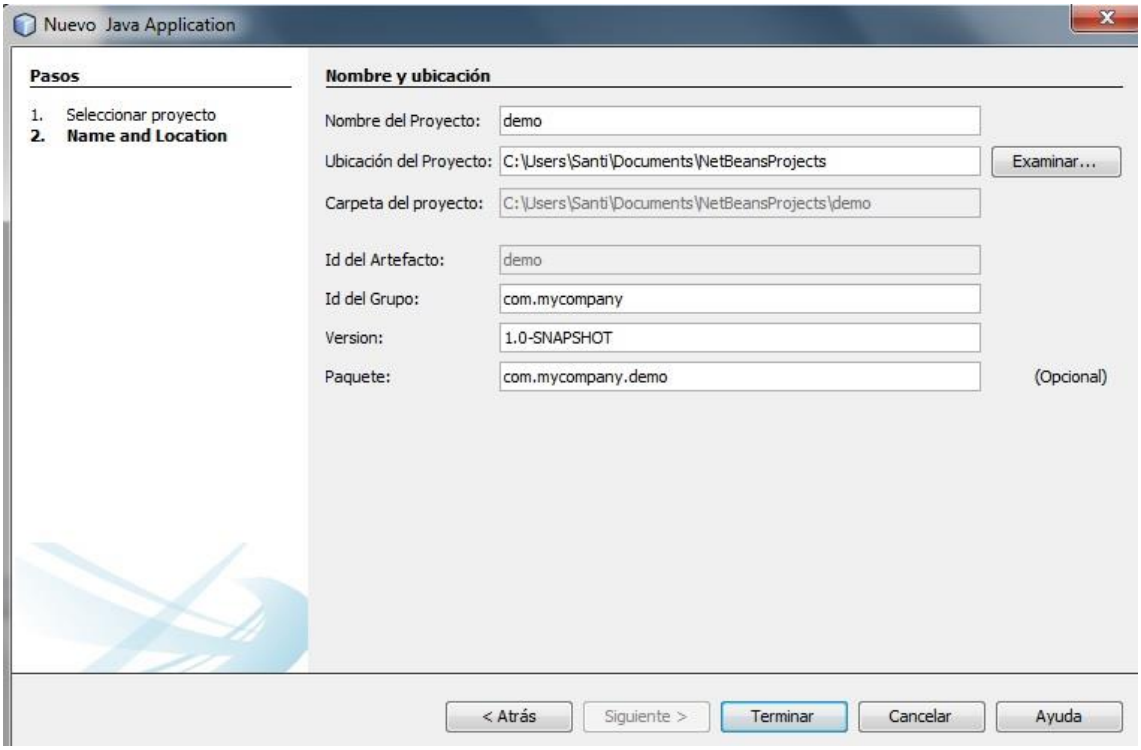
Una vez ya tenemos las dos partes instaladas correctamente, solo nos queda iniciar "*Netbeans* IDE 8.2" y empezar a trabajar.

5.2.12.2 Creación del proyecto

El primer paso que se tiene que hacer es la de la creación del proyecto donde tendremos alojado toda nuestra aplicación. Tal y como se muestra en la captura, se debe crear un proyecto nuevo de categoría "*Maven*" y de tipo "Java *Application*". Se ha elegido "*Maven*" porque tiene un modelo de configuración de construcción más simple que otros de su misma categoría y además está listo para usar en red. Pero una de sus características más a destacar sin duda es la de que utiliza un "*Project Object Model*" (POM) para describir el proyecto de "*software*" a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos.



Una vez se elige el tipo de proyecto que se quiere realizar, se pulsa el botón “Siguiente” y se muestra la siguiente captura de pantalla, en donde se debe definir el nombre y la ubicación del proyecto. Una vez ya hemos rellenado los campos acordes con nuestras necesidades, pulsamos el botón “Terminar” y tenemos ya creado nuestro proyecto.



5.2.12.3 Distribución de las plantillas

En este apartado se va a explicar cómo está distribuido el proyecto, tanto a nivel de paquetes como a nivel de ficheros. En esta captura vemos todas las carpetas que forman parte del proyecto, teniendo cada una su correspondiente importancia.

Se ve desglosada en esta captura cuatro paquetes de una forma clara, en el primero llamado "*es.upv.lemus*", se tienen dos ficheros encargados del arranque de la aplicación. En el siguiente paquete "*es.upv.lemus.controller*" se tienen todos los controladores de la aplicación, destacando los que son de vital importancia que son los que se remarcan en color verde(Contrato, DetalleContrato, DimTiempo, Edificio, Factura).

El siguiente paquete "*es.upv.lemus.controller.excepcions*" contiene los ficheros encargados de controlar las excepciones que se nos pueden aparecer a la hora de la creación del código para la implementación del proyecto en la parte controladora, o mejor dicho a la hora de implementar los distintos controladores de la aplicación.

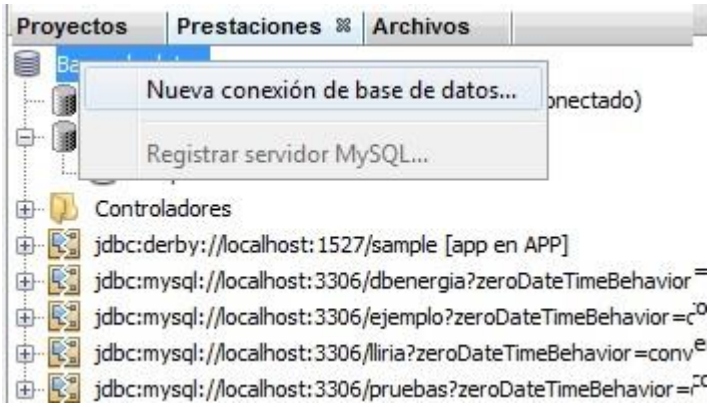
Luego se tiene el paquete "*es.upv.lemus.model*" donde se encuentras las clases Java alojadas, que al igual que en el caso de los controladores, se han definido las clases que resaltan en verde para el proyecto, dichas clases son "Contrato.java", "DetalleContrato.java", "DimTiempo.java", "Edificios.java" y "Factura.java".

Por último tenemos el paquete "*es.upv.lemus.repositiry*", donde se guarda para clase java definida anteriormente su correspondiente repositorio. En estas clases se da la opción de consumir las operaciones CRUD ("*Create*", "*Read*", "*Update*", "*Delete*") dentro de cada uno de los repositorios. Al final de la captura se puede observar como he resaltado el fichero "pom.xml", la principal característica por la que se ha decidido utilizar "*Maven*". En este fichero destacan principalmente las dependencias definidas en el proyecto. A continuación, muestro una captura para mostrarlo.

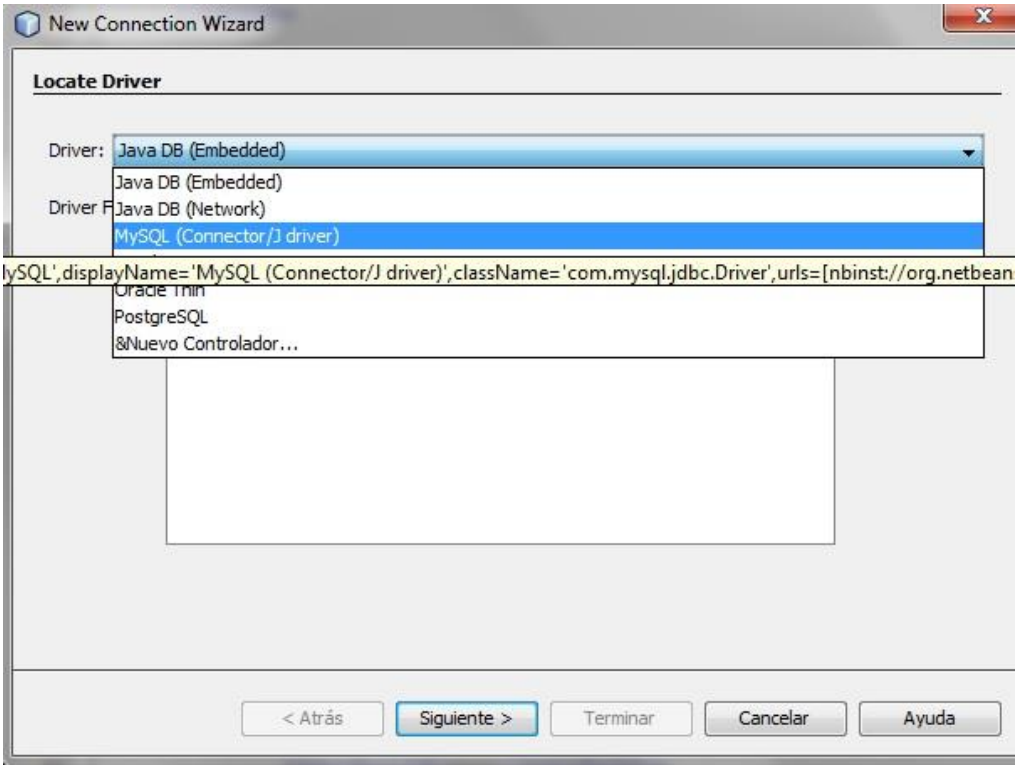
```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-rest</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-log4j2</artifactId>
    </dependency>
    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.8.0</version>
    </dependency>
```

5.2.12.4 Conexión Base de Datos

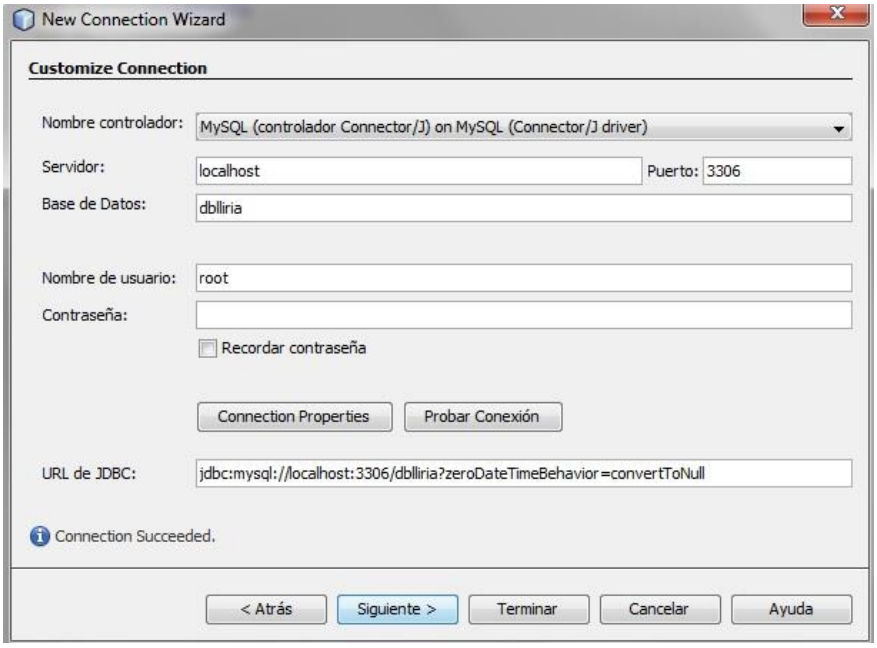
En este apartado se va a proceder a la explicación de cómo se debe realizar la conexión a la base de datos desde Netbeans. Es muy importante tener la base de datos operativa a la hora de la creación, sino Netbeans no podrá encontrarla y nos mostrará que hay un error de conexión. En la siguiente página se muestran diversas capturas para la explicación paso a paso de cómo hacer la correspondiente creación.



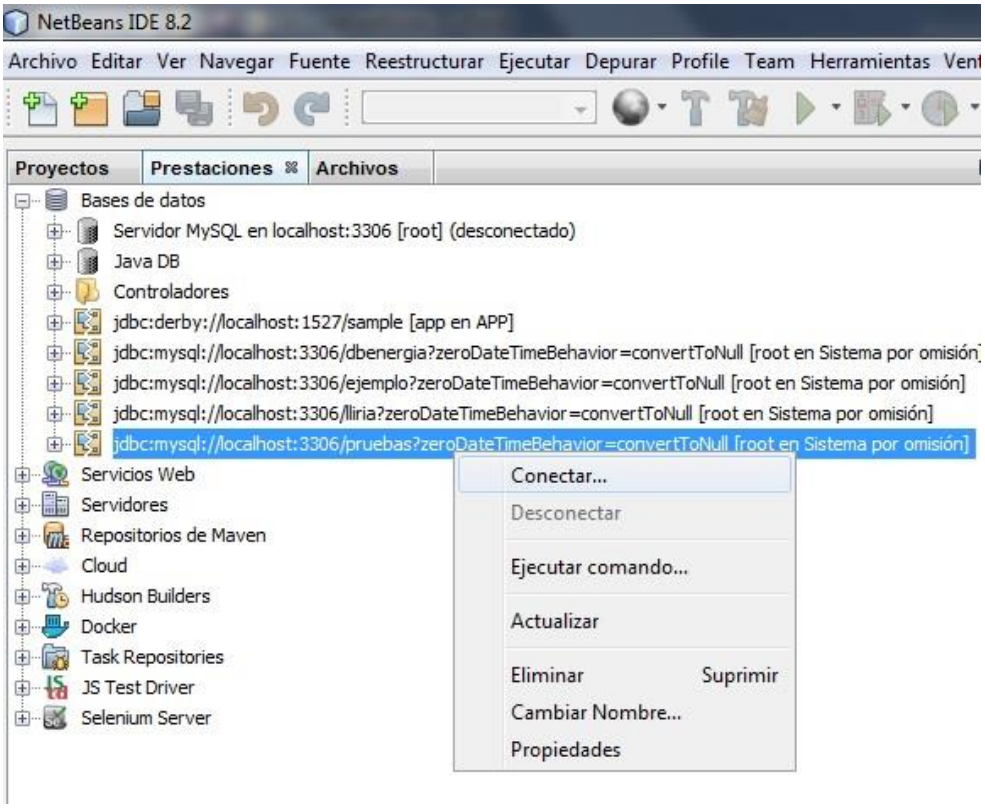
El primer paso es acceder al apartado “Prestaciones” y hacer “*click*” con el botón derecho sobre el icono de la base de datos. Luego solo tenemos que pulsar sobre “Nueva conexión de base de datos”. Cuando se proceda a la opción de crear una nueva conexión de base de datos, se nos muestra la siguiente captura:



Hay que seleccionar el “Driver” correcto, en este caso se selecciona el de MySQL. Una vez se tiene seleccionado hacemos “*click*” en “Siguiente”. A continuación, se muestra la siguiente ventana:



En esta ventana se puede observar como tenemos el controlador seleccionado en la ventana anterior además de los datos del servidor y el nombre de la base de datos. En la parte inferior de la imagen se puede apreciar cómo pone "*Connection Succeeded*", que quiere decir que la conexión con la base de datos ha sido exitosa. Pulsamos el botón "Terminar" y ya tenemos nuestra base de datos creada. Ahora solamente queda como se puede ver en la imagen, seleccionar la conexión que se ha creado a la base de datos, y conectarnos.



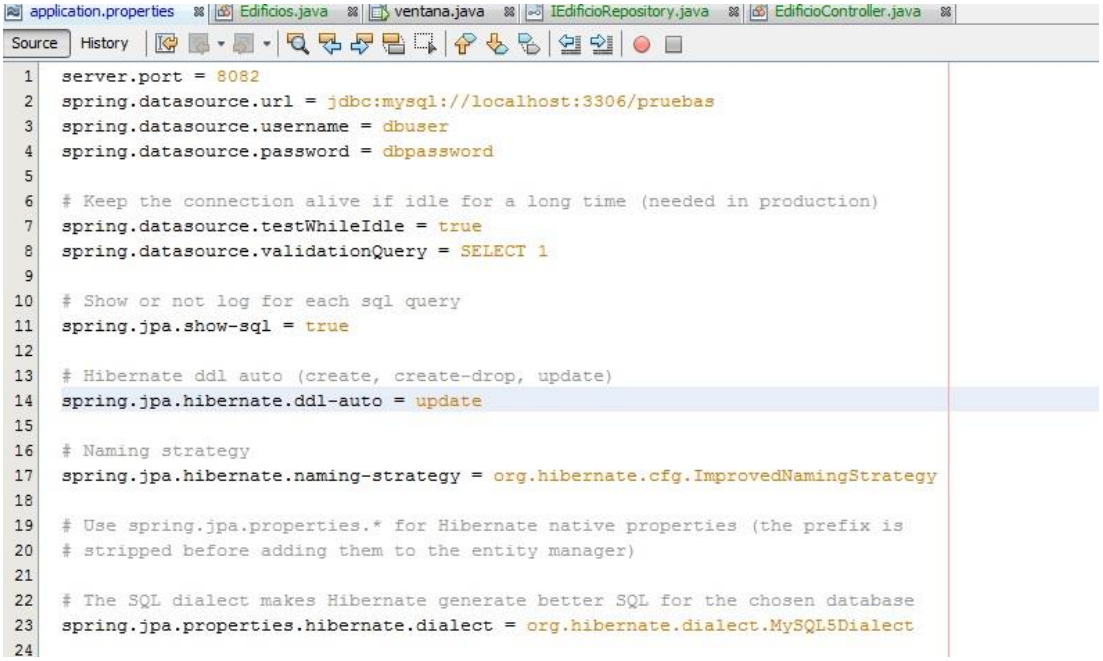
5.2.12.5 Archivos Destacados

En este apartado se van a analizar los archivos más importantes y que tienen más relevancia en el proyecto, donde están, los creados para cada una de las tablas de la base de datos en cada uno de los paquetes anteriormente nombrados.

5.2.12.5.1 Application.properties

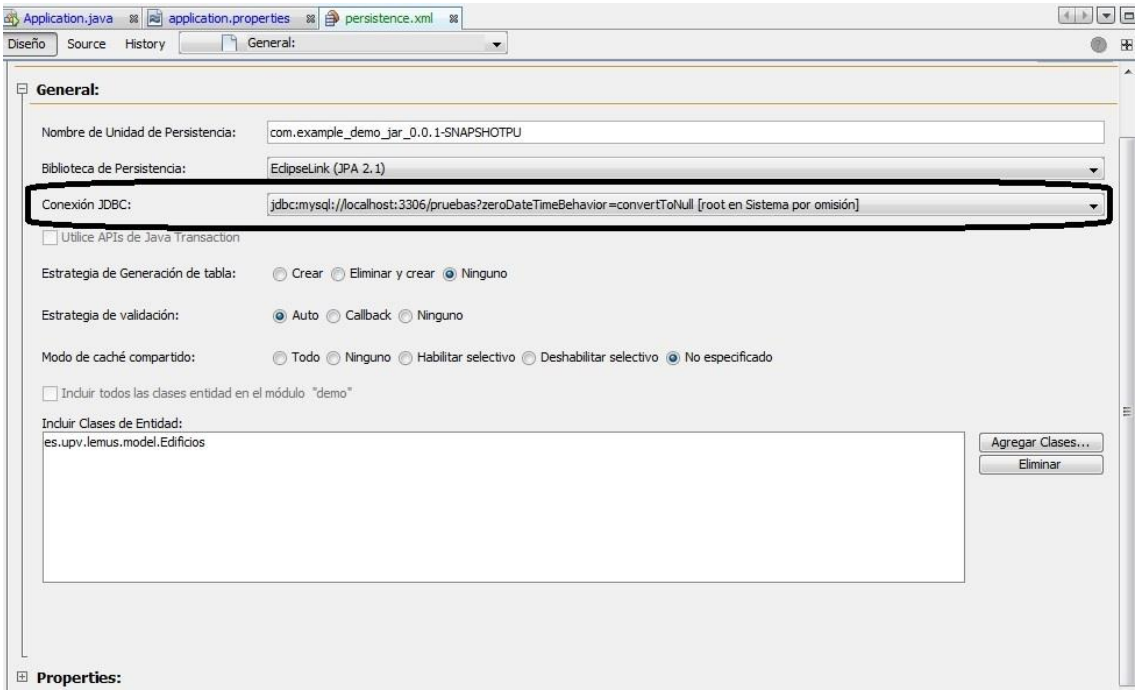
El primer archivo a destacar es el de "*application.properties*". En esta imagen se puede apreciar cómo está configurado dicho fichero. Este fichero sirve para poder conectarnos mediante ARC (*Advanced Rest Client*) a nuestra base de datos y poder consumir datos e información de todas las tablas que tengamos.

Se observa como se ha elegido como numero de puerto el ochenta ochenta-y-dos para poder lanzar el servidor. Luego se tiene la URL de nuestra base de datos, donde todas nuestras tablas están en la base de datos llamada "prueba". Los siguientes campos no hay que configurarlos de ninguna forma específica.



5.2.12.5.2 Persistence.xml

El segundo archivo que se destaca es "*persistence.xml*". Este archivo tiene una característica importante y que hay que recalcar para que nuestra aplicación funcione correctamente. Este fichero pertenece y hace referencia a la unidad de persistencia y como se puede observar en la captura de pantalla que hay en la siguiente página, para que la unidad de persistencia reconozca todas las operaciones, se tiene que definir en el apartado "Conexión JDBC" la conexión jdbc a nuestra base de datos.



5.2.12.5.3 Ventana.java

El tercer archivo a destacar es el de "ventana.java" donde se define la funcionalidad de mostrar los edificios en el desplegable de nuestra interfaz gráfica. Anteriormente ya se ha mostrado mediante capturas de pantalla como se muestra dicha lista a su vez que la propia interfaz. Ahora se va a mostrar el código implementado para llevar a cabo dicha funcionalidad.

```
void mostrarEdificio(){
    try {
        Connection cn = null;
        Class.forName("com.mysql.jdbc.Driver");
        cn = DriverManager.getConnection("jdbc:mysql://localhost:3306/lliria","root", "");
        Statement st = cn.createStatement();
        ResultSet rs = st.executeQuery("select EdificioMunicipal from edificios");
        comboEdificio.removeAllItems();
        while(rs.next()){
            comboEdificio.addItem(rs.getString(1));
        }
        rs.close();
        cn.close();

    } catch (ClassNotFoundException ex) {
        Logger.getLogger(ventana.class.getName()).log(Level.SEVERE, null, ex);
    } catch (SQLException ex) {
        Logger.getLogger(ventana.class.getName()).log(Level.SEVERE, null, ex);
    }
```

Se puede observar como lo primero que se hace es realizar una conexión a la base de datos, para ello se necesita el "*com.mysql.jdbc.Driver*". Luego tenemos que indicar la ruta de donde queremos conectarnos, en este caso será a la base de datos de Llíria. En dicha URL se especifica su dirección "*localhost*" además de su usuario y contraseña para poder acceder a ella. El siguiente paso y último es decirle de que tabla queremos obtener la información que se necesita, en este caso, se desea obtener la lista de edificios que tenemos, por lo tanto, se hace la consulta "*select EdificioMunicipal from edificios*".

Se cierra la conexión a la base de datos y ya está lista la función "*mostrarEdificio*". Como precaución se ha implementado el código dentro de la estructura "*try-catch*" para poder tratas las excepciones que se pudiesen dar. Luego en esta captura de pantalla se observa cómo se realiza la llamada al método creado anteriormente.

```
 */
public ventana() {
    initComponents();
    /**this.setLocationRelativeTo(null);*/
    setVisible(true);
    mostrarEdificio();
}
```

5.2.12.5.4 Creando Controladores

En este apartado se van a definir los ficheros controladores que se han creado en el proyecto. Como son todos iguales, pero cambiando las variables para que no den error y tengan conflicto, procederé a explicar uno de ellos y a decir donde cambian los unos de los otros. El archivo que se va a explicar es el de "*EdificioController.java*".

Estos archivos tienen que implementar las librerías de su clase java y de su repositorio, es decir tienen que estar interconectados los tres ficheros para que la coherencia resulte efectiva.

```
import es.upv.lemus.model.Edificios;
import es.upv.lemus.repository.IEdificioRepository;
```

Ahora voy a adjuntar unas capturas de pantalla del código implementado para que se pueda visualizar y a su vez poder explicarlo y marcar las diferencias que se tendrán en general todos los archivos entre sí.

```
@RestController
public class EdificioController {
    private static final String template = "Hello, %s!";
    private final AtomicLong counter = new AtomicLong();
    @Autowired
    private IEdificioRepository EdificioRepository;

    @PostMapping("/edificios/create")
    public Edificios edificios_add(@RequestBody Edificios m, HttpServletResponse response) {
        if(m.getCod() != -1) {
            response.setStatus(HttpServletResponse.SC_PRECONDITION_FAILED);
            return null;
        }
        Edificios m2 = null;
        try{
            m2 = EdificioRepository.save(m);
            response.setStatus(HttpServletResponse.SC_CREATED);
            System.out.println("==== in create edificio ====" + m.toString());
        }
        catch(Exception e){
            response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
        }
        return m2;
    }
```

```
@GetMapping("/edificios/list")
public ArrayList<Edificios> list_edificios(HttpServletResponse response) {
    ArrayList<Edificios> list = new ArrayList<Edificios>();
    Iterable<Edificios> iterable = null;
    Iterator<Edificios> iterador = null;
    try {
        iterable = EdificioRepository.findAll();
        iterador = iterable.iterator();
        while(iterador.hasNext()){
            list.add(iterador.next());
        }
        response.setStatus(HttpServletResponse.SC_OK);
    }
    catch(Exception e) {
        response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
    }
    return list;
}
```

```
@GetMapping("/edificios/read/{cod}")
public Edificios read_edificios(@PathVariable("cod") long cod, HttpServletResponse response) {
    Edificios m = null;
    try {
        m = EdificioRepository.findOne(cod);
        response.setStatus(HttpServletResponse.SC_OK);
        System.out.println("==== in read edificio ====" + cod);
    }
    catch(Exception e) {
        response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
    }
    return m;
}

@PostMapping("/edificios/update/{cod}")
public Edificios edificios_update(@PathVariable("cod") long cod, @RequestBody Edificios m, HttpServletResponse response) {
    if(m.getCod() == cod && EdificioRepository.exists(cod)) {
       Edificios m2 = null;
        try {
            m2 = EdificioRepository.save(m);
            response.setStatus(HttpServletResponse.SC_ACCEPTED);
            System.out.println("==== in update edificio ====" + m.toString());
        }
        catch(Exception e){
            response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
        }
        return m2;
    }
    response.setStatus(HttpServletResponse.SC_PRECONDITION_FAILED);
    return null;
}
```

```
    @PostMapping("/edificios/delete/{cod}")
     public Boolean delete_edificios(@PathVariable("cod") long cod, HttpServletResponse response) {
         try{
             EdificioRepository.delete(cod);
             response.setStatus(HttpServletResponse.SC_ACCEPTED);
             System.out.println("==== in delete edificio ====" + cod);
         }
         catch(Exception e){
response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
         }
         return true;
     }


}
```

En la primera captura se observa la definición de la clase controladora además de la definición del repositorio, al que hace mención a la hora de definirlo mediante "*private IEdificioRepository EdificioRepository;*". Toda la primera parte de código es la implementación para los servicios Web Rest *("@RestController*"). Unas líneas más abajo, aparece "*@PostMapping*" y se puede observar como el método creado se encarga de crear/añadir nuevos edificios a la base de datos. Hay que destacar de esta parte del código, la parte que hay justo al lado de "@PostMapping", es decir, *"/edificios/create*".

Esa línea irá cambiando en cada una de las clases que se ha creado en el proyecto, concretamente cambiará la palabra edificios, que es el nombre de una de las tablas de la base de datos, por otro nombre de otra tabla que se tenga en la misma.

Del mismo modo se tiene que cambiar el nombre "edificio" de todo el código por el nombre correspondiente a la tabla que se esté implementando en ese momento. Otro dato importante a tener en cuenta es la clave primaria de cada una de las tablas de la base de datos, debido a que cada tabla tiene una clave primaria diferente.

La tabla edificio tiene como clave primaria "*cod*", en la primera captura se puede apreciar en el primer método "*if*", más concretamente en "*if(m.get**Cod**() != -1)*". Dicha clave primaria también hay que ir cambiándola de una tabla a otra. Ahora dejo unos breves ejemplos para un mejor entendimiento de cada clase de cómo cambia su clave primaria y el nombre de la tabla en el "*@PostMapping*".

```
private IContratoRepository ContratoRepository; /**clase contrato*/
@PostMapping("/contrato/create") /** clase contrato*/
if(m.getContrato() != -1) /**clase contrato*/

private IDetalleContratoRepository DetalleContratoRepository;/**clase detalleContrato*/
@PostMapping("/detallecontrato/create")/**clase detalleContrato*/
if(m.getId() != -1)/**clase detalleContrato*/

 private IDimTiempoRepository DimTiempoRepository;/**clase dim_tiempo*/
@PostMapping("/dim_tiempo/create")/**clase dim_tiempo*/
 if(m.getFecha() != -1)/**clase dim_tiempo*/

 private IFacturaRepository FacturaRepository; /** clase factura*/
@PostMapping("/factura/create") /** clase factura*/
 if(m.getId() != -1) /** clase factura*/
```

Ahora pasamos a analizar la segunda captura. En ella se encuentra el "*@GetMapping*" y el método listar. Este método se encarga de listar la tabla seleccionada con todos sus campos de una forma ordenada y en forma de "*Array*". Los resultados se muestran en formato" JSON". Recalcar al igual que en el caso anterior, que hay que cambiar el nombre de la tabla de la base de datos cada vez que se haga un nuevo archivo. En este caso se está explicando el archivo que contiene la tabla edificios, por tanto, hay que cambiar dicho nombre de tabla cada vez que aparezca por el nombre de la tabla que se vaya a implementar en ese momento.

En la tercera captura de pantalla volvemos a ver que se hace referencia al método "*@GetMapping*" en la parte superior junto al método "*read_edificios*". Este método se encarga de leer cada uno de los edificios que se le vaya solicitando al sistema además de responder que ha tenido éxito o fracaso la operación. Para facilitar como sería dicho método con las demás tablas, adjunto una captura donde se puede ver de forma resumida cada uno de ellos.

```
  @GetMapping("/factura/read/{id}") /**clase factura*/
public Factura read_factura(@PathVariable("id") long id, HttpServletResponse response) /**clase factura*/
  Factura m = null; /**clase factura */

  @GetMapping("/dim_tiempo/read/{fecha}")/** clase dimtiempo*/
  public DimTiempo read_dimtiempo(@PathVariable("fecha") Date fecha, HttpServletResponse response){/** clase dimtiem
    DimTiempo m = null; /** clase dimtiempo*/

  @GetMapping("/detallecontrato/read/{id}") /** clase detallecontrato*/
public DetalleContrato read_detallecontrato(@PathVariable("id") long id, HttpServletResponse response){/** clase det
    DetalleContrato m = null; /** clase detallecontrato*/

@GetMapping("/contrato/read/{id}") /**clase contrato*/
public Contrato read_contrato(@PathVariable("contrato") long contrato, HttpServletResponse response){/**clase contra
    Contrato m = null; /**clase contrato*/
```

Se puede observar como para cada una de las clases va variando el nombre de la tabla además de la clave primaria en la parte del método constructor "*read_factura*". Luego tenemos en la segunda parte de la tercera captura, el "*@PostMapping*" junto con el método "*edificios_update*". Este método se encarga de poder actualizar la base de datos mediante el método POST en el ARC, pudiendo actualizar cualquier campo de la propia tabla y luego quedándose guardado. Al igual que en el caso anterior, voy a mostrar una captura donde se aprecien las diferentes tablas y los cambios que hay en cada una de ellas, que vienen a ser como he dicho antes, el nombre de la tabla de la base de datos y la clave primaria de la misma.

```
@PostMapping("/dim_tiempo/update/{fecha}")/** clase dimtiempo */
public DimTiempo dimtiempo_update(@PathVariable("fecha") Date fecha, @RequestBody DimTiempo m, HttpServletResponse
        if(m.getFecha() == fecha && DimTiempoRepository.exists(fecha)) {/** clase dimtiempo */
           DimTiempo m2 = null;/** clase dimtiempo */

@PostMapping("/detallecontrato/update/{id}")/**clase detallecontrato*/
public DetalleContrato detallecontrato_update(@PathVariable("id") long id, @RequestBody DetalleContrato m, HttpServ
        if(m.getId() == id && DetalleContratoRepository.exists(id)) {/**clase detallecontrato*/
           DetalleContrato m2 = null;    /**clase detallecontrato*/

  @PostMapping("/factura/update/{id}") /**clase factura*/
public Factura factura_update(@PathVariable("id") long id, @RequestBody Factura m, HttpServletResponse response) {
        if(m.getId() == id && FacturaRepository.exists(id)) {/**clase factura*/
           Factura m2 = null; /**clase factura*/

 @PostMapping("/contrato/update/{contrato}")/**clase contrato*/
 public Contrato contrato_update(@PathVariable("contrato") long contrato, @RequestBody Contrato m, HttpServletResp
        if(m.getContrato() == contrato && ContratoRepository.exists(contrato)) { /**clase contrato*/
           Contrato m2 = null;   /**clase contrato*/
```

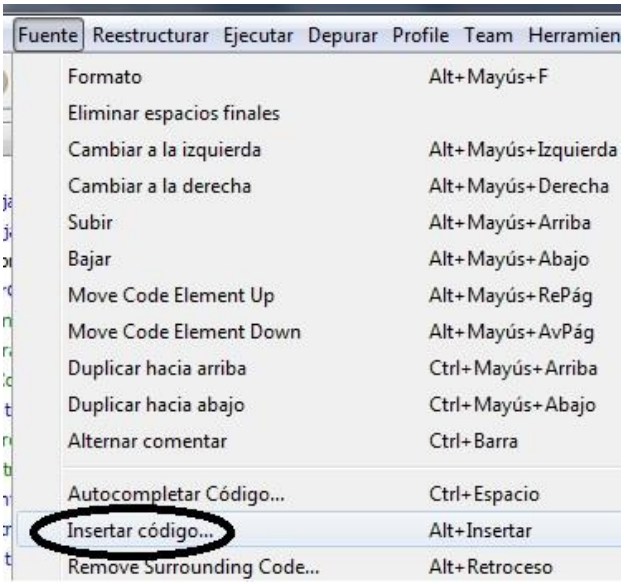
Y, por último, la cuarta captura. Se puede observar cómo se hace referencia otra vez al método "*@PostMapping*" además del método "*delete_edificios*". Este método permite poder eliminar cualquier campo de cualquier edificio además de poder eliminar el edificio entero y quitarlo de la base de datos. Como en la captura que se puede observar en la parte superior, el método "*delete*" es idéntico, pero solo teniendo como línea de código implementada la del método constructor "*public*". Como se ha dicho, para cada clase diferente, habrá que cambiar el nombre de la tabla y su clave primaria.

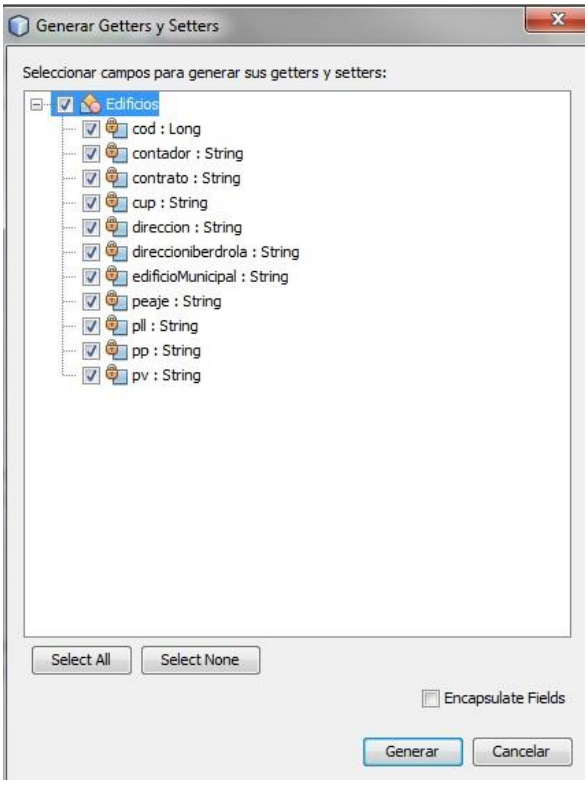
5.2.12.5.5 Creando Clases

En este apartado se van a definir los ficheros de las clases que se han creado en el proyecto. Como son todos iguales, pero cambiando las variables y los atributos para que no den error y tengan conflicto, procedo a explicar uno de ellos y a decir donde cambian los unos respecto de los otros. El archivo que se va a explicar es el de "*Edificio.java*". A continuación, adjunto cinco capturas de pantalla para proceder a su explicación.

```
@Entity
@Table(name = "edificios")
public class Edificios implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)


    private Long cod;

    @Column(name="edificiomunicipal")
    private String edificioMunicipal;
    private String direccion;
    private String cup;
    private String direccioniberdrola;
    private String contador;
    private String peaje;
    private String pp;
    private String pll;
    private String pv;
    private String contrato;
```

```
public Edificios(Long cod, String edificioMunicipal, String direccion,String cup,String direccioniberdrola, Str
    this.cod = cod;
    this.edificioMunicipal= edificioMunicipal;
    this.direccion=direccion;
    this.cup=cup;
    this.direccioniberdrola=direccioniberdrola;
    this.contador= contador;
    this.peaje= peaje;
    this.pp=pp;
    this.pll=pll;
    this.pv=pv;
    this.contrato= contrato;
}
```





```java
@Override
public int hashCode() {
    int hash = 0;
    hash += (cod != null ? cod.hashCode() : 0);
    return hash;
}

@Override
public boolean equals(Object object) {
    // TODO: Warning - this method won't work in the case the id fields are not set
    if (!(object instanceof Edificios)) {
        return false;
    }
    Edificios other = (Edificios) object;
    return !((this.cod == null && other.cod != null) || (this.cod != null && !this.cod.equals(other.cod)));
}

@Override
public String toString() {
    return "es.upv.lemus.model.Edificios[ cod=" + cod + " ]";
}
```

En la primera captura se puede observar cómo se define el nombre de la tabla "*@Table*", este valor va cambiando para cada una de las tablas que tenemos. Luego se puede observar cómo se define la propia clase, además, y esto es muy importante, de cómo implementa "*Serializable*". Por último, en la primera captura se tiene la definición del tipo y de los atributos de la tabla de la base de datos.

En la segunda captura tenemos la estructura del método constructor de una clase java. Se puede ver cómo se definen en el propio método sus atributos y luego en el cuerpo como se le asignan valores. Recalcar que cada una de las tablas de la base de datos tendrán unos atributos y un método constructor distintos.

En la tercera captura vemos como el propio "*Netbeans IDE*" nos da la opción de poder insertar código, opción de gran ayuda en estos casos. Y pasamos a la cuarta captura donde decidimos que código queremos insertar, en este caso nos interesa generar e insertar los métodos "*getters*" y "*setters*" para todos los campos. Este paso se repetirá idénticamente para cada clase de cada tabla que se cree.

Y por último, en la quinta captura podemos observar los métodos "*hasCode*()", "*equals*()" y "*toString*()". Estos tres métodos son generados de la misma manera que los "*getters*" y los "*setters*", mediante "Insertar código" y seleccionando estos mismos. Automáticamente los tenemos generados e implementados. Cada una de las clases como he dicho antes, tiene una clave primaria distinta, debido a que estos métodos tienen como atributo dicha clave primaria de la tabla de la base de datos.

5.2.12.5.6 Creando Repositorios

El sexto y último archivo que se va a explicar van a ser los repositorios de java que se han creado en el proyecto. Para cada una de las tablas de la base de datos se necesita un repositorio diferente en cada uno de los casos. Se va a proceder a la muestra del repositorio de la tabla "edificios" para explicarlo y explicar las diferencias que tendrán los unos de los otros.

Aquí tenemos la captura de pantalla referente a "*IEdificioRepository.java*" que muestra el repositorio creado para la clase Edificios. Destacar que cada repositorio tiene que implementar su propia clase java definida en el paquete "*es.upv.lemus.model*", como se hace en este caso con "*import es.upv.lemus.model.Edificios*". Cada repositorio que implementemos tendrá predefinidas las operaciones CRUD debido a que cada repositorio extiende de la clase "*CrudRepository*". La única diferencia que hay entre cada repositorio, es que en cada uno se hace referencia a la clase que se está implementando en dicho caso.

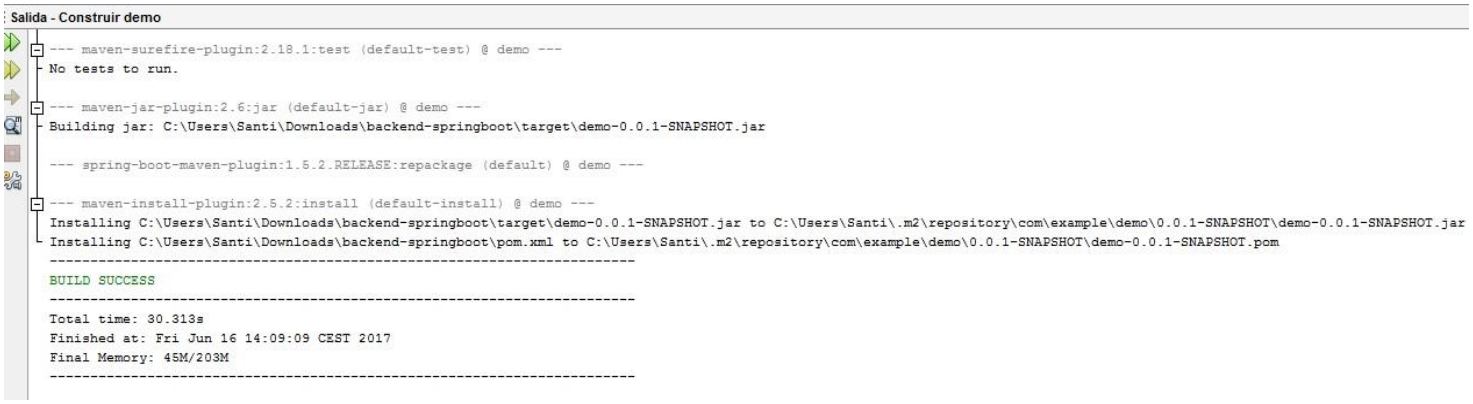
```
package es.upv.lemus.repository;

import es.upv.lemus.model.Edificios;
import org.springframework.data.repository.CrudRepository;

public interface IEdificioRepository extends CrudRepository<Edificios, Long> {

}
```

6. Pruebas y Validación

Una vez finalizado el desarrollo de la primera versión de la aplicación, se debe de proceder a llevar a cabo los procesos de pruebas y validación de la misma. En el caso de nuestra aplicación, la validación viene dada por la aprobación de un auditor energético de que la aplicación recoge los requisitos funcionales que al inicio del proyecto se propusieron.

Mientras que, en las pruebas, el primer paso que debemos de hacer es el de comprobar que no tenemos ningún error en nuestro código y podemos poner en marcha nuestra aplicación. Para ello procedemos a, como se puede ver en la captura de pantalla, "Limpiar y construir".


```
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/C:/Users/Santi/.m2/repository/ch/qos/logback/logback-classic/1.1.11/logback-classic-1.1.11.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/C:/Users/Santi/.m2/repository/org/apache/logging/log4j/log4j-slf4j-impl/2.7/log4j-slf4j-impl-2.7.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [ch.qos.logback.classic.util.ContextSelectorStaticBinder]

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.5.2.RELEASE)

2017-06-25 17:25:32.592  INFO 1492 --- [           main] es.upv.lemus.Application                 : Starting Application on Santi-PC with PID 1492 (C:\Users\Santi\Downloads\backend
2017-06-25 17:25:32.592  INFO 1492 --- [           main] es.upv.lemus.Application                 : No active profile set, falling back to default profiles: default
```



Una vez se selecciona “Limpiar y construir” da comienzo el proceso de carga y compilación de la aplicación, poniendo en marcha todos los paquetes, módulos y dependencias y comprobando que no haya ningún error. Como se puede observar en la segunda imagen, aparece la frase “*BUILD SUCCESS*”, que significa que la compilación de la aplicación ha sido un éxito y que no hay errores de implementación. Por tanto, las pruebas realizadas a la aplicación han sido satisfactorias debido a que no hay errores a la hora de haber realizado dicho test.

Ahora pasamos a la parte de la validación, que como he dicho antes, la validación de la aplicación viene dada por un auditor energético, ya que esta tiene que cumplir los requisitos funcionales que se propusieron al inicio del proyecto. Mediante las pruebas funcionales realizadas se valida el cumplimiento de los requisitos funcionales para detectar posibles errores producidos en la fase de desarrollo.

Este proceso es importante debido a que los errores detectados a tiempo durante la fase de pruebas ayudan a ahorrar tiempo y dinero. Para la realización de las pruebas, el auditor energético ha tenido acceso a la aplicación y a los servicios web creados, y ha podido comprobar que todos los datos y la información que solicitaba se le daba de forma correcta.

7. Conclusión

La idea de realizar este proyecto viene a razón de facilitar el trabajo a la hora de realizar auditorías energéticas a los auditores energéticos. Dichas auditorias son tareas costosas en lo referente a datos de consumo eléctrico y datos de la vivienda o edificio público en cuestión. Con esta aplicación se le ofrece al auditor energético la disponibilidad de poseer toda la información histórica sin necesidad de tener que llevar cientos de papeles y carpetas.

Debido a que realizar dichas auditorias es un proceso largo y costoso, el tener a su disposición toda la información de datos históricos tanto de consumo eléctrico como de facturación a un solo "*click*" es una gran ventaja. Además de que pueden comprobar mediante gráficos como va variando el consumo eléctrico en el tiempo que ellos precisen, ya sea en horas, días, semanas, meses o años.

Para el desarrollo de la herramienta se ha utilizado Netbeans IDE, plataforma que personalmente no había utilizado hasta dicho proyecto, y he de decir que es una plataforma muy completa y con la que se pueden desarrollar proyectos muy sofisticados además de que aprender a usarla es muy sencillo. Por otra parte, tenemos la extensión ARC de servicios Web REST de Google, que facilita el consumo de servicios web de nuestra base de datos mediante nuestras consultas para obtener la información que se necesita.

La herramienta desarrollada es una primera aproximación a la versión final. Será utilizada para terminar de validar la idea y decidir qué cambios y que mejoras se pueden hacer, sobre todo en tema de eficiencia y tiempo de respuesta al solicitar la información.

A lo largo del proyecto he tenido algunos problemas sobre todo a la hora de familiarizarme con SpagoBI, debido a que no había mucha información clara y concisa sobre la herramienta en Internet, hasta que encontré un libro y tuve que comprarlo para poder resolver las dudas y poder generar las primeras gráficas que no conseguía obtener. También a la hora de crear el primer controlador tuve ciertos problemas, pues no conseguía poder compilar la clase y tuve que pedir ayuda a mis tutores.

A nivel personal, el desarrollo de este trabajo de fin de grado ha sido un gran paso para mi formación, tanto académica como laboral. En la rama de especialización que cursé, dicha rama fue tecnologías de la información, no había trabajado ni estudiado a este nivel el tema de los servicios Web REST ni la gestión e integración de bases de datos.

En el trabajo se han tratado temas de diferentes áreas. En primer lugar, más orientado al sector empresarial, se trabaja con una idea y modelo de negocio que actualmente utiliza la OMIE. Seguidamente con el desarrollo e implementación de la aplicación se tocan campos y áreas relacionados con la ingeniería del "*software*" y por último con la configuración de los servicios Web REST se toca la parte de tecnologías de la información más concretamente el tema de Internet y Web.

El realizar un proyecto de este calibre ha supuesto el dedicarle una gran cantidad de tiempo a investigar sobre los diferentes temas a desarrollar y buscar y comparar mucha información para luego poder llevarla a la práctica.

La memoria se ha intentado que sea lo más clara y legible respecto a temas de implementación y desarrollo, debido a que puede servir de ayuda a gente que se encuentre en un futuro en mí misma situación.

El trabajo de final de grado ha sido una experiencia y una oportunidad que no he querido dejar escapar y con la que he podido expresar y mostrar mis conocimientos adquiridos durante estos cuatro años de grado poniéndolos en práctica en un caso real. Todo lo adquirido y aprendido durante este periodo de desarrollo del proyecto hace sentirme preparado para incorporarme el mundo laboral.

8. Bibliografía

1. Auditoria energética. Wikipedia [online] [Fecha de consulta:05-05-2017]. Disponible en: https://es.wikipedia.org/wiki/Auditor%C3%ADa_energ%C3%A9tica

2. UNE-EN 16247-1:2012 Auditorías energéticas. Parte 1: Requisitos Generales. AENOR[online] [Fecha de consulta:08-05-2017]. Disponible en: http://www.aenor.es/aenor/normas/buscadornormas/resultadobuscnormas.asp#.WVNg-ITyjIU

3. UNE-EN 16247-2:2014 Auditorías energéticas. Parte 2: Edificios. AENOR[online] [Fecha de consulta:08-05-2017]. Disponible en: http://www.aenor.es/aenor/normas/buscadornormas/resultadobuscnormas.asp#.WVNivITyjIU

4. UNE-EN 16247-3:2014 Auditorías energéticas. Parte 3: Procesos. AENOR[online] [Fecha de consulta:08-05-2017]. Disponible en: http://www.aenor.es/aenor/normas/buscadornormas/resultadobuscnormas.asp#.WVNjWoTyjIU

5. UNE-EN 16247-4:2014 Auditorías energéticas. Parte 4: Transporte. AENOR[online] [Fecha de consulta:08-05-2017]. Disponible en: http://www.aenor.es/aenor/normas/buscadornormas/resultadobuscnormas.asp#.WVOPg4TyjIU

6. UNE-EN 16247-5:2015 Auditorías energéticas. Parte 5: Competencia de los auditores energéticos. AENOR[online] [Fecha de consulta:08-05-2017]. Disponible en: http://www.aenor.es/aenor/normas/buscadornormas/resultadobuscnormas.asp#.WVOP1oTyjIU

7. Guía TFG Sergio Benavent. Sergio Benavent Mascarell Repositorio Riunet UPV[online] [Fecha de consulta:09-05-2017]. Disponible en: https://riunet.upv.es/handle/10251/54038

8. IEEE. IEEE[online] [Fecha de consulta:12-05-2017]. Disponible en: https://www.ieee.org/index.html

9. Especificación de requisitos según el estándar de IEEE 830. Universidad complutense de informática[online] [Fecha de consulta:15-05-2017]. Disponible en: https://www.fdi.ucm.es/profesor/gmendez/docs/is0809/ieee830.pdf

10. Lenguaje UML. UML[online] [Fecha de consulta:16-05-2017]. Disponible en: http://www.uml.org/

11. Casos de uso. Wikipedia[online] [Fecha de consulta:18-05-2017]. Disponible en: https://es.wikipedia.org/wiki/Diagrama_de_casos_de_uso

12. Diagrama entidad relación. Wikipedia[online] [Fecha de consulta:22-05-2017]. Disponible en: https://es.wikipedia.org/wiki/Modelo_entidad-relaci%C3%B3n

13. Diagramas de actividad. Computadores y Tiempo Real[online] [Fecha de consulta:24-05-2017]. Disponible en: http://www.ctr.unican.es/asignaturas/procodis_3_ii/doc/statediagram.pdf

14. Arquitectura Modelo-Vista-Controlador. Wikipedia[online] [Fecha de consulta:29-05-2017]. Disponible en: https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador

15. Xampp. Wikipedia[online] [Fecha de consulta:31-05-2017]. Disponible en: https://en.wikipedia.org/wiki/XAMPP

16. MariaDB. Wikipedia[online] [Fecha de consulta:31-05-2017]. Disponible en: https://en.wikipedia.org/wiki/MariaDB

17. SQL. Wikipedia[online] [Fecha de consulta:31-05-2017]. Disponible en: https://es.wikipedia.org/wiki/SQL

18. Python. Python Software Foundation[online] [Fecha de consulta:02-06-2017]. Disponible en: https://docs.python.org/3/tutorial/index.html

19. phpMyAdmin. Wikipedia[online] [Fecha de consulta:04-06-2017]. Disponible en: https://es.wikipedia.org/wiki/PhpMyAdmin

20. Spoon. Pentaho Data Integration(Spanish) [online] [Fecha de consulta:08-06-2017]. Disponible en: http://wiki.pentaho.com/display/EAIes/Manual+del+Usuario+de+Spoon

21. Java (lenguaje de programación). Wikipedia[online] [Fecha de consulta:12-06-2017]. Disponible en: https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)

22. XML. Manual de XML[online] [Fecha de consulta:14-06-2017]. Disponible en: http://www.mundolinux.info/que-es-xml.htm

23. Servicios Web RestFul. RNM ELP-DSIC-UPV[online] [Fecha de consulta:15-06-2017]. Disponible en: http://users.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf

24. Servicios Web RestFul. Asociación Desarrolladores Web de España[online] [Fecha de consulta:21-06-2017]. Disponible en: http://www.adwe.es/general/colaboraciones/servicios-web-restful-con-http-parte-i-introduccion-y-bases-teoricas

25. ARC. Hubworks[online] [Fecha de consulta:22-06-2017]. Disponible en: http://api.hubworks.com/wp-content/themes/hub-dev-portal/docs/Rest-Client-Installation.pdf

26. SpagoBI. Wikipedia[online] [Fecha de consulta:23-06-2017]. Disponible en: https://es.wikipedia.org/wiki/SpagoBI

27. SpagoBI Suite. Basic Reporting & Data Exploration Tools[online] [Fecha de consulta:26-06-2017]. Disponible en: https://www.kobo.com/es/es/ebook/basic-reporting-data-exploration-tools

28. SpagoBI Inicio Rápido. OW2 Forge[online] [Fecha de consulta:27-06-2017]. Disponible en: http://download.forge.ow2.org/spagobi/SpagoBI_4.x_quick_start_ES.pdf

29. Learn SpagoBI in two hours. Ogutu[online] [Fecha de consulta:27-06-2017]. Disponible en: http://ogutu.org/books/Learn-SpagoBI-in-Two-Hours.pdf

30. SpagoBI Data Source. Wiki SpagoBI[online] [Fecha de consulta:27-06-2017]. Disponible en: http://wiki.spagobi.org/xwiki/bin/view/spagobi_server/Data+Source

31. SpagoBI Data Set. Wiki SpagoBI[online] [Fecha de consulta:28-06-2017]. Disponible en: http://wiki.spagobi.org/xwiki/bin/view/spagobi_server/data_set

32. Netbeans IDE. Netbeans org[online] [Fecha de consulta:09-06-2017]. Disponible en: https://netbeans.org/index_es.html

33. Instalación Netbeans IDE. Universidad Complutense Informática[online] [Fecha de consulta:09-06-2017]. Disponible en: https://www.fdi.ucm.es/profesor/luis/fp/devtools/NetBeans.html

34. Instalación Netbeans IDE. SambayRedes[online] [Fecha de consulta:09-06-2017]. Disponible en: http://jdeveloper.wikispaces.com/06.1.-+Instalaci%C3%B3n+de+NetBeans+7.2

35. Maven. Wikipedia[online] [Fecha de consulta:29-06-2017]. Disponible en: https://es.wikipedia.org/wiki/Maven

36. Introduction to Spring Web MVC. Netbeans org[online] [Fecha de consulta:30-06-2017]. Disponible en: https://netbeans.org/kb/docs/web/quickstart-webapps-spring.html

37. Iberdrola Clientes. Iberdrola S.A[online] [Fecha de consulta:03-05-2017]. Disponible en: https://www.iberdrola.es/webclifr/#/login

38. Iberdrola Distribución. Iberdrola Distribución Eléctrica[online] [Fecha de consulta:04-05-2017]. Disponible en: https://www.iberdroladistribucionelectrica.com/consumidores/inicio.html#login