Apificación de recursos CSV

PAC - 3

Autor	Fecha
Alex Lleida	29/11/2016

Contenido

1 Abstract....3
2 Introducción....5
2.1 Estado del arte....5
2.1.1 Introducción....5
2.1.2 Desarrollo....8
2.1.3 Conclusiones....14
2.2 Introducción al proyecto....15
2.3 Movimiento Opendata....16
2.4 Objetivos del proyecto....17
2.5 REST....17
3 Método....21
3.1 Arquitectura y herramientas utilizadas....21
3.2 Detalle del proyecto....23
3.3 Clientes....28
4 Conclusiones del trabajo....32
5 Bibliografía....34

1 APIFICACIÓN DE RECURSOS CSV

1 Abstract

Las hojas de cálculo han pasado a ser una herramienta de gestión muy común en las organizaciones. La potencia y sencillez de este tipo de aplicaciones han hecho que cualquier usuario pueda manejarlas pero, al mismo tiempo, han generado un problema a causa de los errores estructurales y de los propios datos, el cual genera una pérdida de consistencia en la información y multitud de errores. El objetivo de este artículo es demostrar cómo utilizando una API REST se puede controlar la información contenida en ficheros CSV para dar servicios a cualquier tipo de aplicación, suministrando y gestionando la información de forma estandarizada para evitar errores en la manipulación. Los usuarios no tienen, de este modo, un acceso directo al fichero CSV sino que lo hacen mediante la API REST correspondiente. Las API REST puede ser un complemento para acceder y trabajar los documentos publicados bajo la filosofía del movimiento Opendata que pone a disposición de todo el mundo información útil sin restricciones de uso.
Palabras clave: Opendata, API Rest, CSV, Apificación, UML.

Spreadsheets have become a very common management tool in organizations. The power and the simplicity of this kind of applications has made that any user can handle them but, at the same time, a problem has been generated due to the structural errors and the data. That causes information inconsistency and can have multiple errors. The purpose of this article is to demonstrate how using a REST API can control the information of CSV files to provide services to any type of application by providing and managing information in a controlled way to avoid errors in handling. Users won't have direct access to the CSV file, but they will do so using the corresponding REST API. These are a complement to access and work published documents under the philosophy of the Opendata movement that puts everyone useful information without restrictions of use.
Keywords: Opendata, API Rest, CSV, API-fication, UML.

2 Introducción

2.1 Estado del arte

2.1.1 Introducción

La generación y edición de hojas de cálculos ha proliferado mucho y se ha hecho muy popular. Se ha convertido en un estándar para llevar a cabo y mantener bases de datos. Todo debido a la facilidad de introducir los datos y de visualizarlo .Esto ha llevado a construir bases de datos sin ningún esquema aparente por usuarios sin conocimiento en normalización y modelización de los esquemas. Las estructuras de las hojas de cálculo, van creciendo conforme las necesidades de los usuarios así lo requieren. Esto hace que la hoja de cálculo se pueda volver inconsistente en cualquier momento ya sea por añadir una nueva columna o por rehacer una formula. Existen diferentes errores que podemos atribuir a los usuarios. El primero es meter la información incorrecta ya sea porque el tipo o formato no es el correcto (por ejemplo meter un texto en una columna donde se espera un campo numérico). Otro tipo de error es el de modificar la estructura de la hoja de cálculo sin tener en cuenta las consecuencias de este. Al añadir una fila o columna podemos estar modificando la estructura de esta hoja y dejando columnas sin sus correspondientes formulas o variar las formulas existente en toda la hoja u hojas relacionadas haciendo que los cálculos sean erróneos. Una simple corta y pegar puede hacer que toda una hoja pierda la consistencia y los datos calculados y mostrados sean erróneos. Un ejemplo de este caso lo tenemos en la empresa TransAlta Corp. [4] que dio unos beneficios incorrectos por culpa de haber modificado sin control una hoja de cálculo.

El objetivo de los estudios analizados busca dar una coherencia y control a las hojas de cálculo para poder realizar cambios posteriores de datos sin que esto afecte a la información ya insertada o a los cálculos que se tiene con esta información. Otro objetivo consiste en controlar la calidad de los datos introducidos evitando errores que puedan afectar al funcionamiento general de la hoja de cálculo. Se busca crear verdaderas aplicaciones de hojas de cálculo que sirvan a las organizaciones para sus fines concretos con la seguridad de que la información está preservada de forma correcta y los usuarios puedan modificarla de forma controlada y segura.

El denominador común de estos artículos es utilizar el modelado UML para estandarizar la información y las relaciones entre los diferentes objetos que pueda haber en la hoja. La forma de conseguirlo es diferente entre ellos, algunos utilizan extensiones de Excel para analizar la información de la hoja de cálculo y extraer la información. Otros se basan en herramientas externas que gracias a patrones creados con anterioridad permite obtener el modelo conceptual de una forma más rápida.

En este estudio se trabajan hojas de cálculos con estructura de calculo que pueden incluir formulas y que la disposición de la información es importante. En estos artículos dejan de lado la conceptualización de modelos de hojas de cálculo más orientadas al almacenamiento de información como si de base de datos se tratase, aunque se puede extrapolar las técnicas a este tipo de hojas de cálculo. Estos últimos ficheros suelen almacenarse en formatos no binarios como el CSV separado por comas y no contiene fórmulas de cálculo. No obstante se puede utilizar el mismo proceso de conceptualización.

Tal como indica [1] existen limitaciones en las ClassSheets. No se puede detectar errores comunes en la entrada de información. Sí que es verdad que con las reglas OCL podemos restringir el rango de datos a insertar, no podemos regular toda la lógica de negocio y podemos tener errores que no se pueden verificar a no ser que se hiciese una comparación y estadística de valores. Por ejemplo se podría indicar una cifra incorrecta de ventas pero el sistema siempre que esté en unos rangos lógicos la dejaría pasar. Otro de las limitaciones es la capacidad de conectar el modelado de ClassSheets con otras herramientas de desarrollo. Como veremos más adelante, la definición UML y las restricciones OCL la podemos extrapolar al desarrollo a través de las clases p.e. definiendo estas clases con Java. Tampoco se puede relacionar ClassSheets entre ellas sin alguna herramienta que pueda controlar esta estructura.

El modelo formal de ClassSheets tal como resumen bien [4] define a todos los elementos que pueden contener una hoja del cálculo. Las celdas pueden ser varios tipos. [3] identifica los siguientes tipos: encabezado, pie, datos y relleno. En cambio [4] indica que los tipos de celdas pueden ser: celdas vacías, formulas y constantes. Si vemos los ejemplos presentados en la mayoría de los casos vemos que el encabezado suelen ser valores constantes (muchas veces de tipo texto), los datos también son constantes pero de tipo numérico, el pie suelen ser las formulas finales que totalizan y por último las celdas vacías y de relleno son lo mismo. Por lo tanto, ambas tipologías coinciden bastante.

Por otra parte nos encontramos que muchas organizaciones están publicando información útil para el público en general bajo el movimiento Opendata. Este es una filosofía de compartir información sin ningún tipo de limitación. Bajo la práctica del copyleft permite a los usuarios que se descargan información usarla esta sin ningún tipo de limitación, redistribuir las copias y modificarla si procede.

También nos encontramos una tecnología, RESTful, que en los últimos años se ha convertido en un estándar para el acceso y manipulación de la información a través de aplicaciones clientes. Esta comunicación se realiza utilizando protocolos ampliamente utilizados en internet como son el http y con formatos de transferencia de la información universales y estandarizados como XML y JSON.

El proyecto busca la aplicación de los 3 elementos analizados:

- Ficheros facilitados bajo el movimiento Opendata, Concretamente los CSV.
- Modelización de estructura de la información del fichero (una parte de los ClassSheets).
- Creación de API REST a partir de los ficheros seleccionados.

Aplicando estos elementos se busca crear APIs que gestionen la información contenida en los CSV y permita a las aplicaciones clientes acceder de forma controlada a la información.

2.1.2 Desarrollo

Todas las técnicas pasan por una serie de fases:

2.1.2.1Análisis y esquematización de la hoja de cálculo del problema.

Cualquier hoja de cálculo tiene una estructura que busca un objetivo concreto a partir de una lógica de negocio. Primero de todo se debe analizarla estructura y el tipo de información de cada celda. Es importante conocer la lógica y el objetivo que hay por detrás para analizar bien la conceptualización del modelo. Aunque las posibilidades de modelos son infinitas, normalmente un porcentaje muy alto de ficheros tienen esquemas comunes en la hoja de cálculo. Esto hace más fácil los procesos de conceptualización ya que permite definir plantillas con modelos.

El objetivo es definir una estructura coherente con los objetivos del negocio y que este normalizada. Contra más definida este la hoja de cálculo más fácil serán los pasos posteriores.

En todos los artículos leídos trabajan con modelos en dos dimensiones, que es lo más típico en hoja de cálculo. Las hojas de cálculos suelen tener varias entidades que se acaban combinando en diferentes niveles de filas y columnas dentro de una misma hoja. El primer paso consiste en definir las entidades que intervienen junto con sus claves principales y buscar las dependencias funcionales. Las dependencias funcionales no son más de las claves foráneas de cada entidad respecto a otra entidad de la misma hoja. Una vez se tiene esta información se analiza si la estructura de la información y las formulas son coherentes.

2.1.2.2Modelización de la estructuras

Se pueden proponer mejoras a la estructura para dejar esta mejor normalizada. Se puede construir un modelo conceptual con UML a partir de la estructura definida.

UML en un lenguaje de modelado que nos permite describir métodos y procesos a través de gráficos. En estos gráficos podemos informar de las relaciones entre diferentes objetos. El objetivo es obtener un modelo de sistema que después nos permitirá extrapolarlo a otros sistemas o herramientas para seguir trabajando. El objetivo aparte de la generación de este modelo es la estandarización de los procesos de modelado.

A partir de las entidades participantes se busca representar la lógica de negocio en un diagrama de clases. En esta modelización estamos representando gráficamente todas las dependencias funcionales que vimos en el punto anterior. Cada hoja tendrá un patrón definido que nos ayudará a crear las clases. Estas pueden venir identificadas por las filas o las columnas de una misma hoja de cálculo. No obstante puede haber otras clases que no se identifiquen a simple vista. Este es el caso de relaciones n..n de clases que se convierten en otra clase más. Por ejemplo [4] propone el caso de ventas de productos por países, aquí aparecen 2 clases derivadas, una la de las ventas por país, que es la información cruzada entre productos por país. Una segunda clase será la del total de ventas. En este caso la clase sale a partir de las fórmulas de los totales que tiene la hoja e calculo. La primera clase derivada la ubicamos en las celdas centrales que unen los productos y los países. En cambio la segunda sale de los extremos opuestos a los literales con la suma de filas y columnas de un determinado país o producto.

En [3] nos presentan la herramienta Gyro que es capaz de analizar las hojas de cálculo y obtener los diagramas de clases de forma automática. Este utiliza patrones para obtener su fin. Realmente va comparando la hoja de cálculos con cada uno de los patrones y cuando da con el correcto extrae el modelo conceptual predefinido así como las clases detectadas. En este caso en [3] se hizo un estudio utilizando unas 4000 hojas de cálculo extraídas del repositorio EUSES. El resultado fue que el 40% de las hojas de cálculo fueron identificadas a partir de los patrones. Para identificar los patrones [3] utiliza gramáticas de patrones con determinadas reglas de producción. El reconocimiento el patrón se basa en una seria de pasos. Primero identifica el rectángulo con información dentro de la hoja de cálculo. Aquí se encontraran los diferentes tipo de información como datos, formulas o etiqueta. Aquí se buscan los extremos y las formulas.

2.1.2.3Definición de reglas de trabajo

Una vez definida la estructura se puede definir todas las reglas. En este apartado se definen las operaciones que se pueden realizar sobre una celda (baldosa) o rango de celdas (embaldosados) [4]. Aquí se declaran hacia sonde puede crecer un rango de celdas, si es horizontal o vertical y en qué sentido se deja crecer. A esto [4] lo llama alicatar, como si fuesen las racholas de un baño. Este aumento de celdas controlado permite que las fórmulas que pueda haber, normalmente al final, puedan adaptarse a las variaciones sufridas y se mantenga la coherencia con todo el sistema. Esta definición de las reglas se puede hacer de varias formas:

Utilizando herramientas visuales [2],[4] como Claos que permite definir el entorno de trabajo de una forma gráfica y sencilla. De hecho, el entorno es muy parecido a la misma hoja de cálculos final por lo que ayuda a tener una contextualización mayor.

Definir plantillas o patrones estándares de hojas de cálculo [3] y se mira si una hoja determinada cumple con una plantillas preestablecida. Si es así se heredan todas las reglas. Esto permite una disminución en el tiempo de modelado. Aunque las posibilidades de combinación son infinitas, casi todas las hojas de cálculo siguen unos patrones similares

A la hora de definir las reglas de alicatado o de crecimiento de los rangos [4], define tres tipos de alicatados:

- Alicatado horizontal: las clases son horizontales y van de lado a lado
- Alicatado vertical: las clases son verticales y van creciendo de la misma forma
- Alicatado tabla: las estructuras son rectangulares y evolucionan en ambas direcciones

Claos implementa las reglas de encadenado para asegurarse de que las ClassSheets estén bien formadas.

Normalmente las hojas de cálculo utilizan las dos dimensiones para jugar con dos clases. Para ver la relación una clase la ponemos en forma vertical y la otra la ponemos horizontal. El producto de las 2 nos da las relaciones entre las dos clases. En crecimiento (alicatado) de estas clases nos lo facilitará la sintaxis del modelo formal de ClassSheets.

Las reglas o restricciones de la hoja de cálculo se documentan en el lenguaje OCL (Object Constraint Languaje o lenguaje de restricción de objetos). Se trata de un lenguaje que se encuentra dentro de la especificación de UML. Este lenguaje permite definir restricciones y consultas a los modelos de datos relacionados a través de reglas con una sintaxis propia. Es un complemento a los diagramas UML y su objetivo es ampliar su semántica. No cambia la información de los objetos sino que impone reglas a cumplir por parte de otras herramientas que si que pueden modificar esta información.

2.1.2.4Exportación de estas reglas a herramientas externas.

Una vez tenemos el modelo definido y todas las reglas bien documentadas es momento de ponerlo en práctica. Aquí nos ofrecen varias posibilidades entre las cuales se destacan
Recoger el modelo conceptual y crear las clases necesarias para poder desarrollar una aplicación que consulte o gestione la información de la hoja de cálculo. En este caso se pone la modelización al servicio de la programación orientada a objetos para controlar el acceso a la información siguiendo las reglas pactada.

Por otra parte se puede traspasar el esquema de modelado así como sus restricciones a Excel. Aunque directamente esta aplicación no dispone de funcionalidad para administrar esa información, existen complementos de Excel que nos puede ayudar para que la hoja de cálculo siga las normas establecidas en casos anteriores.

Las especificaciones de estas reglas son enviadas a herramientas o add-ons de Excel como Gencel a través de un lenguaje visual llamada ViTSL. Este proporciona un método para ayudar a la modelización de una hoja de cálculo indicando como puede ser modificada su estructura. En el este lenguaje se definen por ejemplo los proceso de alicadas que se han trabajado en Claos. Estas reglas con ViTSL se envían a Gencel que las sabe interpretar y controlar en cualquier hoja de cálculo que cumpla con esta plantilla. VITSL puede ser utilizado como plantillas para diferentes hojas de cálculo ya que al ser unas especificaciones se abstrae del contenido de este y como hemos hecho con Giro más atrás podemos reutilizar esta especificación para validar y controlar diferentes hojas de cálculo

El objetivo de Gencel es servir como un sistema de calidad de la hoja de cálculo. Las reglas han sido establecidas a priori y a los usuarios se les deja un margen amplio de libertad para que modifique la hoja de cálculo a su gusto. En el momento que realicen una acción que no se contemple en las reglas importadas a través de la especificación viTSL, los sistemas no dejará seguir con la operación. En el momento que el usuario realiza la operación, si esta es permitida, Gencel creará las formulas necesarias en las celdas variadas para conservar esta integridad.

Otra posibilidad es utilizar la modelización para crear servicios web que trabajen de forma controlada con la información incluida en la hoja de cálculo. Este es el objeto de mi estudio. La idea es, a partir del modelo definido, crear una serie de servicios web REST que realicen todas las operaciones posibles con una hoja de cálculo. En este caso la hoja de cálculo tiene más el aspecto de una base de datos, sin formulas. Las operaciones que se podrán hacer son las básicas que se hacen con una tablas (llamadas CRUD) Create, Read, Update y Delete. Todas estas operaciones se hacen de forma controlada por lo que en ningún momento se pone en peligro la integridad de la hoja de cálculo. Otro beneficio de este desarrollo es que se pueden utilizar los servicios REST para comunicar diferentes hojas de cálculo modelizadas con ClassSheets e integrarlas en una sola aplicación.

En el caso concreto del proyecto que nos lleva las fases del desarrollo sería:

- Seleccionar un fichero a trabajar. Buscar archivos candidatos a partir de los repositorios Opendata de las administraciones públicas.
- Realizar la modelización de su estructura. Crear el esquema UML de las clases que componen los ficheros seleccionados.
- Crear la API REST de gestión de la información a partir de la modelización. Crear las clases del fichero así como todas las funciones de servicios web que nos permitan ejecutar las tareas CRUD de gestión de fichero. Estas principalmente serían: Consulta de información (puede haber varias formas según el contexto del fichero), Creación de registros, Actualización y Borrado de estos.

2.1.3 Conclusiones.

ClassSheets es un modelo que permite dotar a las hojas de cálculo de una integridad para garantizar la calidad de la información. Para lograr este objetivo se debe realizar un trabajo previo de análisis y modelización de la estructura para poderle aplicar las reglas necesarias para acotar los posibles errores. Todos los trabajo intentar desarrollar técnicas basadas en la sintaxis del modelo ClassSheets para lograr la integridad de la información. En algunos casos el medio para llegar a ellos varía en función de la automatización de cada uno de los procesos que intervienen en este proyecto. En algunos casos se han realizado programas especializados en algunas de las partes o complementos a herramientas estándares como puede ser Gencel en MS Excel. También se ha buscado la estandarización de procesos a través de patrones de funcionamiento.

El proceso de modelización con ClassSheets no es difícil pero tampoco es para usuarios básicos. No obstante, el resultado de este proceso de modelización abre la puerta a que cualquier usuario pueda utilizar una hoja de cálculo haciendo cualquier tipo de operación sin que este archivo pierda la integridad en la información y en lo cálculos resultantes.

El objetivo de todos los trabajos mostrados es evitar los errores que se puede producir en las hojas de cálculo. Se trata de una herramienta cada vez más utilizada no solo para gestionar información básica. Algunas veces la criticidad de la información que se maneja puede hacer necesario el tomar medidas para paliar cualquier pérdida de información por un mal uso o lo que es peor, no poder garantizar que los datos que hay en esta hoja de cálculo están bien cálculos. De estos datos pueden depender decisiones estratégicas de la organización. La pérdida de integridad de una hoja de cálculo importante puede causar una pérdida considerable de recursos en una organización ya sean en tiempo de sus trabajadores en investigar la causa del error o ya sea en dinero por haber tomado decisiones erróneas basadas en datos calculados de forma incorrecta. En ambos casos, la empresa tiene que tomar medidas y creo que la modelización con ClassSheets puede ayudar a la organización a ganar calidad en sus procesos informáticos y de control.

En mi proyecto el objetivo es el mismo aunque en vez de controlar el contenido desde el mismo fichero de hoja de cálculo, como hace ClassSheets, yo busco el control de este contenido a través de la gestión que lleva a cabo el API REST creado a tal efecto. Esto además del control nos permite la compartición de la información de la API entre varias aplicaciones e incluso entre otras API REST.

2.2 Introducción al proyecto

En este proyecto he utilizado un tipo de hoja de cálculo especial, un formato abierto CSV (comma-separated values). En esta tipología la información viene dada en formato texto, con los campos separados normalmente por coma o punto y coma. Los registros se separan entre ellos con saltos de línea y contiene una primera línea de cabecera. Si un campo es de texto, podemos delimitarlo con comillas dobles para que la información contenida se pueda mantener sin cambiar la estructura. Por ejemplo, que dentro del campo los datos puedan incluir espacios, comas o puntos y coma.

La estructura es mucho más sencilla que la que pueda tener una hoja de cálculo. La información está organizada por filas y columnas. Las filas son los propios registros (salvo la primera que se destina a la cabecera) y la columnas son los campos (la primera fila de cada columna nos indica el nombre del campo).

Las aplicaciones principales de un formato CSV son la importación y exportación de la información y es debido, principalmente, a su simplicidad y semejanza con la estructura de una tabla de base de datos.

Muchos de los ficheros CSV son publicados por administraciones públicas dentro del movimiento Opendata.

2.3 Movimiento Opendata

Cada vez más, las organizaciones tanto públicas como privadas están poniendo a disposición de los usuarios información útil en formatos estándares. La mayoría de esta información proviene del sector público, ya sean gobiernos centrales, regionales o locales. El movimiento Opendata [1] indica en su propia definición que los ficheros deben tener las siguientes características:

- Los datos deben estar disponibles preferentemente en internet y sin ningún coste.
- Se debe permitir la reutilización, la redistribución y la mezcla de los datos.
- Deben ser publicados en un formato abierto que permita la manipulación.
- No debe haber ningún tipo de restricción ni por el motivo del acceso (comercial o no), ni a personas o grupos.

2.4 Objetivos del proyecto

El objetivo de este proyecto es demostrar la aplicación de aquellos datos publicados bajo el movimiento Opendata y cómo pueden ser aplicados a desarrollos para trabajar la información utilizando como puente de conexión entre el usuario y el CSV una API REST. Aunque casi la totalidad de los ficheros están publicados exclusivamente en modo lectura, también cabría la posibilidad de un mantenimiento de la información en BackOffice gestionado por parte del publicador.

En este proyecto he propuesto dos ejemplos de aplicaciones. En primer lugar, un servicio de gestión de datos llamado CRUD, el cual permite crear, actualizar, borrar y leer información contenida en un fichero CSV a través de llamadas a servicios web REST. En segundo lugar, he creado un ejemplo de aplicación cliente que utiliza tres fuentes de información CSV diferentes. En este caso, únicamente lee y no permite modificar la información contenida.

La información ha sido extraída del catálogo de datos abiertos del Ajuntament de Barcelona . En uno de los casos se ha utilizado el mismo fichero sin modificación alguna. En el otro, dos ficheros auxiliares se han modificado para adaptarlos al fin del proyecto.

2.5 REST

Administrar los ficheros CSV se puede conseguir construyendo un API que permita hacer las operaciones más comunes y que sea fácil de acceder por parte de los clientes, aunque de una forma controlada. Para ello utilizaremos los servicios REST.

Se trata de funciones y métodos accesibles a través del protocolo HTTP por direcciones únicas URI (Uniformed Resource Identifier) que dan acceso a recursos. Se utilizan algunos de los verbos estándares del protocolo HTTP para diferenciar el tipo de acción que se quiere llevar a cabo con los recursos. Por ejemplo, GET para consulta, POST para insertar, PUT para actualizar y DELETE para eliminar.

Cada recurso puede ser representado de varias formas aunque normalmente utilizaremos los formatos JSON o XML. Esto ayuda a la interpretación tanto desde el servidor como desde el cliente, dado que son formatos abiertos. Este punto es clave para poder abstraer los sistemas. Ya no necesitamos tener sistemas idénticos en ambos lados, cliente y servidor, para poder trabajar juntos.

El éxito de REST viene dado porque facilita la comunicación al utilizar formatos abiertos y estructurados para el diálogo entre cliente y servidor. Si hacemos un paralelismo con los elementos que intervienen en la comunicación, tal y como muestra la figura 1, podemos ver que:

- Emisor: es el cliente que tiene una necesidad y debe acceder a los recursos de un servidor para poder cumplir sus objetivos. Las peticiones siempre se inician en un cliente.
- Receptor: es el servidor con toda una serie de recursos publicados. Este queda a la espera de las llamadas de los diferentes emisores o clientes.

- Canal: el medio de comunicación por excelencia es internet o una simple red local. En definitiva, un sistema abierto sin restricciones. Pueden existir restricciones pero éstas son exclusivas del servidor. Tanto emisor como receptor tienen que tener acceso libre a este canal.
- Código: es el formato preestablecido para el envío de la información. En este punto tenemos la ayuda que nos pueda dar el servidor sobre qué espera recibir y qué va a devolver. Por ejemplo, existe la especificación abierta Swagger que ayuda a la publicación de estos requisitos del servidor.
- Mensaje: es la información necesaria para la comunicación. Establecida en el código por el servidor es encapsulada en un formato abierto como XML o JSON.

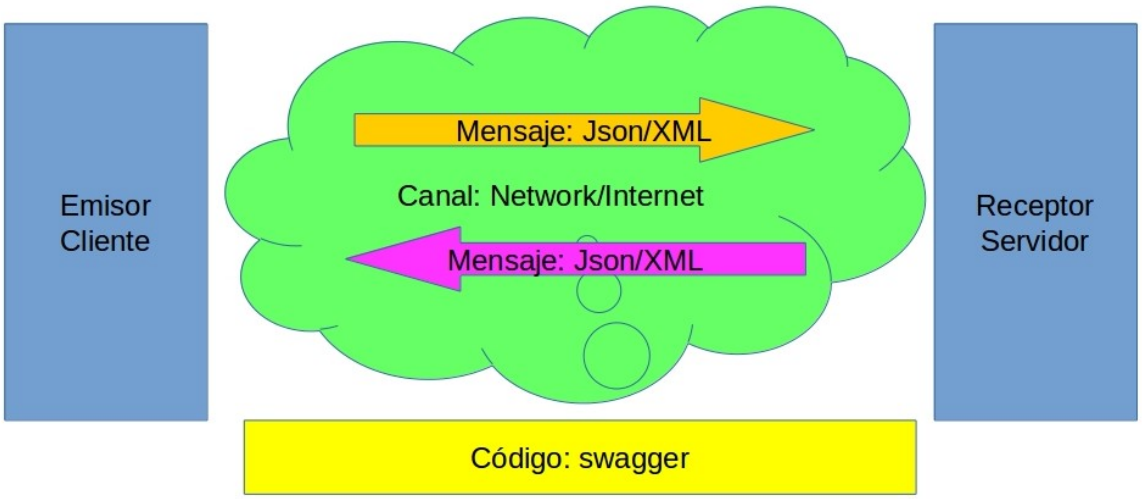


Figura 1

Todos los elementos de la comunicación para un servicio REST se basan en códigos abiertos, lo que permite que los sistemas existentes en cada uno de los lados no tengan la necesidad de ser iguales para poder comunicarse. Únicamente deben hablar el mismo idioma, en este caso también abierto, para poder establecer una buena comunicación. Esta abstracción es el mayor de los beneficios de los servicios REST.

Otro beneficio a destacar es que se trata de una tipología de servicios en donde el servidor no guarda información de estado entre llamada y llamada. Cada llamada debe pasar toda la información como si fuese la primera vez. Cuando el servidor retorna el recurso éste queda totalmente liberado. Esto hace que el servidor economice en recursos como la memoria. Todo el control del estado lo lleva el cliente.

Por lo tanto, REST es ideal para crear funciones API ya que nos aporta una mayor visibilidad de los recursos gracias al protocolo HTTP. También aporta una abstracción de los sistemas y deja de haber dependencia de un sistema concreto. Por último, aumenta el rendimiento de las llamadas al ser procesos concretos y más o menos cortos y al no tener que gestionar estados una vez se acaba la petición.

3 Método

3.1 Arquitectura y herramientas utilizadas

API REST

- S.O. Linux Mint 64x 17.3
- IDE Netbeans v. 8
- Java j2ee v. 1.7
- Tomcat v. 7
- Jersey v. 1.8
- Maven 1.7
- Swagger v. 2.0

Cliente

- S.O. Linux Mint 64x 17.3
- Apache 2.0
- PHP 5.0
- API Google Maps
- JQuery v.1.10
- Bootstrap v.3.3.5.

Para el servidor he utilizado Java EE como plataforma para construir las API por la potencia de sus librerías y servicios así como del soporte que proporciona una gran comunidad de usuarios. Me he basado en el framework Jersey para poder construir más fácilmente los servicios REST con toda la funcionalidad que incorporar. Por último, Swagger me ha dado la posibilidad de documentar los servicios REST para facilitar a los clientes la información necesaria para utilizarlos. Todo el servidor lo he montado bajo un servidor de aplicaciones Tomcat por la simplicidad del mantenimiento. Como entorno de desarrollo IDE he trabajado con Netbeans por la capacidad de integrar todas las herramientas necesarias.

Para los clientes he utilizado PHP como herramienta de desarrollo por ser libre, tener una gran comunidad y disponer de conocimientos a nivel de usuario. Bootstrap ha dado una presentación más agradable a las páginas realizadas. Por último jQuery se ha utilizado muy poco para alguna automatización necesaria. En estos casos solo he utilizado un editor de texto para generar todo el código.

Tanto los clientes como las API servidor han sido alojadas en un servidor Linux por la facilidad a la hora de configurar los distintos servidores.

3.2 Detalle del proyecto

Para llevar a cabo el proyecto he seleccionado un fichero de la lista de puntos Wifi públicos de la ciudad de Barcelona (). Este fichero, además de los datos técnicos incorpora la geolocalización de los puntos Wifi, así como a organización por distritos y barrios de la ciudad. También he incorporado dos ficheros más -modificando los originales- para darle mayor coherencia a la información final. Estos dos ficheros son Barris () y Districtes (extracción realizada a partir de CSV de Puntos Wifi). En la figura 2 podemos ver el esquema de las aplicaciones a montar.

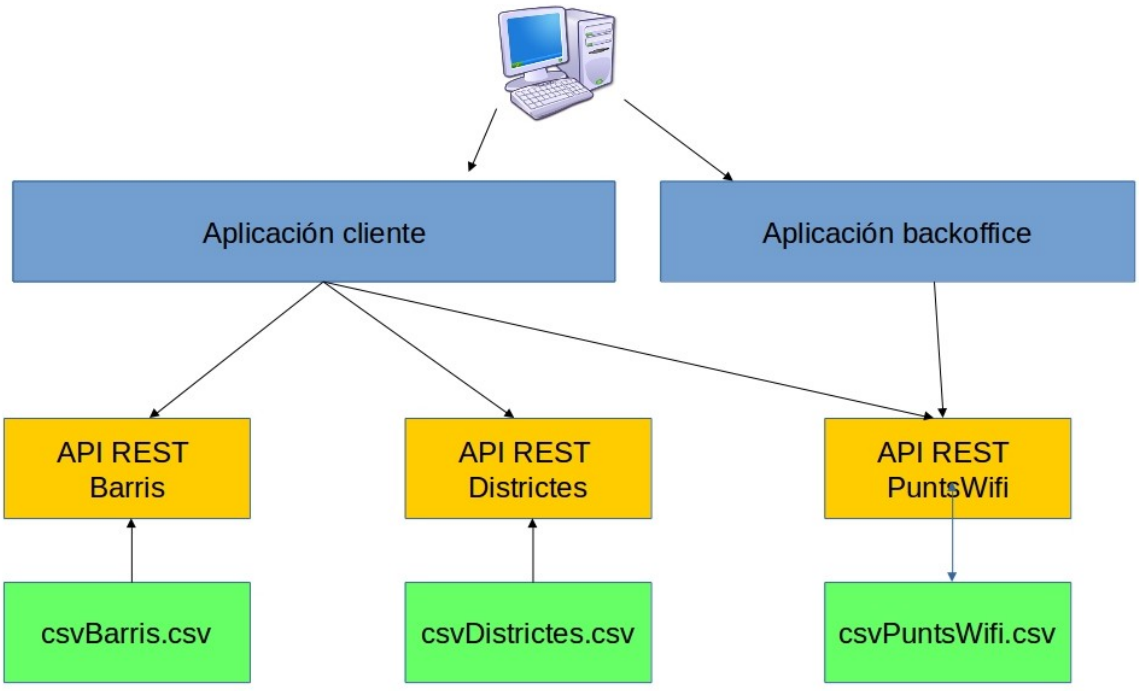


Figura 2

Para poder trabajar con el fichero primero debemos analizar su estructura y propiedades y qué operaciones haremos con esa información. Para declarar toda esta información utilizaremos la modelización UML.

Definir la estructura de la información en este tipo de formato es bastante más sencillo que en las hojas de cálculo, ya que la organización por filas con una cabecera lo simplifica mucho. De este modo, tendremos tantas propiedades como columnas tenga el fichero. No obstante, debido a que existe la posibilidad que no podamos definir un campo clave único, añadiremos la línea del registro como una propiedad más. Si se da el caso de que no existe uno o varios campo clave que presentan unicidad, utilizaremos el campo línea como el campo clave. Justamente en este fichero de puntos WIFI podemos notar la ausencia de un campo que nos identifique cada uno de los registros de forma unívoca. En este caso, podríamos haber utilizado los campos de latitud y longitud para definir la unicidad, pero al ser campos decimales he preferido no hacerlo.

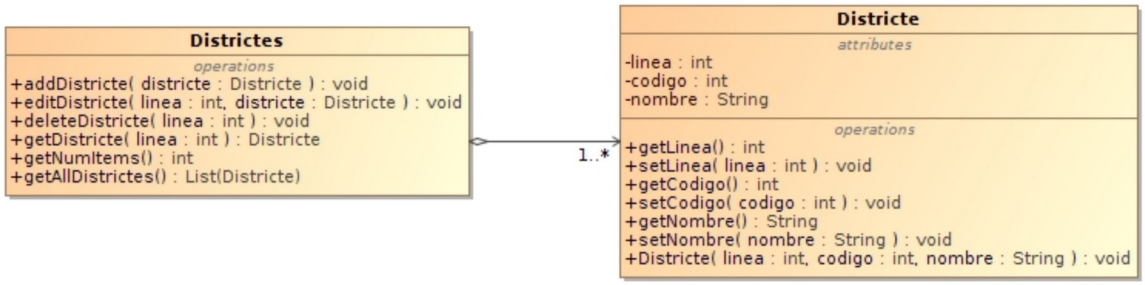
Cada modelizado de un CSV tiene dos clases. La primera hace referencia a un registro (línea) del CSV, salvo la primera fila que se destina a la cabecera. La segunda hace referencia a todo el conjunto de registros del CSV. Los registros son tratados como una lista. Todas las operaciones con los registros se realizan con métodos de esta segunda clase.

En la primera clase, la cual marca la estructura de un registro, únicamente disponemos de los gets y sets para acceder a las propiedades de este registro. Existe además un constructor que nos permite crear el objeto una vez en el momento de la inicialización. Tal y como he comentado anteriormente, se le añade un propiedad más de tipo entero largo para especificar la línea que ocupa el objeto dentro del fichero CSV.

El tipo de dato de cada propiedad vendrá especificado por el tipo de dato de la columna del CSV. Para valorar el tipo que le corresponde a cada columna deberemos analizar los valores de ésta. En muchos casos tendremos el formato claro ya sea por el contexto o incluso por el mismo nombre. En caso de duda, deberemos indicar que el tipo del campo es String (texto) y así nos aseguraremos de no perder posteriormente ningún valor tanto en la lectura como en la escritura.

En la segunda clase, se define una lista con todos los objetos del tipo de la primera clase. Las operaciones de esta clase van dirigidas a realizar tareas relacionadas con los objetos de la lista. Así tenemos métodos para obtener todos los registros o parte de ellos a través del filtrado. También se puede crear nuevos objetos, editar y borrar los registros. Las operaciones se realizan exclusivamente sobre la lista salvo en el caso de búsqueda filtrada por un valor, ya que deberemos leer las propiedades de cada objeto de la lista para poder filtrar.

Las tres Api del proyecto han sido modelizadas previamente con UML.



UML Districte

UML PuntWifi

Toda la operativa de la API se basa en el siguiente flujo de operaciones.

Al no tener estado la API, cada llamada se considera la primera por lo que antes de realizar la operación deberemos extraer del CSV toda la información y guardarla en una estructura de lista de objetos. Una vez tenemos la estructura generada en memoria podemos realizar la operación requerida. Si esta operación implica modificaciones en el fichero CSV, posteriormente deberemos hacer el proceso inverso al anterior y guardar la lista de objetos en el fichero CSV sin alterar su estructura para que pueda ser leído más adelante.

Tanto para leer o escribir tenemos en cuenta los siguientes parámetros:

- Ruta del fichero: en los parámetros se dan dos rutas distintas en dos formatos también distintos.
    - Un formato URI para tener acceso al fichero CSV para leer. Esta ruta es pública y permite a cualquiera acceder al fichero.
    - Un formato de ruta absoluta o relativa para tener acceso al archivo CSV de forma interna. Este acceso se utilizará para poder escribir sobre este fichero. Es importante que el usuario que deba realizar esta acción tenga los permisos correspondientes.
- Separador de campos dentro del registro: este tipo de ficheros suele presentar diferencias en el tipo de separador entre campos. Normalmente se encuentran los campos separados con un punto y coma “;” o bien con una coma “,”. No obstante, podemos encontrar otros caracteres como el tabulador, espacio o cualquier otro. El separador de registro se presupone que siempre es el retorno de carro CR + LF.
- Código de página: es utilizada para guardar la información. Se indica el formato de los conjuntos de caracteres utilizado y cada lenguaje tiene sus propios caracteres.

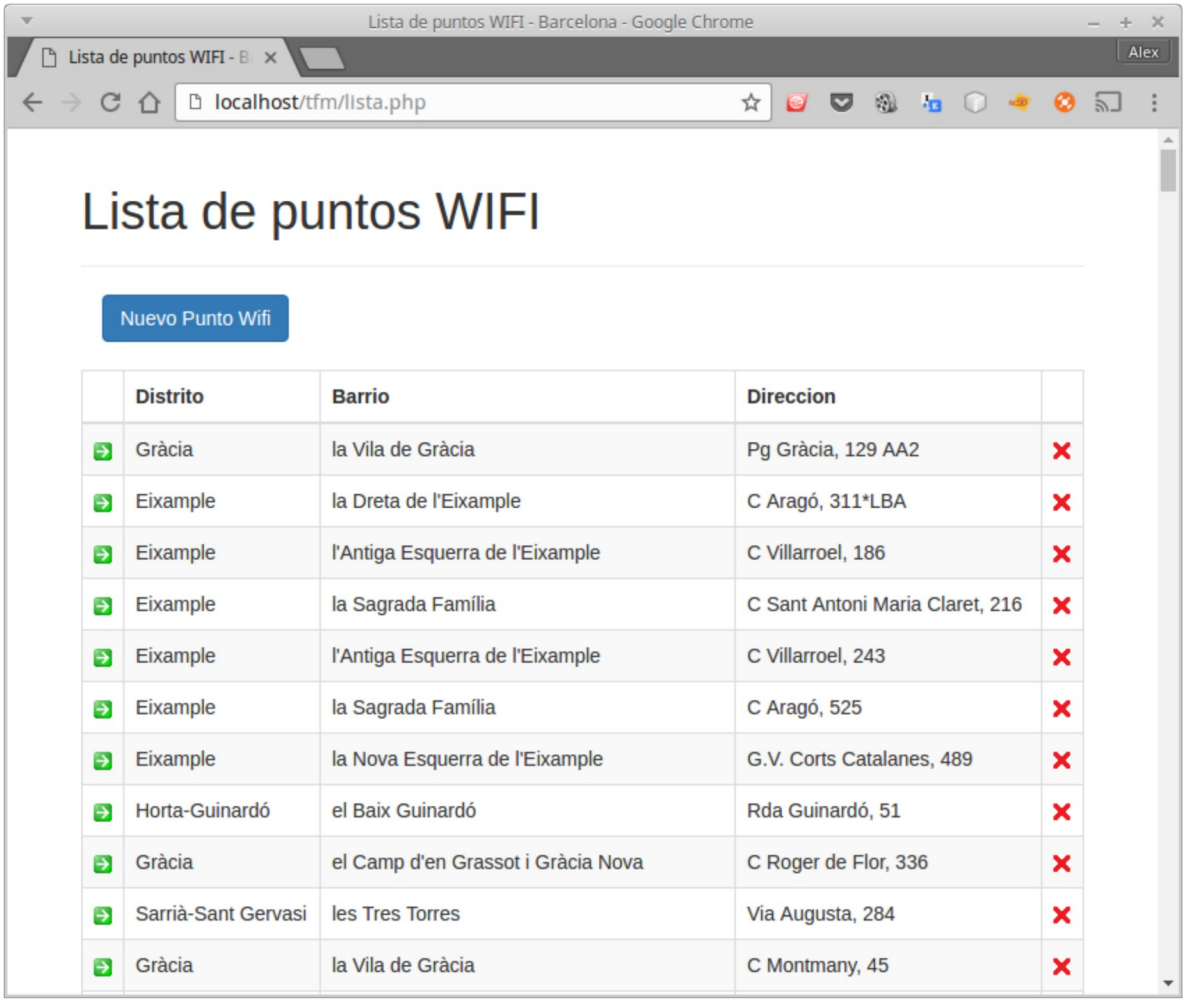
El proceso de lectura empieza abriendo el fichero CSV y, omitiendo la primera línea que es la de los títulos de las columnas o cabecera, va leyendo línea a línea los registros. Para cada línea trocea los campos a partir del separador indicado. Es importante definir el orden de los campos tanto en la lectura como en la escritura. Para cada línea se crea un objeto dentro de la lista y se inicializan los valores en base a los campos obtenidos en dicha línea.

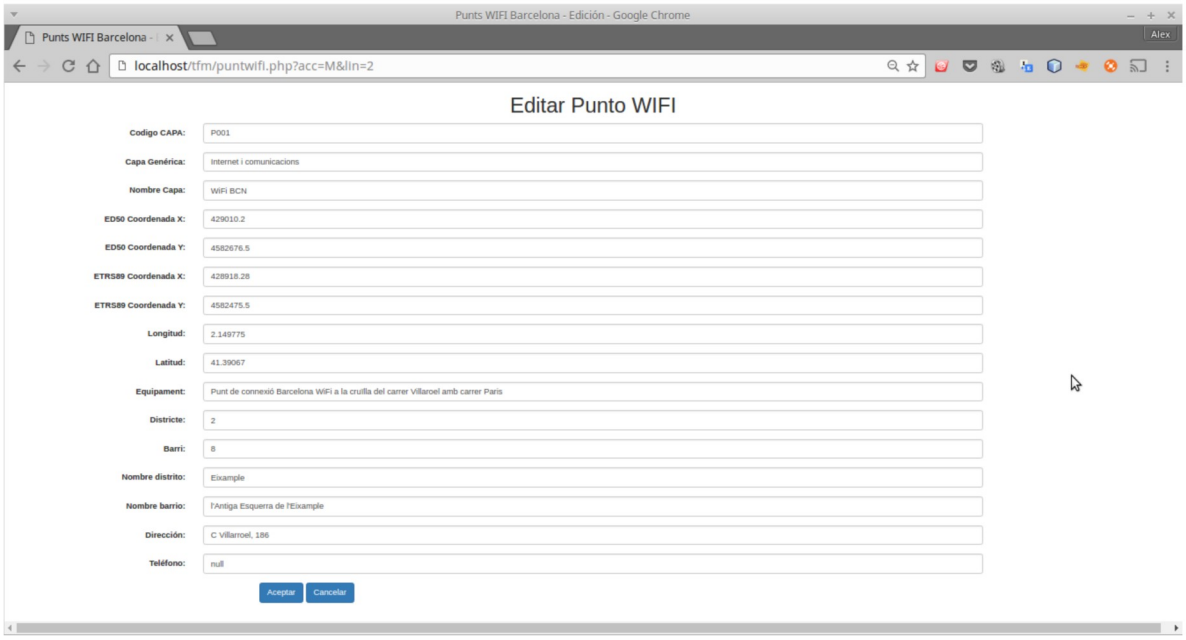
El proceso de escritura del CSV es al contrario. Primero abre el fichero y si no existiera lo creará. La primera línea que carga es la de la cabecera (en este proyecto he puesto toda la línea de golpe ya formateada). Posteriormente, va leyendo de la lista de objetos uno a uno cada objeto que corresponderá con cada una de las líneas del fichero. Para cada objeto extrae todas las propiedades manteniendo el orden de las columnas. Al final cierra el fichero.

Las funciones del servicio REST son documentadas con Swagger. Swagger es un framework utilizado para documentar servicios RESTful.

3.3 Clientes

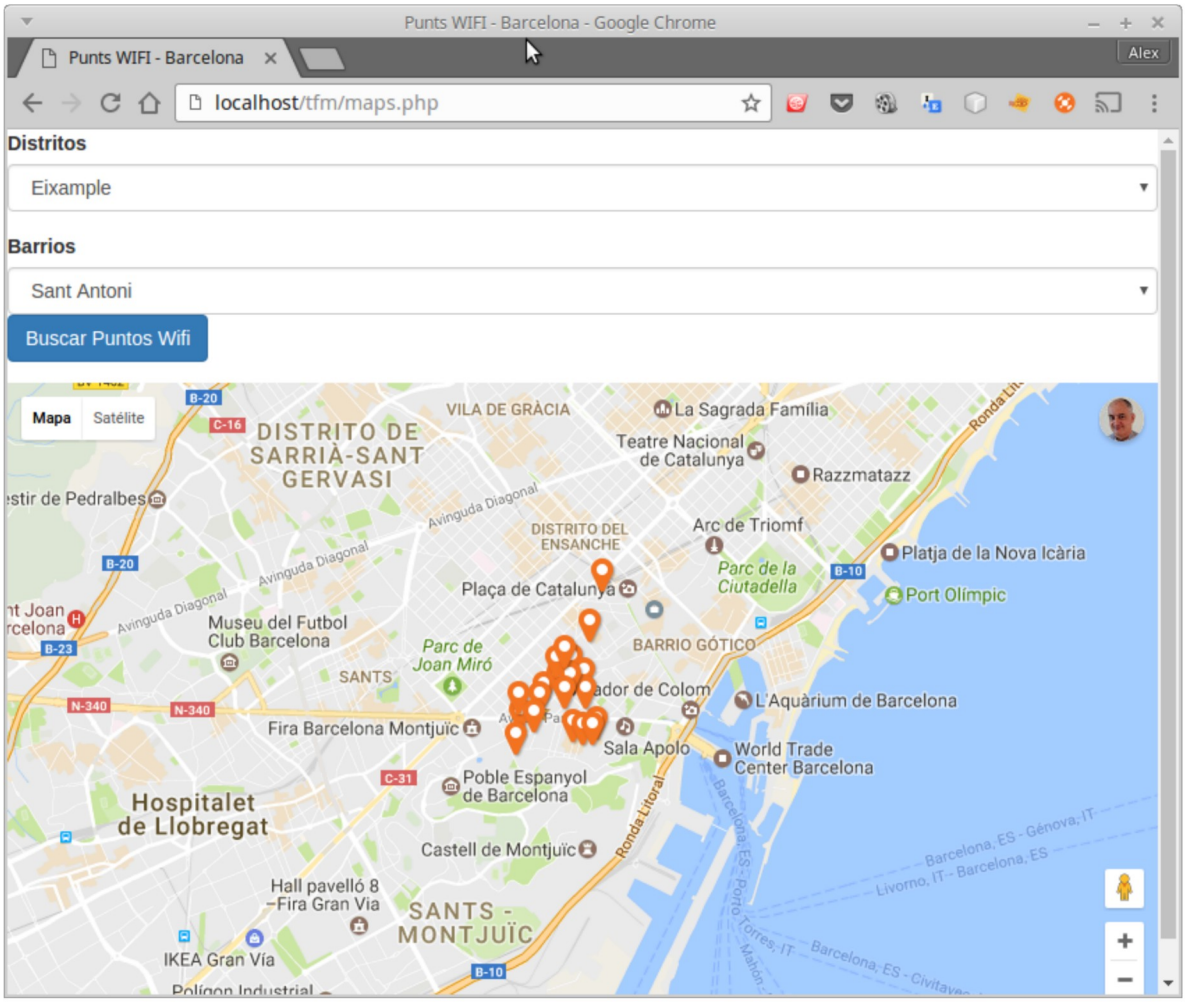
Se han desarrollado dos clientes que consumen los servicios de las APIs creadas. El primer cliente se trata del BackOffice de la API principal basado en el CSV de Puntos Wifi del Ajuntament de Barcelona. En él se pueden realizar todas las gestiones contra el mismo fichero CSV. Éstas son la de leer un registro, crearlo, actualizarlo y borrarlo. Para todas estas operaciones se van realizando llamadas al API REST publicado del csvPuntsWifi.


Lista de puntos WIFI - Barcelona - Google Chrome
Lista de puntos WIFI - B
Alex
localhost/tfm/lista.php
Lista de puntos WIFI
Nuevo Punto Wifi
Distrito
Barrio
Direccion
Gràcia
la Vila de Gràcia
Pg Gràcia, 129 AA2
Eixample
la Dreta de l'Eixample
C Aragó, 311*LBA
Eixample
l'Antiga Esquerra de l'Eixample
C Villarroel, 186
Eixample
la Sagrada Família
C Sant Antoni Maria Claret, 216
Eixample
l'Antiga Esquerra de l'Eixample
C Villarroel, 243
Eixample
la Sagrada Família
C Aragó, 525
Eixample
la Nova Esquerra de l'Eixample
G.V. Corts Catalanes, 489
Horta-Guinardó
el Baix Guinardó
Rda Guinardó, 51
Gràcia
el Camp d'en Grassot i Gràcia Nova
C Roger de Flor, 336
Sarrià-Sant Gervasi
les Tres Torres
Via Augusta, 284
Gràcia
la Vila de Gràcia
C Montmany, 45



Dado que los servicios del API no guardan el estado, cada llamada a los servicios obliga a leer el fichero CSV, cargar los datos en memoria, realizar la operación correspondiente y, si procede, salvar esta información con las modificaciones realizadas en la memoria. Esta API no es multiusuario en los casos de BackOffice ya que no controla los bloqueos de los registros. Esto únicamente es válido para operaciones que no puedan modificar la información del CSV (exclusivamente lectura).

El segundo cliente sólo realiza operaciones de lectura de los ficheros CSV. El objetivo de este cliente es mostrar los puntos wifi de un barrio concreto de la ciudad de Barcelona sobre un mapa de Google Maps incrustado en la misma página. Este cliente trabaja con las tres API REST creadas : la API de Barris, la API de districtes y la API de PuntsWifi. Las dos primeras filtran el contenido a mostrar y la API de PuntsWifi selecciona la información, geolocaliza los puntos seleccionados y muestra información suplementaria. Este cliente utiliza el componente de Google Maps con una clave creada para este proyecto. El acceso a las APIs es exclusivamente de lectura, por lo que no hace falta disponer de ninguna clave de autorización ya que no modificaremos nada.



2

4 Conclusiones del trabajo

La API REST para cada uno de los CSV hace de puente entre el usuario a través de una aplicación y el fichero CSV. Se impide la modificación directa del fichero y las posibles modificaciones de éste se realizan de forma controlada.

Una vez realizado todo el proceso de creación de la API, desde la modelización de la estructura y los métodos para acceder al fichero CSV, y la creación de los clientes he llegado a las siguientes conclusiones.

En primer lugar, el acceso a la información no ha tenido penalizaciones grandes en tiempo debido a que los ficheros eran de tamaño pequeño. Si el fichero CSV es de un tamaño pequeño o medio, el acceso a los servicios REST no se ve penalizado de forma visible. En cambio, si este fichero es de gran tamaño, con infinidad de registros, el acceso a través de los mismos servicios se ve penalizado ya que debe leer todo el fichero cada vez en cualquier operación.

En segundo lugar, el acceso de consulta a la información es ágil y puede llegar a personalizarse según las diferentes necesidades tanto el acceso como el control de la información que se pretende devolver al cliente con las funciones de la API REST.

En tercer lugar, la API no está pensada para acceso multiusuario en aquellos procesos que implique modificar la información del fichero CSV (Update, Insert y Delete). Al no poder los servicios REST guardar un estado, se impide gestionar fácilmente el bloqueo de la información en caso de quererla modificar. Ésto no afecta a las funciones de consulta.

En cuarto lugar, sobre los estudios de ClassSheets en los que me había basado para el proyecto tengo que decir que al final en este proyecto he simplificado el proceso ya que, por ejemplo, no he tenido que controlar el crecimiento de la información por celdas o columnas para no perder consistencia en las fórmulas, ya que dado el formato del tipo de fichero CSV no existe la figura de la fórmula. No obstante, al modelizar y controlar la estructura a través de la API se limita el acceso al fichero y el usuario sólo puede administrar una información controlada.

Finalmente, el proceso de apificación y la obtención de ClassSheets son similares porque empiezan con la modelización UML. A partir de allí, los dos procesos siguen caminos diferentes ya que ClassSheets busca controlar la estructura del fichero mientras está siendo modificada En cambio, con la API REST estableces un puente que administra los flujos de información para mantener la estructura.

Por tanto, la solución API REST para gestionar ficheros CSV es válida porque no requiere controlar operaciones sobre las celdas de información no sobre las fórmulas que pueda tener la hoja de cálculo. El método para apificar el contenido sí que es similar ya que utiliza la modelización UML y si hiciese falta se podría utilizar OCL para aplicar restricciones.

5 Bibliografía

[1] Cunha, J., Fernandes, J. P., &Saraiva, J. (n.d.). From Relational ClassSheets to UML+OCL.

[2] Engels, G., &Erwig, M. (n.d.). ClassSheets: Automatic Generation of Spreadsheet Applications from Object Oriented Specifications.

[3] Hermans, F., Pinzger, M., & Van Deursen, A. (2010). Automatically Extracting Class Diagrams from Spreadsheets. Lecture Notes in Computer Science.

[4] BALS, Jan-Christopher, et.al. ClassSheets-model-based object-oriented design of spreadsheet applications. Journal of Objetc Technology, 2007, vol. 6, no 9, p. 383-398.

[5] JIMENEZ de Parga, Carlos. UML Aplicaciones en Java y C++. RA-MA. 2015. 411. ISBN. 978-84-9964-516-2.

[6] Martin's Francis, Dimitrios S. Kolovos Nicholas Matragkas, and R. F. P. (2013). Adding Spreadsheets to the MDE Toolkit. Model-Driven Engineering Languages and Systems, 35–51.

[7] Ed-Douibi, H., Izquierdo, J. L. C., Gómez, A., Tisi, M., &Cabot, J. (2016, April). EMF-REST: generation of RESTful APIs from models. In Proceedings of the 31st Annual ACM Symposium on Applied Computing (pp. 1446-1453). ACM.

[8] Fielding, Roy Thomas (2000). "Chapter 5: Representational State Transfer (REST)". Architectural Styles and the Design of Network-based Software Architectures (Ph.D.). University of California, Irvine.

[9] Mulloy, B.: Web API Design - Crafting Interfaces that Developers Love. Apigee

[10] Whatis Open? Open Knowledge International

[11] Rest – Restful: ventajas y diferencias Damián Wajser, Technical Team Lead Softtek