Universidad de Valladolid

E.U. DE INFORMÁTICA (SEGOVIA)

Grado en Ingeniería Informática de Servicios y Aplicaciones

Aplicación de Técnicas de Web Scraping al BOCyL

Alumna: Cristina Hernández Herrero

Tutor: Miguel Ángel Martínez Prieto

Desde estas líneas me gustaría expresar mi más sincero agradecimiento

a mi tutor Miguel Ángel, por ayudarme y guiarme
durante todo el proyecto

a mi familia y mi novio, por haberme apoyado y animado
a lo largo de todos estos años

a mis amigos y compañeros,
especialmente a Eme y Sandra,
por los momentos vividos durante estos últimos años

Resumen

Este proyecto se propone estudiar el concepto de *Web Scraping* y cómo mediante su uso, es posible extraer de forma estructurada la información contenida en un conjunto de páginas web con estructura similar. A lo largo del documento se conocerán las técnicas fundamentales de *scraping*, junto con la implementación de una herramienta que permite extraer la información de un sitio web de interés general como es el Boletín Oficial de Castilla y León. (BOCyL), y almacenarlo en una base de datos relacional. Además de conocer la creación de la herramienta, se mostrará un posible uso de la información como es la creación de archivos Open Data. Para finalizar, se plantearán diferentes usos que se pueden dar a la información.

Abstract

This project aims to study the concept of Web scraping, and its uses for extracting structured information from a given website. Web scraping solutions expect that the target website publishes contents following the same structure within the web pages, so it can be "learned" and its data can be extracted and organized according to their semantics. The document describes a brief review of Web Scraping fundamentals and introduces some techniques. This knowledge are then used as basis for developing a project which scraps a website of general interest: the Boletín Oficial de Castilla y León (BOCyL). All extracted data are stored, for potential uses and purposes, in a relational database. In addition to the "scrap & store" workflow, we also deploy a proof-of-concept solution which aims to demonstrate the data value for third-party applications. This application leverages our BOCyL database for releasing the daily bulletins as Open Data.

Tabla de contenido

1. Introducción 7
1.1. Alcance del sistema 8
1.2. Objetivos 9
1.3. Gestión del proyecto 9
1.4. Organización del Documento 15
2. Data Scraping 17
2.1. ¿Qué es web scraping? 18
2.2. Funcionamiento del web scraping 18
2.3. Técnicas usadas para el web scraping 19
2.4. Herramientas para el web scraping 20
2.5. Impedimentos para realizar web scraping. 21
2.6. Ejemplos de proyectos 23
3. Análisis del sistema 25
3.1. BOCyL 26
3.2. Actores del sistema 27
3.3. Requisitos 28
3.4. Casos de uso 33
3.5. Modelo de datos conceptual 35
4. Diseño del Proyecto 37
4.1. Arquitectura lógica 37
4.2. Arquitectura física 39
4.3. Modelo de datos lógico 39
5. Implementación del Proyecto 43
5.1. Paso de la arquitectura lógica a la implementación 43
5.2. Fase 0: Análisis de la estructura del BOCyL. 45
5.3. Fase I: Extracción de datos del BOCyL. 49
5.4. Fase II: Explotación de los datos en Open Data. 65
5.5. Prototipo 69
6. Pruebas 73
7. Conclusiones 79
8. Bibliografía 81
ANEXO I: Manual de usuario 85

Índice de Tablas

Tabla 1: Complejidad de los procesos 10
Tabla 2: Complejidad de Entradas y Consultas 10
Tabla 3: Complejidad de Salidas 10
Tabla 4: Puntos de Función no Ajustados 10
Tabla 5: Grado de Complejidad 11
Tabla 6: Factores de Ajuste 11
Tabla 7: Factor de esfuerzo 12
Tabla 8: Presupuesto Hardware 13
Tabla 9:Presupuesto Software 13
Tabla 10: Distribución de horas 13
Tabla 11: Coste Trabajador 13
Tabla 12: Presupuesto Total 14
Tabla 13: ACT-01 27
Tabla 14: ACT-02 27
Tabla 15: RQF 01 28
Tabla 16: RQF 02 28
Tabla 17: RQF 03 29
Tabla 18: RQF 04 29
Tabla 19: RQF 05 29
Tabla 20: RQNF 01 30
Tabla 21: RQNF 02 30
Tabla 22: RQNF 05 30
Tabla 23: RI 01 31
Tabla 24: RI 02 31
Tabla 25: RI 03 31
Tabla 26: RI 04 32
Tabla 27: RI 05 32
Tabla 28: RI 06 32
Tabla 29: RI 07 32
Tabla 30: Matriz de interacción de requisitos 33
Tabla 31: UC 01 34
Tabla 32: UC02 34
Tabla 33: UC 03 35
Tabla 34: CP- 01 74
Tabla 35: CP- 02 74
Tabla 36: CP- 03 75
Tabla 37: CP- 04 75
Tabla 38: CP- 05 76
Tabla 39: CP- 06 76
Tabla 40: CP- 07 77

Índice de Ilustraciones

Ilustración 1: Tiempo de desarrollo estimado 14
Ilustración 2: Tiempo de desarrollo real 15
Ilustración 3: Estructura de Directorios 16
Ilustración 4: 'Trasparencia de Cuentas Públicas' 23
Ilustración 5: 'El Indultómetro' 23
Ilustración 6: Diagrama de casos de uso 33
Ilustración 7: Modelo E-R 35
Ilustración 8: Arquitectura lógica 38
Ilustración 9: Arquitectura física 39
Ilustración 10: Modelo Relacional 40
Ilustración 11: Página web de un boletín 45
Ilustración 12: Página web de un documento 46
Ilustración 13: Ejecución del script 69
Ilustración 14: Boletín almacenado 70
Ilustración 15: Ejecución con url inválida 74
Ilustración 16: Ejecución de un boletín no publicado 75
Ilustración 17: Prueba validar fecha 76

1. Introducción

Desde siempre el ser humano ha deseado ampliar sus conocimientos con la obtención de datos e información. La aparición de internet y posteriormente, en 1990, de la web permitió satisfacer de forma importante este deseo, al proporcionar a los usuarios información mediante la visualización pasiva de páginas web, que otras personas habían creado para ellos. Más tarde, en 2004 se produjo una revolución en la web, que permitió a los usuarios no sólo poder obtener información y datos con la lectura pasiva de páginas, sino la posibilidad de que el usuario interactúe y colabore como creador de contenido en una comunidad. Ésta revolución dio paso a denominar la web a partir de entonces como web 2.0 o 'de lectura y escritura', y a todo lo anterior como web 1.0 o sólo de lectura. Por ello, la web 2.0 aporta a los usuarios una mayor cantidad de información y datos, al permitir que usuarios sin conocimientos técnicos puedan mostrar y compartir sus conocimientos con los demás de una forma sencilla. Del mismo modo, también facilita la comparación de la información y de sus fuentes para determinar la veracidad de la misma.

La información que se muestra en las páginas web tiene como fin ser entendida y procesada por personas, por ello, resulta muy difícil que una máquina sea capaz de entender un texto que no esté estructurado. Actualmente se está desarrollando lo que ya se ha denominado web 3.0 y que pretende poder crear páginas web con metadatos semánticos que describan el contenido y la relación entre los datos, de forma que sea posible ser evaluadas automáticamente por máquinas. Además, la web 3.0 intenta conseguir que los datos sean identificables dentro de la estructura de internet, es decir, que las búsquedas sean mucho más concretas y fiables.

La imposibilidad actual de interpretar la información, residente en las páginas, mediante el uso de máquinas de forma automática, impide que dicha información pueda ser extraída para ser reutilizada en otra actividad, limitando así las funciones que se le pueden dar a una misma información. Este motivo es una de las principales razones por las que emerge el concepto de *web scraping* que se va a estudiar durante el trabajo. Como se explica más adelante en este capítulo, el objetivo principal del proyecto, es utilizar técnicas de *web scraping* para extraer de forma automática la información, y posteriormente, ser reutilizada para propósitos distintos.

Por otro lado, en los últimos años y debido al gran alcance mundial que otorga Internet muchas entidades, principalmente gubernamentales, han decidido publicar abiertamente la información de sus actividades permitiendo que esta información tenga un acceso público y gratuito. El problema reside cuando esta información se encuentra en una página de forma estática, simplemente se muestra, sin permitir que la información sea utilizada para otros fines como comparaciones entre documentos, etc. Todo esto se desarrolla paralelamente a los movimientos de Open Data que se desarrollan en la actualidad, y que persiguen los objetivos de poner a disposición de la sociedad los datos que gestiona la administración pública en formatos fáciles de manipular.

Por ello, este proyecto tiene la intención de extraer la información de una página web, en concreto de una página gubernamental, y almacenarlos en archivos Open Data que permitan realizar otra actividad con esa información, ampliando sus posibilidades de utilización.

1.1. Alcance del sistema

El alcance del sistema será de forma local en un principio, ya que la obtención y almacenamiento de información se realizará en un único ordenador.

Se pretende realizar una aplicación lo más intuitiva y sencilla de utilizar en el día a día, aunque como existe la posibilidad de que los administradores de la página de la que se pretende obtener la información, modifiquen la página, el usuario deberá de tener conocimientos suficientes para modificar la aplicación.

No se descarta que posteriormente se realice algún proyecto basado en este, permitiendo reutilizar los datos obtenidos en cualquier otra actividad, de modo que se amplíe el alcance que tiene actualmente el sistema.

1.2. Objetivos

Los objetivos que se persiguen con este proyecto es el de conocer de forma teórica la técnica de *Web Scraping* en profundidad, lo que implica, estudiar las herramientas existentes en la actualidad y que permiten realizar su cometido.

Así mismo, se pretende ejemplarizar esta técnica con la obtención de los datos del BOCyL de modo que se almacenarán en una base de datos, permitiendo demostrar que han sido extraídos correctamente. Una vez extraídos los datos de la página web y almacenados en la base de datos, se recopilará la información y almacenará en archivos Open Data siguiendo la filosofía de este movimiento, y posibilitando la reutilización de la información por otras personas para distintas actividades. El formato que van a tener los archivos de Open Data será XML (*extensible markup language*).

1.3. Gestión del proyecto

Estimación de costes económicos y temporales

Mediante el modelo COCOMO se va a determinar el coste asociado al proyecto. Para ello, primero se va a estimar los puntos de función (PF), y después mediante unos factores se va a determinar el esfuerzo temporal y el número de personas necesarias para realizar el proyecto.

Estimación por Puntos de Función (PF)

Los valores de los dominios de información y su complejidad se definen de la forma siguiente:

Número de entradas de usuario: Se cuenta cada entrada de usuario que proporciona diferentes datos orientados a la aplicación.

- Enlace del boletín: complejidad simple.
- Datos del boletín: complejidad alta.

Número de salidas de usuario: Se cuenta cada salida que proporciona al usuario información orientada a la aplicación, informes, pantallas, etc.

- Información del boletín: complejidad alta.
- Información del boletín en Open Data: complejidad alta.

Número de consultas de usuario: Una petición es una entrada interactiva que genera alguna respuesta del software inmediata en forma de salida interactiva.

- Script para la obtención del boletín: complejidad alta.
- Script para la obtención del boletín en Open Data: complejidad alta.

Número de ficheros internos:

- Base de datos: complejidad alta.

Número de ficheros externos:

- Manual de técnico: complejidad media.

El siguiente paso es clasificar los elementos de cada clase según su grado de complejidad (alta, media o baja). La asignación de complejidades a FLI (fichero lógico interno) y FE (fichero de externo) se basa en el número de Tipos de Elementos de Datos (TED) y número de Tipos de Elementos de Registros (TER). Un TED se define como

un campo único, no recurrente y reconocible para el usuario en un FLI o FE. Un TER se define como un subgrupo de elementos de datos reconocibles para el usuario dentro de un FLI o FE. Una vez conocidos los TED y los TER propios de cada fichero podemos establecer el nivel de complejidad apoyándonos en la siguiente tabla:

	1 a 19 TED	20 a 50 TED	51 ó más TED
1 TER	Baja	Baja	Media
2 a 5 TER	Baja	Media	Alta
6 o más TER	Media	Alta	Alta

Tabla 1: Complejidad de los procesos

Para hallar la complejidad de los procesos, entradas externas, salidas externas y consultas, utilizamos la siguiente tabla de ponderaciones establecidas en la técnica de puntos de función.

Entradas. y Consultas	1 a 4 TED	5 a 15 TED	> 51 TED
0-1 TER accedidos	Baja	Baja	Media
2 TER accedidos	Baja	Media	Alta
> 2 TER accedidos	Media	Alta	Alta

Tabla 2: Complejidad de Entradas y Consultas

Salidas	1 a 5 TED	5 a 19 TED	> 19 TED
0-1 TER accedidos	Baja	Baja	Media
2-3 TER accedidos	Baja	Media	Alta
> 3 TER accedidos	Media	Alta	Alta

Tabla 3: Complejidad de Salidas

Por último se obtienen los puntos de función no ajustados (PFNA) mediante una suma ponderada de esas cantidades con los pesos que aparecen en la Tabla 4:

Tipo de función	Complejidad	Total x Complejidad	Total por tipo	Suma
Ficheros internos	Simple Media Alta	0x7 0x10 1x15	15	15
Ficheros externos	Simple Media Alta	0x5 1x7 0x10	7	7
Entradas de usuario	Simple Media Alta	1x3 0x4 1x6	3+6	9
Salidas de usuario	Simple Media Alta	0x4 0x5 2x7	14	14
Consultas de usuario	Simple Media Alta	0x3 0x4 2x6	12	12
Total de puntos de función				57

Tabla 4: Puntos de Función no Ajustados

Una vez obtenidos los PFNA, deben ser ajustados mediante un factor de Ajuste (FA). El cálculo del factor de ajuste está basado en 14 características generales de los sistemas, que miden la funcionalidad general y complejidad/influencia de la aplicación.

A cada característica se le atribuye un peso de 0 a 5, e indica el grado de complejidad/influencia que tiene cada característica.

Grado	Descripción Complejidad	Grado	Descripción Influencia
0	No está presente o su comp. no es tenida en cuenta	0	No está presente o no influye
1	Complejidad mínima	1	Influencia mínima
2	Complejidad moderada	2	Influencia moderada
3	Complejidad promedio	3	Influencia promedio
4	Complejidad significativa	4	Influencia significativa
5	Complejidad fuerte	5	Influencia fuerte

Tabla 5: Grado de Complejidad

Calculamos el grado de complejidad de cada característica para el cálculo del factor de ajuste:

Factores de ajuste	Complejidad
Comunicación de datos	4
Funciones distribuidas	4
Rendimiento	3
Gran carga de trabajo	4
Frecuencia de transiciones	4
Entrada on-line de datos	4
Requisito de manejo del usuario final	2
Actualizaciones on-line	2
Procesos complejos	4
Utilización de otros sistemas	1
Facilidad de mantenimiento	4
Facilidad de operación	4
Instalación en múltiples lugares	2
Facilidad de cambio	3
TOTAL:	45

Tabla 6: Factores de Ajuste

Cálculo del Factor de Ajuste (PF) a partir de la suma de los 14 factores de complejidad.

FA=(0,01x45)+0,65=1.1

Una vez calculado el PF, se procede a obtener el número de líneas de código estimadas tomando como referencia la equivalencia en LDC. El LCD de Python no está determinado pero algunos documentos indican que es menor o igual al lenguaje de programación Perl, por lo que vamos a tomar como mediada el valor para Perl que es 21 LCD por Punto de Función.

PF=57x1,1=62,7

62,7 PF x 21 LDC/PF = 1316,7 LDC aproximadamente 1,4 KLDC

Estimación Mediante COCOMO:

Clasificamos nuestro sistema a desarrollar como un sistema software rígido o empotrado, por tener unos requisitos muy restrictivos y ser un problema único imposibilitando basarse en la experiencia previa. Para pasar al COCOMO intermedio hay que aplicar un factor para el esfuerzo. Este factor son 15 atributos del proyecto, agrupados en cuatro grandes grupos: Atributos del producto, atributos de la

computadora, atributos del personal y atributos del proyecto. Cada atributo se mide en varios grados y cada grado toma un valor.

Esfuerzo nominal=2,8x(1,4)^1,2=4,19

Esfuerzo=4,19x1,15x1,29x1,14x0,7x0,91=4,5 personas

Tiempo=2,5x(4,5)^0,32=4,045 meses

No. Personas=4,5/4,045=1,11 personas -> una persona al mes

Tras obtener el valor de este factor, de manera inmediata obtendremos el tiempo de desarrollo y el número de personas necesarias para hacerlo.

	Valor de los factores					
FACTORES	Muy bajo	Bajo	Medio	Alto	Muy alto	Extra
Fiabilidad requerida	0,75	0,88	1,00	1,15	1,4	
Tamaño de la base de datos		0,94	1,00	1,08	1,16	
Complejidad del software	0,70	0,85	1,00	1,15	1,30	1,65
Restricciones de tiempo de ejecución			1,00	1,11	1,30	1,66
Restricciones de memoria			1,00	1,06	1,21	1,56
Volatilidad del hardware		0,87	1,00	1,15	1,30	
Restricciones de tiempo de respuesta		0,87	1,00	1,07		
Calidad de los analistas	1,46	1,19	1,00	0,86	0,71	
Experiencia con el tipo de aplicación	1,29	1,13	1,00	0,91	0,82	
Experiencia con el hardware	1,21	1,10	1,00	0,90		
Exp. con el lenguaje de programación.	1,14	1,07	1,00	0,95		
Calidad de los programadores	1,42	1,17	1,00	0,86	0,70	
Técnicas modernas de programación	1,24	1,10	1,00	0,91	0,82	
Empleo de herramientas	1,24	1,10	1,00	0,91	0,83	
Restricciones a la duración del proyec.	1,23	1,08	1,00	1,04	1,10	

Tabla 7: Factor de esfuerzo

Presupuesto

Para desarrollar el proyecto se necesitarán medios Hardware y Software cuyo coste proporcional al uso que se le dará en el proyecto, hay que introducirlo en el presupuesto. Naturalmente, también tendrá que incluir el coste de los recursos humanos utilizados.

Presupuesto Hardware: Ordenador para el desarrollo del sistema, la implantación y pruebas del mismo, la generación de la documentación y para el análisis. Conexión a internet para obtención de información y descarga de software. Impresora para imprimir la documentación e información que sea necesaria entregar.

HARDWARE	USO (%)	COSTE TOTAL (€)	COSTE (€)
Ordenador personal	9,375	1000	93,75
Conexión a internet	(4 meses)	40	160
Impresora	2	199	18,65

TOTAL: 272,4€

Tabla 8: Presupuesto Hardware

Presupuesto Software: Se utilizarán las siguientes herramientas con sus costes asociados:

SOFTWARE	USO (%)	COSTE TOTAL (€)	COSTE (€)
Windows 7	9,375	154,60	14,5
MySQL	9,375	0	0
StarUML	9,375	0	0
Microsoft Office 2007	(4 meses)	12,8	51,2

TOTAL: 65,7€

Tabla 9:Presupuesto Software

Presupuesto del Desarrollo del Proyecto: En la siguiente tabla se encuentra un desglose de las tareas necesarias para llevar a cabo el proyecto y la duración estimada de las mismas en horas de trabajo. Teniendo en cuenta que se ha estimado que los trabajos durarán meses, y que cada día se pretende trabajar 8 horas una media de 22 días por mes:

TAREA	DURACIÓN (HORAS)
Estudio de la técnica	60
Requisitos del sistema	65
Análisis de componentes	42
Diseño de componentes	30
Implementación de componentes	188
Pruebas de componentes	95
Pruebas de aplicación	82
Documentación	150
TOTAL:	712 HORAS

Tabla 10: Distribución de horas

Multiplicando por el sueldo por hora de un Titulado de Grado en Informática, se obtiene el coste total de la mano de obra:

	TIEMPO	**COSTE**
Ingeniero	712 HORAS	9 / Hora

TOTAL: 6408 €

Tabla 11: Coste Trabajador

Presupuesto Total: La estimación del presupuesto total es la suma de los presupuestos que hemos estimado anteriormente.

	COSTE
Hardware	272,4
Software	65,7
Desarrollo	6408
TOTAL :	6746,1 €

Tabla 12: Presupuesto Total

Planificación Temporal

La distribución de las actividades a desarrollar dentro del plan temporal establecido anteriormente se refleja en la Ilustración 1.

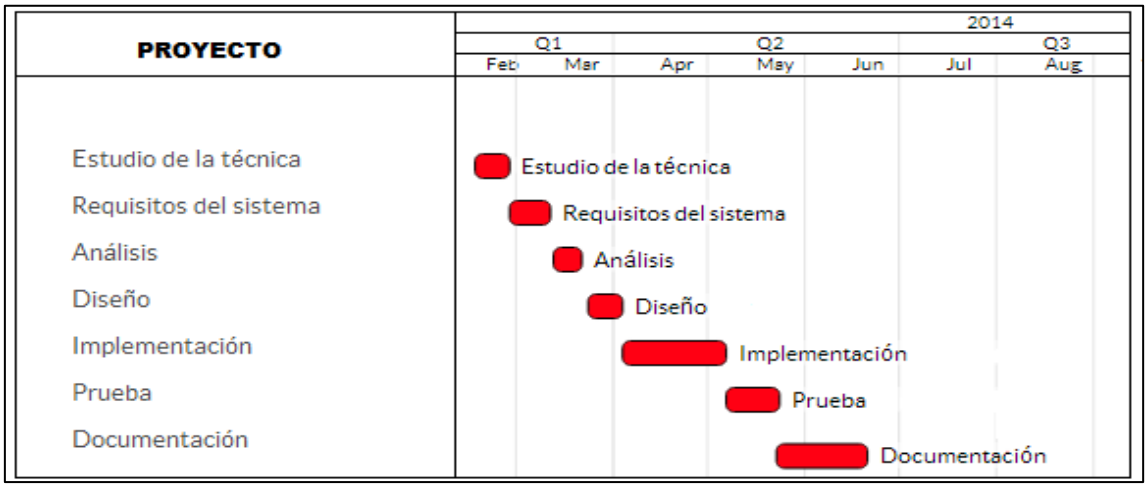

Ilustración 1: Tiempo de desarrollo estimado

La distribución que se realizó finalmente se muestra en la Ilustración 2, de donde se observa que varía notablemente de la que se estimó inicialmente. Entre los motivos que han llevado a esta variación destaca, que durante los primeros meses del desarrollo del proyecto se dedicaban aproximadamente 3 horas diarias en vez de las 8 horas que se estipularon en un principio, además, el desarrollo de la documentación se realizó durante prácticamente todo el desarrollo del proyecto, por lo que el número de horas empleadas en realizar la documentación son las mismas que las estipuladas en un inicio, pero distribuidas por todo el tiempo del proyecto. Por lo cual, la duración en horas de las actividades ha sido la misma, pero distribuidas en un número mayor de meses.

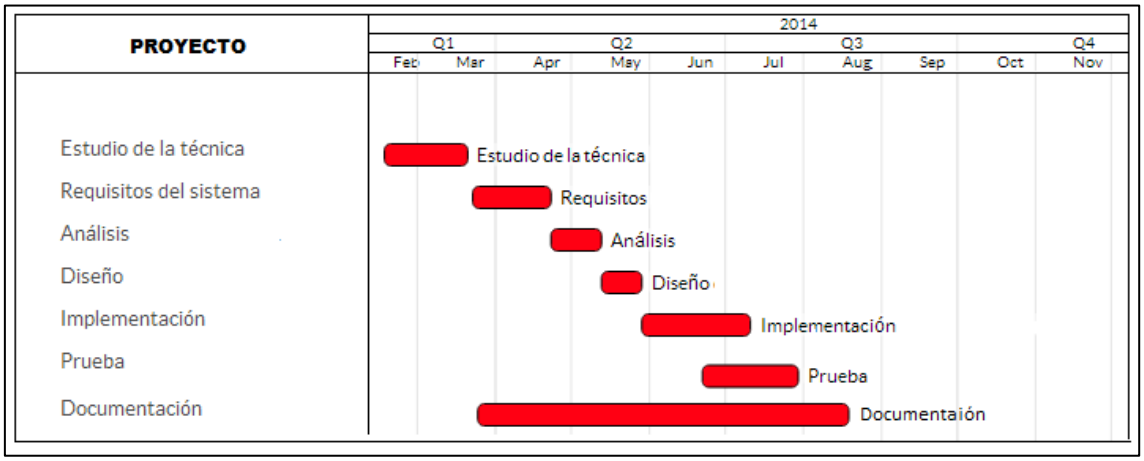

Ilustración 2: Tiempo de desarrollo real

1.4. Organización del Documento

En esta sección se va a describir la estructura de documento de modo que sirva de ayuda al lector. El documento se va a dividir en ocho capítulos.

Capítulo I: Introducción

Este capítulo es en el que nos encontramos y presenta una presentación sobre el documento, así como el presupuesto económico y temporal que conlleva realizar el proyecto.

Capítulo II: Data Scraping

Este capítulo trata de asentar los conceptos referentes al *scraping*. Para ello, explicará qué es el *scraping*, así como su funcionamiento. También expone y explica algunas de las técnicas y herramientas que permiten realizar *web scraping*. Además, explica los impedimentos que nos encontramos a la hora de realizar *web scraping* y muestra unos ejemplos de proyectos que han realizado esta actividad.

Capítulo III: Análisis del sistema

En éste capítulo vamos a describir el dominio del problema en el que nos encontramos. También se presentarán los requisitos y los casos de uso del proyecto, así como el modelo entidad-relación que presenta los datos que se van a almacenar.

Capítulo IV: Diseño del Proyecto

En esta parte del documento se va a proceder a mostrar y explicar la arquitectura lógica y física que presenta el documento. Además de mostrar el modelo de datos lógico que representa la base de datos, y que se obtiene del modelo de datos conceptual.

Capítulo V: Implementación del Proyecto

En este capítulo se va a proceder a describir la implementación necesaria para que el proyecto se lleve a cabo. Antes de comenzar la implementación se va a realizar el paso de la arquitectura lógica que anteriormente se habrá definido a la implementación.

Capítulo VI: Pruebas

En esta parte del documento se van a presentar las pruebas que se han realizado para asegurarnos de que el proyecto se comporta siguiendo los requisitos establecidos

Capítulo VII: Conclusiones

En este capítulo del documento, se mostrarán las conclusiones obtenidas tras la realización del proyecto.

Capítulo VIII: Bibliografía

En este capítulo del documento, se mostrará la información exterior consultada para poder realizar el proyecto.

Contenido del CD-ROM

En la Ilustración 3 se muestra la estructura de directorios que contiene el CD-ROM.

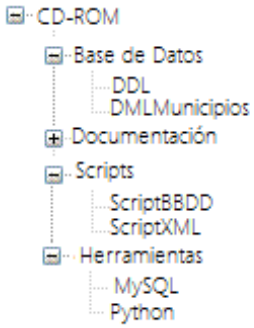

Ilustración 3: Estructura de Directorios

Como se puede ver en la Ilustración 3, el CD-ROM que se entrega junto a la documentación contiene:

Carpeta Base de Datos: en ella residirán todos los archivos necesarios para crear a base de datos y que funcione correctamente. Los archivos serán:

- DDL, necesario para crear la base de datos.
- DMLmunicipios, necesario precargarlo en la tabla municipio de la base de datos.

Carpeta documentación: en ella estará el documento en formato PDF.

Carpeta scripts: en ella estarán los archivos de los scripts que permiten el funcionamiento del programa. Los archivos se denominan 'scriptBBDD' y 'scriptXML'.

Carpeta herramientas: contendrá las herramientas necesarias para el correcto funcionamiento del programa. Las herramientas son: el gestor de base de datos MySQL y el instalador del programa Python.

2. Data Scraping

En este capítulo se va a proceder a describir la técnica de *web scraping*, así como sus técnicas, herramientas e impedimentos. Para comenzar vamos a definir el concepto de *scraping.*

Dar una descripción de lo que es *scraping* resulta complicado. Por ello, quizá la forma más sencilla de describir qué es el *data scraping* es partiendo de la propia traducción del término, el cual significa "raspado de datos". Siendo éste el objetivo fundamental que persigue el *scraping*, el de raspar o arañar datos para que posteriormente, puedan ser usados con otra finalidad diferente.

Existen dos vertientes diferentes relacionadas con el *scraping* de datos. Por un lado, el *screen scraping*, esta técnica se encarga de tomar una presentación de los datos y mediante ingeniería inversa obtener los datos que dieron lugar a esa presentación. Por otro lado, existe la técnica del *web scraping*, la cual se encarga de obtener los datos

mediante el HTML *(HyperText Marckup Language)* de una página y convertirlo en datos que puedan ser almacenados. Este proyecto se va a especializar en la técnica del *web scraping.*

2.1.¿Qué es web scraping?

Como ya se ha indicado, el *web scraping* se encarga de obtener los datos mediante un procesamiento del código HTML que forma la página. En este sentido, cada vez que hemos realizado la acción de *"copy&paste"* por diferentes páginas de la web para obtener datos, y que posteriormente hemos utilizado para otra actividad diferente, lo que se ha realizado ha sido un *"scrapeo"* de datos en la web. Pero esta práctica puede resultar muy costosa al tener que emplear mucho tiempo en el caso de querer obtener muchos datos de diferentes páginas, y posteriormente, organizar y estructurar dichos datos para poder ser utilizados con otra finalidad.

Por ello, lo que pretende permitir la técnica del *web scraping,* es realizar la acción de *"Copy&Paste"* de un modo más eficiente, mediante el uso de la programación. De modo que, programando se pueda obtener y organizar la información residente en las diferentes páginas.

La práctica del *web scraping* es bastante útil en diferentes actividades, uno de los ejemplos más notorios es en el periodismo de datos. Al permitir acceder, recopilar y organizar datos para la realización de reportajes de investigación, así como realizar cruce de datos con otra información. Por ello, en los últimos años se están impartiendo cursos o talleres de periodismo de datos orientados al *scraping* como los que imparte la página web 'Periodismo de Datos'.

En un principio, el realizar *web scraping* significa tener conocimientos de programación, por ello, en los últimos años se han desarrollado multitud de herramientas que permiten la realización del *web scraping* de modo más sencillo mediante el uso de interfaces, que mediante simples *"clicks"* permiten la obtención de la información rápidamente. Aunque, también es cierto, que este tipo de programas están limitados.

2.2.Funcionamiento del web scraping

El funcionamiento del *web scraping* consiste en una simulación de la navegación, que una persona realizaría, ya sea implementado a bajo nivel, como en HTTP (Hypertext Transfer Protocol), o incluido en ciertos navegadores web. Ésta simulación se realiza mediante programas denominados *"bots"*, que crean una clonación del *cliqueo*, la lectura y el *"copy-paste",* automatizando la tarea de búsqueda y de recolección de datos.

De modo que, *web scraping* se centra en la transformación del contenido no estructurado, por lo general en formato HTML, en datos estructurados que pueden ser almacenados y analizados.

2.3. Técnicas usadas para el web scraping

Existen una gran variedad de técnicas que permiten el *web scraping*:

- Recolección manual: es quizás el más conocido y sencillo, consiste simplemente en copiar la información que nos interesa de un sitio web y pegarla. Esta tarea puede resultar un trabajo cansado y tedioso ya que si, en vez de tener que realizarlo en una página, debemos de hacerlo en miles o millones de páginas web sería imposible. A pesar de éste gran inconveniente, a veces, este método es el único que podemos usar cuando un sitio web tiene barreras para prevenir que ciertos softwares puedan realizar tareas automáticas de extracción de datos.
- Expresiones regulares: son una secuencia de caracteres que forman un patrón de búsqueda. Normalmente es usado para la búsqueda o reconocimiento de cadenas de caracteres. De modo que, utilizando motores de búsqueda se puede conseguir la búsqueda de expresiones regulares en una página web, este método resulta sencillo y potente, aunque laborioso y lento. No se recomienda su uso para procesar formatos HTML.
- Protocolo HTTP: se obtienen páginas web mediante peticiones HTTP a un servidor remoto mediante sockets. De modo que, se consigue la página web completa con todos los datos que ésta contiene, y que posteriormente deberán ser almacenados y clasificados.
- Algoritmos de minería de datos: la minería de datos se encarga de extraer información de un conjunto de datos, transformándola en una estructura comprensible para su posterior uso.
  Muchos sitios web tienen páginas web similares, ya que para agilizar la creación hacen uso de plantillas o scripts. Entonces, la minería de datos se encarga de detectar la plantilla y extraer los datos.
- Parsers de HTML: mediante el uso de lenguajes de programación se procesan documentos HTML, recuperando y transformando el contenido. Algunos ejemplos de lenguajes que permiten realizar este procesamiento son Xquery o HTQL (HyperText Query Language).
- Reconocimiento de web semántica: algunas páginas contienen metadatos o información semántica, como anotaciones o comentarios. Esta 'metainformación' puede ser usada para extraer la información deseada. Las anotaciones pueden contenerse en la misma página, siendo de utilidad cuando procesamos el DOM (Document Object Model) del documento; o pueden estar en una capa semántica, que se encuentra almacenada de forma separada, pudiendo obtener estos esquemas antes de analizar los documentos.
- Aplicaciones para *web scraping*: existe una gran variedad de aplicaciones disponibles que permiten obtener información. Para conseguirlo, deben ser capaces de conocer la estructura de una página, o permitir al usuario seleccionar los campos que son de su interés en un documento. Ésta es la solución más recomendable, ya que dispone de diferentes algoritmos que permiten la recolección de la información.

2.4. Herramientas para el web scraping

Existen multitud de herramientas destinadas a la obtención de datos, cada herramienta se encuentra escrita en un lenguaje de programación, como puede ser Python, Ruby, PHP… En éste apartado se va a proceder a explicar algunas de ellas, haciendo una diferenciación según el usuario final a quién van dirigidas:

Usuarios sin conocimientos de programación: las aplicaciones destinadas a este tipo de usuarios se caracterizan por tratarse de interfaces sencillas, que mediante diferentes *"clicks"* el usuario puede obtener la información y en las cuales se evita tener que programar. También es cierto, que están bastante más limitadas en capacidad de obtener datos, que aquellas que están destinadas a usuarios con conocimientos de programación.

- Web HTTrack: [1]es un software libre que permite descargar un sitio web a un directorio local construyendo todos los directorios, obteniendo el HTML, imágenes y otros ficheros desde el ordenador a la computadora. Simplemente se abre la página en el navegador "espejo" y se navega de enlace en enlace como si estuviera en la página web.
  También permite reanudar descargas interrumpidas actualizando el sitio reflejado existente. Además, es configurable y contiene un sistema de ayuda integrado.
- Mozenda[2]: permite extraer información e imágenes de cualquier sitio web para ello hace uso de una interfaz sencilla que pueden crear y editar los agentes encargados de obtener la información. Además, contiene una consola web desde donde se puede programar y ejecutar los agentes.
- WebHarvy[3]: herramienta de *web scraping* que mediante simples *"clicks"* sobre la interfaz, permite la extracción de información de forma sencilla y cómoda. Es capaz de identificar patrones en distintas páginas, identificar por palabras claves y obtener el resultado. También, permite almacenar los datos en multitud de formatos como SQL (Structures Query Language), CSV (Comma-Separeted Values), TSV (Tab Separeted Values), XML (eXtensible Markup Languages), compatibles con JSON (JavaScript Object Notation).
- Visual Web Ripper:[4] dispone de un entorno gráfico y sus plantillas permite realizar un proyecto en poco tiempo y de manera sencilla. También, permite llegar a un nivel de programación bajo para soluciones más específicas o complejas. Permite exportar los datos a diferentes formatos (CSV, Excel, Base de datos,...) A su vez, se puede planificar y monitorizar con facilidad.

Usuarios con conocimientos de programación: las aplicaciones destinadas a este colectivo se caracterizan por permitir al usuario programar los datos que desea obtener de las páginas web.

[1] http://www.httrack.com/

[2] http://www.mozenda.com/web-scraping-software

[3] https://www.webharvy.com/index.html

[4] http://www.visualwebripper.com/

- Scraper.rb: programa escrito en Ruby que navega y recolecta la información, para posteriormente almacenarla y ordenarla en formato XML. Para realizar dicha extracción el programa hace uso fundamentalmente de dos librerías denominadas Watir y Nokogiri.
  Watir permite controlar el comportamiento del navegador, de este modo, el programa es capaz de abrir la página y de navegar por las secciones de ella ejecutando *"clicks"* en los menús de la página que han sido definidos en un archivo denominado rutasMenu.xml. En el caso de modificarse la estructura de la página a la que se va a realizar el scraping, sólo hay que modificar dicho archivo sin tener que modificar el código.
  La librería Nokogiri permite, apoyándose en la librería Watir, recorrer la página y obtener la información recogiéndola en formato XML.
- ScraperWiki[5]: en sus orígenes permitía escribir código en Python, Ruby o PHP y compartir el código con el resto de usuarios, no disponía de GUI (Graphical User Interface), por lo que era necesario tener altos conocimientos de programación.
  Actualmente, permite la realización de scraping mediante el uso de una interfaz sencilla que posibilita programar la extracción de datos de páginas web, ordenando y almacenando la información en tablas que posteriormente pueden ser descargadas en formato CSV o Excel. También pueden realizarse consultas SQL a los datos almacenados, realizar gráficos y compartir los datos obtenidos. Además, se pueden cargar hojas de cálculo en formato CSV o excel para realizar comparativas con los datos obtenidos del scraping.
  Esta herramienta permite programar el proceso de scraping en diferentes lenguajes (C, Clojure, Coffe Script, Lua, Node.js, Perl, PHP, Phyton, R, Ruby y Shell).
  A su vez, también tiene otras opciones que permiten extraer tablas de ficheros PDF, excel o de páginas web de modo sencillo y obtener tweets.
  También proporciona una librería que puede incorporarse a Python permitiendo *"scrapear"* los datos de forma local.

2.5. Impedimentos para realizar web scraping.

Existen diferentes impedimentos que impiden o dificultan la función de *web scraping*, todas ellas las vamos a dividir en dos grupos; por un lado aquellos factores que dificultan la realización de *web scraping*, y por otro, las barreras legales que a lo largo de los años se han desarrollado para evitar que se pueda realizar *web scraping*.

Factores que dificultan el uso de aplicaciones para el web scraping

Existen diversos factores que dificultan la obtención de los datos mediante el uso de herramientas destinadas al *web scraping*, estos factores pueden hacer que realizar la obtención sea más complicado o incluso llegando a impedir que se realice. Algunos de estos factores son:

[5] https://scraperwiki.com/

Códigos HTML mal estructurados, por ejemplo aquellas páginas que fueron creadas hace tiempo y no se han adaptado, de modo que su estructura es muy simple y no es posible extraer de su etiquetado HTML información estructurada. En este tipo de páginas, podremos obtener información pero sin clasificar, de modo que su obtención implica un posterior procesamiento haciendo la práctica más complicada o incluso impidiendo obtener los datos necesarios en el caso de grandes volúmenes de datos.

Aquellas páginas que contiene sistemas de autenticación como códigos y paywalls de CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart). Resulta evidente que conseguir la información mediante la programación de un *scraper* es imposible, ya que para acceder a la página es necesario realizar una autenticación de CAPTCHA que es totalmente diferente en cada petición.

Bloqueo al acceso masivo por los administradores de los servidores. Al programar un *scraper* se realizan muchas peticiones a un servidor, el cual puede detectarlo como un ataque de denegación de servicio y bloquear nuestra IP.

Sistemas que usan cookies para realizar un seguimiento de lo que el usuario realiza.

Barreras legales que limitan el web scraping

Además, existen otras limitaciones relacionadas con las barreras legales, ya que, algunos países reconocen los derechos de bases de datos limitando el derecho a reutilizar la información que se obtiene de lugares ya publicados. De hecho, algunas páginas se protegen del *scraping* declarando en sus condiciones legales la prohibición de realizar *scraping* sobre la página, como en el caso de una autoescuela la cual en una de sus cláusulas legales dice textualmente *"Entre los usos comerciales prohibidos se incluyen, sin limitación: 1.La reventa o redistribción del sitio web, sus contenidos y/o su servicio a través de otro sitio wen. En particular, técnicas de "web scraping" para acceder a los contenidos del sitio web*". [6]

Lo que es cierto, es que no existen leyes que definan el alcance y la legalidad para regular el proceso de *scraping*; debido a esto, se han diseñados diferentes herramientas que permiten detectar y prever el *scraping* en sitios web como por ejemplo *Application Manager* ™ *Security* (ASM), el cual se encarga de proveer diferentes métodos para hacer frente al *scraping*.

En los últimos años, se han producido varios casos de denuncia que distintas compañías han interpuesto contra otras páginas por el hecho de haber realizado *scraping* en sus páginas web. Un ejemplo de esto, es el que involucró a American Airlines contra una empresa llamada FareChase, que se dedicaba a vender un software que permitía comparar las tarifas de los vuelos. El resultado fue que la sentencia dio la razón a American Airlines, impidiendo a la empresa FareChase vender el software si el sitio de American Airlines estaba incluido, al considerar que entraba en los servidores de la compañía sin permiso de ésta.

No existe una idea uniforme sobre la legalidad o ilegalidad que supone la realización de *web scraping*, pero sí podemos extraer la idea general de que en éstas sentencias se está tendiendo más a la protección del contenido propietario en sitios web comerciales.

[6] http://www.clot.es/Legal.aspx

2.6. Ejemplos de proyectos

Se han realizado multitud de proyectos con información obtenida mediante el *web scraping*. Muchos de ellos, han creado páginas que realizan comparativas de precios en hoteles, vuelos, supermercados o con otros fines. Uno de estos proyectos es el denominado "Transparencia en las cuentas públicas"[7], creado por Miguel Fiandor, mediante el uso de la herramienta scraperwiki. El fin de este proyecto es el de mejorar y aumentar el conocimiento sobre las cuentas públicas, y así incentivar una mejor gestión de las mismas. La página principal se muestra en la Ilustración 4.

Ilustración 4: 'Trasparencia de Cuentas Públicas'

Otro ejemplo bastante notorio es "El Indultómetro"[8], cuya página principal se muestra en la Ilustración 5, proyecto desarrollado por la organización sin ánimo de lucro CiViO, en el cual muestran los datos del BOE referentes a los indultos que se han concedido en España desde 1996. De modo que los usuarios puedan investigar y conocer dichos datos más sencillamente.

Ilustración 5: 'El Indultómetro'

[7] http://transparenciadecuentaspublicas.es/

[8] http://www.elindultometro.es/

3. Análisis del sistema

En éste capítulo vamos a describir el dominio del problema, así como los requisitos del proyecto y modelos conceptuales que serán los casos de uso y el modelo de entidad-relación.

La página web sobre la que se ha decidido realizar el proceso de *web scraping* ha sido la del Boletín Oficial de Castilla y León (BOCyL). El motivo por el que ésta página ha sido la escogida para la realización del proyecto, se basa en que desde el 1 de enero de 2010 la Comunidad de Castilla y León decidió hacer una edición electrónica del Boletín Oficial de Castilla y León que permitiera un acceso universal, público y gratuito, pero simplemente se limita a mostrar la información de forma estática sin permitir que se pueda reutilizar. Por ello, se quiere dar una mayor usabilidad a los datos mediante la obtención de los mismos, y posteriormente con la creación de archivos Open Data, para que puedan ser utilizados en cualquier otro proyecto que lo requiera.

3.1. BOCyL

Como el proyecto se ha querido desarrollar sobre el Boletín Oficial de Castilla y León (BOCyL), se va a proceder a describir y documentar qué es el BOCyL, así como describir que contenido y estructura lo forma.

El Boletín Oficial de Castilla y León (BOCyL) tal y como lo describe su página web es "…el periódico oficial de la Comunidad de Castilla y León, a través del cual se da publicidad a los documentos que deben ser objeto de publicación oficial, de acuerdo con el ordenamiento jurídico." [9]. El objetivo que dicen perseguir es permitir un acceso universal, público y gratuito a la información. Teniendo la publicación pleno valor jurídico, como se establece según el decreto 61/2009, de 24 de septiembre.

El boletín deberá ser publicado de lunes a viernes, a excepción de los días declarados inhábiles por la Comunidad de Castilla y León. Excepcionalmente se podrá publicar cualquier otro día en el caso de que ocurran circunstancias que lo hagan necesario. El organismo que se encarga de editarlo es la Consejería de la Presidencia de Castilla y León, mediante el servicio de Secretaría del Boletín Oficial de Castilla y León, que se encuentra adscrito a la Secretaría General.

Contenido

En el BOCyL se podrán publicar las leyes, disposiciones con rango de ley y disposiciones administrativas con carácter general. Así como, los actos de las instituciones y administraciones públicas cuando sean publicadas por el ordenamiento jurídico; los actos de particulares que legalmente deban ser publicados; y cualquier otro documento cuya publicación sea exigida por orden judicial.

Estructura

Cada boletín se compone de diferentes documentos que estarán ordenados mediante diferentes secciones que a su vez estarán divididos en subsecciones:

SECCIÓN I- Comunidad de Castilla y León. Esta sección se subdivide en:

- *Disposiciones generales*, donde se incluirán las Leyes Orgánicas de reforma del Estatuto de Autonomía de Castilla y León, las leyes de las Cortes de Castilla y León, disposiciones con rango de ley y disposiciones administrativas de carácter general de la Administración de la Comunidad de Castilla y León.
- *Autoridades y personal*, incluirá resoluciones de personal relativas a autoridades y personal al servicio de la Comunidad de Castilla y León organizados en: Nombramientos, situaciones e incidencias; Oposiciones y concursos.
- *Otras disposiciones* que incluirán los actos administrativos de la Comunidad de Castilla y León que no corresponde a ninguna de las otras secciones.
- *Anuncios* incluyendo anuncios y notificaciones de la Comunidad de Castilla y León, dividido en: Contratación pública; notificaciones; y Otros anuncios oficiales.

[9] http://bocyl.jcyl.es/

SECCIÓN II- Estado y otras Comunidades Autónomas que se subdivide en:

- *Estado* que incluirá todas las disposiciones, actos y anuncios del Estado que deban publicarse en el BOCyL.
- *Otras Comunidades Autónomas* que incluirán todos los actos y anuncios de otras Comunidades y Ciudades Autónomas que deban publicarse en el BOCyL.

SECCIÓN III- Administración Local que se subdivide en:

- *Disposiciones generales* incluyendo las disposiciones administrativas de carácter general de las entidades locales de la Comunidad.
- *Autoridades y personal* incluyendo las resoluciones de autoridades y personal de las entidades locales de la Comunidad, organizados en: Nombramientos, situaciones e incidencias; Oposiciones y concursos.
- *Otras disposiciones*, incluyendo todos los actos administrativos de las entidades locales de Castilla y León y que no corresponde a ninguno de las otras subsecciones.
- *Anuncios* incluyendo anuncios y notificaciones de las entidades locales de la Comunidad de Castilla y León organizados en: Contratación pública; Notificaciones; y Otros anuncios oficiales.

SECCIÓN IV- Administración de Justicia donde se incluirán los edictos, notificaciones, requisitorias y anuncios de los Juzgados y Tribunales.

SECCIÓN V- Otros anuncios donde se incluirán los documentos que deban publicarse y no correspondan en ninguna de las otras secciones.

A su vez, cada documento estará incluido dentro de uno o varios apartados, que se identificarán por el o los organismos que emiten el documento. Posteriormente se mostrará una breve entrada que nos introducirá al texto del documento que seguidamente se mostrará y que será firmado por uno o varios responsables identificados por el nombre y puesto que ocupan, y donde se mencionará el lugar y la fecha de dicha firma.

Además, el documento podrá tener uno o varios anexos donde se mostrará información complementaria al texto principal.

3.2. Actores del sistema

Los actores especifican un rol jugado por un usuario o cualquier otro sistema que interactúa con el sujeto

ACT-01	Administrador
Versión	1.0 (22\07\2014)
Descripción	Este actor representa a la persona que se encarga del correcto funcionamiento del sistema.
Comentarios	Ninguno

Tabla 13: ACT-01

ACT-02	Sistema de Base de Datos
Versión	1.0 (22\07\2014)
Descripción	Este actor representa al sistema gestor de base de datos que se encarga de almacenar y proporcionar los datos.
Comentarios	Ninguno

Tabla 14: ACT-02

3.3. Requisitos

Requisitos funcionales

Un requisito funcional es aquel que describe una funcionalidad del sistema software o de sus componentes. En el caso de nuestro proyecto se van a realizar los siguientes requisitos funcionales:

- RQF 01.- Validar URL
- RQF 02.- Obtener los datos de un boletín.
- RQF 03.- Almacenar los datos en la base de datos.
- RQF 04.- Validar fecha
- RQF 05.- Obtener los datos en Open Data.

RQF01	Validar URL
Versión	1.0 (22/07/2014)
Dependencias	Ninguna
Descripción	El sistema deberá comprobar que en la URL suministrada por el administrador se publican contenidos válidos del BOCyL.
Importancia	Alta.
Prioridad	Alta.
Comentarios	Ninguno.

Tabla 15: RQF 01

RQF02	Obtener los datos de un boletín
Versión	1.0 (22/07/2014)
Dependencias	• RQF 01 Validar URL
Descripción	El sistema deberá recuperar los datos publicados en el ámbito de una URL previamente validada.
Importancia	Alta.
Prioridad	Alta.
Comentarios	Ninguno.

Tabla 16: RQF 02

RQF03	Almacenar los datos en la base de datos
Versión	1.0 (22/07/2014)
Dependencias	• RQF02.- Obtener los datos de un boletín
Descripción	El sistema deberá preservar los datos recuperados, a partir de una URL válida, de acuerdo con el esquema de la base de datos utilizada.
Importancia	Alta.
Prioridad	Alta.
Comentarios	Ninguno.

Tabla 17: RQF 03

RQF04	Validar fecha
Versión	1.0 (22/07/2014)
Dependencias	Ninguna
Descripción	El sistema deberá comprobar que existen los datos del boletín correspondiente a la fecha que se le indica.
Importancia	Alta.
Prioridad	Alta.
Comentarios	La fecha suministrada por el administrador deberá tener un formato de anio-mes-dia

Tabla 18: RQF 04

RQF05	Obtener los datos en Open Data
Versión	1.0 (22/07/2014)
Dependencias	• RQF03- Almacenar los datos en la base de datos. • RQF 04 Validar fecha
Descripción	El sistema deberá obtener la representación de los datos almacenados en la base de datos utilizando algún formato aceptado para la publicación de Open Data
Importancia	Alta.
Prioridad	Alta.
Comentarios	Ninguno.

Tabla 19: RQF 05

Requisitos no funcionales

Los requisitos no funcionales son aquellos que describen propiedades o cualidades que el producto debe tener. En este proyecto se van a desarrollar los siguientes requisitos no funcionales:

- RQNF 01.- Consistencia de los datos.
- RQNF 02.- Mantenibilidad.
- RQNF 03.- El formato de los ficheros Open Data será XML

RQNF01	Consistencia de los datos
Versión	1.0 (22/07/2014)
Descripción	El sistema deberá almacenar los datos del BOCyL de forma coherente evitando duplicados o falta de los mismos.
Importancia	Alta.
Prioridad	Alta.
Comentarios	Ninguno.

Tabla 20: RQNF 01

RQNF02	Mantenibilidad
Versión	1.0 (22/07/2014)
Descripción	El sistema deberá ser fácilmente mantenible, permitiendo adaptarse a cambios que realicen en el BOCyL en menos de un día.
Importancia	Alta.
Prioridad	Alta.
Comentarios	Ninguno.

Tabla 21: RQNF 02

RQNF05	El formato de los ficheros Open Data será XML
Versión	1.0 (22/07/2014)
Descripción	El sistema deberá crear los ficheros Open Data en el formato XML.
Importancia	Alta.
Prioridad	Alta.
Comentarios	Ninguno.

Tabla 22: RQNF 05

Requisitos de información

Los requisitos de información describen la información que debe almacenar y gestionar el sistema para dar soporte a los procesos de negocio.

De toda la información que se puede encontrar en los boletines y en los documentos que hemos descrito en el apartado de la estructura del boletín, se obtiene que existen siete tipos diferentes de información, los cuales van a describirse a continuación:

RI-1	SECCIÓN
Versión	01.0 (09/07/2014)
Dependencias	
Descripción	El sistema deberá almacenar la información correspondiente a las secciones, éstas son las diferentes secciones existentes en un boletín y que anteriormente se han descrito. La información que se va a almacenar en concreto es: • *Id: identificación único de tipo numérico.* • *Referencia: referencia única de tipo texto acorde con los patrones que define el BOCyL en su página.* • *Nombre: nombre descriptivo de la sección en la que se encuadra el documento.*
Importancia	Alta
Prioridad	Muy alta
Comentarios	Ninguno

Tabla 23: RI 01

RI-2	APARTADO
Versión	01.0 (09/07/2014)
Dependencias	
Descripción	El sistema deberá almacenar la información correspondiente a los apartados, los cuales son los organismos que emiten los diferentes documentos. En concreto almacenará: • *Id*: *identificación único de tipo numérico.* • *nombre_administración: nombre descriptivo del apartado al que pertenece el documento.*
Importancia	Alta
Prioridad	Muy alta
Comentarios	Ninguno

Tabla 24: RI 02

RI-3	DOCUMENTO
Versión	01.0 (09/07/2014)
Dependencias	
Descripción	El sistema deberá almacenar la información correspondiente a cada documento, se trata de todo el texto que contiene un documento. En concreto: • *id_documento: identificación único de tipo numérico.* • *Cabecera: texto que describe brevemente el contenido del documento.* • *Contenido: texto que describe en profundidad el documento.* • *url: enlace al documento de la página del BOCyL.* • *Fecha: fecha en la que se firma el documento.*
Importancia	Alta
Prioridad	Muy alta
Comentarios	Ninguno

Tabla 25: RI 03

RI-4	BOLETÍN
Versión	01.0 (09/07/2014)
Dependencias	
Descripción	El sistema deberá almacenar la información correspondiente al boletín, esta información será la que describa el boletín general que se va a almacenar. En concreto: • *Id: identificación único de tipo numérico.* • *url*: enlace del boletín de la página del BOCyL. • f*echa*: fecha en la que es publicado el boletín
Importancia	Alta
Prioridad	Muy alta
Comentarios	Ninguno

Tabla 26: RI 04

RI-5	ANEXO
Versión	01.0 (09/07/2014)
Dependencias	
Descripción	El sistema deberá almacenar la información correspondiente a los anexos, esta información será texto que describirá alguna característica adicional del documento. En concreto: • *Id: identificación único de tipo numérico.* • *Contenido: contenido del anexo perteneciente a un documento.*
Importancia	Alta
Prioridad	Muy alta
Comentarios	Ninguno

Tabla 27: RI 05

RI-6	RESPONSABLE
Versión	01.0 (09/07/2014)
Dependencias	
Descripción	El sistema deberá almacenar la información correspondiente a los responsables, esta información caracterizará a los encargados de firmar los documentos. En concreto: • *Id: identificación único de tipo numérico.* • *Nombre: nombre del responsable que firma el documento. Puede ser una persona o un organismo.* • *Puesto: puesto que ocupa la persona que firma el documento.*
Importancia	Alta
Prioridad	Muy alta
Comentarios	Ninguno

Tabla 28: RI 06

RI-7	MUNICIPIO
Versión	01.0 (09/07/2014)
Dependencias	
Descripción	El sistema deberá almacenar la información correspondiente a los municipios de Castilla y León. En concreto: • *Id: identificación único de tipo numérico.* • *Municipio: municipio al que pertenece un documento.*
Importancia	Alta
Prioridad	Muy alta
Comentarios	Ninguno

Tabla 29: RI 07

Matriz de trazabilidad

En la tabla 30 se muestra la matriz de trazabilidad entre los requisitos funcionales. En esta tabla se representan las relaciones entre requisitos, de este modo si se produce la modificación de un requisito funcional podemos saber de manera sencilla qué requisitos se verán afectados por la modificación.

	RQF 01	RQF 02	RQF 03	RQF 04	RQF 05
RQF 01		1	0	0	0
RQF 02			1	0	0
RQF 03				0	1
RQF 04					1
RQF 05					

Tabla 30: Matriz de interacción de requisitos

3.4. Casos de uso

Diagrama de casos de uso general

Los diagramas de casos de uso representan el comportamiento que el sistema debe presentar. El diagrama de casos de uso de nuestro sistema es el que se presenta en la Ilustración 3.

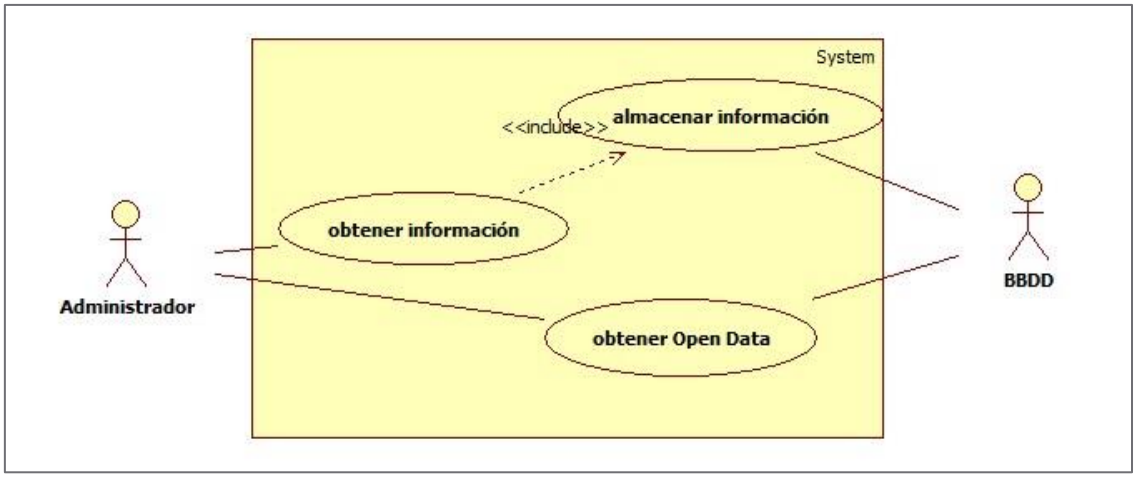

Ilustración 6: Diagrama de casos de uso

Caso de uso Obtener datos

UC01	Obtener información	
Versión	*1.0 (22/07/2014)*	
Dependencias	*Requisitos de los que depende:* • RQF01 Validar URL. • RQF 02 Obtener datos	
Precondición		
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando un usuario quiere obtener los datos de un boletín	
Secuencia Normal	Paso	Acción
	1	El administrador proporciona al sistema el enlace del boletín del que desea obtener la información.
	2	El sistema comprobará la validez del enlace.
	3	El sistema obtiene los datos y los clasifica de acuerdo al esquema de almacenamiento utilizado
Postcondición		
Excepciones	Paso	Acción
	2	La URL no es valida
		1 Mostrará un mensaje indicando que la URL no es válida
Frecuencia	1 vez/día	
Importancia	Alta	
Prioridad	Alta	
Comentarios	Ninguno	

Tabla 31: UC 01

Caso de uso Almacenar la información.

UC02	Almacenar la información	
Versión	*1.0 (22/07/2014)*	
Dependencias	Requisitos de los que depende: • RQF 03 Almacenar los datos en la base de datos.	
Precondición	El sistema deberá haber obtenido y clasificado la información acorde al esquema lógico de la base de datos.	
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando un administrador solicita obtener los datos de un boletín	
Secuencia Normal	Paso	Acción
	1	El administrador proporciona los datos sin procesar al sistema
	2	El sistema procesa la información, dividiéndola de forma que corresponda con la estructura de las tablas de la base de datos.
	3	El sistema proporciona los datos a la base de datos, que será la responsable del almacenamiento.
Postcondición	El sistema deberá haber almacenado los datos en la base de datos.	
Frecuencia	1 vez/día	
Importancia	Alta	
Prioridad	Alta	
Comentarios	Ninguno	

Tabla 32: UC02

Caso de uso Obtener datos en Open Data.

UC03	Obtener datos en Open Data	
Versión	*1.0 (22/07/2014)*	
Dependencias	Requisitos de los que depende: • RQF 04 Validar fecha • RQF 05 Obtener datos en Open Data	
Precondición	El sistema gestor de base de datos deberá haber almacenado en el caso de uso anterior los datos.	
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando un administrador solicita los datos en Open Data.	
Secuencia Normal	Paso	Acción
	1	El administrador proporciona al sistema la fecha del boletín del que desea crear el fichero XML.
	2	El sistema solicita los datos a la base de datos.
	3	La base de datos proporciona al sistema los datos correspondientes al boletín de la fecha.
	4	El sistema almacena en el fichero los datos con el nombre de la fecha.
Postcondición	El sistema deberá haber creado un fichero con los datos en formato XML.	
Excepción	Paso	Acción
	3	La base de datos no proporciona ningún valor al sistema, indicando que no existe en boletín solicitado
		1 El sistema muestra un mensaje indicando que no existe el boletín en la base de datos.
Frecuencia	1 vez/día	
Importancia	Alta	
Prioridad	Alta	
Comentarios	Ninguno	

Tabla 33: UC 03

3.5. Modelo de datos conceptual

Para validar los requisitos de información obtenidos, realizamos el diagrama de entidad-relación que obtendría el sistema. Comprobamos que con los requisitos de información que hemos recogido podríamos generar la base de datos y se almacenaría la información correctamente. El modelo de E-R se muestra en la Ilustración 7.

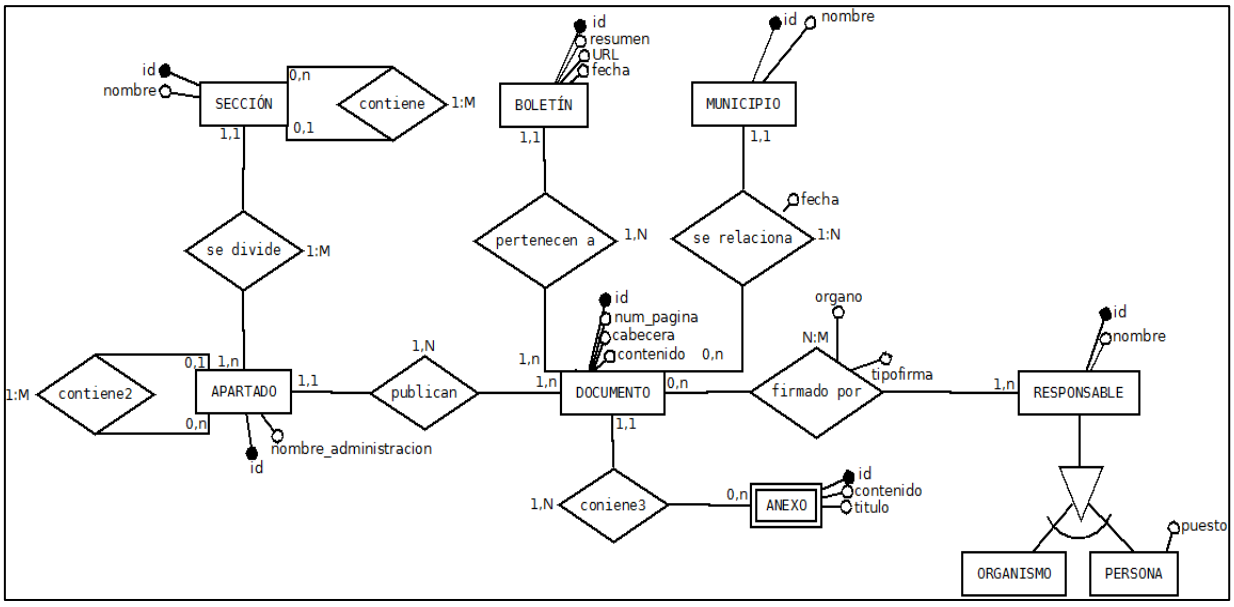

Ilustración 7: Modelo E-R

Las entidades del modelo conceptual corresponden con los requisitos de información anteriormente presentados, en cambio las relaciones entre las entidades son las siguientes:

- Contiene: relación existente entre una sección consigo misma, es la encargada de representar la jerarquía entre las secciones. La cardinalidad que tiene es de 1:N, ya que una sección puede contener una o varias secciones, pero una sección sólo puede ser contenida por máximo una sección.
- Se_divide: relación existente entre una sección y un apartado. La cardinalidad que tiene es de 1:N, ya que una sección puede dividirse en una o varias secciones, pero un apartado sólo puede ser dividido por máximo un apartado.
- Contiene2: relación existente entre un apartado consigo mismo, es la encargada de representar la jerarquía entre los apartados. La cardinalidad que tiene es de 1:N, ya que un apartado puede contener uno o varios apartados, pero un apartado sólo puede ser contenida por máximo un apartado.
- Publican: relación existente entre un apartado y un documento. La cardinalidad que tiene es de 1:N, ya que un apartado puede publicar uno o varios documentos, pero un documento sólo puede ser dividido por máximo un apartado.
- Pertenece_a: relación existente entre un boletín y un documento. La cardinalidad que tiene es de 1:N, ya que a un boletín le pueden pertenecer uno o varios documentos, pero un documento sólo puede pertenecer a máximo un boletín.
- Se_relaciona: relación existente entre un municipio y un documento, representa el lugar en el que se identifican los documentos. La cardinalidad que tiene es de 1:N, ya que un municipio puede relacionarse con uno o varios documentos, pero un documento sólo puede ser relacionado con máximo un documento. En la relación se identifica también la fecha en la que se produce la relación entre ambas entidades.
- Firmado_por: relación existente entre un documento y un responsable. La cardinalidad que tiene es de N:M, ya que un documento puede firmarlos muchos responsables, y un responsable puede firmar más de un documento. En la relación se identifica también el tipo de firma y el órgano asociado al tipo de firma.
- Contiene3: : relación existente entre un anexo y un documento, representa el anexo que va asociado al docuemnto. La cardinalidad que tiene es de 1:N, ya que a un documento le pueden pertenecer uno o varios anexos, pero un anexo sólo puede pertenecer a máximo un documento.

4. Diseño del Proyecto

En esta parte del documento se va a proceder a mostrar y explicar la arquitectura lógica y física que presenta el documento. Además, también se mostrará el modelo de datos lógico que representa la base de datos.

4.1. Arquitectura lógica

La arquitectura lógica expresa cuáles son los componentes lógicos que participan en el programa, y la relación entre ellos. En nuestro caso, la arquitectura lógica que usa en el programa es la que se muestra en la Ilustración 8.

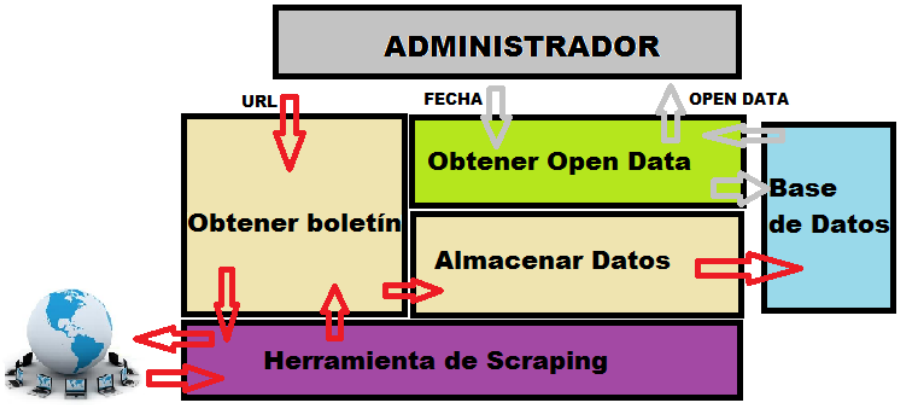

Ilustración 8: Arquitectura lógica

La base de la arquitectura lógica es la 'Herramienta de Scraping' y es la que fundamenta todo el proyecto.

Por encima de la 'Herramienta de Scraping' se encuentran dos componentes. El componente 'Obtener boletín' se encargará de obtener la información de un boletín que se le solicita, y el componente 'Almacenar Datos' se encargará de comunicarse con la base de datos para almacenar los datos que anteriormente se ha obtenido.

Por otro lado tenemos el componente 'Obtener Open Data' que se encargará de conectarse con la base de datos para obtener los datos de un boletín y crear los archivos Open Data.

De este modo, podemos observar que existen dos flujos de trabajo bien diferenciados. Por un lado los que involucran a los componentes 'Obtener boletín' y al componente 'Almacenar Datos', y por otro lado al que involucra al componente 'Obtener Open Data'. El primer flujo de trabajo seguiría los pasos de:

1. El administrador solicita un nuevo boletín, activando el componente 'Obtener boletín'
2. El componente 'Obtener boletín' se comunicará con la 'Herramienta de Scraping'.
3. La 'Herramienta de Scraping' se comunica con la web para recuperar los datos del BOCyL indicado por el administrador.
4. Cuando todos los datos se han recuperado se activa el componente 'Almacenar datos'.
5. El componente 'Almacenar datos' se comunica con la base de datos para que ésta almacene la información.

El segundo flujo de trabajo sería:

1. El administrador solicita un nuevo documento Open Data, activando el componente 'Obtener Open Data'.
2. El componente 'Obtener Open Data' se comunica con el componente 'Base de datos' para solicitar la información.
3. El componente 'Base de datos' devuelve la información requerida por el componente 'Obtener Open Data'.

El administrador se comunicará directamente con el sistema, en concreto con los componentes 'Obtener boletín' y 'Obtener Open Data'.

4.2. Arquitectura física

La arquitectura física expresa cuáles son los componentes físicos que participan en el proyecto, así como la relación existente entre ellos. La arquitectura física que utilizamos en esta plataforma es la que se muestra en la Ilustración 9, y se compone de tres capas:

- Capa de Presentación: encargada de interactuar con el usuario.
- Capa de Negocio: es la aplicación propiamente dicha y es el encargado de controlar y manejar la información residente en la base de datos. Esta capa es la que interactúa con internet, por ello debemos de realizar la comunicación mediante firewalls, para asegurar la comunicación.

  Capa de Datos: es la base de datos. Encargada del almacenamiento de los datos

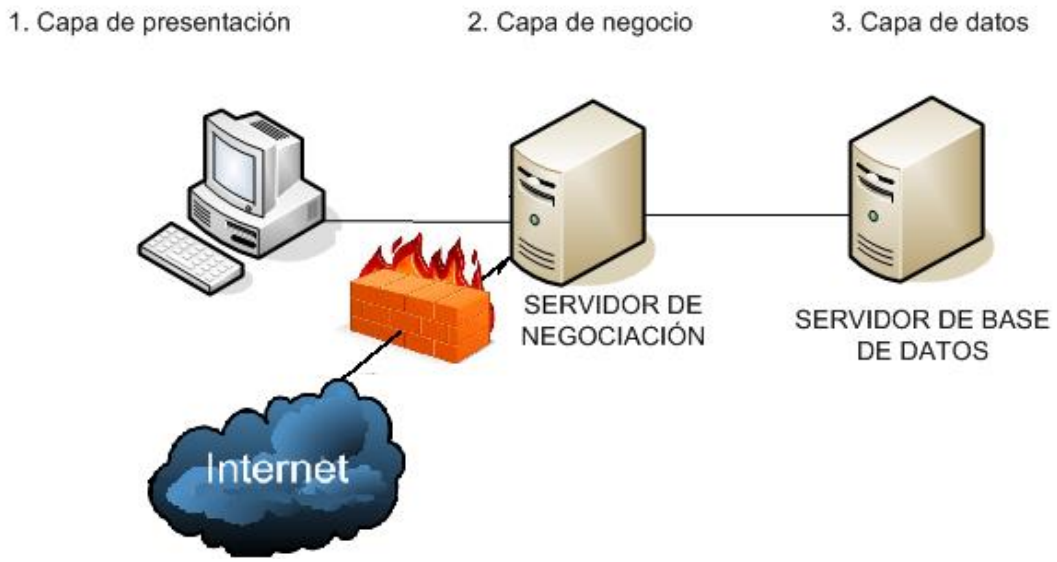

Ilustración 9: Arquitectura física

Esta arquitectura sería la que se desarrollaría una vez llevado el proyecto a producción. Actualmente, el sistema asume todas estas responsabilidades en una misma máquina, por lo que en el capítulo de implementación sólo se encargará de crear los componentes y serán ejecutados en un mismo ordenador.

4.3. Modelo de datos lógico

El modelo de datos lógico o modelo relacional que se representa en la Ilustración 10 se obtiene del modelo de datos conceptual, mediante una serie de pautas:

- Cada conjunto de entidades fuerte se representa con una tabla, cuyas columnas corresponden a los atributos de las entidades.

  Las entidades sección, apartado, documento, anexo, boletín y municipio, pasan a ser tablas cuyas columnas son los atributos que pertenecen a cada entidad.

- Relaciones “uno a varios” se toma el campo llave del extremo uno y se inserta en la tabla del extremo varios.

La relación *contiene*, se transforma añadiendo una columna a la tabla sección cuya clave foránea es el identificador de la tabla sección.

La relación *se_divide*, se transforma añadiendo una columna a la tabla apartado cuya clave foránea es el identificador de la tabla sección.

La relación *contiene2*, se transforma añadiendo una columna a la tabla apartado cuya clave foránea es el identificador de la tabla apartado.

La relación *publican*, se transforma añadiendo una columna a la tabla documento cuya clave foránea es el identificador de la tabla apartado.

La relación *pertenece_a*, se transforma añadiendo una columna a la tabla documento cuya clave foránea es el identificador de la tabla boletin.

La relación *se_relaciona*, se transforma añadiendo una columna a la tabla documento cuya clave foránea es el identificador de la tabla municipio.

La relación *contiene3*, se transforma añadiendo una columna a la tabla anexo cuya clave foránea es el identificador de la tabla documento.

- Relaciones "varios a varios" se representa con una tabla, la cual tiene una columna por cada atributo de las llaves primarias de los conjuntos de entidades a los que participan en la relación, más una o más columnas por cada atributo que fueron necesarios para describir la relación.

  La relación *firmado_por*, se transforma creando una nueva tabla con dos columnas cuyas claves foráneas son el identificador de la tabla documento y el identificador de la tabla responsable.

- En el caso de las transformaciones de generalizaciones se ha decidido que la entidad padre reciba todos los atributos de las entidades hijas, de este modo la entidad padre contiene todos los atributos de las entidades hijas.

  La jerarquía existente entre las entidades responsable, organismo y persona se transforma en una única tabla cuyas columnas serán los atributos de las tres tablas.

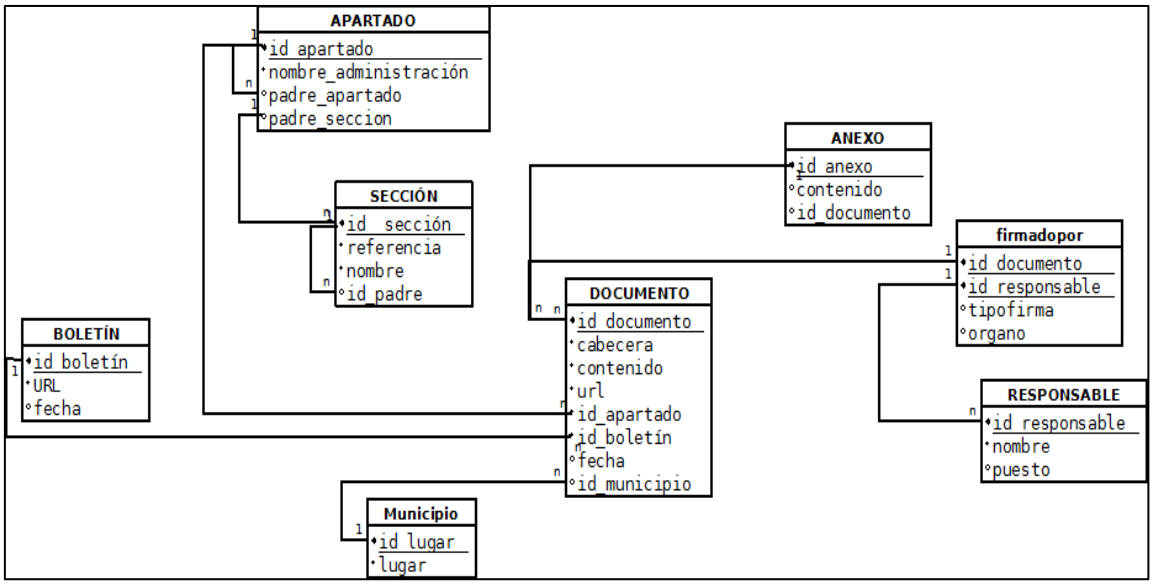

Ilustración 10: Modelo Relacional

Propiedades de las tablas

BOLETÍN, tendrá las siguientes columnas:

- Id_boletin, entero, no nulo, clave primaria.
- url: texto, no nulo.
- Fecha: fecha, no nulo.

SECCIÓN,

- Id_seccion: entero, no nulo, clave primaria.
- Referencia: texto, no nulo.
- Nombre: texto, no nulo.
- Id_padre: entero, no nulo, clave foránea (sección).

APARTADO,

- Id_apartado: entero, no nulo, clave primaria.
- Nombre_administracion: texto, no nulo.
- Padre:apartado: entero, no nulo, clave foránea (apartado).
- Padre_seccion: entero, no nulo, clave foránea (sección).

MUNICIPIO,

- Id_lugar: entero, no nulo, clave primaria.
- Lugar: texto, no nulo.

DOCUMENTO,

- Id_documento: entero, no nulo, clave primaria.
- Cabecera: texto, no nulo.
- Contenido: texto, no nulo.
- url: texto, no nulo.
- id_apartado: entero, no nulo, clave foránea (apartado).
- id_boletin entero, no nulo, clave foránea (boletin).
- fecha: fecha, no nulo.
- id_municipio: entero, no nulo, clave foránea (municipio).

RESPONSABLE,

- Id_responsable: entero, no nulo, clave primaria.
- Nombre: texto, no nulo.
- Puesto: texto, nulo.

FIRMADO_POR,

- Id_documento: entero, no nulo, clave primaria, clave ajena (documento).
- Id_responsable: entero, no nulo, clave primaria., clave ajena (responsable).
- Tipofirma: enumerado (normal, P.D., P.O., P.S., P.S.R. P.D.O.).
- Órgano: texto, nulo.

ANEXO,

- Id_anexo: entero, no nulo, clave primara.
- Contenido: texto, no nulo.
- Id_documento: entero, no nulo, clave ajena (documento).

5. Implementación del Proyecto

En este capítulo se va a proceder a describir la implementación del proyecto.

Antes de comenzar, vamos a describir las herramientas empleadas para realizar la implementación del proyecto. Una vez finalizado esta descripción, se va a proceder a presentar la implementación. Para ello, se va a dividir en varias fases que coinciden con las fases de desarrollo del proyecto: la primera analiza la estructura HTML del documento, la segunda obtiene la información y la tercera obtiene el *Open Data*.

5.1. Paso de la arquitectura lógica a la implementación

La implementación del proyecto se debe realizar basándose en la arquitectura lógica descrita en el capítulo 3. De modo que cada componente propio de la arquitectura lógica sea implementado y desarrolle las actividades que se le encomiendan.

El componente 'Herramienta de scraping', será el lenguaje de programación Python, junto con las librerías propias de este programa que permiten realizar la actividad de *scraping*. En un principio, se pretendía hacer uso de la librería ScraperWiki que la herramienta ScraperWiki tiene para realizar proyectos de *scraping* en local, pero tras realizar ciertas pruebas con ella, se decidió no hacer uso de ella, ya que no era suficiente para este proyecto. El problema que presentaba era, que cuando se solicitaba un tipo de etiqueta, extraía todas las etiquetas de ese tipo existentes en el *html*, lo que no permitía identificar detrás de qué tipo de etiqueta estaba en el *html*, impidiendo conocer a qué tipo de información pertenece. Este problema se presenta en el apartado de Metodología de la Fase I.

Las librerías que se utilizaron al final son:

- *urllib2:* encargada de obtener el código *hml* de la *url*.
- *lxml.etree:* encargada de realizar el procesamiento secuencial de la información.
- *BytesIO;* se encuentra en el paquete *io,* es la encargada de permitir tratar un *array* como un fichero binario.

Los componentes 'Obtener boletín' y 'Almacenar datos' se implementarán en un mismo script. El componente 'Obtener boletín' se encargará de recibir la url que el administrador le suministra y pasárselo al componente 'Herramienta de scraping' para que realice el proceso de *scraping*. Según el componente 'Herramienta de scraping' recopile toda la información se la pasará al componente 'Almacenar datos' que se encargará de crear las sentencias SQL de inserción, con los datos que le han suministrado, en función de la base de datos que utilizamos, y se conectará con el componente 'Base de datos' para que almacene la información de las consultas SQL. El componente 'Base de datos' será el gestor de base de datos MySQL. Para realizar sus funciones, el componente 'Almacenar datos' hará uso de dos librerías:

- *re*: encargada de permitir encontrar los patrones de las expresiones regulares.
- *MySQLdb:* encargada de conectarse con la base de datos para almacenar los datos.

Por otro lado, el componente 'Obtener Open Data' se implementará en un único script mediante el lenguaje de programación Python. Se encargará de recibir la fecha del boletín que el administrador le suministra. Creará las consultas SQL necesarias para obtener los datos, y se conectará con el componente 'Base de datos' para que recopile los datos y se los entregue al componente 'Obtener Open Data'. El componente 'Obtener Open Data', creará el archivo Open Data con los datos recibidos. Las librerías usadas para la realización de este script son:

- *MySQLdb*: encargada de conectarse con la base de datos para obtener los datos.
- *ElementTree* y *Element*: encontradas en el paquete xml.etree.ElementTree y encargadas de crear el árbol y los elementos que compondrán el árbol xml.

Herramientas empleadas para la implementación del Proyecto

Como ya se ha indicado, las herramientas usadas para la implementación son el lenguaje de programación Python, y como base de datos MySQL.

Python: lenguaje de scripting independiente de plataforma y orientado a objetos. Es un lenguaje interpretado, lo que significa que no necesita compilar el código fuente

para poder ejecutarlo. Al no ser necesario la compilación del código fuente permite una mayor rapidez de desarrollo, pero lo limita con una menor velocidad.

El motivo por el que se escogió Python para la implementación, se basó en las características que lo destacan, como son la simplicidad, versatilidad y rapidez de desarrollo. Además, tiene una gran cantidad de librerías integradas de forma estándar a las que se le pueden adjuntar otras librerías, que facilitan el trabajo del programador.

MySQL: sistema de administración de bases de datos para bases de datos relacionales. Así, MySQL no es más que una aplicación que permite gestionar archivos llamados de bases de datos. Es open source, lo que hace que su utilización sea gratuita e incluso se pueda modificar con total libertad, pudiendo descargar su código fuente.

El motivo por el que se escogió MySQL como la base de datos del proyecto, se basó en la facilidad de uso y en la velocidad que proporciona.

5.2. Fase 0: Análisis de la estructura del BOCyL.

Lo primero que ha de realizarse, antes de pasar a desarrollar el proceso de *scraping*, es analizar las páginas web que contienen la información que deseamos obtener.

Por un lado, tenemos la página web que contiene el listado de todos los documentos que pertenecen a un boletín, un ejemplo de esta página web se muestra en la Ilustración 11.

Ilustración 11: Página web de un boletín

Como se puede observar en la Ilustración 11, los documentos se muestran agrupados por la sección o secciones a la que pertenece cada uno, y de cada documento aparece un pequeño resumen y los enlaces que llevan al documento completo. El primer enlace nos lleva al documento en formato *pdf*, y otro en formato *html*. De toda la información que presenta la página web, lo que nos interesa almacenar es el enlace que nos lleva al documento en formato *html*.

Si analizamos el código correspondiente a la página web, vemos que los enlaces correspondientes a cada documento se encuentran en un listado cuya etiqueta es `<ul>` y el atributo que tiene asociado es `'class': 'descargaBoletin'`. En este listado hay dos elementos identificados con la etiqueta `<li>`. El primero es el enlace al documento en formato *pdf*, y el otro es el enlace al documento en *html*; el que deberemos almacenar es el segundo.

Por otro lado, tenemos la página que muestra un documento. Un ejemplo de esta página web es la que se muestra en la Ilustración 12, en la cual se puede apreciar seis divisiones bien marcadas.

Las secciones se encuentran en la parte superior del documento, y vienen referenciadas con letras respetando los diferentes patrones descritos en el capítulo de *Web Scraping*.

Los apartados se encuentran inmediatamente después, seguidos de una cabecera con un pequeño resumen sobre el documento. Este pequeño resumen lo denominaremos cabecera.

El texto será la parte principal del documento. Aquí podremos encontrarnos texto, listados, tablas. La parte del texto finalizará con la firma del responsable del documento que podrá ser un responsable o un organismo. La firma podrá ser de diferentes tipos: por delegación (P.D.), por autorización (P.A.), por orden (P.O.), P.S., P.S.R. y P.D.O.

Por último, también podrá tener un anexo que podrá contener texto, listados y tablas.

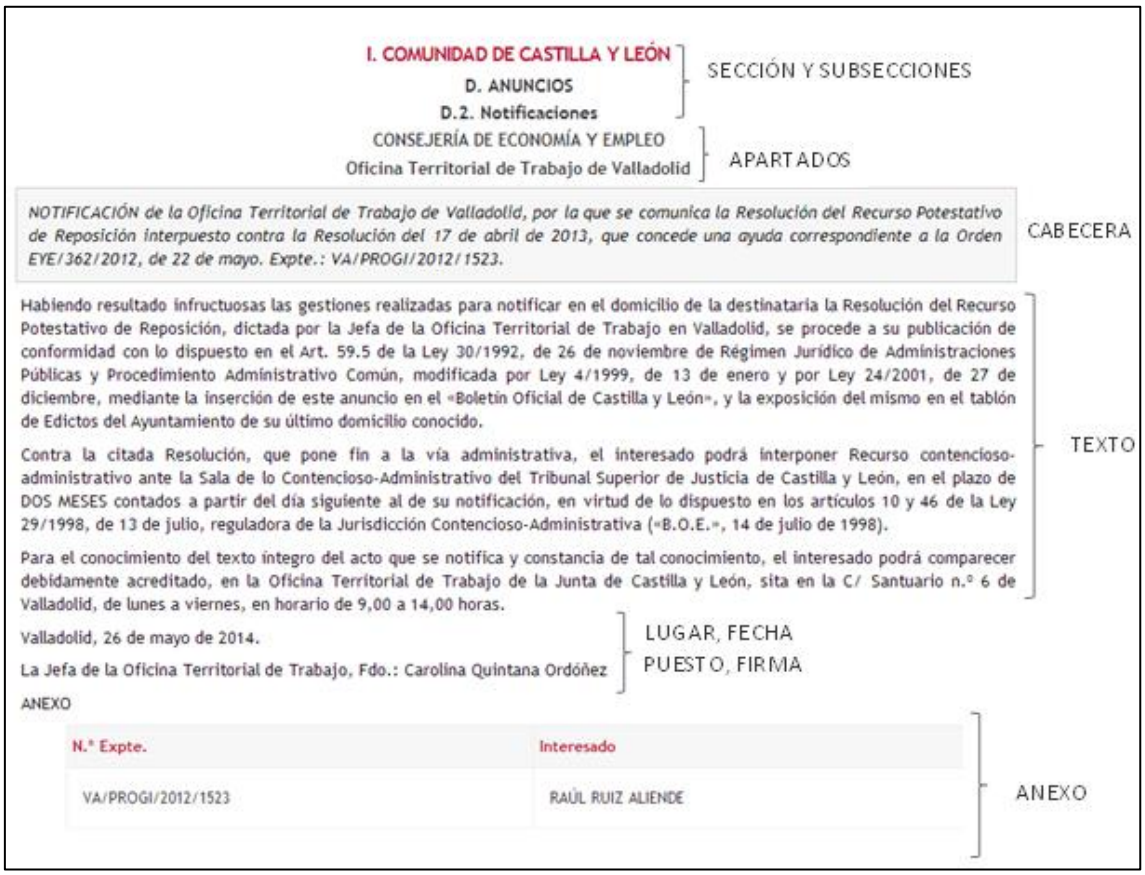

Ilustración 12: Página web de un documento

SECCIONES

Si analizamos el código html, se encuentra que las secciones se identifican mediante etiquetas `<h2>`, `<h3>` y `<h4>`. Estos distintos niveles de etiquetas, se utilizan para mostrar la jerarquía que existe dentro de las secciones. Siempre que aparezca una etiqueta `<h2>` será una sección, en cambio, las etiquetas `<h3>` y `<h4>` pueden indicar que son parte de los apartados o de encabezados que se encuentren dentro del texto. En el caso de pertenecer a alguno de éstos dos últimos casos se identifican mediante unos "attrib" específicos, por ello, se deberá primero comprobar que no pertenecen a ninguno de éstos casos, para asegurarnos que son secciones.

APARTADOS

Inmediatamente después, aparecen los apartados que también pueden tener una jerarquía entre ellos. Para indicar que son apartados se utilizan varias etiquetas que pueden ser `<h3>`, `<h4>`, `<h5>`, `<h6>` y `<p>`. En el caso de las etiquetas `<h3>` y `<h4>`, se indicará que forman parte de los apartados mediante el atributo `class='encabezado5'` o el atributo `class='encabezado6'`. En el caso de la etiqueta `<h5>`, los atributos que indican que se trata de un apartado puede ser `class='encabezado6'`, `class='aparienciaH7'` o ninguno. Las etiquetas `<h6>` y `<p>`, son referenciadas con el atributo `class='aparienciaH7'`. Y en el caso de la etiqueta `<h6>`, también puede pertenecer a los apartados en el caso de no tener atributo. Los apartados también puede ser referenciado mediante la etiqueta `<strong>`, que se encuentra dentro de un `<p>`, siempre que esta última tenga el atributo `class='aparieniaH7'`.

TEXTO

A continuación, aparece el texto mediante etiquetas `<p>`, aunque también pueden ser etiquetas de tipo `<h5>` y `<h6>` con el atributo `class='aparienciaTitulo'`, en el caso de ser títulos de secciones. La fecha y el autor de la firma del documento están referenciados con la etiqueta `<p>`, pero no están referenciados con ningún tipo de etiqueta especial que indique que son los datos de la firma. Por ello, en un principio, serán tratados como texto, y a la hora de almacenar los datos en la base de datos se debe procesar el texto para obtener los datos específicos de la firma que por regla general se encontrarán al final. En los responsables que firman los documentos se ha encontrado un fallo, y es que existen personas que unas veces su puesto aparece completo y otras sólo una parte del mismo. Un ejemplo de este problema ocurre con el responsable que se denomina 'Antonio Silván Rodríguez', unas veces aparece con el puesto 'El Consejero' y otras como 'El Consejero de Fomento y Medio Ambiente', a la vista de una persona resulta evidente que es el mismo puesto, pero el ordenador no es capaz de interpretarlo como lo mismo. Para ello, se decide crear un trigger sobre la tabla responsable, que será explicado más adelante.

ANEXO

Como ya se ha dicho, pueden aparecer dentro del documento, anexos. Los anexos pueden ser instanciados con las etiquetas `<h6>`, `<p>` y `<h5>`, con el atributo `class='aparienciaTitulo'`. También pueden ser etiquetas `<strong>` o `<p>`. Sus atributos coinciden con los que indican secciones dentro del texto por ello, cuando aparecen estas etiquetas y atributos se debe buscar la palabra 'ANEXO', y en el caso de que se encuentre sabremos que todo lo que viene detrás será parte del anexo.

TABLA

Pueden aparecer tablas dentro del texto o del anexo, las tablas se identifican con la etiqueta `<table>` y el atributo `class='presentDocument'`, de modo que cuando aparezca esta etiqueta y el atributo asociado a la etiqueta, se sabrá que hasta la etiqueta

de cierre `</table>` toda la información pertenecerá a la tabla. Por comodidad y sencillez a la hora de almacenar la información en la base de datos, se almacenará la tabla con las etiquetas, de esta forma, se definen perfectamente las filas y columnas de la tabla. Por ello, se deberá almacenar las etiquetas `<td>`, `<tr>` o `<th>` y la información que contiene cada una.

LISTADO

También pueden aparecer listados dentro del texto o del anexo que se identifica con la etiqueta `<li>`.

Para facilitar la compresión de las distintas posibilidades existentes a la hora de representar la información se va a presentar un esquema a modo resumen.

Sección.

- `<h2>`: siempre.
- `<h3>` y `<h4>`: cuando no tenga como atributo `class='encabezado5'`, `class='encabezado6'` o `class='aparienciaTitulo'`.

Apartado.

- `<h3>`, `<h4>`: cuando tenga como atributo `class='encabezado5'`o `class='encabezado6'`
- `<h5>`: cuando tenga como .atributo `class='encabezado6'`, `class='aparienciaH7'` o ninguno.
- `<h6>` : cuando tenga como atributo `class='aparienciaH7'` o ninguno.
- `<p>`: cuando tenga como atributo `class='aparienciaH7'`.
- `<strong>`: cuando se encuentra dentro de un `<p>` siempre que esta última tenga el atributo `class='aparieniaH7'`.

Texto.

- `<p>`: siempre que no forme parte del anexo.
- `<h5>`y `<h6>`: cuando tenga como atributo `class='aparienciaTitulo'`.

Fecha y responsable.

- `<p>`: al no tener ningún atributo que lo identifique lo almacenaremos como texto y después se procesará.

Anexo.

- `<h6>`, `<p>` y `<h5>`: con el atributo `class='aparienciaTitulo'` y que tenga la palabra anexo dentro.
- `<strong>`: cuando se encuentra dentro de un `<p>` siempre que contenga la palabra anexo.

Tablas.

Las tablas pueden ser encontradas dentro del texto o el anexo.

- `<table>`: cuando su atributo sea `class='presentDocument'`.

Listados.

Los listados pueden ser encontrados dentro del texto o del anexo.

- `<li>`: siempre.

5.3. Fase I: Extracción de datos del BOCyL.

Para poder describir esta sección vamos a apoyarnos en un ejemplo de un documento perteneciente a un boletín que se muestra en la Ilustración 12.

Metodología empleada

Tras estudiar y analizar las diferentes herramientas que se han descrito en el apartado 2.4, como ya se ha dicho se decidió, en un principio, realizar el proceso de *scraping* haciendo uso de la herramienta ScraperWiki que parecía bastante completa, pero tras realizar varias pruebas con ella se llegó a la conclusión de que era insuficiente para la envergadura de este proyecto. El principal fallo encontrado, es que al procesar el código HTML que correspondía a una página, por ejemplo, si quisiéramos obtener los párrafos con la etiqueta '<p>' del documento deberíamos realizarlo mediante el código:

```
[1] html= scraperwiki.scrape(url)
[2] root=lxml.html.fromstring(html)
[3] for p in root.cssselect('p'):
[4]    texto.append(p.text)
```

De éste modo lo que estamos haciendo es pasar por la librería "scraperwiki" la url donde se encuentra la página de la que queremos obtener la información; después la pasamos a la librería "lxml.html" que lo que hace es obtener el código html de la página en texto. Ahora le decimos que recorra la página en texto almacenando en un *array* que denominamos texto todos los párrafos `<p>`. Aquí encontramos el problema ya que almacena todos los `<p>` que existen en la página, de este modo si resultase que en el anexo también existiesen párrafos también nos lo almacenaría en el *array* del texto, por esta razón no es recomendable este tipo de herramienta para nuestro proyecto.

Una vez encontrado el problema que tenemos al usar 'Scraperwiki', y por lo que no era posible usarlo en este proyecto, se comenzó a indagar entre las librerías de Python buscando una que permitiera navegar por las etiquetas de nuestra página de forma secuencial. De este modo, se podría crear una herramienta propia que permitiera obtener los datos de forma eficiente. La librería en Python que permite hacer un recorrido secuencial por el código *html* es *lxml.etree*.

Para acceder a las etiquetas de forma secuencial, lo haremos mediante el siguiente fragmento de código que se encuentra en la aplicación:

```
[1] htmltostring=urllib2.urlopen(linkOriginal).read()
[2] var=BytesIO(htmltostring)
[3] for event,element in lxml.etree.iterparse(var,
            html=True, events=("start","end")):
```

Primero se pasa por la función 'urlopen' (perteneciente a la librería urllib2) la *url* de la página de la cual queremos realizar el procesado, de modo que la función se encarga de obtener el *html* de la página. Después, pasa el resultado por 'BytesIO' que permite tratar un *array* binario como un fichero y es almacenado en una variable.

Posteriormente, haciendo un bucle `for` por los `event` y `element` del código *html*, a través de la función `interparse` perteneciente a la librería 'lxml.etree', a la cual se le pasa como argumentos:

- `var` es la variable que contiene la página.

- `html=true` que permite que analice la página como un *html* y no como un *xml* que es como lo hace si no lo indicamos nada.
- `events=("start","end")` que son los eventos que debe analizar, es decir los de entrada y salida.

Los `element` tienen diferentes atributos de los cuales los necesarios para este proyecto son:

- `tag`: nos indica el tipo de etiqueta en la que estamos *(*`table`, `p`, `il`*...)*
- `attrib`: nos indica si tiene atributos la etiqueta y cual es.
- `text`: indica el texto que pertenece a cada etiqueta.

Así, dependiendo de la etiqueta, y del atributo al que pertenezca cada etiqueta, podremos saber si el texto pertenece a uno u otro de los diferentes tipos de información que podemos encontrar en el BOCyL.

Por lo cual, esta fase se encuentra dentro de diferentes tecnologías de las anteriormente descritas en el apartado 3 del capítulo 2. Por un lado, la de protocolos HTTP, la cual se hace uso cuando se pasa a la librería urllib2.urlopen el *link* de la página que se quiere procesar para obtener la información. Por otro, la tecnología de parsers HTML que será utilizada cuando se hace uso de la librería lxml.etree, ya que lo que se está haciendo es procesar el código HTML. Por último, también se usan expresiones regulares para conseguir la obtención de los firmantes de un documento. El uso de expresiones regulares se hará cuando el que firma no es una persona y no podemos reconocer la firma mediante 'Fdo:', o también, en el caso de que la firma no cumple la regla dentro del documento de encontrarse en las últimas posiciones del texto. De este modo mediante patrones se identifique dónde están las fechas de las firmas, ya que sabemos que después de las firmas aparecerán los responsables que firman el documento. Las expresiones regulares que serán usadas y que pueden ser encontradas en el proyecto son:

```
[1] patron1=re.compile('^\n.*, ([0-9]|[1-3][0-9]) de
      (enero|febrero|marzo|abril|mayo|junio|julio|agosto|septiembre|
      octubre|noviembre|diciembre) de 2014.')
[2] patron2=re.compile('^\n.*, a .* de
      (enero|bebrero|marzo|abril|mayo|junio|julio|agosto|septiembre|
      octubre|noviembre|diciembre) de dos mil catorce.')
```

Implementación

La implementación de esta parte del proyecto se encuentra en el fichero denominado '*scriptBBDD'* , en este script se van a realizar los componentes 'Obtener boletín' y 'Almacenar datos', de la arquitectura lógica .

Como ya se ha dicho, para poder obtener los datos y almacenarlos en la base de datos, debemos hacer uso de las librerías urllib2, lxml.etree, BytesIO, re y MySQLdb.

La implementación del código de *scraping*, se realiza mediante tres funciones principales que se van a ir explicando a continuación.

ObtenerURLs. La primera función que se invoca es `def obtenerURLs(linkOriginal)`. Esta función se encargará de obtener las *urls* de cada documento, correspondientes a un boletín. Para ello, se le pasará a la función el enlace perteneciente al boletín en cuestión.

Para obtener la información de la página de forma secuencial, lo haremos mediante el fragmento de código anteriormente explicado, en el apartado de metodología de este capítulo. Posteriormente, iremos tratando la información para obtener únicamente la necesaria.

Como ya hemos dicho, lo que nos interesa son los enlaces que nos llevan a los documentos en formato *html* y que son uno de los elementos del listado con el atributo `'descargaBoletin'`.

De modo que, cuando encuentra una etiqueta de listado (`<ul>`), cuyo atributo es `"class=descargaBoletin"` y el evento que lo acompaña es de inicio, es decir `event=="start"`, indica a una variable (`descargaBoletin`), que ha entrado en un listado de descarga, para ello pone la variable `descargaBoletin` a *True*. De este modo, indica que en las siguientes etiquetas va a estar el enlace a un documento del boletín en formato *html*, el cual deseamos almacenar. Los enlaces al documento en html, tienen la extensión .do, esta extensión indica que se trata de una aplicación web hecha con el lenguaje de programación Java. Las siguientes etiquetas, cuando encuentren un enlace con la extensión .do almacenarán este enlace en un *array*. Consiguiendo almacenar únicamente los enlaces a páginas *html* y no los *pdf*.

Cuando vuelve a encontrar una etiqueta `<ul>` cuyo atributo es `"class=descargaBoletin"` y el evento que lo acompaña es de fin, es decir, `event=="end"`, indica a la variable que se ha cerrado la etiqueta de tipo listado estableciendo la variable `descargaBoletin` a *False*.

A su vez, esta función también se encarga de almacenar los datos pertenecientes al boletín del que estamos obteniendo los datos.

```
[1] fechacompleta=linkOriginal.split('=')
[2] fechadividida=fechacompleta[1].split('/')
[3] insertBoletin="INSERT IGNORE INTO `boletin` (`url`,
      `fecha`)VALUES ('"+linkOriginal+"', '"+ fechadividida[2]+"-
      "+fechadividida[1]+"-"+ fechadividida[0]+"');\n"
[4] insertBoletin=insertBoletin.encode('utf-8')
[5] insertarBBDD(insertBoletin)
```

Los datos que necesitamos almacenar en la base de datos sobre el boletín, son la *url* y la fecha del boletín. En este momento lo único que tenemos es la *url* cuya estructura es:

`http://bocyl.jcyl.es/boletin.do?fechaBoletin=`*dd*/*mm*/*aa*

de modo que debemos de obtener de la *url*, los datos de la fecha. Mediante la función `Split`, propia de Python, se divide un *string* por una cadena de caracteres que se le indica y lo almacena en una variable de tipo *array*. En este caso, le decimos que divida la *url* por el carácter "=" y lo almacene en un *array*. Ahora este *array* tendrá dos datos, en la primera posición tendrá la *url* hasta el carácter que se encuentra antes del "=" (`http://bocyl.jcyl.es/boletin.do?fechaBoletin`) y en la segunda posición almacenará la fecha en formato dd/mm/aa (*dd*/*mm*/*aa*).

Para almacenar la fecha en la base de datos, debemos de pasarla al formato aa-mm-dd, por lo que necesitamos descomponer la fecha que tenemos almacenada en la segunda posición del *array*. Para ello, volvemos a hacer uso del método `Split`, pero ahora le indicamos que queremos que nos lo divida por el carácter "/". El resultado de dividir por el carácter se almacena en otro *array*.

Ahora se crea la consulta SQL de insertado, que permitirá almacenar los datos del boletín en la base de datos. Llamamos a la función `insertarBBDD` pasándola la sentencia SQL. La función `insertarBBDD` se encargará de almacenar la información, en caso de producirse un error, deshace todo lo que hubiera modificado mediante un `rollback`.

Una ver se realiza la inserción de los datos en la tabla boletín, la función retorna el *array* con los enlaces de cada uno de los documentos. .

Una vez finalizada esta función tendremos los datos de los enlaces en un *array* y los datos del boletín insertados en la base de datos. Ahora se pasa a realizar el proceso de obtención de datos de todos los documentos.

ObtenerInfo. La siguiente función que se invoca es `def obtenerInfo(url, linkOriginal)`. Esta función se encarga de obtener toda la información de cada documento. Para ello, se le debe pasar el *array* que contiene las *urls* de cada documento, que se han obtenido en la función anterior, y el enlace del boletín con el que se está trabajando. El enlace al boletín será utilizado únicamente, para poder crear las consultas SQL.

Para ir recopilando la información de todas las *urls*, es decir de todos los documentos, se realiza un bucle *for* por todas las posiciones del *array* que contiene las *urls*. Con el bucle for conseguimos recorre todas las *urls* y en cada *url* recopilar la información, y almacenarla en distintos *arrays* en función del tipo de información que sea. Se deben declarar cuatro *arrays* distintos (`sección`, `apartado`, `texto` y `anexo`), a su vez, también se declaran diferentes variables booleanas que nos indicarán en qué punto del documento estamos (`existeanexo`, `existetabla` y `existeapartado`). Las variables booleanas permiten saber en que *array* se debe almacena la información. También se declarará un *string* denominado "tabla", que almacenará la tabla, en caso de existir, con las etiquetas necesarias.

Para poder procesar el código de cada documento de forma secuencial, lo haremos mediante el fragmento de código anteriormente explicado, en el apartado de metodología de este mismo capítulo.

Primero comprueba si la variable 'existetabla' tiene el valor `True`, esto indicará que estamos dentro de una tabla en el documento. En caso afirmativo, se dispondrá a realizar varias comprobaciones:

- Si la etiqueta es `<tr>`, `<td>` o `<th>`, y si el evento tiene el valor `'start'`, es decir, inicio. En caso afirmativo comprueba que el texto que pertenece al elemento es diferente de *None* y si así es adjunta al *string* 'tabla' la etiqueta y el texto. En caso contrario, que el texto fuese *None* adjunta al *string* 'tabla' la etiqueta del elemento.

- Si la etiqueta es `<tr>`, `<td>` o `<th>` y si el evento tiene el valor `'end'`, es decir, fin. Adjunta al *string* 'tabla' la etiqueta del elemento con el símbolo de fin.
- Si la etiqueta fuese `<p>`, comprueba que el texto no sea *None* y si es así, adjunta al *string* 'tabla' el texto del elemento y limpia el elemento mediante la ejecución de `element.clear()` de modo que ignorará la etiqueta de cierre del elemento.
- Si la etiqueta fuese `<table>` y el elemento que lo desencadena de tipo `'end'`, es decir, de fin. Adjuntaría al *string* 'tabla' la etiqueta de fin de tabla (`</table>`) y comprobaría si la variable `existeanexo` tuviera valor True, es decir, que la tabla se encuentra dentro de un anexo entonces si es afirmativo almacena el *string* 'tabla' en el anexo. En caso contrario, lo almacenaría en el *string* 'texto'. Además, establece el valor de la variable `existetabla` a *False*.

El código perteneciente a esta parte es:

```
[1]  if (existetabla==True):
[2]      if ((element.tag=='tr' or element.tag=='td' or
           element.tag=='th') and event=="start"):
[3]          if element.text!=None:
[4]              tabla=tabla+'<'+element.tag+'>'+element.text
[5]          else:
[6]              tabla=tabla+'<'+element.tag+'>'
[7]      if ((element.tag=='tr' or element.tag=='td' or
            element.tag=='th') and event=="end"):
[8]              tabla=tabla+'</'+element.tag+'>'
[9]      if (element.tag=='p'):
[10]         if element.text!=None:
[11]             tabla=tabla+element.text
[12]             element.clear()
[13]     if (element.tag=='table' and event=="end"):
[14]         tabla=tabla+'</table>'
[15]         if (existeanexo==True):
[16]             anexo.append(tabla)
[17]         else:
[18]             texto.append(tabla)
[19]             existetabla=False
```

En caso de no tener valor True la variable *existetabla*, comprueba si la etiqueta del elemento es `<table>`, el atributo del elemento es `{'class': 'presentDocument'}` y el evento que lo desencadena es de tipo `'start'`, es decir, de tipo inicio. En este caso, establece el valor de la variable `existetabla` a *True* y en el *string* 'tabla' almacena la etiqueta.

```
[1]elif (element.tag=='table' and element.attrib=={'class':
     'presentDocument'} and event=="start"):
[2]      existetabla=True
[3]      tabla='<table>'
```

Después, comprueba si el texto del elemento es diferente de *None* y si la variable `existetabla` es igual a *False*.

```
[1] if (element.text!=None and existetabla==False):
```

En caso afirmativo realiza diferentes comprobaciones.

- Si la etiqueta del elemento es `<h2>` almacena en el *array* sección el texto.
- Si la etiqueta es `<h3>` o `<h4>`, comprueba si el atributo es `{'class': 'encabezado5'}` o `{'class': 'encabezado6'}`. En caso afirmativo, lo almacena en el *array* apartado. En caso contrario, comprueba si el atributo es `{'class': 'aparienciaTitulo'}` y si así es comprueba si la variable `existeanexo` tiene valor *True.* Si es así, almacena el texto en el *array* `anexo`; si no es así, busca en el texto la palabra ANEXO y si lo encuentra almacena el texto en el *array* `anexo` y cambia el valor de la variable `existeanexo` a *True.* En caso contrario, de no encontrar la palabra ANEXO, lo almacena en la variable texto.
  En el caso de que no tuviera como atributo `encabezado5, encabezado6` o `aparienciaTitulo` el texto será almacenado en el *array* `sección`.

```
[1]  if (element.tag=='h3') or (element.tag=='h4'):
[2]   if (element.attrib=={'class': 'encabezado5'} or
       element.attrib=={'class': 'encabezado6'}):
[3]         apartado.append(element.text)
[4]   elif (element.attrib=={'class': 'aparienciaTitulo'}):
[5]         if (existeanexo==True):
[6]               anexo.append(element.text)
[7]         elif (element.text.find('ANEXO')!=-1):
[8]               existeanexo=True
[9]               anexo.append(element.text)
[10]        else:
[11]              texto.append(element.text)
[12]  else:
[13]        seccion.append(element.text)
```

- Si las etiquetas son `<h5>` o `<h6>`, si el atributo es diferente a `{'class': 'aparienciaTitulo'}` almacena el texto en el *array* `apartado`. Si en el texto encuentra la palabra ANEXO establece el valor de la variable `existeanexo` a *True* y almacena el texto en el *array* `anexo`, en caso contrario comprueba si la variable `existeanexo` tiene valor *False*. De ser así, indicará que no estamos en la parte del anexo y lo almacenará en el *array* `texto`. En caso de no ser cierto, significará que nos encontramos dentro de un `anexo` y almacenará el `texto` en el *array* `anexo`.

```
[1] if (element.tag=='h5') or (element.tag=='h6'):
[2]   if (element.attrib!={'class':'aparienciaTitulo'}):
[3]         apartado.append(element.text)
[4]   elif (element.text.find('ANEXO')!=-1):
[5]         existeanexo=True
[6]         anexo.append(element.text)
[7]   elif (existeanexo ==False):
[8]         texto.append(element.text)
[9]   else:
[10]        anexo.append(element.text)
```

- Si la etiqueta es `<strong>` realiza unas comprobaciones. En caso de encontrar en el texto la palabra ANEXO establecerá la variable `existeanexo` a *True* y almacenará en el *array* `anexo` el texto. En caso de no ser así, comprueba si la variable `existeapartado` tiene valor *True,* indicando que ha sido establecido este valor en la etiqueta anterior que era de tipo `<p>`, de ser así, almacenará el valor en el *array* `apartado` y establecerá la variable `existeapartado` a *False* de nuevo. Si no es así, comprobará si la variable `existeanexo` tiene valor *True* indicando que nos encontramos dentro de un anexo y si es cierto almacenará el texto en el *array* `anexo`. En el caso de no ser cierto ninguna de las anteriores comprobará si la variable `existeanexo` tiene valor *False,* indicando que estamos en el texto, por lo que almacenará el texto en el *array* `texto`.

```
[1] if (element.tag=='strong'):
[2]   if (element.text.find('ANEXO')!=-1):
[3]         existeanexo=True
[4]         anexo.append(element.text)
[5]   elif existeapartado==True:
[6]         apartado.append(element.text)
[7]         existeapartado=False
[8]   elif (existeanexo==True):
[9]         anexo.append(element.text)
[10]  elif (existeanexo==False):
[11]        texto.append(element.text)
```

- En caso de ser una etiqueta `<p>` y que el atributo sea diferente a `{'class': 'errorGenerico'}`, ya que este atributo es utilizado para informar en un documento que el documento en formato *pdf* contiene imágenes. En este caso comprobará si el atributo es `{'class': 'aparienciaTitulo'}` y el texto contiene la palabra ANEXO, en caso de ser así establecerá la variable `existeanexo` a *True* y almacenará en el *array* `anexo` el texto. En caso contrario, comprobará si el atributo es `{'class': 'erroraparienciaH7'}` y de ser así si el texto es diferente de vacío almacena en el *array* `apartado` el texto; en caso contrario, cambia el valor de la variable `existeapartado` a *True* para que la siguiente etiqueta que será la de `<strong>` sepa que es del apartado. En caso contrario comprobará si el texto es ANEXO y de ser así cambiará el valor de la variable `existeanexo` a *True* y almacenará en el *array* `anexo` el texto, en caso contrario comprobará si la variable `existeanexo` tiene

valor *True*, y de ser así almacena en el *array* `anexo` el texto. En caso contrario, si el texto son dos saltos de línea los ignorará. Y en caso de no cumplirse ninguno de los anteriores casos, almacenará en el *array* `texto` el texto.

```
[1] if (element.tag=='p' and element.attrib!={'class':
     'errorGenerico'}):
[2]    if (element.attrib=={'class':'aparienciaTitulo'}
         and element.text.find('ANEXO')!=-1):
[3]         existeanexo=True
[4]         anexo.append(element.text)
[5]    elif(element.attrib=={'class': 'aparienciaH7'}):
[6]         if element.text.find('ANEXO')!=-1:
[7]               existeanexo=True
[8]               apartado.append(element.text)
[9]         else:
[10]              existeapartado=True
[11]        elif (element.text=='\nANEXO\n'):
[12]              existeanexo=True
[13]              anexo.append(element.text)
[14]        elif (existeanexo==True):
[15]              anexo.append(element.text)
[16]        elif (element.text=='\n\n'):
[17]              element.clear()
[18]   else:
[19]        texto.append(element.text)
```

- En el caso de ser la etiqueta `<li>` y su atributo diferente de `{'class': 'ultimo'}` comprobará si la variable `existeanexo` está con valor *True* y si es así, almacenará en el *array* `anexo` el texto. De no ser así almacenará en el *array* `texto` el texto.

```
[1] if(element.tag=='li' and element.attrib! =
     {'class':'ultimo' }):
[2]   if (existeanexo==True):
[3]         anexo.append(element.text)
[4]   else:
[5]         texto.append(element.text)
```

Para que resulte más intuitivo y a modo resumen se va a realizar un esquema con las distintas posibilidades existentes.

Variable existetabla=True

- Etiqueta tr, td, th y evento=inicio
  Almacena en tabla la etiqueta y el texto.
- Etiqueta tr, td, th y evento=fin
  Almacena en tabla la etiqueta.
- Etiqueta p
  Almacena en tabla el texto.
- Etiqueta table y evento= fin
  Almacena en la tabla la etiqueta de cierre.

- Variable existeanexo=True
  anexo.append(tabla)
- Sino
  texto.append(tabla)

Variable existetabla=false, etiqueta=table,atributo='class': 'presentDocument' y evento=inicio

- Etiqueta existetabla=True
- Almacena en tabla la etiqueta

Variable existetabla=false

- Etiqueta h2
  Almacena en apartado el valor
- Etiqueta h3, h4'
  - atributo 'class': 'encabezado5', 'class': 'encabezado6'
    Almacena en apartado el valor.
  - atributo 'class': 'aparienciaTitulo' y variable existeanexo=true
    Almacena en anexo el valor.
  - atributo 'class': 'aparienciaTitulo' y encuentra palabra ANEXO
    Almacena en anexo el valor.
    Cambia valor de existeanexo a true
  - atributo 'class': 'aparienciaTitulo' y variable existeanexo=false
    Almacena en texto el valor.
  - Ninguna de las otras opciones
    Almacena en seccion el valor.
- Etiqueta h5, h6
  - atributo'class':'aparienciaTitulo'
    almacena en apartado el valor
  - Encuentra la palabra ANEXO
    Almacena en anexo el valor.
    Cambia valor de existeanexo a true
  - Variable existeanexo =False
    Almacena en texto el valor.
  - Variable existeanexo =true
    Almacena en texto el valor
- Etiqueta strong
  - Encuentra la palabra ANEXO
    Almacena en anexo el valor.
    Cambia valor de existeanexo a true
  - Variable existeapartado =False
    Almacena en apartado el valor
  - Variable existeapartado=False y existeanexo=true
    Almacena en anexo el valor
  - Variable existeapartado=False y existeanexo=false
    Almacena en texto el valor
- Elemento p
  - Atributo 'class':'aparienciaTitulo' y encuentra palabra ANEXO
    Almacena en anexo el valor.
    Cambia valor de existeanexo a true
  - Atributo 'class': 'aparienciaH7 y encuentra palabra ANEXO
    Almacena en anexo el valor.
    Cambia valor de existeanexo a true
  - Atributo 'class': 'aparienciaH7 y no encuentra palabra ANEXO
    Cambia valor de existeanexo a true
  - Encuentra palabra ANEXO
    Cambia valor de existeanexo a true
    Almacena en anexo el valor.

    - Variable existeanexo==True
        Almacena en anexo el valor.
    - Ninguna de las otras opciones
        Almacena en texto el valor.
- Etiqueta li y atributo 'class':'ultimo'
    - Variable existeanexo==True
        Almacena en anexo el valor
    - Variable existeanexo==false
        Almacena en texto el valor

Para finalizar la ejecución llama a la función `almacenarBBDD` pasándole los *arrays* que contienen la información mediante la sentencia `almacenarBBDD(seccion, apartado, texto, anexo, linkOriginal, url)`. Esta función se encargará de almacenar los datos en la base de datos.

almacenarBBDD. Esta función se encarga de almacenar en la base de datos todos los datos que anteriormente hemos recopilado y almacenado en los *arrays*. Como parte de esta información no está procesada, se realizará un proceso antes de almacenarla en la base de datos.

SECCIONES

Lo primero que almacena son las secciones, esta información la encontramos en el *array* `seccion`. Como las secciones representan una jerarquía, debemos de insertar las secciones representando esta jerarquía. Para ello, en la tabla de sección de nuestra base de datos, tenemos una columna que será el `id_padre`, es decir, la sección que se encuentra encima en la jerarquía de la sección que queremos almacenar. En el caso de que la sección que queremos almacenar se encuentra en la primera posición del *array* `seccion` el `id_padre` será 0, es decir una sección precargada que no tiene ningún valor y que nos permite saber que la sección que pretendemos almacenar es la primera de la jerarquía. De este modo, para insertar las secciones, debemos de obtener la sección que se encuentra por encima de la sección que se pretende insertar (sección padre), y a su vez la sección padre de la sección que se encuentra encima de la sección que pretendemos almacenar. Por ejemplo, para el *array* seccionejemplo=[‘I. COMUNIDAD DE CASTILLA Y LEÓN’, ‘B. AUTORIDADES Y PERSONAL’, ‘B.1. NOMBRAMIENTOS, SITUACIONES E INCIDENCIAS’]

Cuando queremos almacenar el elemento de la posición 0, la sentencia de insertado sería:

"INSERT IGNORE INTO seccion(referencia,nombre,id_padre) SELECT 'I.','COMUNIDAD DE CASTILLA Y LEÓN',0"

En cambio, cuando almacenamos el elemento de la posición 1, la sentencia de insertado será:

"INSERT IGNORE INTO seccion(referencia,nombre,id_padre) SELECT 'I.','COMUNIDAD DE CASTILLA Y LEÓN' (SELECT id_seccion FROM seccion WHERE referencia='B.1.'and nombre= ' NOMBRAMIENTOS, SITUACIONES E INCIDENCIAS'' and id_padre=0)”

Y la sentencia de insertado para el elemento de la posición 2, sería:

"INSERT IGNORE INTO seccion(referencia,nombre,id_padre) SELECT 'I.','COMUNIDAD DE CASTILLA Y LEÓN',(SELECT id_seccion FROM seccion WHERE referencia='B.'and nombre= 'AUTORIDADES Y PERSONAL' and id_padre= (SELECT id_seccion FROM seccion WHERE referencia='B.1.'and nombre= ' NOMBRAMIENTOS, SITUACIONES E INCIDENCIAS'' and id_padre=0))"

Para poder realizar la inserción se recorre el *array* `sección` mediante un bucle for, y para cada iteración del bucle, llama a una función (`SQLseccion`) pasándola el *array* `sección`, y la posición en la que se encuentra. Esta función es la encargada de realizar la sentencia SQL de inserción respetando la jerarquía de las secciones. La función `SQLseccion` es:

```
[1] def SQLseccion(seccion, i):
[2]    parte1="INSERT IGNORE INTO seccion(referencia,nombre,id_padre)
       SELECT'"+seccionref(seccion,i)+"','"+seccion tex(seccion,i)+"',"
[3]    if i==0:
[4]         parte1=parte1+"0"
[5]    else:
[6]         for j in reversed(range(i)):
[7]              if j==0:
[8]                   parte1=parte1+"(SELECT id_seccion FROM seccion
                     WHERE referencia='"+seccionref(seccion,j)+"'and
                     nombre='"+seccion tex(seccion,j)+"' and
                     id_padre=0"
[9]              else:
[10]                  parte1=parte1+"(SELECT id_seccion FROM seccion
                     WHERE referencia='"+seccionref(seccion,j)+"'and
                     nombre='"+seccion tex(seccion,j)+"' and
                     id_padre="
[11]   for k in range(i):
[12]        parte1=parte1+")"
[13]   return parte1+";\n"
```

APARTADOS

Tras almacenar las secciones se procede a insertar los apartados, esta información la encontramos en el *array* `apartado`. Al igual que con las secciones, los aparatados también contienen una jerarquía que hay que respetar. La jerarquía de los apartados debe mantener también la jerarquía de las secciones. Para ello, en la tabla apartado existen dos columnas `padre_seccion` y `padre_apartado`. La columna `padre_seccion` indica la sección padre que contiene el apartado en el que nos encontramos, y la columna `padre_apartado` indica el apartado que se encuentra encima del apartado que estamos intentando insertar. En el caso de tener un `padre_apartado`, el valor de `padre_seccion` será 0; y en el caso de tener un `padre_seccion` el `padre_apartado` será 0.

Para ello, se recorre el *array* de apartados mediante un bucle *for*. En el caso de que la posición sea cero, es decir, el apartado está debajo de una sección en la jerarquía, crea la sentencia de insertado y llama a la función `SQLapartado(sección,apartado,i)`, que es la que se encarga de crear la jerarquía en

el insert, y establece la columna de `padre_apartado` a cero. En caso de no ser igual a 0 creará la sentencia de insert estableciendo la columna `padre_seccion=0` y llamando a la función `SQLapartado(sección,apartado,i)` para crear la jerarquía. La función que genera la parte de la jerarquía de la consulta es:

```
[1] def SQLapartado(seccion, apartado, i):
[2]   parte1=""
[3]   if i==0:
[4]         for j in reversed(range(len(seccion))):
[5]               if j==0:
[6]                     parte1=parte1+"(SELECT id_seccion FROM seccion
                         WHERE referencia='"+seccionref(seccion,j)+"'
                         and nombre='"+seccion tex(seccion,j)+"' and
                         id_padre=0"
[7]               else:
[8]                     parte1=parte1+ "(SELECT id_seccion FROM seccion
                        WHERE referencia='"+seccionref(seccion,j)+"'
                        and nombre='"+seccion tex(seccion,j)+"' and
                        id_padre="
[9]         for k in range (len (seccion)):
[10]              parte1=parte1+")"
[11]              parte1=parte1+",0"
[12]  else:
[13]        for j in reversed(range(i)):
[14]              if j==0:
[15]                    parte1=parte1+"(SELECT id_apartado FROM
                         apartado WHERE nombre_administracion='"+
                         apartado[j].strip().title()+"'and
                         padre_seccion="
[16]                    for p in reversed(range(len(seccion))):
[17]                          if p==0:
[18]                                parte1=parte1+"(SELECT id_seccion
                                    FROM seccion WHEREreferencia=
                                    '"+seccionref(seccion,p)+"' and
                                    nombre='"+seccion tex(seccion,p)+"'
                                    and id_padre=0"
[19]                          else:
[20]                                parte1=parte1+ "(SELECT id_seccion
                                    FROM sección WHEREreferencia='"
                                    +seccionref(seccion,p)+"' and
                                    nombre='"+seccion tex(seccion,p)+"'
                                    and id_padre="
[21]              else:
                        parte1=parte1+"(SELECT id_apartado FROM
                        apartado WHERE nombre_administracion='"+
                        apartado[j].strip().title()+"' and
                        padre_apartado="
[22]        for m in range (len (seccion)+i):
[23]              parte1=parte1+")"
[24]  return parte1
```

DOCUMENTO, RESPONSABLEY FIRMADOPOR

Después, debe almacenar la información de las tablas documento, responsable y firmadopor, toda esta información la encontramos en el *array* `texto`.

Del *array* `texto` debe obtener los responsables que firman el documento y la fecha en la que se produjo. El responsable suele aparecer por regla general al final del texto del documento pero algunas veces esta regla no se cumple y hay que buscarlo. Para encontrarlo se hace uso de las expresiones regulares que anteriormente se mencionaron, ya que los responsables aparecen inmediatamente después de la fecha.

Primero se intenta obtener el responsable mediante la búsqueda de la cadena 'Fdo.:' en la última posición del *array* `texto`. En caso de encontrar la cadena 'Fdo.:' en el último elemento del array texto, lo almacena en un array denominado responsable, y el elemento de la posición anterior en un string. Mediante un bucle while comprueba si el string contiene la cadena 'Fdo.:', en ese caso almacena la cadena en el array responsable y establece como valor del string el elemento de la posición anterior. Este procedimiento se realiza ya que puede existir más de un responsable.

Si en la última posición del array texto no ha encontrado la cadena 'Fdo.:', pasará a realizar la misma operación pero con el array anexo, ya que hay veces que los documentos no respetan los patrones establecidos y la firma del documento se encuentra en el anexo.

En el caso de no encontrar el responsable en las últimas posiciones del *array* `texto` y del *array* `anexo`. Se realizará un bucle *while* de forma inversa en el que se buscará la fecha y lugar mediante las dos expresiones regulares indicadas anteriormente, en caso de coincidir algún elemento del array texto con alguna de las dos expresiones regulares, el elemento pasará a ser el valor del *string* `fechafirma` y el elemento de la siguiente posición se almacenará en el *array* `responsable`.

En cualquier caso, cuando se encuentre un responsable o la fecha, el elemento del *array* que los contenga pasará a vacío para no tener la información duplicada.

Una vez hemos extraído del *array* `texto` los elementos que forman la firma del documento, se procede a obtener el texto en un único *string*. Mediante un bucle *for* que recorre todos los elementos del *array* `texto` obtenemos el texto de un documento en un único *string*.

En la variable `fechafirma` tenemos la fecha y el lugar en el que se firmó el documento. Para poder almacenar la información en la base de datos, debemos de obtener esta información de forma independiente, y en el caso de la fecha debemos de obtenerla en formato día, mes y año. El formato que tiene la cadena que contiene el lugar y la fecha es `[lugar], [dia] de [mes] de [año].`

Primero debemos obtener los valores de la fecha y la firma en dos variables independientes. Mediante la función `split`, se divide la variable `fechafirma` por el carácter ','. Después, debemos de obtener los valores de la fecha en tres variables diferentes, para poder almacenarlo en la base de datos correctamente. Para ello, se divide la parte de la fecha del mismo modo pero por el carácter 'de'.

Una vez, se tienen los datos de la fecha y lugar de firma separados, y los datos de la fecha separados en dia, mes y año. Debemos obtener la fecha en número para

poder almacenarla correctamente. Por regla general los días y los años están en número, y los meses en letra. Pero existen excepciones en las que los días se presentan en letra, o que los días vienen precedidos de la cadena ‘a ’. Para conseguir obtener los días y meses en número se realizan bucles *for* que busquen coincidencia del día y del mes con los existentes en un *array* de días y meses respectivamente. El *array* de días, tendrá todos los días de un mes, y el *array* de mese tendrá todos los meses de un año en letra.

Ahora ya tenemos la información que necesitamos sobre el documento, por ello vamos a realizar la inserción de la misma en la tabla documento. La información que necesitamos almacenar sobre el documento es la cabecera, contenido, url, idapartado, idboletin, fecha, y por ultimo idmunicipio. En el idapartado almacenaremos el identificador del último apartado perteneciente al boletín, respetando la jerarquía de apartados y secciones, del mismo modo que lo hacíamos cuando insertamos los apartados.

Después, se inserta la información de las tablas responsable y firmadopor. Para ello, se recorre el array responsable, dividiendo el responsable por el puesto y el nombre. El formato que tiene el responsable es [puesto],‘Fdo.:’ [nombre], por lo que dividiremos cada responsable por la cadena ‘Fdo.:’. A su ver el puesto puede tener un tipo firma, por lo que en cada puesto se comprobará si tiene alguno de estos tipos de firma y en caso de tenerlo dividirá por el tipo de firma que tenga, para obtener el puesto, el tipo de firma y el órgano que corresponde al tipo de firma.

En la inserción de responsables existe un problema, como ya se ha indicado, y es que existen personas que unas veces su puesto aparece completo y otras sólo una parte del mismo. Para comprender mejor este problema vamos a ver un ejemplo.

La persona cuyo nombre es Antonio Silván Rodríguez unas veces aparece con el puesto ‘El Consejero’ y otras como ‘El Consejero de Fomento y Medio Ambiente’, a la vista de una persona resulta evidente que es el mismo puesto, pero el ordenador no es capaz de interpretarlo como lo mismo. Para ello, se decide crear el siguiente trigger sobre la tabla responsable.

TRIGGER SOBRE TABLA RESPONSABLE

El trigger será lanzado antes de insertar los datos en la tabla responsable, y hará uso de una tabla adicional, denominada tmp, que tendrá tres columnas: columna nombre, puesto y puestoaux. En la columna puestoaux se insertará el valor del puesto que forma una subcadena de otro puesto, ambos con mismo nombre.

El trigger comprobará si ya existe un responsable insertado que tenga el mismo nombre, pero diferente puesto. En caso de existir el responsable en la base de datos, selecciona el puesto que ocupa y comprueba si el puesto del responsable existente en la base de datos es una subcadena del puesto que estamos intentando insertar, le denominaremos ‘puestobbdd’, en ese caso comprueba si existe un valor en la columna puesto de la tabla tmp, donde el valor de nombre y puestoaux son los mismos que los valores del nombre y puesto que se pretenden insertar; de ser así, significaría que existe un responsable cuyo valor de puesto contiene como subcadena el puesto que estamos intentando insertar. En caso contrario, de no existir ningún valor en la columna puesto de la tabla tmp, donde el valor de nombre y puestoaux son los mismos que los valores del nombre y puesto que se pretenden insertar, modificará los registros que puedan

existir en la tabla tmp se indique que el puestobbdd es mayor que otro puesto, poniendo como valor mayor el puesto que se pretende insertar, e insertando en la tabla tmp los valores que representen que el puesto que intentamos insertar, es mayor que el denominado puestobbdd.

En el caso contrario, en el que el puesto que intentamos insertar sea la subcadena de un registro almacenado en la tabla responsable con el mismo nombre y distinto puesto, almacenará en la tabla tmp los valores que indiquen que el puesto existente en la tabla responsable es mayor que el que se pretende insertar.

```
[1]  CREATE TRIGGER `tresponsable` BEFORE INSERT ON `responsable`
[2]  FOR EACH ROW BEGIN
[3]  DECLARE Vidres INT;
[4]  DECLARE total,i,V1,V2,VnewL, VoldL, VsharedL INT;
[5]  DECLARE Vpuesto, puesto1, puesto2 VARCHAR(200);
[6]  SELECT COUNT(*) INTO total FROM responsable WHERE
      nombre=NEW.nombre and puesto<>NEW.puesto;
[7]  IF total<>0 THEN
[8]   SET i=0;
[9]   WHILE (i<total) DO
[10]        SELECT id_responsable,puesto INTO Vidres,Vpuesto FROM
            responsable WHERE nombre=NEW.nombre and
            puesto<>NEW.puesto     ORDER BY puesto DESC LIMIT i,1;
[11]        IF Vidres<>0 THEN
[12]              SELECT LENGTH(NEW.puesto) INTO VnewL;
[13]              SELECT LENGTH(Vpuesto) INTO VoldL;
[14]              IF (VoldL < VnewL) THEN
[15]                    SELECT INSTR(NEW.puesto,Vpuesto) INTO
                        VsharedL;
[16]                    IF (VsharedL <> 0) THEN
[17]                          SELECT puesto INTO puesto2 FROM tmp
WHERE                         nombre=NEW.nombre and
                              puestoaux=NEW.puesto;
[18]                          IF puesto2<>'' THEN
[19]                                INSERT IGNORE INTO tmp (nombre,
                                    puesto, puestoaux) VALUES
                                    (NEW.nombre, puesto2, NEW.puesto);
[20]                          ELSE
[21]                                UPDATE tmp SET puesto=NEW.puesto
                                    WHERE nombre=NEW.nombre and
                                    puesto=Vpuesto;
[22]                                INSERT IGNORE INTO tmp(nombre,
                                    puesto, puestoaux) VALUES
                                    (NEW.nombre, NEW.puesto,Vpuesto);
[23]                          END IF;
[24]                    END IF;
[25]              END IF;
[26]              IF (VoldL>VnewL) THEN
[27]                    SELECT INSTR(Vpuesto, NEW.puesto) INTO
                        VsharedL;
[28]                    IF (VsharedL <> 0) THEN
[29]                          SELECT puesto INTO puesto1 FROM tmp
```

```
                              WHERE nombre=NEW.nombre and
                              puestoaux=Vpuesto;
[30]                          IF puesto1<>'' THEN
[31]                                INSERT IGNORE INTO tmp (nombre,
                                    puesto, puestoaux) VALUES
                                    (NEW.nombre, puesto1 ,NEW.puesto);
[32]                          ELSE
[33]                                INSERT IGNORE INTO tmp (nombre,
                                    puesto, puestoaux) VALUES
                                    (NEW.nombre, Vpuesto, NEW.puesto);
[34]                          END IF;
[35]                    END IF;
[36]              END IF;
[37]        END IF;
[38]        SET i=i+1;
[39]  END WHILE;
[40] END IF;
[41] END
```

Una vez finaliza de insertarse todos los datos en nuestra base de datos, se ejecuta la siguiente sentencia SQL que se encargará de borrar todos los registros de la tabla responsable cuyo nombre y puesto es igual al `nombre` y el `puestoaux` de la tabla `tmp`.

```
[1] DELETE r FROM responsable r JOIN tmp t where r.nombre =
      t.nombre AND r.puesto=t.puestoaux
```

Cuando se ejecuta la sentencia anterior de borrado se lanza el siguiente *trigger* sobre la tabla responsable. Este *tigger* se ejecuta antes de borrar cualquier registro de la tabla responable, y se encarga de modificar cualquier registro de la tabla firmadopor cuyo idresponsable identifique un valor de responsable que se va a borrar, y cambiándolo por el responsable que se sustituye.

```
[1] CREATE TRIGGER `tdeleterespons` BEFORE DELETE ON `responsable`
[2] FOR EACH ROW BEGIN
[3] DECLARE puesto1 VARCHAR(200);
[4] SELECT puesto INTO puesto1 FROM tmp WHERE nombre=OLD.nombre and
      puestoaux=OLD.puesto;
[5] UPDATE firmadopor SET id_responsable=(SELECT id_responsable FROM
      responsable WHERE nombre=OLD.nombre and puesto=puesto1) WHERE
      id_responsable=(SELECT id_responsable FROM responsable WHERE
      nombre=OLD.nombre and puesto=OLD.puesto);
[6] END
```

ANEXO

Lo último es almacenar el anexo, en caso de existir. Para ello recorremos el *array* `anexo` mediante un bucle *for* y anidamos todas los elementos en un mismo *string*. Después crea la sentencia SQL necesaria para insertar el anexo en la tabla anexo, para ello obtiene el valor del documento en el que nos encontramos haciendo uso de la *url* del documento, e inserta los valores mediante la función `insertarBBDD`.

5.4. Fase II: Explotación de los datos en Open Data.

Esta segunda fase se encarga de crear el documento en Open Data, para que los datos correspondientes a los boletines estén disponibles de forma libre, permitiendo que sean reutilizados por cualquier máquina de forma automática.

El formato en el que se han decidido presentar los datos es XML, el motivo de escoger este tipo de formato es que permite de una forma clara conocer qué tipo de información existe en el documento. Sí es cierto que al ser un documento en Open Data, y por lo tanto que va a ser procesado por máquinas, la claridad de la información no es necesaria pero de este modo permitimos a la persona que va a trabajar con los datos conocer qué tipo de información se va a encontrar de una forma sencilla y esquemática, y facilitarle el trabajo.

El esquema que va a tener el XML será el siguiente:

```
<boletin fecha=" ">
      <documento URL=" ">
            <seccion>
                  <seccion>
                        <seccion>
                              <apartado>
                                    <apartado>
                                          <apartado></apartado>
                                    </apartado>
                              </apartado>
                        </seccion>
                  </seccion>
            </seccion>
            <cabecera> </cabecera>
            <contenido> </contenido>
            <anexo> </anexo>
            <firma>
                  <tipofirma> </tipofirma>
                  <organo> </organo>
                  <responsable>
                        <nombre> </nombre>
                        <puesto> </puesto>
                  </responsable>
                  <fecha> </fecha>
                  <municipio> </municipio>
            </firma>
      </documento>
</boletin>
```

La etiqueta `boletín` englobará al boletín completo y tendrá como atributo la fecha del boletín, dentro de ésta estarán todos los documentos cada uno será una etiqueta con el atributo *url* que será la *url* propia del documento. Dentro de cada documento estarán las secciones y apartados representando la jerarquía existente, posteriormente estarán la cabecera, el contenido, el anexo y la firma. La firma constará del tipofirma, el órgano si existe, el responsable con el nombre y puesto, y la fecha y municipio.

La implementación del *script* que obtiene los datos en Open Data se encuentra en el fichero denominado *'scriptXML'* y que se va a presentar de forma detallada a continuación.

Para poder realizar el documento XML hacen falta importar las librerías `MySQLdb`, `ElementTree` y `Element`.

El procedimiento de creación del árbol es similar para todos los elementos: se crea el elemento nuevo, se le da valor y atributo en caso necesario, y por último, se añade al elemento del árbol del que cuelga.

Primero se crea el '*root'*, es decir la raíz del documento, que se denominará boletín y lo establecemos como elemento del árbol `tree`. Después se establece el atributo `fecha` con la fecha del boletín para el que se está almacenando en XML, la fecha es la que el administrador ha proporcionado al sistema para realizar el fichero.

```
[1] root=Element('boletin')
[2] tree=ElementTree(root)
[3] root.set('fecha',fechanecesaria)
```

Una consulta a la base de datos obtiene los enlaces de los documentos que corresponden al boletín de la fecha establecida. El resultado son todos los enlaces de los documentos. Para cada enlace, es decir, para cada documento, se obtiene la información de cada tabla que se va añadiendo al árbol, y se crea un elemento documento que colgará del elemento raíz (root), y del que colgará toda esta información. Cada elemento 'documento' se le establece como atributo su url.

SECCIÓN y APARTADO

Como se deben insertar los elementos de las secciones y apartados respetando la jerarquía, primero debemos obtener todas las secciones y apartados, de forma ascendente, es decir, del apartado más cercano al documento, hasta la sección de la que dependen todos.

Se realiza una consulta a la base de datos para obtener el `nombre_administracion`, `padre_seccion` y `padre_apartado` de la tabla `apartado` cuyo id_apartado sea el mismo que el que se encuentra en el documento en el que nos encontramos. Y almacena el nombre de la administración en el *array* apartado. Como en padre_apartado encuentra el apartado que se encuentra encima de el en la jerarquía, realiza la misma consulta para obtener el apartado, hasta que el padre_apartado sea cero. Cuando el padre_apartado es cero, tendremos en la variable padre_seccion a la sección que se encuentra encima en la jerarquía, por lo que realizamos otro bucle del mimo modo que para los apartados pero consultando en la sección. Las secciones se almacenan en el *array* sección.

Ahora debemos recorrer los *arrays* que contienen la información de la jerarquía. Esta información ha sido almacenada de forma ascendente, del apartado más cercano al documento hasta la sección principal a la que pertenece el documento. Por este motivo, para almacenar la información debemos recorrernos los *arrays* de forma inversa, para así, almacenar la jerarquía de forma correcta.

Como tenemos que almacenar la información de forma automática, y cada documento tiene un número variable de apartados y documentos haremos uso de

diccionarios. Los diccionarios son estructuras de datos variables que almacenan los datos como pares clave-valor.

De este modo, mediante un bucle *for* que recorra el *array* secciones de forma inversa iremos creando los nodos del árbol y estableciendo los valores de estos. Si nos encontramos en la última posición del *array*, es decir la primera de la jerarquía, crearemos una clave en el diccionario y le definiremos como elemento `'seccion'` del árbol, después añadiremos al elemento 'seccion' el valor que le corresponde.

En el caso de que no nos encontremos en la última posición del *array* `secciones` crearemos una clave en el diccionario y la definiremos como elemento `'seccion'`. Le daremos el valor del *array* en la posición en la que nos encontramos y asociaremos el elemento que acabamos de crear con el anterior que hemos creado.

En el caso de los apartados es exactamente igual, salvo en el caso de encontrarnos en la última posición del *array*, ya que en este caso asociaremos el elemento que acabamos de crear con el primer elemento del *array* secciones

```
[1] diccionarioseccion={}
[2] for j in reversed(range(len(secciones))):
[3]   if j==len(secciones)-1:
[4]         diccionarioseccion["grupo%d"%j]=Element('seccion')
[5]         documento.append(diccionarioseccion["grupo%d"%j])
[6]         diccionarioseccion["grupo%d"%j].text=secciones[j].decode
              ("unicode_escape")
[7]   else:
[8]         diccionarioseccion["grupo%d"%j]=Element('seccion')
[9]         diccionarioseccion["grupo%d"%j].text=secciones[j].decode
            ("unicode_escape")
[10]        diccionarioseccion["grupo%d"%(j+1)].append(
              diccionarioseccion["grupo%d"%j])
[11] diccionarioapartado={}
[12] for k in reversed(range(len(apartado))):
[13]  if k==len(apartado)-1:
[14]        diccionarioapartado["grupo%d"%k]=Element('apartado')
[15]        diccionarioseccion["grupo0"].append(
              diccionarioapartado["grupo%d"%k])
[16]        diccionarioapartado["grupo%d"%k].text=apartado[k].decode
            ("unicode_escape")
[17]  else:
[18]        diccionarioapartado["grupo%d"%k]=Element('apartado')
[19]        diccionarioapartado["grupo%d"%k].text=apartado[k].decode
              ("unicode_escape")
[20]        diccionarioapartado["grupo%d"%(k+1)].append(
              diccionarioapartado["grupo%d"%k])
```

CABECERA y CONTENIDO

Ahora debemos de obtener los datos de la cabecera y el contenido. Para ello, obtenemos la cabecera, el contenido y la fecha del documento cuando el enlace corresponde con el que nos encontramos, mediante una sentencia SQL a la base de

datos. La fecha será utilizada en la parte de la firma. Creamos el elemento ‘`cabecera`’ y lo añadimos al documento y establecemos como valor la cabecera obtenida de la consulta. De manera similar lo hacemos para el contenido, pero poniendo como valores `contenido`.

ANEXO

Para el anexo es de forma similar a lo que hemos realizado para la cabecera y el contenido. Ejecuta una consulta en la base de datos que nos obtenga el contenido de la tabla anexo cuyo id_documento es igual al identificador del documento en el que estamos. En el caso de que el resultado no sea vacío, ya que un documento puede no tener anexo, crearemos el elemento del árbol ‘anexo’, lo añadiremos al documento y lo daremos el valor que la consulta nos ha devuelto.

FIRMA

Ahora debemos de crear la firma del documento, para ello creamos un elemento y lo denominamos firma y lo añadimos al elemento documento del árbol. Dentro de este elemento firma, estará toda la información correspondiente a la firma, es decir, el tipo de firma, el órgano, el nombre del firmante, el puesto que ocupa, la fecha y el lugar en el que se firmó. Realizamos una consulta a la base de datos para obtener el tipofirma, el órgano, el nombre y el puesto de las tablas `responsable` y `firmadopor`. Como puede haber más de un firmante en un documento realizamos un bucle *for* por el *array* que contiene el resultado de la consulta, para cada posición del *array* del resultado creamos un elemento denominado `tipofirma`, lo añadimos al elemento `firma` y le establecemos como valor el `tipofirma` que hemos obtenido de la consulta. Comprobamos si el órgano que nos devuelve la consulta es diferente de ‘null’, y en caso de ser así crearemos el elemento órgano lo asociaremos con el elemento firma y estableceremos como valor el órgano que nos ha devuelto la consulta.

Después creamos el elemento `responsable`, y lo vinculamos con el elemento `firma`, creamos el elemento `nombre`, lo añadimos al elemento `responsable` y establecemos como valor el nombre que nos ha devuelto la consulta. Del mismo modo hacemos para el puesto, creamos el elemento `puesto` lo añadimos al `responsable` y le damos como valor el puesto que nos devuelve la consulta.

Realizamos una nueva consulta a la base de datos para obtener el municipio de la tabla municipio que tiene el mismo identificador que el que tiene el documento en el que nos encontramos. Creamos el elemento `municipio` y lo añadimos al elemento `firma` dándole como valor el resultado de la consulta. Creamos el elemento `fecha`, lo añadimos al elemento `firma` y le damos el valor de la fecha que hemos obtenido en la consulta que hemos realizado anteriormente a la tabla documento, cuando hemos obtenido la cabecera y el contenido. El valor de la fecha que obtenemos debemos de pasarlo a *string* mediante la función `strtime('%d/%m/%y')`.

Estas operaciones las realizamos para todos los documentos, es decir, para todas las urls que hemos obtenido al principio. De modo que conseguimos crear el árbol con toda la información de los documentos.

Como se puede observar, el hecho de tener los datos almacenados en la base de datos facilita enormemente la reutilización de los mismos para la creación del archivo Open Data en formato xml. Mediante simples consultas a la base de datos obtenemos

toda la información, que después puede ser reutilizada para diferentes actividades, en este caso, la creación del archivo xml.

5.5. Prototipo

En este apartado se va a proceder a mostrar el funcionamiento del prototipo, para lo que se va a hacer uso del boletín del día 29/08/2014.

Para que el prototipo se pueda ejecutar es necesario establecer un entorno de explotación que contenga todas las herramientas necesarias. Este entorno de programación se obtiene siguiendo las pautas que se indican en el manual de instalación que se encuentra en la Anexo I. Una vez instalado todo, podremos comenzar a utilizar nuestro sistema:

Primer paso: Obtención del boletín.

Para ejecutar este paso nos iremos al programa Python y desde allí, accederemos al archivo 'scriptBBDD' que se encuentra en la misma carpeta de Pytuon. Una vez accedamos al archivo deberemos indicar la *url* del boletín que queremos *scrapear* como parámetro de entrada, en nuestro caso *http://bocyl.jcyl.es/boletin.do?fechaBoletin=29/08/2014*. Para ello daremos la url como valor a la variable linkOriginal que se encuentra en la línea 18 del archivo. Una vez hemos definido la url como parámetro de entrada al sistema, deberemos ejecutarlo. Para ello, tenemos dos opciones, una ir a la barra de herramientas, al botón de 'Run' y una vez se despliega el submenú se pincha sobre el botón 'Run Module', otra opción es ejecutar el script mediante la tecla F5, que se encuentra en la parte superior del teclado.

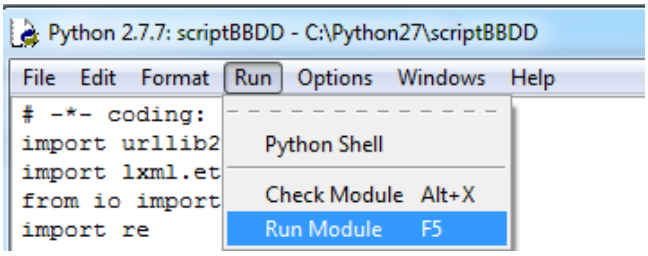

Ilustración 13: Ejecución del script

Una vez ejecutado, este script será el encargado, en un primer paso, de obtener la información de un boletín. Para ello hará uso de una serie de *arrays* donde almacenará la información para que posteriormente sea procesada.

Cuando la primera parte de este script termina de ejecutarse, el programa pasará los *arrays* a una función que será la encargada de procesar la información y de crear las sentencias SQL de inserción de los datos. El script se conectará con la base de datos de MySQL y ejecutará las sentencias que ha creado anteriormente, encargándose de almacenar la información. Una vez se ejecuta el script, el resto de las operaciones se realizaran en segundo plano, sin que el usuario que lo ejecuta tenga que realizar nada.

Una vez finaliza el script podemos observar en la base de datos cómo la información referente al boletín del día 29/08/2014 está almacenada.

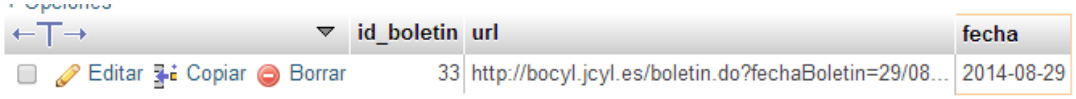

Ilustración 14: Boletín almacenado

Segundo paso: Obtención del Open Data.

Para ejecutar este paso nos iremos al programa Python y desde allí, accederemos al archivo 'scriptXML'. Este archivo será el encargado de recopilar toda la información que existe en la base de datos referente al boletín que el usuario desea.

Una vez accedamos al archivo deberemos indicar la *fecha* del boletín que queremos *scrapear* como parámetro de entrada, para ello daremos la fecha como valor a la variable fechanecesaria que se encuentra en la línea 9 del archivo, en nuestro ejemplo el valor será 2014-08-29. Una vez hemos definido la fecha como parámetro de entrada al sistema, deberemos ejecutarlo. Para ello, tenemos dos opciones, una ir a la barra de herramientas, al botón de 'Run' y una vez se despliega el submenú se pincha sobre el botón 'Run Module' como se muestra en la Ilustración 13, otra opción es ejecutar el script mediante la tecla F5, que se encuentra en la parte superior del teclado.

Cuando se ejecuta el script se conectará con la base de datos para recopilar la información y genera el archivo en formato XML, el cual será almacenado de forma automática en el ordenador. Una vez el script es ejecutado las operaciones que se realizan por detrás el usuario no las conoce.

Cuando finaliza de ejecutarse tenemos un archivo en la carpeta XML con el archivo Open Data en formato XML, y cuya fecha será la fecha que el usuario ha indicado al principio.

El prototipo ha sido ejecutado para obtener los boletines de los últimos dos meses, desde el 1 de julio de 2014 hasta el 29 de agosto de 2014, almacenando la información en la base de datos. Además de cada boletín se ha generado un archivo XML siguiendo la filosofía de Open Data. En términos de almacenamiento, la base de datos ha ocupado un total de 20,4Mb, y los archivos XML han ocupado un total de 16 Mb.

Por último para la creación y uso del prototipo, se han tomado las decisiones sobre los tipos de motores de almacenamiento e índices que serán usados en las tablas de nuestra base de datos.

Motor de almacenamiento

La opción elegida para todas las tablas es InnoDB, esta elección se debe a que InnoDB proporciona bastantes beneficios como puede ser:

- Robustez ante fallos de hardware o software, finalizando los cambios realizados antes del fallo y deshaciendo los que comenzaron pero no alcanzaron el estado de comprometido.
- Cachea los datos e índices que se usan frecuentemente de modo que se puedan consultar en memoria.
- Integridad sobre claves foráneas, si insertamos o borramos datos las relaciones se actualizarán automáticamente.
- Optimiza las relaciones que se hacen sobre claves primarias permitiendo mayor velocidad.

Índices

El índice de una base de datos es una estructura de datos que permite agilizar las consultas a las tablas. En el proyecto se deciden crear índices sobre la columna lugar de la tabla municipio, y sobre la columna url de la tabla documento.

El hecho de decidir crear un índice sobre la columna lugar de la tabla municipio, se fundamenta en que la tabla municipio es consultada cada vez que se inserta un documento, y en cada consulta en el peor de los casos se deberá recorrer 2249 registros, que es el número total de registros existentes en la tabla.

En el caso de la tabla documento, se ha decidido crear el índice sobre la columna url, ya que cada vez que se inserten datos, en las tablas anexo y firmadopor deberá recorrerse la columna url para buscar el identificador del documento. La tabla documento puede llegar a tener unas dimensiones elevadas, ya que tomando como medida la cantidad de documentos almacenados en la base de datos en dos meses, nuestra tabla documento llega a tener 11850 registros.

6. Pruebas

Esta última etapa contiene todas las pruebas realizadas con el objetivo de detectar los posibles errores de codificación. La ejecución de las pruebas no certifica la ausencia de errores, ya que como este proyecto depende de terceros, es decir, de los administradores de la página del BOCyL, no se pueden predecir qué cambios van a realizar para adaptarnos a ellos. Pero lo que si conseguimos con la realización de pruebas es demostrar la existencia de los mismos.

Existen dos tipos de pruebas:

Pruebas de caja blanca: se realizan sobre las funciones internas de un módulo, clase, etc.

Pruebas de caja negra: comproban que los requisitos funcionales se han respetado y cumplido. Es decir, permite obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa.

.

Las pruebas de caja blanca se han ido realizando a la vez que se ha ido implementando el proyecto. En cambio, las pruebas de caja negra se van a presentar a continuación, su diseño se creó en la fase de análisis, pero al ejecutarse en esta parte del proyecto se muestra toda la información en este punto.

PRUEBA I: Obtener los datos de un boletín

CP- 01	Obtener los datos de un boletín
Propósito	Comprobar la correcta obtención de un boletín, comprobando tanto la parte de recolección de datos y almacenamiento en los arrays necesarios
Prerrequisito	
Datos de entrada	Url del boletín
Pasos	1.- El sistema obtiene los datos del boletín
Resultado esperado	Los datos del boletín son almacenados en bruto en los arrays.
Resultado obtenido	Correcto

Tabla 34: CP-01

El caso de prueba anterior, sólo es un ejemplo de los más de cuarenta que se realizaron para verificar el correcto funcionamiento del caso de uso.

PRUEBA II y III: Validar URL

CP- 02	Validar URL
Propósito	Comprobar que el sistema valida correctamente la url que el administrador le pasa.
Prerrequisito	
Datos de entrada	http://bocyl.jcyl.es/boletin.do?fechaBoletin=29/08/2014
Pasos	1.- El administrador entrega la url al sistema
Resultado esperado	Sistema muestra el mensaje 'URL inválida'
Resultado obtenido	Correcto

Tabla 35: CP- 02

En la ilustración 15, se puede comprobar cómo el sistema muestra correctamente el mensaje que se indica en la prueba CP- 02.

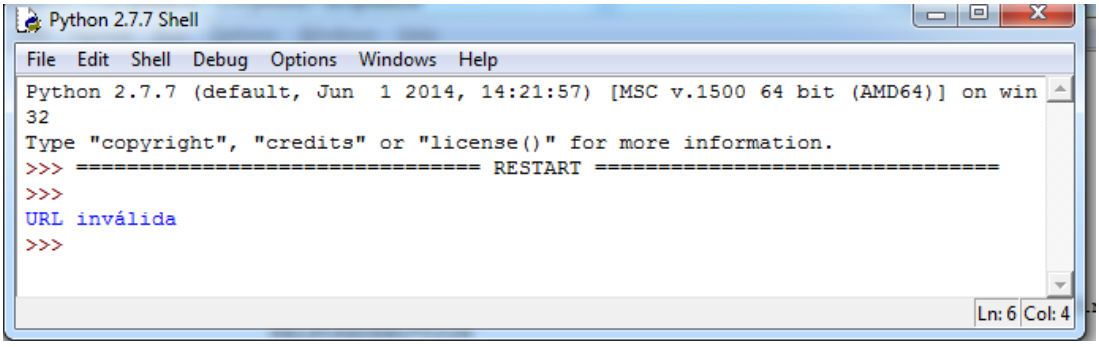

Ilustración 15: Ejecución con url inválida

CP- 03	Validar URL
Propósito	Comprobar que el sistema valida correctamente la url que el administrador le pasa.
Prerrequisito	
Datos de entrada	'http://bocyl.jcyl.es/boletin.do?fechaBoletin=12/12/2014'
Pasos	1.- El administrador entrega la url al sistema
Resultado esperado	Sistema muestra el mensaje 'No existe boletín con url = http://bocyl.jcyl.es/boletin.do?fechaBoletin=12/12/2014'
Resultado obtenido	Correcto

Tabla 36: CP-03

En la ilustración 16, se puede comprobar cómo el sistema muestra correctamente el mensaje que se indica en la prueba CP-03.

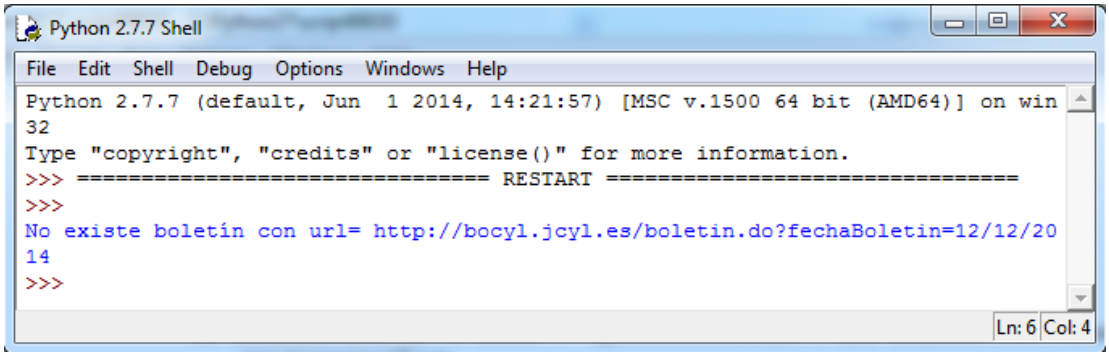

Ilustración 16: Ejecución de un boletín no publicado

PRUEBA IV: Almacenar en la base de datos

CP- 04	Almacenar en la base de datos
Propósito	Comprobar la correcta inserción de los datos que están en los arrays, en la base de datos.
Prerrequisito	
Datos de entrada	Arrays con la información tras finalizar la prueba I
Pasos	1.- El sistema obtiene los arrays.
Resultado esperado	Los datos son almacenados en la base de datos
Resultado obtenido	Correcto

Tabla 37: CP- 04

El caso de prueba anterior, sólo es un ejemplo de los más de cuarenta que se realizaron para verificar el correcto funcionamiento del caso de uso.

PRUEBA V: Creación del archivo Open Data

CP- 05	Creación del Open Data
Propósito	Comprobar la correcta creación de un archivo XML
Prerrequisito	Deberán estar los datos almacenados en la base de datos
Datos de entrada	2014-08-29
Pasos	1.- El administrador entrega la fecha al sistema
Resultado esperado	Archivo creado con toda la información, respetando las jerarquías.
Resultado obtenido	Correcto

Tabla 38: CP-05

PRUEBA VI: Validar fecha

CP- 06	Validar fecha
Propósito	Comprobar que el sistema valida correctamente la fecha que el administrador le pasa.
Prerrequisito	
Datos de entrada	2014-31-31
Pasos	1.- El administrador entrega la fecha al sistema
Resultado esperado	Sistema muestra el mensaje 'URL inválida'
Resultado obtenido	Correcto

Tabla 39: CP- 06

En la ilustración 17, se puede comprobar cómo el sistema muestra correctamente el mensaje que se indica en la prueba CP-06.

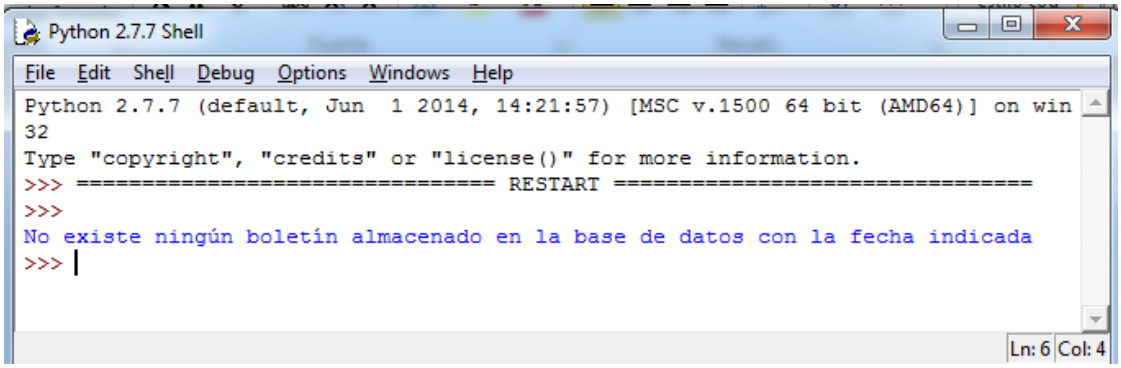

Ilustración 17: Prueba validar fecha

PRUEBA VII: Eliminación de los responsables.

CP- 07	Eliminación de los responsables
Propósito	Comprobar que cuando aparecen responsables duplicados que contienen varios puestos y en los que uno es subcadena del otro, el sistema borra los registros de responsable que contienen el mismo nombre y como puesto el puesto de subcadena.
Prerrequisito	
Datos de entrada	SELECT count(*) FROM responsable r JOIN tmp t where r.nombre = t.nombre AND r.puesto=t.puestoaux
Pasos	1.- El sistema gestor de base de datos ejecuta la sentencia
Resultado esperado	0
Resultado obtenido	Correcto

Tabla 40: CP- 07

7. Conclusiones

Por último, se van a mostrar las conclusiones finales que se extraen tras la realización del proyecto.

PRINCIPALES DIFICULTADES

Las dificultades encontradas durante la realización del trabajo fin de grado se describen en los siguientes puntos:

- Completo desconocimiento de la técnica de web scraping. Cuando comencé el proyecto sólo tenía una breve idea de lo que era esta técnica, pero nunca me imaginaba lo que ésta podía llegar a hacer. Durante la etapa inicial de investigación logre comprender en detalle la técnica y sus variantes, lo que me ha permitido, enfocar correctamente el proyecto hacia sus objetivos. Es necesario mencionar, que la información residente en la web es ambigua y cuesta obtener una idea consistente sobre lo que es y para lo que sirve.

- Desconocimiento sobre el entorno de desarrollo. Para realizar el proyecto decidí hacer uso del lenguaje de programación Python, por sus grandes beneficios. Esta dificultad fue solventada eficientemente debido a la gran cantidad de información que existe sobre este lenguaje de programación.
- BOCyL no se adapta a los patrones que define. Existen edictos y documentos que no se ajustan al patrón definido, y que por lo tanto se debe realizar un tratamiento más exhaustivo para lograr obtener la información. El hecho de que existan documentos que no siguen los patrones complica en gran medida el procesado, ya que se debe tener en cuenta todas las diferentes variables que puede tomar el documento. Esto implica que el administrador del sistema tenga que estar continuamente realizando pruebas para saber si hay alguna variante que el programa no considere y que impida la correcta obtención de la información.

OBJETIVOS ALCANZADOS

Como se ha dicho a lo largo del proyecto, el objetivo principal consistía en la obtención y almacenamiento de la información del BOCyL en una base de datos que permita reutilizar la información para cualquier actividad diferente. Este objetivo se ha conseguido de forma bastante satisfactoria.

Además, de forma complementaria, se ha conseguido dar una reutilización a la información en forma de archivos XML, siguiendo la filosofía de Open Data.

CONOCIMIENTOS ADQUIRIDOS

En primer lugar, destacar que la realización de este trabajo me ha permitido realizar una aplicación de cierta envergadura, lo que implica tener que realizar las tareas de planificación, análisis, diseño, implementación y pruebas. La realización del trabajo me ha permitido una primera toma de contacto con lo que es la realización de un proyecto completo, lo que supone una acercamiento con lo que será el día a día una vez me incorpore al mundo laboral.

En segundo lugar, la investigación llevada a cabo en la fase de estudio de la técnica de *web scraping*, me ha permitido ampliar mis conocimientos y obtener una mayor facilidad a la hora de sintetizar información, así como de hacer comparación entre los diferentes tipos de información que me he podido encontrar.

Por último, la creación de la aplicación de web scraping me ha permitido conocer un lenguaje de programación con el que nunca había trabajado.

POSIBLES MEJORAS

El estado actual del proyecto permite crear una base de datos con la información del BOCyL, la cual resulta muy interesante a la hora de poder reutilizar los datos para diferentes usos. Una de las ideas que se podrían llevar a cabo con esta información es la realización de un BOCyL, pero que permitiera navegar entre los documentos que se encuentren relacionados entre sí, ya que actualmente, existen documentos que son una ampliación de otros documentos anteriormente publicados y que para navegar de un documento a otro debemos de buscarlo por nuestra cuenta, de forma manual.

8. Bibliografía

WEB SCRAPING

- Javier Ibañez, Screen Scraping, http://riunet.upv.es/bitstream/handle/10251/10104/PFC_Javier_Iba%C3%B1ez.pdf, (25/06/2014)
- Open Knowledge Foundation, Data Journalism Handbook, http://datajournalismhandbook.org/1.0/en/getting_data_3.html, (25/06/2014)
- Open Knowledge Foundation, School of Data, 2013, http://schoolofdata.org/2013/11/25/scraping-the-web/, (25/06/2014)
- Code Proyect, 2007,http://www.codeproject.com/Articles/676672/Web-Scraping-Problems-Solutions, (25/06/2014)
- BBC, Conozca a los "scrapers", los ladrones de páginas de internet, 2013, http://www.bbc.co.uk/mundo/noticias/2013/10/131002_tecnologia_screen_scrapers.shtml, (28/06/2014)

- Nicolás Marin Torres, Web Scraper con PHP, 2013, http://www.nicolasmarin.com/web-scraper-con-php/, (28/06/2014)
- Escuela de Datos, Introducción a la extracción de datos de sitios web: scraping, http://es.schoolofdata.org/introduccion-a-la-extraccion-de-datos-de-sitios-web-scraping/, (26/08/2014)
- Powerbits, Talleres de Scraping, 2013, http://blog.powerbits.eu/category/webscraping/, (25/06/2014)

LEGALIDAD

- Gómez- Acebo & Pombo, Lexology, 2013, http://www.lexology.com/library/detail.aspx?g=270c4dc5-b937-4321-9783-a62b285a1f1e, (25/06/2014)
- BudgetAir, 2010, http://www.budgetair.es/aviso-legal, (20/06/2014)
- Autoescola Clot, http://www.clot.es/Legal.aspx, (24/06/2014)
- Tonido, Web Scraping and Legal Uses, 2013, http://www.tonido.com/blog/index.php/2013/12/28/web-scraping-and-legal-issues/#.VACAN_l_tu4, (25/06/2014)
- Imperva, Detecting and Blocking Site Scraping Attacks, 2014, http://www.imperva.com/docs/wp_detecting_and_blocking_site_scraping_attacks.pdf, (27/05/2014)
- Open Congress, Web Scraping Legal Uses, 2012, http://www.opencongress.org/wiki/Web_Scraping_Legal_Issues, (25/05/2014)
- Custis Smolar, How Legal is content scraping, http://venturebeat.com/2011/05/30/how-legal-is-content-scraping/, (28/06/2014)

HERRAMIENTAS

- Datknosys, http://blog.tecnologico.datknosys.com/index.php/comparativa-de-herramientas-de-scraping/, (27/03/2014)
- Google Code, 2011, https://code.google.com/p/aap-etsiit-ugr/wiki/MarketScraperFinal, (15/06/2014)
- Mozenda,2013, http://www.mozenda.com/web-scraping-software, (18/06/2014)
- Scraperwiki, 2012, https://scraperwiki.com/, (10/06/2014)
- Wikipedia, http://es.wikipedia.org/wiki/GNU_Wget, (25/04/2014)
- GNUWget, https://www.gnu.org/software/wget/manual/wget.pdf, (03/05/2014)
- Web Harby, 2013, https://www.webharvy.com/articles/web-scraper-use-cases.html, (28/05/2014)

PROYECTOS SIMILARES

- Miguel Fiandor, Transparencia en las Cuentas Públicas, 2012, http://transparenciadecuentaspublicas.es/, (13/07/2014)
- Juan Elosua, David Cabo y Eva Belmone, El Indultómetro, 2013, http://www.elindultometro.es/, (13/07/2014)

PYTHON

- Lutz Prechelt, Are Scripting Languages Any Good?A Validation of Perl, Python, Rexx, and Tcl against C, C++, and Java, 2002, http://www.inf.fu-berlin.de/inst/ag-se/pubs/jccpprt2_advances2003.pdf (11/05/2014).
- Servicio del Boletín Oficial de Castilla y León, Boletín Oficial de Castilla y León,2010 http://bocyl.jcyl.es/ (28/08/2014)
- PyAr, 2013, http://revista.python.org.ar/2/es/html/virtualenv.html, (15/08/2014)
- CodeHero, 2014, http://codehero.co/python-desde-cero-bases-de-datos/, (15/08/2014)
- Guido Van Rosum, El tutorial de Python, 2009, http://docs.python.org.ar/tutorial/pdfs/TutorialPython2.pdf, (12/08/2014)
- Alex Martelli, Python. Guía de Referencia, Editorial ANAYA, 2008

ANEXO I: Manual de instalación

Para el correcto funcionamiento del proyecto se deberá tener instalado en el ordenador, tanto el sistema de gestor de base de datos MySQL, como el programan Python. La instalación de ambos programas se realizará en Windows, por lo que resulta sencilla. Simplemente, se ejecutará los archivos con extensión .exe que se encuentran en la careta herramientas del CD-ROM. Después se deberán seguir una serie de pautas:

1. Accedemos a MySQL, para ello en la máquina de comandos escribimos:
   mysql -h localhost -u root –p

2. Creación de la base datos en MySQL. El nombre que deberá tener la base de datos es ‘bbdd’.
   CREATE DATABASE ‘bbdd’;
   USE bbdd;

3. Importar el archivo ddl contenido en la carpeta Base de Datos del CD-ROM. Este archivo se encargará de crear la estructura que almacenará los datos.
   mysql --host localhost --user root-p bbdd<DDL.sql
4. Importar el archivo denominado ‘DMLMunicipio’, contenido en la carpeta Base de Datos del CD-ROM: Este archivo se encargará de crear los registros de todos los municipios existentes en la comunidad de Castilla y León.
5. mysql --host localhost --user root-p bbdd<DMLMunicipio.sql

6. Creación del usuario administrador mediante las sentencias.
   CREATE USER 'administrador'@'localhost' IDENTIFIED BY 'administrador';
   GRANT ALL PRIVILEGES ON `bbdd` . * TO 'administrador'@'localhost';