Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Plataforma SOA para un catálogo de cómics hentai

Trabajo Fin de Máster

Máster Universitario en Gestión de la Información

Autor: Adrià Girbés Ferrer

Tutor: Pedro J. Valderas Aranda

2017-2018

Resumen

Se explica cómo se construye un catálogo unificador de la oferta hecha por tiendas en línea de cómics hentai digitales mediante las técnicas de rastreo (crawling) y scraping. Este catálogo está soportado por una plataforma que sigue la arquitectura orientada al servicio (SOA).

Palabras clave: Crawling, Scraping, Catálogo, SOA.

Abstract

In this master's thesis it is explained how a unifying catalogue of the digital hentai comic books that are offered in online shops was created through crawling and scraping web techniques. This catalogue works in a service-oriented architecture (SOA) platform.

Keywords : Crawling, Scraping, Catalogue, SOA.

Tabla de contenidos

Índice de figuras

Índice de tablas

1. Introducción

"Hentai" es la palabra que se utiliza fuera de Japón para referirse al género al que pertenecen videojuegos, cómics, animaciones e ilustraciones de origen nipón con el estilo del manga de tipo erótico o pornográfico (Mangirón, 2012, p. 32).

El hentai aparece por una evolución de las obras *shunga*, un arte erótico japonés durante la era Edo (siglos XVI-XIX) (Burdzik, 2014, p. 343). El origen del hentai genuino es discutido: unas fuentes consideran como la pionera la obra de Go Nagai, *Harenchi Gakuen* (1968) (Mirón Martínez, 2017, p. 24) y otras hablan de *Senya ichiya monogatari* (1969), de Osamu Tezuka (Yegulalp, 2017).

A partir de la década de los setenta, su difusión no ha parado de crecer hasta lograr durante la mitad de la década del 2010 la fama mundial (Martínez Galán, 2017, p. 9). Es destacable el aumento de la producción y consumo de obras anime hentai durante los ochenta por el desarrollo del mercado de vídeos reproducibles en casa (Yegulalp, 2017), década en la que también se introdujo en Estados Unidos (Patten, 1998). Durante los noventa entraron a Estados Unidos los primeros mangas hentai traducidos al inglés (Perper & Cornog, 2015, p. 3).

En esta década, la del 2010, han aparecido y crecido tiendas en línea que venden cómics de este género (admin., 2014; «Digital Manga Plans Project-H Imprint with 3 Hentai Books», 2011; «Hentai Website Fakku Enters Publishing Business with Japan's Wanimagazine», 2014).[1]

Un consumidor puede encontrar placer en estas obras porque:

- Tienen temáticamente variedad y cierta complejidad narrativa que choca con el estereotipo que tienen los europeos y norteamericanos de la pornografía en general.[2]
- Las posibilidades creativas sobrepasan los límites de las fotografías y los fotogramas (Uidhir & Pratt, 2013, p. 8).
- Al representar personajes que de manera obvia son ficticios se provoca un sentimiento de superioridad moral al compararse con el consumo de pornografía no metonímica (Uidhir & Pratt, 2013, pp. 10-11).
- Como es una pornografía que muestra explícitamente una irrealidad es más fácil que sean consideradas arte estas obras por lo que los consumidores se sienten justificados de poseerlas y apreciarlas (Uidhir & Pratt, 2013, p. 11).

Ante el éxito que ha tenido el género hentai (Martínez Galán, 2017, p. 9), el cómic implica un objeto de interés para el público, lo que puede suponer una satisfacción para

[1] Se observa que una de las tiendas, 2D Market, se creó su cuenta de Twitter el 2012 («2D Market - Twitter», 2012).

[2] Aunque Barancovaitė-Skindaravičienė (Barancovaitė-Skindaravičienė, 2013, p. 9) lo dice de las series de animación, se puede extrapolar a los cómics.

el cliente potencial de estos productos el presentarle un catálogo agregado de la oferta en formato digital, teniendo en cuenta que el cliente potencial está globalizado con el acceso a Internet y el uso del inglés como *lingua franca*.

En este trabajo se presenta detalladamente la creación de un catálogo en línea en forma de API RESTful en el que se encuentre información bibliográfica y comercial básica de todos los cómics hentai digitales que se ofrecen a la venta en tiendas en línea (web).

Los objetivos de este trabajo son:

- Extraer datos bibliográficos y comerciales de los cómics hentai de las tiendas web.
- Permitir consultar esos datos mediante una API REST y una aplicación web que beba de esta API, una aplicación cliente.

El resto del documento se organiza de la manera siguiente:

- En la sección 2 se explican las tecnologías utilizadas en este trabajo.
- En la sección 3 se explican los pasos seguidos para el logro del trabajo.
- En la sección 4 se describe la arquitectura de la solución creada para el objetivo del trabajo.
- En la sección 5 se describen las fuentes explicando su historia y su contenido, se explica por qué son elegidas o por qué son descartadas, se describe los datos que aporta cada una y finalmente se presenta un modelo de datos a extraer y organizar a partir de los datos aportados por las tiendas.
- En la sección 6 se explica el proceso aplicado de extracción de datos (scraping).
- En la sección 7 se explica cómo se diseñó, creó y pobló la base de datos.
- En la sección 8 se explica cómo se diseñó la API REST, determinando las interacciones posibles con esta a partir de determinar las necesidades de los usuarios.
- En la sección 9 se explica cómo se creó una aplicación cliente de la API REST desarrollada en el paso anterior.
- En la sección 10 se resume el trabajo hecho, se determina cuánto se han cumplido los objetivos, se presentan futuros desarrollos posibles del proyecto logrado y se presenta una reflexión personal sobre lo aprendido durante el camino.

2. Contexto tecnológico

El contexto tecnológico para el desarrollo de este trabajo se resume en los siguientes programas informáticos.

2.1. Python

Python es un lenguaje de programación de propósito general que se utilizó en este trabajo. Éste tiene una licencia de software libre certificada (Challenger-Pérez, Díaz-Ricardo, & Becerra-García, 2014, p. 7) desde su primera versión, es un lenguaje interpretado y se caracteriza por su sencillez (Challenger-Pérez et al., 2014, p. 2). También es destacable la riqueza de sus bibliotecas de código estándar (Challenger-Pérez et al., 2014, p. 6).

Ha sido utilizado como herramienta para el aprendizaje y para crear conocidos programas de código libre como Blender y Gimp. Entre sus usuarios se cuentan grandes empresas como Google y Lucasfilm (Challenger-Pérez et al., 2014, pp. 9-10).

Las estructuras de datos que maneja, como lenguaje de alto nivel que es, son las listas, los conjuntos, los diccionarios y las tuplas (Challenger-Pérez et al., 2014, p. 3). Los diccionarios son una estructura similar al formato de datos JSON.

Su característica de ser un lenguaje interpretado facilita el trabajo de comprobación del funcionamiento de instrucción en instrucción (Challenger-Pérez et al., 2014, pp. 5-6). A esta ventaja se le añade la alta legibilidad del código desde el punto de vista humano (Challenger-Pérez et al., 2014, p. 5).

Se utilizó Python 3.6.1, bajo la distribución Anaconda Python 4.4.0 (para 64 bits). Esta distribución incluye muchas librerías de código (las cuales la Python Software Foundation llama paquetes, en inglés *packages*) ya instaladas.

2.2. Anaconda Python

Anaconda Python es un producto de Anaconda, Inc, de código abierto. Entre los elementos que incluye, destaco el entorno de desarrollo integrado Jupyter. La versión de Jupyter que utilicé fue la 4.3.0 y es útil por su cuaderno para edición de código Jupyter Notebook.

Jupyter Notebook (ver Figura 1) es un editor de código de varios lenguajes de programación, de entre los cuales nos interesa Python. Para la programación con Python utiliza la funcionalidad de iPython, un intérprete de órdenes que agrega funcionalidades a las propias de Python.

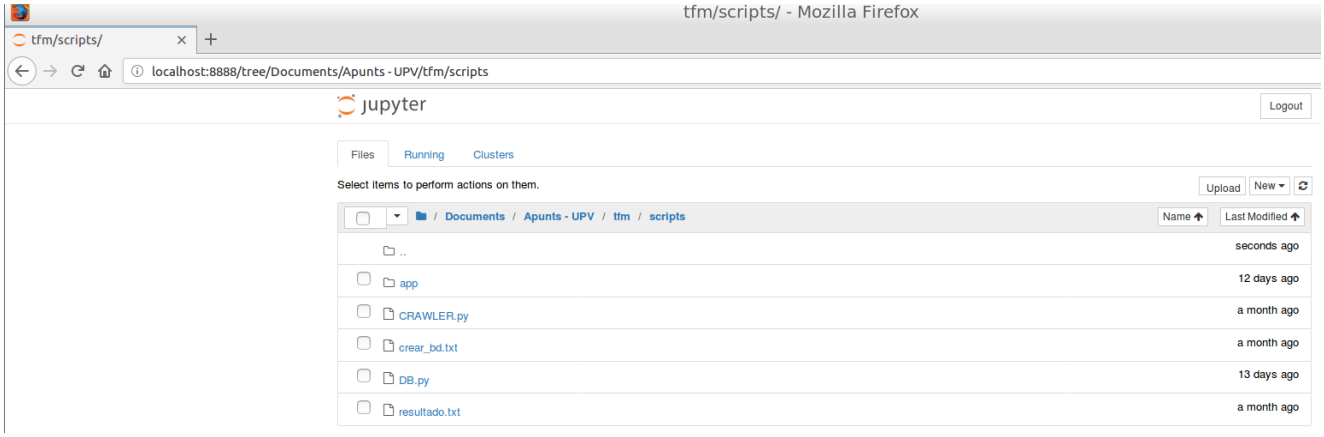

Figura 1. Captura del navegador de directorios de Jupyter.

Este editor de código permite editar ficheros con extensión python (.py) y ficheros con el formato para el notebook para Python (.ipynb). Cuando se trabaja con los ficheros .ipynb el código se reparte en celdas (ver Figura 2), aunque no necesariamente las celdas deben contener código. Cada celda se ejecuta de manera independiente a las otras.

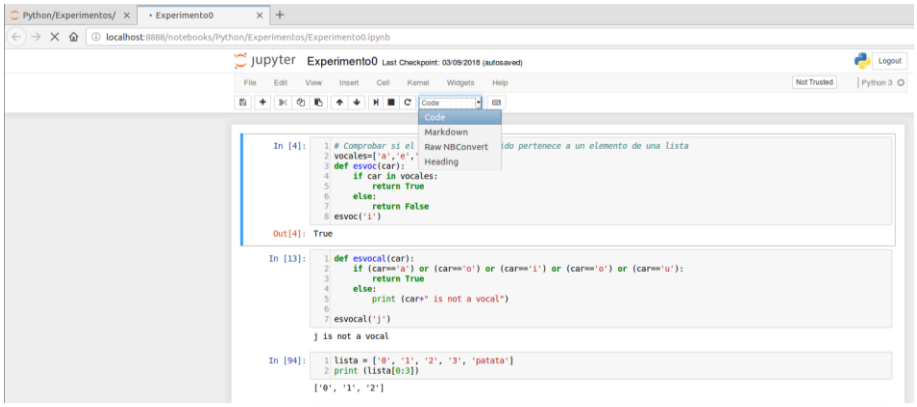

Figura 2. Un fichero .ipynb siendo leído por Jupyter. Nótese las posibilidades de formato de la celda a crear en la selección desplegable.

2.3. XAMPP for Linux

XAMPP es un grupo de software libre (Arriola Cruz, 2017, p. 142) que suelen ir juntos, lo que se llama un conjunto de soluciones. El grupo incluye: el servidor Web Apache, un gestor de base de datos (que solía ser MySQL y desde recientes versiones es MariaDB (Segarra Cabedo, 2016, pp. 21-22)) y los lenguajes de programación PHP y Perl (López Montalbán, Castro Vázquez, & Ospino Rivas, 2014, p. 18).

La versión que se utilizó fue XAMPP for Linux 7.1.10-0, el cual incluye 10.1.28-MariaDB, Apache Server 2.4.28, Perl 5.16.3 y PHP 7.1.10.

MariaDB es un sistema de gestión de bases de datos relacionales que derivó de MySQL, ambos gratuitos y de código abierto. Puede utilizarse mediante la interfaz gráfica web PhpMyAdmin (ver Figura 3).

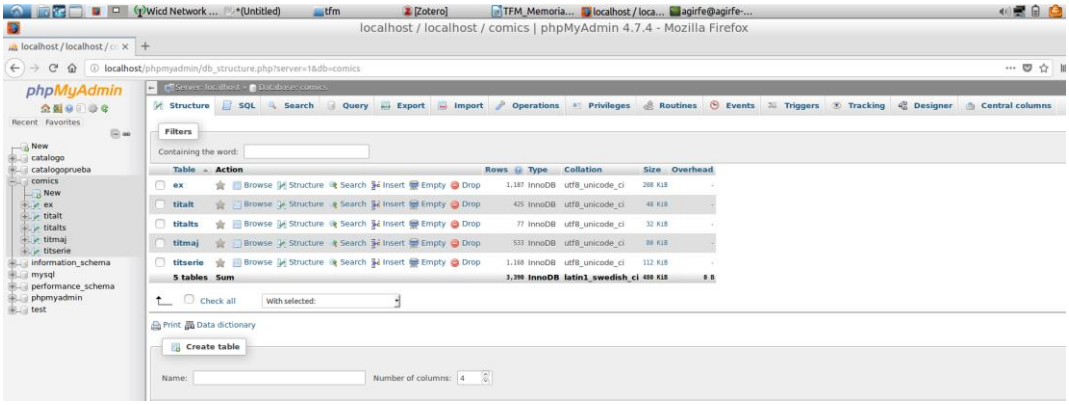

Figura 3. Pantallazo de la interfaz gráfica PhpMyAdmin.

Las bases de datos relacionales son conjuntos integrados de datos estructurados de manera coherente e interrelacionados. Aplican la lógica de predicados y la teoría de conjuntos, lo que las hace muy potentes (López Montalbán et al., 2014, p. 12). SQL (Structured Query Language) es un lenguaje de programación estandarizado por la ISO para manejar este tipo de bases de datos (López Montalbán et al., 2014, pp. 13, 16). Este tipo de base de datos es superado por otros tipos ante las necesidades de distribución de la base de datos en varias máquinas. Aun así, sigue siendo útil para trabajos que no requieren cantidades masivas almacenadas de datos (Judd, 2008).

PHP (PHP Hypertext Preprocessor) es un lenguaje de programación de alto nivel para el desarrollo web, también usado como lenguaje de propósito general. Es ejecutado por el lado del servidor, es decir desde donde se reciben las peticiones. Tiene una licencia de código abierto (Cruz Heras & Zumbado Rodríguez, 2003, p. 93).

PHP es especialmente útil para trabajar con MySQL (y por tanto con MariaDB) porque incluye funciones predefinidas para este fin (Cruz Heras & Zumbado Rodríguez, 2003, p. 135).

2.4. JavaScript

JavaScript es un lenguaje de programación basado en objetos creado el 1995. Entre sus características (Pérez Castaño, 2014, p. 29), destaco que está orientado a eventos,[3] y que es utilizado para generar código HTML, extendiendo las capacidades del lenguaje de marcado (Pérez Castaño, 2014, p. 32). De hecho, la última revisión de HTML, HTML5, permite la posibilidad de insertar scripts de JavaScript (Refsnes Data, s. f.).

[3] Los eventos son acciones causadas por la interacción del usuario.

3. Metodología

Los pasos generales para la creación del catálogo y la puesta en marcha de sus servicios de consulta fueron los siguientes (ver Figura 4):

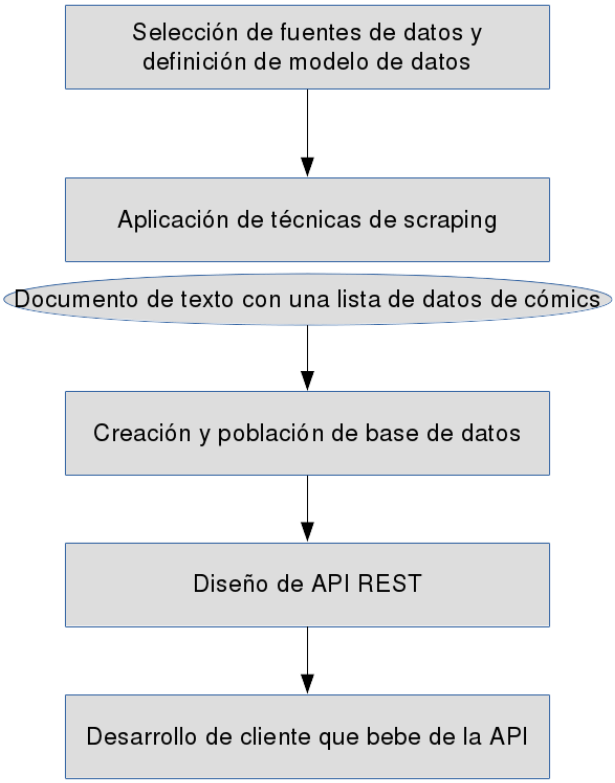

Figura 4. Esquema de la metodología

(1) Selección de las fuentes de datos: Se listan las tiendas que cumplen el requisito básico, se seleccionan las que se consideran útiles, se analiza los datos que proporcionan y se define un modelo de datos para los datos a extraer.

(2) Aplicación de las técnicas de extracción de datos: Se programan y ejecutan programas sobre las webs de las tiendas elegidas en el paso anterior. El resultado, una lista de listas de datos sobre los cómics, se guarda en un documento de texto.

(3) Creación y población de una base de datos: Se diseña y crea una base de datos y luego se guardan los datos resultado del paso anterior.

(4) Diseño de una API REST que interactúa con la base de datos creada en el paso anterior. Para la creación del servicio REST se analizan los datos disponibles y se consideran las necesidades.

(5) Prueba de concepto desarrollando un cliente de búsqueda que utilice la API REST.

4. Arquitectura de la solución

La arquitectura general de la solución que se presenta en este trabajo está esquematizada en la siguiente figura:

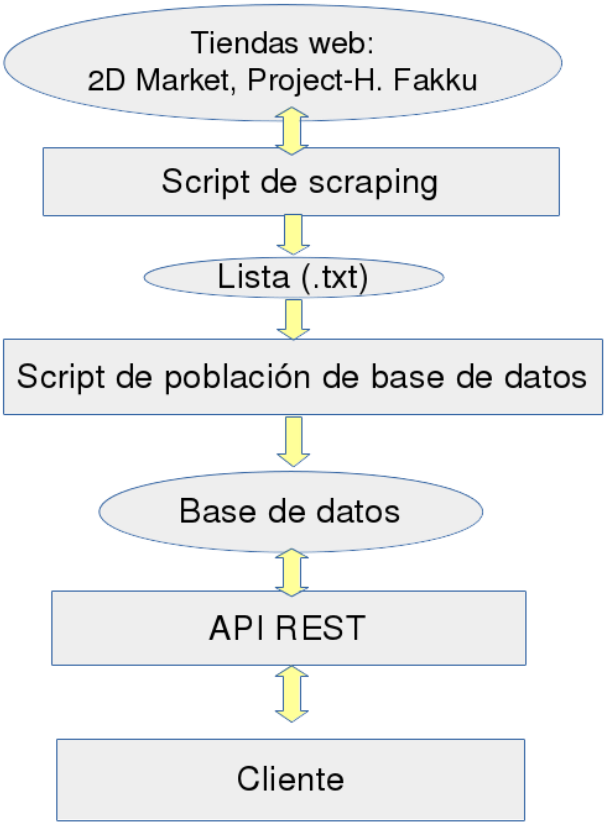

Figura 5. Arquitectura de la solución.

El script de extracción de datos (hecho con Python) se conecta y lee las páginas web, extrae los datos y los guarda en forma de lista de Python en formato txt. Otro script hecho con Python se conecta con el gestor de base de datos (MySQL) y puebla la base de datos. Un script de Python pone en funcionamiento la API REST, interactuando con la base de datos a través de las URIs. La API REST devuelve resultados en formato JSON. Cuando esta API está en funcionamiento junto al gestor de la base de datos, el cliente, que consiste en un documento HTML que ejecuta código Javascript, interactúa con la API presentando los resultados en formato HTML en el mismo documento.

5. Selección de fuentes y definición de modelo de datos

A partir de conocer las tiendas de cómics digitales existentes, se seleccionaron y cribaron las tiendas que cumplían los requisitos de este trabajo. Los datos presentados por las tiendas que habían pasado la selección fueron examinados para definir un modelo de datos.

5.1. Selección de fuentes

Las fuentes de las que extraer los datos son las tiendas de cómics hentai digitales, o al menos principalmente digitales.

A partir de unas búsquedas web y una revisión bibliográfica [4] se detectaron las siguientes tiendas que podían cumplir el perfil:

- 2D Market (https://2d-market.com/)

2D Market es una tienda especializada en obras originalmente autopublicadas (en japonés *doujinshi*) en formato digital. Está gestionada por Digitalna Knjižara Multimedije, una empresa croata.

- Fakku (https://store.fakku.net/)

Fakku es una tienda estadounidense que vende cómics hentai en versión digital y en versión impresa. Llega a incluir en su catálogo algún cómic que no es hentai.

Se inició como sitio web pirata especializado en cómics hentai el 2007. Se convirtió en una plataforma respetuosa con los derechos de propiedad intelectual a partir del 2014 a causa del acuerdo con Wanimagazine.

- Project-H (https://www.projecthentai.com/)

Project-H (también llamado Project Hentai y Project-H Books) es una tienda gestionada por Digital Manga, una editorial estadounidense. Vende cómics principalmente disponibles en formato digital y también vende algunos solamente en formato físico.

Opera desde el 2011.

- eManga (https://www.emanga.com/)

Tienda que al igual que Project-H está gestionada por Digital Manga. Opera desde el 2008 y ofrece cómics manga digitales de muchos géneros, incluyendo el hentai.

- SuBLime Manga (https://www.sublimemanga.com/)

[4] La revisión bibliográfica que hice la publiqué en el artículo de la Wikipedia en catalán sobre Hentai bajo el nombre de usuario Hienafant.

Tienda especializada en el género yaoi, tanto de contenido sexualmente explícito como no explícito, que vende en formato digital y, en algunos, en formato impreso. El yaoi es el género de manga y anime centrado en relaciones románticas y/o sexuales homosexuales entre hombres. Está gestionada por Viz Media, una empresa estadounidense.

Se descartaron las dos últimas tiendas por distintas razones: en eManga se avisaba que las obras pertenecientes al género serían trasladadas a Project-H Books y en SuBLime Manga no explicaban si los cómics considerados "Mature" necesariamente contienen escenas suficientemente explícitas para cumplir la definición de hentai que se exige en este trabajo. Se envió un correo a SuBLime Manga pidiendo la definición concreta de la certificación, pero nunca fue respondido.

Se observó que en algunos casos había cómics que solamente se vendían en formato físico, por lo que se tomó la decisión de exceptuar el cumplimiento estricto del objetivo del trabajo (referido a cómics digitales) e incluirlos. Ésto obedece la necesidad de no complicar el trabajo.

5.2. Análisis de los datos disponibles

Se analizaron los datos bibliográficos y comerciales disponibles de cada cómic vendido en la tienda para determinar el modelo de datos a extraer.

Hay dos tipos de datos que son comunes a todas las tiendas: cada cómic es vendido por una tienda y tiene una URL.

En 2D Market los tipos de datos ofrecidos son: el título traducido, el título original, el responsable de la autoría (en este caso suelen ser círculos de autores), la descripción, los tags, las categorías, el año de publicación, el precio, el número de páginas y la fuente de inspiración.

La fuente de inspiración son los nombres de las obras originales en las que se basan las versiones propias de los autores, en el caso en el que la obra esté inspirada. Por ejemplo: en *Half of Prism* la autoría pertenece al círculo Kabayakiya y los personajes provienen de varias obras, entre otras *Sword Art Online*, una obra de manga y anime creada por autores ajenos al círculo Kabayakiya.

Hay casos en 2D Market en los que el título traducido tiene dos presentaciones. Este caso se da en los cómics que pertenecen a una serie, siendo los cómics los capítulos de la serie. El título presenta dos formas: la forma que consiste solamente en el título del capítulo y la forma que presenta el nombre del capítulo en el contexto de la serie. El título traducido se presenta solamente bajo la segunda forma. Un ejemplo es el capítulo *Sleeping Beauties* de la serie *Victim Girls*, que es presentado como *Sleeping Beauties* y como *Victim Girls Ch.14: Sleeping Beauties*.

En Fakku solamente interesan los productos en venta que no formen parte de la suscripción. Los tipos de datos ofrecidos son distintos según si son obras doujin o libros (cada uno de esta clase se encuentra en la sección correspondiente).

En las obras doujin se ofrecen los siguientes tipos de datos: el título traducido, el responsable de la autoría (que incluye autor y círculo al que pertenece en campos separados), el número de páginas, el idioma, si tiene censura, la fuente de inspiración y el precio. En algunos casos se añaden los datos como el nombre de la editorial (por ejemplo *Asuna Strategy guide*) y el evento en el que se presentó el cómic (es el caso de *Erina's Secret Recipe*). Hay un caso, *Hurl Ryuu*, en el que solamente se muestran el título traducido y el precio.

En las obras que son libros los tipos de datos son: el título traducido, la fecha de publicación, el responsable de la autoría, el número de páginas,, el número de capítulos, el número de páginas a color, las características extras añadidas, el precio, la editorial, el idioma y si está censurado. Hay casos, como *Urotsukidoji : Legend of the Overfiend Volume 3* en el que no se dan el número de capítulos ni el número de páginas a color, aunque en este último tipo de datos se justifica porque no hay páginas a color. Hay un caso parecido al de *Hurl Ryuu* de la sección Doujin, que es *Metempsychosis*, en el que se muestra el título traducido, una descripción y el precio. No siempre aparece una descripción.

En el caso del dato de las editoriales se observa que hay obras editadas por 2D Market que se venden en Fakku (por ejemplo *Anna & Witch's Tentacles*).

En Project-H los tipos de datos que se ofrecían eran: el título traducido, el título original escrito en escritura japonesa y en rōmaji, el precio, la descripción, si hay censura, el responsable de la autoría (solamente muestra a los autores, no los círculos a los que pertenecen) y los tags. En algún caso, como *Cute Devil Girlfriend 2nd Edition*, en la descripción se muestra una lista de los capítulos.

En las tres tiendas se informa en general de si la obra no tiene censura, Fakku informa mediante un campo específico, 2D Market y Project-H mediante un tag “Uncensored”.

Se recogió cada tipo de datos y su presencia mayoritaria en cada tienda para así poder determinar qué datos escoger para el modelo de datos en la siguiente tabla:

Tipo de datos	**Número de tiendas en las que aparece el tipo de datos en la mayoría de casos**
Título traducido	3/3
Título original	2/3
Responsable de autoría	3/3
Descripción	2/3

Tags y categorías	2/3
Fecha de publicación	2/3
Precio	3/3
Número de páginas	2/3
Fuente de inspiración	2/3
Número de capítulos	1/3
Número de páginas a color	1/3
Censura	3/3
Idioma	2/3
Editorial	1/3
Evento	1/3
Características extra	1/3

Tabla 1. Tipos de datos en tiendas.

5.3. Modelo de datos

A partir de la información anterior y los criterios dados se elaboró el modelo de datos.

El criterio general para el modelo de datos es que los tipos de datos sean comunes a todos los casos a ser posible o en casi todos los casos, que los datos sean de interés y que incluyan datos comerciales y bibliográficos.

El modelo de datos (ver Figura 6) consiste en que cada cómic tiene una **tienda**, un **enlace**, un **título**, un **autor**, un **número de páginas**, la indicación de si hay alguna **censura** y los **precios**.

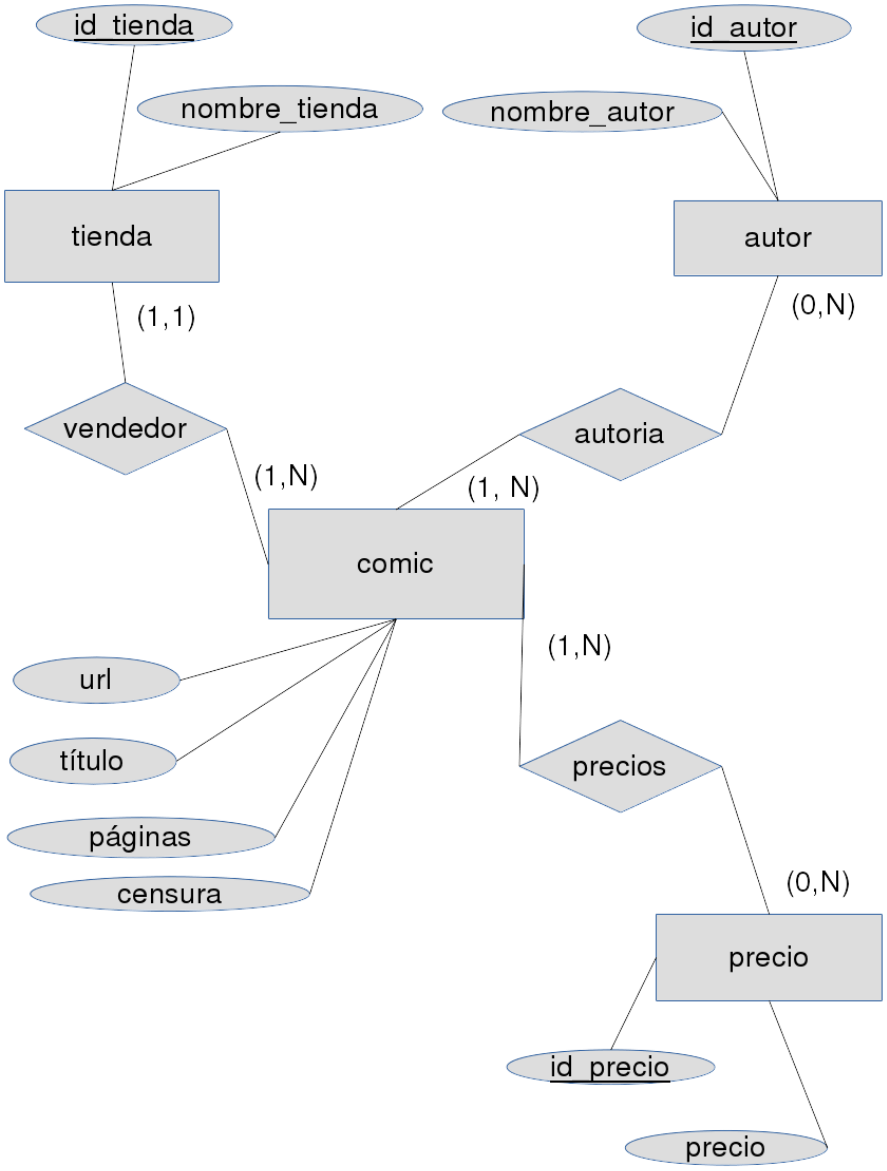

Figura 6. Diagrama entidad-relación.

En el modelo de datos se entiende que cada cómic es un producto ofrecido por una sola tienda. Se estableció esta declaración en el modelo porque en los casos de un mismo título ofrecido por varias tiendas ocurre que el precio varía (probablemente por razones de plusvalía). Por lo tanto, el cómic definido en el modelo de datos se identifica unívocamente como un título vendido en una tienda, y no un título que puede ser vendido en una o más tiendas.

El título que se establece en el modelo es el título traducido. Considerar el título original, además de no estar presente en las tres tiendas, supone añadir complejidad. Para los casos de la tienda 2D Market como el de *Sleeping Beauties* se resuelven eligiendo el nombre con contexto por tratarse de títulos con información añadida sin ser considerados un exceso de información.

El **autor** se establece que tiene que ser el autor y el círculo al que pertenece (éste como si fuera otro autor más). Si solamente se diese el círculo de autores, se añadiría el círculo de autores. Se unifica bajo autor tanto la persona autora como su círculo para evitar complicaciones por la inconsistencia entre datos aportados por tiendas.

El **número de páginas** se considera que es un elemento cuantitativo para calcular el precio merecido al ponerlo en relación con los precios ofrecidos. El hecho de que no esté presente en la tienda Project-H no supone un inconveniente por ser un dato bastante importante.

La **censura** indicará si está totalmente sin censura, si no lo está o si no se indica si lo está totalmente o no.

Los **precios** indican no solamente un número de unidades monetarias, sino también el formato y/o soporte al que va ligado, en caso de que se especifiquen.

En cuanto a los tipos de datos descartados:

- Los títulos traducidos solamente se consideraría apropiado añadirlos en el caso de que la catalogación automática tuviera un modelo de datos en el que el cómic se definiría como título que puede encontrarse en varias tiendas, volviendo innecesariamente complicado el modelo de datos. La comparación de los precios de los títulos vendidos en más de una tienda se hace posible con este modelo de datos al buscar por autor (concretamente buscando por círculo de autores).
- La descripción y la fecha de publicación no están presentes en todos los casos. La fecha no supone un dato de mucho interés y la descripción puede dar problemas para la población de la base de datos.
- Los tags y las categorías no están presentes en las tres tiendas, por lo que la búsqueda por etiqueta supondría un sesgo grande en detrimento de la tienda Fakku.
- La fuente de inspiración está presente en un solo tipo de género de los cómics. Añadir este tipo de datos supondría tener que diferenciar los cómics, haciendo complejo de manera innecesaria el modelo de datos.

- El número de capítulos y el número de páginas a color no están presentes en todos los casos y no añaden mucha más información interesante a la que presenta el total de páginas.
- El idioma se supone que es el inglés en todos los casos, de manera que no aporta nada.
- La editorial no es un dato interesante para el comprador y no aparece en todas las tiendas.
- El evento en que se presentó el doujinshi es una característica que solamente aparece en un tipo de cómic, cosa que complicaría el modelo de datos de ser aceptado. Además no es interesante para el comprador.

6. Extracción de los datos

La extracción de datos web, en inglés *web data extraction,* también conocida como cosecha web (*web harvesting*) y *web scraping*, es un conjunto de técnicas automáticas mediante el uso de programas para extraer información de los sitios web (Vargiu & Urru, 2012, p. 44). Estas técnicas recogen datos encontrados en páginas web de manera más o menos guiada para reunirlos en un lugar de manera estructurada para poder trabajarlos más cómodamente.

La aplicación de estas técnicas siguen estos pasos: el acceso al sitio web mediante el protocolo HTTP (Glez-Peña, Lourenço, López-Fernández, Reboiro-Jato, & Fdez-Riverola, 2013, p. 789), el análisis (*parsing*) y extracción del código HTML y finalmente la presentación estructurada de los datos extraídos (Glez-Peña et al., 2013, p. 790).

Un ejemplo de aplicación de las técnicas de scraping sería el descrito en la siguiente figura:

Figura 7: Ejemplo de aplicación de técnicas de web scraping.

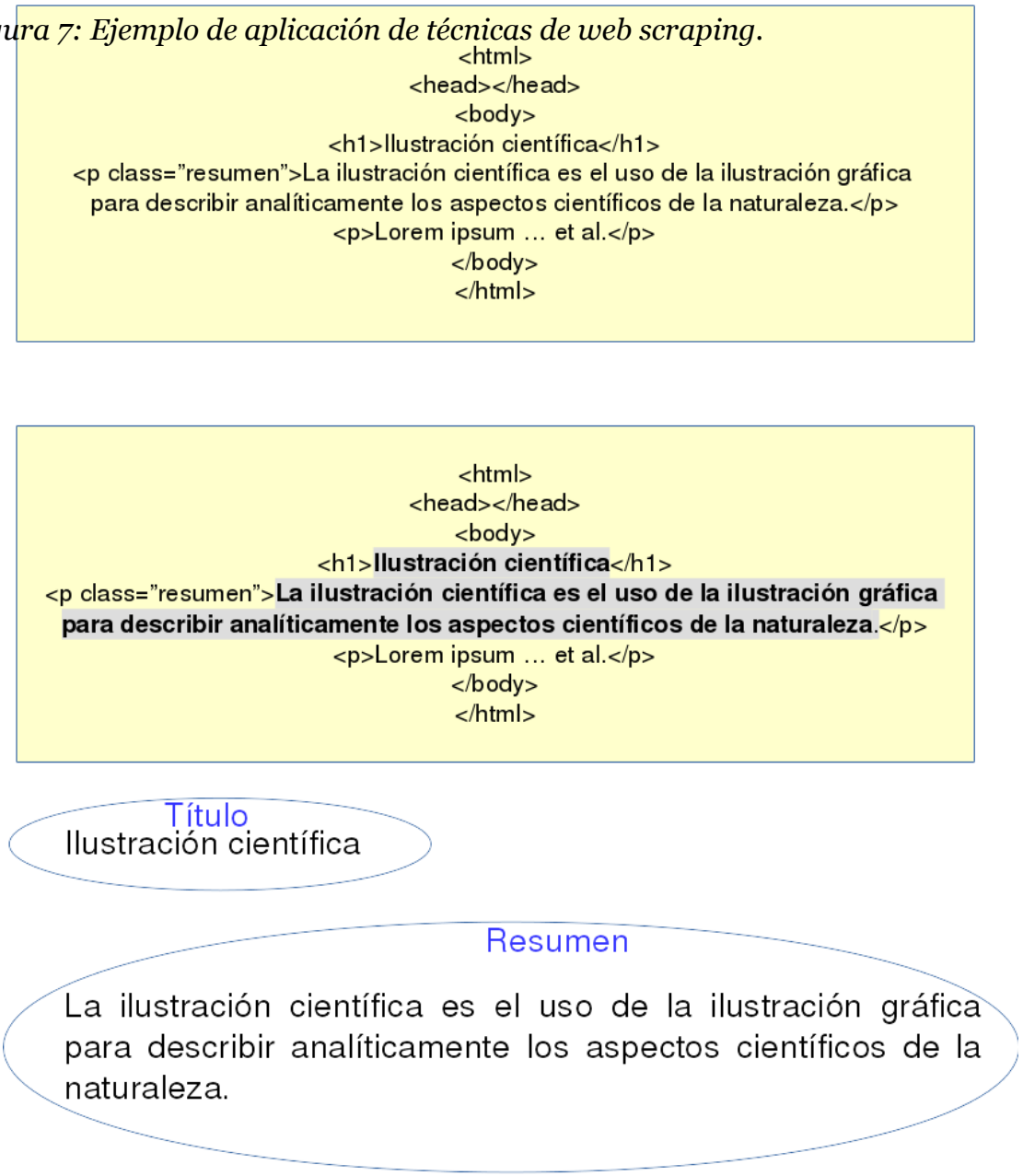

Figura 7. Ejemplo de aplicación de técnicas de web scraping.

A partir de un documento web en HTML, el programa detectaría el texto interesante, lo

clasificaría y lo guardaría, de manera que en el resultado del proceso se pierde texto (o caracteres) que no interesan y se puede acceder directamente a los datos según han sido categorizados en Título o Resumen.

El programa ejecutor se le suele llamar *web scraper* o *web robot* (Glez-Peña et al., 2013, p. 789) y puede consistir en un programa hecho a medida mediante el uso de bibliotecas de código, como por ejemplo BeautifulSoup para Python; entornos de trabajo (*frameworks*), como Scrapy para Python, o programas sueltos, como Screen-scraper (Glez-Peña et al., 2013, pp. 789-790) y Kimono (VanHemert, 2014). Cuando se producen cambios en las webs, los robots deben ser actualizados para que funcionen correctamente, lo que los vuelve en programas que requieren una actualización constante (Glez-Peña et al., 2013, p. 791).

En un caso abstracto, estas técnicas son útiles para pasar de los datos encontrados en una fuente de información no estructurada (como son los archivos HTML) a tener estos datos de manera estructurada almacenados en algún sitio de destino (Vargiu & Urru, 2012, p. 44). En casos más concretos, se utilizan para elaborar mashups webs, control de cambios en sitios web y comparación de precios en línea, entre otros (Vargiu & Urru, 2012, p. 45). Son una de las técnicas aplicadas en el tratamiento de los datos masivos (Polidoro, Giannini, Conte, Mosca, & Rossetti, 2015, p. 166).

La utilidad del scraping para este trabajo radica en que, ante la gran cantidad de páginas web por visitar para recopilar exhaustivamente los datos del mercado de cómics hentai, un proceso automático de recogida de datos permite recoger cíclicamente los datos.

Los pasos para la extracción de los datos fueron los siguientes:

1) Reunir los enlaces de los cómics.
2) Lograr el acceso a las páginas web a través de los enlaces.
3) Lograr los datos de los cómics
4) Guardar los datos en variables.
5) Almacenar los datos en un documento de texto (.txt).

Se utilizó el lenguaje de programación Python 3.6.1 con la ayuda de la plataforma Anaconda 4.4.0.

La extracción se hizo el 28 de abril de 2018, quedando como suficientemente satisfactorio el trabajo de diseño del script hecho hasta ese momento. A partir de algún momento en el tiempo posterior a la ejecución del script de extracción de los datos hubo un cambio importante en la web de Project-H, ya que hicieron que el contenido de la etiqueta body fuera cargada con Javascript, cosa que dificultaría la acción del script con mis conocimientos técnicos.

6.1. Preparación de las páginas web

Para extraer los datos (conectar a la página web, leer el código, copiar y guardar los datos a extraer), se necesita primero tener estos enlaces para que la máquina los pueda procesar. Estos enlaces los listamos, ya que las listas son una estructura de datos ofrecida Python que resulta sencilla y suficiente.

Para reunir los enlaces mediante listas de URLs se definió una función que partía de uno o varios enlaces iniciales en los que la máquina inicia su navegación para su recolección.

Se utilizaron las bibliotecas re (para utilizar expresiones regulares para la búsqueda dentro de las páginas web), urllib.request (para pedir ("GET") mediante el protocolo HTTP las páginas web) y beautifulsoup4 (para buscar etiquetas de HTML, complementando la funcionalidad de la biblioteca re).

Se creó un algoritmo especializado en cada tienda.

La conexión no dio ningún error HTTP en ningún caso, temiendo especialmente el error 401 (conexión no autorizada).

En cada tienda se buscó manualmente las páginas suficientes para la recolección de las URLs. Estas páginas eran secciones del sitio web de la tienda en las que se muestran los enlaces a las páginas donde se localizan los datos a extraer. En el código son llamadas URLs iniciales.

Para determinar estas páginas me fijé en el código de la página para saber guiar los algoritmos lo más eficientemente que se me ocurriera.

6.1.1. Fakku

En el script la función fue nombrada getlinks_F y es descrita con el siguiente pseudocódigo:

```
1  Conecta con enlace inicial

2  Lee el código html como texto

3  Lista de enlaces vacía a llenar

4  Si el enlace inicial es el de la sección Book

5        Se añaden los enlaces que sigan la estructura
   "/products/caracteres" a la lista de enlaces

6  Fin

7  En caso contrario los enlaces iniciales son
   ['https://store.fakku.net/collections/doujinshi?page=1',
   'https://store.fakku.net/collections/doujinshi?page=2']

8        Se conecta y lee como texto en cada de estos enlaces
```

```
9       Se añaden los enlaces que sigan la estructura
   "/products/caracteres" a la lista de enlaces
10 Fin
11 Se asegura que no haya enlaces repetidos en la lista de enlaces
12 Se devuelve la lista de enlaces sin repetidos
```

En el caso de Fakku se encontró primero que en el subdominio store se encontraban los enlaces a los productos de nuestro interés: productos que se vendían independientemente de la suscripción.

Para diferenciar automáticamente con mis capacidades técnicas los enlaces que nos interesaban de los que no, no se debía tener en cuenta la construcción de la URL sino en qué sección se encuentran. Así, en las secciones Books (https://store.fakku.net/collections/manga-doujinshi) y Doujin (https://store.fakku.net/collections/doujinshi) solamente había cómics hentai digitales no efímeros. Por tanto, no sujetos a suscripción, sino a compra final.

Para tener los datos que nos interesan se determinó que se ejecutaría el algoritmo empezando por la página web de las secciones Books (https://store.fakku.net/collections/manga-doujinshi) y Doujin (https://store.fakku.net/collections/doujinshi). De estas secciones, la que contiene todos los productos en una sola carga, es decir, no requiere cargarlos usando el scroll ni navegando por páginas, solamente es la sección Books. La segunda presentaba dos páginas de navegación que inserté manualmente en el algoritmo.

Para el funcionamiento del algoritmo, desde estas secciones (estas páginas web introducidas en la función) se permitió recoger solamente los enlaces que empiecen por “/products/”.

Para evitar la repetición de enlaces la lista cruda resultante fue convertida en un conjunto y luego otra vez en una lista.

6.1.2. 2D Market

En el script la función fue nombrada getlinks_2D y es descrita con el siguiente pseudocódigo:

```
1  Conecta con enlace inicial
2  Lee el código html como texto
3  Lista de enlaces de navegación vacía a llenar
4  Se selecciona el código con los enlaces de navegación
5  Se guardan los enlaces de navegación en la lista de enlaces
6  Seleccionar el último enlace, que es la última página de
   navegación
```

```
7  Se extrae el número de esta página de navegación

8  Se vacía la lista de enlaces de navegación

9  A partir de este número se llena la lista con todas las páginas
   de navegación mediante una cuenta atrás que termina con el
   número 1

10 Se crea una lista vacía de enlaces de cómics

11 Se accede a cada enlace de la lista de enlaces de navegación y
   se guardan los enlaces de cada cómic en la lista de enlaces de
   cómics

12 Se asegura que la lista de enlaces de cómics no contenga
   elementos repetidos

13 Devuelve esa lista
```

El hecho que todos los productos fueran cómics objetivo facilitó el trabajo.

Como los productos estaban dispuestos en una lista de navegación se hizo que el algoritmo:

- Primero indexara las páginas de navegación sobre la primera página de la sección Browse (https://2d-market.com/Browse), pudiendo inferir todas las páginas de navegación a partir de sacar cuál es la última página. Estos enlaces se guardarían en una lista de páginas de navegación.
- Y luego, desde cada página de esta lista se conseguirían todos los enlaces de los cómics, los cuales empezaban por "/Comic/", un número y un guión más posibilidad de texto.

Los enlaces de los cómics podían acortarse utilizando solamente el número. Así, por ejemplo https://2d-market.com/Comic/89-Noshiron-Commandeered es tan válida como https://2d-market.com/Comic/89. Se recogieron solamente las URLs en su versión más corta para la eficiencia del procesamiento y para la posterior población de la base de datos.

6.1.3. Project-H

En el script la función fue nombrada getlinks_PH y es descrita con el mismo pseudocódigo que 2D Market.

El caso de Project-H tiene los mismos elementos básicos a considerar para reunir las URLs que 2D market: todos los productos fueron cómics objetivo y estaban dispuestos en una lista de navegación.

Los cómics se caracterizan por seguir la forma "/collections/all/products/" más el nombre del cómic.

6.2. Ejecución de las técnicas de extracción

Una vez ya reunidos en listas los enlaces de cada cómic, se procedió a conectarse a la página web, leer y analizar el código, y guardar los datos que son el objetivo de la extracción.

Para la ejecución de estas técnicas, se utilizó, junto a las bibliotecas de código usadas en la preparación de las páginas web, la biblioteca time. Esta biblioteca fue utilizada para impedir saltarse los límites de peticiones por segundo impuestos por los servidores que almacenan los sitios web de las tiendas.

Se creó un algoritmo especializado en cada tienda.

Tal como se esquematiza en la siguiente tabla, se creó una lista (nivel 0) formada por listas (nivel 1) en la que cada lista (nivel 1) recogiera los datos de cada producto siguiendo el esquema establecido en el modelo de datos: tienda, enlace, título, autores, número de páginas y precios. En el caso de autores y precios podía darse el caso de haber varios autores o precios por lo que el elemento "autores" y "precios" resultaron en otras listas (nivel 3).

Nivel de profundidad	**Contenidos**						
0	**Lista de listas**						
1	**Lista por producto**						
2	**Tienda**	**Enlace**	**Título**	**Autores**	**Páginas**	**Censura**	**Precios**
3				**Autor**			**Precio**

Tabla 2. Esquema de niveles de profundidad.

Cuando una característica estaba ausente se insertaba el valor EMPTY.

Dado que el resultado era una lista muy grande, se hacían controles de calidad del algoritmo comprobando: que el número de productos (Nivel 1) coincidiera con el número de enlaces conseguidos con los algoritmos que reúnen enlaces de los cómics y que el número de elementos en la lista de cada producto (Nivel 2) fuera de 7 elementos.

6.2.1. Fakku

Tuve que corregir varias veces el algoritmo por las particularidades que se encontraron.

Los casos arquetípicos eran el de *Pandemonium* en la sección Books y el de *Living with Succubus 4* en la sección Doujin.

Había dos casos sin datos más allá del título traducido y el precio, ya nombrados en la sección 4.2 Análisis de los datos disponibles. Se resolvieron con la inserción del valor EMPTY donde no estaba disponible el dato.

Otros casos muy particulares fueron el de *Hunt Hunt 1*, con la etiqueta span donde estaba el nombre del autor diferente al resto de casos con la etiqueta span por contener un atributo anómalo llamado mce-data-marked y el de los cómics de la sección Doujin que eran como *Victim Girls 22: Queen Kashima's Training Journal*, con la etiqueta td que contiene el nombre del artista y el del círculo con un atributo, cosa diferente a muchos otros cómics de la sección Doujin.

El algoritmo se cambió porque las etiquetas con las que se había pensado que contenían la descripción del campo Artist y Circle no eran solamente la etiqueta strong dentro de un td, sino que también había algún caso en el que estaban directamente dentro de la etiqueta td sin una etiqueta strong. También los datos de los nombres de los autores a veces estaban dentro de una etiqueta span y otras directamente dentro de una etiqueta td.

En el script la función fue nombrada scraper_F y es descrita con el siguiente pseudocódigo:

```
1  Se introduce la lista de enlaces logrado con getlinks_F, llamada
   "getlinks"

2  Se crea la lista de cómics vacía list_fakku

3  Para cada enlace de getlinks:

4        Se crea una lista vacía temporal (list_temp)

5        Se conecta a la página web y se lee el código HTML como
   string

6        Se añade "Fakku" y el enlace a la lista vacía temporal

7        Se busca el título y se añade

8        Se recoge en la variable elementos_tabla los elementos de
   la etiqueta td como string

9        Si esta variable está vacía, se añaden tres "EMPTY"
   consecutivos

10       Si no está vacía:

11             Se divide tomando como separador ", <" creando una
   lista llamada elementos_tabla

12             Para cada td de elementos_tabla:

13                   Si el td tiene ">Artist</":

14                         Se define la variable artist que contiene
   el td siguiente al que contiene ">Artist</"

15                         Se busca en artist si hay contenido
   dentro de la etiqueta span. Esto mediante la variable tipo_span

16                               Si tipo_span está vacío, se busca
   dentro de "></td" los artistas y se crea una lista de autores
```

```
   dividiendo por ", " que se añade esta lista de autores a
   list_temp

17                              Si tipo_span no está vacía, se
   busca dentro de "></span" los artistas y se crea una lista de
   autores dividiendo por ", " que se añade esta lista de autores a
   list_temp

18                  Si el td tiene ">Circle</":

19                          Se define la variable circle que contiene
   el td siguiente al que contiene ">Circle</"

20                          Se busca en circle si hay contenido
   dentro de la etiqueta span. Esto mediante la variable tipo_span

21                              Si tipo_span está vacío, se busca
   dentro de "></td" los artistas y se crea una lista de autores
   dividiendo por ", " que se añade esta lista de autores a
   list_temp

22                              Si tipo_span no está vacía, se
   busca dentro de "></span" los artistas y se crea una lista de
   autores dividiendo por ", " que se añade esta lista de autores a
   list_temp

23                  Si el td tiene ">Inside the Book</":

24                          Se define la variable pages que contiene
   el td siguiente al que contiene ">Inside the Book</"

25                          Se selecciona lo contenido entre "><"

26                          Se seleccionan los números contenidos

27                          Se selecciona el número más grande

28                          Se añade este número a list_temp

29                  Si el td tiene ">Uncensored</":

30                          Se define la variable uncensored que
   contiene el td siguiente al que contiene ">Uncensored</"

31                          Se selecciona lo contenido entre "><"

32                          Si este contenido es "Yes", se añade "T"
   a list_temp

33                          Si este contenido es otra cosa, se añade
   "F" a list_temp

34                  Se define la variable elementos_tabla_juntos
   juntando los elementos de la lista elementos_tabla

35                  Si ">Uncensored</" no se encuentra en
   elementos_tabla_juntos, se añade "EMPTY" a list_temp
```

```
36       Se recogen las etiquetas option en la variable options
37       Se crea una lista de precios vacía
38       Para option en options:
39             Se convierte en string cada option
40             Se eliminan espacios en blanco de distintos tipos
41             Se añade a la lista de precios la lista de option
42       Se añade la lista de precios a list_temp
43       Se añade la list_temp a list_fakku
44       Se espera 0,8 segundos
45 Devuelve list_fakku
```

6.2.2. 2D Market

El pseudocódigo que explica el funcionamiento de la función es el siguiente:

```
1  Se introduce la lista de enlaces logrado con getlinks_2D, llamada "getlinks"
2  Se crea una lista de cómics vacía llamada list_2D
3  Para cada enlace de getlinks:
4        Se conecta al enlace y se lee el código html como string
5        Se crea una lista de enlaces vacía llamada link_temp
6        Se le añade a link_temp el nombre de la tienda y el enlace
7        Se reúnen las etiquetas h2 con la variable h2_tag
8        Se lee h2_tag como string
9        Se divide h2_tag por ", " convirtiéndolo en una lista
10       Se define la variable area_title como el primer elemento de h2_tag
11       Para searchtitle en los grupos de caracteres encontrados entre "<span" y "/span>" en area_title:
12             Se busca dentro de searchtitle contenido entre "<a" y "</a>" definido como string en la variable area_title_serie
13             Si area_title_serie está vacía:
14                   Se define la variable total_matched que incluye los caracteres que se encuentren entre ">" en la variable searchtitle
15                   Añade total_matched a list_temp
```

```
16            Si no:
17                  Se define title_serie como los caracteres encontrados dentro de “a></a>” en la variable area_title_serie
18                  Se define title_cap como los caracteres encontrados entre “a>” y “</span” en la variable searchtitle
19                  Se define la variable title como un string vacío
20                  Se reescribe la variable title como una combinación de title concatenado con title_serie y title_cap
21                  Se añade title a list_temp
22      Se define la variable author conteniendo en ella la etiqueta span con el atributo itemprop con el valor author
23      Dentro de author se busca los caracteres que estén entre “<meta content="” y “" itemprop="name">”, definidos con la variable autor
24      Se divide por “ and “, convirtiendo autor en una lista
25      Se añade autor a list_temp
26      Se busca en el html de toda la página los caracteres que estén entre “<span itemprop="numberOfPages">” y “<\/span>”, definidos con la variable pages
27      Se añade pages a list_temp
28      Se define la variable span_kw_tag conteniendo en ella la etiqueta span con el atributo itemprop con el valor keywords
29      Se busca en span_kw_tag si contiene “>Uncensored<”, definiendo esta búsqueda en la variable uncensored
30      Se busca en span_kw_tag si contiene “>Partially uncensored<”, definiendo esta búsqueda en la variable partiallyuncensored
31      Si ambas variables definidas anteriormente están vacías, se añade “EMPTY” a list_temp
32      Si aparecen ambas variables, se añade “F” a list_temp
33      Si solamente la variable partiallyuncensored está vacía, se añade “T” a list_temp
34      Si solamente la variable uncensored está vacía, se añade “F” a list_temp
35      Se define la variable p_offers_tag conteniendo en ella la etiqueta span con el atributo itemprop con el valor offers
36      Se crea la lista vacía list_style
```

```
37      Para cada price de cada resultado de búsqueda de los
   caracteres entre "<meta content="" y "" itemprop="price"\/>" en
   p_offers_tag:

38            Se añade a la list_style el price concatenado con "€"

39            Se añade a list_temp la list_style

40      Se añade a list_2D la list_temp

41      Se espera 0,6 segundos

42 Devuelve list_2D
```

Con esta tiendo no hubo ningún problema a la hora de lograr que el algoritmo funcionase.

Aunque no se encontró ningún caso de ambigüedad, para la determinación del nivel de censura se pensó qué decisión debería tomar el algoritmo en caso de mostrarse en las etiquetas datos sobre censura contradictorios.

6.2.3. Project-H

El pseudocódigo que describe el funcionamiento del algoritmo es el siguiente:

```
1  Se introduce la lista de enlaces logrado con getlinks_PH,
   llamada "getlinks"

2  Para cada enlace de getlinks:

3       Se conecta al enlace y se lee el código HTML como string

4       Se crea una lista vacía llamada list_temp

5       Se le añade a list_temp el nombre de la tienda y el enlace

6       Se busca el title entre "<h2 class="collection-title">" y
   "<\/h2>"

7       Se añade title a list_temp

8       Se define la variable autoria en la que se contienen las
   etiquetas a que están dentro de las etiquetas p con la clase
   (class) product-vendor

9       Se crea la lista vacía list_style

10      Se busca en autoria los caracteres entre "><" y se añade a
   list_style y éste a list_temp

11      Se añade "EMPTY" a list_temp

12      Se define una variable, li_tag, en la que se reúnen las
   etiquetas li

13      Se busca en li_tag si contiene "/collections/all/censored",
   definiendo esta búsqueda en la variable uncensored
```

```
14      Se busca en li_tag si contiene "/collections/all/uncensored", definiendo esta búsqueda en la variable partiallyuncensored

15          Si ambas variables definidas anteriormente están vacías, se añade "EMPTY" a list_temp

16      Si aparecen ambas variables, se añade "F" a list_temp

17      Si solamente la variable partiallyuncensored está vacía, se añade "T" a list_temp

18      Si solamente la variable uncensored está vacía, se añade "F" a list_temp

19      Se añade "EMPTY" a list_temp

20      Se añade list_temp a list_PH

21      Se espera 1,2 segundos

22 Devuelve list_PH
```

Project-H establece los precios mediante combinaciones de formatos y soportes que suponían demasiada complejidad. Por defecto se dejó los precios como EMPTY, igual que su ausente muestra de número de páginas.

Al igual que en el caso de 2D Market, se pensó qué decisión debería tomar el algoritmo en caso de mostrarse en las etiquetas datos sobre censura contradictorios.

6.3. Resumen

La extracción de datos se ejecuta mediante un script que a grandes rasgos queda así:

```
1  Se definen las funciones que reúnen los enlaces. Una función por cada tienda.

2  Se definen las funciones que acceden estos enlaces, leen y analizan los datos y guardan los datos en forma de lista.

3  Se definen los enlaces iniciales en variables.

4  Mediante variables que llaman a las funciones, se procede a la extracción de datos de la siguiente manera:

5       Por cada tienda, se reúnen los enlaces a partir de sus enlaces iniciales (Fakku es la única tienda con dos enlaces iniciales). A partir de estos enlaces reunidos en una lista, se extraen los datos, guardados en forma de lista de listas.

6  Cuando se ha terminado esta extracción de datos, se juntan en una variable y se guardan en un documento de texto.
```

7. Creación y población de una base de datos

Se diseñó la base de datos siguiendo el diagrama entidad-relación (ver Figura 6, en la página 22).

Se codificó el nombre de la tienda para ahorrar espacio, resultando en una tabla para las tiendas (la tabla *tienda*) y otra que relaciona la tabla *tienda* con la tabla *comic*, llamada *vendedor*.

Se creó la base de datos (“catalogo·) y sus tablas utilizando PhpMyAdmin, obteniéndose la base de datos organizada como indica la Figura 8.

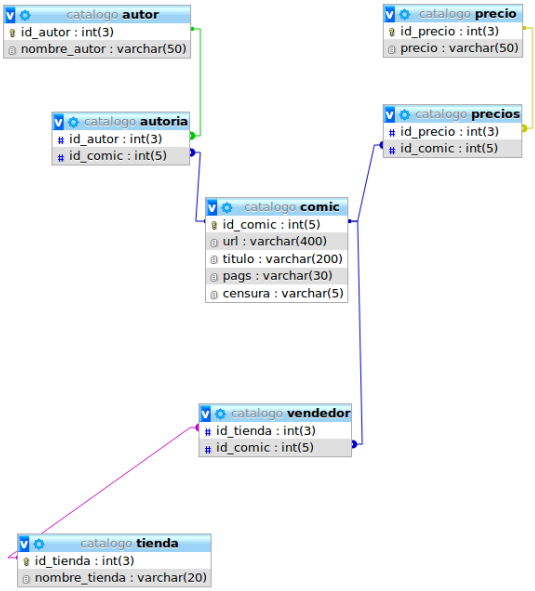

Figura 8. Esquema de relaciones generado por PhpMyAdmin.

Las tiendas se encuentran en la tabla “tienda” (identificadas por un número y descritas por un nombre), los productos en sí estarían en la tabla “comic” (identificados por un número, localizados con una URL y descritos mediante un título), la relación entre los productos y las tiendas se encuentra en la tabla “vendedor”, los precios se encuentran en la tabla “precio” (identificados por un número y descritos por el precio en sí), la relación entre los precios y los cómics a los que pertenecen se encuentra en la tabla “precios”, los autores se encuentran en la tabla “autor” (identificados por un número y descritos por el nombre en sí) y la relación de autoría se encuentra en la tabla “autoria”.

El algoritmo que se ejecutó es descrito en el siguiente pseudocódigo:

```
1  Se lee el documento de texo

2  Se interpreta el contenido del documento como una lista
```

```
3  Conecta con la base de datos
4  Se define una función cuya entrada es gran_lista:
5        Se crea una lista de tiendas vacía
6        Para producto en gran lista:
7              Se añade el primer elemento del producto a la lista
   de tiendas
8        Fin
9        Se asegura de que la lista de tiendas no contiene elementos
   repetidos
10       Se ejecuta mediante SQL la introducción de los datos de la
   lista de tiendas a la tabla tienda
11       Se crea una lista de autores vacía
12       Para producto en gran lista:
13             Se añade el cuarto elemento del producto a la lista
   de autores
14       Se asegura de que la lista de autores no contiene elementos
   repetidos
15       Se ejecuta mediante SQL la introducción de los datos de la
   lista de autores a la tabla autor
16       Se crea una lista de cuatro características (url, título,
   páginas y censura) vacía
17       Para producto en gran_lista:
18             Se añade a la lista de cuatro características el
   segundo, el tercer, el quinto y el sexto del producto (url,
   título, páginas y censura)
19             Se ejecuta mediante SQL la introducción de los datos
   de la lista de cuatro características a la tabla comic
20             Se vacía la lista de cuatro características
21       Para producto en gran_lista:
22             Se ejecuta una búsqueda por url en SQL para encontrar
   el id_comic.
23             Se guarda en una variable el resultado de la búsqueda
   SQL: id_comic
24             Se ejecuta una búsqueda por tienda en SQL para
   encontrar el valor id_tienda
25             Se guarda el resultado en la variable id_tienda
```

```
26            Se ejecuta mediante SQL la introducción de datos en
   la tabla vendedor con los datos guardados en las variables
   id_comic y id_tienda

27            Se encuentra el valor autor en producto teniendo en
   cuenta que puede ser una lista o un string

28            Se busca mediante SQL en la tabla autor, devolviendo
   el id_autor, el cual se guarda en la variable id_autor

29            Se ejecuta mediante SQL la introducción de datos en
   la tabla autoria con los datos guardados en las variables
   id_comic y id_autor

30      Se crea una lista de precios vacía

31      Para producto en gran_lista:

32            El séptimo elemento de producto es añadido a la lista
   de precios

33      Se asegura de que la lista de precios no contiene elementos
   repetidos

34      Se crea el diccionario vacío dict_precios

35      Se define la variable ii = -1

36      Para pprreeccioo en lista de precios:

37            Se suma 1 a ii

38            Se define para dict_precios la clave ii con el valor
   pprreeccioo

39      Se define l_keys como una lista formada por las claves de
   dict_precios

40      Se define l_values como una lista formada por los valores
   de dict_precios

41      Se insertan en la tabla precio los datos de cada clave y
   valor en los campos id_precio y precio, correspondientemente

42      Se crea una lista vacía llamada resultado_codificado

43      Cada producto en gran_lista:

44            Se crea la lista vacía producto_codificado

45            Se añade a producto_codificado el enlace (es decir el
   segundo elemento de producto)

46            Si el séptimo elemento de producto es una lista:

47                  Se crea la lista vacía lista_cod_precios

48                  Para precio en el séptimo elemento de producto:
```

```
49                          Si precio está en l_values:

50                              Se define la variable cod_precio, que es el elemento de l_keys en el que el número de índice es el número de índice en el que el valor de l_values coincide con precio

51                              Se añade a lista_cod_precios el cod_precio

52              Si no es así:

53                  Si el séptimo elemento de producto se encuentra en l_values:

54                          Se define la variable cod_precio, que es el elemento de l_keys en el que el número de índice es el número de índice en el que el valor de l_values coincide con el séptimo elemento de producto

55                          Se añade a producto_codificado el cod_precio

56              Se añade a resultado_codificado el producto_codificado

57          Cada producto en resultado_codificado:

58              Búsqueda en MySQL: Se busca id_comic en la tabla comic en la que el valor del campo url coincida con el primer elemento de producto

59              Se extrae el resultado de la búsqueda anterior, consiguiendo un número, definido en la variable id_comic

60              Si el segundo elemento de producto es una lista:

61                  Para precio en el segundo elemento de producto:

62                          Se introduce en la tabla precios los valores de las variables id_comic y precio en los campos correspondientes id_comic e id_precio

63              Si no:

64                  La variable precio se define como el segundo elemento de producto

65                  Se introduce en la tabla precios los valores de las variables id_comic y precio en los campos correspondientes id_comic e id_precio
```

Se conectó y comunicó el documento de texto que contiene el resultado de la extracción de datos con el sistema de gestión de basos de datos relacionales MariaDB mediante la biblioteca PyMySQL y para el manejo de los datos de la transferencia se utilizó las bibliotecas ast y re. Con estas bibliotecas de código de Python se creó el algoritmo que

partiendo del documento de texto en el que los algoritmos de scraping dejaban su resultado poblaba la base de datos.

El resultado de la población fueron un total de 2.633 filas, repartidas entre las tablas: autor (259), autoria (656), comic (535), precio (41), precios (604), tienda (3) y vendedor (535).

8. Diseño de la API REST

La Arquitectura Orientada a Servicios (concepto conocido por sus siglas en inglés SOA) es el estilo de arquitectura de un programa informático en el que sus procesos ofrecen servicios. Un servicio es una tarea concreta autónoma relacionada con una actividad que realiza el programa cuando se accede a este a través de una infraestructura en red (González Quiroga, 2011, p. 23). Permite que la interconexión de los servicios se realice sin tener que entender el funcionamiento interno (tecnologías implementadas) del servicio que se obtiene (Milena Caicedo, Bustos R., & Rojas Diaz, 2008, p. 178). En otros palabras, la arquitectura orientada a servicios implica la implementación de *middleware* (Velasco Melo & Caicedo Rendón, 2010, pp. 66-67).

Esta arquitectura se emplea para que a partir de un programa (A) se puedan crear otros programas (B). Lo que se dice que los programas B beben de A. Por ejemplo, de un programa que representa mapas cartográficos (A) se podrían utilizar los datos de representación geográfica que aporta en forma de servicio para unirlos a datos aportados por usuarios donde señalan incidentes ocurridos allí.

REST (REpresentational State Transfer) es un estilo de arquitectura aplicada a los servicios para la creación de sistemas hipermedia distribuidos (Navarro Marset, 2007, p. 4), como las aplicaciones web. En el caso de los servicios web que siguen los principios REST se comunican mediante el uso de URIs (identificador universal de recurso) que operan con el protocolo HTTP, permitiendo realizar las operaciones CRUD (Create, Read, Update and Delete) (Navarro Marset, 2007, p. 7; W3C Working Group, 2004).

REST no es considerada ni por su propio creador la solución a todos los problemas de integración de aplicaciones (Navarro Marset, 2007, p. 6). Otros estilos de arquitectura de servicios web serían: Remote Procedure Calls, SOAP y SOA sin REST (Navarro Marset, 2007, pp. 3, 10). Aun estas alternativas, los servicios web REST son adecuados para situaciones de escaso ancho de banda y cuando el servicio web no tiene estado[5] (Navarro Marset, 2007, p. 16).

Una API REST o API RESTful es una interfaz de programación de aplicaciones (es decir, unas especificaciones que indican cómo interactuar con una aplicación mediante otra aplicación) que sigue los principios del estilo REST.

Las API permiten la reutilización de los datos proporcionados mediante ellas. Tal como dice Navarro Marset (2007, p. 8), reutilizar formatos de representación de los recursos existentes facilita esta reutilización.

Los principios de las API REST son (Navarro Marset, 2007, pp. 5-6): seguir un protocolo que no recuerde interacciones anteriores, una sintaxis uniforme para describir

[5] Un servicio o recurso no tiene estado cuando las interacciones con éste no implican cambios sobre el recurso accedido.

los recursos (identificación de recursos mediante una URI unívoca) y posibilidad de realizar unas operaciones completamente definidas (como es por ejemplo GET).

Una API RESTful para servicios web que emplee JSON para el formato de datos implica una estandarización de los argumentos (las instrucciones mediante operaciones GET y otras), del formato de datos (al utilizar el formato estándar JSON) y del transporte o transmisión de la API (al utilizar el protocolo HTTP) (Gómez Fajardo & Lara Silva, 2014, p. 56).

La metodología de diseño de la API REST sigue estos pasos (Navarro Marset, 2007, pp. 13-14):

1. Identificar recursos que pueden ser utilizados como servicios.

2. Definir URLs para identificarlos y así poder ser utilizados.

3. Diferenciar las operaciones que poder hacer sobre estos recursos (GET, POST u otros).

4. Implementar el servidor Web.

Para definir las operaciones que se permite realizar en una API que utilice el protocolo HTTP, hay que tener en cuenta si estas son idempotentes y seguras. Se considera una operación segura la que no modifica el estado del recurso y se considera idempotente las que tienen un resultado al ser ejecutadas que no cambia aunque se ejecuten más de una vez. Las operaciones realizadas mediante el protocolo HTTP que son consideradas seguras e idempotentes son GET (devuelve un recurso, a modo de consulta), HEAD (devuelve un resumen de lo que se conseguiría con la operación GET) y OPTIONS (devuelve operaciones que el recurso puede responder).

Se consideran buenas prácticas en una API REST:[6]

- No incluir verbos en los literales de las URIs.
- Consistencia en el uso del número gramatical: o todo singular o todo plural.
- Codificar relaciones de recursos como subrecursos: la estructura sería "recurso/identificación_del_recurso/subrecurso/identificación_del_subrecurso".
- No utilizar GET para cambiar el estado de los recursos.
- Utilizar la cabecera HTTP, mediante el campo Accept, para informar del formato del contenido a recuperar.
- Permitir filtrar, ordenar y acceder por campo.
- Indicar la versión de la API en la URI.
- Permitir la sobrescritura de métodos HTTP.
- Permitir que el usuario elija el formato de la información recuperada.

[6] Estas recomendaciones fueron sacadas de unas diapositivas de un curso de administración electrónica facilitadas por mi tutor. Las diapositivas fueron escritas por mi tutor y Manoli Albert (malbert@dsic.upv.es). También se encontró un documento con recomendaciones similares (Navarro Marset, 2007, p. 11).

- Permitir operaciones de prueba que no afecten a datos reales.

También se recomienda que se paginen los resultados para no sobrecargar al usuario.

A partir del modelo de datos se diseñaron las opciones para la API RESTful, considerando cuáles son las posibilidades de consultas posibles.

El código Python de la aplicación realiza los siguientes pasos:

1. Conecta con la base de datos.
2. Realiza una consulta y guarda el resultado de la consulta en una variable.
3. Fila por fila guarda en una lista de listas la información asegurándose de que no se repita el mismo cómic en la lista de listas.
4. Lista a lista las convierte en diccionarios y los guarda en una lista. La conversión a diccionario se justifica porque la estructura de datos de Python del tipo diccionario coincide con la estructura de datos del formato JSON.
5. Finalmente, esta lista de diccionarios es convertida al formato JSON (ver Figura 9) y la devuelve.

```
    return (jsonify(resultat))

    for comic in lista_comics:
        fila_dict = {}
        fila_dict['autor']=comic[0]
        fila_dict['url']=comic[1]
        fila_dict['titulo']=comic[2]
        fila_dict['paginas']=comic[3]
        fila_dict['censura']=comic[4]
        fila_dict['precio']=comic[5]
        fila_dict['tienda']=comic[6]
        resultat.append(fila_dict)
    return (jsonify(resultat))
```

Figura 9. Conversión de lista a diccionario y introducción de los diccionarios en una lista para luego darles el formato JSON.

Se configuró que la orden "/api/v2/comic" enviara todos los datos, "/api/v2/comic/autor/<nombre_autor>" permitiera la búsqueda por responsable de la autoría y "/api/v2/comic/tienda/<nombre_tienda>" permitiera la búsqueda por tienda.

Se permitió buscar solamente por esos dos campos porque no se busca por título, sino por autor o círculo. La búsqueda por círculo permite comparar precios cuando el mismo cómic es vendido por distintas tiendas (es el caso de los cómics de 2D Market vendidos en Fakku a un precio que suele ser más caro).

La única operación HTTP posible era GET.

Aquí una captura de cómo se ve la API REST en su ejecución:

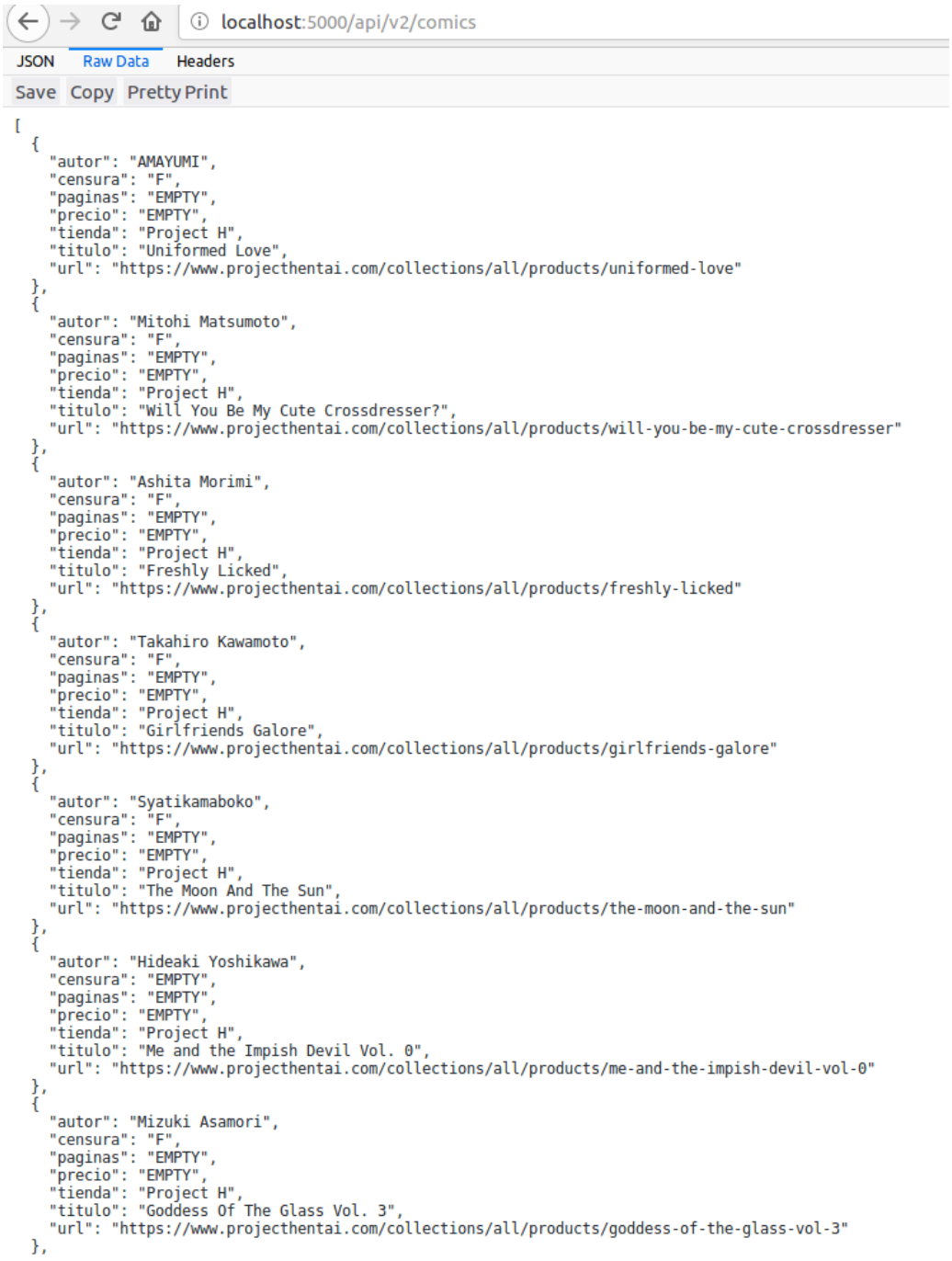

Figura 10. Captura del JSON resultante de la interacción con la API.

9. Desarrollo de un cliente

La aplicación cliente consiste principalmente en un documento HTML que es interactuado mediante el código Javascript mientras que la API REST está en ejecución. El funcionamiento sigue este esquema (ver Figura 11):

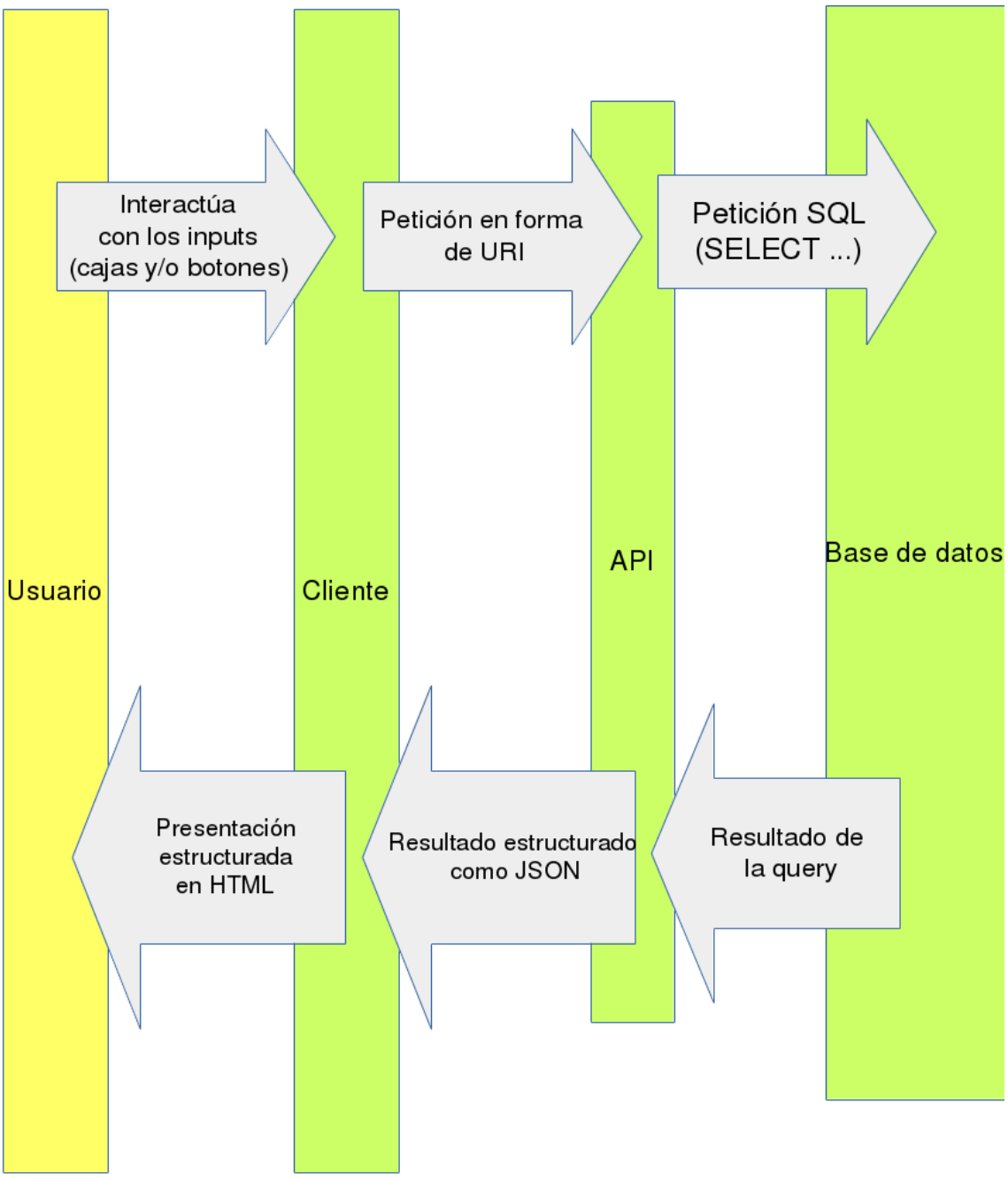

Figura 11. Esquema del funcionamiento del cliente.

El usuario interactúa con los elementos de entrada de datos del cliente, los cuales envían peticiones en forma de URI (tal como define la API) a la API, la cual las traduce en peticiones SQL del tipo “SELECT … FROM ...” realizadas en el gestor de base de datos (MySQL), que las devuelve en un formato que la API entiende y las presenta

dándoles el formato JSON, este fichero JSON es interpretado por el código Javascript y lo presenta al usuario en forma de contenido HTML generado e insertado en la página HTML visible al usuario.

Las búsquedas por tienda y por autor requieren de pulsar el botón manualmente, porque pulsar la tecla de intro no equivale a pulsar el botón correspondiente desde el punto de vista usable.

El código JavaScript que actúa como AJAX (interacción entre JavaScript y XML) dirige los datos introducidos para ser lanzados como peticiones a la API (ver Figura 12), el cual devuelve un fichero JSON (ver Figura 13) cuyos datos son utilizados para generar la alteración de la interfaz web (ver Figura 14).

```
var urlPorTienda="http://localhost:5000/api/v3/comic/tienda/";
var urlPorAutor="http://localhost:5000/api/v3/comic/autor/";
var urlTodos="http://localhost:5000/api/v3/comic";

$(document).ready(function(){
  $("#busq_tienda").click(function(){
    var tienda=$("#input_tienda").val();
    buscarComics(urlPorTienda+tienda);
  });

  $("#busq_autor").click(function(){
    var autor=$("#input_autor").val();
    console.log(autor);
    buscarComics(urlPorAutor+autor);
  });

  $("#todos").click(function(){
    buscarComics(urlTodos);
  });

  $("#resultados").empty();
});
```

Figura 12. AJAX: Introducción de datos y construcción de URIs para las peticiones.

```
function buscarComics(url){
  $.ajax({
    url:"proxy.php",
    data:{
      url: url
    },
    contentType: "application/json",
    dataType: "json",
    success: function(comics) {
      $("#resultados").empty();
      if(comics.length>0){
        $.each(comics,function( index, comic ){
          addComic(comic);
        });
      }else{
        sinResultados();
      }
    },
    error:function(e) {
      console.log(e);
      sinResultadosx();
    }
  })
}
```

Figura 13. AJAX: Devolución de un fichero JSON.

```
function addComic(comic){
  var html='';
  html+='<p class="output_titulo"><a href="'+comic.url+'">'+comic.titulo+'</a></p>';
    html+='<ul class="output_autor">';
    $.each(comic.autor, function(index, autor){
       html+='<li>'+autor+'</li>';
    });
    html+='</ul>';
    html+='<p class="output_paginas">'+comic.paginas+' páginas</p>';
    html+='<p class="output_censura">Sin censura?: '+comic.censura+'</p>';
    html+='<ul class="output_precio">';
    $.each(comic.precio, function(index, precio){
       html+='<li>'+precio+'</li>';
    });
    html+='</ul>';
    html+='<hr>';

    $("#resultados").append(html);
}
```

Figura 14. AJAX: Generación del código HTML a partir de los datos tomados del fichero JSON.

La interfaz del cliente se ve así:

Figura 15. Interfaz del cliente.

El cliente es interactuado con un botón que recupera la totalidad de los cómics y las barras de búsqueda que permiten buscar por tienda y por autor (o círculo).

Los resultados aparecen en la misma página como un continuo, sin páginas de navegación. Véase la correspondencia entre la API y el cliente en las siguientes imágenes:

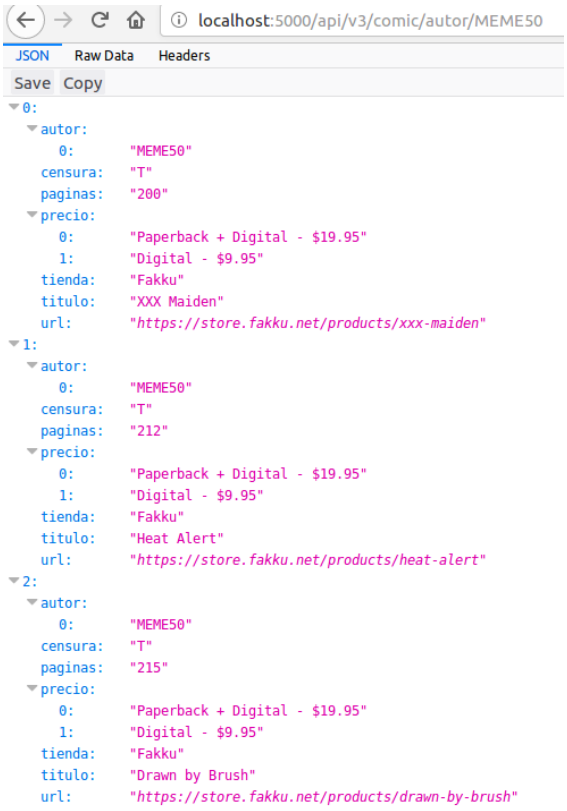

Figura 16. Búsqueda por autor (en este caso: MEME50) en la API REST.

Figura 17. Interfaz del cliente alterado por una interacción: una búsqueda por autor introduciendo MEME50 como criterio de búsqueda..

El código WYSIWYM alterado por la búsqueda por autor, siendo el autor buscado MEME50, es el siguiente:

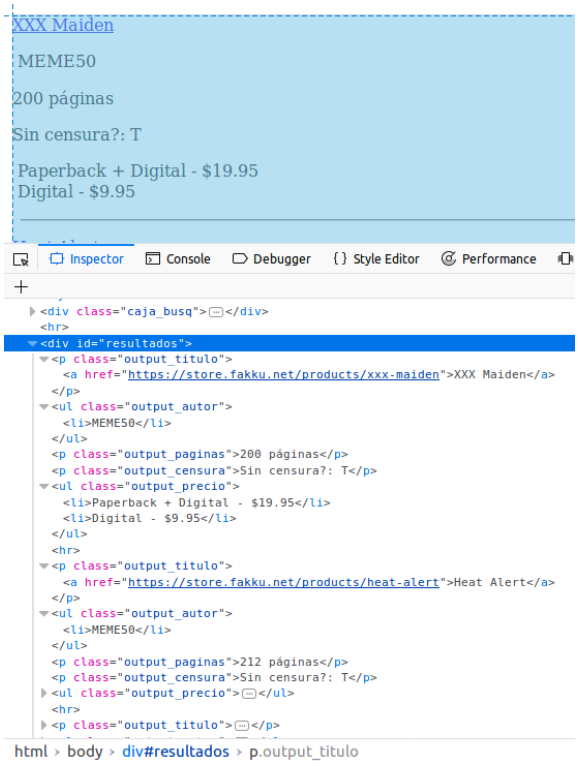

Figura 18. Código generado por la interacción.

10. Conclusiones

En este trabajo se ha presentado cómo a partir de una selección de fuentes de datos (unas tiendas elegidas por sus características), se recolectan los enlaces de las páginas a extraer, se extraen los datos interesantes, se guardan estructurados en listas en un fichero de texto, de este fichero se almacenan en una base de datos relacional, una API RESTful se comunica con esta base de datos y bebiendo de esta API se crea un servicio que tiene una interfaz web (hecha con código HTML y Javascript) con la que un usuario común puede utilizar.

Se cumplieron los objetivos de manera satisfactoria. Uno de los principales problemas que podrían haberse dado pero que no ocurrieron era que las webs tuvieran una protección contra el scraping y los ataques por denegación de servicio obligándome a utilizar estrategias de evasión como el falseamiento de header o la simulación de navegación (Mele, 2015).

Las tecnologías disponibles permitieron que alguien que no tenía una experiencia extensa programando siendo ayudado por un tutor experto en servicios REST pudiese crear un catálogo de manera automatizada. Otra posibilidad tecnológica podría haber sido utilizar MongoDB, un sistema de gestión de bases de datos NoSQL, en vez de MySQL, pero no contaba con suficiente confianza para su manejo.

Los pasos explicados detalladamente en este trabajo permiten reproducir proyectos similares a las personas que, como yo, no son expertas en programación.

Aun así hay que tener en cuenta que la última actualización de la web de Project-H implica que mi trabajo ha podido funcionar poco tiempo antes de ésta, ya que la actualización probablemente esconde código legible mediante la biblioteca de código urllib. En el caso de actualizar el código que he creado en este trabajo, tendría que utilizar la biblioteca Selenium.

Trabajos similares con una utilidad de cara al mundo académico podrían ser los vaciados (o extracción o *scraping*) de enlaces y DOIs de las webs de editoriales con prácticas depredadoras (Monte Luna, Cruz Agüero, Carmona, & Reyes Bonilla, 2014). Estos vaciados, si son validados como rigurosos, pueden servir para generar una API REST con la que editores (sean de Wikipedia o pertenecientes a consejos editoriales serios) o incluso investigadores, puedan comprobar si los trabajos que están refiriéndose provienen realmente de una fuente fiable por la que pasan revisiones de calidad mínimos (llamados revisiones por pares).

Para un futuro se podría estudiar la implementación en un servidor web, creando un servicio en línea. Para ello se tendría que tener conocimientos para cubrir los aspectos de seguridad y de automatización de todo el proceso (es decir, automatizar el ciclo que empieza con la extracción de datos hasta la creación de la base de datos). Estos conocimientos superan mis propios conocimientos.

Otros aspectos a considerar para un futuro serían:

- La revisión de la optimización del código de mis scripts, ya que durante la ejecución hay variables que se mantienen ineficientemente en la memoria.
- El posible control de títulos en varios idiomas considerando la extracción de estos de las propias tiendas (como en el caso de 2D Market) o de Baka-Updates Manga (https://www.mangaupdates.com/index.html).

11. Agradecimientos

Agradecimientos a Andriy Yatsyk por ayudarme con las expresiones regulares y con la conexión con la base de datos y a María Ángeles Espinosa Molina por ayudarme en la biblioteca PyMySql.

12. Referencias

2D Market - Twitter. (2012). Recuperado a partir de https://twitter.com/2DMarket

admin. (2014, agosto 24). Blog update. Recuperado 17 de septiembre de 2017, a partir de http://blog.2d-market.com/2014/08/blog-update/

Arriola Cruz, M. E. (2017). *Desarrollo de un Aplicativo Web que Permita el Análisis de los Registros de Auditoria del Sistema de Gestión de Base de datos MYSQL Utilizando Herramientas de Software Libre* (Tesis). Universidad de Guayaquil, Guayaquil, Ecuador. Recuperado a partir de http://repositorio.ug.edu.ec/handle/redug/19621

Barancovaitė-Skindaravičienė, K. (2013). Construction of gender images in Japanese pornographic anime. *Regioninės studijos*, *7*, 9-29.

Burdzik, T. (2014). Hentai–erotyka z mangi i anime. *Kultura I Edukacja*, *3*(103), 340–353. https://doi.org/10.6084/m9.figshare.1335808

Challenger-Pérez, I., Díaz-Ricardo, Y., & Becerra-García, R. A. (2014). El lenguaje de programación Python. *Ciencias Holguín*, *20*(2), 1-13.

Cruz Heras, D. de la, & Zumbado Rodríguez, C. D. (2003). *Flash, PHP y MySQL□: contenidos dinámicos*. Madrid, España: Anaya Multimedia.

Digital Manga Plans Project-H Imprint with 3 Hentai Books. (2011, junio 11). *Anime News Network*. Recuperado a partir de http://www.animenewsnetwork.com/news/2011-06-11/digital-manga-plans-project-h-imprint-with-3-hentai-books

Glez-Peña, D., Lourenço, A., López-Fernández, H., Reboiro-Jato, M., & Fdez-Riverola, F. (2013). Web scraping technologies in an API world. *Briefings in bioinformatics*, *15*(5), 788–797. https://doi.org/10.1093/bib/bbt026

Gómez Fajardo, H. J., & Lara Silva, E. A. (2014). Diseño de un modelo funcional de gestión soportado en servicios RESTful para gestión integrada de redes y servicios de T-learning. *Sistemas & Telemática*, *12*(29), 49-65.

González Quiroga, M. (2011). *Estudio de arquitecturas de redes orientadas a servicio* (Tesis de maestría). Universidad Politécnica de Cataluña. Recuperado a partir de http://hdl.handle.net/2099.1/12312

Hentai Website Fakku Enters Publishing Business with Japan's Wanimagazine. (2014, junio 22). *Anime News Network*. Recuperado a partir de http://www.animenewsnetwork.com/news/2014-06-22/hentai-website-fakku-enters-publishing-business-with-japan-wanimagazine/.75802

Judd, D. (2008, agosto 7). Scale Out with Hypertable. *Linux Magazine*. Recuperado a partir de http://www.linux-mag.com/id/6645/

López Montalbán, I., Castro Vázquez, M. de, & Ospino Rivas, J. (2014). *Bases de datos□: Técnico Superior en Desarrollo de Aplicaciones Multiplataforma y Web DAM y DAW* (2ª). Madrid, España: Garceta.

Mangirón, C. (2012). Manga, anime y videojuegos japoneses: análisis de los principales factores de su éxito global. *Puertas a la lectura*, *24*, 28-43.

Martínez Galán, S. (2017). *Pornografía animada japonesa. El éxito del hentai* (Trabajo final de grado). Universidad Politécnica de Valencia. Recuperado a partir de http://hdl.handle.net/10251/89941

Mele, M. (2015, febrero 21). Selenium Tutorial: Web Scraping with Selenium and Python. Recuperado 3 de mayo de 2018, a partir de http://www.marinamele.com/selenium-tutorial-web-scraping-with-selenium-and-python

Milena Caicedo, S., Bustos R., L. S., & Rojas Diaz, J. (2008). Integración de procesos utilizando la arquitectura orientada a servicios - SOA. *Scientia Et Technica*, *14*(40), 177-182.

Mirón Martínez, A. (2017). *¿Es el Hentai el reflejo de la sociedad japonesa?□: análisis e investigación del Manga erótico japonés* (Trabajo final de grado). Universitat Autònoma de Barcelona. Recuperado a partir de https://ddd.uab.cat/record/180993

Monte Luna, P. del, Cruz Agüero, G. de la, Carmona, R., & Reyes Bonilla, H. (2014). Impacto académico de las revistas ilícitas de libre acceso. *CIENCIA ergo-sum*, *21*(2), 140-142.

Navarro Marset, R. (2007, septiembre 17). REST vs Web Services. Recuperado a partir de http://users.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf

Patten, F. (1998). The Anime «Porn» Market. *Animation World Magazine*, *3*(4). Recuperado a partir de https://www.awn.com/mag/issue3.4/3.4pages/3.4patten.html

Pérez Castaño, A. (2014). *JavaScript Fácil* (1.ª ed.). Barcelona, España: Marcombo.

Perper, T., & Cornog, M. (2015). Anime and manga. En *The International Encyclopedia of Human Sexuality*. Oxford, Reino Unido: John Wiley & Sons.

Polidoro, F., Giannini, R., Conte, R. L., Mosca, S., & Rossetti, F. (2015). Web scraping techniques to collect data on consumer electronics and airfares for Italian HICP compilation. *Statistical Journal of the IAOS*, *31*(2), 165-176. https://doi.org/10.3233/sji-150901

Refsnes Data. (s. f.). HTML Javascript. En *HTML5 Tutorial*. W3Schools. Recuperado a partir de https://www.w3schools.com/html/html_scripts.asp

Segarra Cabedo, P. (2016). *Diseño e implementación de un sitio web que presente retos tipo CTF o Wargame de seguridad informática* (Trabajo final de maestría). Universitat Oberta de Catalalunya. Recuperado a partir de http://hdl.handle.net/10609/56464

Uidhir, C. M., & Pratt, H. (2013). Pornography at the Edge: Depiction, Fiction, & Sexual Predilection. En H. Maes & J. Levinson (Eds.), *Art & Pornography: Philosophical Essays* (pp. 1-18). Oxford, Reino Unido:

Oxford University Press. Recuperado a partir de https://philpapers.org/archive/MAGPAT.pdf

VanHemert, K. (2014, marzo 4). This Simple Data-Scraping Tool Could Change How Apps Are Made. *Wired*. Recuperado a partir de https://web.archive.org/web/20150511050542/http://www.wired.com/2014/03/kimono

Vargiu, E., & Urru, M. (2012). Exploiting web scraping in a collaborative filtering-based approach to web advertising. *Artificial Intelligence Research*, *2*(1), 44-54. https://doi.org/10.5430/air.v2n1p44

Velasco Melo, X., & Caicedo Rendón, Ó. M. (2010). MIDDIS: arquitectura de referencia para la interacción de servicios basados en SOA e IMS. *Ciencia e Ingeniería Neogranadina*, *20*(2), 65-85.

W3C Working Group. (2004, febrero 11). Web Services Architecture. W3C. Recuperado a partir de http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

Yegulalp, S. (2017, marzo 8). Hentai. ThoughtCo. Recuperado a partir de https://www.thoughtco.com/hentai-definition-145446