Recopilación automática de contenido web para agregación de datos y presentación clasificada

---

Automatic collection of web content for data aggregation and classified presentation

Fernando González Barrado - Grado en Ingeniería Informática
Daniel González Montero - Grado en Ingeniería de Software

FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo de Fin de Grado en Ingeniería Informática e Ingeniería de Software

Madrid, septiembre de 2021

Directores:

Bernabé García, Sergio
Muñoz Fernández, Emilio José

Agradecimientos

Gracias a nuestros dos tutores Sergio y Emilio por aceptarnos para desarrollar este Trabajo Fin De Grado, y por guiarnos durante todo el año académico.

Agradecer por todos los servicios prestados a la facultad de informática de la complutense y a los profesores por la paciencia que han tenido.

Sin embargo, quisiera agradecer especialmente a mi familia por haberme ayudado durante toda mi vida de estudiante.

Primero quiero agradecer a mi padre, Fernando González Gamella, porque esté donde esté sé que estará muy orgulloso de mí, y a mi madre, Josefa Barrado Díaz, porque sin ellos no tendría lo que he conseguido. Mi padre dejó el colegio a los 6 años y mi madre, por unas circunstancias difíciles, tuvo que abandonar antes de llegar al nivel de Secundaria Superior. Los dos han conseguido que sus tres hijos seamos estudiantes universitarios, cuando el 80 por ciento de los jóvenes con padres sin estudios superiores no consiguen títulos universitarios.

Gracias a mis dos hermanas, que durante mi etapa universitaria han sido un gran apoyo y, además, juntos hemos superado todas las dificultades que han aparecido a lo largo de estos años.

Quisiera agradecer a mi madre Maria Luz Montero Canales por su incesante atención, dedicación y aprecio. Sin su apoyo nada de esto hubiera sido posible. También dar las gracias a mi padre Alfonso González Fernández por su preocupación, ayuda constante y sus útiles consejos. Gracias a él escogí esta carrera.

Gracias a mi hermana Rocío González Montero por comprenderme y ayudarme en todo lo posible. Y agradecer a todos mis amigos por haber estado apoyando siempre y por hacerme desconectar de vez en cuando.

A todos ellos, gracias.

Índice general

Índice de figuras

Índice de tablas

Resumen

Cuando tratamos de buscar información en la red, recurrimos a buscadores web que nos ofrecen resultados aproximados sobre la información objetivo. Si estamos buscando información detallada dispersa en varias páginas web nos enfrentamos a un gasto de tiempo excesivo. Esto puede resultar para algunos usuarios inaceptable, por lo que concluirá en búsquedas parciales e incompletas del total de posibilidades ofrecidas en internet.

Con tanta cantidad de información accesible hoy en día, los buscadores web genéricos como Google, Bing, Safari, Mozilla Firefox e Internet Explorer no pueden dedicar recursos a contenidos excesivamente concretos. Aquí es donde entran en juego los buscadores especializados y comparadores web, cada vez más usados en la actualidad. Hablamos de casos como Trivago, un comparador de hoteles; Rastreator, un comparador de seguros; eDreams, un buscador de vuelos o Booking, un buscador de alojamientos, entre otros.

Para realizar este trabajo nos hemos fijado en la web de Zahoribo, un buscador de boletines oficiales del estado español. Esta página usa un motor de búsqueda basado en la indexación llamado Sphinx, el cual tiene muy buenas críticas por su gran velocidad. Y esta fama no es casual: en la web de Zahoribo con más de 6 millones de páginas indexadas, una búsqueda por palabras clave se resuelve en milisegundos.

Otra problemática a tener en cuenta entre la población más joven es que elegir la carrera supone una gran decisión. Solo en la Universidad Complutense de Madrid hay más de 60 grados a elegir. Esto, sumado a la incertidumbre y desconocimiento de un futuro estudiante universitario, hace que la tarea de encontrar el grado ideal sea una labor sumamente tediosa y compleja. Con el deseo de ayudar a los futuros estudiantes cuyos planes pasan por ir a la universidad, nos hemos planteado realizar una aplicación para facilitar esta decisión que, sin duda, puede cambiar completamente la vida de los estudiantes.

Para este Trabajo Fin de Grado, hemos fusionado estas dos ideas -la búsqueda de carreras y el motor de búsqueda Sphinx- para crear **Unisurfing**, un buscador de carreras universitarias y másteres. Con información universitaria extraída directamente de páginas

web oficiales, **Unisurfing** nos permite hacer una búsqueda por nombre de las asignaturas y por universidad. De esta manera podemos ver las carreras o másteres que poseen las asignaturas preferidas de los usuarios. Todo lo anterior explotable en una interfaz clara y sencilla que nos permita tener la información actualizada y clasificada.

Palabras clave

Buscador web, comparador web, motor de búsqueda, indexación, Zahoribo, gema, Sphinx, Ruby on Rails.

Abstract

When we try to find information on the internet, we turn to web search engines that give us approximate results on the objective information. If we are looking for detailed information scattered on several web pages we face an excessive waste of time. This may be unacceptable for some users, which will lead to partial and incomplete searches of the total possibilities offered on the internet.

With so much information accessible today, generic web search engines such as Google, Bing, Safari, Mozilla Firefox and Internet Explorer cannot dedicate resources to excessively specific content. This is where the specialized search engines and web comparators, increasingly used today, come into play. We talk about cases like Trivago, a hotel comparator; Rastreator, an insurance comparator; eDreams, a flight search engine or Booking, an accommodation search engine, among others.

To carry out this work we have looked at the Zahoribo website, a search engine for official bulletins of the Spanish state. This page uses an indexing-based search engine called Sphinx, which gets rave reviews for its great speed. And this fame is not accidental: on the Zahoribo website with more than 6 million indexed pages, a keyword search is resolved in milliseconds.

Another problem to take into account among the younger population is that choosing a career is an important decision. Only at the Complutense University of Madrid there are more than 60 degrees to choose from. This, added to the uncertainty and ignorance of a future university student, makes the task of finding the ideal degree an extremely tedious and complex task. With the desire to help future students whose plans include going to university, we have considered making an application to facilitate this decision that, without a doubt, can completely change the lives of students.

For this Degree Final Project, we have merged these two ideas -the career search and the Sphinx search engine- to create **Unisurfing**, a university degree and master's degree search engine. With university information taken directly from official web pages, **Unisurfing**

allows us to do a search by name of the subjects and by university. In this way we can see the careers or masters that have the preferred subjects of the users. All of the above exploitable in a clear and simple interface that allows us to have updated and classified information.

Keywords

Web search engines, web comparator , search engine, indexing, Zahoribo, gem, Sphinx, Ruby on Rails.

Capítulo 1

Introducción

1.1. Motivación

La creciente oferta de carreras universitarias y másteres hace que la elección de carrera sea cada vez más difícil y caótica. Por experiencia personal, sabemos que buscar una carrera universitaria, junto con la facultad donde es impartida, no es tarea fácil. Por ello, hemos desarrollado un sistema centralizado de información donde se pueda tomar decisiones de la forma más precisa, cómoda y visual posible. Este sistema ahorrará tiempo y disgustos a muchos estudiantes indecisos.

Los estudiantes elegimos universidad en función de diferentes filtros: por las distintas salidas laborales, por las asignaturas a cursar, por por el precio del grado, por la localización de la facultad o por reseñas de otros estudiantes entre otras. Modestamente intentaremos cubrir algunos de estos filtros de búsqueda y con ello facilitar esta importante toma de decisión en la medida de lo posible.

Motivados por el descubrimiento de un motor de búsqueda llamado Sphinx, hemos querido desarrollar un buscador de grados y másteres. Gracias a su potencial basado en la indexación, permite hacer una búsqueda instantánea de altas cantidades de información, tal y como hemos visto en un buscador de boletines oficiales del Estado, Zahoribo[1].

[1] https://zahoribo.com/

1.2. Objetivos

Nuestra principal meta consiste en desarrollar una aplicación web para centralizar la información de las carreras y másteres universitarios y con ello poder implementar distintos tipos de búsqueda. Con este fin, necesitaremos extraer la información directamente de las páginas web oficiales de las respectivas universidades. Además, se deberá añadir la posibilidad de tener la información actualizada para asegurarnos la corrección de la información.

En la primera versión funcional se ha tratado de reunir la información básica de cada grado junto con sus asignaturas, con la característica de hacer búsquedas por nombre de asignatura y por universidad. Otro de los objetivos será lograr abarcar el mayor número de carreras posibles ofrecidas en la Comunidad de Madrid, con la posibilidad de desarrollar la búsqueda de másteres universitarios.

Otro de los objetivos es la creación de una interfaz clara y sencilla con la que poder navegar por las diferentes opciones y poder hacer búsquedas con distintos filtros. Estas búsquedas deberán ser ejecutadas preferiblemente con el motor de búsqueda Sphinx, ya que se ha demostrado su excelente velocidad.

La consecución de los objetivos generales anteriormente mencionados se resuelven en la presente memoria, abordando una serie de objetivos específicos, los cuales se enumeran a continuación:

- Extracción efectiva del mayor número de grados posible.
- Tratamiento de datos para un almacenamiento estructurado.
- Configuración de una búsqueda eficiente usando el motor de búsqueda *Sphinx*.
- Desarrollo de una interfaz ordenada en la que disponer la información.
- Unión de la extracción y tratamiento de datos en la aplicación.
- Implementación de distintas búsquedas desde la interfaz web.

1.3. Plan de trabajo

Después de haber expuesto los objetivos principales de este trabajo, se determina la planificación de su desarrollo en el marco temporal establecido para su consecución. Para ello, se ha diseñado un diagrama de Gantt (ver Figura 1.1) y se ha definido de forma visual la duración y tareas a cumplir para el correcto funcionamiento.

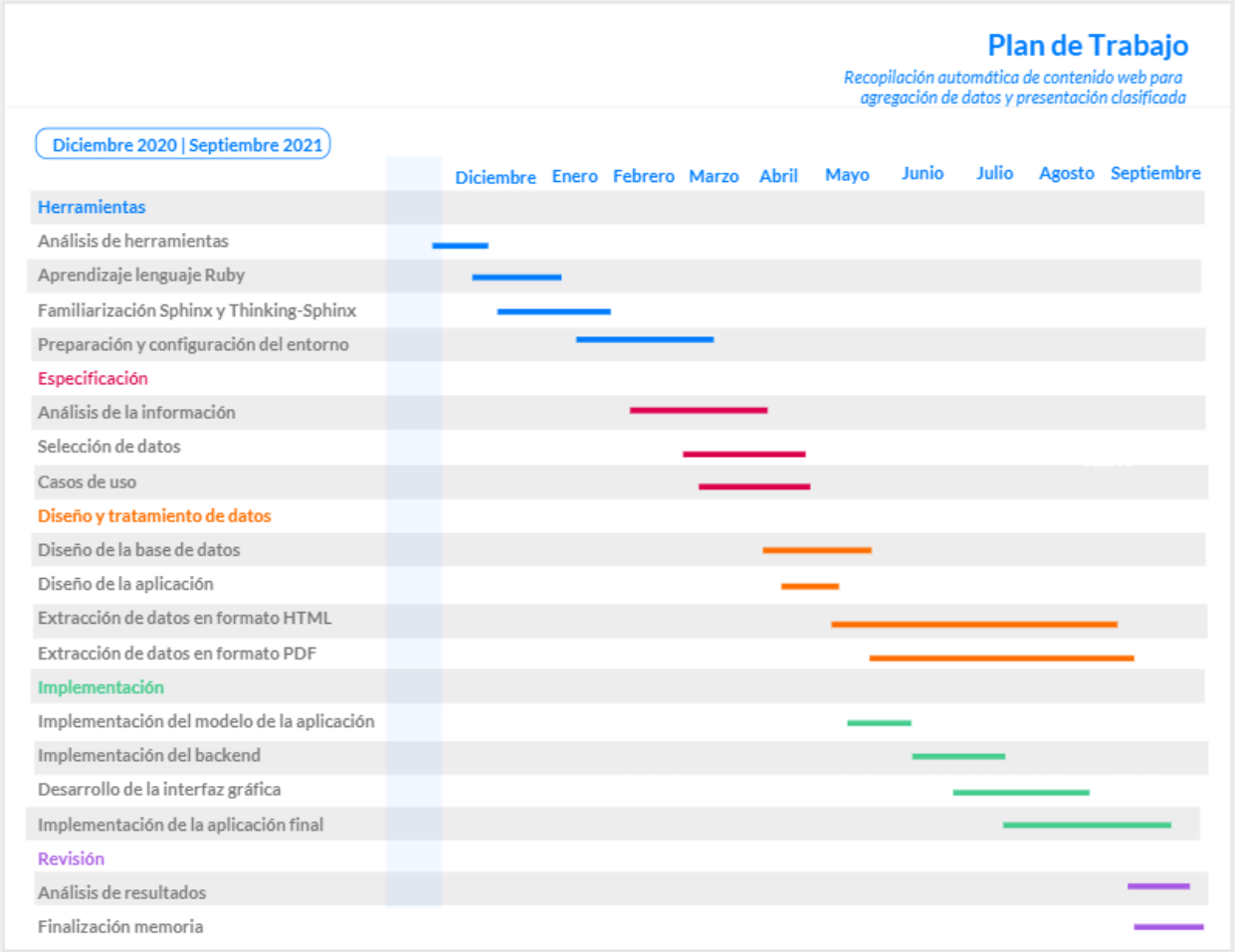


Figura 1.1: *Plan de trabajo.*

1.4. Organización de la memoria

Este informe detalla un resumen inicial, un resumen en inglés, seis capítulos, la bibliografía y cuatro apéndices. Esta sección describirá breve y ordenadamente el contenido de

cada capítulo:

- **Capítulo 1. Introducción**: Se describirán las razones que motivaron a la creación de esta iniciativa. Se mostrarán las metas iniciales previstas a cumplir. También se describirán las tareas en un marco temporal, mediante un diagrama de Gantt.

- **Capítulo 2. Estado del Arte**: Exposición de las diferentes herramientas implementadas y metodologías usadas durante el proceso de desarrollo, tanto de ámbito software como hardware. Se hará una comparativa de las tecnologías utilizadas con otras alternativas disponibles en el mercado para el desarrollo de la aplicación, junto con algunos ejemplos de uso reales.

- **Capítulo 3. Especificación de la Aplicación**: Planteamiento en detalle del proyecto, planificación y preparación previa. Listado de las universidades elegidas para extraer el contenido deseado. Presentación de la información elegida unido a las características de esta. También se analizarán los datos elegidos junto con su formato correspondiente.

- **Capítulo 4. Diseño para la Extracción y Tratamiento de los Datos**: Esquematización y presentación de los datos junto con diagramas aclaratorios. Se describirá detalladamente las metodologías y herramientas usadas para la recolección y transformación de información y su introducción en una base de datos. Al final se explicarán los algoritmos de extracción de datos.

- **Capítulo 5. Implementación de la Aplicación**: Diseño de la aplicación tanto *front-end* como en *back-end*. Explicación en profundidad del framework utilizado y del modelo elegido. Se expondrá a través de diferentes imágenes las vistas finales y casos de uso. También se definirán las posibles opciones de búsqueda junto con sus algoritmos y ajustes necesarios para la actualización de contenido. También se analizará el rendimiento de la aplicación, usando herramientas para desarrolladores.

- **Capítulo 6. Conclusiones y Trabajo Futuro**: Resumen del proyecto y sobre la utilidad de sus soluciones. Eficacia de la herramienta ante las diferentes universidades elegidas y su uso real. Futuras funcionalidades de la aplicación que por falta de tiempo no se pudieron implementar. Valoración personal global sobre el proyecto.

Capítulo 2

Estado del Arte

En este capítulo se comentarán algunas de las tecnologías y técnicas de extracción y tratamiento de datos y desarrollo web más populares disponibles del mercado. Estas herramientas se describirán brevemente y se compararán con las que se han elegido para esta aplicación.

2.1. Web scraping

Web scraping o raspado web en español, se trata de un conjunto de técnicas desarrolladas para la obtención de información a partir de sitios web. Con estas técnicas se pueden hacer multitud de funcionalidades, como por ejemplo: la transformación de información, la extracción selectiva, el análisis y minería de datos, reconocer estructuras de código HTML y la monitorización entre otros.

Estas técnicas son basadas en la indexación, como las que realizan los motores de búsqueda de Google. Hay dos tipos de motores de búsqueda: los manuales y los automáticos. Por un lado, el *scraping manual* trata de copiar y pegar información concreta, por lo que es más laborioso. Por otro lado, el *scraping automático* se ayuda de software o algoritmos para el análisis y extracción de contenido web.

Las utilidades del *web scraping* son casi ilimitadas, desde la caza de tendencias y optimización de precios hasta la monitorización de la competencia[1]. En este trabajo usamos estas

[1] https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/que-es-el-web-scraping/

técnicas con el fin de la extracción selectiva de datos y su presentación ordenada, ya que en la mayoría de universidades muestran su información a través del formato HTML. También remarcar que el *scraping* es legal siempre y cuando los datos recabados estén disponibles libremente para terceros en la web.

En este capítulo se mostrarán cuales son las técnicas más populares del mercado y nuestra elección para el proyecto. Hay varias técnicas para cada objetivo y veremos cual es la ideal para la implementación en nuestro proyecto.

2.1.1. Tecnologías revisadas

En la web hay diferentes herramientas para *scraping*, la mayoría son programas en los que se va configurando a partir de una página web concreta, con un proceso bastante automatizado. Estos software como *Bright Data*, *ScrapeStorm* y *Techvic* son programas que requieren de pago para la versión completa. Además, este proyecto requiere acoplar esta extracción de información a la aplicación y con estas herramientas sería demasiado complejo y costoso.

Otra opción interesante sería la de desarrollar esta parte con Python y sus diferentes librerías como *Scrapy*, *Selenium* y *BeautifulSoup*. Este lenguaje de programación resulta ideal para este tipo de funciones e interacciona bien con el ámbito de minería de datos y *data science*. A parte de sus múltiples librerías y su predisposición para el tratamiento de datos, Python cuenta con una gran cantidad de documentación y una comunidad que lo soporta. Esta opción se desestimó por la posible dificultad de acoplamiento con la aplicación desarrollada en Ruby.

Con la elección del lenguaje de programación Ruby, al ser un lenguaje con un porcentaje de uso pequeño entre la comunidad de desarrolladores, nuestra lista de técnicas se ha visto limitada. A continuación veremos algunas de estas técnicas, analizaremos las ventajas y desventajas, para facilitar la elección de la técnica más apropiada para nuestro proyecto. En este lenguaje a las librerías se les denomina gemas, existen multitud de ellas que aportan

funcionalidades a las diferentes aplicaciones desarrolladas. El uso de estas gemas proporciona un rápido y flexible desarrollo, ya que sólo hay que modificar o añadir unas pocas líneas de código para adaptar dichas gemas a tu programa.

Otra de las opciones sobre el papel fue *Scraping Bee*, una API de raspado web para Ruby. Esta redirige a una lista de APIs, en la que cada una extrae automáticamente una serie de parámetros. Estos parámetros abarcan desde palabras en inglés, multimedia y hasta geolocalización. Esta opción no fue elegida por exigir pagos para la versión completa y por estar destinada a un rascado web menos selectivo[2].

También barajamos *Anemon* un framework gratuito y multihilo en Ruby que ayuda a generar una *web spider*. Entre otras características puedes escribir tareas para generar algunas estadísticas interesantes en un sitio con solo darle la URL. Esta opción fue desechada por no poseer mucha documentación, estar poco actualizada y tener una comunidad poco activa.

2.1.2. Tecnología elegida: Kimurai

Finalmente hemos elegido Kimurai, un framework de raspado web en Ruby. Este marco de trabajo permite operar no sólo con las páginas estáticas HTML, sino también con páginas dinámicas y renderizados de Javascript. Esta funcionalidad puede resultar muy útil ya que algunos sitios web de universidades tienen páginas dinámicas y se deben poder procesar. Además de ser el marco de trabajo más popular, Kimurai tiene abundante documentación actualizada y una buena comunidad que lo respalda.

Kimurai usa la gema "Nokogiri" para páginas web estáticas HTML. "Nokogiri" se trata de una biblioteca de software de código abierto para analizar HTML, SAX y RSS en Ruby. Esta gema proporciona acceso a los elementos basados en selectores XPath, AT y CSS3, esenciales para contenido web. Las principales razones del uso de esta gema son: facilidad de uso y una gran comunidad usándola, ya que se trata de una gema con más de 400 millones

[2] https://www.scrapingbee.com/blog/web-scraping-ruby/

de descargas[3].

Este framework también usa otras gemas importantes como por ejemplo extiende a la gema "Capybara" [1] para utilizar determinados métodos como clickar, rellenar, seleccionar y retroceder entre otros, para interactuar con las páginas web. También utiliza la librería "Open Uri" para leer URLs http, https o ftp como si fuera un archivo y "Selenium" para navegar imitando la interacción del usuario, pudiendo usar el ratón y el teclado como lo haría un usuario humano.

2.2. Motores de búsqueda

Un motor de búsqueda o buscador es un mecanismo que recopila la información disponible en los servidores web y la distribuye a los usuarios por medio del proceso de *crawling*, en el que las arañas de los buscadores mapean los datos almacenados en la red. Para encontrar tales archivos, los buscadores web recurren a la identificación de la palabra clave empleada por la persona que realiza la búsqueda y devolver una lista de resultados[4].

El trabajo de estos motores de búsqueda se puede dividir en tres fases: análisis, catalogación y respuesta. La primera fase es realizada por programas llamados *Anemon spider*, robot o *crawler*. Estos programas escanean la web de forma metódica y automatizada. En la última fase las páginas seleccionadas por los rastreadores analizan y guardan el texto de estas páginas. Este paso requiere de la capacidad de los motores de búsqueda para ordenar los sitios con base en los datos por relevancia.

En este documento también haremos distinción entre buscadores genéricos y buscadores de contenido específico. Estos últimos hacen una búsqueda más precisa, actualizada y detallada del ámbito al que se refiere. Con este tipo de buscadores hay bastantes páginas web populares, como es el caso de Trivago, Rastreator, eDreams y Booking entre otros muchos. Estos buscadores muestran más información relevante, de una manera más clara, ordenada y fácilmente comparable que otros motores de búsqueda más genéricos no ofrecen.

[3] https://sloboda-studio.com/blog/how-to-do-web-scraping-with-ruby/
[4] https://rockcontent.com/es/blog/motores-de-busqueda/

2.2.1. Alternativas contempladas

- **Ransack**: Se trata de una reescritura de *MetaSearch*, un motor de búsqueda que permite la creación de formularios de búsqueda simples y avanzados para su aplicación en Ruby on Rails. En su modo simple funciona de manera muy similar a *MetaSearch* y requiere muy poco esfuerzo de configuración. En su modo avanzado, las búsquedas utilizan la funcionalidad de atributos anidados de Ruby on Rails para generar consultas complejas con agrupaciones *AND/OR* anidadas.

  Un inconveniente notable de estas búsquedas es que el tamaño aumentado de la cadena de parámetros, normalmente lo obligará a utilizar el método *HTTP POST* en lugar de *GET*. Esto significa que se necesitará modificar las rutas[5].

- **Elasticsearch**: Específico para Rails, es un motor de búsqueda de código abierto basado en JSON que nos permite buscar datos indexados de forma rápida y con opciones que no proporcionan los almacenes de datos clásicos. La información se guarda en una base de datos, se indexa en una instancia de Elasticsearch y luego se podrán buscar los datos indexados[6].

2.2.2. Alternativa seleccionada: Sphinx

Sphinx es un servidor de búsqueda de texto completo de código abierto, diseñado desde cero teniendo en cuenta el rendimiento, la relevancia (también conocida como calidad de búsqueda) y la simplicidad de integración. Está escrito en C++ y funciona en Linux (RedHat, Ubuntu, etc.), Windows, MacOS, Solaris, FreeBSD y algunos otros sistemas operativos.

Sphinx permite indexar por lotes y buscar información almacenada en una base de datos SQL, almacenamiento NoSQL o simplemente archivos de forma rápida y sencilla, o indexar y buscar datos sobre la marcha, trabajando con Sphinx prácticamente como con un servidor de base de datos.

[5] https://www.rubydoc.info/gems/ransack/1.7.0
[6] https://iridakos.com/programming/2017/12/03/elasticsearch-and-rails-tutorial

Se puede ajustar Sphinx para los requisitos particulares de su aplicación a través de un conjunto de funciones de procesamiento y una serie de funciones de relevancia que aseguran que también pueda ajustar la calidad de búsqueda. La búsqueda a través de *SphinxAPI* es tan simple como tres líneas de código y la consulta a través de *SphinxQL* es aún más simple, con consultas de búsqueda expresadas en un buen SQL antiguo.

Aunque su curva de aprendizaje es alta, hemos escogido la herramienta Sphinx por el rendimiento y escalabilidad:

- Rendimiento de indexación: Sphinx indexa hasta 10-15 MB de texto por segundo por núcleo de CPU único, es decir, más de 60 MB/seg por servidor (en una máquina de indexación dedicada).

- Búsqueda de rendimiento: La búsqueda a través de un millón de documentos, colección de texto de 1,2 GB que usamos para el desarrollo y las pruebas diarias se ejecuta a más de 500 consultas/seg en una máquina de escritorio de dos núcleos con 2 GB de RAM.

- Escalabilidad: El clúster Sphinx más grande que se conoce indexa más de 25.000 millones de documentos, lo que da como resultado más de 9 TB de datos.

Para usar la integración de Sphinx en Ruby hemos usado la gema "thinking-sphinx". Se trata de una biblioteca Ruby concisa y fácil de usar que conecta *Active Record* [2] con el proceso demonio de búsqueda Sphinx, administrando la configuración y la búsquedas (es requisito necesario tener instalado Sphinx en el sistema operativo). El desarrollador Pat Allan es quien está detrás de Thinking Sphinx. Pat Allan tiene multitud de preguntas respondidas a la comunidad de usuarios de Thinking Sphinx.

Para el uso de Thinking Sphinx, primero tienes que instalar en tu sistema operativo Sphinx. En nuestro caso al usar el sistema operativo Linux, lo hemos instalado con el comando "`sudo apt-get install sphinxsearch`". Además la gema "thinking-sphinx" solo funciona con la base de datos MySQL y Ruby on Rails por defecto se instala con SQlite3.

Para cambiar la instalación por defecto, primero hay que instalar MySQL de forma correcta para que funcione con Ruby on Rails (una de las cosas más tediosas del proyecto) [3]. Después hay que crear el proyecto con una opción (-d) que indica que la Base de Datos es MySQL. El comando quedaría de la siguiente forma: "`rails new nombreProyecto -d mysql`" (ver Figura 2.1).

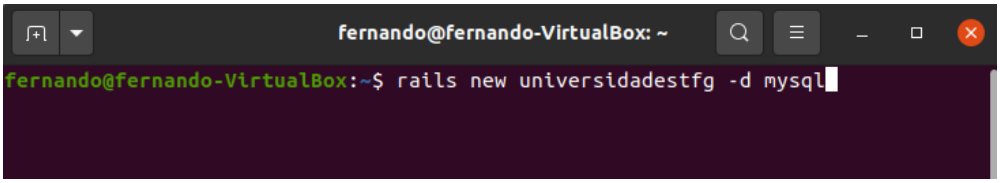


Figura 2.1: *Comando para crear proyecto Ruby on Rails con la base de datos MySQL.*

Una vez creado el proyecto en Ruby on Rails e instalada la gema "thinking-sphinx" en el archivo *Gemfile*, procedemos a su configuración para la indexación de contenido. Para poder instalar Sphinx, hay que añadir dos líneas de código al archivo *.gitignore* (ver Figura 2.2).

```
/db/sphinx
/config/*.sphinx.conf
```

Figura 2.2: *Modificación del archivo .gitignore para el funcionamiento de la gema "thinking-sphinx" en Ruby on Rails.*

A continuación para la configuración tenemos que usar el siguiente comando: "`rake ts:configure`". Después se generará en el proyecto dentro de la carpeta "*config*", el archivo "*development.sphinx.conf*".

Ahora podemos arrancar por primera vez Thinking Sphinx con el comando "`rake ts:start`" y pararlo con el comando "`rake ts:stop`". Al iniciarlo no funcionaría, ya que no hemos configurado los índices. La configuración de los índices y cómo hacer funcionar Sphinx, lo explicaremos en la Sección 5.3, como parte de la implementación del back-end.

2.3. Tecnologías para aplicaciones web

Después de analizar y estudiar las diferentes tecnologías para realizar el proyecto, vimos que lo más sencillo era usar un lenguaje de programación que tuviera un framework disponible para el entorno web.

2.3.1. Herramientas y lenguajes

Analizamos tres posibles lenguajes, con sus cuatro posibles frameworks asociados: PHP (framework Laravel), Java (framework Spring), Python (framework Django) y Ruby (framework Ruby on Rails). A continuación pondremos el análisis realizado de cada uno de ellos:

PHP (framework Laravel): PHP es un lenguaje conocido principalmente para el desarrollo de aplicaciones web. Después de consultar varias fuentes, su framework, tenía un peor rendimiento que Ruby on Rails y una curva de aprendizaje mayor que Django (Python).

Java (framework Spring): Java era el lenguaje de programación en el que más cómodos nos sentíamos, ya que es un lenguaje impartido en diferentes asignaturas del grado. La principal razón para no elegir el framework de trabajo Spring fue que no encontramos una buena integración con el motor de búsqueda Sphinx.

Python (framework Django): Python es un lenguaje de programación cada vez más demandado para el tratamiento de datos. Después de descartar tanto Java (framework Spring) como PHP (framework Laravel), estuvimos considerando usar Python y su framework Django. Finalmente no nos decidimos a usarlo porque había muchas funcionalidades que teníamos que hacer desde cero. Además la librería recomendada para hacer *web scraping*, llamada "Selenium", tenía una dificultad de aprendizaje mucho mayor que la librería "Nokogiri" de Ruby.

Ruby (framework Ruby on Rails): Tiene una curva de aprendizaje considerablemente elevada. Es más, el lenguaje de programación Ruby era el único lenguaje que ninguno habíamos visto durante todo el grado. Finalmente fue el lenguaje y framework elegido, ya

que una vez superada la elevada curva de aprendizaje, Ruby on Rails ofrece múltiples herramientas que facilitan y acortan el tiempo de desarrollo. En la siguiente sección, se explicará con más detalles.

2.3.2. Entorno escogido: Ruby on Rails

Cuando nos decantamos por Ruby y su framework Ruby on Rails, nos gustó mucho uno de los principios que sigue: *"Don't Repeat Yourself (DRY)"*. Básicamente quiere decir que, con sólo unas pocas líneas de código es posible crear aplicaciones web. Como ejemplo de esto, los desarrolladores de Twitter implementaron la primera versión después del primer día de desarrollo con Ruby on Rails[7]. Aunque este framework tiene una cuota de mercado relativamente pequeña, tiene un gran potencial, ya que aplicaciones web como GitHub o Airnbnb están desarrolladas en Ruby on Rails.

Además analizando las gemas disponibles[8], teníamos gemas disponibles para las dos principales características de nuestra aplicación: el motor de búsqueda Sphinx (gem "thinking-sphinx") y para *web scraping* (gem "kimurai"). Cuando creas un nuevo proyecto en Ruby on Rails, se genera un archivo automático con las gemas instaladas por defecto. Este archivo está en el directorio raíz del proyecto y se llama *Gemfile*. Además de estas gemas, hemos usado las siguientes para el desarrollo de la aplicación:

Gem "bootstrap" [4] y **Gem "jquery-rails"** [5]. Son necesarias para usar el framework de front-end *Bootstrap* en nuestra aplicación. En la Sección 3.4, se entrará en detalle con *Bootstrap*.

Gem "simpleform" [6]. Facilita el uso de formularios y cuadros de búsqueda dentro de la aplicación.

Gem "devise" [7]. Proporciona un servicio de autenticación flexible. Es una gema ampliamente usada en aplicaciones con Ruby on Rails con más de 100 millones de descargas. La razón principal es que tiene múltiples funcionalidades, pudiendo configurar las deseadas

[7] https://blog.desafiolatam.com/10-respuestas-a-ruby-on-rails/
[8] https://rubygems.org/?locale=es

en función del proyecto desarrollado. Algunas de estas funcionalidades son:

- Creación: Registra un *hash* y almacena una contraseña en la base de datos para validar la autenticidad de un usuario al iniciar sesión.
- Recuperación: Restablece la contraseña del usuario y envía instrucciones de restablecimiento.
- Confirmación: Envía correos electrónicos con instrucciones de confirmación y verifica si una cuenta ya está confirmada durante el inicio de sesión.

Para utilizar la gema "devise", además de agregarla en el *Gemfile*, necesitas instalarla en la aplicación con el comando: "`rails generate devise:install`". Al realizar la instalación puedes configurar qué funcionalidades usar. La configuración de las vistas de forma automática se hace mediante el comando "`rails generate devise:views`".

Gem "kimurai" [8]. Esta gema es necesaria para la aplicación ya que implementa la librería "Nokogiri", para realizar *web scraping*. Hablaremos en más detalle de ella en la Sección 4.3.1.

Gem "thinking-sphinx" [9]. Nos proporciona la librería para el motor de búsqueda Sphinx, hablaremos de ella en la Sección 3.3.

Gem "pundit" [10] y **Gem "rolify"** [11]. Nos proporcionan librerías para otorgar roles a los usuarios de nuestra aplicación. Por defecto nosotros tenemos dos roles para los usuarios registrados, rol "alumno" (se otorga al registrarse) y rol "admin" (usuarios con identificador 1 y 2). Con la gema "pundit" podemos permitir o prohibir determinadas acciones a cada rol de usuario.

Gem "pdf-reader" [12]. Es la gema que nos da la funcionalidad para poder extraer datos de los PDF. Hablaremos en más detalle en la Sección 4.3.2.

Para instalar estas gemas en nuestra aplicación, además de escribirlas en el fichero *Gemfile*, tenemos que usar el comando "`bundle install`" para instalar las gemas y sus depen-

dencias en el proyecto. Una vez instaladas, podemos proceder al uso de las funcionalidades de cada gema (ver Figura 2.3).

Gemfile

```
source 'https://rubygems.org'
git_source(:github) { |repo| "https://github.com/#{repo}.git" }

ruby '3.0.0'

# Bundle edge Rails instead: gem 'rails', github: 'rails/rails', branch: 'main'
gem 'rails', '~> 6.1.4'
# Use mysql as the database for Active Record
gem 'mysql2', '~> 0.5'
# Use Puma as the app server
gem 'puma', '~> 5.0'
# Use SCSS for stylesheets
gem 'sass-rails', '>= 6'
# Transpile app-like JavaScript. Read more: https://github.com/rails/webpacker
gem 'webpacker', '~> 5.0'
# Turbolinks makes navigating your web application faster. Read more: https://github.com/turbolinks/turbolinks
gem 'turbolinks', '~> 5'
# Build JSON APIs with ease. Read more: https://github.com/rails/jbuilder
gem 'jbuilder', '~> 2.7'
# Use Redis adapter to run Action Cable in production
# gem 'redis', '~> 4.0'
# Use Active Model has_secure_password
# gem 'bcrypt', '~> 3.1.7'
gem 'bootstrap', '~> 5.1.0'
gem 'jquery-rails'
gem 'simple_form', '~> 5.1'
gem 'devise', '~> 4.2'
gem 'kimurai', '~> 1.2'
gem 'thinking-sphinx', '~> 5.3'
gem "pundit"
gem "rolify"
gem 'pdf-reader', '~> 2.5'
```

Figura 2.3: *Archivo Gemfile, con las gemas adicionales instaladas.*

Capítulo 3

Especificación de la Aplicación

En este capítulo se describirá la funcionalidad de la aplicación llamada *Unisurfing* de forma detallada. Hablamos en primer lugar de selección de datos a utilizar y su formato particular para llegar a copilar toda la información en la aplicación. En segundo lugar, no podemos olvidarnos de mencionar la herramienta utilizada para la búsqueda, Thinking Sphinx. En tercer lugar, se mencionará el proceso para la preparación del entorno junto con las herramientas elegidas. Para finalizar se expondrán los casos de uso de forma resumida.

3.1. Selección de datos entrada y salidas

Una vez elegidas las tecnologías, necesitamos saber cuál va a ser el flujo de información. Para ello, analizaremos el ámbito alrededor de los datos disponibles y la información objetivo a la que queremos llegar y mostrársela al usuario final. Por lo tanto, lo más práctico es acceder a las páginas oficiales de las universidades y hacer una búsqueda exhaustiva de información. Entonces, se han buscado manualmente los datos más comunes a todos los grados analizando los datos de distintas páginas web, siempre que fuesen de interés general.

3.1.1. Páginas web de universidades elegidas

El ámbito de la universidades está restringido únicamente a la Comunidad de Madrid por elección propia, debido a ser nuestro lugar de residencia y por consiguiente poseemos más conocimientos sobre su relevancia. También debido a que es, junto con Barcelona, la

zona más amplias y diversa a nivel universitario.

El criterio seleccionado para la elección de universidades es tanto la reputación como la amplia gama de grados y másteres a elegir. El otro criterio igualmente relevante es la facilidad de extracción de información, que engloba tanto el nivel de estructura de la información como la facilidad la sencillez en el proceso de recolección y transformación. Según estos filtros nos queda la siguiente elección de universidades:

- **Universidad Complutense de Madrid** : La información proporcionada viene en documentos PDF internos (ver Figura 3.1), en los que se incluye el nombre del grado, los créditos por tipo de asignatura que hay que cursar y las asignaturas por curso con sus créditos correspondientes. También es común encontrar algunos grados con información en PDF y semi-estructurada en HTML como es el caso del grado de Farmacia (ver Figura 3.2). Fue elegida gracias al gran abanico de grados disponibles aunque la información, si bien visualmente es fácil de entender, no lo es a la hora de leer y recolectar estos datos[1].

- **Universidad Politécnica de Madrid**: La información proporcionada viene en documentos PDF internos, en los que se incluye el nombre del grado y están ordenados por curso las asignaturas con su nombre, créditos, tipo y semestre correspondientes (ver Figura 3.3). El criterio de elección de esta universidad fue la gran oferta de carreras y su relevancia, ya que está enfocada principalmente a las ingenierías[2].

- **Universidad Carlos III de Madrid**: Toda la información de esta universidad viene la información estructurada en la propia página web (ver Figura 3.4). La información que incluye el nombre del grado y ordenada por cursos y cuatrimestres el nombre de las asignaturas junto con sus créditos, tipos y el lenguaje en las que se imparten. El motivo de su elección fue su amplio abanico de grados y la facilidad para su extracción[3].

[1] https://www.ucm.es/estudios/grado
[2] https://www.upm.es/Estudiantes/EstudiosTitulaciones/EstudiosOficialesGrado
[3] https://www.uc3m.es/grado/estudios

Plan de Estudios

Tipo de Asignatura	ECTS
Formación Básica	60
Obligatorias	111
Optativas	60*
Trabajo Fin de Grado	9
Total	240

* Incluye 6 ECTS de Prácticas Externas

Primer Curso	ECTS
Bioestadística	6
Biología Celular e Histología	12
Biomatemáticas	6
Bioquímica	12
Física Aplicada a la Biología	6
Geología Aplicada a la Biología	6
Métodos en Biología	6
Química Aplicada a la Biología	6

Segundo Curso	ECTS
Botánica	12
Genética	12
Microbiología	12
Zoología	12
Dos Optativas	12

Tercer Curso	ECTS
Biología Evolutiva	12
Ecología	12
Fisiología Animal	12
Fisiología Vegetal	12
Dos Optativas	12

Cuarto Curso	ECTS
Biología Experimental	9
Proyectos y Estudios en Biología	6
Cinco Optativas de Mención	30
Una Optativa de Perfil Profesional	6
Trabajo Fin de Grado	9

Optativas de 2º Curso	ECTS
Biogeografía	6
Fundamentos de Ingeniería Genética y Genómica	6
Organografía Microscópica	6
Regulación del Metabolismo	6

Optativas de 3er Curso	ECTS
Antropología Física	6
Biología del Desarrollo	6
Ecología de los Recursos Naturales	6
Etología	6

Optativas de 4º Curso	ECTS
Perfil Profesional	
Historia, Enseñanza y Difusión de la Biología	6
Iniciación a la Investigación	6
Neurobiología	6
Prácticas Externas	6
Mención: Biología Ambiental	
Análisis de la Biodiversidad Animal	6
Análisis de la Biodiversidad Vegetal	6
Biología de la Conservación	6
Biología de la Contaminación	6
Descripción y Valoración Ambiental de Ecosistemas	6
Gestión Sostenible del Medio Natural	6
Mención: Biotecnología	
Análisis Biológico del Control de Calidad	6
Biología Aplicada a la Producción Animal y Vegetal	6
Biorremediación y Control Ambiental	6
Biotecnología de Enzimas	6
Biotecnología Microbiana	6
Cultivos Celulares y Transgénesis	6
Mención: Biología Sanitaria	
Bioquímica Clínica y Patología Molecular	6
Fisiopatología y Farmacología	6
Genética Humana	6
Inmunología y Análisis Clínicos	6
Microbiología Clínica y Epidemiología	6
Parasitología y Vectores de Transmisión	6

Créditos de Participación	ECTS
Cualquier curso	6

Figura 3.1: *Grado en Biología, Universidad Complutense de Madrid.*

	CURSO	1º		2º		3º		4º		5º		TOTAL
	SEMESTRE	1S	2S	3S	4S	5S	6S	7S	8S	9S	10S	
	MATERIA											
MÓDULO 1.- Química	Química	6	6									60
	Química Analítica		6	6								
	Química Orgánica		6	6								
	Físico-Química Farmacéutica			4,5	4,5							
	Química Farmacéutica					4,5	4,5	6				
MÓDULO 2.- Física y Matemáticas	Estadística	6										12
	Física	6										
MÓDULO 3.- Biología	Biología	6										54
	Bioquímica			4,5	10,5							
	Botánica		6									
	Parasitología			4,5	4,5							
	Farmacognosia y Fitoterapia							4,5	4,5			
	Microbiología					4,5	4,5					
MÓDULO 4.- Farmacia y Tecnología	Biofarmacia y Farmacocinética							6				27
	Tecnología Farmacéutica					4,5	4,5		6	6		
MÓDULO 5.- Medicina y Farmacología	Fisiología		6	4,5	4,5	6						81
	Inmunología				6							
	Bromatología					6						
	Nutrición						6					
	Farmacología					4,5	4,5	4,5	4,5			
	Toxicología									6		
	Atención Farmacéutica									6		
	Bioquímica Aplicada y Clínica						6					

Figura 3.2: *Grado en Farmacia, Universidad Complutense de Madrid.*

GRADO EN CIENCIAS AGRARIAS Y BIOECONOMÍA
(Código 20BI)

PRIMER CURSO

CÓDIGO	ASIGNATURAS	CRÉDITOS	TIPO	SEMESTRE
205000001	MATEMÁTICAS I	6	Bás	1º
205000003	FÍSICA	6	Bás	1º
205000004	QUÍMICA	6	Bás	1º
205000007	BIOLOGÍA VEGETAL Y ANIMAL	6	Bás	1º
205000011	AGRICULTURA Y ALIMENTACIÓN	3	Bás	1º
205000020	CLIMATOLOGÍA	3	Obl	1º
205000002	MATEMÁTICAS II	6	Bás	2º
205000005	QUÍMICA AGRÍCOLA	6	Bás	2º
205000006	BIOQUÍMICA	6	Bás	2º
205000012	GEOLOGÍA	5	Bás	2º
205000016	BOTÁNICA AGRÍCOLA	4	Obl	2º
205000017	PROGRAMACIÓN PARA ESTADÍSTICA	3	Obl	2º

Figura 3.3: *Grado en Ciencias Agrarias y Bioeconomía, Universidad Politécnica de Madrid.*

Curso 1 - Cuatrimestre 1

Asignaturas	ECTS	TIPO	Idioma
Técnicas de búsqueda y uso de la información	3	FB	
Técnicas de expresión oral y escrita	3	FB	
Filosofía Política	6	FB	
Habilidades: Humanidades	6	FB	
Historia de España	6	FB	
Teoría de la comunicación mediática	6	O	

Curso 1 - Cuatrimestre 2

Asignaturas	ECTS	TIPO	Idioma
Comunicación y participación ciudadana en la red	6	FB	
Estadística aplicada al periodismo	6	FB	
Estructura del sistema de medios	6	FB	
La noticia periodística	6	O	
Lengua española aplicada a los medios	6	FB	

Curso 2 - Cuatrimestre 1

Asignaturas	ECTS	TIPO	Idioma
Derecho de la Información	6	O	
Economía	6	FB	
Periodismo y cambio social en España	6	O	
Radio Informativa	6	O	
Teoría y análisis del documental audiovisual	6	O	

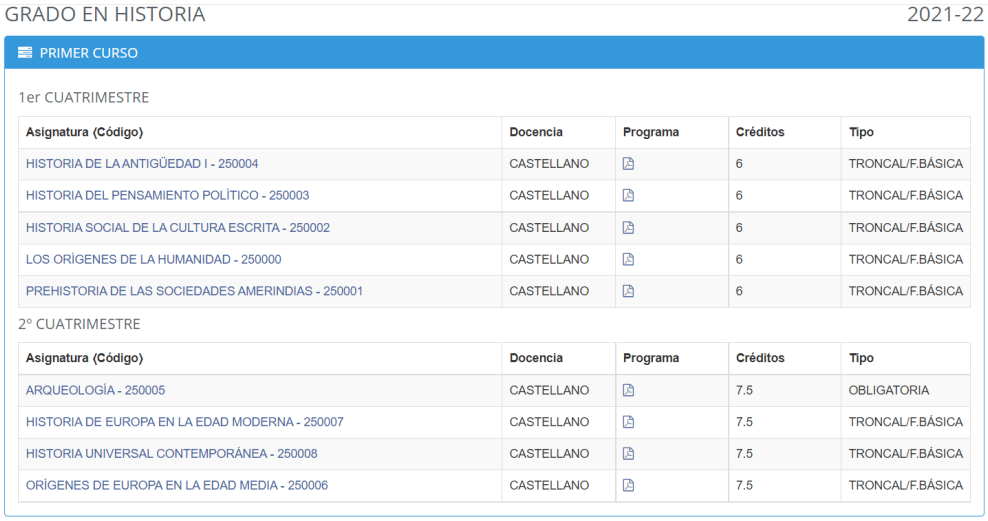
Curso 2 - Cuatrimestre 2

Asignaturas	ECTS	TIPO	Idioma
Habilidades profesionales interpersonales	3	FB	
Hojas de cálculo. Nivel Intermedio	3	FB	
Diseño en medios periodísticos	6	O	
Metodologías de investigación en periodismo	6	O	
Periodismo en la Red	6	O	
Televisión Informativa	6	O	

Figura 3.4: *Grado en Periodismo, Universidad Carlos III.*

- **Universidad Alcalá de Henares**: La información de esta universidad viene estructurada en la propia página web y también en documentos PDF semi-estructurados. Entre algunos datos importantes incluye el nombre del grado y ordenada por cursos y cuatrimestres el nombre de las asignaturas junto con sus créditos, tipos y el lenguaje en las que se imparten (ver Figura 3.5). El motivo de su elección fue la facilidad para

su extracción de sus datos[4].



GRADO EN HISTORIA 2021-22

PRIMER CURSO

1er CUATRIMESTRE

Asignatura (Código)	Docencia	Programa	Créditos	Tipo
HISTORIA DE LA ANTIGÜEDAD I - 250004	CASTELLANO		6	TRONCAL/F.BÁSICA
HISTORIA DEL PENSAMIENTO POLÍTICO - 250003	CASTELLANO		6	TRONCAL/F.BÁSICA
HISTORIA SOCIAL DE LA CULTURA ESCRITA - 250002	CASTELLANO		6	TRONCAL/F.BÁSICA
LOS ORÍGENES DE LA HUMANIDAD - 250000	CASTELLANO		6	TRONCAL/F.BÁSICA
PREHISTORIA DE LAS SOCIEDADES AMERINDIAS - 250001	CASTELLANO		6	TRONCAL/F.BÁSICA

2º CUATRIMESTRE

Asignatura (Código)	Docencia	Programa	Créditos	Tipo
ARQUEOLOGÍA - 250005	CASTELLANO		7.5	OBLIGATORIA
HISTORIA DE EUROPA EN LA EDAD MODERNA - 250007	CASTELLANO		7.5	TRONCAL/F.BÁSICA
HISTORIA UNIVERSAL CONTEMPORÁNEA - 250008	CASTELLANO		7.5	TRONCAL/F.BÁSICA
ORÍGENES DE EUROPA EN LA EDAD MEDIA - 250006	CASTELLANO		7.5	TRONCAL/F.BÁSICA

Figura 3.5: *Grado en Historia, Universidad de Alcalá de Henares.*

- **Universidad Alfonso X "el sabio"**: La información de esta universidad viene estructurada en la propia página web. Entre algunos datos importantes incluye el nombre del grado y ordenada por cursos y cuatrimestres el nombre de las asignaturas junto con sus tipos, sus créditos y el lenguaje en las que se imparten (ver Figura 3.5). El motivo de su elección fue la facilidad para su extracción de sus datos[5].

Primer Curso

PRIMER CUATRIMESTRE

Código	Asignaturas	Carácter*	Créditos
0131000	Bioestadística	FB	6
0131001	Biología	FB	6
0131002	Estructura Y Función del Cuerpo Humano	FB	6
0131003	Idioma Moderno	FB	6
0131004	Química	FB	6
TOTAL:			**30**

Figura 3.6: *Grado en Nutrición Humana y Dietética, Universidad Alfonso X "el Sabio".*

[4] https://www.uah.es/es/estudios/estudios-oficiales/grados/

[5] https://www.uax.com/grados

- **Universidad Europea de Madrid**: En esta universidad la información viene estructurada en la propia página web. Entre algunos datos importantes incluye el nombre del grado y, además, está ordenada por cursos y cuatrimestres con el nombre de las asignaturas junto con sus créditos, tipos e idioma en el que se imparte (ver Figura 3.7). El motivo de su elección fue la facilidad para su extracción de sus datos[6].

Programa de estudios

Código Asignatura / Subject Code	Materia / Coursework	ECTS	Tipo / Type	Idioma / Language
9954002101	Cálculo I	6	BASICA	Español (es)
9954002102	Cálculo II	6	BASICA	Español (es)
9954002103	Fundamentos Físicos de la Ingeniería	6	BASICA	Español (es)
9954002104	Sistemas de Representación y Dao	9	BASICA	Español (es)
9954002105	Geología	9	BASICA	Español (es)
9954002106	Álgebra y Estadística	6	BASICA	Español (es)
9954002107	Mecánica de Estructuras	6	OBLIGATORIA	Español (es)
9954002108	Química de Materiales	6	OBLIGATORIA	Español (es)
9954002109	Taller de Proyectos: Modelado 3d en Bim	6	BASICA	Español (es)

PRIMER CURSO / FIRST YEAR

Figura 3.7: *Grado en ingeniería civil, Universidad Europea.*

- **Universidad Camilo José Cela**: La información viene estructurada en la propia página web de la universidad. Entre algunos datos importantes a destacar están que incluye el nombre del grado, que viene ordenada por cursos, que el nombre de las asignaturas se detalla junto con sus créditos, tipos y semestre (ver Figura 3.8). El motivo de su elección fue la facilidad para su extracción de sus datos[7].

[6] https://universidadeuropea.com/grados-madrid/
[7] https://www.ucjc.edu/estudios-universitarios/grados/oficiales/

1º CURSO | 2º CURSO | 3º CURSO

ASIGNATURA	CRÉDITOS	TIPO	SEMESTRE
Fundamentos de la fotografía y el dibujo	6	Básica	1º
Lenguaje y expresión audiovisual	6	Básica	1º
Fundamentos de la creatividad	6	Básica	1º
Fundamentals of multimedia programming	6	Básica	1º
Derecho de la comunicación publicitaria	6	Obligatoria	1º
Usability, interface and user experience	6	Básica	2º

Figura 3.8: *Grado en ciencias de la actividad física y del deporte, Universidad Camilo José Cela.*

- **Universidad de Nebrija**: Esta universidad presenta la información tanto estructurada dentro de la propia página web como en PDF adjuntos. Estos PDF no son adecuados para nuestra aplicación por estar dividido en dos partes. Sin embargo, en la página web también están los datos estructurados, más adecuados para su extracción (ver Figura 3.9).

  Entre algunos datos importantes se incluye el nombre del grado, el nombre de las asignaturas junto con sus créditos semestre, tipos y la materia. El motivo de su elección fue la facilidad para su extracción de sus datos[8].

[8] https://www.nebrija.com/carreras-universitarias/carreras.php

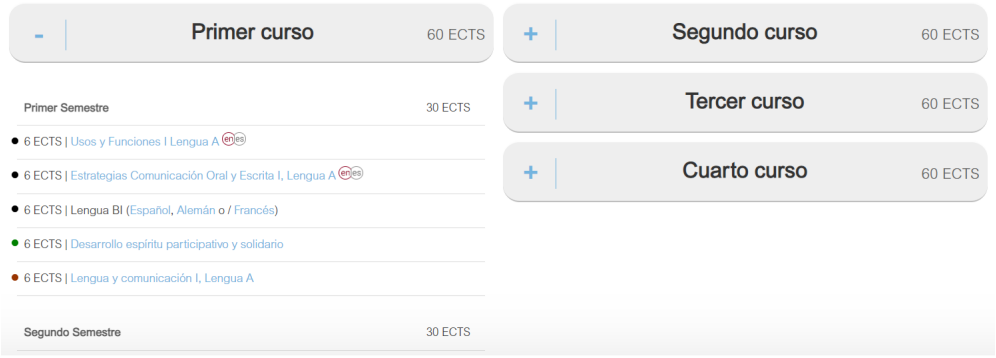


Figura 3.9: *Grado en Lenguas Modernas, Universidad de Nebrija.*

- **Universidad Rey Juan Carlos**: Esta universidad presenta la información estructurada dentro de la propia página web. En la página web están los datos estructurados en HTML, adecuados para su extracción (ver Figura 3.10).

  Entre algunos datos importantes se incluye el nombre del grado, el nombre de las asignaturas junto con sus créditos, el semestre y tipos de asignatura. El motivo de su elección fue la facilidad para su extracción de sus datos y su gran cantidad de grados ofrecidos[9].

CURSO 1º

Semestre	Asignatura	Carácter	Créditos
1	Álgebra Lineal	FBR	6
1	Fundamentos Biológicos	FBR	6
1	Lógica	OB	6
1	Introducción a la Programación	FBC	6
1	Matemática Discreta	OB	6
2	Estructuras Algebraicas	OB	6
2	Cálculo	FBR	6
2	Fundamentos Físicos	FBR	6
2	Ética, Legislación y Protección de Datos	FBC	6
2	Probabilidad	OB	6
Total de créditos a cursar: 60			

Figura 3.10: *Grado en Matemática, Universidad Rey Juan Carlos.*

- **Universidad CEU San Pablo**: Esta universidad presenta la información tanto estructurada dentro de la propia página web como en PDF adjuntos. Estos PDF no

[9] https://www.urjc.es/estudios/grado

son adecuados para nuestra aplicación por su compleja distribución de contenido. Sin embargo, en la página web también están los datos estructurados, más adecuados para su extracción (ver Figura 3.11).

Entre algunos datos importantes se incluye el nombre del grado, el nombre de las asignaturas junto con sus créditos, el tipo de asignatura y el idioma en el que esta impartida. El motivo de su elección fue la facilidad para su extracción de sus datos[10].

CURSO 1

ASIGNATURA	ECTS	CARÁCTER	IDIOMAS
ANÁLISIS DE FORMAS ARQUITECTÓNICAS I	6	FORMACIÓN BÁSICA	ESPAÑOL/INGLÉS
GEOMETRÍA DESCRIPTIVA I	6	FORMACIÓN BÁSICA	ESPAÑOL/INGLÉS
FUNDAMENTOS MATEMÁTICOS DE LA ARQUITECTURA I	3	FORMACIÓN BÁSICA	ESPAÑOL/INGLÉS
FUNDAMENTOS FÍSICOS DE LA ARQUITECTURA I	3	FORMACIÓN BÁSICA	ESPAÑOL/INGLÉS
INTRODUCCIÓN A LA ARQUITECTURA	6	OBLIGATORIA	ESPAÑOL/INGLÉS
CLAVES DE HISTORIA Y LITERATURA	6	OBLIGATORIA	ESPAÑOL/INGLÉS
ANÁLISIS DE FORMAS ARQUITECTÓNICAS II	6	FORMACIÓN BÁSICA	ESPAÑOL/INGLÉS
GEOMETRÍA DESCRIPTIVA II	6	FORMACIÓN BÁSICA	ESPAÑOL/INGLÉS
DIBUJO ARQUITECTÓNICO I	6	FORMACIÓN BÁSICA	ESPAÑOL/INGLÉS
FUNDAMENTOS MATEMÁTICOS DE LA ARQUITECTURA II	3	FORMACIÓN BÁSICA	ESPAÑOL/INGLÉS
FUNDAMENTOS FÍSICOS DE LA ARQUITECTURA II	3	FORMACIÓN BÁSICA	ESPAÑOL/INGLÉS
ARQUITECTURA Y SOCIEDAD	6	OBLIGATORIA	ESPAÑOL/INGLÉS

Figura 3.11: *Grado en Arquitectura, Universidad CEU San Pablo.*

3.1.2. Análisis y cribado de datos disponibles

Después de haber analizado los datos disponibles mencionados en la anterior sección, ahora toca analizar y seleccionar los atributos más adecuados e informativos teniendo en cuenta las características del proyecto. Estas características son: la duración del proyecto con un tiempo limitado, que será alrededor de 300 horas (lo especificado en teoría para cualquier TFG), la implementación una búsqueda con información útil para los usuarios y la limitación del formato de la información, que en algunos casos esta poco estructurada.

Habiendo observado detalladamente las páginas web de las universidades más destacadas, se localizó una serie de patrones en las características de las asignaturas como es su nombre,

[10] https://www.uspceu.com/oferta/grado

créditos, curso y tipo. También encontramos otra serie de atributos comunes interesantes aunque no tan fáciles de encontrar que podrán ser implementados en futuras versiones. Este es el caso de la descripción del grado, el precio por crédito, las notas de corte y dirección de la facultad.

Por estos motivos, cogeremos solo los datos más importantes para el buen funcionamiento de esta primera versión, haciendo más énfasis en las asignaturas que en los grados en sí. Estos datos son el nombre y dirección web de la universidad, su comunidad autónoma y si es pública o privada, el nombre y dirección web del grado y respecto a las asignaturas: el nombre, los créditos, el tipo (básica, obligatoria u optativa). Cabe destacar que no todas las universidades resaltan el tipo de las asignaturas como es el caso de la Universidad Complutense de Madrid.

3.2. Formatos de información encontrados

Los formatos en los que se encuentra la información son increíblemente heterogéneos, en casi todas las páginas web viene de forma diferente. En algunas páginas toda la información se encuentra en formato HTML dentro de la propia sitio web; en otras páginas viene en un documento PDF adjunto al grado o máster en cuestión; por último en las restantes vienen en formato híbrido: algunos datos en HTML y otros en un documento adjunto. Cada universidad usa un patrón para mostrar la información y al extraer la información hay que diseñar un algoritmo específico de extracción de datos por cada universidad.

3.2.1. Datos en HTML

HTML es un lenguaje de Marcas de Hipertexto, del inglés *HyperText Markup Language*. Es el componente más básico de la Web. Para el procesamiento de datos en HTML hemos usado la gema "Kimurai", que facilita mediante las funciones *xpath()*, *css()* y *at()* la extracción de datos HTML [13, 14].

Al ser un lenguaje basado en etiquetas, cada universidad usa su propia forma de estructurar el contenido y cada una tiene sus propias etiquetas. Incluso dentro de cada universidad

en función de la rama del grado, la información tiene diferentes etiquetas. En cambio en los másteres la información viene peor estructurada, ya que la duración de los mismos es diferente, hay másteres impartidos de forma conjunta por varias universidades.

3.2.2. Datos en PDF

PDF por sus siglas en inglés *Portable Document Format* o en español "Formato de Documento Portátil" es un formato de almacenamiento para documentos digitales. Es un formato muy popular para representar documentos por diversas razones, aquí se exponen algunas: es un formato multiplataforma, no pierde el formato con el envío, se puede cifrar fácilmente, está soportado por muchas aplicaciones y puede contener elementos multimedia[11].

El uso de este formato se ha extendido tanto que se ha convertido en la principal manera para guardar información oficial. Por este motivo algunas universidades lo usan para describir sus grados de forma más visual, pero este método de almacenar información dificulta en exceso su extracción, por problemas como la poca estructuración y otros que se mostrarán más adelante.

3.3. Búsquedas en Sphinx: Thinking Sphinx

Sphinx es un motor de búsqueda basado en la indexación de contenido. Se usa principalmente en lenguaje Python, pero al elegir el lenguaje Ruby y el framework Ruby on Rails para la implementación necesitamos de una gema que conecte Sphinx con la arquitectura Modelo Vista-Controlador usada por Ruby on Rails. Esta gema o librería es "thinking-sphinx".

La librería "thinking-sphinx" nos ayuda a vincular Sphinx con *Active Record* y, así, podemos buscar en múltiples campos usando SQL de manera sencilla. *Active Record* es la parte del modelo en el patrón de arquitectura Modelo Vista-Controlador, con la que se puede generar fácilmente con un comando la estructura básica a partir del modelo [9].

[11] https://es.wikipedia.org/wiki/PDF

3.4. Preparación del entorno: Ruby on Rails

Después de haber analizado y seleccionado la información necesaria sobre las asignaturas de los grados y saber cómo va a funcionar la búsqueda con Sphinx, el siguiente paso corresponde a desarrollar la estructura de la aplicación. Como ya se mencionó anteriormente, se eligió el framework Ruby on Rails, un marco con una arquitectura predefinida Modelo Vista-Controlador, muy adecuada para este tipo de aplicación con varias capas de implementación.

Gracias a Ruby on Rails la creación de un proyecto se vuelve bastante sencilla, después de instalar la gema "Rails", con un simple comando se forma toda la estructura Modelo Vista-Controlador ("`rails new "nombre del proyecto"`"). Además se puede añadir algunas opciones como elegir la base de datos con la opción "-d" y en el caso de nuestro proyecto resulta vital usar una base de datos MySQL para poder enlazarla con Sphinx.

Después de la creación del proyecto se instalan las gemas necesarias como la de "thinking-sphinx" o la gema "devise" y se ejecuta el comando "`bundle install`" que nos ayuda a instalar todas las dependencias necesarias automáticamente.

Para la preparación del entorno de vistas hemos usado *Bootstrap*, un framework que proporciona librerías de estilos prediseñadas. Hay una gran comunidad detrás de *Bootstrap*, proporcionando plantillas y componentes listos para añadir a la aplicación. Para la preparación del entorno se han usado componentes prediseñados [15].

También hay otra librería interesante llamada *Scaffold* [16], la cual ayuda a crear automáticamente la estructura según la arquitectura de *Rails*. Estos archivos incluyen:

- Un controlador.
- Un modelo.
- Vistas para cada acción estándar del controlador (indexar, editar, mostrar, nuevo).

En nuestro proyecto, un ejemplo para la creación de una estructura creada con *Scaffold* (ver Figura 3.12), podemos ver el comando *Scaffold* y la estructura generada sobre

Asignatura_Master.

```
fernando@fernando-VirtualBox:~/pdfReader$ rails g scaffold AsignaturaMaster nombre:string curso:string tipo:string creditos:integer master:references
Running via Spring preloader in process 7450
      invoke  active_record
      create    db/migrate/20210914152659_create_asignatura_masters.rb
      create    app/models/asignatura_master.rb
      invoke    test_unit
      create      test/models/asignatura_master_test.rb
      create      test/fixtures/asignatura_masters.yml
      invoke  resource_route
       route    resources :asignatura_masters
      invoke  scaffold_controller
      create    app/controllers/asignatura_masters_controller.rb
      invoke    erb
      create      app/views/asignatura_masters
      create      app/views/asignatura_masters/index.html.erb
      create      app/views/asignatura_masters/edit.html.erb
      create      app/views/asignatura_masters/show.html.erb
      create      app/views/asignatura_masters/new.html.erb
      create      app/views/asignatura_masters/_form.html.erb
      invoke    resource_route
      invoke    test_unit
      create      test/controllers/asignatura_masters_controller_test.rb
      create      test/system/asignatura_masters_test.rb
      invoke    helper
      create      app/helpers/asignatura_masters_helper.rb
      invoke      test_unit
      invoke    jbuilder
      create      app/views/asignatura_masters/index.json.jbuilder
      create      app/views/asignatura_masters/show.json.jbuilder
      create      app/views/asignatura_masters/_asignatura_master.json.jbuilder
      invoke  assets
      invoke    scss
      create      app/assets/stylesheets/asignatura_masters.scss
      invoke  scss
   identical    app/assets/stylesheets/scaffolds.scss
```

Figura 3.12: *Comando Scaffold con Asignatura_Master.*

3.5. Casos de uso

En esta sección, se encuentran los casos de uso que engloban las funcionalidades básicas de cualquier aplicación que tenga usuarios en sus bases de datos, es decir, el registro, el inicio de sesión y cerrar sesión. Además, las incluye para ver la información de las entidades (comunidades, universidades, grados, másteres, asignaturas de grado y asignaturas de másteres). También incluye las funcionalidades de hacer búsquedas de grados y másteres. Los usuarios con permisos de administrador o "admin" también podrán actualizar la información por cada universidad. Aquí se muestra un diagrama mostrando los casos de uso (ver Figura 3.13) y a continuación tablas describiendo los casos de uso.

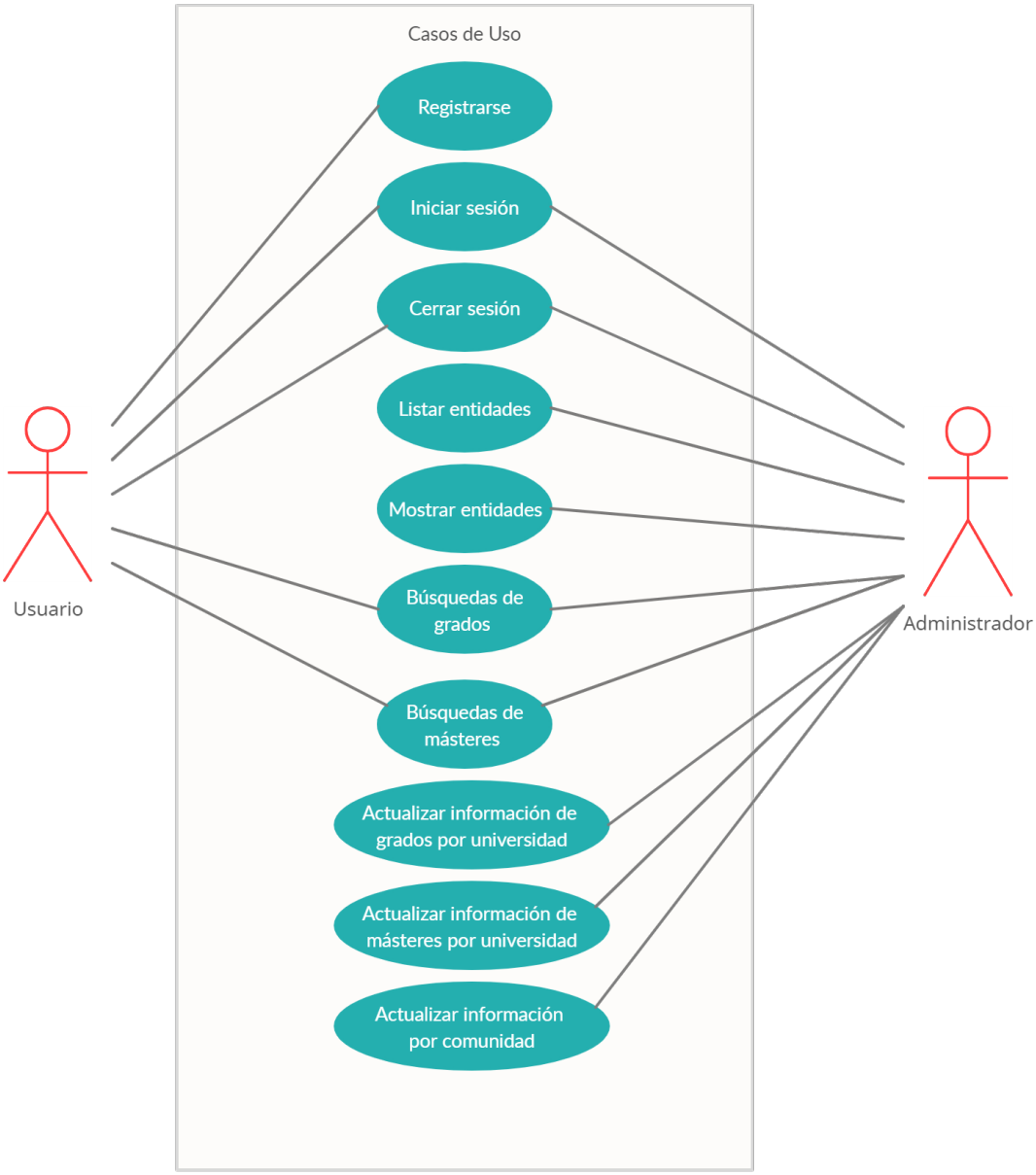


Figura 3.13: *Casos de uso.*

01 Registrarse	
Actores	Alumno.
Descripción	El usuario se registra en la aplicación solo con el rol de alumno.
Precondición	El usuario no debe de estar dado de alta previamente.
Secuencia normal	1. El sistema muestra un botón «Sign up». 2. Al pinchar en el botón redirige a una página de rellenar un formulario para registrarse. 3. Al introducir los datos correctamente da de alta el usuario.
Postcondición	El sistema genera automáticamente un registro con los datos. El sistema informa al usuario que se ha dado de alta.
Excepciones	Si se introducen los campos del formulario con formatos inválidos hace que salte un error, notificando al usuario de ello.
Comentarios	El rol de administrador solo se puede dar de alta introduciéndolo directamente en la base de datos.

Tabla 3.1: *Caso de uso 01. Registrarse.*

02 Iniciar sesión	
Actores	Alumno y administrador.
Descripción	El alumno o administrador puede iniciar sesión para poder utilizar las funcionalidades de la aplicación.
Precondición	El usuario debe de estar dado de alta previamente.
Secuencia normal	1. El sistema muestra un botón «Log in». 2. Al pinchar en el botón redirige a una página de rellenar un formulario para validarse en el sistema. 3. Al introducir los datos correctamente se valida en el sistema.
Postcondición	El sistema genera automáticamente una sesión y puede acceder a sus funcionalidades.
Excepciones	Si los datos del formulario no son correctos, salta un error notificando de esto al usuario.
Comentarios	N/A

Tabla 3.2: *Caso de uso 02. Iniciar sesión.*

03 Cerrar sesión	
Actores	Alumno y administrador.
Descripción	El usuario puede cerrar sesión para iniciar otra sesión o registrar a otro usuario.
Precondición	El usuario debe de estar dado de alta previamente.
Secuencia normal	1. El sistema muestra un botón «Log out». 2. Al pinchar en el botón redirige a una página de rellenar un formulario para validarse en el sistema. 3. Al introducir los datos correctamente se valida en el sistema.
Postcondición	El sistema genera automáticamente una sesión y puede acceder a sus funcionalidades.
Excepciones	Si los datos del formulario no son correctos, salta un error notificando de esto al usuario.
Comentarios	N/A

Tabla 3.3: *Caso de uso 03. Cerrar sesión.*

04 Listar entidades	
Actores	Administrador.
Descripción	El administrador puede acceder a un listado de información de las diferentes entidades: comunidades, universidades, grados, másteres, asignaturas de grados y asignaturas de másteres; para ver sus respectivos atributos.
Precondición	El usuario debe de estar validado previamente.
Secuencia normal	1. El sistema muestra una barra de navegación en donde hay botones con las entidades (comunidades, universidades, grados, másteres, asignaturas de grados y asignaturas de másteres). 2. Al pinchar un botón se redirige a una página con un listado de datos de la entidad seleccionada con sus respectivos atributos.
Postcondición	Se genera un listado de la entidad elegida.
Excepciones	No se mostrará nada si no hay entidades dadas de alta en el sistema.
Comentarios	N/A

Tabla 3.4: *Caso de uso 04. Listar entidades.*

05 Mostrar entidades	
Actores	Administrador.
Descripción	El administrador puede acceder a ver una información de cualquier entidad: comunidades, universidades, grados, másteres, asignatura de grados y asignatura de másteres. Puede ver sus respectivos atributos en detalle.
Precondición	El usuario debe de estar validado previamente.
Secuencia normal	1. El sistema muestra una barra de navegación en donde hay botones con las entidades (comunidades, universidades, grados, másteres, asignaturas de grados y asignatura de másteres). 2. Al pinchar un botón se redirige a una página con un listado de datos de la entidad seleccionada. Cada entidad muestra un botón «Show». 3.Al pinchar en el botón «Show» te dirige a una vista detallada de atributos de esa entidad.
Postcondición	Se muestra la entidad elegida con sus atributos.
Excepciones	N/A
Comentarios	N/A

Tabla 3.5: *Caso de uso 05. Mostrar entidades.*

06 Búsquedas de grados	
Actores	Alumno y administrador.
Descripción	El alumno o administrador puede hacer búsquedas de asignaturas de grados con varios filtros: por palabras contenidas en el nombre de la asignatura y por universidad.
Precondición	El usuario o administrador deben de estar validados previamente.
Secuencia normal	1. El sistema muestra un botón en la barra de navegación que pone «UniSurfing». 2. Al pinchar un botón redirige a la página principal donde se pueden hacer búsquedas por grado y por máster. Se muestra un botón «Buscar Grado». 3. Al pinchar el botón «Buscar Grado» te redirige a una página de búsqueda donde hay 2 filtros: por nombre y por universidad. 4. Se introduce en el recuadro de grados la palabra a buscar y/o se selecciona la universidad y se pincha el botón «buscar». 5. Redirige a un listado de grados que contienen esa palabra en el nombre de una asignatura y/o la universidad elegida.
Postcondición	Se genera un listado de asignaturas.
Excepciones	No se mostrará nada si no hay asignaturas dadas de alta en el sistema que no contengan esa palabra y/o la universidad seleccionada.
Comentarios	N/A

Tabla 3.6: *Caso de uso 06. Búsquedas de grados.*

07 Búsquedas de másteres	
Actores	Alumno y administrador.
Descripción	El alumno o administrador puede hacer búsquedas de asignaturas de másteres con varios filtros: por palabras contenidas en el nombre de la asignatura y por universidad.
Precondición	El alumno o administrador deben de estar validados previamente.
Secuencia normal	1. El sistema muestra un botón en la barra de navegación que pone «UniSurfing». 2. Al pinchar un botón redirige a una página donde se pueden hacer búsquedas por grado y por máster. Se muestra un botón «Buscar Máster». 3. Al pinchar el botón «Buscar Máster» te redirige a una página de búsqueda donde hay un filtro por nombre y por universidad. 4. Se introduce en el recuadro de másteres la palabra a buscar y/o se selecciona la universidad y se pincha el botón «buscar». 5. Redirige a un listado de grados que contienen esa palabra en el nombre de una asignatura y/o la universidad elegida.
Postcondición	Se genera un listado de asignaturas.
Excepciones	No se mostrará nada si no hay asignaturas dadas de alta en el sistema que no contengan esa palabra y/o la universidad.
Comentarios	N/A

Tabla 3.7: *Caso de uso 07. Búsquedas de másteres.*

08 Actualizar información de grados por universidad	
Actores	Administrador.
Descripción	Los administradores pueden actualizar toda la información de los grados de una universidad.
Precondición	El usuario debe de estar validado previamente.
Secuencia normal	1. En la barra de navegación hay un botón «universidades». 2. Si se pincha en el botón te dirige a la página listar universidades. En cada universidad hay un botón «*Scrape*». 3. Si se pulsa el botón «*Scrape*» se actualizará la información de los grados de esa universidad.
Postcondición	Se actualizan la información de la universidad elegida.
Excepciones	Si no se consigue extraer información exitosamente de un grado no se actualizará esa información.
Comentarios	N/A

Tabla 3.8: *Caso de uso 08. Actualizar información de grados por universidad.*

09 Actualizar información de másteres por universidad	
Actores	Administrador.
Descripción	Los administradores pueden actualizar toda la información de los másteres de una universidad.
Precondición	El usuario debe de estar validado previamente.
Secuencia normal	1. En la barra de navegación hay un botón «universidades». 2. Si se pincha en el botón te dirige a la página listar universidades. En cada universidad hay un botón «*ScrapeMaster*». 3. Si se pulsa el botón «*ScrapeMaster*» se actualizará la información de los másteres de esa universidad.
Postcondición	Se actualizan la información de la universidad elegida.
Excepciones	Si no se consigue extraer información exitosamente de un máster no se actualizará esa información.
Comentarios	N/A

Tabla 3.9: *Caso de uso 09. Actualizar información de másteres por universidad.*

10 Actualizar información por comunidad	
Actores	Administrador.
Descripción	Los administradores pueden actualizar toda la información de las universidades de una comunidad.
Precondición	El usuario debe de estar validado previamente.
Secuencia normal	1. En la barra de navegación hay un botón «Comunidades». 2. Si se pincha en el botón te dirige a la página listar comunidades. En cada universidad hay un botón «*Scrape*». 3. Si se pulsa el botón «*Scrape*» se actualizará la información de los grados y másteres de esa universidad.
Postcondición	Se actualizan la información de la universidad elegida.
Excepciones	N/A
Comentarios	El proceso puede tardar bastante tiempo (alrededor de una hora).

Tabla 3.10: *Caso de uso 10. Actualizar información por comunidad.*

Capítulo 4

Diseño para la Extracción y Tratamiento de los Datos

En este capítulo trataremos el tema del diseño hecho para aclarar los datos objetivos que queremos junto con su estructura, tanto en la web como en nuestra base de datos. Este diseño y los preparativos para la extracción son vitales para que un proyecto sea transparente y tenga clara sus metas. Esta fase también es importante ya que es donde se redefinen los objetivos a partir de los conocimientos desarrollados en la elaboración del diseño y el análisis exhaustivo de los datos preseleccionados.

4.1. Información a gestionar

En esta sección se detallarán los datos seleccionados que compondrán las entidades de la aplicación. Parte de estos datos han sido extraídos de páginas web y otros han sido introducidos por los usuarios administradores. A continuación se detallarán las entidades de la aplicación:

- **Comunidad**: Aquí se guardarán las comunidades implementadas en el sistema. Para esta versión se introducirá solo la de la Comunidad de Madrid. Esta solo tiene el campo nombre de la comunidad.

- **Universidades**:
  - Comunidad: Referencia a la comunidad autónoma a la que pertenece.
  - Universidad: Nombre de la universidad.
  - URL: Dirección de la página web oficial en la sección de grados.
  - Tipo: Tipo de universidad, si es pública o privada.
- **Grado**:
  - Grado: Nombre del grado.
  - URL: Dirección de la página web oficial en la sección del grado en cuestión.
  - Universidad: Referencia a la universidad que pertenece.
- **Asignatura**:
  - Asignatura: Nombre de la asignatura.
  - Tipo: Indica si se trata de una asignatura básica, obligatoria o optativa.
  - Creditos: Número de créditos de vale la asignatura.
  - Curso: Año académico en el que se cursa la asignatura.
  - Grado: Referencia al grado al que pertenece.
- **Master**:
  - Nombre: Nombre del máster.
  - URL: Dirección de la página web oficial en la sección del máster en cuestión.
  - Universidad: Referencia a la universidad.
- **Asignatura_Master**:
  - Asignatura: Nombre de la asignatura.

- Tipo: Indica si se trata de una asignatura básica, obligatoria o optativa.
- Creditos: Número de créditos de vale la asignatura.
- Curso: Año académico en el que se cursa la asignatura.
- Master: Referencia al máster al que pertenece.

4.2. Diagrama de diseño

Para esta sección se presentarán el modelo de diseño de la base de datos a un nivel más esquemático. A continuación se mostrarán el diagrama de diseño entidad-relación de la base de datos, haciendo más visual e interpretable esta parte. El diagrama relacional de desarrollará más adelante en la memoria (ver Figura 4.1).

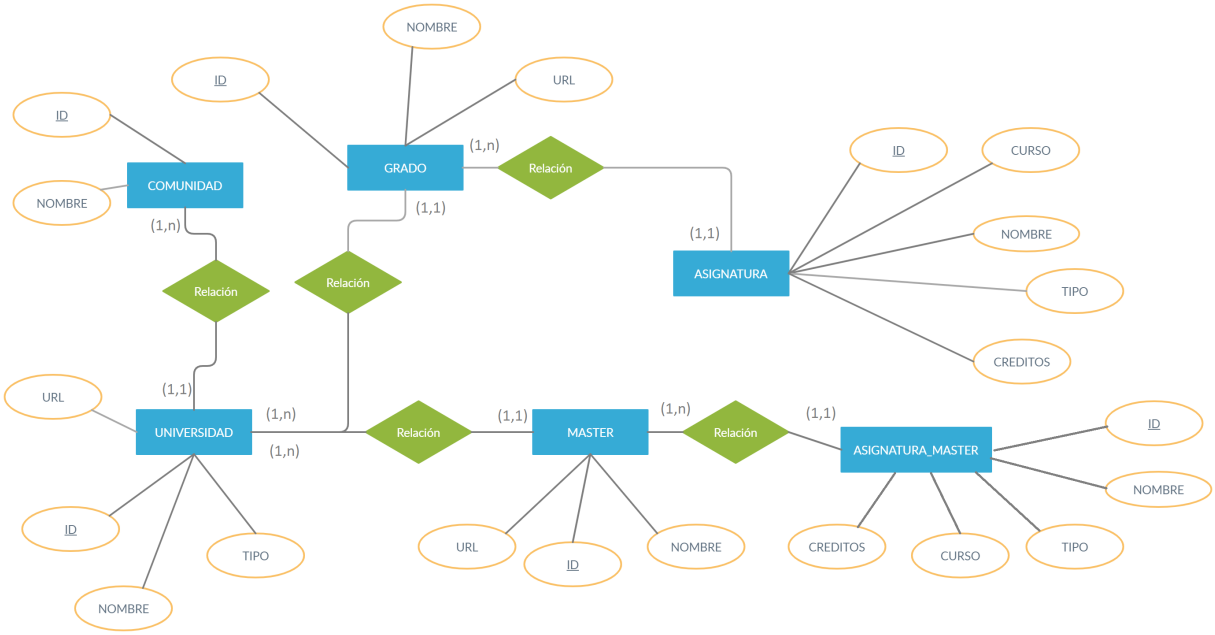


Figura 4.1: *Diagrama Entidad-Relación.*

4.3. Datos de entrada

4.3.1. Extracción de datos desde HTML

Para la extracción de información de un grado de una universidad concreta se han de sacar patrones comunes de diseño por etiquetas HTML. Para ello todas las universidades tienen una dirección web que contiene un listado de grados/másteres impartidos en la propia universidad. Un ejemplo de esta práctica es la lista de grados de la Universidad Carlos III de Madrid (ver Figura 4.2).

Figura 4.2: *Listado de grados de la Universidad Carlos III de Madrid.*

Ahora hay que analizar las etiquetas HTML para sacar un patrón común, la mayor complicación es que cada universidad usa formatos y estructuras completamente diferentes. Además, cada universidad en función de la rama de estudio del grado (ciencias, ingeniería, ciencias sociales, etc) usa diferentes formatos. En los másteres es peor, ya que hay másteres impartidos por diferentes facultades, de diferente duración, etc.

A pesar de esta diferencia de formatos, hay pasos comunes para la extracción de datos con todas las universidades. El primer paso es crear un objeto "Nokogiri" (objeto implementado en la gema "Kimurai"), pasándole la URL en HTML, obtenida con el comando "`CURL`". El segundo paso es elegir uno de los selectores que ofrece la gema "Kimurai" *css()*, *at()* y *xpath()*, podemos analizar el código HTML para sacar patrones comunes para extraer el texto deseado. Por ejemplo,"objetoNokogiri.css("td/text()")" devolvería el texto contenido, pero en una fila concreta este texto viene con la etiqueta "`<strong>`", esa función no devolverá nada.

Para que devuelva el contenido, habría que hacerlo así:
"`objetoNokogiri.css("td/strong/text()")`". Además para extraer la dirección URL que contiene la información concreta, viene dentro de la etiqueta "`<a href=ürl»</a>`". Por lo que para poder coger esa "`url`" de dentro del "`href`", habría que hacerlo de la siguiente forma: "`objetoNokogiri.css('a/@href')`".

Finalmente, la identificación de etiquetas concretas como "`div`", "`table`" u otras similares, se puede hacer siempre que contengan el parámetro "`id`", que las hace únicas de la forma "`<div id='asignatura'></div>`". Para esta etiqueta sería con el selector
"`objetoNokogiri.css("div[@id='asignatura']")`". Si por el contrario en vez de tener el identificador lo que se tiene es una clase: "`<div class='asignatura'></div>`" al hacer "`objetoNokogiri.css("div[@class='asignatura']")`", devolvería un *array* de objetos *Nokogiri* que contendría todos los elementos que tengan esa etiqueta clase. Por ejemplo, si se quiere solo el segundo "`div`" de este tipo, se haría con el selector "`at`", de la siguiente forma: "`objetoNokogiri.at("div[@class='asignatura'][2]")`".

4.3.2. Extracción de datos desde ficheros PDF

Hemos usado la gema "PDF-reader" para la extracción de datos de PDF. Esta gema tiene configuradas diferentes funciones y, en concreto, hemos usado una función que extrae línea a línea de cada página. Aunque cada universidad usa formatos en PDF diferentes, dentro de

cada universidad la información de las asignaturas se organiza con un formato más o menos fijo. El análisis ha sido el siguiente:

- Contar el número de espacios entre cada tipo de información (código de asignatura, nombre, tipo asignatura y créditos).
- En función del tamaño en caracteres de dicha información, aunque en la Politécnica el código de asignatura tiene diferente longitud en función del grado.
- Si es una información que empieza por letra o por número.
- Cada línea de marcaje de curso contiene una palabra clave, ya sea ECTS, conteniendo la palabra curso o código.

En este proceso, los algoritmos los creamos de cero manualmente, el análisis y la implementación de nuevas soluciones se validan con prueba y error. Primero se analizan las variables obtenidas y se intentan descubrir patrones que faciliten la identificación de las diferentes entidades y atributos. Después se genera un algoritmo sobre un posible patrón, común a todos los grados de la universidad. En último lugar se desarrolla el código y se comprueba la autenticidad de los datos obtenidos en la ejecución, con los contenidos en el PDF de donde se extraen esos datos.

Los pasos generales en el código quedarían de la siguiente forma: primero se leen las páginas del PDF cogiendo solo las que contengan información relevante. Posteriormente, se itera por cada línea y se convierte la línea *string* actual en un *array* separado por espacios. Después se salta el contenido hasta encontrar el primer curso. Se sigue iterando si la línea es nula, vacía o las líneas que no contienen los datos objetivo (tienen diferente longitud). En último lugar se leen los datos de la asignatura si tienen identificador o número de créditos y se almacenan en una variable.

Durante el proceso el nombre de asignatura se tiene que convertir a *string* recorriendo el *array*. También son leídos los créditos si son mayores que 0 y menor o igual que 18, ya

que 18 créditos son el máximo de créditos por asignatura que se ha visto (en la mayoría de grados son 12). Además se lee el tipo si tiene y se transforma en un *string* común a todos.

El problema más común de este proceso viene cuando cada asignatura viene en dos o más líneas. No es tan sencillo como añadir esa línea sin créditos o identificador numérico a la anterior. En el ejemplo de la Universidad Complutense de Madrid, la librería de "PDF-reader" no sigue un patrón claro, a veces lee los créditos en una línea a parte, otras en cualquiera de las líneas céntricas. Esto es debido a que el PDF de la Complutense esta en formato díptico y si bien visualmente se puede comprender fácilmente, es más difícil para un intérprete software.

También hay otros problemas añadidos como la longitud variable de los elementos que en algunos casos hace muy difícil distinguir contenido sin relevancia de los datos objetivos. La distribución de optativas en itinerarios, repetición de asignaturas, cambio de idioma que dificulta la identificación de palabras clave y por su puesto el cambio de estructura del documento.

Capítulo 5

Implementación de la Aplicación

5.1. Arquitectura de la aplicación

Con el framework Ruby on Rails para crear aplicaciones web se usa por defecto el patrón de arquitectura MVC (Modelo Vista-Controlador). Además se utiliza *Active Record* con un ORM *(Object-Relational Mapping)* es un modelo de programación que permite mapear estructuras de una base de datos relacional, sobre una estructura lógica de entidades. Para crear las rutas se sirve de REST (*REpresentational State Transfer*) que es un paradigma para definir rutas en aplicaciones web. En base a REST, las aplicaciones de *Ruby on Rails* determinan qué parte de aplicación mostrar y cómo responder a las solicitudes del usuario[1]. La arquitectura se puede ver a continuación (ver Figura 5.1).

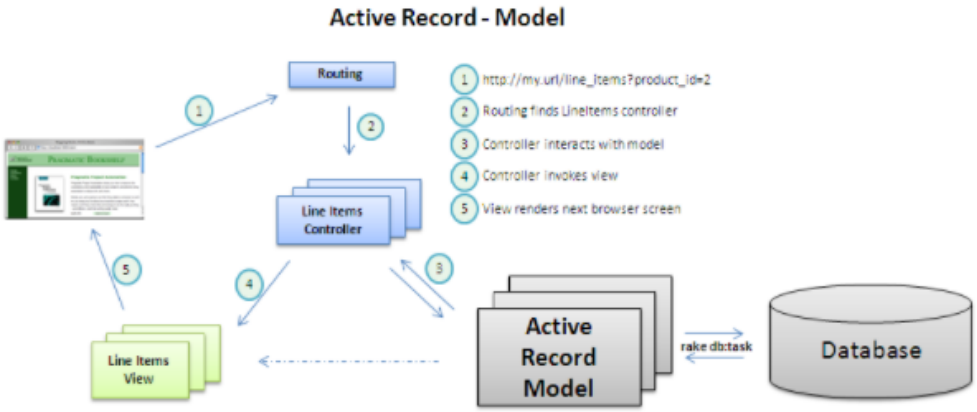


Figura 5.1: *Arquitectura Ruby on Rails.*

[1] https://dylanninin.com/blog/2013/11/25/rails4_ar.html/

5.2. Base de datos: construcción del modelo de datos

El diseño de la base de datos se hizo de forma que todas las tablas estuvieran relacionadas (ver Figura 5.2). Estas tablas están desarrolladas para que la aplicación pueda ampliar su ámbito fuera de la Comunidad de Madrid. También está pensada para poder añadir en el futuro más tipos de estudios que pueda tener una universidad, como cursos de formación o ciclos superiores de formación.

Con el objetivo futuro de hacer ampliable la aplicación a más comunidades autónomas, se ha decidido añadir la tabla Comunidad aunque solo tengamos la Comunidad de Madrid. Asimismo de los grados y los másteres, como las asignaturas de grados y las asignaturas de másteres, quedando así una base de datos con seis tablas:

- **Tabla Comunidad**. Esta tabla es para poder ampliar la aplicación a más comunidades autónomas.
- **Tabla Universidad**. Esta tabla tiene las universidades de las que la aplicación tiene información. Tiene una clave foránea que referencia a la comunidad que pertenece.
- **Tabla Grado**. Esta tabla contiene la información de los grados, tiene una clave foránea que referencia a su universidad.
- **Tabla Master**. Esta tabla contiene la información de los másteres, tiene una clave foránea que referencia a su universidad.
- **Tabla Asignatura**. Esta tabla contiene la información de los asignaturas impartidas en los grados, tiene una clave foránea que referencia al grado dónde se imparte.
- **Tabla Asignatura_Master**. Esta tabla contiene la información de los asignaturas impartidas en los másteres, tiene una clave foránea que referencia al grado dónde se imparte.

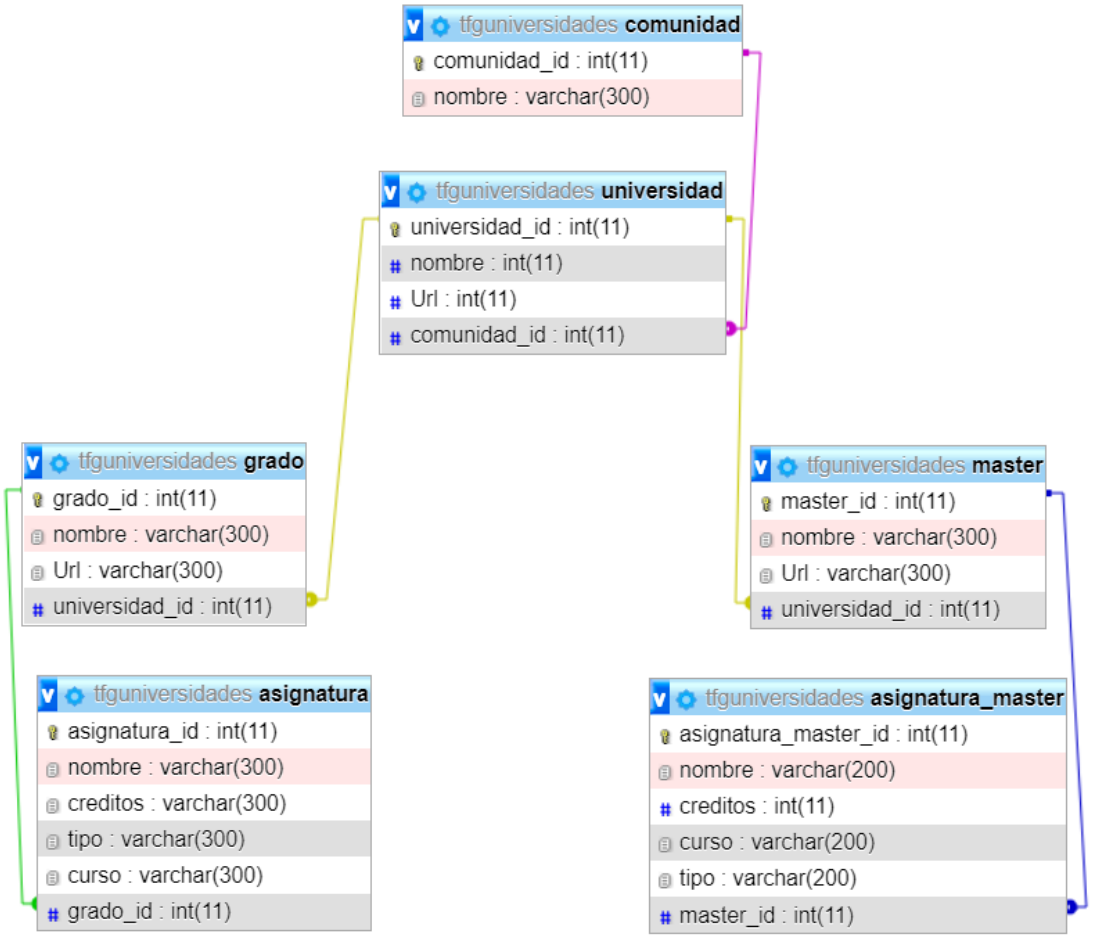


Figura 5.2: *Diagrama relacional de la base de datos.*

5.3. Implementación de funcionalidades: back-end

La aplicación tiene diferentes funcionalidades dependiendo del rol. Los usuarios registrados, por defecto se les otorga el rol "estudiante". Esto lo hemos implementado con la gema "Rolify" y asignando en una función del *modelo User*, una función para los usuarios nuevos registrados (ver Figura 5.3). Estos usuarios registrados y los usuarios no registrados tienen las mismas funcionalidades.

```ruby
class User < ApplicationRecord
        rolify
        # Include default devise modules. Others available are:
        # :confirmable, :lockable, :timeoutable, :trackable and :omniauthable
        devise :database_authenticatable, :registerable,
               :recoverable, :rememberable, :validatable

        after_create :assign_default_role

        def assign_default_role
           self.add_role(:estudiante) if self.roles.blank?
        end
end
```

Figura 5.3: *Asignar rol "estudiante" por defecto a usuarios registrados.*

La funcionalidad principal de la aplicación es buscar el grado o el máster que contenga una asignatura dada, la pueden realizar todos los usuarios, registrados (rol "estudiante" y rol "admin") y no registrados. A continuación explicamos cómo hemos implementado esta funcionalidad:

El buscador de grados y másteres es un formulario HTML, que en el äctionÏlama a su función correspondiente del controlador *static_pages*, dependiendo de si se buscan másteres se llamará a la función del controlador *static_pages* "`buscadorMaster`" y si se buscan grados se llamará a la función del controlador static_pages "`buscador`" (ver Figura 5.4).

```erb
  <div class="row justify-content-center" >
    <div class="col-lg-2 col-md-2 col-sm-12 align-content-lg-center">
      <!--select1-->
      <%= form_tag("/static_pages/buscador", method: "get") do %>
  <%= label_tag(:nombre, "Search for:") %>
  <%= text_field_tag(:nombre) %>
  <%= submit_tag("Search") %>
<% end %>
```

Figura 5.4: *Formulario para buscar grados.*

Las dos funciones del controlador *static_pages* ("`buscador`" y "`buscadorMaster`") funcionan de forma idéntica, solo se diferencian si se buscan grados o másteres. Primero declaramos un *array* de asignaturas vacío y si hemos definido el parámetro de búsqueda "`:nombre`",

se procede a realizar una búsqueda con el motor de búsqueda Sphinx con el nombre de asignatura indicado en el parámetro "`:nombre`".

Para configurar Sphinx hay que crear dentro del directorio *app/*, una carpeta *index/*, que contendrá un archivo por cada modelo de la base de datos a indexar. Para ello se recomienda que estos archivos se llamen igual que el modelo indexado, seguido de una barra baja y la palabra *index*. También hay que indicar qué campos de la base de datos queremos indexar con Sphinx y, adicionalmente, se pueden añadir atributos, como que esos campos sean ordenables.

Para terminar de configurar Sphinx y que cada búsqueda solo muestre resultados si la búsqueda es exacta. Es decir, solo mostraría resultados en caso de encontrar literales. Para que muestre resultados parciales, tenemos que añadir un nuevo archivo en el carpeta *config*, este archivo se llama *thinking_ sphinx.yml*, y aquí se puede elegir una configuración avanzada de Sphinx.

Una vez creado el archivo de la forma *asignatura_ index.rb*, y configurando los parámetros deseados, se puede hacer un consulta con Sphinx. Es en este archivo donde tienes que incluir sobre qué atributos quieres hacer los índices en Sphinx. Además puedes seleccionar diferentes opciones sobre estos parámetros, en nuestro caso, solo hemos puesto que se puedan ordenar con el parámetro "`:sortable =>true`" (ver Figura 5.5).

app > indices > asignatura_index.rb

```
1   ThinkingSphinx::Index.define :asignatura, :with => :active_record do
2       indexes nombre, :sortable => true
3       indexes grado_id, :sortable => true
4
5
6
7
8
9
10
11
12  end
```

Figura 5.5: *Archivo de configuración índices de Sphinx.*

Ahora vamos a definir las diferentes implementaciones del rol "admin". Los usuarios

registrados con este rol, que de momento solo son los usuarios con el identificador 1 o 2, son los únicos que pueden hacer *web scraping* de cada universidad. Esta decisión la tomamos porque cada universidad de forma individual tarda entre 5 y 10 minutos en hacer el *web scraping* de sus grados o másteres.

Para hacer el *web scraping*, el primer paso es hacer una petición *CURL* [17]. Con esta petición y el parámetro user-agent configurado como "`Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)`", conseguimos el contenido de la URL sobre la que estamos haciendo la petición *CURL*. Si la dirección existe, este comando devuelve el código HTML que contiene la dirección URL sobre la que se ha hecho *CURL*. El resultado de esta petición es el que le pasamos a una función de la gema "Kimurai", para poder extraer el código. Un ejemplo del comando *CURL* sería el siguiente comando (ver Figura 5.6).

```
paga = `curl --user-agent "Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)" https://www.uc3m.es/grado/estudios`
```

Figura 5.6: *Comando CURL usado.*

Una vez creado el objeto *Nokogiri*, usando los selectores implementados en las funciones *xpath()*, *css()* y *at()*. Cada universidad tiene una URL concreta donde encontramos todo el listado de grados que ofertan y otra URL concreta para todo el listado de másteres que ofertan. En estas URLs mediante los selectores descritos antes, seleccionamos la URL concreta y creamos el objeto grado o máster correspondiente a cada estudio, con su nombre y URL correspondiente. Un ejemplo de código de extracción de URLs concretas que contienen la información de cada grado es el siguiente (ver Figura 5.7).

Por cada objeto tipo máster o tipo grado, llamamos a otro método que se llama "`parse_url()`", donde volvemos a realizar otra petición CURL con la URL que contienen la información de cada grado o máster concreto. Aunque hay alguna universidad, que necesita que se repita otro CURL, porque tienen la información de las asignaturas en otra URL diferente donde tienen otra información concreta del grado. Dependiendo de si la información de la asignatura está contenida en un PDF o en HTML, procedemos a extraer la información de las

asignaturas(ver Figura 5.8). La explicación de los algoritmos seguidos en la extracción de datos está explicada en la Sección 4.3.1, Extracción de datos HTML y en la Sección 4.3.1, Extracción de datos PDF.

```
response2.xpath("//div[@class='col span_12']").each do |tabla|

curso=tabla.css('h2').text

tabla.css("table").each do |proba|

  proba.css("tr").each do |asignatura|

     nombre_asignatura   = asignatura.css("td[1]/a/text()").to_s
     ects = asignatura.css("td[2]/text()").to_s

     if nombre_asignatura != "Trabajo Fin de Grado" && nombre_asignatura != "Optativas: Recomendado elegir 12 créditos"

          if ects.length>0

             item2 = data2

             item2[:nombre]      = nombre_asignatura
             item2[:creditos] = ects
             tipo_asignatura      = asignatura.css("td[3]/text()").to_s
```

Figura 5.7: *Código extracción para crear un grado.*

```
response2.xpath("//div[@class='col span_12']").each do |tabla|

curso=tabla.css('h2').text

tabla.css("table").each do |proba|

  proba.css("tr").each do |asignatura|

     nombre_asignatura   = asignatura.css("td[1]/a/text()").to_s
     ects = asignatura.css("td[2]/text()").to_s

     if nombre_asignatura != "Trabajo Fin de Grado" && nombre_asignatura != "Optativas: Recomendado elegir 12 créditos"

          if ects.length>0

             item2 = data2

             item2[:nombre]      = nombre_asignatura
             item2[:creditos] = ects
             tipo_asignatura      = asignatura.css("td[3]/text()").to_s
```

Figura 5.8: *Código extracción de la información para crear una asignatura.*

Para asegurarnos de que no se repiten las asignaturas y los grados, cada vez que hacemos *scraping* (para los grados o para los másteres) de una universidad, borramos todos los grados o másteres con sus asignaturas de esa universidad. Además, después de introducir todas las

asignaturas de un grado en la base de datos, comprobamos que si existen otros grados en la base de datos, que tengan el mismo nombre, que pertenezcan a la misma universidad y que se borren todos los anteriores, para que la información del grado se actualice.

5.4. Visualización de las salidas: front-end

Hemos decidido hacer un diseño práctico y funcional con el framework front-end de *Bootstrap*. En la página principal puedes elegir si quieres buscar másteres o grados. Y una vez elegido qué quieres buscar, en la correspondiente vista, tienes un sencillo buscador para elegir qué grado o máster buscar que contenga una determinada asignatura.

Además, como en las vistas donde solo tienen acceso los usuarios con el rol "admin", hemos optado por un diseño básico de tablas por cada tabla de la base de datos. La primera diferencia evidente es el menú de navegación, los usuarios con rol "admin" tienen más pestañas para navegar. A continuación explicaremos, cada pestaña de la barra de navegación para los usuarios con rol "admin": Comunidades, Universidades, Grados, Másteres, Asignaturas de grados y Asignaturas de másteres (ver Figura 5.9).

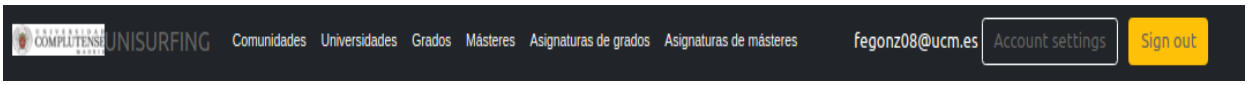


Figura 5.9: *Menú de navegación para usuarios con rol "admin".*

A destacar, en la vista universidades, los usuarios con rol "admin" pueden ver cuántas universidades hay en el sistema, y realizar *scraping* de grados de cada universidad, al hacer *click* en el botón "Scrape grados". Al hacer *click* en el botón "Scrape másteres" pueden realizar *scraping* de los másteres. Además se puede borrar todos los grados o másteres asociados a una universidad concreta. Si se quiere realizar *scraping* a la vez de todas las universidades de una Comunidad Autónoma, podemos ir a la pestaña Comunidades, y al hacer click en el botón "Scrape" correspondiente de cada universidad (ver Figura 5.10).

Id	Comunidad	Nombre	Url	Tipo	Grados	Másteres	Mostrar Universidad	Scrape grados	Scrape másteres	Borrar grados	Borrar másteres
1	Comunidad de Madrid	Universidad de Alcalá	https://www.uah.es/es/estudios/estudios-oficiales/grados/	Pública	33	1	Show	Scrape grados	Scrape másteres	Borrar grados	Borrar másteres
2	Comunidad de Madrid	Universidad Rey Juan Carlos	https://www.urjc.es/estudios/grado	Pública	145	0	Show	Scrape grados	Scrape másteres	Borrar grados	Borrar másteres
3	Comunidad de Madrid	Universidad Carlos III de Madrid	https://www.uc3m.es/grado/estudios	Pública	53	134	Show	Scrape grados	Scrape másteres	Borrar grados	Borrar másteres
4	Comunidad de Madrid	Universidad Alfonso X el Sabio	https://www.uax.com/grados	Privada	50	36	Show	Scrape grados	Scrape másteres	Borrar grados	Borrar másteres
5	Comunidad de Madrid	Universidad Nebrija	https://www.nebrija.com/lp/2021/grado/marca/?Cod_TipoFRM=2039&gclsrc=aw.ds&ds_rl=1291044	Privada	29	0	Show	Scrape grados	Scrape másteres	Borrar grados	Borrar másteres
6	Comunidad de Madrid	Universidad Europea	https://universidadeuropea.com/grados-universitarios/?locations_filter=1%2C9	Privada	50	0	Show	Scrape grados	Scrape másteres	Borrar grados	Borrar másteres
7	Comunidad de Madrid	Universidad Camilo José Cela	https://www.ucjc.edu/estudios-universitarios/grados/oficiales/	Privada	27	0	Show	Scrape grados	Scrape másteres	Borrar grados	Borrar másteres

Figura 5.10: *Vista universidades, con la tabla de las universidades.*

También es posible hacer *scraping* de todas las universidades de cada comunidad autónoma (en este caso solo disponible para Madrid, única comunidad autónoma implementada). Para ello tenemos que ir a la pestaña comunidades, y hacer click en el botón "Scrape" de la comunidad. Realizando esta acción se hará *scrape* de todos los másteres y grados de la comunidad autónoma (ver Figura 5.11).

Figura 5.11: *Botón "Scrape" de la vista Comunidades.*

Además es posible ir a cada universidad, donde se mostrará una pequeña tabla, con el número de grados, másteres y asignaturas asociadas. Y el listado de cada grado y máster, con sus atributos, además haciendo click en el botón "Show", podemos ver el grado o máster concreto y sus asignaturas (ver Figura 5.12).

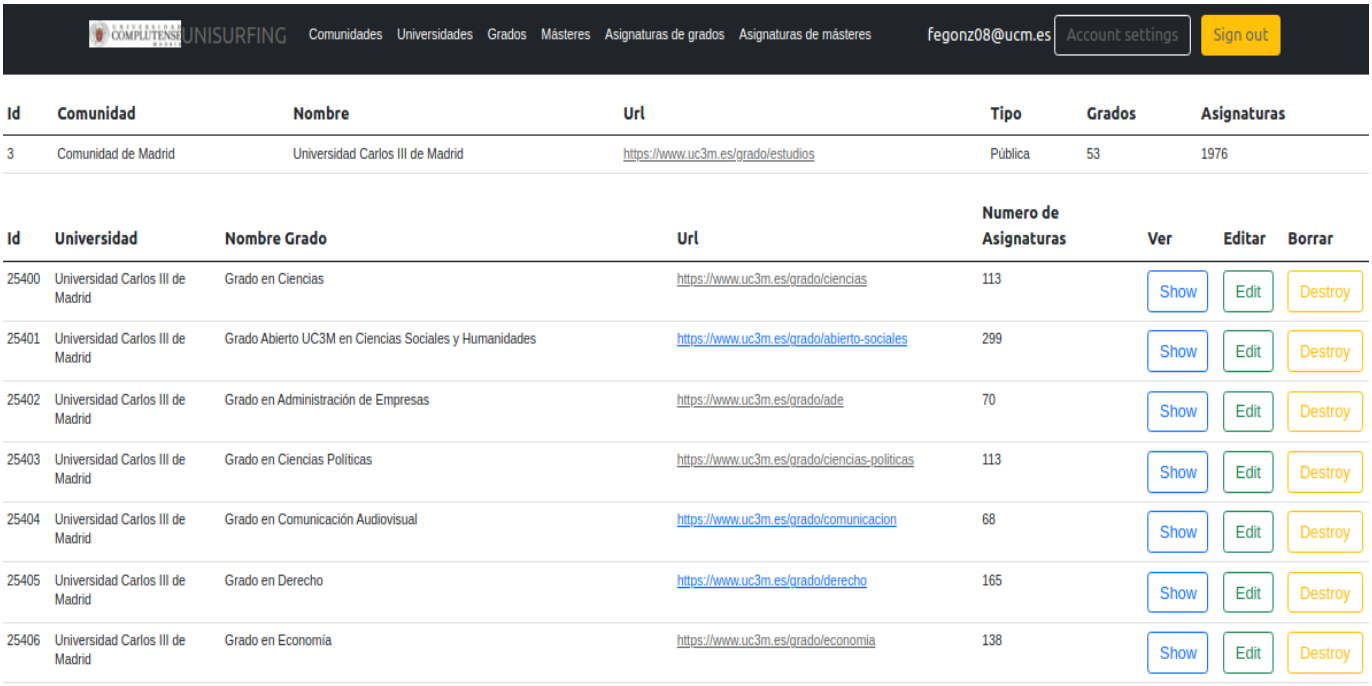


Figura 5.12: *Universidad Vista Concreta.*

5.5. Flujo de uso y resultados de ejecución (rendimiento)

En este capítulo veremos lo que serían los caminos para navegar en la aplicación y representaremos lo que podría ser varios ejemplos de uso. Apoyándonos en capturas de pantalla, iremos explicando como interactuar con la aplicación, para ver todas las funcionalidades implementadas en la aplicación.

Lo primero que se tendría que hacer en la aplicación es tener por defecto uno o varios usuarios con el rol administrador introducidos directamente en la base de datos. Lo segundo sería hacer el *scraping* o raspado web de todas las universidades y ya tendríamos el entorno preparado. Una vez efectuado el raspado web nos saldremos de la sesión de administrador para dejar paso a los usuarios. Nos registraremos con email y contraseña para iniciaremos sesión. En este punto podemos hacer una búsqueda por nombre de asignatura tanto de grados como de másteres.

Un posible flujo de uso como usuario no registrado, sería el siguiente:

Primero elegimos, si queremos buscar un grado o un máster concreto en la vista principal (ver Figura 5.13).

Figura 5.13: *Vista principal búsquedas.*

En este caso, hemos elegido buscar un grado. Este grado tiene que tener alguna asignatura que contenga la palabra "algebra" (sin tilde), y se ha podido estudiar en una universidad pública o privada indistintamente.

Las búsquedas de másteres, son exactamente igual, con el mismo formato. Así que ahora vamos a simular un flujo de uso, siendo administrador, en este caso con el usuario con "id=1". Primero vamos a iniciar sesión, en la vista login (ver Figura 5.14).

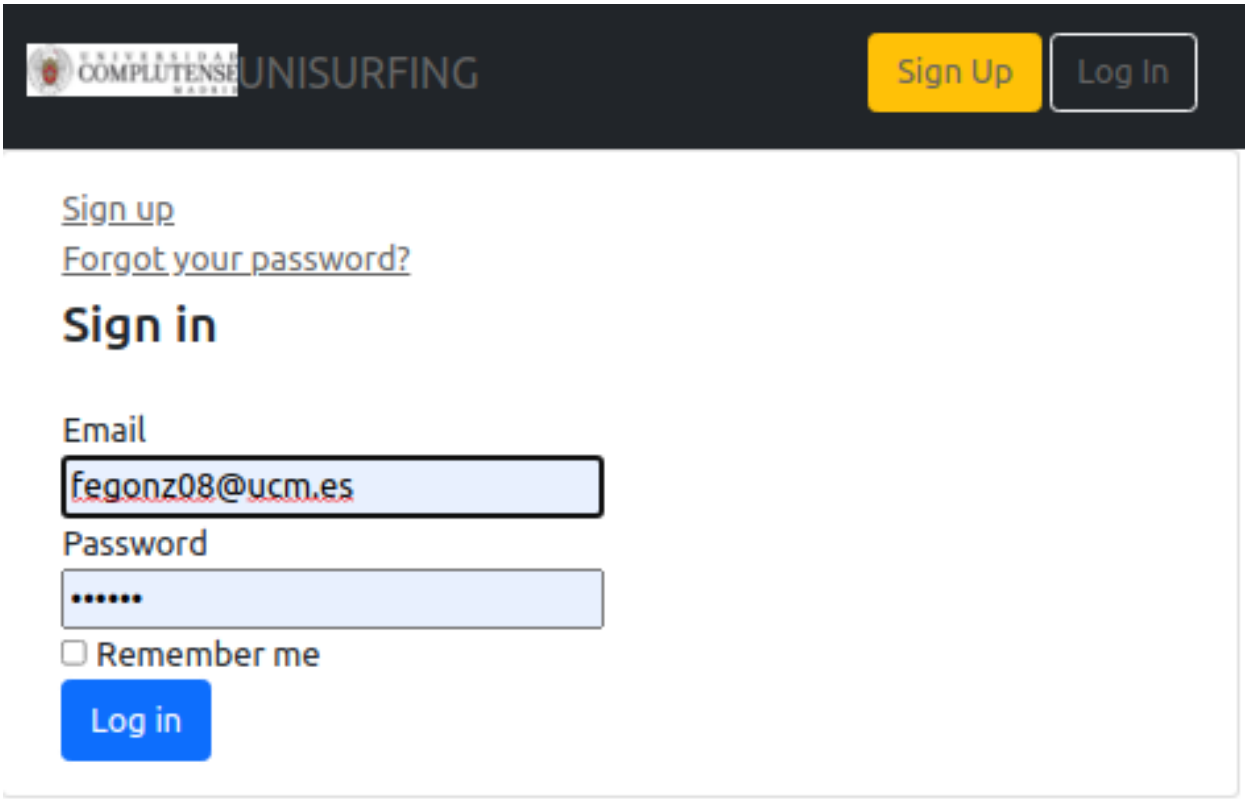


Figura 5.14: *Vista del login.*

Una vez introducidos los datos, se procesará el formulario de inicio de sesión. A continuación nos saldrá un mensaje en verde diciéndonos si hemos tenido éxito al realizar el inicio de sesión (ver Figura 5.15). Si por el contrario introducimos mal el email o la contraseña, saldrá un mensaje de error en fondo rojo (ver Figura 5.16).

Figura 5.15: *Inicio de sesión correcto.*

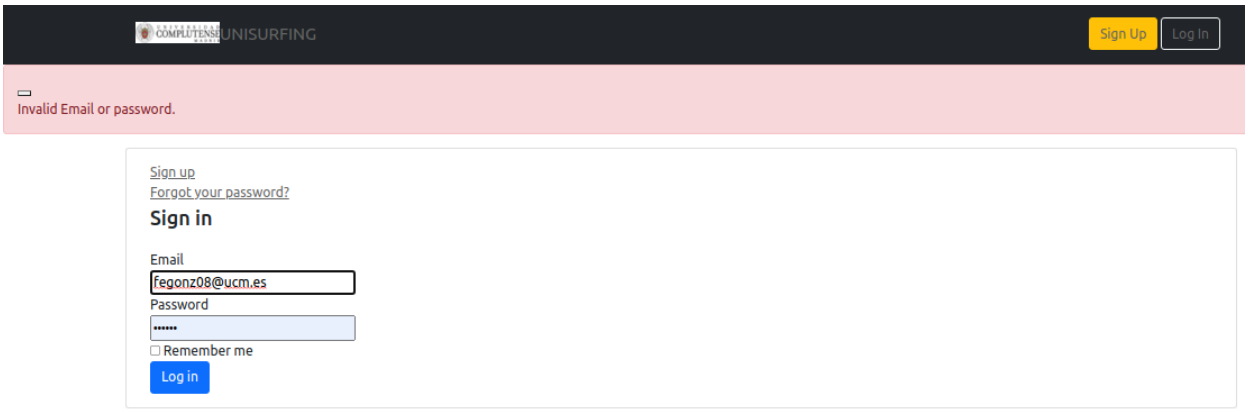


Figura 5.16: *Inicio de sesión incorrecto.*

Si hemos iniciado sesión de forma correcta, al ser un usuario con rol "admin", tendremos en el navegador diferentes pestañas, que dan acceso a diferentes funcionalidades, explicadas en el punto más arriba. Ahora haremos click en comunidades, a continuación nos saldrá una tabla con las comunidades autónomas introducidas en la aplicación. Si es la primera vez que iniciamos sesión como administradores, los contadores de másteres y grados, tendrán valor "0". Tenemos dos posibles acciones a realizar:

- Acción *Scrape*. Al hacer click en el botón "Scrape", se procederá hacer *scraping* de todas las universidades de la comunidad autónoma (ver Figuras 5.17 y 5.18).
- Acción Borrar. Al hacer click en el botón "Borrar grados y másteres", se procederá a borrar todos los grados y másteres de todas las universidades de la comunidad autónoma.

Si al realizar alguna de las dos acciones anteriores (*Scrape* o Borrar), se verá una barra azul de progreso justo debajo de la barra de navegación en el navegador.

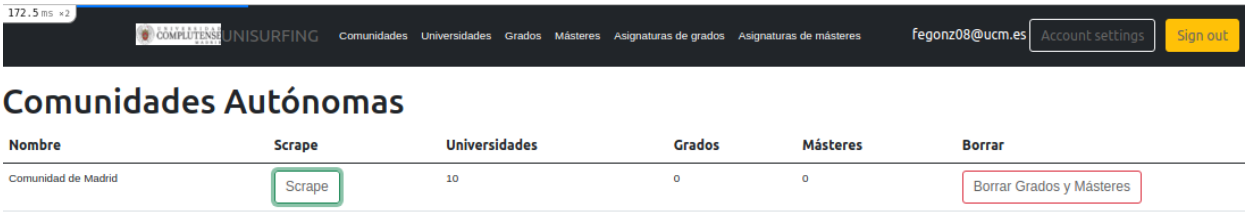


Figura 5.17: *Vista barra azul de progreso.*

Figura 5.18: *Vista después de haber hecho scrape en la Comunidad de Madrid.*

Además, podemos hacer *scrape* sobre grados o los másteres concretos de una universidad. Para ello tenemos que ir a la pestaña universidades, para hacer *scrape* a los grados de la universidad en concreto, tenemos que hacer click en el botón "Scrape grados". Si queremos hacer *scrape* sobre los másteres, haremos click en el botón "Scrape másteres". También podremos borrar todos los grados y másteres asociados a una universidad concreta, haciendo click en el botón 'Borrar grados' y "Borrar másteres".

Ahora hablaremos del rendimiento de la aplicación. En cuanto al raspado web son operaciones bastante costosas en cuanto a tiempo de ejecución. Ya que después de realizar cada

petición *CURL*, hay que analizar las etiquetas HTML para extraer la información deseada. Al hacer *scrape* sobre todas las universidades introducidas de la Comunidad de Madrid, analizando másteres y grados, el resultado tarda aproximadamente 53 minutos analizado con la herramienta para desarrolladores del navegador Google Chrome [4] (ver Figura 5.19). Otro ejemplo individual al realizar *scrape* concreto sobre los grados de la Universidad Carlos III de Madrid, tardando aproximadamente 7,5 minutos (ver Figura 5.20).

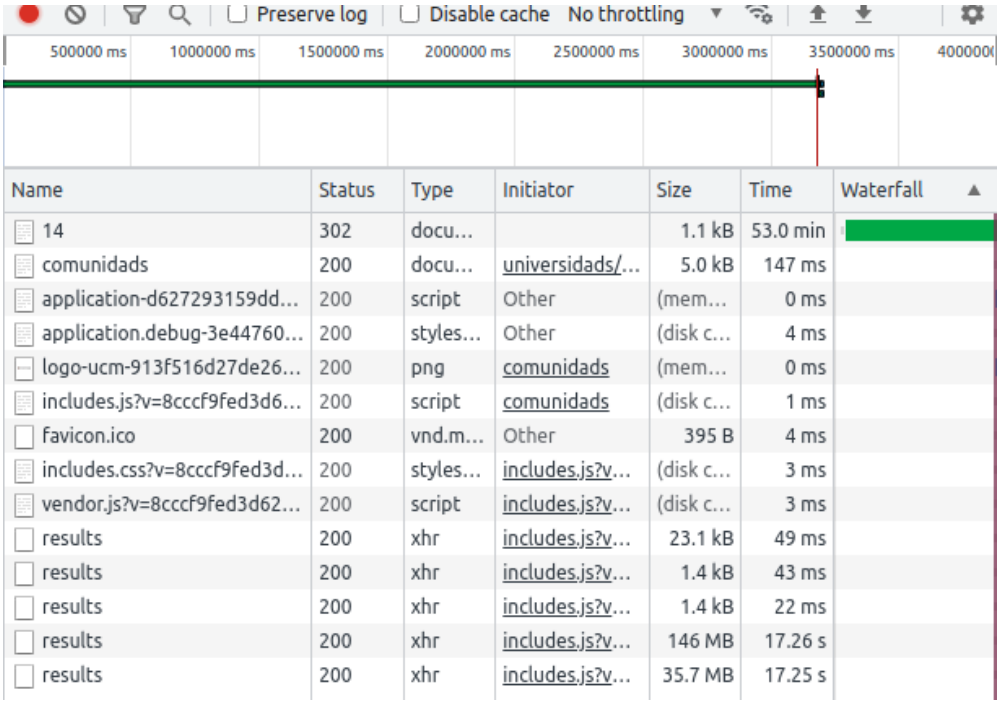


Figura 5.19: *Tiempo de ejecución de realizar scrape de toda la Comunidad de Madrid.*

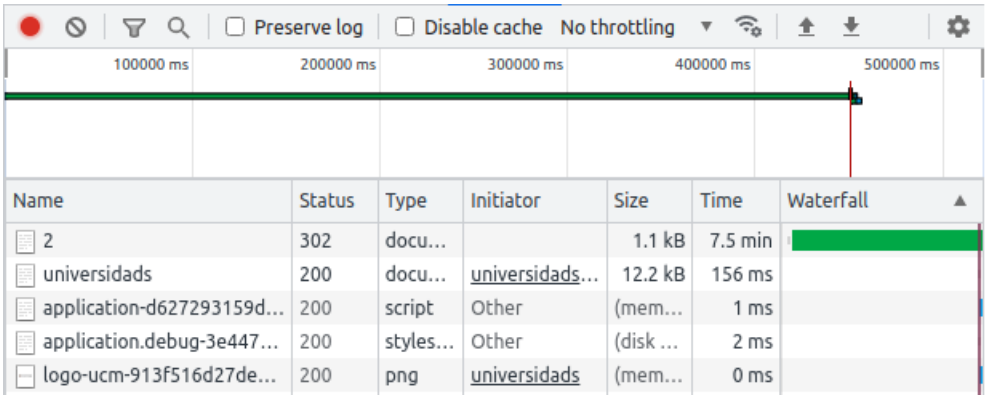


Figura 5.20: *Tiempo de ejecución de realizar scrape de los grados de la Universidad Carlos III de Madrid.*

En cuanto al tiempo de ejecución, hemos analizado la búsqueda de un grado, dado el nombre de una asignatura, ya que con una base de datos con más de 39.000 asignaturas y más de 600 grados. La búsqueda de un grado que contenga la asignatura "algebra" (sin tilde), tarda 229 ms (ver Figura 5.21).

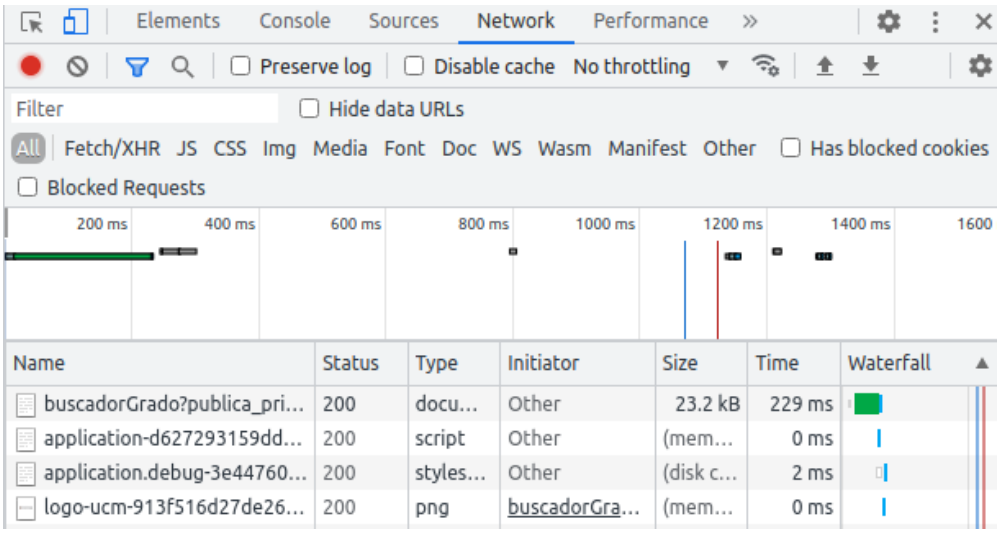


Figura 5.21: *Tiempo de ejecución búsqueda de grados con la asignatura "algebra" (sin tilde).*

Capítulo 6

Conclusiones y Trabajo Futuro

6.1. Conclusiones

A título personal hemos quedado bastante satisfechos con el resultado de la aplicación *UniSurfing*. Si bien han surgido problemas en el proceso de instalación, por la documentación desactualizada y por la escasa unificación del formato de datos, el resultado es bastante alentador. En cuanto a los objetivos a cumplir, hemos logrado realizar casi todas las metas propuestas en la especificación del TFG.

El primer objetivo era crear una aplicación web funcional en la cual poder extraer información de diferentes páginas oficiales, algo que se ha implementado escrupulosamente. También se mostró predisposición a la posibilidad de tener la información actualizada, hecho que se ha implementado como funcionalidad de forma efectiva dentro de la aplicación.

Otro objetivo principal consistía en desarrollar una aplicación con la cual almacenar información básica de los datos extraídos y poder ponerlos a disposición de los usuarios de esta, algo que claramente se consiguió, con la posibilidad de listar los grados y sus asignaturas. También ofrece la funcionalidad de hacer búsquedas por nombre y créditos de la asignatura, algo que hemos logrado con la posibilidad de una búsqueda con varios filtros. También se pedía recoger datos del máximo número de universidades y hemos almacenado datos de diez universidades. También se logró cumplir la implementación de másteres aunque en menor medida.

Por último, se pedía implementar una interfaz clara y sencilla en la que poder navegar por sus elementos y hacer búsquedas por filtros. Esta meta también se cumplió, pudiendo acceder a listas de los diferentes los diferentes elementos y pudiendo ejecutar búsquedas múltiples tanto de grados como de másteres, estas últimas a través del motor de búsqueda Sphinx.

El resultado es *UniSurfing*, una aplicación para la extracción de datos de grados y másteres de universidades de la Comunidad de Madrid. Con *UniSurfing* podemos extraer información y hacer búsquedas extraordinariamente veloces a través de filtros por nombre de la asignatura y por universidad.

6.2. Trabajo futuro

El ámbito del proyecto actual es únicamente la Comunidad de Madrid, pero es una aplicación fácilmente escalable al resto de comunidades autónomas de España. Hemos creado una base de datos pensando en la escalabilidad del proyecto y se podría añadir fácilmente más universidades para poder abarcar el espectro completo de ofertas del Estado español.

Además de tener estudios de grado y másteres, hay universidades que ofertan otros tipos de titulaciones como ciclos superiores, cursos formativos, cursos para convalidación de créditos, etc. El diseño de la base de datos también está pensado para poder añadir más tipos de oferta formativa.

En cuanto a información adicional para añadir a grados, másteres y sus funciones, las posibilidades son bastante amplias. Se podría implementar una descripción con palabras clave para aumentar el abanico de búsquedas. También se podría desarrollar reseñas de otros estudiantes que las hayan cursado, añadir el precio por crédito y la nota de corte y señalar los posibles horarios e itinerarios posibles.

En cuanto a las características de las universidades se puede añadir otra entidad intermedia, las facultades. Estas englobarían varios grados y másteres y en donde se puede remarcar tanto la localización como los servicios ofrecidos en la institución, junto a la información de

contacto. Los servicios que ofrecen las facultades pueden marcar la diferencia entre elegir una u otra.

También se podrían implementar vistas y funciones que ayuden en la comparación de grados, con diferentes filtros para facilitar la elección de uno frente a otros. En definitiva, el potencial de esta aplicación es enorme, gracias a su escalabilidad se pueden implementar un sinfín de posibilidades y con una búsqueda de elementos bastante satisfactoria debido a su velocidad. Disponiendo los recursos necesarios para su desarrollo y mantenimiento podría llegar a convertirse incluso en un referente del mercado a medio-largo plazo.

Bibliografía

[1] Jonas Nicklas Thomas Walpole. Documentación gema capybara. *RubyDoc.info*, 2018.

[2] Fernando Salcido. ¿qué es active record y cómo funciona? *Medium*, 2020.

[3] Bernardo Gómez. Configurando ruby on rails 5 con mysql en ubuntu 14.04 (con accesos remotos a bases de datos). *Medium*, 2016.

[4] Usar herramientas de desarrollo de chrome para revisar etiquetas.

[5] André Arco. Documentación gema jquery-rails_gem. *RubyDoc.info*, 2010.

[6] Rafael França José Valim, Carlos Antonio. Documentación gema simple_form. *RubyDoc.info*, 2019.

[7] Carlos Antonio José Valim. Documentación gema devise_gem. *Github*, 2019.

[8] Victor Afanasev. Documentación gema kimurai. *RubyDoc.info*, 2018.

[9] Pat Allan. Thinking sphinx. *https://freelancing-gods.com/*, 2020.

[10] Varvet AB Jonas Nicklas. Documentación gema pundit_gem. *RubyDoc.info*, 2018.

[11] Wellington Cordeiro Florent Monbillard. Documentación gema rolify_gem. *RubyDoc.info*, 2015.

[12] James Healy. Documentación gema pdf-reader_gem. *RubyDoc.info*, 2009.

[13] Flavor Jones. Parsing an html5 document. *https://nokogiri.org/*, 2017.

[14] Luke Duncan. How to scrape a pdf for keywords using ruby. *Medium*, 2020.

[15] Componentes de bootstrap. *https://getbootstrap.com.*

[16] Javier Vázquez y Daniel Martínez. *Ruby On Rails, Aprende a crear aplicaciones web desde cero*. 2020.

[17] José Quijado. ¿qué es curl? 2019.

[18] Jesús Martínez. Ruby on rails 5 curso completo desde 0 a experto (2019). *Udemy*, 2018.

Apéndice A

Introduction

A.1. Motivation

The increasing offer of university degrees and master's degrees makes the choice of career increasingly difficult and chaotic. From personal experience, we know that seeking a university degree, together with the faculty where it is taught, is not an easy task. For this reason, we have developed a centralized information system where decisions can be made in the most precise, comfortable and visual way possible. This system will save many indecisive students time and annoyance.

The students choose university based on different filters: by the different job opportunities, by the subjects to be studied, by the price of the degree, by the location of the faculty or by reviews from other students, among others. We will modestly try to cover some of these search filters and thereby facilitate this important decision making as much as possible.

Motivated by the discovery of a search engine called Sphinx, we wanted to develop a search engine for degrees and masters. Thanks to its potential based on indexing, it allows an instant search of large amounts of information, as we have seen in a search engine for official state bulletins, Zahoribo[1].

[1] https://zahoribo.com/

A.2. Objectives

Our main goal is to develop a web application to centralize information on university majors and degrees and thus be able to implement different types of search. To this end, we will need to extract the information directly from the official websites of the respective universities. In addition, the possibility of having updated information should be added to ensure the correctness of the information.

In the first functional version, an attempt has been made to gather the basic information of each degree together with its subjects, with the characteristic of searching by subject name and by university. Another objective will be to cover the largest possible number of careers offered in the Community of Madrid, with the possibility of developing the search for university master's degrees.

Another objective is the creation of a clear and simple interface with which to navigate through the different options and to be able to search with different filters. These searches should preferably be executed with the Sphinx search engine, as its excellent speed has been proven.

The achievement of the aforementioned general objectives are resolved in this report, addressing a series of specific objectives, which are listed below:

- Effective extraction of the greatest number of degrees possible.
- Data processing for structured storage.
- Setting up an efficient search using the *Sphinx* search engine.
- Development of an orderly interface in which to arrange the information.
- Union of the extraction and processing of data in the application.
- Implementation of different searches from the web interface.

A.3. Workplan

After having exposed the main objectives of this work, the planning of its development is determined in the time frame established for its achievement. To do this, a Gantt chart has been designed (see Figure A.1) and the duration and tasks to be completed for proper operation have been visually defined.

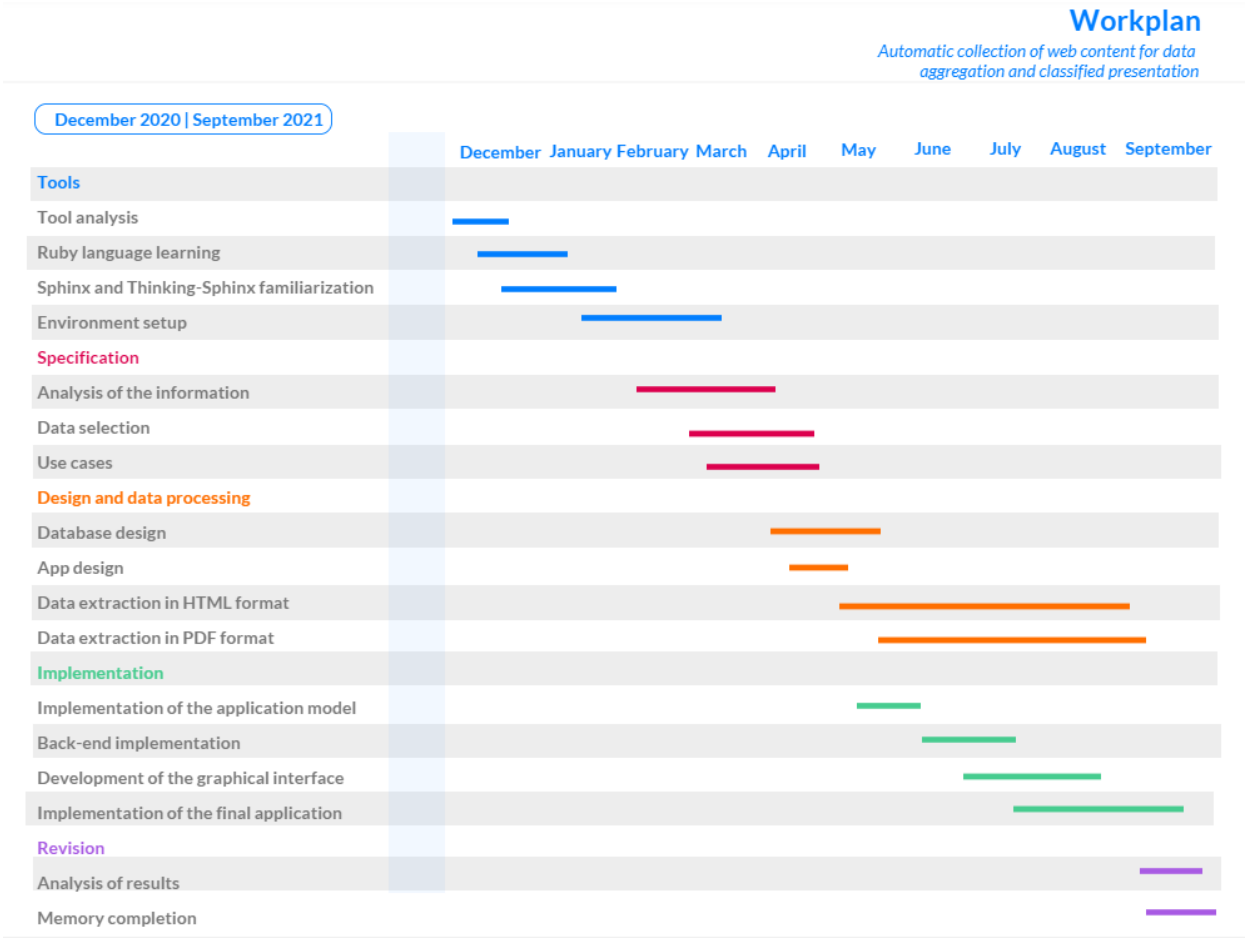


Figura A.1: *Workplan.*

A.4. Memory organization

This report details an initial summary, a summary in English, six chapters, the bibliography, and four appendices. This section will briefly and orderly describe the contents of

each chapter:

- **Chapter 1. Introduction**: The reasons that motivated the creation of this initiative will be described. The initial goals expected to be met will be displayed. Tasks will also be described in a time frame, using a Gantt chart.

- **Chapter 2. State of the art**: Exhibition of the different tools implemented and methodologies used during the development process, both in the software and hardware fields. A comparison will be made of the technologies used with other alternatives available in the market for the development of the application, together with some examples of real use.

- **Chapter 3. Application specification**: Detailed approach to the project, planning and prior preparation. List of the universities chosen to extract the desired content. Presentation of the chosen information together with its characteristics. The chosen data will also be analyzed along with its corresponding format.

- **Chapter 4. Design for data extraction and treatment**: Schematization and presentation of the data together with explanatory diagrams. The methodologies and tools used for the collection and transformation of information and its introduction into a database will be described in detail. At the end the data extraction algorithms will be explained.

- **Chapter 5. Implementation of the application**: Application design both front-end and back-end. In-depth explanation of the framework used and the model chosen. The final views and use cases will be exposed through different images. The possible search options will also be defined along with their algorithms and necessary settings for content updating. The performance of the application will also be analyzed, using developer tools.

- **Chapter 6. Conclusions and future work**: Summary of the project and the usefulness of its solutions. Efficacy of the tool before the different chosen universities and its actual use. Future functionalities of the application that could not be implemented due to lack of time. Overall personal assessment of the project.

Apéndice B

Conclusions and Future Work Lines

B.1. Conclusions

On a personal note, we have been quite satisfied with the result of the *UniSurfing* application. While problems have arisen in the installation process, due to outdated documentation, and poor data format unification, the result is undeniably encouraging.

As for the objectives to be met, we have all met. The first objective was to create a functional web application in which to extract information from different official pages, something that has been scrupulously implemented. There was also a predisposition to the possibility of having updated information, a fact that has been implemented as a functionality effectively within the application.

Another main objective was to develop an application in which to show basic information of the extracted data and to be able to make it available to its users, something that was clearly achieved, with the possibility of listing the degrees and their subjects. It also offers the functionality to search by name and credits of the subject, something that we have achieved with the possibility of a search with several filters. It was also requested to collect data from the maximum number of universities and we have stored data from 9 universities. The implementation of master's degrees was also achieved, although to a lesser extent.

Finally, it was asked to implement a clear and simple interface in which to be able to navigate through its elements and search by filters. This goal was also met, being able to

access lists of the different elements and being able to execute multiple searches for both degrees and masters, the latter through the Sphinx search engine.

The result is *UniSurfing*, an application for extracting data from bachelor's and master's degrees from universities in the Community of Madrid. With *UniSurfing* we can extract information and do extraordinarily fast searches through filters by subject name and by university.

B.2. Future work lines

The scope of the current project is only the Community of Madrid, but it is an easily scalable application to the rest of the autonomous communities of Spain. We have created a database thinking about the scalability of the project and more universities could easily be added to cover the full spectrum of offers in the Spanish State.

In addition to having undergraduate and master's degrees, there are universities that offer other types of degrees such as higher cycles, training courses, courses for credit validation, etc. The design of the database is also thought to be able to add more types of training offer.

As for additional information to add to degrees, masters and their functions, the possibilities are quite wide. A description with keywords could be implemented to increase the range of searches. You could also develop reviews of other students who have taken them, add the price per credit and the cut-off mark and point out the possible schedules and possible itineraries.

Regarding the characteristics of the universities, another intermediate entity can be added, the faculties. These would include several degrees and masters and where you can highlight both the location and the services offered in the institution, along with the contact information. The services offered by colleges can make the difference between choosing one or the other.

You can also implement views and functions that help in the comparison of degrees,

with different filters to facilitate the choice of one over others. In short, the potential of this application is enormous, thanks to its scalability, endless possibilities can be implemented and with a quite satisfactory search for elements due to its speed. Throwing in hours of work can even become a benchmark in the market in the medium-long term.

Apéndice C

Reparto de Trabajo

En éste apéndice se hablará de como nos hemos repartido el trabajo y que metodologías y dinámicas hemos practicado para llevar un buen ritmo de trabajo.

El reparto de trabajo se ha ido haciendo gracias a una metodología ágil, esto quiere decir que aun teniendo la mayoría de tareas planeadas y prefijadas, han ido surgiendo imprevistos que hemos podido afrontar gracias a una flexibilidad de modificación de las tareas. Estas se repartían una a una y por consenso mutuo, pero esto no impedía que ayudáramos en la tarea del compañero e incluso en algunos casos intercambiar tareas por diferentes motivos.

C.1. González Barrado, Fernando

Éste fue el primer miembro del equipo y se introdujo en agosto. Acudió a la primera reunión donde se discutió el tema a implementar, se valoraron diferentes alternativas, como un bucador de torneos de pádel, un buscador de noticias de periódicos, además del tema finalmente elegido el buscador de universidades. También se propuso el motor de búsqueda (Sphinx) y el lenguaje a utilizar (Ruby). Después tuvieron lugar varias reuniones en donde se especificaban las posibles funcionalidades, el alcance y las tecnologías a usar.

En la primera fase Fernando se dedicó a aprender Ruby como lenguaje de programación y Ruby on Rails como framework. Para ello se compró un curso de Ruby en la plataforma Udemy [18]. Después le tocó buscar documentación del motor de búsqueda Sphinx para su posterior implementación. Logró instalar y configurar Thinking-Sphinx para su uso en Ruby

on Rails. Para ello, realizó la parte técnica más tediosa del proyecto, instalar MySQL de forma que fuera compatible con el framework elegido, realizando hasta un total de diecisiete instalaciones diferentes, hasta dar con la correcta instalación de las librerías necesarias.

En la segunda fase este miembro se encargó del análisis de la información a escoger de las páginas web, en concreto describir el formato que tenían y los datos que mostraban. También se dedicó a la búsqueda de información sobre la extracción y tratamiento de datos de páginas web en formato HTML con la gema Kimurai.

En la tercera fase, diseño y tratamiento de datos, Fernando se dedicó al diseño de la base de datos, para poder hacer un proyecto escalable al resto del territorio español con el tiempo. Otra tarea importante fue la extracción y tratamiento de datos sobre grados en formato HTML, tarea que se alargó en el tiempo porque cada universidad que había que añadir, proporciona datos diferentes sobre sus grados, y usa etiquetas HTML diferentes, como hemos explicado en la Sección 4.3.1.

En la cuarta fase, tuvo las tareas de una vez implementada la estructura final, desarrollar la aplicación en Ruby on Rails tanto el back-end como el front-end y a unir la parte de extracción de datos a la aplicación. Para el front-end, optó por usar el framework Bootstrap, cuya instalación en el framework Ruby On Rails, también tuvo cierta complejidad, ya que para hacerlo funcionar hay que instalar otras librerías de las que depende la librería de Bootstrap.

En la quinta fase, también realizó la extracción de datos de PDF de la Universidad Politécnica de Madrid, para ello primero instaló una gema "Kristin", que convierte un PDF a formato HTML, pero no lo hacía correctamente. Después lo realizó con la gema "pdf-reader ", para la extracción datos se desarrollaron los algoritmos descritos anteriormente.

En la sexta fase, realizó la extracción de datos de másteres de diferentes universidades. Como cada máster concreto de cada, tiene una duración determinada y hay algunos impartidos por diferentes universidades, esto provocaba un algoritmo de extracción diferente por cada máster.

En la fase final, se dedicó a analizar los resultados para comprobar su eficiencia en cuanto a velocidad. A parte se dedicó a la finalización de la memoria.

C.2. González Montero, Daniel

Este miembro se introdujo en el equipo a partir de la segunda reunión. Hubo varias reuniones iniciales en donde se discutieron las especificaciones del TFG. Estas especificaciones iban desde las tecnologías propuestas (Sphinx y Ruby on Rails) hasta la funcionalidad de la aplicación.

En la primera fase este miembro tuvo que aprender el lenguaje Ruby y el uso del framework Ruby on Rails. Para ello también se compró un curso de Ruby en la plataforma Udemy [18]. Después estuvo buscando documentación del motor de búsqueda Sphinx para su posterior comprensión de uso. También comentar que consiguió instalar y configurar la base de datos MySQL para su posterior implementación en Ruby on Rails. Para finalizar desarrolló una posible estructura para la aplicación en Ruby on Rails, la cual sirvió de modelo para la futura implementación.

En la segunda fase Daniel se hizo cargo de la búsqueda sobre posibles formas de extraer información dentro del framework Ruby on Rails, en concreto sobre la extracción y tratamiento de PDF con la gema "PDF_Reader". También se especificaron la primera versión sobre los casos de uso, ya que después se iteraría sobre esta versión.

En la tercera fase, diseño y tratamiento de datos, este miembro se dedicó al diseño de la base de datos para que esta tuviera una estructura relacional. Otra tarea a destacar fue la extracción de datos en formato PDF tarea que se alargó hasta casi el final del proyecto. Esta tarea de extracción de información fue sumamente tediosa y compleja debido a que había que diseñar los algoritmos de cero. Además la forma de revisar esos algoritmos era implementado el código y comprobándolo el resultado manualmente.

En la cuarta fase, la fase de implementación, Daniel se encargó de desarrollar la estructura final para albergar la aplicación configurando la base de datos. También se dedicó a continuar la extracción y tratamiento de datos de PDF y a su vez implementar esta parte dentro de la aplicación. Otra tarea fue comprobar el buen funcionamiento de la aplicación y ayudar en su desarrollo.

En la quinta fase terminó el desarrollo del tratamiento de datos en PDF y su corrección dentro de la aplicación. A pesar de la dificultad se logró un resultado satisfactorio. También la revisión de la aplicación puliendo detalles del front-end, como la implementación y eliminación de botones y sus funcionalidades.

En la sexta fase, Daniel se dedicó al desarrollo de la memoria del TFG. Esta tenía que seguir ciertas directrices que se consensuaron con los tutores a través de reuniones online. Esta memoria se desarrolló en una plataforma llamada Overleaf dedicada a la creación de documentos PDF. Esta plataforma tiene diferentes comandos y opciones en formato pseudo código que facilitan tareas como la creación de los índices, implementación de imágenes y creación de tablas entre otras.

En la fase final, su tarea fue analizar los resultados para comprobar la eficacia de todas las funcionalidades. También se dedicó a la finalización de la memoria y su revisión.