ESCUELA SUPERIOR DE INGENIERÍA

Ingeniería Informática

Sitic: framework para generar páginas web estáticas

José Jesús Marente Florín

Cádiz, 5 de septiembre de 2017

ESCUELA SUPERIOR DE INGENIERÍA

Ingeniería Informática

Sitic: framework para generar páginas web estáticas

DEPARTAMENTO: Ingeniería Informática.
DTOR. DEL PROYECTO: José Miguel Mota Macías.
AUTOR DEL PROYECTO: José Jesús Marente Florín.

Cádiz, 5 de septiembre de 2017

Fdo.: José Jesús Marente Florín

Este documento se halla bajo la licencia FDL. Según estipula la licencia, se muestra aquí el aviso de copyright. Se ha usado la versión inglesa de la licencia, al ser la única reconocida oficialmente por la FSF.

Agradecimientos

Me gustaría dar las gracias a todos esos compañeros que he conocido durante la carrera y me han ayudado a lo largo la misma y desarrollo de este proyecto. Así como a mi familia, pareja y amigos por todo el apoyo que me han dado. También querría dar las gracias al tutor del proyecto José Miguel Mota por el apoyo, supervisión y consejos que me ha dado.

Resumen

Sitic es un *framework* generador de sitios web estáticos. El usuario escribirá los contenidos de la web en ficheros de textos plano y la herramienta generará una web completamente estática a partir de estos ficheros, transformándolos en html.

Entre otras características, la herramienta usa un sistema de plantillas que permite al usuario personalizar la forma en la que se mostrarán todos los contenidos. Así mismo, permitirá al usuario definir los atributos básicos de cada página y ofrecerá otras posibilidades como generación de menús o generación de RSS.

Estará totalmente desarrollado usando el lenguaje de programación Python y usará como base el sistema de plantillas Jinja [8].

Palabras clave: Internet, Web, HTML, Python, Jinja.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación

La idea de desarrollar este proyecto surge a raíz de usar otras herramientas con un objetivo parecido, pero que no llegaba a satisfacer todas las funcionalidades que necesitaba, viéndote obligado a suplirlas con otras herramientas que no eran totalmente compatibles con las originales.

En noviembre de 2015 forme parte en el desarrollo de una página web totalmente estática con información bastante importante a la que accederían personas de distintos idiomas. Para el desarrollo de la web mencionada, se usó Hugo [7], uno de los generadores estáticos más conocidos. Pero a pesar de ser uno de los más usados, la herramienta carecía de funcionalidades que considerábamos totalmente imprescindibles en el desarrollo de una plataforma tan importante.

De ahí nace la motivación para intentar desarrollar una herramienta que aunque siga los mismos principios, añada funcionalidades de las que las herramientas actuales carecen y no tiene pensado añadir en un futuro próximo.

También he de añadir que tras conocer abiertamente el mundo del Software libre, gracias a la importancia que se le presta en la Universidad de Cádiz. Se decidió que el proyecto fuera software libre bajo licencia GPL. Y así cualquier persona interesada en el proyecto y en el software libre en general, pudiera usar los recursos del proyecto o colaborar libremente.

1.2. Objetivos

A la hora de definir los objetivos de un sistema, podemos agruparlos en dos tipos diferentes: funcionales y transversales. Los primeros se refieren a qué debe hacer la herramienta que vamos a desarrollar, e inciden directamente en la experiencia del usuario y de potenciales desarrolladores.

Por otro lado, los objetivos transversales son aquellos invisibles al usuario final, pero que de forma inherente actúan sobre el resultado final de la aplicación y sobre la experiencia de desarrollo de la misma.

1.2.1. Funcionales

- Crear una herramienta que permita a los usuarios crear una web completamente estática con todas las funcionalidades que una web normal y dinámica tienen.
- Dar la posibilidad a los usuarios de personalizar en la medida de todo lo posible la configuración inicial de la herramienta de forma que puedan adaptar lo máximo posible a sus necesidades.
- Dar la posibilidad que cualquier usuario medio Linux pueda crear una web desde cero sin tener ningún conocimiento de programación o de servidores web.

1.2.2. Transversales

- Aplicar mis conocimientos sobre el desarrollo web en general.
- Adquirir soltura en el uso del lenguaje de programación Python en herramientas de terminal.
- Utilizar un enfoque de análisis, diseño y codificación orientado a objetos, de una forma lo más clara y modular posible, para permitir ampliaciones y modificaciones sobre la aplicación por terceras personas.
- Hacer uso de herramientas básicas en el desarrollo de software, como son los sistemas de control de versiones para llevar un control realista del desarrollo del software, así como hacer de las veces de sistema de copias de seguridad.

1.3. Estructura del documento

Este documento está compuesto por las siguientes partes:

- **Introducción**: pequeña descripción del proyecto, así como los objetivos y estructura del documento.
- **Descripción general**: descripción más amplia sobre el proyecto, así como todas las características relevantes que tendrá.
- **Planificación**: exposición de la planificación del proyecto y las distintas etapas que esta compuesto el mismo.

- **Análisis**: fase de análisis del sistema, empleando la metodología seleccionada. Se definirán los requisitos funcionales del sistema, diagramas de caso de uso, diagramas de secuencia y contrato de las operaciones.
- **Diseño**: realización del diseño del sistema, diagramas de secuencia y clases aplicadas al diseño.
- **Implementación**: aspectos más relevantes durante la implementación del proyecto. Y problemas que han aparecido durante el desarrollo.
- **Pruebas**: pruebas realizadas a la aplicación, con el fin de comprobar su correcto funcionamiento y cumplimiento de las expectativas.
- **Conclusiones**: conclusiones obtenidas tras el desarrollo de la aplicación.
- **Apéndices**:
  - **Manual de instalación**: manual para la correcta instalación del proyecto en el sistema.
  - **Manual de usuario**: manual de usuario para el correcto uso de la aplicación.
- **Licencia GPL**: texto completo sobre la licencia GPL, por la cual se rige el proyecto.
- **Bibliografía**: libros y referencias usadas durante el desarrollo del proyecto.

Capítulo 2

Descripción general del proyecto

2.1. Descripción

Sitic es un *framework* para hacer sitios web de uso general. Técnicamente hablando, Sitic es un generador de sitios web estáticos. A diferencia de otros sistemas que genera dinámicamente una página cada vez que un visitante la solicita, Sitic crea el sitio una única vez, cuando creas su contenido. Dado que los sitios web se ven con mucha más frecuencia de lo que se edita.

Los sitios construidos con Sitic son bastante más rápidos y seguros que un sitio generado de forma dinámica. Se pueden alojar en cualquier lugar, incluyendo GitHub Pages, Google Cloud Storage o Amazon S3 entre otros. Los sitios de Sitic se ejecutan sin depender de tiempos de ejecución costosos como Ruby, Python o PHP y sin dependencia de ninguna base de datos, en lo que se refiere al lado del servido, solo el tiempo que el navegador del usuario necesite para descargar la página visitada. En referencia al lado del cliente como puede ser la ejecución del Javascript que el usuario haya desarrollado, dependerá del navegador y equipo del usuario.

2.1.1. Diferencia con los generadores dinámicos

Los generadores de sitios web crean contenidos en ficheros HTML. La mayoría son "generadores dinámicos". Esto significa que el servidor HTTP (que es el programa que se ejecuta en su sitio web con el que el navegador del usuario habla) ejecuta el generador para crear un nuevo fichero HTML cada vez que un usuario desea ver una página.

Crear la página de forma dinámica significa que la máquina que aloja el servidor HTTP tiene que tener suficiente memoria y CPU para ejecutar el generador de forma eficaz durante todo el día. Si no, entonces el usuario tiene que esperar a que la página se genere.

Nadie quiere que los usuarios esperen más de lo necesario, por lo que los generadores de sitios dinámicos programaron sus sistemas para almacenar en caché los ficheros HTML. Cuando un fichero se almacena en caché, una copia se almacena temporalmente en el equipo. Es mucho más rápido que el servidor envíe esa copia la próxima vez que se solicite la página en lugar generarla desde cero.

Sitic intenta llevar el almacenamiento en caché un paso más allá. Todos los ficheros HTML se representan en su máquina. Puede revisar los ficheros antes de copiarlos en la máquina que aloja el servidor HTTP. Dado que los ficheros HTML no se generan dinámicamente, decimos que Sitic es un "generador estático".

No tener que ejecutar la generación de HTML cada vez que se recibe una petición tiene varias ventajas. Entre ellas, la más notable es el rendimiento, los servidores HTTP son muy buenos en el envío de ficheros. Tan bueno que puede servir eficazmente el mismo número de páginas con una fracción de memoria y CPU necesaria para un sitio dinámico.

Sitic tiene dos componentes para ayudarle a construir y probar su sitio web. El que probablemente usará más a menudo es el servidor HTTP incorporado. Cuando ejecuta el servidor, Sitic procesa todo su contenido en ficheros HTML y luego ejecuta un servidor HTTP en su máquina para que pueda ver cómo son las páginas.

El segundo componente se utiliza una vez que el sitio esté listo para ser publicado. Ejecutar Sitic sin ninguna acción reconstruirá su sitio web completo utilizando la configuración `base_url` del fichero de configuración de su sitio. Eso es necesario para que sus enlaces de página funcionen correctamente con la mayoría de las empresas de alojamiento.

2.1.2. Características principales

En términos técnicos, Sitic toma un directorio fuente de ficheros y plantillas y los usa como entrada para crear un sitio web completo.

Sitic cuenta con las siguientes características:

General

- Tiempos de generación rápidos
- Fácil instalación
- Posibilidad de alojar su sitio en cualquier lugar

Organización

- Organización sencilla
- Soporte para secciones

- URL personalizables
- Soporte para taxonomías configurables que incluyen categorías y etiquetas.
- Capacidad de clasificar el contenido como se desee
- Generación automática de tabla de contenidos
- Creación dinámica de menús
- Soporte de URLs legibles

Contenido

- Soporte nativo para contenido escrito en Markdown, Textile y Reestructured text
- Soporte internacionalización
- Soporte para especificar metadatos, usando TOML [19] y YAML [25] como formatos, en los contenidos
- Páginas completamente personalizables

Capítulo 3

Planificación

En este capítulo procederemos a detallar la planificación que ha seguido el proyecto. En particular, se describe las etapas por las que ha pasado el proyecto, la planificación temporal y la organización del personal involucrado.

En general, el desarrollo del proyecto se ha ajustado razonablemente bien al calendario inicialmente dispuesto en la planificación, con algunas etapas durando más de lo previsto pero otras terminando antes de lo planificado.

3.1. Fase inicial

La primera fase consistió en plantear la idea del proyecto, con la ayuda del tutor. Tras plantear varios enfoques, se decidió realizar este proyecto debido a las motivaciones escritas anteriormente.

También se pensó en qué lenguaje se desarrollaría el proyecto, así como las principales bibliotecas que se usarían durante la realización del mismo, priorizando siempre opciones libres. Finalmente se optó por utilizar el lenguaje Python por su documentación, comunidad y amplia biblioteca estándar.

3.2. Fase de análisis

Esta etapa está dividida principalmente en las dos partes siguiente:

- Especificación de los requisitos: estudio de los requisitos que deberá cumplir el proyecto.
- Recursos necesarios: recursos necesarios que deberemos usar durante el desarrollo del proyecto.

3.3. Fase de aprendizaje

Aunque ya había realizado otros proyectos en Python, decidí dedicarle tiempo a perfeccionar mis conocimientos sobre el lenguaje, así como de la librería estándar y las posibles bibliotecas que me serían útiles en el desarrollo.

Esta fase se caracterizó por intentar entender código ya escrito así como leer tutoriales y documentación oficial. Se podría dividir en estas etapas:

- **Perfeccionamiento del lenguaje Python**: investigar sobre el propio lenguaje, biblioteca estándar así como metodologías a utilizar.
- **Aprendizaje de otras bibliotecas**: investigar bibliotecas externas desarrolladas por la comunidad y que podrían ser útiles en el desarrollo.
- **Investigar sobre lenguajes de marcado**: dado que el usuario escribirá el contenido de la web en ficheros de texto plano y haciendo uso de lenguajes de marcado. Invertí tiempo en ver los más usados, para así poder darles soporte.

3.4. Fase de desarrollo

Tras la consecución de las etapas anteriores, se comenzó el desarrollo del proyecto. Esta etapa del desarrollo es la más extensa de todas, como es comprensible. Y me fue posible llevarla a cabo gracias a los prototipos que iba implementando conforme aprendía.

- **Generador básico**: Como primer paso se realizó un generador básico de ficheros a partir de ficheros de texto plano. Esto incluía la generación estructural de toda la web resultante.
- **Adaptar sistema de plantillas**: Otro de los aspectos más importantes fue adaptar el sistema de plantillas usado (Jinja) con el motor de generación de forma que los HTMLs generados fueran totalmente personalizables por los usuarios.
- **Mejor de output**: Al ser una herramienta que se ejecuta por la linea de comandos, se invirtió bastante tiempo en que el output ofrecido al usuario fuera realmente verbose y útil, dando información relevante.
- **Internacionalización**: Sistema de internacionalización fácil e intuitivo de usar, que permitirá al usuario escribir plantillas HTMLs una única vez para todos los idiomas configurados.
- **Ampliación de funcionalidades**: una vez que ya se había desarrollado toda las funcionalidades básicas, se dedicó tiempo a mejorar las ya existentes, así como a implementar nuevas.

3.5. Pruebas y correcciones

Una de las etapas más importantes del desarrollo de cualquier proyecto. Esta etapa se realizó en paralelo a la de desarrollo, ya que conforme se implementaron nuevas funcionalidades, se iban probando exhaustivamente bajo el contexto de las distintas situaciones que pudieran darse, hasta obtener el comportamiento esperado.

3.6. Redacción de la memoria

La redacción de la memoria se ha realizado conforme se iba avanzando en el desarrollo del proyecto. Pero tras la finalización de éste, se le ha dedicado más tiempo a su finalización, corrigiendo puntos que finalmente no se han adecuado al producto final.

3.7. Diagrama de Gantt

A continuación se muestra la planificación anteriormente comentada, en su correspondiente diagrama de Gantt:

3.8. Costes

El coste final del producto, sería la suma del coste de los recursos humanos, con los recursos materiales. El cálculo se realiza con los tiempos estimados.

Los recursos materiales los podemos dividir en dos:

- Recursos software: conjunto de herramientas propuestas a usar en el desarrollo. Todo lo utilizado es software libre, por lo tanto el coste de los recursos software es 0€.
- Recursos hardware: para el desarrollo de la herramienta se ha utilizado un portátil Asus con un precio de 720€.

Los recursos humanos necesarios para el desarrollo serían:

- Analista: Sueldo medio anual de un analista funcional es de 45.000€. Siendo la hora a 23€
- Desarrollador: Sueldo medio anual de un desarrollador web es de 35.000€. Siendo la hora a 18€.
- Encargado de Pruebas: Sueldo medio anual de un especialista encargado de realizar pruebas técnicas y funcionales es de 30.000€. Siendo la hora a 15€.

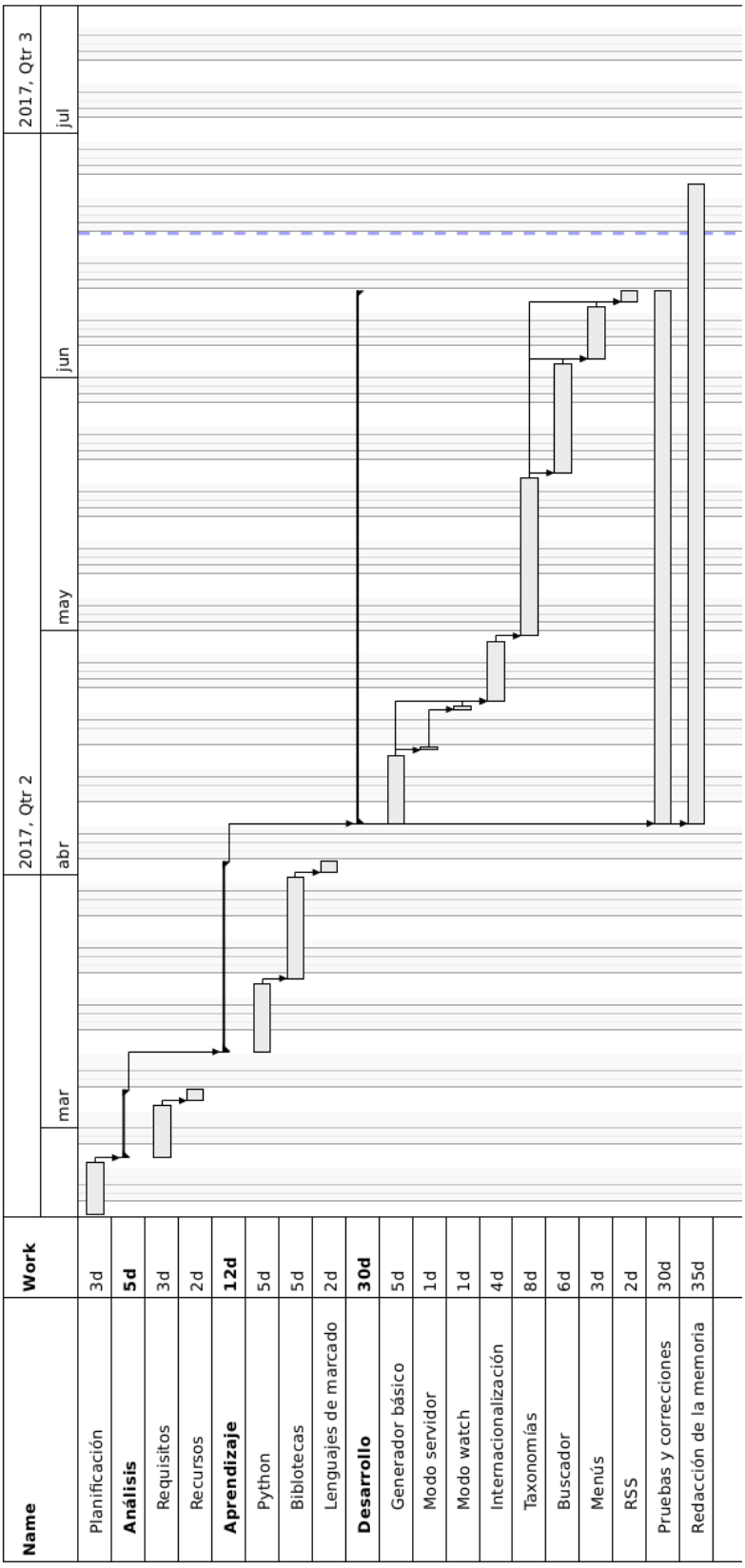

Figura 3.1: Diagrama de Gantt

Los costes de cada rol han sido obtenidos a través de la web http://espana.jobtonic.es.

Coste total de desarrollo:

Concepto	**Coste unitario**	**Unidades**	**Coste total**
Portatil Asus	720€	3	2.160€
Analista	23€/hora	50 horas	1.150€
Desarrollador	18€/hora	300 horas	5.400€
Encargado de pruebas	15€/hora	60 horas	900€
Total			**9.610€**

Tabla 3.1: Tabla de planificación de costes

Capítulo 4

Requisitos del sistema

En este capítulo procederemos a describir de manera formal los objetivos que se presentaron anteriormente de manera genérica.

En particular, se detallan:

- Los objetivos del sistema, que resumen a muy grandes rasgos la funcionalidad del proyecto.
- Los requisitos funcionales, que definen las funciones del sistema software y sus componentes.
- Los requisitos de información, que detallan los datos que el sistema usará durante el desarrollo y ejecución para llevar a cabo de forma adecuada su funcionalidad.
- Los requisitos no funcionales, que, en general, especifican criterios de diversas categorías que el software debe cumplir para garantizar un buen nivel de calidad más allá de su funcionamiento.

Para terminar el capítulo se presentan diversas alternativas tecnológicas para cubrir las necesidades surgidas en los requerimientos del sistema, detallando las decisiones tomadas.

4.1. Objetivos del sistema

Objetivos principales de Sitic, definidos a alto nivel son los siguientes:

Título	Crear contenido HTML a partir ficheros de texto plano
Descripción	Sitic permitirá generar una web completamente estática a partir de ficheros de texto plano, que se escribirán siguiendo alguno de los lenguajes de marcado soportados.

Tabla 4.1: OBJ-1, objetivo del sistema 1

Título	Personalización
Descripción	Sitic permitirá a los usuarios personalizar de la forma que desee la representación que tendrás cada uno de los contenidos.

Tabla 4.2: OBJ-2, objetivo del sistema 2

Título	Configuración
Descripción	Sitic permitirá a los usuarios configurar cualquier aspecto del sistema que se desee cambiar, así como configurar los atributos que el usuario necesite en cada uno de los contenidos.

Tabla 4.3: OBJ-3, objetivo del sistema 3

4.2. Catálogo de requisitos

Se presentan a continuación los requisitos del sistema, tanto a nivel funcional como de información y no funcionales.

4.2.1. Requisitos funcionales

En esta subsección se introducen los requisitos funcionales, que describen las funciones del sistema software.

Título	Escribir contenido haciendo uso de lenguajes de marcado
Descripción	Se deberá de poder crear los contenidos de la web escribiendo los contenidos haciendo uso de un lenguaje de marcado más sencillo e intuitivo que HTML.

Tabla 4.4: RQF-1, requisito funcional 1: contenidos

Título	Internacionalización
Descripción	Se deberán de poder configurar tantos idiomas con se desee generando sitios independientes por idiomas de sólo aquellos contenidos en el idioma generado.

Tabla 4.5: RQF-2, requisito funcional 2: internacionalización

Título	Búsqueda
Descripción	Debe de ser posible realizar búsquedas de contenidos en la web generada.

Tabla 4.6: RQF-3, requisito funcional 3: búsqueda

Título	Categorización
Descripción	Los contenidos deben de poder ser categorizable en secciones y taxonomías, dando como resultado grupos de contenidos en el sitio generado.

Tabla 4.7: RQF-4, requisito funcional 4: categorización

Título	Paginación
Descripción	La herramienta debe de proporcional paginación en los contenido agrupados, evitando así generar páginas demasiado grandes.

Tabla 4.8: RQF-5, requisito funcional 5: paginación

4.2.2. Requisitos de información

A continuación se presentan los requisitos de información, que describen los datos necesarios que requiere la aplicación para operar.

Título	Configuración del sitio
Descripción	El sitio tendrá un fichero de configuración básica: ▪ Idiomas ▪ Atributos referentes al sitio(paginación, urls, etc) ▪ Menús

Tabla 4.9: RQF-6, req. de información 6: configuración

Título	Fichero con contenido
Descripción	Los ficheros de contenido se compondrá de: ▪ Metadatos ▪ Contenido ▪ Menús

Tabla 4.10: RQF-7, req. de información 7: fichero

Título	Plantillas HTML
Descripción	Plantillas que se usarán para renderizar los contenidos

Tabla 4.11: RQF-8, req. de información 8: plantillas

4.2.3. Requisitos no funcionales

Seguidamente se presentan los requisitos no funcionales, que especifican criterios usados para juzgar la operación del software a través de varias categorías, más allá de su comportamiento específico.

Requisitos de fiabilidad

Título	Generación robusta
Descripción	El sitio generado deberá ser completamente fiable en el sentido que se generen tantos contenidos como los definidos.

Tabla 4.12: RQF-9, req, no funcional 9: generación robusta

Requisitos de accesibilidad y usabilidad

Título	Alta capacidad de configuración
Descripción	El sistema deberá de ser tan configurable como sea posible, de forma que cualquier usuario lo pueda adaptar lo máximo posible a sus necesidades.

Tabla 4.13: RQF-10, req. no funcional 10: configuración

Requisitos de eficiencia

Título	Generación de gran cantidad de documentos
Descripción	El sistema debería de ser capaz de generar un sitio a partir de una gran cantidad de contenidos en un tiempo razonable.

Tabla 4.14: RQF-11, req. no funcional 11: eficiencia

4.3. Alternativas de solución

En esta sección, se presentan diferentes alternativas tecnológicas que permiten satisfacer los requerimientos del sistema, presentando las alternativas elegidas y detallando las razones de esta decisión.

4.3.1. Biblioteca para crear simples interfaces por línea de comandos

Dada la mayor familiaridad del desarrollador con el lenguaje, solo se barajaron *frameworks* de desarrollo basados en el lenguaje Python.

- **Click** [3]: paquete de Python para crear interfaces de línea de comandos de una manera sencilla a partir del código. Es un "Kit de creación de interfaz de línea de comandos". Es altamente configurable. Su objetivo es hacer que el proceso de escritura de herramientas de línea de comandos sea rápido y divertido, al tiempo que evita cualquier frustración causada por la imposibilidad de implementar una CLI API.

- **Fire** [13]: es una biblioteca para generar automáticamente interfaces de línea de comandos desde cualquier objeto Python.
- **Argparse** [1]: pertenece a la biblioteca estándar de Python y facilita la escritura de interfaces de línea de comandos fáciles de usar. El programa define qué argumentos requiere, y ArgParse descubrirá cómo analizar los que están fuera de sys.argv. El módulo ArgParse también genera automáticamente mensajes de ayuda y uso y emite errores cuando los usuarios dan al programa argumentos inválidos.

Se tomó la decisión de optar por **Click** ya que parecía más sencillo y las opciones que ofrecía sería mucho más útiles.

4.3.2. Sistema de plantillas

El sistema contará con un motor de plantillas que permitirá al usuario definir y extenderlas de forma que pueda reutilizar prácticamente todo el código HTML que genere para las propias plantillas de su sitio. Las alternativas barajadas son:

- **Jinja** [8]: es un motor de plantillas completo para Python. Cuenta con soporte unicode completo, un entorno de ejecución de espacio de trabajo integrado opcional, ampliamente utilizado y licenciado por BSD.
- **Chamaleon** [2]: es un motor de plantilla HTML / XML para Python. Está diseñado para generar la salida de documentos de una aplicación web, normalmente marcado HTML o XML.

Finalmente se decidió usar **Jinja** por ser el sistema de plantilla más robusto y con mayor número de utilidades.

4.3.3. Lenguajes de marcado soportados

El sistema convertirá los contenidos escritos en ficheros de texto plano a HTML, estos ficheros deberá de estar escritos usando algún lenguaje de marcado. Las opciones que se barajaron fueron las siguientes:

- **Markdown** [11]: es un lenguaje de marcado ligero que trata de conseguir la máxima legibilidad y facilidad de publicación tanto en su forma de entrada como de salida, inspirándose en muchas convenciones existentes para marcar mensajes de correo electrónico usando texto plano. Se distribuye bajo licencia BSD.
- **RestructuredText** [15]: es un lenguaje de marcas ligero creado para escribir textos con formato definido de manera cómoda y rápida. Es parte del proyecto DocUtils dentro de la comunidad de Python. Es una evolución de Structured Text.

- **Textile** [18]: es un lenguaje de marcado ligero que partiendo de un texto con marcas genera código HTML. Se utiliza para generar cualquier tipo de contenido escrito que deba ser publicado en internet.
- **Wikitexto** [24]: es un texto elaborado mediante un lenguaje de marcado especial para la creación de páginas wikis en servidores que tengan instalado algún software para wikis. El lenguaje de marcado para wikis se denomina lenguaje wikitexto y no existe un estándar que defina su sintaxis, sus características y su estructura, como la tiene el lenguaje HTML. Por el contrario, depende del software wiki utilizado.

En un primer momento se decidió usar únicamente **Markdown**, ya que parecía el mas intuitivo, sencillo y usado por la comunidad. Finalmente, se decidió también por dar soporte a **RestructuredText** y **Textile**, ofreciendo al usuario la posibilidad de elegir aquel con el que se sienta más cómodo.

4.3.4. Formato de configuración para los metadatos de los ficheros

En cada fichero de texto plano, se podrá definir una serie de metadatos, usando un formato concreto. Las alternativas pensadas fueron las siguientes:

- **JSON** [10]: es un formato de texto ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript aunque hoy, debido a su amplia adopción como alternativa a XML, se considera un formato de lenguaje independiente.

  Ejemplo de formato:

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "spouse": null
}
```

- **TOML** [19]: es un formato de archivo de configuración que es fácil de leer debido a la semántica obvia que pretende ser "mínima". TOML está diseñado para hacer asignaciones a un diccionario sin ambigüedad.

  Ejemplo de formato:

```
title = "TOML Example"
```

```
[instance]
server = "192.168.1.1"
ports = [ 8001, 8001, 8002 ]
connection_max = 5000
enabled = true
```

- **YAML** [25]: formato de serialización de datos legible por humanos inspirado en lenguajes como XML, C, Python, Perl.

  Ejemplo de formato:

  ```
  men: [John Smith, Bill Jones]
  women:
    - Mary Smith
    - Susan Williams
  ```

Finalmente se decidió usar dos alternativas, tanto **YAML** como **TOML**, de forma que el usuario use aquel que encuentre más sencillo.

Capítulo 5

Análisis

En este capítulo se dispone el análisis realizado en base a los requisitos presentados en el capítulo 3, haciendo uso del lenguaje de modelado UML. Concretamente, el capítulo describe:

- El modelo conceptual de los datos usados en la aplicación, elaborado a partir de los requisitos de información.
- Los casos de uso, que describen los pasos que componen los procesos habituales del proyecto.

5.1. Modelo conceptual

De acuerdo a lo reflejado en la sección anterior se presenta el diagrama 5.1 en el que se identifican los principales tipos de datos, sus atributos y las relaciones entre ellos.

A continuación se detallan cada uno de los tipos de datos reflejados en el modelo conceptual.

5.1.1. Sitio web

Representan el sitio web generado a partir de todos los contenidos que lo componen.

Relaciones:

- Esta compuesto por varios contenidos.
- Está compuesto por una configuración.

Objetivos:

- Contiene todos los elementos necesarios en un sitio web.

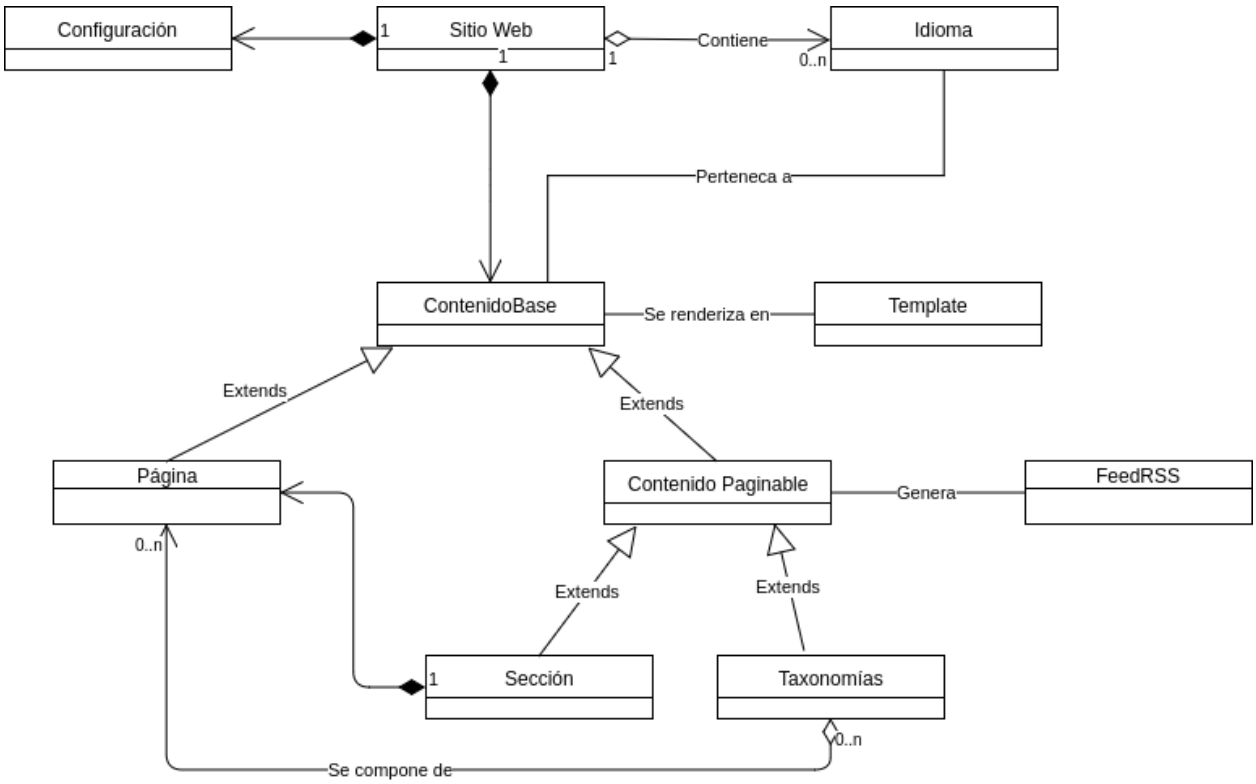

Figura 5.1: Modelo conceptual

- Se encarga de representar los elementos de la forma adecuada.

5.1.2. Configuración

Representa la configuración básica.

Relaciones:

- Pertenece a un sitio web.

Objetivos:

- Almacena todos los elementos configurables del sitio web.
- Aplica valores determinados a los parámetros no especificados.
- Provee a cada una de las partes los parámetros de configuración necesarios.

5.1.3. Idioma

Representa los idiomas disponibles.

Relaciones:

- Pertenece a un sitio web.

Objetivos:

- Contiene los distintos idiomas disponibles en la web.
- Gestiona los contenidos que están en cada idioma.
- Gestiona la traduccion de la interfaz general del sitio web.

5.1.4. Contenido base

Representa cualquier contenido que existe en un sitio web.

Relaciones:

- Pertenece a un sitio web.
- Se relaciona con una plantilla en la que se tiene que renderizar.
- Se relaciona con un idioma.

Objetivos:

- Convertir los contenidos añadidos por los usuarios en formato HTML.

5.1.5. Contenido paginable

Representa a un contenido, que a su vez está compuesto por otros contenidos, lo que implica que es paginable.

Relaciones:

- Extiende un contenido base.

Objetivos:

- Organizar cada uno de los contenidos de los que está compuesto en distintas páginas.
- Facilitar al usuario la gestión de las páginas.

5.1.6. Página

Representa la página que redacta un usuario.

Relaciones:

- Extiende un contenido base.
- Pertenece a una sección.

- Se clasifica en varias taxonomías.

Objetivos:

- Convertir a HTML los contenidos escritos en el lenguaje de marcado usado por el usuario.
- Gestionar los metadatos definidos por los usuarios en cada contenido.

5.1.7. Sección

Representa a un conjunto de contenidos.

Relaciones:

- Está compuesto por varias páginas.
- Extiende un contenido base.

Objetivos:

- Agrupar y clasificar páginas.
- Permitir mostrar las páginas en las distintas secciones a las que pertenece.

5.1.8. Taxonomías

Representa la clasificación de contenidos.

Relaciones:

- Extiende un contenido base.
- Contiene varias páginas.

Objetivos:

- Agrupar y clasificar páginas de una forma más concreta.
- Permitir mostrar las páginas en las distintas taxonomías a las que pertenece.

5.1.9. Template

Representa la plantilla HTML en la que se renderiza un contenido

Relaciones:

- Se relaciona con un contenido.

Objetivos:

- Renderizar el contenido en una plantilla HTML.

- Personalizar la forma en la que aparecerán los contenidos.
- Posibilidad de reaprovechar plantillas para evitar repetir código.

5.1.10. Feed RSS

Representa la representación RSS de un contenido paginable

Relaciones:

- Pertenece a un contenido paginable.

Objetivos:

- Lista los contenidos en un formato que puedan ser procesado por lectores RSS.

5.2. Modelo de casos de uso

El modelo de casos de uso representa las interacciones entre los actores y el sistema, desarrollado a partir de los requisitos descritos anteriormente.

Actores

En este apartado se describen los diversos roles que juegan los usuarios del sistema. Para ese caso en concreto, sólo existirá un usuario ya que es el propio usuario el que realiza todas las acciones para la generación de los contenidos.

Figura 5.2: Diagrama de actores

Caso de uso: configuración del sitio

Descripción El usuario debe de poder configurar todas las opciones disponibles a través de un fichero general de configuración

Actores *Usuario*.

Escenario principal

1. El *Usuario* escribe el contenido del fichero de configuración.
2. El *Usuario* ejecuta la ejecución de contenido.
3. El sitio se genera satisfactoria siguiendo todos los parámetros configurados.

Flujo alternativo

3a Existe algún error de formato en el fichero y el sistema avisa al usuario.

Caso de uso: creación de contenidos

Descripción El usuario debe de poder definir contenidos y generarlos en HTML.

Actores *Usuario*.

Escenario principal

1. El *Usuario* define contenidos en la carpeta de contenidos.
2. El *Usuario* ejecuta la generación del sitio.
3. El contenido de los ficheros es generado correctamente en HTML

Flujo alternativo

2a El formato de alguno de los ficheros no es correcto y el sistema informa al usuario.

Caso de uso: metadatos de contenidos

Descripción El usuario debe de poder definir metadatos al inicio de los ficheros.

Actores *Usuario*.

Escenario principal

1. El usuario añade al inicio del fichero metadatos que necesita en la renderización de éstos.
2. El usuario ejecuta la generación del sitio.
3. El usuario puede usar esos metadatos al renderizar los contenidos en la plantillas.

Flujo alternativo

2a Los metadatos no siguen el formato correcto y el sistema notifica al usuario.

Caso de uso: clasificación de contenidos

Descripción El usuario debe de poder definir taxonomías y poder usarlas en los ficheros

Actores *Usuario*.

Escenario principal

1. El usuario define las taxonomías necesarias en el fichero de configuración.
2. El usuario usa las taxonomías en los metadatos de los ficheros para clasificarlos.
3. El usuario ejecuta la generación del sitio.
4. Se generan tantas urls como taxonomías definidas con los contenidos asociados.

Flujo alternativo

3a Se produce algún error en la generación y el sistema avisa al usuario.

Caso de uso: internacionalicación

Descripción El usuario debe de poder definir más de un idioma.

Actores *Usuario*.

Escenario principal

1. El *Usuario* define los idiomas deseados en la configuración.
2. El *Usuario* define contenido en todos esos idiomas.
3. El *Usuario* genera el sitio.
4. El sitio se genera con cada contenido definido en el idioma que le corresponde

Flujo alternativo

3a Existe algún problema en alguna de las configuraciones, el sistema avisa del error y cancela la ejecución.

Caso de uso: definición de plantillas

Descripción El usuario debe de poder definir plantillas HTML en función del tipo de contenido.

Actores *Usuario*.

Escenario principal

1. El *Usuario* define distintas plantillas en función del tipo de contenido.
2. El *Usuario* genera el sitio.
3. El sitio renderiza correctamente los contenidos en las plantillas correspondientes.

Flujo alternativo

3a Existe algún problema en la generación y el sistema avisa del error y cancela la ejecución.

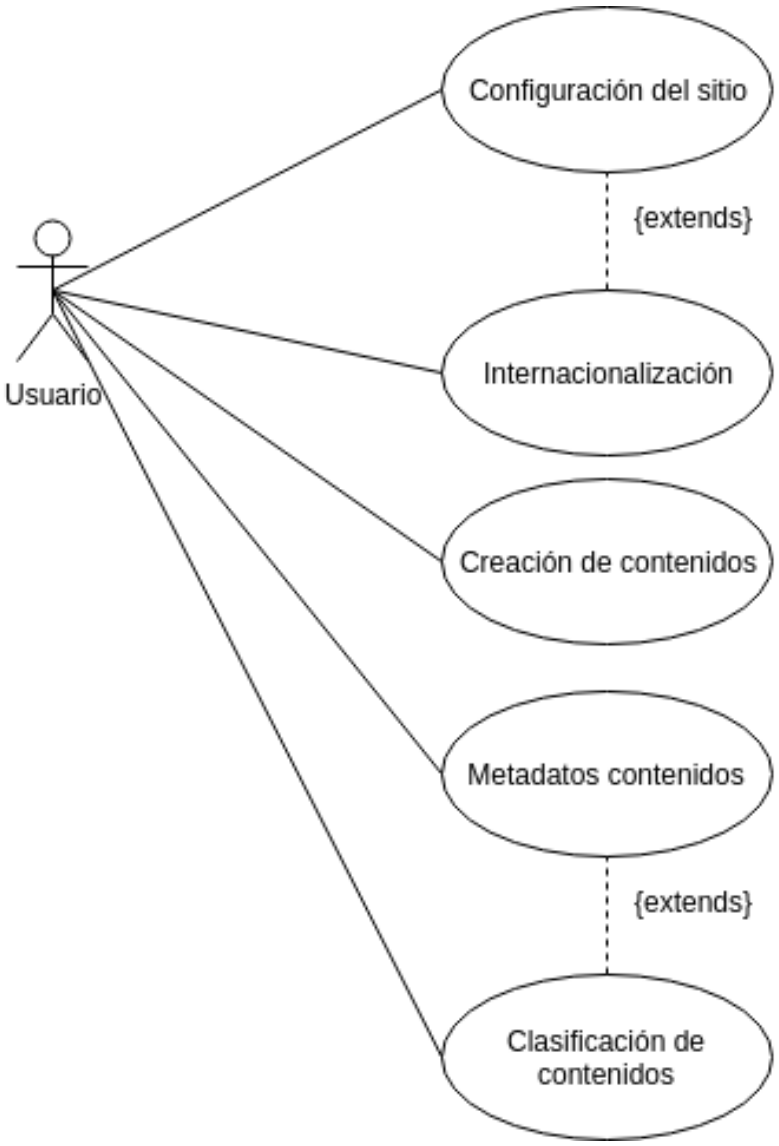

Figura 5.3: Casos de uso

Capítulo 6

Diseño

En este capítulo se presentan los detalles de diseño del proyecto, basándonos en el análisis mostrado en los anteriores apartados. Se detallan la arquitectura general y el diseño físico de datos entre otros aspectos.

6.1. Arquitectura del sistema

6.1.1. Arquitectura física

En este apartado, describimos los principales componentes hardware que forman la arquitectura física de nuestro sistema.

Hardware El hardware mínimo indispensable para la correcta ejecución del motor del proyecto se detalla en la siguiente lista:

- 512 MiB de memoria RAM como mínimo.
- 10 GiB de disco duro como mínimo.

Software En cuanto al software necesario para la ejecución del proyecto, se detallan los siguientes elementos, que es necesario instalar para desplegar el sistema:

- Sistema operativo **GNU/Linux**, preferiblemente basado en paquetería Debian.
- Código fuente del proyecto **Sitic**.
- Intérprete de **Python**, versión mínima 2.7.
- Soporte de entornos virtuales **VirtualEnv** para la encapsulación de dependencias.

6.1.2. Arquitectura lógica

La arquitectura lógica del sistema está formada por los elementos software (servicios, aplicaciones, librerías, etc.) que componen el software base, más el software desarrollado para cumplir los requisitos de la aplicación. En esta sección se muestra la organización de los distintos elementos software que componen el proyecto así como la comunicación entre ellos.

En la figura 6.1 se puede ver un esquema de cómo se comunican los diferentes elementos que conforma el sistema a alto nivel. En detalle, el flujo que sigue en cada capa es el siguiente:

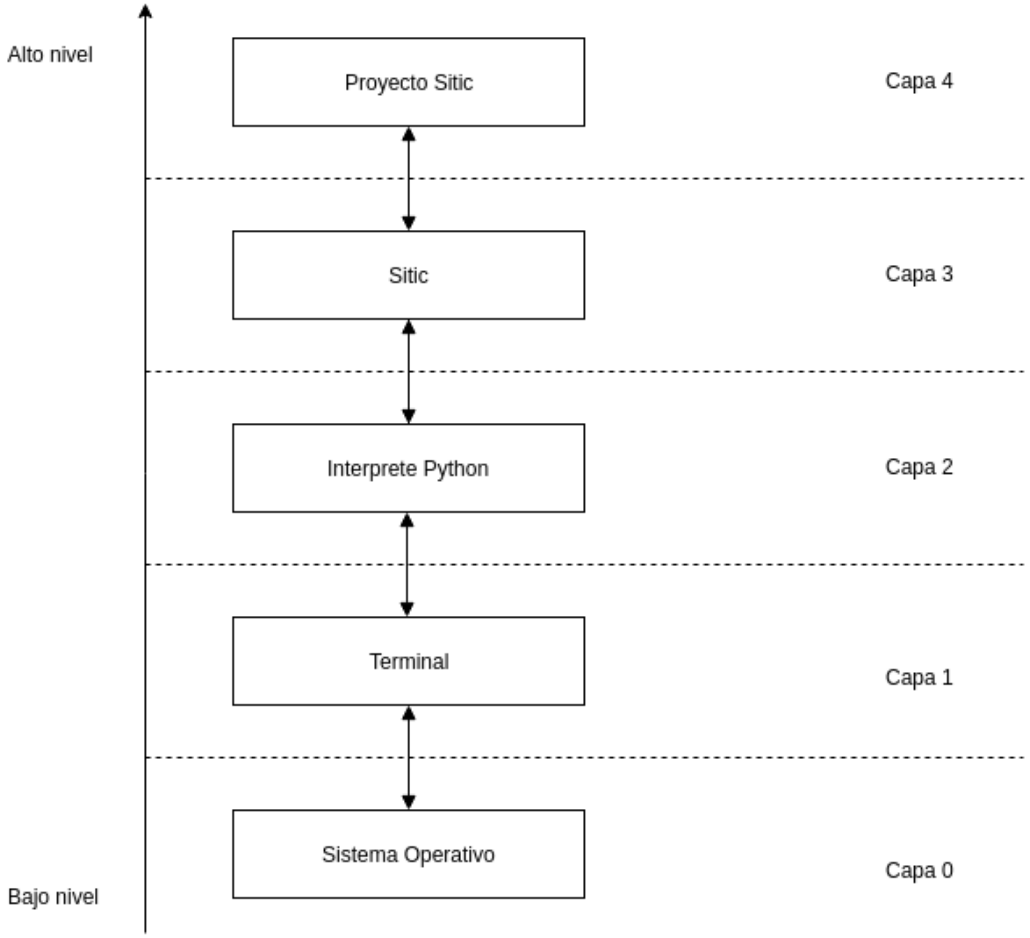

Figura 6.1: Arquitectura lógica

De esta forma, en este apartado de arquitectura lógica, vamos a describir todos los elementos que componen el software base de la aplicación, además del propio software a desarrollar. De esta forma, podemos citar los siguientes componentes software:

- Primeramente, como acabamos de comentar, nos encontramos el elemento software que compone a nuestro proyecto, como ya se ha descrito anteriormente, un generador de sitios web estáticos.

- Esta herrameinta está desarrollado haciendo uso del lenguaje de programación Python, por lo que es indispensable que la máquina donde se use la herramienta, tenga el lenguaje disponible.
- A su vez, dicho proyecto se ejecutará usando una terminal. La terminal del sistema debe de tener como requisito poder ejecutar código Python.
- Por último, el sistema operativo donde se vaya a instalar todos los componentes anteriores, debe de tener soporte tanto para el servidor web donde vamos a incluir nuestro proyecto, como para la tecnología Python, ya que el resto de componentes comentados, funcionan sobre estas dos piezas. Por suerte, Python es un lenguaje de programación multiplataforma, el cual tiene soporte para los principales sistemas operativos existentes en la actualidad.

6.2. Diseño detallado de componentes

En esta sección se detalla el diseño concreto de algunos componentes de la aplicación, especificando las clases involucradas y el flujo de mensajes entre ellas.

En particular, se han elegido el proceso parseo y generación de contenidos por ser el que mayor importancia y complejidad tiene.

Parseo y generación de contenidos

El proceso de parseo y generación de contenidos es un proceso que involucra bastantes elementos y es el componente esencial de la herramienta.

En primer lugar, cuando el usuario ejecuta la generación de contenidos, ejecutando el comando principal de la herramienta, el sistema recorre todos y cada uno de los ficheros que se encuentran en la carpeta de contenidos definida.

Cada fichero lo separa en dos partes, la parte que contiene los metadatos del contenido y la parte que contiene el contenido redactado. Una vez hecho esto parsea cada una de las partes para convertirlos en un tipo de datos que sea de fácil manipulación, los metadatos los convierte a un tipo de datos diccionario del lenguaje Python y el contenido en sí, lo parsea en función de la extensión que se le haya puesto al fichero, que determinará que lenguaje de marcado hemos utilizado.

Una vez que todos los ficheros han sido parseados, determina qué ficheros se van a generar, ya que el usuario puede haber definido en los metadatos que el fichero haya caducado o que simplemente no esté listo para ser publicado aún.

Teniendo ya los ficheros que se van a publicar, se recopilan el resto de contenidos a partir de estos, es decir, todos aquellos contenidos que depende de estos como son la página inicial, las secciones o las taxonomías, ya que si, por ejemplo, existe una sección con un único contenido, se evita que dicha sección se genere completamente vacía.

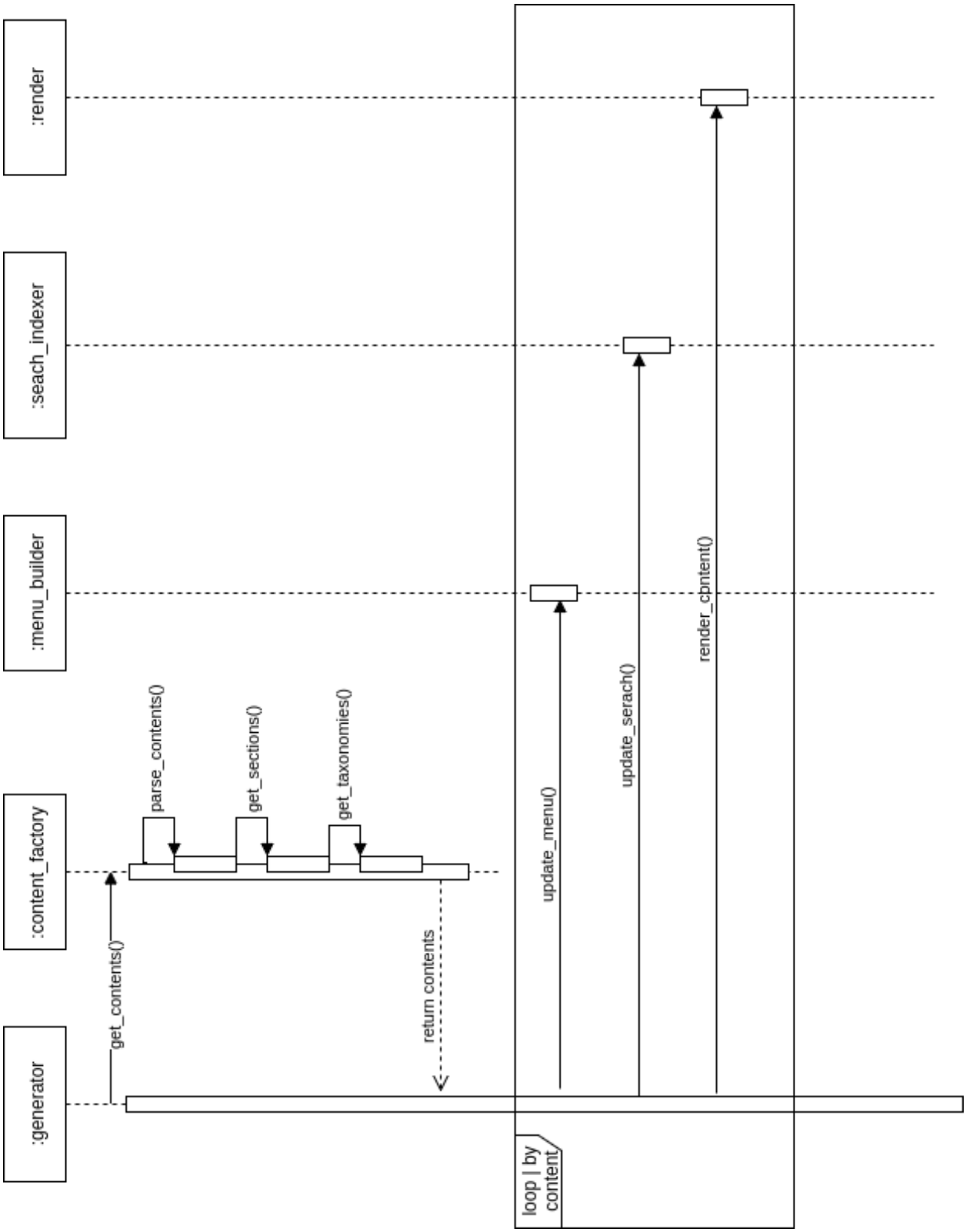

Figura 6.2: Diagrama de secuencia

Tras esto, se recorre cada uno de los ficheros, determinando que plantilla se debe usar como base para renderizarlos, y se le comunica al motor de plantillas Jinja, el contenido que debe de renderizar.

Por último, el sistema mueve todos los ficheros estáticos (CSS, imágenes, Javascript) que la web pueda necesitar a la carpeta con todos los contenidos generados.

6.3. Diagrama de clases de diseño

A continuación se muestra el diagrama de clases de diseño. Que se ha tenido que dividir en dos partes para su correcto visionado.

En el diagrama de clases de pueden apreciar clases que no están relacionadas con ninguna otra del diagrama. Estas clases a las que nos referimos siguen un patrón Singleton [16], que permite que cualquier clases pueda acceder a ella desde cualquier parte del sistema. Sólo existirá una única instancia de cada una de las clases Singleton, que se creará en la primera llamada que se realice a ellas.

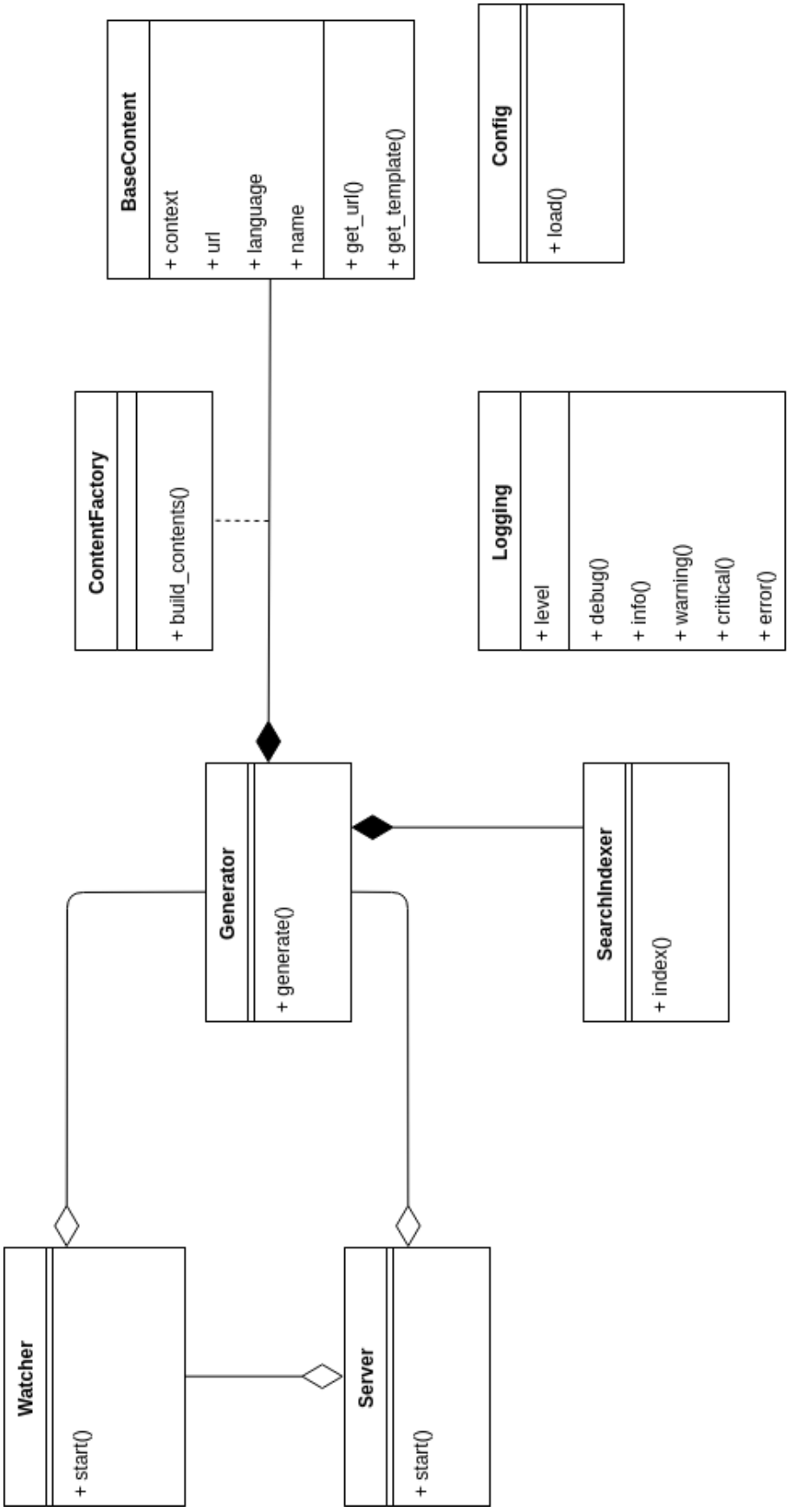

Figura 6.3: Diagrama de clases de diseño 1

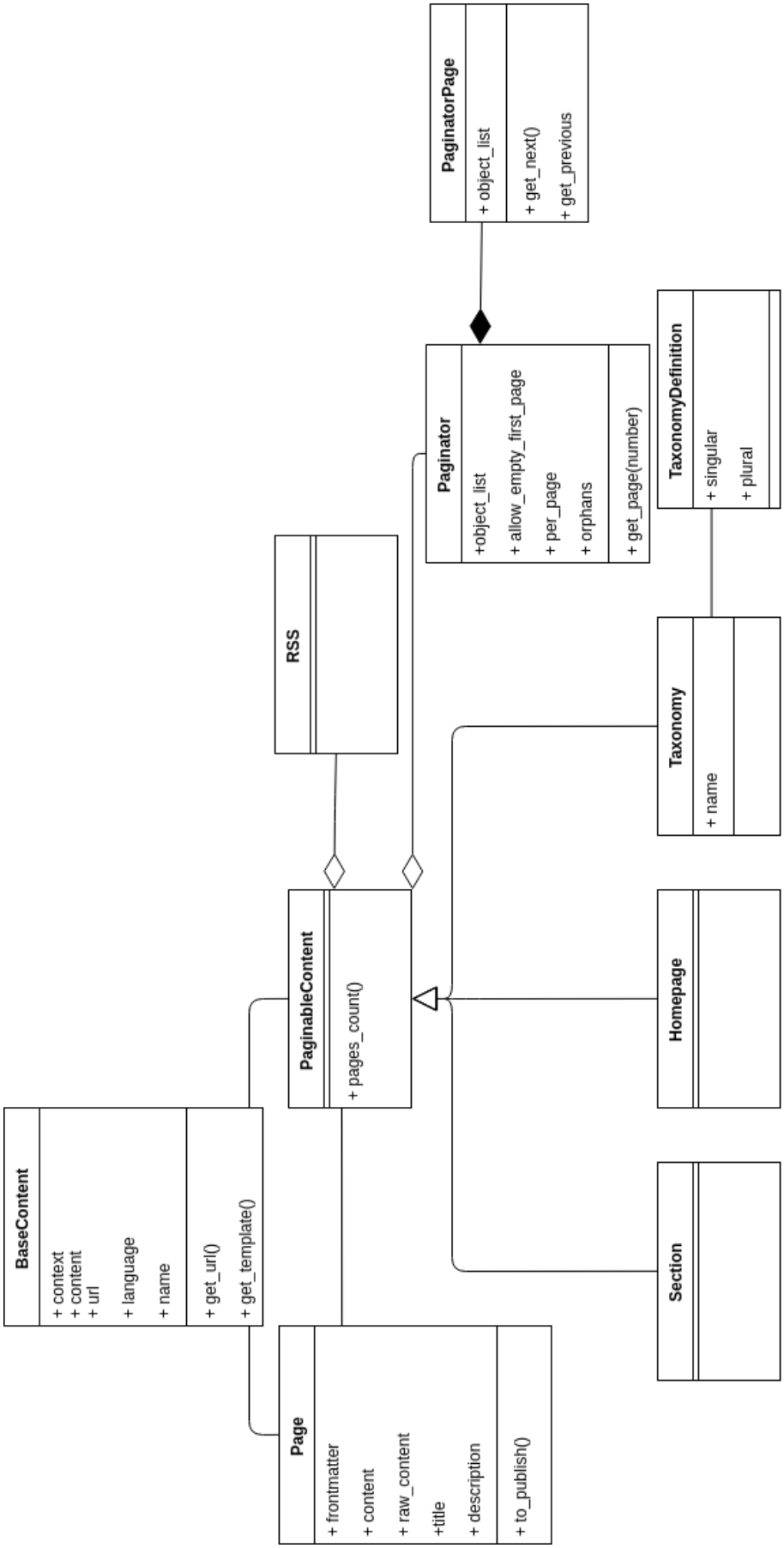

Figura 6.4: Diagrama de clases de diseño 2

Capítulo 7

Implementación

7.1. Entorno de construcción

7.1.1. Herramientas de diseño y desarrollo

Como editor principal se ha utilizado Vim [20], una versión mejorada del editor de texto vi, presente en todos los sistemas UNIX.

Se ha utilizado draw.io [4] para la generación de diagramas de casos de uso, secuencia, diagramas de flujo y similares. Se trata de una herramienta web, actualmente integrable en Google Drive, con un enorme número de elementos de dibujo y la capacidad de exportar en un gran número de formatos, entre ellos en formato vectorial PDF (Portable Document Format).

(a) Logo de Vim

(b) Logo de Draw.io

7.1.2. Gestión de dependencias

Para facilitar la gestión de las dependencias del proyecto se han utilizado VirtualEnv [21] y VirtualEnvWrapper [22]. Estas herramientas permiten generar entornos virtuales para cada proyecto, en los que se instalan las dependencias necesarias. Estos entornos se activan y desactivan, de forma que las bibliotecas instaladas

en el entorno virtual de un proyecto no son accesibles desde el entorno del otro proyecto. Esto evita la polución del nivel general de bibliotecas y facilita el control estricto de las versiones de las dependencias.

Junto a Virtualenv, la herramienta pip [12] permite guardar en un fichero anexo la lista de dependencias de un proyecto, de forma que sea fácil reinstalarlas todas si hubiese que repetir la instalación en otro sistema.

7.1.3. Control de versiones

Todo el código fuente del proyecto se encuentra alojado en un repositorio público en GitHub, haciendo uso de los planes gratuitos. GitHub es un repositorio que utiliza el sistema de control de versiones Git [6]. Además del alojamiento de código, GitHub provee numerosas funcionalidades adicionales, tanto a nivel social (permitiendo a los repositorios tener seguidores, por ejemplo) como a nivel funcional (ofreciendo sistemas de tickets, estadísticas, etcétera).

El uso de un control de versiones es fundamental por varios motivos. En primer lugar, sirve como sistema de copia de seguridad. En segundo lugar, permite deshacer cambios en el código que no funcionen bien, siendo siempre posible volver atrás. Por último, sirve como cuaderno de bitácora improvisado, ya que se guarda el historial de commits que el desarrollador va enviando junto a los mensajes, siendo posible ver en una línea temporal el progreso del trabajo.

Figura 7.2: Logo de Git

7.1.4. Lenguaje de programación

Como se ha comentado en numerosas ocasiones en la presente memoria, el lenguaje de programación elegido para el desarrollo del proyecto es Python [14], un lenguaje interpretado de alto nivel desarrollado por Guido Van Rossum en 1991. Python soporta múltiples paradigmas de programación, desde la orientación a objetos hasta la programación funcional, pasando por el clásico estilo imperativo. Su principal uso ha sido como lenguaje de *scripting*, pero también tiene su hueco en contextos más amplios como lenguaje principal.

Su facilidad de aprendizaje, lo simple de su sintaxis (basada en la indentación para marcar los bloques) y su extensibilidad (sobre todo gracias al uso de métodos

mágicos y metaprogramación) han hecho que el lenguaje sea muy popular y su uso en los últimos años se haya expandido enormemente.

Figura 7.3: Logo de Python

7.1.5. Bibliotecas de terceros

En el proyecto se han utilizado un gran número de bibliotecas auxiliares para facilitar el desarrollo y ampliar la funcionalidad de forma sencilla.

Jinja2 [8] Es un lenguaje de plantillas moderno y diseñado para Python, modelado a partir de las plantillas de Django. Es rápido, ampliamente utilizado y seguro con el entorno de ejecución.

Características:

- Sistema de escapado de HTML para la prevención XSS.
- Herencia de la plantillas.
- Compilación opcional de plantillas por adelantado.
- Fácil de depurar.
- Sintaxis configurable.

Figura 7.4: Logo de Jinja2

Watchdog [23] API para monitorizar eventos en el sistema de ficheros.

- Multiplataforma.
- Herramientas para responder a cambios en directorios.

Click [3] Es un paquete de Python para crear interfaces de línea de comandos de una manera sencilla. Es el "Kit de creación de interfaz de línea de comandos". Es altamente configurable.

Características:

- Anidamiento arbitrario de comandos.
- Generación de ayuda automática.
- Soporta la carga de subcomandos en tiempo de ejecución.

Six [17] Proporciona utilidades simples para envolver las diferencias entre Python 2 y Python 3. Pretende que soporte las bases de código que funcionan tanto en Python 2 como en 3 sin modificación. Consta de un solo archivo Python, por lo que es muy sencillo de añadir a un proyecto.

7.2. Organización del código fuente

En esta sección se detalla la organización del código fuente del proyecto, describiendo la utilidad de los ficheros y directorios.

Como complemento a la lectura de este capítulo se recomienda tener una copia local del repositorio del proyecto, disponible para su libre descarga desde el repositorio oficial [9].

El código fuente cuenta con los siguientes elementos en el directorio raíz:

- `doc`: directorio con toda la documentación del proyecto.
- `examples`: directorio con ejemplos de webs haciendo uso de la herramienta.
- `sitic`: código fuente.
- `README.md`: fichero de presentación para el repositorio.
- `requirements.txt`: fichero con dependencias instalables con Pip.

7.2.1. Código fuente

El código se encuentra íntegramente contenido en el directorio sitic del proyecto, que contiene lo siguiente:

- `cli_parser.py`: definición de comandos disponibles.
- `compilemessages.py`: comando para compilar traducciones.
- `config.py`: encargado de parsea la configuración.
- `content`: directorio con los ficheros que gestionan los contenidos.

    - `base_content.py`: implementa el contenido base del que heredan el resto de contenidos.
    - `content_factory.py`: implementa el patrón factory para obtener contenidos.
    - `frontmatter_handlers.py`: parseado de metadatos de contenidos.
    - `homepage.py`: implementa la homepage.
    - `menu_builder.py`: implementa el menú.
    - `menu.py`: implementa un elemento del menú.
    - `page_parser.py`: implementa el parseador de contenidos.
    - `page.py`: implementa el contenido escrito por el usuario.
    - `paginable_content.py`: implementación de contenidos paginables.
    - `paginator.py`: implementa paginación de secciones y taxonomías.
    - `rss.py`: implementa el rss.
    - `section.py`: implementa las secciones.
    - `sitemap.py`: implementa el sitemap de la web.
    - `taxonomy.py`: implementa las taxonomías.
- `files`: directorio con plantillas necesarios.
    - `pot template.jinja`: plantilla de fichero de traducciones.
    - `rss.xml`: plantilla para la generación del RSS.
    - `search.js`: fichero javascript necesario para la búsqueda.
    - `sitemap.xml`: plantilla de sitemap.xml.
- `generator.py`: generador de contenidos.
- `logging.py`: logging de la aplicación.
- `makemessages.py`: comando para obtener todos los mensajes traducibles del sitio web.
- `scoper.py`: implementa funcionalidad para modificación de variables en las plantillas.
- `search.py`: implementa el buscador.
- `server.py`: comando para montar servidor local de pruebas.
- `template`: directorio con el código encargado de la renderización de las plantillas.
    - `filters.py`: filtros propios para usar en plantillas.
    - `render.py`: renderizador de plantillas.

- `utils`: directorio con código común entre distintos módulos.
  - `constants.py`: constantes del proyecto.
  - `enums.py`: enumerados usados en el proyecto.
- `watcher.py`: Comando encargado de regenerar el contenido cada vez que existe algún cambio.

7.3. Detalles de implementación

En esta sección se describen algunos detalles en la implementación del proyecto que han resultado de interés suficiente para ser dignos de mención, ya sea por los conflictos que dieron, por su complejidad o por otras razones subjetivas.

Implementación estructura de ficheros

Uno de los primeros retos a la hora de la implementación, era ofrecer al usuario una forma sencilla de dividir el contenido de los ficheros y los metadatos de estos.

Para que fuera sencillo tanto para los usuario, como para la herramienta poder diferencias esas dos partes de forma inequívoca, se decidió que los ficheros de contenidos usará delimitadores que marcaran que parte eran metadatos y debía ser tratados como tal y que parte era el contenido real del fichero.

Dado que el sistema ofrece escribir los metadatos en dos formatos, `YAML` y `TOML`, se decidió que cada uno tuviera su propio carácter delimitador:

- El formato `TOML` usaría el carácter `+`.
- El formato `YAML` usaría el carácter `-`.

A continuación se puede ver un ejemplo de un fichero de contenido `Markdown` con los metadatos en formato `TOML`:

```
+++++++++++++++
title='Post 1'
tags=['tag1', 'tag2']
description="Post 1 description"
+++++++++++++++

## Variarum loricaeque

Lorem markdownum mihi. Avidoque doloris stabit; omen turaque est,
ante ergo nostraque, tibi.

    cpl_monitor_clean += html_window(management_ddl * 3, hacker);
    ccd -= -3;
```

```
    alignment_sdram += netbios_optical_party.webHardening(recycle_horizontal(5,
            networkCmsOpen, bandwidth_computer - text), -1);

## Curam vires adhuc domum

Tendere posuisse fatorum pharetras e diversis *lacertos*: ius cum vacuaque denos
quem, ausa transit puero. Undique foret et restatque supremaque accessi utile
mollire [bracchia Othryn](http://www.ratibus.net/curvavit.html) manumque; vimque
exprobravit sola ferrum diverso inania terramque. Qui illic et Romulus, ad erat
 Pergama daedalus annis, fudit.

> Hunc medio, Argus plus pro lecto nimiumque dextera. Inde pondere ipse,
> lacertos [Troianae](http://faciunt.net/) habet rex, membra, lacrimisque
> onerosa. Dedi grata vani visurus corpora pariter ferarum **est quodque
> stimuletur**, vestri discrimina sceptrum, abit doceri septem iuvenumque. Ego
> est forte inpetus coniugis, auras inania regnat, tum languescuntque Byblida
 > nisi, mentes. Vident candidus flectitur humana.
```

Internamente el sistema automáticamente divide el contenido del fichero en dos partes y las procesa por separado. A continuación se puede ver el fragmento de código que se encarga de realizar dicha tarea:

```
from sitic.content.frontmatter_handlers import YamlHander, TomlHandler

# Define every supported metadata handler
handlers = [handler() for handler in [YamlHander, TomlHandler]]

def __get_handler(text):
    """ Detects if text metadata is in TOML o YAML format """
    text_handler = None

    # Check file with every handler and return the correct one
    for handler in handlers:
        if handler.has_format(text):
            text_handler = handler
            break

    return handler


def __parse(text, handler):
    """ Process the file and splits the content in metadata and text """
    text = text.strip()
    metadata = {}

    handler = handler or __get_handler(text)
```

```
        # If no handler found, return metadata as empty
        if not handler:
            return metadata, text

        # If handler found, split the content in metadata and text
        try:
            metadata, text = handler.split(text)
        except ValueError:
            return metadata, text

        # Parse metada into internal format
        metadata = handler.load(metadata)

        return metadata, text


    def load(file_path, handler=None):
        """ Giving a file path, returns parsed content """
        text = ''

        with open(file_path, 'r') as f:
            text = f.read()

        text = text.strip()

        # Get handler if none given
        handler = handler or __get_handler(text)

        # Once we have read the file, use loads function to handle content
        return loads(text, handler)


    def loads(text, handler=None):
        """ Giving a text and an optional handle, returns parsed content """

        # Get handler if none given
        handler = handler or __get_handler(text)

        # Return content parsed
        return __parse(text, handler)
```

Implementación de búsqueda

Hoy en día, cualquier sitio web con un número de páginas elevado, necesita un buscador con el que se puedan encontrar los contenidos de forma sencilla.

Dado que `Sitic` genera un sitio web completamente estático, lo que implica que no tenemos ningún servidor al que hacerle peticiones con los términos que se desean buscar, se decidió implementar una búsqueda acorde a las características de la herramienta.

Para conseguirlo, la implementación consistía en varias partes, la primera de ellas, generar un índice de contenidos en un fichero JSON que incluyera todos los contenidos publicados.

EL JSON resultante tendría el siguiente formato:

```
[
    {
        "content": "Variarum loricaeque\nLorem markdownum mihi...",
        "title": "Post 1",
        "url": "/blog/post1"
    },
    {
        "content": "Variarum loricaeque\nLorem markdownum mihi...",
        "title": "Post 2",
        "url": "/blog/post2"
    },
    {
        "content": "Variarum loricaeque\nLorem markdownum mihi...",
        "title": "Post 6",
        "url": "/blog/post6"
    }
]
```

Este fichero se añadiría a la carpeta del sitio generado.

La tercera parte requería implementar la lógica necesaria en `Javascript` que usara el índice generado y devolviera los resultados encontrados. Para ello se hizo uso de la biblioteca `Lunr.js` para `Javascript` que permite realizar búsquedas en índices adaptados previamente a su formato.

La tercera y última parte de la implementación consiste en que el usuario pueda de forma sencilla añadir dicho buscador en su web, para ello, se debe añadir un formulario con un id específico que el código `Javascript` anteriormente comentado consultará y por último generar una plantilla llamada `search-results.js` con un bloque determinado donde se mostrarán los resultados de la búsqueda.

Implementación internacionalización

Para ofrecer internacionalización en las posibles webs que se desarrollen usando `Sitic`, se ha hecho uso de la famosa biblioteca `gettext`, que es compatible con el motor de templates usado `Jinja2`.

En primer lugar, el usuario debe de poder configurar qué idiomas van a estar disponibles en su web. Esto debe hacerlo en el fichero configuración del sitio. A continuación podéis ver una configuración para dos idiomas, estableciendo como idioma principal el inglés:

```
main_language: 'en'

languages:
    en:
        title: "English"
    es:
        title: "Español"
```

Una vez hecho esto, el usuario deberá indicar al sistema de que idioma se trata cada contenido, añadiendo como sufijo justo antes de la extensión, el código del idioma, en el caso de que no se añada ninguno, el sistema supondrá que el formato del fichero es el idioma marcado como principal.

En el caso de las templates, el usuario tiene disponible un filtro propio de `Jinja2`.

El uso de dicho filtro se usaría de la siguiente forma dentro de una plantilla:

```
<div id="header">{% trans %}This is a test{% endtrans %}</div>
<p>{% trans user='test'%}Hello {{ user }}!{% endtrans %}</p>
```

Como se puede apreciar en el ejemplo anterior, el sistema permite añadir variables en las cadenas traducibles.

Por último, si el usuario tuviera que crear y manipular de forma totalmente manual los ficheros necesarios por `gettext`, esto sería de poca utilidad. Por ello se implementaron dos comandos que facilitaran esta labor al usuario:

- **makemessages**: Extrae las cadenas de traducciones y genera los ficheros .po necesarios.
- **compilemessages**: Compila los ficheros .po en .mo usados por gettext.

A continuación se puede ver un fragmento del código principar de `makemessages` encargado de extraer las traducciones de las plantillas:

```
def extract_translations(self):
    # Loop through every file in the templates path
    for root, directory, files in os.walk(config.templates_path):
        for f in files:

            # Skip the file if it doesn't have a valid template name
            if not f.endswith(tuple(constants.VALID_TEMPLATES_EXTENSIONS)):
                continue

            # Get complete file path
            file_path = os.path.join(root, f)
            relative_file_path = file_path.replace(config.base_path, '')
```

```
            with open(file_path, 'r') as content:
                # Extract every translation from file
                translations = self.env.extract_translations(content.read())

                # Insert every translation in array to handle it later
                for t in translations:
                    t = list(t)
                    t.insert(0, relative_file_path)
                    self.translations.append(t)
```

Capítulo 8

Pruebas

El diseño de casos de pruebas para un proyecto de las características de *Sitic* es algo complicado, ya que necesitamos un escenario en el que interactúen todos los elementos entre sí. Pero las pruebas son algo necesario si deseamos un software con una calidad aceptable.

Todos los módulos que componen la aplicación han sido probados individualmente, como pueden ser aquellos módulos encargados de la gestión de idiomas, generación de contenidos o categorización.

También se realizaron pruebas de integración, ya que había módulos, una vez probado en solitario, debían realizar distintas acciones junto con otros módulos.

Otras pruebas que se realizaron fueron de generación, tanto yo como personas ajenas al desarrollo de proyecto probaron el mismo ofreciendo sus opiniones sobre aspectos que deberían ser modificados o errores que aparecían a lo largo de la ejecución y en el resultado obtenido.

En definitiva, la organización de los casos de pruebas fue la siguiente:

1. Tras finalizar la implementación de cada módulo, se realizaban pruebas unitarias sobre estos.
2. A medida que distintos módulo que anteriormente probados individualmente, debían colaborar entre ellos, se llevaban a cabo pruebas de integración.
3. Con las distintas versiones usables se realizaban pruebas de generación.

8.1. Pruebas unitarias

Estas pruebas se realizaron junto a la fase de implementación, conforme se implementaban nuevos módulos para la aplicación, se realizaban pruebas individuales sobre los mismos. De esta forma se buscaban todos los caminos posibles que podría dar cada módulo, teniendo en cuenta aquellos que fueran más predispuesto a fallos.

De esta forma todas las sentencias se ejecutaban como mínimo una vez y los posibles fallos se encontraban de una forma más sencilla. Por lo que también eran más fácil localizar donde estaba el problema y afrontar la solución.

8.2. Pruebas de integración

Conforme aparecían nuevos módulos, cuya implementación era necesaria y a su vez estos requerían el uso de otro módulos que posteriormente habían sido probados individualmente, se realizaban pruebas de integración entre dichos módulos.

Conforme se avanzaba en el desarrollo, se realizaban pruebas de integración a mayor escala. No solo entre módulos del mismo sistema, sino entre varios sistemas.

Una de las mejores formas que se pensaron para realizar este tipo de pruebas, fue desarrollar dos webs que usaran todos los elementos que se iban implementando.

En concreto se crearon las dos siguientes webs:

- **Blog**: ejemplo de sitio web perfecto y sencillo, donde se pueden usar elementos como la paginación, taxonomías, comentarios, secciones y buscador. Se puede ver el resultado en la figura 8.1.
- **Documentación web**: otro ejemplo perfecto en el que se pueden ver somo se usan elementos como menús, internacionalización y buscador. Se puede ver el resultado en la figura 8.2.

Ambos sitios están disponibles en el respositorio oficial del proyecto [9] bajo la carpeta `examples`.

Durante el desarrollo se descubrieron distintos errores y se mejoraron muchos aspectos de usabilidad del proyecto, como puede serel uso de las taxonomías o la usabilidad de la funcionalidad para la paginación de secciones.

8.3. Pruebas de aceptación

Conforme se avanzaba en el desarrollo del proyecto y se conseguían versiones estables del mismo, se envíaban estas versiones, junto con el manual del proyecto, a un grupo de usuarios con los conocimientos necesarios, capaces de interpretar los requerimientos especificados por los futuros usuarios de la herramienta.

El manual se iba actualizando con las nuevas funcionalidades que se iban añadiendo. De forma que los usuarios pudieran identificar y probar correctamente todas las características de la herramienta.

Cada uno de estos usuarios devolvía un pequeño informe sobre la herramienta, en la que detallaban los distintos problemas que se habían encontrado, errores que

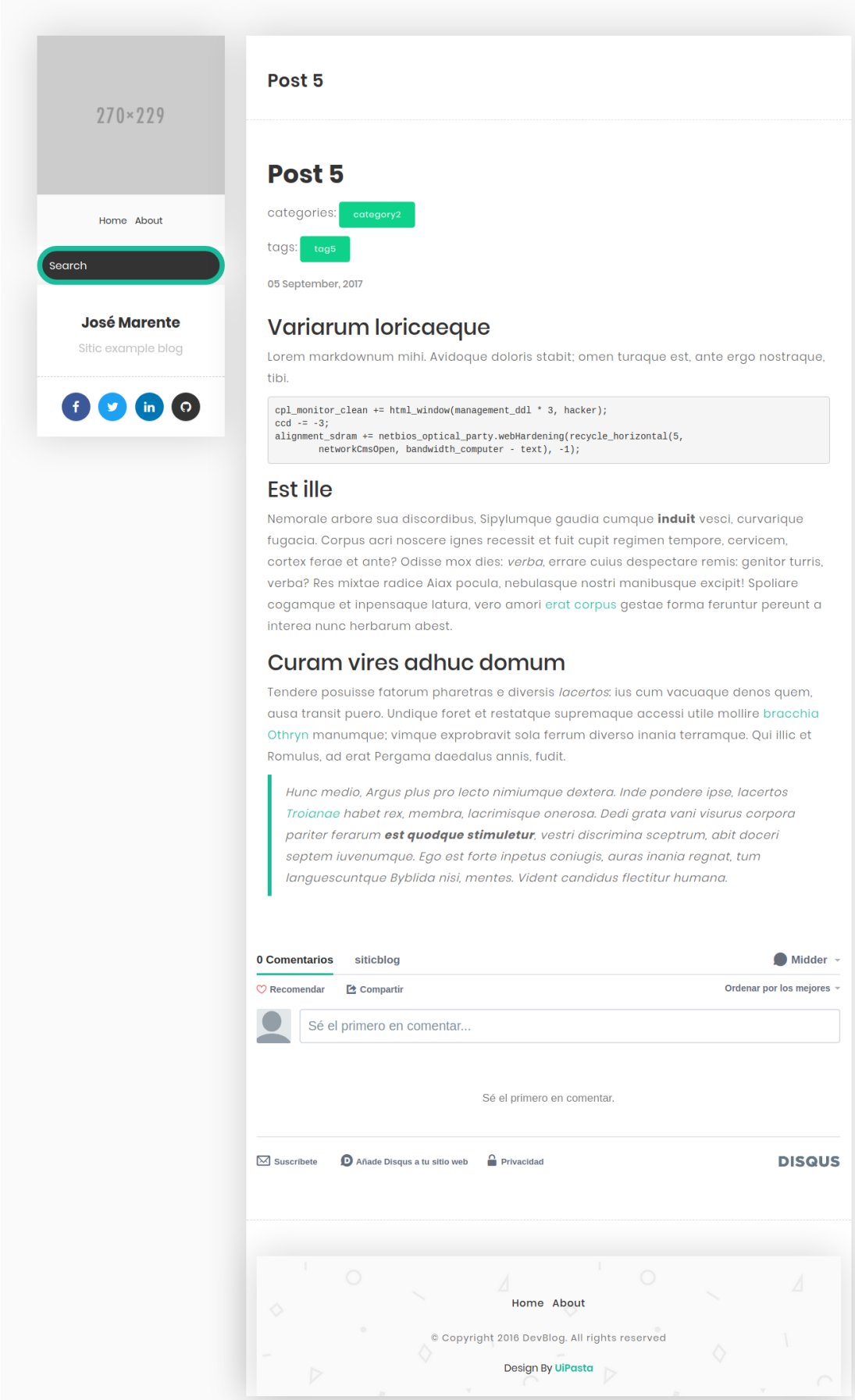

Figura 8.1: Ejemplo de blog construido

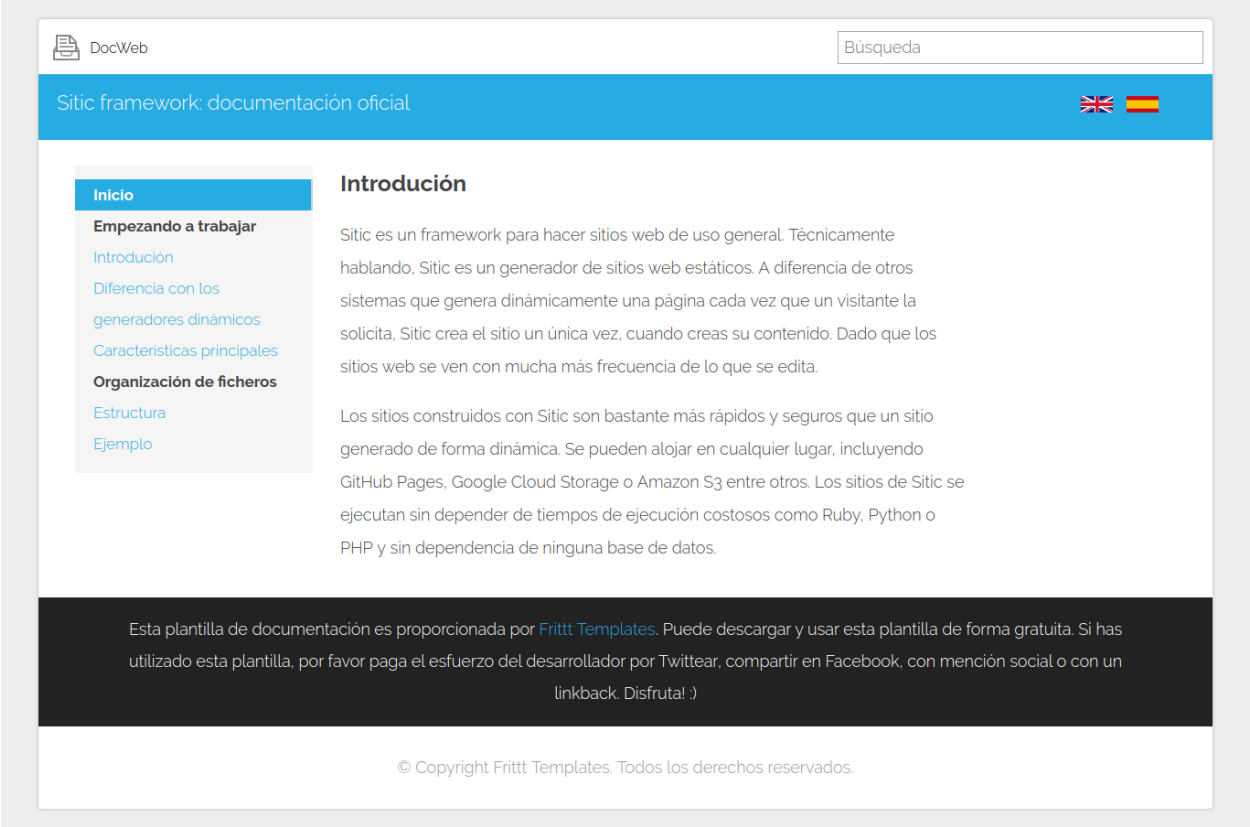

Figura 8.2: Ejemplo de documentación web construido

necesitaban ser corregidos y su opinión sobre funcionalidades, indicando cuales podría ser ampliadas para que fueran más útiles o más sencillas de usar.

Con cada uno de estos informes, se trabajaba para mejorar la herramienta en versiones posteriores, dando prioridad a aquellas que se consideraban más críticas.

Entre las principales quejas de los usuarios, se encontraban muchos aspectos referentes a los contenidos, ya que algunos usuario consideraban que se debería permitir generar contenidos sin necesidad de generar ficheros en la carpeta de contenidos, por lo que se trabajó en una solución para corregirlo.

8.4. Pruebas de generación

Cada vez que estaban disponibles nuevas versiones estables de la herramienta, se pedían la ayuda a personas, totalmente ajenas al desarrollo, que probaran las distintas versiones disponibles. De esta forma cada uno de los colaboradores daba su opinión sobre distintos aspectos como la usabilidad, dificultad, respuesta o eficiencia. Tras recopilar la información que todos ellos proporcionaron, se procedía a realizar los ajustes necesarios a los distintos parámetros requeridos.

Entre los distintos aspectos a probar que se le recomendaban a los colaboradores

que hicieran especial hincapié, son los siguiente:

- **Internacionalización**: definición de varios idiomas con contenidos divididos, así como pruebas de generar los mismos contenidos sin ningún idioma configurado.
- **Urls de ficheros**: definir distintas urls en los ficheros, comprobando que la generación se la ruta se generaba de forma correcta, en función de la rutas definidas
- **Paginación**: configurar distintos valores para la paginación de elementos por página, así como la desactivación del mismo, comprobando que se generaban tantas páginas estáticas por sección/taxonomías como páginas debería de tener.
- **Menús**: definición de menús con elementos estáticas y dinámicos.
- **Categorización**: definir distintas taxonomías y mezclarlas entre los idiomas, comprobando finalmente que cada contenido se había organizado de la forma correcta.

Capítulo 9

Conclusiones

Durante el transcurso del desarrollo de proyecto y sobre todo al término del mismo, se han obtenido unas conclusiones y unos resultados, tanto de forma personal como para con la comunidad, que intentaremos reflejar en este capítulo.

9.1. Resumen de objetivos

Es evidente que su realización no me ha dejado indiferente. No ha sido fácil construir una idea clara sobre lo que se quería hacer. Así como solucionar los distintos problemas que han ido apareciendo a lo largo del desarrollo.

También decir que el proyecto me ha ocupado bastante más tiempo del esperado en un principio. Tuve muchos problemas y alguna que otra duda en algunas fases de desarrollo, que me tuvieron bloqueado durante un tiempo hasta encontrar la solución más adecuada para estos. A pesar de todo, estoy muy satisfecho con el resultado final.

Se puede decir que el proyecto goza de buena calidad. Se ha intentado hacer un software sencillo, intuitivo, fácil de usar y entretenido para el usuario.

9.2. Conclusiones personales

Durante el desarrollo del proyecto se han aprendido muchísimas cosas: como hacer distintas ramas de desarrollo, plantear y crear calendarios, usar las herramientas adecuadas, tomar decisiones importantes durante el desarrollo, documentación del código, organización, etc. Ya que durante la carrera se han realizado distintas prácticas y trabajos de complejidad, pero nada con el tamaño y duración que requiere un Proyecto de fin de carrera. Una vez finalizado, creo que tengo la experiencia necesaria para afrontar otro proyecto con buenos resultados.

Puedo decir que he profundizado y consolidado bastante en el lenguaje de programación Python. Además he obtenido más práctica a la hora de manejar bibliotecas externas, así como entender su documentación e integrarlas en un proyecto.

En definitiva, este proyecto me ha hecho madurar como persona, estudiante y profesional. He aprendido a buscar bibliografía, opiniones en otras personas, compartir ideas, seguir un horario, cumplir una fechas de entrega y enfrentarme a un proyecto de estas características.

9.3. Mejoras y ampliaciones

Las posibles mejoras y ampliaciones que se podrían añadir al proyecto en futuras versiones, se comentan a continuación:

- **Soporte Filtros/funciones propias** de forma que cada usuario puede personalizar aún mas funciones a usar en las plantillas.
- **Método watch que recargue automáticamente el navegador**: permitir que el modo watch recargue automáticamente el navegador cuando se detecte algún cambio.
- **Permitir temas**: permitir la creación de temas, es decir, poder usar temas definidos por otros usuarios y únicamente preocuparte por la redacción del contenido.
- **Permitir fichero con contenido para taxonomía**: permitir definir metadatos en las taxonomías tal y como se puede hacer en los contenidos y secciones.
- **Permitir publicar un solo idioma**: no verse obligado a generar el sitio completo y podre generar un único idioma o página.

Apéndice A

Manual de instalación

A continuación se presentan las instrucciones de instalación de la herramienta.

Antes de continuar se presentan los requisitos de hardware y software mínimos para la correcta ejecución de la plataforma web.

Hardware

- 512MiB de memoria RAM como mínimo.
- 10GiB de disco duro como mínimo.
- Acceso a Internet.

Software

- Sistema operativo **GNU/Linux**, preferiblemente basado en paquetería Debian.
- Intérprete de **Python**, versión mínima 2.7 o 3.5.
- Soporte de entornos virtuales **VirtualEnv** para la encapsulación de dependencias.

A.1. Instalación

A.1.1. Descarga de código

Para instalar la plataforma web es necesario clonar el repositorio [9] con el código de la aplicación. Primero, creamos una ubicación donde alojar el código desde la terminal:

```
$ mkdir ~/sitic -p
$ cd ~/sitic
```

Una vez ahí, instalamos el software de control de versiones **git** y clonamos el repositorio desde Github:

```
$ sudo apt-get install git
$ git clone https://github.com/josemarente/sitic.git .--
```

Instalación de dependencias

Como se ha comentado previamente, el proyecto utiliza **Virtualenv** y **Virtualenvwrapper** para una gestión más limpia de las dependencias. La instalación de estos dos elementos es sencilla, en primer lugar se instala el gestor de paquetes y los dos paquetes mencionados:

```
$ sudo apt-get install python-pip
$ sudo pip install virtualenv
$ sudo pip install virtualenvwrapper
```

En segundo lugar, añadimos el código de *bootstrapping* de Virtualenvwrapper a nuestro perfil de terminal, habitualmente `.bashrc`:

```
$ cat >> ~/.bashrc

if [ -f /usr/local/bin/virtualenvwrapper.sh ]
then
source /usr/local/bin/virtualenvwrapper.sh
fi

EOF
```

Hecho esto, será necesario reiniciar la terminal o volver a cargar el perfil del a terminal, para el caso de `.bashrc`, se puede hacer de la siguiente forma:

```
$ source ~/.bashrc
```

Tras esto, podremos crear el entorno virtual e instalar las dependencias:

```
$ mkvirtualenv sitic
$ sudo apt-get install python-dev
$ pip install -r requirements.txt
```

El proceso de instalación de dependencias puede tardar entre 5 y 10 minutos aproximadamente.

A.1.2. Comando sitic

Por último sólo quedaría que sitic estuviera disponible en el sistema de forma global, es decir, que podamos ejecutar su comando sin necesidad de en que ruta está y sin necesidad de poner la ruta completa para lanzarlos, para ello bastaría con hacer un enlace simbólico del ejecutable de sitic en `/usr/local/bin` de la siguiente manera:

```
$ ln -s ~/sitic/main.py /usr/local/bin/sitic
```

A.2. Pruebas de implantación

Para probar que la instalación es correcta, lo ideal es que pruebe la aplicación a fondo, revisando toda la funcionalidad. El itinerario que se recomienda es el siguiente. Si tiene dudas sobre cómo realizar alguna de las acciones descritas, revise el Manual de usuario.

El primer paso antes de nada es lanzar la batería de tests que viene incluida en el proyecto, mediante el comando

```
$ python tests.py test
```

Todos los tests deberían funcionar correctamente. Si no lo hacen, es que ha habido un fallo en el sistema o en la aplicación.

Tras esto, el código clonado dispone de varios ejemplos implementados usando la herramienta, por lo que bastaría con ir a a la carpeta `examples` y ejecutar sitic en cualquier de los ejemplos disponibles. Los comandos para este caso serían los siguientes:

```
$ workon sitic
$ cd ~/sitic/examples/blog
$ sitic
```

En el caso de que la generación del sitio de ejemplo no se llevara a cabo correctamente, es que no se habrá realizado adecuadamente alguno de los pasos de instalación.

Apéndice B

Manual de usuario

En este capítulo se explicarán en detalle las instrucciones de uso de la herramienta.

B.1. Introducción a Sitic

Sitic es un *framework* para hacer sitios web de uso general. Técnicamente hablando, Sitic es un generador de sitios web estáticos. A diferencia de otros sistemas que genera dinámicamente una página cada vez que un visitante la solicita, Sitic crea el sitio un única vez, cuando creas su contenido. Dado que los sitios web se ven con mucha más frecuencia de lo que se edita.

Los sitios construidos con Sitic son bastante más rápidos y seguros que un sitio generado de forma dinámica. Se pueden alojar en cualquier lugar, incluyendo GitHub Pages, Google Cloud Storage o Amazon S3 entre otros. Los sitios de Sitic se ejecutan sin depender de tiempos de ejecución costosos como Ruby, Python o PHP y sin dependencia de ninguna base de datos.

B.1.1. En qué se diferencia con los generadores dinámicos

Los generadores de sitios web generan contenidos en ficheros HTML. La mayoría son "generadores dinámicos". Esto significa que el servidor HTTP (que es el programa que se ejecuta en su sitio web con el que el navegador del usuario habla) ejecuta el generador para crear un nuevo fichero HTML cada vez que un usuario desea ver una página.

Crear la página de forma dinámica significa que la máquina que aloja el servidor HTTP tiene que tener suficiente memoria y CPU para ejecutar el generador de forma eficaz durante todo el día. Si no, entonces el usuario tiene que esperar a que la página se genere.

Nadie quiere que los usuarios esperen más de lo necesario, por lo que los generadores de sitios dinámicos programaron sus sistemas para almacenar en caché los

ficheros HTML. Cuando un fichero se almacena en caché, una copia se almacena temporalmente en el equipo. Es mucho más rápido que el servidor envíe esa copia la próxima vez que se solicite la página en lugar generarla desde cero.

Sitic intenta llevar el almacenamiento en caché un paso más allá. Todos los ficheros HTML se representan en su máquina. Puede revisar los ficheros antes de copiarlos en la máquina que aloja el servidor HTTP. Dado que los ficheros HTML no se generan dinámicamente, decimos que Sitic es un "generador estático".

No tener que ejecutar la generación de HTML cada vez que se recibe una petición tiene varias ventajas. Entre ellas, la más notable es el rendimiento, los servidores HTTP son muy buenos en el envío de ficheros. Tan bueno que puede servir eficazmente el mismo número de páginas con una fracción de memoria y CPU necesaria para un sitio dinámico.

Sitic tiene dos componentes para ayudarle a construir y probar su sitio web. El que probablemente usará más a menudo es el servidor HTTP incorporado. Cuando ejecuta el servidor, Sitic procesa todo su contenido en ficheros HTML y luego ejecuta un servidor HTTP en su máquina para que pueda ver cómo son las páginas.

El segundo componente se utiliza una vez que el sitio esté listo para ser publicado. Ejecutar Sitic sin ninguna acción reconstruirá su sitio web completo utilizando la configuración `base_url` del fichero de configuración de su sitio. Eso es necesario para que sus enlaces de página funcionen correctamente con la mayoría de las empresas de alojamiento.

B.1.2. Características principales

En términos técnicos, Sitic toma un directorio fuente de ficheros y plantillas y los usa como entrada para crear un sitio web completo.

Sitic cuenta con las siguientes características:

General

- Tiempos de generación rápidos
- Fácil instalación
- Posibilidad de alojar su sitio en cualquier lugar

Organización

- Organización sencilla
- Soporte para secciones
- URL personalizables
- Soporte para taxonomías configurables que incluyen categorías y etiquetas.

- Capacidad de clasificar el contenido como usted desea
- Generación automática de tabla de contenido
- Creación dinámica de menú
- Soporte de URLs legibles

Contenido

- Soporte nativo para contenido escrito en Markdown, Textile y Reestructured text
- Soporte internacionalización
- Soporte para los metadatos TOML y YAML en los contenidos
- Páginas completamente personalizables

B.2. Estructura de ficheros

Sitic toma un directorio y lo usa como entrada para crear una página web completa.

El nivel más alto del directorio principal tendrá los siguientes elementos:

```
+--content/
+--data/
+--locales/
+--templates/
+--static/
+--sitic.yml
```

El proposito para cada fichero/directorio se decribe a continuación:

- **content**: Aquí es donde se almacenan los contenidos de la web, se crearán sub-directorios para crear las distintas secciones de la web. Supongamos, que nuestra web tiene cuatro secciones: `blog`, `news`, `about` y `contact`, entonces será necesario crear una carpeta para cada una de ellas.
- **data**: Este directorio contiene distintos ficheros de configuración que pueden ser usado mientras se genera la web. El contenido de estos ficheros puede estar en format YAML, JSON o TOML.
- **locales**: Ficheros con las traducciones de las cadenas usadas en las plantillas.
- **templates**: Los contenidos dentro de este directorio especifican como se convertirán los contenidos en una web estática.
- **static**: Directorio usado almacenar todos los contenidos estáticos que la web necesitará como imágenes, CSS, Javascript u otro tipo de contenido estático.

- **sitic.yml**: Todo proyecto hecho con Sitic debe de tener un fichero de configuración en la raíz del proyecto. Este debe de tener el nombre `sitic.yml`, usando el formato YAML [25]. Esta configuración se aplica a todo el siti completo, que incluye la `base_url` y `title` de la página web.

B.2.1. Ejemplo

Un ejemplo de completo tendría el siguiente aspecto:

```
+-- content
|   +-- about.md
|   +-- blog
|       +-- post1.md
|       +-- post2.md
|       +-- post3.textile
|       +-- post4.rst
+-- locales
+-- dara
+-- sitic.yml
+-- static
|   +-- css
|   +-- fonts
|   +-- images
|   +-- js
+-- templates
    +-- base.html
    +-- default
    |   +-- list.html
    |   +-- page.html
    +-- section
        +-- about.html
```

B.3. Configuración

La estructura de directorios de un sitio web de Sitic, o más exactamente de los ficheros de origen que contienen su contenido y plantillas, proporciona la mayor parte de la información de configuración que Sitic necesita. Por lo tanto, en esencia, muchos sitios web realmente no necesitan un fichero de configuración. Esto se debe a que Sitic está diseñado para reconocer ciertos patrones de uso típicos (y los espera por defecto).

Sin embargo, Sitic busca un fichero de configuración con un nombre en particular en la raíz del directorio de su sitio web. El fichero en concreto debe tener el nombre `./sitic.yaml`.

En este fichero de configuración puede incluir las instrucciones necesarias de cómo se debe procesar su sitio, así como definir sus menús y establecer otras opciones.

B.3.1. Ejemplos

A continuación se muestra el fichero de configuración básico:

```
base_url: "http://Sitic.example.com/"
```

Seguidamente podemos un fichero un poco más completo con distintos elementos:

```
base_url: "www.Sitic.net"
paginable: "1"
main_language: "en"

lazy_menu: "main"

sitemap:
    change_frequency: "monthly"
    priority: 0.5

menus:
    main:
        - title: "title test"
          url: "/test-url"
          id: "test1"
        - title: "title test2"
          url: "/test-url2"
          id: "test2-custom"
          parent: "test1"
    footer:
        - title: "title footer"
          url: "/test-url-footer"
          id: "test-footer"


languages:
    en:
    es:
        menus:
            footer:
                - title: "title footer"
                  url: "/test-url-footer"
                  id: "test-footer"
```

En este último ejemplo se pueden ver distintas configuraciones, como la configuración de menús o idiomas, de las que hablaremos mas en detalle en las siguientes

secciones.

B.4. Contenidos

B.4.1. Organización

El contenido debe estar organizado de la misma manera que se pretende que aparezca en la web. Sin ningún tipo de configuración adicional, a continuación se puede apreciar un ejemplo sencillo:

```
content/
+-- about.md            // http://example.com/about
+-- blog                // http://example.com/blog/
    +-- post1.md        // http://example.com/blog/post1
    +-- post2.md        // http://example.com/blog/post2
    +-- post3.textile   // http://example.com/blog/post3
    +-- post3.rst       // http://example.com/blog/post4
```

Como se puede apreciar, Sitic usará la ruta definida en en el disco a partir del directorio `content`, para definir la url del contenido, esto se puede sobreescribir usando el atributo `url` en los metadatos de los contenidos.

Se puede usar el nivel de directorios que se desee, Sitic tomará siempre como `sección` los directorios que se encuentren en el primer nivel.

B.4.2. Formatos soportados

Sitic soporta los siguientes formatos para los contenidos:

- **Markdown**: Se identificarán aquellos ficheros con la extensión `.md`.
- **reStructuredText**: ficheros con la extensión `.textile`
- **Textile**: ficheros con cualquier de las extensiones `.txt` `.rst` `.rest` `.restx`

B.4.3. Metadatos

Sitic usa ficheros con encabezados para definir los metadatos, comúnmente llamados `front matter`. Sitic usa la organización proporcionada para su contenido para minimizar cualquier configuración adicional, aunque se puede sobrescribir en los metadatos.

Estos metadatos se identificarán a partir de unos caracteres delimitadores, los formatos soportados son:

- **TOML**: identificado por +++.

- **YAML**: identificado por - - -.

Ejemplo de TOML:

```
+++
title='Post 2'
description='Post 2 description'
tags=['tag1', 'tag2', 'tag3']
+++

El contenido va aquí
```

Ejemplo de YAML:

```
---
title: 'Post 2'
description: 'Post 2 description'
tags: ['tag1', 'tag2', 'tag3']
---

El contenido va aquí
```

B.4.4. Secciones

Sitic está hecho de forma que la organización su contenido tenga un propósito. La estructura usada para organizar el contenido de origen se utiliza para organizar el sitio generado. Siguiendo este patrón Sitic utiliza el nivel superior de su organización de contenido como la Sección.

Sitic creará automáticamente páginas para cada raíz de sección que enumere todo el contenido de esa sección. Consulte la sección dedicada a las plantillas para obtener detalles sobre la personalización de la forma en que aparecen.

Las páginas de sección también pueden tener un fichero de contenido con metadatos. Para ello basta con definir un fichero con el nombre `index.md`, se tomará como el contenido propio de la sección en lugar de un contenido de la sección.

Secciones como ficheros

Puede existir ocasiones en las que queramos tener una sección que carecerá de contenidos adicionales, es decir, que no tendrá otros contenidos relacionados que se listarán dentro de la página de la sección.

En ese caso, cualquier fichero definido en la raíz del directorio de contenidos, se tomará como una sección en la que se podrá definir directamente el contenido y metadatos sin necesidad de seguir el proceso de creación de carpetas y fichero `index.md`.

B.4.5. Taxonomías

Sitic también ofrece la posibilidad del uso de taxonomías, por defecto existen dos tipos de taxonomías, las `tags` y `categories`. Pero Sitic también ofrece al usuario la opción de configurar nuevas taxonomías desde el fichero general de configuración.

B.4.6. Definiendo nuevas taxonomías

Para configurar nuevas taxonomías debemos hacerlo, añadiendo un nuevo elemento `taxonomies` a la configuración principal, junto con las definiciones en plural y singular de las nuevas taxonomías, un ejemplo, sería el siguiente:

```
taxonomies:
    tag: "tags"
    category: "categories"
    series: "series"
```

B.4.7. Relacionando taxonomías y contenidos

Una vez que las taxonomías ya están definidas en la configuración, la forma en la que se deben de relacionar con los contenidos, es a través de los metadatos, creando una variable con el nombre plural definido de la taxonomía, seguido de cada uno de los términos de la taxonomía a la que pertenece el contenido.

A continuación se muestra un ejemplo usando las taxonomías `tags` y `categories`, en un fichero con los metadatos en formato `TOML`:

```
+++
title='Post 2'
description='Post 2 description'
tags=['tag1', 'tag2', 'tag3']
categories=['cat1', 'cat3']
+++

El contenido va aquí
```

B.4.8. Página inicial

La página inicial de una web, suele ser de un aspecto distinto al del resto de secciones o contenidos de la web, para añadir un fichero con contenidos y metadatos, a la página inicial de un sitio construido con Sitic, simplemente tendremos que añadir un fichero `index.md` en la raíz de la carpeta de contenidos.

B.5. Plantillas

Sitic usa la librería `Jinja2` [8] como motor de plantillas. Tiene todas las funcionalidades necesarias para construir plantillas de forma rápida y personalizable.

Si nunca has usado `Jinja2` anteriormente, te recomiendo visitar si página oficinal [8], donde podrás ver todas las funcionalidades que ofrece y todas disponibles en Sitic.

Entre otras funcionalidades, ofrece la posibilidad de extender plantillas reduciendo así el número de código duplicado entre plantillas.

B.5.1. Tipos de plantillas

Existen tres tipos de plantillas que se definen a continuación:

- **Page**: Plantilla para renderizar un único contenido.
- **List**: Plantilla para listar contenidos, como pueden ser las secciones y las taxonomías.
- **Index**: La plantilla para la página inicial de tu web.

B.5.2. Ruta de plantillas

Como se ha comentado anteriormente, el directorio `templates`, es donde se deben de añadir todas las plantillas. Por defecto un Sitio hecho con Sitic debe de tener un directorio `default`, con las siguientes plantillas:

- **page.html**: Plantilla por defecto para todas las páginas.
- **list.html**: Plantilla por defecto para secciones, taxonomía y homepage.

Pero Sitic te ofrece la posibilidad de personalizar las plantillas en función del tipo que sean.

Page

La plantilla para un contenido normal puede estar en cualquier de las siguientes rutas. Sitic buscará la plantilla en el orden que se muestra a continuación:

- /templates/`TYPE`/`TEMPLATE`.html
- /templates/`SECTION`/`TEMPLATE`.html
- /templates/`TYPE`/page.html
- /templates/`SECTION`/page.html

- /templates/default/page.html

Tanto `TYPE` como `TEMPLATE`, son los valores que se definan en los metadatos del contenido en variable con el mismo nombre, totalmente en minúsculas.

En el caso de `SECTION`, se usa el nombre de la sección a la que pertenezca el contenido.

Sección

Para las secciones las rutas que Sitic busca son las siguientes:

- /templates/section/`SECTION`.html
- /templates/`SECTION`/section.html
- /templates/`SECTION`/list.html
- /templates/default/section.html
- /templates/default/list.html

Taxonomía

En el caso de las taxonomias:

- /templates/taxonomy/`SINGULAR`.html
- /templates/taxonomy/list.html
- /templates/default/taxonomy.html
- /templates/default/list.html

Homepage

Por último en la home:

- /templates/index.html
- /templates/default/list.html
- /templates/default/page.html

B.5.3. Elementos disponibles en las plantillas

Cada plantilla dispondrá de una serie de elementos en función del tipo de plantilla de la que estemos hablando.

Todas las plantillas tendrán acceso a la variable `site` en la que estarán todos los parámetros presentes en la configuración del sitio y además tendrá acceso a la variable node, que representará el nodo que se está renderizando, ya sea una página, sección o taxonomía. En este elementos están disponibles todo los atributos de dicho elemento.

B.5.4. Ejemplo

A continuación se muestra una seria de ejemplo de uso de plantillas.

En primer lugar podemos ver la plantilla base de una web, con el nombre *base.html*:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- Meta Tag -->
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Personal Blog Template</title>

    <link rel="stylesheet" type="text/css" href="/static/css/style.css">
  </head>
  <body>

    <div id="main">
      <div class="container">
          <h2>
            {% if node.title %}
                {{ node.title }}
            {% else %}
                My Blog
            {% endif %}
          </h2>
        </div>
        <div class="col-md-12 content-page">
          {% block content %}
          {% endblock %}
        </div>
      </div>
    </div>

    <script type="text/javascript" src="/static/js/scripts.js"></script>
  </body>
```

```
</html>
```

Se puede apreciar en el documento anterior que es un documento HTML con ciertos peculiaridades del sistema de plantillas *Jinja2*.

En concreto podemos ver como se usan estructuras de control y como se define el bloque donde irá el contenido, concretamente:

```
<div class="container">
    <h2>
      {% if node.title %}
          {{ node.title }}
      {% else %}
          My Blog
      {% endif %}
    </h2>
  </div>
  <div class="col-md-12 content-page">
    {% block content %}
    {% endblock %}
  </div>
</div>
```

A continuación podemos ver la plantilla de un contenido, extendiendo de la plantilla base y definiendo el contenido necesario, así como haciendo uso de las taxonomías.

```
{% extends 'base.html' %}

{% block content %}
<div class="col-md-12 blog-post">

  <div class="post-title">
    <h1>{{ node.title }}</h1>
  </div>

  <div class="post-info">
    <span>
        {{ node.publication_date | datetimeformat("%d %B, %Y") }}
        {% if node.author %}
            / by <a href="#" target="_blank">{{ node.author }}</a>
        {% endif %}
    </span>
  </div>

  {{ node.content }}
</div>
{% endblock %}
```

Podemos ver que ya no es necesario volver a definir el HTML completo para cada uno de los contenidos, sino que podemos extender de algunas de las plantillas ya existentes y definido qué bloques vamos a completar.

En este caso se extiende de la plantilla base.html mostrada anteriormente y se define el contenido que tendrá el bloque denominado content, a su vez se accede a los distintos metadatos del contenido en este caso a los campos title, author, publication_date y content, este último está compuesto del contenido de la página en HTML.

B.6. Internacionalización

Sitic ofrece la posibilidad de tener una web multi-idioma, haciendo uso de la poderosa herramienta gettext [5], que Python soporta de base.

B.6.1. Configuración

Para que Sitic sepa que el sitio contendrá más de un idioma, es necesario configurarlo, a continuación podemos ver un ejemplo:

```
main_language: 'en'

languages:
    en:
        name: 'English'
    es:
        name: 'Español'
```

De esta forma configuramos dos idiomas, indicándole a Sitic que el idioma inglés, es el idioma principal de la web. En el caso de que no se especificara qué idioma es el principal, Sitic tomaría cualquier de los configurados como principal.

B.6.2. Contenido

A la hora de escribir contenidos en distinto idiomas, sólo es necesario añadir el idioma al que pertenece el contenido en al nombre de éste, justo antes de la extensión del fichero. En el caso de que no se indique el idioma, Sitic dará por hecho de que el contenido pertenece al idioma principal configurado.

En el siguiente ejemplo, se puede ver un ejemplo con una sección con contenidos en inglés y español:

```
content/
+-- section
|   +-- content1.es.md
```

```
+   +-- content1.md
```

B.6.3. Plantillas

Para ver como traducir las cadenas presentes en las plantillas, los mejor es visitar la página oficial de Jinja2 [8], donde se explican los distintos filtros disponibles.

Aunque la forma más común de traducir texto en las plantillas se realiza usando el filtro `trans`, se puede ver un ejemplo a continuación:

```
<h1>{% trans %}Texto a traducir{% endtrans %}</h1>
```

B.6.4. Generación y gestión

Sitic ofrece la posibilidad de generar los ficheros .po y .mo, usado por gettext para realizar la traducción de un sitio. Para ello ofrece dos comandos.

makemessages Este comando buscará todas las cadenas traducibles definidas en las plantillas y generará los ficheros .po en la carpeta locales.

```
$ sitic makemessages
```

Un ejemplo de la carpeta locales generada por sitic:

```
locales/
+-- en
|   +-- LC_MESSAGES
|       +-- sitic.po
+-- es
|   +-- LC_MESSAGES
+       +-- sitic.po
```

compilemessages Una vez que hayamos completado los ficheros .po con las traducciones necesarias, es necesario compilar dichos ficheros, para que gettext pueda obtener las traducciones correctamente. El comando para generar los ficheros .mo necesario es:

```
$ sitic compilemessages
```

B.7. Buscador

Sitic trae de base un buscador de forma que sea sencillo encontrar cualquier contenido del sitio, bastante útil en los casos que creemos webs con muchos contenidos.

Para que el buscador funcione correctamente hay que realizar los siguientes pasos:

- Añadir un formulario con las clases necesarias
- Crear la plantilla en la que el buscador mostrará los resultados

B.7.1. Añadir formulario

Como es obvio, es necesario añadir un formulario en el que realizar las busquedas.

El formulario debe de enviarse como `GET` y como `action` debe de tener la url del buscador, que podemos obtener usando la función `get_search_url`.

Además deberá de tener un `input` con el valor `sitic-search-input` como id.

A continuación se puede ver un ejemplo del como sería el HTML necesario de un buscador válido.

```
<form method="get" action="{{ get_search_url() }}" novalidate="true">
  <input id="sitic-search-input" type="text" name="search">
</form>
```

B.7.2. Plantilla del buscador

También necesitaremos que se genere una url donde se muestren los resultado de la búsqueda.

En este caso tendremos que añadir la plantilla `search.html` en el directorio de plantillas `templates`.

Esta plantilla deberá de tener los siguientes elementos:

- Elemento `ul` con la clase `sitic-search-results` en la que se mostrarán los resultados.
- Cualquier tipo de contenedor con la clase `sitic-search-no-results` con el mensaje que queramos mostrar en el caso de que no se encuentre ningún resultado.
- Dos elementos para controlar la paginación, con las clases `sitic-search-previous-page` y `sitic-search-next-page`

Un ejemplo del contenido de esta plantilla, recopilando todo lo anterior sería el siguiente:

```
{% extends 'base.html' %}

{% block content %}
<ul id="sitic-search-results">
</ul>
```

```
<p id="sitic-search-no-results">No results found</p>

<div class="col-md-12">
  <nav aria-label="...">
    <ul class="pager">
      <li id="sitic-search-previous-page"><a>Previous</a></li>
      <li id="sitic-search-next-page"><a>Next</a></li>
    </ul>
  </nav>
</div>
{% endblock %}
```

B.8. Menús

Es posible gestionar los menús que queramos que estén disponibles en la web y qué contenidos queremos que aparezcan en ciertos menús.

B.8.1. Asignación en contenidos

Para definir qué contenidos estén en ciertos menús, basta con añadir en los metadatos del contenido el atributo `menus` en el que indicaremos los menús en los que deberá aparecer los contenidos.

Ejemplo:

```
+++
title='Post 1'
description='Post 1 description'
menus=['main', 'footer']
weight=3
+++

El contenido va aquí
```

En el ejemplo, el contenido se asignará a los menús `mainy` `footer`.

B.8.2. Elementos personalizados

También tenemos la posibilidad de añadir elementos a los menús de forma manual, sin necesidad de que tenga que ser contenidos de la web, para ello deberemos de indicar los elementos del menú en el fichero de configuración principal del sitio.

Ejemplo:

```
menus:
    main:
        - title: "title test"
          url: "/test-url"
          id: "test1"
        - title: "title test2"
          url: "/test-url2"
          id: "test2-custom"
          parent: "test1"
    footer:
        - title: "title footer"
          url: "/test-url-footer"
          id: "test-footer"
```

Como podemos ver, hemos definidos ciertos elementos el los menús `main` y `footer`, indicando el título, la url e identificador de los elementos. Con el atributo `parent`, indicaremos que ese elemento del menú será hijo del elemento con el id especificado.

B.8.3. Menú automático

Otra opción es dejar que Sitic genere un menú de forma totalmente automática a partir de todos los contenidos del sitio. Para ello basta con establecer la opción `lazy_menu` en la configuración principal, especificando qué menú se completará de forma automática.

Ejemplo:

```
lazy_menu: "main"
```

B.8.4. Usando menús en las plantillas

Por último, podremos acceder a todos los menús definidos desde las plantillas, accediendo al atributo `menus` de la variable `site`.

```
<ul>
{% for menu in site.menus.main %}
<li><strong>{{ menu.title }}</strong></li>
  {% if menu.has_children %}
    {% for child in menu.children %}
    <li><a href="{{ child.url }}">{{ child.title }}</a></li>
    {% endfor %}
  {% endif %}
{% endfor %}
</ul>
```

Como se ve en el ejemplo, podemos comprobar si un menú tiene más elementos y también recorrerlos.

B.9. Comandos

En esta sección se indican los distintos comandos disponibles en sitic, aparte de los ya comentados en secciones anteriores.

watch Con el objetivo de no tener que estar generando el sitio cada vez que se produce algún cambio en algunos de los fichero que compone la web, está disponible el comando `watch`, que constantemente está comprobando si existe algún cambio en algún fichero para regenerar el sitio.

```
$ sitic watch
```

servidor Con el siguiente comando, Sitic levanta un servidor local en el puerto que se desee, siendo por defecto el 8000, de forma que el usuario pueda probar el funcionamiente de su web sin necesidad de instalar y configurar ningún otro software:

```
$ sitic server
```

Este comando a su vez incluye la funcionalidad del comando `watch` anteriormente comentado. Este comando está pensado para usarse durante el proceso de desarrollo de un sitio.

Apéndice C

GNU Free Documentation License

Version 1.3, 3 November 2008

http://fsf.org/

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited

in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated

HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as

long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the

public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover

Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works

permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your

rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your

documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with . . . Texts.” line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Bibliografía y referencias

[1] argparse, parser for command-line options, arguments and sub-commands. https://docs.python.org/3/library/argparse.html.

[2] Chameleon, documentation. https://chameleon.readthedocs.io/en/latest/.

[3] Click documentation. http://click.pocoo.org/5/.

[4] draw.io, a free online diagram drawing application. http://draw.io.

[5] Gettext. https://www.gnu.org/software/gettext/.

[6] Git, a distributed version control system. http://git-scm.com/.

[7] Hugo, a static site generator. https://gohugo.io/.

[8] Jinja2, offcial website. http://jinja.pocoo.org/docs/2.9/.

[9] Josemarente's github - repositorio de sitic. https://github.com/jmarente/sitic.

[10] Json, acrónimo de javascript object notation. http://www.json.org/json-es.html.

[11] Markdown, wikipedia. https://en.wikipedia.org/wiki/Markdown.

[12] pip, a tool for installing python packages. http://www.pip-installer.org/.

[13] Python-fire, github. https://github.com/google/python-fire.

[14] Python, the official website. https://www.python.org/.

[15] restructuredtext documentación. http://docutils.sourceforge.net/rst.html.

[16] Singleton pattern. https://en.wikipedia.org/wiki/Singleton_pattern.

[17] Six: Python 2 and 3 compatibility library. https://pythonhosted.org/six/.

[18] Textile documentation. https://txstyle.org/.

[19] Tom's obvious, minimal language. https://github.com/toml-lang/toml.

[20] Vim, offical website. http://www.vim.org/.

[21] Virtualenv, a tool to create isolated python environments. http://www.virtualenv.org/.

[22] Virtualenvwrapper, a set of extensions to virtualenv. http://virtualenvwrapper.readthedocs.org/.

[23] Watchdog documentation. http://pythonhosted.org/watchdog/.

[24] Wikitexto, wikipedia. https://es.wikipedia.org/wiki/Wikitexto.

[25] Yaml, the official website. http://yaml.org/.