Sitic: framework para generar páginas web estáticas

Alumno: José Jesús Marente Florín
Tutor: José Miguel Mota Macías

Septiembre de 2017

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 templates que permite al usuario personalizar la forma en la que se mostrarán todos los contenidos. También puede ser ampliada, añadiendo filtros y funciones propias que se puedan usar posteriormente en las templates durante la generación de contenido. 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 [4] y usará como base el sistema de templates Jinja [2].

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

1. Introducción

1.1. Contexto y 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 necesidades 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 [1], 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. Planificación

El proyecto se ha desarrollado siguiendo un calendario basado en fases, utilizando un modelo de desarrollo iterativo incremental.

1.3.1. 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 que 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 lenguage Python por su documentación, comunidad y amplia biblioteca estándar.

1.3.2. 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.

1.3.3. 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 posible 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 lenguaje de marcado. Invertí tiempo en ver cuales eran los más usados, para así poder darles soporte en la herramienta.

1.3.4. 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 html 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 dedico tiempo a mejorar las ya existentes, así como a implementar nuevas.

1.3.5. 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 pudiera darse, hasta obtener el comportamiento esperado.

1.3.6. Redacción de la memoria

La redacción de la memoria se ha redactado 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.

1.3.7. Diagrama de Gantt

Diagrama de Gantt de la planificación comentada (Figura 1).

2. 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 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, en

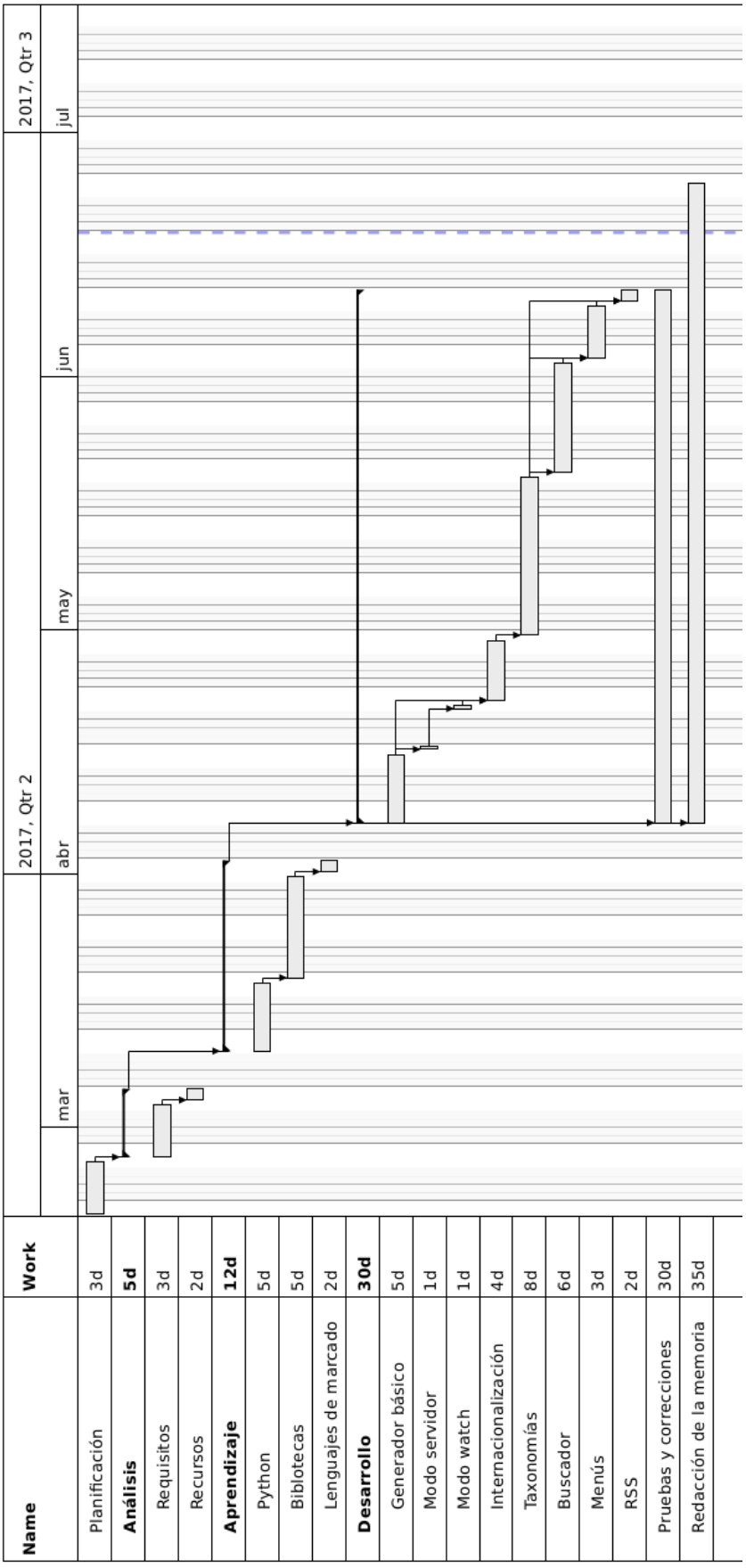

Figura 1: Diagrama de Gantt

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 el ejecución del Javascript que el usuario haya desarrollado, dependerá del navegador y equipo del usuario.

2.1. 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.

2.2. 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 [3], Textile [6] y Reestructured text [5]
- Soporte internacionalización
- Soporte para los metadatos TOML [7] y YAML [8] en los contenidos
- Páginas completamente personalizables

3. Desarrollo del sistema

El desarrollo de la plataforma web de **Sitic** supuso un reto por el cúmulo de tecnologías involucradas y las cuestiones que surgieron durante la implementación. El stack tecnológico utilizado se detalla a continuación:

- Como lenguaje principal de desarrollo se ha usado **Python** [4] por su documentación, comunidad y amplia biblioteca estándar.
- Para el motor de plantillas quse ha usado **Jinja2** [2], es un motor de plantillas completo para Python. Cuenta con so- porte unicode completo, un entorno de ejecución de espacio de trabajo inte- grado opcional, ampliamente utilizado y licenciado por BSD.
- Los lenguajes de marcados soportados para que el usuario pueda escribir los contenidos de las web han sido **Markdown** [3], **Textile** [6] y **Reestructured text** [5], a pesar de que el primer de ellos nombrados, es el mas usado debido a sus sencillez, se decidión que el sistema soportara más de uno de forma que el usuario pudiera usar aquel con el que se sintiera más cómodo.

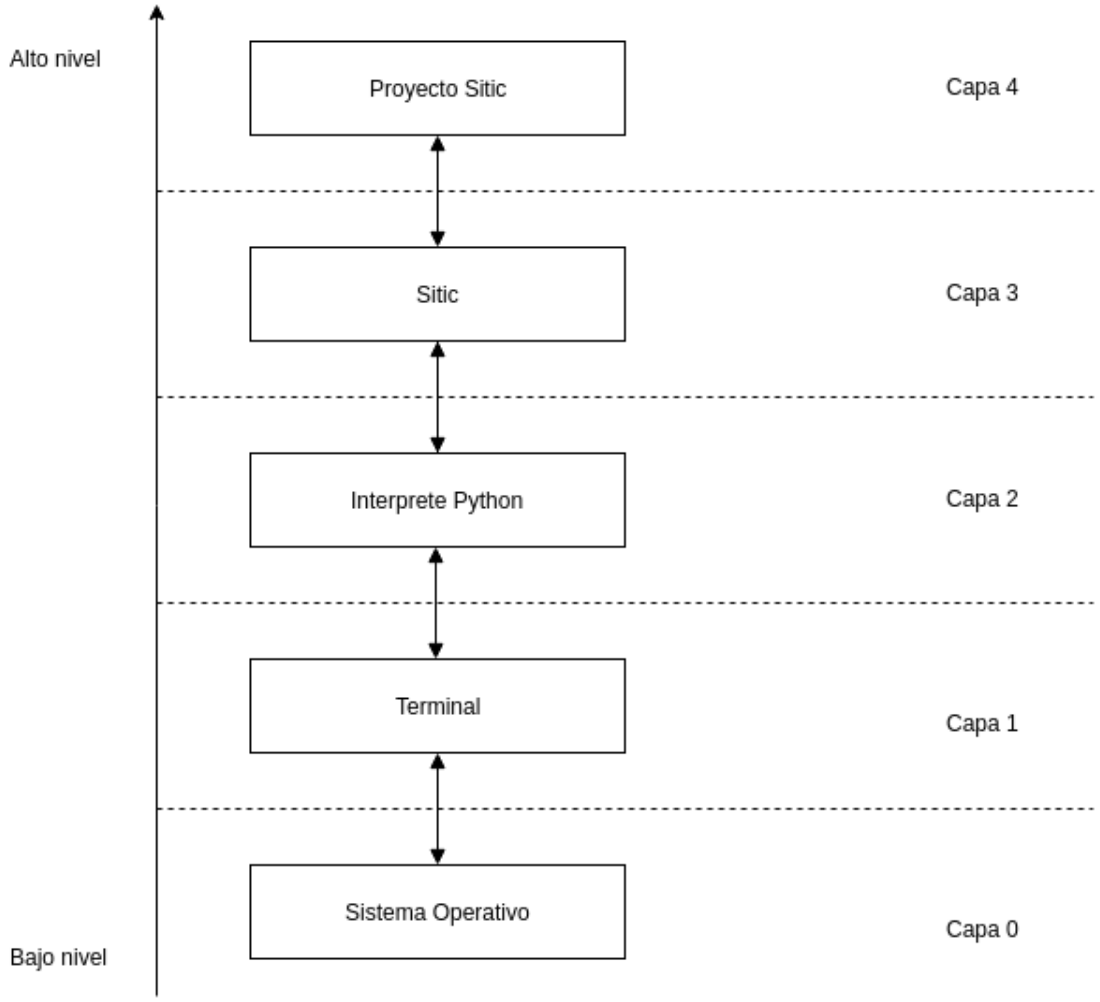

Figura 2: Arquitectura lógica

- La forma en la que el usuario pudiera configurar información extra en los ficheros de contenidos, se decidió usar los formatos **YAML** [7] y **TOML** [8], por su simplicidad y flexibilidad.

Además de las mencionadas, se han utilizado muchas otras tecnologías y herramientas de apoyo al desarrollo, que se detallan más profundamente en la memoria del proyecto.

3.1. Detalles implementación de 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` [3] 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:

```
# -*- condig: utf-8 -*-

from sitic.content.frontmatter_handlers import YamlHander, TomlHandler


handlers = [handler() for handler in [YamlHander, TomlHandler]]

def __get_handler(text):
    text_handler = None

    for handler in handlers:
        if handler.has_format(text):
            text_handler = handler
            break

    return handler


def __parse(text, handler):
    text = text.strip()
    metadata = {}

    handler = handler or __get_handler(text)
    if not handler:
        return metadata, text


    try:
        metadata, text = handler.split(text)
    except ValueError:
        return metadata, text

    metadata = handler.load(metadata)

    return metadata, text


def load(file_path, handler=None):
```

```
    text = ''

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

    text = text.strip()
    handler = handler or __get_handler(text)
    return loads(text, handler)
```

3.2. Verificación y pruebas

El diseño de casos de pruebas para un proyecto de las características de *Sitic* es algo complicado, ya que estamos en un escenario simulando como interactúan muchísimos 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.
4. Pruebas de interfaz sobre los distintos menús que se implementaban.

3.2.1. Pruebas unitarias

Esta pruebas se realizaron junto a la fase de implementación, conforme se implementaban nuevos módulos necesarios para la aplicación se realizaban pruebas individuales sobre estos módulos. 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 de este.

3.2.2. 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 pesaron para realizar este tipo de pruebas, es desarrollar una web que usara todos los elementos que se iban implementando.

3.2.3. 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 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 la siguientes versiones, dando prioridad a aquellas que se consideraban más críticas.

3.2.4. 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.

4. Conclusiones

Durante el transcurso del desarrollo de Sitic, 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.

4.1. 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 de este.

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 del desarrollo de proyecto, 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 que se ha obtenido.

Se puede decir que el proyecto goza de buena calidad. Se ha intentado hacer un software sencillo, intuitivo, fácil de manejar y entretenido para el jugador. Algo esencial para un juego de estas características, en el que se busca que cualquier persona pueda echar algún rato de su tiempo libre y qué menos que disfrute durante ese tiempo.

4.2. 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, hacer decisiones importantes para el desarrollo de este, 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 este 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.

4.3. 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 templates.
- **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.

Referencias

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

[2] *Jinja2, Offcial website*. http://jinja.pocoo.org/docs/2.9/.

[3] *Markdown, Wikipedia*. https://en.wikipedia.org/wiki/Markdown.

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

[5] *reStructuredText Documentación*. http://docutils.sourceforge.net/rst.html.

[6] *Textile Documentation*. https://txstyle.org/.

[7] *Tom's Obvious, Minimal Language*. https://github.com/toml-lang/toml.

[8] *YAML, the official website*. http://yaml.org/.