



**Universidad
Rey Juan Carlos**

**GRADO EN INGENIERÍA EN SISTEMAS AUDIOVISUALES
Y MULTIMEDIA**

Curso Académico 2022/2023

Trabajo Fin de Grado

**EVALUACIÓN SISTEMÁTICA DE PROYECTOS
DE SOFTWARE LIBRE**

Autor : Iván Miguel Molinero

Tutor : Dr. Gregorio Robles Martínez

Trabajo Fin de Grado/Máster

Evaluación Sistemática de Proyectos de Software Libre

Autor : Iván Miguel Molinero

Tutor : Dr. Gregorio Robles Martínez

La defensa del presente Proyecto Fin de Carrera se realizó el día de
de 202X, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 202X

*Hazlo o no lo hagas,
pero no lo intentes.*

Maestro Yoda

Agradecimientos

Aquí vienen los agradecimientos... Aunque está bien acordarse de la pareja, no hay que olvidarse de dar las gracias a tu madre, que aunque a veces no lo parezca disfrutará tanto de tus logros como tú... Además, la pareja quizás no sea para siempre, pero tu madre sí.

Resumen

Aquí viene un resumen del proyecto. Ha de constar de tres o cuatro párrafos, donde se presente de manera clara y concisa de qué va el proyecto. Han de quedar respondidas las siguientes preguntas:

- ¿De qué va este proyecto? ¿Cuál es su objetivo principal?
- ¿Cómo se ha realizado? ¿Qué tecnologías están involucradas?
- ¿En qué contexto se ha realizado el proyecto? ¿Es un proyecto dentro de un marco general?

Lo mejor es escribir el resumen al final.

Summary

Here comes a translation of the “Resumen” into English. Please, double check it for correct grammar and spelling. As it is the translation of the “Resumen”, which is supposed to be written at the end, this as well should be filled out just before submitting.

Índice general

1. Introducción	1
1.1. Estructura de la memoria	2
2. Objetivos	3
2.1. Objetivo general	3
2.2. Objetivos específicos	3
2.3. Planificación temporal	4
3. Estado del arte	7
3.1. Python	7
3.2. PyGithub	9
3.2.1. get_repository()	9
3.2.2. get_wiki()	9
3.3. Github	10
3.3.1. Repositorios	10
3.3.2. Pull request	10
3.3.3. Commits	10
3.3.4. Wiki	10
3.4. Django	11
3.5. HTML	11
3.6. CSS	13
3.7. Javascript	13
3.8. OpenBRR	14

4. Diseño e implementación	17
4.1. Arquitectura general	17
4.1.1. Fichero views.py	18
4.1.2. PyGithub	20
4.1.3. Django	21
4.1.4. Javascript	25
4.1.5. Plantillas HTML	25
4.1.6. Ficheros CSS	25
4.1.7. Ejemplo de uso	26
5. Experimentos y validación	31
5.1. Perceval	31
5.2. PyGithub	31
5.3. Pruebas con la aplicación	33
5.3.1. Pruebas con views.py	33
6. Resultados	47
7. Conclusiones	49
7.1. Consecución de objetivos	49
7.2. Aplicación de lo aprendido	49
7.3. Lecciones aprendidas	50
7.4. Trabajos futuros	50
A. Manual de usuario	51
Bibliografía	53

Índice de figuras

3.1. Fragmento HTML	12
3.2. Esquema de análisis de OpenBRR	15
4.1. Arquitectura de la aplicación	18
4.2. Apartado para generar la contraseña de Gmail	20
4.3. Objeto repositorio de PyGithub	22
4.4. Overflow.html	22
4.5. Página principal con el formulario relleno	23
4.6. Página de datos con las pestañas ocultas.	26
4.7. Primera parte de la página de datos.	27
4.8. Segunda parte de la página de datos.	27
4.9. Tercera parte de la página de datos.	28
4.10. Primera parte de la página de resultados.	28
4.11. Segunda parte de la página de resultados.	29
4.12. Tercera parte de la página de resultados.	29
5.1. Programa de prueba de Perceval.	32
5.2. Primera parte del programa de prueba de PyGithub.	34
5.3. Segunda parte del programa de prueba de PyGithub.	35
5.4. Tercera parte del programa de prueba de PyGithub.	36
5.5. Salida del programa de prueba de PyGithub.	36
5.6. Número de commits de PyGithub.	36
5.7. Número de forks de PyGithub.	37
5.8. Lenguajes utilizados en el repositorio de PyGithub.	37

5.9. Página principal antes de analizar PyGithub.	38
5.10. Pestaña de comunidad de PyGithub.	39
5.11. Obtención de la calificación en función del número de commits.	39
5.12. Obtención de la calificación en función del número de forks.	40
5.13. Obtención de la calificación en función del número de suscriptores.	41
5.14. Obtención de la calificación en función de si pertenece a una organización. . . .	41
5.15. Obtención de la calificación en función de la fecha de última actualización. . .	42
5.16. Calificación de comunidad obtenida por la aplicación.	42
5.17. Calificación de seguridad obtenida por la aplicación.	42
5.18. Calificación de funcionalidad obtenida por la aplicación.	43
5.19. Calificación de soporte obtenida por la aplicación.	43
5.20. Calificación de calidad obtenida por la aplicación.	43
5.21. Calificación de usabilidad obtenida por la aplicación.	43
5.22. Calificación de adopción y nota final obtenida por la aplicación.	43
5.23. Formulario rellenado con mi propia dirección de correo.	44
5.24. Mensaje enviado por la aplicación.	45

Capítulo 1

Introducción

Actualmente se comparten muchos proyectos de software libre en Internet en general y en Github [3]¹ en particular. Algunos ejemplos de proyectos que se comparten son paquetes de Python que ayudan a desarrolladores a no tener que reinventar la rueda”, es decir, que si necesitas hacer un proyecto que lea URLs en uno de sus pasos (por ejemplo) y alguien ya ha hecho un paquete para ello, debes usar ese paquete para poder avanzar en tu proyecto de una forma más rápida y eficaz.

Un proyecto de software libre es un desarrollo que es público; los demás desarrolladores pueden verlo y mejorarlo a través de Pull Request y también descargarlo gratuitamente para su uso por eso muchos usuarios lo prefieren a proyectos de código cerrado.

Ahora bien, en Github se comparten infinidad de proyectos de software libre y no siempre es fácil distinguir entre los que son buenos y los que no lo son tanto: ¿cuál es más seguro? o ¿cuál es más eficiente? Para resolver estas y más preguntas es que, gracias al paquete PyGithub², he desarrollado esta aplicación web que permite al usuario introducir el nombre de un repositorio y analizarlo según los parámetros que más le convengan.

El proyecto[10] nace como una extensión de OpenBRR³ que es una forma de analizar proyectos en función a los siguientes 7 parámetros: funcionalidad, calidad, soporte, comunidad, adopción y usabilidad. A estos parámetros se les asigna un peso y a partir de ahí, da una calificación final sobre 5 del proyecto. A parte de OpenBRR existen otras formas de analizar

¹<https://github.com/>

²<https://github.com/pygithub/pygithub>

³https://link.springer.com/chapter/10.1007/978-3-642-13244-5_18

proyectos como OSMM, OpenBQR o MOSST entre muchos otros pero me decanté por esta opción al ser la que más se centraba en analizar los proyectos de software libre.

Esta herramienta ayudará a docentes y desarrolladores a distinguir entre los miles de repositorios de Github cuáles son los que merecen la pena agilizando mucho la búsqueda. Pongamos un ejemplo práctico para que se entienda mejor: un desarrollador busca un paquete que realice una función específica para su proyecto pero si tiene que buscar entre las miles de opciones cuál es la mejor, tardaría un tiempo excesivo ya que tendría que analizar cada código por si mismo (con todo lo que ello supone como entenderlo si no está bien comentado), buscar vulnerabilidades, descargarlo, probarlo , etc. En cambio, gracias a este proyecto podrá buscar su mejor opción configurando el análisis según le convenga pudiendo poner el foco solo en seguridad, en seguridad y funcionalidad o en los 7 apartados que tiene OpenBRR.

1.1. Estructura de la memoria

- Capítulo 1: se explica brevemente el origen del proyecto, sus bases y funcionamiento .
- Capítulo 2: se muestran los objetivos del proyecto.
- A continuación se presenta el estado del arte en el capítulo 3.
- ...

Capítulo 2

Objetivos

2.1. Objetivo general

Mi trabajo de fin de grado consiste en crear una aplicación web que analice repositorios de Github para darles una calificación sobre 5 basándose en los parámetros de OpenBRR.

2.2. Objetivos específicos

A la hora de realizar este proyecto, he abordado los siguientes objetivos específicos:

- Estudiar, aprender y utilizar Django¹.
- Estudiar qué es OpenBRR y saber cómo analizar un código para poder calificar cada uno de sus parámetros.
- Analizar el paquete PyGithub en busca de las funciones que necesitaba para analizar repositorios.
- Rellenar y enviar formularios con Django.
- Manipular los datos enviados por el usuario y enviar los resultados.
- Probar el proyecto tanto con repositorios vacíos, como con repositorios teóricamente buenos para comprobar si los resultados eran coherentes.

¹<https://www.djangoproject.com/>

- Solucionar errores como que el usuario introduzca un repositorio que no exista.
- Introducir mejoras tales como permitir al usuario introducir una dirección de correo electrónico para que le lleguen los resultados ahí también.
- Probar aún más repositorios haciendo que la aplicación se centrase en diferentes parámetros para comprobar que hacía un correcto análisis.

2.3. Planificación temporal

El proyecto se empezó a pensar a finales de Febrero de 2022 pero no fue hasta el verano de ese año que se empezó a trabajar en serio en él ya que anteriormente tenía que compaginarlo con las prácticas externas y no disponía del tiempo necesario. De Marzo a Junio de 2022 dedicaba las mañanas de los sábados y domingos a investigar qué era OpenBRR, en qué consistía y cómo podía usarlo para este proyecto. Asimismo mi tutor me recomendó usar el paquete Perceval² de Python para analizar los repositorios de GitHub pero tras investigar por mi cuenta di con el paquete PyGithub el cual analizaba los repositorios de una forma más rápida y cómoda para el desarrollador ya que únicamente con introducir el nombre del repositorio es capaz de sacar todos los datos necesarios.

Durante el verano, pasaba las mañanas de lunes a viernes trabajando en el TFG y conseguí entender el funcionamiento completo de PyGithub así como analizar todos los datos que me devolvía y relacionarlos con los 7 parámetros de OpenBRR intentando que cada dato perteneciera al parámetro que más le correspondiera.

En el mes de Septiembre decidí implementar Django en la aplicación para ser capaz de leer, manipular y crear páginas web a través de las peticiones del usuario. No tenía conocimientos sobre esta tecnología así que usé los tutoriales de DjangoGirls³[2] para aprender a utilizarla dedicando 3 tardes de lunes a viernes y las mañanas de los sábados y domingos.

Finalmente, de Octubre de 2022 a Enero de 2023, empecé con el desarrollo de la aplicación web dedicando el mismo reparto de tiempo que para aprender Django ya que tenía clases de la universidad por las mañanas salvo en las vacaciones de Navidad que le dediqué todas las mañanas sin excepción. Durante el desarrollo surgieron varias dudas como qué datos asignar a

²<https://perceval.readthedocs.io/en/latest/perceval/git.html>

³[https://django.org/es/](https://.djangoproject.org/es/)

qué parámetros y, sobretodo, qué peso asignarle a cada dato pero tras varias reuniones con mi tutor decidimos que el usuario pudiera manejar los pesos a su antojo para así poder centrarse en analizar los repositorios de la forma que más le convenga. A finales de Diciembre comencé a hacer pruebas con repositorios profesionales como el propio de PyGithub o los de myTeachingURJC⁴ (los cuales son utilizados por los profesores de la universidad para compartir contenido de sus asignaturas) y con repositorios de peor calidad como mis primeros repositorios en busca de evaluaciones coherentes y algún error. Para principios de Enero la aplicación ya era funcional pero no tenía ningún tipo de estilo así que dediqué el resto del tiempo a personalizar su aspecto, así como a corregir y controlar errores que surgían durante las diferentes pruebas que realizaba. Por tanto, dediqué las mañanas de finales de Enero y todo Febrero para la realización de la memoria con el objetivo de poder presentar el proyecto en Marzo.

⁴<https://github.com/myTeachingURJC>

Capítulo 3

Estado del arte

Para la realización de este proyecto se ha utilizado Python, PyGithub, Github, Django, HTML, CSS, Javascript y OpenBRR.

3.1. Python

Python[9] es un lenguaje de programación interpretado que le da importancia a la legibilidad del código teniendo una sintaxis muy limpia. Está orientado a objetos y permite ser un buen lenguaje para aprender a programar ya que tiene una curva de aprendizaje muy suave gracias a su sintaxis muy cercana al lenguaje humano. Es un lenguaje de código abierto, lo que le permite estar en constante mejora por parte de los usuarios y ser adaptado para diferentes usos como el Machine Learning.

Python fue creado por Guido van Rossum en 1991 mientras trabajaba en el sistema operativo Amoeba¹ teniendo como objetivo controlar excepciones y manejar interfaces de dicho sistema operativo. A finales del año 2000 nació Python 2.0 el cual soportaba Unicode por primera vez y empezó a ser desarrollado por la comunidad bajo la mirada de Guido. Finalmente, en 2008 se lanzó Python 3.0 siendo la versión más completa de Python pero incompatible con las anteriores por lo que muchas características se implementaron también en Python 2.6.

La filosofía de Python se basa en los siguientes puntos para hacer la vida más fácil al desarrollador:

- Hermoso es mejor que feo.

¹<https://www.cs.vu.nl/pub/amoeba/>

- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Plano es mejor que anidado.
- Disperso es mejor que denso.
- El código legible cuenta.
- Casos especiales no son lo suficientemente especiales para romper las reglas.
- Casi siempre lo práctico vence a lo formal
- Los errores no deben pasar nunca desapercibidos, a menos que se especifique este comportamiento.
- Ante una ambigüedad, descarte la tentación a adivinar.
- Debe haber una, y preferentemente una sola, manera obvia de lograr algo, aunque esta generalmente no está clara a primera vista a menos que seas un genio.
- Ahora es mejor que nunca, aunque en muchas ocasiones nunca es mejor que ahora mismo.
- Si la implementación es difícil de explicar, entonces es una mala idea.
- Si la implementación es fácil de explicar, entonces pudiera ser una buena idea.
- Los espacios de nombre son una buena idea, hagamos más de eso.

Para finalizar, aquí podemos ver un ejemplo de un fragmento de código en Python correspondiente a un desarrollo que hice por mi cuenta² :

```
class MiCuchillo(Sprite):
    def __init__(self, image, dir, x, y):
        super().__init__(image)
        self.position = (x, y)
        self.direccion = dir
        self.cshape = AARectShape(self.position, self.width/5, self.height/5)
```

²<https://github.com/ivanmiguelmolinero/Python2D>

```

def update(self, dt):
    coor_cuchillo = manejador_scroll.world_to_screen(self.x, self.y)[0]
    if coor_cuchillo > 1280 or coor_cuchillo < 0:
        self.kill()
    if self.direccion == 'd':
        self.position += Vector2(20, 0)
    if self.direccion == 'i':
        self.position -= Vector2(20, 0)
    self.cshape.center = Vector2(self.position[0], self.position[1])

```

3.2. PyGithub

PyGithub[8] se trata de la librería utilizada en la aplicación para analizar los datos del repositorio. Adicionalmente, también permite administrar recursos de Github como repositorios, perfiles de usuario y organizaciones. Es una librería de código abierto lo que permite que esté en constante mejora y desarrollo por parte de la comunidad.

Algunas funciones de interés son:

3.2.1. get_repository()

Esta función permite obtener todos los datos del repositorio únicamente pasándole el nombre del repositorio. Ejemplo de uso sacado directamente de este proyecto:

```

# Obtenemos el repositorio introducido por el usuario
repo = get_repository(request.GET['text'])

```

3.2.2. get_wiki()

Esta función es un ejemplo de los datos que nos permite analizar PyGithub. En concreto, “getwiki()” nos devuelve si el repositorio en cuestión tiene wiki o no. Otros ejemplos de funciones de este tipo son “repo.homepage”, “repo.subscriberscount”, “repo.getcommits().totalCount”...

3.3. Github

Github es una herramienta esencial para ingenieros de software que consiste en un servicio basado en la nube donde los desarrolladores suben sus proyectos de código abierto a unos repositorios para compartirlos con la comunidad, permitiendo así realizar cambios y mejoras en el proyecto. Se ha convertido en una herramienta muy popular llegando a alojar más de 100 millones de repositorios.

Github fue desarrollado por Chris Wanstrath, P. J. Hyett, Tom Preston-Werner y Scott Chacon usando Ruby on Rails, y comenzó a funcionar en 2008. Desde 2018 pertenece a Microsoft.

A continuación se explican los diferentes mecanismos que permiten que funcione Github.

3.3.1. Repositorios

Lugar donde los desarrolladores suben sus proyectos. Por ejemplo, este proyecto está subido en el repositorio “ivanmiguelmolinero/TFG”.

3.3.2. Pull request

Es la funcionalidad que permite a los desarrolladores colaborar entre ellos sugiriendo cambios del programa en cuestión.

3.3.3. Commits

Funcionalidad que permite realizar y subir al repositorio cambios en el proyecto junto con un breve mensaje descriptivo. Se puede ejecutar mediante la interfaz de un editor de código con VSCode³ o bien ejecutando “git commit -m [mensaje descriptivo entre comillas]” en una ventana de comandos.

3.3.4. Wiki

Sección del repositorio donde se explica la funcionalidad del proyecto, el manual de usuario y los diferentes cambios que se van implementando. Muchos proyectos también usan el README del repositorio para estas funcionalidades.

³<https://code.visualstudio.com/>

3.4. Django

Django es un framework de Python de alto nivel que fue creado para gestionar páginas web facilitando la creación de sitios web complejos. Fue lanzado inicialmente en Julio de 2005.

La distribución principal de Django permite aplicaciones con un sistema de comentarios, herramientas para juntar contenido vía RSS⁴ o Atom⁵, plantillas que permiten gestionar páginas de contenido sin necesidad de escribir controladores o vistas de esas páginas y un sistema de redirección de URLs.

Las características principales de Django son:

- Framework muy rápido, es decir, permite a los desarrolladores terminar sus proyectos lo más rápido posible.
- Es muy seguro: ayuda a evitar los errores de seguridad más comunes.
- Es muy escalable: a medida que crece la aplicación da facilidades para añadirle las nuevas funcionalidades.

3.5. HTML

HTML[4] (lenguaje de marcas de hipertexto) es el componente más básico de una página web. Es usado para estructurar el contenido de la misma. Normalmente se utiliza junto con CSS y Javascript que se explicarán en los siguientes apartados. Hipertexto se refiere a los enlaces que conectan las páginas web entre sí.

Un elemento HTML se distingue de otro mediante etiquetas que se definen entre los símbolos ” < ” y ” > ”.

En la siguiente figura se puede ver un ejemplo de HTML

⁴<https://www.rssboard.org/rss-specification>

⁵<https://www.ietf.org/rfc/rfc4287>

```
1  <!DOCTYPE html>
2  <html lang="es" dir="ltr">
3      <head>
4          <meta charset="utf-8">
5          <title>Calculadora Iván Miguel Molinero</title>
6
7          <!-- HOJA DE ESTILO -->
8          <link rel="stylesheet" href="micss.css">
9
10         <!-- Programa en Javascript -->
11         <script <script src="mjs1.js"></script>
12
13     </head>
14
15     <body onload="main();">
16         <div class="pantalla">
17             <h1 id="display">0</h1>
18         </div>
19         <div class="teclado">
20             <button type="button" name="button" id="boton7" class="boton">7</button>
21             <button type="button" name="button" id="boton8" class="boton">8</button>
22             <button type="button" name="button" id="boton9" class="boton">9</button>
23             <button type="button" name="button" id="botonac" class="boton">AC</button>
24             <br> [REDACTED]
25             <button type="button" name="button" id="boton4" class="boton">4</button>
26             <button type="button" name="button" id="boton5" class="boton">5</button>
27             <button type="button" name="button" id="boton6" class="boton">6</button>
28             <button type="button" name="button" id="operacionpor" class="boton">x</button>
29             <button type="button" name="button" id="operaciondivision" class="boton">/</button>
30             <br> [REDACTED]
31             <button type="button" name="button" id="boton1" class="boton">1</button>
32             <button type="button" name="button" id="boton2" class="boton">2</button>
33             <button type="button" name="button" id="boton3" class="boton">3</button>
34             <button type="button" name="button" id="operacionmas" class="boton">+</button>
35             <button type="button" name="button" id="operacionmenos" class="boton">-</button>
36             <br> [REDACTED]
37             <button type="button" name="button" id="boton0" class="boton">0</button>
38             <button type="button" name="button" id="botoncoma" class="boton">.</button>
39             <button type="button" name="button" id="operacionigual" class="boton">=</button>
40         </div>
41     </body>
42 </html>
```

Figura 3.1: Fragmento HTML

3.6. CSS

CSS[1] (del inglés Cascading Style Sheets) es utilizado para dar estilo a las páginas web creadas con HTML o XML. Ha tenido varias versiones pero es desde CSS3 que su alcance es tal que cada módulo empezó a mostrar varias diferencias por lo que el W3C⁶ decidió empezar a realizar capturas de las últimas especificaciones estables.

Su sintaxis más básica consiste en hacer referencia a las diferentes etiquetas de HTML y definir su estilo entre corchetes. Por ejemplo:

```
div {  
    text-align: center;  
}
```

```
img {  
    height: 20px;  
    margin-left: 0px;  
}
```

3.7. Javascript

Javascript[5] (abreviado JS) es un lenguaje de programación ligero, interpretado y compilado “just-in-time”. Se usa para secuencias de comandos en páginas web aunque también se utiliza en otros entornos como Node.js⁷.

Se originó en el año 1995 para el navegador Netscape como una forma de agregar programas a páginas web. Desde entonces, el lenguaje ha sido adoptado por todos los demás navegadores gráficos principales. Ha hecho posibles las aplicaciones web modernas, aplicaciones con las que puede interactuar directamente sin hacer una recarga de página para cada acción.

Algunas de las características de Javascript son las siguientes:

- Lenguaje del lado del cliente
- Orientado a objetos

⁶<https://www.w3.org/>

⁷<https://developer.mozilla.org/es/docs/Glossary/Node.js>

- De tipado débil o no tipado
- De alto nivel
- Lenguaje interpretado
- Muy utilizado por los desarrolladores

3.8. OpenBRR

OpenBRR[7] (del inglés Business Readiness Rating) es un modelo de evaluación de software libre. Su objetivo es estandarizar una fórmula de evaluación de software. Este método fue patrocinado por el Centro de Investigación de Código abierto e Intel.

Se considera necesario ya que proporciona a las compañías una forma segura, estandarizada y eficaz de evaluar un software antes de adoptarlo. Hasta el momento ha permitido a compañías tanto elegir un software ideal para misiones críticas como descartar otros que a primera vista parecían buenos pero que después de analizarlos dieron con graves problemas de seguridad.

El esquema de análisis se puede ver en la figura 3.2. Aparte de los parámetros que aparecen en la figura se decidió introducir el parámetro de seguridad al considerarse de importancia también.

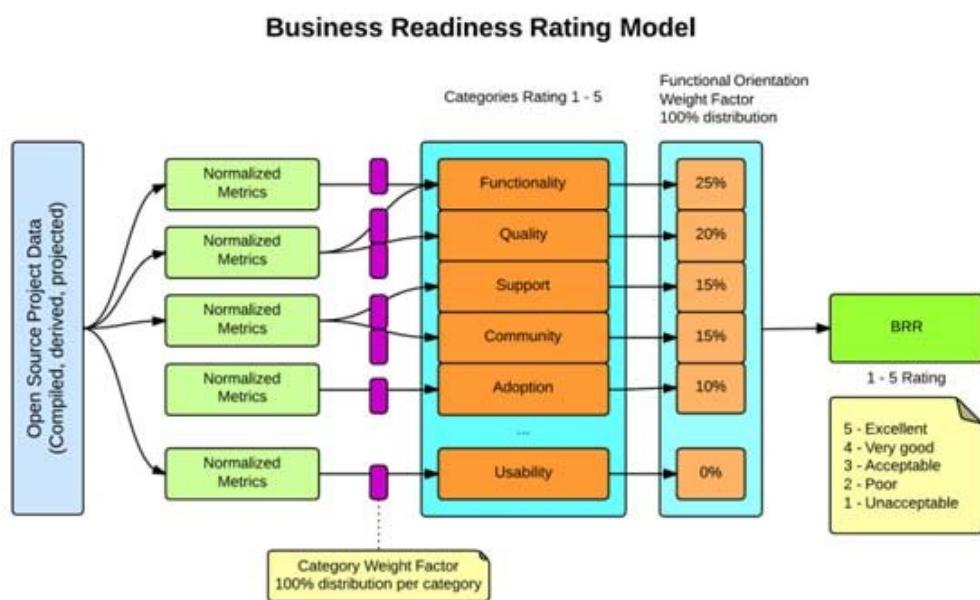


Figura 3.2: Esquema de análisis de OpenBRR

Capítulo 4

Diseño e implementación

En este capítulo se detallará cómo funciona la aplicación desde el código hasta las funciones que debe realizar el usuario.

4.1. Arquitectura general

Como se puede observar en la figura 4.1 el programa principal depende del usuario, PyGithub y de GitHub. Como se ha explicado anteriormente, el usuario debe introducir la dirección de un repositorio, compuesta por ”usuario_de_GitHub/nombre_del_repositorio” y PyGithub se encarga de pedir los datos a Github y devolvérselos al programa principal. Vamos a ver esta parte con más detalle.

Una vez que el usuario introduce el repositorio, el programa principal, gracias a Django, manda una petición GET¹ a través de la cual se obtiene el repositorio con la función get_repository del paquete analize_repo.py creado por mí. Este paquete contiene las funciones necesarias para obtener los datos del repositorio. A continuación se crean 7 listas, una por cada parámetro de OpenBRR, en el fichero views.py donde se irán guardando los datos del repositorio.

¹<https://developer.mozilla.org/es/docs/Web/HTTP/Methods/GET>

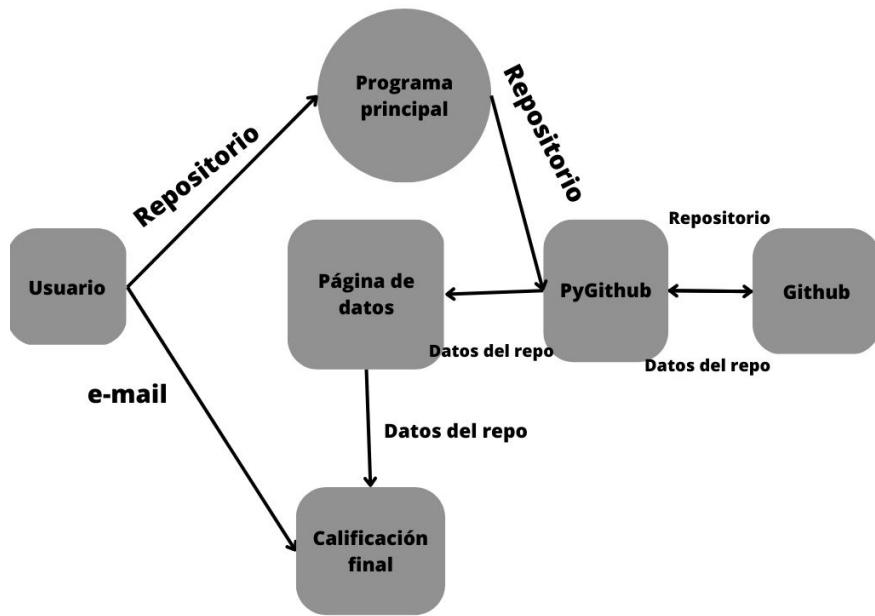


Figura 4.1: Arquitectura de la aplicación

4.1.1. Fichero views.py

Este fichero forma parte de la estructura de Django y tiene dos funciones principales: recibir las peticiones GET y, en función de dichas peticiones, servir la plantilla HTML correspondiente y organizar los datos extraídos del repositorio a analizar. Funciona de la siguiente manera:

1. El usuario manda mediante una petición GET el nombre del repositorio, el fichero views.py lo recibe y le pide a PyGithub los datos del repositorio para mostrar al usuario la plantilla "repo_data.html" donde podrá leer los datos y manipularlos para su posterior calificación. En caso de que no exista el repositorio escrito por el usuario se devolverá la plantilla "error.html" donde se le da la posibilidad de volver a la página principal para volver a escribir el nombre correctamente.
2. Una vez extraídos los datos del repositorio, views.py los organiza y guarda en 7 listas correspondientes a los parámetros de OpenBRR(posts, post_sec, post_func, post_supp, post_qual, post_usab y post_adop correspondientes a los parámetros de comunidad, seguridad, funcionalidad, soporte, calidad, usabilidad y adopción respectivamente). Consigue analizar estos datos gracias a las diferentes funciones de PyGithub.

3. Como se verá más adelante, en esta plantilla el usuario podrá manejar los datos a su antojo y, una vez esté de acuerdo, mandarlos para su calificación junto con su dirección de correo para recibir el análisis en su bandeja de entrada si así lo desea.
4. Mediante otra petición GET se mandan todos estos datos y se calculan las calificaciones.
5. Views.py puntúa el repositorio en función de los datos obtenidos y/o introducidos por el usuario y el valor que les haya querido dar con la siguiente fórmula:

```
nota ponderada = nota * valor
```

siendo la nota la calificación del 1 al 10 del dato en cuestión y valor el número que haya introducido el usuario en ese apartado dividido por 100. En apartados posteriores se explica cómo se obtiene cada nota más en profundidad.

6. Como el estándar de OpenBRR da una calificación del 1 al 5, la nota final de cada parámetro se divide entre 2 con un redondeo de 2 decimales. Se aplica la siguiente fórmula:

```
nota final parámetro = suma notas ponderadas / 2
```

7. La nota final del repositorio se obtiene como la media de las notas de los 7 parámetros:

```
nota final = suma de las notas de los parámetros / 7
```

8. Finalmente, views.py sirve la plantilla "result.html" y manda un correo al usuario (si así lo ha indicado) con los resultados del análisis y las calificaciones sobre 5.

Envío del correo

Para el envío del correo se utiliza la función "send_mail" de django.core.mail la cual nos permite enviar correos definiendo el asunto, el mensaje y la dirección de correo desde la que se enviará. Se debe configurar previamente el fichero "settings.py" de Django añadiendo los siguientes campos:

```
EMAIL_BACKEND = "django.core.mail.backends.smtp.EmailBackend"
EMAIL_HOST = "smtp.gmail.com"
```

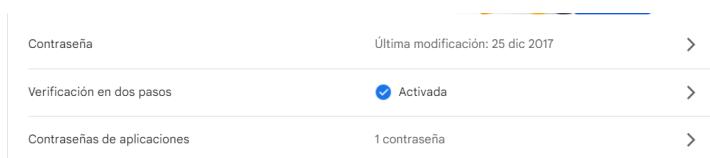


Figura 4.2: Apartado para generar la contraseña de Gmail

```
EMAIL_USE_TLS = True
EMAIL_PORT = 587
EMAIL_HOST_USER = "tu_direccion_de_correo@gmail.com"
EMAIL_HOST_PASSWORD = config["EMAIL_PASSWORD"]
```

”EMAIL_PASSWORD” es la contraseña generada por Gmail² para estos tipos de servicios y que va guardada en un fichero ”credentials.env” para no ser desvelada en el repositorio. Para generar esta contraseña debes entrar en el apartado de seguridad de tu cuenta de Google <https://myaccount.google.com/security> pulsando en el apartado ”Contraseñas para aplicaciones” tal y como se indica en la figura 4.2.

4.1.2. PyGithub

PyGithub es la librería encargada de recibir la dirección del repositorio y obtener los datos necesarios para su posterior análisis. Funciona de la siguiente manera:

1. En primer lugar obtiene el repositorio con la función ”get_repo()” a la que se le pasa como argumento la dirección del repositorio.

²<https://www.gmail.com/mail/help/intl/es/about.html?iframe>

2. Una vez hecho esto se obtiene un objeto repositorio como el de la figura 4.3 donde se ha puesto de ejemplo el repositorio "moodle/moodle"³
3. A partir de aquí se obtienen los datos correspondientes a cada apartado gracias a las funciones que nos brinda PyGithub y que podemos observar en la figura 4.3 (size, pushed_at o subscribers_count son algunos ejemplos).

Cabe añadir la restricción con la que cuenta esta librería. PyGithub limita las peticiones que puedes realizar durante una hora y, si excedes esas peticiones, devolverá la excepción "RateLimitExceeded" con el siguiente mensaje:⁴

```
github.GithubException.RateLimitExceeded: 403
{"message": "API rate limit exceeded for 84.78.50.170.
(But here's the good news: Authenticated requests get
a higher rate limit. Check out the documentation
for more details.)", "documentation_url":
"https://docs.github.com/rest/overview/
resources-in-the-rest-api#rate-limiting"}
```

lo cual ha sido controlado en el fichero views.py visto anteriormente haciendo que muestre la plantilla "overflow.html" que se corresponde con una página que informa al usuario del error y le permite volver a la página principal como se observa en la figura 4.4.

4.1.3. Django

Django es el framework de Python que se encarga de gestionar la página web mediante el envío de formularios. Consta de varios ficheros, entre ellos el fichero views.py explicado en el apartado 4.1.1.

Los formularios son enviados mediante el método GET como en el siguiente ejemplo:

```
text=ivanmiguelmolinero%2FTFG
```

³<https://github.com/moodle/moodle>

⁴<https://docs.github.com/rest/overview/resources-in-the-rest-api#rate-limiting>

```
> repos_repository['full_name']=='moodle/moodle"]> special_variables> function variables> CHECK_AFTER_INIT_FLAG: Falseallow_merge_commit: Nonelallow_rebase_merge: Nonelallow_squash_merge: Nonearchive_url: "https://api.github.com/repos/moodle/moodle/{archive_format}{{/ref}}'archived: Falsesigned_name: "https://api.github.com/repos/moodle/moodle/assignees{/user}'blobs_url: "https://api.github.com/repos/moodle/moodle/git/blobs{/sha}'branches_url: "https://api.github.com/repos/moodle/moodle/branches{/branch}'clone_url: "https://github.com/moodle/moodle.git'collaborators_url: "https://api.github.com/repos/moodle/moodle/collaborators{/collaborator_id}'commits_url: "https://api.github.com/repos/moodle/moodle/commits{/number}'commits_url: "https://api.github.com/repos/moodle/moodle/commits{/sha}'compare_url: "https://api.github.com/repos/moodle/moodle/compare/{base}...{head}"contents_url: "https://api.github.com/repos/moodle/moodle/contents{/path}'contributors_url: "https://api.github.com/repos/moodle/moodle/contributors'"created_at: datetime.datetime(2009, 12, 16, 23, 1, 23)'default_branch: "master'"delets_branch_on_merge: Nonehomepage: "https://moodle.org/"hooks_url: "https://api.github.com/repos/moodle/moodle/hooks'"html_url: "https://github.com/moodle/moodle'"id: 439861is_template: Falsesource_issue_comment_url: "https://api.github.com/repos/moodle/moodle/issues/comments{issue_id}'source_issues_url: "https://api.github.com/repos/moodle/moodle/issues/events{issues_url}'source_issues_url: "https://api.github.com/repos/moodle/moodle/issues{/number}'keys_url: "https://api.github.com/repos/moodle/moodle/keys{/key_id}'labels_url: "https://api.github.com/repos/moodle/moodle/labels{/name}'language: "PHP'"languages_url: "https://api.github.com/repos/moodle/moodle/languages'"last_modified: "Mon, 06 Feb 2023 10:51:25 GMT'"master_branch: Nonemerges_url: "https://api.github.com/repos/moodle/moodle/merges'"milestones_url: "https://api.github.com/repos/moodle/moodle/milestones{/number}'network_url: "git@github.com:moodle/moodle.git'"name: "moodle'"network_count: 5824'notifications_url: "https://api.github.com/repos/moodle/moodle/notifications'"open_issues: 0'open_issues_count: 5'organization: Organization(login="moodle")'owner: NumedisUser(login="moodle")'parent: Nonerights: Nonerelations: Nonesource_issue_comment_url: "https://api.github.com/repos/moodle/moodle/pulls{/number}'source_issues_url: "https://api.github.com/repos/moodle/moodle/pulls/events{/issue_id}'source_issues_url: "https://api.github.com/repos/moodle/moodle/pulls{/number}'stargazers_url: "https://api.github.com/repos/moodle/moodle/pulls{/id}'size: 665720'source: Nonessh_url: "https://github.com/moodle/moodle.git'"stargazers_count: 4528'stargazers_url: "https://api.github.com/repos/moodle/moodle/stargazers'"statuses_url: "https://api.github.com/repos/moodle/moodle/statuses{/sha}'subscribers_count: 410'subscribers_url: "https://api.github.com/repos/moodle/moodle/subscribers'"subscription_url: "https://api.github.com/repos/moodle/moodle/subscription'"svn_url: "https://github.com/moodle/moodle'"tags_url: "https://api.github.com/repos/moodle/moodle/tags'"
```

Figura 4.3: Objeto repositorio de PyGithub



Límite de peticiones a Github alcanzado.
Inténtelo de nuevo más tarde.

Volver

Figura 4.4: Overflow.html



Figura 4.5: Página principal con el formulario relleno

Se usan parejas de clave-valor, donde la clave es el nombre del campo a llenar y valor lo que haya introducido el usuario. El ejemplo mostrado es el correspondiente a introducir en la pantalla principal el valor "ivanmiguelmolinero/TFG" como se puede ver en la figura 4.5.

A continuación se pasan a explicar cada uno de los ficheros de importancia de Django.

urls.py

Es el fichero encargado de gestionar las peticiones que le llegan comunicándose con views.py. El usuario lanza una petición y, en función de esta, este fichero activa una función u otra de views.py. Por ejemplo, si el usuario introduce la dirección raíz, urls.py se encarga de que se acabe mostrando la página principal mediante la función views.post_main. Las otras dos URLs que debe gestionar se corresponden con el envío del formulario de la página principal (figura 4.5) y el envío del formulario de repo_data.html

forms.py

Fichero que define el modelo de formulario a utilizar

models.py

Fichero que da forma al modelo definido en forms.py

settings.py

Este es el fichero de configuración de Django, en él se deben especificar:

- Directorio donde está guardado el fichero credentials.py correspondiente a las credenciales de Gmail necesarias para el envío del correo.
- Directorio base a partir del cual funcionará la aplicación.
- Clave secreta.
- Hosts permitidos: en este caso son 127.0.0.0 (localhost) y pythonanywhere.com.
- Las apps instaladas en el sitio.
- La lógica de intercambio de información o middleware[6].
- La definición de las plantillas.
- La base de datos donde se guardarán diferentes valores si se necesitara.
- El idioma del lenguaje.
- La zona horaria.
- La URL y el directorio raíz.
- Los valores necesarios para el envío del correo: EMAIL_BACKEND, EMAIL_HOST, EMAIL_USE_TLS, EMAIL_PORT, EMAIL_HOST_USER y EMAIL_HOST_PASSWORD.

manage.py

Es el fichero encargado de iniciar el servidor como se indica en el manual de usuario y gestionar el error de no haber instalado correctamente Django. Funcion mediante el siguiente comando:

```
python manage.py runserver
```

4.1.4. Javascript

Los ficheros javascript son los encargados de la interacción aplicación-usuario. En el caso de esta aplicación, la plantilla repo_data.html funciona gracias al fichero datos.js

datos.js

Controla que el usuario pueda ocultar y mostrar las pestañas de los parámetros de OpenBRR, controla que no pueda introducir valores superiores al 100 % (función edit_max_value_[parámetro] y get_suma) y que pueda modificar los datos mostrados (función save_input_[parámetro]). Además, controla que solo pueda enviar el formulario si se están mostrando para evitar el error de enviar el formulario vacío.

4.1.5. Plantillas HTML

Es lo que se muestra al usuario una vez renderizadas. La aplicación cuenta con las siguientes:

- **Main.html:** Correspondiente a la página principal muestra el nombre de la aplicación y el formulario para enviar el nombre del repositorio a analizar.
- **Repo_data.html:** Muestra los datos del repositorio en diferentes pestañas que el usuario puede mostrar u ocultar. También puede modificar esos datos. Al final hay un formulario para introducir su dirección de correo donde recibirá los resultados de su análisis.
- **Result.html:** Muestra la calificación final de cada uno de los parámetros y del repositorio.
- **Error.html:** Página que se muestra si el repositorio introducido no existe. Permite al usuario volver a la página principal.
- **Overflow.html:** Página que se muestra cuando se excede el número de peticiones a Github. Permite al usuario volver a la página principal.

4.1.6. Ficheros CSS

Son los que dan estilo a cada una de las plantillas HTML.



Figura 4.6: Página de datos con las pestañas ocultas.

4.1.7. Ejemplo de uso

De la figura 4.5 a la figura 4.12 tenemos un ejemplo analizando el repositorio de este TFG.

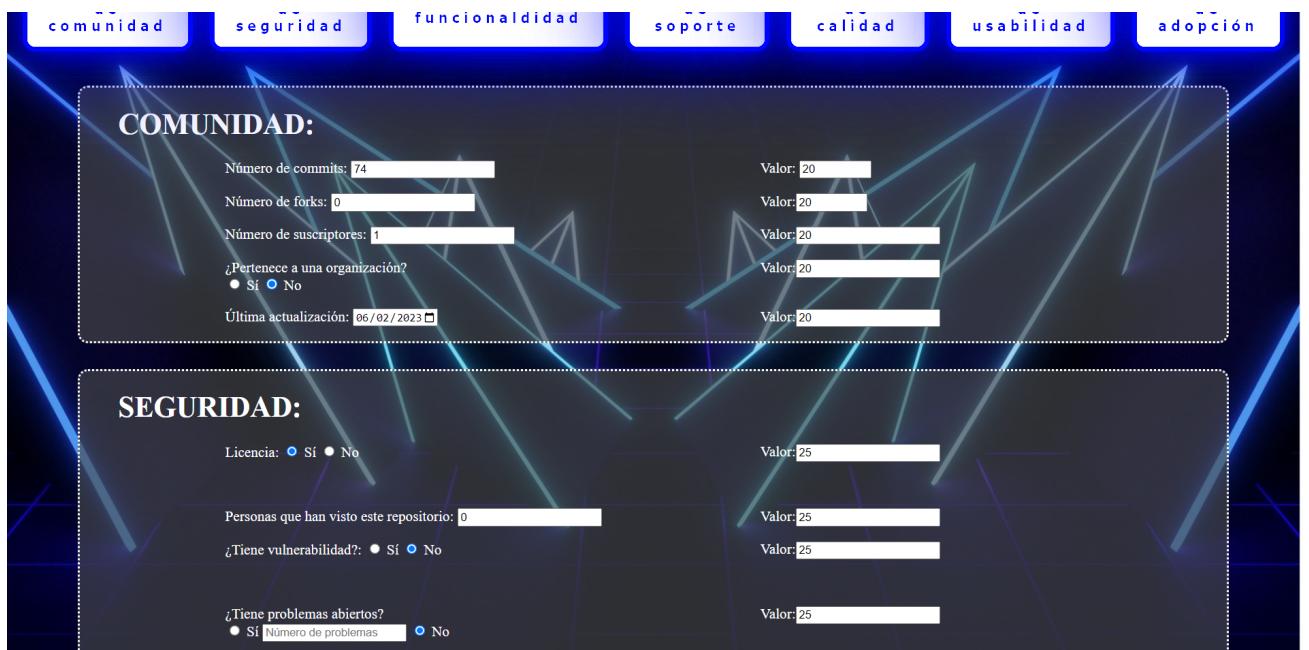


Figura 4.7: Primera parte de la página de datos.

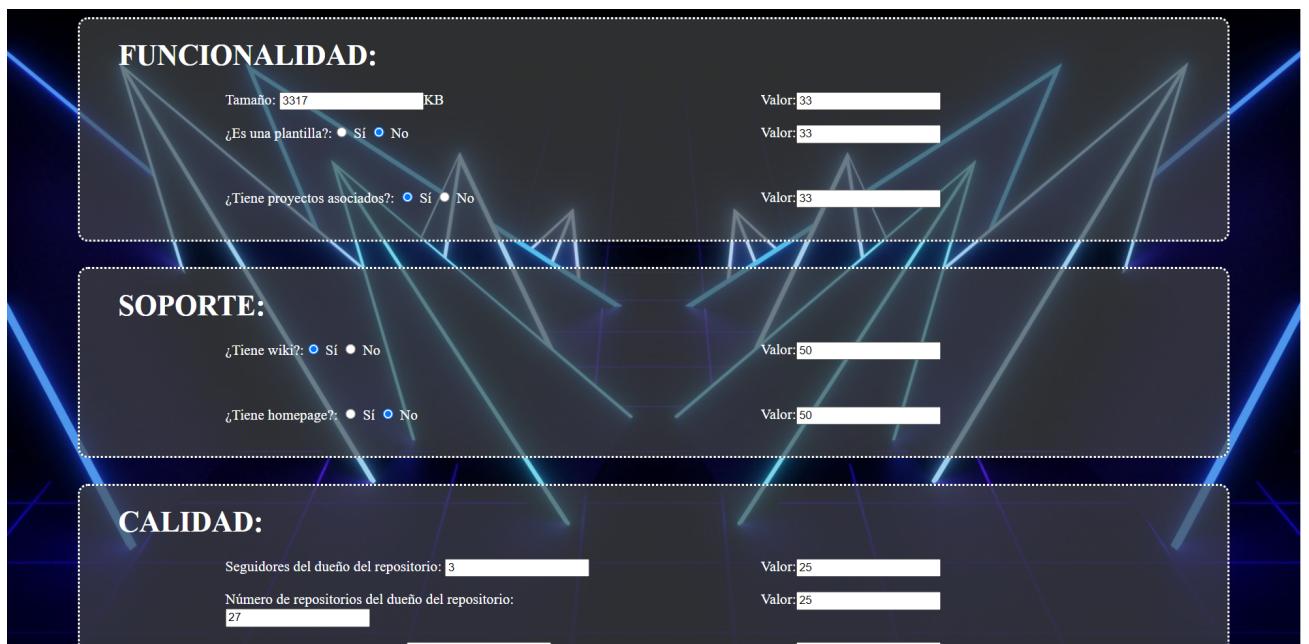


Figura 4.8: Segunda parte de la página de datos.

CALIDAD:

- Seguidores del dueño del repositorio: 3 Valor: 25
- Número de repositorios del dueño del repositorio: 27 Valor: 25
- Seguidores de la organización: 0 Valor: 25
- Número de repositorios de la organización: 0 Valor: 25

USABILIDAD:

- Número de lenguajes utilizados: 4 Valor: 50
- ¿Tiene README?: Sí No Valor: 50

ADOPCIÓN:

- ¿Tiene descargas?: Sí No Valor: 100

Dirección de correo a la que enviar los resultados (OPCIONAL):

Figura 4.9: Tercera parte de la página de datos.



Figura 4.10: Primera parte de la página de resultados.

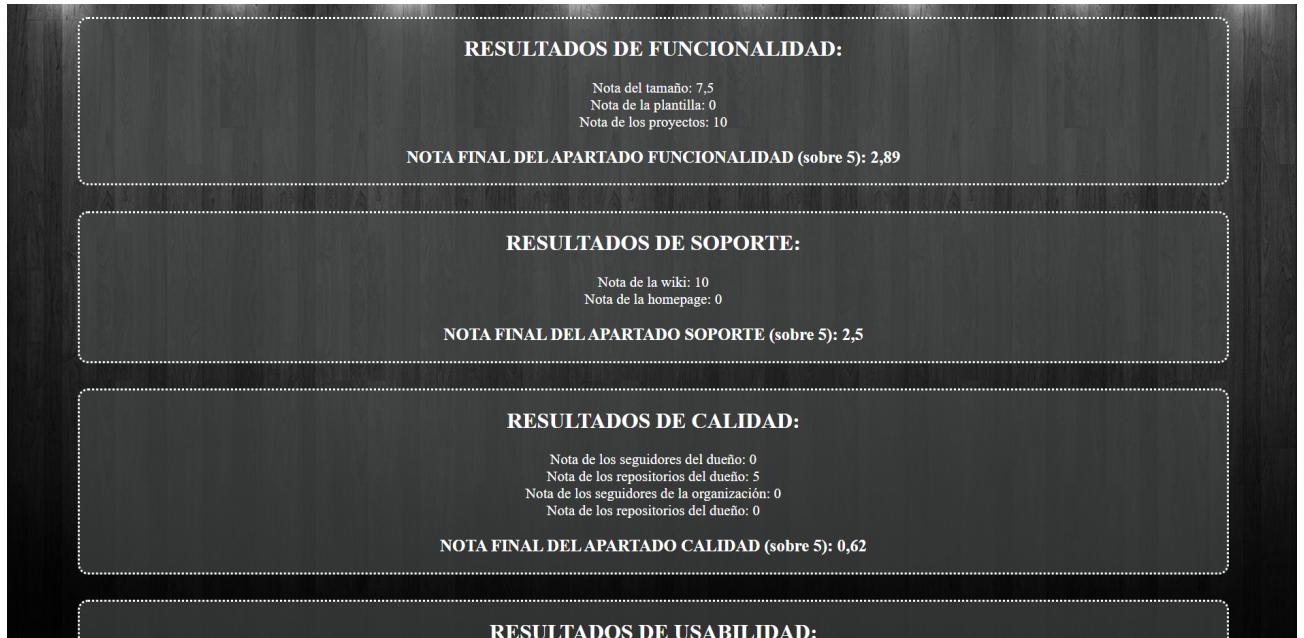


Figura 4.11: Segunda parte de la página de resultados.



Figura 4.12: Tercera parte de la página de resultados.

Capítulo 5

Experimentos y validación

En este capítulo se explicarán las diferentes pruebas que se fueron haciendo durante el desarrollo de este proyectos con los diferentes paquetes y frameworks.

5.1. Perceval

Esta fue la librería propuesta inicialmente por mi tutor. Realicé varias pruebas siguiendo los pasos indicados en su página¹ pero debido a su complejidad y que solo conseguía que funcionara en Linux² se terminó descartando en favor de PyGithub. En la figura 5.1 podemos observar un programa de prueba con Perceval que extraía los "pull request" y los "issues" de un repositorio.

5.2. PyGithub

Por los motivos explicados anteriormente se descartó Perceval y se reemplazó por PyGithub. Al principio, se hicieron diferentes pruebas con diferentes funciones de esta librería para comprobar su correcto funcionamiento antes de implementarla en la aplicación final. Se seleccionó como repositorio de pruebas el propio de PyGithub³ y se extrajeron diferentes datos que fueran fácilmente comprobables desde la página de Github. De la figura 5.2 a la figura 4.9 podemos ver el programa de prueba resultante y en la figura 5.5 su salida. Finalmente, de la figura 5.6 a

¹<https://perceval.readthedocs.io/en/latest/perceval/git.html#using-perceval-as-a-python-module>
²<https://www.linux.org/>
³<https://github.com/pygithub/pygithub>

```
import argparse

from perceval.backends.core.github import GitHub

# Parse command line arguments
parser = argparse.ArgumentParser(
    description= "Simple parser for Github issues and pull requests"
)
parser.add_argument("-t", "--token",
                    '--nargs', nargs='+',
                    help= "Github token")
parser.add_argument("-r", "--repo",
                    help= "Github repository, as 'owner/repo'")
args = parser.parse_args()

# Owner and repository names
(owner, repo) = args.repo.split('/')

# Create a Git object, pointing to repo_url, using repo_dir for cloning
repo = GitHub(owner=owner, repository=repo, api_token=args.token)
# Fetch all issues/pull requests as an iterator, and iterate it printing
# their number, and wether they are issues or pull requests
for item in repo.fetch():
    if 'pull_request' in item['data']:
        kind = 'Pull request'
    else:
        kind = 'Issue'
    print(item['data']['number'], ':', kind)
```

Figura 5.1: Programa de prueba de Perceval.

la figura 5.8 observamos los datos mostrados por el propio repositorio y comprobamos que se corresponden con los obtenidos por la salida del programa de prueba. Por tanto, el programa funciona correctamente.

5.3. Pruebas con la aplicación

Una vez comprobado el correcto funcionamiento de PyGithub, se implementa en la aplicación junto con Django. Para corroborar que las peticiones se envían y el repositorio se analiza correctamente, rellenamos el formulario inicial con el repositorio de PyGithub (como en el apartado anterior) para ver si los datos obtenidos coinciden. La página principal queda tal y como observamos en la figura 5.9. Finalmente, en la ventana de datos que podemos ver en la figura 5.10 observamos que los datos obtenidos coinciden con los obtenidos en la figura 5.5 y que, por tanto, la implementación ha sido un éxito.

5.3.1. Pruebas con views.py

Como se ha explicado en el apartado 4.1.1 este fichero es muy importante para el correcto funcionamiento de la aplicación, por ello es de vital importancia asegurarse de que funciona perfectamente. Debido a su extensión, se explicarán las pruebas que se hicieron poniendo de ejemplo la pestaña comunidad de la figura 5.10 y se darán por explicadas todas las demás ya que, además, su funcionamiento es exactamente el mismo.

Views.py se encarga, entre otras muchas cosas, de recoger los datos enviados por el usuario en el formulario y obtener la calificación de cada apartado y la calificación final del repositorio. Para la pestaña comunidad, como no se han modificado los valores de la aportación de cada dato, se obtiene su calificación final sobre 5 con la siguiente fórmula:

```
nota_final = (nota_commits * 0.2 + nota_forks * 0.2  
+ nota_suscriptores * 0.2 + nota_organización * 0.2  
+ nota_actualización * 0.2) / 2
```

Para el cálculo de la calificación de cada dato en concreto se usan los métodos que podemos ver de la figura 5.11 a la figura 5.15 en función del número introducido por el usuario en el formulario de la figura 5.10. Cabe destacar que, en caso de tratarse de un formulario de "sí o

```
from github import Github
from github.GithubException import RateLimitExceededError
from datetime import datetime

def to_valid_format(date):
    if (date < 10):
        return '0' + str(date)
    else:
        return str(date)

# Introducimos la token del usuario
print("Key (si no la tiene pulse INTRO):")
key = input()
# Si tenemos la clave también podemos hacer esto
if key != "":
    g = Github(key)
    #Contamos los commits de cada repositorio del usuario
    for repoGit in g.get_user().get_repos():
        print(repoGit.name, repoGit.get_commits().totalCount)
else:
    g = Github()
repo_path = "PyGithub/PyGithub"

try:
    # Introducimos la dirección del repo
    repo = g.get_repo(repo_path)
    contents = repo.get_contents("") # Obtenemos su contenido
    for content_file in contents: # Comprobamos si alguno es el "LICENSE"
        if (content_file.name[:7] == "LICENSE"):
            print("TIENE LICENCIA. CHECK")
```

Figura 5.2: Primera parte del programa de prueba de PyGithub.

```
if (content_file.name[:7] == "LICENSE"):
    print("TIENE LICENCIA. CHECK")
    file = content_file.name
    print("Content name:", content_file.name)

# Mostramos los commits del repo seleccionado
print(repo.name)
print("COMMITES: ", repo.get_commits().totalCount)

# Mostramos lenguajes utilizados
print("Lenguaje: ", list(repo.get_languages().keys()))

#Mostramos la última vez que se actualizó
fecha = repo.pushed_at
print("Actualizado por última vez:", repo.pushed_at)

print("Tiene wiki:", repo.has_wiki)

print("Forks:", repo.forks_count)

organizacion = repo.organization
print("¿Pertenece a una organización?", repo.organization)
#print("Login: ", organizacion.login)

has_dld = repo.has_downloads
downloads = repo.get_downloads()
#download = repo.get_download()
#deployment = repo.get_deployment()
deployments = repo.get_deployments()
print("DESCARGAS: ", has_dld, downloads.totalCount)
readme = repo.get_readme()
```

Figura 5.3: Segunda parte del programa de prueba de PyGithub.

```

print("Número de suscriptores: ", repo.subscribers_count)
fecha_d = str(fecha.date())
fecha_h = fecha.ctime()
now = str(datetime.now())
day = to_valid_format(repo.pushed_at.day)
month = to_valid_format(repo.pushed_at.month)
now_day = to_valid_format(datetime.now().day)
num_leng = len(list(repo.get_languages().keys()))
vulne = repo.get_vulnerability_alert()
work = repo.get_workflows()
#traffic = repo.get_views_traffic()
print("p")
#stats = repo.get_stats_code_frequency()
#work = repo.get_workflow()
#print(work)
except RateLimitExceededError:
    print("Número máximo de peticiones alcanzadas. Inténtelo de nuevo más tarde.")

```

Figura 5.4: Tercera parte del programa de prueba de PyGithub.

```

PS C:\Users\ivijo\Documents\TFG> & C:/Python310/python.exe c:/Users/ivijo/Documents/TFG/main.py
Key (si no la tiene pulse INTRO):

PyGithub
COMMIT: 2011
Lenguaje: ['Python', 'Shell']
Actualizado por última vez: 2023-02-06 17:29:57
Tiene wiki: False
Forks: 1595
¿Pertenece a una organización? Organization(login="PyGithub")
DESCARGAS: True 0
Número de suscriptores: 114

```

Figura 5.5: Salida del programa de prueba de PyGithub.

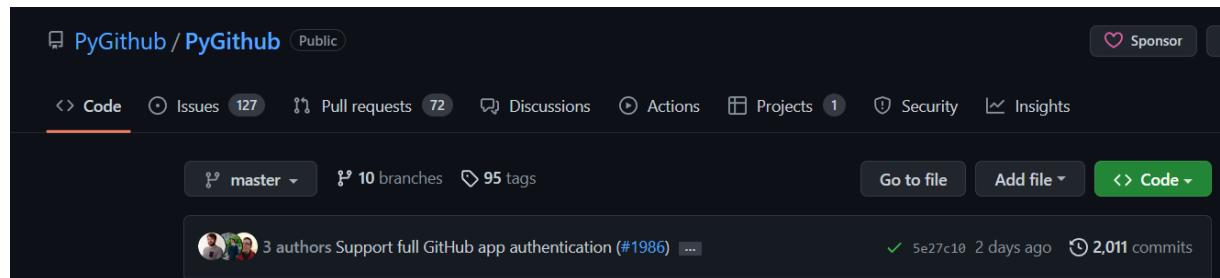


Figura 5.6: Número de commits de PyGithub.

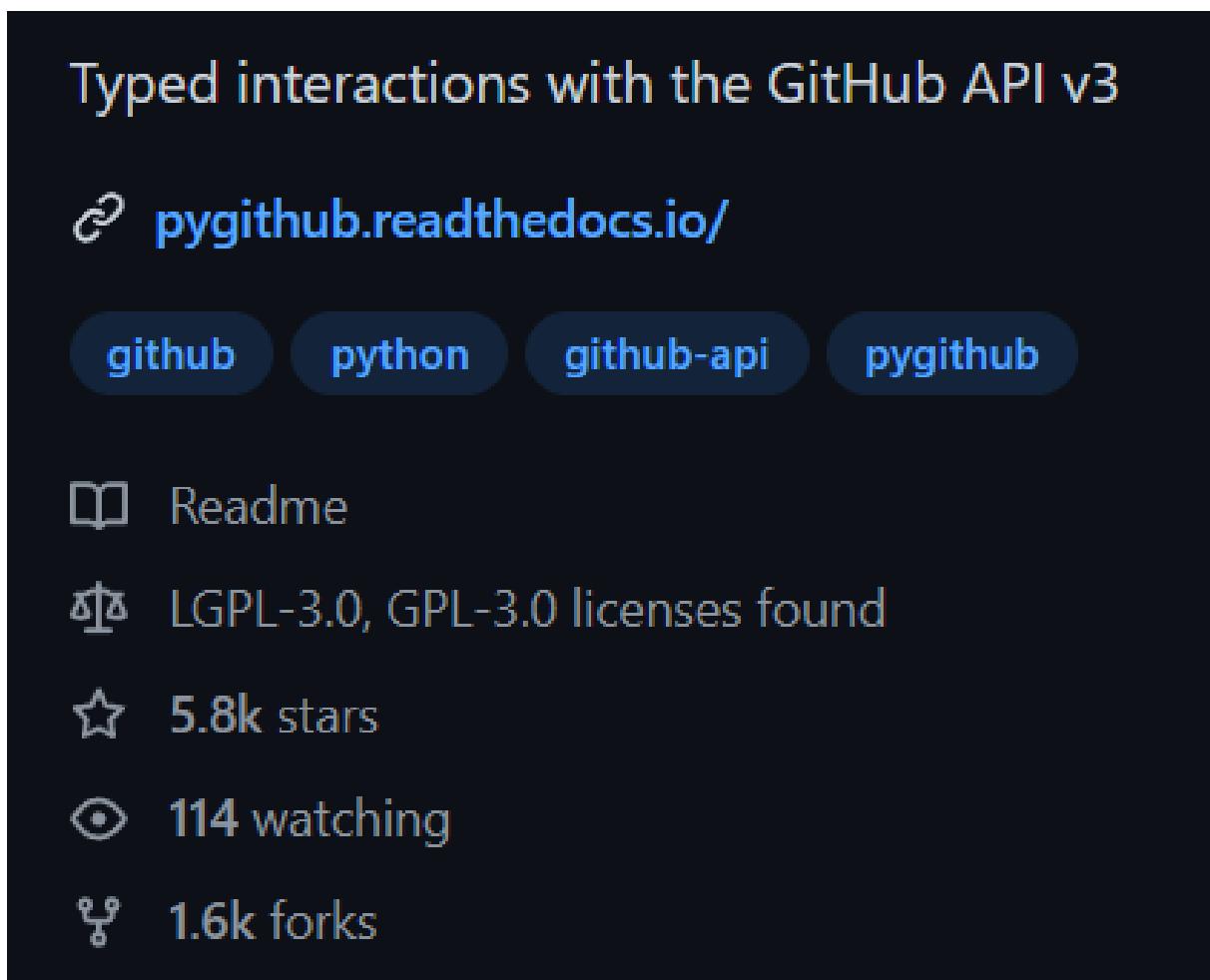


Figura 5.7: Número de forks de PyGithub.

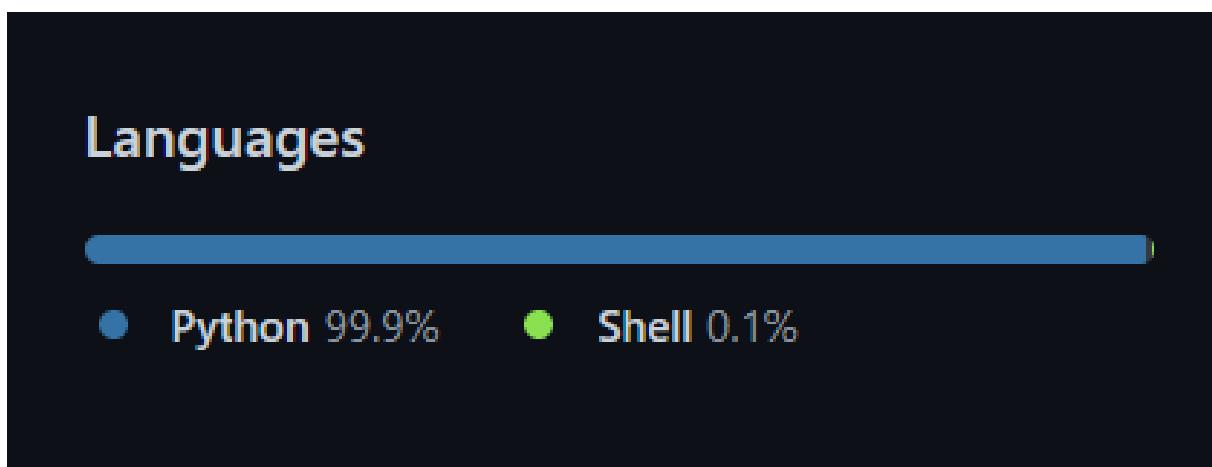


Figura 5.8: Lenguajes utilizados en el repositorio de PyGithub.



Figura 5.9: Página principal antes de analizar PyGithub.

no” la nota es binaria, es decir, un 10 si cumple el requisito o un 0 si no lo cumple. Esto lo podemos ver en la figura 5.14.

Si obtenemos la nota final a mano obtenemos que la nota de cada apartado es:

```
nota commits = 10
nota forks = 10
nota suscriptores = 10
nota organización = 10
nota actualización = 7.5
```

Por tanto, aplicando la fórmula anterior obtenemos una calificación de 4.75 que es lo mismo que ha calculado la aplicación como vemos en la figura 5.16 así que funciona correctamente. Por último, comprobamos que obtiene de manera correcta la nota media de todos los apartados. A mano, calculamos la nota final sumando las notas de cada apartado, que podemos observar de la figura 5.16 a la figura 5.22 y dividiendo entre 7 para obtener una nota final de:

```
nota final = (4.75 + 4.38 + 2.06 + 2.5 + 0 + 5 + 5) / 7 = 3.38
```

Como vemos en esta última figura, la aplicación ha calculado la nota correctamente y podemos concluir que funciona.

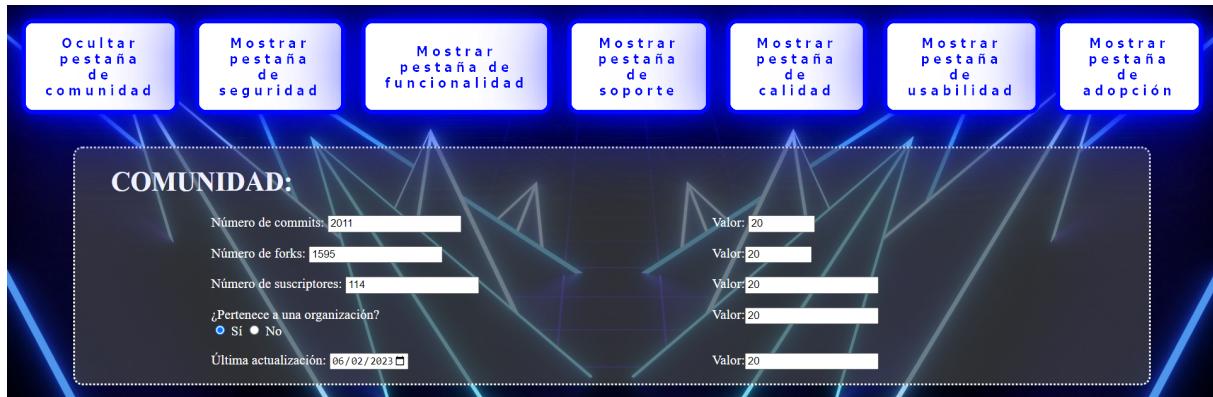


Figura 5.10: Pestaña de comunidad de PyGithub.

```

def calc_nota_commits(com):
    # Calculamos la nota de los commits
    if (com > 0) and (com <= 20):
        nota_commits = 2.5
        print(com)
    elif (com > 20) and (com <= 60):
        nota_commits = 5
    elif (com > 60) and (com <= 100):
        nota_commits = 7.5
    elif com > 100:
        nota_commits = 10
    else:
        nota_commits = 0
    return nota_commits #-- Nota ponderada con su peso

```

Figura 5.11: Obtención de la calificación en función del número de commits.

```
def calc_nota_forks(forks):
    # Calculamos la nota de los forks
    if (forks > 0) and (forks <= 50):
        nota_forks = 2.5
    elif (forks > 500) and (forks <= 150):
        nota_forks = 5
    elif (forks > 160) and (forks <= 500):
        nota_forks = 7.5
    elif forks > 500:
        nota_forks = 10
    else:
        nota_forks = 0
    return nota_forks
```

Figura 5.12: Obtención de la calificación en función del número de forks.

```
def calc_nota_sus(sus):
    # Calculamos la nota de las suscripciones
    if (sus > 5) and (sus <= 20):
        nota_sus = 2.5
    elif (sus > 20) and (sus <= 60):
        nota_sus = 5
    elif (sus > 60) and (sus <= 100):
        nota_sus = 7.5
    elif sus > 100:
        nota_sus = 10
    else:
        nota_sus = 0
    return nota_sus
```

Figura 5.13: Obtención de la calificación en función del número de suscriptores.

```
def calc_nota_org(org):
    # Calculamos la nota de pertenecer o no a una organización
    if org == 'Sí':
        nota_org = 10
    elif org == 'No':
        nota_org = 0
    return nota_org
```

Figura 5.14: Obtención de la calificación en función de si pertenece a una organización.

```

def calc_nota_update(upd):
    # Calculamos la nota de la actualización
    now = datetime.now()
    day = to_valid_format(now.day)
    month = to_valid_format(now.month)
    year = str(now.year)
    # Separamos el día, el mes y el año de la última actualización
    upd_separado = upd.split('-')
    if year == upd_separado[0]:
        if month == upd_separado[1]:
            if day == upd_separado[2]:
                nota_upd = 10 # Se ha actualizado hoy
            else:
                nota_upd = 7.5 # Se ha actualizado este mes
        else:
            nota_upd = 5 # Se ha actualizado este año
    else:
        nota_upd = 2.5 # No se ha actualizado este año
    return nota_upd

```

Figura 5.15: Obtención de la calificación en función de la fecha de última actualización.



Figura 5.16: Calificación de comunidad obtenida por la aplicación.

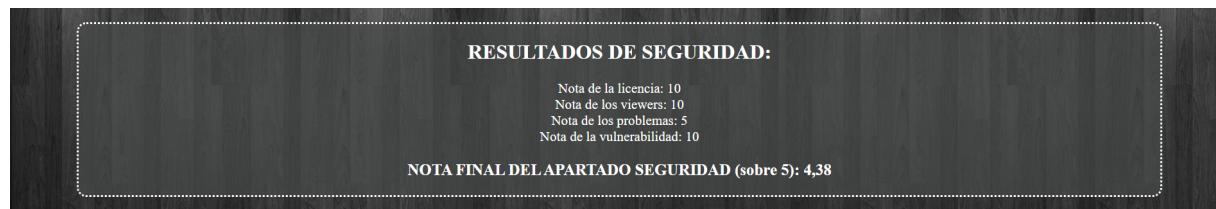


Figura 5.17: Calificación de seguridad obtenida por la aplicación.



Figura 5.18: Calificación de funcionalidad obtenida por la aplicación.



Figura 5.19: Calificación de soporte obtenida por la aplicación.



Figura 5.20: Calificación de calidad obtenida por la aplicación.

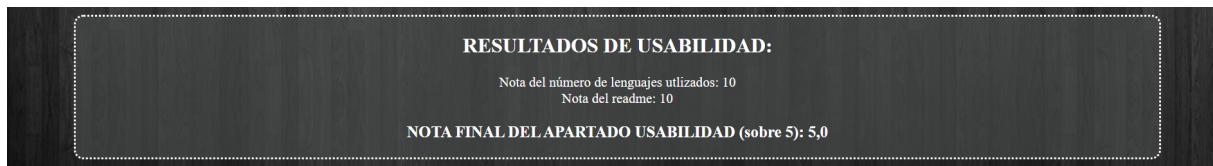


Figura 5.21: Calificación de usabilidad obtenida por la aplicación.



Figura 5.22: Calificación de adopción y nota final obtenida por la aplicación.



Figura 5.23: Formulario relleno con mi propia dirección de correo.

Envío del correo

Finalmente, el fichero `views.py` envía al usuario un mensaje a su dirección de correo que ha indicado con los mismos datos que podemos observar de la figura 5.16 a la figura 5.22 por si quiere guardar ese análisis. Para comprobar su correcto funcionamiento indico mi dirección de correo en el formulario de la figura 5.23. Como podemos observar en la figura 5.24 los datos coinciden. El envío del correo es un éxito.

i.miguel.molinero@gmail.com
para mí ▾

COMUNIDAD
Nota de los commits: 10
Nota de los forks: 10
Nota de los suscriptores: 10
Nota de la organización: 10
Nota update: 7.5
NOTA FINAL DE LA COMUNIDAD: 4.75

SEGURIDAD
Nota de la licencia: 10
Nota de los viewers: 10
Nota de los problemas: 5
Nota de la vulnerabilidad: 10
NOTA FINAL DE LA SEGURIDAD: 4.38

FUNCIONALIDAD
Nota del tamaño: 2.5
Nota de la plantilla: 0
Nota de los proyectos: 10
NOTA FINAL DE LA FUNCIONALIDAD: 2.06

SOPORTE
Nota de la wiki: 0
Nota de la homepage: 10
NOTA FINAL DE SOPORTE: 2.5

CALIDAD
Nota de los seguidores del dueño: 0
Nota de los repositorios del dueño: 0
Nota de los seguidores de la organización: 0
Nota de los repositorios de la organización: 0
NOTA FINAL DE CALIDAD: 0.0

USABILIDAD
Nota del número de lenguajes: 10
Nota del README: 10
NOTA FINAL DE USABILIDAD: 5.0

ADOPCIÓN
Nota de las descargas: 10
NOTA FINAL DE ADOPCIÓN: 5.0

NOTA FINAL DEL REPOSITORIO: 3.38

Figura 5.24: Mensaje enviado por la aplicación.

Capítulo 6

Resultados

En este capítulo se incluyen los resultados de tu trabajo fin de grado.

Si es una herramienta de análisis lo que has realizado, aquí puedes poner ejemplos de haberla utilizado para que se vea su utilidad.

Capítulo 7

Conclusiones

7.1. Consecución de objetivos

Esta sección es la sección espejo de las dos primeras del capítulo de objetivos, donde se planteaba el objetivo general y se elaboraban los específicos.

Es aquí donde hay que debatir qué se ha conseguido y qué no. Cuando algo no se ha conseguido, se ha de justificar, en términos de qué problemas se han encontrado y qué medidas se han tomado para mitigar esos problemas.

Y si has llegado hasta aquí, siempre es bueno pasarle el corrector ortográfico, que las erratas quedan fatal en la memoria final. Para eso, en Linux tenemos aspell, que se ejecuta de la siguiente manera desde la línea de *shell*:

```
aspell --lang=es_ES -c memoria.tex
```

7.2. Aplicación de lo aprendido

Aquí viene lo que has aprendido durante el Grado/Máster y que has aplicado en el TFG/TFM. Una buena idea es poner las asignaturas más relacionadas y comentar en un párrafo los conocimientos y habilidades puestos en práctica.

1. a

2. b

7.3. Lecciones aprendidas

Aquí viene lo que has aprendido en el Trabajo Fin de Grado/Máster.

1. Aquí viene uno.
2. Aquí viene otro.

7.4. Trabajos futuros

Ningún proyecto ni software se termina, así que aquí vienen ideas y funcionalidades que estaría bien tener implementadas en el futuro.

Es un apartado que sirve para dar ideas de cara a futuros TFGs/TFMs.

Apéndice A

Manual de usuario

Esto es un apéndice. Si has creado una aplicación, siempre viene bien tener un manual de usuario. Pues ponlo aquí.

Bibliografía

[1] Css.

<https://developer.mozilla.org/es/docs/Web/CSS>.

[2] Djangogirls.

<https://djangogirls.org/es/>.

[3] Github.

<https://docs.github.com/es>.

[4] Html.

<https://developer.mozilla.org/es/docs/Web/HTML>.

[5] Javascript.

<https://developer.mozilla.org/es/docs/Web/JavaScript>.

[6] Middleware.

<https://web.archive.org/web/20050507151935/http://middleware.objectweb.org/>.

[7] Openbrr.

<https://web.archive.org/web/20050803022846/http://www.openbrr.org/>.

[8] Pygithub.

<https://github.com/pygithub/pygithub>.

[9] Python.

<https://web.archive.org/web/20200224120525/https://luca-d3.com/es/dataspeaks/diccionario-tecnologico/python-lenguaje>.

- [10] Repositorio de mi trabajo de fin de grado.
[https://github.com/ivanmiguelmolinero/TFG.](https://github.com/ivanmiguelmolinero/TFG)