



**Universidad
Rey Juan Carlos**

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

Curso Académico 2022/2023

Trabajo Fin de Grado

**EVALUACIÓN DE REPOSITORIOS DE GITHUB
CON OPENBRR**

Autor : Iván Miguel Molinero

Tutor : Dr. Gregorio Robles Martínez

Trabajo Fin de Grado

Evaluación de Repositorios de GitHub con OpenBRR

Autor : Iván Miguel Molinero

Tutor : Dr. Gregorio Robles Martínez

La defensa del presente Proyecto Fin de Carrera se realizó el día de 2023, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a _____ de _____ de 2023

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

Maestro Yoda

Agradecimientos

Siete años después he conseguido llegar hasta aquí, aunque hubo momentos en los que quise abandonar, nunca me rendí y he logrado terminar la carrera. Esto no hubiera sido posible, en primer y principal lugar, sin mis padres. Siempre han estado ahí, apoyándome, animándome, pagándome todo lo que necesitara, valiera lo que valiera como el ordenador desde el que escribo estas líneas o las asignaturas en las que llegué a tercera matrícula o incluso quinta. Gracias, Jose y Yoli, gracias, papá y mamá, sin vosotros esto hubiera sido imposible. Os quiero. De esta carrera también me llevó a una nueva familia. ¿Quién iba a decirme que terminaría amando lo que en 2017 describía como una pedanía? Hablo de Jaén y del hermano y hermana que me llevó para siempre. Gracias por confiar en mí incluso cuando ni yo lo hacía. Siempre habéis estado ahí para mí, escuchando cada audio durase lo que durase, hablase de lo que hablase, desde lo enfadado que estaba por que había perdido el Madrid hasta mis penas y alegrías. Me enseñasteis lo que es una amistad de verdad, gracias. Agradecer también a personas que pasaron por mi vida durante esta etapa y que me enseñaron a “no ocuparme antes de ocuparme”, a mi psicóloga que me enseñó que casi nada esta vida es catastrófico y me dio las herramientas para poder sacar mi mejor versión cuando ni yo sabía lo que me pasaba. También quiero agradecer a todos los compañeros que he tenido en la carrera, entre todos nos hicimos la vida un poco más fácil. Gracias, claro, a mi equipo, el Real Madrid que me enseñó a luchar “hasta el final” como reza el tatuaje que llevo en el brazo. Para terminar, dar las gracias al C.E.I.P. Pío Baroja, al I.E.S. Manuel de Falla de Móstoles y a la Universidad Rey Juan Carlos de Fuenlabrada por formarme como persona y como profesional así como a los espectaculares profesores que he tenido a lo largo de mi vida como Sara, Diego, Susana, Juan y, en especial, mi tutor del TFG quien desde el minuto uno me transmitió su pasión por Python y la programación. Gracias a todos y a por más.

Resumen

Inicié este proyecto con el objetivo de seguir creciendo como desarrollador y, a la vez, crear una herramienta que fuera de utilidad para más desarrolladores como yo. A la hora de buscar herramientas en GitHub impresiona la gran cantidad de ellas que hay y, para desarrolladores novatos como yo (y no tan novatos) es difícil distinguir las mejores opciones.

Es por eso que nació este proyecto, para permitir a los desarrolladores analizar los repositorios de una manera más rápida y eficaz. El objetivo final sería poder analizar una gran cantidad de repositorios al mismo tiempo con esta aplicación.

Este proyecto permitirá a los desarrolladores evaluar los repositorios de GitHub con el sistema de análisis de OpenBRR que da una calificación sobre 5 de proyectos de código abierto. Además, el análisis podrá ser personalizado, permitiendo al usuario centrarse en los parámetros que más le convengan.

Estos objetivos no son nuevos, ya los han tenido diferentes proyectos como OpenBRR, el cual se eligió usar y ampliar ya que se centra en los aspectos más importantes de los proyectos. Además, es de código abierto, lo que siempre facilita las cosas.

Pero el uso de esta aplicación no está centrado solo en desarrolladores noveles, también se puede usar en el ámbito educativo para que los profesores puedan elegir qué repositorios recomendar a sus alumnos.

Finalmente, indicar que este proyecto se ha realizado con Python, Django, HTML, CSS y JS, tecnologías que he ido aprendiendo durante mi estancia en la universidad.

Summary

I started this project with the aim of continuing to grow as a developer and, at the same time, to create a tool that would be useful for more developers like me. When looking for tools on GitHub, it is impressive how many tools there are and, for novice developers like me (and not so novice) it is difficult to distinguish the best options.

That's why this project was born, to allow developers to analyse repositories in a faster and more efficient way. The ultimate goal would be to be able to analyse a large number of repositories at the same time with this application.

This project will allow developers to evaluate GitHub repositories with the OpenBRR analysis system that gives a score out of 5 for open source projects. In addition, the analysis will be customisable, allowing the user to focus on the parameters that suit them best.

These goals are not new, they have already been achieved by different projects such as OpenBRR which was chosen to use and extend as it focuses on the most important aspects of the projects. Moreover, it is open source, which always makes things easier.

But the use of this application is not only focused on novice developers, it can also be used in the educational field so that teachers can choose which repositories to recommend to their students.

Finally, this project has been done with Python, Django, HTML, CSS and JS, technologies that I have been learning during my stay at the university.

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. Commits	10
3.3.3. Pull request	10
3.3.4. Wiki	11
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. Envío del correo	18
4.1.2. PyGitHub	19
4.1.3. Django	21
4.1.4. Javascript	35
4.1.5. Plantillas HTML	36
4.1.6. Ficheros CSS	37
5. Experimentos y validación	39
5.1. Perceval	39
5.2. PyGitHub	39
5.3. Pruebas con la aplicación	41
5.3.1. Ejemplo de uso	41
6. Resultados	51
6.1. Prueba con myTeachingURJC/Arq-computadores-01	51
6.1.1. Análisis de myTeachingURJC/Arq-computadores-01	51
6.1.2. Resultados de myTeachingURJC/Arq-computadores-01	51
6.2. Prueba con moodle/moodle	54
6.2.1. Análisis de moodle/moodle	54
6.2.2. Resultados de moodle/moodle	58
6.3. Prueba con un repositorio vacío	58
6.3.1. Análisis del repositorio vacío	58
6.3.2. Resultados del repositorio vacío	62
6.3.3. Modificación de los datos del repositorio vacío	62
7. Conclusiones	67
7.1. Consecución de objetivos	67
7.1.1. Objetivos conseguidos	67
7.1.2. Solución de los objetivos no conseguidos	68
7.2. Aplicación de lo aprendido	68

ÍNDICE GENERAL	XI
7.3. Lecciones aprendidas	69
7.4. Trabajos futuros	70
A. Manual de usuario	71
A.1. Instalación	71
A.1.1. Configuración del correo.	71
A.2. Iniciar la aplicación.	72
A.3. Uso de la aplicación.	72
Bibliografía	73

Índice de figuras

3.1. Fragmento de 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	19
4.3. Diagrama de funcionalidad de PyGitHub	20
4.4. Objeto repositorio de PyGitHub	22
4.5. Overflow.html	23
4.6. Página principal con el formulario relleno	23
4.7. Diagrama de la función principal de views.py	24
4.8. Obtención de la calificación en función del número de commits.	27
4.9. Obtención de la calificación en función del número de forks.	27
4.10. Obtención de la calificación en función del número de suscriptores.	28
4.11. Obtención de la calificación en función de si pertenece a una organización.	28
4.12. Obtención de la calificación en función de la fecha de última actualización.	29
4.13. Pestaña de comunidad de PyGitHub.	29
4.14. Calificación de comunidad obtenida por la aplicación.	30
4.15. Calificación de seguridad obtenida por la aplicación.	30
4.16. Calificación de funcionalidad obtenida por la aplicación.	30
4.17. Calificación de soporte obtenida por la aplicación.	30
4.18. Calificación de calidad obtenida por la aplicación.	30
4.19. Calificación de usabilidad obtenida por la aplicación.	31
4.20. Calificación de adopción y nota final obtenida por la aplicación.	31
4.21. Página principal antes de analizar PyGitHub.	31

4.22. Formulario rellenado con mi propia dirección de correo.	32
4.23. Mensaje enviado por la aplicación.	33
4.24. Plantilla repo_data.html	36
4.25. Plantilla result.html	36
5.1. Programa de prueba de Perceval.	40
5.2. Primera parte del programa de prueba de PyGitHub.	42
5.3. Segunda parte del programa de prueba de PyGitHub.	43
5.4. Tercera parte del programa de prueba de PyGitHub.	44
5.5. Salida del programa de prueba de PyGitHub.	44
5.6. Número de commits de PyGitHub.	44
5.7. Número de forks de PyGitHub.	45
5.8. Lenguajes utilizados en el repositorio de PyGitHub.	45
5.9. Página principal con el formulario lleno	46
5.10. Página de datos con las pestañas ocultas.	46
5.11. Primera parte de la página de datos.	47
5.12. Segunda parte de la página de datos.	47
5.13. Tercera parte de la página de datos.	48
5.14. Primera parte de la página de resultados.	48
5.15. Segunda parte de la página de resultados.	49
5.16. Tercera parte de la página de resultados.	49
6.1. Vista inicial del repositorio de “Arquitectura de computadores”.	52
6.2. Datos del repositorio de “Arquitectura de computadores”.	52
6.3. Datos del repositorio de “Arquitectura de computadores”.	53
6.4. Datos del repositorio de “Arquitectura de computadores”.	53
6.5. Resultados del repositorio de “Arquitectura de computadores”.	54
6.6. Resultados del repositorio de “Arquitectura de computadores”.	55
6.7. Resultados del repositorio de “Arquitectura de computadores”.	55
6.8. Vista inicial del repositorio de Moodle.	56
6.9. Datos del repositorio de Moodle.	56
6.10. Datos del repositorio de Moodle.	57

6.11. Datos del repositorio de Moodle.	57
6.12. Resultados del repositorio de Moodle.	58
6.13. Resultados del repositorio de Moodle.	59
6.14. Resultados del repositorio de Moodle.	59
6.15. Vista inicial del repositorio vacío.	60
6.16. Datos del repositorio vacío.	60
6.17. Datos del repositorio vacío.	61
6.18. Datos del repositorio vacío.	61
6.19. Resultados del repositorio vacío.	62
6.20. Resultados del repositorio vacío.	63
6.21. Resultados del repositorio vacío.	63
6.22. Datos modificados del repositorio vacío.	64
6.23. Resultados modificados del repositorio vacío.	64
6.24. Resultados finales modificados del repositorio vacío.	65
7.1. Alerta mostrada cuando hay una pestaña oculta.	69

Capítulo 1

Introducción

Actualmente se comparten muchos proyectos de software libre en Internet en general y en GitHub [5]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, puedes 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 dicho motivo, 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 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 [17] nace como una extensión de OpenBRR [1] 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. Además de OpenBRR existen otras formas de analizar proyectos como OSMM [13], OpenBQR [11] o MOSST [10] entre muchos otros, pero me decanté por esta opción al ser la que más se centraba en analizar los proyectos de software libre.

¹<https://github.com/>

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 sí 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

La memoria de este proyecto se estructura en los siguientes capítulos:

- **Capítulo 1:** se explica brevemente el origen del proyecto, sus bases y funcionamiento.
- **Capítulo 2:** se muestran los objetivos del proyecto.
- **Capítulo 3:** se explica cada tecnología usada en este proyecto.
- **Capítulo 4:** aquí se entra en detalle en el funcionamiento de la aplicación.
- **Capítulo 5:** se muestran los diferentes experimentos y se prueba el correcto funcionamiento de la aplicación.
- **Capítulo 6:** se explican los resultados obtenidos poniendo de prueba varios repositorios.
- **Capítulo 7:** se indican los resultados conseguidos y la solución a los objetivos no conseguidos.

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/>

- Tratar 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 Perceval[14] 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 DjangoGirls [3] 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 qué parámetros y, sobre todo, 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 myTeachingURJC² (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.

²<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 [16] 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 natural. 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[4] :

- 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 [15] es una biblioteca que se puede utilizar para analizar los datos del repositorio. Adicionalmente, también permite administrar recursos de GitHub como repositorios, perfiles de usuario y organizaciones. Es una biblioteca 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. Aquí podemos observar su uso en este proyecto, donde “text” hace referencia al nombre del repositorio:

```

# 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. 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.

3.3.3. Pull request

Es la funcionalidad que permite a los desarrolladores colaborar entre ellos sugiriendo cambios del programa en cuestión. El usuario hace un “pull” del repositorio para tener los cambios realizados por otros desarrolladores en local y poder trabajar sobre ellos para, posteriormente, hacer un “commit” con los cambios que considere pertinentes. De esta manera, esta es la funcionalidad que permite a los desarrolladores colaborar entre sí para la elaboración de proyectos.

³<https://code.visualstudio.com/>

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.

3.4. 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 [6] (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 de HTML

3.6. CSS

CSS [2] (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, es decir, debido a la gran cantidad de módulos diferentes y las incompatibilidades entre ellos, se decidió estandarizar la última versión estable. 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 [7] (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 [12] (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.

Como se puede ver en la figura, OpenBRR obtiene el código fuente del proyecto para analizarlo a través de unas métricas normalizadas y, posteriormente, asignarles un peso diferente a cada métrica con una distribución del 100 % por categoría. A continuación, se decide a qué parámetro, de los 7 a evaluar, pertenece cada métrica y se distribuye el peso de cada parámetro para, finalmente, obtener una calificación sobre 5 del proyecto analizado.

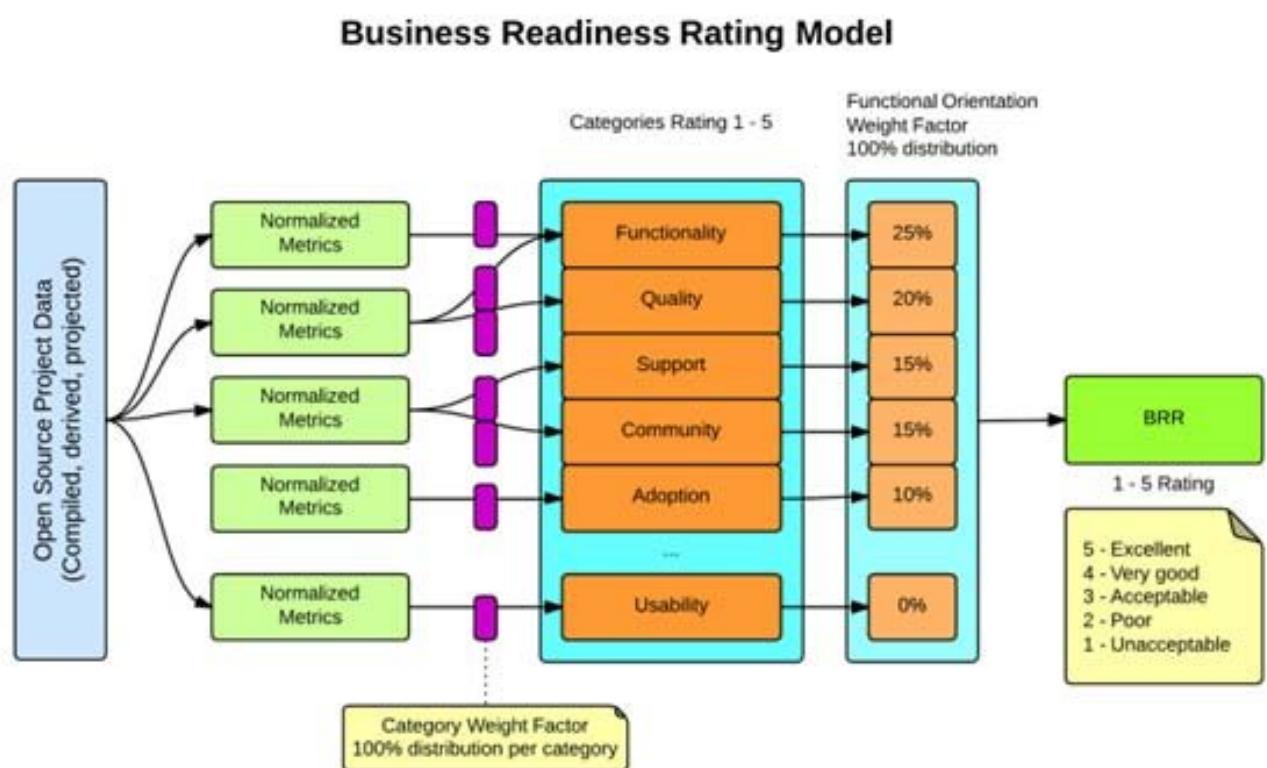


Figura 3.2: Esquema de análisis de OpenBRR

Capítulo 4

Diseño e implementación

En este capítulo se detallará cómo funciona la aplicación desde el código hasta un ejemplo de las diferentes opciones de las que dispone el usuario.

4.1. Arquitectura general

Como se puede observar en la figura 4.1 el programa principal depende del repositorio que introduzca el usuario, los datos obtenidos por PyGitHub y del repositorio de GitHub a analizar. El usuario debe introducir la dirección de un repositorio, compuesta por “usuario_de_GitHub/nombre_del_repos” 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. Estos parámetros son mostrados al usuario en la página de datos, donde se podrán manipular tanto los mismos datos como sus pesos correspondientes según las necesidades del usuario. Además, el usuario podrá introducir su dirección de correo en esta página para que le llegue un mensaje con los resultados del análisis. Finalmente, se llega a la página de calificación final donde se podrán observar los resultados del análisis.

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

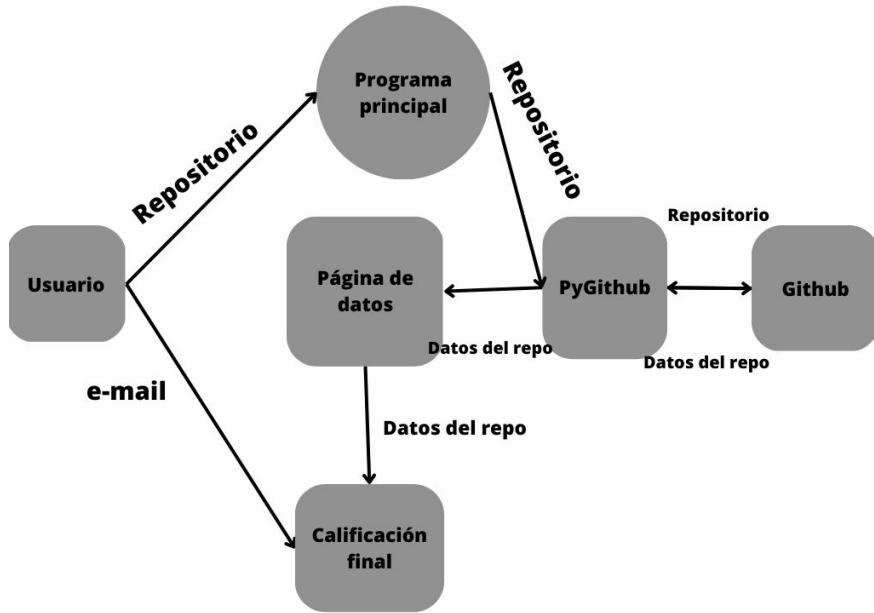


Figura 4.1: Arquitectura de la aplicación

4.1.1. Envío del correo

Para el envío del correo se utiliza la función “send_mail” de django.core.mail que 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"
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 Google³ pul-

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

³<https://myaccount.google.com/security>

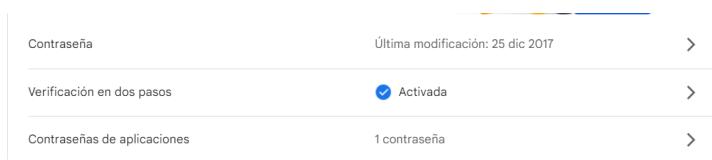


Figura 4.2: Apartado para generar la contraseña de Gmail

sando en el apartado “Contraseñas para aplicaciones”, tal y como se indica en la figura 4.2.

4.1.2. PyGitHub

PyGitHub es la biblioteca encargada de recibir la dirección del repositorio y obtener los datos necesarios para su posterior análisis. Funciona de la siguiente manera (ver figura 4.3):

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. Una vez hecho esto se obtiene un objeto repositorio como el de la figura 4.4 donde se ha puesto de ejemplo el repositorio “moodle/moodle”⁴
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 4.4 (size, pushed_at o subscribers_count son algunos ejemplos).

⁴<https://github.com/moodle/moodle>

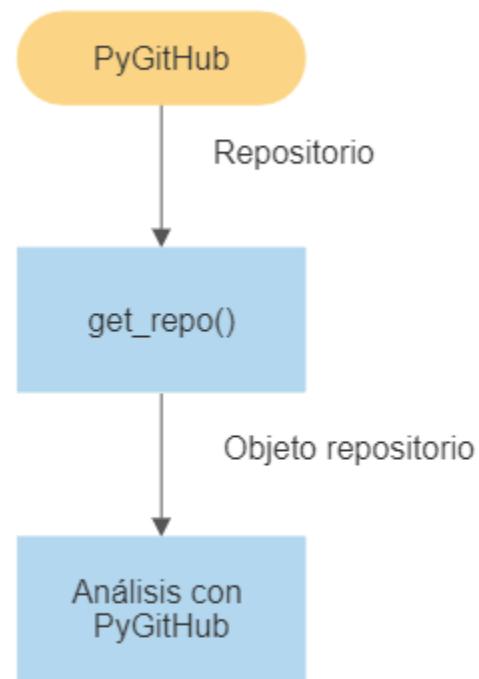


Figura 4.3: Diagrama de funcionalidad de PyGitHub

Cabe añadir la restricción con la que cuenta esta biblioteca. PyGitHub limita las peticiones que puedes realizar durante una hora y, si excedes esas peticiones, devolverá la excepción “RateLimitExceededException” con el siguiente mensaje:⁵

```
github.GitHubException.RateLimitExceededException: 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 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 4.5 donde vemos la página de error con el botón que permite volver a la página principal.

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 que es el más importante. Los formularios son enviados mediante el método GET como en el siguiente ejemplo:

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

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

⁵<https://docs.github.com/rest/overview/resources-in-the-rest-api#rate-limiting>

Figura 4.4: Objeto repositorio de PyGitHub



Figura 4.5: Overflow.html



Figura 4.6: Página principal con el formulario relleno

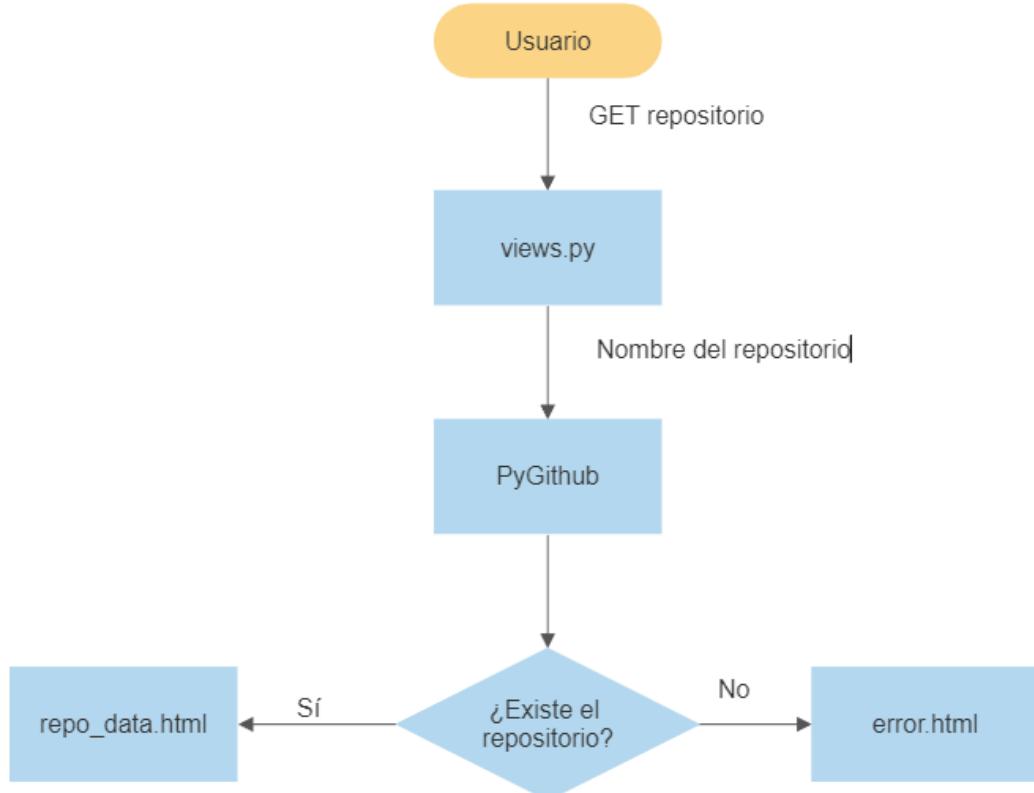


Figura 4.7: Diagrama de la función principal de `views.py`

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. Todo esto se puede observar en la figura 4.7
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 procedimiento 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.

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 4.13 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
```

Views.py también se encarga del cálculo de las calificaciones. Para el cálculo de la calificación de cada dato en concreto se usan los métodos que podemos ver de la figura 4.8 a la figura 4.12 en función del número introducido por el usuario en el formulario de la figura 4.13. Cabe destacar que, en caso de tratarse de un formulario de “sí o 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 4.11. 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 4.14 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 4.14 a la figura 4.20 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.

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 la figura 4.14 a la figura 4.20 por

```
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 4.8: 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 > 50) 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 4.9: 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 4.10: 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 4.11: 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 4.12: Obtención de la calificación en función de la fecha de última actualización.

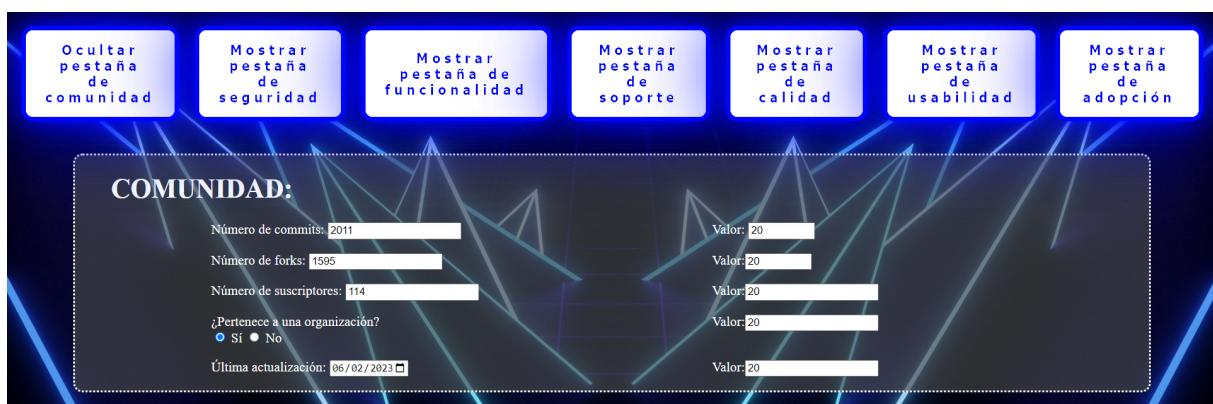


Figura 4.13: Pestaña de comunidad de PyGitHub.



Figura 4.14: Calificación de comunidad obtenida por la aplicación.

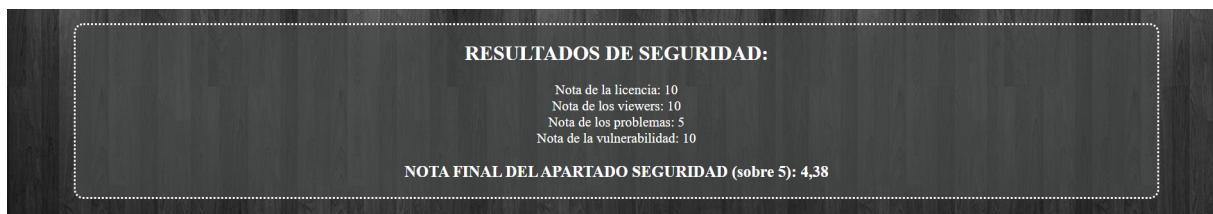


Figura 4.15: Calificación de seguridad obtenida por la aplicación.

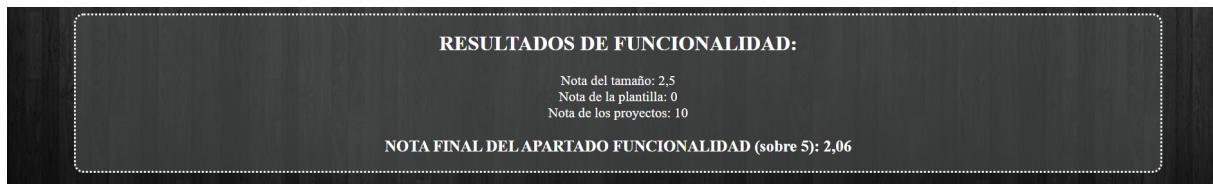


Figura 4.16: Calificación de funcionalidad obtenida por la aplicación.



Figura 4.17: Calificación de soporte obtenida por la aplicación.

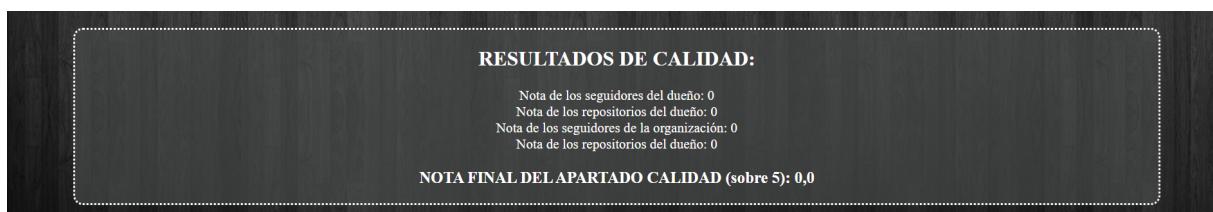


Figura 4.18: Calificación de calidad obtenida por la aplicación.

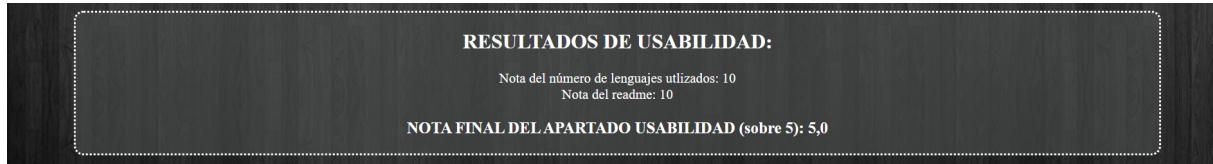


Figura 4.19: Calificación de usabilidad obtenida por la aplicación.



Figura 4.20: Calificación de adopción y nota final obtenida por la aplicación.

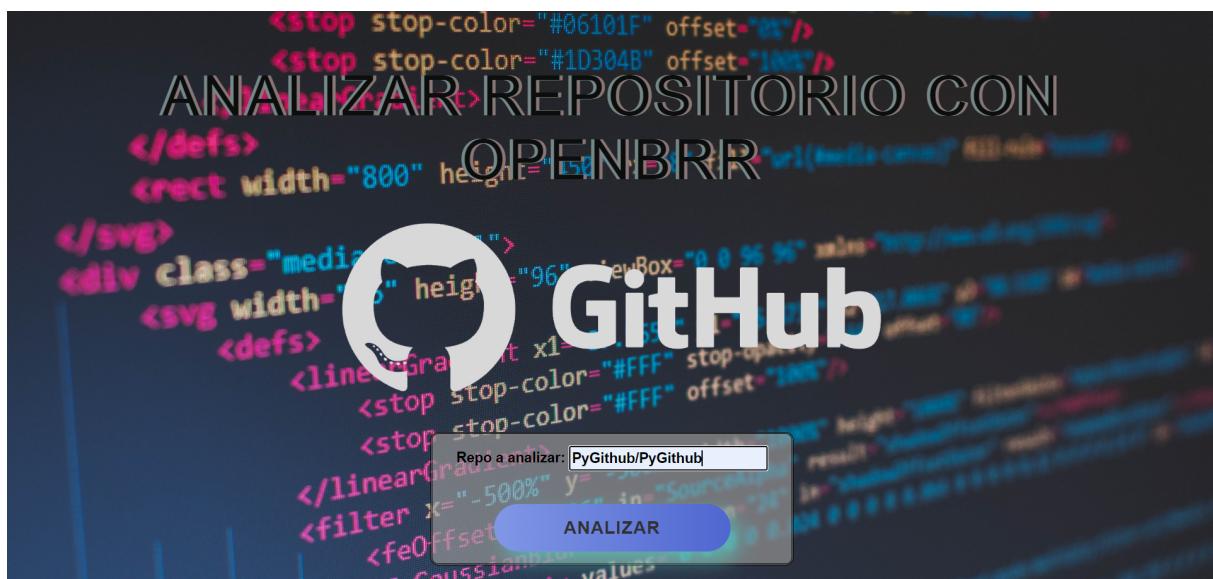


Figura 4.21: Página principal antes de analizar PyGitHub.



Figura 4.22: Formulario relleno con mi propia dirección de correo.

si quiere guardar ese análisis. Para comprobar su correcto funcionamiento indico mi dirección de correo en el formulario de la figura 4.22. Como podemos observar en la figura 4.23 los datos coinciden. El envío del correo es un éxito.

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 4.6) y el envío del formulario de repo_data.html. Las peticiones y la función que inician se recogen en la siguiente tabla:

Petición	Función (plantilla HTML)
/	views.post_main (main.html)
post	views.post_repo (repo_data.html)
get-data	views.get_data (result.html)

forms.py

Fichero que define el modelo de formulario a utilizar, el cual es simplemente un par clave-valor, donde la clave es el nombre del campo a llenar y el valor es lo que introduzca el usuario.

 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 4.23: Mensaje enviado por la aplicación.

models.py

Fichero que da forma al modelo definido en forms.py. Contiene la función “`__str__`” que permite devolver el valor introducido por el usuario.

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 [8].
- 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.

Campo	Valor
Directorio credentials.env	mysite\credentials.env
Directorio base	Path(__file__).resolve().parent.parent
Clave secreta	Clave secreta de Django
Hosts	127.0.0.0 y pythonanywhere.com
Apps instaladas	Apps necesarias para el funcionamiento de Django y OpenBRR
Idioma del lenguaje	es-es
URL y directorio raíz	/static/
Valores para el envío del correo	EMAIL_BACKEND, EMAIL_USE_TLS, EMAIL_HOST_USER EMAIL_HOST_PASSWORD EMAIL_HOST, EMAIL_PORT, y

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. Función 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.

```
<div id="content-adoption" class="information">
    <h1>ADOPCIÓN:</h1>
    {% for key, value in post_adop.items %}
        {% if key == 'downloads' %}
            {% if value == 'Sí' %}
                <p><span id="dato">¿Tiene descargas?: <input type="radio" value="Si" name="downloads" id="descargas-si" oninput="save_input_adoption('descargas-si');" checked> Si
                    <input type="radio" value="No" name="downloads" id="descargas-no" oninput="save_input_adoption('descargas-no');"> No</span>
            {% else %}
                <p><span id="dato">¿Tiene descargas?: <input type="radio" value="Si" name="downloads" id="descargas-si" oninput="save_input_adoption('descargas-si');"> Si
                    <input type="radio" value="No" name="downloads" id="descargas-no" oninput="save_input_adoption('descargas-no');" checked> No</span>
            {% endif %}
            <span id="valor">Valor:<input type="number" name="valor-descargas" value="100" step="0.1" min="0" max="100" id="valor-descargas" oninput="edit_max_value_adoption('descargas')";</p><br></span>
        {% endif %}
    {% endfor %}
</div>
```

Figura 4.24: Plantilla repo_data.html

```
<div id="result_content">
    <h2>RESULTADOS DE USABILIDAD:</h2>
    {% with nota_num_lenguajes=nota_num_lenguajes nota_readme=nota_readme nota_usabilidad=nota_usabilidad %}
        <p>Nota del número de lenguajes utilizados: {{ nota_num_lenguajes }} <br>
        Nota del readme: {{ nota_readme }} <br>
        <h3>NOTA FINAL DEL APARTADO USABILIDAD (sobre 5): {{ nota_usabilidad }}</h3>
    {% endwith %}
</div>
<div id="result_content">
```

Figura 4.25: Plantilla result.html

4.1.5. Plantillas HTML

Una plantilla HTML lo que se muestra al usuario una vez renderizadas.

Como se ve en la figura 4.25, correspondiente con repo_data.html, la plantilla se estructura sobre varios bucles “for” que van recorriendo los pares clave-valor, enviados mediante un GET, y van colocando los datos en la plantilla. En la figura ?? vemos la estructura de result.html, la cual se basa en un “with” que recoge los valores y los escribe en la plantilla de forma similar al “for” anterior.

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. En ellos, se especifica cada bloque de HTML y se define el estilo que va a tener. En el caso de este proyecto, se ha elegido unos tonos azules y negros para todas las páginas con el fin de mantener una uniformidad.

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 biblioteca 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 biblioteca para comprobar su correcto funcionamiento antes de implementarla en la aplicación final. Se seleccionó como repositorio de pruebas el propio de PyGitHub³ y se trajeron diferentes datos que fueran fácilmente comprobables desde la página de GitHub. De la figura 5.2 a la figura 5.13

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.

podemos ver el programa de prueba resultante y en la figura 5.5 su salida. Finalmente, de la figura 5.6 a 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 4.21. Finalmente, en la ventana de datos que podemos ver en la figura 4.13 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. Ejemplo de uso

De la figura 5.9 a la figura 5.16 tenemos un ejemplo analizando el repositorio de este TFG. El funcionamiento de la aplicación es el siguiente:

1. En la primera ventana el usuario introduce el repositorio que quiere analizar.
2. Posteriormente, la aplicación obtiene los datos de ese repositorio y se los muestra por si desea manipularlos o dar más importancia a algún parámetro. Todo esto funciona con formularios.
3. Al final de esta ventana se le da la opción de introducir su dirección de correo por si quisiera que le llegaran los resultados por dicha vía.
4. Una vez enviados los datos, se califican con OpenBRR dividiéndolos en 7 parámetros y dando una calificación final sobre 5 que se muestra en la última ventana.

```
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
COMMITs: 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.

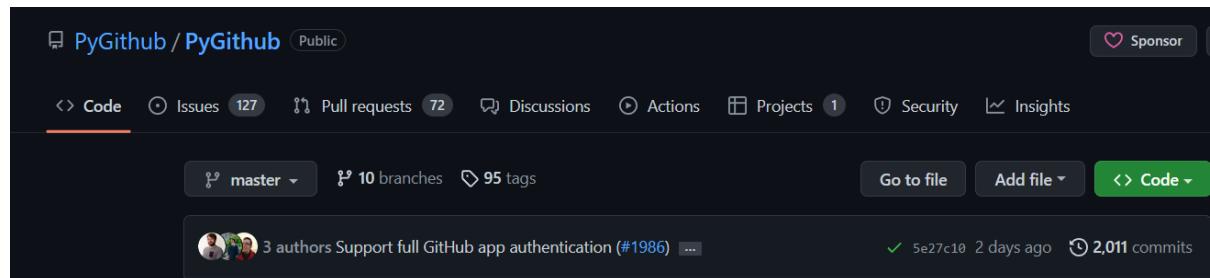


Figura 5.6: Número de commits de PyGitHub.

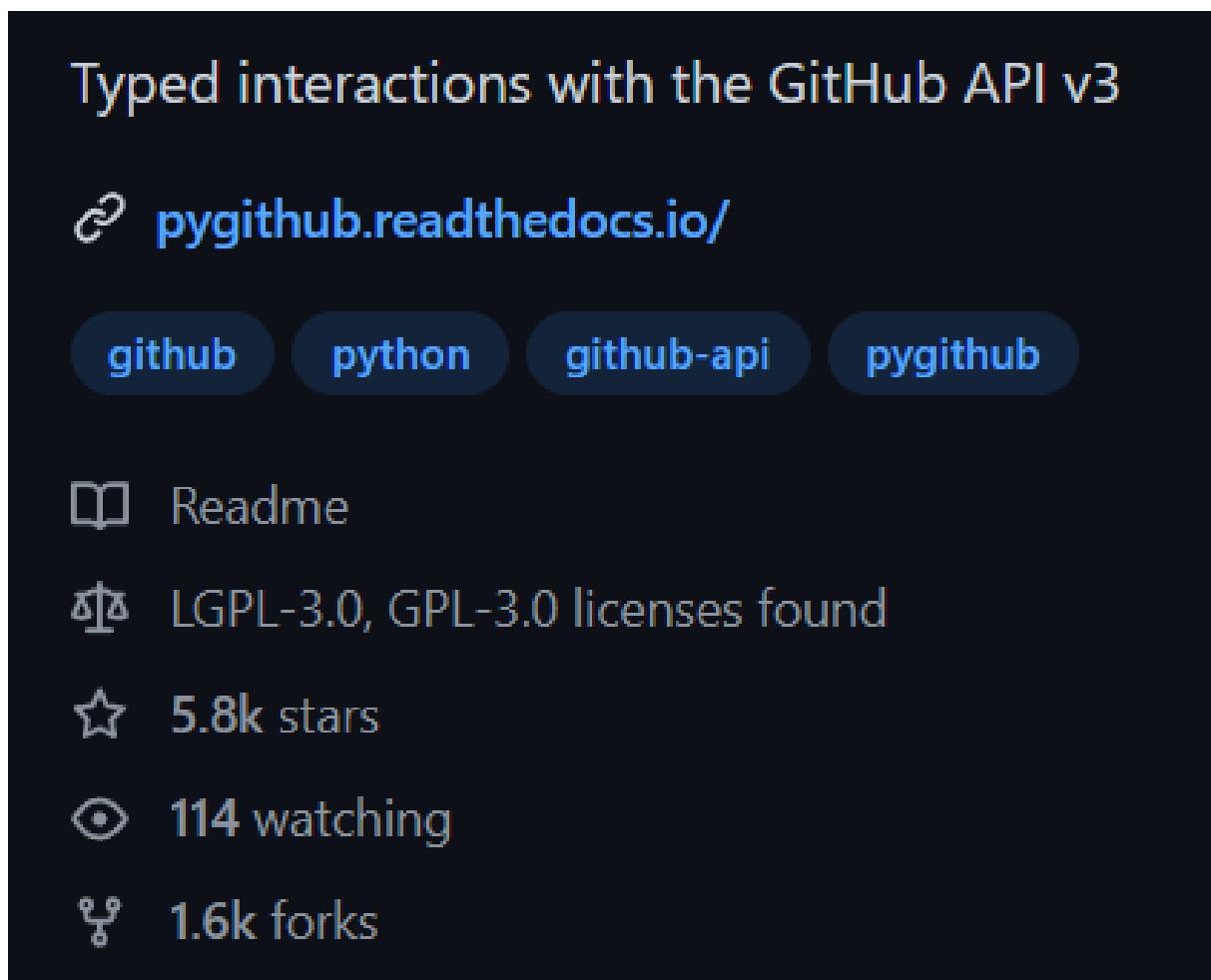


Figura 5.7: Número de forks de PyGitHub.

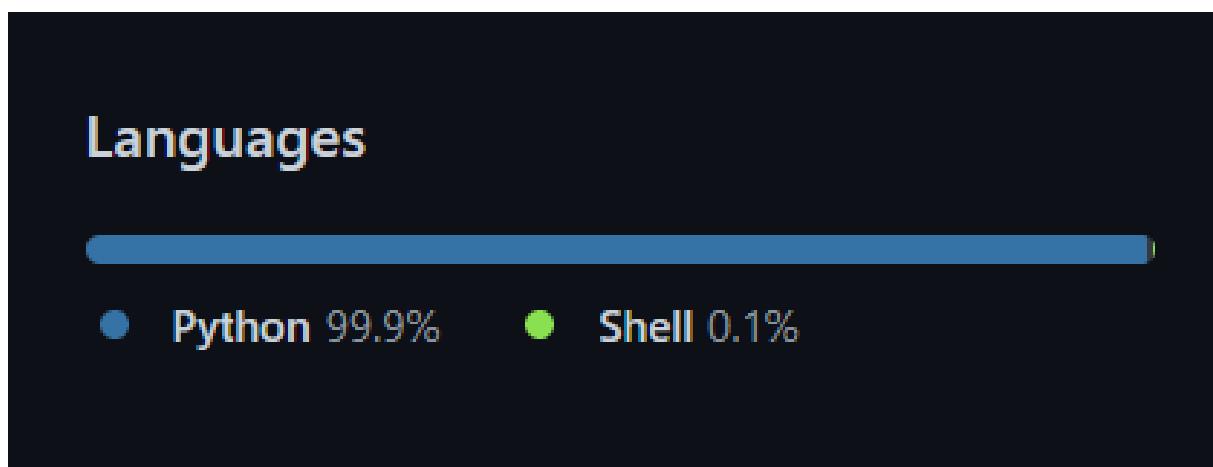


Figura 5.8: Lenguajes utilizados en el repositorio de PyGitHub.



Figura 5.9: Página principal con el formulario relleno



Figura 5.10: Página de datos con las pestañas ocultas.

COMUNIDAD:

- Número de commits: 74
- Número de forks: 0
- Número de suscriptores: 1
- ¿Pertenece a una organización?
 - Sí
 - No
- Última actualización: 06/02/2023

Valor: 20

Valor: 20

Valor: 20

Valor: 20

Valor: 20

SEGURIDAD:

- Licencia: Sí No
- Personas que han visto este repositorio: 0
- ¿Tiene vulnerabilidad?:
 - Sí
 - No
- ¿Tiene problemas abiertos?:
 - Sí | Número de problemas
 - No

Valor: 25

Valor: 25

Valor: 25

Valor: 25

Figura 5.11: Primera parte de la página de datos.

FUNCIONALIDAD:

- Tamaño: 3317 KB
- ¿Es una plantilla?:
 - Sí
 - No
- ¿Tiene proyectos asociados?:
 - Sí
 - No

Valor: 33

Valor: 33

Valor: 33

SOPORTE:

- ¿Tiene wiki?:
 - Sí
 - No
- ¿Tiene homepage?:
 - Sí
 - No

Valor: 50

Valor: 50

CALIDAD:

- Seguidores del dueño del repositorio: 3
- Número de repositorios del dueño del repositorio: 27

Valor: 25

Valor: 25

Valor: 25

Figura 5.12: 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?: <input checked="" type="radio"/> Sí <input type="radio"/> No	Valor: 50

ADOPCIÓN:

¿Tiene descargas?: <input checked="" type="radio"/> Sí <input type="radio"/> No	Valor: 100
---	------------

Dirección de correo a la que enviar los resultados (OPCIONAL):
direccion@ejemplo.com

Figura 5.13: Tercera parte de la página de datos.



Figura 5.14: Primera parte de la página de resultados.

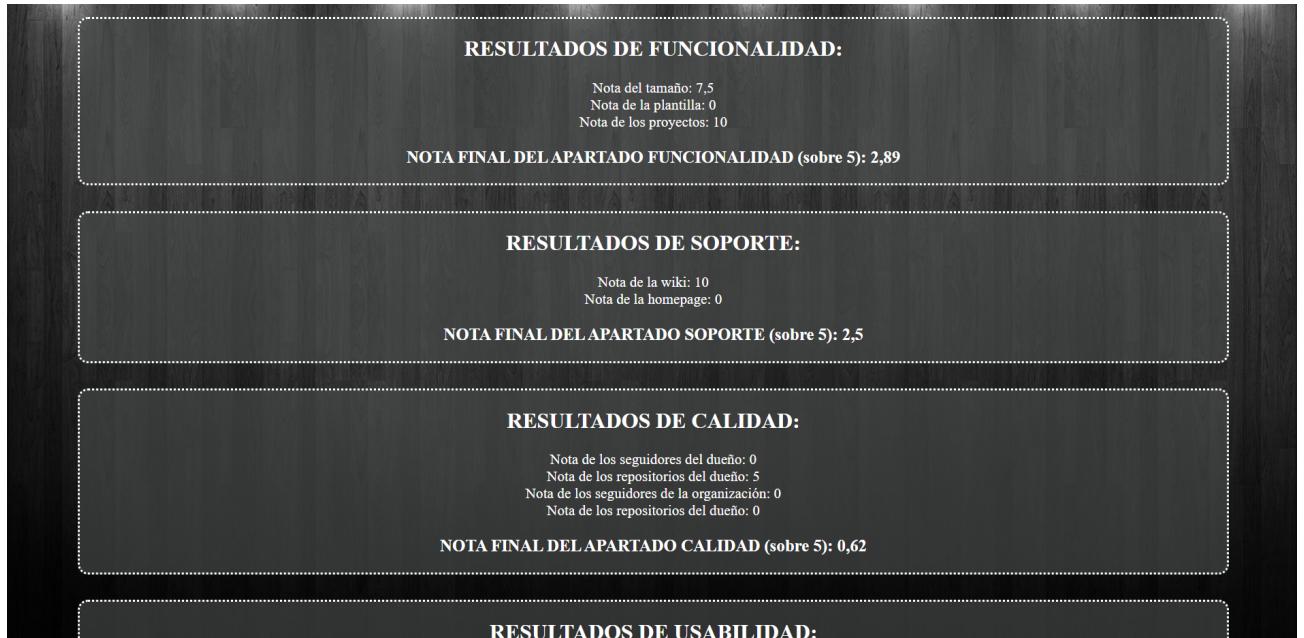


Figura 5.15: Segunda parte de la página de resultados.



Figura 5.16: Tercera parte de la página de resultados.

Capítulo 6

Resultados

Una vez hechas las pruebas explicadas en el capítulo 5 se pasa a comprobar que la aplicación devuelve resultados coherentes, es decir, que a repositorios teóricamente buenos les da una buena calificación y a repositorios prácticamente vacíos (como el de la prueba que se mostrará en el apartado 6.3) les da una mala calificación.

6.1. Prueba con myTeachingURJC/Arq-computadores-01

Este repositorio¹ es el de la asignatura “Arquitectura de computadores” de la URJC. A primera vista, como vemos en la figura 6.1, se podría decir que se trata de un buen repositorio. Vamos a analizarlo.

6.1.1. Análisis de myTeachingURJC/Arq-computadores-01

De la figura 6.2 a la figura 6.4 tenemos los datos del repositorio que vamos a mandar a través del formulario para su calificación. Como hemos visto en el capítulo 5, los datos son correctos.

6.1.2. Resultados de myTeachingURJC/Arq-computadores-01

A continuación, se ven los resultados del análisis del repositorio. Véase de la figura 6.5 a la figura 6.7. Obtenemos una calificación final de 3.13 sobre 5, lo que es una calificación coherente

¹<https://github.com/myTeachingURJC/Arq-computadores-01>

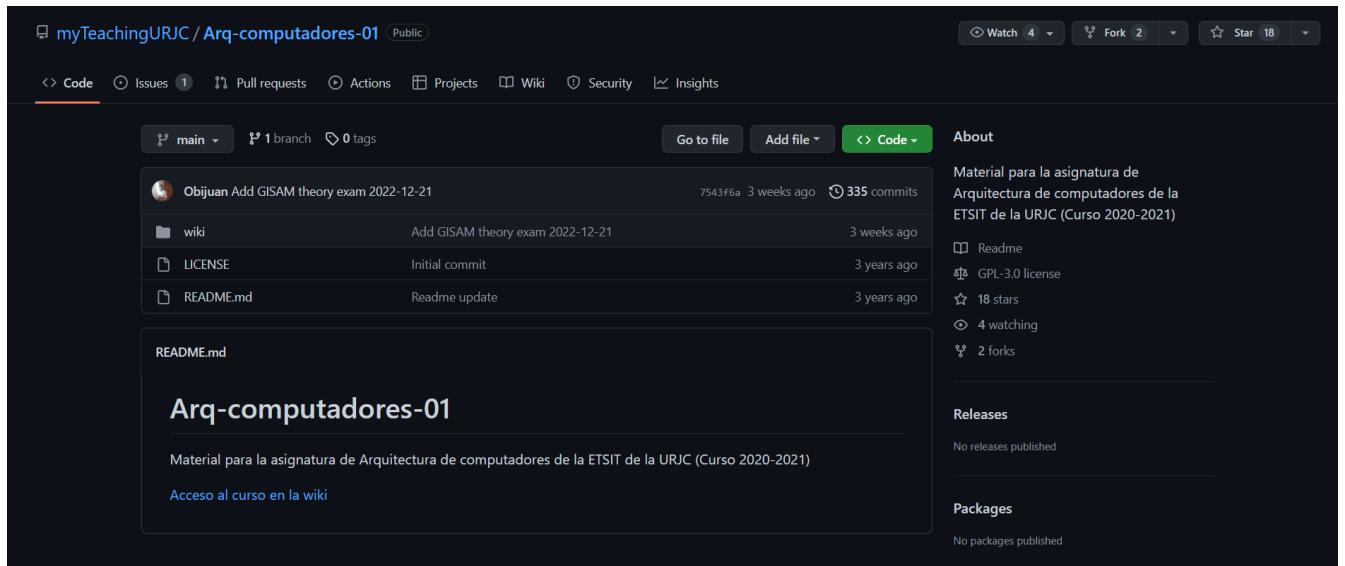


Figura 6.1: Vista inicial del repositorio de “Arquitectura de computadores”.

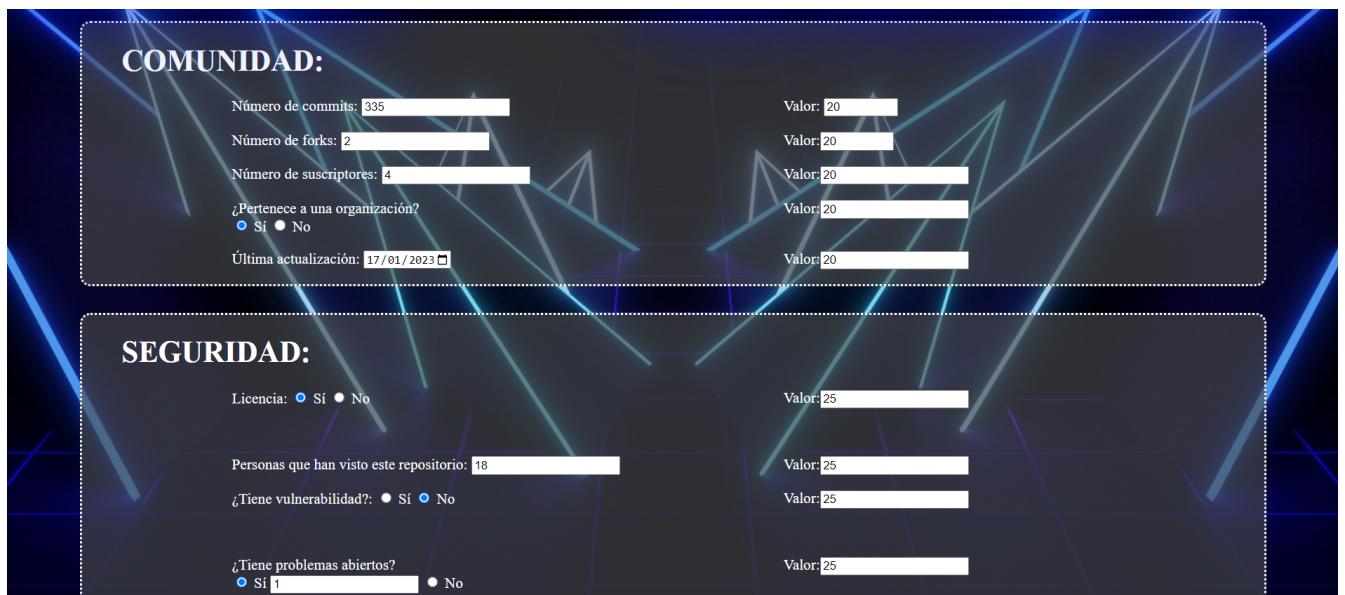


Figura 6.2: Datos del repositorio de “Arquitectura de computadores”.

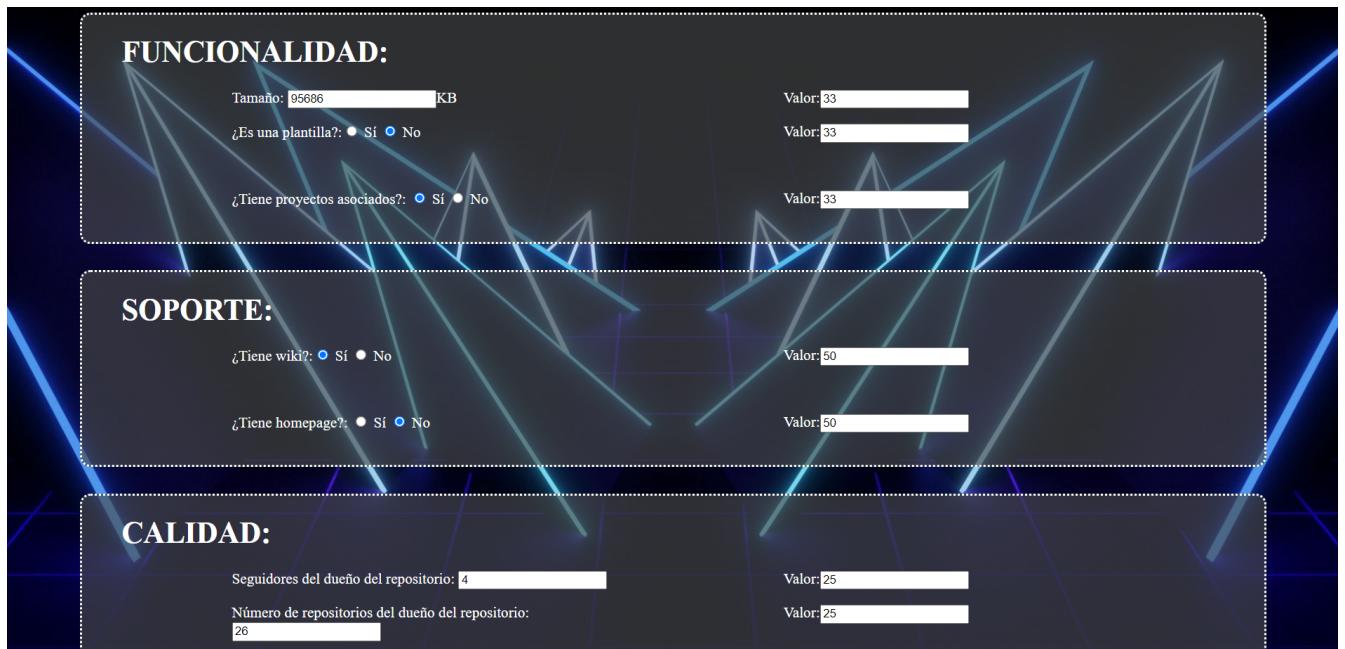


Figura 6.3: Datos del repositorio de “Arquitectura de computadores”.

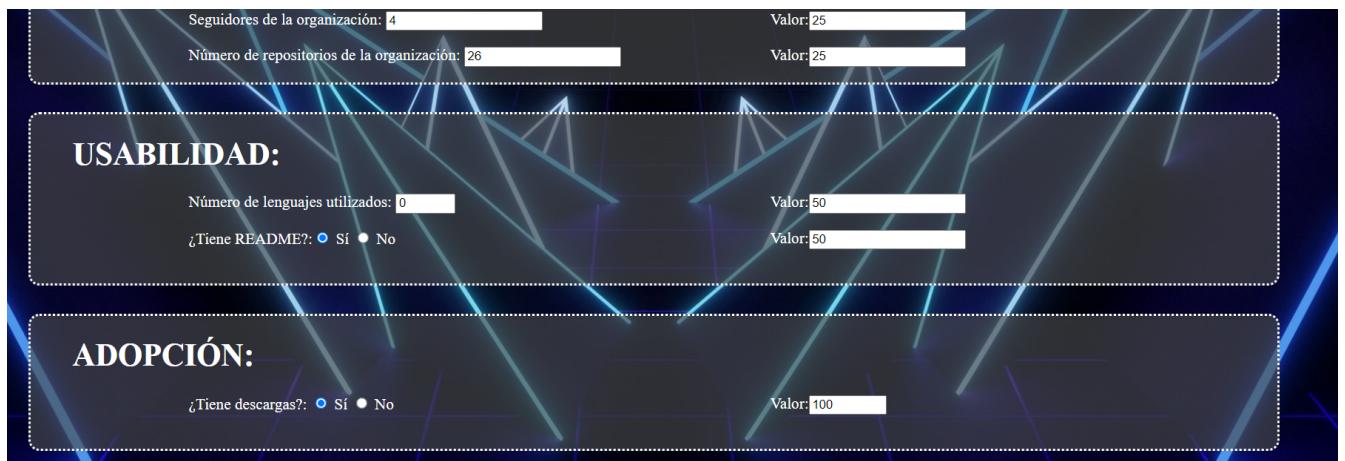


Figura 6.4: Datos del repositorio de “Arquitectura de computadores”.

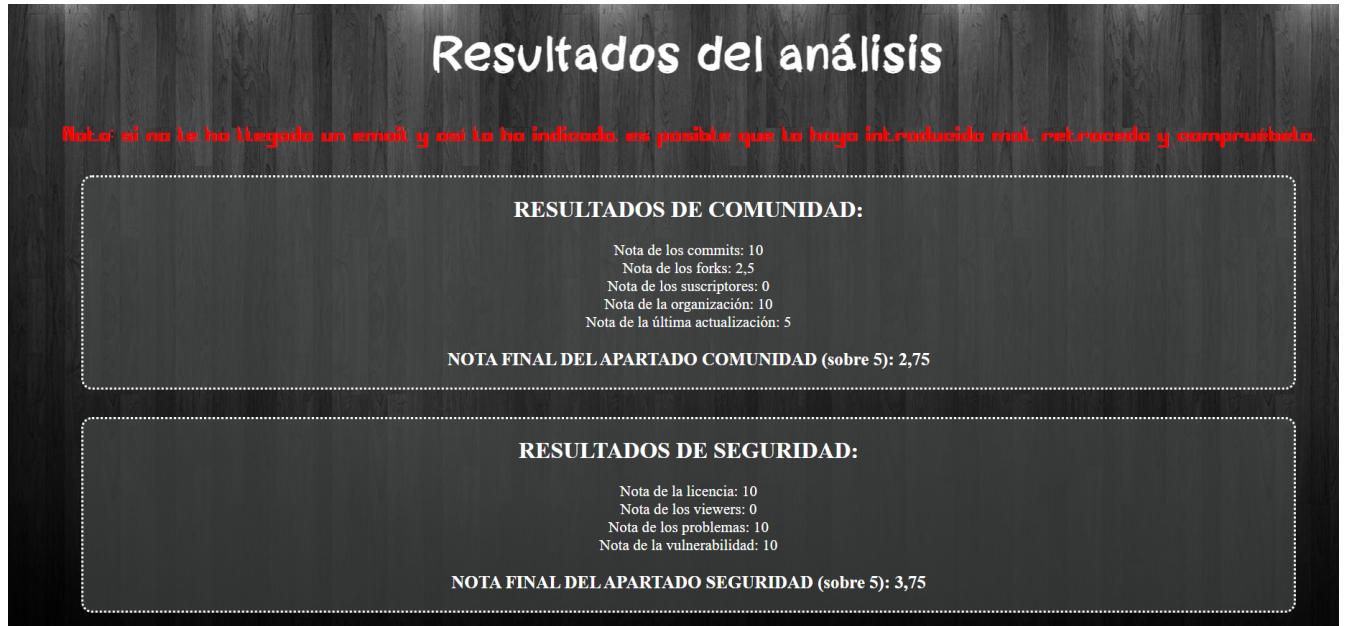


Figura 6.5: Resultados del repositorio de “Arquitectura de computadores”.

con este repositorio.

6.2. Prueba con moodle/moodle

Ahora, vamos a probar con un repositorio teóricamente completo como es el de Moodle² que tenemos en la figura 6.8. Moodle [9] es un sistema de gestión de aprendizaje, gratuito y de código abierto que utilizan muchos centros educativos. Por ejemplo, la URJC lo usa para la gestión de su Aula Virtual³

6.2.1. Análisis de moodle/moodle

Como siempre, en primer lugar obtenemos los datos del repositorio que mandaremos a través del formulario para su posterior calificación. Lo vemos de la figura 6.9 a la figura 6.11.

²<https://github.com/moodle/moodle>

³<https://www.aulavirtual.urjc.es/moodle/login/index.php>



Figura 6.6: Resultados del repositorio de “Arquitectura de computadores”.



Figura 6.7: Resultados del repositorio de “Arquitectura de computadores”.

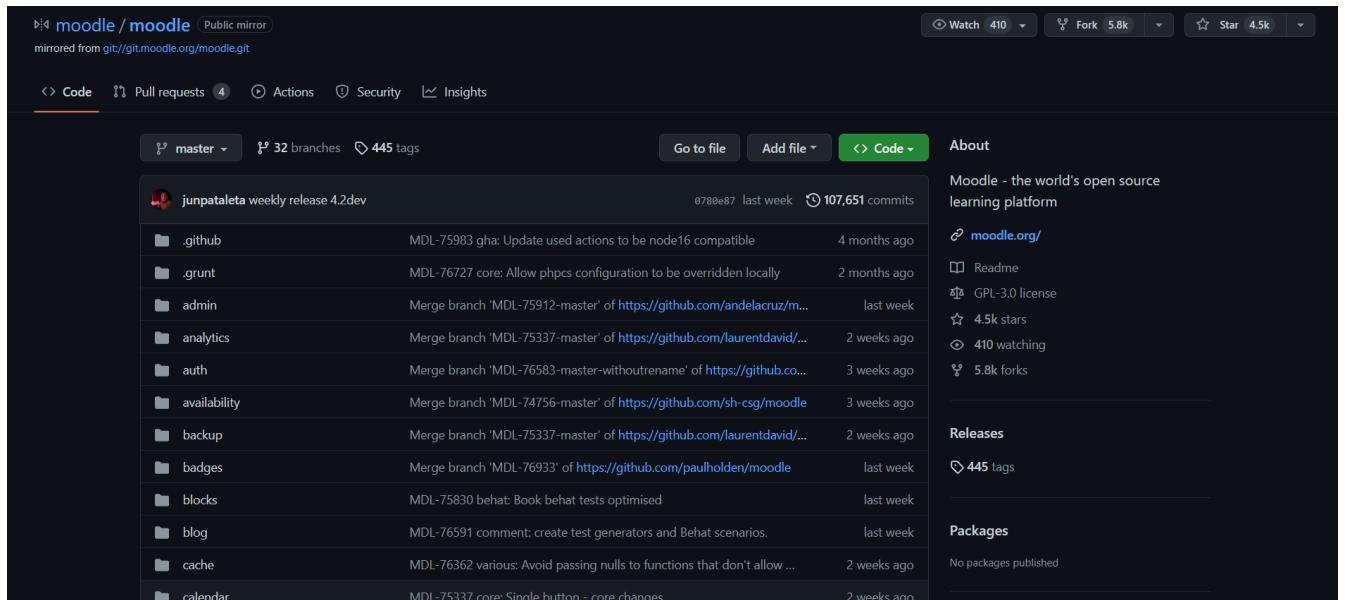


Figura 6.8: Vista inicial del repositorio de Moodle.

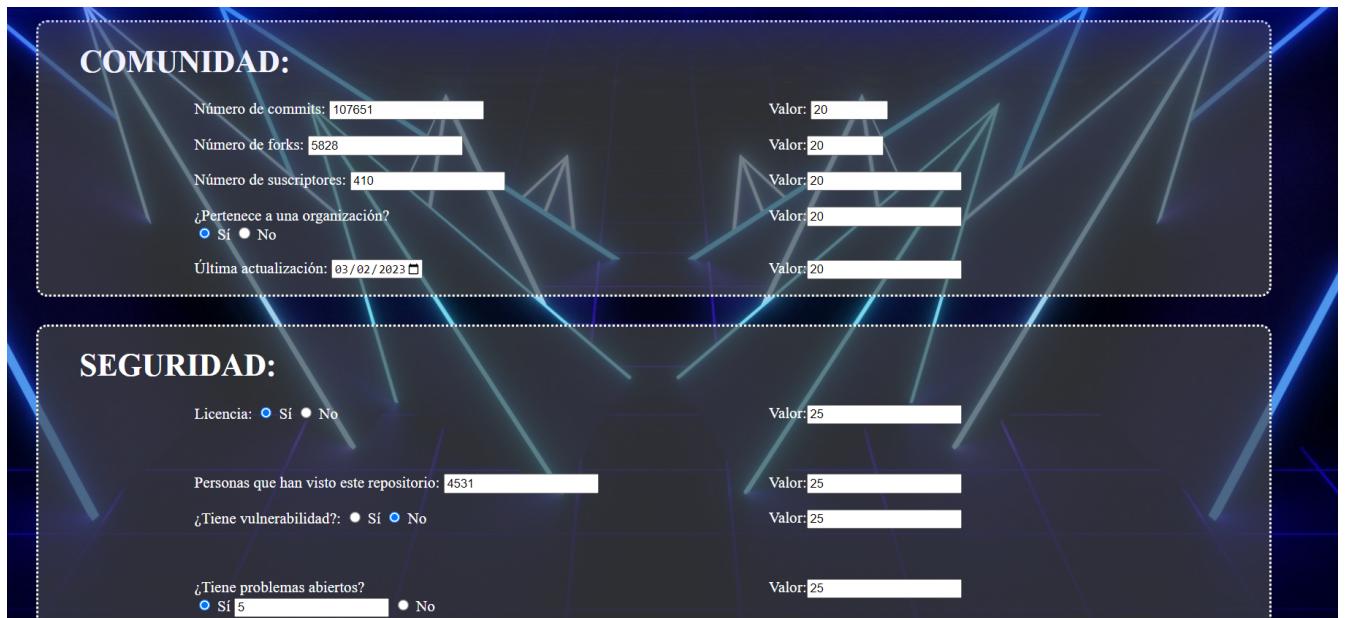


Figura 6.9: Datos del repositorio de Moodle.

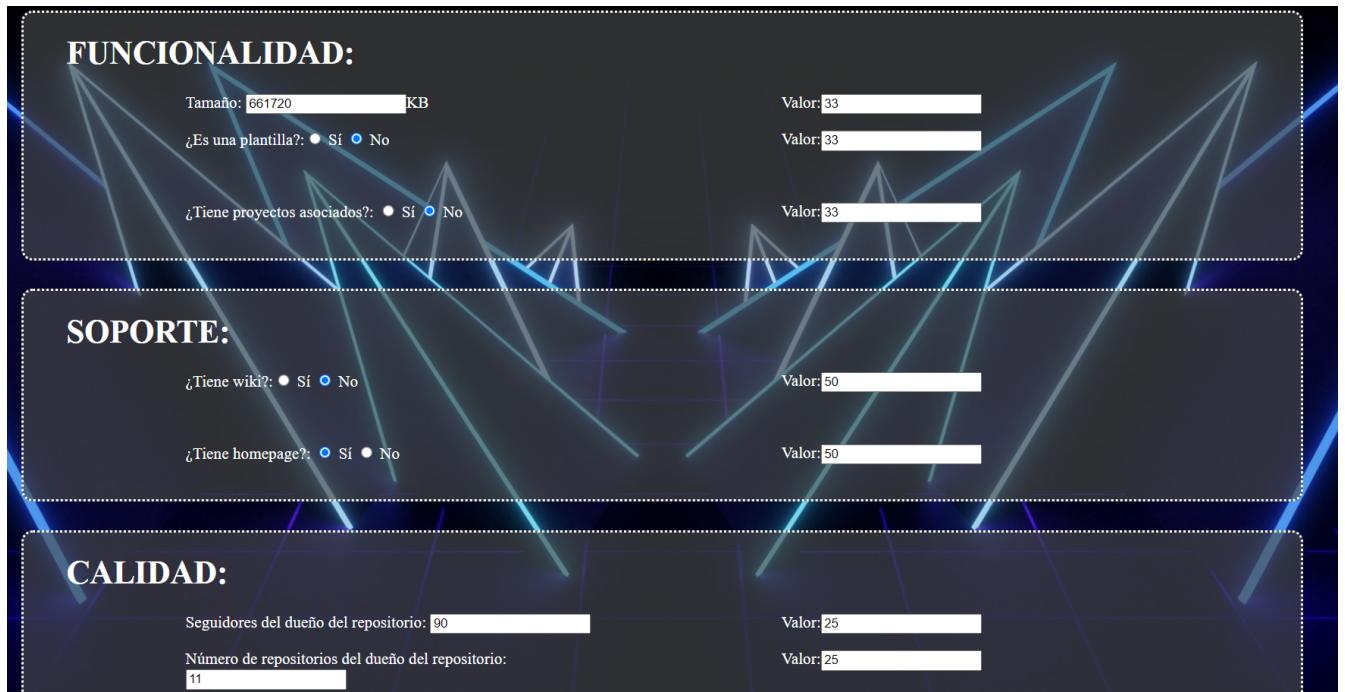


Figura 6.10: Datos del repositorio de Moodle.

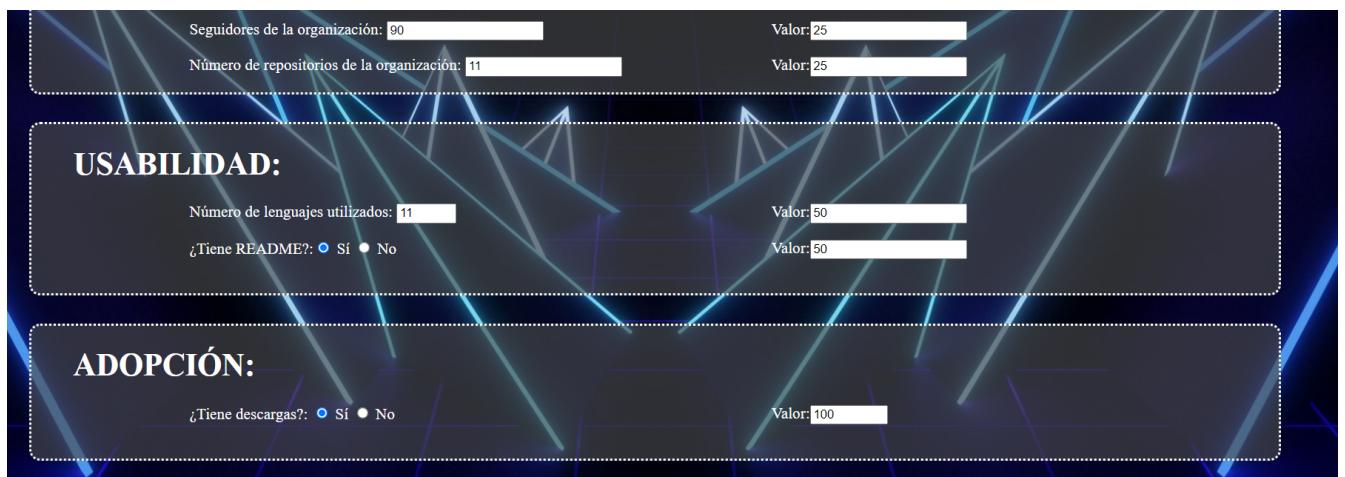


Figura 6.11: Datos del repositorio de Moodle.



Figura 6.12: Resultados del repositorio de Moodle.

6.2.2. Resultados de moodle/moodle

Finalmente, vemos los resultados del análisis del repositorio de Moodle. Véase de la figura 6.12 a la figura 6.14. Obtenemos una calificación final de 3.09 sobre 5, lo que nos vuelve a indicar que es un buen repositorio, es decir, es de fiar.

6.3. Prueba con un repositorio vacío

Finalmente, probaremos la aplicación con un repositorio que no contenga nada, únicamente el README inicial, para demostrar que mantiene la coherencia a la hora de calificar. Adicionalmente, se modificarán los datos para probar cómo cambian los resultados. El repositorio elegido para esta prueba es “ivanmiguelmolinero/repositorio_vacio”⁴. Lo podemos ver en la figura 6.15.

6.3.1. Análisis del repositorio vacío

Volvemos a ver los datos recogidos del análisis de un repositorio, esta vez del repositorio “ivanmiguelmolinero/repositorio_vacio”. Los datos van de la figura 6.16 a la figura 6.18.

⁴https://github.com/ivanmiguelmolinero/repositorio_vacio



Figura 6.13: Resultados del repositorio de Moodle.



Figura 6.14: Resultados del repositorio de Moodle.

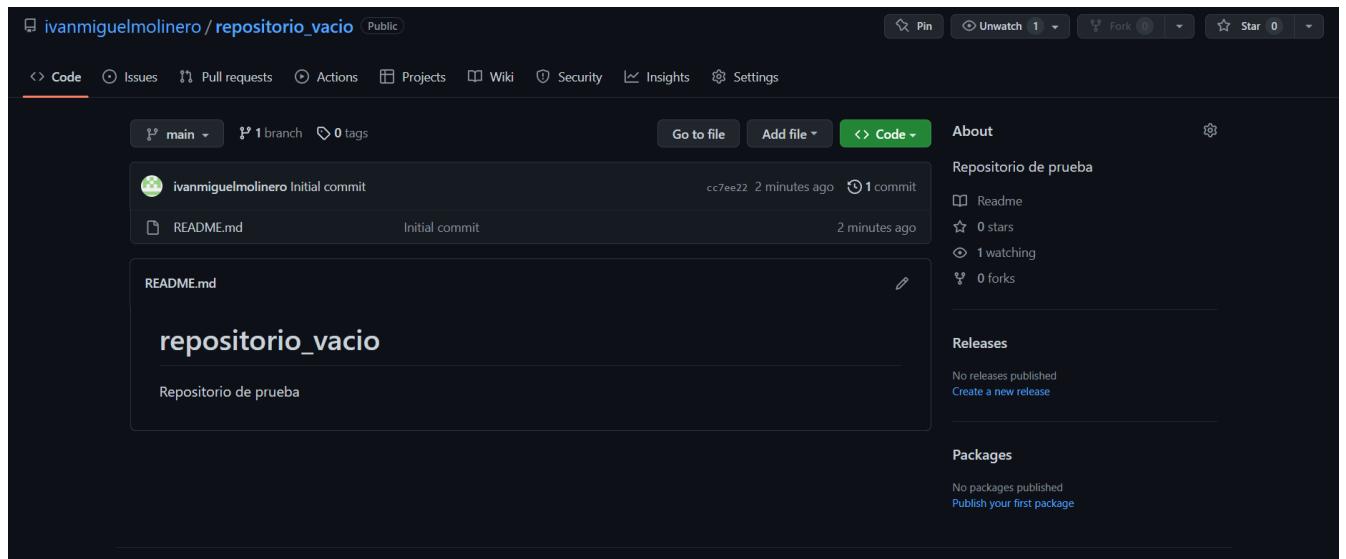


Figura 6.15: Vista inicial del repositorio vacío.

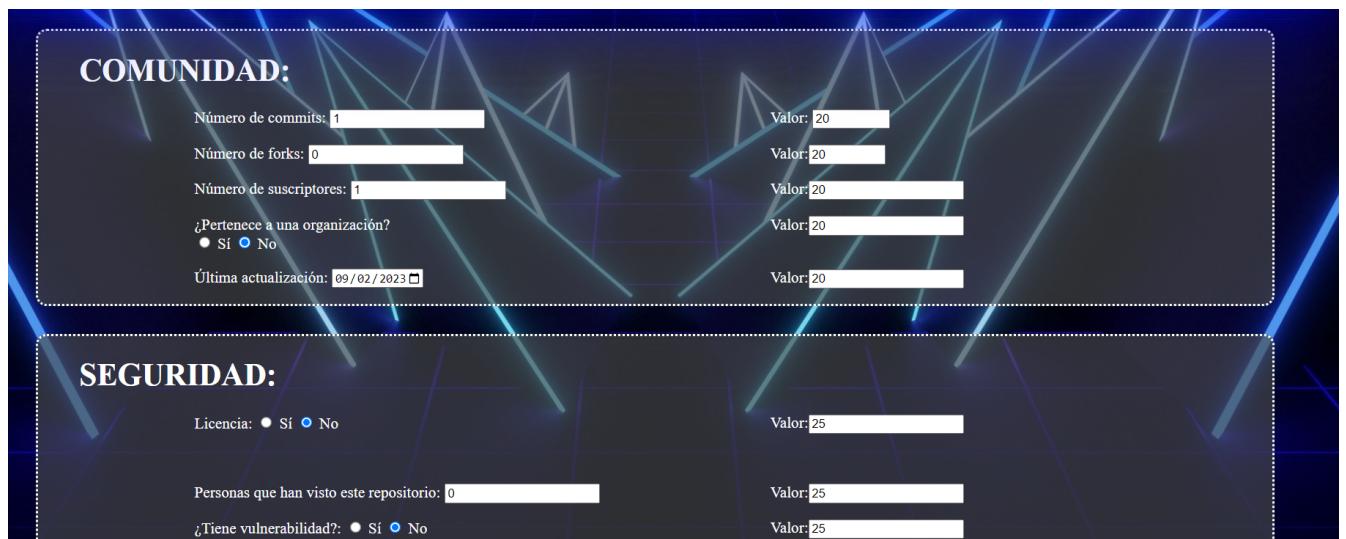


Figura 6.16: Datos del repositorio vacío.

¿Tiene problemas abiertos?

- Sí
- No

Valor: 25

FUNCIONALIDAD:

Tamaño: KB

¿Es una plantilla?

- Sí
- No

¿Tiene proyectos asociados?

- Sí
- No

Valor: 33

Valor: 33

Valor: 33

SOPORTE:

¿Tiene wiki?

- Sí
- No

Valor: 50

¿Tiene homepage?

- Sí
- No

Valor: 50

Figura 6.17: Datos del repositorio vacío.

CALIDAD:

Seguidores del dueño del repositorio:

Número de repositorios del dueño del repositorio:

Seguidores de la organización:

Número de repositorios de la organización:

Valor: 25

Valor: 25

Valor: 25

Valor: 25

USABILIDAD:

Número de lenguajes utilizados:

¿Tiene README?

- Sí
- No

Valor: 50

Valor: 50

ADOPCIÓN:

¿Tiene descargas?

- Sí
- No

Valor: 100

Figura 6.18: Datos del repositorio vacío.



Figura 6.19: Resultados del repositorio vacío.

6.3.2. Resultados del repositorio vacío

Vemos que los resultados son bastante peores en comparación con los dos casos anteriores, obteniendo una calificación final de 2.17, luego es una calificación coherente. Vemos los resultados de la figura 6.19 a la figura 6.21.

6.3.3. Modificación de los datos del repositorio vacío

Como se indica en el apartado 6.3 vamos a pasar a modificar los datos y los valores que hemos obtenido al analizar el repositorio vacío. Esta funcionalidad ha sido añadida para que el usuario pueda dar más importancia a la análisis de según que apartados según le convenga para sus menesteres o por si quiere comprobar como cambiaría la calificación de su repositorio en caso de que cambiara cosas antes de realizar esas modificaciones. Se han modificado los datos de los parámetros de comunidad y seguridad (el resto se han dejado igual) como vemos en la figura 6.22. Vemos en la figura 6.23, pues, que los resultados de estos parámetros han mejorado y consecuentemente la nota final del repositorio como se puede observar en la figura 6.24.



Figura 6.20: Resultados del repositorio vacío.



Figura 6.21: Resultados del repositorio vacío.



Figura 6.22: Datos modificados del repositorio vacío.



Figura 6.23: Resultados modificados del repositorio vacío.



Figura 6.24: Resultados finales modificados del repositorio vacío.

Capítulo 7

Conclusiones

Tras la realización de este proyecto se han conseguido varios de los objetivos planteados inicialmente. En este capítulo se explicarán cuáles se han conseguido y cómo se han paliado los objetivos que no.

7.1. Consecución de objetivos

En primer lugar se explicarán los objetivos conseguidos y a continuación la solución a los objetivos que no se han podido conseguir.

7.1.1. Objetivos conseguidos

- Se ha aprendido a manejar Django gracias a los tutoriales de DjangoGirls recomendados al principio del proyecto por mi tutor.
- Se ha entendido la utilidad y necesidad de usar tecnologías como OpenBRR para poder analizar qué códigos y/o repositorios son mejores que otros. Hay una cantidad ingente de proyectos y estas tecnologías son necesarias para distinguir cuáles son los más válidos. OpenBRR hace un análisis inteligente ya que se basa en los parámetros más esenciales.
- La biblioteca PyGitHub se ha entendido a la perfección a la hora de sacar los datos necesarios para un análisis de repositorio. Esta biblioteca también es útil para gestionar GitHub pero ha quedado pendiente aprender esta utilidad ya que no era necesaria para el proyecto.

- El envío de formularios con Django funciona correctamente en la aplicación.
- A la hora de modificar los datos se encontró el problema de que la aplicación no recordaba los cambios pero se pudo solucionar con las funciones “`save_input`” de “`datos.js`”.
- En un principio no sabía qué ponderación dar a cada dato/parámetro pero con ayuda de mi tutor decidimos que lo mejor es que el usuario decidiera este asunto por lo que se le dio la opción de poder modificarlo quedando un reparto equitativo como opción por defecto. Así se consiguieron resultados coherentes.
- Se solucionó error causado cuando el usuario introduce un repositorio que no existe dándole la posibilidad de volver a la página inicial.
- Investigando las diversas funcionalidades de Django di con una que permitía el envío de correos por lo que decidí implementarlo de forma extra en mi proyecto para dar la posibilidad al usuario de guardar los resultados en un sitio seguro.

7.1.2. Solución de los objetivos no conseguidos

- No se ha conseguido solucionar el error de PyGitHub que limita el número de peticiones que puede realizar el usuario por lo que se ha puesto una pantalla de error que informa al usuario del error y le permite volver a la página principal indicándole que debe esperar antes de volver a analizar un repositorio.
- A la hora de mostrar y ocultar las pestañas en la pantalla de datos no se ha conseguido que el usuario pueda mandar los datos con las pestañas ocultas ya que la aplicación detecta que los formularios que están ocultos están vacíos por lo que se ha optado por no permitir al usuario enviar los datos con alguna pestaña oculta. Cuando lo intenta, se le muestra la alerta que vemos en la figura 7.1.

7.2. Aplicación de lo aprendido

Para la realización de este TFG he aplicado los conocimientos adquiridos en las diferentes asignaturas de programación que hay durante el grado así como un asignatura de primero a la hora de redactar la memoria.

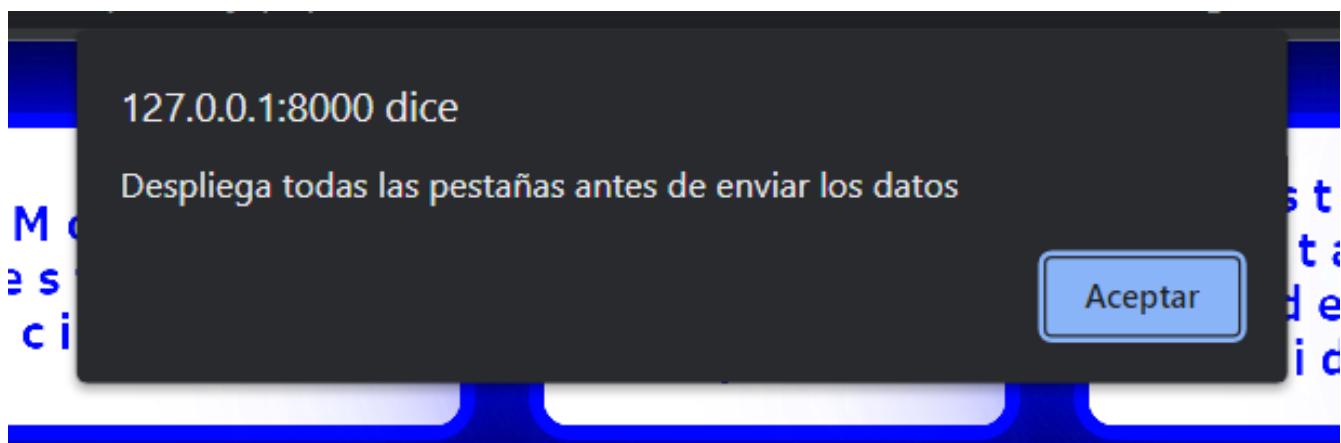


Figura 7.1: Alerta mostrada cuando hay una pestaña oculta.

1. **Expresión Oral y Escrita y Búsqueda de Información:** Me ha ayudado a realizar una correcta redacción de la memoria y buscar información de forma eficaz.
2. **Informática I:** Cuando estudie esta asignatura en 2016, me enseñaron Picky pero aún así pude aprender los fundamentos de la programación y las sintaxis básicas.
3. **Protocolos para la Transmisión de Audio y Vídeo en Internet:** En esta asignatura aprendí a programar en Python, el lenguaje con el que está desarrollada esta aplicación en su mayoría.
4. **Construcción de Servicios y Aplicaciones Audiovisuales en Internet:** Gracias a esta asignatura aprendí HTML, CSS y Javascript y me inicié en el desarrollo web del lado del cliente. Con estos lenguajes funcionan las diferentes ventanas de este proyecto.
5. **Laboratorio de Tecnologías Audiovisuales en la Web:** En esta asignatura aprendí a desarrollar el lado del servidor en una página web lo que ha sido de mucha ayuda a la hora de entender fácilmente Django.

7.3. Lecciones aprendidas

Con lo aprendido durante el grado no fue suficiente así que tuve que adquirir diversos conocimientos para poder realizar el TFG.

1. Aprender a gestionar y aprovechar mejor el tiempo en un proyecto más largo que cualquier trabajo de clase.
2. Entender y utilizar Django.
3. Entender y utilizar PyGitHub.
4. Avanzar en mis conocimientos de GitHub y aprender que no se trata simplemente de un repositorio donde subir tus proyectos si no que también ayuda a gestionarlos, buscar errores, colaborar con otros desarrolladores, etc.
5. Aprender a realizar un manual de usuario.
6. Afianzar mis conocimientos sobre programación y programar de una manera más limpia.
7. Aprender a utilizar L^AT_EX para la redacción de la memoria.

7.4. Trabajos futuros

Como todo proyecto de software, este TFG no está terminado ni mucho menos. Siempre se puede mejorar su optimización o añadir nuevas mejoras y funcionalidades. A continuación se mencionan algunas:

- Corregir el error que no permite enviar el formulario si las pestañas están ocultas.
- Solucionar el error que limita el número de peticiones de PyGitHub.
- Añadir una funcionalidad que permita comparar varios repositorios a la vez y que devuelva cuál es el mejor según la necesidad del desarrollador.
- Añadir más datos a analizar en cada parámetro de OpenBRR.

Apéndice A

Manual de usuario

Esta aplicación permite analizar un repositorio de GitHub según los parámetros de OpenBRR.

A.1. Instalación

Para instalar y poder usar esta aplicación en tu ordenador personal debes dirigirte al repositorio “ivanmiguelmolinero/TFG”¹ y clonarlo en un directorio local. A continuación deberás instalar Django en tu entorno. Para ello abre una ventana de comandos y ejecuta el siguiente comando:

```
pip install django
```

Se empezará a instalar y, una vez te indique que la instalación ha sido realizada con éxito, estará todo listo para comenzar a usar la aplicación.

A.1.1. Configuración del correo.

Para configurar la funcionalidad del envío del correo tendremos que dirigirnos a la carpeta “mysite” y:

1. En el fichero settings.py cambiar el valor de “EMAIL_HOST_USER” por tu propia dirección de correo.
2. Crear un fichero “credentials.env” donde introduciremos lo siguiente:

¹<https://github.com/ivanmiguelmolinero/TFG>

```
EMAIL_USER = ejemplo@gmail.com  
EMAIL_PASSWORD = contraseña
```

3. El valor del campo contraseña se obtiene en <https://myaccount.google.com/security> en el apartado “Iniciar sesión en Google” > “Contraseñas de aplicaciones”.

A.2. Iniciar la aplicación.

Para iniciar la aplicación debes abrir una ventana de comandos en la carpeta “Django_BRR” y ejecutar:

```
python manage.py runserver
```

A.3. Uso de la aplicación.

Finalmente para utilizar esta aplicación, abrimos un navegador y nos dirigimos a la dirección “<http://127.0.0.1:8000/>” y se nos abrirá la ventana principal (figura 4.21). Una vez ahí, podremos introducir el repositorio que queramos analizar siguiendo la estructura “usuario/nombre_del_repositorio” por ejemplo “ivanmiguelmolinero/TFG” donde ivanmiguelmolinero es el usuario del dueño del repositorio y TFG el nombre del repositorio. Pulsamos en “ANALIZAR” y nos llevará a la siguiente ventana. Ya en la ventana de datos (de la figura 6.9 a la figura 6.11) nos aparecerán los 7 parámetros analizados ocultos en sus correspondientes pestañas. Para poder verlo debemos pulsar en sus correspondientes botones. Podemos cambiar los datos analizados e incluso la ponderación que aporta cada uno. Al final de esta página, de manera opcional, podremos introducir una dirección de correo a la que nos llegaría un mensaje con los resultados del análisis. Pulsamos en “ENVIAR DATOS” y nos llevará a la última ventana. Finalmente, en la ventana de resultados (de la figura 6.12 a la figura 6.14), vemos la calificación de cada apartado y la calificación final del repositorio.

Bibliografía

[1] Análisis con openbrr.

<https://dl.ifip.org/db/conf/oss/oss2010/PetrinjaSS10.pdf>.

[2] Css.

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

[3] Djangogirls.

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

[4] Filosofía de python.

<https://datademia.es/blog/que-es-python>.

[5] Github.

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

[6] Html.

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

[7] Javascript.

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

[8] Middleware.

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

[9] Moodle.

<https://moodle.org/>.

[10] Mosst.

<https://dl.ifip.org/db/conf/oss/oss2009/BiancoLMT09.pdf>.

[11] Openbqr.

<https://flosshub.org/sites/flosshub.org/files/OpenBQR.pdf>.

[12] Openbrr.

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

[13] Osmm.

<https://optaresolutions.com/osm-assessments/>.

[14] Perceval.

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

[15] Pygithub.

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

[16] Python.

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

[17] Repositorio de mi trabajo de fin de grado.

<https://github.com/ivanmiguelmolinero/TFG>.