Universidad
Rey Juan Carlos

ESCUELA DE INGENIERÍA DE FUENLABRADA

TITULACIÓN EN MAYÚSCULAS

TRABAJO FIN DE GRADO

ANÁLISIS DEL USO DE LENGUAJES DE PROGRAMACIÓN MINORITARIOS EN PROYECTOS DE SOFTWARE LIBRE

Autor : Raúl Gómez Cantador

Tutor : Dr. Gregorio Robles Martínez/a

Curso académico 2024/2025

Trabajo Fin de Grado/Máster

ANÁLISIS DEL USO DE LENGUAJES DE PROGRAMACIÓN MINORITARIOS EN PROYECTOS DE SOFTWARE LIBRE

Autor : Raúl Gómez Cantador

Tutor : Dr. Gregorio Robles Martínez/a

La defensa del presente Proyecto Fin de Carrera se realizó el día de de 2024, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente **Calificación:**

Fuenlabrada, a de de 202X

Dedicado a

mi familia / mi abuelo / mi abuela

II

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

Índice de figuras

Capítulo 1

Introducción

Durante los últimos años, a través de plataformas de desarrollo de software como GitHub o GitLab, una gran cantidad de desarrolladores contribuyen diariamente a proyectos de software libre. Por supuesto, al haber tanta variedad de proyectos y desarrolladores, existe una gran diversidad de lenguajes de programación que se usan en los distintos repositorios de cada uno de los proyectos. Este trabajo de fin de grado tiene como objetivo realizar el análisis del uso de esos lenguajes de programación, dividiéndolos en dos clases: lenguajes mayoritarios y lenguajes minoritarios, si bien, nos centraremos en mayor medida en estos últimos. Estos lenguajes minoritarios, si bien no son los más usados por la gran mayoría de desarrolladores, pueden llegar a cumplir roles específicos y críticos para el funcionamiento de una aplicación o proyecto.

Para llevar a cabo este análisis, se ha desarrollado una aplicación en Python cuyo objetivo es la recopilación y representación de datos provenientes de los repositorios de distintos proyectos para comprobar como es el uso de estos lenguajes minoritarios. Con todos estos datos, se ha creado una base de datos SQL que contiene la información de uso de los distintos lenguajes por usuario y repositorio de todos los proyectos que se han analizado a lo largo de la ejecución del trabajo de fin de grado.

Además, este proyecto cuenta con una aplicación web para poder facilitar la recolección y representación de los datos de manera mucho más fácil y cómoda, y que en el futuro podría servir para seguir recopilando datos de repositorios distintos a los que se han usado para realizar este análisis.

1.1. Estructura de la memoria

La memoria se estructura de la siguiente manera:

- **Capítulo 1: Introducción.** Se presenta la motivación y una breve descripción de lo que se afronta en este proyecto, además de informar de la estructura del mismo.
- **Capítulo 2: Objetivos.** Se describen los objetivos a alcanzar durante toda la consecución del proyecto, incluyendo tanto planificación temporal como objetivos técnicos.
- **Capítulo 3: Estado del arte.** Explicación de las distintas tecnologías usadas a lo largo del proyecto. Incluye la información sobre la aplicación web, y como se realiza la recopilación y guardado de datos en la base de datos.
- **Capítulo 4: Diseño e implementación.** Se realiza una explicación sobre el diseño en conjunto de todo el proyecto, y el por qué de como se ha implementado.
- **Capítulo 5: Resultados.** Se realiza el análisis final de los resultados obtenidos.
- **Capítulo 6: Conclusiones.** Incluye la conclusión final del proyecto.

Capítulo 2

Objetivos

2.1. Objetivo general

Mi trabajo de fin de grado consiste en obtener datos de los repositorios de proyectos de software libre que se encuentren disponibles en la plataforma de desarrollo software GitHub y analizar en profundidad los desarrolladores implicados en los mismos como los lenguajes de programación utilizados para llevarlos a cabo, principalmente aquellos que son minoritarios, pero de uso común.

2.2. Objetivos específicos

Se han seguido los siguientes objetivos específicos para llegar a cumplir el objetivo final. Cada objetivo específico cuenta con sus propios puntos:

- **Obtención de datos de los repositorios de plataformas Git**
  - Búsqueda de una herramienta para obtener los datos usando Python
  - Investigación del módulo de python Perceval usado para la obtención de commits realizados en los respositorios
  - Investigación de llamadas a la API de GitHub para obtener información adicional
  - Guardado de datos en formato JSON para su posterior tratamiento
- **Tratamiento de los datos de los repositorios**

    - Organización de los datos obtenidos
    - Parsear ficheros JSON de las llamadas a la API de GitHub para guardar únicamente los datos relevantes para el proyecto:
        - Commits realizados sobre el repositorio
        - Usuario que ha participado en cada uno de los commits
        - Lenguaje de programación usado en cada commit
        - Total de usuarios que han participado en el repositorio
    - Creación de un fichero JSON para almacenar todos los datos resultantes del punto anterior:

- **Creación de una aplicación web para permitir un uso más fácil del aplicativo**
    - Uso de Flask para construir la aplicación web
    - Conectividad con la base de datos creada
    - Permitir al usuario la obtención de los datos de un repositorio a su elección
    - Representación de los datos de los repositorios
- **Creación de una base de datos donde almacenar los análisis de los repositorios**
    - Almacenar repositorios analizados
    - Almacenar usuarios contribuyentes a cada repositorio analizado
    - Almacenar usuarios con todas sus contribuciones a cada uno de los repositorios
    - Almacenar lenguajes usados en cada repositorio

2.3. Planificación temporal

El total de duración del proyecto ha sido de aproximadamente 1 año. Contacté por primera vez con Gregorio; mi tutor, en octubre del año 2023, con la intención de comenzar con este proyecto, dando inicio desde noviembre y terminándolo a lo largo del anterior curso académico 2023/2024. Sin embargo, no ha sido hasta este presente curso 2024/2025 en el que he podido finalizar el trabajo, sobre todo debido a algunos parones que hice durante algunos meses del año

anterior y a que, principalmente, se ha hecho en mi tiempo libre y durante los fines de semana. En la siguiente figura, se encuentra la planificación general final que ha seguido el proyecto a lo largo de todo este año en un diagrama de Gantt:

Capítulo 3

Estado del arte

En este proyecto se han utilizado múltiples tecnologías, siendo el lenguaje de programación Python el principal, y en el que se basan prácticamente todos los recursos que he utilizado.

3.1. Python

Python [5] es uno de los lenguajes de programación más utilizados durante los últimos años. Se debe, sobre todo, a su accesibilidad y facilidad que ofrece a los desarrolladores para realizar múltiples tareas.

Es un lenguaje de programación de código abierto y gratuito creado por Guido van Rossum a principios de la década de los noventa, pero no ha hecho nada más que evolucionar durante todo este tiempo para agregar nuevas características en cada una de las versiones. Para este trabajo se ha utilizado la versión 3.11 de Python, sin embargo, aún sigue actualizandose, siendo la última versión en el momento de escribir esta memoria la versión 3.13.

Las principales ventajas [6] que ofrece sobre otros lenguajes son:

- **Es un lenguaje interpretado**, es decir, no es necesario compilar el código para su ejecución, sino que en su lugar existe un intérprete que se encarga de la lectura del fichero y su ejecución.

- **Es un lenguaje multiplataforma**, por lo que su ejecución no está limitada a un sistema o software específico. Es muy flexible desde este punto de vista.

- **Facilidad de la sintaxis**. Es uno de los lenguajes preferidos para comenzar en el mundo de la programación debido a ello.

- **Existencia de un garbage collector** que permite la limpieza automática de memoria y facilita en gran cantidad evitar gastos de memoria inútiles.

En mi caso, lo he escogido en este proyecto ya que, además de ser el lenguaje que más he usado tanto a lo largo de la carrera universitaria como en el mundo laboral, y por tanto ser el lenguaje del que más conocimiento tengo, quería aprender más sobre él usándolo en otros ámbitos que no conocía antes, como la interacción con una base de datos, o la creación de la plataforma web usando el módulo Flask que ofrece el propio Python.

3.2. Perceval

Perceval es la librería de Python que me recomendó mi tutor para poder conseguir la mayoría de información necesaria para realizar los análisis de este proyecto. Consiste en un módulo que consigue recuperar datos relacionados con el desarrollo software de varias posibles fuentes como pueden ser Confluence, Bugzilla o Slack, aunque en este trabajo solo se ha usado para obtener datos de repositorios Git.

Del modo que se ha usado Perceval en mi caso, ha sido para poder recuperar los commits de los distintos repositorios en formato JSON, lo que me ha permitido ver el historial de cambios de los repositorios y las personas que han contribuido a los mismos para realizar los análisis.

3.3. Flask

Flask [1] es un módulo de Python que consiste en un framework ligero para la creación de aplicaciones web. Está diseñado para ser sencillo y rápido, pero escalable a la construcción de aplicaciones complejas. Se usa en distintos ámbitos como aplicaciones web, como es el caso de este proyecto, pero también en microservicios, desarrollos de APIs...

Se ha escogido Flask sobre otros frameworks para Python como puede ser Django ya que ofrece más flexibilidad y me permitía elegir los demás módulos que fueran necesarios para la personalización de mi web y hacerla de la manera que he pensado.

3.4. Jinja2

Jinja2 [4] es el motor de plantillas que usa Flask. Permite la generación de contenido web dinámico en la aplicaciones de manera eficiente y estructurada. Una de sus características principales que se ha usado en este proyecto es la utilización de una estructura base que cualquier otra plantilla puede heredar, lo que permite ahorrar bastante tiempo de desarrollo HTML o CSS.

Su uso me ha permitido crear las páginas dinámicas en la que aparecen los análisis realizados de los repositorios, permitiéndome representarlos de manera sencilla y cómoda para mí como desarrollador, pero también, a imagen del usuario que podría visitar la web, fácil de entender.

3.5. Git

Git [2] es un sistema de control de versiones usado en proyectos de desarrollo de software. Realiza un seguimiento de los cambios en archivos de un proyecto, por lo que es muy útil en los casos donde un grupo de personas realizan estos cambios en los mismos ficheros al mismo tiempo.

Permite a los desarrolladores conocer todo el historial de modificaciones que se han realizado sobre el proyecto, lo que facilita la organización y el alineamiento entre los propios contribuyentes. Se utiliza tanto en proyectos de código abierto y gratuito como este, como en proyectos comerciales de empresas, que utilizan Git para la gestión de versiones del software que crean.

En este proyecto, además de para los repositorios Git que he escogido para realizar el análisis, se ha utilizado para hacer también, el control de versiones de mi propio proyecto, por lo que es fácil ver el progreso que ha seguido mi software a lo largo del tiempo, pues es posible ver el historial de cambios hechos durante este año.

3.6. GitHub

GitHub [3] es una plataforma para almacenar, compartir y trabajar junto a otros usuarios en proyectos de desarrollo. Sus principales características son:

- Compartir el trabajo junto a otras personas.

- Seguir los cambios en el código a lo largo del tiempo, para el control de versiones de un proyecto.
- Colaborar en proyectos con multitud de personas.
- Permitir a otros usuarios de la plataforma que puedan revisar el código y realizar sugerencias para su mejora.
- Creación y gestión de repositorios donde almacenar todo el código de un proyecto.

Todos los repositorios analizados en este proyecto forman parte de la propia plataforma de GitHub, incluyendo también mi propio trabajo de fin de grado.

3.7. SQLAlchemy

SQLAlchemy es una librería de Python usada para la interacción con una gran variedad de bases de datos. Permite crear modelos de datos y consultas de una manera sencilla y cercana a las clases de Python.

Es un ORM (Object Relational Mapper), por lo que permite el mapeo de tablas de las bases de datos sin tener que escribir las consultas, sino que mediante los objetos de Python, podemos hacer esta función.

Lo he usado en mi implementación con la base de datos SQLite que se ha hecho y que contiene el análisis de todos los repositorios.

3.8. SQLite

SQLite es una base de datos integrada de código abierto. Su principal característica y diferencia con la mayoría de bases de datos SQL es que no tiene un proceso independiente al de la aplicación, sino que se encuentra contenida en la propia aplicación.

Lo he usado por la facilidad que ofrece para la configuración de la propia base de datos, ya que, al estar dentro de la propia aplicación, no necesita configuración de red, lo que facilita bastante el uso de bases de datos para personas como yo, que no son expertos en ello.

3.9. HTML

HTML es un lenguaje de marcado que utiliza una serie de etiquetas con el objetivo de dar estructura a un contenido web. Representa el estándar de la visualización de páginas web y es utilizado por todos los grandes navegadores actuales.

Todas las páginas que forman la aplicación web se han creado y estructurado mediante ficheros HTML.

3.10. CSS

CSS es el lenguaje mayoritariamente utilizado para dar estilo a una página web. Este lenguaje permite vincular los documentos de texto en formato HTML con hojas de estilo que contiene la información topográfica de los elementos visuales de la página, y que además, permite separar completamente la estructura de los contenidos, del estilo que estos van a tener ya en la visualización en la web.

En este proyecto se ha utilizado CSS en todas las páginas para dar el estilo visual a los contenidos, utilizando además la librería Bootstrap para facilitar un diseño simple y que sirviera de plantilla para tener consistencia visual en toda la aplicación.

3.11. Bootstrap

Bootstrap es un proyecto de código abierto que consiste en un framework de front-end que facilita el desarrollo de aplicaciones mediante un conjunto de herramientas y componentes prehechos como botones o formularios que permiten la personalización del diseño de una web.

Se ha usado Bootstrap en el diseño de todas las páginas incluyendo botones, formularios, o el encabezado de la página web.

Capítulo 4

Diseño e implementación

En este apartado de la memoria se detalla el funcionamiento de la aplicación y todos sus componentes.

4.1. Arquitectura general

Este proyecto sigue un modelo cliente-servidor. El cliente en este caso, para el uso que está pensado, sería la aplicación web que se ejecuta desde el navegador y que permite realizar las peticiones de una manera más cómoda, accesible y sencilla al servidor, aunque obviamente, se podrían hacer estas peticiones .[a] mano"sin utilizar la aplicación web, y seguiría siendo el cliente. El servidor mientras tanto, espera a atender las peticiones que se envían a los endpoints que tiene definidos para finalmente dar la respuesta a los clientes.

Aunque en puntos posteriores entraremos más a fondo en cada uno de elementos de la aplicación, el flujo que sigue, a modo general es el siguiente:

Un cliente cualquiera accede a la página web del proyecto, a la que he llamado OSSAnalyzer (viene de Open Source Software Analyzer), y haciendo click en el botón "Búsqueda de un repositorio concreto.[en] el encabezado de la página principal, provoca que el navegador lanze una petición GET al endpoint principal de la aplicación (/userrepo). El servidor, después, ofrece en la respuesta un formulario donde el usuario deberá introducir el nombre del repositorio y el nombre del propietario del repositorio, con el objetivo de hacer una petición POST al servidor, también al endpoint /userrepo. Al llamar a este endpoint con estos datos y con el método POST, el servidor realiza toda la parte interna de recolección de datos usando Perceval, realiza también

el análisis de los datos obtenidos y además, también realiza la representación de este análisis en la propia aplicación web. No hay que olvidar también la base de datos que forma parte de la aplicación; ya que los datos de los repositorios que se vayan a analizar, es decir, que se vayan a introducir en el endpoint /userrepo se van a guardar en la base de datos para tenerlos almacenados.

4.2. Estructura del código del proyecto

Para hacer más sencilla la explicación de partes del código en puntos posteriores, hablaré de la estructura de código del proyecto:

```
OSSAnalyzer/
|---> main.py
|
|---> utils.py
|
|---> requests_to_github_api.py
|
|---> readme.md
|
|---> config/
|      |
|      |---> OSSAnalyzerConfig.json
|
|---> logs/
|      |
|      |--->OSSAnalyzer.log
|
|---> web/
      |
      |---> endpoints.py
```

```
|
|---> models.py
|
|---> __init__.py
|
|---> database.db
|
|---> static/
|      |
|      |---> mainpage.css
|      |
|      |---> mainpage.js
|      |
|      |---> userrepo.css
|      |
|      |---> userrepo.js
|      |
|      |---> formulario.css
|      |
|      |---> formulario.js
|      |
|      |---> images/
|           |
|           |---> logoaplicacion.png
|
|---> templates/
       |
       |---> base.html
       |
       |---> mainpage.html
       |
```

```
|---> userrepoForm.html
|
|---> userrepoResult.html
|
|---> usersearchForm.html
|
|---> usersearchResult.html.
```

- **main.py**: Contiene la información necesaria para arrancar la aplicación.
- **utils.py**: Contiene todas las funciones usadas para la recolección de datos de los repositorios obtenidos usando la librería Perceval. También contiene métodos para calcular los lenguajes minoritarios y los contribuyentes que han participado en los mismos.
- **requests_to_github_api.py**: Contiene todas las funciones usadas para la recolección de datos extra mediante llamadas a la API de GitHub.
- **readme.md**: Es un fichero Markdown con una breve introducción a la aplicación. Se usa sobre todo para que al entrar en la página del repositorio en GitHub, cualquier usuario pueda ver fácilmente de qué trata el proyecto.
- **config/OSSAnalyzerConfig.json**: Es un fichero de configuración en formato JSON que se usa principalmente para guardar configuraciones que sean modificables fácilmente y no haya que cambiar más código si fuera necesario. Contiene información sobre la IP y el puerto donde se inicia la aplicación, configuración de los logs, el token que se usa en todas las llamadas contra la API de GitHub y un diccionario con una gran cantidad de extensiones y el lenguaje que representan.
- **logs/OSSAnalyzer.log**: En este fichero de log se escribe continuamente el estado de la aplicación, las peticiones que llegan y sus resultados, errores e información extra. Realmente este fichero tendría sentido si la aplicación estuviera corriendo en un servidor continuamente, ya que sirve para investigar posibles fallos.
- **web/models.py**: En este fichero se definen los modelos de las tablas con sus correspondientes columnas en la base de datos.

- **web/database.db**: En este fichero se almacena la base de datos SQLite3.
- **web/endpoints.py**: En este fichero se definen todos los endpoints de la aplicación y las acciones que realizan cada uno de ellos.
- **web/__init__.py**: Contiene la información para crear la aplicación y cargar todos los elementos, la base de datos, y los endpoints.
- **web/static**: En este directorio se almacenan las imágenes, y los ficheros CSS y JavaScript de las páginas.
- **web/templates**: Este directorio almacena los ficheros HTML con los templates que se modifican.

4.3. Comenzando a usar la aplicación

El primer paso para comenzar a usar la aplicación es, obviamente, acceder a la web desde un navegador. Al entrar se presenta una página de inicio en la que se informa al usuario de la finalidad del proyecto y una pequeña guía para facilitar el uso de la aplicación, aunque es bastante sencillo.

El usuario, debe escoger en ese momento entre dos opciones que se dan en el encabezado de la página:

- Búsqueda de un repositorio concreto: Llama al endpoint /userrepo de la aplicación usando un método GET.
- Búsqueda por usuario: Llama al endpoint /usersearch de la aplicación usando un método GET.

4.4. Endpoint /userrepo

En el endpoint /userrepo es donde se encuentra la gran mayoría de la lógica de la aplicación.

Al llamar a este endpoint con un método GET, el servidor responde a la petición con un formulario con dos campos que el cliente deberá rellenar: el nombre del repositorio, y el nombre

del propietario del repositorio. También, se entrega en la respuesta un botón de envío, que al pulsarlo, provoca que el navegador lance una petición al endpoint /userrepo, pero ahora usando un método POST. Al ser una petición POST, los parámetros se encuentran en el body, y en este caso son dos: username (nombre de usuario del propietario del repositorio) y repo (nombre del repositorio).

Una vez que llega al servidor esa petición POST, comienza la lógica del análisis del repositorio que el usuario ha escogidp:

La primera tarea que realiza el servidor es la obtención de la rama por defecto del repositorio y el identificador del último commit que se ha subido en esa misma rama. El sentido de esto tiene que ver con la implementación con la base de datos, ya que desde un punto de vista de rendimiento, no tendría sentido guardar de nuevo en la base de datos un repositorio cuya última versión ya tenemos almacenado. La comprobación que realiza el servidor para saber si ese repositorio ya está almacenado en la base de datos se hace mediante una consulta contra la base de datos. Si ese repositorio no está almacenado, o si la fecha del último commit es anterior a la que se ha obtenido anteriormente, se procede con la recolección de datos del repositorio.

Como hemos comentado en puntos anteriores, la recolección de datos de los repositorios se ha hecho usando la libería Perceval. Perceval nos permite obtener todos los datos de los commits realizados sobre un repositorio Git en formato JSON, lo que facilita después la selección de datos que nos interesan, y descartar aquellos datos inservibles para nuestro análisis.Añadir ejemplo con un repo pequeño

Los datos que podemos sacar del uso de Perceval y que nos interesan en nuestro análisis principalmente, son:

- Lista de usuarios que han realizado commits sobre el repositorio: para ello se ha creado la función obtain_users en utils.py, que genera esta lista accediendo al campo Author de cada uno de los objetos (cada uno de los objetos representa un commit) del JSON obtenido del uso de Perceval.

- Obtener los ficheros que ha modificado cada uno los usuarios de la lista creada anterior: usando la función obtain_users_files de utils.py, se genera un diccionario de listas con los archivos en los que ha trabajado cada usuario que ha participado en el repositorio.

- Obtener las extensiones de los ficheros que ha modificado cada usuario para conocer el

lenguaje sobre el que se han hecho las modificaciones: con la función obtain_files_extension del fichero utils.py se logra crear un diccionario con los lenguajes. Para ello se usa un diccionario llamado extensiones que se encuentra almacenado en un fichero de configuración JSON (OSSAnalyzerConfig.json). En este diccionario de extensiones se almacena como clave la extensión, y como valor el lenguaje de programación. Para la construcción de este diccionario se ha usado el fichero YAML languages.yml del repositorio Linguist de GitHub, que contiene una gran cantidad de lenguajes.

- Obtener el total de modificaciones realizadas sobre un mismo lenguaje de programación de cada uno de los contribuyentes al repositorio: para ello se ha creado la función counter_ext, almacenada también en utils.py y cuya salida consiste en un diccionario con la suma de modificaciones realizadas sobre todos los lenguajes que cada uno de los usuarios ha realizado.

Además para enriquecer algo más el análisis y ofrecer otros datos que no podemos obtener usando Perceval, se han incluido también algunas llamadas a la API de GitHub para obtener los siguientes datos:

- Porcentaje de uso de cada uno de los lenguajes usados en el repositorio. Mediante la función obtain_used_languages_on_repo del fichero requests_to_github_api.py se obtienen todos los ficheros almacenados en el repositorio. Una vez tenemos la lista con todos los ficheros, usando su extensión, se convierte a ”lenguaje”de igual manera que se hace con los datos que hemos obtenido de Perceval anteriormente, es decir, usando el diccionario extensiones del fichero de configuración OSSAnalyzerConfig.json. Con ello ya tenemos el total de ficheros de cada lenguaje, por lo que es fácil obtener el porcentaje de cada uno de los lenguajes sobre el total.

- Número total de ficheros del repositorio. Se obtiene usando la función obtain_num_files del fichero requests_to_github_api.py.

Por último, para terminar nuestra recolección de datos, puesto que nos centramos en los lenguajes minoritarios, necesitamos saber qué lenguajes son minoritarios en un repositorio, y la información sobre los mismos. Entendemos que un lenguaje es minoritario en un repositorio si representa una cuota menor del 5 % del total de un repositorio. Por ejemplo:

Un repositorio contiene 5 lenguajes de programación: Python, HTML, CSS, JavaScript y un Dockerfile. Python representa el 80 % del repositorio, HTML un 10 %, CSS un 7 %, JavaScript, un 2 % y Dockerfile, el 1 % restante. En este caso, los lenguajes minoritarios serán JavaScript y Dockerfile.

Teniendo esto en mente, debido a que se obtiene el porcentaje de uso de cada uno de los lenguajes usado en los repositorios como se ha comentado anteriormente, se puede saber cuales de los lenguajes que forman parte de un repositorio son minoritarios. Ahora que tenemos los lenguajes minoritarios, buscamos el resto de información sobre estos: los commits hechos sobre esos lenguajes y quién de los usuarios contribuyentes han realizado estos commits.

Todo esto se hace de la siguiente manera:

- Obtención de los lenguajes minoritarios: con el diccionario de porcentajes de cada uno de los lenguajes de un repositorio, usando la función obtain_min_languages del fichero utils.py nos quedamos con una lista con solo los lenguajes que representan menos del 5 % que hemos definido.

- Número total de commits realizados sobre esos lenguajes minoritarios: se realiza un diccionario en el que se hace la suma total de commits realizados sobre cada lenguaje minoritario usando la lista anterior y los datos que hemos obtenido de los commits con Perceval con el total de modificaciones realizadas cada uno de los lenguajes por cada contribuyente al repositorio. Esta función la realiza el método obtain_total_commits_min_languages del fichero utils.py.

- Obtención de los contribuyentes y su participación en los lenguajes minoritarios: mediante la función top_contribuyebtes_por_lenguaje del fichero utils.py se construye un diccionario con el número de commits realizados por los usuarios que han contribuido en estos lenguajes.

Ya se tienen todos los datos necesarios para nuestro análisis, por tanto, se procede a su almacenamiento en la base de datos.

En caso de que el repositorio no esté almacenado,se crea una fila en la tabla Repository de la base de datos con diferentes columnas con cada uno de los datos de interés para el análisis. El otro caso que se puede dar es que el repositorio que se haya analizado sea uno que ya existía

en la tabla Repository, pero de una versión posterior, por lo que en lugar de crear una nueva fila, se actualizan las columnas de la fila ya existente, de manera que no se gasta espacio innecesario en la base de datos.

También, además de almacenar cada repositorio en la tabla Repository, se guardan todos los usuarios que han participado en cada uno de los repositorios analizados junto a los repositorios que ha participado y los commits que ha realizado en los mismos. Esto nos permite ver el expertise de cada uno de los usuarios y conocer en qué lenguaje se especializa y qué número de lenguajes totales ha usado en todos los repositorios que haya participado y se hayan analizado previamente en la aplicación. Todo esto se almacena en la tabla User_expertise de la base de datos.

Una vez almacenados los distintos datos, solo queda la representación de los mismos en la aplicación web. Se presenta de la siguiente manera: imagen

En la parte superior central, se informa al usuario del repositorio que se ha pedido analizar y el total de ficheros que tiene el repositorio.

En la parte izquierda, se presenta una gráfica circular con la representación de porcentaje de cada uno de los lenguajes usados junto a una leyenda para inforar al usuario del número de porcentaje y el color que representa. La gráfica se genera a partir del diccionario con el porcentaje de uso de cada uno de los lenguajes usados en el repositorio que se ha obtenido en la recolección de datos usando la librería de python Matplotlib. Además, se presentan los lenguajes minoritarios del repositorio y el número de commits totales realizados sobre cada uno de ellos. Si se clicka en cualquiera de ellos, muestra a los usuarios que han realizado los commits y la cantidad.

En la parte derecha, se muestran todos los usuarios que han contribuido, y al hacer click sobre cualquiera de ellos, muestra los commits realizados sobre cada uno de los lenguajes del repositorio.

Por otro lado, si el repositorio que el usuario pide analizar ya tiene los datos de la última versión almacenada en la base de datos, estos se recuperan mediante una consulta en la que se pide a la tabla Repository la fila correspondiente al repositorio, y con ella se recuperan de nuevo todos los datos para representarlos en la web.

4.5. Endpoint /usersearch

El objetivo del endpoint /usersearch es facilitar al usuario todos los repositorios públicos que pertenecen a un mismo propietario de manera que pueda localizar fácilmente los repositorios de un mismo usuario sobre los que hacer los análisis de lenguajes minoritarios. Sobre todo, está dirigido de cara a usuarios propietarios que consistan en un grupo de personas que suelen participar en los mismos proyectos.

El funcionamiento es de la siguiente manera:

- En caso de que llegue una petición al endpoint con método GET, se ofrece en la respuesta un formulario donde el usuario debe indicar el nombre de usuario de GitHub del cual quiera consultar los repositorios junto a un botón de envío para provocar el envío de una petición al mismo endpoint /userrepo, pero en este caso, con método POST cuyo cuerpo contiene un parámetro llamado username y su valor es lo que haya intoducido el usuario.

- Cuando llega esa petición POST, el servidor procede a buscar los repositorios públicos del usuario que ha llegado en el body de la petición. Esto se hace mediante la función obtain_user_repos del fichero requests_to_github_api.py, cuya salida es un diccionario con el nombre de todos los repositorios que posee ese usuario. Este diccionario es lo que se forma la respuesta final del endpoint.

Al finalizar, se representa de la siguiente manera en la aplicación web:

En la parte superior se informa al usuario del nombre de usuario que ha escogido.

En la parte inferior se muestran todos los repositorios públicos del usuario.

Lo que hace que facilite al usuario la experiencia de analizar varios repositorios de un mismo usuario es que, al clickar en cualquiera de los repositorios que aparecen en la lista, se realiza una llamada POST al endpoint /userrepo con el nombre de usuario y el nombre del repositorio. Como se ha explicado en el punto anterior, en el endpoint /userrepo es donde se realizan los análisis.

Capítulo 5

Experimentos y validación

Este capítulo se introdujo como requisito en 2019. Describe los experimentos y casos de test que tuviste que implementar para validar tus resultados. Incluye también los resultados de validación que permiten afirmar que tus resultados son correctos.

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] Flask documentation.
https://flask.palletsprojects.com/en/stable/.

[2] Git.
https://git-scm.com/.

[3] Github.
https://github.com/.

[4] Jinja documentation.
https://jinja.palletsprojects.com/en/stable/.

[5] Python.
https://www.python.org/.

[6] A. F. Montoro. *Python 3 al descubierto*. Alfaomega, 2013.