UNIVERSIDAD
REY JUAN CARLOS

INGENIERÍA DE TELECOMUNICACIÓN

Curso Académico 2016/2017

Proyecto Fin de Carrera

ANÁLISIS DE PROYECTOS PYTHON DE GITHUB CON COALA

Autor : Raúl Sánchez López

Tutor : Dr. Gregorio Robles

Proyecto Fin de Carrera

Análisis de proyectos Python de GitHub con Coala

Autor : Raúl Sánchez López

Tutor : Dr. Gregorio Robles Martínez

La defensa del presente Proyecto Fin de Carrera se realizó el día de de 2017, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2017

Dedicado a
mi familia.

II

Agradecimientos

¡Por fin llegó el día! Con la entrega de este proyecto doy por terminada esta etapa de mi vida. Ha sido un camino largo, muchas horas de estudio, días de biblioteca, alegrías y decepciones, pero todo esfuerzo tiene su recompensa. Al final de esta etapa no solo me llevo el título, también me llevo un trocito de todas esas personas con las que he compartido mis días y que han estado ahí para hacerlo todo más llevadero.

A Celia, por tu amistad y apoyo incondicional; a Carlos, por todos esos momentos compartidos dentro y fuera de la universidad; a Álvaro, por esos días en su casa agobiados con las prácticas y las discusiones futboleras; a Borja, por esos primeros días y todos los viajes camino a la universidad; a Joel, Pedro, César, Larraco, Victor, Sergio, Hugo, Alberto, y a todos los que me dejo, con quienes he compartido muchas horas de estudio, dudas, alegrías, confidencias y muchísimas risas.

A mi tutor, Gregorio, por su ayuda y consejo para la realización de este trabajo.

A mis amigos, que fueron un apoyo y una vía de escape en momentos de agobio.

Y por supuesto a mi familia, que siempre me ha apoyado en todos los sentidos. Vero, gracias por estar siempre. Abuelos, por vuestra forma de afrontar las adversidades y demostrarme que es posible superar cualquier cosa. Pero sobre todo a mis padres, por todo su apoyo y el enorme esfuerzo que han hecho para que yo pudiera conseguir este objetivo, todo esto es tan mío como vuestro.

¡Gracias por todo!

Resumen

Este proyecto tiene como objetivo la creación de una aplicación web para el análisis de proyectos de GitHub con Coala. Este análisis puede dar una idea acerca de la calidad de los proyectos analizados, ya que a través de los diferentes 'bears' con los que cuenta, es capaz de detectar un gran número de problemas tanto en el código como en el proyecto en sí mismo y mostrarlos de forma clara y ordenada.

El diseño está basado en el modelo cliente-servidor, de modo que aunque ambos puedan estar alojados en la misma máquina, existe una separación que permite por un lado centrarse en la lógica de la aplicación y por otro en la presentación de los resultados.

La aplicación web se ha desarrollado con Python, utilizando Django, y tecnologías como Coala, BootStrap, CSS, HTML, JSON y JavaScript. El proyecto está enfocado al análisis de proyectos Python alojados en GitHub, aunque con unos pequeños cambios podría ser utilizado para analizar código de cualquier otro de los muchos lenguajes de programación que soporta Coala.

Los datos obtenidos tras el análisis con Coala se almacenan en una base de datos que el usuario puede a) consultar a través de la página de la aplicación y b) utilizar las diferentes funcionalidades.

Summary

This project aims to create an application web in order to analyse GitHub projects with Coala. The analysis can provide an idea about the quality of the analysed projects, since through the variety of their 'bears' it is able to detect a large number of problems both in the code and in the project itself and show the results clearly and organized.

The design is based on the client-server model, so that both can be hosted in the same computer but there is a separation between the application logic and the presentation of the results.

The application web has been developed with Python, using the Django framework, and technologies as Coala, BootStrap, CSS, HTML, JSON and JavaScript. The project is focused on the analysis of Python projects hosted in Github, however, with a few changes, it could be used to analyse code of any other language supported by Coala.

The collected data after Coala analysis are stored in a database that the user a) can consult through the application web and b) use the application funcionalities.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Presentación

El presente documento es la memoria del trabajo realizado para mi proyecto fin de carrera de Ingeniería de Telecomunicación. El proyecto consiste en la creación de una herramienta para analizar proyectos Python alojados en la plataforma GitHub, apoyado en la herramienta Coala.

A la hora de desarrollar un proyecto es importante localizar y solucionar los posibles problemas que se presenten. Si contamos con una herramienta que pueda detectar posibles errores y, adicionalmente, alertar sobre discrepancias con la guía de estilo del lenguaje de programación que estemos usando, tendremos la información necesaria para ayudarnos a solucionar los errores y mejorar la calidad de nuestro proyecto.

La aplicación web almacena la información recogida tras analizar los proyectos y la muestra de forma ordenada a varios niveles, facilitando la visualización de los datos. En este caso, la aplicación se ha orientado al análisis de proyectos Python, aunque puede servir como punto de partida para, haciendo algunos cambios, analizar proyectos desarrollados en otros lenguajes de programación, gracias a que Coala es compatible con un gran número de lenguajes de programación.

Básicamente contamos con dos funcionalidades principales: analizar proyectos y consultar los datos almacenados. A la hora de crear la interfaz web he pensado en la sencillez, de modo que el usuario pueda hacer uso de forma rápida, sencilla e intuitiva de las características de la herramienta, mostrando la información de forma clara y ordenada.

1.2. Estructura de la memoria

La memoria está estructurada del siguiente modo:

- **Capítulo 1. Introducción.** Descripción general del proyecto y breve explicación de la estructura del mismo.
- **Capítulo 2. Objetivos.** Exposición del objetivo general del trabajo, así como los objetivos más específicos que se pretenden alcanzar con este trabajo. Finalmente se incluye la planificación temporal seguida.
- **Capítulo 3. Estado del arte.** Descripción de las diferentes tecnologías utilizadas para el desarrollo de la aplicación web.
- **Capítulo 4. Diseño e implementación.** Explicación del desarrollo, estructura y funcionamiento del proyecto basado en las tecnologías descritas en el punto anterior.
- **Capítulo 5. Resultados.** Análisis de los resultados obtenidos.
- **Capítulo 6. Conclusiones.** Reflexión y conclusiones finales sobre el proyecto.

Capítulo 2

Objetivos

2.1. Objetivo general

El objetivo principal de este trabajo es desarrollar una aplicación web que sirva a analizar el código fuente de proyectos Python, así como mostrar los resultados de estos análisis.

2.2. Objetivos específicos

Para alcanzar el objetivo principal se han perseguido los siguientes objetivos específicos:

- Estudiar las funcionalidades de Coala y de los 'bears' compatibles con Python.
- Examinar los resultados del análisis con Coala a partir del fichero JSON generado y mapear estos datos para tratarlos posteriormente.
- Crear la primera versión de la herramienta capaz de analizar una lista de proyectos Python alojados en GitHub.
- Trasladar la herramienta a la aplicación web con Django, adaptando las vistas y organizando la base de datos para mostrar la información obtenida de forma sencilla.
- Optimizar el código y aplicar mejoras visuales que faciliten el uso de la herramienta de forma que esta sea más rápida y sencilla de utilizar.

2.3. Planificación temporal

La planificación temporal seguida para la elaboración de este proyecto, teniendo en cuenta que entre hitos he mantenido reuniones con mi tutor, ha sido la siguiente:

- Reunión con el tutor en busca de ayuda y consejo para la elaboración del proyecto. En ella acordamos tomar como base Coala para analizar proyectos Python.
- Estudio de la herramienta Coala, para aprender como se usa, la diferentes funcionalidades, funcionamiento de los 'bears' y ficheros de configuración, etc. Además del estudio de los resultados del análisis de Coala y los ficheros JSON generados dependiendo de la configuración utilizada.
- Definición de la estructura de la base de datos, mapeo de JSON obtenidos y creación de una primera versión de la herramienta, con una salida rudimentaria para mostrar los resultados.
- Desarrollo de la aplicación web con Django, llevando el programa principal a las vistas y la estructura de la base de datos a los modelos de Django.
- Mejora de aspecto de la web e inclusión de nuevas funcionalidades como la búsqueda por URL o fichero, búsqueda de resultados y presentación de datos.
- Optimización de la aplicación, incluyendo tablas de resumen con enlaces a los resultados para disminuir los tiempos de ejecución de los análisis.
- Elaboración de la memoria.

Capítulo 3

Estado del arte

Para la realización de este proyecto me he apoyado en tecnologías ya existentes que serán presentadas en este capítulo, desde la herramienta de análisis con la que se procesa cada proyecto hasta las herramientas utilizadas para mostrar los resultados en la aplicación web.

3.1. Coala

Coala es una herramienta que proporciona una interfaz unificada para encontrar y arreglar problemas en el código con un único fichero de configuración (coafile), independientemente del lenguaje de programación utilizado. La herramienta permite obtener los resultados del análisis en un fichero JSON fácilmente analizable, o personalizarlo acorde a las necesidades. [2]

Coala dispone de un conjunto de plugins denominados 'bears' compatibles con diferentes lenguajes de programación incluyendo Python, C/C++, JavaScript, CSS o Java entre otros, además de plugins compatibles con todos los lenguajes.

En el presente proyecto, se ha utilizado Coala para analizar proyectos Python, por lo que sólo se han utilizado 'bears' compatibles con este lenguaje de programación. Una vez ejecutada la herramienta, se ha analizado el fichero JSON generado por Coala.

3.1.1. coafile

Fichero de configuración[1] de Coala. Se pueden utilizar hasta tres coafiles para configurar la herramienta:

- Coafile por proyecto: Por convención este fichero es nombrado '.coafile' y se aloja en el directorio raíz del proyecto. Con esta convención, solo hay que ejecutar Coala y la herramienta tomará la configuración del fichero.

  Los ajustes definidos en este fichero de configuración anulan la configuración dada por el resto de ficheros de configuración y solo pueden ser anulados por aquellas opciones de configuración que introduzcamos por línea de comandos.

  El fichero de configuración se puede dividir en secciones y puede contener los 'bears' que vamos a utilizar y las opciones de configuración para cada uno de ellos. Hay que tener en cuenta que la mayor parte de las opciones de configuración de los 'bears' son opcionales, por lo que sólo estaremos obligados a incluir aquellas que sean necesarias para realizar el análisis correctamente.

- Coafile por usuario: Este fichero, nombrado como '.coarc', se coloca en el 'home' para establecer los ajustes por defecto del usuario. Estos ajustes serán tomados en cuenta para todos los proyectos que vayan a ser analizados por el usuario, y solo anulan los que sean especificados por el fichero de configuración del sistema.

- Coafile por sistema: Es el fichero de configuración de más baja prioridad. El denominado 'default_coafile' debe situarse en el directorio de instalación de Coala y es válido para todos los usuarios que utilicen Coala en el sistema.

3.1.2. Bears

Los bears[2] son los plugins utilizados por Coala para análizar código. Actualmente cuenta con 78 bears que cubren un total de 54 lenguajes de programación, pero en este caso me centraré en aquellos compatibles con Python:

[1] `http://docs.coala.io/en/latest/Users/coafile.html`
[2] `https://coala.io/#!/languages`

BanditBear

Con este bear se analiza el código Python en busca de los problemas de seguridad más comunes con la herramienta Bandit[3]. Bandit procesa cada uno de los ficheros, construye un AST[4], ejecuta los plugins adecuados contra los nodos AST y genera un informe. La herramienta cuenta con una lista de test que se identifican mediante un ID.

- Lenguajes: Python, Python 2, Python 3
- Configuración
  - BANDIT_SKIPPED_TESTS: De manera opcional, podemos indicar en el fichero de configuración el ID de los test que no queramos ejecutar. Por defecto, este bear no ejecuta los plugin: B105 hardcoded_password_string, B106 hardcoded_password_funcarg, B107 hardcoded_password_default, B404 import_subprocess, B603 subprocess_without_shell_equals_true, B606 start_process_with_no_shell y B607 start_process_with_partial_path

FilenameBear

Este bear se puede aplicar a cualquier lenguaje de programación. Comprueba que el nombre de los ficheros siguen una convención.

- Lenguajes: Todos
- Configuración
  - FILE_NAMING_CONVENTION: De manera opcional, se puede indicar una de las siguientes convenciones de nombrado: camel (thisIsCamelCase) - pascal (ThisIsPascalCase) - snake (this_is_snake_case) - space (This Is Space Case). Por defecto, elige 'snake'.
  - IGNORE_UPPERCASE_FILENAMES: De manera opcional, se puede indicar si se quiere ignorar los nombres de fichero completamente en mayúsculas. Por defecto se ignoran.

[3] `https://github.com/openstack/bandit`

[4] `https://docs.python.org/2/library/ast.html`

InvalidLinkBear

Busca enlaces en los ficheros y comprueba si son válidos. Considera válido el enlace si el servidor responde con un código 2xx. Este bear hará peticiones HEAD a todas las URLs del fichero, por lo que es potencialmente peligroso si el código contiene URLs que no deben ser visitadas. Por ejemplo, si el fichero contiene una línea como la siguiente do_not_ever_open = 'https://api.acme.inc/delete-all-data', este bear la visitará, borrando todos los datos.

- Lenguajes: Todos
- Configuración
  - NETWORK_TIMEOUT o TIMEOUT: Se trata de un dict[5] mapeando URLs y tiempos de espera para cada enlace. Las URLs con el mismo host que las contenidas en el dict, esperarán el tiempo indicado. También puede contener un campo comodín con la clave '*' para el resto de URLs. Es opcional y por defecto viene vacío.
  - LINK_IGNORE_REGEX o IGNORE_REGEX: Una expresión regular para identificar las URLs a ignorar. Es opcional y tiene un valor por defecto para URLs del tipo 'example.com'.
  - LINK_IGNORE_LIST: Lista de URLs a ignorar. Es opcional y por defecto está vacía.
  - FOLLOW_REDIRECTS: Si está en 'True' corrige los enlaces con errores, pero ignora las URLs que son muy diferentes a la URL original. Es opcional y por defecto su estado es 'False'.

MypyBear

Comprueba la escritura estática usando la herramienta Mypy[6]. La herramienta chequea el código en busca de los errores de tipado más comunes. Estas anotaciones referentes al tipo dentro del código no interfieren al ejecutar el programa al tratarse de un analizador estático.

Mypy puede comprobar que el código sigue las reglas básicas de anotaciones basadas en comentarios para Python 2 y Python 3 (utilizando la PEP 484[7]).

[5] https://docs.python.org/2/library/stdtypes.html#typesmapping
[6] http://mypy.readthedocs.io/en/latest/basics.html
[7] https://www.python.org/dev/peps/pep-0484/

- Lenguajes: Python, Python 2, Python 3
- Configuración
  - LANGUAGE: Con 'Python' o 'Python 3', chequea código Python 3.x, mientras que la opción 'Python 2' chequea código en Python 2.x. Es opcional y la opción por defecto es 'Python 3'.
  - ALLOW_UNTYPED_FUNCTIONS: Si su valor es 'True', permite funciones sin anotaciones de tipo o con anotaciones de tipo incompletas. Por defecto el valor es 'True'.
  - STRICT_OPTIONAL: Habilita comprobaciones estrictas relacionadas con tipos opcionales, comprobando que no se intentan realizar operaciones no admitidas en valores 'None', o que no se utilizan valores 'None' cuando se espera un valor no opcional. Esta característica es todavía experimental[8]. Es opcional y por defecto su valor es 'False'.
  - PYTHON_VERSION: Permite especificar la versión de Python concreta. Es opcional y por defecto su valor es 'None'.
  - CHECK_UNTYPED_FUNCTION_BODIES: Permite no chequear el interior de las funciones que no tengan anotaciones de tipo. Es opcional y por defecto su valor es 'False'.
  - ALLOW_UNTYPED_CALLS: Permite la llamada a funciones sin anotaciones de tipo desde funciones tipadas. Es opcional y por defecto su valor es 'True'.

PEP8Bear

Detecta y corrige código que no cumpla con las normas de la guía de estilo para Python (PEP8[9]). Este bear no modifica la funcionalidad del código.

- Lenguajes: Python, Python 2, Python 3
- Configuración

[8] `http://mypy-lang.blogspot.com.es/2016/07/mypy-043-released.html`
[9] `https://www.python.org/dev/peps/pep-0008/`

- MAX_LINE_LENGTH: Permite definir el número máximo de caracteres por línea. Es opcional y por defecto son 79.
- INDENT_SIZE o TAB_WIDTH: Permite definir el número de espacios para cada nivel de indentación. Es opcional y por defecto vale 4.
- PEP_IGNORE: Lista de errores o advertencias a ignorar. Es opcional y por defecto aparece vacía.
- PEP_SELECT: Lista de errores o advertencias para aplicar exclusivamente. Es opcional y por defecto aparece vacía.
- LOCAL_PEP8_CONFIG: Si está a 'True', 'autopep8'[10] (usado para corregir el código para que cumpla con la guia de estilo) deberá usar un fichero de configuración indicado. Es opcional y por defecto su valor es 'False'.

PyCommentedCodeBear

Detecta código fuente comentado en Python. No tiene opciones de configuración.

- Lenguajes: Python, Python 2, Python 3

PycodestyleBear

Analiza el código con la herramienta Pycodestyle[11], formalmente conocida como PEP8. Sirve para comprobar que el código sigue las convenciones de estilo definidas para Python en PEP8.

Pycodestyle permite agregar nuevas normas de forma sencilla gracias a su estructura. Además, su salida es parseable, es ligero y contiene una completa *suite* de pruebas. Actualmente consta de un completo listado[12] de códigos de error y advertencia. Hay que tener en cuenta que en la configuración por defecto se ignoran los códigos E121 (línea de continuación insuficientemente indentada), E123 (la indentación del corchete de cierre no coincide con la de la línea del corchete de apertura), E126 (línea de continuación sobreindentada), E133 (corchete de cierre sin indentación), E226 (espacio en blanco faltante alrededor de un operador aritmético), E241

[10] `https://pypi.python.org/pypi/autopep8`
[11] `https://pycodestyle.readthedocs.io/en/latest/`
[12] `https://pycodestyle.readthedocs.io/en/latest/intro.html#error-codes`

(múltiples espacios después de una ‘,’), E242 (tabulación después de una ‘,’), E704 (múltiples declaraciones en la definición de una función) y W503 (salto de línea antes de un operador binario), ya que se corresponden con reglas que no están aceptadas de forma unánime y PEP8 no obliga a que sean cumplidas. El código E133 es mutuamente exclusivo con el código E123.

- Lenguajes: Python, Python 2, Python 3
- Configuración
  - MAX_LINE_LENGTH: Permite definir el número máximo de caracteres por línea. Es opcional y por defecto son 79.
  - PYCODESTYLE_IGNORE: Lista con los errores a ignorar de entre todos los códigos[13]. Es opcional y por defecto está vacío.
  - PYCODESTYLE_SELECT: Lista con los errores a detectar de entre todos los códigos. Es opcional y por defecto está vacío.

PyFlakesBear

Analiza los ficheros Python en busca de errores con la herramienta PyFlakesBear[14], que puede detectar problemas de sintaxis, código sin uso y elementos indefinidos.

Es una herramienta que no tiene en cuenta el estilo. Es más limitada que otras herramientas debido a que sólo examina el árbol de sintaxis de los archivos de forma individual, pero esta característica también lo hace más rápida. No tiene disponibles opciones de configuración con Coala.

- Lenguajes: Python, Python 3

PyImportSortBear

Detecta los casos relacionados con la forma en que se ordenan los imports y es capaz de añadir comentarios basados en las muchas opciones de configuración que posee. Se basa en la herramienta ‘isort’[15], que se utiliza para ordenar los ‘imports’ alfabéticamente y por secciones.

[13] `http://www.pydocstyle.org/en/latest/error_codes.html`
[14] `https://github.com/PyCQA/pyflakes`
[15] `https://isort.readthedocs.io/en/latest/`

- Lenguajes: Python, Python 2, Python 3
- Configuración
  - USE_PARENTHESES_IN_IMPORT:'True' si se han de utilizar paréntesis en la declaración de los imports. Es opcional y por defecto es 'True'.
  - FORCE_ALPHABETICAL_SORT_IN_IMPORT: Si su valor es 'True', fuerza a que todos los imports sean ordenados en una sola sección, en lugar de estar dentro de otros grupos. Es opcional y por defecto es 'False'.
  - FORCE_SORT_WITHIN_IMPORT_SECTIONS: Si su valor es 'True', los imports estarán ordenados según su sección sin importar el tipo. Es opcional y por defecto es 'True'.
  - FROM_FIRST_IN_IMPORT: Si es 'True', los imports usando 'from' se muestran encima del resto. Es opcional y por defecto es 'False'.
  - INCLUDE_TRAILING_COMMA_IN_IMPORT: Si es 'True' añadirá una coma final a los imports. Ejemplo: 'from abc import (a,b,c,)'. Es opcional y por defecto es 'False'.
  - COMBINE_STAR_IMPORTS: Si es 'True', se asegura de que haya un 'import *', y nada más es importado desde ese 'namespace'. Es opcional y por defecto es 'True'.
  - COMBINE_AS_IMPORTS: Si es 'True', isort combinará los imports dentro de la misma línea para las declaraciones de import. Es opcional y por defecto es 'True'.
  - LINES_AFTER_IMPORTS: Fuerza que haya un cierto número de líneas después de la declaración de los imports y antes de la primera línea funcional de código. Por defecto es '-1', que utiliza 2 líneas si la primera línea de código es una clase o función y una línea para el resto de casos.
  - ORDER_IMPORTS_BY_TYPE: Si es 'True', isort crea secciones separadas dentro de los 'from' para constantes, clases y módulos/funciones. Es opcional y por defecto es 'False'.
  - BALANCED_WRAPPING_IN_IMPORTS: Si es 'True', para cada import multilínea isort ajustará dinámicamente la longitud del import a una que guarde el equilibrio con la máxima longitud definida. Es opcional y por defecto es 'False'.

- IMPORT_HEADING_LOCALFOLDER: Comentario colocado encima de los imports que comienza por '.'. Es opcional y por defecto está vacío.
- IMPORT_HEADING_FIRSTPARTY: Comentario colocado encima de los imports del proyecto. Es opcional y por defecto está vacío.
- IMPORT_HEADING_THIRDPARTY: Comentario colocado encima de los imports de terceros. Es opcional y por defecto está vacío.
- IMPORT_HEADING_STDLIB: Comentario colocado encima de los imports de la librería estándar. Es opcional y por defecto está vacío.
- IMPORT_HEADING_FUTURE: Comentario colocado encima de futuros imports. Es opcional y por defecto está vacío.
- DEFAULT_IMPORT_SECTION: Sección por defecto para colocar los imports si su sección no puede ser determinada automáticamente. Es opcional y por defecto es 'FIRSTPARTY'.
- FORCE_GRID_WRAP_IMPORTS: Fuerza a los 'from' a estar dentro de la cuadrícula independientemente de la longitud de línea. Es opcional y por defecto es 'False'.
- FORCE_SINGLE_LINE_IMPORTS: Si es 'True', cada import es forzado a mostrarse en una sola línea en lugar de tener imports multi-línea. Es opcional y por defecto es 'True'.
- SORT_IMPORTS_BY_LENGTH: Ordena los imports por longitud en lugar de alfabéticamente. Es opcional y por defecto es 'False'.
- USE_SPACES: Será 'True' para utilizar espacios en lugar de tabulaciones. Es opcional y por defecto es 'True'.
- INDENT_SIZE: Número de espacios por nivel de indentación. Es opcional y por defecto es 4.
- FORCED_SEPARATE_IMPORTS: Lista de módulos que deseamos que aparezcan en una sección propia. Es opcional y por defecto está vacía.
- ISORT_MULTI_LINE_OUTPUT[16]: Entero que representa como se muestran los imports si no son lo suficientemente largos como para mostrarse en múltiples líneas.

[16] `https://isort.readthedocs.io/en/latest/#multi-line-output-modes`

Es opcional y por defecto su valor es 4 (vert-grid).

- KNOWN_FIRST_PARTY_IMPORTS: Lista de imports que está forzado a mostrar dentro de los imports del proyecto. Es opcional y por defecto está vacía.
- KNOWN_THIRD_PARTY_IMPORTS: Lista de imports que está forzado a mostrar dentro de la categoría de imports de terceros. Es opcional y por defecto está vacía.
- KNOWN_STANDARD_LIBRARY_IMPORTS: Lista de imports que está forzado a mostrar dentro de la categoría estándar de imports. Es opcional y por defecto está vacía.
- MAX_LINE_LENGTH: Máximo número de caracteres por línea. Es opcional y por defecto es 79.
- IMPORTS_FORCED_TO_TOP: Fuerza a que la lista de imports aparezca encima de su respectiva sección. Es opcional y por defecto está vacía.
- TAB_WIDTH: Número de espacios por nivel de indentación. Es opcional y por defecto es 4.

PyLintBear

Comprueba el código con PyLint[17], ejecutándolo para cada fichero por separado. PyLint detecta errores de sintaxis, código no usado, errores de formato, duplicidad y problemas de seguridad, gracias al amplio listado de comprobaciones[18] que lleva a cabo.

La herramienta cuenta con un listado de errores[19] que serán mostrados por Coala.

- Lenguajes: Python, Python 2, Python 3
- Configuración
  - PYLINT_DISABLE: Inhabilita los mensajes, reportes, categoría o chequeos correspondientes a las IDs dadas. Es opcional y por defecto es una lista vacía.
  - PYLINT_ENABLE: Habilita los mensajes, reportes, categoría o chequeos correspondientes a las IDs dadas. Es opcional y por defecto es una lista vacía.

[17] `https://www.pylint.org/`
[18] `http://pylint-messages.wikidot.com/all-messages`
[19] `http://pylint-messages.wikidot.com/all-codes`

- PYLINT_RCFILE: Especifica el RCFile[20] para PyLint. Es opcional y por defecto está vacío.
- PYLINT_CLI_OPTIONS: Opciones por línea de comandos que se quieran pasar a PyLint. Es opcional y por defecto está vacío.

PyUnusedCodeBear

Detecta código en desuso. Por defecto está limitado a declaraciones 'pass' e imports innecesarios.

- Lenguajes: Python, Python 2, Python 3
- Configuración
  - REMOVE_ALL_UNUSED_IMPORTS: Si es 'True', elimina todos los imports que no se utilizan. Puede tener efectos secundarios. Es opcional y por defecto es 'False'.

PyDocStyleBear

Comprueba la semántica y convenciones relacionadas con los 'docstrings'[21] de Python.

- Lenguajes: Python, Python 2, Python 3
- Configuración
  - PYDOCSTYLE_SELECT: Lista que especifica los errores que se van a chequear. No puede usarse a la vez que 'PYDOCSTYLE_IGNORE'. Es opcional y por defecto está vacía.
  - PYDOCSTYLE_IGNORE: Lista que especifica los errores que se van a ignorar. No puede usarse a la vez que 'PYDOCSTYLE_SELECT'. Es opcional y por defecto está vacía.

[20] `https://en.wikipedia.org/wiki/RCFile`
[21] `https://www.python.org/dev/peps/pep-0257`

PyromaBear

Comprueba que se llevan a cabo buenas prácticas en lo referente al empaquetado utilizando Pyroma[22]. La herramienta Pyroma está destinada a calificar como de bien cumple con las mejores prácticas un proyecto Python, así como un listado con los puntos que podrían ser mejorados.

Pyroma puede ayudar a diseñar un proyecto agradable y fácilmente utilizable, así como mejorar el software Python de terceros.

No cuenta con opciones de configuración en Coala.

- Lenguajes: Python, Python 3

RadonBear

Radon sirve para calcular la complejidad de un fichero dado.

- Lenguajes: Python, Python 2, Python 3
- Configuración
  - RADON_RANKS_INFO: Categorías, dadas por Radon, para tratar la gravedad INFO. Es opcional y por defecto es una lista vacía.
  - RADON_RANKS_NORMAL: Categorías, dadas por Radon, para tratar la gravedad NORMAL. Es opcional y por defecto es '('C', 'D')'.
  - RADON_RANKS_MAJOR:Categorías, dadas por Radon, para tratar la gravedad MAJOR. Es opcional y por defecto es '('E', 'F')'.

VultureBear

Comprueba el código Python utilizando Vulture[23]. Esta herramienta busca en el código clases, funciones y variables que no son utilizadas, lo que ayuda a limpiar y encontrar errores en el código.

Aunque Vulture es una herramienta muy útil, hay que tener en cuenta una serie de consideraciones. Debido a la naturaleza dinámica de Python, los analizadores de código estáticos

[22] `https://github.com/regebro/pyroma`

[23] `https://github.com/jendrikseipp/vulture`

como Vulture probablemente pueden saltarse partes del código en desuso. Además, el código que solamente es llamado implícitamente puede ser reportado como código no utilizado.

No cuenta con opciones de configuración en Coala.

- Lenguajes: Python, Python 3

coalaBear

Comprueba que la ortografía de 'coala' es correcta dentro del código. No cuenta con opciones de configuración.

- Lenguajes: Todos

SpaceConsistencyBear

Comprueba y corrige el espaciado. Incluye el uso de tabulaciones en lugar de espacios, espacios en blanco a la izquierda o al final de la línea y las nuevas líneas de texto que falten al final del fichero.

- Lenguajes: Todos
- Configuración
  - ALLOW_TRAILING_WHITESPACE: Si es 'True', se permiten espacios o tabulaciones después del último carácter que no sea un espacio en blanco en la línea antes del salto de línea. Es opcional y por defecto es 'False'.
  - INDENT_SIZE: Número de espacios por nivel de indentación. Es opcional y por defecto es 4.
  - ENFORCE_NEWLINE_AT_EOF: Fuerza o no a que haya una nueva línea al final del fichero. Es opcional y por defecto es 'True'.
  - TAB_WIDTH: Número de espacios por tabulación. Es opcional y por defecto es 4.
  - USE_SPACES: Si es 'True', deben usarse espacios en lugar de tabulaciones. Es obligatorio.

3.2. Python

Python [4] es un lenguaje de programación de código abierto optimizado para la calidad, productividad, portabilidad e integración creado a finales de los ochenta por Guido van Rossum [9] [5]. Sus principales características son:

- Lenguaje de alto nivel: automatiza la mayoría de las tareas a bajo nivel y hace hincapié en que su sintaxis sea sencilla, de forma que se adapta de forma adecuada a la capacidad cognitiva humana.
- Interpretado: de forma que solo necesita de un interprete para ser ejecutado directamente, sin necesidad de compilador.
- Multiparadigma: soporta más de un paradigma de programación, es decir, permite programar utilizando más de un estilo de programación (orientada a objetos, funcional, dinámica,...)
- Tipado dinámico: una variable puede tomar valores de distinto tipo en distinto momento. Los tipos se asignan en tiempo de ejecución y no es necesario declararlos con la variable.

Python ha sido utilizado en todo el proyecto debido a su facilidad de uso y su compatibilidad con DJANGO. Además, se trata de un lenguaje ampliamente utilizado y que cuenta con mucho material de apoyo.

3.3. Django

Django [3] [6] es un entorno de trabajo de alto nivel diseñado para alentar un desarrollo rápido y limpio, dando prioridad a un diseño pragmático. Con este entorno de trabajo solo hay que preocuparse de desarrollar la aplicación, sin tener en cuenta las consideraciones generales del desarrollo de aplicaciones web. Es un entorno versátil, flexible, rápidamente escalable, gratuito y de código abierto.

Django está diseñado para ayudar a los desarrolladores a desarrollar aplicaciones web lo más rápido posible. Incluye docenas de extras que podemos utilizar para manejar las tareas más comunes del desarrollo web, tales como autenticación de usuarios o la administración de contenidos entre otros.

Otro de los aspectos importantes de Django es la seguridad. Django ayuda a evitar errores de seguridad comunes como la inyección de SQL, cross-site scripting[24], etc. Su sistema de autenticación de usuarios proporciona una forma segura de administrar cuentas de usuario y contraseñas.

He utilizado Django para desarrollar la aplicación por su facilidad a la hora de crear las vistas y el interfaz web con el que se interactúa con la aplicación. Adicionalmente, con Django se simplifica considerablemente el manejo de bases de datos, un aspecto muy importante de este proyecto.

3.4. GitHub

GitHub[25] es una plataforma donde se almacenan repositorios, utilizando el control de versiones Git. Una de sus principales características es el desarrollo colaborativo.

Cada proyecto tiene su propia página web, la posibilidad de hacer ramificaciones (forks), control de versiones, wikis, gestión de errores y la posibilidad de que otros desarrolladores colaboren en el proyecto.

En la realización de este proyecto he utilizado la plataforma para obtener proyectos Python que analizar, clonados en local con Git, así como para almacenar tanto la aplicación como la memoria del proyecto. De este modo, ambas son públicas y están disponibles para su consulta o contribuciones de otros desarrolladores.

3.5. Git

Git [7] es un sistema de control de versiones que sirve para gestionar los cambios que sufre un proyecto durante su desarrollo. Una vez añadimos los elementos que debe gestionar, podemos hacer los cambios necesarios y estos quedarán registrados, manteniendo un registro histórico de las acciones que realicemos y contando con la posibilidad de volver a un estado anterior del proyecto.

Se trata de una de las principales herramientas de control de versiones. Se ha utilizado en la

[24] `http://www.cgisecurity.com/xss-faq.html#whatis`
[25] `https://github.com/`

realización de este proyecto tanto para clonar los proyectos Python que analiza la herramienta, a partir de las URLs de los proyectos en GitHub, como para llevar el control del propio proyecto.

3.6. HTML

HTML (HyperText Markup Language)[26] es el lenguaje de marcado estándar utilizado para la creación de páginas web y sus elementos forman los bloques de construcción de todas las páginas web. [10]

En este proyecto se ha utilizado para la creación de las diferentes páginas web que muestra Django y con las que interactúa el usuario para analizar o ver la información almacenada sobre los proyectos.

3.7. BootStrap

BootStrap [1] es un framework de código abierto desarrollado originalmente por diseñadores y desarrolladores de Twitter en 2010. Sirvió para el desarrollo de herramientas internas en la empresa durante más de un año, hasta que fue lanzado públicamente en 2011.

La herramienta consta de una colección de elementos personalizables y funciones para facilitar el diseño de páginas web. Hace uso de CSS, elementos HTML diseñados y mejorados con clases extensibles, un sistema avanzado de cuadrícula, componentes reutilizables como iconos o listas desplegables y JavaScript.

BootStrap se puede utilizar de diferentes formas, pero en este caso he descargado la versión precompilada, que permite hacer un uso rápido de ella con las fuentes que incluye, y la versión precompilada de su CSS y JavaScript. Además, he usado la plantilla 'Bootstrap theme' para la creación de las diferentes páginas con las que cuenta el proyecto, añadiendo o eliminando los elementos necesarios para mostrar la información deseada.

[26] `https://www.w3.org/html/`

3.8. CSS

CSS (Cascading Style Sheets) permite definir la forma en la que se muestra un contenido, facilitando a los desarrolladores el control sobre el formato y estilo de los documentos. [10]

La principal característica de CSS es que permite separar el contenido de la presentación. Un desarrollador puede controlar el formato y estilo de muchas páginas al mismo tiempo, ya que todas las páginas reflejaran el aspecto marcado en el CSS.

En este proyecto se utiliza CSS como parte de BootStrap, haciendo uso de las clases definidas por esta herramienta para mejorar la presentación de la aplicación web.

3.9. JavaScript

JavaScript[27] es un lenguaje ligero e interpretado. Puede ser utilizado orientado a objetos, al ser multi-paradigma; está basado en prototipos, dinámico e imperativo. Puede decirse que es el lenguaje de programación de las webs, porque los navegadores a día de hoy sólo tienen una *máquina* nativa de JavaScript, aunque también es utilizado en otros entornos.

JavaScript sirve para especificar el comportamiento de las páginas web, y forma parte de la 'triada' de tecnologías básicas para el desarrollo web junto con HTML y CSS. [8]

En este proyecto se ha utilizado JavaScript para ocultar o mostrar parte del contenido de las páginas. A la hora de mostrar la información de un proyecto o fichero, o tras analizar una lista de ellos, se presenta toda la información del mismo y gracias a JavaScript podemos ocultar parte de los datos para poder tener una visión global.

3.10. JSON

JSON[28] (JavaScript Object Notation) es un formato de texto ligero para el intercambio de datos. Es sencillo de leer e interpretar para humanos y puede ser fácilmente generado e interpretado por máquinas. Además, es un formato de texto totalmente independiente del lenguaje de programación y está formado por estructuras universales que son soportadas por todos ellos, por lo que es el lenguaje ideal para el intercambio de datos.

[27] `https://developer.mozilla.org/es/docs/Web/JavaScript`
[28] `http://www.json.org/`

JSON está constituido por dos estructuras: una colección de pares nombre/valor y una lista ordenada de valores.

En este proyecto se ha utilizado JSON como formato de salida de los análisis con Coala. Al tratarse de estructuras básica y fácilmente interpretables, es perfecto para su posterior análisis e inclusión de resultados en la base de datos una vez hecho el mapeo.

Capítulo 4

Diseño e implementación

4.1. Arquitectura general

La aplicación desarrollada está basada en el modelo cliente-servidor. En este modelo, cada una de las partes ejecuta acciones diferentes: el cliente realiza peticiones a la aplicación a través de la interfaz web; y el servidor atiende estas peticiones y da una respuesta al cliente. Además, hay que tener en cuenta que la mayoría de las peticiones, sino todas, conllevan el acceso a la base de datos, ya sea para almacenar o extraer información.

Aunque cliente y servidor pueden estar alojados en la misma máquina, existe una separación lógica entre ambas partes.

La aplicación dispone de las siguientes funcionalidades: análisis de proyectos dada una URL perteneciente a un proyecto alojado en GitHub, análisis de proyectos dado un fichero de texto que contenga URLs de proyectos alojados en GitHub, búsqueda de datos almacenados en la base de datos (proyectos, ficheros o bears encontrados), eliminación de proyectos de la base de datos, resúmenes de proyectos, ficheros y bears almacenados en la base de datos.

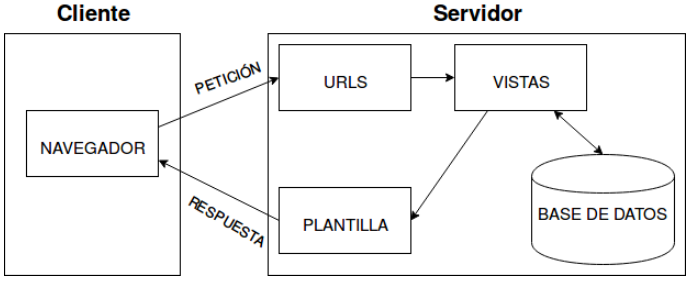


Figura 4.1: Esquema general de la aplicación.

4.1.1. Servidor

En el servidor es donde tenemos la aplicación en sí. Django dispone de un servidor por defecto, que es el que he utilizado para la realización de este proyecto.

A la hora de lanzar el servidor, Django permite elegir la IP y puerto en el que debe escuchar, pero se decidió utilizar la configuración por defecto: 'localhost:8000', y todas las funcionalidades de la herramienta colgarán de 'localhost:8000/analyzer/'.

Toda la lógica de la aplicación se encuentra en el fichero views.py. El cliente realiza una petición al servidor mediante la solicitud de una URL. La petición llega al controlador, que consulta en el fichero urls.py la vista asociada a esa URL y se hace la llamada a esa vista del fichero views.py. En este momento se ejecutan las acciones definidas en la vista y se consulta o actualiza la base de datos dependiendo de la acción concreta. Finalmente, se renderiza la página web a partir del template definido para cada caso y se envía al navegador mediante HTTP para que el cliente visualice el resultado de la consulta.

4.1.2. Cliente

En la parte del cliente es donde se encuentra el usuario de la aplicación, que accederá a ella a través de un navegador. El cliente realizará alguna de las peticiones descritas con anterioridad y las enviará al servidor. Una vez procesadas, recibirá la respuesta en forma de código HTML y será mostrada en el navegador.

Algunas templates utilizados en este proyecto cuentan con código JavaScript que permite esconder o mostrar parte de la información, por tanto el cliente también debe interpretar y ejecutar estos códigos.

4.1.3. Base de datos

La base de datos o el modelo como se conoce en Django, se define en el fichero models.py. Dentro del proyecto, la base de datos se organiza como muestra la figura 4.2.

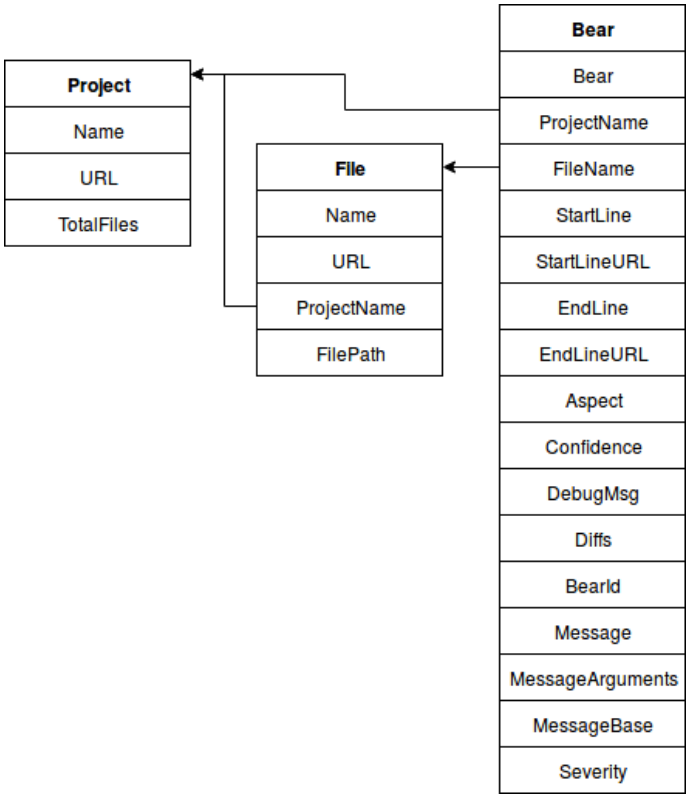


Figura 4.2: Estructura de la base de datos utilizada.

Se encuentra dentro del servidor y es consultada o actualizada por este en función de las peticiones del cliente. En ella se almacenan los datos obtenidos en los análisis de proyectos Python. Las distintas tablas se relacionan entre ellas por medio de los distintos campos, se trata de una base de datos relacional.

De este modo, cada Bear está asociado a un fichero y proyecto concreto; y cada fichero asociado a un proyecto tal y como se observa en la figura 4.2.

4.2. Diseño e implementación del servidor

El servidor se encarga de procesar las peticiones y mostrar la información al cliente. Cada vez que el cliente hace una petición, el servidor hace las consultas necesarias a la base de datos, introduce los datos de los nuevos proyectos o actualiza los existentes.

4.2.1. Análisis de proyectos

Una de las principales características de la aplicación es el análisis de proyectos. Para hacer este análisis, existen dos posibilidades que el cliente puede elegir dentro de la página correspondiente: hacer el análisis de un solo proyecto a partir de la URL del proyecto alojado en GitHub o hacer un análisis masivo de proyectos desde un fichero de texto que contenga las URLs.

Cuando queremos analizar un solo fichero a partir de su URL, la aplicación se comporta según el diagrama de la figura 4.3.

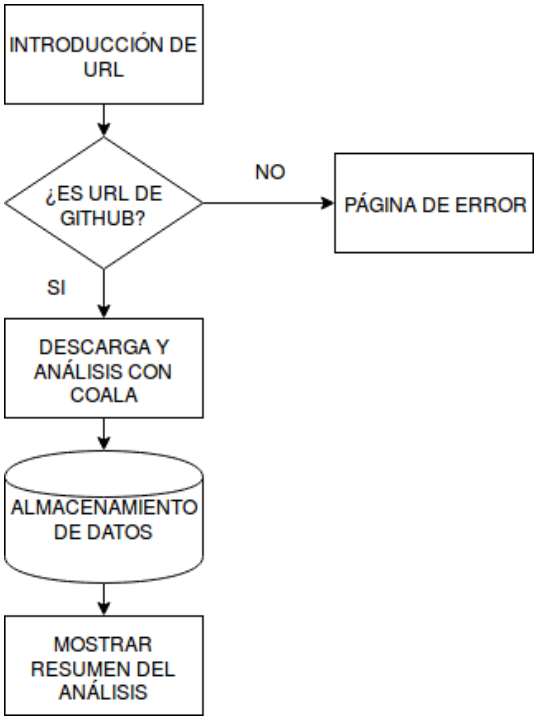


Figura 4.3: Diagrama de flujo en el análisis de proyectos por URL.

Cuando el cliente introduce una URL en el campo de texto, la aplicación comprueba si se corresponde con una dirección de GitHub. Si la dirección no es correcta, se mostrará una página de error; en caso contrario se clonará el proyecto con git en un directorio local. Si el proyecto ya existe en la base de datos, se sobrescribirá y se realizará el análisis sobre la última versión del proyecto.

Una vez descargado el proyecto, se analiza con Coala. Como solo vamos a analizar los ficheros Python, hay que ejecutar Coala sobre los ficheros con extensión '.py'. Además queremos que los resultados se muestren con la estructura de un JSON y redirigirlo a un fichero para su posterior análisis. Teniendo en cuenta todo esto, se ejecuta Coala del siguiente modo, utilizando

un *pipe* de Python:

```
p1 = Popen(["coala", "--json", "--files", python_files], stdout=PIPE)
p2 = Popen(["tee", settings.CONSTANTS['jsonFile']], stdin=p1.stdout, stdout=PIPE)
p1.stdout.close()
output = p2.communicate()[0]
```

Donde 'python_files' son los ficheros con extensión '.py' y 'settings.CONSTANTS['jsonFile']' es la ruta definida en el fichero 'settings' de Django para almacenar el fichero JSON con los datos del análisis. En este punto se elimina el proyecto del almacenamiento local, se lee el fichero JSON, se extraen los datos siguiendo el mapeo definido y se almacena todo en la base de datos[1]. Finalmente, se monta el código HTML sobre la plantilla y se envía para mostrar la información.

A continuación se muestra el aspecto del fichero JSON generado por Coala y desde donde se extraen los datos que se almacenan en la base de datos:


```
{
  "results": {
    "default": [
      {
        "additional_info": "",
        "affected_code": [
          {
            "end": {
              "column": null,
              "file": "/tmp/awesome-python/sort.py",
              "line": 14
            },
            "file": "/tmp/awesome-python/sort.py",
            "start": {
              "column": null,
              "file": "/tmp/awesome-python/sort.py",
              "line": 14
            }
          }
        ],
        "aspect": "NoneType",
```


[1]En la base de datos sólo se almacenan los ficheros analizados que contengan algún bear.

```
        "confidence": 100,
        "debug_msg": "",
        "diffs": {
          "/tmp/awesome-python/sort.py": "--- \n+++ \n@@ -12,6 +12,7 @@\n
            This could be extended by having nested blocks, sorting them
            recursively\n and flattening the end structure into a list of
            lines. Revision 2 maybe ^.^.\n \"\"\"\n+\n \n def sort_blocks
            ():\n     # First, we load the current README
            into memory\n"
        },
        "id": 189705908116436986214728821576105144 17,
        "message": "The code does not comply to PEP8.",
        "message_arguments": {},
        "message_base": "The code does not comply to PEP8.",
        "origin": "PEP8Bear",
        "severity": 1
      }
    ]
  }
}
```


Cuando queremos analizar una lista de proyectos a partir de un fichero de texto, la aplicación se comporta según el diagrama de la figura 4.4.

El fichero únicamente debe contener las URLs de GitHub correspondientes a los proyectos que se desee analizar y cada URL debe ir en una línea distinta.

Cuando el cliente introduce la ruta del fichero que contiene las URLs, el servidor comprueba que la ruta es válida y en caso contrario muestra una página de error. Una vez comprobado, el servidor lee el fichero línea a línea y sigue el mismo proceso que en el apartado anterior, que recordemos es: comprobar que la URL pertenece a un proyecto de GitHub, descarga del proyecto, análisis y almacenamiento de datos. Si alguno de los proyectos no puede ser analizado, la aplicación pasa a la siguiente línea.

Una vez terminado el análisis, se cargan los datos en el template correspondiente y se envía el código HTML al cliente para mostrar un resumen de todos los proyectos analizados. Para mostrar este resumen, cada vez que analizamos un proyecto se incluye en una lista y al finalizar se muestran los datos almacenados para los proyectos de esa lista.

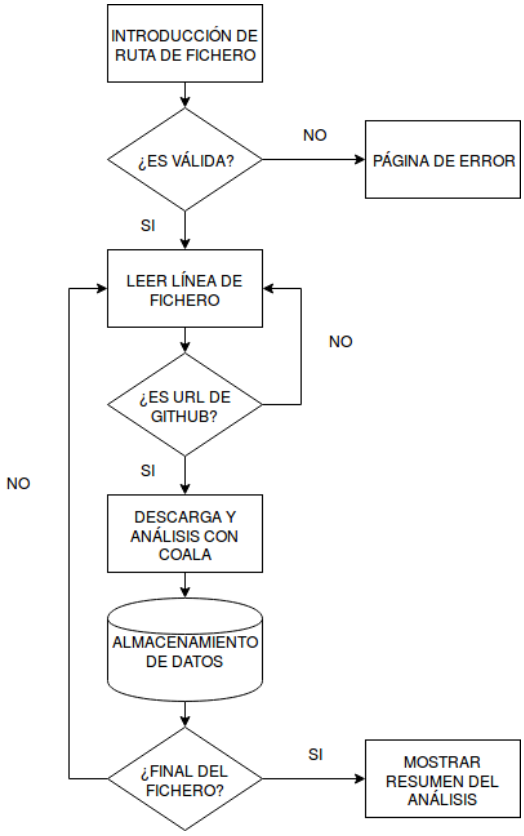


Figura 4.4: Diagrama de flujo en el análisis de proyectos por fichero.

4.2.2. Búsqueda en la base de datos

La aplicación permite hacer una búsqueda en la base de datos. Dentro de la sección correspondiente, el cliente debe elegir la categoría en la que quiere buscar: proyecto, fichero o bear. Una vez relleno el campo de texto y seleccionada la categoría, se lanza la búsqueda en el servidor.

El servidor recibe la consulta y la opción de búsqueda deseada y muestra todos los registros de la base de datos que encajan con la búsqueda junto con un pequeño resumen y links hacia la información al completo. En caso de no encontrar nada, mostrará un mensaje informando de la situación.

Hay que tener en cuenta que dependiendo de los criterios de búsqueda la operación tardará más o menos tiempo. La cantidad de bears almacenados suele ser muchísimo mayor que el número de ficheros y proyectos por lo que puede llevar bastante tiempo en comparación con otras búsquedas.

4.2.3. Resumen de proyectos, ficheros y bears almacenados

Otra de las funcionalidades de la aplicación es mostrar un resumen de todos los datos almacenados en la base de datos.

Resumen de proyectos almacenados

Cuando el cliente envía una petición para mostrar los datos de proyectos, el servidor recorre la lista de ficheros almacenados y cuenta todos los pertenecientes a ese proyecto. Repite la operación con los bears almacenados para contar la cantidad de bears encontrados. Finalmente, monta una tabla con estos datos, el enlace a la página que contiene todos los detalles del proyecto y el porcentaje de ficheros afectados respecto al número total de ficheros del proyecto antes de enviar el código HTML al cliente.

Resumen de ficheros almacenados

Cuando la petición del cliente es para mostrar los datos de los ficheros afectados por algún bear, el servidor recorre la lista de bears almacenados y cuenta todos aquellos que pertenecen a cada fichero, mostrando los bears totales por fichero. Finalmente monta una tabla de resumen con estos datos, el nombre el proyecto y enlaces a las páginas de detalle del fichero y proyecto al que pertenecen, y envía el código HTML al cliente.

Resumen de bears almacenados

Por último, cuando la consulta es acerca de los bears almacenados, el servidor recorre la lista de bears y anota el tipo de bear de cada uno. Al terminar, muestra todos los tipos de bear almacenados en la base de datos y la cantidad de cada uno de ellos, y envía el código HTML al cliente.

En esta lista pueden aparecer repetidos algunos bears con un código al final, esto ocurre porque algunos bears se corresponden con herramientas que analizan diferentes aspectos del código. Por tanto, aunque el bear sea el mismo, el código hace referencia al aspecto concreto que se ha analizado en cada caso.

4.2.4. Detalles de proyectos, ficheros y bears almacenados

El cliente puede hacer una búsqueda para encontrar la información detallada de un proyecto, fichero o bear concreto, o acceder a los datos a través de alguno de los enlaces que aparecen en las diferentes páginas.

Detalle de proyecto

El cliente envía una petición al servidor cuando accede a la página de un proyecto a través de una URL del siguiente tipo:

```
http://localhost:8000/analyzer/project/CREADOR_PROYECTO/NOMBRE_PROYECTO
```

Al acceder a la página de información de un proyecto, el servidor busca el proyecto en la base de datos y en caso de que no exista la respuesta del servidor se mostrará en una página de error. Si el proyecto existe, muestra todos los campos almacenados, el número total de ficheros afectados y el porcentaje de ficheros afectados respecto al total de ficheros Python del proyecto. Además, recorre la lista de los ficheros relacionados con este proyecto y muestra un resumen del tipo y cantidad de bears encontrados para cada uno.

La información de los ficheros analizados está oculta por defecto y puede mostrarse si el usuario lo desea gracias al código JavaScript que contiene la página. Esto se hace para no amontonar la información en pantalla, de forma que el usuario vea un resumen de los datos y se muestre el resto si así lo desea.

Detalle de fichero

El cliente envía una petición al servidor cuando accede a la página de un fichero a través de una URL del siguiente tipo:

```
http://localhost:8000/analyzer/file/CREADOR_PROY/NOMBRE_PROY/FICHERO
```

Al acceder a la página de información de un proyecto, comprueba que existan en la base de datos tanto el proyecto como el fichero en cuestión. Si no existen se mostrará una página de error y en caso contrario se saca toda la información del fichero (nombre, proyecto al que pertenece, URL en GitHub y path local del fichero). Además, se recorren los bears asociados

a este fichero y se extrae parte de la información como el nombre del bear, la línea del fichero donde se encontró, el mensaje del bear y su Id.

Como en el caso anterior, la información de bears encontrados aparece oculta por defecto y puede desplegarse gracias al código JavaScript insertado en el código HTML.

Detalle de bear

El cliente envía una petición al servidor cuando accede a la página de un bear a través de una URL del siguiente tipo:

```
http://localhost:8000/analyzer/bear/ID_BEAR
```

Cada ID es único, por tanto se comprueba que el bear correspondiente a esa Id se encuentra almacenado en la base de datos y en caso contrario se muestra una página de error. Si el bear existe en la base de datos, se extrae toda la información[2] y se monta el código HTML para en viarlo al cliente. Cada bear se corresponde con un proyecto y fichero concreto.

4.2.5. Eliminación de proyectos almacenados

Esta funcionalidad no aparece en la barra de navegación del cliente ya que está pensada para que el administrador pueda eliminar proyectos de la base de datos. Es posible que ocurra un error durante el análisis que deje datos incompletos del proyecto y por tanto es útil contar con la función de borrado a través de una URL construida de la siguiente forma:

```
http://localhost:8000/analyzer/remove/CREADOR_PROYECTO/NOMBRE_PROYECTO
```

4.3. Diseño e implementación del cliente

Las peticiones al servidor se realizan a través de la parte del cliente, además el cliente se encarga de mostrar al usuario los resultados de esas consultas después de interpretar el código HTML recibido.

Gracias a Django, es posible hacer una separación entre la presentación final de la aplicación y la lógica de la misma. Para ello, he definido una serie de templates que servirán de base para

[2] Aspect, MessageArguments y MessageBase no se muestran por no aportar información extra al usuario.

el código HTML que se va a mostrar en cada caso. En el fichero 'views.py' añadiremos la información en contexto dentro del template una vez procesada la petición del cliente para mostrar la respuesta.

Para la creación de estos templates, me he ayudado de BootStrap, que permite crear sitios web atractivos a la vista y con algunas funcionalidades realmente útiles. Gracias a BootStrap, el diseño de la aplicación se adapta al tamaño del dispositivo permitiendo una correcta visualización en cada caso, es decir, es un diseño 'responsive'. Además, todas las páginas diseñadas cuentan con una barra de navegación superior que facilita el acceso a las diferentes funcionalidades que tiene la aplicación.

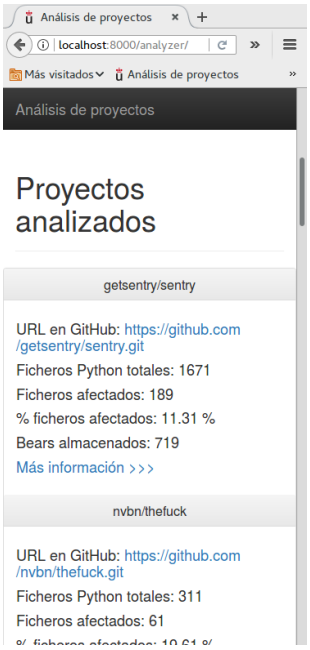


Figura 4.5: Página principal adaptada a una ventana pequeña.

La página principal muestra una breve explicación el proyecto y acto seguido una lista con todos los proyectos analizados ordenados en orden ascendente por el porcentaje de ficheros afectados por algún bear. Aunque la cantidad de bears encontrados depende en gran medida del tamaño del proyecto y los ficheros analizados, con un primer vistazo ya podemos ver un resumen de lo que nos encontraremos.

El usuario puede pinchar en 'Más información' para acceder a la página de un proyecto dentro de la aplicación, o en el enlace de GitHub para inspeccionar el proyecto en la plataforma.

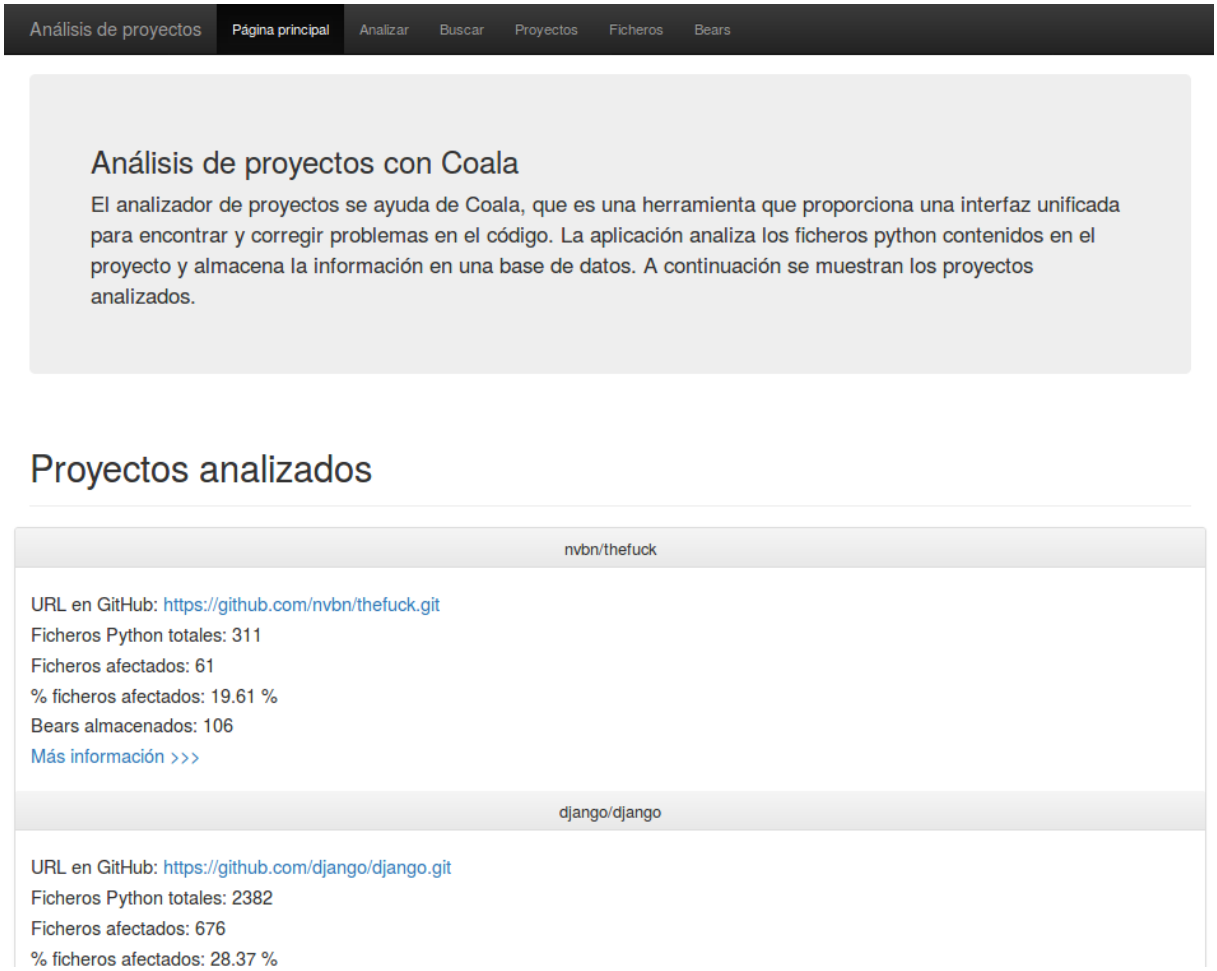


Figura 4.6: Página principal de la aplicación web.

Si el usuario quiere analizar un proyecto, o varios, al entrar en el apartado de análisis encontrará las dos posibles opciones: analizar a partir de una URL o analizar a partir de un fichero de texto. Al pulsar en la opción deseada, se mostrará la página correspondiente en la que el usuario debe introducir la URL o la ruta absoluta del fichero que contiene las URLs que se van a analizar.

Al concluir el análisis, en ambos casos se muestra un resumen de los proyectos analizados. En este resumen se incluye la información del proyecto y un resumen de ficheros y bears encontrados que podemos mostrar u ocultar pinchando en 'Mostrar/ocultar ficheros analizados'. Si queremos ver los bears de cada tipo, al pinchar en el enlace haremos una búsqueda en la base de datos de ese tipo de bear o también podemos acceder a la página de información para el proyecto o alguno de los ficheros.

Figura 4.7: Páginas de análisis. Arriba, la página principal; en el medio, la página de análisis a partir de un fichero; y abajo, la página de análisis a partir de una URL.

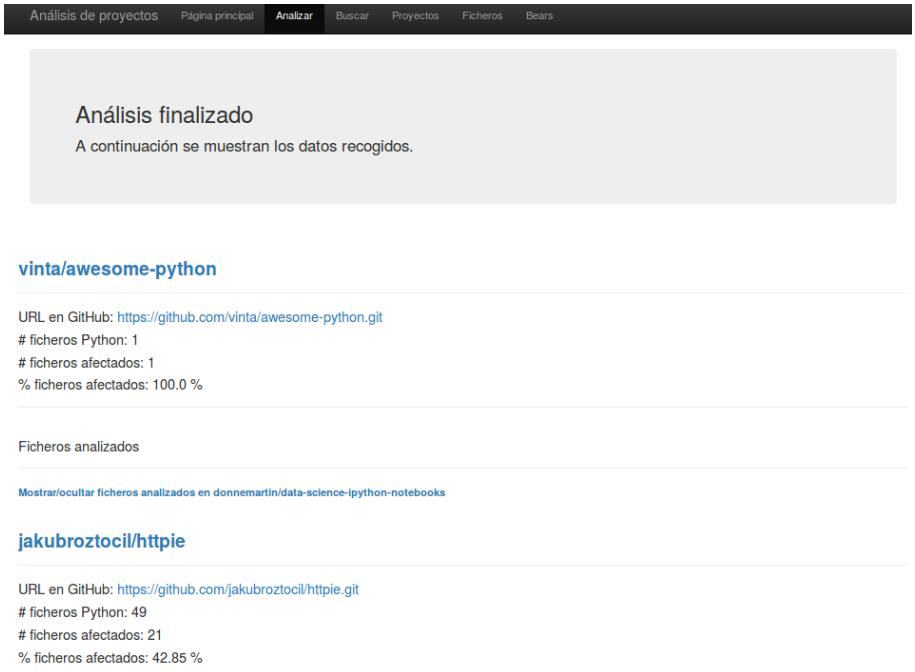


Figura 4.8: Resumen del análisis realizado.

La siguiente funcionalidad es la de buscar en la base de datos. Al acceder a esta página, nos muestra un cuadro de texto y las opciones entre las que podemos elegir para iniciar la búsqueda.

Figura 4.9: Página de búsqueda.

Una vez terminada, se muestra un listado con todos los datos encontrados. Si por ejemplo queremos buscar el fichero 'sort.py', nos mostrará todos los ficheros con ese nombre almacenados, en este caso solo hay uno. En caso de que el fichero no se encuentre en la base de datos, mostrará un mensaje de error.

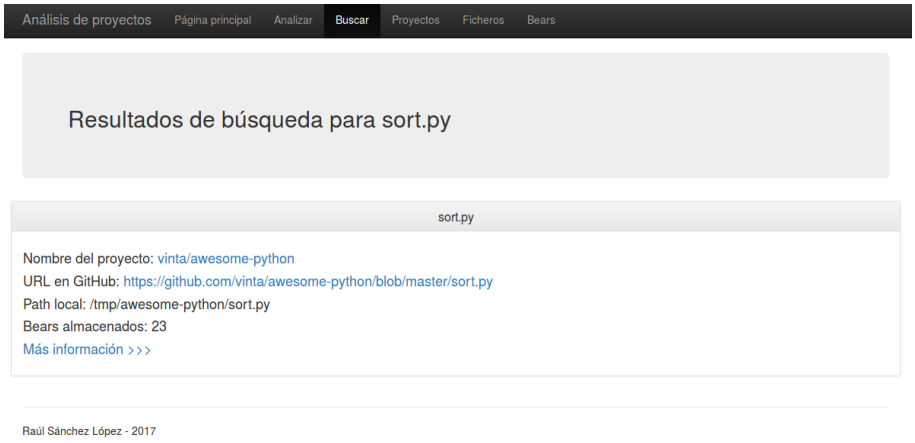


Figura 4.10: Ejemplo de búsqueda por fichero.

Si queremos ver un resumen de todos los proyecto, ficheros y bears almacenados, habrá que dirigirse al apartado correspondiente de la web. Los proyectos están ordenados de mayor a menor porcentaje de ficheros afectados por algún bear, los ficheros aparecen ordenados por cantidad de bears encontrados, sin tener en cuenta de que tipo son, y en el caso de los bears, aparecen ordenados los bears encontrados por tipo.

Cantidad de bears por proyecto

#	Bears	% ficheros afectados	Ficheros afectados	Ficheros Python	Proyecto
1	5	100.0	1	1	vinta/awesome-python
2	3	100.0	3	3	StevenBlack/hosts
3	10099	96.01	193	201	clips/pattern
4	21117	94.73	162	171	powerline/powerline
5	6158	94.33	300	318	nltk/nltk
6	1002	90.32	28	31	locustio/locust
7	4002	88.61	358	404	sqlmapproject/sqlmap
8	753	82.84	140	169	getredash/redash
9	1793	82.32	177	215	spotify/luigi
10	202	82.14	23	28	codelucas/newspaper
11	5543	81.52	278	341	ipython/ipython
12	154	81.25	13	16	wifiphisher/wifiphisher
13	4104	77.66	560	721	bokeh/bokeh
14	1035	77.54	259	334	pytorch/pytorch
15	527	77.22	78	101	nicolargo/glances

Figura 4.11: Página de resumen de proyectos almacenados ordenados por porcentaje de ficheros afectados.

Cantidad de bears por fichero

#	Bears	Fichero	Proyecto
1	1561	tests/test_python/test_segments.py	powerline/powerline
2	815	mitmproxy/contrib/wbxml/ASWBXML.py	mitmproxy/mitmproxy
3	807	powerline/__init__.py	powerline/powerline
4	786	tests/test_python/test_configuration.py	powerline/powerline
5	741	powerline/lint/checks.py	powerline/powerline
6	732	pattern/text/en/wordnet/pywordnet/wordnet.py	clips/pattern
7	699	faker/providers/address/nl_BE/__init__.py	joke2k/faker
8	690	tests/modules/vim.py	powerline/powerline
9	662	pattern/vector/__init__.py	clips/pattern
10	660	thirdparty/pydes/pyDes.py	sqlmapproject/sqlmap
11	643	powerline/lint/spec.py	powerline/powerline
12	636	tests/test_python/test_lib.py	powerline/powerline
13	573	powerline/segments/vim/__init__.py	powerline/powerline
14	512	doc/tutorial/machine_learning_map/pyparsing.py	scikit-learn/scikit-learn

Figura 4.12: Información mostrada en la página de resumen de ficheros almacenados ordenados por cantidad de bears encontrados.

Cantidad de bears por tipo

#	Cantidad	Bear
1	256318	MypyBear
2	46859	PyDocStyleBear
3	29297	PyLintBear (C0103)
4	26962	PycodestyleBear (E501)
5	26345	PyLintBear (C0111)
6	13832	PEP8Bear
7	11026	PyLintBear (W0614)
8	5818	PyLintBear (E0401)
9	5632	PyLintBear (C0301)
10	3818	PyLintBear (R0903)
11	2764	PyLintBear (W0212)
12	2615	PyImportSortBear
13	2247	B101
14	2163	PyLintBear (C0413)

Figura 4.13: Información mostrada en la página de resumen de tipos bears almacenados.

Finalmente, tenemos las páginas de información de cada proyecto, fichero y bears. En ellas se muestra toda la información almacenada en la base de datos. Cuando analizamos un proyecto, podemos consultar la página del proyecto y a partir de ahí navegar por los diferentes ficheros y consultar los bears que se han encontrado, los mensajes que lanza cada uno de ellos, etc.

La página de proyecto muestra una información parecida a la que nos encontramos en la página de inicio, pero además tiene información sobre los ficheros almacenados para ese proyecto concreto. Esa información se puede mostrar u ocultar pinchando en el enlace correspondiente.

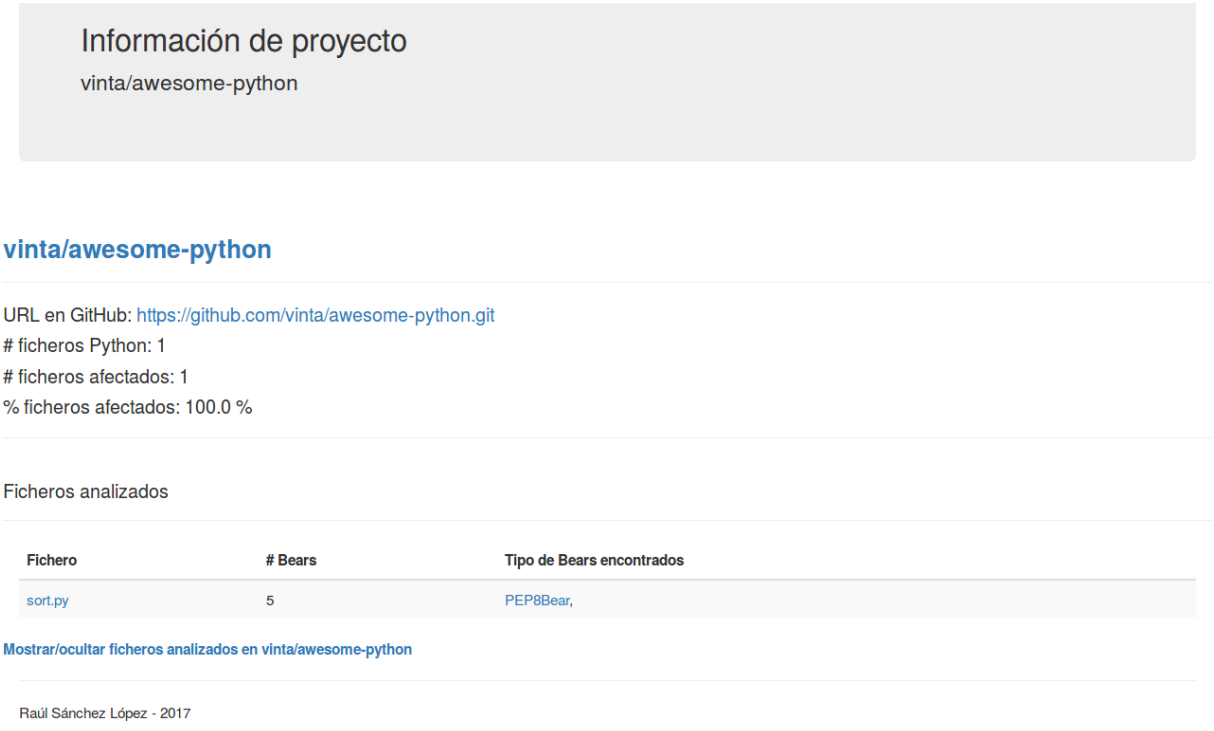
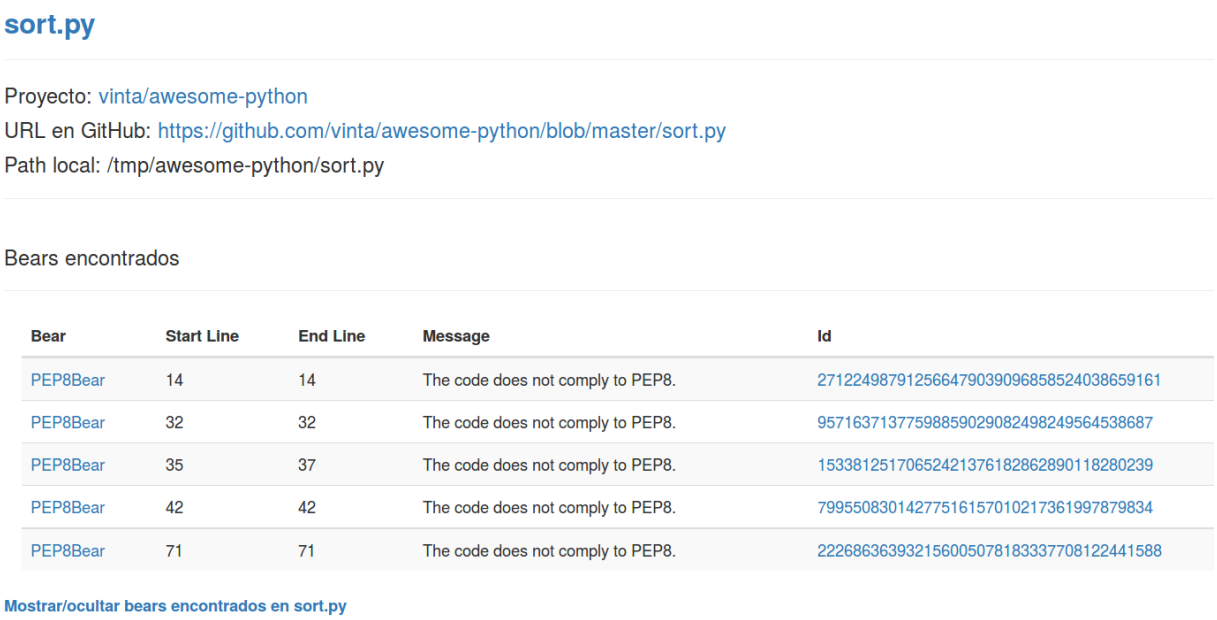


Figura 4.14: Información mostrada en la página de información de proyecto.

La página de información del fichero muestra los datos del fichero, además de una tabla con todos los bears encontrados. Esta tabla tiene los datos básicos, como son el tipo de bear, el mensaje lanzado, las líneas afectadas y el identificador (Id) de cada uno de los bears.

La página de información del bear muestra todos los datos almacenados para cada bear. Esta es la página con mayor nivel de detalle tras hacer el análisis, puesto que en ella podemos ver el problema concreto que ha detectado la herramienta.



sort.py

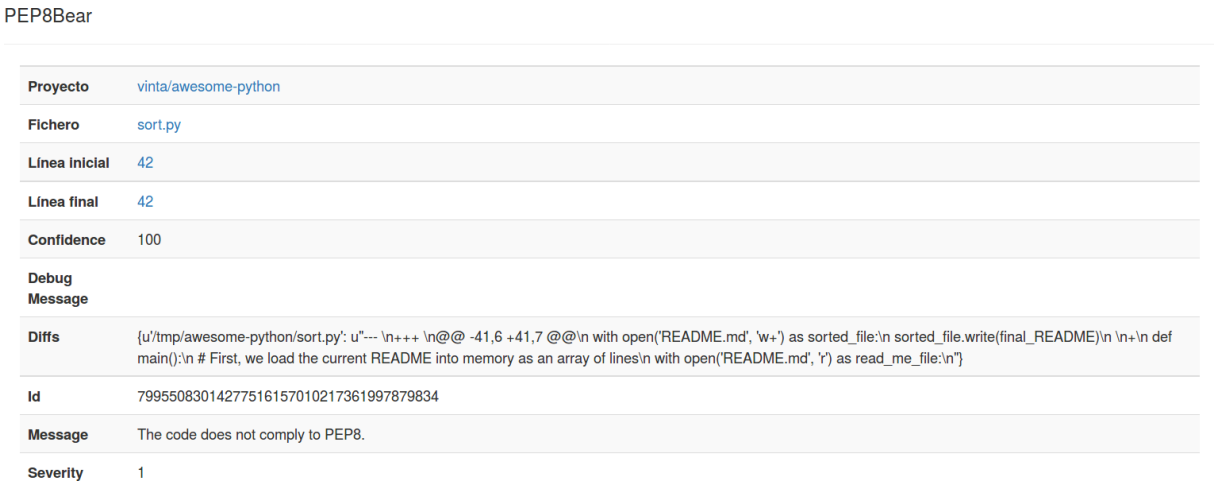
Proyecto: vinta/awesome-python
URL en GitHub: https://github.com/vinta/awesome-python/blob/master/sort.py
Path local: /tmp/awesome-python/sort.py

Bears encontrados

Bear	Start Line	End Line	Message	Id
PEP8Bear	14	14	The code does not comply to PEP8.	271224987912566479039096858524038659161
PEP8Bear	32	32	The code does not comply to PEP8.	957163713775988590290824982495645386 87
PEP8Bear	35	37	The code does not comply to PEP8.	153381251706524213761828628901182802 39
PEP8Bear	42	42	The code does not comply to PEP8.	799550830142775161570102173619978798 34
PEP8Bear	71	71	The code does not comply to PEP8.	222686363932156005078183337708122441588

Mostrar/ocultar bears encontrados en sort.py

Figura 4.15: Información mostrada en la página de información de fichero.



PEP8Bear

Proyecto	vinta/awesome-python
Fichero	sort.py
Línea inicial	42
Línea final	42
Confidence	100
Debug Message	
Diffs	{u'/tmp/awesome-python/sort.py': u"--- \n+++ \n@@ -41,6 +41,7 @@\n with open('README.md', 'w+') as sorted_file:\n sorted_file.write(final_README)\n \n+\n def main():\n # First, we load the current README into memory as an array of lines\n with open('README.md', 'r') as read_me_file:\n"}
Id	799550830142775161570102173619978798 34
Message	The code does not comply to PEP8.
Severity	1

Figura 4.16: Información mostrada en la página de información de bear.

4.4. Diseño e implementación de la base de datos

Es una base de datos relacional, en la que las distintas tablas están relacionadas entre sí por medio de algunos campos, tal y como se observaba en la figura 4.2.

4.4.1. Project

La tabla Project almacena la información de los proyectos. Para identificar un proyecto sólo es necesario tener su nombre y la URL en GitHub, aunque se añade la cantidad total de ficheros para su posterior uso.

- Name: Nombre del proyecto analizado.
- URL: URL donde está alojado en proyecto en GitHub. Es la URL que hay que proporcionar a la aplicación para analizar proyectos.
- TotalFiles: Cantidad total de ficheros Python que forman el proyecto.

4.4.2. File

La tabla File almacena la información de los ficheros analizados en cada proyecto.

- Name: Nombre del fichero.
- URL: URL donde se aloja el fichero en GitHub.
- ProjectName: Proyecto al que pertenece el fichero analizado.
- FilePath: Path local del fichero. Podemos definir donde se descargán dentro del fichero 'settings' de Django.

4.4.3. Bear

Para la creación de la tabla Bear que contiene la información de los bears encontrados se han utilizado los datos de salida del análisis de Coala. Esta información aparece ordenada en el fichero JSON que se procesa durante el análisis.

- Bear: Nombre del bear que ha detectado el problema. Algunos bears son herramientas que analizan muchos aspectos diferentes, por lo que este nombre puede ir seguido del código interno de esos bears o directamente puede tener el código como nombre.
- ProjectName: Proyecto al que pertenece la incidencia encontrada por el bear.

- FileName: Fichero en el que se ha encontrado la incidencia.
- StartLine: Línea donde comienza la incidencia detectada por el bear. En caso de que la incidencia encontrada no se refiera a ninguna línea en concreto por tratarse de un problema en el propio fichero, como por ejemplo convenciones de nombrado, será 0. También será 0 cuando la incidencia no se refiera a un fichero en concreto, sino a a un problema general del proyecto analizado.
- StartLineURL: URL de GitHub que lleva a la línea donde comienza la incidencia. Será 'None' en caso de que la línea sea 0 por los supuestos anteriores.
- EndLine: Línea donde termina la incidencia detectada por el bear. También será 0 en los supuestos descritos para 'StartLine'.
- EndLineURL: URL de GitHub que lleva a la línea donde termina la incidencia. También será 'None' en los supuestos descritos para 'StartLine'.
- Aspect: Hoja de la clase 'Aspect' a la que pertenece este resultado. Para los términos de este trabajo se incluyó por formar parte de la salida de Coala, pero en principio no aporta mas información.
- Confidence: Número entre 0 y 100 que describe la probabilidad de que la incidencia sea un problema real.
- DebugMsg: Mensaje que puede ayudar al usuario a averiguar por qué se obtuvo el resultado.
- Diffs: Diccionario con nombres de fichero asociados a la secuencia de código que provocó la incidencia. No todos los bear ofrecen esta información, por lo que en muchas ocasiones puede aparecer vacío.
- BearId: Número de identificación del bear.
- Message: Mensaje que lanza el bear informando del problema encontrado.
- MessageArguments: Argumentos extra del mensaje. Para los términos de este trabajo se incluyó por formar parte de la salida de Coala, pero en principio no aporta mas información.

- MessageBase: Texto base del mensaje. Para los términos de este trabajo se incluyó por formar parte de la salida de Coala, pero en principio no aporta mas información.
- Severity: Severidad del mensaje.

Capítulo 5

Resultados

Para comprender la utilidad de la aplicación, vamos a analizar una lista de 50 proyectos[1] de entre todos aquéllos que cuentan con más estrellas en GitHub. Para ello, vamos a utilizar un fichero de configuración en el que incluiremos, en lugar de todo el listado disponible para Python, una selección de bears para centrarnos solo en algunos aspectos y evitando aquellos bears que hacen un análisis completo por si mismos. De este modo se pretende clasificar los resultados por los datos concretos que analiza cada uno de estos bears.

5.1. Fichero de configuración

Para la prueba utilizaré el siguiente '.coafile':

```
[Default]
bears = FilenameBear,PEP8Bear,PyCommentedCodeBear,VultureBear,
SpaceConsistencyBear
files = *.py
use_spaces = True
```

'FileNameBear' comprobará que el nombre de los ficheros sigue la convención 'snake', 'PEP8Bear' comprobará que se sigue la guía de estilo descrita en PEP8, 'PyCommentedCodeBear' buscará código fuente comentado, 'VultureBear' buscará clases, funciones y variables que no son utilizadas y finalmente 'SpaceConsistencyBear' comprobará el espaciado. Todos los

[1] De los 52 con más estrellas, dos han sido excluidos debido a su gran tamaño.

bears utilizaran la configuración por defecto y en el caso de ‘SpaceConsistencyBear’ habilitamos la opción que obliga al uso de espacios en lugar de tabulaciones.

5.2. Análisis de datos recogidos

Tras finalizar el análisis, consultamos el resumen de Bears. El tipo de bear que se ha dectectado un mayor número de veces es el ‘PEP8Bear’, de donde se puede concluir que, dentro de este escenario, el principal problema de los proyectos con mejor valoración tiene que ver con el estilo, aunque este bear no especifica exactamente en su mensaje el error encontrado.

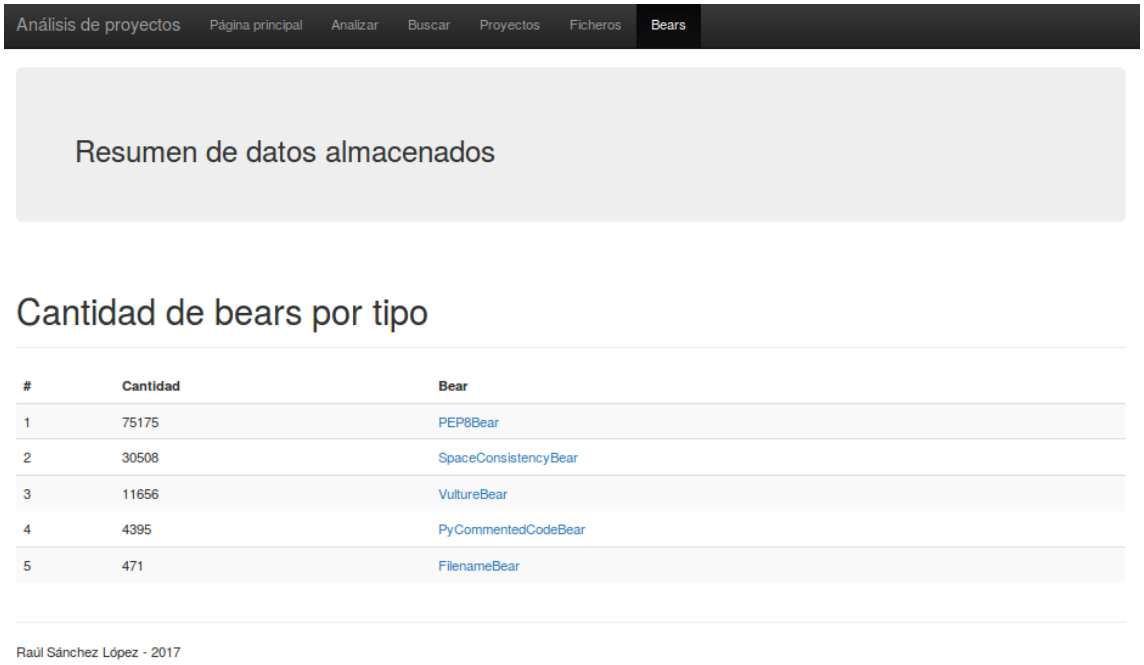


Figura 5.1: Página de resumen de bears recogidos.

El siguiente bear de la lista es ‘SpaceConsistencyBear’, por lo que muchos de estos proyectos cuentan con problemas de espaciado y utilizan tabulaciones en lugar de espacios. El tercer elemento de la lista es ‘VultureBear’, seguido de ‘PyCommentedCode’ y terminando por ‘FileNameBear’ en un número bastante inferior.

De todo lo anterior se deduce que incluso los proyectos mejor valorados tienen problemas de estilo, pero sin embargo es menos frecuente que sus ficheros no sigan la convención ‘snake’.

Ahora vamos a hacer una comparación entre proyectos. Uno de los datos que podemos visualizar con la aplicación es el porcentaje de ficheros que cuentan con, al menos, un bear

detectado, es decir, un problema a priori:

#	Bears totales	% ficheros afectados	Ficheros afectados	Ficheros Python	Bears por fichero afectado	Proyecto
1	5	100.0	1	1	5,00	vinta/awesome-python
2	3	100.0	3	3	1,00	StevenBlack/hosts
3	10099	96.01	193	201	52,33	clips/pattern
4	21117	94.73	162	171	130,35	powerline/powerline
5	6158	94.33	300	318	20,53	nltk/nltk
6	1002	90.32	28	31	35,79	locustio/locust
7	4002	88.61	358	404	11,18	sqlmapproject/sqlmap
8	753	82.84	140	169	5,38	getredash/redash
9	1793	82.32	177	215	10,13	spotify/luigi
10	202	82.14	23	28	8,78	codelucas/newspaper
11	5543	81.52	278	341	19,94	ipython/ipython
12	154	81.25	13	16	11,85	wifiphisher/wifiphisher
13	4104	77.66	560	721	7,33	bokeh/bokeh
14	1035	77.54	259	334	4,00	pytorch/pytorch
15	527	77.22	78	101	6,76	nicolargo/glances
16	356	74.19	46	62	7,74	dbcli/pgcli
17	14342	73.14	1604	2193	8,94	ansible/ansible
18	569	72.95	116	159	4,91	beetbox/beets
19	491	72.81	75	103	6,55	binux/pyspider
20	12629	72.49	1621	2236	7,79	saltstack/salt
21	531	71.34	122	171	4,35	keon/algorithms
22	140	68.96	20	29	7,00	coursera-dl/coursera-dl
23	2528	68.94	262	380	9,65	divio/django-cms
24	3284	68.37	214	313	15,35	reddit/reddit
25	122	68.11	47	69	2,60	faif/python-patterns
26	10	66.66	2	3	5,00	sivel/speedtest-cli
27	398	66.25	53	80	7,51	docker/compose
28	880	62.30	81	130	10,86	donnemartin/gitsome
29	1228	62.07	221	356	5,56	scrapy/scrapy
30	1277	60.94	259	425	4,93	kivy/kivy
31	361	57.14	28	49	12,89	donnemartin/data-science-ipython-notebooks
32	146	57.14	20	35	7,30	eliangcs/http-prompt
33	261	55.88	19	34	13,74	requests/requests
34	12299	55.57	1325	2384	9,28	django/django
35	1156	54.70	128	234	9,03	certbot/certbot
36	2610	50.86	236	464	11,06	mitmproxy/mitmproxy
37	3152	50.26	840	1671	3,75	getsentry/sentry
38	44	50.0	6	12	7,33	wting/autojump
39	489	49.22	127	258	3,85	donnemartin/interactive-coding-challenges
40	1471	47.23	128	271	11,49	joke2k/faker
41	552	46.44	137	295	4,03	explosion/spaCy
42	381	45.26	110	243	3,46	ajenti/ajenti
43	1423	43.89	259	590	5,49	pandas-dev/pandas
44	1249	43.84	299	682	4,18	scikit-learn/scikit-learn
45	53	42.85	21	49	2,52	jakubroztocil/httpie
46	146	40.25	31	77	4,71	pallets/flask
47	32	38.46	10	26	3,20	donnemartin/system-design-primer
48	972	32.74	370	1130	2,63	home-assistant/home-assistant
49	20	27.02	10	37	2,00	getpelican/pelican
50	106	19.61	61	311	1,74	nvbn/thefuck

Figura 5.2: Lista de proyectos analizados.

En la tabla anterior se ha añadido una columna con información que no procede de la aplicación, obtenida de hacer una división entre la cantidad de bears encontrados y la cantidad de

ficheros afectados: ‘Bears por fichero afectado’.

Dentro de este escenario, el proyecto con un menor porcentaje de ficheros Python afectados por algún bear es ‘thefuck’, cuyo creador es ‘nvbn’. ¿Esto hace de ‘thefuck’ el proyecto con menos ‘errores’ del listado? La respuesta es que depende. Es el que menos ficheros tiene afectados por algún bear en términos absolutos, y a su vez uno de los mejores en cantidad de bears por fichero afectado. Además, el fichero que cuenta con más errores tiene 8, relacionados con el estilo, código en desuso y el uso de espacios.

nvbn/thefuck

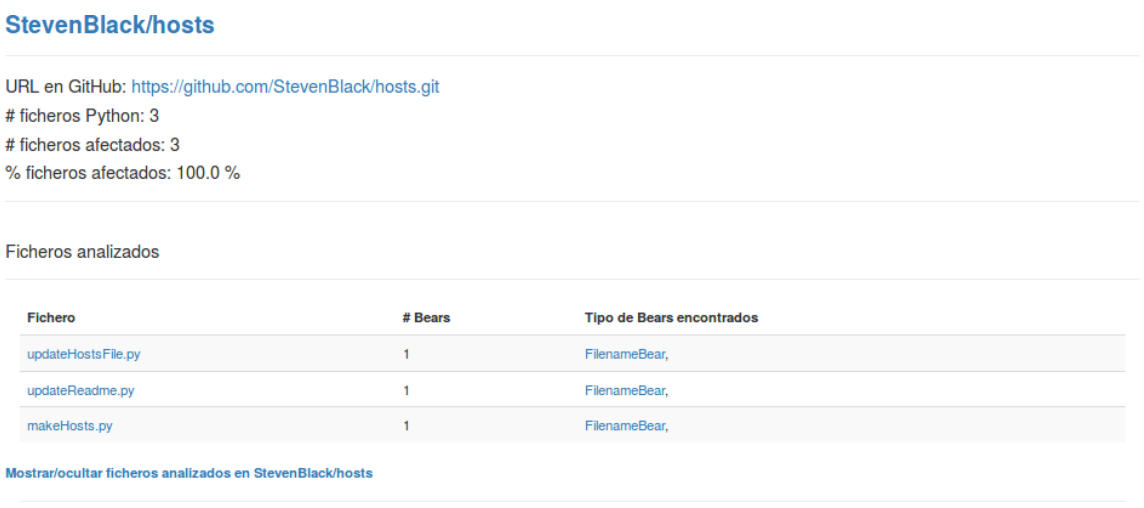
URL en GitHub: https://github.com/nvbn/thefuck.git
ficheros Python: 311
ficheros afectados: 61
% ficheros afectados: 19.61 %

Ficheros analizados

Fichero	# Bears	Tipo de Bears encontrados
tests/rules/test_yarn_command_not_found.py	8	SpaceConsistencyBear, PEP8Bear, VultureBear,
tests/rules/test_yarn_help.py	6	SpaceConsistencyBear,
tests/test_utils.py	6	VultureBear,
tests/test_types.py	5	PEP8Bear, VultureBear,
tests/test_conf.py	5	PEP8Bear, VultureBear,
tests/shells/test_fish.py	4	PEP8Bear, VultureBear,
tests/conftest.py	4	VultureBear,
tests/test_ui.py	3	PEP8Bear, VultureBear,
tests/rules/test_systemctl.py	3	PEP8Bear,
tests/rules/test_vagrant_up.py	3	PEP8Bear,
tests/rules/test_no_such_file.py	2	PEP8Bear,
tests/rules/test_ssh_known_host.py	2	PEP8Bear,
tests/rules/test_mvn_unknown_lifecycle_phase.py	2	SpaceConsistencyBear,
tests/rules/test_mvn_no_command.py	2	SpaceConsistencyBear,
tests/rules/test_mkdir_p.py	2	PEP8Bear,

Figura 5.3: Extracto de lista de ficheros afectados en ‘thefuck’.

Por ejemplo el proyecto ‘hosts’ de ‘StevenBlack’ tiene errores en todos sus ficheros Python, pero son errores sobre la convención de nombrado de los ficheros (‘FilenameBear’), en total es en el que menos bears se han encontrado, pero se trata de un proyecto pequeño en cuanto al número de ficheros totales en comparación con el resto de proyectos de la lista. Además, estos problemas son fácilmente reparables si queremos utilizar la convención de nombrado tipo ‘snake’, ya que basta con renombrarlos.



StevenBlack/hosts

URL en GitHub: https://github.com/StevenBlack/hosts.git
ficheros Python: 3
ficheros afectados: 3
% ficheros afectados: 100.0 %

Ficheros analizados

Fichero	# Bears	Tipo de Bears encontrados
updateHostsFile.py	1	FilenameBear,
updateReadme.py	1	FilenameBear,
makeHosts.py	1	FilenameBear,

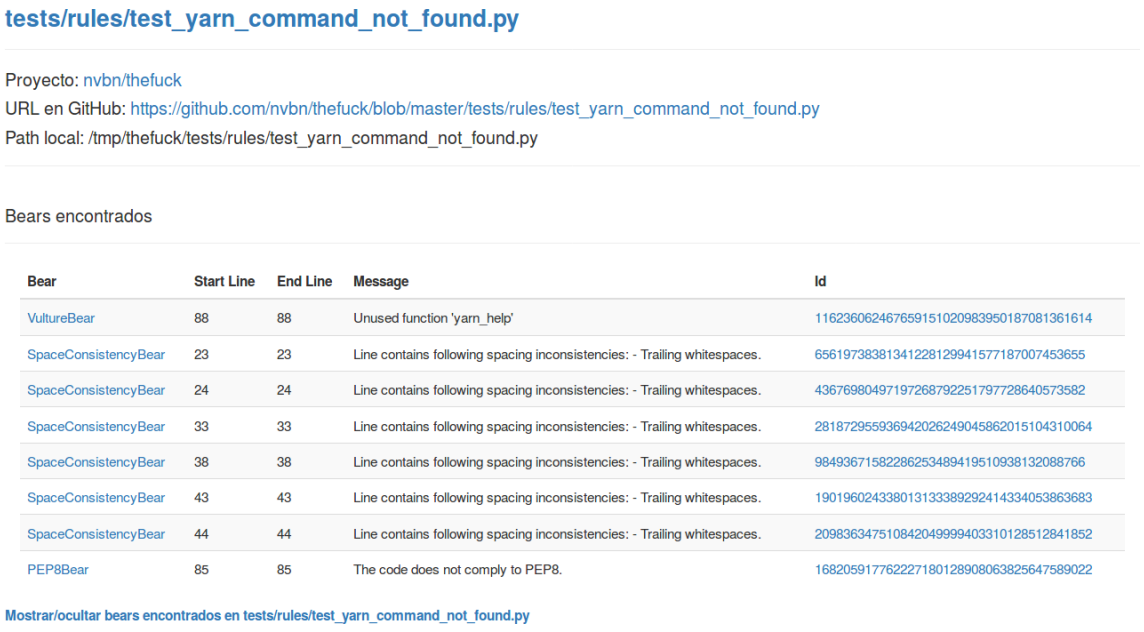
Mostrar/ocultar ficheros analizados en StevenBlack/hosts

Figura 5.4: Información mostrada en la página del proyecto ‘hosts’.

Si tenemos en cuenta el número total de ficheros Python de cada proyecto, sin duda ‘thefuck’ destaca sobre el resto.

Para recabar más información sobre el proyecto ‘thefuck’, habría que inspeccionar los ficheros almacenados. Por ejemplo, el fichero ‘test_yarn_command_not_found.py’, que es el que mayor número de bears acumula en este proyecto.

Como se observa en la figura 5.5, la mayor parte de errores vienen de la inclusión de espacios al final o al inicio de una sentencia.



tests/rules/test_yarn_command_not_found.py

Proyecto: nvbn/thefuck
URL en GitHub: https://github.com/nvbn/thefuck/blob/master/tests/rules/test_yarn_command_not_found.py
Path local: /tmp/thefuck/tests/rules/test_yarn_command_not_found.py

Bears encontrados

Bear	Start Line	End Line	Message	Id
VultureBear	88	88	Unused function 'yarn_help'	116236062467659151020983950187081361614
SpaceConsistencyBear	23	23	Line contains following spacing inconsistencies: - Trailing whitespaces.	65619738381341228129941577187007453655
SpaceConsistencyBear	24	24	Line contains following spacing inconsistencies: - Trailing whitespaces.	436769804971972687922517977286405735 82
SpaceConsistencyBear	33	33	Line contains following spacing inconsistencies: - Trailing whitespaces.	28187295593694202624904586201510431006 4
SpaceConsistencyBear	38	38	Line contains following spacing inconsistencies: - Trailing whitespaces.	98493671582286253489419510938132088766
SpaceConsistencyBear	43	43	Line contains following spacing inconsistencies: - Trailing whitespaces.	19019602433801313338929241433405386368 3
SpaceConsistencyBear	44	44	Line contains following spacing inconsistencies: - Trailing whitespaces.	20983634751084204999403310128512841852
PEP8Bear	85	85	The code does not comply to PEP8.	16820591776222718012890806382564758902 2

Mostrar/ocultar bears encontrados en tests/rules/test_yarn_command_not_found.py

Figura 5.5: Detalle de bears encontrados en un fichero de ‘thefuck’.

Podemos ver en más detalle uno de los bears de la lista a través del enlace. Ahí se muestra toda la información que ha recopilado Coala.

SpaceConsistencyBear

Proyecto	nvbn/thefuck
Fichero	tests/rules/test_yarn_command_not_found.py
Línea inicial	24
Línea final	24
Confidence	100
Debug Message	
Diffs	{u'/tmp/thefuck/tests/rules/test_yarn_command_not_found.py': u"--- \n+++ \n@@ -21,7 +21,7 @@\n --offline trigger an error if any required dependencies are not available in local cache\n --prefer-offline use network only if dependencies are not available in local cache\n --strict-semver \n- --json \n+ --json\n --ignore-scripts don't run lifecycle scripts\n --har save HAR output of network traffic\n --ignore-platform ignore platform checks\n"}
Id	436769804971972687922517977286405735 82
Message	Line contains following spacing inconsistencies: - Trailing whitespaces.
Severity	1

Figura 5.6: Detalle de uno de los bears encontrados por la aplicación.

Para ver exactamente dónde se encuentra el fallo, podemos inspeccionar el fichero a través del enlace a la línea exacta desde la información de bear, en este caso la línea 24. Una vez abierto, comprobaremos que existe al menos un espacio en blanco a la derecha de '–json' que ha detectado el 'SpaceConsistencyBear', ya que con la configuración utilizada para este análisis no permite estos espacios.

```
15
16    Options:
17
18      -h, --help                  output usage information
19      -V, --version               output the version number
20      --verbose                   output verbose messages on internal operations
21      --offline                   trigger an error if any required dependencies are not available in local cache
22      --prefer-offline            use network only if dependencies are not available in local cache
23      --strict-semver
24      --json
25      --ignore-scripts            don't run lifecycle scripts
26      --har                       save HAR output of network traffic
27      --ignore-platform           ignore platform checks
28      --ignore-engines            ignore engines check
29      --ignore-optional           ignore optional dependencies
30      --force                     ignore all caches
31      --no-bin-links              don't generate bin links when setting up packages
32      --flat                      only allow one version of a package
33      --prod, --production [prod]
```

Figura 5.7: Detalle de uno de los bears encontrados en GitHub.

De este modo, la aplicación nos ayuda a detectar errores y corregir el código en caso de ser

necesario.

Uno de los principales ejemplos de uso sería analizar un proyecto propio de GitHub y, a partir de los resultados del análisis, evaluar los errores que ha detectado la aplicación y corregir aquellos que consideremos necesarios para mejorar el funcionamiento del proyecto o ajustarnos a las normas de estilo establecidas.

Capítulo 6

Conclusiones

6.1. Consecución de objetivos

Esta sección es la sección espejo del capítulo de objetivos, donde se revisará si se han alcanzado los objetivos marcados al inicio de este trabajo. Para hacer esta valoración iré de los objetivos más específicos hasta el objetivo general.

Para poder desarrollar la aplicación el primer paso era aprender a utilizar Coala y estudiar cada uno de los bears que podemos utilizar. Para ello estudié el funcionamiento de la aplicación y los bears descritos en el Capítulo 2. Cada uno de estos bears tiene una funcionalidad distinta y consta de diferentes opciones de configuración.

Una vez estudiado el funcionamiento de Coala, era necesario mapear la salida del fichero JSON para poder almacenar la información correctamente. Tras esto, la aplicación era capaz de mostrar todos los campos contenidos en ese fichero sin ningún tipo de problema, tratando los datos de forma que se acoplaran a los resultados obtenidos en cada análisis, como por ejemplo, contemplando casos en los que no existe una línea afectada por tratarse de un bear que se refiere a todo el fichero o la configuración del proyecto en su totalidad.

Antes de poner en marcha la aplicación en Django, había que probar que todo encajaba correctamente. La primera versión del programa servía para testear que las asunciones hechas anteriormente eran las correctas y el análisis arrojaba los resultados esperados. Tras una serie de pruebas, la herramienta era capaz de analizar un listado de proyectos procedentes de un fichero.

El siguiente paso era trasladar aquella lógica a Django, para presentar los resultados en forma de página web y almacenar la información en una base de datos. Django facilita mucho

la tarea de lidiar con bases de datos debido a su simplicidad a la hora de almacenar y consultar la información.

Finalmente quedaba crear todos los 'templates' necesarios para presentar la información al usuario. Justo en este punto se empieza a optimizar el funcionamiento de la aplicación y se añaden nuevas funcionalidades como el resumen de datos, la búsqueda, análisis de ficheros a partir de una URL, eliminación de proyectos almacenados (aunque sea una función que no se presente al usuario por estar pensada para el administrador), ordenar la información o incluir el porcentaje de ficheros afectados en el proyecto.

El objetivo principal de este trabajo era el desarrollo de una aplicación web que sirviera para analizar código fuente Python. Revisando todo lo anterior llegamos a la conclusión de que el objetivo principal se ha cumplido.

6.2. Aplicación de lo aprendido

En la elaboración de este trabajo se han utilizado numerosas tecnologías. Algunas de ellas eran totalmente desconocidas para mí, pero otras formaban parte del temario de asignaturas cursadas a lo largo de la carrera:

1. Conocimiento adquiridos en SAT: es la asignatura en la que aprendí la mayor parte de de los conocimiento aplicados en este proyecto. En ella adquirí conocimientos de Python, Django, HTML, CSS, etc. por lo que está estrechamente relacionada con este proyecto.

2. Arquitectura cliente-servidor: vista en más de una asignatura, llegando a presentar numerosas prácticas con este modelo aunque no solo en Python, también utilizando otros lenguajes como Java.

6.3. Lecciones aprendidas

Para la realización de este proyecto he aprendido a utilizar herramientas y tecnologías que desconocía o que no había usado hasta este momento:

1. Coala: no tenía conocimiento de la existencia de esta herramienta hasta que mi tutor me habló de ello. Es de gran utilidad para el análisis de código gracias a que a partir de una

sola herramienta y gracias a sus numerosos bears podemos hacer análisis de todo tipo.

2. Git/GitHub: es otra de las claves de este proyecto. Me ha servido tanto para encontrar los proyectos que he analizado como para alojar el proyecto en sí mismo. Facilita mucho el control de versiones y la posibilidad de colaborar en proyectos o que otros desarrolladores colaboren en tu proyecto.

3. BootStrap: para mí ha sido un gran descubrimiento. Es de enorme utilidad a la hora de diseñar páginas web. Ha sido de mucha ayuda para mejorar el aspecto de las diferentes vistas de la aplicación.

4. JSON: conocía la tecnología, pero no lo había utilizado tanto hasta la elaboración de este trabajo. Me ha facilitado mucho el tratamiento de los datos procedentes de los análisis.

6.4. Trabajos futuros

Ningún software se termina, todas las aplicaciones necesitan nuevas funcionalidades y requieren de constantes actualizaciones. Es proyecto no se escapa a este aspecto y, como todos, está abierto a mejoras.

Una de las funciones más interesantes sería la elaboración de una clasificación o ranking de 'calidad' de los proyectos a partir de los datos recogidos. Aunque se analizan los ficheros y se da una idea a partir de los datos recogidos, se podría entrar más en detalle clasificando la gravedad de cada bear respecto al resto y no solo la cantidad. Dentro de este ámbito de la 'calidad', se podrían hacer muchas clasificaciones diferentes y tener diferentes rankings dependiendo de factores concretos.

Otra funcionalidad extra podría apoyarse en la capacidad de algunos bears de reparar los problemas detectados, configurando los bears compatibles para solucionar estos problemas y hacer un pull request al creador del proyecto de forma automática.

Apéndice A

Manual de usuario

A.1. Instalación de Coala

Antes de poder utilizar la aplicación es necesario instalar Coala en el equipo. Para ello hay que ejecutar en una terminal el siguiente comando para su instalación simple:

```
$ pip3 install coala-bears
```

Sin embargo, si se va a utilizar un entorno virtual:

```
$ pip3 install virtualenv
$ virtualenv venv        # On Linux
$ venv\scripts\activate  # On Windows
```

A.2. Instalación de Django

Para poder utilizar la aplicación es necesario tener instalado Django. Para ello hay que ejecutar:

```
$ pip install django
```

Para lanzar el servidor hay que abrir una terminal en la ruta en la que se encuentra el proyecto y ejecutar:

```
$ python manage.py runserver
```

Esto lanzará el servidor con la configuración por defecto, que es la utilizada en este proyecto.

A.3. Uso del analizador

Se puede descargar o clonar el proyecto desde su página de GitHub:

```
https://github.com/rsanlo/prAnalysis
```

Antes de ejecutar el servidor, hay que configurar las rutas por defecto en el fichero de Django dentro de '/prAnalysis/settings.py'. Los datos a editar son 'workspace', que es el lugar donde descargará los proyectos para analizarlos y donde buscará el fichero de configuración '.coafile', y 'jsonFile', que es donde almacenará y consultará el fichero JSON.

```
# GLOBAL CONSTANTS
CONSTANTS = {
'workspace': '/tmp',
'jsonFile': '/tmp/coala_analysis.json',
```

Una vez hecho esto, hay que situar el fichero '.coafile' en la ruta definida. Su estructura debe ser como sigue:

```
[Default]
bears = Bear1, Bear2, Bear3,...
opcionA = valorA
opcionB = valorB
.
.
.
```

En 'bears' añadimos los bears que queremos utilizar y debajo las opciones que vayamos a configurar para cada uno de ellos.

El cliente accederá a la página principal a través de la siguiente URL:

```
http://localhost:8000/analyzer
```

Las URLs de GitHub utilizadas para realizar los análisis deben tener la siguiente estructura:

```
http://github.com/CREADOR/PROYECTO.git
```

Bibliografía

[1] Bootstrap.
http://getbootstrap.com/.

[2] Coala.
https://coala.io.

[3] Django project.
https://www.djangoproject.com/.

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

[5] D. M. Beazley. *Python Essential Reference, Fourth Ed.* Pearson Education, Inc., 2009.

[6] J. Bennet. *Practical Django Projects*. Apress, Inc., 2009.

[7] S. Chacon and B. Straub. Pro git.
https://git-scm.com/book/es/v1.

[8] D. Flanagan. *JavaScript: The Definitive Guide, Sixth Ed.* O'Reilly Media, Inc., 2011.

[9] M. Lutz. *Programming Python, Third Ed.* O'Reilly Media, Inc., 2006.

[10] E. Robson and E. Freeman. *Head First HTML and CSS*. O'Reilly Media, Inc., 2012.