UNIVERSIDAD
COMPLUTENSE
MADRID

Desarrollo de Aplicaciones y Servicios Inteligentes

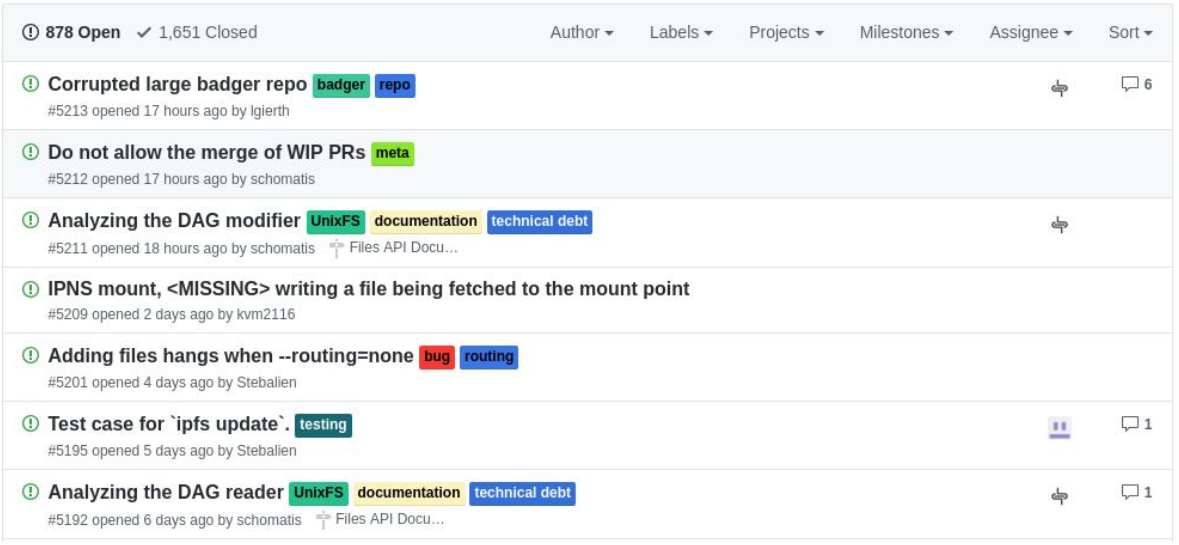
Máster de Ingeniería Informática
de la Universidad Complutense de Madrid

Alumno: David Llop Vila
Profesor: Rubén Fuentes Fernández
Proyecto: **Brown Dispatcher**

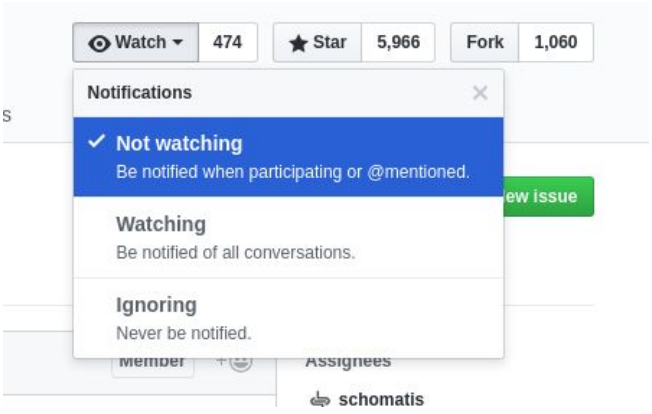
Índice

Descripción del proyecto

Las comunidades de código abierto desarrollan su software gracias a la colaboración de miles de voluntarios alrededor del mundo. Github es una de las plataformas más usadas para colaborar en la creación de código colectiva, y muchos proyectos tienen allí cientos de issues abiertas a la espera de voluntarios que deseen echar una mano.



El problema que tienen muchos desarrolladores a la hora de colaborar es que no existe granularidad en qué notificaciones reciben de un proyecto concreto. Si se suscriben (watch) un proyecto recibirán todas las actualizaciones de todas las issues. Por otro lado, si interactúan con una issue (comentando, por ejemplo), se suscribirán a esa issue concreta, pero no recibirán notificaciones de otras issues que podrían interesarles.



Por lo tanto, los desarrolladores no pueden seguir todos los proyectos que les gustaría, ni saben qué issues se adaptan mejor a sus gustos y capacidades. Es por eso que proponemos un recomendador de issues como proyecto para esta asignatura.

El recomendador inferirá issues que pueden parecer interesantes a un usuario basándose en aquellas en las que ya haya interactuado. El recomendador buscará issues similares basándose en la similitud que hay entre los textos de las issues, sus labels y los autores que han participado en éstas.

Descripción de la aplicación

Introducimos nuestro nombre de usuario y un proyecto en que estamos interesados y el sistema analiza:

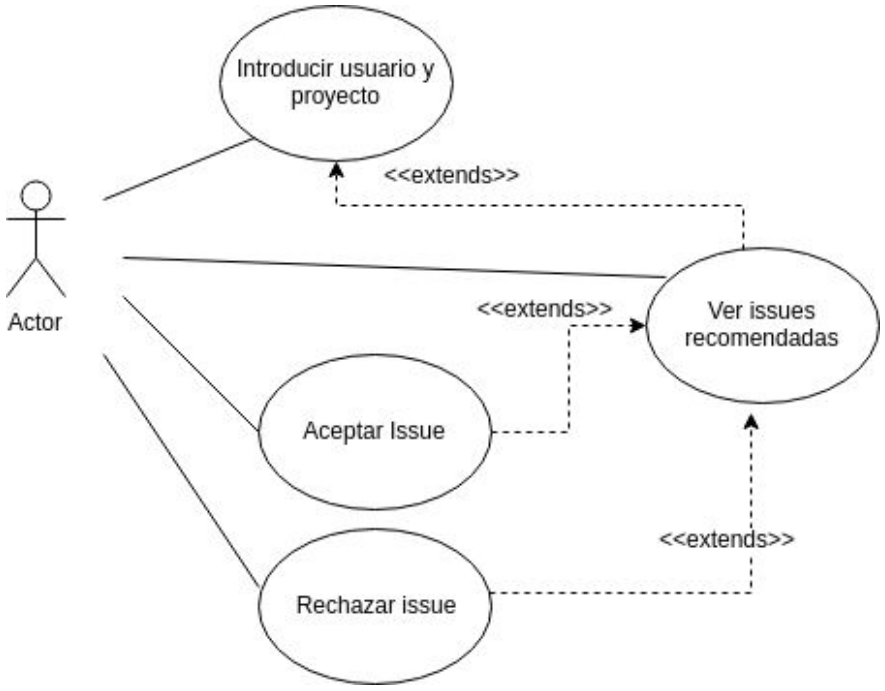
1. La lista de issues en la que hemos hecho alguna aportación.
2. Las issues abiertas en el proyecto en el que estamos interesados.

El sistema infiere una **lista de issues propuestas** a la que nos podría interesar aportar. Por cada una de ellas podemos:

1. Aceptar: La issue se añade en la **lista de issues interesantes** y sirve de refuerzo para el sistema.
2. Rechazar: La issue se elimina de la **lista de issues propuestas** y ya no vuelve a salir recomendada.

Especificación de requisitos

Requisitos funcionales



Caso de uso ID	Introducir usuario y proyecto
Objetivo en contexto	El usuario debe introducir unos parámetros iniciales para que el sistema disponga de información para poderle recomendar issues.

<table>
<tr><td>Entradas</td><td>Nombre de usuario, contraseña y proyecto en el que se quiere participar</td></tr>
<tr><td>Precondiciones</td><td>Ninguna</td></tr>
<tr><td>Salidas</td><td>Dos ficheros: userIssues.csv y projectIssues.csv</td></tr>
<tr><td>Postcoindición si éxito</td><td>Los ficheros estarán guardados en disco</td></tr>
<tr><td>Postcondición si fallo</td><td>Los ficheros no están guardados</td></tr>
<tr><td>Actores</td><td>El usuario</td></tr>
<tr><td>Secuencia normal</td><td><table>
<tr><td>Paso</td><td>Acción</td></tr>
<tr><td>1</td><td>El usuario introduce su nombre de usuario y contraseña y el proyecto en el que quiere participar.</td></tr>
<tr><td>2</td><td>Se usa la API de Github para encontrar las issues en las que el usuario ha estado involucrado (userIssues) y las issues abiertas del proyecto en el cual está interesado (projectIssues).</td></tr>
<tr><td>3</td><td>Se guardan los dos datasets en disco</td></tr>
</table></td></tr>
<tr><td>Secuencias alternativas</td><td><table>
<tr><td>Paso</td><td>Acción</td></tr>
<tr><td>S1</td><td>El usuario introduce mal el nombre de usuario o contraseña, o el nombre del proyecto.</td></tr>
<tr><td>S2</td><td>El sistema para la ejecución y devuelve una excepción</td></tr>
</table></td></tr>
</table>

Caso de uso ID	Ver issues recomendadas
Objetivo en contexto	Mostrar al usuario las issues similares a otras que ha elegido anteriormente de manera que pueda aceptarlas o

<table>
<tr><td></td><td>rechazarlas.</td></tr>
<tr><td>Entradas</td><td>Los datasets de userIssues y projectIssues.</td></tr>
<tr><td>Precondiciones</td><td>Tener disponibles ambos datasets. Ya no es necesario consultar la API de Github.</td></tr>
<tr><td>Salidas</td><td>Top 10 issues de projectIssues similares a las 5 más recientes de userIssues.</td></tr>
<tr><td>Postcoindición si éxito</td><td>Disponemos de 10 issues que mostramos al usuario.</td></tr>
<tr><td>Postcondición si fallo</td><td>Ninguna</td></tr>
<tr><td>Actores</td><td>Usuario</td></tr>
<tr><td>Secuencia normal</td><td><table>
<tr><td>Paso</td><td>Acción</td></tr>
<tr><td>1</td><td>Calcular distancias entre el texto del body de las 5 issues del usuario más recientes y todas las del proyecto.</td></tr>
<tr><td>2</td><td>Hacer lo mismo con las labels.</td></tr>
<tr><td>3</td><td>Hacer lo mismo con los autores de los comentarios de las issues.</td></tr>
<tr><td>4</td><td>Ponderar las distancias, dándole especial importancia a la similitud por label y por autores de comentarios.</td></tr>
<tr><td>5</td><td>Por cada una de las 5 últimas issues de usuario, mostrar 2 issues recomendadas de proyecto (en total 10 issues).</td></tr>
</table></td></tr>
<tr><td>Secuencias alternativas</td><td>Ninguna</td></tr>
</table>

Caso de uso ID	Aceptar issue
Objetivo en contexto	Al aceptar una issue, esta ya no se recomienda más y su información se usa para recomendar nuevas issues similares a esta.

<table>
<tr><td>Entradas</td><td>Los datasets de userIssues y projectIssues.</td></tr>
<tr><td>Precondiciones</td><td>Que ambos datasets estén disponibles.</td></tr>
<tr><td>Salidas</td><td>Ninguna</td></tr>
<tr><td>Postcoindición si éxito</td><td>La issue ha pasado del dataset projectIssues a encabezar el dataset userIssues.</td></tr>
<tr><td>Postcondición si fallo</td><td>Ninguna</td></tr>
<tr><td>Actores</td><td>Usuario</td></tr>
<tr><td>Secuencia normal</td><td><table><tr><td>Paso</td><td>Acción</td></tr><tr><td>1</td><td>El usuario acepta la issue</td></tr><tr><td>2</td><td>El sistema la borra del dataset projectIssues</td></tr><tr><td>3</td><td>El sistema la inserta en el dataset de userIssues</td></tr><tr><td>4</td><td>Se persisten los cambios</td></tr></table></td></tr>
<tr><td>Secuencias alternativas</td><td>Ninguna</td></tr>
</table>

<table>
<tr><td>Caso de uso ID</td><td>Rechazar issue</td></tr>
<tr><td>Objetivo en contexto</td><td>Las issues rechazadas se hacen desaparecer del sistema</td></tr>
<tr><td>Entradas</td><td>El dataset projectIssues</td></tr>
<tr><td>Precondiciones</td><td>Tener disponible el dataset mencionado</td></tr>
<tr><td>Salidas</td><td>Ninguna</td></tr>
<tr><td>Postcoindición si éxito</td><td>La issue desaparece de projectIssues</td></tr>
<tr><td>Postcondición si fallo</td><td>Ninguna</td></tr>
<tr><td>Actores</td><td>Usuario</td></tr>
<tr><td>Secuencia normal</td><td><table><tr><td>Paso</td><td>Acción</td></tr><tr><td>1</td><td>El usuario rechaza la issue</td></tr></table></td></tr>
</table>

	2 | La issue es borrada de projectIssues 3 | Los cambios se persisten
Secuencias alternativas	Ninguna

Modelo de dominio

Utilizamos la API de Github para obtener los siguientes campos de las issues:

- repo: Nombre del repositorio al que pertenece la issue
- title: Título de la issue
- body: Cuerpo de la issue
- labels: Etiquetas asignadas a la issue, muy útiles para categorizarlas
- state: Estado de la issue, puede ser “open” o “closed”
- url: Sitio web de la issue
- created_at: Fecha de creación
- updated_at: Fecha de última actualización
- users: Lista de usuarios que han comentado en la issue, incluyendo el autor

Utilizamos la librería pandas para trabajar con un DataFrame que contiene los datos de las issues en formato de tabla, con las columnas descritas anteriormente.

Librerías empleadas

Pandas
Usamos la librería Pandas para el modelado de las issues. Tratamos la información de sus respectivos comentarios y agregamos los resultados a las issues.

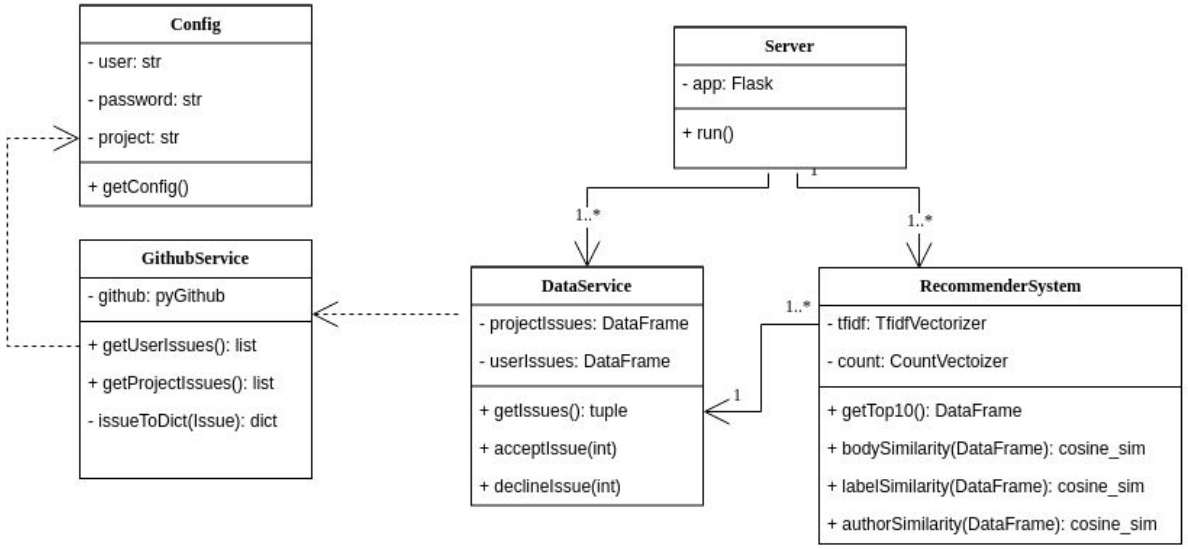
SkLearn
Usamos una matriz TF-IDF para obtener la frecuencia de las palabras usadas en las descripciones de las issues. Luego calculamos la distancia entre matrices usando la similaridad de coseno por las cinco primeras issues del usuario y cada issue candidata. Seleccionamos las issues que obtienen puntuaciones mayores, es decir, se parecen a alguna de las que el usuario ya ha aportado.

Flask
Usamos el framework Flask para desarrollar tal interfaz web que permita al usuario interactuar con el sistema. La interfaz permite visualizar las issues recomendadas y aceptar o rechazar cada una de ellas.

pyGithub
Usamos esta librería para poder acceder más fácilmente a la API de Github.

Diseño del sistema final

Diagrama de clases



La clase principal instancia un **DataService**, un **RecommenderSystem** y un **Server**.

El **Server** hace de vista en el patrón Modelo-Vista-Controlador y muestra al usuario 10 issues recomendadas, con las opciones de aceptar o rechazar por cada una de ellas.

El **RecommenderSystem** hace de controlador y sirve para obtener 10 issues recomendadas basándose en el body, las labels y los autores de las 5 últimas issues con las que el usuario ha interactuado.

Utilizamos una matriz Tf-idf para contar el número de ocurrencias que hay de palabras en el cuerpo de una issue, eliminando las palabras frecuentes del inglés. Luego calculamos la similaridad de coseno para encontrar las issues más parecidas a las que el usuario acaba de interactuar.

El **DataService** hace de modelo en el patrón MVC y obtiene las issues de los ficheros *userIssues.csv* y *projectIssues.csv*. En su defecto, crea esos ficheros cargando los datos de la clase **GithubService**. Las funciones `acceptIssue` y `declineIssue` modifican los ficheros csv para que los cambios en las recomendaciones se mantengan a lo largo del tiempo.

El **GithubService** utiliza la API de github para obtener dos listas de issues, las abiertas en un proyecto concreto y las que se ha visto involucrado un usuario concreto (del que tenemos su contraseña). La lista está compuesta por diccionarios python con los campos vistos en la sección modelo de dominio.

La clase **Config** carga la configuración de usuario y contraseña de github y nombre de proyecto en el que se está interesado desde un fichero llamado `config`.

Diagrama de secuencia

Al hacer una petición al **Server**, éste le pregunta al **RecommenderEngine** por las 10 issues más recomendables para el usuario. Éste, a su vez, consulta a **DataService** por los DataFrames userIssues y projectIssues. Si **DataService** no los tiene disponibles en disco, los tiene que pedir a **GithubService**, que utiliza la API de github para construir esos ficheros.

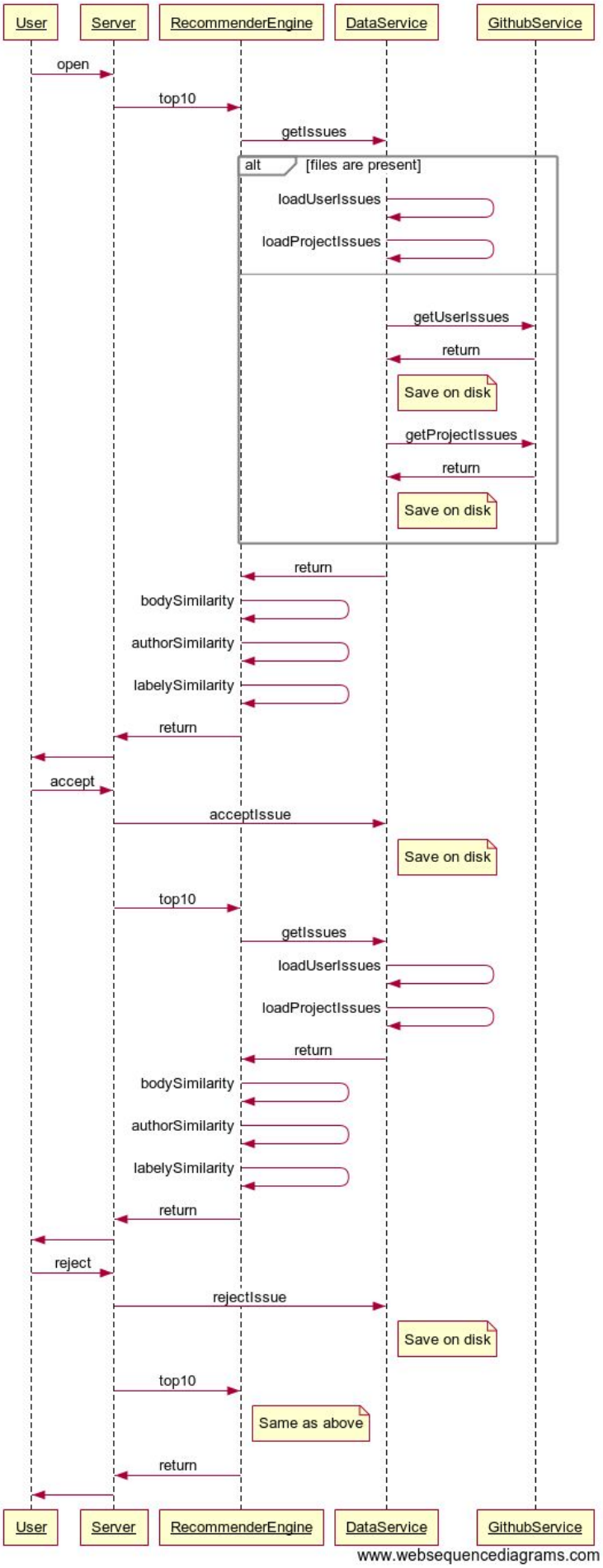
Una vez tenemos los dos DataFrames, procedemos a calcular la similitud por coseno de las matrices TfiIdfVectorizer y CountVectorizer de los cuerpos y las labels y autores de las issues respectivamente para ver cuales son más similares a las últimas 5 issues con las que el usuario ha interactuado. Se ponderan las puntuaciones y se obtinee la lista de 10 issues.

El usuario ve la lista de issues en el navegador, y puede navegar a cada una de ellas, aceptarlas o rechazarlas.

Al aceptarlas, borra la issue de projectIssues por lo que no se le volverá a recomendar más y pasa a ser la última issue con la que el usuairo ha interactuado, es decir, la primera del dataset userIssues, sobre la que se recomienda un nuevo top 10.

Al rechazarlas, secillamente se borra la issue de projectIssues y no se vuelve a recomendar esa issue en concreto.

Open, Accept, and Reject Sequence



www.websequencediagrams.com

Manual de instalación

El sistema requiere de python 3 y de algunas dependencias que se pueden instalar con pip usando el siguiente comando (probado en Ubuntu 18.04):

```
$ sudo pip3 flask pandas sklearn pygithub
```

Si no tiene instalado pip3, puede instalarlo usando el gestor de paquetes así:

```
$ sudo apt install python3-pip
```

Junto al código distribuido se pueden encontrar los ficheros pregenerados userIssues.csv y projectIssues.csv. Corresponden a las issues del usuario sembrestels y del proyecto ipfs/go-ipfs, con los que hemos realizado las pruebas de esta práctica. Para ejecutar el programa sólo es necesario escribir en la terminal:

```
$ python3 BrownDispatcher.py
```

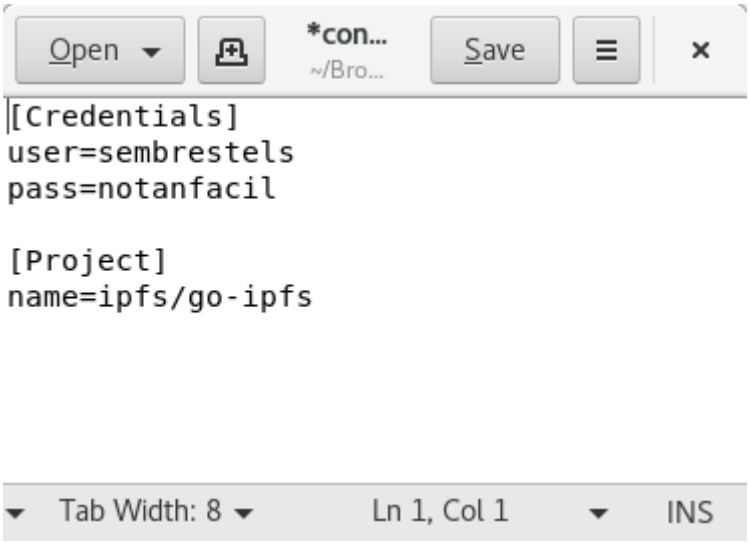
Y veremos las recomendaciones para ese usuario por defecto. Si queremos personalizar las preferencias para otro usuario o otro proyecto, hay generar esos dos ficheros de nuevo. Para hacerlo, es necesario borrar ambos ficheros y crear un fichero de configuración llamado `config` con el usuario y contraseña de github y el proyecto deseado, con este formato:

```
[Credentials]
user=sembrestels
pass=notanfacil

[Project]
name=ipfs/go-ipfs
```

Se puede encontrar un ejemplo en el fichero config-default, en esa misma carpeta. Se puede copiar y modificar:

```
$ cp config-default config
$ gedit config
```

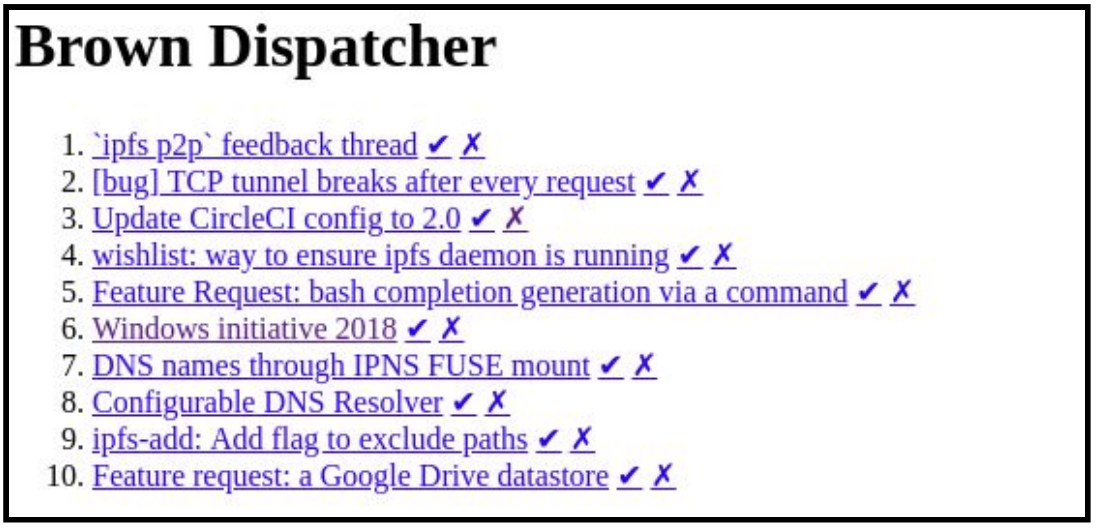


Manual de usuario

Tal como hemos explicado en el manual de instalación, para ejecutar el sistema usamos el comando:

```
$ python3 BrownDispatcher.py
```

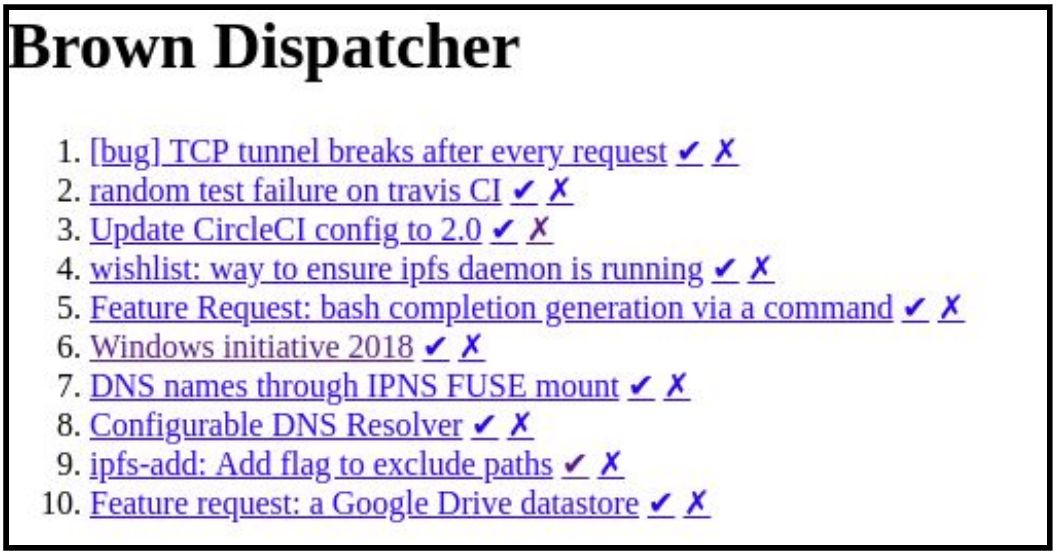
Eso iniciará un servidor en el puerto 5000, permitiendo acceder a la aplicación web desde la URL: http://127.0.0.1:5000/. Si accedemos, encontramos la siguiente lista de issues recomendadas:



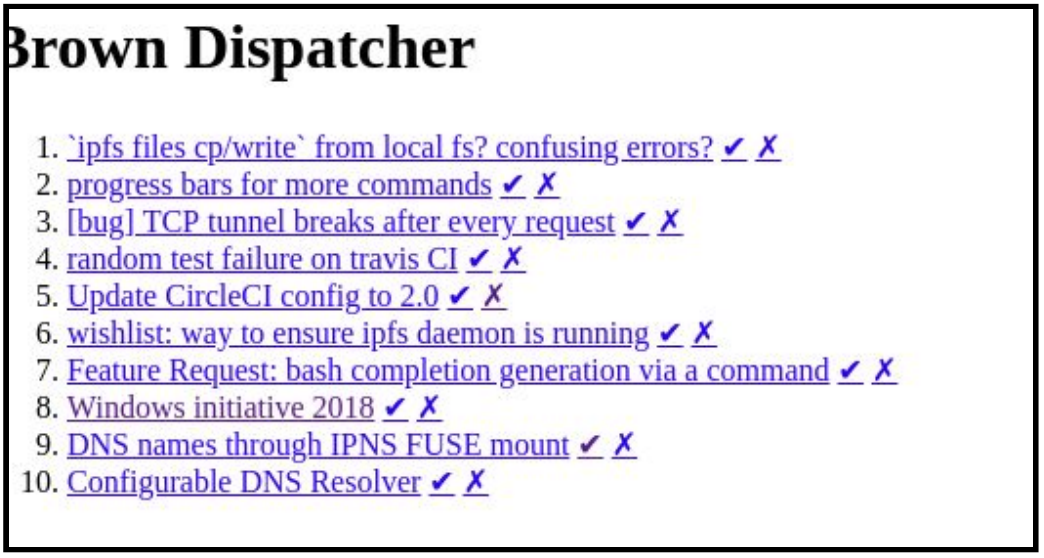
Aparecen 10 issues correspondientes a las 5 últimas issues que el usuario ha interactuado.

Si hacemos click encima del título, se abre una nueva pestaña con la URL de la issue en Github.

Si rechazamos la primera issue, vemos que la segunda pasa a ser la primera y aparece una nueva en segundo lugar.



Si aceptamos la 9 (ipfs-add: Add flag to exclude paths), por ejemplo, vemos que esta ya no aparece más y ahora hay dos nuevas issues en primer lugar que son similares a esta:



Si seguimos dando feedback al sistema, este va recomendando issues similares a las últimas 5 aceptadas, mientras guarda todas las issues aceptadas para no volverlas a mostrar.

También, si paramos el proceso python (Ctrl+C en la terminal) y volvemos a iniciarlo, vemos que los cambios persisten entre sesión y sesión, por lo que el aprendizaje perdura en el tiempo.