

Decide-Ío-Postprocesado

Grupo 2

ID de opera: 108

Miembros del grupo:

Cabello Colmenares, Carlos Manuel: 2

Morales Moreno, Sergio: 5

Rebollo Lobo, Pablo: 5

Sevilla Barceló, Carlos: 5

Enlaces de interés:

[Repositorio del proyecto](#)

[Aplicación en Heroku](#) (Superuser: admin, Pass: iopostproc)

Contenido

1. Resumen.....	3
2. Introducción y contexto	3
3. Descripción del sistema.....	4
4. Planificación del proyecto	5
5. Entorno de desarrollo	5
6. Gestión de incidencias.....	6
6.1. Etiquetas usadas por miembros del equipo de desarrollo y terceros	7
Etiquetas de petición de cambios: Color azul.	7
Etiquetas de feedback: Color morado.....	7
6.2. Etiquetas usadas exclusivamente por el equipo de desarrollo.....	7
Etiquetas para marcar el estado de la incidencia: Color verde.....	7
Etiquetas para marcar incidencias que no serán resueltas: Color rojo.....	8
Etiquetas para indicar el área de procesos/ rol designado que afronta la incidencia: Color amarillo.....	8
7. Gestión de depuración	8
8. Gestión del código fuente	8
9. Gestión de la construcción e integración continua.....	10
10. Gestión de liberaciones, despliegue y entregas.....	10
11. Mapa de herramientas.....	11
12. Ejercicio de propuesta de cambio	11
13. Conclusiones y trabajo futuro	12

1. Resumen

El proyecto ha tratado sobre la introducción de cambios a la ya existente aplicación de Decide, en concreto a uno de sus submódulos, el de postprocesado, y la gestión adecuada de los cambios realizados. Estos cambios que se han realizado se tratan de nuevas funcionalidades añadidas al módulo que permiten ordenar las opciones votadas en una encuesta de diferentes formas, según unos parámetros que se reciben. La solución propuesta han sido diferentes funciones en las vistas del proyecto que serán usadas dependiendo de un parámetro en los datos recibidos que indica el tipo de procesado que se desea. Consideramos que esta solución es adecuada porque respeta el cómo estaba estructurado el código de este módulo antes de realizar estos cambios. Como conclusión, pensamos que los cambios a implementar han sido sencillos y que las dificultades han venido principalmente por la parte de gestión del código y la automatización del despliegue de la aplicación.

2. Introducción y contexto

Decide se trata de una aplicación desarrollada en Django que ofrece la creación de encuestas online y el registro de usuarios para que voten en estas. Se trata de un proyecto educativo, con el fin de ayudar al estudio de sistemas de votación, y que actualmente está bajo una licencia de copyleft GNU AGPL 3.0.

Decide está dividida en varios submódulos, aprovechando el cómo Django se estructura mediante diferentes apps. Cada módulo se encarga de un proceso en concreto de los que requiere esta aplicación para funcionar, e interactúan entre ellos a través de llamadas API a las vistas que ofrecen cada uno de estos módulos. Por supuesto, las funcionalidades de estas APIs pueden ser accedidas de forma externa, enviando peticiones POST por diferentes canales, como puede ser un cliente REST como Restlet.

Los módulos que constituyen esta aplicación son los siguientes:

- Authentication
- Booth
- Census
- Mixnet
- Postproc
- Store
- Visualizer
- Voting

Nuestro grupo ha trabajado en el módulo de Postproc o postprocesado, y el trabajo desarrollado ha consistido en añadir nuevas funciones que permitan procesar los resultados de las votaciones de forma diferente, así como la implementación de los tests necesarios para asegurar el correcto funcionamiento de la aplicación una vez añadidos estos cambios. También se ha elaborado una guía acerca del uso de las funcionalidades que ofrece este módulo y cómo hacer correctamente peticiones a la API ofrecida, a petición del grupo encargado del módulo de cabina, para que estos puedan realizar llamadas a nuestra aplicación standalone que se encuentra desplegada en Heroku.

3. Descripción del sistema

En el módulo sobre el que se hay desarrollado este proyecto, todo el código que realiza las acciones necesarias para el funcionamiento de la aplicación se encuentra en el archivo `views.py`. Este archivo está estructurado de la siguiente forma:

- Diferentes funciones, cada una encargada de implementar un tipo de postprocesado. Estas funciones, a partir de los datos proporcionados en la petición recibida, generarán una respuesta en JSON con los resultados de la votación, en la que se ordenan las opciones votadas según el tipo de postprocesado
- Una función “post” que recibe una petición que puede provenir de dos sitios, bien desde el módulo de votación o desde una llamada externa. Este método mira qué tipo de postprocesado se está pidiendo en esa llamada y devuelve el resultado de la función que se corresponda a ese método.

Inicialmente, sólo existía un tipo de postprocesado, el llamado “IDENTITY”, que simplemente ordenaba las opciones de mayor a menor número de votos obtenidos. Nuestro grupo ha implementado los siguientes tipos de procesado:

- `weightedOptions`: Cada opción tiene un peso asignado. Para ordenarlas, se multiplicará el peso de cada opción por su número de votos.
- `randomSelection`: Se elige una opción de forma aleatoria y esta es la que se devuelve como resultado de la votación. El número de votos que tenga esa votación influirá en el porcentaje de probabilidades que tendrá para salir elegida, cuantos más votos respecto al resto de opciones tenga, mas probable será que sea la opción ganadora.
- `weightedRandomSelection`: Se trata de una combinación de los dos métodos anteriores, se escogerá una opción al azar pero en este caso, las probabilidades se verán afectadas por el número de votos que tenga una opción multiplicado por su peso.
- `borda_count`: Implementamos el algoritmo de recuento borda, un algoritmo para procesos electorales propuesto por Jean-Charles de Borda. Este algoritmo se basa en votaciones múltiples, donde eliges las opciones en orden de interés del votante. El orden de los votos se comporta como el peso, y es el elemento principal de este algoritmo.
- `genderParity`: Se implementa un algoritmo que devuelve ordenadas las opciones de forma que las opciones de votación con género masculino sumen la misma cantidad de votos que en las que el género sea femenino.
- `ageLimit`: Las opciones a devolver deberán pasar un filtro respecto a la edad de los presentados en las votaciones.
- `hondt`: Se aplicará la Ley d’Hondt al resultado de las votaciones

4. Planificación del proyecto

Inicialmente, se establecieron cuales iban a ser las funcionalidades principales que se iban a realizar y quién iba a estar al cargo de cada una, si bien es cierto que a lo largo del desarrollo del proyecto han ido surgiendo ideas para funcionalidades nuevas que han sido autoasignadas de forma independiente por la persona que ha pensado en ellas, siempre previa consulta y aprobación con el resto de integrantes del grupo. Por cada uno de los milestones de la asignatura se realizó una iteración, lo que significa que este proyecto ha tenido un desarrollo que abarca 3 iteraciones. Con el fin de preparar cada milestone, el día previo a cada milestone se organizaba una reunión no presencial en la que se exponía que hitos se habían cumplido y cuales eran los siguientes pasos a seguir.

Los avances realizados en cada iteración fueron los siguientes:

- Iteración 1: Creación del repositorio y elaboración de la metodología a seguir a la hora de realizar commits y crear issues, si bien es cierto que previamente se habían realizado algunos commits que por tanto no cumplen exactamente la metodología posteriormente acordada. Implementación de Travis para la gestión de la construcción y de la integración continua. Al final de esta iteración, Travis ya se encargaba de lanzar los tests creados cada vez que se actualizaba el repositorio. Implementación del postprocesado “weightedOptions” junto sus tests.
- Iteración 2: Despliegue de la aplicación en Heroku, implementación del método “randomSelection” junto a sus correspondientes tests.
- Iteración 3: Documentación acerca del uso de la API que nuestro módulo ofrece, implementación de los métodos “weightedRandomSelection”, “borda_count”, “genderParity”, “ageLimit” y “hondt” junto a sus correspondientes tests. Automatización del despliegue en Heroku.

5. Entorno de desarrollo

Todos los integrantes del equipo han usado en el desarrollo del proyecto el mismo software con el mismo tipo de versiones, a excepción del IDE, que se decidió dejar a preferencia personal de cada miembro. Los IDEs que han sido usados por los miembros del grupo han sido PyCharm y Visual Studio Code.

Alguno de los elementos de software más relevantes junto a las versiones usadas han sido los siguientes:

- Python 3.7
- PostgreSQL 10
- Django 2.0
- DjangoRestFramework 3.7.7
- Heroku 7.19
- Travis CI Command line client 1.8.9

Para instalar el software necesario para trabajar en el entorno del desarrollo, desde la consola de comandos accedemos al directorio del proyecto en el que se encuentra el archivo de texto “requirements.txt” y se lanzará la instrucción “pip install -r requirements.txt”

Será también necesario configurar la base de datos en PostgreSQL, para ello, accedemos a la SQL Shell que Postgre ofrece y lanzamos los siguientes comandos:

- "create user decide with password 'decide'"
- "create database decide owner decide"

El siguiente paso será configurar un archivo "local_settings.py" para el despliegue del proyecto en local. Para esto, se cuenta con un archivo llamado "local_settings.example.py" que sirve de plantilla, y en el que habrá que realizar los siguientes cambios para generar el archivo que deseamos:

- Cambiar todas las urls en API y BASEURL a direcciones de la máquina local "localhost" o "127.0.0.1", y asignarle un puerto. En nuestro caso, le hemos asignado el puerto 8000, por tanto, la dirección por la que se sustituirán todas las urls que aparecen será "http://localhost:8000", pudiendo sustituirse ese 8000 por otro número de puerto si fuera necesario.
- En DATABASES, indicar que el USER y PASSWORD de la base de datos PostgreSQL es "decide". Además, el campo HOST deberá tener asignada la dirección local 127.0.0.1.

Una vez realizado estos cambios, el entorno de desarrollo estará funcional, si queremos desplegar la aplicación, deberemos desde la consola de comandos movernos al directorio en el que se encuentra el archivo manage.py, que generalmente se encontrará en /decide.

Una vez situados en ese directorio, actualizaremos las tablas de la base de datos lanzando el comando "python manage.py migrate" y arrancaremos el servidor con "python manage.py runserver". Si por algún motivo las migraciones no están al día con los modelos de la aplicación, siempre podemos lanzar "python manage.py makemigrations" y luego volver a realizar el migrate.

Para trabajar con Heroku y Travis CI command line client, instalaremos Heroku en el repositorio usando pip install y Travis CI usando "gem install Travis" (requiere tener Ruby en el entorno de la máquina en la que se está instalando)

6. Gestión de incidencias


Las incidencias pueden y deben ser etiquetadas tanto por los miembros del equipo de desarrollo como por terceros. Según si la persona que está etiquetando la incidencia pertenece a una de estas dos categorías, se deberán usar una serie de etiquetas. Las etiquetas de una misma categoría estarán agrupadas por colores. Las plantillas usadas para realizar peticiones de cambio y reportar bugs se encuentran [aquí](#). Cada miembro del equipo decidirá qué incidencias se asignará. En el momento en el que una persona se asigne una incidencia, será el responsable tanto de trabajar en ella como de actualizar sus etiquetas conforme sea necesario. Cuando se determine que se deba cerrar esa incidencia, será el miembro asignado quien se encargará de cerrarla. Cuando se realice un commit relacionado con una incidencia, en el subject se referenciará a dicha incidencia colocando Issue #numerodelaissue. En cualquier caso, más adelante se mostrará el formato que deberán tener los commits que se realicen.

Un ejemplo de una incidencia en la que se ha seguido este proceso es la incidencia #2:

Implementar sistema de elección aleatoria #2

Edit New iss

 Closed pabreblob opened this issue on 3 Dec 2018 · 0 comments



pabreblob commented on 3 Dec 2018

Owner + 👤 ...

¿Su sugerencia está relacionada con algún problema? Por favor descríballo.

Se desea ofrecer una funcionalidad en la que se escoja como ganadora a una opción de forma aleatoria, teniendo en cuenta que las opciones que hayan sido más votadas deberán tener más opciones de salir elegidas.

Describe una solución que le gustaría ver


Establecer un método de postprocesado que obtenga el porcentaje de cada opción votada, genera un número aleatorio entre 1 y 100 y devuelva una de las opciones según el número obtenido y el porcentaje de votos de cada pregunta.


Describe alternativas que haya considerado

Otras posibilidades: Devolver todas las opciones ordenadas de ésta forma en vez de sólo la primera elegida, implementar un sistema de dble tirada para elegir el número a partir del cual saldrá la opción elegida, aplicar pesos a las diferentes opciones.


Información adicional

No hay información adicional

 pabreblob added **Nueva funcionalidad** **Nueva** **Desarrollador** labels on 3 Dec 2018

 pabreblob self-assigned this on 3 Dec 2018

Assignees

 pabreblob

Labels

Cerrada

Desarrollador

Nueva funcionalidad


Projects

None yet

Milestone

No milestone

Notifications

 Unsubscribe

You're receiving notifications bec you modified the open/close stat

1 participant

Cuando se desee reportar incidencias a otros subsistemas, se seguirán las pautas que estos hayan establecido.

6.1. Etiquetas usadas por miembros del equipo de desarrollo y terceros

Etiquetas de petición de cambios: Color azul.

Estas etiquetas se usarán siempre al crear una incidencia para solicitar un nuevo cambio en el sistema. Se usarán las siguientes etiquetas:

- Nueva funcionalidad: Se sugiere la adición de una nueva funcionalidad.
- Mejora: Se solicita una mejora de una funcionalidad ya existente.
- Bug: Se informa de un mal funcionamiento del sistema
- Documentación: Se solicita la creación o ampliación de documentación relativa al sistema.

Etiquetas de feedback: Color morado.

Estas etiquetas se utilizan para crear una incidencia cuyo objetivo es el de solicitar u ofrecer algún tipo de feedback:

- Duda: Se realiza una pregunta sobre el funcionamiento o diseño del sistema.
- Sugerencia: Se ofrece algún tipo de feedback acerca del sistema.
- Discusión: Se propone una discusión abierta acerca de alguna funcionalidad.

6.2. Etiquetas usadas exclusivamente por el equipo de desarrollo

Etiquetas para marcar el estado de la incidencia: Color verde.

Estas etiquetas se utilizan para seguir el estado de la incidencia:

- Nueva: El equipo de desarrollo ha leído la incidencia pero no ha empezado a trabajar en ella.
- En proceso: Se está trabajando en la incidencia.
- En pruebas: Se ha ejecutado la incidencia pero aún sigue en proceso de pruebas.
- Cerrada: Incidencia resuelta

Etiquetas para marcar incidencias que no serán resueltas: Color rojo.

Estas etiquetas se utilizan para indicar que una incidencia no será resuelta:

- Duplicada: Esta incidencia ya existe.
- Inválida: La incidencia no cumple con las condiciones necesarias para que sea considerada.
- Descartada: El equipo de desarrollo ha decidido no resolver por alguna razón.

Etiquetas para indicar el área de procesos/ rol designado que afronta la incidencia:

Color amarillo

Estas etiquetas se utilizan para categorizar las incidencias según el rol de la persona que la afronte:

- Análisis: Incidencia relacionada con el diseño del sistema.
- Desarrollador: Incidencia relacionada con el código del sistema y su implementación.
- Testeo: Incidencia relacionada con las pruebas de una funcionalidad y la detección de bugs.
- Integración: Incidencia relacionada con la integración del código y posibles conflictos que hayan surgido.

7. Gestión de depuración

Cuando se encuentre un funcionamiento anómalo en el sistema, se creará una incidencia en la que se indique. Existe una plantilla para crear este tipo de incidencias que se encuentra [aquí](#).

Es importante que a la hora de reportar un bug, o funcionamiento incorrecto, se indiquen los pasos a seguir para replicarlo, el comportamiento que se esperaba y el comportamiento anómalo que se ha detectado. Una vez creada esta incidencia, la persona a cargo de la funcionalidad que está funcionando incorrectamente comenzará a trabajar en la incidencia siguiendo las mismas pautas que se indican en el apartado relacionado con la gestión de incidencias.

8. Gestión del código fuente

El repositorio estará estructurado de la siguiente forma:

- La rama principal o master será la que contenga todas las nuevas funcionalidades implementadas y testeadas, y a partir de esta rama se unirán todos los cambios generados al repositorio general del proyecto.
- La rama dev es una rama creada a partir de master. Sirve como rama de desarrollo/integración y será donde se sitúe el código generado por cada miembro del grupo del grupo y haya sido de forma individual. Para poder pasar los cambios de la rama de desarrollo a la rama master es necesario haber probado la versión del código que se encuentra en esta rama y comprobar que no hay conflictos entre los cambios añadidos por cada desarrollador. Esta tarea recaerá sobre el coordinador del grupo

principalmente, aunque cualquier otro miembro podrá realizarla previa autorización del coordinador.

- Cada miembro del equipo de desarrollo tendrá una rama personal que parte desde la rama de desarrollo. Sobre esta rama desarrollará las funcionalidades que tenga asignadas. Un desarrollador hará commits a esta rama cada vez que tenga una funcionalidad nueva implementada y con sus correspondientes tests o una vez cada tres días, en caso de que se hayan producido cambios. Los cambios en las ramas personales no se pasarán a la rama de desarrollo hasta que se haya completado una funcionalidad con sus tests y se haya comprobado el correcto funcionamiento de esta.

A la hora de realizar un commit, el mensaje de commit seguirá el siguiente formato:

Subject: [Descripción del commit][Issue #numero de la issue]. Si no hay ninguna issue relacionada, se describirá brevemente la razón del commit.

Body:[Fecha del commit]


[Descripción de la funcionalidad añadida]

Issue status: [New/In development/ Testing/Closed]


Todo el mensaje de commit se escribirá en inglés, para mantener una cierta coherencia, ya que los commits del repositorio padre ya se encontraban en inglés.

A continuación se muestran algunos commits realizados en la rama personal de uno de los integrantes del equipo:


Fixed error in heroku url Issue #5 ...

 pabreblob committed 28 days ago ✓


17/12/2018
Fixed the url where the app will be deployed
Issue status: In process.

 0ea5087

Added Heroku deployment files Issue #5 ...


 pabreblob committed 28 days ago ✓

17/12/2018
Made changes so that the app can be deployed to Heroku. Automatic deployment using Travis has yet to be implemented.
Issue status: In process.


 5179d06

Commits on Dec 13, 2018

Added more randomSelection tests Issue #3 ...


 pabreblob committed on 13 Dec 2018 ✓

13/12/2018
Added more tests related to the random selection functionality.
Issue status: Testing.


 023ee1a

Commits on Dec 9, 2018

Added randomSelection test Issue#2 ...

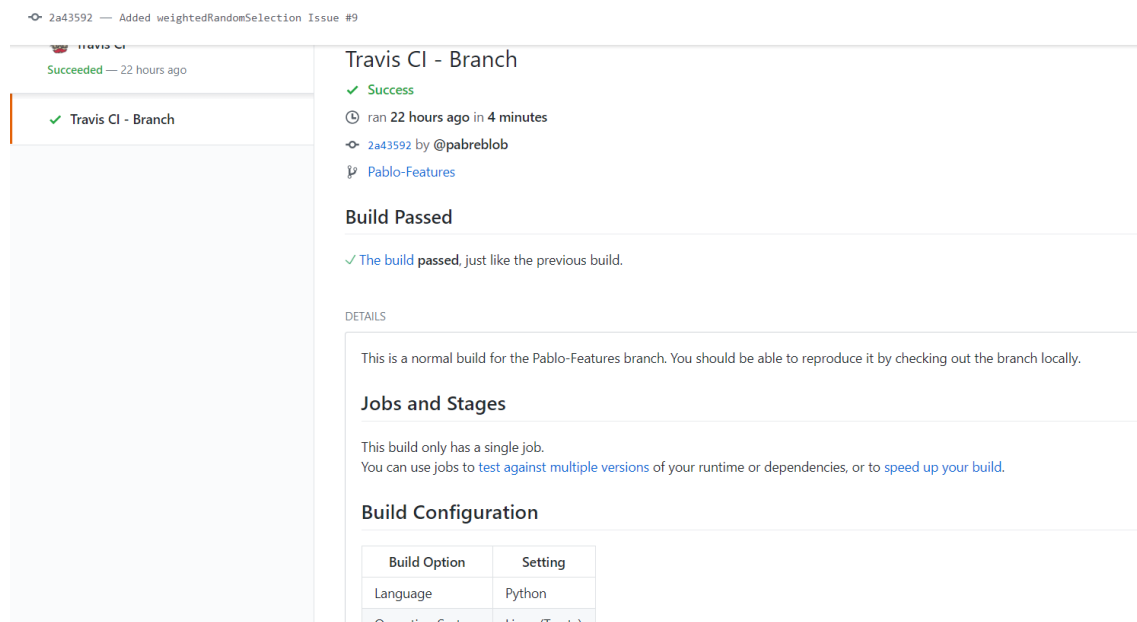
 pabreblob committed on 9 Dec 2018 ✓

09/12/2018
Added a test related to the randomSelection function.
Issue status: Closed.

 f6e85e6

9. Gestión de la construcción e integración continua

Se usará Travis para dar soporte a la integración continua. En cada commit a la rama de desarrollo o a la master, se lanzarán los tests de la aplicación mediante un job de Travis para asegurar la calidad del código subido. Un ejemplo del uso de Travis funcionando cuando se realiza un commit es el siguiente:



2a43592 — Added weightedRandomSelection Issue #9

Travis CI

Succeeded — 22 hours ago

✓ Travis CI - Branch

Travis CI - Branch

✓ Success

🕒 ran 22 hours ago in 4 minutes

🔗 2a43592 by @pabreblob

🏷️ Pablo-Features

Build Passed

✓ The build passed, just like the previous build.

DETAILS

This is a normal build for the Pablo-Features branch. You should be able to reproduce it by checking out the branch locally.

Jobs and Stages

This build only has a single job.
You can use jobs to [test against multiple versions](#) of your runtime or dependencies, or to [speed up your build](#).

Build Configuration

Build Option	Setting
Language	Python
Operating System	Linux (Trusty)

10. Gestión de liberaciones, despliegue y entregas.

La licencia con la que se decide lanzar esta aplicación es GNU AGPL 3.0. El motivo principal por el que se usa esta licencia es porque la aplicación original de la que nace nuestro proyecto usa esta misma licencia. Se trata de una licencia copyleft que obliga a hacer público el código de las modificaciones realizadas sobre la aplicación en caso de que se lance una versión actualizada del software original.

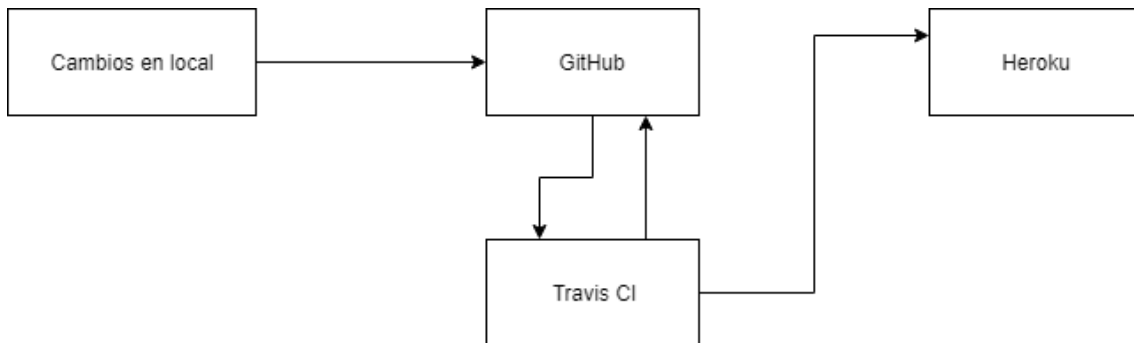
La aplicación se encuentra desplegada en <https://decideiopostproc.herokuapp.com/admin/>. Los credenciales de un superuser para poder acceder al admin de django son user: admin, pass: iopostproc .

El despliegue en Heroku se encuentra automatizado, usando para ello travis de forma que cada vez que se hicieran cambios en la rama master, la aplicación se actualiza.

Cabe destacar que dado que en la aplicación desplegada sólo hay cambios relacionados con el módulo de postprocesado y que para poder enviar peticiones en las que se soliciten usar los nuevos tipos de procesado eran necesarios hacer cambios dentro del módulo de voting, si se trata de hacer el recuento de una votación desde la propia interfaz del admin de django, no será posible aplicar estos nuevos métodos. Para poder comprobar el correcto funcionamiento de estos, aparte de ejecutar los tests en local, es posible realizar peticiones post a la API ofrecida. La documentación relativa a como realizar peticiones para probar estas nuevas funcionalidades se encuentra [aquí](#).

11. Mapa de herramientas

Las herramientas que se han usado en este proyecto han sido GitHub, Travis CI y Heroku.



Los cambios realizados en local serán subidos al repositorio en GitHub. Cuando se detecta un cambio en GitHub, se lanza una automatización de Travis en la que se construye la aplicación y se lanzan los tests y, si el repositorio que ha sido cambiado es el master, se desplegará automáticamente la aplicación en Heroku con los cambios correspondientes. Se informará a GitHub si esta automatización se ha realizado de forma correcta o no.

12. Ejercicio de propuesta de cambio

Un cambio que se podría realizar es la detección y corrección de un bug en uno de los métodos implementados. Supongamos que se detecta un error en el método “weightedOptions”. Los pasos a seguir serían los siguientes:

- Crear una incidencia en la que se indique el bug. Para ello, se seguirá la plantilla indicada y se le asignarán las etiquetas “Bug”, “Nueva”, y alguna relacionada con el rol de la persona encargada de solucionar el bug, como puede ser “Desarrollador” o “Testeo”
- La persona relacionada con la funcionalidad que está fallando se asignará esta incidencia cuando la vea, y una vez que se ponga a trabajar en ella, le quitará la etiqueta “Nueva” para poner “En proceso”.
- El encargado de solucionar el problema comenzará a trabajar en resolverlo desde su rama personal. Para ello, configurará el entorno de desarrollo, lo cual incluye actualizar la estructura de la base de datos usando “python manage.py migrate” Para poder depurar el error, lo primero será lanzar los tests, usando el comando “python manage.py test postproc/”. Preferiblemente, se establecerán breakpoints en el IDE que se esté usando en los métodos donde puede estar el error. Si se localiza el error, se procederá a corregirlo y a volver a ejecutar los tests para comprobar si se ha solucionado correctamente. Si no, el siguiente paso a seguir será lanzar la aplicación en el servidor local usando el comando “python manage.py runserver” y comenzar a debugear desde allí. En cualquier caso, una vez que se ha solucionado el error, se comprobará que la aplicación funciona correctamente, tanto por medio de los tests como usando la aplicación desplegada en local.
- Para subir los cambios al repositorio, se usará el comando “git add .” para añadir los cambios al commit, luego “git commit -m ‘subjectmessage’” para realizar el commit y por último “git push origin nombredelarama”. Una vez subido el cambio a la rama, deberá cambiarse a la rama de desarrollo usando “git checkout dev” y hacer merge de los cambios en la rama personal con “git merge nombredelarama” y luego hacer otro

git push. Este proceso se repetirá cuando se quieran pasar los cambios a la rama master.

- Se sustituirá la etiqueta de la incidencia “En proceso” por “Cerrada” y se cerrará la incidencia

13. Conclusiones y trabajo futuro

Como lecciones aprendidas que hemos sacado de este trabajo, se debe mencionar que la comunicación con otros módulos ha sido bastante mejorable, sólo nos hemos integrado con un módulo y esa integración se ha limitado a ofrecerles una documentación. El módulo de votación, el cual era importante para mostrar el funcionamiento de nuestros cambios, no fue capaz de dar respuesta a lo que le propusimos, si bien es cierto que de cara a mostrar nuestras interacciones con ellos cometimos el error de no poner issues en primer lugar sino comunicarnos con ellos por privado, y pese a que se nos dio una respuesta negativa, no hay incidencias que dejen constancia de ello. Como algo positivo, queda el haber aprendido el funcionamiento de una aplicación de Django y el cómo funcionan tecnologías como Travis o Heroku.

A modo de trabajo futuro, queda el implementar el procesado aplicando la ley Hondt o el implementar una forma de poder seleccionar el tipo de recuento a hacer dentro del admin de Django