



DECIDE-ORTOSIA-POSTPROCESADO

Documento del proyecto

Grupo: 1

ID de opera: 124

Miembros del grupo:

- Cantón Fernández, Adrián: 5
- Carpio Camacho, Daniel: 5
- Fresno Aranda, Rafael: 5
- Rodríguez Méndez, Raúl: 1
- Segura Jiménez, Antonio: 5

Enlaces de interés

- Repositorio: <https://github.com/pablotabares/decide/tree/ortosia-postproc>
- Heroku: <https://decide-ortosia-postproc.herokuapp.com/>
(Usuario: adminname Contraseña: adminpasswd)
- Docker web: https://hub.docker.com/r/raffrearaus/ortosia-postproc_web
- Docker nginx: https://hub.docker.com/r/raffrearaus/ortosia-postproc_nginx

Contenido

1. Resumen.....	2
2. Introducción y contexto	2
3. Descripción del sistema.....	3
4. Planificación del proyecto	4
4.1. Descripción de la planificación.....	4
4.2. Reparto de tareas.....	4
5. Entorno de desarrollo	4
6. Gestión de incidencias.....	8
6.1. Issues	10
6.2. Gestión de incidencias internas	13
6.3. Gestión de incidencias externas.....	13
7. Gestión de depuración	13
8. Gestión del código fuente	15
9. Gestión de la construcción e integración continua.....	18
10. Gestión de liberaciones, despliegue y entregas.....	20
11. Mapa de herramientas.....	25
12. Ejercicio de propuesta de cambio	25
13. Integración con otros módulos	27
14. Conclusiones y trabajo futuro	27

1. Resumen

Decide es una herramienta de código libre bajo la licencia AGPL-3.0 y que tiene como objetivo permitir el voto online de manera segura y anónima. El grupo está dividido en subgrupos los cuales trabajan en los distintos módulos de la aplicación.

Nuestro grupo se encarga del subsistema de postprocesado. Tras finalizar una votación y obtener los votos de la base de datos, nuestro módulo es el encargado de analizarlos y devolver una respuesta que es la que se muestra a los usuarios. Los módulos se comunican entre ellos mediante llamadas a APIs REST, lo cual permite que la implementación interna de cada módulo sea invisible al resto de los módulos pudiendo ser implementados en el lenguaje deseado, aunque en nuestro caso hemos seguido la tecnología usada por los creadores.

Nuestra labor ha sido la de implementar más métodos de postprocesados. Para nuestra organización hemos utilizado la plataforma Github en el cual definíamos issues para identificar las distintas tareas a realizar y las fechas en las cuales deben estar terminadas.

Se ha añadido también Travis como asistente de integración continua que permite ejecutar pruebas automáticas, permitiendo también el despliegue automático de versiones estables a Heroku. Podemos obtener el sistema directamente del repositorio, descargándonos la imagen disponible para Docker o accediendo a la plataforma de Heroku

2. Introducción y contexto

Decide es una plataforma de voto electrónico seguro que cumple con una serie de garantías básicas, como el anonimato y el secreto de voto. Es una plataforma educativa, por lo que prima la simplicidad de la misma, para que sea entendible por parte de los estudiantes.

El proyecto está dividido en subsistemas que se conectan mediante APIs permitiendo nivel de desacople entre los mismos mayor. Cada subsistema puede estar desplegado en servidores distintos o implementados en lenguajes diferentes siempre y cuando se cumpla que las entradas y salidas de estos sigan unas especificaciones. Los distintos subsistemas de los que se componen son:

- Autenticación: encargado de la autenticación de los votantes.
- Censo: gestiona el grupo de personas que puede participar en una votación.
- Votación: es el que define las preguntas.
- Cabina de votación: ofrece una interfaz para participar en una votación.
- Almacenamiento: guarda los votos cifrados en una base de datos y la relación con el votante
- Recuento: se encarga de la parte criptográfica, es el que asegura el anonimato de los votos. Se desliga por tanto el voto del votante. Se divide en autoridades y estas deben descifrar una parte de los votos totales.
- Postprocesado: recibe la lista con los números de votos de las distintas opciones y los traduce a un resultado coherente en función del tipo de voto.
- Visualización: encargado de coger los datos que devuelve el método de postprocesado y mostrarlos de forma entendible a los votantes.

Nuestro grupo se ha encargado de evolucionar el subsistema de postprocesado añadiendo más tipos de procesamiento de los datos.

Para que estos subsistemas funcionen es necesario de un proyecto base que es el que conecta los distintos subsistemas entre sí y le da sentido. El proyecto está desarrollado usando Django aunque cualquier subsistema puede ser reemplazado de forma individual.

3. Descripción del sistema

El subsistema que estamos evolucionando, como ya sabemos es el de postprocesado. En este caso, nuestra labor es la de añadir más tipos de procesamiento de los datos.

Nuestro sistema recibe los datos obtenidos del módulo de recuento al finalizar una votación, y debe devolver un resultado para que el sistema de visualización represente datos con sentido. A continuación, vamos a describir los métodos de postprocesado tanto los que venían inicialmente, como los que hemos implementado nosotros:

- **Identity:** este método venía implementado por los desarrolladores de Wadobo y simplemente consideraba que si una votación tenía 3 votos para la opción a y 2 para la b, el sistema devolvía 3 votos para la a y 2 para la b.
- **Weight:** el primer método implementado por nuestro equipo. Para que este método tenga sentido, es necesario desde el módulo de votación establecer un peso a cada opción de una pregunta. El sistema devuelve para la opción a, el número de votos recibido por el módulo de recuento de la opción a multiplicado por el peso, y así sucesivamente con las múltiples opciones.
- **WeightedRandomSelection:** este método devuelve una opción aleatoria, pero con mayor probabilidad de aquellas que tienen más votos.
- **Sistema de Hondt:** método de procesado bastante conocido, por ser el usado actualmente en las elecciones de nuestro país. Se indica el número de asientos a repartir, y en función de los votos se hace el reparto. El método devuelve el número de asientos de cada opción en función de los votos obtenidos. Para ello es necesario establecer previamente el número de asientos a repartir.
- **Borda:** método en el cual al votar, no se selecciona una opción, si no que se establecen unas prioridades, que son las almacenadas en la base de datos, y al hacer el recuento, por cada voto (lista de prioridades), a la primera opción de la lista obtiene n votos, a la segunda n-1, y así hasta la última que obtiene 1 voto. Esto se debe hacer por cada voto almacenado y finalmente, se devuelve la lista de opciones con los votos a cada una de ella.
- **Multiquestion:** este método permite procesar múltiples preguntas a la vez. Tiene como objetivo permitir hacer encuestas de más de una pregunta. El tipo de procesado que se realiza aquí es el de identidad.
- **GenderBalanced:** método de procesado cremallera, las opciones tienen un “genero”, y el método devuelve la lista de las opciones alternando los géneros.
- **DroopQuota:** sistema de reparto de escaño, en el cual se calcula el número de votos necesarios para cada escaño, se reparten los escaños en función de los votos de cada opción y si quedan escaños, en el caso de que la división de escaños entre votos no sea exacta se reparten en función de mayor a menor votos.
- **SainteLague:** sistema de reparto de escaños en función de los votos, similar al sistema de Hondt, pero los asientos se reparten siguiendo otra fórmula.

Todos estos métodos están implementados en:

<https://github.com/pablotabares/decide/blob/ortosia-postproc/decide/postproc/views.py>

Para todos estos métodos hemos implementado tests que aseguran el correcto funcionamiento de estos y que serán utilizados más adelante para garantizar la integración continua. Los tests podemos verlos en:

<https://github.com/pablotabares/decide/blob/ortosia-postproc/decide/postproc/tests.py>

También se ha creado una colección de tests para la aplicación Postman disponible en:

https://github.com/pablotabares/decide/blob/ortosia-postproc/decide/postproc/doc/Postproc_tests.postman_collection.json

4. Planificación del proyecto

4.1. Descripción de la planificación

Al inicio del desarrollo, el equipo tuvo una reunión para establecer los métodos de postprocesado que se iban a implementar y hacer un primer reparto de tareas.

El proyecto ha sido desarrollado mediante tres iteraciones. En la primera iteración teníamos como objetivo tener el sistema desplegado en local y además de tener implementado los siguientes métodos: weight, weightedrandomselection, borda, genderbalanced y multiquestion.

En la segunda iteración, se implementaron los demás métodos de postprocesado y se incluyeron tests automáticos mediante la herramienta Travis, para garantizar la integración continua. Además se automatizó el proceso de despliegue en Heroku.

Finalmente en la tercera iteración, se generó la documentación requerida, se solucionaron bugs y se integró con los cambios de los demás subsistemas en la medida de lo posible.

Antes de finalizar cada iteración se realizó una reunión para ver los problemas que teníamos cada uno y preparar los milestones.

4.2. Reparto de tareas

- Adrián Cantón Fernández: implementación weight, sistema de Hondt
- Daniel Carpio Camacho: implementación weighted random selection, gender balanced, droop quota
- Rafael Fresno Aranda: implementación multipreguntas, automatización tests y despliegue
- Antonio Segura Jiménez: implementación Borda y Sainte Lague

El trabajo realizado se puede ver con mayor detalle en el diario del equipo disponible en:

<https://github.com/pablotabares/decide/blob/ortosia-postproc/decide/postproc/doc/Diario%20del%20equipo.pdf>

5. Entorno de desarrollo

Todos los integrantes del grupo han usado estas tecnologías instaladas en sus respectivos ordenadores, usando las mismas versiones en todos ellos:

Python 3.7.1 (más información en <https://www.python.org/>)

Django 2.0.0 (más información en <https://www.djangoproject.com/>)

PostgreSQL 11.1 (más información en <https://www.postgresql.org/>)

Estas tecnologías son necesarias para el correcto funcionamiento del software.

Respecto a la hora de editar el código, a cada miembro del grupo se le permite utilizar el software con el que se sintiese más cómodos a la hora de trabajar. Finalmente se acabó utilizando:

- PyCharm

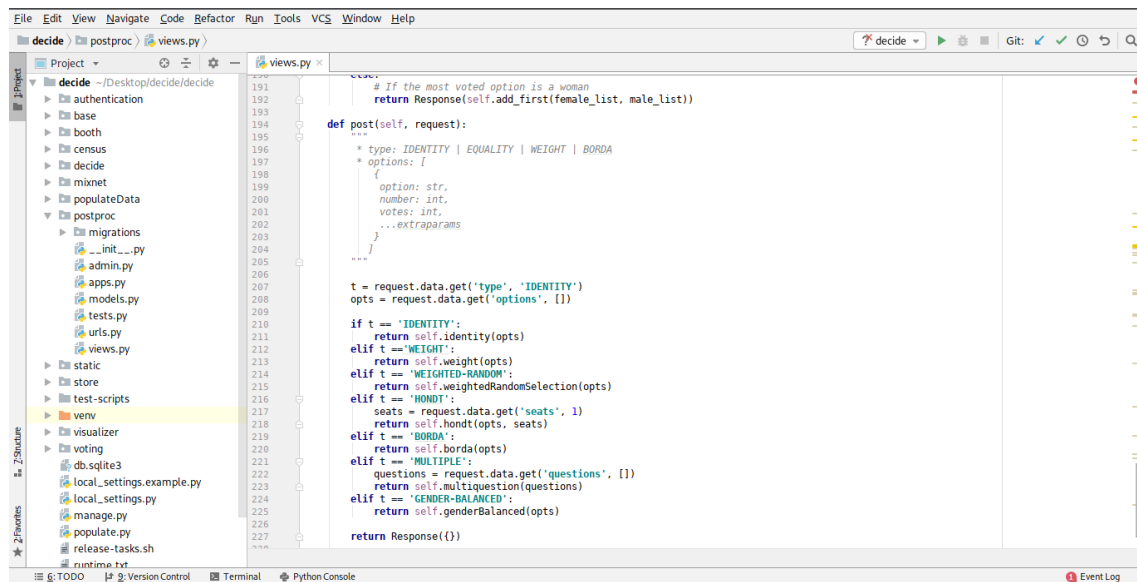


Imagen 1: Entorno de desarrollo Pycharm

Disponible en <https://www.jetbrains.com/pycharm/>.

- Visual Studio Code

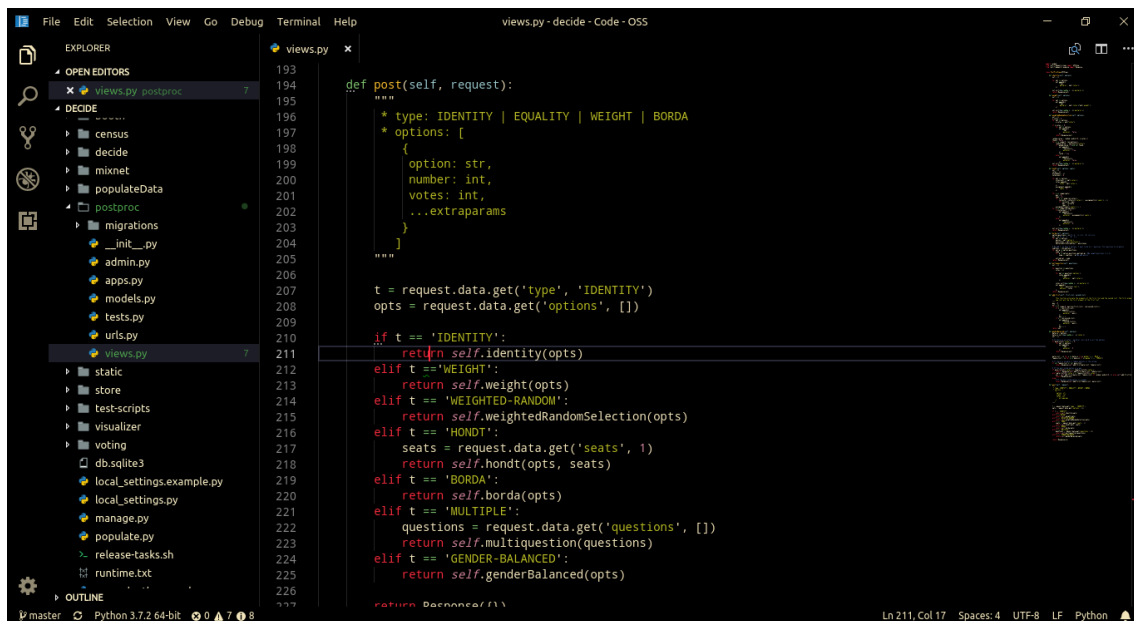


Imagen 2: Entorno de desarrollo Visual Studio Code

Disponible en <https://code.visualstudio.com/>,

Para la instalación del software Decide en un ordenador en local, se deben seguir estos pasos:

1. Instalar todos los paquetes necesarios. Esto se puede usar mediante el comando:
`pip install -r requirements.txt`
en la carpeta donde se encuentre el archivo `requirements.txt`. Se requiere acceso como superusuario.

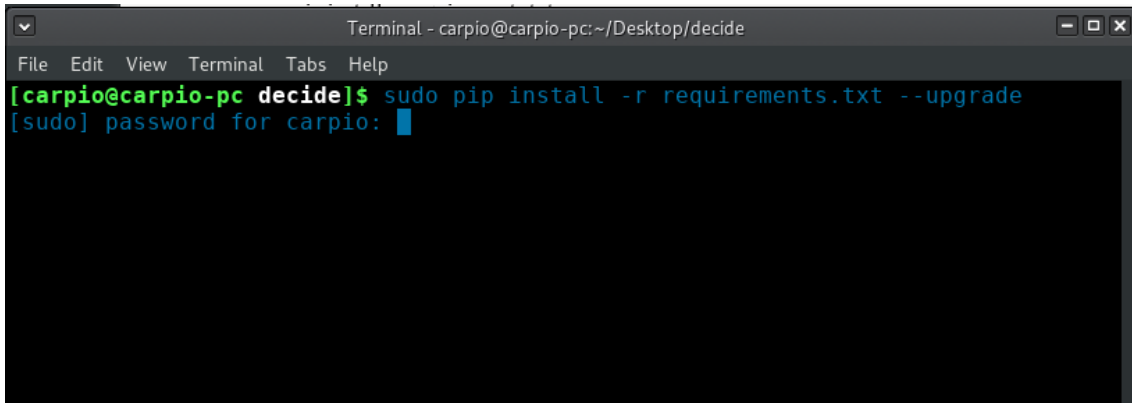


Imagen 3: Instalación de paquetes necesarios

2. Crear el usuario en la base de datos PostgreSQL. Para ello, se debe lanzar la siguiente sentencia SQL:
`create user decide with password 'decide';`
3. Crear la base de datos. Para ello, deberá lanzar la siguiente sentencia SQL:
`create database decide owner decide;`

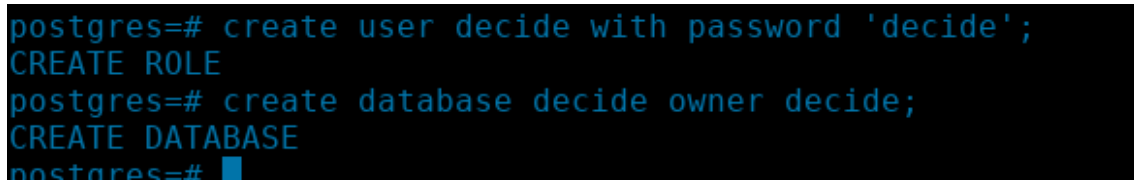


Imagen 4: Creación de usuario y base de datos

4. Con la configuración actual, se podrá desplegar el software correctamente. No obstante, si desea lanzar los tests que ya vienen en el propio proyecto, será necesario que también lance la siguiente sentencia. Esto permite al usuario 'decide' crear bases de datos. Todos los tests se ejecutarán en una base de datos temporal para que no afecten a los datos reales del proyecto.
`alter user decide createdb;`



Imagen 5: Permitir crear bases de datos al usuario decide

5. Crear el archivo `local_settings.py` a partir del archivo `local_settings.example.py`. El nuevo archivo creado deberá localizarse en la misma ruta que el archivo original. Además, será necesario editar los valores de las variables que se encuentran dentro. Los cambios que se deberán realizar son editar las direcciones donde se encuentran las APIs, que pasarán a ser <http://localhost:8000>, editar la constante `BASEURL` por la misma dirección que antes y editar los datos de la base de datos, cuyo `HOST` será `'localhost'` y cuyo nombre, usuario y contraseña será `'decide'`.

```
4  MODULES = [  
5      'authentication',  
6      'base',  
7      'booth',  
8      'census',  
9      'mixnet',  
10     'postproc',  
11     'store',  
12     'visualizer',  
13     'voting',  
14 ]  
15  
16  APIS = {  
17     'authentication': 'localhost:8000',  
18     'base': 'localhost:8000',  
19     'booth': 'localhost:8000',  
20     'census': 'localhost:8000',  
21     'mixnet': 'localhost:8000',  
22     'postproc': 'localhost:8000',  
23     'store': 'localhost:8000',  
24     'visualizer': 'localhost:8000',  
25     'voting': 'localhost:8000',  
26 }  
27  
28  BASEURL = 'localhost:8000'  
29  
30  DATABASES = {  
31     'default': {  
32         'ENGINE': 'django.db.backends.postgresql',  
33         'NAME': 'postgres',  
34         'USER': 'postgres',  
35         'PASSWORD': 'postgres',  
36         'HOST': '127.0.0.1',  
37         'PORT': '5432',
```

Imagen 6: Fichero `local_settings.py`

6. Ejecutar la primera migración lanzando en la terminal:

python manage.py migrate



```
[carpio@carpio-pc decide]$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, authtoken, base, census, contenttypes, mixnet, sessions, store, voting
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying authtoken.0001_initial... OK
  Applying authtoken.0002_auto_20160226_1747... OK
  Applying base.0001_initial... OK
  Applying base.0002_auto_20180921_1056... OK
  Applying base.0003_auto_20180921_1119... OK
  Applying census.0001_initial... OK
  Applying mixnet.0001_initial... OK
  Applying mixnet.0002_auto_20180216_1617... OK
  Applying voting.0001_initial... OK
  Applying voting.0002_auto_20180302_1100... OK
  Applying voting.0003_auto_20180605_0842... OK
  Applying mixnet.0003_mixnet_auth_position... OK
  Applying mixnet.0004_auto_20180605_0842... OK
  Applying sessions.0001_initial... OK
  Applying store.0001_initial... OK
  Applying store.0002_vote_voted... OK
  Applying store.0003_auto_20180921_1522... OK
[carpio@carpio-pc decide]$
```

Imagen 7: Migración de la base de datos

7. Ejecutar el servidor:

python manage.py runserver

Finalmente el software se estará ejecutando en <http://localhost:8000>.

6. Gestión de incidencias

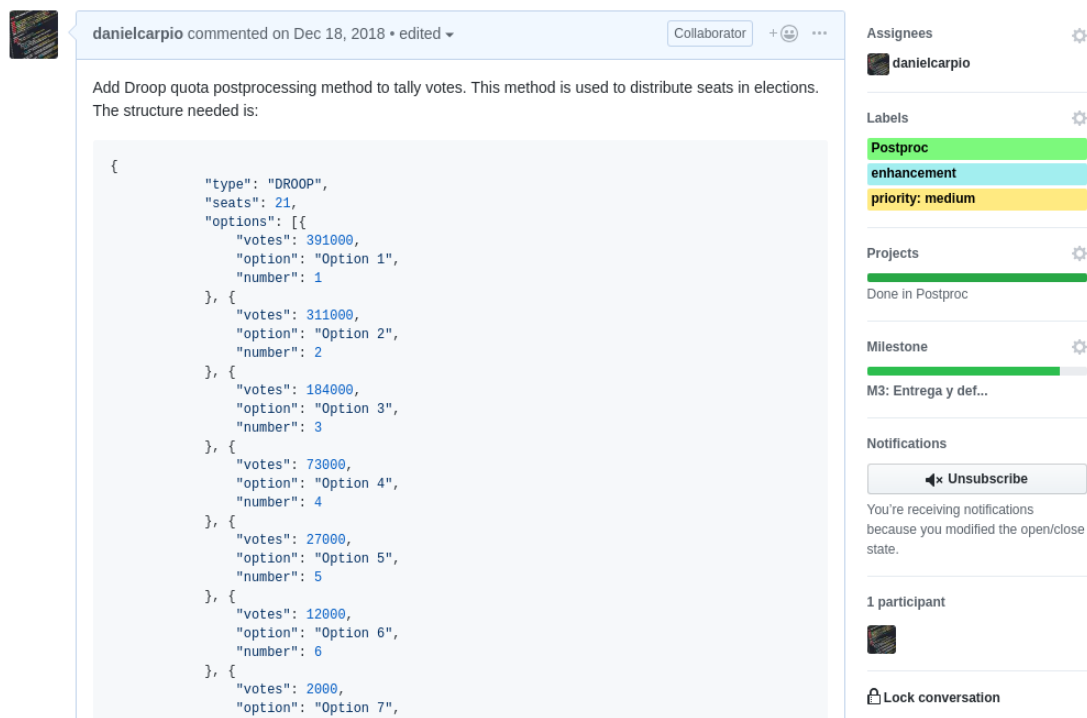
Para la gestión de incidencias, se utilizará el sistema de issues que nos proporciona GitHub.

Toda incidencia debe cumplir estas indicaciones:

- Debe llevar un título breve y descriptivo. Debido a que nuestro grupo está trabajando en el mismo repositorio que otros, toda issue creada debe empezar por "Postproc - " para diferenciarse del resto.
- Debe llevar una descripción del problema lo más amplia posible. Si dicha issue es para añadir una funcionalidad a nuestro software, debe llevar un ejemplo de la estructura del JSON que se usará en dicha funcionalidad.
- Debe llevar etiquetas para poder clasificar las issues. Debe llevar como mínimo tres etiquetas: la etiqueta 'Postproc' para designar a nuestro equipo de trabajo, la prioridad establecida y una etiqueta que indique el objetivo de la issue, como por ejemplo 'enhancement', 'bug' o 'optimization'. Se pueden añadir más etiquetas si se desea.
- Debe asignarse a todos los miembros que tengan alguna relación con la issue, ya sea de nuestro equipo de trabajo o de otro equipo. Si nuestra issue afecta a algún otro equipo, también se le debe notificar vía mensajería instantánea.

- Toda issue debe indicar el milestone donde se presentará la funcionalidad. Toda issue debe estar cerrada y finalizada antes de la fecha del milestone. De no haberse podido finalizar a tiempo, se debe cambiar el milestone establecido en la issue.
- Toda issue debe tener asignado el proyecto creado por el grupo en GitHub, llamado 'Postproc'. Durante el transcurso de la tarea, se debe ir modificando el estado de la issue en el proyecto a cada una de las opciones disponibles: 'To do', estado en el que se encuentra inicialmente que indica que todavía no se ha comenzado a trabajar en dicha tarea; 'In Progress', que indica que se está trabajando actualmente; 'To integrate', que indica que ha sido finalizada por nuestro grupo de trabajo pero todavía está pendiente a que se finalice por parte de otros equipos de trabajo; y 'Done', que indica que ha sido finalizada.
- Todo commit que se realice debe incluir en su descripción una referencia a la issue que se esté tratando en ese momento.

Este es un ejemplo de una issue creada correctamente. Corresponde con la issue #105. Se puede acceder mediante esta URL: <https://github.com/pablotaabares/decide/issues/105>.



The screenshot shows a GitHub issue comment by user 'danielcarpio' on December 18, 2018. The comment describes a method for adding a Droop quota postprocessing method to tally votes. It includes a JSON structure with the following data:

```
{
  "type": "DROOP",
  "seats": 21,
  "options": [
    {
      "votes": 391000,
      "option": "Option 1",
      "number": 1
    },
    {
      "votes": 311000,
      "option": "Option 2",
      "number": 2
    },
    {
      "votes": 184000,
      "option": "Option 3",
      "number": 3
    },
    {
      "votes": 73000,
      "option": "Option 4",
      "number": 4
    },
    {
      "votes": 27000,
      "option": "Option 5",
      "number": 5
    },
    {
      "votes": 12000,
      "option": "Option 6",
      "number": 6
    },
    {
      "votes": 2000,
      "option": "Option 7",
      "number": 7
    }
  ]
}
```

The right sidebar of the GitHub interface shows the following details for the issue:

- Assignees:** danielcarpio
- Labels:** Postproc (green), enhancement (blue), priority: medium (yellow)
- Projects:** Done in Postproc (green progress bar)
- Milestone:** M3: Entrega y def... (green progress bar)
- Notifications:** Unsubscribe button, with a note: "You're receiving notifications because you modified the open/close state."
- Participants:** 1 participant (danielcarpio)
- Lock conversation:** Lock icon and text

Imagen 8: Ejemplo de commit

Si se desea ver todas las issues creadas por el módulo Postproc, se pueden ver en la siguiente URL:
<https://github.com/pablotaabares/decide/issues?utf8=%E2%9C%93&q=is%3Aissue+label%3APostproc>.

6.1. Issues

Id	2
Enlace	https://github.com/pablotabares/decide/issues/2
Título	Integrate weighted postprocessing
Creada por	Adrián Cantón Fernández
Asignada a	Adrián Cantón Fernández
Descripción	Se incluye un nuevo método de recuento de votos: votos con peso. En esta issue van incluidos tanto el código que añade la funcionalidad como los tests.

Id	8
Enlace	https://github.com/pablotabares/decide/issues/8
Título	Integrate weighted random selection postprocessing
Creada por	Daniel Carpio Camacho
Asignada a	Daniel Carpio Camacho
Descripción	Se incluye un nuevo método de recuento de votos: selección aleatoria ponderada. En esta issue van incluidos tanto el código que añade la funcionalidad como los tests.

Id	10
Enlace	Enlace https://github.com/pablotabares/decide/issues/10
Título	Integrate Hondt postprocessing
Creada por	Adrián Cantón Fernández
Asignada a	Adrián Cantón Fernández
Descripción	Se incluye un nuevo método de recuento de votos: el sistema d'Hondt. En esta issue van incluidos tanto el código que añade la funcionalidad como los tests.

Id	16
Enlace	https://github.com/pablotabares/decide/issues/16
Título	Integrate Borda postprocessing
Creada por	Antonio Segura Jiménez
Asignada a	Antonio Segura Jiménez
Descripción	Se incluye un nuevo método de recuento de votos: el recuento Borda. En esta issue van incluidos tanto el código que añade la funcionalidad como los tests.

Id	17
Enlace	https://github.com/pablotabares/decide/issues/17
Título	Integrate multiple questions
Creada por	Rafael Fresno Aranda
Asignada a	Rafael Fresno Aranda
Descripción	Se incluye un nuevo método de recuento de votos: múltiple preguntas. En esta issue van incluidos tanto el código que añade la funcionalidad como los tests.

Id	36
Enlace	https://github.com/pablotabares/decide/issues/36
Título	Failure in Hondt test
Creada por	Antonio Segura Jiménez
Asignada a	Adrián Cantón Fernández
Descripción	Fallaba un test cuando se ejecutaban los del sistema d'Hondt.

Id	43
Enlace	https://github.com/pablotabares/decide/issues/43
Título	Failure in Hondt method
Creada por	Daniel Carpio Camacho
Asignada a	Adrián Cantón Fernández
Descripción	Se encontró un bug cuando se ejecutaba el sistema d'Hondt. Este daba valores incorrectos cuando en una votación no había votado nadie.

Id	45
Enlace	https://github.com/pablotabares/decide/issues/45
Título	Integrate gender balanced postprocessing
Creada por	Daniel Carpio Camacho
Asignada a	Daniel Carpio Camacho
Descripción	Se incluye un nuevo método de recuento de votos: el recuento "cremallera". En esta issue van incluidos tanto el código que añade la funcionalidad como los tests.

Id	47
Enlace	https://github.com/pablotabares/decide/issues/47
Título	Reduce cognitive complexity
Creada por	Daniel Carpio Camacho
Asignada a	Daniel Carpio Camacho
Descripción	SonarCloud detectó que el código escrito para el recuento "cremallera" comentado en la issue #45 (https://github.com/pablotabares/decide/issues/45) tenía una complejidad cognitiva alta, lo que provocaba que futuros desarrolladores tardasen más de lo necesario en comprender su funcionalidad.

Id	78
Enlace	https://github.com/pablotabares/decide/issues/78
Título	Integrate Heroku with Travis CI
Creada por	Rafael Fresno Aranda
Asignada a	Rafael Fresno Aranda
Descripción	Integración de la herramienta Heroku en Travis CI para que se pueda desplegar automáticamente el sistema en un servidor cada vez que se suba el código a GitHub.

Id	105
Enlace	https://github.com/pablotabares/decide/issues/105
Título	Integrate Droop quota
Creada por	Daniel Carpio Camacho
Asignada a	Daniel Carpio Camacho
Descripción	Se incluye un nuevo método de recuento de votos: cociente Droop.

Id	108
Enlace	https://github.com/pablotabares/decide/issues/108
Título	Add Droop quota postprocessing method tests
Creada por	Daniel Carpio Camacho
Asignada a	Daniel Carpio Camacho
Descripción	Descripción Incluye diferentes tests para comprobar la funcionalidad “cociente Droop”, descrito en la issue #105 (https://github.com/pablotabares/decide/issues/105)

Id	130
Enlace	https://github.com/pablotabares/decide/issues/130
Título	Integrate Sainte-Laguë method
Creada por	Antonio Segura Jiménez
Asignada a	Antonio Segura Jiménez
Descripción	Se incluye un nuevo método de recuento de votos: método Sainte-Laguë. En esta issue van incluidos tanto el código que añade la funcionalidad como los tests.

Id	156
Enlace	https://github.com/pablotabares/decide/issues/156
Título	Unable to save votes with multiple questions
Creada por	Pablotabares
Asignada a	Pablotabares
Descripción	Aviso a todos los módulos debido a un problema al guardar los votos al crear múltiples preguntas

Id	166
Enlace	https://github.com/pablotabares/decide/issues/166
Título	Create and upload Docker images
Creada por	Rafael Fresno Aranda
Asignada a	Rafael Fresno Aranda
Descripción	Creación de las imágenes y envío a Docker Hub para poder obtenerlas fácilmente en un futuro. Estas imágenes se harán de la rama ‘ortosiapostproc’.

Id	194
Enlace	https://github.com/pablotaabares/decide/issues/194
Título	Create Postman tests collection
Creada por	Adrián Cantón Fernández
Asignada a	Adrián Cantón Fernández
Descripción	Creación de un conjunto de tests para comprobar la funcionalidad en el servidor de Heroku.

Id	204
Enlace	https://github.com/pablotaabares/decide/issues/204
Título	Reduce tests redundancy
Creada por	Rafael Fresno Aranda
Asignada a	Rafael Fresno Aranda
Descripción	Tras evaluar la gestión del código, se comprueba que se ejecutan los tests un número elevado de veces. Se desea modificar el archivo Travis para reducirlo.

6.2. Gestión de incidencias internas

Toda comunicación entre los integrantes del grupo se hará de dos formas: reuniones en persona o vía mensajería instantánea. Las reuniones, cuya fecha y hora se establecerá tras mutuo acuerdo entre todos los integrantes del grupo, se reservarán para temas mayores, mientras que la mensajería instantánea se reservará para dudas menores.

Se deberá informar a todos los elementos del grupo de toda funcionalidad que se vaya a añadir o problema encontrado en el software.

6.3. Gestión de incidencias externas

Toda comunicación entre integrantes de distintos grupos se realizará mediante mensajería instantánea. Solo habrá una persona en nuestro grupo que se encargue de todas las comunicaciones externas, siendo este el responsable de informar a los demás miembros del grupo posteriormente.

Se deberá informar de todas las funcionalidades que se vayan a añadir o problemas encontrados en el software a otro equipo de trabajo siempre que afecte a su parte.

Una vez informados, serán ellos los que se organicen de acuerdo a su forma de actuar, siendo responsables de asignar a uno o varios miembros para trabajar en la issue. Será esta persona la encargada de asignarse en la tarea correspondiente.

7. Gestión de depuración

Para todo bug que se produzca en nuestro software será necesario que la persona que lo haya encontrado proporcione toda la información posible, por ejemplo los pasos que había realizado hasta llegar al fallo o ciertos datos de su ordenador, como el sistema operativo y las versiones de los paquetes, para poder replicarse el error. Cuando se haya asignado a un miembro del equipo para tratar dicho problema, que normalmente será el que haya tenido más relación con esa parte del código debido a que conocerá mejor su funcionamiento, procederá a ejecutar estos pasos:

1. Intentar replicar el error. Se podrá usar máquinas virtuales para conseguir una réplica más exacta del entorno de la persona que ha encontrado el bug.
2. Reinstalación de los paquetes. Posteriormente se deberá comprobar si se ha solventado el error.
3. Si no se ha solucionado, se deberá localizar la parte del código en la que se produce el error. Esto se podrá hacer de la forma que el miembro del equipo considere más óptimo, ya sea con las herramientas que ofrezcan los editores de código o con logs que vaya informando de los valores en variables son correctos.
4. En el caso de que no haya localizado el error, añadirá la etiqueta 'help wanted' en la issue, buscando a más miembros para arreglar el bug. Cuando ocurra, se le asignará en la issue y volverán al paso 3.
5. La persona asignada procederá a arreglar el código.
6. En el caso de que no pueda corregir el error, añadirá la etiqueta 'help wanted' en la issue, buscando a más miembros para arreglar el bug. Cuando ocurra, se le asignará en la issue y volverán al paso 3.
7. En el caso de que se haya arreglado el bug, cambiará el estado de la issue en el proyecto a 'Done'.

Incorporamos el siguiente diagrama para facilitar la comprensión de los pasos a seguir.

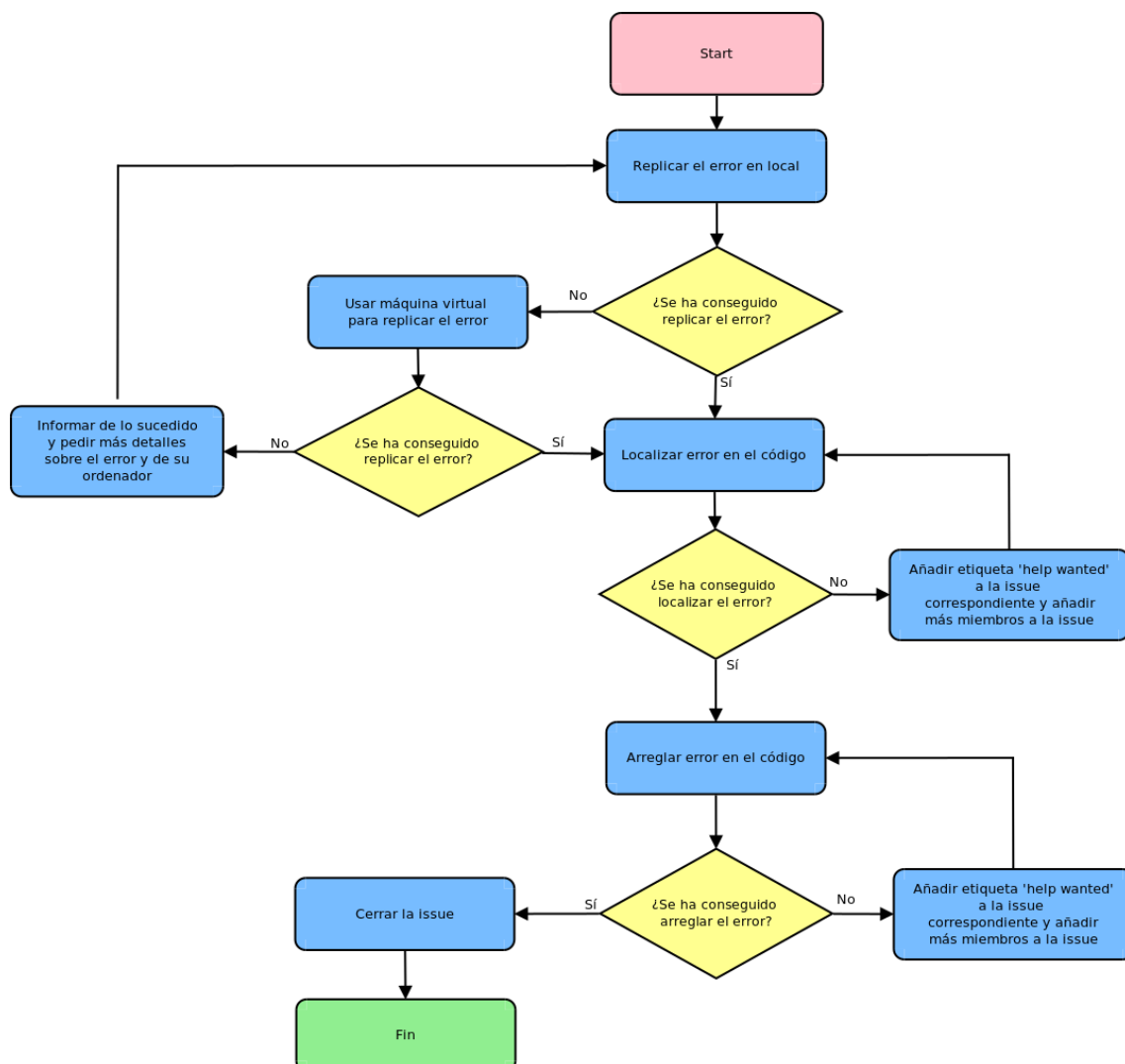


Imagen 9: Diagrama de depuración

A continuación se mostrará un ejemplo real la gestión de depuración de nuestro grupo.

Un miembro de nuestro equipo encontró un bug en el método de postprocesado de Hondt. Acto seguido, creó la issue #43 de acorde a lo establecido en la gestión de incidencias.

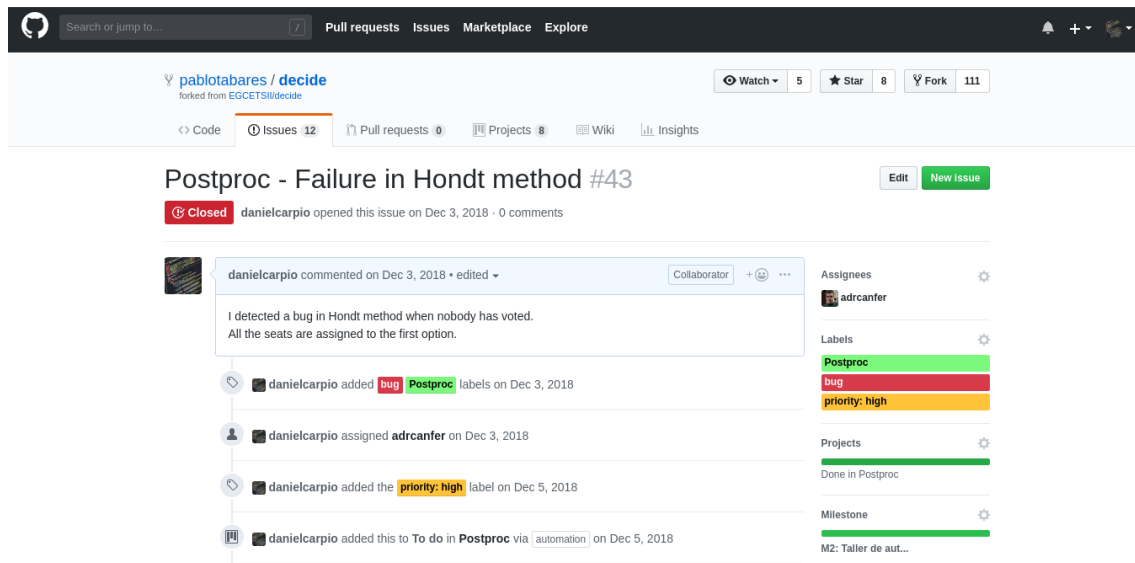


Imagen 10: Ejemplo de bug detectado

Se puede acceder a la issue mediante <https://github.com/pablatabares/decide/issues/43>.

Posteriormente, se estableció a un miembro del equipo para poder solucionarlo. Una vez empezó a trabajar, replicó el error en su ordenador, para una mayor comprensión del problema a solucionar. Tras poder replicarlo y entenderlo, empezó a resolver el problema.

Tras resolver los errores, se ejecutó varias veces para comprobar que se había solucionado correctamente. Posteriormente, añadió más tests que incluyesen ese caso de uso que no había tenido en cuenta anteriormente. Una vez que el problema se solucionó y todos los tests se ejecutaron sin problemas, realizó el commit indicando la issue en la descripción. Finalmente, movió el estado de la issue a 'Done' y la cerró.

8. Gestión del código fuente

Para gestionar el código fuente del proyecto, se hace necesario utilizar alguna herramienta que nos permita disponer de diferentes versiones de los archivos, así como que cada miembro pueda trabajar por separado en las características que esté desarrollando sin que los cambios de otros miembros puedan interferir, y que posteriormente todos los cambios se puedan unir resolviendo los conflictos que pudieran surgir. Por lo tanto, el proceso general que se sigue en nuestro caso es programar por separado las funcionalidades que se han asignado a cada miembro, y, una vez terminadas y probadas, unirlas con el resto de adiciones o correcciones que han realizado los demás miembros del equipo.

Como herramienta principal para conseguir este fin, utilizamos el sistema de control de versiones Git. Este sistema está disponible para todos los sistemas operativos y su instalación es muy simple. Una vez instalado, se debe configurar el nombre de usuario y el correo electrónico, datos que se usarán para identificar a los autores de los diferentes cambios que se realicen sobre el proyecto. Posteriormente, cada miembro deberá "clonar" a partir de un repositorio remoto (veremos más tarde dónde se aloja este repositorio) el proyecto base, y luego tendrá que crear una "rama" para realizar sus cambios (la rama principal de cualquier proyecto en Git se

denomina “master”). Para escribir cambios en el repositorio remoto, se realiza un “push”, mientras que para obtener cambios remotos que no se hayan incluido en local se utiliza un “pull”. Cuando se realizan las modificaciones pertinentes, se deben preparar los cambios para ser registrados en Git, lo que se conoce como “stage”, y después se realiza un “commit” para guardarlos definitivamente. En una rama se pueden realizar modificaciones sin que afecten a los demás miembros del equipo, y cuando se hayan completado los cambios y se haya probado su correcto funcionamiento, una rama puede unirse a otra haciendo uso de un “merge” o de un proceso de “pull request” (aunque este concepto no pertenece al propio sistema Git, lo comentamos aquí ya que su uso es muy extendido en páginas de alojamiento de repositorios). Todo el proceso detallado que un miembro de nuestro equipo debe seguir para realizar sus cambios se explica más adelante. Cabe mencionar que Git es un sistema ideado para funcionar mediante comandos introducidos en un terminal, pero existen diferentes aplicaciones gráficas que facilitan el uso de las acciones más comunes que se pueden realizar. Entre ellas, destacan algunas como SourceTree o GitKraken. Dado que esta última es la única disponible para sistemas Linux y en nuestro equipo solamente utilizamos distribuciones del mismo, los miembros que prefieren utilizar una interfaz gráfica usan GitKraken, aunque para algunas tareas eventualmente se usa el terminal. Para obtener más información sobre Git, puede visitar su página oficial: <https://git-scm.com>.

Como se comentó antes, el proyecto se debe obtener de un repositorio remoto, Existen diversas páginas que permiten alojar repositorios de Git, siendo la más popular GitHub (<https://github.com>), que es la que utilizaremos ya que nuestro proyecto es un “fork” (es decir, una copia) de otro repositorio alojado en esta página. Tras registrarse, un usuario puede crear repositorios públicos o privados (estos últimos eran exclusivos de usuarios de pago o estudiantes verificados, aunque recientemente se permitió su uso, con ciertas restricciones, a cualquier usuario), y subir (mediante el “push” mencionado anteriormente) sus repositorios locales a la web, desde la cual otros usuarios pueden clonarlos si son públicos o tienen el permiso correspondiente.

El repositorio base, “pablotabares/decide” (<https://github.com/pablotabares/decide>), es un fork hecho por un miembro de Decide-Ortosia desde el repositorio “EGCETSII/decide”, que es a su vez un fork hecho por los profesores sobre el proyecto original “wadobo/decide”. Dentro de “pablotabares/decide”, existe una rama principal de cada módulo, con el nombre “ortosia-<módulo>”, y que en el caso de postprocesado se llama “ortosia-postproc” (<https://github.com/pablotabares/decide/tree/ortosia-postproc>). Esta rama sería equivalente a una rama “master” dentro de nuestro módulo. Además, cada miembro de nuestro equipo tiene una rama personal, llamada “ortosia-postproc-<uvus>”. En estas ramas será donde cada miembro realice sus cambios antes de unirlos con la rama “ortosia-postproc”.

Cuando se realiza un commit, se debe incluir un breve título del mismo. También es necesario añadir una descripción con algo más de detalle para explicar los motivos por los que se realiza el commit. Además, si el commit está relacionado con alguna incidencia que esté registrada como issue en GitHub, se deberá incluir una referencia a dicha issue en la descripción, para así poder seguir mejor la evolución de las issues. Un ejemplo de commit es

<https://github.com/pablotaabares/decide/commit/5679d640339c41284942154322f0a6d89f8e5676>, del cual se muestra una captura a continuación:

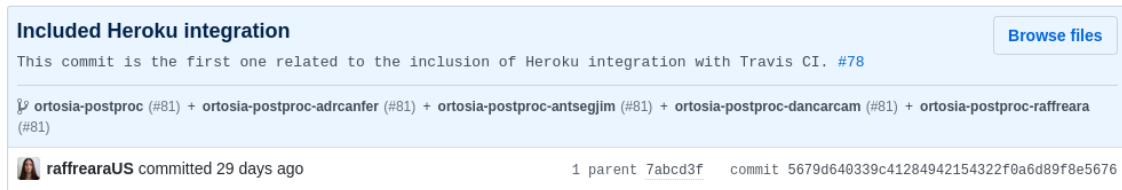


Imagen 11: Ejemplo de commit

Cuando un miembro quiere incluir sus cambios en la rama “ortosia-postproc”, debe primero llevarse los cambios de “ortosia-postproc” a su rama personal y solucionar los conflictos que puedan haber surgido. Una vez haya finalizado, puede realizar una pull request a “ortosia-postproc”. Una pull request, como se mencionó antes, no es un concepto del sistema Git y debe realizarse directamente desde la página de GitHub. En la descripción de la misma debe indicar las issues que se han solucionado, y es necesario añadir las etiquetas de “Postproc”, tipo (“bug”, “enhancement”, u “optimization”) y prioridad (“critical”, “high”, “medium” o “low”), así como la milestone para la que se pretendía incluir estos cambios. También se asignará la pull request al miembro que la ha abierto. Si los tests, ejecutados utilizando Travis CI, no reportan ningún error, el mismo usuario que abrió la pull request puede aceptarla. Una pull request de nuestro equipo es <https://github.com/pablotaabares/decide/pull/101>, mostrada en la siguiente captura de pantalla:

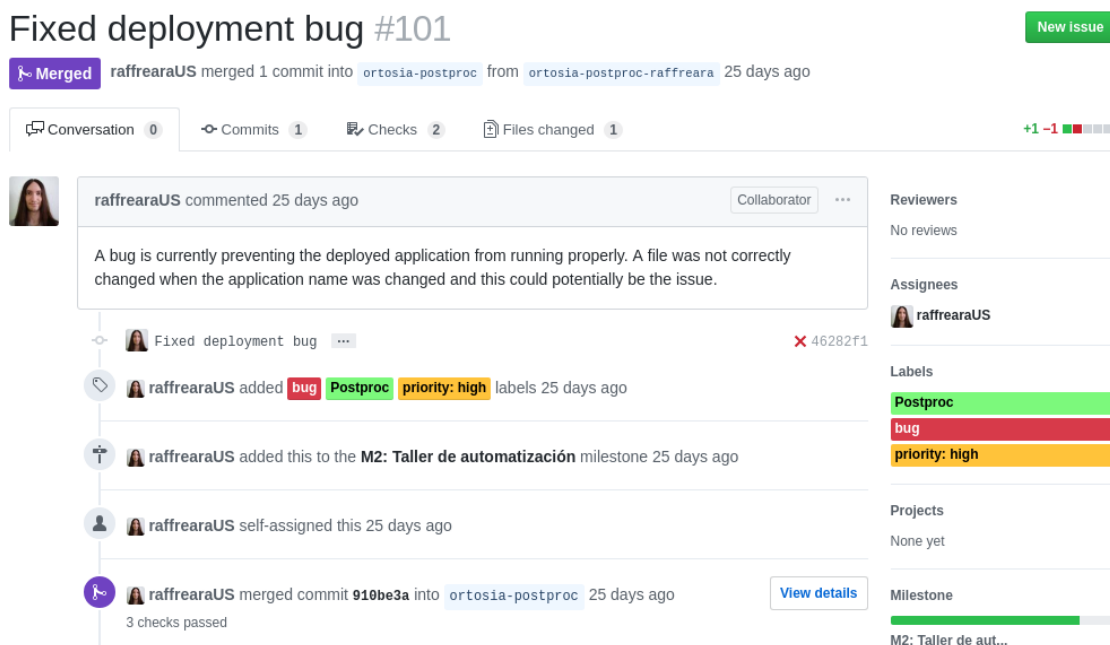


Imagen 12: Ejemplo de pull request

Cuando el equipo lo vea adecuado, los cambios de “ortosia-postproc” pueden unirse a la rama “ortosia-prepro”, que equivale a una rama “develop” dentro del repositorio. La rama “develop” es un concepto de GitFlow, una forma propuesta de uso de Git muy extendida. El objetivo de esta rama es reunir el conjunto de cambios finalizados y probados del resto de ramas antes de pasarlos a “master”, que debe contener únicamente versiones completamente funcionales (y preferiblemente desplegadas) del sistema. Un miembro debe primero llevarse los

cambios de “ortosia-prepro” a “ortosia-postproc” y solucionar los posibles conflictos. Posteriormente, debe crear una pull request y añadir como revisor a algún otro miembro de nuestro equipo. Si los tests se ejecutan correctamente y el miembro revisor da su aprobación, este puede aceptar la pull request. Una pull request realizada sobre “ortosia-prepro” es <https://github.com/pablotabares/decide/pull/54>.

A modo de lecciones aprendidas sobre nuestra gestión del código fuente, podemos mencionar que la decisión de utilizar ramas para cada módulo dentro de “pablotabares/decide” no fue muy buena, ya que esto permitía que cualquier miembro de cualquier módulo pudiera tener acceso a las ramas del resto de módulos y de sus correspondientes miembros, lo cual no es recomendable. Nuestro módulo propuso crear forks para cada módulo, de forma que el repositorio quedara más ordenado, pero el resto de subgrupos preferían utilizar ramas por lo que así se hizo finalmente.

9. Gestión de la construcción e integración continua

Para llevar a cabo la integración continua en nuestro proyecto, queremos disponer de algún servicio que nos permita ejecutar los tests de la aplicación de forma ajena a nuestros entornos locales, para poder asegurar que estos funcionan en cualquier equipo y que no dependen de nuestras configuraciones. Queremos, además, poder ejecutar los tests de todo el proyecto, no solo los de nuestro módulo, para poder verificar que nuestros cambios no afectan al resto de funcionalidades del sistema. Por lo tanto, lo que buscamos es poder, de alguna forma, que las pruebas del proyecto se ejecuten automáticamente cada vez que algún miembro del equipo realice un cambio y lo suba a GitHub.

Existen varias herramientas que dan soporte a esta tarea, entre los que destaca el sitio Travis CI, que, conectándose a GitHub, puede ejecutar una serie de comandos previamente definidos cada vez que un usuario realice un push sobre un repositorio o se acepte una pull request. Es importante mencionar que existen dos sitios diferentes de Travis, <https://travis-ci.org> (disponible exclusivamente para proyectos públicos en GitHub) y <https://travis-ci.com> (que puede utilizarse con proyectos públicos y privados). El proyecto “decide” está configurado para utilizar ambos sitios; sin embargo, algunas funcionalidades como el despliegue automático de la aplicación requerirían configuraciones diferentes para cada dominio. Por ello, en nuestro módulo nos centramos principalmente en <https://travis-ci.org>, mientras que en el otro sitio es posible que en algunos casos se produzcan problemas y se reporte algún error.

Para comenzar a utilizar Travis CI, solamente hay que iniciar sesión en su página con las mismas credenciales usadas en GitHub, y posteriormente se debe activar el soporte para Travis en los proyectos que se quieran integrar con esta herramienta. Eso es todo por parte de la web de Travis CI. Por otro lado, en la raíz del proyecto a integrar se debe añadir un archivo “.travis.yml”. Este fichero contendrá una serie de comandos y parámetros de configuración que determinarán cómo se ejecutarán los tests y demás funcionalidades adicionales en Travis.

En nuestro caso, el archivo antes mencionado es el siguiente, y comentaremos su contenido a continuación:

```

language: python
python:
  - "3.6.6"
install:
  - pip install -r requirements.txt
sudo: required
services:
  - docker
notifications:
  email: false
before_install:
  - cd docker
  - docker-compose up -d
  - cd ..
jobs:
  include:
    - stage: tests
      script:
        - cd docker
        - docker exec -ti decide_web ./manage.py test
        - cd ..
      if: branch = ortosia-postproc AND type = pull_request
    - stage: "Postproc tests"
      script:
        - cd docker
        - docker exec -ti decide_web ./manage.py test postproc
        - cd ..
      if: branch =~ ^ortosia-postproc-.{9}$
    - stage: deploy
      script: skip
      deploy: &heroku
        provider: heroku
        api_key:
          ortosia-postproc:
            secure: Pc+jLY1rIuKcyz0AB4s8vwtrJx7v13p/5/ha8Rmq7n
        app:
          ortosia-postproc: decide-ortosia-postproc
      if: branch = ortosia-postproc AND type = push

```

Imagen 13: Fichero .travis

Primero, se debe especificar el lenguaje de la aplicación y la versión del mismo que se desea usar, en este caso, Python 3.6.6. Luego, se indica el comando que se debe ejecutar para instalar las dependencias del proyecto. Además, se especifica que no deseamos recibir correos electrónicos cuando se produzcan cambios en Travis. Posteriormente se “dockeriza” la aplicación (es decir, se crea un contenedor de Docker), gracias al uso de archivos Dockerfile y de un ocker-compose. Después, en el archivo se indican diferentes “stages”. Una “stage” es un concepto de Travis que consiste en una serie de comandos agrupados que pueden ejecutarse en paralelo, de forma que luego se pueden combinar diferentes “stages” en un “job” para que se ejecuten secuencialmente y en base a unas condiciones específicas. En nuestro caso,

definimos tres etapas: una donde se ejecutan todos los tests de la aplicación (llamada “tests”), otra en la que solo se ejecutan los tests de nuestro módulo (llamada “postproc tests”), y otra en la que se despliega la aplicación en Heroku (llamada “deploy”, el despliegue automático se explica con más detalle en el siguiente apartado). La primera etapa solamente se ejecuta si Travis se ha disparado tras crear una pull request sobre nuestra rama “ortosia-postproc”. La segunda se lanza al realizar cualquier cambio en alguna de nuestras ramas personales (que, como se comentó anteriormente, tienen como nombre “ortosia-postproc-<uvus>”, de ahí la expresión regular utilizada). La tercera solo se ejecuta al hacer un push en “ortosia-postproc”, lo que ocurrirá cada vez que una pull request se acepte. Esto se hace de esta forma para eliminar redundancias a la hora de ejecutar los tests, ya que lanzarlos todos cada vez que se realiza cualquier cambio es una práctica poco recomendable puesto que consume recursos innecesarios en los servidores de Travis, por lo que es más viable ejecutar más o menos tests en función de lo que consideremos oportuno.

Anteriormente, se ejecutaba también el comando “sonar-scanner” que permitía detectar diferentes problemas de forma en el código. Sin embargo, este comando no funcionaba del todo bien por problemas de configuración y finalmente se optó por eliminarlo.

Un ejemplo de ejecución de tests en una pull request puede encontrarse en <https://travis-ci.org/pablotabares/decide/builds/485010365>, y a continuación se muestra una captura del mismo:

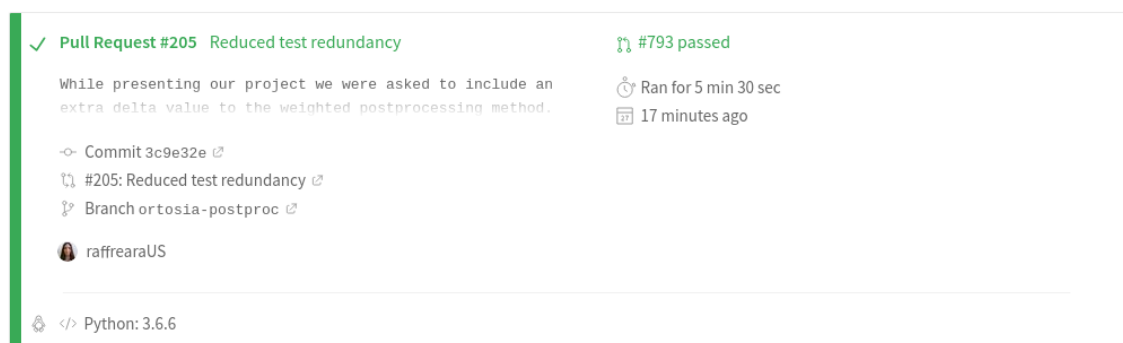


Imagen 14: Ejemplo de ejecución de tests en Travis

Todas las ejecuciones de Travis del repositorio “pablotabares/decide” se encuentran en <https://travis-ci.org/pablotabares/decide/builds>, mientras que un agrupamiento de las mismas por ramas está en <https://travis-ci.org/pablotabares/decide/branches>.

La principal lección aprendida de la gestión de la integración continua es la importancia de que exista la menor redundancia posible a la hora de ejecutar los tests de la aplicación, ya que no solo se alivia la carga de trabajo de los servidores de Travis CI, sino que además permite agilizar más el proceso completo de realización y unión de un cambio, porque se reducen los tiempos que un miembro del equipo debe esperar para verificar que sus cambios no reportan ningún error.

10. Gestión de liberaciones, despliegue y entregas

Para disponer de la aplicación desplegada en Internet de forma cómoda, necesitamos algún mecanismo que nos permita realizar el despliegue automáticamente cada vez que un cambio se haya unido y probado. Además, dado que no disponemos de los recursos y la

necesarios para alojar una página web en servidores propios, se hace muy interesante la opción de utilizar servidores en la nube que ofrezcan alojamiento, preferiblemente gratuito, de sitios web con bases de datos. Por lo tanto nuestro proceso general para desplegar la aplicación es que, tras aceptar una pull request sobre la rama “ortosia-postproc”, se deben subir automáticamente los últimos cambios a un servidor en la nube.

La herramienta principal usada en este caso es Heroku, (<https://www.heroku.com>), un proveedor de tipo PaaS (plataforma como servicio), que con una configuración muy simple permite desplegar aplicaciones web en diferentes lenguajes. Para comenzar a utilizar este servicio, hay que registrarse en su página e instalar la utilidad de línea de comando que ofrece. Una vez hecho, hay que ejecutar el comando “heroku login” e introducir las credenciales de nuestra cuenta. Posteriormente, mediante el comando “heroku ocker <nombreApp>”, crearemos la aplicación correspondiente en la nube, que tendrá como URL base “<nombreApp>.herokuapp.com”. Si no se proporciona un nombre en el comando, Heroku creará uno automáticamente. Además, es importante mencionar que una cuenta gratuita donde no se ha proporcionado ninguna tarjeta de crédito como forma de verificación solo puede tener cinco aplicaciones desplegadas simultáneamente. Luego se deben crear en la raíz del proyecto dos archivos, un “Procfile” donde se incluyen las instrucciones que necesita Heroku para iniciar la aplicación (en nuestro caso, migrar la base de datos y lanzar el servidor con la utilidad “gunicorn”), y un “runtime.txt” donde se especifica la versión de Python a utilizar (en este caso es Python 3.7.1). Tras crear estos archivos, basta con ejecutar el comando “git push heroku master” para subir el proyecto a la nube. Nótese que se utiliza Git para este fin; eso quiere decir que lo que se subirá a Heroku será todo lo que esté registrado en Git, es decir, que si tenemos cambios a los que no se les haya hecho “commit” estos no se verán reflejados en la aplicación desplegada.

Para realizar el despliegue automático, podemos valernos de nuevo de Travis CI. Este sitio permite configurar fácilmente el despliegue en diversos proveedores de alojamiento, siendo Heroku uno de ellos. Para conseguir la automatización deseada, simplemente tenemos que añadir al archivo “.travis.yml” una sección “deploy”, tal y como se puede ver en la captura de la sección anterior. En esa parte del archivo se especifica una clave encriptada, conseguida mediante el comando “travis encrypt (\$heroku auth:token)” (la utilidad “travis” es una “gem” de Ruby por lo que debe instalarse con “gem install travis”, se puede encontrar más información en <https://github.com/travis-ci/travis.rb>). También se indica que el nombre de nuestra aplicación desplegada es “decide-ortosia-postproc”, y se especifica que tanto la clave encriptada como la aplicación indicada solamente se usarán en la rama “ortosia-postproc”.

La URL de nuestra aplicación en la nube es <https://decide-ortosia-postproc.herokuapp.com>. Cada vez que una pull request sobre la rama “ortosia-postproc” se acepta, el commit resultante dispara la ejecución de Travis CI y la aplicación se despliega en la URL indicada, y está disponible públicamente. El usuario con permisos de administrador de Django tiene como nombre “adminname” y como contraseña “adminpasswd” (para conectarse a la aplicación de Heroku para crear un administrador, se debe ejecutar el comando “heroku run -a <nombreApp> “sh -c ‘cd decide && ocker manage.py createsuperuser’”). Actualmente se pueden crear votaciones y emitir votos en la página desplegada; sin embargo, no se puede realizar el recuento de los votos ya que esta acción requiere cierto tiempo para completarse y Heroku define un límite no ampliable de 30 segundos tras los cuales, si la aplicación no ha respondido, se devuelve un error y no se completa el recuento.

Para identificar las distintas versiones del sistema desplegado, utilizamos el propio versionado que proporciona Heroku, que tiene el formato “vX”. Cada vez que se realiza un nuevo despliegue, el número de versión aumenta una unidad.

Otra forma de entregar el proyecto es utilizar Docker (<https://www.docker.com>), que permite crear imágenes de aplicaciones que se pueden ejecutar en cualquier sistema, sin tener que instalar manualmente dependencias y de forma que no existan problemas relativos a que distintos usuarios tengan diferentes versiones que funcionan o no en sus equipos. Estas imágenes se lanzarán dentro de los denominados “contenedores”. Para usar Docker, simplemente hay que descargarlo e instalarlo; está disponible de forma nativa en sistemas Linux, mientras que en el resto es necesario utilizar algún sistema de virtualización.

Para generar una imagen, se debe crear un archivo, comúnmente llamado “Dockerfile”, aunque puede tener el nombre que se desee siempre que luego se especifique donde sea oportuno. Para utilizar Decide se necesitan tres contenedores: uno que llamaremos “decide_web” que contiene la aplicación de Django así, otro llamado “decide_nginx” que contiene un servidor Nginx que sirve de proxy para la aplicación web, y otro llamado “decide_db” que es la base de datos PostgreSQL. Los tres contenedores se lanzarán simultáneamente utilizando la herramienta “docker-compose”, que permite, a partir de un archivo que veremos posteriormente, definir cómo deben lanzarse las imágenes y cómo se conectarán entre ellas.

Para crear la imagen que usaremos en el contenedor “decide_web”, necesitamos un Dockerfile que contenga las instrucciones relativas a su construcción:

```
from python:alpine

RUN apk add --no-cache git postgresql-dev gcc libc-dev
RUN apk add --no-cache gcc g++ make libffi-dev python3-dev build-base

RUN pip install gunicorn
RUN pip install psycopg2
RUN pip install ipdb
RUN pip install ipython

WORKDIR /app

ADD . .
RUN pip install -r requirements.txt

WORKDIR /app/decide

# local settings.py
ADD ../docker/docker-settings.py local_settings.py

RUN ./manage.py collectstatic
```

Imagen 15: DockerFile

Aquí estamos definiendo que la imagen base será “python:alpine” (Alpine es una distribución de Linux muy ligera), luego instalamos los paquetes necesarios, añadimos todo el contenido de nuestro proyecto a la imagen e instalamos las dependencias de Python correspondientes, y finalmente añadimos un archivo de configuración de Django. Con la estructura actual de Decide, este archivo se llama “Dockerfile” y reside en una carpeta llamada “docker”. Para construir la imagen, nos movemos en un terminal hasta la carpeta “docker” y ejecutamos “docker build .. -f Dockerfile --tag <nombreImagen>”, comando que lanzará su creación con el nombre que hayamos especificado.

Para crear la imagen del contenedor “decide_nginx”, el Dockerfile será:

```
from nginx:alpine
```

```
ADD docker-nginx.conf /etc/nginx/conf.d/default.conf
```

Imagen 16: Fragmento de código para crear la imagen del contenedor

Aquí solamente definimos que la imagen base será “nginx:alpine” y añadimos un fichero de configuración. Dado que este archivo también se encuentra en la carpeta “docker” y se llama “Dockerfile-nginx”, para crear la imagen usaremos “docker build . -f Dockerfile-nginx --tag <nombreImagen>”.

Para el contenedor “decide_db” no necesitamos crear ninguna imagen, ya que podemos usar directamente la imagen que proporciona PostgreSQL en Docker Hub.

En nuestro equipo hemos generado las imágenes de los contenedores “decide_web” y “decide_nginx” y las hemos publicado en Docker Hub para facilitar su obtención y uso posterior. Para publicar imágenes hay que registrarse en Docker Hub, ejecutar el comando “docker login” para iniciar sesión y ejecutar “docker push <usuario>/<imagen>”. Las imágenes creadas están disponibles en https://hub.docker.com/r/raffrearaus/ortosia-postproc_web y https://hub.docker.com/r/raffrearaus/ortosia-postproc_nginx para los contenedores “decide_web” y “decide_nginx” respectivamente. Dado que solo tenemos una versión de estas imágenes, no encontramos necesario establecer una política de versionado para las mismas, por lo que nos limitaremos a identificarlas con la etiqueta “latest” que Docker añade automáticamente.

Como se mencionó antes, para lanzar los contenedores simultáneamente usamos la utilidad “docker-compose”, la cual puede requerir una instalación aparte en algunas distribuciones de Linux. Tenemos que definir un archivo “docker-compose.yml” con las instrucciones correspondientes a cómo se ejecutarán las imágenes. Mostramos a continuación el archivo que haría falta para utilizar Decide con las imágenes publicadas por nuestro equipo, en texto en lugar de en una captura para que se pueda copiar y pegar si es necesario:

```
docker : '3.4'
```

```
services:
```

```
  db:
```

```
    restart: always
```

```
    container_name: decide_db
```

```
    image: postgres:alpine
```



```

    volumes:
      - db:/var/lib/postgresql/data
    networks:
      - decide
  web:
    restart: always
    container_name: decide_web
    image: raffrearaus/ortosia-postproc_web
    command: ash -c " ocker manage.py migrate && gunicorn -w 5 decide.wsgi
-timeout=500 -b 0.0.0.0:5000"
    expose:
      - "5000"
    volumes:
      - static:/app/static
    depends_on:
      - db
    networks:
      - decide
  nginx:
    restart: always
    container_name: decide_nginx
    image: raffrearaus/ortosia-postproc_nginx
    volumes:
      - static:/app/static
    ports:
      - "8000:80"
    depends_on:
      - web
    networks:
      - decide


volumes:
  static:
    name: decide_static
  db:
    name: decide_db

networks:
  decide:
    driver: bridge
  ipam:
    driver: default
    config:
      - subnet: 10.5.0.0/16

```

Aquí se definen los nombres de los contenedores, las imágenes que se usarán en cada uno de ellos, ocker comandos adicionales, una serie de volúmenes de almacenamiento y una subred en la que estarán disponibles. Este fichero se debe guardar en cualquier sitio como "ocker-compose.yaml" o "ocker-compose.yml", y posteriormente, en un terminal que esté en el directorio donde se encuentre el archivo, hay que ejecutar el comando "ocker-compose up -d". Esto descargará todas las imágenes necesarias y lanzará todos los contenedores con los nombres, comandos y demás parámetros de configuración que se especificaron en el archivo. Además, dado que en todos los contenedores se indicó la opción "restart: always", estos se iniciarán automáticamente cada vez que Docker se reinicie. Se puede detener y borrar los contenedores si se ejecuta "ocker-compose down". Para verificar el correcto funcionamiento de la aplicación utilizando Docker, se puede encontrar una guía en la wiki del repositorio original de Decide: <https://github.com/wadobo/decide/wiki/Como-functiona-Decide>.

En cuanto a la licencia, utilizamos GNU Affero General Public License, en su versión 3, dado que es la que los creadores originales del proyecto “decide” incluyeron en su repositorio. Esta licencia permite el uso tanto público como privado del código, pero obliga a que, si se publica una versión actualizada del sistema, todo el código de las modificaciones realizadas debe hacerse público.

 wadobo/decide is licensed under the
GNU Affero General Public License v3.0

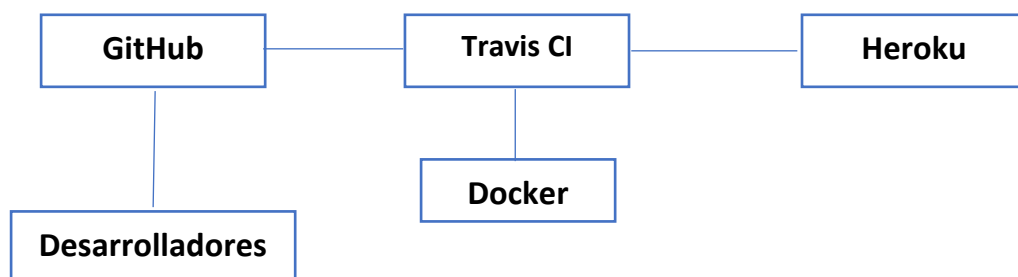
Permissions of this strongest copyleft license are conditioned on making available complete source code of licensed works and modifications, which include larger works using a licensed work, under the same license. Copyright and license notices must be preserved. Contributors provide an express grant of patent rights. When a modified version is used to provide a service over a network, the complete source code of the modified version must be made available.

Permissions	Limitations
✓ Commercial use	✗ Liability
✓ Modification	✗ Warranty
✓ Distribution	
✓ Patent use	
✓ Private use	

Imagen 17: Licencia de Decide

11. Mapa de herramientas

Este proyecto ha sido elaborado con diferentes herramientas para desplegar y conseguir una buena gestión del código. Para ello, hemos utilizado Git para el control de versiones y GitHub para el alojamiento de repositorios, los cuales nos han servido para gestionar nuestro código viendo los cambios en archivos y a coordinarnos entre todos los grupos. Además, GitHub nos ha ayudado a la gestión de incidencias. Para la integración continua hemos utilizado Travis CI que se trata de un sistema distribuido libre integrado con GitHub, lo cual nos permite probar que nuestro código es correcto y que no hemos creado errores en otros módulos. También podemos automatizar la ejecución y construcción de imágenes con Docker. Por último, Travis CI nos ha ayudado también a desplegar nuestro código en un servidor web si no hay errores, en este caso se trata de Heroku.



12. Ejercicio de propuesta de cambio

En este apartado vamos a analizar una propuesta de cambio real en nuestro proyecto. Se detecta un fallo en un método de post-procesado:

- Creamos una issue en la cual ponemos como título *Failure in Hondt test* y además daremos una breve explicación del error y en qué lugar del código se ha producido.
- Asignamos esa issue al responsable y le añadimos las etiquetas **bug**, **priority**, **postproc**.

Postproc - Failure in Hondt test #36

Edit New issue

Closed AntSegJim opened this issue on 30 Nov 2018 · 1 comment

AntSegJim commented on 30 Nov 2018

I detected a failure in hondt when I executed it.

AntSegJim assigned adrcanfer on 30 Nov 2018

adrcanfer added **bug** **Postproc** labels on 1 Dec 2018

adrcanfer changed the title **Failure in Hondt test** to **Postproc - Failure in Hondt test** on 1 Dec 2018

adrcanfer commented on 1 Dec 2018

This issue was fixed in commit `bdf845b`, but another issue was accidentally referenced in its message.

adrcanfer closed this on 3 Dec 2018

adrcanfer added the **priority: high** label 23 days ago

adrcanfer added this to To do in **Postproc** via automation 23 days ago

Assignees: adrcanfer

Labels: **Postproc**, **bug**, **priority: high**

Projects: Done in Postproc

Milestone: M2: Taller de aut...

Notifications: Unsubscribe

You're receiving notifications because you authored the thread.

2 participants

Imagen 18: Bug detectado

- Antes de comenzar a arreglar el fallo, obtenemos todos los cambios realizados por nuestros compañeros. Miramos el repositorio en el cual nos encontramos con **'git branch'**, y si no estamos en nuestra rama de trabajo, ejecutamos la instrucción **'git checkout branch-name'**. Debemos asegurarnos que nuestra rama esta actualizada, en caso de que no lo esté ejecutamos **'git pull'**.
- Ahora que ya tenemos nuestra rama actualizada comenzamos a trabajar, abrimos nuestro entorno de desarrollo y nos vamos hasta el lugar donde está el error. En este caso, como es un método de post-procesado nos iríamos a `decide > decide > postproc`.
- Buscamos el error. Si es necesario ejecutamos los tests con la instrucción **'python3 manage.py test postproc'**. Una vez detectado, cambiamos el código que está mal y ejecutamos de nuevo los tests, si estos no fallan lanzamos el servidor en local con **'python3 manage.py runserver'**. En algunos casos, también hacemos pruebas con la aplicación Postman para poder hacer peticiones a nuestra API y probar solo nuestro módulo.
- Cuando este esté arreglado, volvemos a la consola de comando y usamos **'git status'** para ver los cambios pendientes de seguimiento. Para resolver esto, escribimos en la consola **'git add nombre-del-fichero'** y luego hacemos **'git commit'** y **'git push origin HEAD:nombre-de-la-rama'** para enviar los cambios al repositorio.
- Estos cambios se encuentran en nuestra rama, por lo que debemos actualizar nuestra rama con los cambios existentes en la rama master mediante el comando **'git merge nombre-rama-master'** y resolver los conflictos si hubiera. Una vez resuelto habría que introducir los cambios en nuestra rama con **'git push origin HEAD:nombre-de-la-rama'**.
- Estos cambios se han subido a nuestra rama por lo que, en la interfaz online de GitHub pulsamos **'New pull request'**, seleccionamos de que rama a que rama queremos pasar los datos, escribimos un comentario y solicitamos la petición.
- Por último, una vez se ejecuten los tests automáticos y salgan bien, se pasa a introducir los datos en la rama master.

13. Integración con otros módulos

Tras implementar una serie de métodos, se ha solicitado a los distintos módulos que integren nuestra aportación. El módulo de votación implementó el método weight. Su tarea consistía en que al crear una pregunta se solicitara al usuario un peso por cada opción. Debido a esto, y a otros cambios solicitados por otros módulos, el formato del JSON que utiliza la cabina de votación se vio afectado, de modo que no se puede votar. Pese a la existencia de dos grupos encargados de la parte de cabina, ninguno ha estado trabajando en cabina versión web, si no en una aplicación móvil y en bots para votar desde Telegram o Slack. Es por este motivo por el cual los cambios implementados desde votación no los podemos añadir a nuestra rama, aunque se encuentran en la rama master. También podemos ver como algunas issues creadas por nosotros para que los distintos módulos afectados integren nuestros métodos, han sido asignadas pero no completadas.

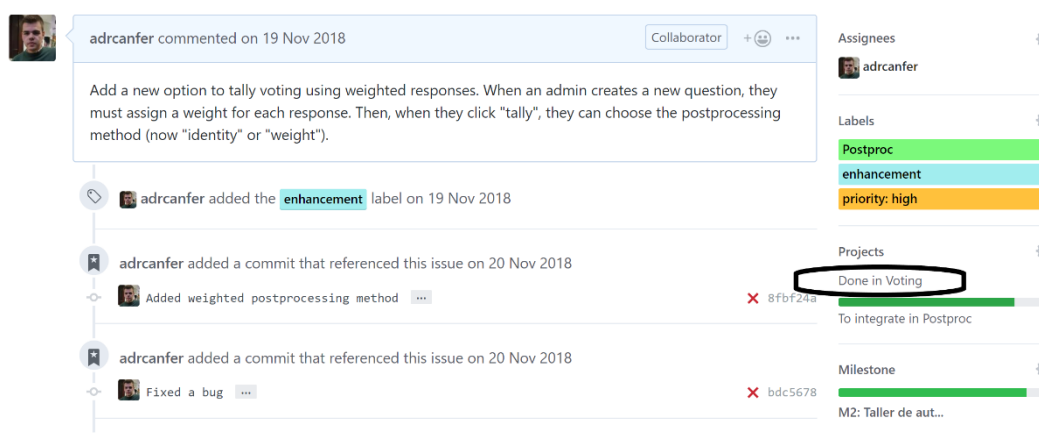


Imagen 19: Ejemplo de actividad con otro módulo

14. Conclusiones y trabajo futuro

Conclusiones: Ha sido un trabajo muy interesante y entretenido, dado que se ha realizado entre un gran número de alumnos y ha supuesto un gran reto para todos tratar de conseguir nuestros objetivos y conjuntamente los objetivos de los demás. Ser capaces de ponernos de acuerdo e intentar lograr un objetivo común nos ha ayudado a comprender la idea de empresa, el trabajo en grupos grandes y el compañerismo. Es muy importante la organización y la comunicación, puesto que si no la hay, cada uno hace lo que desea sin tener en cuenta que otros están haciendo otra cosa y no se pueden integrar. Hay que trabajar en equipo.

A futuro: Implementar otro método similar a Sainte-Laguë al cual se le denomina Sainte-Laguë modificado ya que introduce un pequeño cambio en la fórmula original y podría ser interesante comprobar que con ese pequeño cambio pueden salir resultados, en algunos casos, muy diferentes.

Sería una buena idea llegar en un futuro a la integración de todas las partes del proyecto. Por lo tanto, otra propuesta es que todos los métodos implementados en post-procesado sean utilizados para obtener diferentes tipos de recuentos y visualizados para que los pueda ver un usuario final.