



# 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: x
- Segura Jiménez, Antonio: 5

### Enlaces de interés

- Repositorio: <https://github.com/pablotabares/decide/tree/ortosia-postproc>
- Heroku: <http://decide-ortosia.herokuapp.com/> (Usuario: adminname Contraseña:adminpasswd)

## Contenido

1. Resumen.....	2
2. Introducción y contexto .....	2
3. Descripción del sistema.....	3
4. Planificación del proyecto .....	4
5. Entorno de desarrollo .....	4
6. Gestión de incidencias.....	5
6.1. Gestión de incidencias internas .....	5
6.2. Gestión de incidencias externas.....	6
7. Gestión de depuración .....	6
8. Gestión del código fuente .....	7
9. Gestión de la construcción e integración continua.....	9
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

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.

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.

## 4. Planificación del proyecto

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 `multiquuestion`.

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.

## 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
- Django 2.0.0
- PostgreSQL 11.1

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 permitía utilizar el software con el que se sintiesen más cómodos a la hora de trabajar. Finalmente se acabó utilizando PyCharm y Visual Code Studio.

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`** con acceso como superusuario y en la carpeta donde se encuentre el archivo `requirements.txt`.
2. Crear los usuarios en la base de datos postgresQL. Para ello, se deben lanzar las siguientes sentencias SQL:
  - **`create user decide with password 'decide'`**
  - **`create database decide owner decide`**
3. 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 HOSTS será 'localhost' y cuyo nombre, usuario y contraseña será 'decide'.

4. Ejecutar la primera migración ejecutando: **'python manage.py migrate'**
5. 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, deberá llevar un ejemplo de la estructura del JSON que se usará en dicha funcionalidad.
- Deberá llevar etiquetas para poder clasificar las issues. Deberá 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.
- Se deberá asignar 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 notificará vía mensajería instantánea.
- Toda issue deberá indicar el milestone donde se presentará la funcionalidad. Toda issue deberá estar cerrada y finalizada antes de la fecha del milestone. De no haberse podido finalizar a tiempo, se deberá cambiar el milestone establecido en la issue.
- Toda issue deberá tener asignado el proyecto creado por el grupo en GitHub, llamado 'Postproc'. Durante el transcurso de la tarea, se deberá 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 deberá incluir en su descripción una referencia a la issue que se esté tratando en ese momento.

### 6.1. 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.

## 6.2. 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.

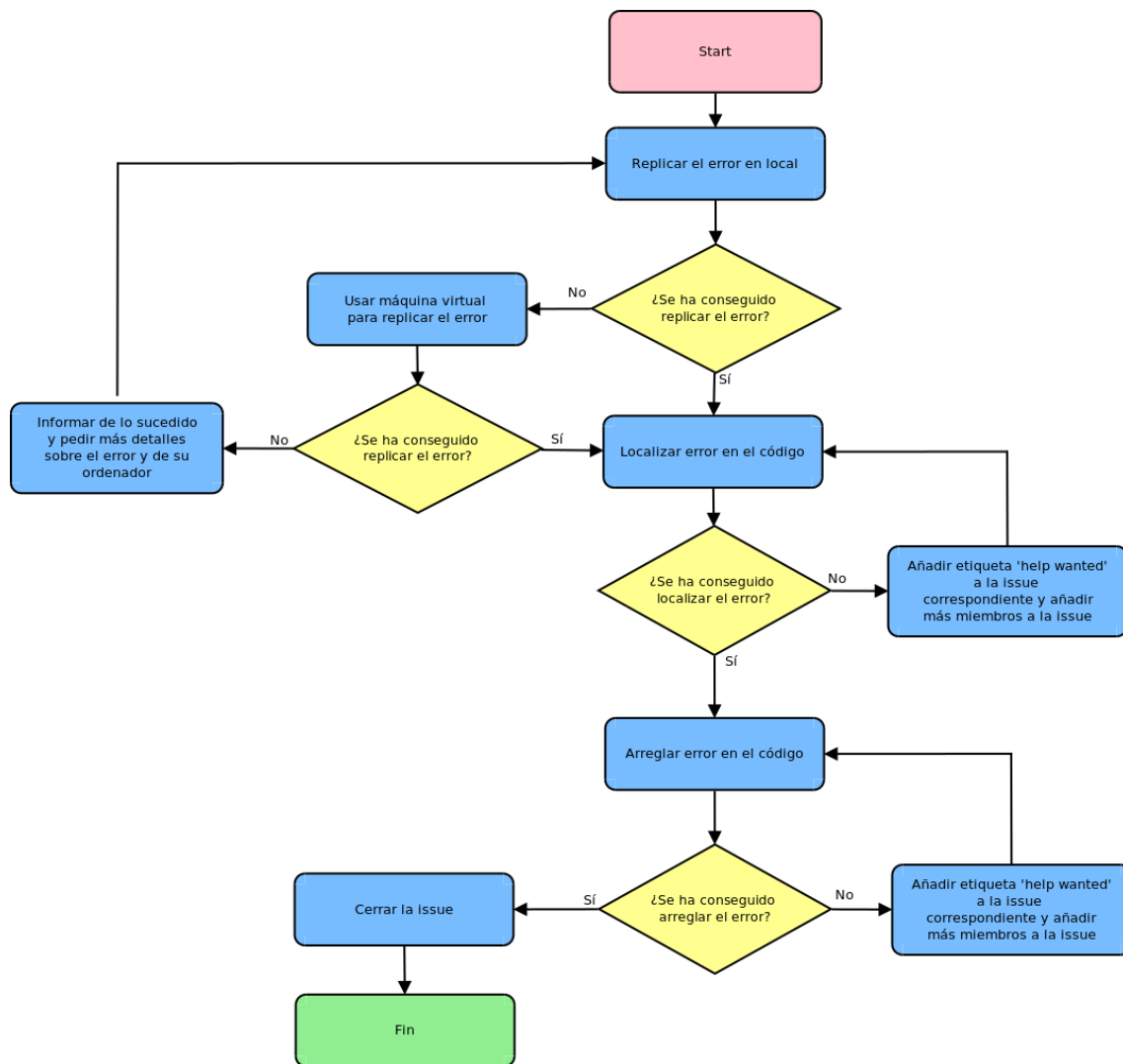
Una vez informados, serán ellos los que se organicen de acuerdo a su forma de actuar, siendo responsables de asignar a uno o varios miembros para trabajar en la issue. Será esta persona la encargada de asignarse en la tarea correspondiente.

## 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 1: 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. 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, utilizamos el sistema de control de versiones Git. Cada miembro puede usar libremente la terminal o una interfaz gráfica, según le resulte más cómodo en cada momento. En nuestro caso, la única interfaz que usamos es GitKraken.



El repositorio base, “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”. Estos repositorios están alojados en el sitio web GitHub. Dentro de “pablotabares/decide”, existe una rama llamada “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.



*Imagen 2: 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”. En la descripción de la misma debe indicar las issues que se han solucionado, y es necesario añadir las etiquetas de “Postproc”, tipo (“bug” o “enhancement”) y prioridad (“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.


# Fixed deployment bug #101

[New Issue](#)

**Merged** raffrearaUS merged 1 commit into ortosia-postproc from ortosia-postproc-raffreara 25 days ago

Conversation 0 Commits 1 Checks 2 Files changed 1

+1 -1



**raffrearaUS** commented 25 days ago

Collaborator

...

A bug is currently preventing the deployed application from running properly. A file was not correctly changed when the application name was changed and this could potentially be the issue.

Fixed deployment bug

46282f1

raffrearaUS added

bug

Postproc

priority: high

labels 25 days ago

raffrearaUS added this to the

M2: Taller de automatización

milestone 25 days ago

raffrearaUS self-assigned this 25 days ago

raffrearaUS merged commit 910be3a into ortosia-postproc 25 days ago

View details

3 checks passed

Reviewers

No reviews

Assignees

raffrearaUS

Labels

Postproc

bug

priority: high

Projects

None yet

Milestone

M2: Taller de aut...

Imagen 3: Ejemplo de issue

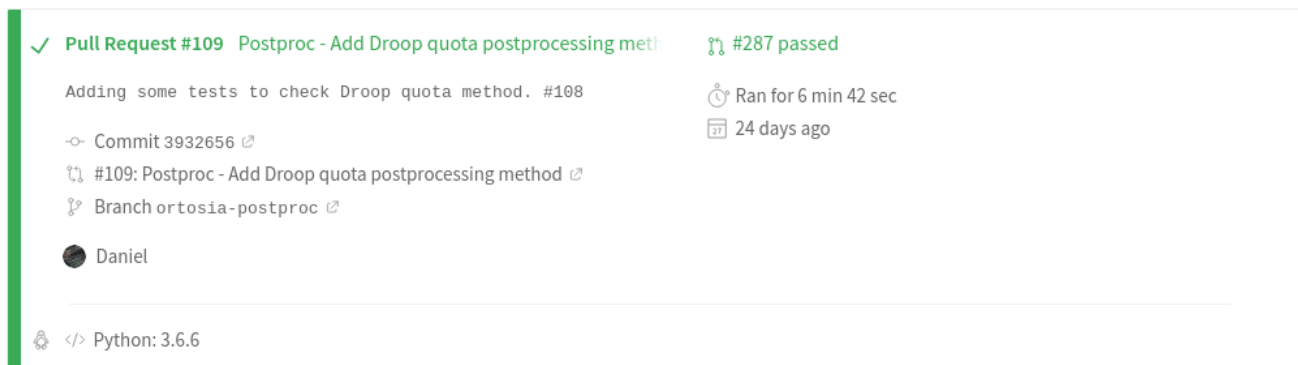
Cuando el equipo lo vea adecuado, los cambios de “ortosia-postproc” pueden unirse a la rama “ortosia-prepro”, que equivale a una rama “develop” dentro del repositorio. Para ello, un miembro debe primero llevarse los cambios de “ortosia-prepro” a “ortosia-postproc” y solucionar los posibles conflictos. 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.

## 9. Gestión de la construcción e integración continua

Para llevar a cabo la integración continua en nuestro proyecto, utilizamos 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.

En nuestro caso, disponemos de un archivo “.travis.yml” que contiene los comandos que se ejecutarán cada vez que se suban cambios al repositorio. La aplicación se “dockeriza”, gracias al uso de archivos Dockerfile y de un docker-compose. Posteriormente, los tests del proyecto se ejecutan. En caso de que no se reporte ningún error y que además los cambios se hayan hecho sobre la rama “ortosia-postproc”, se realizará un despliegue automático a Heroku. Esto se describirá con mayor detalle en el siguiente apartado.

Anteriormente, se ejecutaba también “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.



*Imagen 4: Travis CI*

## 10. Gestión de liberaciones, despliegue y entregas

Para realizar el despliegue de la aplicación, disponemos de una página alojada en Heroku, cuya URL 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, si los tests se ejecutan correctamente (lo que debería ocurrir ya que en caso de no hacerlo la pull request no se habría aceptado), 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”. 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.

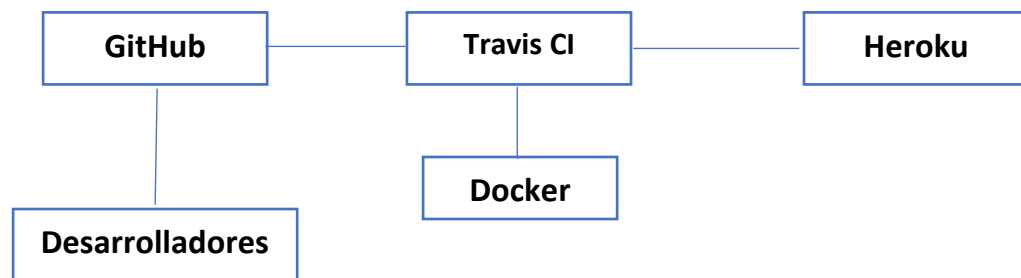
Para identificar las distintas versiones del sistema desplegado, utilizamos el propio versionado que proporciona Heroku, que tiene el formato “vX”. Cada vez que se realiza un nuevo despliegue, el número de versión aumenta una unidad.

Como forma alternativa de entrega del proyecto, se proporcionan dos imágenes de Docker, nombradas “raffrearaus/ortosia-postproc\_web” y “raffrearaus/ortosia-postproc\_nginx”. Estas imágenes están disponibles en Docker Hub. Para utilizarlas, se incluye un archivo “docker-compose.yaml” que, además de crear un contenedor con una base de datos de PostgreSQL, descarga y ejecuta las dos imágenes antes mencionadas, de forma que se puede probar el sistema en un entorno local. 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.

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.

## 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

**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

2 participants

Imagen 5: 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. 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.