

Decide-Part-Pícaro-Votaciones

Grupo 2

Curso escolar: 2020/2021

Asignatura: Evolución y gestión de la configuración

Milestone en el que se entrega la documentación: M3

Miembros del equipo: escala de 1 al 10 con el esfuerzo hecho en el proyecto (10 mayor implicación, 1 menor implicación)

- Caballero Domínguez, Ángel:10
- Correa López, Florentina: 10
- Granja Naranjo, Javier: 10
- Morante Caraballo, Abel: 10
- Rodríguez Holgado, Gonzalo: 10
- Toro Valle, Daniel: 3

Enlaces de interés:

Repositorio de código: <https://github.com/pabfrasan/decide>

Sistema desplegado: <https://picaro-decide.herokuapp.com/admin/>

Índice

1. Resumen ejecutivo	3
1.1 Indicadores del proyecto	4
1.2 Integración con otros equipos.....	5
1.3 Descripción del sistema	5
1.4 Visión global del proceso de desarrollo	8
1.5 Entorno de desarrollo	10
1.6 Gestión de incidencias	15
1.6.1 Gestión interna	15
1.6.2 Gestión externa	20
1.7 Gestión del código fuente	22
1.8 Gestión de la construcción e integración continua	24
1.9 Gestión de liberaciones, despliegue y entregas	25
1.9.1 Proceso definido para las liberaciones	25
1.9.1.1 Licencia de software para su proyecto	25
1.9.2 Proceso definido para el despliegue	25
1.9.2.1 Despliegue en local	26
1.9.2.2 Despliegue con Docker	27
1.9.2.3 Cómo desplegar en Docker	28
1.9.2.4 Despliegue con Heroku.....	28
1.9.2.5 Integración con Travis.	29
1.9.3 Proceso definido para las entregas	32
1.9.4 Política de nombrado e identificación de los entregables	32
1.10 Ejercicio de propuesta de cambio	33
1.11 Conclusiones y trabajo futuro	35

1. Resumen ejecutivo

El objetivo de este proyecto consiste en la extensión de las funcionalidades del sistema heredado decide. Formando parte esto del currículum de la asignatura EGC del curso 2020/2021. La finalidad de esta tarea reside en la captación de capacidades de gestión de un proyecto además de la familiarización con Django, Heroku, GitHub, Docker, Travis entre otras tecnologías.

En concreto podríamos especificar que el Alcance del proyecto está constituido por las implementaciones siguientes:

- Creación de una plantilla para preguntas del tipo Sí / No.
- Creación de preguntas que permitan una ordenación de las respuestas por preferencia.
- Poder introducir una fecha de finalización de la votación a la hora de crearla, tras la cual no se podrá participar en dicha votación.
- Almacenar los resultados de una votación en un fichero del sistema.
- Creación de un parámetro “link” en cada votación para permitir el acceso a dicha votación desde una dirección personalizada.
- Votación con varias preguntas.

Durante el desarrollo de este proyecto hemos tenido que lidiar con el abandono de uno de los miembros (Daniel Toro Valle), provocando esto que la última de las implementaciones propuestas originalmente no haya llegado a completarse. No solo esto, sino que sus implementaciones tempranas e incompletas han supuesto horas de corrección para solucionar los conflictos creados. A pesar de esto valoramos su participación en la redacción de este documento, aligerando así el peso de su decisión

1.1 Indicadores del proyecto

En esta tabla se muestra el trabajo realizado por parte de los miembros del equipo siguiendo la guía de desarrollo del Documento del Proyecto.

Miembro del equipo	Horas	Commits	LoC	Test	Issues	Incremento
Caballero Domínguez, Ángel	62	15	234	6	12	Creación de una plantilla para preguntas del tipo Sí/No.
Correa López, Florentina	65	14	277	8	14	Creación de preguntas que permitan una ordenación de las respuestas por preferencia.
Granja Naranjo, Javier	65	13	262	6	9	Poder introducir una fecha de finalización de la votación a la hora de crearla, tras la cual no se podrá participar en dicha votación.
Morante Caraballo, Abel	65	14	327	6	11	Almacenar los resultados en un fichero del sistema.
Rodríguez Holgado, Gonzalo	56	11	215	4	9	Posibilidad de personalización de la url de una votación.
Toro Valle, Daniel	-	-	-	-	-	Posibilidad de creación de votaciones con varias preguntas.
TOTAL	313	67	1315	30	55	

Para la realización de esta tabla cada miembro del equipo ha realizado un seguimiento particular de su desempeño en el proyecto utilizando herramientas como GitHub.

1.2 Integración con otros equipos

Los grupos que componen **decide-pícaro** son **decide-pícaro-visualización**, **decide-pícaro-autenticación** y **decide-pícaro-votación**, debido a que decide-pícaro-visualización quería integrarse con nosotros desde un primer momento y posteriormente se unió pícaro-autenticación tiempo después por mediación del profesorado.

A pesar de la integración inesperada, no ha existido ningún problema y ha sido una buena experiencia con ambos grupos en general por nuestra parte. Todas las integraciones han sido de forma no presencial lo cual no ha supuesto un problema de comunicación.

1.3 Descripción del sistema

Decide es un sistema de voto electrónico educativo que permite el voto electrónico asegurando la privacidad del voto mediante algoritmos criptográficos. Además, es un sistema modular compuesto por distintos subsistemas que contienen la funcionalidad y los cuales pueden ser extendidos para añadir más funcionalidades.

Este sistema de software libre (código <https://github.com/wadobo/decide>) aporta una implementación base a partir de la cual se puede extender con cualquier tecnología usando una API web básica.

Gracias a la modularidad de Decide, no dependemos de un lenguaje, podemos distribuir la carga en diferentes máquinas, es más sencillo a la hora de abordar una tarea y todos estos módulos están conectados mediante API (requests).

Los módulos son los siguientes:

- Base: Tiene funcionalidades comunes y es similar a una librería
- Auth: Autenticación de votantes que actualmente funciona con usuario/contraseña
- Census: Funciona como un censo que puede guardar quien vota y quién ha votado, de cara a cada votación.
- Creación de una plantilla para preguntas del tipo Sí / No: Se ha añadido un checkbox a la vista de creación de una pregunta. Para que funcione, solamente hay que marcarlo, sin tener que completar ninguna opción ni su número asociado, y guardamos la pregunta. Cuando accedemos a la pregunta ya creada, podemos comprobar que se han creado automáticamente las opciones de sí y no. Así mismo, en el caso de que se añadieran una o varias opciones que no sean las de si o no a una pregunta que está marcada como de sí o no, estas no se guardarán al actualizar la pregunta.
- Creación de preguntas que permitan una ordenación de las respuestas por preferencia: Se ha añadido un atributo al modelo que se llama pref_number que

permite que posteriormente en cabina se indique ese número para cada opción.y estando en el censo de esa votación, realizar su voto y emitirlo.

- Poder introducir una fecha de finalización de la votación a la hora de crearla, tras la cual no se podrá participar en dicha votación: Para este apartado se ha añadido una fecha de finalización opcional en la creación de votación, además de una restricción para no poder introducir una fecha de finalización pasada, con esta fecha tras la cual ya no se puede votar, al intentar votar en una votación con la fecha de finalización cumplida no podrá realizar la acción debido a que esta ya está desautorizada.
- Almacenar los resultados de una votación en un fichero del sistema: Para una votación que ya haya finalizado, una vez hecho el recuento de votos de ésta, se podrá guardar (en el servidor) seleccionándola y haciendo click en el botón de Save (aparece en el desplegable junto a las opciones de Start, Stop y Tally). Tras hacerlo, si accedemos a esa votación, encontraremos un campo File en el cual aparecerá un link con el nombre del fichero que se ha guardado. Si hacemos click nos abrirá dicho documento y con un click derecho en la página podremos descargar ese archivo en nuestro equipo.
- Creación de un parámetro “link” en cada votación para permitir el acceso a dicha votación desde una dirección personalizada. Se ha añadido el parámetro Link a la creación de votaciones, además de añadir la ruta /booth/url/<link> para acceder a la misma, sin eliminar la forma anterior a través de la id y sin obligar a rellenar dicho nuevo campo, para evitar conflictos con los tests anteriores.os votos recibidos se almacenen en una base de datos guardando una relación entre el votante y el voto. Sin embargo, estos votos estarán cifrados, para asegurar que no se puede descubrir el contenido del mismo, pero sabiendo quién ha votado y no.
- Postproc: Aquí se obtienen los votos de cada votación y encarga de traducir esa lista de números a un resultado coherente con el tipo de votación.
- Visualizer: En este subsistema, se muestran los resultados obtenidos por el post-procesado de una forma que sea atractiva y sobre todo entendible por alguna persona, de modo que sepa cómo ha ido la votación.

En este apartado se describe las funcionalidades implementadas en decide-pícaro-votación, estas son las funcionalidades propuestas:

- Creación de una plantilla para preguntas del tipo Sí / No: Se ha añadido un checkbox a la vista de creación de una pregunta. Para que funcione, solamente hay que marcarlo, sin tener que completar ninguna opción ni su número asociado, y guardamos la pregunta. Cuando accedemos a la pregunta ya creada, podemos comprobar que se han creado automáticamente las opciones de sí y no. Así mismo, en el caso de que se añadieran una o varias opciones que no sean las de si o no a una pregunta que está marcada como de sí o no, estas no se guardarán al actualizar la pregunta.

- Creación de preguntas que permitan una ordenación de las respuestas por preferencia: Se ha añadido un atributo al modelo que se llama `pref_number` que permite que posteriormente en cabina se indique ese número para cada opción.
- Poder introducir una fecha de finalización de la votación a la hora de crearla, tras la cual no se podrá participar en dicha votación: Para este apartado se ha añadido una fecha de finalización opcional en la creación de votación, además de una restricción para no poder introducir una fecha de finalización pasada, con esta fecha tras la cual ya no se puede votar, al intentar votar en una votación con la fecha de finalización cumplida no podrá realizar la acción debido a que esta ya está desautorizada.
- Almacenar los resultados de una votación en un fichero del sistema: Para una votación que ya haya finalizado, una vez hecho el recuento de votos de ésta, se podrá guardar (en el servidor) seleccionándola y haciendo click en el botón de Save (aparece en el desplegable junto a las opciones de Start, Stop y Tally). Tras hacerlo, si accedemos a esa votación, encontraremos un campo File en el cual aparecerá un link con el nombre del fichero que se ha guardado. Si hacemos click nos abrirá dicho documento y con un click derecho en la página podremos descargar ese archivo en nuestro equipo.
- Personalizar la url de una votación: Creación de un parámetro "link" en cada votación para permitir el acceso a dicha votación desde una dirección personalizada. Se ha añadido el parámetro Link a la creación de votaciones, además de añadir la ruta `/booth/url/<link>` para acceder a la misma, sin eliminar la forma anterior a través de la id y sin obligar a rellenar dicho nuevo campo, para evitar conflictos con los tests anteriores.

En un principio, estaba prevista la implementación de una funcionalidad más, pero debido a que un compañero tuvo que abandonar el equipo, esta funcionalidad no se implementó y se eliminó su avance para evitar algunos fallos que provocaba. Esta funcionalidad es la siguiente:

- Posibilidad de crear un voting con varias preguntas: Se podrá crear una votación que incluya varias preguntas, de modo que cuando se acceda a esa votación en cabina, se pueda enviar una respuesta para cada pregunta de la votación.

1.4 Visión global del proceso de desarrollo

Para desarrollar el proyecto, a nivel interno del equipo de votación, hemos partido el trabajo en partes lo más equitativas que hemos considerado, y, a continuación, las hemos asignado al azar entre todos los miembros del equipo. Hemos procedido de esta manera porque ningún miembro tenía especial interés en hacer una parte concreta, y, además, nos ahorra posibles disputas. Para repartir las tareas utilizamos herramientas de sorteo online como <https://echaloasuerte.com> o <https://www.sortea2.com/>.

Una vez que cada miembro tenía claro cuál era su parte a desarrollar, se ha trabajado de manera colaborativa, de modo que hemos intercambiado ideas y conocimientos para facilitar el trabajo de los demás, y, en caso de que algún miembro tuviera problemas con su parte, ser ayudado por el resto para seguir avanzando hacia el objetivo común.

Para trabajar de manera conjunta, hemos usado las siguientes herramientas:

- WhatsApp: Mediante un grupo formado por todos los integrantes del equipo, se ha llevado a cabo el seguimiento y control del proyecto.
- Discord: A través de un servidor de esta herramienta, hemos realizado una serie de llamadas para solucionar las incidencias que nos han surgido durante el desarrollo del proyecto, consultar dudas al resto del equipo o intercambiar cualquier información relacionada con el proyecto.

A nivel general de *Pícaro* (conjunto de equipos de votación, visualización y autenticación), nos hemos comunicado de igual manera, a través de un grupo de WhatsApp con todos los miembros de todos los equipos para propuestas y consultas breves y seguimiento y control general del proyecto y de un servidor de Discord para reuniones más largas. Así hemos evitado conflictos o desacuerdos debidos a problemas de comunicación.

En el grupo de WhatsApp han participado todos los miembros activamente, pero en el servidor de Discord, cada equipo proponía a algunos miembros para estas reuniones. Esto se decidió así para no hacer una llamada con muchas personas y evitar el exceso de ruido o de interrupciones. Tras finalizar las llamadas, cada representante que había participado en ésta, comunicaba por escrito al resto del equipo las decisiones tomadas, bien por el propio servidor de Discord, o bien por el grupo de WhatsApp. En nuestro caso (equipo de votación), elegíamos a los representantes por disponibilidad horaria de éstos.

Además de emplear las herramientas hasta ahora nombradas, también hemos trabajado con:

- Travis CI: Para la integración continua (ver sección Gestión de la construcción e integración continua).
- Codacy: Para la gestión de la calidad.

- Heroku: Para desplegar nuestra aplicación. Puedes acceder a través de <https://picaro-decide.herokuapp.com>

Para terminar, exponemos un ejemplo de un cambio propuesto para el sistema y cómo hemos abordado todo el ciclo hasta tener ese cambio en producción:

El cambio en cuestión es el de la creación de una plantilla para preguntas del tipo Sí/No.

1. Lo primero es asignar esta tarea a un miembro del equipo, y para ello se echa a suertes como se explicó anteriormente.
2. Una vez que está asignada a un miembro, se crea una issue en el tablero kanban de github de nuestro grupo (visualización), con las etiquetas correspondientes, la persona a la que está asignada, y, opcionalmente, una descripción de dicha tarea.
3. A continuación, se desarrolla dicha tarea y pasa a la parte de “In progress”.
4. Una vez desarrollada esta tarea, se realiza un commit con el formato acordado (ver sección “Gestión del código fuente”) haciendo referencia al issue correspondiente.
5. Después, se hace push a la rama correspondiente.
6. Llegados a este punto, el encargado de desarrollar la tarea, la muestra al resto de miembros del equipo de Votación, y si estos la ven correcta, se pasa este issue a “Done”.
7. Por último (una vez que hay varios issues como “Done”), cuando el grupo de Votación considere suficiente los cambios realizados en su rama principal (developVotación), se pondrá en contacto con los otros subsistemas de Pícaro para establecer un día y hora en el que realizar el merge a la rama principal (master), como lo establece la política de merges preestablecida por los grupos de Pícaro y se lleva a cabo toda la integración continua de pruebas en Travis y el despliegue en Heroku.

1.5 Entorno de desarrollo

Para el desarrollo del proyecto Decide-Part-Pícaro-Votaciones ha habido diversos entornos de desarrollo utilizados por los integrantes del equipo:

- Ángel Caballero Domínguez ha utilizado Windows 10 + máquina virtual Oracle Virtual Box versión 6.1.2 con Ubuntu 20.04 y su editor de código fuente ha sido Visual Studio 1.52.1.
- Florentina Correa López ha utilizado Windows 10 + máquina virtual Oracle Virtual Box versión 6.1.2 r135662 con Ubuntu 20.04 y como editor de código Visual Studio Code versión 1.52.1.
- Javier Granja Naranjo tiene en su portátil Ubuntu 20.04.1 LTS en una partición de su disco y para el código ha utilizado Visual Studio Code 1.52.1.
- Abel Morante Caraballo tiene de sistema operativo Windows 10 con Oracle Virtual Box versión 6.0.24 con Ubuntu 20.04, para el código ha escrito en Visual Studio Code 1.52.1.
- Gonzalo Rodríguez Holgado ha utilizado Windows 10 + máquina virtual Oracle Virtual Box versión 6.1.2 r135662 con Ubuntu 20.04 y como editor de código Visual Studio Code versión 1.52.1.
- ~~Daniel Toro Valle~~

Si quiere utilizar Oracle Virtual Box versión X con Ubuntu 20.04 siga estos pasos, si va a utilizar Ubuntu como sistema operativo salte este apartado.

Para instalar Virtual Box puede seguir el manual de la asignatura EGC en este [enlace](#) y posteriormente si quiere añadirle una interfaz gráfica con estos comandos:

```
#-----  
# Actualizamos toda nuestra máquina virtual  
#-----  
sudo apt get update  
sudo apt get upgrade  
  
#-----  
# Instalamos una herramienta que nos permitirá autoconfigurar el sistema  
#-----  
  
sudo apt install taskel  
  
#-----  
# Instalamos un gestor de display  
#-----
```

```

sudo apt install slim

#-----
# Instalamos un entorno de ventanas
#-----
# En el video uso xfce, por el balanceo de consumo y comodidad que ofrece,
pero podeis instalar cualquier otro dependiendo de que máquina tengais y
cuanta memoria podais dedicar a la VM. Por ejemplo
# sudo tasksel install ubuntu-mate-core # Para instalar mate
# sudo tasksel install lubuntu-core # para instalar lightweight gui
# sudo tasksel install ubuntu-desktop # Esto es para gnome

sudo tasksel xubuntu-core

#-----
# Arrancamos un entorno de ventanas
#-----

sudo service slim start

#-----
# Instalamos, por ejemplo, visual studio code desde la snap store o bien
desde su página web abriendo el # navegador (tenemos chromium instalado desde
la práctica)
#-----
#sudo snap install --classic code

```

También hay un video tutorial [aquí](#) a partir del minuto 10:10 aproximadamente.

Todo listo con Ubuntu, procederemos a descargar Venv y PostgreSQL con los comandos en consola:

```

sudo apt-get update
sudo apt-get install python3-venv
sudo apt-get install postgresql

```

Crearemos un espacio virtual escribiendo `python3 -m venv <myenvname>`

Cambiaremos al espacio virtual `source <myenvname>/bin/activate`

Clonamos el siguiente repositorio de Decide <https://github.com/EGCETSII/decide.git>

Cambiamos al directorio donde hayamos clonado y buscamos dentro de la carpeta el archivo requirements.txt, debe quedar con la siguiente información:

```
Django==2.0
pycryptodome==3.6.6
django-rest-framework==3.7.7
django-cors-headers==2.1.0
requests==2.18.4
django-filter==1.1.0
psycopg2==2.8.4
django-rest-swagger==2.2.0
coverage==4.5.2
django-nose==1.4.6
jsonnet==0.12.1
```

Instalaremos estas dependencias con

```
pip install -r requirements.txt
```

```
pip install wheel
```

Tras esto tendremos que crearnos nuestra base de datos con postgres:

```
sudo su - postgres
```

```
psql -c "create user decide with password 'decide'"
```

```
psql -c "create database decide owner decide"
```

```
exit
```

Ahora tendremos que hacer una copia al archivo local_settings_example.py y llamarlo local_settings.py y cambiar la parte de DATABASES cambiando el NAME, USER y añadiendo el campo PASSWORD, cambie también la BASEURL a la dirección de su localhost. La parte cambiada le quedará así:

```

APIS = {
    'authentication': 'http://localhost:8000',
    'base': 'http://localhost:8000',
    'booth': 'http://localhost:8000',
    'census': 'http://localhost:8000',
    'mixnet': 'http://localhost:8000',
    'postproc': 'http://localhost:8000',
    'store': 'http://localhost:8000',
    'visualizer': 'http://localhost:8000',
    'voting': 'http://localhost:8000',
}

BASEURL = 'http://localhost:8000'

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'decidedb3',
        'USER': 'decide3',
        'PASSWORD': 'decide',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}

```

Realizamos la primera migración para preparar la base de datos que utilizaremos:

```
./manage.py migrate
```

Cree el super usuario con el comando e introduzca los campos que le pida:

```
./manage.py createsuperuser
```

Por último, ya podremos ejecutar el módulos o módulos seleccionados en la configuración de la siguiente manera:

```
./manage.py runserver
```

Si quiere usar el navegador de su sistema operativo ejecute la siguiente (8000 indica el puerto que indicó en local_settings.py.)

```
./manage.py runserver 0:8000
```

Ya puede probar la página de administración yendo a la BASEURL que indicó anteriormente.

Para crear pruebas, tenemos que editar los permisos de nuestro usuario de la base de datos:

```
sudo su - postgres
```

```
psql -c "ALTER USER tuusuario CREATEDB"
```

Para ejecutar pruebas vaya al directorio de decide general si quiere correr todos los test o vaya a la carpeta del subsistema deseado. Allí:

```
./manage.py test
```

Si está en Virtual Box, escriba `./manage.py test --exe`

Para probar la cobertura de nuestras pruebas usaremos la aplicación coverage:

```
pip install coverage
```

Para lanzar el analisis de cobertura:

```
coverage run ./manage.py test -v 2
```

Añadiremos la opción "--source ." para sólo analizar nuestro código de decide pero no las bibliotecas incluidas.

```
coverage html
```

Finalmente podemos ver el contenido del html abriendo "decide/htmlcov/index.html"

Para pruebas de navegación puede usar la herramienta Selenium:

```
pip install selenium
```

Instalamos chromiun el driver para chromium que se encargará de reproducir el código dentro del navegador

```
sudo apt install chromium-browser chromium-chromedriver
```

También tendrá que descargar geckodriver para que funcione Selenium. El \$ indica una línea nueva.

```
$wget
https://github.com/mozilla/geckodriver/releases/download/v0.27.0/geckodrive
r-v0.27.0-linux64.tar.gz
$tar -xzf geckodriver-v0.27.0-linux64.tar.gz
$chmod +x geckodriver
$sudo cp geckodriver /usr/bin/
$rm geckodriver-v0.27.0-linux64.tar.gz
```

Código del script de Python para probar el funcionamiento de Selenium:

```
from selenium import webdriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC

options = webdriver.FirefoxOptions()
options.headless = True
driver = webdriver.Firefox(options=options)
driver.get("https://www.google.com/")
print('Title: %s' % driver.title)
driver.quit()
```

Para ejecutar los test de estrés utilizando locust, necesitaremos tener instalado locust:

```
Pip install locust
```

Tras esto, necesitaremos un archivo de configuración de locust de lo que vayamos a ejecutar llamado locustfile.py. Después, ejecutamos el siguiente comando (para probar Authentication por ejemplo):

```
Locust Authentication
```


Esto abrirá un servidor, y nos dirá en qué puerto. Si accedemos nos dejará personalizar el número de peticiones, cómo crecen en el tiempo y finalmente iniciar el test.

1.6 Gestión de incidencias

La gestión de incidencias se divide en dos, una interna la cual es la de nuestro equipo Decide-Part-Pícaro-Votaciones y otra externa en las que participan los equipos Decide-Part-Pícaro-Authenticación y Decide-Part-Pícaro-Visualización.

1.6.1 Gestión interna

Cualquier miembro del equipo Decide-Part-Pícaro-Votaciones puede hacer una incidencia, para ello hay que registrarla en el tablero de projects de nuestro repositorio de Decide-Part-Pícaro.

pabfrasan / decide  Watch 0 Star 0 Fork 376

forked from EGCETSII/decide

[Code](#) [Issues 47](#) [Pull requests 3](#) [ZenHub](#) [Actions](#) [Projects 3](#) [Wiki](#) [Security](#) [Insights](#)

is:open New project

3 Open 0 Closed Sort


Pícaro-Authenticación Updated 1 hour ago	Proyecto de decide del grupo Pícaro-Authenticación	...
Pícaro-Visualización Updated 1 minute ago	Proyecto de decide del grupo Pícaro-Visualización	...
Pícaro-Votación Updated 1 hour ago	Proyecto de decide del equipo Pícaro-Votación	...

También se usará las Labels o etiquetas para diferenciar que subsistema es dentro de Pícaro y que el tipo sea desarrollo.



A continuación, se muestra una [issue](#):



Creación de preguntas con respuesta por preferencia #32



Closed flocorlop opened this issue 9 days ago · 0 comments Edit New issue



 flocorlop commented 9 days ago · edited Collaborator ...



implementar un número de preferencia en la opción de una pregunta



  flocorlop created this issue from a note in **Pícaro-Votación (Done)** 9 days ago


  flocorlop self-assigned this 9 days ago



  flocorlop closed this 9 days ago


  flocorlop added the **enhancement** label 9 days ago


  javgranar added the **Votacion** label 7 days ago

Assignees   flocorlop

Labels  **Votacion** **enhancement**

Projects   **Pícaro-Votación**
Done


Milestone  No milestone

Linked pull requests  Successfully merging a pull request may close this issue.

Un ejemplo textual sería:

Título: Implementación de la fecha de finalización de una votación

Cuerpo: Poder introducir una fecha de finalización de la votación a la hora de crearla, tras la cual no se podrá participar en dicha votación.

 [javgranar](#) created this issue from a note in [Pícaro-Votación](#) (**To do**)

 [javgranar](#) moved this from **To do** to **In progress** in [Pícaro-Votación](#)

 [javgranar](#) self-assigned this

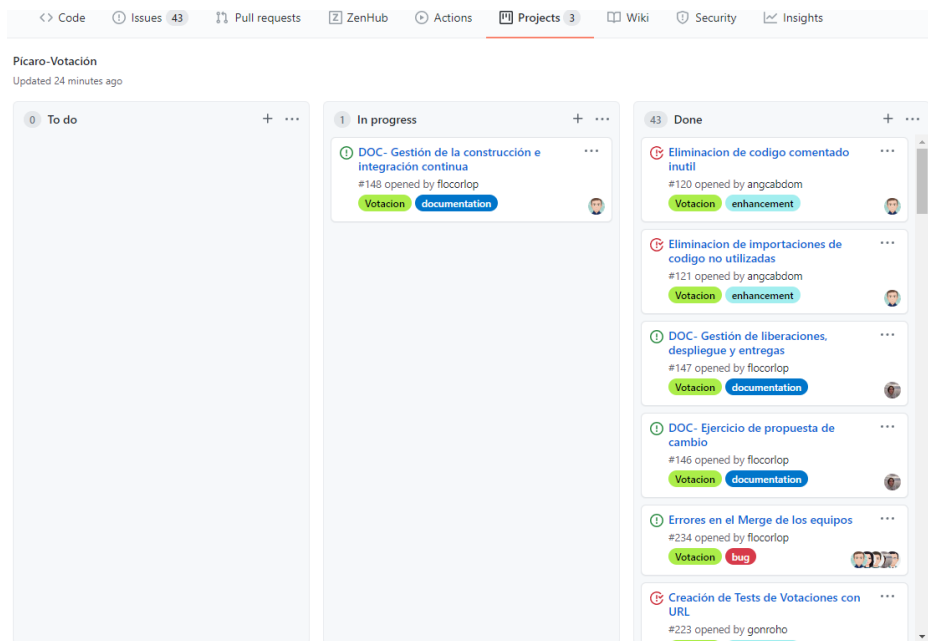
 [javgranar](#) added **enhancement** **Votacion**

 [javgranar](#) added a commit that referenced this issue

 [javgranar](#) moved this from **In progress** to **Done** in [Pícaro-Votación](#)

Una tarea siempre sigue el proceso de estar en la columna To Do, luego In Progress cuando se empieza a desarrollar, y por último se coloca en la columna Done dando por finalizada la tarea como último paso:

Ya en la pestaña, hay varias columnas:



El tablero contiene tres columnas: To Do para las tareas que hay que hacer aún no empezadas, In progress para las que están en progreso y Done para las ya realizadas.

Para crear una incidencia:

Primero tendrá que hacer click en el icono + de la columna de To Do ya que primero se crea una nota. En los tres puntos de la nota se procede a convertir a Issue y hay que introducir un título y cuerpo.

Para una incidencia de tipo error:

Si procede y es posible, incluir la traza del error, o el log del servidor.

Etiqueta bug obligatoria.

The screenshot shows a GitHub issue thread for the repository 'Picaro-Votación'. The issue is titled 'Arreglado fallo al guardar una votacion en local sin tally' and is marked as a 'bug' and 'Votacion'. The thread shows a series of actions by the user 'abemorcadc':

- Commented 4 days ago: 'Al intentar guardar una votación en local sin haberla iniciado, terminado ni hecho el tally, da error debido a que al guardar se usan la fecha de inicio y fin entre otros.'
- Created this issue from a note in 'Picaro-Votación (To do)' 4 days ago.
- Self-assigned this 4 days ago.
- Added 'bug' and 'Votacion' labels 4 days ago.
- Moved this from 'To do' to 'In progress' in 'Picaro-Votación' 4 days ago.
- Added a commit that referenced this issue 4 days ago: '<fix> Arreglado fallo al guardar una votacion en local sin tally' (commit hash 503275f).
- Moved this from 'In progress' to 'Done' in 'Picaro-Votación' 3 days ago.
- Closed this 3 days ago.

The right sidebar shows the following metadata:

- Assignees:** abemorcadc
- Labels:** Votacion, bug
- Projects:** Picaro-Votación (Done)
- Milestone:** No milestone
- Linked pull requests:** None yet
- Notifications:** A 'Subscribe' button is present.

O una tarea en forma textual:

Título: Arreglar la cabina

Cuerpo: Hay que arreglar la cabina porque no muestra las opciones. Creo que se debe a la opcion de multiples preguntas y no está implementado en booth.html

[flocorlop](#) created this issue from a note in [Pícaro-Votación](#) (**To do**)

[flocorlop](#) assigned [flocorlop](#) and [dantorval-us](#)

[flocorlop](#) added [bug](#) [Votacion](#) labels

[flocorlop](#) moved this from **To do** to **Done** in [Pícaro-Votación](#)

A continuación, una tarea de documentación:

Su etiqueta es la de documentación y también es asignada a una persona. Cada integrante del grupo le corresponden 2 tareas de documentación.

DOC- Diario del equipo #144

[Edit](#)[New issue](#)[Open](#)

flocorlop opened this issue yesterday · 0 comments



flocorlop commented yesterday

Collaborator



Resumen de total de reuniones empleadas en el equipo.
Actas de acuerdos de las reuniones en las que se tomaron decisiones importantes

DOC- Diario del equipo has no dependencies

[Add dependency](#)flocorlop created this issue from a note in [Pícaro-Votación \(To do\)](#) yesterdayflocorlop added [documentation](#) [Votacion](#) labels yesterdayflocorlop assigned [angcabdom](#) yesterday

Pipelines

[EGC-Votación](#)
[New Issues](#)[+ Show 2 more...](#)

Manage this Issue's pipeline per Workspace.
Change its pipeline to show progress in other Workspaces. [See related Workspaces](#)

Assignees

[angcabdom](#)

Labels

[Votacion](#) [documentation](#)

De forma textual:

Título: DOC- Ejercicio de propuesta de cambio

Cuerpo: Se presentará un ejercicio con una propuesta concreta de cambio en la que a partir de un cambio que se requiera, se expliquen paso por paso (incluyendo comandos y uso de herramientas) lo que hay que hacer para realizar dicho cambio. Debe ser un ejercicio ilustrativo de todo el proceso de evolución y gestión de la configuración del proyecto.

[flocorlop](#) created this issue from a note in [Pícaro-Votación \(To do\)](#)[flocorlop](#) added [documentation](#) [Votacion](#) labels[flocorlop](#) assigned [gonroho](#)

Leyenda de etiquetas disponibles:

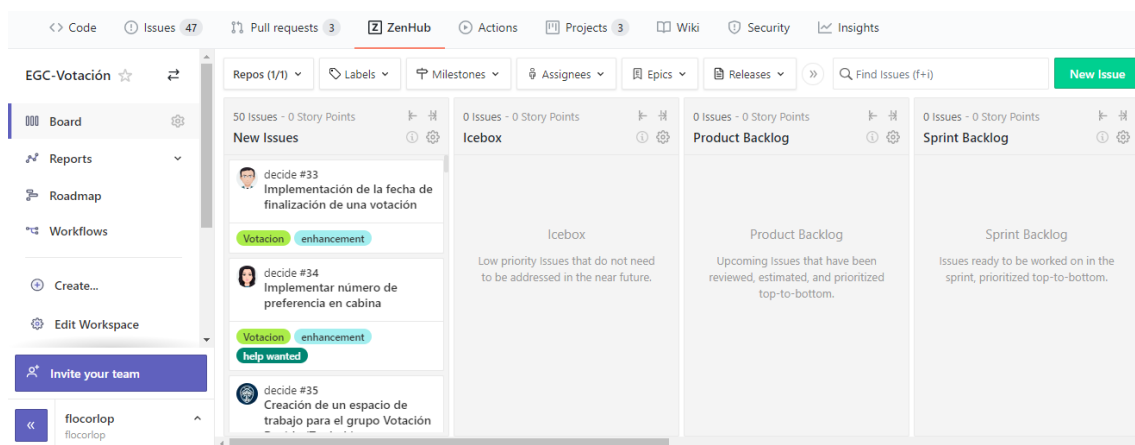
- [autenticacion](#) Indica que todo lo marcado es referente al subsistema de autenticación
- [bug](#) Algo no funciona
- [documentation](#) Improvements or additions to documentation
- [duplicate](#) This issue or pull request already exists
- [enhancement](#) New feature or request
- [good first issue](#) Good for newcomers
- [help wanted](#) Anyone is welcome to open a pull request to fix this issue.
- [Incidencia intergrupala](#) Incidencia que afecta a dos o más grupos que componen el proyecto
- [question](#) Further information is requested
- [rechazado](#) Rechazado del proyecto, no se continuará
- [Visualizer](#) Indica que todo lo marcado con esta label ha sido reportado por el grupo de visualización
- [Votacion](#) Indica que todo lo marcado con esta label ha sido reportado por el grupo de votación
- [wontfix](#) This will not be worked on

1.6.2 Gestión externa

Ya que existen dos equipos más en el proyecto Decide-Part-Pícaro nos tenemos que integrar con ellos.

Para una incidencia externa se sigue un proceso parecido al de la gestión interna. Primero hay que tener ZenHub instalado como plugin de GitHub.

Ya en la pestaña, hay varias columnas:



New Issues: Nuevas tareas por completar.

Icebox: Tareas de baja prioridad que no necesitan ser tratadas

Product Backlog: Lista de tareas del proyecto en general priorizadas

In Progress: Tareas que se encuentran realizándose.

Review Q/A: Tareas que necesitan una revisión y aceptación.

Closed: Tarea cerrada.

Como es una integración de equipos, aquí las incidencias que afecten a los demás llevarán la etiqueta Incidencia intergrupala y la etiqueta del subsistema.

La plantilla de incidencias externas contiene:

- Creación de labels concretas para cada subgrupo.

- Creación de label que indique incidencia grupal.

- Título: No más de 10 palabras. Al leerlo, debe obtenerse una idea clara del problema

- Cuerpo: Explicación detallada. Debe contener enlaces a commits que puedan clarificar lo sucedido.

Un ejemplo de incidencia intergrupala ha sido esta issue:

Introducir código en el modulo voting #107

Open pabfrasan opened this issue 3 days ago · 0 comments

pabfrasan commented 3 days ago

Estoy implementando el código para un bot de telegram. Este bot notificará por un grupo de Telegram cuando se inician, se cierran y cuando se hace recuento de las votaciones. También notifica de los resultados de las votaciones.

Abro esta incidencia para ver si tengo el visto bueno del grupo de votaciones para comenzar estos cambios, ya que deberé añadir código en model.py y en admin.py del modulo de voting.

Espero vuestra respuesta.

Introducir código en el modulo voting has no dependencies

Add dependency

pabfrasan added Incidencia intergrupala Votacion labels 3 days ago

Pipelines

- EGC-Votación
- New Issues
- Show 2 more...

Manage this Issue's pipeline per Workspace. Change its pipeline to show progress in other Workspaces. See related Workspaces

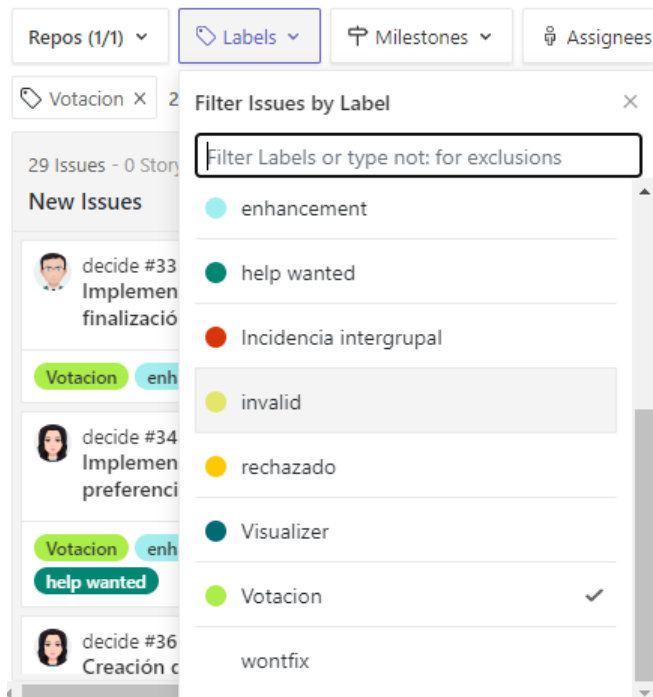
Assignees

No one—assign yourself

Labels

- Incidencia intergrupala
- Votacion

Debido a que en la columna de New Issues aparecen todas las tareas también de los demás equipos hay que filtrar por etiqueta de equipo Votación.



Todo esto se acordó en una reunión con dos integrantes de cada equipo.

1.7 Gestión del código fuente

Para la gestión del código fuente, vamos a distinguir entre gestión interna (equipo de votación) y gestión general (conjunto de equipos de Pícaro).

A nivel general, el código fuente con el que trabaja el equipo se encuentra en el repositorio de GitHub <https://github.com/pabfrasan/decide.git> y es accesible por todos los grupos que forman decide-pícaro. El código que se encuentra ahí es el de toda la plataforma de Decide y se obtuvo de su repositorio original <https://github.com/EGCETSII/decide.git>.

Nuestro repositorio se encuentra dividido en varias ramas según el subsistema de la plataforma en el que se va a trabajar, y que permiten a los subgrupos trabajar de forma separada sobre su subsistema. Para el equipo de votación, denotaremos tres importantes:

- master: la rama más importante porque es aquella en la que convergen las ramas de todos los subgrupos y es desplegada, y por tanto accesible mediante Heroku. Es fundamental que esta rama no posea ningún fallo pues es la que establece si la plataforma funciona como es debido en todo momento o no. Es accesible mediante la URL <https://github.com/pabfrasan/decide/tree/master> (al estar definida como la rama default del repositorio también es accesible por la URL <https://github.com/pabfrasan/decide>).

- **developVotación:** esta es la rama dedicada al trabajo de nuestro equipo en concreto. Sirve para unificar las aportaciones de los miembros de nuestro subgrupo y para preparar un único bloque de modificaciones funcionales para ser unificadas a master. Esta rama solo debe ser modificada por los miembros de nuestro subgrupo, puesto que los otros equipos poseen su propia rama que cumple la misma función. Es accesible mediante la URL <https://github.com/pabfrasan/decide/tree/developVotacion>.
- **picaroVotacionNombre:** (donde nombre corresponde al nombre de un desarrollador del equipo): esta es una rama personal y reservada a cada programador. Todos los miembros del equipo poseen una y son libres de aportar su código en ella. En principio, están reservadas al desarrollador al que hacen referencia, pero otro puede utilizarla si así lo permite su propietario; por ejemplo, en el caso de que el propietario necesite ayuda de otro programador para solucionar un fallo propio de su rama. Estas ramas son accesibles mediante las URL <https://github.com/pabfrasan/decide/tree/picaroVotacionNombre>, sustituyendo "Nombre" por Abel, Angel, Flor o Gonzalo.

A nivel interno del equipo de votación, decidimos que los cambios introducidos en las ramas personales, únicamente se unieran a la rama developVotacion cuando no se detecten fallos aparentes, y una vez unidos, volver a comprobar que todo funciona como se espera en esa rama. En el caso de detectar algún problema, lo solucionará la persona que lo haya introducido, pero también será ayudada por el resto del equipo.

Para gestionar los commits, también a interno del equipo de votación, creamos una plantilla para que tuvieran un formato definido. Esta plantilla tiene la siguiente estructura:

<tipo> titulo

Descripción

<Tarea: #>

Siendo:

- **<tipo>:** Tipo de commit. Este tipo podría ser:
 - feat: para funcionalidades nuevas
 - fix: para corregir algún fallo
 - docs: para documentos
 - style: para formato
 - refactor: para refactorización del código
 - test: para añadir o modificar algún test.
- **titulo:** Un título para identificar fácil y rápidamente un commit
- **Descripción:** Una descripción detallada de ese commit
- **<Tarea: #>:** Referencia al número que tiene asociado el issue con el que está relacionado el commit.

Se pueden ver ejemplos de uso de estas plantillas en el siguiente enlace: <https://github.com/pabfrasan/decide/commits/developVotacion>.

Para realizar los commits, cada miembro es libre de usar GitHub online o la versión de escritorio.

1.8 Gestión de la construcción e integración continua

Las herramientas utilizadas para la realización de este apartado son las siguientes:

- Github
- Travis CI
- Codacy
- Heroku

Nuestro repositorio de Github está asociado con Travis CI, de manera que cada vez que se haga un commit en cualquiera de las ramas del repositorio se realizará una build de la aplicación y se ejecutarán todos los tests. Una vez que Travis CI finalice estos procesos, aparecerá indicado junto al commit si se ha creado correctamente la build y si se han superado los tests, así como una opción que permite ver en detalle la consola de Travis CI. La implementación de esta herramienta se ha realizado como se ha explicado durante las prácticas de la asignatura, exceptuando algunos cambios que ha realizado el grupo de visualización al fichero de “.travis.yml”.

Utilizamos Heroku para el despliegue online de la aplicación mediante una asociación con el repositorio que levanta solamente el código que se encuentra dentro de la rama de master. Dicho despliegue se realiza en la dirección <https://picaro-decide.herokuapp.com/admin/>.

Por último, utilizamos Codacy para analizar la calidad del proyecto. Cada vez que se realiza un commit a una de las ramas de develop de alguno de los grupos de pícaro, esta herramienta comprueba en que estado se encuentra el código tras el commit, indicando junto a este si la calidad del código es buena o ha empeorado. La implementación de Codacy se ha realizado como se ha explicado durante las prácticas de la asignatura.

Para comprobar la calidad de las build se han utilizado los siguientes indicadores:

- Número de issues: Codacy nos indica cuáles son los bugs y posibles problemas que tenemos dentro de cada rama, además de indicar porque es perjudicial en el caso de problemas de formateo del código. Esto nos permite solucionar estos problemas de manera más sencilla, sin tener que mirar las líneas de código una a una.

- Complejidad: Codacy utiliza la métrica de la complejidad ciclomática para encontrar ficheros complejos dentro de la rama. Esto nos permite evitar crear código excesivamente complejo y evitar perder tiempo innecesario creando una mayor cantidad de tests de las verdaderamente necesarias.
- Duplicación: Codacy nos muestra el porcentaje de líneas de código duplicadas dentro de la rama. Esto nos permite comprobar si estamos repitiendo líneas de código de manera innecesaria en vez de, por ejemplo, crear un método que implemente solamente una vez esas líneas.
- Cobertura de tests: Codacy nos muestra el porcentaje de líneas de código cubiertas por nuestros tests. Esto nos permite comprobar que parte del código no cubrimos con los tests, para así poder crear nuevos tests que la prueben.
- Líneas de código: Este indicador mide la longitud del código del proyecto. Esto nos permite tener controlado el tamaño de nuestro de proyecto para que no se desproporcione, necesitando una refactorización en este caso.

1.9 Gestión de liberaciones, despliegue y entregas

1.9.1 Proceso definido para las liberaciones

Para el entregable M3 se realizará una única liberación conjunta entre todos los grupos que será la versión final del proyecto. En esta deberán integrarse todas las nuevas funcionalidades propuestas por cada grupo de forma que el proyecto debe estar en su totalidad en la rama master.

1.9.1.1 Licencia de software para su proyecto

La licencia heredada para el proyecto Decide-Pícaro-Votación es de tipo copyleft en concreto AGPL.

Esta licencia es la que se aplica al proyecto original de Decide y especifica que todo producto derivado debe integrar esta misma licencia.

1.9.2 Proceso definido para el despliegue

El proyecto se puede desplegar tanto en local como con Heroku y Docker.

1.9.2.1 Despliegue en local

Utilizando un entorno virtual se puede desplegar el proyecto siguiendo estos pasos:

1.- Configurar el entorno

Tras entrar al entorno virtual mediante el comando *source* más la ruta del “activate” se deberán instalar los requisitos o dependencias del proyecto. Para esto tendremos que ubicarnos en la carpeta raíz del proyecto y ejecutar la instalación almacenada en el archivo *requirements.txt* . Para hacer eso utilizaremos el comando *pip3 install -r requirements.txt*

Con esto nuestro entorno reunirá las condiciones necesarias para ejecutar el proyecto. Lo que haremos siguiendo estas indicaciones.

2.- Preparar la base de datos

Primero acceder a la subcarpeta *decide* donde se haya el archivo *manage.py* . A través de él prepararemos la base de datos que se tendrá que crear previamente mediante *postgres*. Configuraremos el archivo *settings* para que apunte apropiadamente a las bases de datos creadas. Con los siguientes comandos haremos las migraciones necesarias para establecer la base de datos.

```
python3 ./manage.py makemigrations
python3 ./manage.py migrate
```

Creamos el superusuario con

```
python3 ./manage.py createsuperuser
```

para acceder al sistema

3.- Correr el proyecto

Ejecutamos el proyecto con

```
python3 ./manage.py runserver
```

Previamente a este paso se pueden ejecutar los test con

```
python3 ./manage.py test
```

para verificar que el sistema funciona correctamente.

Desde el momento en el que el servidor está corriendo se podrá acceder al sistema en modo administrador desde la dirección: <http://localhost:8000/admin> introduciendo las credenciales anteriormente creadas con el superuser

1.9.2.2 Despliegue con Docker

Mediante Docker podremos ejecutar el sistema decide a través del uso de contenedores e imágenes, permitiéndonos automatizar el despliegue

Configuración de Docker: Dockerfile

El archivo dockerfile es el que secuencia las instrucciones para el despliegue desde Docker, siendo este para nuestro proyecto:

```
from python:3.7-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 psycpg2
RUN pip install ipdb
RUN pip install ipython

WORKDIR /app

RUN git clone https://github.com/pabfrasan/decide .
RUN pip install -r requirements.txt

WORKDIR /app/decide

# local settings.py
ADD docker-settings.py /app/decide/local_settings.py
```

```
RUN ./manage.py collectstatic
```

```
#CMD ["gunicorn", "-w 5", "decide.wsgi", "--timeout=500", "-b 0.0.0.0:5000"]
```

1.9.2.3 Cómo desplegar en Docker

El primer paso para configurar Docker es instalarlo mediante el comando

```
sudo apt instal docker.io
```

Para desplegar desde Docker tendremos que tener una instalación limpia, es decir, haber eliminado las imágenes y contenedores previos que puedan entorpecer con el despliegue actual. Una vez eliminados podemos seguir.

Accedemos al directorio raíz del proyecto y ejecutamos los siguientes comandos:

```
cd docker docker-compose up -d docker exec -ti decide_web
```

y

```
./manage.py createsuperuser
```

Una vez creado el superusuario podremos acceder tal y como hacemos en el despliegue local a través de la dirección: <http://10.5.0.1:8000/admin>

1.9.2.4 Despliegue con Heroku

Heroku es una plataforma como servicio (PaaS) de computación en la Nube que soporta distintos lenguajes de programación. Esta plataforma nos permitirá ejecutar las operaciones directamente en la nube. Puede sincronizarse con Travis CI para asegurar una integración continua completa. Travis revisará el correcto funcionamiento del sistema y tras verificarlo lo enviará para su despliegue a Heroku.

Antes de integrarlo con Travis configuraremos Heroku siguiendo estos pasos.

Primero nos situaremos una vez más en la ruta de nuestro proyecto o repositorio y desde ahí crearemos la aplicación mediante el comando

```
heroku create "nombre"
```

Iniciaremos sesión en Heroku y a partir de ahí se nos habrá creado una aplicación que se asociará al repositorio.

Es importante disponer de un archivo Procfile en nuestro proyecto. Este archivo define parámetros importantes para la plataforma como el servidor, instrucciones pre-despliegue y otras utilidades.

```
% prepara el repositorio para su despliegue.  
  
release: sh -c 'cd decide && python manage.py migrate'  
  
% especifica el comando para lanzar Decide  
  
web: sh -c 'cd decide && gunicorn decide.wsgi --log-file -'
```

Este sería el contenido del nuestro.

Desde este momento podremos desplegar en Heroku únicamente ejecutando el comando

```
git push heroku master
```

Esto completará el despliegue desde la rama master de nuestro proyecto, para indicar una rama distinta tendrá que añadirse el nombre de la rama antes de master, dividido por dos puntos.

Con esto solo haría falta unos pequeños detalles para poder acceder al sistema, introduciendo los siguientes comandos crearemos una serie de credenciales para el acceso:

```
$ heroku run -a nombre-app bash  
$ ./decide/manage makemigrations  
$ ./decide/manage migrate  
$ ./decide/manage createsuperuser
```

1.9.2.5 Integración con Travis.

Para configurar la sincronización con Travis será necesario editar el archivo .travis.yml y el archivo settings previamente mencionado. A continuación, exponemos los cambios pertinentes.

A través del archivo .travis.yml se establece la configuración necesaria para definir el nombre de nuestra aplicación, la preferencia de despliegue a través de heroku, la estrategia de git y los parámetros apropiados para configurar la base de datos, además de una API key que vincularía Travis y heroku.

Aquí nuestro archivo actual:

```

dist: xenial

services:

- postgresql

addons:

  postgresql: '9.4'

before_script:

- psql -U postgres -c "create user decide password 'decide'"
- psql -U postgres -c "create database decidedb owner decide"
- psql -U postgres -c "ALTER USER decide CREATEDB"

language: python

python:

- '3.6'

install:

- pip install -r requirements.txt
- pip install codacy-coverage

script:

-
wget
https://github.com/mozilla/geckodriver/releases/download/v0.27.0/geckodriver-
v0.27.0-linux64.tar.gz
- tar -xzf geckodriver-v0.27.0-linux64.tar.gz
- chmod +x geckodriver
- sudo cp geckodriver /usr/bin/
- rm geckodriver-v0.27.0-linux64.tar.gz
- sudo chmod 755 chromedriver
- sudo mv chromedriver /usr/local/bin
- wget -q -O - https://dl.google.com/linux/linux_signing_key.pub | sudo apt-key add -
- sudo sh -c 'echo "deb [arch=amd64] http://dl.google.com/linux/chrome/deb/ stable
main" >> /etc/apt/sources.list.d/google-chrome.list'

```

```
- sudo apt update
- sudo apt install google-chrome-stable
- cd decide
- cp travis_local_settings.py local_settings.py

- coverage run --branch --source=. ./manage.py test authentication --keepdb
- coverage run --branch --source=. ./manage.py test visualizer --keepdb
- coverage run --branch --source=. ./manage.py test voting --keepdb

- coverage xml
- python-codacy-coverage -r coverage.xml
deploy:
  provider: heroku
  app: picaro-decide
  strategy: git
  api_key:
    Secure: XXXXXXXXXXXXXXXX
```

Este archivo permitirá además establecer que ramas del repositorio tienen un build válido que pueda desplegarse en Heroku,

Para completar el apartado de API-key se ejecutará el siguiente comando, tras haber creado la aplicación en Heroku.

```
travis encrypt $(heroku auth:token) --add deploy.api_key
```

Dentro del archivo settings.py es importante editar este apartado para configurar la aplicación de Heroku creada:

```
BASEURL = 'https://picaro-decide.herokuapp.com'

APIS = {
    'authentication': BASEURL,
```

```
'base': BASEURL ,  
'booth': BASEURL ,  
'census': BASEURL ,  
'mixnet': BASEURL ,  
'postproc': BASEURL ,  
'store': BASEURL ,  
'visualizer': BASEURL ,  
'voting': BASEURL ,  
}
```

Por último, habrá que crear el archivo `travis_local_settings.py` que copiaremos del archivo `local_settings.py` dentro de la carpeta `decide`.

A partir de aquí y tras hacer un commit y subir dicho commit al repositorio Travis se ejecutará de forma automática. Una vez se realice el despliegue correctamente tendremos que configurar de nuevo Decide como se ha hecho en apartados anteriores, migrando las bases de datos y creado en superusuario pertinente.

1.9.3 Proceso definido para las entregas

Antes de constituir los archivos que conformarán la entrega, se verificará el correcto funcionamiento de los incrementos implementados por cada integrante del grupo. Además, se realizará un merge para confirmar que todas las nuevas funcionalidades coexisten sin conflicto en una única rama y en una única ejecución del proyecto.

Para verificar esto se ejecutarán los test necesarios junto con el resto de los grupos que conforman Pícaro-Decide.

Tras estas verificaciones podremos rellenar los documentos apropiados establecidos en la entrega redactada por el profesor, siendo estos entregados finalmente por el Project Manager.

1.9.4 Política de nombrado e identificación de los entregables

Los archivos entregados deberán ser fácilmente identificables, siguiendo el patrón requerido por la asignatura. En caso de no ser proporcionado ningún modelo o ejemplo deberá ser completamente clara la diferenciación entre documentos además de identificar al grupo Pícaro-votación como autores de aquellos documentos que puedan no autodefinirse por su contenido.

1.10 Ejercicio de propuesta de cambio

La propuesta de cambio elegida es la de crear un nuevo parámetro en las votaciones que responda al esquema de Tag o Etiqueta. Como funcionalidad únicamente será un nuevo campo a la hora de crear y consultar una votación, pero que creemos que podría abrir el camino a nuevas formas de consulta, ordenación y recomendación de votaciones.

El primer paso a la hora de abordar una nueva tarea es crear la issue apropiada en el tablero de GitHub. Esta issue deberá responder al formato, o plantilla, de <feat>, dado que añade una nueva característica al sistema.

Una vez creada, cuando se asigne y se comience a trabajar en la misma deberá moverse a la columna "In progres"

El encargado principal de esta tarea tendrá que empezar a trabajar en su rama para implementar el código necesario, tras completar una solución aproximada deberá trasladarse a la rama develop donde en un futuro se verificará que no cause conflictos.

Antes de empezar a trabajar deberemos ejecutar un pull a la rama en la que estemos trabajando para evitar el mayor número de conflictos con los cambios ya implementados.

Para implementar la funcionalidad abriremos el repositorio desde nuestro entorno de programación favorito, accediendo a la subcarpeta decide. Dentro de la carpeta voting deberemos preocuparnos principalmente del archivo models.py para introducir el nuevo parámetro en el modelo de votación además de en serializers donde se verifican los distintos parámetros.

Para probar los cambios recién implementados eliminaremos la base de datos especificada en el archivo settings y la volveremos a crear, evitando así posibles conflictos.

```
sudo su - postgres  
  
psql -c "drop database decide"  
  
psql -c "create database decide owner decide"
```

Tras esto debemos migrar la base de datos con el siguiente comando, es posible que sea necesario aplicar el segundo comando si así nos lo pide.

```
python ./manage.py migrate  
python ./manage.py makemigrations
```

Una vez realizado esto podremos ejecutar el servidor con el comando

```
./manage.py runserver
```

y comprobar que funciona accediendo a través del navegador a localhost:8000/admin para iniciar sesión deberíamos haber creado un super-usuario previamente con el comando.

```
python3 ./manage.py createsuperuser
```

Tras esto, introducimos las credenciales oportunas y accederemos al panel de control de administrador.

Clicamos en votings y añadimos una nueva votación, se nos abrirá un formulario donde debería aparecer el nuevo campo implementado Tags, que rellenaremos de la forma oportuna. Verificamos que al completar la votación no saltan errores y continuamos con el proceso.

En cuanto a los aspectos negativos en general hemos tenido retrasos a la hora de los plazos de las entregas y el desconocimiento de Django ha causado un avance mucho más lento al esperado.

```
python3 ./manage.py test
```

Si los tests se ejecutan con normalidad y sin errores aparentes, continuaremos con el commit, siguiendo la plantilla previamente guardada en el archivo template, con los comandos

```
git config commit.template PATH  
git add  
git commit  
git push (tras haber rellenado la plantilla)
```

En este momento empezaría a trabajar Travis, comprobando que existe un despliegue funcional que poder ejecutar desde Heroku. Tras esta verificación (exitosa) de Travis deberemos volver al tablero de issues y situar esta tarea como completada.

Si todo continúa sin conflictos podemos hacer un merge desde la rama en la que estamos desarrollando a la rama master.

Tras hacer esto se volverá a realizar la verificación de Travis, solo que esta vez se desplegará automáticamente en Heroku.

1.11 Conclusiones y trabajo futuro

Durante el proyecto nos hemos dado cuenta de algunos puntos positivos y negativos en relación al desarrollo y gestiones para la realización del proyecto.

En cuanto a los aspectos positivos consideramos que la comunicación entre los miembros del equipo y el resto de los equipos ha sido buena, a pesar de ser todo de forma online, y todas las lecciones que hemos aprendido sobre la gestión del proyecto y sobre las herramientas empleadas como GitHub.

En cuanto a los aspectos negativos en general hemos tenido retrasos a la hora de los plazos de las entregas y el desconocimiento de Django ha causado un avance mucho más lento al esperado.

Como conclusión, la experiencia en general ha sido positiva para nosotros, a pesar de los inconvenientes de enfrentarse a una tecnología que no controlamos tanto.