DOCUMENTACIÓN ADMIN. CENSOS

Grupo 1

ID Opera: 13

Índice

Componentes del grupo

- Serrano Guerrero, Manuel (Coordinador) : 5
- López Heredia, Gonzalo : 4

Enlaces de interés

Repositorio de código:

https://github.com/mansergue/AdminCensos-EGC-G1-M5

Sistema desplegado:

https://g1admcensos.egc.duckdns.org

Grupo en ópera:

http://opera.eii.us.es/egc/public/trabajo/ver/id/86

Wiki:

https://github.com/mansergue/AdminCensos-EGC-G1-M5/wiki

1. Resumen

En este proyecto nuestro equipo se encarga de resolver un problema que se produce a la hora de realizar una votación. Para que una votación se considere democrática y legítima solo podrán ejercer su voto las personas censadas en esa votación, además, una persona no podrá votar dos veces en una misma votación. Es por ello que nuestro equipo crea este subsistema resolviendo el problema de la administración de un censo electoral.

2. Introducción y contexto

Nuestro subsistema es Administración de Censo, es el encargado de administrar los diferentes censos que surgen cada vez que se crea una nueva votación y gestionar los usuarios que pueden participar en cada una de estas. El administrador de la votación podrá decidir qué personas están censadas en su votación. Además, el censo determinará qué personas han realizado su voto que para que el subsistema de cabina de votaciones no permita realizar de nuevo el voto. En principio decidimos utilizar el código heredado del curso anterior, este estaba realizado en Java junto al framework Spring, utilizando como herramienta Eclipse Indigo. Tenía el sistema de librerías de Maven y utilizaba una base de datos MySQL. Montamos y arreglamos todo el entorno de desarrollo ya que tenía algunos problemas. Empezamos a trabajar en ese entorno. Con este punto de partida nuestros objetivos eran optimizar el código heredado refactorizando los métodos e incluir nuevas funcionalidades como realizar el censo de los usuarios de manera masiva.

Tras la reunión inicial de todos los grupos encargados de los subsistemas desde integración nos informaron que la base de datos utilizada sería MariaDB lo cual complicaba mucho nuestra conexión a esta. Por lo tanto, intentamos implementar el uso de esta nueva base de datos, pero fue imposible. Finalmente, no se pudo integrar y tras una reunión con el equipo se decidió por unanimidad reescribir todo el código en Python usando el framework Django, que nos facilitaba la conexión con la base de datos.

3. Descripción del sistema

El sistema de Administración de censo se trata de un sistema escrito en Python junto al framework Django. Se comunica con una base de datos que usa una tecnología derivada de MySQL con licencia GPL titulada MariaDB. Tanto esta base de datos como nuestro subsistema están desplegados en un contenedor Docker. En cuanto a la relación con otros subsistemas, nuestro proyecto tiene una relación directa con el subsistema de autenticación ya que es imprescindible, para realizar cualquier modificación o creación de un censo, primero hay que pasar por autenticación.

Las funcionalidades que implementa son:

- Creación de un censo
- Edición de un censo
- Eliminación de un censo
- Login
- Inscripción a un censo

4. Entorno de desarrollo

El sistema operativo escogido para nuestro entorno de desarrollo ha sido Ubuntu, una de las distribuciones de Linux más conocidas. Debajo se detallan las tecnologías utilizadas:

Lenguaje de desarrollo: python2.7

Base de datos: MariaDB 10.0.31

Framework: Django 1.5

Docker: Despliegue de la base de datos mariaDB

IDE: PyCharm

Instalación del subsistema

Desde la distribución Ubuntu de linux haremos el despliegue del entorno de desarrollo.

1. En primer debermos instalar docker en nuestro equipo, para ello seguiremos este tutorial: https://www.digitalocean.com/community/tutorials/como-instalar-y-usar-docker-en-ubuntu-16-04-es
2. Una vez instalado docker, instalaremos Docker Compose utilizando los siguientes comandos:

```
sudo curl -L https://github.com/docker/compose/releases/download/1.18.0/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
```

```
sudo chmod +x /usr/local/bin/docker-compose
```

3. Para desplegar la base de datos ejecutaremos el siguiente comando en la raíz del archivo docker-compose.yml de la base de datos mariaDB que proporciona el grupo de integración. Gracias a docker tendremos la base de datos funcionando en pocos segundos.

```
docker-compose up -d --build
```

4. Instalar las dependencias del proyecto. Para poder instalarlas necesitaremos el gestor de dependencias, pip de python2.7:
   En el caso de que no lo tengamos instalado, ejecutaremos los siguientes comandos:

```
sudo apt-get update && sudo apt-get -y upgrade
```

```
sudo apt-get instal python-pip
```

Para instalar las dependencias ejecutaremos los siguientes comandos en la raíz de nuestro proyecto:

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

```
pip install MySQL-python
```

5. Finalmente ejecutaremos el servidor de django con el comando:

```
python manage.py runserver
```

5. Gestión del cambio, incidencias y depuración

Gestión del cambio

Para la gestión del cambio en el proyecto se sigue el siguiente protocolo:

Las nuevas funcionalidades se propondrán y discutirán en una reunión o mediante mensajería instantánea. Una vez se haya llegado a un acuerdo, el coordinador será el encargado de crear una nueva Issue para generar este nuevo cambio en el código.

El coordinador se encarga de asignar a los desarrolladores las tareas en sus correspondientes Issues. A partir de este punto se sigue el procedimiento indicado con respecto a la gestión de incidencias descrito en el apartado “Gestión de incidencias” y se procederá a la creación de su rama correspondiente (Feature) tal y como se describe en el apartado “Gestión del código fuente”.

Gestión de incidencias

Todas las incidencias se gestionan mediante la interfaz que ofrece GitHub de administración de. Issues que ofrece el propio repositorio. Tenemos que diferenciar entre las incidencias internas que serán las originadas en nuestro grupo de trabajo y las incidencias externas que serán las que recibiremos de otros grupos.

Obligatoriamente todas las incidencias deben estar catalogadas con su prioridad y su estado correspondiente.

La prioridad puede ser: Alta, Media, Baja

El estado puede ser:

- Backlog (Significará que está pendiente de asignar)
- En progreso
- Realizado

Además, las incidencias se podrán etiquetar por:

- Documentación (Trata sobre algo relacionado con la documentación)
- Bug (Se trata de un bug encontrado)
- Enhancement (Mejora)
- Hotfix (Trata de un bug/característica que necesita ser solventada con urgencia)
- Invalid (Incidencia inválida)
- Investigación (Trata sobre algo relacionado con investigación)
- Question (Una pregunta)
- Feature (Una nueva funcionalidad)
- Duplicate (Incidencia duplicada)

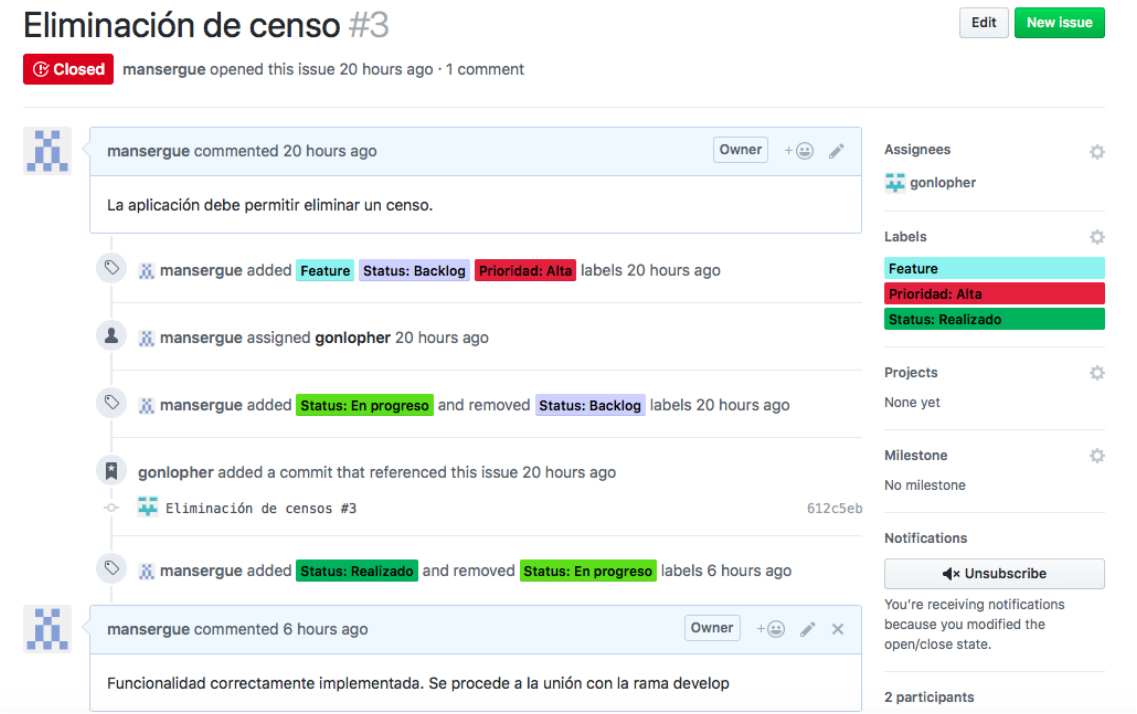
Incidencias Internas: Una vez se encuentra una incidencia en el subsistema se abre una Issue en GitHub. Se etiquetará siguiendo la especificación indicada arriba. En cada incidencia se intentará aportar la mayor información posible, posteriormente el coordinador del grupo se encargará de asignar a la persona correspondiente para que se encargue de resolverla.

En el caso de que la incidencia se encuentre en estado “Backlog” significará que está pendiente de asignación y será el coordinador el encargado de asignarla a un miembro del equipo y la pasara al estado “En progreso”.

El desarrollador encargado solventará la incidencia referenciando los commits siguiendo el protocolo descrito en el apartado “Gestión de código fuente” y “Commits”.

Una vez el desarrollador haya terminado su trabajo, cambiará el estado “Realizado”, entonces será el turno del coordinador que se encargará de comprobar que el código está correcto, funciona perfectamente y está solventada la incidencia en su totalidad. Si el código estuviera correcto, el coordinador cerrará la Issue y se encargará de realizar la unificación de ramas tal y como se describe en el apartado “Gestión del código fuente”. En el caso de hubiera cualquier error y no se cumplieran las condiciones anteriormente descritas, el coordinador describirá en la propia incidencia el nuevo problema detectado y volverá a poner la incidencia en Estado “En progreso”.

Aquí se puede ver un ejemplo donde se ha seguido el proceso de una incidencia correspondiente a una nueva Feature: https://github.com/mansergue/AdminCensos-EGC-G1-M5/issues/3



Incidencias Externas: Para las incidencias externas realizadas por otro grupo, se seguirá el mismo proceso que en las internas. Además, el coordinador se pondría en contacto personalmente con el coordinador del otro grupo si fuera necesario aclarar algún aspecto. El caso de las incidencias que reportemos a otros grupos, se comunicará al coordinador y si este la aprueba, se encargará de realizarla en el repositorio del grupo correspondiente siguiendo su propia plantilla de gestión de incidencias (deberá informarse anteriormente).

Gestión del código fuente

La gestión del código fuente se realizará a través de la herramienta Git. Además, usaremos GitHub como repositorio de código. Para el uso de esta herramienta definimos un proceso a seguir para la gestión del código fuente del proyecto.

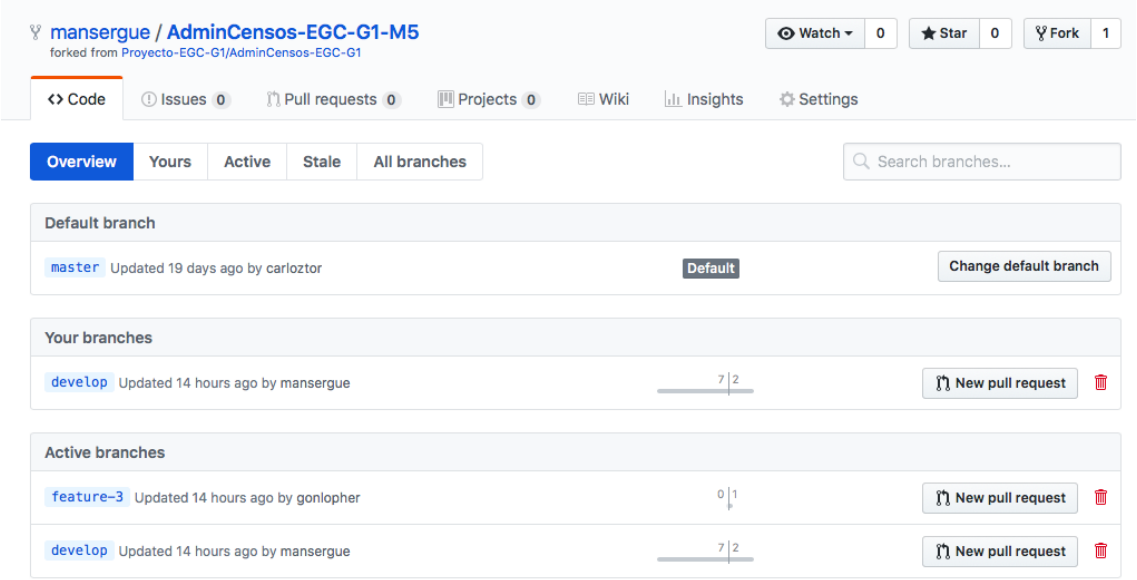
Modelo de uso

Siguiendo las buenas prácticas propuestas por Gitflow, en nuestro proyecto usamos una versión un poco simplificada debido a la pequeña envergadura del proyecto. Contamos con las ramas Master, Develop, Feature y Hotfix. Prescindimos por lo tanto de la rama Release.

La gestión de ramas se hace de la siguiente manera:

Una vez se ha creado una incidencia, el desarrollador que tenga asignado dicha incidencia hará la modificación respectiva en el código. En el caso de que la incidencia esté catalogada con la etiqueta Feature, creará una nueva rama que partirá del contenido de la rama Develop y se titulará con la palabra Feature-X donde X corresponde al número de la Issue.

Podemos visualizar un ejemplo de una rama feature-3 que implementa una nueva funcionalidad.



En el caso de que la incidencia esté catalogada como Hotfix, entonces realizará el cambio en la rama Hotfix que parte de la rama Master.

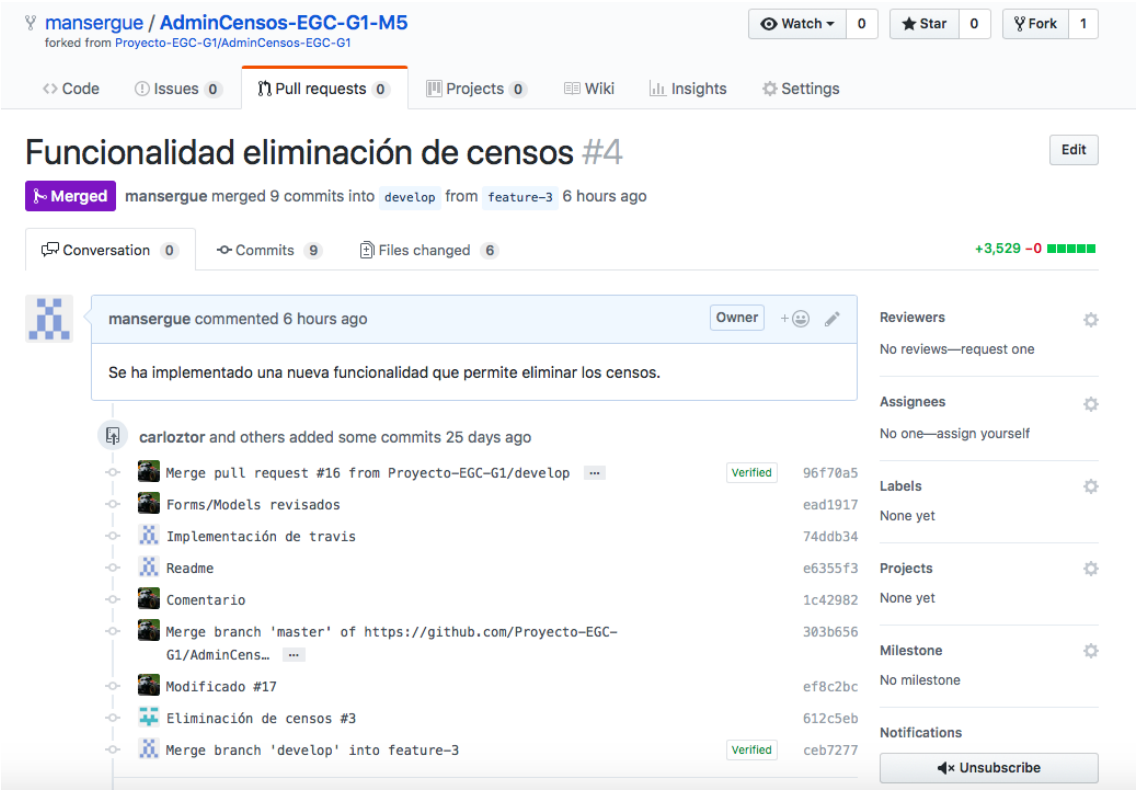
Las ramas que catalogamos como Feature solo se usarán como su propio nombre indica, para nuevas características del sistema. Si se desea corregir un bug de una funcionalidad ya implementada, se corregirá en la rama Develop.

Para realizar cualquier commit en cualquier rama se deberá seguir el protocolo descrito en el apartado *"Commits"*.

Una vez se haya seguido el proceso completo de gestión de incidencias especificado en el apartado "Gestión de incidencias" y la Issue esté cerrada, entonces el coordinador del proyecto (Manuel Serrano), se encargará de unificar la rama. La unificación se hará mediante pull request. En el caso de que sea una rama Feature se realizará merge con la rama Develop y se eliminará la rama Feature correspondiente.

Aquí podemos ver un ejemplo de un pull request realizado a la rama Develop. Se une con la rama feature-3:

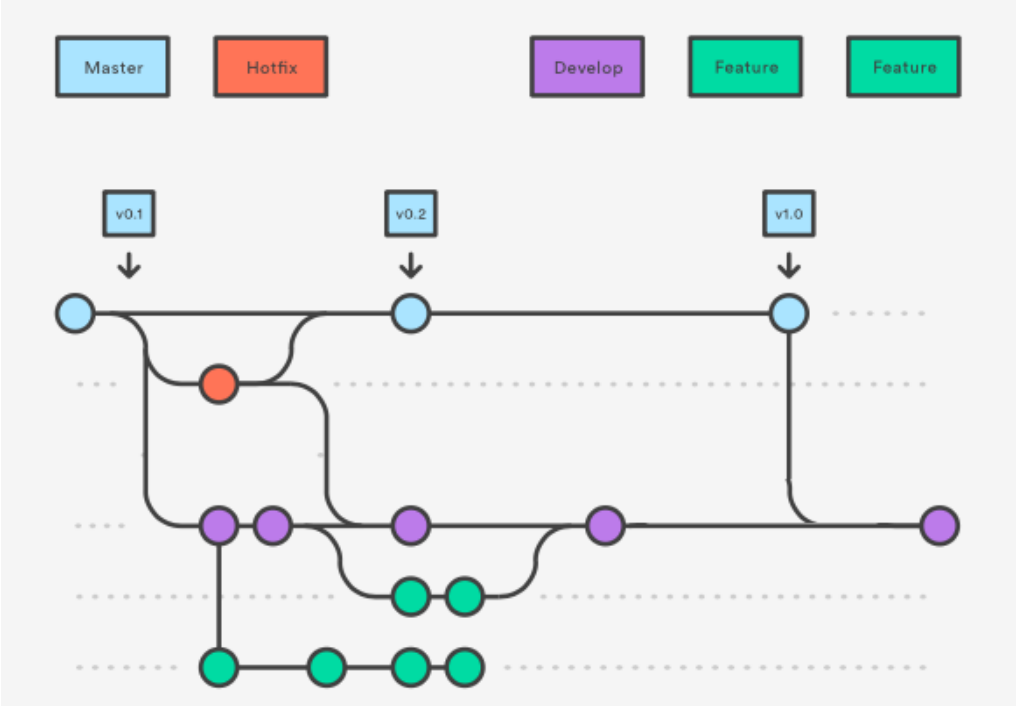
https://github.com/mansergue/AdminCensos-EGC-G1-M5/pull/4



La rama Master concentrará todo el código preparado para subir a producción. La rama Develop concentrará todo el código de las nuevas versiones del proyecto, además se preparará para subir a master una vez esté la versión completamente estable y lista para subir a producción. La rama Hotfix se utilizará para errores encontrados o cambios que necesiten ser solventados lo antes posible (sean urgentes).

Cualquier consideración o problema con respecto a las ramas se comunicará al coordinador que se encargará de solucionarlo.

El mapa de ramas finalmente quedará asi:



Commits

Sólo se podrá realizar commit si la mejora implementada está completamente funcional. No se subirá una funcionalidad inacabada o con errores conocidos. Además, los commits deben seguir el siguiente formato:

FORMATO: **Titulo +** Espacio en blanco **+ Cuerpo +** Espacio en blanco **+ Pie**

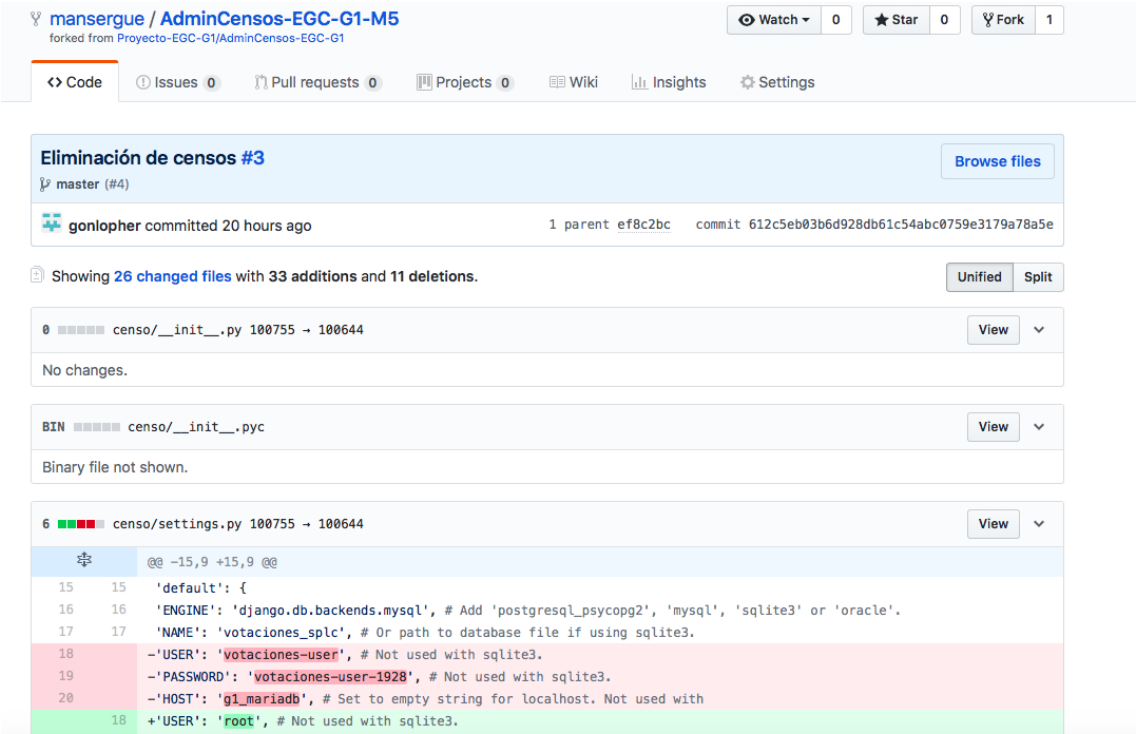
Título: Breve descripción de qué se ha cambiado

Cuerpo: Descripción detallada del cambio que se ha realizado. (Si es necesario)

Pie: Referencia con una almohadilla con el número de la issue a la que hace referencia

Aquí un ejemplo de un commit realizado en el proyecto que se puede visualizar a través de la siguiente url:

https://github.com/mansergue/AdminCensos-EGC-G1-M5/commit/612c5eb03b6d928db61c54abc0759e3179a78a5e



6. Gestión de la construcción e integración continua

Como método de construcción e integración continua, en nuestro proyecto hemos empleado Docker junto a Docker Hun y Travis CI.

Docker: https://github.com/mansergue/AdminCensos-EGC-G1-M5/blob/master/Dockerfile

DockerHub: https://hub.docker.com/u/manusg/

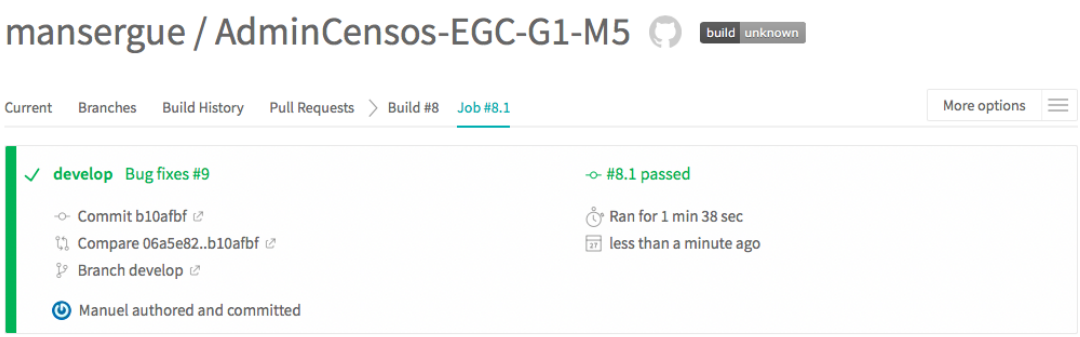
Travis: https://github.com/mansergue/AdminCensos-EGC-G1-M5/blob/master/.travis.yml

Integramos nuestro proyecto en un contenedor docker para poder utilizarlo tanto en el entorno de despliegue con Docker Hub. Creamos el archivo Dockerfile con la información necesaria para la construcción del contenedor añadiendo nuestro proyecto.

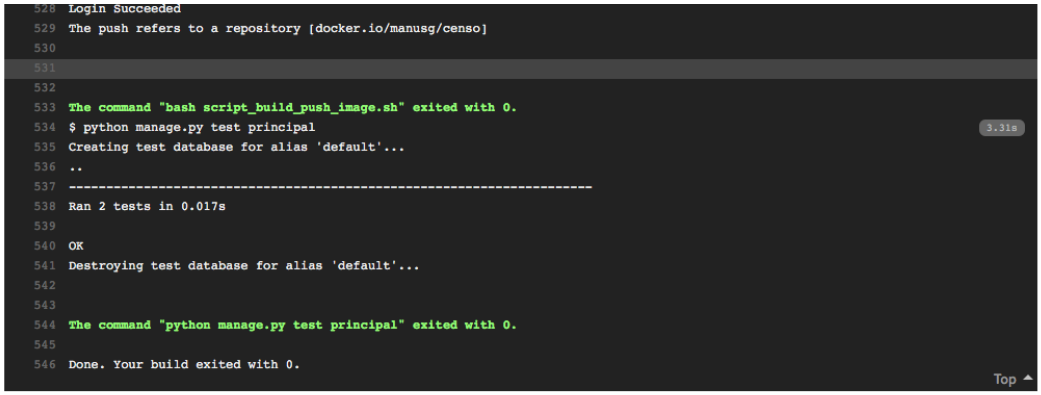
Con cada push que se realice en el repositorio, automáticamente Travis se encargará de construir el proyecto y verificar que no hay ningún error de compilación en el proyecto. Además creará una nueva imagen de docker gracias al archivo Dockerfile y subirá esta nueva imagen a DockerHub. Una vez subida, se desplegará en el servidor de producción automáticamente. Esta última funcionalidad ha sido creada por el Equipo de Integración, un servicio se encarga de comprobar que todos los contenedores se encuentran actualizados a su última versión, en el caso que no lo esté, parará el contenedor existente de forma segura y desplegará la nueva versión. Una vez desplegado mandará un correo automáticamente notificando de los cambios realizados y estado de los contenedores.

Aquí podemos ver un ejemplo de la construcción del proyecto después de hacer commit:

https://travis-ci.org/mansergue/AdminCensos-EGC-G1-M5/builds/338155131?utm_source=github_status&utm_medium=notification



Podemos observar cómo Travis CI ejecuta automáticamente los test:



En cada commit realizará la construcción y la carga de los test del proyecto.

7. Gestión de liberaciones, despliegue y entregas

Gestión de liberaciones

En GitHub gestionamos todas las versiones liberadas que finalmente se despliegan en el servidor. Como se indica en el apartado de Gestión del código fuente, todas las versiones que se despliegan en el servidor son las que se encuentran en la rama Master de nuestro repositorio. En cada versión el equipo indica los cambios y mejoras implementados en esa versión.

Se puede acceder a nuestras versiones a través de este enlace:

https://github.com/mansergue/AdminCensos-EGC-G1-M5/releases

La nomenclatura usada para las versiones comienza con una v siguiéndole el número de versión. El primer digito contempla una actualización/mejora muy relevante de la aplicación. El segundo digito contempla mejoras menores en el proyecto. Un ejemplo de cómo quedaría: v2.2.5

Todas las versiones finales de la rama master se consideran candidatas a ser entregables. El coordinador decidirá qué versión será utilizará para los entregables.

Gestión de despliegue

Para la gestión del despliegue se utiliza TRAVIS CI, para ello creamos un archivo titulado .travis.yml (debajo podemos ver en detalle el archivo) en la rama master donde detallamos la configuración necesaria. Travis CI se encargará de desplegar el proyecto en el servidor una vez se haya realizado push en dicha rama. Creará una nueva imagen docker a partir del script y la subirá al repositorio en DockerHub. Finalmente, gracias al equipo de integración, una vez realizado el push en DockerHub, automáticamente se desplegará la nueva versión del contenedor en el servidor.

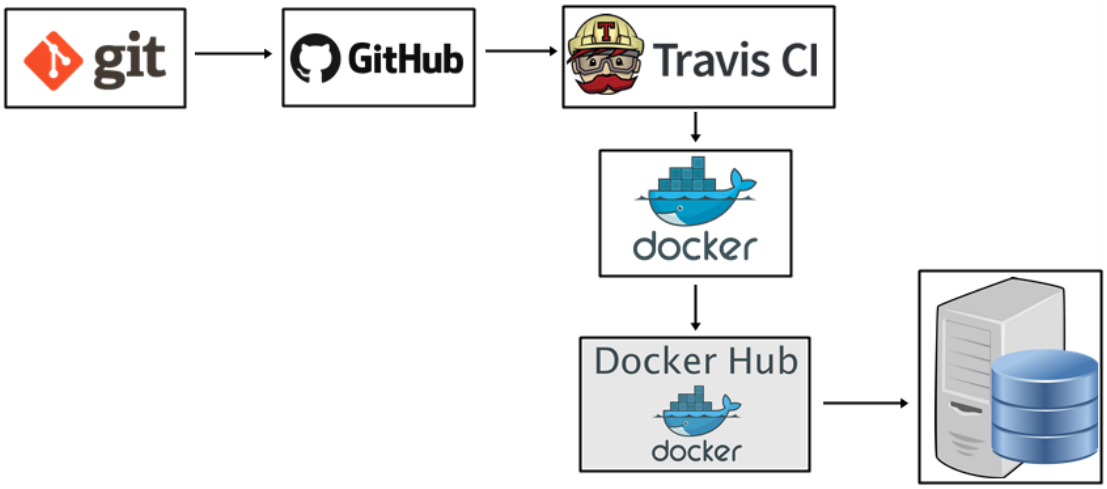
El archivo travis en cuestión se encuentra:

https://github.com/mansergue/AdminCensos-EGC-G1-M5/blob/master/.travis.yml

30 lines (21 sloc) | 777 Bytes — Raw | Blame | History

```
language: python

sudo: required

python:
    - "2.7.10"

install:
   - "pip install -r requirements.txt"
   - "pip install mysql-python"

script:
  - bash script_build_push_image.sh
  - python manage.py test principal

services:
  - mysql
  - docker

before_script:
  - echo "USE mysql;\nUPDATE user SET password=PASSWORD('your_password') WHERE user='root';\nFLUSH PRIVILEGES;\n" | mysql -u ro

before_install:
  - export DJANGO_SETTINGS_MODULE=censo.settings

  - mysql -e 'DROP DATABASE IF EXISTS votaciones_splc'
  - mysql -e 'CREATE DATABASE IF NOT EXISTS votaciones_splc;'
  - mysql -u root --default-character-set=utf8 votaciones_splc < scripts/script_create_votaciones_splc.sql
  - mysql -u root --default-character-set=utf8 votaciones_splc < scripts/splc2017.sql
```

8. Mapa de herramientas



Git - Sistema de control de versiones.

GitHub - Repositorio usado junto al sistema de control de versiones comentado

anteriormente.

Travis - Herramienta para la automatización de la construcción, realización de test, base de datos, despliegue e integración.

Docker - Herramienta usada para la creación de la imagen del subsistema y posterior despliegue usando Travis.

DockerHub - Repositorio de imágenes creadas por Docker.

9. Conclusiones y trabajo futuro

Hemos aprendido mucho en cuanto a temas de gestión del código fuente, gestión de incidencias en equipos. Es una buena manera de trabajar, productiva y organizada. Tambíen hemos trabajado con nuevas tecnologías como Django, mariaDB, Docker, DockerHub, Travis CI que nos han ayudado a la integración continua en el proyecto, automatización de pruebas, automatización la construcción además de la automatización de la entrega y despliegue del sistema.

Como trabajos futuros planteamos:

- Permitir la creación de un censo de usuarios de manera masiva
- Filtrar la lista de censos creados
- Permitir al usuario visualizar en qué censos está censado.