
EVOLUCIÓN Y GESTIÓN DE LA CONFIGURACIÓN

DOCUMENTO DEL PROYECTO
2017 - 2018



Francisco Javier García Parrales

Daniel Lozano Portillo

María Ruiz Gutiérrez

Miguel Ternero Algarín

Laura Vera Recacha

GRUPO ADMINISTRACIÓN DE CENSOS

Índice:

Resumen	2
Introducción y contexto	3
Descripción del sistema	5
Descripción del sistema del punto de vista funcional y arquitectónico	5
Descripción de los componentes del sistema.	5
Relación con otros subsistemas.	6
Cambios con respecto al curso anterior.	7
Planificación del proyecto	8
Entorno de desarrollo	11
Gestión del cambio, incidencias y depuración	12
Gestión del código fuente	16
Gestión de la construcción e integración continua	18
Gestión de liberaciones, despliegue y entregas	20
Mapa de herramientas	22
Ejercicio de propuesta de cambio	24
Conclusiones y trabajo futuro	26
Anexos	27

1. Resumen

En este proyecto hemos tratado la solución para resolver la necesidad de la creación de Censos para controlar el acceso a determinadas votaciones del sistema Nvotes.

Una de nuestras funciones principales era la creación y edición de estos Censos, que agrupaban a los votantes en diferentes grupos (grupos provistos por el subsistema de Autenticación), y se establecen unas fechas de validez para los que dichos votantes podían o no acceder a determinada votación (id determinado por el subsistema de Adm. de Votaciones), algo crucial para controlar las votaciones y poder proveer al sistema de una función básica.

Hemos ido desarrollando las distintas soluciones y mejorandolas con la aparición de nuevas necesidades de otros subsistemas, y acabado desarrollando una estructura de datos estable y funcional, así como un conjunto de funcionalidades útiles y más que básicas para el correcto funcionamiento (crear censo, editar, ver si un usuario puede o no votar, etc).

2. Introducción y contexto

El contexto en el que nos encontramos será el de desarrollar o mejorar un subsistema dentro de un proyecto de mayor envergadura llamado Nvotes, el cual es un sistema de votaciones por internet.

El subsistema que debemos elegir será heredado del curso anterior elaborado por otros alumnos de la escuela, siempre y cuando esto sea posible. La evolución del proyecto es una de las partes mas importantes de la asignatura, asi pues en lo máximo posible se deberá de tener muy presente el código heredado para desarrollar el subsistema de este año.

Antes de la elección del subsistema se debían formar los equipos de trabajo, siendo estos de 5 a 7 personas. Tras la formación del grupo este debía de elegir el subsistema a desarrollar por el equipo. En un primer momento el equipo optó por realizar el subproyecto de visualización, pero nos topamos con el problema de que había dos grupos más interesados en este subproyecto, para arreglar este problema se procedió a realizar un sorteo para ver quien lo realizaría. Este grupo no salió victorioso del sorteo entonces se tuvo que realizar una charla improvisada para decidir entre todo el equipo que subsistema de los que no estaban escogidos realizaríamos. Decidido ya cuál va a ser el nuevo subproyecto del equipo nos registramos en opera e introducimos el subsistema a realizar por nosotros, este subsistema fue “administración de censos”.

El cometido de nuestro subsistema es la administración de censos que incluye el crearlos, modificarlos, listarlos y borrarlos, además de transmitir quien puede votar o no.

Tras tener asignado el subsistema y estar estable el equipo de trabajo, se procedió a buscar y examinar el código heredado del año pasado, tras varios días buscándolo contactamos con el profesor que más tarde nos facilito donde estaba. El trabajo realizado los alumnos en este subsistema el curso anterior fue examinado por este equipo y se llego al consenso de que no se iba a heredar ningún trozo de código, ni usarse el mismo lenguaje ni IDE, es decir se decidió implementar la funcionalidad que ya había y mejorarla en otro lenguaje, este lenguaje ha sido pyhton. Por tanto, el trabajo tuvo que realizarse partiendo de cero.

Por último comentar que se han conseguido implementar satisfactoriamente las funciones que el equipo consideró relevantes para el sistema así como sus

correspondientes test, otras funciones menos importantes se dejaron a un lado para usarse en el caso de que el trabajo a realizar fuese escaso.

3. Descripción del sistema

a. Descripción del sistema del punto de vista funcional y arquitectónico

Desde el punto de vista funcional nuestro subsistema depende de autenticación y creación y administración de votaciones ya que nuestro subsistema tiene la función de decidir qué usuarios pueden votar en las votaciones.

El desarrollo de este subsistema se ha realizado mediante Django REST framework en su versión 1.11 con el lenguaje Python en su versión 3.6. El código está estructurado de la siguiente manera:

- Censos/api: esta carpeta contiene todo el código del desarrollo de la funcionalidad de la API, las urls, los tests, los modelos y todo lo demás relacionado con la API.
- Censos/censos: esta carpeta contiene archivos de configuración para la gestión de las conexiones con la BD y demás configuraciones.
- A parte de esas dos carpetas nos encontramos con censos/manage.py este es un archivo el cual nos permite correr el servidor donde esta la API, hacer las migraciones a la base de datos, realizar la ejecución de los test, es tareas de administrador.

b. Descripción de los componentes del sistema.

Ahora explicaremos un poco el contenido de la carpeta censos que es donde se especifica la API. Dentro de esa carpeta nos encontramos los siguientes archivos:

- init.py: este archivo esta presente para especificar que todo lo que hay bajo esa carpeta es de lenguaje python.
- Apps.py: en este archivo se especifica el nombre que tendrá la aplicación para el administrador.
- Models.py: aquí se deberán especificar los modelos que se van a persistir en la base de datos. Este fichero realiza la conexión directa con la BD Se especificará el nombre de la tabla y el nombre de las columnas, así como el tipo que son cada una.

- Serializers.py: este archivo guarda como se van a serializar los objetos, es decir traducir el modelo de django a otro tipo de formato, en nuestro caso de serializar a json. En django se puede usar tanto el concepto de serializar como el concepto de deserializar. Es decir nos sirve para traducir respuestas y al revés.
- Tests.py: aquí es donde están almacenado todos los tests para comprobar las funciones creadas.
- Urls.py: este archivo contiene la especificación de cada url que existe en nuestra API, cada url se corresponde con el método de la vista que debe ejecutar.
- Views.py aquí se especifica el método de cada vista, por ejemplo, el delete_censo, cuando se hace una petición a delete desde la url se llama a este método que será el que se ejecute.

c. Relación con otros subsistemas.

En cuanto a la relación con los demás subsistemas nos encontramos con que nuestro subsistema tiene una relación muy fuerte con autenticación pues consumiendo su API obtendremos información sobre el usuario como puede ser el rol ya que este atributo no puede ser nulo para un censo, debido a que nos tenemos que integrar con autenticación hemos cambiado la estructura de la base de datos para hacer aparecer el atributo "rol". Otra relación que tiene nuestro subsistema es con administracion de votaciones, la relación con este subsistema es a través del atributo id_votacion.

4. Cambios con respecto al curso anterior.

Según lo visto en el trabajo del curso anterior nos encontramos con una wiki un poco pobre en documentación sobre el trabajo, pues en la wiki no aparece ninguna especificación de la API. Nosotros hemos especificado el uso de la API de manera muy detallada en un apartado llamado API en nuestra wiki.

El equipo encargado de administración y creación de censos del curso anterior desarrollo el trabajo con el framework de spring. Este framework hemos pensado nosotros que está enfocado a proyectos de mayor envergadura y complejidad que el desarrollo de una API como la de este subsistema. Este framework requiere del uso de muchos archivos para llegar a la construcción final de la API. Durante el curso 2017-2018 se ha decido prescindir de este framework y usar el framework de django que está más enfocado a la construcción de API rest.

En cuanto a funcionalidad se ha implementado la misma funcionalidad, pero claro esta en vez de usar java se ha usado Python.

El equipo del curso anterior hizo una mala gestión del código pues en el repositorio no disponían un archivo básico de git como es .gitignore, esto puede ahorrar muchos archivos cache o basura en el repositorio. Nosotros por nuestra parte desde la creación del entorno de trabajo se completó este archivo.

La base de datos del trabajo del anterior curso es Mysql que es la misma que este solo nosotros en la base de datos tenemos distintos campos creados para un censo quedando mucho más claro cuando se recupera de la base de datos el objeto.

Los alumnos del anterior curso tuvieron la mal idea de que los objetos se iban a recuperar solo por su id, nosotros en cambio en nuestra API se puede especificar a través de qué campo se puede recuperar un objeto con la funcionalidad filter, existe otra función que es un get que si se encarga de recuperar el objeto en base al id pasado como parámetro. La función de filter está más orientada para que la usen los demás subsistemas.

Otra diferencia con el curso anterior es que en el tema de los censos este año enlazamos el censo con la votación mediante un id..

5. Planificación del proyecto

Para la realización del trabajo hemos tenido la siguiente planificación; En cuanto al grupo tenemos los roles de Coordinador, este trabajo ha sido realizado por Francisco Javier García Parrales. Otro rol que destacar es el gestor de incidencias, realizado por Miguel Ternero Algarín, el cual está explicado en el apartado [Gestión del cambio, incidencias y depuración](#). Los demás integrantes del grupo tienen el papel de desarrolladores, Daniel Lozano Portillo, María Ruiz Gutiérrez y Laura Vera Recacha, junto con los dos componentes mencionados anteriormente.

Las tareas principales que tiene nuestro trabajo y quién las ha realizado se describe a continuación:

- Crear el entorno de desarrollo, por Fco. Javier García Parrales
- Creación y desarrollo de la wiki, por Fco. Javier García Parrales
- Gestionar las incidencias, por Miguel Ternero Algarín
- Realizar las funcionalidades que pertenecen a nuestra parte del sistema de votación, estas son las funciones: de:
 - Obtener un censo, realizado por Fco. Javier García Parrales
 - Crear un censo, por Laura Vera Recacha
 - Filtrar un censo (por su id de votación, rol, nombre, fecha de inicio o fecha de finalización del censo), por María Ruiz Gutiérrez
 - Actualizar un censo, por Miguel Ternero Algarín
 - Borrar un censo, por Daniel Lozano Portillo
 - Decidir quién puede votar y quién no, por Fco. Javier García Parrales
- Para cada una de las funciones se han hecho varios test:
 - Test para obtener un censo, por Fco. Javier García Parrales
 - Test para crear un censo, por Laura Vera Recacha
 - Test para filtrar un censo, por María Ruiz Gutiérrez

- Test para actualizar un censo, por Miguel Ternero Algarín
- Test para borrar un censo, por Daniel Lozano Portillo
- Test para la funcionalidad “can_vote”, por Fco. Javier García Parrales
- Desplegar el sistema, en esta tarea ha sido un poco más complicada por el desconocimiento de la tecnología, y por ello ha habido involucración por todos los miembros del equipo.
- Y por último la realización de los documentos necesarios para los milestone de la asignatura. Esta tarea también ha sido repartida entre todos los miembros del grupo.

Para ver un seguimiento de las tareas y diario de grupo adjuntamos un enlace de trello: <https://trello.com/b/dA4Pf6h5/egc-censos>

Ahora la planificación la dividiremos por los milestone de la asignatura:

En el milestone 1, en el caso de tener código heredado, se lleva el ecosistema preparado. En nuestro caso no hemos heredado código de otros años. Por tanto, este hito ha sido para hablar con el grupo de integración y aclararnos sobre nuestra funcionalidad.

En el milestone 2, teníamos todo el entorno de desarrollo en funcionamiento con todas sus dependencias. El entorno de desarrollo ha sido creado por el coordinador del grupo. Otra tarea para este hito ha sido realizar una presentación explicando partes, como la gestión de incidencias, la gestión del código, si ha implementado nuevas funcionalidades... De esta documentación se ha encargado el resto del grupo. Para este *milestone* también estaba toda la documentación de la API en la wiki; encargándose de ello el jefe de proyecto.

En el milestone 3, teníamos todas las funcionalidades, descritas anteriormente, desarrolladas. También el sistema estaba desplegado aisladamente. Para realizarlas ha habido la implicación de todo el grupo.

En el milestone 4, desarrollamos todos los test, desplegar el sistema integrándose con los demás y realizar la documentación requerida. Para desplegarlo hemos usado las instrucciones de integración. Para todas estas tareas han intervenido todos los miembros del grupo.

La tarea de gestionar las incidencias ha estado presente en todos los milestone.

Por último, resumimos el grado de implicación de cada miembro del grupo.

Tomando los siguientes valores, 0 al que menos implicación en el trabajo ha tenido y 10 el que mayor implicación ha tenido.

Apellidos, Nombre

Grado de implicación

García Parrales, Fco. Javier	10
Lozano Portillo, Daniel	9
Ruiz Gutiérrez, María	9
Ternero Algarín, Miguel	9
Vera Recacha, Laura	9

Hay que tener en cuenta, que al trabajar en una máquina virtual de ubuntu y no estar familiarizado con la tecnología de git, hay commit que no tienen sus credenciales y por tanto no se le aplican a la persona correspondiente. Así que el número de commit puede ser aproximadamente una cifra. Como se puede ver la persona que se encarga de integrar todo el código tiene una cifra mucho más elevada.

6. Entorno de desarrollo

Hemos usado todos los desarrolladores el mismo sistema y versiones, ya que dicho sistema ha sido distribuido entre nosotros (una máquina virtual con versiones de los servicios usados ya instalada) y gestionado las nuevas dependencias mediante el archivo requirements.txt

Cada desarrollador ha trabajado sobre una máquina virtual (VirtualBox) corriendo sobre Ubuntu en su versión 16.04 LTS. Dentro de la misma hemos trabajado con el framework de Django, en su versión 1.11.7, corriendo con Python 3.6 y encapsulado en un entorno virtual (Virtualenv) para alojar las dependencias de python de otros proyectos. Dichas dependencias han sido instaladas y gestionadas por el gestor pip (pip3, para Python 3.6) y grabadas en requirements.txt. Como servicios, hemos hecho uso de mysql (en su versión más actual) para la base de datos, travis para la gestión de integración, despliegue y testing continuo, y git (con GitHub como interfaz) para la gestión de código.

Para instalar y arrancar el sistema deberíamos seguir (suponiendo que estamos en la terminal de Ubuntu)

- *apt-get install mysql && apt-get install git* <En caso de no estar instalados>
- *git clone <https://github.com/EGC-G2-Trabajo-1718/egc-censos.git>*
- *cd egc-censos/*
- *virtualenv .* <Esto crearía el entorno virtual en la carpeta egc-censos>
- *source bin/activate* <Esto activaría el entorno virtual>
- *(venv) pip3 install -r requirements.txt* <Esto instalaría las dependencias de
- *[En caso de cambios al modelo] (venv) python3 censos/manage.py makemigrations*
- *(venv) python3 censos/manage.py migrate* <Esto crearía la BD desde el modelo>
- *(venv) python3 censos/manage.py runserver* <Esto correría el servidor>

7. Gestión del cambio, incidencias y depuración

Protocolo seguido para la gestión de incidencias (issues):

- En caso de recibir una incidencia externa: El gestor de las incidencias las pasara al grupo una vez comprobado realmente que el cambio es necesario. El grupo decidirá a quien se le asigna la incidencia. Será el gestor de incidencia el que cree en el gestor de tareas (trello) la tarea y la asignará a la persona acordada en el grupo. Un ejemplo de todo esto:

Ejemplo de issue que no era necesario ningún cambio:

PabloTinoco commented on 6 Dec 2017

Member

Prioridad: Media

Desde Administración de Votaciones necesitamos la consulta API que nos proporcione la lista de todos los censos.

migueltern added the **enhancement** label on 12 Dec 2017

migueltern self-assigned this on 12 Dec 2017

migueltern removed the **enhancement** label on 12 Dec 2017

migueltern removed their assignment on 12 Dec 2017

migueltern commented on 12 Dec 2017 • edited

Member

La funcionalidad que pedís está resuelta en el GET /filter sin ningún parámetro, se ha añadido en la wiki para explicar con más detalle. https://1984.lsi.us.es/wiki-egc/index.php?title=Administraci%C3%B3n_de_censos_-_17_18_-_G2

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

Milestone

No milestone

Notifications

Unsubscribe

You're receiving notifications because you modified the open/close state.

2 participants


Lock conversation

Se adjunta Url para que pueda comprobar:
<https://github.com/EGC-G2-Trabajo-1718/egc-censos/issues/1>

- En caso de recibir una incidencia interna: Las incidencias internas se pasan al gestor de incidencias para que las describa en github y la asigne al equipo interno encargado en el gestor de tareas (trello). Una vez que el gestor de incidencias ve las tareas con estado finalizado en trello se dispondra a cerrar la issue.

Gestionar autenticación externa #3

Open migueltern opened this issue on 12 Dec 2017 · 1 comment



migueltern commented on 12 Dec 2017

Prioridad: medio
Estado: pendiente.
Descripción: Mediante un parámetro en la URL te puedas autenticar en el censo.

migueltern added the **enhancement** label on 12 Dec 2017

migueltern assigned KirinAnks and Daniellp20 on 12 Dec 2017

Assignees

KirinAnks
Daniellp20

Labels

enhancement

Projects

None yet

Milestone

No milestone

Notifications

Unsubscribe

Se adjunta Url para que pueda comprobar:
<https://github.com/EGC-G2-Trabajo-1718/egc-censos/issues/3>

- En caso de enviar una incidencia: La persona que detecta la incidencia la comunicara al grupo, en caso de que se acepte por el grupo y se crea necesaria, el gestor de las incidencias la documentara y la publicará al sistema de la api donde nos tengan que dar respuesta.

Ejemplo de esto:


EGC-G2-Trabajo-1718 / **autenticacion**

Watch 2 Star 0 Fork 1

Code Issues 19 Pull requests 0 Projects 0 Wiki Insights

Funcionalidad de "obtener los usuarios de un rol/grupo" a partir del nombre del rol/grupo. #7

Closed migueltern opened this issue on 13 Dec 2017 · 1 comment



migueltern commented on 13 Dec 2017

Prioridad: alta
Estado: pendiente
Descripción:
Desde administración de censos necesitamos obtener un ROL/GRUPO para agrupar a los Usuarios de ese ROL/GRUPO en un censo.

Opción 1: Convertir atributo "role" de User en tabla "grupo" y añadir funcionalidad de "/getRole" o "/getGroup":
Opción un poco compleja ya que requiere de cambiar vuestro modelo. Consiste en que el atributo "role" de un User sea un objeto/tabla relacionado con User, o en su defecto que creen el objeto/tabla/clase Group(id, name) para que almacene los valores que vosotros habéis definido como "role" (ASISTENTE, PONENTE, AMBOS, ETC).

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Se adjunta Url para que pueda comprobar:
<https://github.com/EGC-G2-Trabajo-1718/autenticacion/issues/7>

Por último vamos a mostrar un ejemplo de una issue donde se hace referencia a un commit:

The screenshot displays a GitHub issue thread. At the top, a comment by 'migueltern' (Member) from 4 days ago states: 'Prioridad: urgente. Descripción: No tenemos el fichero deploy.enc en nuestro repositorio, según dice vuestra wiki vuestro equipo tiene que proveernos de ella. Somos el grupo censos.' This comment has 'urgente' and 'travis' labels added by 'robgc' 4 days ago. Below, 'robgc' (Owner) comments: 'Para poder añadir el fichero deploy.enc la herramienta Travis CLI requiere la existencia del fichero .travis.yml en vuestro repositorio. No he encontrado dicho fichero en ninguna rama de vuestro repositorio, cuando lo hayáis subido añadiré los elementos necesarios a la rama correspondiente.' Then, 'migueltern' (Member) comments 3 days ago: 'Prioridad: urgente. Descripción: Acabamos de subir el archivo .travis.yml para que podáis añadir los elementos necesarios. 1faed1f'. Finally, 'robgc' (Owner) comments 2 days ago: 'El siguiente commit cierra este issue: EGC-G2-Trabajo-1718/egc-censos@ 1faed1f'. The issue is closed by 'robgc' 2 days ago. On the right sidebar, it shows 'Assignees: No one assigned', 'Labels: travis, urgente', 'Projects: None yet', 'Milestone: No milestone', 'Notifications: Unsubscribe', and '2 participants'.

Se adjunta Url para que pueda comprobar:

<https://github.com/EGC-G2-Trabajo-1718/integracion/issues/3>

El formato para las incidencias es el publicado por el equipo de desarrollo.

Título: <breve título sobre la incidencia>

Prioridad: a seleccionar entre distintos valores: *urgente, alto, medio, bajo*.

Estado: *pendiente, en curso, finalizado*. Los dos primeros estados deberían meterse como etiquetas en GitHub, el último estado se produce cuando se cierra la incidencia en GitHub.

Descripción: <descripción detallada del error>

La descripción puede incluir imágenes o la salida emitida por el fallo.

Etiquetas: <etiquetas de GitHub para clasificar las incidencias>

enhancement: propuesta de mejora

bug: fallos encontrados en el sistema

help wanted: incidencia que puede ser resuelta por un miembro del equipo pero que ha sido atendida previamente por otro

question: (a usar solo entre miembros del equipo) dudas sobre un commit en concreto, hay que referenciar el commit en cuestión

8. Gestión del código fuente

La gestión del código fuente se realiza a través del correspondiente repositorio en github, este es: <https://github.com/EGC-G2-Trabajo-1718/egc-censos> . Cada integrante del equipo posee una rama en el repositorio en el que irá alojando su código. En esta rama se puede hacer un push cada vez que se quiera pero el responsable de hacer ese push será siempre el miembro al que le corresponda la rama.

Una vez que se ha acabado una funcionalidad o una determinada tarea en la rama del miembro este código se queda ahí en esa rama hasta que el encargado de integración del equipo se encarga de revisar el código y pasarlo a la rama de development.

¿Como sabe el encargado de integración del equipo cuando tiene que pasar una rama a development?. El encargado de integración para pasar la funcionalidades a la rama development irá revisando el gestor de tareas usado para el proyecto, si percibe que la tarea esta acabada acudira a la rama del miembro examinará el código y lo pasara a la rama development si corresponde. En el caso de que haya una duda, pregunta o aclaración acudira a un chat creado para tratar de especificar cuándo se harán las reuniones y ciertos temas.

Para que todos los miembros del equipo sepan qué tarea está desarrollando cada uno se creó un tablero en la herramienta Trello.

En la rama development se encuentra el código que esta completo pero esta en fase de desarrollo por si hubiera más cambios, esta rama es la usada en el entorno de desarrollo, es decir que en nuestro IDE usamos la rama development para hacer las correspondientes pruebas además de poder usar cada uno su rama para desarrollar funcionalidades y tareas como pueden ser la realización de tests. Cuando se sabe que ese código no va a ser modificado mas se pasa a la rama master para que se pueda hacer una integración continua y el correspondiente despliegue.

Los commit realizados en las ramas de cada miembro deben de tener un título aclaratorio de que se ha hecho en ese commit, pudiéndose añadir una descripción. Los commits realizados a la rama master que es la que el equipo de integración usará deben de tener un formato como se especifica en la wiki de integración.

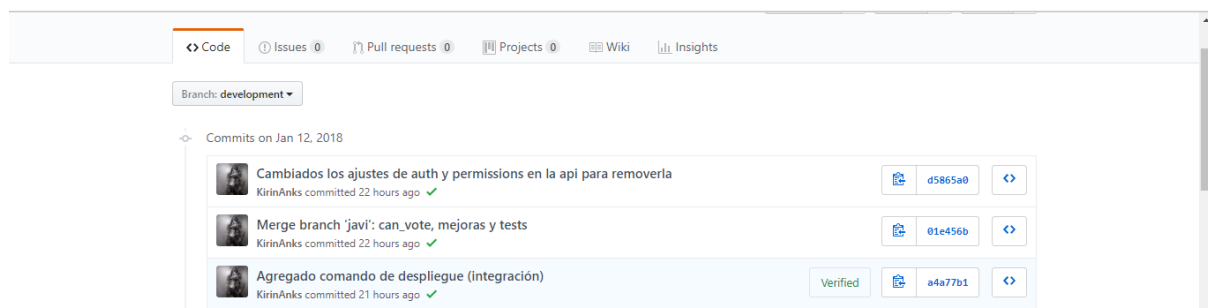
Las versiones del código estarán identificados por los commits en la rama development, pudiendo ser una version inicial la que solo contenga el modelo para la base de datos, otra version podria ser una con el modelo y el serializador, otra con la función get, y así sucesivamente hasta llegar a una versión final y estable.

Un ejemplo de un commit hecho en la rama daniel es el que se muestra en la imagen, estos commit pueden ser intermedios de tareas a realizar o funciones a hacer, es decir desde que se empieza hasta que se acaba una funcionalidad se pueden hacer tantos commits como se

quieran en la rama. La imagen muestra un arreglo a un archivo final que contenía importaciones innecesarias, además de otro commit para arreglar tests y crear más tests. El enlace para ver los commits de esa rama es el siguiente: <https://github.com/EGC-G2-Trabajo-1718/egc-censos/commits/daniel>



Ahora se pondrá un ejemplo de un commit realizado a la rama development, como he dicho anteriormente a esta rama solo se le hacen commits o push cuando esta revisado el código y una tarea concreta acabada en las demás ramas.



Como se ha podido ver en la imagen de arriba se muestran tres commits, dos de ellos son para ajustes previos al despliegue que después serán pasados a master. Se ha realizado un merge de la rama "javi" a development debido a que la rama javi ya tenía la funcionalidad de can_vote acabada con sus tests y otras mejoras. Los commits de la rama development se pueden ver en el siguiente enlace: <https://github.com/EGC-G2-Trabajo-1718/egc-censos/commits/development>

Aclarar que cuando el equipo decide crear una funcionalidad nueva cada miembro deberá de traerse mediante un merge todo el código de development a su rama y luego implementar la funcionalidad si le corresponde.

9. Gestión de la construcción e integración continua

Para la construcción se usan las herramientas provistas por defecto por el framework de Django. Tanto el modelo (BD), conectado a mysql como el servidor, están gestionados por esta herramienta, mediante comandos y archivos de ajustes, podemos proceder a construir el sistema (migrate, runserver, test...), cuyos pasos fueron previamente descritos en el apartado *<Entorno de desarrollo>*. Las dependencias de las que hace uso esta construcción están gestionada por otra herramienta de Python, como es pip, y dichas dependencias están grabadas en requirements.txt.

```
(egc-censos) djangodev@egc:~/Escritorio/egc-censos$ pip3 freeze
certifi==2017.11.5
chardet==3.0.4
Django==1.11.7
djanoorestframework==3.7.3
idna==2.6
mysqlclient==1.3.12
pytz==2017.3
requests==2.18.4
urllib3==1.22
```

```
(egc-censos) djangodev@egc:~/Escritorio/egc-censos$ python3 censos/manage.py migrate
Operations to perform:
  Apply all migrations: admin, api, auth, contenttypes, sessions
Running migrations:
  No migrations to apply.
```

```
(egc-censos) djangodev@egc:~/Escritorio/egc-censos$ python3 censos/manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
January 12, 2018 - 20:30:15
Django version 1.11.7, using settings 'censos.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Para la integración continua se ha seguido la sugerencia del grupo de Integración y se ha usado travis (conectada directamente al código en github), la cual es gestionada mediante el fichero .travis.yml:

```

language: python

python:
  - 3.6

services:
  - mysql

env:
  - DJANGO_VERSION=1.11.7

before_install:
  - mysql -e 'CREATE DATABASE IF NOT EXISTS censos;'
  - mysql -uroot censos < create_database.sql
install:
  - pip install -q Django==$DJANGO_VERSION
  - pip install -r requirements.txt

script:
  - python censos/manage.py test censos/api/

```

**** poner captura del nuevo****

Dicho fichero contiene las instrucciones para instalar las dependencias necesarias. Según se ha podido observar en la documentación oficial de travis, para python NO ES NECESARIO añadir los ajustes y creación del virtualenv (previamente explicado en el apartado Entorno de desarrollo), ya que dicho servicio está automatizado.

Antes de ejecutar el script de pruebas, se instalan las dependencias mediante pip y se crea la base de datos, para luego ejecutarse los tests e informarnos del correcto funcionamiento de la última integración hecha en development / master.

EGC-G2-Trabajo-1718 / egc-censos build passing

[Current](#) [Branches](#) [Build History](#) [Pull Requests](#)

✓ **master** Merge branch 'development'

→ #105 passed

→ Commit a214dbd [↗](#)

🕒 Ran for 43 sec

🔗 Compare eace269..a214dbd [↗](#)

📅 39 minutes ago

🌿 Branch master [↗](#)

👤 KirinAnks authored and committed

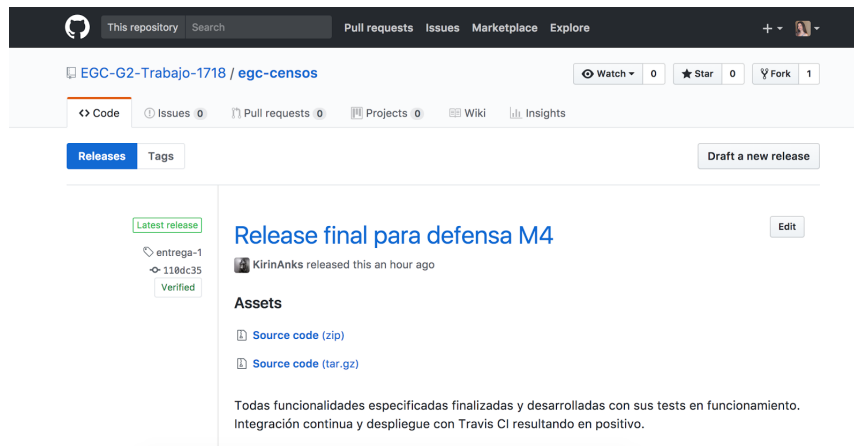
10. Gestión de liberaciones, despliegue y entregas

Gestión de liberaciones:

BASELINE	
CENSOS 20.11	Creación del repositorio
CENSOS 23.11 v0.0	<p>Nada de código. Los lenguajes elegidos para la elaboración del proyecto son:</p> <p>Lenguaje/Herramienta: Django 1.11 -> Python 3.6</p>
CENSOS 1.12 v1.1	Estructura principal del proyecto. Settings, virtualenv, dependencias...
CENSOS 4.12 v1.1	Creado modelo 'Censo' para la base de datos.
CENSOS 12.12 v1.1	Agregado un nuevo campo, id, al modelo para la base de datos.
CENSOS 16.12 v1.1	<p>Creación de los métodos CRUD</p> <ul style="list-style-type: none">• create• delete• update• filter• get
CENSOS 4.1 v1.1	Realización de los tests para los métodos CRUD
CENSOS 10.1 v1.1	Añadiendo fichero .travis.yml para el despliegue con TRAVIS
CENSOS 10.1 v1.1	Modificado un campo del modelo para la base de datos a petición del grupo de autenticación.
CENSOS 12.1 v1.1	Mejora de los tests
CENSOS 13.1 v1.1	Agregado comando de despliegue que nos proporciona el grupo de integración.

Gestión de entrega:

Se entregará a través de una release desde github, que se refiere al último commit donde estaría toda la funcionalidad.



Gestión del despliegue:

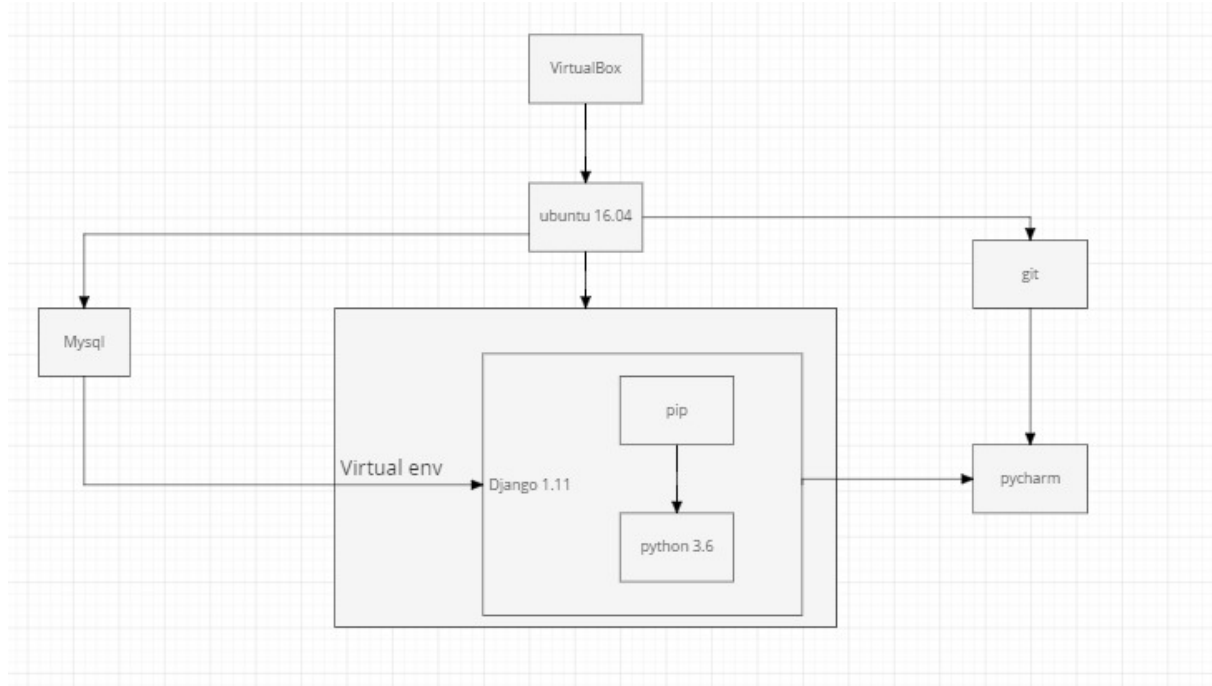
1. despliegue local

Se gestiona mediante lo que se ha especificado en apartados anteriores, instalando las dependencias con el entorno virtual y corriendo el servidor con *manager.py runserver*.

2. Despliegue remoto

Se hace a través del fichero `.travis.yml` con las configuraciones que se ponen en el mismo fichero, las líneas que están comentadas para el despliegue. Se hace a partir de la rama master.

11. Mapa de herramientas



VirtualBox: Con esta herramienta se virtualiza el sistema operativo ubuntu, que contendrá en local la api censos y todo el software necesario para esto.

Ubuntu: En el sistema se instalarán tanto python, pip, django y el framework.djangorestframework 3.7.3, pycharm, github como

django: contiene django framework con un conjunto de herramientas potente y flexible para crear API web.

Mysql: base de datos que usará el framework django.

virtualenv: aislar recursos como librerías y entorno de ejecución, del sistema principal.

python: lenguaje de programación con el que se desarrollará la aplicación

pip: herramienta que instalará los paquetes necesarios de python.

pycharm: Entorno de desarrollo con el que usará el framework django.

github: para la gestión de versiones del código, al cambiar en nuestro entorno de desarrollo cualquier clase detectará el cambio.

12. Ejercicio de propuesta de cambio

El ejercicio propuesto para nuestro proyecto será añadir la información de Comunidad autónoma para saber la localización de ese censo.

Para realizar dicho cambio los pasos a seguir son los siguientes:

1. Primero el gestor de incidencias la tramitará. Esto conlleva a que le dé el formato definido. También definirá este cambio en la herramienta que usamos, llamada Trello, y se le asigna a cada persona su parte correspondiente.
2. Como este cambio lleva modificar el modelo de datos, funcionalidades, test y descripción de la API. Todas estas tareas se pondrán en Trello y se asignará a cada persona correspondiente dependiendo de su funcionalidad.

Gestión de la incidencia y tareas-> Miguel

Documentación de la API -> Miguel

Modelo de datos y serializer -> Javi

Funcionalidad de /create -> Laura

Funcionalidad de /update -> Daniel

Funcionalidad de /filter -> María

Test de /create -> Laura

Test de /update -> Daniel

Test de /filter -> María

Integración a development, master y prueba de CI -> Javi

Cada desarrollador habrá de correr el servidor y los tests localmente para comprobar su correcto funcionamiento antes de comunicar que se ha completado la tarea y de hacer su correspondiente push.

Una vez que todo esté probado y estemos seguros de que todo funciona correctamente se integrará a la rama development, donde tras correr las pruebas

integradas en local se hará push y posteriormente comprobado que la CI en Travis resulte positiva. Cuando todo esté bien, se pasará a master para que se tenga en cuenta en la versión de despliegue.

13. Conclusiones y trabajo futuro

Todo el desarrollo del proyecto está basado en la integración continua de un producto software que está en constante evolución. Trabajar en un repositorio común con el resto de compañeros y apartar nuestros entornos y herramientas para que estén en común con los demás no han sido tareas en absoluto sencillas.

A continuación, detallaremos algunas de las lecciones aprendidas durante el desarrollo de la asignatura:

- GitHub: La mayoría de los integrantes del grupo estábamos acostumbrados a usar SVN como gestor de versiones y profundizar en el uso de GitHub nos va a hacer empezar a usar este repositorio día a día. Su sencillez y la gran cantidad de funcionalidad que ofrece se ve acrecentada por su toque social. Compartir un repositorio con el resto de la comunidad siempre es beneficioso para todos si se hace como es debido.
- Código heredado: No hemos trabajado con código heredado ya que se nos aconsejó que no lo hiciéramos debido a que el código del año anterior era bastante inestable. Por lo tanto, hemos empezado un proyecto desde 0. Esto no es común que se haga hoy en día en cualquier trabajo relacionado con la informática, lo normal es tener ya un código con el que empezar a trabajar.
- Gestión de incidencias: Al principio del proyecto, se elaboró un documento en el cual se explica cómo íbamos a gestionar las incidencias ocurridas durante el desarrollo del mismo. A Miguel Ternero Algarín, miembro del grupo se le asignó el rol de gestor de incidencias, el cual detecta la incidencia y se la comunica al resto del equipo. También se desarrolló un formato para las incidencias.

Propuestas de mejora de funcionalidades futuras:

- Incluir un buscador de censos.
- Añadir si se precisa más campos al modelo de datos.
- Especificar más a fondo quién puede votar y quién no

14. Anexos

- [wiki censos](#)
- [Trello](#)
- [github censos](#)
- [Integración continua y despliegue de la API censos](#)