## 1. Resumen ejecutivo

Nos hemos encargado del subsistema censo , el cual se encarga de asociar los votantes con las votaciones para poder votar. Para agilizar el proceso del mismo hemos realizado un frontend no autogenerado por django para hacer la interfaz más amigable, también hemos incluido las funcionalidades de exportar e importar de manera que se pueda crear y revisar los censos rápidamente, y por último hemos implementado una función probabilística permitiendo estimar en este caso, la intención de voto de los usuarios en una votación dada.

## 1.1. Indicadores del Proyecto

| Miembros del equipo | Horas dedicadas | Commit realizados | Test | Líneas de código | Issues | Incremento funcional |
|---|---|---|---|---|---|---|
| Rafael Giraldez Liébana | 30 | 19 | 9 | 145 | 7 | Referente al trabajo que he realizado en el proyecto, casi la totalidad del tiempo se basó en la búsqueda de distribuciones probabilísticas para añadirlas al proyecto y que fuesen usadas por otros subsistemas, que fue algo más trivial. Además, implementé tests que confirmasen el correcto funcionamiento de estos métodos y he realizado varios documentos requeridos para el entregable. Mis últimas tareas se basaron en conseguir que travis diese en todo momento un build correcta. |
| José Antonio Sosa Cifuentes | 50 | 21 | 5 | 550 | 9 | Durante el tiempo de desarrollo, conseguí hacer una página personalizable para censos que te permite ver, crear y eliminar censos. Además, hice una página de inicio de sesión para censo, hice tests de selenium sobre la página que he creado, y me he peleado con travis y heroku, consiguiendo que la aplicación desplegara en heroku, y los test ejecutaran en travis. |
| Jesús Elías Rodríguez | 31 | 17 | 6 | 106 | 8 | En cuanto al desarrollo me he dedicado a incluir las funcionalidades de importar y exportar censos, las cuales han sido implementadas de manera que se puedan realizar en tres formatos diferentes(csv , json y yaml).También , he ayudado a José Antonio un poco a realizar la página personalizable. |

## 2. Integración con otros equipos:

En este proyecto planeamos hacer una integración con los otros tres equipos presentes en el proyecto Giratina, estos equipos son los siguientes:

- Giratina-Almacenamiento
- Giratina-Visualización
- Giratina-Votación

La integración con todos estos equipos se llevará a cabo utilizando un repositorio común para los cuatro grupos componentes del proyecto, cada uno con una rama propia, para luego hacer un merge a una rama master común para todos, todo esto se explicará más adelante en el apartado de gestión del código fuente. Además, se utilizara telegram como mecanismo de comunicación entre los cuatro equipos.
Una vez realizado el desarrollo por parte de los equipos se procederá a unir los módulos de cada grupo, para comprobar el funcionamiento de la integración se realizarán pruebas de integración que consistirán en realizar el proceso completo de votación en la página. Si estas pruebas son exitosas se finalizara el proceso de integración.

# 3. Descripción del sistema

El sistema que hemos desarrollado se trata de Censo, el cual es un subsistema de decide, una plataforma de voto electrónico.

Censo dentro de esta plataforma se encarga concretamente en decidir quién puede o no votar en una votación, por lo que está relacionado fuertemente con autentificación debido a que tienen que haber usuarios registrados en la plataforma para poder darle ese permiso de votar o no en una votación, y por otro lado con votación , ya que si no existen votaciones a las que asignar votantes censo no podría ejecutar su labor. Estos subsistemas se interconectan implementando una API, y solo habría que coordinar el desarrollo con los otros subsistemas, si los cambios/mejoras realizadas por nosotros les afectarán directamente.

Los cambios que hemos desarrollados en censo son los siguientes:

-Realizar todas las funciones de censo desde un frontend no autogenerado por django , más concretamente hemos realizado un frontend donde se muestran una lista de censos, donde se muestra el id del censo, el nombre del votante y el nombre de la votación a la que puede votar , además pueden ser eliminados y agregados fácilmente.

-Hemos ampliado las funcionalidades de este subsistema , agregando una posible importación y exportación de los datos del censo . De esta forma podemos importar y exportar censos en formatos csv ,json y yaml directamente a decide.

-Hemos implementado la función probabilística de la distribución de la Normal ,permitiendo estimar en este caso, la intención de voto de los usuarios en una votación dada.

*Nota: En un principio íbamos a realizar más cambios y mejoras en censo , pero no hemos podido llevarlos a cabo ,debido a la baja de tres miembros de nuestro equipo .

Como se puede observar en los cambios realizados anteriormente , no existe ninguno que afecte directamente a otro subsistema , por lo que no nos hemos tenido que coordinar en cuanto al desarrollo del mismo .

# 4. Entorno de desarrollo:

En un principio, en nuestro equipo de trabajo se propusieron distintos entornos de desarrollo, entre ellos se encontraban Eclipse, Atom y Pycharm. Tras la primera toma de contacto con decide, y después de hacer la primera reunión de grupo, decidimos que la mejor forma de afrontar este proyecto era con **Pycharm**, ya que este IDE está desarrollado principalmente para trabajar con Python y te ayuda a ser más productivo debido a que cuenta con un terminal dentro de la propia interfaz, con una consola Python interactiva, con asistencia inteligente a la codificación y otras muchas funcionalidades y herramientas más.

Además, gracias a que somos estudiantes de universidad podemos obtener gratis la versión profesional de este IDE, actualmente se encuentra en la **versión 2019.3.**

Una vez descargado Pycharm, aparecerá una carpeta en la cual hay un archivo txt (Install-Linux-tar) donde se explica paso a paso como instalar Pycharm de una manera muy sencilla a través de la consola. Lo único que tendremos que hacer será ejecutar el comando ./pycharm.sh dentro de la carpeta bin para que se nos inicie el IDE.  Una vez dentro de Pycharm solo tenemos que abrir el proyecto con el que vayamos a trabajar y seleccionar el intérprete de Python que queramos utilizar, en nuestro caso se trata de **Python3.6.** Para tener una mayor seguridad frente a conflictos entre diferentes versiones, tenemos instalado un entorno virtual en el cual se encuentra el intérprete de python mencionado anteriormente.

Para crear un entorno virtual solo tenemos que ejecutar el siguiente comando :

- python3 -m venv <myenvname>

Una vez creado, para trabajar dentro de este entorno tenemos que ejecutar:

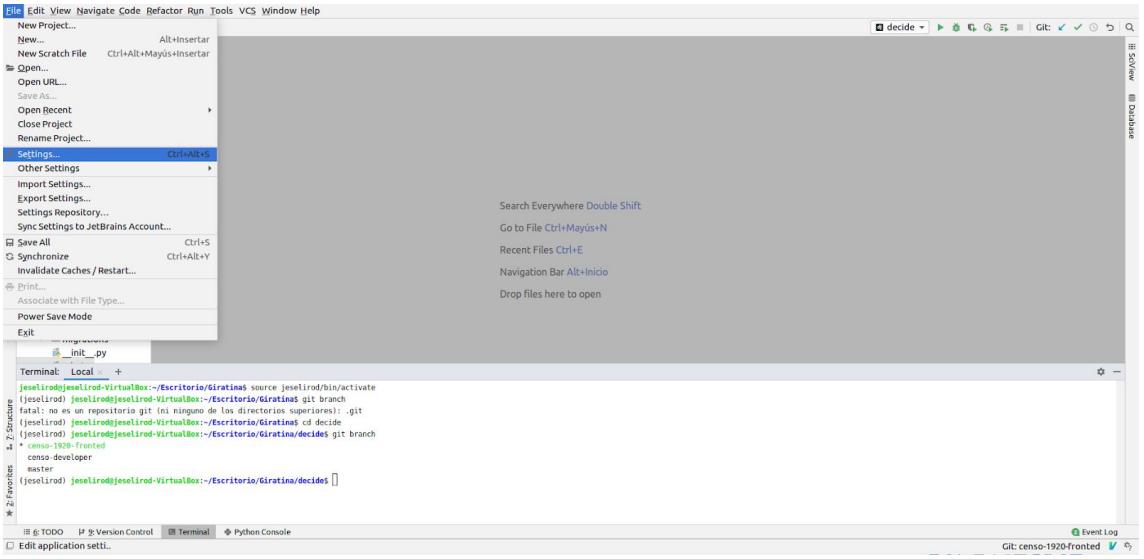
- source <myenvname>/bin/activate

Y en caso de querer salirnos de este entorno :

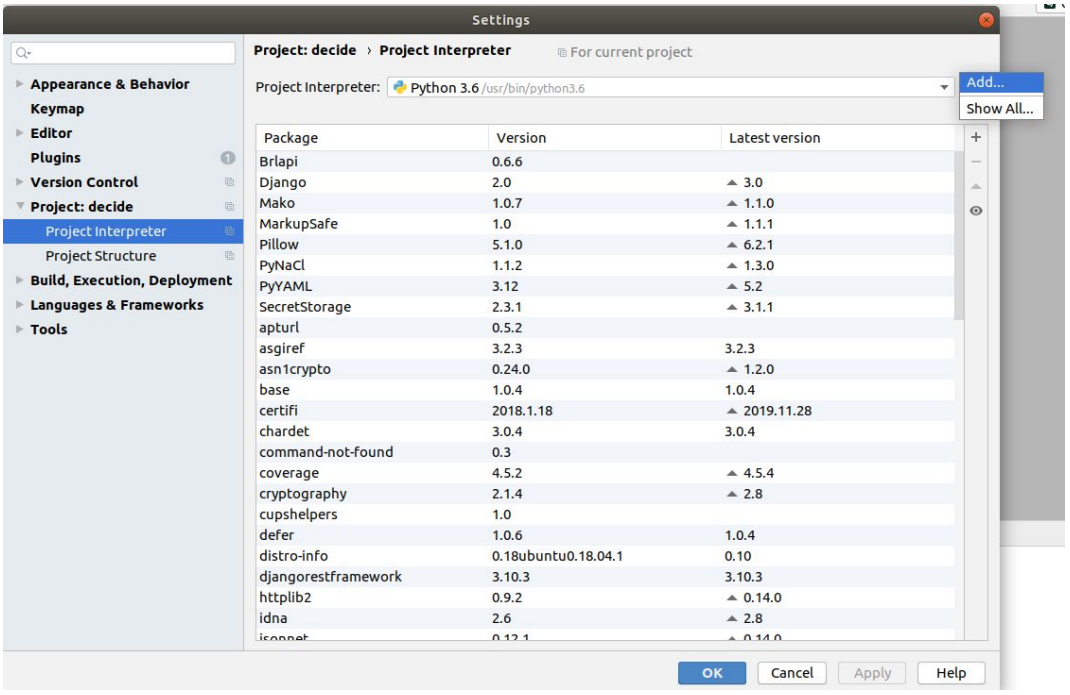
- deactivate

Por último, para trabajar desde Pycharm con el intérprete de python  que tenemos dentro del entorno virtual tendremos que :

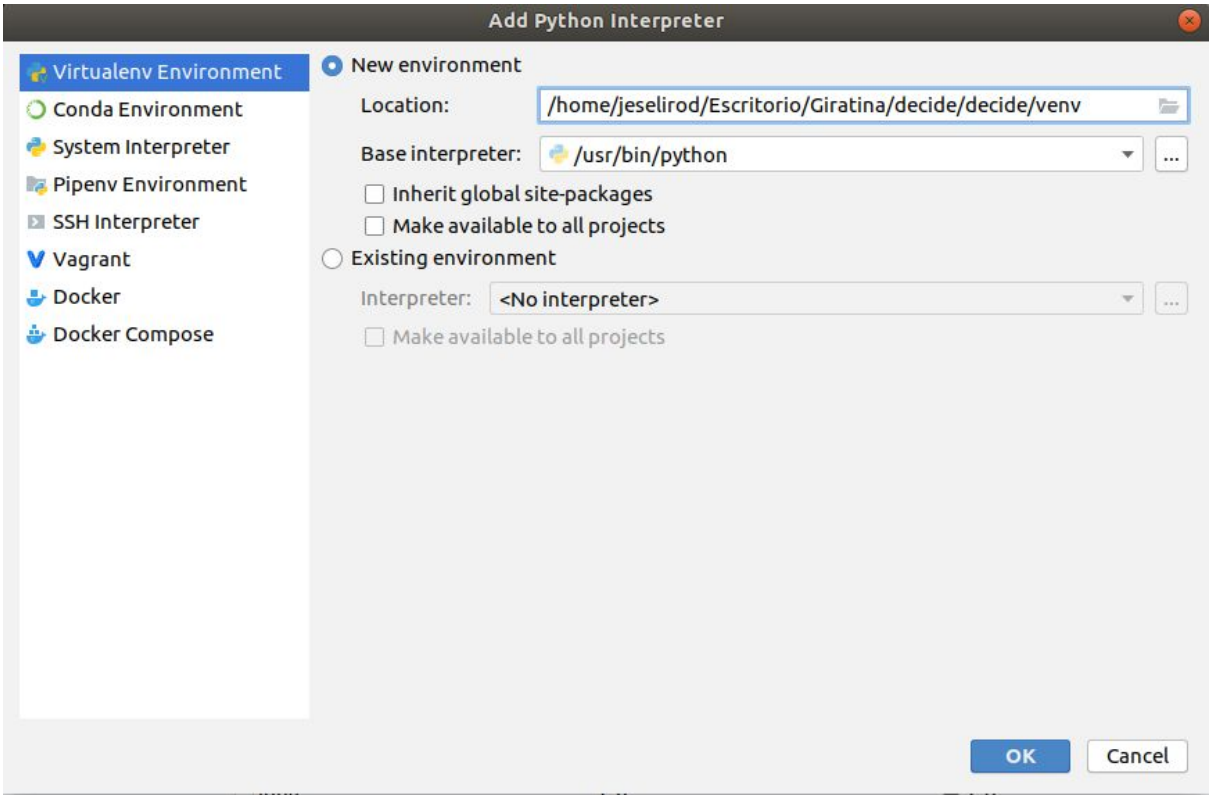
1. Irnos al menú superior de Pycharm y seleccionar file
2. Seleccionar setting




3. Dentro de la ventana settings , irnos a nuestro proyecto y seleccionar project interpreter
4. Pinchar en Add…



5. Dentro de Virtualenv Environment añadir el intérprete que tengamos dentro de nuestro entorno virtual, en el apartado Existing environment.




# 5. Gestión de incidencias

## 5.1. Objetivo

En este apartado se definirán los distintos significados que tiene el término **incidencia,** así como los distintos modos de gestionarla.

## 5.2. Definiciones

**- Incidencia:** Cosa que se produce en el transcurso de un asunto, un relato, etc., y que repercute en él alterándolo o interrumpiéndolo. En el contexto de una empresa se pueden encontrar diversos tipos, entre los cuales podemos encontrar las sugerencias, quejas y reclamaciones relacionadas con nuestros clientes, proveedores o personal, como en labores de corte o entrega. Las incidencias podrán tener su origen en el incorrecto diseño de uno o varios procesos o en la incorrecta ejecución de estos.

**- No conformidad:** falta de cumplimiento de los requisitos fijados para el proyecto.

- **Reclamación:** comunicación de una falta de satisfacción en un servicio prestado (plazo de entrega, falta de documentación) o un producto suministrado (software defectuoso).

**- Tratamiento de una no conformidad:** acción que se realiza para tratar un elemento o proceso que presenta una “no conformidad” con el fin de resolverla.

**- Acción correctora:** acción que se toma para solucionar una “no conformidad” concreta ocurrida.

## 5.3. Gestión de incidencias internas al equipo

Para la resolución de incidencias de manera interna al equipo, nuestro grupo de trabajo ha decidido trabajar en el fork **EGC-GIRATINA/decide**. Dentro de este se ha creado un proyecto llamado **Project-Giratina-Censo** con su respectivo tablero para organizar las distintas issue a realizar por el equipo, y en caso de encontrar conflictos poder asignar tareas de corrección a los distintos miembros del equipo.

El proceso sería el siguiente:

1.- Un miembro del equipo descubre un posible fallo en la aplicación.

2.- Se verifica que el fallo es real verificando con los requisitos del sistema.

3.- Se accede a GitHub, y se crea una issue con un título que especifique de manera simplificada el problema encontrado.

4.- Dentro de la descripción se explicará el problema con más detalle y se asignará a dicha tarea a los miembros del equipo implicados en la realización de dicha parte que está generando problemas.

5.- El equipo empezará a realizar la tarea de corrección, pasando dicha issue a la columna “In Progress”.

6.- Una vez solucionado el problema se pasará la tarea a la columna “Done”. En función de la importancia del fallo, el equipo expondrá los cambios pertinentes al resto de sus compañeros.

7.- Por último, recogeremos dicha incidencia junto con su solución en este documento de control.

## 5.4. Gestión de incidencias externas al equipo

Para la gestión de incidencias externas al equipo se ha optado por hacer uso de un documento de solicitud de cambio para los distintos equipos del proyecto **Decide-Giratina.**

Los pasos a realizar para notificar una incidencia externa al equipo serán los propuestos en el siguiente documento de solicitud de cambio:

# SOLICITUD DE CAMBIO

| Datos | |
|---|---|
| Fecha petición | |
| Grupo que solicita el cambio | |
| Grupo al que se le solicita el cambio | |
| Resultado de solicitud (Aceptado/Rechazado) | PENDIENTE |

## Formulario que rellenar

| Datos del solicitante | |
|---|---|
| Nombre | |
| Apellidos | |
| Correo de contacto | |

Como podemos observar en esta primera parte del documento, se deberá rellenar la primera tabla en la que especificaremos la fecha de petición del cambio, junto con el grupo solicitante y el grupo al que se le solicita el cambio. Dicha solicitud estará por tanto pendiente de arreglo.

El solicitante deberá rellenar además una tabla con sus datos personales para tener constancia de las personas solicitantes de cambios.

| Qué se solicita |
| --- |
| -<br>-<br>-<br>-<br>- |

| Para qué se solicita |
| --- |
| |

**Firmado:**

| Observación del equipo al que se le solicita |
| --- |
| |

**Firmado:**

En esta segunda parte del documento se deberá especificar los cambios o problemas encontrados en la aplicación del otro equipo y poner para qué se solicita.

Realizaremos esto en cada una de las incidencias para informar al equipo de las posibles influencias que produce el error en nuestro sistema, para así poder solucionarlo lo más rápido y ordenadamente posible.

En la casilla de observaciones al equipo que se le solicita el cambio, intentaremos comunicarles posibles observaciones que le faciliten el trabajo sobre el arreglo a tratar.

Finalmente, el equipo **Giratina** se ha organizado para mandar estas peticiones de cambios externos por correo electrónico a uno de los miembros del equipo, que será el encargado de comunicar la incidencia a sus compañeros. En caso de no recibir respuesta, como medida excepcional se comunicará mediante un grupo de Telegram compuesto por todos los integrantes.

# 6. Gestión del código fuente:

Como ya explicamos anteriormente en el apartado de entorno de desarrollo el IDE que utilizaremos será **pycharm,** con esta herramienta gestionaremos todo el código de nuestro proyecto, todo este código se gestionará de la siguiente manera:

- Dentro del proyecto decide hay varias carpetas, como pueden ser census, mixnet, base, visualizer, etc… Sin embargo nuestro equipo encargado del censo solo trabajara en la carpeta census dentro de decide, esta carpeta se corresponde con nuestro módulo.
- Dentro del nuestro módulo se utilizará la carpeta templates para colocar todas las vistas html relacionadas con nuestro módulo.
- En la carpeta static dentro de decide se colocaran todos los archivos CSS relacionados con nuestro módulo.

## 6.1. Gestión del repositorio general

Como herramienta de gestión de versiones utilizaremos git en la cual crearemos un repositorio común para todo el proyecto Giratina, la forma de gestionar este repositorio se hará de la siguiente manera:

- Dentro del repositorio se creará la rama **Master2** en la cual se harán los merge de todas las ramas de los demás módulos.
- Cada módulo tendrá asignado a un Merge Manager que será el encargado de realizar los merge a la rama principal del proyecto.
- Cada módulo poseerá una rama propia en la cual desarrollar su parte del proyecto, esta rama será gestionada internamente por el equipo en cuestión.

## 6.2. Gestión de ramas

Nuestro módulo encargado de Censo gestionará su rama de la siguiente manera:

- Nuestra rama principal será **Censo-1920-Giratina.**
- De esta rama principal crearemos una rama secundaria llamada **censo-developer**, en esta rama realizaremos los merge de aquellas terceras ramas finalizadas.
- De la rama **censo-developer** crearemos otras terceras ramas en cada una de esas ramas cada miembro desarrollara una funcionalidad concreta de nuestro módulo.
- Una vez la funcionalidad de esas terceras ramas esté finalizada y sin errores se hará un merge por el Merge manager a la rama developer.
- Las terceras ramas deberán seguir el patrón de nombres Censo-<Nombre corto de la funcionalidad>-<Nombre de la persona que la realiza>. Ejemplo: Censo-CambiarId-Adrian.
- Cuando la rama developer contenga una versión estable del módulo de censo se hará un merge a la rama principal.
- Una vez estén realizados todos los cambios al módulo Censo se hará un merge a la rama principal del proyecto Giratina.

## 6.3. Gestión de los commits.

La gestión de los commits de nuestro proyecto se va a realizar de la siguiente manera:

- Se realizará un commit una vez que se haya añadido una parte de la funcionalidad, siempre y cuando esta parte funcione.
- Solo se realizarán push al repositorio cuando una funcionalidad esté acabada.
- Nunca se realizarán push al repositorio si el proyecto contiene errores.
- El patrón de mensaje que debe seguir cada commit debe ser el siguiente: Como título del commit deberá incluir [Censo] [Nombre corto del cambio realizado][Nombre de la persona] y en el cuerpo del commit se deberá explicar el cambio realizado.

## 6.4. Gestión de las pruebas

La gestión de las pruebas las realizaremos de la siguiente manera:

- Habrá unas pruebas generales que se encargará de comprobar la funcionalidad del módulo al completo.
- Cada persona responsable de una funcion del modulo creara las pruebas de dicha funcionalidad una vez finalizada esta.
- Antes de solicitar al merge manager realizar el merge a la rama developer se deberán realizar pruebas a las funcionalidades implementadas
- Al hacer el merge se ejecutarán las pruebas del módulo para comprobar que lo añadido no altera la funcionalidad general del módulo. Si al ejecutar las pruebas estas salen erróneas se reportara la incidencia al equipo y se creará un issue asignado a la persona que creó la funcionalidad.

  .

## 6.5. Ejemplo de gestión

A continuación se detalla un ejemplo de cómo gestionar el repositorio y código:

- Se me ha asignado la tarea de crear una vista para la visualización de las listas de censo, para llevar a cabo esta tarea primero creó una rama desde developer llamada Censo-VisualizarListas-Adrian, en esta rama realizo todos los cambios oportunos colocando todos los html en la carpeta templates. Una vez he realizado bastantes cambios, ya la funcionalidad está hecha y lo único faltante sería modificar el html para dejar la vista más bonita, hago un commit en el cual pongo de titulo [Censo] [Visualizar vistas funcionalidad] [Adrián] y de cuerpo describo dicha funcionalidad y comentó que falta modificar un html. Al día siguiente finalizo la modificación del html y hago otro commit, de título [Censo][Visualizar vistas finalizado][Adrian] y de cuerpo comento que esta vez ya he finalizado lo que faltaba en el commit anterior. Luego de esto creo las pruebas necesarias para comprobar la funcionalidad que he añadido, hago un commit llamado [Censo][Visualizar vistas pruebas][Adrian] y en el cuerpo describo las pruebas que he realidado. Esta vez al haber finalizado la tarea y las pruebas hago un push al repositorio y comunicó al Merge manager que mi rama está finalizada. El merge manager hará merge de mi rama a developer, hará las pruebas necesarias y si ve oportuno otro merge a la rama principal.

El enlace a nuestro repositorio es el siguiente: https://github.com/EGC-GIRATINA/decide.git

# 7. Gestión de la integración continua

## 7.1. Objetivo

En este apartado se definirán las pruebas y los procedimientos a seguir para confirmar y añadir mejoras sustanciales al sistema de votación Decide.

## 7.2 Pruebas locales y remotas con TravisCI

Una vez que se halla integrado algún cambio en el código que aporte alguna mejora, se actualizará el documento *manage.py* añadiendo pruebas sobre dicha mejora. Tras esto, gracias al comando *python manage.py test* lanzado desde consola se realizarán todas las pruebas del sistema de forma local.

Tras comprobar que las pruebas son correctas, se realizará un *push* hacia la rama principal del subsistema, provocando el lanzamiento de las pruebas automáticamente en remoto en TravisCI. Los resultados se pueden comprobar junto al *commit* en GitHub.

Seguidamente se realizará un merge de la rama local a la rama *developer* que es común a todo el sistema y se revisará que todo está correcto en TravisCI. Ahora será un coordinador el encargado de realizar un *pull request* de la rama *developer* a la rama *master* siempre y cuando antes se halla asegurado que el cambio no sólo es efectivo, sino que no aportará ningún error al sistema.

## 7.3. Pruebas sobre vistas

Antes de realizar la *pull request* anteriormente mencionada, ya sea el coordinador o quien ha desarrollado la mejora, deberá también reportar que las pruebas en cuanto a la vistas genera un build correcta.

Para dichas pruebas, en este caso, usaremos *Selenium*, un framework que nos permite simular la actuación de un usuario final frente a nuestra aplicación.

En cuanto a las pruebas, podemos realizarlas de forma autónoma o ejecutarlas desde dentro del propio sistema. El mecanismo para realizar dichas pruebas es similar a como habíamos hecho con las pruebas locales en cuanto a nuevas funcionalidades en el **apartado 1**.

## 7.4. Despliegue en Heroku a través de TravisCI

Anteriormente se mencionó a TravisCI para la realización de pruebas en remoto. No obstante, TravisCI no sólo servirá para esto, sino que también se encargará de subir la aplicación a Heroku una vez la build de las pruebas es correcta.

Para esto, primero se deberá configurar correctamente el archivo *requirements.txt* y el archivo *settings.py*, que servirá para establecer la configuración de despliegue en Heroku. Además, será necesario crear el archivo *travis_local_settings.py* para guardar los valores del settings para TravisCI.

Tras todo esto, se creará el archivo *Procfile* y se modificará el archivo *.travis.yml* para que Heroku pueda permitir la comunicación con TravisCI (se deberá añadir un API key). Una vez configurado todos estos componentes, cada vez que se realice un *commit* y las pruebas se realicen correctamente, se subirá a Heroku de manera automática a través de TravisCI.

# 8. Gestión de las liberaciones, despliegue y entregas:

## 8.1. Liberación

Entre los grupos encargados de hacer mejoras para el grupo de Giratina para el proyecto de votaciones on-line “decide”, se decidió que, una vez realizados los incrementos que consideremos que funcionen como es esperado por cada grupo, se hace un “merge” entre la rama de desarrollo de el proyecto en general y el de cada grupo, para así probar que todo el código de todo el mundo funciona, y así hacer la integración continua. Una vez todo el código funciona, se le pone a esa release una etiqueta de la forma “X.Y”, donde X corresponde con el milestone en el que nos encontremos, y la Y representa el número de release dentro de ese milestone.

Trístemente, por falta de tiempo además de la falta de personal, ese tipo de release no han podido realizarse por parte de nuestro equipo.

## 8.2. Despliegue

Para la gestión de despliegue, lo que se ha acordado es crear una rama nueva en github llamada **“censo-heroku”** en la que, tras pasar las pruebas pertinentes, se realice un merge de esta rama a la rama de desarrollo, y desde aquí se realizan los cambios deseados para poder desplegar en Heroku. Para ello, tras realizar los cambios, hacemos un push a la rama master de heroku ejecutando en la terminal el comando **“git push heroku censo-heroku:master”**. Una vez que heroku nos de la confirmación de que el proyecto esté desplegado, entonces podremos considerar que el proyecto está desplegado.

## 8.3. Entrega

Para cada uno de los milestones, los documentos que se piden tienen que estar finalizados y el sistema tiene que funcionar de acuerdo con nuestros criterios de calidad. Para ello, antes de la fecha límite de entrega de cada milestone, los documentos requeridos tienen que estar finalizados y los incrementos tiene que ejecutarse sin errores detectables.

# 9. Ejercicio y propuesta de cambio

Para la realización de este apartado se detallarán los pasos seguidos para implementar la distribución Normal que nos aportará una posible muestra con las personas que votarían la opción sobre la que buscamos.
Inicialmente, se abrirá un incidencia en el formato propuesto y seguido por todo el grupo de trabajo, quedando de la siguiente manera:
**TÍTULO: [CENSO][IMPLEMENTACIÓN] MÉTODOS PROBABILÍSTICOS DE INTENCIÓN DE VOTO**
Descripción: Se procederá a implementar la función probabilística de la distribución de

la Normal permitiendo estimar, en este caso, la intención de voto de los usuarios en una votación dada.
De la misma forma, se implementará la función de la distribución Normal para poblaciones mucho mayores para, de nuevo, estimar el tamaño de la población (muestra) que votará una opción en una votación dada. (Para realizar el correcto funcionamiento de este método por parte de otro subsistema, se deberá tener en cuenta el uso de una tabla de Distribución Normal para un nivel de confianza dado).
Asignees: rafgirlie.
Labels: enhancement, censo.
Projects: Project-Censo-Giratina.
Tras realizar esta issue/incidencia se procederá a incluirla en el árbol de trabajo To Do y posteriormente, en cuanto se trabaje en ella, se cambiará al árbol WIP (Work In Progress). Teniendo presente que el repositorio esté a punto, de manera local, para trabajar en el proyecto, se creará una rama para trabajar sobre el cambio.
La secuencia de comandos que usaremos será:
**$ git branch normal_distribution**
**$ git checkout normal_distribution**
Tras esto, tenemos todo preparado para trabajar el código de nuestro cambio, realizando dicho código en los archivos models.py y tests.p en la carpeta census del proyecto.
Una vez implementado todo, y habiendo añadido los tests correspondientes, se comprobará que lo implementado funciona correctamente con el comando: $ python3 manage.py test census
En caso de que surjan errores habrá que corregirlos hasta recibir el “OK” por parte de la terminal desde la que lanzamos el comando. Por otro lado, si los tests y los métodos funcionan correctamente se procederá a realizar un commit, push y merge para incluir este cambio en la rama censo-developer, por tanto los comando a seguir serán:
**$ git add *;**
**$ git commit -m “Implementacion distribucion normal”;**
**$ git push –set-upstream origin normal_distribution**

Todo esto se realiza desde la rama local creada anteriormente, que ahora ya está subida al servidor. A continuación, nos moveremos a la rama de censo-developer que es la rama principal de nuestro subsistema y realizaremos el merge siguiendo esta lista de comandos:
**$ git checkout censo-developer;**
**$ git pull;**
**$ git merge normal_distribution**

Si al realizar el merge surgen conflictos, desde esta misma rama, se procederá a solucionarlos y se volverá a realizar un commit y el push correspondiente. Es importante tener en cuenta que, al subir a la rama principal de censo una implementación, la build generada en Travis debe ser correcta. De nuevo, en caso de que no sea así, se volverá a operar sobre la misma la rama para arreglar el conflicto y de nuevo se seguirán los comandos pertinentes para la subida al servidor. Por otro lado , si la build generada por Travis es correcta , el mismo se encargará de desplegar

la versión commiteada a Heroku .
Como paso final, se cerrará la issue/incidencia relacionada con esta implementación y se pasará al árbol de trabajo Done.

# 10. Conclusiones y trabajo futuro

Tras ver cómo fue el trascurso del desarrollo y cómo hemos llegado al final de nuestro proyecto en la asignatura, tras encontrarnos ante los problemas que nos hemos encontrado, podemos destacar algunas conclusiones a partir de los problemas que nos hemos encontrado a lo largo del desarrollo de este proyecto.
El problema más destacado que nos hemos encontrado ha sido la falta de actividad de algunos miembros del grupo, algunos de ellos no haciendo nada destacable, y la mitad de los integrantes del grupo abandonando el proyecto a unas semanas de la entrega final. Así, un grupo de 6 personas se convirtió en uno de 3 a unas semanas de entregar el proyecto y sin haber hecho nada destacable en él, haciendo que algunos miembros del proyecto, esperando que hubieran tenido algo hecho que funcionase para que el resto del grupo pudiera utilizarlo, se encontrara con un proyecto incompleto, con muchos puntos indispensables por hacer, y a solo unas semanas de entregar, aumentando su carga de trabajo mucho más de lo que debería para este trabajo.

Con esto, podemos sacar la conclusión de que se necesita un mayor control del trabajo realizado por los compañeros del grupo. Se optó por una estrategia de confianza en todo el mundo, la cual funciona bien cuando todos los miembros trabajan de forma continua y en armonía con el resto de integrantes del grupo, pero que aquí habría sido necesaria otra estrategia, como pueden ser los daily standups, o reuniones semanales de control.

Por otro lado, habría sido necesario un protocolo de estimación de abandono de los miembros del grupo. Esto quiere decir que a lo mejor habría que haber exigido un tiempo mínimo antes del milestone en el que si se ve que no se va a participar en el proyecto, se avise de antemano para así regular mejor las cargas de trabajo que se lleva cada participante del grupo que sí va a trabajar.

Una vez dicho esto, si tuviéramos más milestones para trabajar en el proyecto, tenemos una serie de mejoras que teníamos planteadas y que habría estado bien implementar, pero que no hemos podido realizar.

Estas mejoras son, por ejemplo, la creación de filtros para buscar censos por nombres de votaciones, por ids, por sexo de votantes…
Otra posible mejora hubiera sido la de modificar la página de creación para que en vez de dos celdas de inserción para los ids de la votación y votante, hubieran sido tablas en las que eliges los votantes que quieres que estén censadas para qué votación.
Además de esto, podríamos haber hecho que en la tabla de muestreo de datos pudiéramos elegir qué datos queremos ver.

Si hubiéramos tenido más tiempo y hubiéramos tenido los mismos equipos de compañeros para el proyecto nombrado “Giratina”, podríamos haber estado más en contacto con el resto de equipos, proponiendo mejoras en conjunto, como una página principal que llevara a cada uno de los módulos en los que nos estamos trabajando para hacer más fácil el acceso a las funcionalidades del proyecto “decide”.

Sin embargo, por la falta de tiempo, de personal y de conocimiento del entorno en el que hemos desarrollado, no hemos podido realizar estas tareas.