RESUMEN EJECUTIVO

En este documento describiremos el trabajo realizado por los miembros de Zapdos-Censo.

El objetivo de nuestras diferentes actividades era implementar nuevas funcionalidades en el sistema, que facilitaran su uso por parte tanto del administrador del sistema, como de los usuarios registrados.

Más concretamente, nos hemos centrado en automatizar operaciones del censo de Decide, así como exportación e importación de los datos del censo, adición automática de censos para usuarios con un determinado perfil, y la mejora de las interfaces gráficas del módulo.

1. Indicadores del proyecto

Miembro del equipo	Horas	Commits	Líneas de código	Test	Issues	Incremento
Esquivel Moreno, Eva	30	12	215	4	5	Creación automática de censos para usuarios con un perfil dado.
Cobos Ramos, Pedro Manuel	41	40	208	6	5	Importación de censos desde archivos tipo .csv y .txt
Rodríguez Gómez, Francisco José	32	3	222	6	4	Poder copiar (mover) los votantes de una votación a un censo nuevo.
Perea Ruiz, Sergio	20	8	98	2	4	Exportar censos como CSV y XLS
Coello Vasconi, Jesús	35	2	215	5	4	Nuevos campos y creación de censos a partir de IDs.
TOTAL	158					Descripción breve

La tabla contiene la información de cada miembro del proyecto y el total de la siguiente forma:

- Commits: solo contar los commits hechos por miembros del equipo, no lo commits previos
- LoC (líneas de código): solo contar las líneas producidas por el equipo y no las que ya existían o las que se producen al incluir código de terceros

- Test: solo contar los test realizados por el equipo nuevos
- Issues: solo contar las issues gestionadas dentro del proyecto y que hayan sido gestionadas por el equipo
- Incremento: principal incremento funcional del que se ha hecho cargo el miembro del proyecto

2. Integración con otros equipos

Los equipos con los que nos hemos integrado han sido:

- **Zapdos-Autenticación:** El propósito de la integración con este equipo ha sido compartir la adición de distintos campos asociados a los usuarios. Ambos grupos han trabajado en sus respectivos incrementos funcionales teniendo en cuenta este cambio en las mejoras.

  El propósito de censo era implementar sus nuevas funcionalidades haciendo uso de los nuevos campos almacenados:

  - Provincia: Indicador de la provincia de origen del votante. Estará almacenado como un tipo cadena.
  - Localidad: De la cual proviene el votante. Estará almacenada como un tipo cadena.
  - Fecha de nacimiento: Campo que será utilizado en alguna de las funcionalidades implementadas para calcular la edad del votante. Estará almacenada como un tipo fecha.
  - Género: Indica si el votante es hombre o mujer.

  El equipo de Censo-Zapdos ha utilizado los campos para implementar funciones como la adición de censos según un parámetro: consiste en para una votación determinada, permitir que todos los usuarios con un valor común puedan participar en dicha votación.

  Por su parte, el equipo Autenticación-Zapdos, los ha utilizado para el proceso de registro y acceso de usuarios al sistema. Todo votante, está registrado como usuario, y por tanto su información relativa a estos campos, va a estar almacenada en el sistema.

  La razón por la que el grupo Censo-Zapdos propuso la integración con el equipo Autenticación-Zapdos, es porque consideró imprescindible que ambos grupos deberían trabajar acorde con la misma información. El objetivo era proveer de facilidad de uso y concordancia al sistema. Autenticación-Zapdos consideró que era buena idea la integración, pues tuvo en cuenta que Censo-Zapdos haría uso posterior de la información que ellos decidieran guardar sobre los usuarios que serían autenticados en el sistema.

- **Zapdos-Votación:** La integración también ha consistido en hacer uso de datos comunes. En este caso, el grupo Zapdos-Censo propuso utilizar las votaciones

generadas por el equipo Zapdos-Votación. Cada votación tiene asociada preguntas con sus respectivas respuestas.

Zapdos-Censo pretendía crear en la vista de administrador un desplegable con las distintas preguntas almacenadas en el sistema. Dicha funcionalidad, también está asociada a la creación de censos por un determinado parámetro. El propósito es que, al crear un censo dado un determinado valor, en el momento en el que el administrador asocie la votación, no visualice los identificadores de las votaciones, sino las preguntas asociadas a ellas. De tal manera, será posible generar permisos de usuarios para votar de una forma más intuitiva, dado que directamente sabemos a qué preguntas queremos que responda el votante.

El motivo que más importante por el que Censo-Zapdos propuso la integración con Votación-Zapdos fue haber considerado la posibilidad de proveer de cierta facilidad de uso al usar las mejoras implementadas en la vista de administrador, en la parte de Censo. Por otro lado, Votación-Zapdos consideró interesante proveer datos que son encargados de generar, para la implementación de funcionalidades por parte de otros equipos.

3. Descripción del sistema

Este apartado se divide en sí, en varios subapartados, es por esto que es acompañado de una breve guía introductoria mediante la cual, el lector, podrá obtener mayor y mejor compresión del sistema software desarrollado.

Sea esta guía:
- **Descripción funcional** del sistema al completo, en la que se obtiene una perspectiva genérica del funcionamiento del proyecto a gran escala.
- **Descripción arquitectónica**, obtenemos una visión estructurada de los diferentes componentes del sistema, esto es, “paquetes” de código del proyecto Decide.
- **Relación con el resto de módulos del sistema,** siendo todos los módulos existentes:
    - Autenticación
    - Censo (nuestro módulo)
    - Votación
    - Cabina de votación
    - Almacenamiento de votos (cifrado)
    - Recuento (MixNet)
    - Post-procesado
    - Visualización de resultados

  Se explicará con cuáles de estos módulos tiene relación con el módulo de censo, en el cual se ha basado nuestro proyecto/sistema.
-**Cambios** implementados con éxito.

Descripción funcional:

Antes de comenzar a describir nuestro proyecto desarrollado, es necesario propiciar una breve idea general del proyecto Decide, en torno al que giran todos los subproyectos desarrollados por los distintos equipos de un grupo de clase.

Decide es un proyecto desarrollado en Django, se concibió con la idea de implementar una plataforma segura de voto electrónico, esta plataforma cedería una serie de garantías básicas, siendo estas, la anonimidad y el secreto del voto.
El proyecto Decide (a partir de ahora, cuando nos referimos a proyecto, hacemos referencia al proyecto origen principal, Decide, y nos referimos de manera equiparable, a sus subproyectos, que serán en este curso, tres subproyectos por grupo de clase, así mismo, cada subproyecto se compone de ocho módulos, ya citados, los cuales están desacoplados entre sí).

Cada módulo o subsistema se encarga de una tarea concreta en el sistema de votación.
Pueden existir tareas con cierta complicación, que requieran coordinar desarrollo y cooperación entre dos o más módulos diferentes, para que puedan comunicarse correctamente, (dentro de un mismo subproyecto, que serían: Zapdos, Articuno y Fortress).

Cada módulo será asignado a un equipo de desarrollo de entre 5 y 6 miembros, idealmente.

En nuestro caso se nos ha asignado gestionar el módulo de Censo, del subproyecto Zapdos, del proyecto Decide.

Un censo consiste en una lista oficial de los habitantes de una población o de un estado, en el cual existen indicaciones de diferentes condiciones sociales, tales como económicas, de género, edad, etc.

Tras tener claro, la definición exacta de un censo, concretamos la implementación de nuestro subsistema.

La parte de Censo es muy importante, en el sistema Decide, consiste pues en el grupo de personas que pueden votar en una votación, gestionar el censo es esencial para definir una votación, y poder controlar quienes son votantes o no de dicha votación.

La peculiaridad que tiene nuestro subsistema con respecto a otros módulos reside en que, prácticamente casi todos los módulos existentes pueden enviar peticiones sobre datos de un censo, por ejemplo, a nuestro equipo de desarrollo nos puede encargar el módulo X, los habitantes de Sevilla, varones que existen en Y votación.
Este tipo de peticiones, se gestionarán por medio de actas de colaboración, donde queda reflejado oficialmente cooperación entre dos o más módulos.
Estas peticiones se llevarán a cabo a razón de que la petición se pueda implementar con éxito por nuestro equipo de desarrollo, y la petición del otro módulo sea coherente con la actividad a la que nos concibe, sólo censos.
Tenemos pues un módulo que puede implicar gran cooperación con el resto de subsistemas.

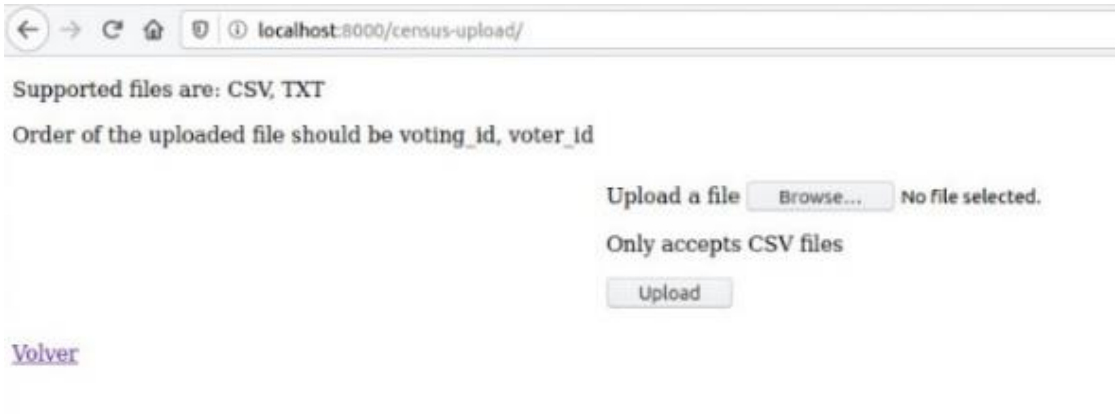
Nuestro sistema básicamente consiste en en crear censos, implementando diferentes funcionalidades.

Al desplegar nuestro sistema, en local por ejemplo podemos ver la página Admin de Decide, en Django, en la cual tenemos las funciones de crear censos, votaciones, usuarios votantes, etc.

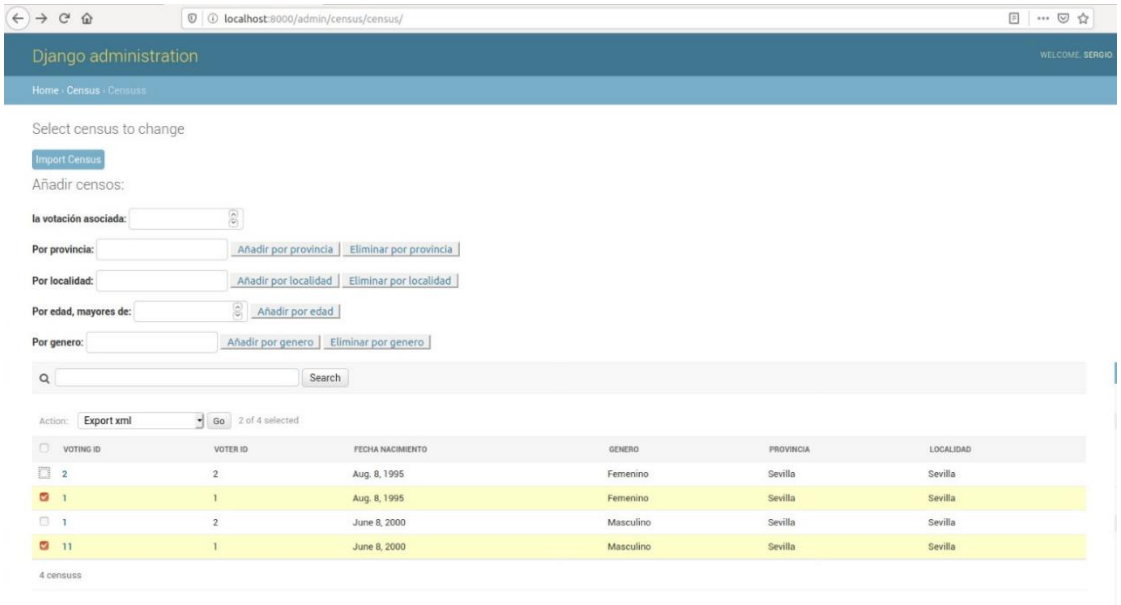
Cada miembro del equipo se ha encargado de implementar una funcionalidad relevante, que sea un añadido al proyecto base de censo.

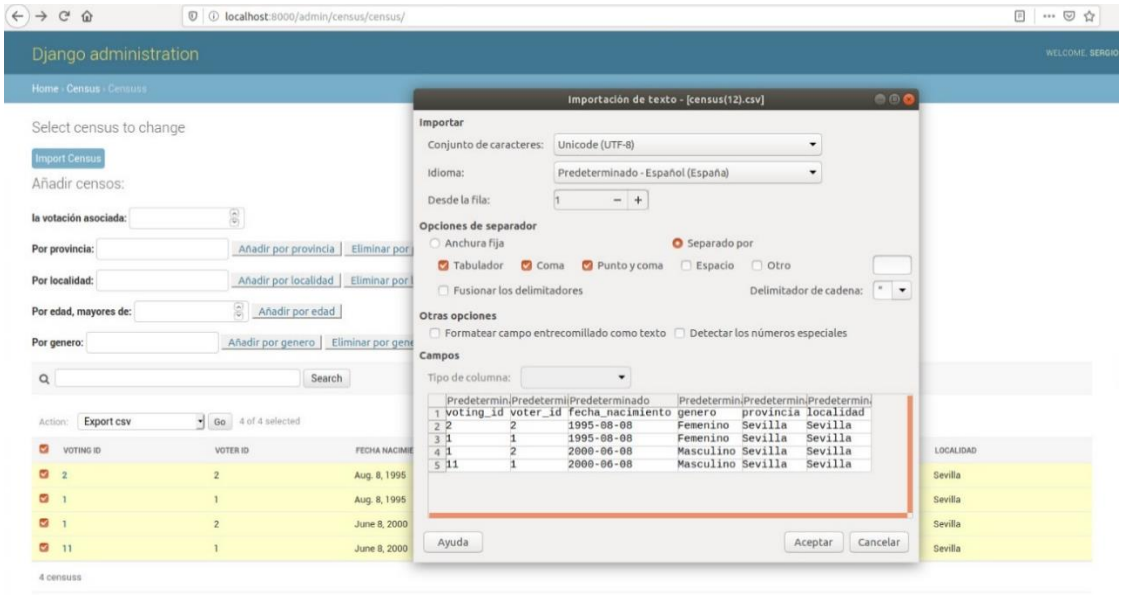
Nuestro sistema contiene entonces, varias nuevas funciones, de las que antes carecía:

- **Funcionalidad importar censo:** se ha implementado un botón en la vista de censo, contenida en la vista admin del sistema, dicho botón te redirige a una página dónde puedes subir un censo en formato .csv o .txt , en caso de no ser el formato el correcto, aparece un mensaje de error, en caso de si importar un archivo correcto, puede volver a la vista de censo, mediante otro botón.



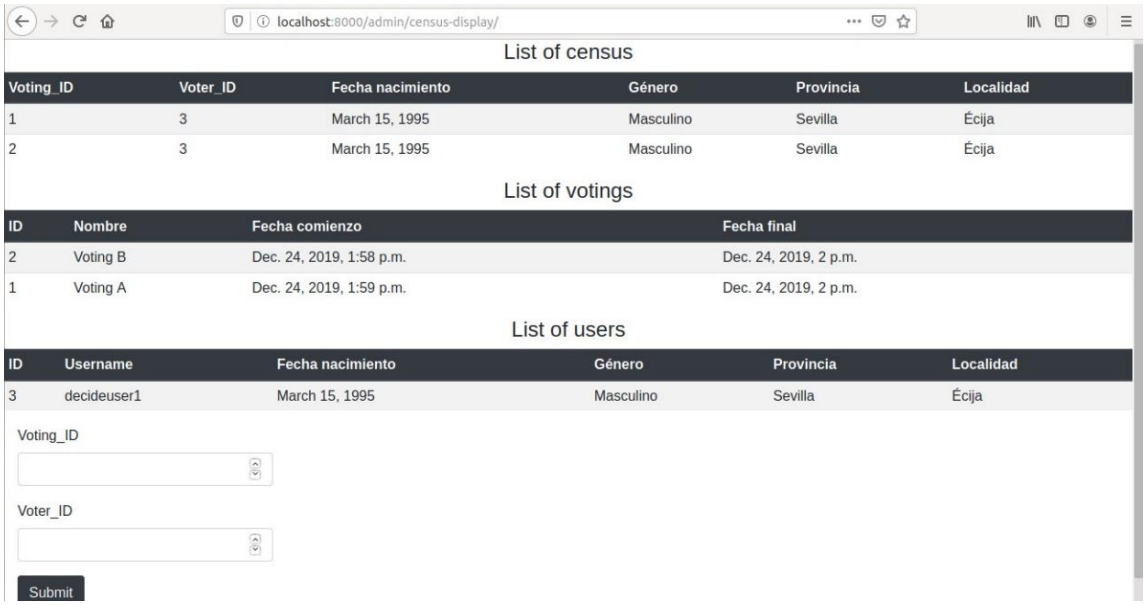
- **Funcionalidad exportar censo:** dentro de la vista de censos, en admin, aparace la opción ya existente de eliminar censos, seleccionándolos mediante checkbox, se ha implementado ahí, la exportación de uno o varios censos, en formato .csv o .xml.



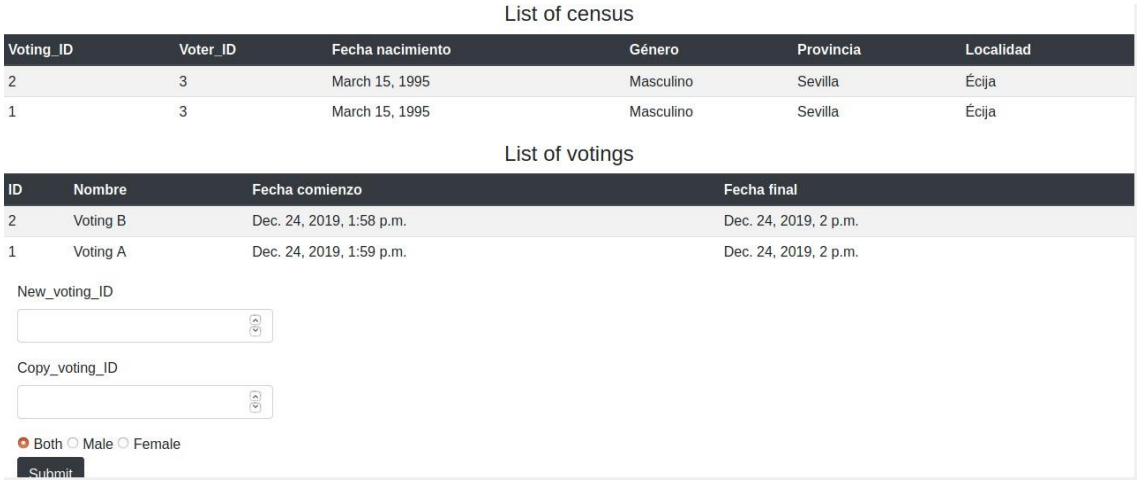


	A	B	C	D	E	F
1	Voting_id	Voter_id	Fecha_nacim	Genero	Provincia	Localidad
2	1	1	34919	Femenino	Sevilla	Sevilla
3	11	1	36685	Masculino	Sevilla	Sevilla
4						

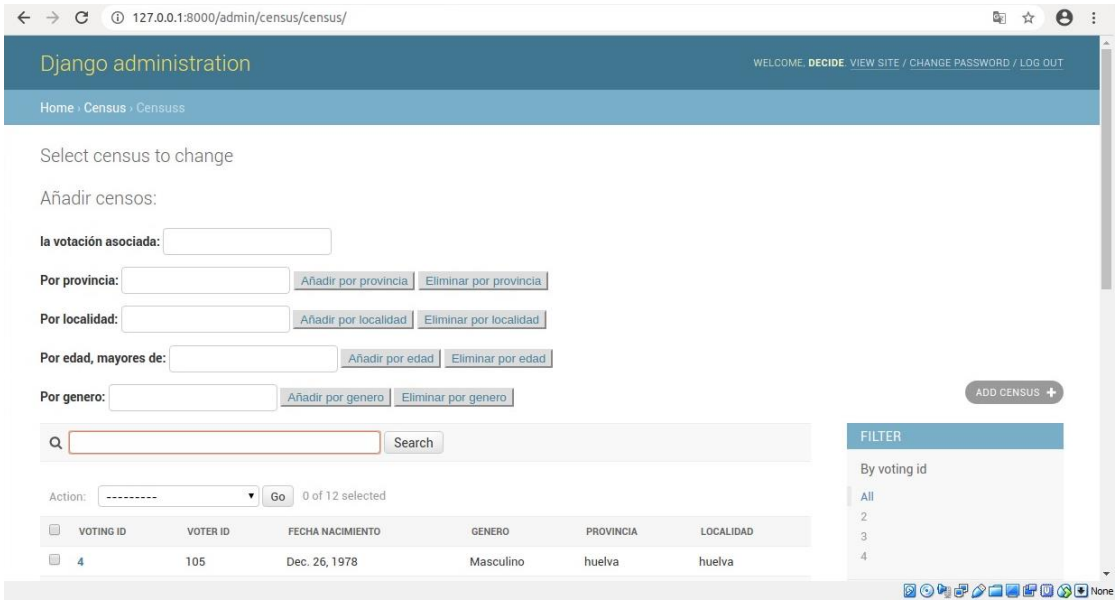
- **Funcionalidad mostrar censo:** la vista muestra tres tablas: los censos, las votaciones y los usuarios del sistema, y un formulario con dos campos: voting_id y voter_id. Al introducir valores correctos, es decir, ids de votaciones y usuarios que existan en la base de datos y que el censo no exista ya, el sistema busca el usuario con el id introducido en voter_id y crea un censo con las ids introducidas y los campos del usuario encontrado.



- **Funcionalidad copiar censo:** la vista muestra dos tablas: los censos y votaciones del sistema y un formulario con dos campos: new_voting_id y copy_voting_id. Al introducir datos correctos, es decir, id de votación a copiar que exista y no repetir censos ya existentes, el sistema busca la votación cuya id sea copy_voting_id y crea censos cuyo voting_id sea new_voting_id y el resto de campos los propios de la votación encontrada.



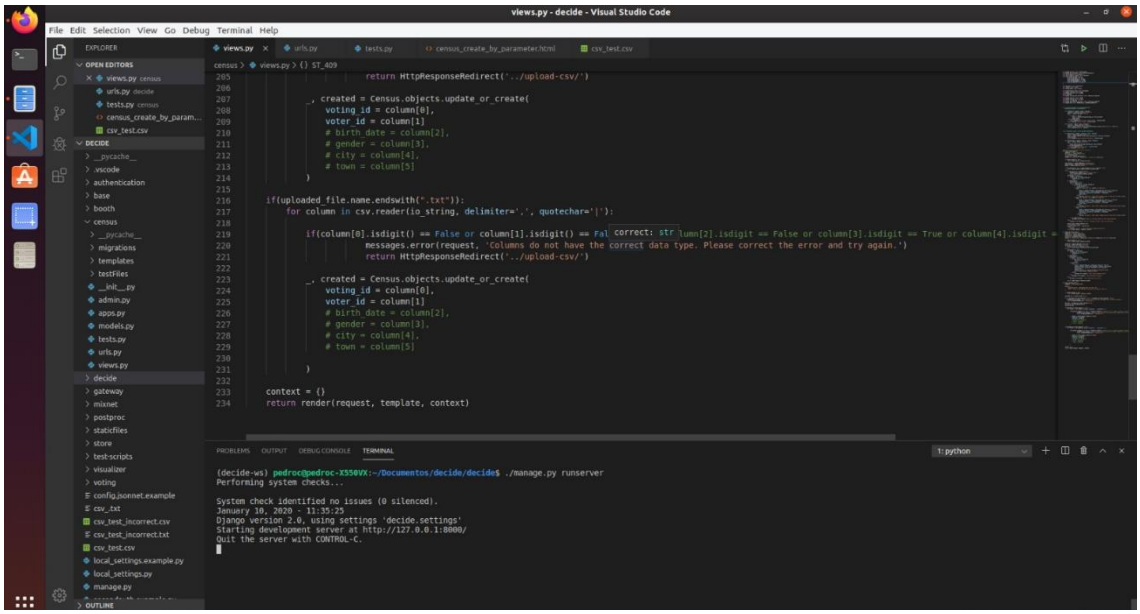
- **Funcionalidad creación de censos por categoría:** accediendo a admin.py, al apartado de Censos, podemos crear censos por provincia, edad y género.
Es necesario introducir un id de votación y uno de los campos citado anteriormente.
Un ejemplo podría ser: voting_id = 3, provincia = Sevilla.
Se crearía pues Censos para la votación 3, con todos los usuarios de Sevilla.



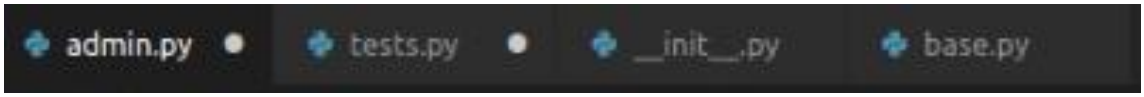
Descripción estructural:

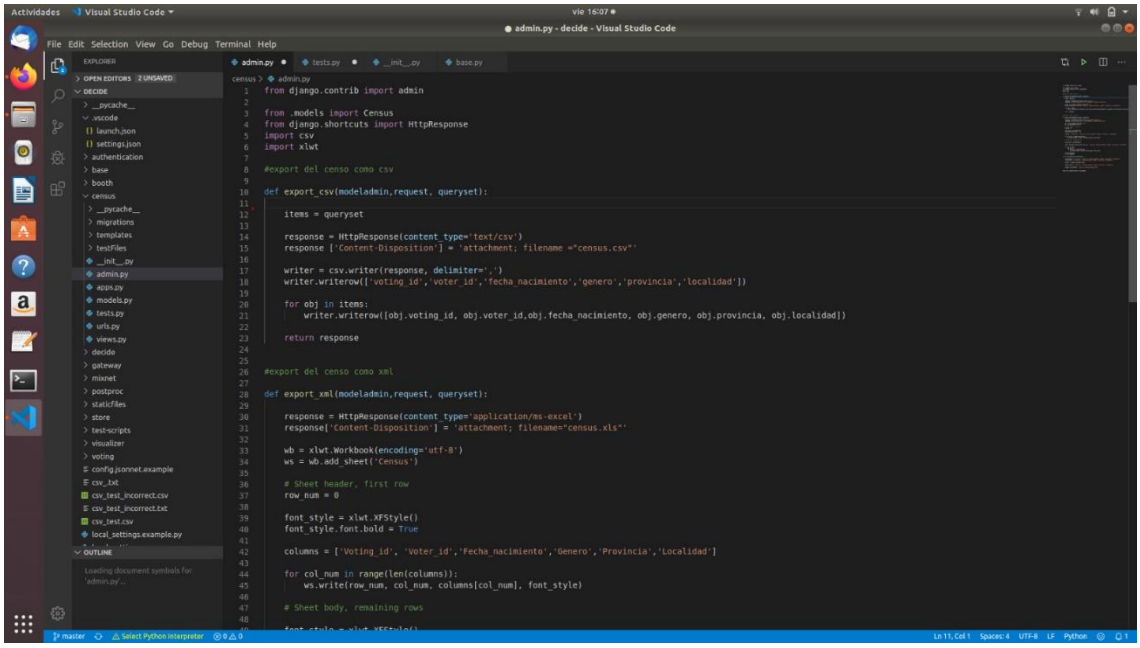
Entendemos una descripción de los componentes, paquetes y archivos correspondientes a todo lo modificado por los miembros del equipo de desarrollo, a raíz del proyecto base Decide.

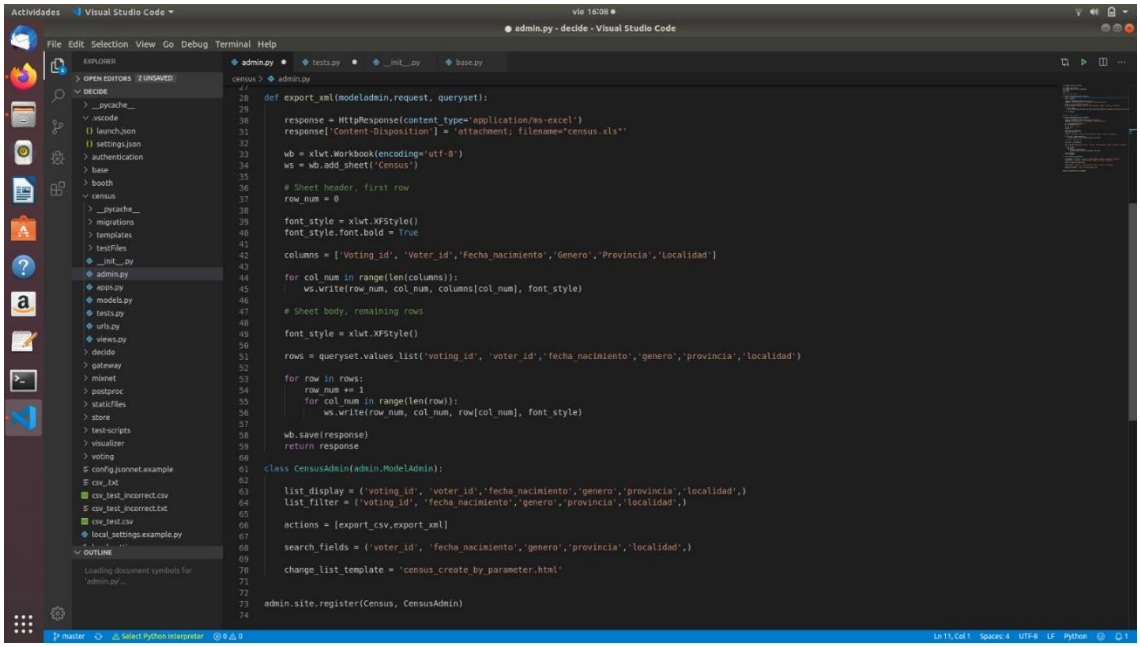
Respecto a la funcionalidad **importar censo**, se han creado/ modificado los siguientes archivos:



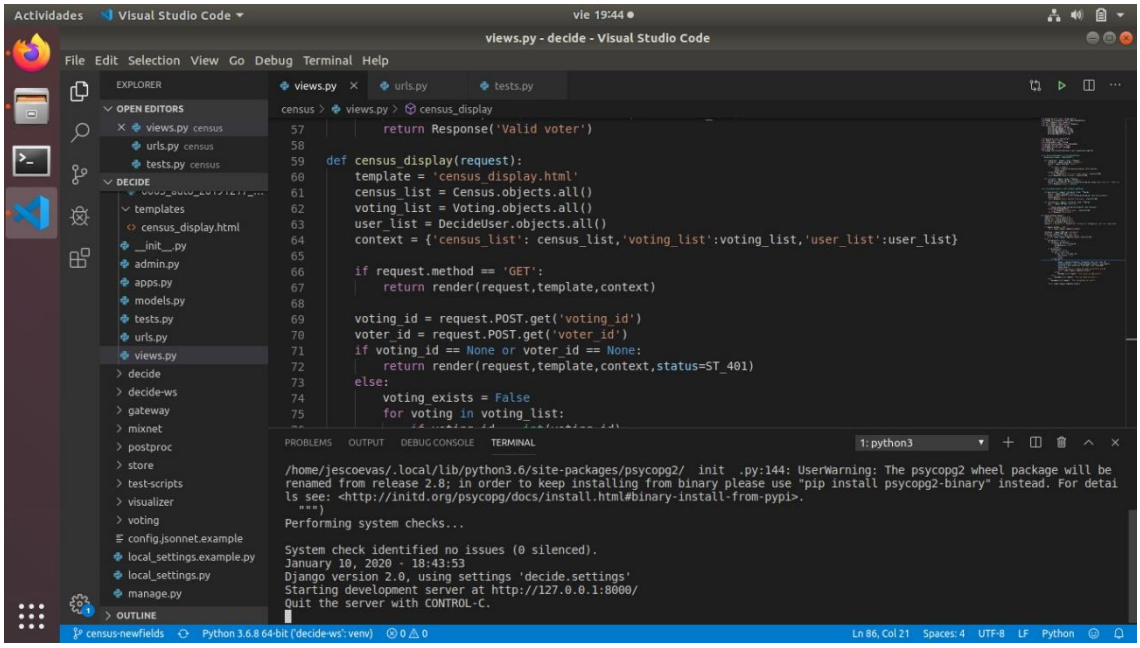
Respecto a la funcionalidad **exportar censo**, se han creado/ modificado los siguientes archivos:



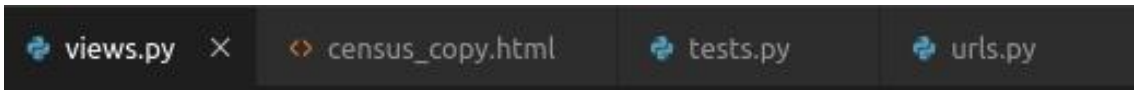


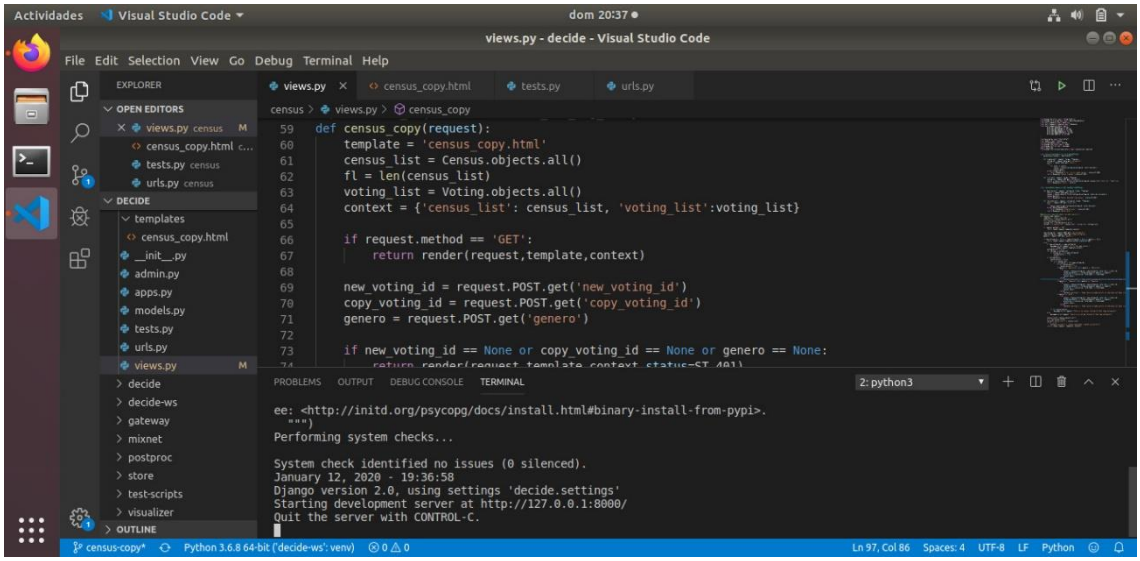


Respecto a la funcionalidad **mostrar censos**, se han creado/ modificado los siguientes archivos:



Respecto a la funcionalidad **copiar censos** se han creado/ modificado los siguientes archivos:



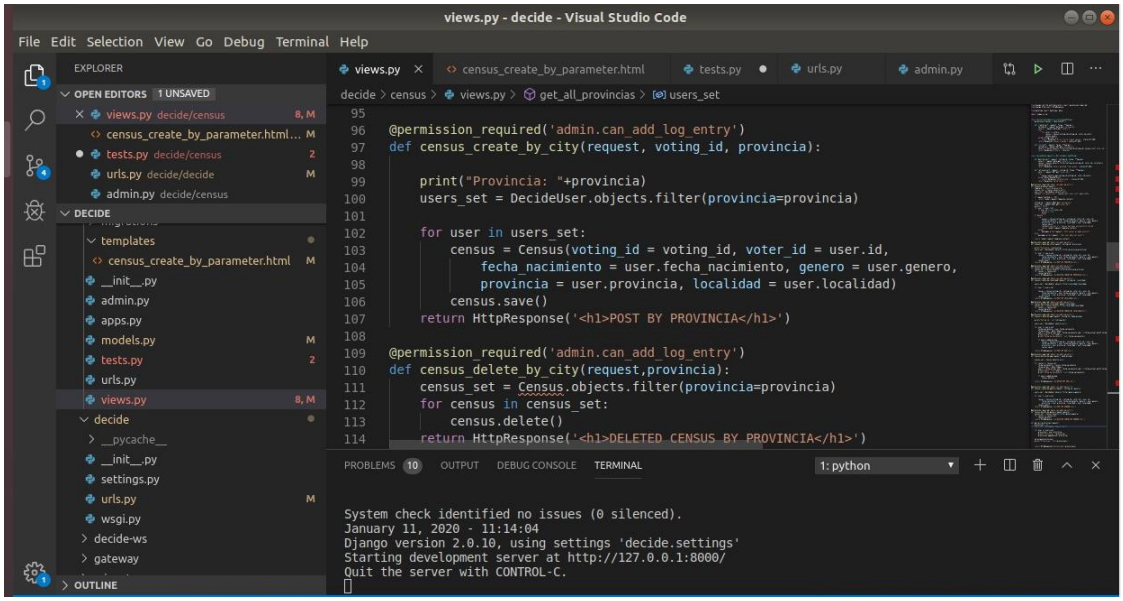


```python
def census_copy(request):
    template = 'census_copy.html'
    census_list = Census.objects.all()
    fl = len(census_list)
    voting_list = Voting.objects.all()
    context = {'census_list': census_list, 'voting_list':voting_list}

    if request.method == 'GET':
        return render(request,template,context)

    new_voting_id = request.POST.get('new_voting_id')
    copy_voting_id = request.POST.get('copy_voting_id')
    genero = request.POST.get('genero')

    if new_voting_id == None or copy_voting_id == None or genero == None:
        return render(request,template,context,status=ST_401)
    else:
        if new_voting_id == copy_voting_id:
            messages.error(request,'It is the same census')
            return render(request,template,context)
        voting_exists = False
        for voting in voting_list:
            if voting.id == int(new_voting_id):
                voting_exists = True
                break
```

```python
if voting_exists:
    census_exists = False
    for c in census_list:
        if c.voting_id == int(copy_voting_id):
            if not census_exists:
                census_exists = True
            if genero == 'masculino' and c.genero == 'Masculino':
                try:
                    census = Census(voting_id = new_voting_id, voter_id = c.voter_id,
                    fecha_nacimiento = c.fecha_nacimiento, genero = c.genero,
                    provincia = c.provincia, localidad = c.localidad)
                    census.save()
                except:
                    context['warning'] = 'Some census already exists so they have not been created'
            if genero == 'femenino' and c.genero == 'Femenino':
                try:
                    census = Census(voting_id = new_voting_id, voter_id = c.voter_id,
                    fecha_nacimiento = c.fecha_nacimiento, genero = c.genero,
                    provincia = c.provincia, localidad = c.localidad)
                    census.save()
                except:
                    context['warning'] = 'Some census already exists so they have not been created'
```

```python
            if genero == 'both':
                try:
                    census = Census(voting_id = new_voting_id, voter_id = c.voter_id,
                    fecha_nacimiento = c.fecha_nacimiento, genero = c.genero,
                    provincia = c.provincia, localidad = c.localidad)
                    census.save()
                except:
                    context['warning'] = 'Some census already exists so they have not been created'

    if not census_exists:
        messages.error(request,"There is no census refered to that copy_voting_id")
else:
    messages.error(request,'There is no voting refered to that new_voting_id')

census_list2 = Census.objects.all()
sl = len(census_list2)
context['census_list'] = census_list2
if sl != fl:
    context['success'] = 'Census have been created successfully'
return render(request, template, context)
```

Respecto a la funcionalidad **crear censos por categoría** se han creado/ modificado los siguientes archivos:



Módulos con los que Censo tiene relación:

Como hemos explicado anteriormente, nuestro proyecto particular y módulo en el cual se basa, podría tener relación con el resto de módulos de Decide-Zapdos, en nuestro caso particular. Ya que, como ya se dijo, cualquier módulo podría realizarnos una petición, respecto a un censo. Aunque esto difiere de la realidad, ya que durante el transcurso del proyecto, desde sus inicios, censo ha intentado tener algún tipo de cooperación con varios módulos de Zapdos , aún así, ha sido posible integrarnos con estos dos, Autenticación y Votación, con los cuáles existe oficialmente cada acta de integración, respectivamente. En resumen, tenemos relación con todos los módulos pero nos hemos podido integrar únicamente con dos de ellos.
Además, los módulos con los que nos integramos mediante alguna petición, son Auntenticación y Votación, podríamos decir que, Censo se sitúa en medio, de estos dos módulos, en términos de dependencias.
Censo necesitaría de Autenticación, y Votacion de Censo.
Autenticación <- Censo <-Votación

Procedemos a explicar brevemente los módulos, Autenticación y Votación, además adjuntaremos una descripción sobre que tarea se ha de realizar conjuntamente, entre dichos módulos y censo, acordados en cada acta de integración.

Autenticación

Consiste en autenticar a los votantes, comprobando que un votante no vota más de una vez en la misma votación y de que puede votar en dicha votación. Esto se consigue autenticando a los usuarios votantes con fines de evitar algún tipo de fraude, tales como la suplantación de identidad. Esto se puede conseguir, usando enlaces de voto electrónico de acceso por un único medio, certificados, etc.

Explicamos resumidamente la integración existente con autenticación:

-Ambos grupos añadirán a los usuarios que se registren en el sistema nuevos cambios:

- Provincia del usuario y de tipo texto.
- Localidad del usuario y de tipo texto.
- Fecha de nacimiento del usuario y de tipo fecha.
- Género del usuario, de tipo texto, y que tomará los valores 'Masculino' y 'Femenino'

Estos cambios serán usados como convenga en cada módulo del sistema, se añadirán a las mejoras existentes individuales de cada equipo.

Votación

Consiste en definir una o varias preguntas, con opciones distintas y diferentes configuraciones, siendo votación de tipo voto simple, una lista ordenada de opciones, preguntas con elección multiple, etc.

Explicamos resumidamente la integración existente con votación:

- Votación nos cede una serie de preguntas asociadas a las encuestas,
  Nuestro equipo intentará elaborar un desplegable en el archivo admin.py,
  En vez de mostrar identificadores de las votaciones, se muestren las
  Preguntas registradas en el sistema.

Cambios realizados:

A parte de las cinco funciones implementadas, se han realizado pruebas CRUD, unitaria, …, para cada una de las nuevas funcionalidades del sistema.
En resumen, se han realizado estos cambios al proyecto, todos conseguidos con éxito.
•Mover personas entre censos (copiar censos).
•Creación de censos por categoría.
•Añadir campos al censo (mostrar censos).
•Exportar e importar censos en diversos formatos.
• Front-end de cada funcionalidad (realizada por cada miembro, según las necesidades de la tarea implementada).
• Ejecución de las pruebas en Travis, obteniendo "luz verde".
• Se ha conseguido desplegar el sistema en Heroku, funcionando todas las nuevas funcionalidades correctamente.

se explicará el sistema desarrollado desde un punto de vista funcional y arquitectónico. Se hará una descripción tanto funcional como técnica de sus componentes y su relación con el resto de subsistemas. Habrá una sección que enumere explícitamente cuáles son los cambios que se han desarrollado para el proyecto.

4. Visión global del proceso de desarrollo

Zapdos-Censo tenía en mente la idea de implementar nuevas funcionalidades para Decide asociadas a dicho módulo. El equipo realizó varias reuniones, tanto para generar nuevas ideas acerca de las posibles mejoras aplicables al sistema, como para asignar tareas a los miembros del grupo y revisar el trabajo desarrollado hasta el momento de la reunión.

Se acordó con el grupo utilizar como entorno Visual Code, el proyecto base Decide para el cuál usamos Python3, y la versión de Django compatible asociada al proyecto.

En estas reuniones del grupo, cada una de las posibles mejoras que finalmente se propusieron:

- **Creación automática de censos dado un parámetro:** Teniendo en cuenta que los usuarios se almacenaban con distintos campos, teníamos cierto juego para implementar esta funcionalidad.

  El objetivo era crear censos automáticamente según un parámetro. Un ejemplo sería: El administrador del sistema desea dar acceso a todos los votantes mayores de 18 años a una determinada votación. En la vista de administrador, encontraría una entrada asociada a la edad en la que introducirá el número 18. Al enviar los datos, estamos consiguiendo que todos los usuarios registrados en el sistema, mayores de 18 tengan acceso a la votación en cuanto esta sea abierta.

  El proceso comenzó generando un conjunto de usuarios de prueba, los cuales serían utilizados en la creación de censos. Una vez hecho esto, dado que la idea era asociar usuarios a una votación, se crearon los distintos métodos en views.py como parte principal de esta funcionalidad.

  Haciendo uso del modelo Census, creado anteriormente, en el archivo models.py se generaron métodos de creación de censos en views.py:

  - Uno de ellos es census_create_by_city(voting_id, provincia), que dado un id de votación y una determinada provincia, permite añadir a todos los votantes de esa provincia a los autorizados para votar en esa votación.
  - Otro de ellos que puede resultar interesante es census_create_by_age (voting_id, edad_minima), que dado un id de votación, y una edad a partir de la cual es posible participar en ella, autoriza a todos los mayores a esa edad registrados en el sistema a votar en dicha votación. Para ello, fue necesario calcular la edad de cada uno de los votantes, dado que la información recogida era la fecha de nacimiento de los usuarios.

- También están creados los métodos de eliminación de censos dado un parámetro.

Además, para ello fue necesario crear un fichero .html que sería incluido en el admin.py del proyecto. Este contenía entradas de texto para los distintos campos por los cuales es posible crear censos. Cada uno de los botones que envían la url de creación de censo, está asociado a una función formulada en JavaScript.

La razón por la que no está hecho con un formulario es porque es más fácil darle el formato que queremos a la url si la escribimos en el href del botón.

Las urls que dan paso a los métodos de creación de censos, están definidas en decide/urls.py. A cada una de ellas se les pasa los parámetros necesarios para ejecutar el método.

El resultado de ejecutar los distintos métodos es la correcta adición o eliminación de censos según un parámetro dado.

- **Importación de censos:** Consiste en, dado un determinado fichero que contenga los datos de un determinado censo, es posible importarlo subiendo dicho archivo a la plataforma. Por ejemplo, ficheros con formato .csv pueden ser subidos con total facilidad. Es necesario explorar nuestro navegador de archivos y seleccionar el documento .csv deseado. De esta manera, importando el fichero, podemos leerlo satisfactoriamente, y crear los censos deseados, especificados en el archivo dado.

- **Exportación de censos:** El objetivo de la exportación de censos, es tener la posibilidad de generar un archivo descargable con los datos de un determinado censo.

  Para la implementación de está funcionalidad fue necesario aprender a leer ficheros y generar ficheros. Por este motivo, fue necesario buscar algunos tutoriales:

  - Tutorial para exportar arhivos .csv en Django
  - Crear acciones en Django

  El resultado de esta mejora de Decide, fue la posibilidad de exportar censos en distintos formatos. También fue necesario realizar algunas búsquedas en Stack Overflow.

- **Front-End de Administración de censo:** Un incremento bastante útil del sistema es la creación de unas interfaces gráficas agradables al usuario. Para ello, ha sido necesario el uso de Bootstrap, que provee librerías de HTML, JavaScript y CSS de código abierto. De esta forma, se simplifica un poco la dificultad de implementación del Front-End de la plataforma.

  El resultado ha sido la obtención de una interfaz gráfica intuitiva y agradable al usuario.

- **Reutilización de censo entre diferentes votaciones:** Para ejecutar esta funcionalidad se esperaba poder copiar censos. Es decir, crear un nuevo censo para la misma votación que el que estamos teniendo en cuenta, y con los mismos votantes.

Para cada una de estas nuevas funcionalidades se han creados tests que comprueban que se ejecutan correctamente. Estas están recogidas en el archivo census/tests.py. Herramientas como TravisCI permiten la ejecución de estas pruebas automáticamente, de forma que podamos visualizar su resultado de forma global cada vez que añadamos nuevos cambios al proyecto. También se ha tenido en cuenta Codacy para comprobar la calidad de nuestro código generado.

El resultado de todas estas mejoras ha sido la construcción de un nuevo Decide, con funcionalidades asociadas al censo que proveen facilidad de uso y mayor comodidad en cuanto al tratamiento de la información.

5. Entorno de desarrollo

Nuestro equipo ha hecho uso de Visual Studio Code (VSCode) sobre Ubuntu 18.04 ya que todas las utilidades de las que requeríamos para el desarrollo del proyecto se encuentran con mayor facilidad y disponibilidad bajo este sistema operativo.

Por lo general el equipo ha hecho uso de máquinas virtuales sobre Windows 10 a excepción de uno de los integrantes que optó por reservar una partición del disco para instalar Ubuntu como sistema operativo conjunto a Windows 10, esto fue así ya que el uso de máquinas virtuales en su ordenador personal le estaban dando problemas por lo que tuvo que optar por instalar ambos sistemas operativos en su máquina.

Para la instalación del sistema de votación se ha tomado como referencia base el código que nos ofrece Wadobo, el cual se basa en el framework implementado el Python 3.7.5 conocido como Django, usaremos la versión 2.0 del mismo para el desarrollo de la aplicación.

Todos los miembros del grupo han realizado la instalación en un entorno virtual de Python, bajo el cual se ha procedido a la instalación del framework. Se decidió el uso de esta opción gracias a la facilidad de despliegue en un entorno local que ofrece, ya que en caso de ser necesario tan solo haría falta la creación de un entorno virtual e instalar dentro las dependencias indicadas en el archivo 'requirements.txt'.

Para la instalación del framework se ha ejecutado el archivo 'requirements.txt' el cual contiene todas las dependencias, así como sus versiones concretas que se van a usar. Para el incremento que se ha realizado en nuestro grupo tan solo se han añadido dos dependencias extra, las cuales son necesarias para el despliegue de la aplicación en la plataforma Heroku. Estas dependencias son 'Django-heroku' y 'gunicorn', ambas en su versión más reciente.

La base de datos de la cual haremos uso será PostgreSQL versión 11.5, la cual hace falta instalarla por separado del sistema anterior y previo a la instalación del framework ya que sin la base de datos la instalación de las dependencias devolverá un error. Tras instalar la base de datos tendremos que crear un usuario en la base de datos el cual tendremos que llamar

“decide”, al cual le asignaremos una base de datos que también deberemos de crear a la cual volveremos a llamar de la misma forma, es decir tanto la base de datos como el usuario asociado a ella tendrán el mismo nombre, “decide”.

Una vez todas las dependencias hayan sido instaladas satisfactoriamente, será necesario realizar todas las migraciones de la base de datos, lo que incluye todos los módulos en los que se subdivide la aplicación final.

Ya que toda la aplicación se ha realizado en la parte de administrador de Django, será necesario crear un usuario administrador con el cual podremos acceder a todas las funcionalidades que se implementen en el sistema. Sin este usuario no se podrá hacer uso de la aplicación por lo que una vez el sistema funcione desplegado en local, tendremos que proceder a crear un usuario administrador, también conocido como “super usuario”

Cuando todo lo anterior haya sido instalado, para comprobar si la aplicación base funciona correctamente antes de proceder con el desarrollo del incremento debemos ejecutarla mediante el comando “. /manage.py runserver”, el cual lanza la aplicación si todo ha ido bien. Se debe comprobar que el archivo “local_settings.py” contenga como URL de todos los módulos, así como la URL base sean ‘localhost’ en el puesto ‘8080’ que será donde realizaremos las pruebas antes de finalmente proceder al despliegue.

Como software de control de versiones se ha hecho uso de Git, el cual se ha de instalar por separado del resto de componentes. Será con esta herramienta con la que subiremos el código a GitHub y con la nos moveremos por las diferentes versiones del código que cada uno irá desarrollando independientemente del resto de compañeros.

6. Gestión de incidencias

En cuanto a la gestión de incidencias se a seguir la siguiente metodología. Para tratar las incidencias internas vamos a usar una herramienta que nos proporciona el software de control de versiones del que se ha hecho uso, GitHub. En concreto, vamos a las ‘issues’ o incidencias en Español, que podemos crear y asociar a un tablero, como para este proyecto solo tenemos un tablero serán en él dónde especificaremos todas las ‘issues’ que surjan o sean necesarias a lo largo del proyecto.

Lo primero que vamos a hacer será definir una plantilla para que todas las incidencias posteriores tengan la misma estructura. Se ha de diferenciar entre incidencias y errores o bugs.

Si se trata de una petición, tendremos que precisar un título conciso que resuma lo mejor posible. Además, se deberá detallar en la descripción todo lo relacionado con la petición, así como una posible propuesta de implementación. Una vez definido, el equipo decidirá a quien se le asigna dicha petición, junto con una etiqueta que indique la prioridad de la incidencia a tratar. La prioridad de la incidencia podrá ser baja, media o alta.

En caso de tratarse de un error o bug, se deberá crear un título para la ‘issue’ igual que hacíamos con el caso de una petición, este deberá ser breve y lo mas resumido posible para

que de un vistazo se pueda identificar el error que se trata. Además, en la descripción se tendrá que explicar el error de la forma mas detallada posible, debiendo indicar los parámetros de configuración utilizados para poder replicar el error, así como una propuesta de solución. Como ocurre con las peticiones, el bug deberá ser asignado a un miembro del equipo, normalmente se asignará al miembro que encontró o generó el error, aunque es posible que dicho miembro requiera de ayuda de otro miembro para la resolución de este, en cuyo caso se añadirá también a la petición de solución del error.

De forma predeterminada tanto las incidencias como los errores se encuentran en estado de abierto lo cual nos indica que el problema en cuestión no ha sido resuelto. Una vez solventado, dentro de la incidencia se tendrá que añadir la solución encontrada y cerrarla una vez comprobada que funciona correctamente. Cerrar la incidencia también servirá para indicar al resto del equipo que se ha completado dicha parte de la implementación y, en caso de solventar un error, en caso de volver a surgir dicho problema, este podrá tener una solución directa.

Enlace de incidencias internas: https://github.com/pedcobram/decide/issues

Para las incidencias externas se va a seguir una estrategia muy similar a la estrategia seguida para tratar las incidencias internas ya que se va a hacer uso de una plantilla parecida en la que además del resto de datos se añade el grupo al que se refiere la incidencia en cuestión.

Los pasos que seguir para crear una incidencia son los siguientes: Primero abrir el repositorio en el cual queramos crearla, dependiendo si es interna o externa. Una vez en el repositorio, pulsamos sobre la pestaña de issues y le damos al botón ‘New issue’. Después de esto, podremos ver que tenemos añadida una plantilla y solo tendremos que pulsar en ‘Get Started’ para proceder a rellenar dicha plantilla como indicamos anteriormente. Una vez se hayan completado todos los apartados sólo tendremos que pulsar en ‘Submit’ y ya estaría publicada nuestra incidencia. Esto deberemos hacerlo cuando tengamos un problema que no podamos resolver, cuando nos encontremos con un bug de algo que debería funcionar, pero en dicho caso está dando algún problema, cuando necesitemos alguna información o código o ante cualquier otra petición que requiera de colaboración.

Una incidencia concreta de nuestro grupo fue por ejemplo cuando tuvimos un problema en el proceso de integración continua, que era el causante de no poder desplegar la aplicación en heroku. Para ellos se inició un proceso de incidencia, el cual comenzó con la creación de esta. Primero se rellenó la plantilla antes descrita, especificando que se trataba de un bug y añadiendo una descripción para saber cuándo se producía dicho bug. Además, se adjuntó la traza de Travis en la que se podía observar el fallo y se añadió una posible solución. Esta incidencia la tuvo asignada un solo miembro del grupo, que fue el encargado de ejecutar el despliegue, aunque se animó al resto del equipo a proporcionar posibles soluciones que fueron probadas previo a encontrar la solución el problema. Después de ser publicada, le llegó una notificación a todos los miembro para ser informados y se procedió a intentar resolverla. Una vez se supo cuál había sido el problema, se procedió a añadir la resolución en la incidencia la cual fue eliminar el fichero local_settings.py de la rama máster, debido a que dicho fichero sobrescribe lo que aparezca en settings.py. Por último, se publicó la resolución y se cerró la incidencia.

Link de la incidencia del ejemplo: https://github.com/pedcobram/decide/issues/20

7. Gestión del código fuente

Usage model

Mediante una reunión del grupo de trabajo Zapdos en horario de teoría se decidió crear un proyecto principal y distintos subproyectos cada uno correspondiente a un módulo del sistema Decide. La herramienta usada para la gestión del código ha sido Github ya que es la más óptima para gestionar el código. Los subproyectos se crearán mediante forks del proyecto principal.

Proyecto principal

El proyecto principal es un fork del repositorio de la ETSII, tal y como ha sido desarrollado, pero con pequeñas modificaciones para adaptarlo a la asignatura. En la reunión se eligió un dueño para gestionar este proyecto principal. Sus funciones son principalmente atender las pull requests realizadas desde los forks (subproyectos) y comprobar que el resultado final de todo el desarrollo funcione correctamente, esto es, que pase todas las pruebas y tests de calidad y que el sistema en sí funcione según lo esperado. En este proyecto no se debe codificar a no ser que sea estrictamente necesario, ya que el desarrollo se debe realizar en los subproyectos y posteriormente realizar pull requests. El enlace del proyecto principal es https://github.com/LorenRd/decide.

Subproyecto

Cada grupo ha realizado en Github un fork del proyecto principal y ha trabajado sobre él. En cuanto al desarrollo de estos subproyectos se ha trabajado mediante ramas. El dueño de repositorio de cada fork ha creado las ramas necesarias según las funcionalidades pensadas y cada desarrollador se la ha clonado en local para trabajar en ella. En nuestro subproyecto, Censo, el dueño del repositorio es Pedro Manuel Cobos Ramos. Después se han realizado los commits y pushes necesarios y, después de resolver los conflictos encontrados y demás incidencias, se han realizado pull requests las cuales debe gestionar el dueño del proyecto principal. Este es el enlace del subproyecto: https://github.com/pedcobram/decide y estas las ramas creadas durante el desarrollo:

- Funcionalidad *Copiar censos*: https://github.com/pedcobram/decide/tree/census-copy
- Funcionalidad *Mostrar censos*: https://github.com/pedcobram/decide/tree/census-newfields
- Funcionalidad *Importar archivos csv*: https://github.com/pedcobram/decide/tree/census-csvupload
- Funcionalidad *Crear censos por campos*: https://github.com/pedcobram/decide/tree/census-filtering
- Funcionalidad *Exportar archivos csv*: https://github.com/pedcobram/decide/tree/census-csvexport

Commits

Se ha diseñado una plantilla de commits a usar para llevar un mejor orden y control de los mismos. Después de varios diseños optamos por la más sencilla y clara:

Título: < Asunto : Explicación a alto nivel de los cambios realizados >

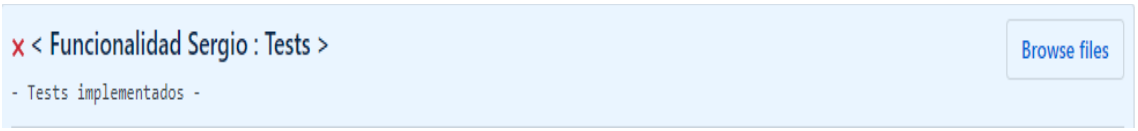
Cuerpo: - Explicación específica de los cambios realizados –

Esta plantilla no se muestra directamente a la hora de realizar el commit para que el desarrollador edite las partes, sino que hay que escribirla completa, ya que no supone un esfuerzo significativo.

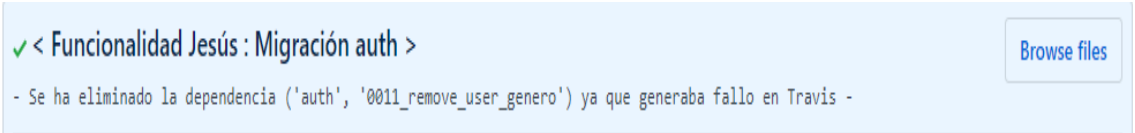
No se ha fijado una dinámica fija que determine cuándo realizar los commits, pero se han procurado hacer cuando se han realizado cambios importantes en el sistema como funcionalidades ya productivas o ciertos cambios de importancia en ellas como validaciones o corrección de errores, aunque también se han hecho cuando se han realizado cambios correspondientes a las pruebas (Travis, Codacy, etc.) o despliegue (Heroku). Se ha usado la plantilla anterior con los commits en local, pero no se ha seguido a la hora de mergear las ramas. Es posible que haya commits locales que no sigan la plantilla ya que fueron realizados en etapas tempranas del proyecto y todavía no se había diseñado.

A continuación se muestran varios ejemplos de commits:

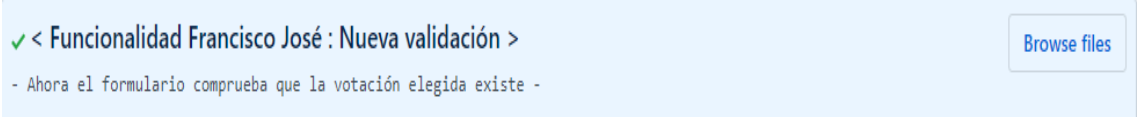
En este primer caso se hizo el commit al realizar los tests de la funcionalidad *Exportar csv*. El enlace es el siguiente: https://github.com/pedcobram/decide/commit/90989d6b82dbe9981781f351ce7ee508948bfcb8



En este segundo caso hubo que eliminar ciertas líneas de código en las migraciones de Authentication para que Travis pudiera correr las pruebas correctamente. El enlace es el siguiente: https://github.com/pedcobram/decide/commit/1baded2df932f95276991ebdf74f0c4292a34e31



En este tercer ejemplo hubo que implementar una nueva validación en el formulario de la funcionalidad *Copiar censos*. El enlace es el siguiente: https://github.com/pedcobram/decide/commit/2c25ca7b33966b1d8cd78c3207e8021d917ec079



8. Gestión de la construcción e integración continua

Para la integración continua hemos decidido usar Travis CI y Codacy.

Codacy sirve para detectar posibles mejoras en el código y está integrado en Travis a través de un Token de codacy asociado al proyecto (CODACY_PROJECT_TOKEN) . Dicho Tocken se inserta en el archivo de configuracion de Travis.

Travis sirve para probar las pruebas del proyecto ,y los resultados de dichas pruebas se muestran de la siguiente forma: si alguna prueba no pasa el código se considerará 'broken' y habrá que solucionar los errores en la funcionalidad o prueba correspondiente. En el caso de que todas las pruebas pasen los tests, el código se considerará 'passed' y se desplegará en el servidor, en nuestro caso se desplegará en Heroku.

Para integrarse con Travis hay que crearse una cuenta en travis_ci.org para enlazar dicha cuenta con el repositorio de GitHub. En nuestro caso la cuenta se la ha creado Pedro Cobos, ya que el repositorio de GitHub de Decide-Zapdos-Censo fue creado por el, y el repositorio de GitHub es https://github.com/pedcobram/decide .

En cuanto a la manera de integrar el proyecto en Travis, hemos decidido que vamos a integrar todas las ramas de Decide-Zapdos-Censo para pasar las pruebas de cada una de las ramas y, una vez que las ramas estén en 'passed' , hacer un merge a la la rama master de Decide-Zapdos-Censo , la que debe pasar las pruebas en Travis. Como hemos comentado anteriormente el proyecto se desplegará en Heroku una vez Travis devuelva las pruebas como 'passed'.

Como indicadores de calidad hemos decidido que la build realizada por Travis debe obtener un resultado de 'passed' y en codacy se debe obtener un resultado mínimo del 50% de aprobación.

9. Gestión de liberaciones, despliegue y entregas

En el equipo de Decide-Zapdos-Censo hemos decidido usar ramas hijas de la rama Master. En cada una de las ramas se realiza una funcionalidad específica y sus pruebas(cada funcionalidad será ejecutada por un único desarrollador). Una vez finalizadas las tareas y probar que funcionan correctamente las pruebas y pasados los tests en Travis, se realiza un merge a la rama Master del proyecto, donde tras ser mergeadas todas las ramas, se volverán a pasar los test y probados en Travis.

En la rama master se deberán encontrar las funcionalidades completas antes de la finalización de los milestones pertinentes. La fecha límite para tener todas las

funcionalidades en la rama master ,funcionando y probadas, es el Viernes 10 de Enero y la fecha para tener completos los documentos el Domingo 12 de Enero.

En cuanto al despliegue, vamos a realizarlo en Heroku a traves de Travis, como ya hemos comentado anteriormente solo se tomará encuenta la rama master de Decide-Zapdos-Censo. La URL de la aplicación desplegada en heroku es la siguiente: https://decide-zapdos-censo.herokuapp.com/admin/ . El usuario de la aplicación es : admin . La contraseña es: adminegc

Politica de nombrado e identificacion de los entregables

En Decide-Zapdos-Censo hemos decidido realizar los entregables con el siguiente formato :Decide-zapdos-censo-X.Y

La X hace referencia al Milestone en cuestión y la Y a la versión de ese Milestone. Por ejemplo Decide-zapdos-censo-3.0 : el milestone en cuestion es el milestone 3 y version actual del proyecto es la 0.

Licencias del proyecto

Unicamente tenemos una licencia heredada del proyecto inicial Decide y la licencia se llama GNU AFFERO GENERAL PUBLIC LICENSE. Dicha licencia se encuentra en decide/LICENSE.

10. Ejercicio de propuesta de cambio

Se desea añadir un campo nuevo a Censo: Barrio, de tipo CharField, e introducirlo en la funcionalidad *Census-Display*. Lo primero que se debe hacer es actualizar los modelos DecideUser y Census.

```
class DecideUser(User):
    fecha_nacimiento = models.DateField(null=True)
    genero = models.CharField(max_length=10, null=True,choices=(
        ('Masculino','Masculino'),
        ('Femenino','Femenino')
    ))
    provincia = models.CharField(max_length=20,null=True)
    localidad = models.CharField(max_length=20,null=True)
    barrio = models.CharField(max_length=20,null=True)
```

```python
class Census(models.Model):
    voting_id = models.PositiveIntegerField()
    voter_id = models.PositiveIntegerField()
    fecha_nacimiento = models.DateField(null=True)
    genero = models.CharField(max_length=10, null=True,choices=(
        ('Masculino','Masculino'),
        ('Femenino','Femenino')
    ))
    provincia = models.CharField(max_length=20,null=True)
    localidad = models.CharField(max_length=20,null=True)
    barrio = models.CharField(max_length=20,null=True)

    class Meta:
        unique_together = (('voting_id', 'voter_id'),)
```

A continuación deben realizarse las migraciones con los comandos makemigrations y migrate.

```
(decide-ws) jescoevas@jescoevas-VirtualBox:~/decide/decide$ python3 ./manage.py makemigrations
/home/jescoevas/.local/lib/python3.6/site-packages/psycopg2/__init__.py:144: UserWarning: The psycopg2 wheel package will be
renamed from release 2.8; in order to keep installing from binary please use "pip install psycopg2-binary" instead. For detai
ls see: <http://initd.org/psycopg/docs/install.html#binary-install-from-pypi>.
  """)
Migrations for 'authentication':
  authentication/migrations/0002_decideuser_barrio.py
    - Add field barrio to decideuser
Migrations for 'census':
  census/migrations/0004_census_barrio.py
    - Add field barrio to census
(decide-ws) jescoevas@jescoevas-VirtualBox:~/decide/decide$
```

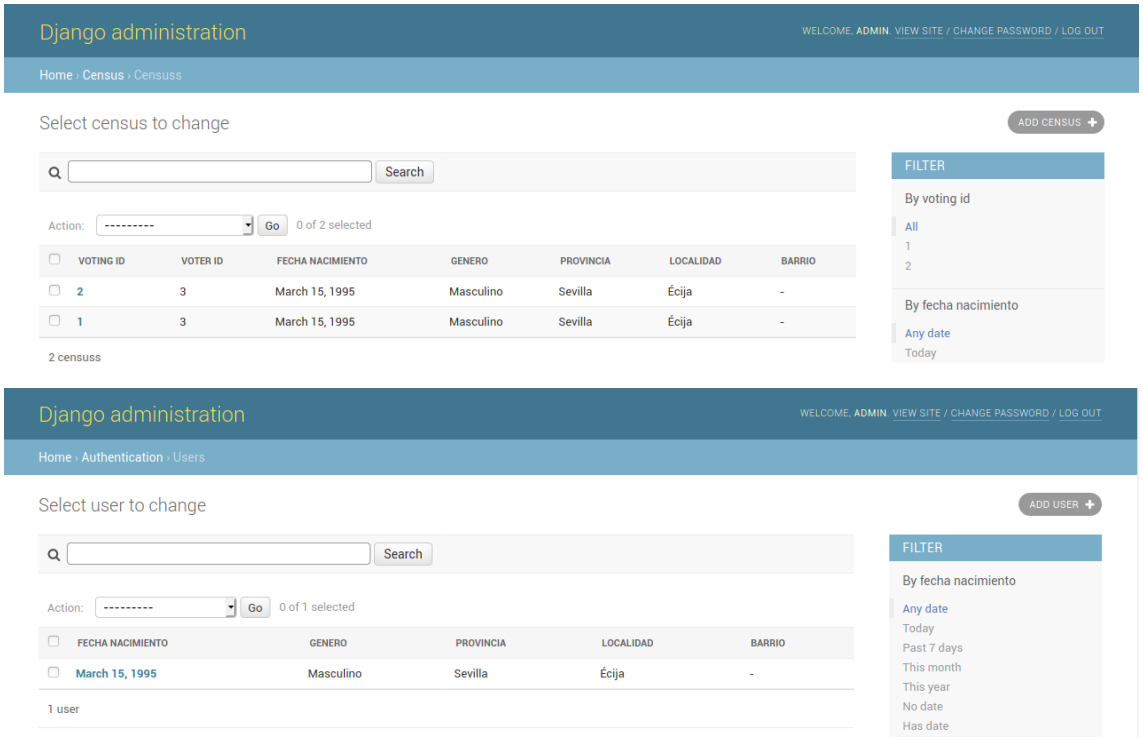
```
(decide-ws) jescoevas@jescoevas-VirtualBox:~/decide/decide$ python3 ./manage.py migrate
/home/jescoevas/.local/lib/python3.6/site-packages/psycopg2/__init__.py:144: UserWarning: The psycopg2 wheel package will be
renamed from release 2.8; in order to keep installing from binary please use "pip install psycopg2-binary" instead. For detai
ls see: <http://initd.org/psycopg/docs/install.html#binary-install-from-pypi>.
  """)
Operations to perform:
  Apply all migrations: admin, auth, authentication, authtoken, base, census, contenttypes, mixnet, sessions, store, voting
Running migrations:
  Applying authentication.0002 decideuser barrio... OK
  Applying census.0004 census barrio... OK
(decide-ws) jescoevas@jescoevas-VirtualBox:~/decide/decide$
```

Ahora deben actualizarse los archivos admin.py de los dos módulos.

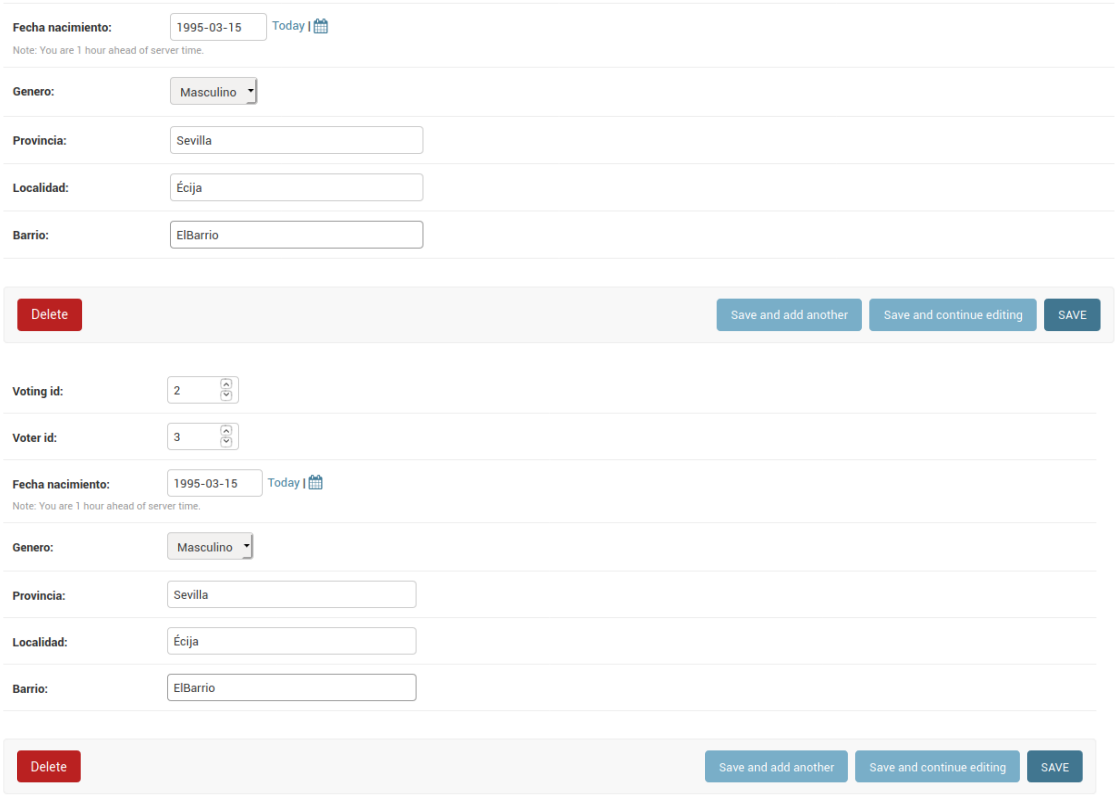
```python
class DecideUserAdmin(admin.ModelAdmin):
    list_display = ('fecha_nacimiento','genero','provincia','localidad','barrio',)
    list_filter = ('fecha_nacimiento','genero','provincia','localidad','barrio',)

    search_fields = ('fecha_nacimiento','genero','provincia','localidad','barrio',)


admin.site.register(DecideUser, DecideUserAdmin)
```

```python
class CensusAdmin(admin.ModelAdmin):
    list_display = ('voting_id', 'voter_id','fecha_nacimiento','genero',
    'provincia','localidad','barrio',)
    list_filter = ('voting_id', 'fecha_nacimiento','genero',
    'provincia','localidad','barrio',)

    search_fields = ('voter_id', 'fecha_nacimiento','genero','provincia','localidad','barrio',)


admin.site.register(Census, CensusAdmin)
```

Ejecutamos el comando runserver y comprobamos en el panel de administrador si la nueva propiedad es visible.



Se debe añadir el barrio desde el menú de edición del panel.



El modelo ya está actualizado tanto en base de datos como en panel de administrador. Lo siguiente es actualizar la funcionalidad.

Lo primero es actualizar las tablas del template census_display.html.

```
<thead class="thead-dark">
    <tr>
    <th scope="col">Voting_ID</th>
    <th scope="col">Voter_ID</th>
    <th scope="col">Fecha nacimiento</th>
    <th scope="col">Género</th>
    <th scope="col">Provincia</th>
    <th scope="col">Localidad</th>
    <th scope="col">Barrio</th>
    </tr>
</thead>
<tbody>
    {% for census in census_list %}

    <tr>
        <td>{{census.voting_id}}</td>
        <td>{{census.voter_id}}</td>
        <td>{{census.fecha_nacimiento}}</td>
        <td>{{census.genero}}</td>
        <td>{{census.provincia}}</td>
        <td>{{census.localidad}}</td>
        <td>{{census.barrio}}</td>
    </tr>

    {% endfor %}
</tbody>
```

```
<thead class="thead-dark">
    <tr>
        <th scope="col">ID</th>
        <th scope="col">Username</th>
        <th scope="col">Fecha nacimiento</th>
        <th scope="col">Género</th>
        <th scope="col">Provincia</th>
        <th scope="col">Localidad</th>
        <th scope="col">Barrio</th>
    </tr>
</thead>
<tbody>
    {% for user in user_list %}

    <tr>
        <td>{{user.id}}</td>
        <td>{{user.username}}</td>
        <td>{{user.fecha_nacimiento}}</td>
        <td>{{user.genero}}</td>
        <td>{{user.provincia}}</td>
        <td>{{user.localidad}}</td>
        <td>{{user.barrio}}</td>
    </tr>

    {% endfor %}
</tbody>
```

Comprobamos la vista en la ruta: localhost:8000/admin/census-display/. Se ha eliminado un censo de la base de datos para poder crearlo y ver que funciona correctamente.

List of census

Voting_ID	Voter_ID	Fecha nacimiento	Género	Provincia	Localidad	Barrio
2	3	March 15, 1995	Masculino	Sevilla	Écija	ElBarrio

List of votings

ID	Nombre	Fecha comienzo	Fecha final
2	Voting B	Dec. 24, 2019, 1:58 p.m.	Dec. 24, 2019, 2 p.m.
1	Voting A	Dec. 24, 2019, 1:59 p.m.	Dec. 24, 2019, 2 p.m.

List of users

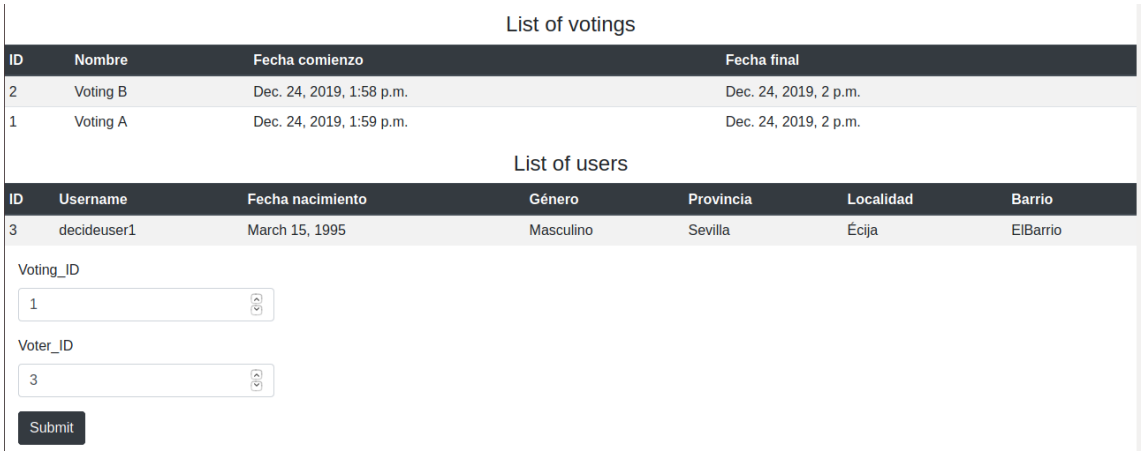
ID	Username	Fecha nacimiento	Género	Provincia	Localidad	Barrio
3	decideuser1	March 15, 1995	Masculino	Sevilla	Écija	ElBarrio

Lo siguiente es actualizar la función census_display en views.py.

```
try:
    census = Census(voting_id = voting_id, voter_id = user.id,
    fecha_nacimiento = user.fecha_nacimiento, genero = user.genero,
    provincia = user.provincia, localidad = user.localidad, barrio = user.barrio)
    census.save()
    context['success'] = 'Census has been successfully stored'
    return render(request,template,context)
except:
    messages.error(request, "This census already exists")
```

Y ya solo queda probar que funciona correctamente.

Introducimos datos correctos, ya que la validación para datos no válidos no se ha tocado y sigue funcionando correctamente.



List of votings

ID	Nombre	Fecha comienzo	Fecha final
2	Voting B	Dec. 24, 2019, 1:58 p.m.	Dec. 24, 2019, 2 p.m.
1	Voting A	Dec. 24, 2019, 1:59 p.m.	Dec. 24, 2019, 2 p.m.

List of users

ID	Username	Fecha nacimiento	Género	Provincia	Localidad	Barrio
3	decideuser1	March 15, 1995	Masculino	Sevilla	Écija	ElBarrio

Voting_ID

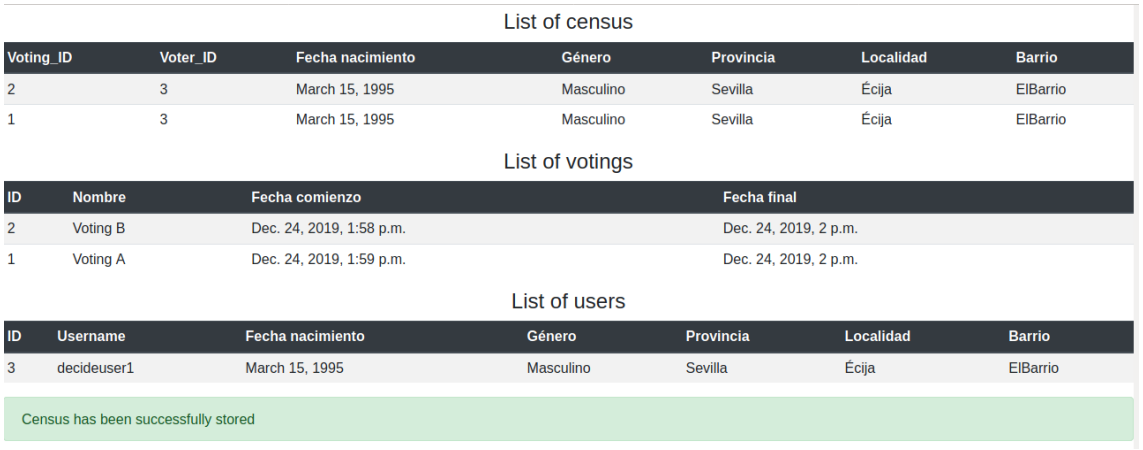
1

Voter_ID

3

Submit

Y el nuevo censo aparece con los datos correctos.



List of census

Voting_ID	Voter_ID	Fecha nacimiento	Género	Provincia	Localidad	Barrio
2	3	March 15, 1995	Masculino	Sevilla	Écija	ElBarrio
1	3	March 15, 1995	Masculino	Sevilla	Écija	ElBarrio

List of votings

ID	Nombre	Fecha comienzo	Fecha final
2	Voting B	Dec. 24, 2019, 1:58 p.m.	Dec. 24, 2019, 2 p.m.
1	Voting A	Dec. 24, 2019, 1:59 p.m.	Dec. 24, 2019, 2 p.m.

List of users

ID	Username	Fecha nacimiento	Género	Provincia	Localidad	Barrio
3	decideuser1	March 15, 1995	Masculino	Sevilla	Écija	ElBarrio

Census has been successfully stored

Como paso final realizamos el commit y push necesarios.

```
(decide-ws) jescoevas@jescoevas-VirtualBox:~/decide/decide$ git add .
(decide-ws) jescoevas@jescoevas-VirtualBox:~/decide/decide$ git commit
```

```
< Propuesta de cambio : Añadir propiedad Barrio >

- Se ha añadido la propiedad barrio a DecideUser y Census, y se ha actualizado la funcionalidad Census-Display -

# Cambios a ser confirmados:
#       modificado:     authentication/admin.py
#       nuevo archivo:  authentication/migrations/0002 decideuser barrio.py
#       modificado:     authentication/models.py
#       modificado:     census/admin.py
#       nuevo archivo:  census/migrations/0004 census barrio.py
#       modificado:     census/models.py
#       modificado:     census/templates/census display.html
#       modificado:     census/views.py
#
```

```
(decide-ws) jescoevas@jescoevas-VirtualBox:~/decide/decide$ git add .
(decide-ws) jescoevas@jescoevas-VirtualBox:~/decide/decide$ git commit
[census-newfields 77029a8] < Propuesta de cambio : Añadir propiedad Barrio >
 8 files changed, 52 insertions(+), 8 deletions(-)
 create mode 100644 decide/authentication/migrations/0002 decideuser barrio.py
 create mode 100644 decide/census/migrations/0004 census barrio.py
(decide-ws) jescoevas@jescoevas-VirtualBox:~/decide/decide$ git push
```

11. Conclusiones y trabajo futuro

Conclusiones

Entre las conclusiones sacadas, la mayor es la importancia de una buena gestión de los commits. Aunque la llevada a cabo es mejorable, sí nos ha permitido tener un mejor control de las versiones del código, sobre todo de forma visual en Github. Para mejorar la gestión actual se debería diseñar una plantilla antes de empezar a desarrollar y debería usarse también a la hora de mergear ramas.

También cabe destacar Codacy y Heroku. Codacy ha permitido tener un código más limpio y legible, lo cual es muy beneficioso a la hora de estudiar trabajo de otro desarrollador. Heroku ha resultado ser una forma muy sencilla de desplegar el proyecto en la nube, el cual es posible que usemos en futuros proyectos.

Trabajo futuro

1. Se han añadido los campos nuevos y nuevas formas de crear censos, pero no se ha implementado el filtro a la hora de votar. Debería implementarse, por ejemplo, que en una votación solo puedan votar usuarios de cierta localidad.
2. Añadir campos nuevos como: Parado, de tipo BooleanField; Ingresos anuales, de tipo IntegerField; Años trabajados, de tipo IntegerField.
3. Implementar Ley d'Hondt.