Decide-Articuno-Censo

Grupo	1
ID Opera	151 (sin loguear) / 159 (logueados)
Curso	2019/2020
Asignatura	Evolución y Gestión de la Configuración
Milestone	M3

Miembros del equipo	Esfuerzo (1-10)
Burdallo Narváez, Alba	10
García Borrego, Carlos	10
Guerrero Cuenca, Claudia	10
González Mendoza, Diego	10
López Hernández, Iván	10
Valle Zarza, Félix	10

Enlaces de interes	Enlace
Repositorio	https://github.com/egc-articuno/decide/tree/articuno-census
Sistema desplegado	https://decide-articuno.herokuapp.com/

Indice

RESUMEN EJECUTIVO

El proyecto que se va a mostrar a continuación recibe el nombre de **Articuno-Census**. Proviene de una plataforma que nos ofrecen, Decide.

Consiste en la división de Decide en distintos módulos donde cada uno de ellos es tratado por un equipo distinto, con el objetivo final de realizar una integración de todos estos módulos dando lugar a un sistema de integración continua y al trabajo en conjunto de todos estos equipos.

Para poder llevar a cabo este trabajo con la integración de muchos equipos, inicialmente se estableció de manera conjunta para todos una serie de características a cumplir por cada uno de ellos para poder tener una cierta coordinación y trabajar del mismo modo. Algunas de estas características son:

> Gestión del código, métricas a cumplir para la Buena calidad del código, formato de los commits, procedimiento de la gestión de incidencias, automatización de la integración continua del proyecto de Articuno, automatización de pruebas o despliegue en remoto.

Para este equipo, el modulo a tratar ha sido el módulo de **Censo**. Lo primero fue establecer una serie de funcionalidades que se iban a implementar en nuestro proyecto (una por miembro del equipo) para realizar un incremento de las funcionalidades que ya traía Decide. Algunas de estas funcionalidades han sido:

> Creación de censo por código postal, clonar censos, importer o exporter censos mediante CSV, …

Una vez establecido esto y siguiendo las mismas características para el flujo de trabajo que las establecidas en Articuno, cada el miembro del grupo procede a implementar su funcionalidad y los test pertinentes para cada una de ellas.

Cuando ya se tienen todas se van añadiendo a la rama de Articuno-Census; y cuando esta esté lista, se pasan las ramas de cada uno de los módulos a la rama Develop, en la cual quedaría el sistema completo con la integración de todos los equipos.

En cuanto a las automatizaciones y a los despliegues:

Se ha realizado la automatización de pruebas y la de integración continua tanto para el proyecto de Articuno entero como para nuestro proyecto de Articuno-Census en concreto.

El despliegue en Heroku también ha sido realizado para ambos proyectos (Articuno y Articuno-Census).

Como conclusiones a la hora del trabajo el equipo se pueden sacar las siguientes conclusiones:

- ✓ A nivel de Articuno entero, se han tenido que establecer suficientes características para definer la metodología de trabajo; ya que al ser un grupo muy numeroso es necesaria la coordinación.
- ✓ A nivel de Articuno-Census, se ha llevado a cabo un buen trabajo en equipo ya que la comunicación ha sido bastante frecuente lo que nos ha ayudado a lleva el proyecto actualizado.

1. Indicadores del Proyecto

Miembros del equipo	Horas	Commits	LoC	Test	Issues	Incremento
Burdallo Narváez, Alba	55	4	103	0	2	Export CVS
García Borrego, Carlos	55	14	325	2	2	Move people between census and delete all registered census
Guerrero Cuenca, Claudia	58	15	237	2	2	Create census by municipalities
González Mendoza, Diego	65	6	463	2	2	Authentication module models and basic functions of census
López Hernández, Iván	56	10	171	2	1	Clone/Reuse Census
Valle Zarza, Félix	56	15	204	1	1	Import CSV

2. Integración con otros equipos

Para la integración de los distintos equipos, se llegó a un acuerdo entre todos los módulos que componen decide-articuno.
En este acuerdo se estableció que implementaríamos el sistema de forma que se realicen distintas votaciones donde cada una de ellas corresponde a la votación en los habitantes que compartan en mismo **código postal**.
Por tanto, las votaciones se dividirán por el código postal que tenga cada usuario del sistema.
Nuestra función como censo será especificar a qué censo pertenece cada persona agrupándolo por código postal.

(anexo acuerdo formal del grupo)

Nuestro objetivo es integrarnos con aquellos módulos que tengan relación con el nuestro, para que pueda haber una buena y mejor coordinación; llevando a un buen funcionamiento del sistema.

Estos módulos o subsistemas con los que vamos a integrarnos son:

Decide – Articuno – Votación
El objetivo de integrarnos con votación es poder crear distintas votaciones donde las personas que puedan votar en cada una de ellas corresponda a un determinado censo.
Esto se da por el .JSON que censo le proporciona a votación con los datos de cada censo creado y los usuarios que componen cada uno de ellos.

(anexo .JSON proporcionado)

Decide – Articuno – Autenticación
El objetivo de integración con autentificación es pedir que nos proporcionen en el .JSON que nos envíen, el código postal de cada usuario del sistema; para que nosotros desde censo podamos listar y filtrar los distintos censos por ese atributo.

(anexo del documento donde requerimos el atributo)

3. Descripción del Sistema

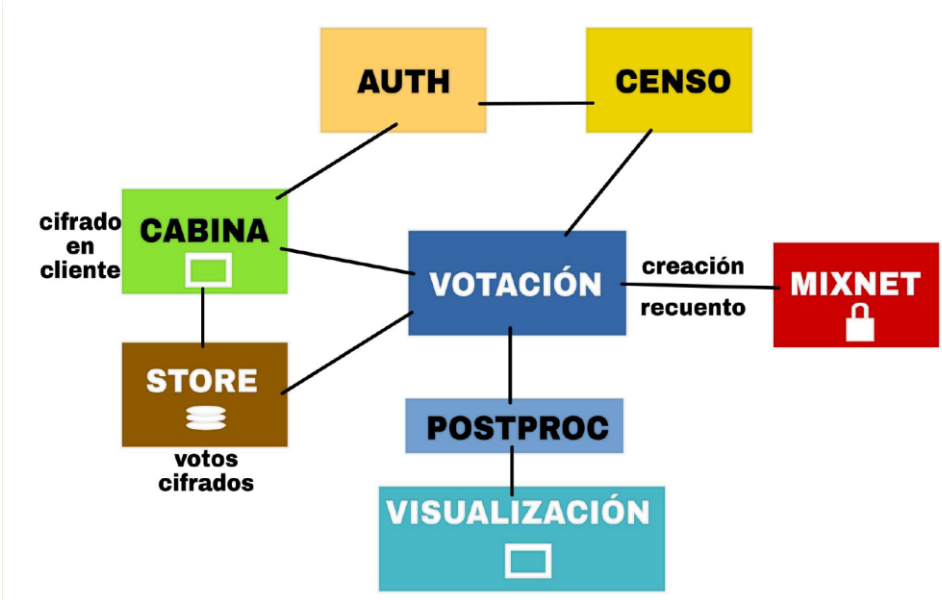
El sistema desarrollado a lo largo del curso en la asignatura Evolución y Gestión de la Configuración (EGC) ha sido DECIDE, DECIDE es un sistema que consiste en una plataforma de voto online.

Los sistemas de votos surgen a partir de la necesidad de tomar decisiones, existen numerosas formas de votar (a mano alzada, papel en urna) y también existen diferentes tipos de votaciones (pública, privada). El voto electrónico es el uso de ordenadores para realizar una votación pudiendo ser presencial o a distancia, este surge como un avance del voto. Algunas ventajas del voto electrónico serian el decrecimiento del coste a la hora de realizar una votación, ya que se ahorra bastante dinero al no usar papel, entorno de votación, mayor numero en el personal necesario, también supone una mejora para las personas con movilidad reducida, ancianos y personas que vivan fuera del país en el momento de la votación. Los sistemas de voto electrónico también suponen algunos inconvenientes como el ser mucho más complejo de auditar, es más fácil de atacar (servidores o clientes) o puede llegar a facilitar la compra de votos o extorsión a la hora de votar. A la hora de votar existen distintos tipos de votos, los votos pueden valer lo mismo, que cada voto tenga un peso, votación directa (en la que gana la opción más votada) o votación por preferencia.

DECIDE es un sistema de voto secreto electrónico y criptográficamente seguro punto a punto, modular y extensible, aporta una implementación de referencia y permite la extensión en cualquier tecnología cumpliendo una API mínima. DECIDE es un proyecto de software libre, publico con el código en GitHub y este está basado en agora-voting.

En lo referente a la arquitectura de DECIDE, está compuesta por módulos independientes, no depende de ningún lenguaje de programación, es posible distribuir la carga en diferentes maquinas dada su modularización y es más sencillo a la hora de abordar una tarea, estando los distintos módulos conectados mediante una API.

Los módulos por los que está compuesto DECIDE son base, auth, census, voting, booth, mixnet, store, postproc y visualizar.



A continuación, se explican algunas características de los distintos módulos:

- Base:
  - Mantiene las funcionalidades comunes.
  - Es similar a una librería para el resto de los módulos.
- Authentication:
  - Su principal utilidad es la autenticación de los votantes de las votaciones.
  - Inicialmente funciona con usuario y contraseña.
- Census:
  - Fuertemente relacionado con el módulo authentication.
  - Guarda un registro de quien puede votar y quien ha votado en las votaciones.
- Booth:
  - Este es esencialmente la cabina de votación donde entrará el votante para emitir su voto.
- Mixnet:
  - Es el modulo encargado de la parte criptográfica de DECIDE.
  - La mixnet vendría a ser una red de ordenadores los cuales generan una clave compartida para cifrar los votos y solo podrán ser descifrados cuando todos los integrantes se pongan de acuerdo.
  - En el recuento, el votante se desliga del voto, ya que cada autoridad realiza un barajado y se lo pasa al siguiente, anonimizando el voto en el proceso.
- Store:
  - Es el modulo encargado del cifrado de los votos.
- Postproc:
  - Postprocesado del voto, es donde con los números obtenidos en el recuento, se genera un resultado coherente al tipo de la votación.
  - En postprocesado se aplican todas las reglas necesarias, dependiendo del tipo de votación (simple, ley de paridad o ley de d'Hondt).
- Visualizer:
  - Este módulo se encarga de la visualización de los datos, vendría a ser un frontend, donde con los resultados obtenidos, los pintaría de forma bonita mediante graficas o tablas.

Como puede verse en la imagen cada uno de los distintos módulos tiene relación con otros:

- El módulo de booth tiene relación con los módulos authentication, store y voting.
- El módulo de store tiene relación con los módulos de booth y voting.
- El módulo de authentication tiene relación con los módulos de booth y census.
- El módulo de census tiene relación con los módulos de authentication y voting.
- El módulo de voting tiene relacionado con los módulos de census, booth, store, Postproc y mixnet.
- El módulo de mixnet tiene relación con el módulo de voting.
- El módulo de postproc tiene relación con los módulos de voting y visualizer.
- El módulo de visualizer está relacionado con el módulo de Postproc.g

Los cambios realizados desde nuestro grupo se han realizado sobre el módulo de census, encargado de guardar un registro de quien puede votar y quién no. Para el desarrollo del proyecto se propusieron algunas mejoras para el estado en el que se encontraba el módulo census en el momento de empezar. Los cambios propuestos fueron los siguientes:

- Importar archivos en formato CSV (comma-separated values).
- Exportar los censos existentes a archivos CSV.
- Mover a personas de un censo a otro distinto.
- Borrar todos los censos existentes.
- Clonar censos existentes.
- Se implementó una página principal nueva ya que la existente no dejaba trabajar con ella bien.
- Se añadió la posibilidad de crear censos mediante el código postal asociado a los usuarios.
- Se implementó la entidad 'usuario' añadiendo el atributo código postal.

Estas fueron las nuevas características añadidas al módulo census, con adición de la implementación de la entidad usuario correspondiente a authentication, ya que no disponíamos de otro grupo con el que integrarnos que tuviera dicho módulo.

4. Visión global de proceso de desarrollo

El proceso para el desarrollo de software, también denominado ciclo de vida del desarrollo de software es una estructura aplicada al desarrollo de un producto de software. Hay varios modelos a seguir para el establecimiento de un proceso de desarrollo de software, cada uno de los cuales describe un enfoque diferente para diferentes actividades que tienen lugar durante el proceso.

Los modelos de desarrollo software son una representación abstracta de una manera en particular. Realmente no representa cómo se debe desarrollar el software, sino que se trata de un enfoque común. Puede ser modificado y adaptado de acuerdo a las necesidades del software en proceso de desarrollo. Hay varios modelos para perfilar el proceso de desarrollo, cada uno de los cuales cuenta con pros y contras. Algunos de ellos son el modelo de cascada, modelo de espiral, desarrollo iterativo e incremental, desarrollo ágil, codificación y corrección y orientado a la reutilización. En el caso de nuestro proyecto, hemos optado por el desarrollo iterativo e incremental.

El desarrollo iterativo e incremental, que no es más que un conjunto de tareas agrupadas en pequeñas etapas repetitivas (iteraciones), es uno de los más utilizados en los últimos tiempos ya que, como se relaciona con novedosas estrategias de desarrollo de software y una programación extrema, es empleado en metodologías diversas.

El modelo consta de diversas etapas de desarrollo en cada incremento, las cuales inician con el análisis y finalizan con la instauración y aprobación del sistema.

En este modelo de desarrollo se planifica un proyecto en distintos bloques temporales que se le denomina iteración. En una iteración se repite un determinado proceso de trabajo que brinda un resultado más completo para un producto final, de forma que quien lo utilice reciba beneficios de este proyecto de manera creciente.

Lo que se busca es que en cada iteración los componentes logren evolucionar el producto dependiendo de los completados de las iteraciones antecesoras, agregando más opciones de requisitos y logrando así un mejoramiento mucho más completo.

La idea principal detrás del mejoramiento iterativo es desarrollar un sistema de programas de manera incremental, permitiéndole al desarrollador sacar ventaja de lo que se ha aprendido a lo largo del desarrollo anterior, incrementando, versiones entregables del sistema. Los pasos claves en el proceso son comenzar con una implementación simple de los requerimientos del sistema, e iterativamente mejorar la secuencia evolutiva de versiones hasta que el sistema completo esté implementado. En cada iteración, se realizan cambios en el diseño y se agregan nuevas funcionalidades y capacidades al sistema.

El proceso en sí mismo consiste de:

- Etapa de inicialización
- Etapa de iteración
- Lista de control de proyecto

Etapa de inicialización

Se crea una versión del sistema. La meta de esta etapa es crear un producto con el que el usuario pueda interactuar, y por ende retroalimentar el proceso. En nuestro caso, esta etapa ya se da por concluida de primeras, ya que en la asignatura se nos proporciona un primer código de base que funciona sin que tengamos realizar ninguna modificación excepcional, más allá de instalar ciertos programas o modificar los archivos con la configuración del proyecto para que funcione. Para la modificación de estos archivos se ha utilizado la herramienta Visual Studio Code y para el despliegue se ha utilizado una aplicación en Windows llamada Ubuntu, que permite usar la terminar de Ubuntu y correr líneas de comando de Ubuntu que incluyen bash, ssh, git, apt y muchas otras. Para visualizar el despliegue del proyecto se ha utilizado el navegador Google Chrome.

Para guiar el proceso de iteración se crea una lista de control de proyecto, que contiene un historial de todas las tareas que necesitan ser realizadas. Incluye cosas como nuevas funcionalidades para ser implementadas. Esta lista de control se revisa periódica y constantemente. Para la realización de esta lista de control se ha utilizado la herramienta Word.

Etapa de iteración

Esta etapa involucra el rediseño e implementación de una tarea de la lista de control de proyecto, y el análisis de la versión más reciente del sistema. La meta del diseño e implementación de cualquier iteración es ser simple, directa y modular, para poder soportar el rediseño de la etapa o como una tarea añadida a la lista de control de proyecto. El análisis de una iteración se basa en la retroalimentación del usuario y en el análisis de las funcionalidades disponibles del programa. La lista de control del proyecto se modifica bajo la luz de los resultados del análisis. En nuestro caso no ha sido necesario modificar la lista de control del proyecto, ya que hemos realizado nuestras tareas sin necesidad de modificaciones.

En esta etapa se han implementado las tareas de la lista de control del proyecto tal y como se creó en la etapa anterior, siempre tratando de que fueran tareas sencillas, por si en algún futuro tuvieran que sufrir un rediseño. Las tareas se han implementado modificando los archivos del proyecto con la herramienta Visual Studio Code y haciendo correr los tests con los comandos pertinentes en la aplicación Ubuntu para Windows, nombrada anteriormente. Cada vez que se implementó una iteración, se visualizó en la herramienta Google Chrome realizando previamente una serie de comandos en la herramienta Ubuntu.

Propuesta de cambio al sistema

En este apartado se propone un cambio al sistema para explicar cómo se abordaría todo el ciclo anteriormente descrito hasta tener todo el cambio en producción.

En el caso de nuestro proyecto, somos los encargados de la parte de censo, por lo que un cambio que se le podría realizar al proyecto, sería el de poder borrar varios censos de una sola vez, en lugar de tener que borrar de uno en uno, ya que es más tedioso y facilitaría la experiencia del cliente a la hora de usar el proyecto.

Comenzamos con la etapa de inicialización, en la que ya tendríamos nuestra primera versión del sistema funcional. Se añadiría entonces a la lista de control de proyecto la tarea de realizar este cambio propuesto en el sistema. Dado que no es un cambio muy grande, no haría falta dividirlo en tareas más sencillas para su posterior implementación, simplemente se le asignaría a un miembro del grupo.

En la etapa de iteración, el miembro con la tarea nueva asignada la realizaría modificando archivos con la herramienta Visual Studio Code y añadiendo, si fuera necesario, vistas nuevas. En este caso, podría añadir un método en el archivo views.py de la carpeta census que realice el borrado de varios censos al ser estos seleccionados. Además, podría añadir una vista específica en la que se muestren todos los censos disponibles para ser borrados con un botón que debería añadir.

Una vez realizado todo esto, se le entregaría la nueva versión del proyecto al cliente para que la testeara y viera si fuera necesario un rediseño de la tarea recientemente implementada, ya fuera una ampliación de la tarea, un rediseño de la vista, o cualquier otro cambio.

5. Entorno de desarrollo

El entorno de desarrollo usado ha sido Visual Studio Code. Visual Studio Code es un editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Este incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. También es personalizable, por lo que los usuarios pueden cambiar el tema del editor, los atajos de teclado y las preferencias. Además, es gratuito y de código abierto. Su descarga se puede realizar desde la página oficial de Visual Studio Code.

Visual Studio Code es compatible con varios lenguajes de programación (en nuestro caso nos interesa para Python) y un conjunto de características que pueden o no estar disponibles para un idioma dado. Muchas de las características de Visual Studio Code no están expuestas a través de los menús o la interfaz de usuario. Más bien, se accede a través de la paleta de comandos o a través de archivos .json. La paleta de comandos es una interfaz de línea de comandos. Sin embargo, desaparece si el usuario hace clic fuera de él o presiona una combinación de teclas en el teclado para interactuar con algo que está fuera de él. Esto también se aplica a los comandos que requieren mucho tiempo. Cuando esto sucede, el comando en progreso se cancela.

Visual Studio Code se puede extender a través de complementos, disponibles a través de un repositorio central. Esto incluye adiciones al editor y soporte de idiomas. Una característica notable es la capacidad de crear extensiones que analizan código, como linters y herramientas para análisis estático, utilizando el Protocolo de Servidor de Idioma.

La versión de Visual Studio Code usada es la 1.41.1.

A continuación, se pasará a describir los pasos a seguir para instalar el sistema y hacerlo funcionar al completo.

Primero deberemos descargar Git para Windows, ya que con esta herramienta podremos clonar el repositorio necesario en nuestro ordenador. Para la descarga e instalación de esta herramienta, podremos seguir los pasos de este link: https://filisantillan.com/como-instalar-git/#windows. La versión de Git usada es la 2.21.0.

Una vez instalado Git, en nuestro escritorio, con el botón derecho del ratón seleccionaremos la opción git bash here, a continuación, podremos poner el comando git clone, seguido del repositorio a clonar, en nuestro caso sería https://github.com/egc-articuno/decide.git. Así ya tendríamos creada una carpeta con nuestro proyecto en el escritorio.

A continuación, instalaremos la aplicación Ubuntu para Windows, a través de la propia Microsoft Store, a la cual podremos acceder desde la pestaña Windows de nuestro ordenador. La versión de Ubuntu usada es la 1804.2019.521.0.

Una vez instalada y abierta la aplicación, deberemos introducir una serie de comandos en cierto orden para instalar algunos requisitos previos, para que al final podamos desplegar en local nuestro proyecto sin problemas. Puede que algún comando nos requiera instalar previamente otra cosa, en ese caso, instalamos lo que nos pida con el comando que nos proporcione. Los comandos a introducir son los siguientes:

- cd /mnt/c/users/nombreDeUsuario/desktop/decide (en nombreDeUsuario pondremos el nombre de usuario que tengamos en nuestro ordenador)
- pip3 install Django –user (para instalar Django)
- pip3 install psycopg2
- pip3 install –r requirements.txt
- sudo apt-get install postgresql (para instalar postgresql)
- sudo service postgresql start (para arrancar postgresql)
- sudo su – postgres
- psql -c "create user decide with password 'decide'" (con esto creamos un usuario)
- psql -c "create database decide owner decide" (con esto creamos la base de datos y le decimos que el dueño es el usuario previamente creado)
- Ahora saldremos de postgres escribiendo el comando “exit”
- cd decide (para meternos otra carpeta que se llama decide)
- python3 ./manage.py migrate (para que todas las tablas sean creadas en la base de datos)
- python3 ./manage.py createsuperuser (para crear un super usuario)
- python3 ./manage.py runserver (con este último comando arrancaremos nuestro proyecto y podremos visualizarlo)

Para poder visualizar el proyecto una vez hemos realizado los siguientes comandos y que no hayamos obtenido ningún error, en Google Chrome (versión 79.0.3945.88) introduciremos la url: localhost:8000/admin para ver la vista de administrador.

Si quisiéramos ir a la vista principal de nuestro módulo, el de censo, la url sería: localhost:8000/census/listCensus.

Una vez que hayamos comprobado que todo funciona, para parar el proceso lo único que tendremos que hacer es en la aplicación Ubuntu pulsar la combinación de botones Ctrl+c.

6. Gestión de incidencias

En este apartado se especifica cómo se han gestionado las incidencias tanto en el equipo de Articuno como en nuestro módulo específico, el censo.

1. Incidencias externas

En cuanto a Articuno, se ha acordado un sistema para crear las issues bastante fácil de entender y que todos debemos poner en común. Hay tres tipos de incidencias generales, para las features, para los bugs y para las pull request donde se han creado las respectivas plantillas para cada una. Estas plantillas se pueden encontrar en el repositorio de GitHub dentro de la ruta .github/ISSUE_TEMPLATE.

Se ha acordado que no se abrirá una incidencia para preguntar ayudas sobre el código, se proporciona el enlace a Stack Overflow donde todos los miembros del grupo pueden buscar sus dudas individualmente. También, el proyecto solo aceptará pull request de las incidencias que estén abiertas, si se quiere sugerir una nueva funcionalidad o cambio, se deberá debatir en la issue primero y si la pull request es para solucionar un bug, éste deberá estar descrito en otra issue con la plantilla de bug correspondiente.

Para las incidencias de features se seguirá el siguiente patrón:

- El título de la issue, que deberá empezar con [FEATURE] y seguidamente el nombre de la tarea que se va a implementar.
- ¿En qué módulo se está trabajando? Donde se especifica cuál es el módulo al que pertenece esta incidencia.
- ¿De qué trata la nueva feature? Aquí se deberá especificar una descripción breve y concisa del problema que hay para que se necesite hacer la tarea concreta.
- Descripción de la solución, donde se describe lo que se quiere que pase al realizar la tarea.
- Descripción breve y concisa de las soluciones o features alternativas que se han considerado.
- Texto adicional o capturas de pantalla sobre la feature específica.

Para las incidencias sobre los bugs se ha creado la plantilla teniendo en cuenta lo siguiente:

- El título de la issue, que debe empezar con [BUG] y seguidamente una muy breve descripción del problema.
- El comportamiento esperado, que describe qué debería ocurrir si este bug no se diera.
- El comportamiento actual, qué es lo que ocurre en vez de lo esperado.
- Una posible solución, si se conoce, para arreglar el problema dado
- Especificar los pasos que llevan al error mediante un enlace a un ejemplo en vivo o una lista de pasos a seguir, indicando si es necesario las partes del código que hay que reproducir.

- El contexto en el que el error ocurre, como afecta, que es lo que se está tratando de llevar a cabo. Proveer un contexto que ayude a dar con una solución que sea útil.

- Una descripción más detallada del cambio que se está proponiendo.

- Si es posible, sugerencia de una idea para la implementación.

Para crear las incidencias de pull request, se seguirán los siguientes pasos:

- El título comenzará con [Pull request] y el nombre que se vea adecuado para su breve descripción.

- Se describirá en detalle los cambios que se hayan realizado a la hora de hacer esta pull request.

- Especificar la incidencia a la que está relacionada y un enlace a ésta.

- ¿Por qué se ha requerido el cambio? ¿Qué problema soluciona? Establecer un contexto y, si es necesario, enlazar la incidencia abierta a la que le ha dado la solución.

- Describir en detalle cómo se han probado los cambios, incluyendo detalles del entorno de testing y los test que se han ejecutado para ver como los cambios afectan a las otras áreas del código.

- Si es necesario, incluir capturas de pantalla.

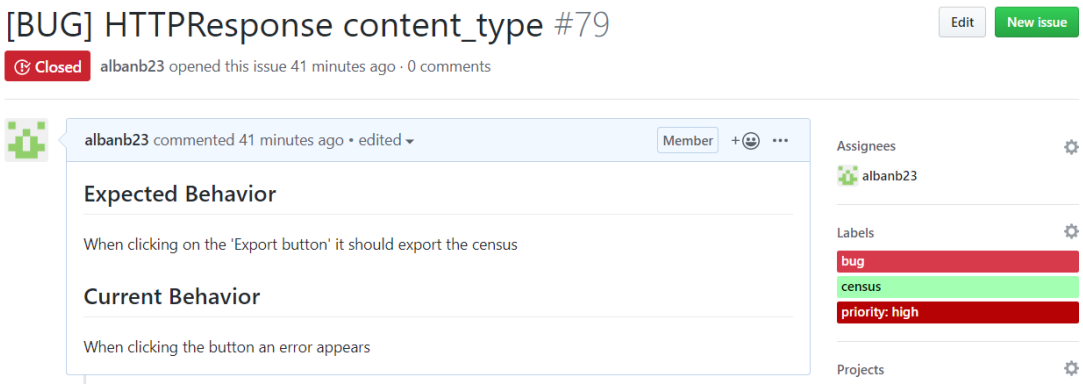
A parte de las plantillas, cada incidencia tendrá asignadas unas etiquetas que ayuden a saber de un vistazo qué módulo la ha creado y de que se trata. A parte de las ya establecidas en el propio GitHub por defecto se han creado una para cada módulo (booth, census, postproc, voting), una para integraciones para las issues de pull request y otras para especificar la prioridad (high, medium, low).

También para cada incidencia se establece a qué milestone pertenece y cada módulo asignará individualmente al miembro del grupo correspondiente a esa issue (bien el encargado de realizar esa feature, el que ha encontrado el bug o el que se dispone a hacer una pull request) según el tipo.

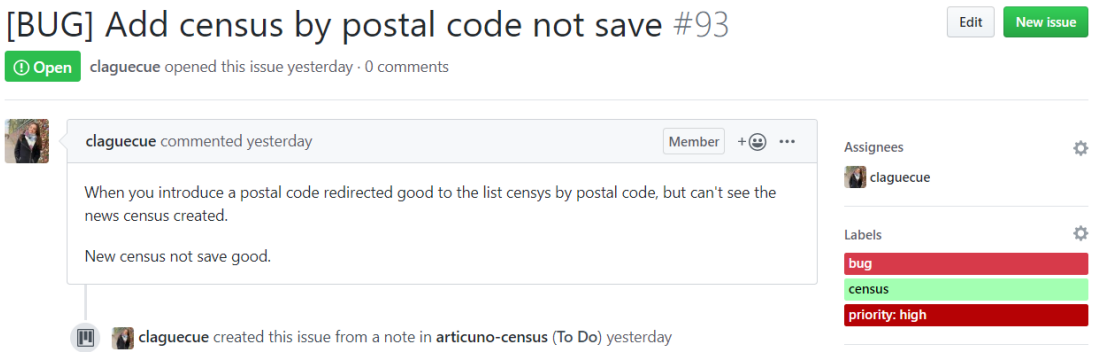
2. Incidencias internas

Nuestro módulo individualmente ha acordado gestionar las incidencias internamente de la misma forma que lo hacemos con el equipo de Articuno al completo ya que así es más fácil de gestionar todo conjuntamente. Seguiremos tanto las plantillas establecidas para las issues como las etiquetas personalizadas.

Algunos ejemplos de incidencias que han ocurrido en nuestro equipo han sido las siguientes:



Como se ve en la imagen, se ha seguido la plantilla establecida en el proyecto para las issues de bug. En esta incidencia en concreto, sólo se han especificado cómo debería funcionar y cómo funciona la feature debido al bug ya que se trataba de un error básico. También, se indican las etiquetas correspondientes.



Esta incidencia se trata de un error a la hora de guardar censos. Como vemos, se ha especificado qué es lo que falla a la hora de darle al botón de guardar. Se indican las etiquetas y las plantillas.

7. Gestión del código fuente

En este apartado, explicaremos los procesos, técnicas y herramientas que hemos utilizado para gestionar nuestro proyecto tanto a nivel general (equipo Articuno), como a nivel de subgrupo, en nuestro caso Censo.

Para explicar nuestra gestión del código, debemos empezar hablando de Git [1] y GitHub [2].
El primero es un software de control de versiones que nos permite administrar nuestro proyecto. Y el segundo es la plataforma donde se almacena nuestro código.

Hay dos razones por las que hemos usado las dos cosas mencionadas anteriormente. La primera es porque partimos de un proyecto ya subido a esta plataforma y la segunda se debe al gran rendimiento y facilidad de usabilidad con respecto a sus otras competencias como puede ser Subversion, CVS, Perforce o ClearCase.
Cómo acabo de decir, nuestro proyecto Articuno parte de otro, para ello hemos realizamos un fork [3] del proyecto EGCETSII, que a su vez parte de otro, llamado Wadobo.

Para la organización entre los distintos subgrupos (auth, cabina, censo, postprocesado, visualización y votación), estuvimos discutiendo sobre si trabajar usando ramas [4] para cada subgrupo o si trabajar con fork.
Las conclusiones a las que llegamos para ambos casos, fueron las siguientes:

Ramas
Para ello utilizaríamos GitFlow [5], crearíamos una rama por grupo con la siguiente estructura: articuno-subgrupo. Y posteriormente, cada subgrupo podría trabajar con más subramas.
Algunos inconvenientes que se plantearon fueron que, al trabajar de esta forma, cualquier otro compañero podría eliminar tu rama por accidente o algún caso similar.

Forks
Tiene la ventaja de que trabajarías de forma aislada, lo que puede evitar problemas con otros subgrupos. Aunque tenía un gran inconveniente, el hacer pull request [6] daría muchos problemas y una importante pérdida de tiempo para solucionar esta parte en un futuro.

Tras votarlo, la estrategia que íbamos a seguir sería la siguiente:
Desde Articuno, lo primero que íbamos a hacer es la creación de una rama denomida develop, la cual será donde se compruebe que todo funciona correctamente antes de subirla a la rama máster, que es la rama que será entregada con la finalización del trabajo. Posteriormente, cada subgrupo realizará la creación de una rama con la estructura anteriormente comentada “articuno-subgrupo”. Y luego, cada subgrupo a nivel interno puede trabajar con ramas o con un fork. En nuestro caso como miembros de censo trabajaremos con ramas, una para feature con la siguiente estructura: “articuno-censo-feature”. Luego cada proyect manager, se tiene que encargar de unir las ramas correspondientes y subirlas a develop. Para ser mergeado a dicha rama, dos proyect manager distintos han de supervisarlo y darle el visto bueno antes de unirlo, para así evitar conflictos entre tantos miembros del equipo.

En cuanto a los commit para la subida de archivos al repositorio, hemos preparado una pequeña guía que seguir para que sea más fácil y comprensible el poder identificar alguna

parte concreta de un compañero sin tener que preguntarle a este directamente, o que sea un quebradero de cabeza, buscar una parte concreta que quieres recuperar.

Los commit van a tener que seguir la siguiente estructura:

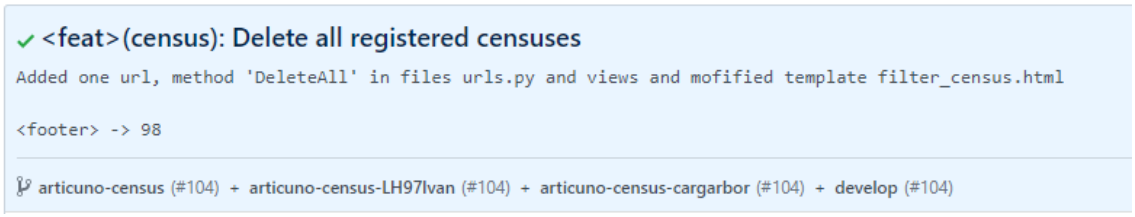
<tipo>(<subgrupo>): </tipo>
<cuerpo>
<pie página> -> Referencias de los problemas

Tipos:

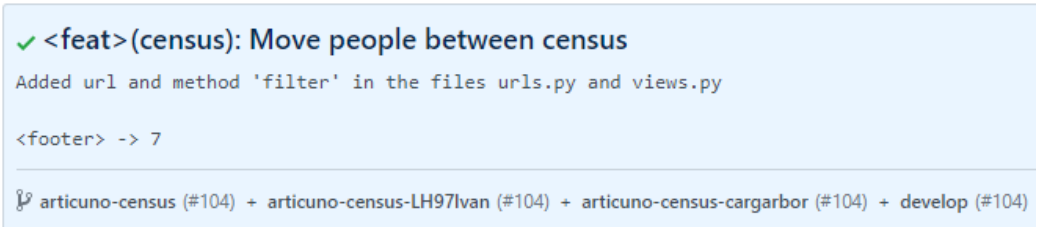
- Feat: Nuevo característica.
- Fix: Arreglar bug.
- Docs: Cambiar documentación.
- Style: Formato, colores, y cualquier cosa relaciona, que no requiere de código que no sea css.
- Refactor: Refactorizar el código de producción.
- Test: Añadir o actualizar pruebas.
- Chore: Actualizar tareas difíciles, sin cambiar el código de producción.

Vamos a explicarlo, primero indicamos a cuál de los siete tipos pertenece el archivo que vamos a subir al repositorio, luego indicamos a que subgrupo o módulo afecta el archivo. Después hacemos una breve descripción de lo que estamos haciendo, para así cualquier persona ajena al código puede comprender lo que se ha subido, y por último, indicamos a que tarea (número) pertenece el documento subido. Otra cosa que no comenté anteriormente, es que hemos acordado hacer los commit en inglés, para así internacionalizar los comentarios que dejemos.

Vamos a mostrarles un par de ejemplos que hemos hecho con sus enlaces correspondientes.



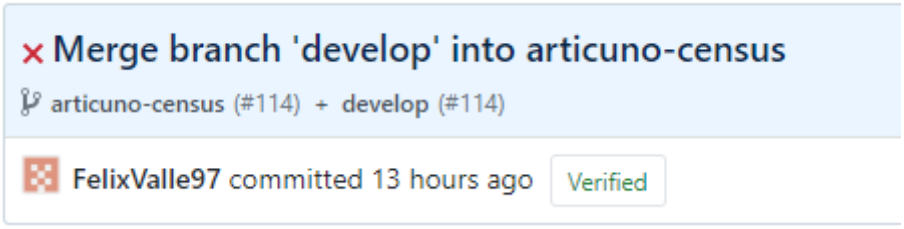
Enlace: https://github.com/egc-articuno/decide/commit/53011d7a2939d66c9ff6533d425f4ad71ca38494



Enlace: https://github.com/egc-articuno/decide/commit/a78fd28a0c94eac88daa58816735d36f1dd0d171

Otro asunto a tratar es el momento de realizar los commit, hemos acordado tanto a nivel interno como con los otros grupos ir subiendo nuestro código cada vez que realicemos una feature o realicemos un cambio en el código ya subido, para que así quede reflejado cada vez que vamos añadiendo cosas, lo que nos permite tener un flujo temporal real de lo que hemos ido haciendo, en lugar de subir todo el código a la vez y que pueda generar confusiones o problemas.

Por último, se puso como recomendación que el código debía estar preparado el día anterior a la entrega para cada subgrupo (censo, cabina, votación, postprocesado) y posteriormente subido a la rama develop para que así los project manager de cada subgrupo prueben que todo funciona correctamente y en el caso de que se produzca algún problema, poder corregirlo. Y posteriormente hacer un pull request a la rama máster.



Enlace: https://github.com/egc-articuno/decide/commit/9631c2f76404c530d7a69dec209117d775207cb2

Anexo

[1] Git: https://git-scm.com/ .

[2] GitHub: https://github.com/ .

[3] Fork: https://help.github.com/en/enterprise/2.17/user/getting-started-with-github/fork-a-repo .

[4] Ramas/Branch: https://git-scm.com/book/es/v2/Ramificaciones-en-Git-%C2%BFQu%C3%A9-es-una-rama%3F .

[5] GitFlow: https://danielkummer.github.io/git-flow-cheatsheet/index.es_ES.html .

[6] Pull Request: https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests .

8. Gestión de la construcción e integración continua

En esta sección se explican los procesos, técnicas y herramientas empleadas para el correcto desarrollo de la integración continua en el proyecto tanto a nivel general (equipo Articuno), como a nivel de subgrupo, en nuestro caso Censo.

Para evaluar y controlar la calidad del código, se acordó por consenso con el resto de los equipos integrantes de Articuno el uso de Codacy, estableciendo unos criterios mínimos que deben superarse a nivel de subgrupo antes de realizar la integración a nivel general.

Para comprobar la integridad del código y su correcto funcionamiento se acordó emplear Travis. Mediante esta herramienta, se crea un proceso en el cual se instalan las dependencias necesarias en un entorno cerrado, y se realizan una serie de comprobaciones para verificar la integridad del código, todo esto queda especificado en un archivo oculto llamado *.travis.yml*.

Se configuró uno común para todos los subgrupos de Articuno, y se lanza una fase de build cada vez que un miembro realiza un commit a Github, bien sea su rama personal o no, para así detectar posibles errores en etapas tempranas del desarrollo de las nuevas funcionalidades que puedan afectar o interferir con el resto.

Debido a esto, para hemos acordado realizar los commits de cara a la integración con el resto de los subgrupos mediante pull requests, ya que así podrá comprobarse antes de unir los submódulos que cada uno de ellos cumple los requisitos establecidos de forma independiente, para así no disminuir criterios como el porcentaje de cobertura del código a nivel grupal y afectar así negativamente al resto de equipos. Dichas pull request podrán ser creadas por cualquier integrante de Articuno, pero es conveniente que sea otro el que la acepte, para así tener un mínimo de control en la rama.

Una vez se alcance una versión estable del proyecto en la rama develop, deberán llevarse dichas funcionalidades a la rama master. Esta tarea recae sobre los coordinadores de cada subgrupo, que son los que poseen permisos en dicha rama. Uno de los coordinadores deberá realizar una pull request, y otros dos serán los encargados de revisarla y, una vez se cercioren de que todo funciona correctamente y no se observa ningún posible problema, aceptarla.

9. Gestión de liberaciones, despliegue y entregas

En esta sección vamos a explicar cómo nos gestionamos, tanto a nivel de subgrupo 'Census' como a nivel general 'Articuno'.

PROCESO PARA LAS LIBERACIONES

En la creación de un proceso para realizar las liberaciones, se define un proceso en el cual se crea una planificación con los momentos específicos en los que cada miembro del grupo podrá subir sus implementaciones y pruebas individuales al repositorio común de todo el equipo.

Establecer esta planificación concreta se da para proteger los servicios ya existentes y que la introducción de nuevos no influya en estos. También se usa para que no se haga una integración de nuevos servicios de forma paralela, pudiendo llevar esto a grandes conflictos.

Para la gestión de las liberaciones podemos distinguir 2 procesos: uno que se llevará a cabo por nuestro equipo interno conformado por los integrantes del módulo de censo, y otro que es el establecido a nivel común por todos los módulos que componen Decide-Articuno.

A) En el equipo interno – Censo

De acuerdo con todos los miembros del equipo que componen censo se ha establecido el siguiente proceso para hacer las liberaciones:

Cada miembro del equipo trabajará en su funcionalidad asignada, tanto la implementación como las pruebas de esta. Cuando se tenga una de ellas lista, se procederá a subirse a la rama de Censo, donde se une todo el trabajo de este módulo.

Para no producir conflictos en la integración de varios servicios a la vez, se establece un día y franja horaria para cada miembro del equipo. De modo que cada uno de ellos cuando quieran liberar su trabajo deben hacerlo en su horario establecido:

- Lunes, 9:00 a 15:00 → Diego
- Lunes, 15:00 a 21:00 → Félix
- Martes, 9:00 a 15:00 → Iván
- Martes, 15:00 a 21:00 → Claudia
- Miércoles, 9:00 a 15:00 → Alba
- Miércoles, 15:00 a 21:00 →Carlos

B) En el equipo completo – Articuno

Aquí se seguirá el proceso que se estableció de forma común (en una reunión) en todos los grupos que conforman Decide-Articuno, para que la integración de estos pueda darse de manera satisfactoria.

El proceso a seguir para la integración de los distintos grupos es el siguiente:

Cada grupo procederá a liberar el trabajo realizado en su rama y unirlo a la rama Master una vez cada dos semanas (empezando la semana del 25 de noviembre). Si existe algún grupo sin nuevo trabajo funcional nuevo, procederá a no hacer nuevas liberaciones hasta su nuevo turno.

En las semanas que sí se hacen liberaciones se establecen estos días de la semana para que cada grupo haga su trabajo en el día correspondiente:

- Lunes → Cabina
- **Martes → Censo**
- Miércoles → Votacion
- Jueves → PostProcesado
- Viernes → Authentification

LICENCIA SOFTWARE DEL PROYECTO

Elegir una licencia software consiste en establecer una serie de condiciones bajo las cuales se puede usar el software elaborado por este equipo.

La licencia software que hemos establecido para el uso interno de nuestro código es la misma que la establecida a nivel general para el equipo de Articuno (con todos sus módulos), ya que compartimos reposiotiro, y es la siguiente:

Dominio público (Public Domain)

Todo el código realizado por el equipo de Articuno, y por cada uno de sus módulo en particular, se encuentra subido a un repositorio de GitHub el cual está abierto públicamente. Esto implica que está a disposición de todas las personas que lo requieran para poder usarlo, copiarlo, modificarlo, … o cualquier otra actividad.

PROCESO PARA EL DESPLIEGUE

Antes de realizarse un despliegue para una determinada entrega, se comprueba en **CODACY** que se cumplen las validaciones que se establecieron comunes en todos los módulos de Decide-Articuo. Estas validaciones son:

- Test coverage > 65 %
  (La cobertura de los test tiene que cubrir más del 65 %)

- Cyclomatic complexity < 8 %
  (La complejidad ciclomática del código no debe ser mayor del 8 %)

- Duplicated code < 15 %
  (No se debe superar el 15 % de código duplicado)

- Unused code < 5 %
  (No se debe superar el 5 % de código sin uso)

- 100 % test passed
  (El 100 % de los test tienen que haber sido pasados)

Tres tipos de despliegue distintos dependiendo el momento del proyecto en el que nos encontremos:

a) Mientras se hacen los incrementos en las funcionalidades:
Cada miembro del equipo ha conseguido el despliegue local de la aplicación en su ordenador mediante la consola de Ubuntu. Esto hace que podamos visualizar el contenido de esta en nuestros ordenadores para hacer las pruebas funcionales que sean necesarias.

b) Antes de llevarse a cabo una entrega:
Se procederá a hacer el despliegue en Heroku de nuestro proyecto en particular (Censo). Es decir, contar con el sistema corriendo en Heroku de manera aislada.

c) Antes de la entrega final, cuando se tenga la integración completa de todos los módulos:
El despliegue de Heroku en cuanto al repositorio del equipo completo de Decide-Articuno también se ha llevado a cabo de forma común a todos los módulos. De forma que la rama Master del repositorio completo de Decide-Articuno también se encuentra desplegada en Heroku.

PROCESO PARA LAS ENTREGAS

Cada componente del equipo se encargará individualmente de subir a la rama Articuno-Census las implementaciones y pruebas correspondientes con sus tareas (siguiendo el proceso descrito anteriormente).

Cada dos semanas, si existe algún incremento en esta nuestra rama, se subirán en el día correspondiente a la rama master que corresponde al proyecto de Decide-Articuno.

A medida que se va implementando las funcionalidades, se va rellenando el documento plantilla que creamos con la información necesaria del proyecto.

El proceso para llevar a cabo una entrega del proyecto se define en los siguientes pasos:

1. Cuando hay una entrega, se requiere que estén las funcionalidades listas y en las ramas correspondientes.
2. Se comprueba mediante CODACY que se cumplen las validaciones establecidas sobre características del código. Si no es así, se procede a corregir las partes necesarias para que cumplan dichas validaciones.
3. Se procede a hacer el despliegue en Heroku de nuestra rama (nuestro proyecto).
4. Se procede a hacer el despliegue en Heroku del proyecto con la integración de los otros módulos.
5. Se entregará un documento Word, que estará disponible en la wiki de la asignatura, en el cuál se incluye la URL del repositorio usado para llevar a cabo la gestión del proyecto y la URL donde está desplegada la aplicación con los incrementos e integraciones realizados.

POLÍTICA DE IDENTIFICACIÓN DE LOS ENTREGABLES

Cada entregable que se haga será subido y publicado en la wiki de la asignatura. En la wiki se ubicará dentro del enlace de proyecto.

Se identificará con un título que indique el nombre del proyecto completo, y seguidamente se dividirá entre los distintos módulos a lo componen. Dentro de esta división se encontrará subido el documento Word donde estará toda la información y los enlaces necesarios para acceder al proyecto.

El título de este documento será EX-PROYECTO-MÓDULO; donde X = número de la entrega, PROYECTO = nombre del proyecto completo, y MÓDULO = módulo correspondiente.

Quedaría de la forma:

Decide – Articuno

Cabina

Censo

E1 - Decide- Articuno – Censo.docx

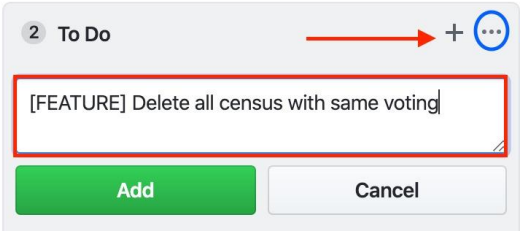
Posprocesado

10. Ejercicio de propuesta de cambio

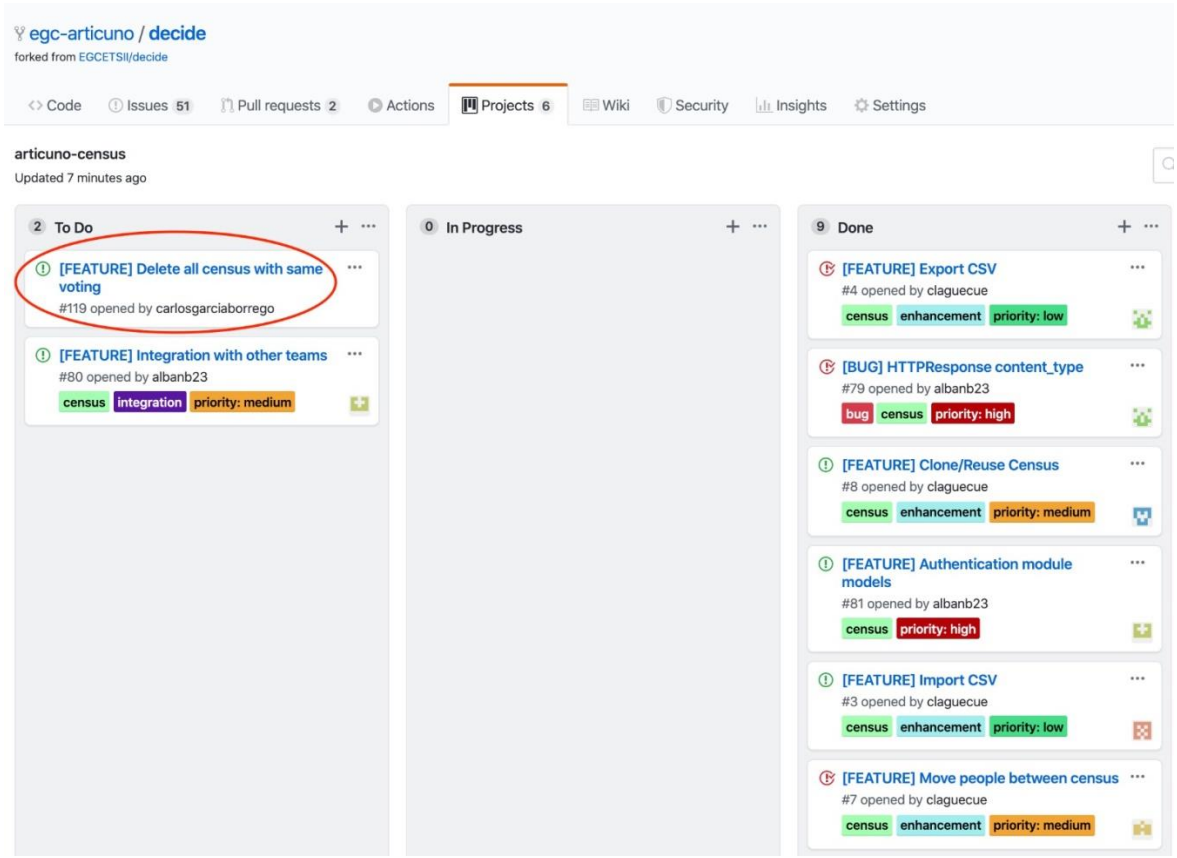
El ejercicio propuesto que vamos a realizar es: **“Eliminar todos los censos que tengan el mismo voting”**.

Lo primero que debemos hacer es irnos a nuestro proyecto en github ‘egc-articuno/decide’, luego nos vamos a la pestaña ‘Projects’ y seleccionamos ‘articuno-census’.

Una vez allí, creamos una nota haciendo click en ‘+’ (flecha roja) y escribimos la tarea.
Al tratarse de una nueva tarea ponemos [FEATURE] junto con el nombre de la tarea que vamos a realizar, luego le damos a ‘Add’.



Por último, la convertimos en una issue haciendo click en el círculo azul.

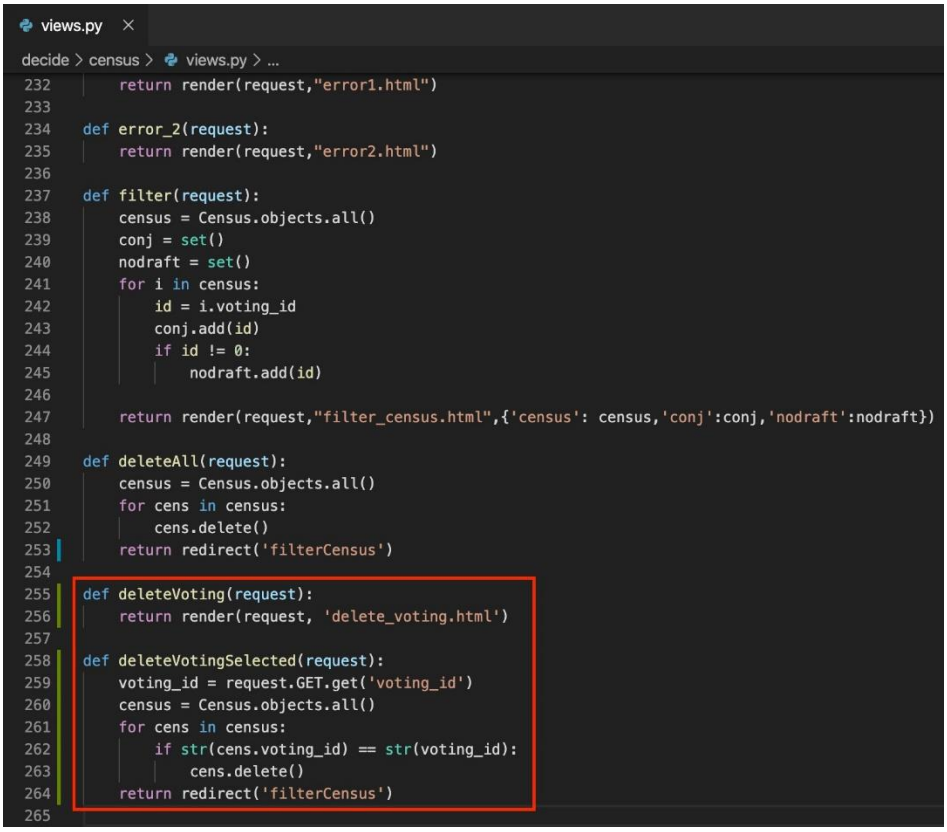


Ahora nos vamos a nuestro entorno de desarrollo, en nuestro caso Visual Studio.

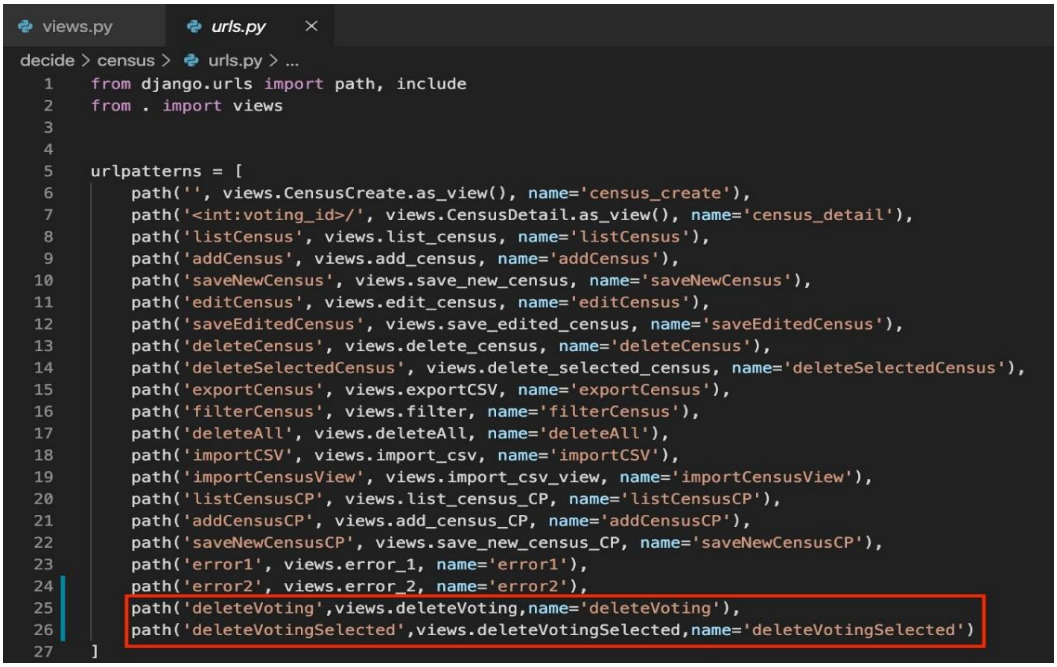
Primero accedemos al archivo views.py en la ruta decide/census/ y creamos 2 métodos que los llamaremos ‘deleteVoting’ y ‘deleteVotingSelected’.

El primero nos mandará a un archivo html que creamos posteriormente. Y el segundo método realizará la función de borrar todos los censos que tengan como voting_id el número que le pasemos.

El código nos quedará de la siguiente manera:



A continuación, nos vamos al archivo urls.py con la ruta decide/census/ y añadimos las siguientes 2 líneas de código:



Después, nos vamos a la carpeta templates y creamos un archivo html. Vamos a llamarlo ‘delete_voting.html’ como hemos anteriormente en el método ‘deleteVotingSelected’ del archivo views.py y añadimos el código para que quede de la siguiente manera:

delete_voting.html

decide > census > templates > delete_voting.html > html > form.ui.form.container

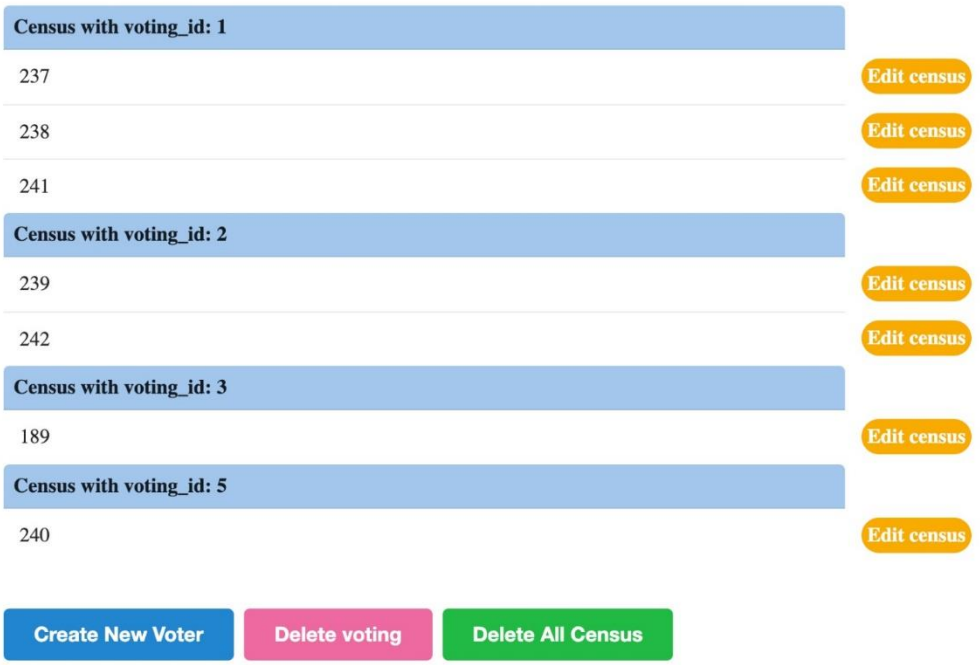
```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" type="text/css" href="https://cdnjs.cloudflare.com/ajax/libs/semantic-ui/2.4.1/components/button.css">
        <link rel="stylesheet" type="text/css" href="https://cdnjs.cloudflare.com/ajax/libs/semantic-ui/2.4.1/components/divider.css">
        <link rel="stylesheet" type="text/css" href="https://cdnjs.cloudflare.com/ajax/libs/semantic-ui/2.4.1/components/icon.css">
        <link rel="stylesheet" type="text/css" href="https://cdnjs.cloudflare.com/ajax/libs/semantic-ui/2.4.1/components/breadcrumb.css">
        <link rel="stylesheet" type="text/css" href="https://cdnjs.cloudflare.com/ajax/libs/semantic-ui/2.4.1/components/form.css">
        <link rel="stylesheet" type="text/css" href="https://cdnjs.cloudflare.com/ajax/libs/semantic-ui/2.4.1/components/header.css">
</head>

<nav>
  <div class="ui breadcrumb">
    <a href="/" class="section">Home</a>
    <i class="right angle icon divider"></i>
    <a href="{% url 'filterCensus' %}" class="section">Census</a>
    <i class="right angle icon divider"></i>
    <a href="" class="active section">Delete voting</a>
  </div>
</nav>

<h3 class="ui center aligned header">Delete  voting</h3>
  <form action="{% url 'deleteVotingSelected' %}" method="get" class="ui form container">
    <div class="field">
        <h4>Voting ID to delete</h4>
            <input type="number" name="voting_id" style="width:80%"/>
    </div>
    </br>
    </br>
    <button class="ui primary button" type="submit">Save</button>
    <a class="ui button" href="{% url 'filterCensus' %}" >Back</a>
  </form>
```

Por último, nos vamos al archivo filter_census.html y le añadimos la siguiente línea de código para poder acceder a la vista que acabamos de crear.

```
<a href="{% url 'deleteVoting' %}" class="positive ui button" style="display: inline-block; background-color: rgba(236, 82, 146, 0.877);">Delete voting</a>
```

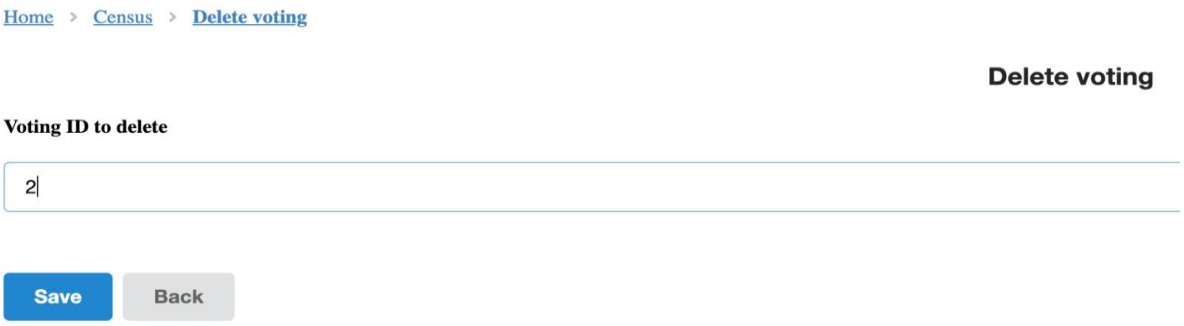
Ahora podemos probar que la tarea funciona correctamente.

Accedemos a la url: localhost:8000



/census/filterCensus y veremos la siguiente vista.

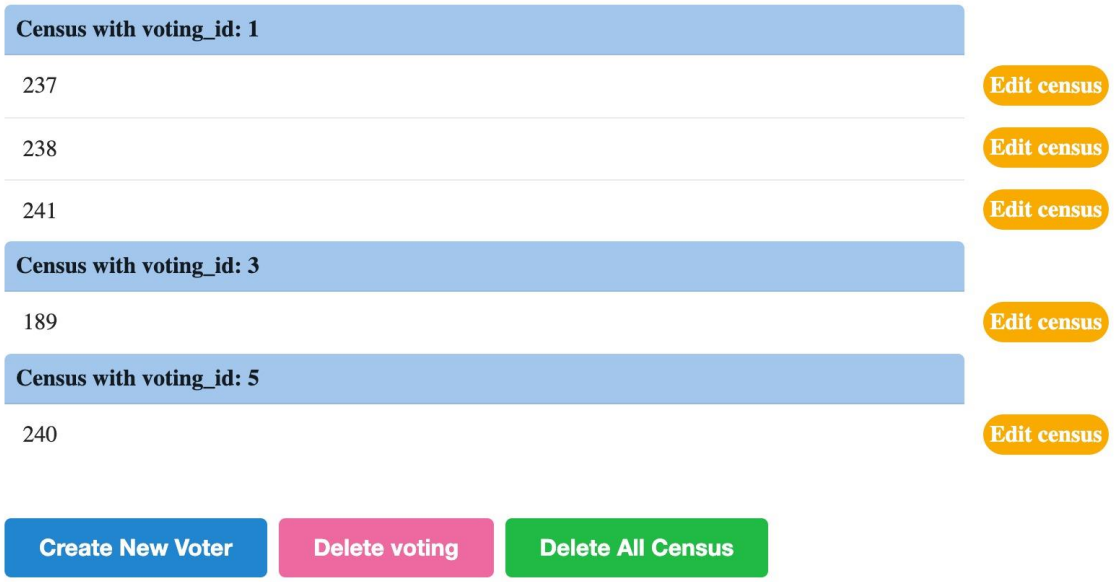
Pulsamos en el botón 'Delete voting' y nos llevará a la vista creada anteriormente.



Insertamos un voting y le damos a 'Save'.

Nota: Si insertemos un voting_id que estuviera en la vista anterior borrará todos los censos con ese voting_id. En el caso de que no existiera ninguno en la vista anterior, no hará nada el método.

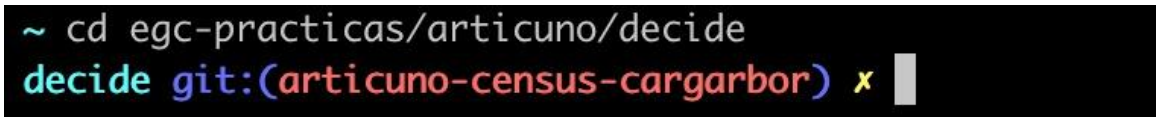
Veamos cómo nos queda la vista tras realizar la acción:



Vemos como hemos borrado todos los censos que tenían como voting_id=2.

Ahora tendremos que subir los cambios a nuestra rama en github, por tanto, nos vamos a nuestra consola de comandos.

Desde la consola nos posicionamos en la carpeta que tengamos guardado el archivo, en mi caso:



Podemos hacer un **git status**, para ver los archivos que se han creado o modificado.

Luego hacemos **git add 'rutafichero'** o '**git add .'** en el caso de que queramos subirlos todos.

Ahora hacemos **'git commit'** y nos mandará a un editor en el que vendrá una estructura del contenido que debemos poner en el commit, esta estructura es la que usamos en el equipo articuno, para subir archivos a github.

```
#<type>(<module>): <subject>
#
#<body>
#
#<footer> -> #Refenrenced_Issue
#
#       Types:
#
#       -feat (new feature)
#       -fix (bug fix)
#       -docs (changes to documentation)
#       -style (formatting, missing semi colons, etc; no code change)
#       -refactor (refactoring production code)
#       -test (adding missing tests, refactoring tests; no production -code change)
#       -chore (updating grunt tasks etc; no production code change)
```

Para poder escribir pulsamos en la tecla **'i'** para que nos permita insertar datos, hacemos los cambios necesarios, quedando de la siguiente manera:

```
<feat>(census): Delete all census with same voting

Added the method which we can delete all census with same voting

<footer> -> 119
#
#       Types:
#
#       -feat (new feature)
#       -fix (bug fix)
#       -docs (changes to documentation)
#       -style (formatting, missing semi colons, etc; no code change)
#       -refactor (refactoring production code)
#       -test (adding missing tests, refactoring tests; no production -code change)
#       -chore (updating grunt tasks etc; no production code change)
```

El cuadrado amarillo lo ponemos para indicar que es una nueva feature lo que vamos a subir, la línea naranja indica el subgrupo al que pertenece el commit, luego añadimos un título y un cuerpo para indicar que hace lo que vamos a subir y por último indicamos a que issue corresponde, si observamos la imagen la issue que creamos en github, vemos que su enumeración es 119.

Tras finalizar el commit, pulsamos en la tecla **'Esc'** y escribimos **':wq'** para salir del editor.

Ahora solo nos falta subirlo a github.

```
decide git:(articuno-census-cargarbor) ✗ git push origin articuno-census-cargarbor
```

Con esto, tendríamos nuestro cambio realizado correctamente y subido al repositorio, vamos adelante deberemos hacer pull request desde Github a la rama articuno-census para unirlo con el resto de código de nuestro grupo.

11. Conclusiones y trabajo futuro (lecciones aprendidas)

LECCIONES APRENDIDAS

A) Lenguaje de programación

El lenguaje de programación ha sido uno de los conocimientos que se ha adquirido por alguno de los miembros del equipo que desconocían las características que este contiene sobre Django, otros lo han usado con anterioridad (minoría).

En primer lugar, la mayoría han tenido que estudiarlo desde el inicio para entender lo que contenía y hacía el código ya existente.
En segundo lugar, se ha estudiado nuevas características de este para poder aumentar las funcionalidades haciendo código nuevo creado por nosotros.

Por tanto, podemos decir que el lenguaje de programación era algo bastante desconocido para el equipo y hemos aprendido su contenido.

B) Tecnología de Git

En cuanto a la tecnología que hemos elegido, que ha sido hacer uso de las ramas de Git, teniendo el proyecto entero en un repositorio de GitHub, y haciendo uso de los comandos de Git para las liberaciones; podemos decir que era algo bastante conocido por todos nosotros.
No obstante, se han aprendido posibles acciones nuevas que se pueden hacer por consola usando Git que se desconocía. Estas nuevas acciones son las vistas en prácticas de laboratorio.

C) Tecnologías de despliegue

En clase de laboratorio se ha conseguido desplegar la aplicación y mantenerla en funcionamiento de distintas formas. Una de ellas ha sido mediante Heroku, algo que era desconocido pero a través de las prácticas se ha conseguido el conocimiento necesario para desplegar en Heroku.

D) Validaciones en Codacy

Se ha adquirido una medida buena para que el código que se produzca sea código bueno. Esta medida es el establecimiento de una serie de validaciones y porcentajes que debe cumplir el código y que pueden ser medidas a través de Codacy, el cual se conecta con nuestro repositorio.

E) Importancia de las pruebas y su tratamiento

Hemos aprendido la importancia que tiene realizar distintos tipos de pruebas, comprobando lo que conlleva la no realización de estas.
También se ha aprendido a llevar a cabo un proceso por el que se debe pasar si se encuentra algún fallo en el código.

F) Trabajo en equipos numerosos

La coordinación y la comunicación frecuente son cosas que se vuelven primordiales cuando se trabaja con grupos muy numerosos. Es algo que hemos adquirido a medida que pasaba el curso, junto con la importancia que tiene establecer una serie de características comunes en algunos rasgos del proyecto.

MEJORAS PARA EL FUTURO

- Tener una mejor visión sobre cuál va a ser el trabajo que hay que hacer sobre el proyecto ya existente algo antes de tiempo.
- Usar un lenguaje de programación ya conocido.
- Usar un proyecto, para trabajar en él, que resulte familiar por ser visto con anterioridad; para que no ocurra el no conocer absolutamente nada del código recibido y que se pueda perder demasiado tiempo en aprenderlo.
- En las prácticas se podría enseñar algo del lenguaje de programación a usar.