DOCUMENTO DEL PROYECTO

decide-single-mulhacén-2

Grupo 2

Curso escolar: 2022/2023

Asignatura: Evolución y Gestión de la Configuración

Miembro	Implicación (*)
Ballestero López, Jesús	10
Bernal Martín, Ángela	10
Cabra Morón, Manuel	10
Iglesias Martín, Mercedes	10
Martín Sánchez, Paola	10
Mena Vargas, Juan Antonio	10

(*)implicación comprendida entre el 0 y el 10, siendo el 0 la nota más baja y el 10 la más alta

Enlaces de interés:

- Repositorio de código: https://github.com/jbl0107/decide-single-mulhacen-2.git
- Sistema desplegado: https://decide-single-mulhacen-2.azurewebsites.net/
  Usuario y contraseña de superuser:
  Usuario: admin
  Contraseña: admin

Índice

Control de versiones

Versión	Realizada por	Fecha	Descripción
V0.0	Jesús Ballestero López, Ángela Bernal Martín, Manuel Cabra Morón, Mercedes Iglesias Martín, Paola Martín Sánchez, Juan Antonio Mena Vargas	10/10/2022	Creación del documento
V1.1	Ángela Bernal Martín	22/10/2022	Realización de apartados
V1.2	Manuel Cabra Morón	02/12/2022	Realización de apartados
V1.3	Mercedes Iglesias Martín	02/12/2022	Realización de apartados
V1.4	Juan Antonio Mena Vargas	09/12/2022	Realización del apartado implementación de visualizaciones para Telegram
V1.5	Juan Antonio Mena Vargas, Ángela Bernal Martín	14/12/2022	Realización de apartados
V1.6	Mercedes Iglesias Martín	17/12/2022	Realización del apartado *Entorno de desarrollo*
V1.7	Jesús Ballestero López, Ángela Bernal Martín, Manuel Cabra Morón, Mercedes Iglesias Martín, Paola Martín Sánchez, Juan Antonio Mena Vargas	18/12/2022	Finalización de apartados y retoque de estilos

Observaciones

No hemos realizado pruebas unitarias para todos los incrementos debido a que no a todos los incrementos que hemos realizado tiene sentido hacerles pruebas unitarias.
Por ejemplo, para el subsistema de traducciones hemos realizado pruebas de vistas; para el bot de telegram e inicios de sesión de Google y Twitter no es necesario hacerle pruebas unitarias a las APIs porque ya están probadas, además de que, aunque es posible hacerlo, requeriría simular telegram y google, cosa que sería demasiado complicada y engorrosa (sin ser además el objetivo de la asignatura), y las gráficas correspondientes a visualizaciones de resultados de votaciones no tienen pruebas.
Por lo tanto, en cuanto a pruebas unitarias se las hemos realizado al incremento funcional de registro e inicio de sesión implementado.

Por otro lado, comentar que aunque todos los miembros han realizado pruebas, algunos como por ejemplo Juan Antonio, Jesús y Paola se han dedicado más a otras tareas, como por ejemplo hacer que la aplicación funcione en docker, o hacer mayor cantidad de documentación, dividendo así el trabajo necesario para poder completar el proyecto de forma satisfactoria.

Importante:

- Para poder usar las funciones de Google y Twitter, debe ejecutar el servidor en localhost:8000 o localhost:8080

- Para comprobar el correcto funcionamiento de todos los tests implementados, concretamente los tests de la clase *visualizer* con nombre “test_funcionamiento_pagina_visualizer”, “test_english_translation”, “test_spanish_translation” y “test_german_translation” se debe tener al menos una votación cerrada (hecho el tally), y el id de esa votación que esté entre 1 y 10.

- Para que todos los test funcionen, el servidor debe estar ejecutándose.

Indicadores del proyecto

Miembro del equipo	Horas	Commits	LoC	Test	Issues	Incremento

Jesús Ballestero López	62	18	9365	3	23	• **Traducciones:**
						  ○ Hacer la interfaz traducible
						  ○ Traducir la interfaz al español
						  ○ Traducir la interfaz a otros idiomas (alemán)

Ángela Bernal Martín	62	25	705	6	33	• **Traducciones:**
						  ○ Hacer la interfaz traducible
						  ○ Traducir la interfaz al español
						  ○ Traducir la interfaz a otros idiomas (alemán)
						• **Visualización de resultados:**
						  ○ Pintado de gráficas y estudio de datos

Manuel Cabra Morón	60	20	2265	6	15	• **Traducciones:**
						  ○ Hacer la interfaz traducible
						  ○ Traducir la interfaz al español
						  ○ Traducir la interfaz a otros idiomas (alemán)
						• **Autenticación:**
						  ○ Autenticación con redes sociales: Google
						  ○ Autenticación con redes sociales: Twitter
						  ○ Registro de usuarios. Formulario de registro que crea nuevos usuarios

Mercedes Iglesias Martín	62	25	1118	8	30	● **Traducciones:** ○ Hacer la interfaz traducible ○ Traducir la interfaz al español ○ Traducir la interfaz a otros idiomas (alemán) ● **Visualización de resultados:** ○ Pintado de gráficas y estudio de datos
Paola Martín Sánchez	63	24	1432	5	23	● **Traducciones:** ○ Hacer la interfaz traducible ○ Traducir la interfaz al español ○ Traducir la interfaz a otros idiomas (alemán)
Juan Antonio Mena Vargas	62	12	544	3	27	● **Traducciones:** ○ Hacer la interfaz traducible ○ Traducir la interfaz al español ○ Traducir la interfaz a otros idiomas (alemán) ● **Visualización de resultados:** ○ Pintado de gráficas y estudio de datos ○ Implementar visualizaciones para diferentes plataformas: telegram

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

- **Horas:** número de horas empleadas en el proyecto.
- **Commits:** solo contar los commits hechos por miembros del equipo, no los 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.

Integración con otros equipos

No nos hemos integrado con ningún otro equipo, somos un equipo de tipo *single* compuesto por seis integrantes.

Resumen ejecutivo

En nuestro caso, decidimos añadir incrementos al proyecto como el de traducciones a distintos idiomas, del inglés al español y al alemán, con respecto a la autenticación hemos implementado la autenticación con redes sociales (Google y Twitter), el registro de usuarios mediante un formulario de registro que crea nuevos usuarios, la implementación de un bot de Telegram y un pintado de gráficas y estudio de datos **(8 incrementos en total).**

En primer lugar, en cuanto a la implementación de *traducciones*, nos dividimos el trabajo de forma equitativa para todos los miembros del grupo, implementando los tres incrementos de traducciones propuestos en la Wiki (Hacer la interfaz traducible, traducir la interfaz al español, traducir la interfaz a otros idiomas (alemán)).
Hicimos la traducción de las páginas ya implementadas “visualizer.html” y “booth.html”, pero además, creamos varias páginas más relacionadas con la asignatura con el lenguaje de marcado HTML y CSS, sobre las jornadas Innosoft, el proyecto educativo decide, sobre la ETSII y por último sobre el subsistema de traducciones. Como bien hemos escrito antes, hicimos las traducciones de esas páginas a los idiomas descritos anteriormente (español y alemán).

En segundo lugar, para la *autenticación con redes sociales* hemos implementado la autenticación con *Google*, para ello se ha tenido que crear un proyecto en la consola para desarrolladores de Google autorizando a decide a poder usar la API de autenticación de Google, Google genera un par de claves pública y privada que se introducen en el settings.py del proyecto para confirmar a Google que en efectivo es el proyecto decide el que envía la petición de inicio de sesión. En el booth.html se ha añadido un botón para Iniciar sesión con Google lo que una vez iniciada sesión te devuelve a la votación en la que estaba el usuario. Se ha creado una función de autenticación que le pide a google los siguientes datos sobre el usuario: Email y Perfil del cual extraemos email, nombre y apellido del usuario en cuestión para guardarlo en la base de datos como un nuevo user si no existía o se actualiza el user en caso de que existiese ya y hubiese que actualizar alguno de sus datos y a continuación se inicia la sesión con el nuevo usuario.

En tercer lugar, para la *autenticación con redes sociales* hemos implementado la autenticación con *Twitter* para la cual ha habido que crear un proyecto en Twitter autorizando a las direcciones en las cuales la aplicación ha sido desplegada. Twitter genera dos pares de claves públicas y privadas, que se usarán posteriormente para autorizar las peticiones a la API de Twitter. Se usa el 3-legged-authentication de Twitter usando OAuth1. Se empieza con un POST a la API de Twitter que devuelve unos tokens para poder redirigir al usuario a la pantalla de autorización de Twitter y una vez se autoriza Twitter redirige a la votación en la que se encontraba el usuario

En cuarto lugar, para el *registro de nuevos usuarios* se ha creado una vista de *registro* "register.html" a la cual se le pasa un parámetro next para devolver al usuario a la vista de la que venía. Se ha creado una función de registro que usa el formulario predefinido por Django para la creación de usuarios.

En quinto lugar, para el *pintado de gráficas* dentro de la carpeta visualizer, en el archivo "views.py" se coge una votación ya cerrada, y con un for se recorre esa votación para obtener las opciones y los votos resultantes de esa votación. Con esto, se rellenan dos listas, una de las opciones y otra de los votos. Por último creamos las gráficas con plt pasándole como parámetro estas dos listas obtenidas.

En sexto lugar, para las *visualizaciones en la aplicación de Telegram*, dentro de la carpeta visualizer, en el archivo "views.py", dentro de la función por defecto del proyecto base se le pasa como variables un *bot_token* y un chat_id previamente obtenidos con la creación del bot en la propia aplicación de Telegram. Con esto, y utilizando el pintado de gráficas le pasamos al bot como parámetro el *chat_id* para definir al bot a que chat mandar los resultados, las gráficas generadas y un mensaje.

Descripción del sistema

Traducciones

En cuanto al subsistema de traducciones hemos realizado el bloque entero de traducciones definido en la Wiki, que son hacer la interfaz traducible, traducir la interfaz al español, traducir la interfaz a otros idiomas (en nuestro caso a alemán).
El sistema consiste en un sistema de funcionalidad de traducción del inglés al español y al alemán.

Para realizar el subsistema de traducciones, hemos instalado el paquete `gettext` con `sudo apt-get install gettext`.

Por lo tanto, hemos seguido los siguientes pasos para realizar las traducciones a los idiomas indicados:

- Configuraciones iniciales

  Nuestro proyecto llamado `decide-single-mulhacen-2` es un proyecto django son una aplicación llamada `decide`.

  El primer paso, es asegurarnos de que tenemos activadas las traducciones en nuestra configuración. Para hacer esto, debemos hacer los siguientes cambios en `decide/settings.py`:

  ```
  #decide/settings.py

  LANGUAGE_CODE = 'en'
  TIME_ZONE = 'UTC'

  USE_I18N = True
  USE_L10N = True
  USE_TZ = True
  ```

  Para este caso hemos definido que nuestro proyecto se encuentra en inglés por defecto.

- Marcar lo que vamos a traducir

  El siguiente paso es marcar los textos que queremos traducir dentro de nuestro proyecto, para esto existen diferentes formas dependiendo del texto que vayamos a traducir.

- Traducir en templates

  Para explicar cómo traducir templates, en cualquier html de nuestro proyecto, habría que añadir las sentencias {%trans 'Texto que queremos traducir'% }, y además es necesario que en cada html que queramos traducir hay que añadir {% load i18n %} al principio del html

  Por tanto como resumen, habría que importar en los templatetags de i18n el `{% load i18n %}` y que para traducir una línea sencilla utilizamos el templatetag `trans`.

- Crear archivos de traducción

El siguiente paso es crear los archivos de traducción para cada uno de los lenguajes que queremos traducir, para esto agregaremos unas cuantas líneas más a la configuración `decide/settings.py`:

```
# some_project/settings.py

from django.utils.translation import ugettext_lazy as _

...

MIDDLEWARE_CLASSES = (
    ...,
    'django.middleware.locale.LocaleMiddleware',
    ...,
)

...

LANGUAGES = [
    ('en', ('English')),
    ('es', ('Spanish')),
    ('de', ('German'))
]

LOCALE_PATHS = [
    os.path.join(BASE_DIR, 'locale')
]
…
```

En este caso le estamos indicando a Django que utilice el `LocaleMiddleware` el cual se encargará de escoger el lenguaje adecuado para cada usuario basado en información extraída del request (Es importante ubicar este middleware entre `SessionMiddleware` y `CommonMiddleware` si estás usando ambos, de lo contrario podría no funcionar bien). También especificamos mediante la variable `LANGUAGES` qué lenguajes va a soportar nuestro proyecto y por último, especificamos la ruta de la carpeta en la cual se guardarán los archivos de traducción mediante la variable `LOCALE_PATHS`. Vale la pena aclarar que debemos crear dicha carpeta `decide/locale`.

Teniendo todo configurado, ahora sí, procedemos a crear los archivos de traducción. Para esto utilizamos el siguiente comando desde la ruta del proyecto:

```
django-admin makemessages -l es
django-admin makemessages -l en
```

```
django-admin makemessages -l de
```

Esto creará los archivos de traducción para el lenguaje español y alemán que son los que utilizaremos para nuestro trabajo, es importante saber que los lenguajes que le pasemos como parámetro a este comendo deben seguir la notación de locale name, de lo contrario, Django no reconocerá las traducciones.

- Traducir archivos .po

Al crear los archivos de traducción obtuvimos algo como esto:

```
#: booth/templates/booth/booth.html:55
msgid "Go"
msgstr ""

#: booth/templates/booth/booth.html:63
msgid "Username"
msgstr ""

#: booth/templates/booth/booth.html:71
msgid "Password"
msgstr ""

#: booth/templates/booth/booth.html:86
msgid "Login"
msgstr ""
```

Estos archivos son los que utilizaremos para especificarle a Django cómo traducir cada palabra en determinado idioma, entonces simplemente para cada uno de los strings que aparezcan en `msgid` podemos especificar su traducción con los strings `msgstr`.

```
#: booth/templates/booth/booth.html:55
msgid "Go"
msgstr "Vamos!"

#: booth/templates/booth/booth.html:63
msgid "Username"
msgstr "Usuario"

#: booth/templates/booth/booth.html:71
msgid "Contraseña"
msgstr ""

#: booth/templates/booth/booth.html:86
msgid "Login"
```

```
msgstr "Inicia sesión"
```

- Compilar archivos de traducción

  Una vez tengamos nuestros archivos de traducción completos, procederemos a compilarlos con el siguiente comando:

  ```
  django-admin compilemessages
  ```

  Cuando compilamos nuestros archivos, ya podemos utilizar nuestro proyecto en el idioma especificado. Cada vez que uno de nuestros usuarios consulte nuestro proyecto aparecerá en su idioma.

  Es importante ejecutar el comando anteriormente descrito para poder ejecutar los tests de traducciones (también tiene que estar el servidor ejecutándose al ser pruebas de vistas).

Autenticación con Google

Para adaptar el nuevo sistema de registro e inicio de sesión con Google se ha tenido que modificar los módulos de Booth y Vote, en los cuales se ha tenido que cambiar la forma en la cual se verifica el usuario que ha iniciado sesión para adaptarlo a usar funciones nativas a Django como “user.is_authenticated” para verificar si el usuario ha iniciado sesión y así modificar el booth.html para saber si el usuario puede ver o no la votación, para poder votar se tiene que ser añadiendo el usuario al census por un administrador, tanto si se registra como inicia sesión con Google, debe ser autorizado previamente por un administrador.

Autenticación con Twitter

Para implementar el sistema de inicio de sesión con Twitter, se ha tenido que añadir las claves al setting.py, los cambios aplicados mediante la autenticación con Google hacen que Twitter funcione de la misma forma. Los usuarios creados mediante Twitter tienen un nombre de usuario especial siendo @ + Su apodo en Twitter. Para poder votar los usuarios han de ser autorizados previamente por un admin.

Registro de usuarios

Para admitir el registro de usuario ha habido que mover el inicio de sesión de la página de votación a una nueva ventana, que al terminar de registrarse o iniciar sesión te redirige a la

última página en la que estaba el usuario. Para poder votar el usuario registrado ha de ser autorizado por un admin primero.

Pintado de gráfica y estudio de datos

Para implementar esta tarea del subsistema de Visualización de resultados, dentro de la carpeta visualizer, en el archivo views.py modificamos la función que viene por defecto del proyecto base añadiendo un bloque try en el que en primer lugar en una variable dicc almacenamos un diccionario pasándole la votación con su id, se vería tal que así:

[{'id': 8, 'name': 'Serie favorita', 'desc': 'Serie favorita', 'question': {'desc': 'Cual es tu serie favorita', 'options': [{'number': 1, 'option': 'Breaking Bad'}, {'number': 2, 'option': 'Juego de Tronos'}, {'number': 3, 'option': '1899'}]}, 'start_date': '2022-12-08T22:27:23.010270Z', 'end_date': '2022-12-08T22:30:38.714016Z', 'pub_key': {'p': 6975082029817921656039808389931166574465896725813514914726858004799371701073 9, 'g': 2630144380998311997309909671712206314564147478413380462625900342006494401221 2, 'y': 4449011851158449801443728663302293343933624302258575289747519411480219683230 2}, 'auths': [{'name': 'localhost', 'url': 'http://localhost:8000', 'me': False}], 'tally': [2, 3, 1], 'postproc': [{'votes': 1, 'number': 1, 'option': 'Breaking Bad', 'postproc': 1}, {'votes': 1, 'number': 2, 'option': 'Juego de Tronos', 'postproc': 1}, {'votes': 1, 'number': 3, 'option': '1899', 'postproc': 1}]}]

Con esto, creamos dos listas vacías llamadas opciones y votos. Utilizando un bucle for, recorremos este diccionario, añadiendo con .append a cada lista el valor del diccionario [“option”] y [“votes”] respectivamente. Finalmente, con estas dos listas ya podemos generar las gráficas. En nuestro caso, creamos una gráfica de barras horizontales pasándole como parámetro las dos listas, opciones y votos.

```
plt.barh(opciones,votos)
```

Y creamos una gráfica de pastel pasándole como parámetro las dos listas y algunos valores más que hace que se vea más estético:

```
plt.pie(votos,labels=opciones,autopct=”%0.1f %%” , colors=colores)
```

Por último,con la aplicación en ejecución, cuando accedemos a /visualizer/’id’ siendo id el número identificativo de la votación, con dicha votación cerrada y con el Tally realizado, se genera el png de las gráficas en el proyecto fuente, pudiendo visualizar los resultados mediante las gráficas creadas.

Implementación de visualizaciones para Telegram

Para implementar esta tarea del subsistema de Visualización de Resultados primero en la aplicación de Telegram teníamos que crear un bot en el BotFather con el comando:

```
/newbot
```

Luego de esto, seleccionamos un nombre y un usuario para nuestro bot en el mismo chat de Telegram. Cuando se completó el registro, el BotFather te facilitaba un Token_id para luego poder enlazar el bot con nuestra aplicación decide.

A continuación, en una nueva pestaña del navegador colocamos:

`https://api.telegram.org/bot(yourtoken)/getUpdates`

Dónde ("yourtoken") es el Token anterior que nos facilitó el BotFather. Nos apareció un mensaje y de él teníamos que copiar el campo "id", chat_id que junto con el Token_id nos haría falta para poder enlazar el bot.

Una vez obtenidos los id, modificamos el archivo visualizer/views.py introduciendo en `def get_context_data(self, **kwargs):` las variables bot_token y chat_id entre otras, pegando en ellas los números copiados anteriormente.

Luego, lo que hicimos fue recorrer la votación para obtener los votos y las opciones de una votación determinada con su id y una vez obtenidos, creamos dos tipos de gráficas simples, una de barras horizontales y otra gráfica de pastel que tenían como parámetro los votos y opciones de esa votación.

Por último, declaramos mediante:

`bot.sendPhoto`

pasándole por parámetros el chat_id de nuestro bot, la photo png de las gráficas generadas y un mensaje. Cuando se cierra una votación y se hace el Tally para el recuento de votos, cuando ingresamos en visualizer/{id} el id de la votación en cuestión, el bot emite dos gráficas en tu chat de Telegram, con el recuento de votos.

Visión global del proceso de desarrollo

Con respecto a nuestro proyecto, para poder llevar a cabo un análisis continuo del código y del estado de las pruebas, hemos hecho uso de la herramienta Codacy, que previamente vinculamos a nuestro proyecto en el repositorio de GitHub y que analiza cada una de las ramas cada vez que se realiza una pull-request.

Para el seguimiento de las horas invertidas en el trabajo hemos utilizado la herramienta Clockify, ya que es muy intuitiva y además es gratuita. Esta herramienta funciona como un

contador de tiempo en el que podemos obtener gráficas para visualizar el tiempo invertido por tarea o miembro del equipo.

Es un buen apoyo para mostrar un registro del rendimiento realizado por cada persona individual y saber si el tiempo que le hemos dedicado al trabajo cumple con los tiempos esperados.

Para comunicarnos entre nosotros hemos utilizado principalmente el canal de comunicación de Discord y también reuniones de manera presencial, de las reuniones formales hemos redactado actas de reuniones.

Además, hemos usado la aplicación WhatsApp para coordinar las fechas de las reuniones, ya fueran a través de Discord o presenciales, y también para comentar pequeños cambios realizados o cualquier tipo de información que considerábamos importante.
Para llevar a cabo la documentación necesaria, utilizamos la herramienta Google Drive, que nos permite subir archivos a la carpeta compartida del equipo y simultáneamente modificar un mismo archivo de forma sincrónica, aumentando la productividad del equipo y evitando posibles errores y superposiciones de diferentes versiones de archivos. En esta plataforma, cargamos la plantilla del documento, en la que escribimos todas las diferentes secciones, que se distribuyen por igual entre los miembros del grupo.

Hemos usamos Github como sistema de control de versiones para albergar el repositorio y registrar los cambios que se hacían al código. Así, tenemos un control de los commits que nos permite saber el avance de cada miembro en el proyecto y además trabajar de forma simultánea. Para organizarnos correctamente, creamos una rama por cada miembro del grupo a partir de la rama develop, siendo esta una copia de la rama original *master*. Una vez que el trabajo de todos los miembros del equipo es revisado y unificado en la rama *develop*, se comprueba que la aplicación funciona correctamente, no se localicen errores y posteriormente, se fusiona con la rama *master*.

A continuación mostraremos un ejemplo de proceso llevado a cabo para realizar uno de los incrementos funcionales:

1. Primero se crea la incidencia en GitHub, se le asignan las etiquetas correspondientes, los puntos de estimación y el milestone al que pertenece y se le asigna mínimo a un miembro del equipo, que será el que solucione la incidencia.
2. Una vez se empieza a trabajar con la incidencia, colocamos esa incidencia en la columna *In Progress* del tablero de GitHub. Una vez finalizado el desarrollo de la incidencia, la incidencia pasará a la columna *Done*. En caso de haber realizado una incidencia de tipo 'documentación', se deberá pasar a la columna Review y la revisará otro miembro del equipo al que se le haya asignado previamente el rol de revisor.

Una vez que la incidencia haya sido revisada y aprobada por dicho miembro, la tarea se deberá pasar a la columna *Done*. De haberse realizado una incidencia de tipo documentación, el proceso habrá finalizado, sin necesidad de realizar los siguientes puntos del proceso. En caso de que la incidencia afecte al código del proyecto, se realizará un commit de la rama en la que se ha trabajado, creando un pull-request y haciendo merge a la rama de desarrollo general develop. Si el merge ha sido exitoso, se pasará la incidencia a la columna *Done*.

3. Estando los cambios individuales compartidos en una misma rama de desarrollo, se comunicará al resto del equipo el cambio realizado y ya implementado en la rama de desarrollo. Con la mayor brevedad posible, todos los miembros del equipo tendrán que comprobar el funcionamiento del sistema. En caso de encontrar algún bug o incongruencia, se comunicará al equipo y se creará una incidencia de tipo bug. En el caso de que se produjera un error muy complejo, se abre una rama hotfix a raíz de la *develop* para solucionar el error, por el contrario, si el error detectado es simple, se solventará en la rama de trabajo del encargado de resolver la incidencia.
4. Se repiten los apartados 1, 2 y 3 para la realización de los tests asociados a los incrementos funcionales, en caso de que sean necesarios.

Por último, una vez implementados todos los cambios y comprobado el correcto funcionamiento en conjunto, se fusiona la rama *develop* a la rama *master* para cerrar el proyecto.

Entorno de desarrollo

En nuestro caso, para llevar a cabo este proyecto, todos los miembros del equipo han utilizado el entorno de desarrollo Visual Studio Code, con la versión 1.74.1 en sus ordenadores. También es importante destacar, que 3 miembros del equipo han ejecutado el proyecto en una máquina virtual de Virtual Box con el sistema operativo Ubuntu 22.04.1, y los otros 3 miembros han instalado el sistema operativo Ubuntu 22.04 mediante una partición de discos en sus portátiles.

- Entorno común en Django

  Una vez instalado Ubuntu, y la herramienta de Visual Studio Code para hacer cambios en el código del proyecto, es necesario tener instalado un entorno virtual en Ubuntu. Para ello necesitamos Python, que será el lenguaje de programación principal, cuya versión descargada y utilizada por todos los miembros del equipo es la 3.8.14. Para crear el entorno virtual, lo hicimos con los siguientes comandos:

  ```
  python3.8 -m venv pyEnv
  ```

Con esto tendremos un entorno virtual activado creado. Podemos activar dicho entorno situándonos dentro de este y escribiendo lo siguiente:

```
source pyEnv/bin/activate
```

Con el entorno virtual activado, a continuación, podemos instalar los requisitos necesarios para que funcione el proyecto. Nos iremos al directorio donde se encuentra el archivo "requirements.txt" y ejecutamos lo siguiente:

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

Todos los requisitos necesarios para que funcione el proyecto son los siguientes:

```
Django==2.0
pycryptodome==3.6.6
djangorestframework==3.7.7
django-cors-headers==2.1.0
requests==2.18.4
django-filter==1.1.0
psycopg2-binary==2.8.4
django-rest-swagger==2.2.0
coverage==4.5.2
django-nose==1.4.6
jsonnet==0.12.1
selenium
numpy
matplotlib
oauth2==1.9.0.post1
telegram
aspose.words
locust
django-bootstrap4
django-crispy-forms
python-telegram-bot
```

Tras la instalación de estos requisitos, será necesario tener una base de datos en local, con las credenciales definidas en el fichero local_settings.py. Podemos crear dicha base de datos con sus credenciales de la siguiente manera:

```
sudo su - postgres
psql -c "create user decide with password 'decide'"
psql -c "create database decide owner decide"
psql -c "ALTER ROLE decide CREATEDB"
psql -c "ALTER USER decide CREATEDB"
exit
```

Con esto tendremos lista nuestra base de datos. Luego, sería necesario crear un usuario para decide, podemos hacerlo situándonos en donde se encuentra manage.py y ejecutando lo siguiente:

```
python ./manage.py createsuperuser
```

Finalmente sólo nos quedaría realizar las migraciones necesarias. Para ello, ejecutamos los siguientes comandos:

```
python ./manage.py makemigrations
python ./manage.py migrate
```

Con esto tenemos ya todos los requisitos necesarios para que funcione correctamente la aplicación en el entorno virtual. Tan sólo queda lanzar la aplicación con el siguiente comando:

```
python ./manage.py runserver
```

- Despliegue en Docker

  La aplicación la podemos desplegar en Docker, que automatiza el despliegue de aplicaciones dentro de contenedores software.

  En primer lugar, necesitamos tener instalado Docker, para ello hemos seguido los siguientes pasos:

  1. Desinstalar versiones antiguas que tengamos con el comando:
     ```
     sudo apt-get remove docker docker-engine docker.io containerd runc
     ```
     .
  2. Instalamos las dependencias con los comandos:
     ```
     sudo apt-get install \
          ca-certificates \
          curl \
          gnupg \
          lsb-release
     ```
  3. Instalamos la llave de cifrado:
     ```
     sudo mkdir -p /etc/apt/keyrings
     curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
     ```
  4. Añadimos el repositorio con el comando:
     ```
     echo \
     "deb [arch=$(dpkg --print-architecture)signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
          $(lsb_release-cs)stable"|sudo tee/etc/apt/sources.list.d/docker.list > /dev/null
     ```
  5. Instalamos docker:
     ```
     sudo apt-get update
     ```

`sudo apt-get install docker-ce docker-ce-cli containerd.io`

6. Instalamos docker compose:
   `sudo apt-get install docker-compose-plugin`
7. Añadimos el usuario a docker:
   `sudo usermod -aG docker $USER`

Posteriormente a la instalación de Docker, configuramos nuestro proyecto para lanzarlo con la herramienta, para ello seguimos los siguientes pasos:

1. Accedemos a la carpeta Docker: /decide-single-mulhacen-2/docker
2. Creamos el archivo *Dockerfile,* donde quedan reflejados los pasos a seguir para configurar el despliegue. En él se definen las dependencias a instalar, se clona el repositorio y se instala los requisitos, y por último, volcamos el contenido de *local_settings.py* en el archivo *docker_settings.py*.

```
from python:3.8-alpine

RUN apk add --no-cache git postgresql-dev gcc libc-dev
RUN apk add --no-cache gcc g++ make libffi-dev python3-dev build-base
RUN apk add gettext
RUN pip install --upgrade pip

RUN pip install gunicorn
RUN pip install psycopg2
RUN pip install ipdb
RUN pip install ipython

WORKDIR /app

RUN git clone https://github.com/jbl0107/decide-single-mulhacen-2.git .
RUN pip install -r requirements.txt

WORKDIR /app/decide

# local settings.py
ADD docker-settings.py /app/decide/local_settings.py

RUN ./manage.py collectstatic
RUN django-admin compilemessages
```

3. Modificamos el contenido del archivo *docker-compose.yml* donde quedan definidos los contenedores que vamos a lanzar.
4. Para finalizar, lanzamos la aplicación con Docker con el comando: `docker compose up`.
5. Accedemos a la URL "http://localhost:8000", ya que el puerto 8000 ha sido el puerto escogido en nuestro caso.

- Despliegue en remoto:

  Para el despliegue en remoto se ha elegido Microsoft Azure como plataforma de despliegue, ya que permite el despliegue de aplicaciones multi-contenedor.

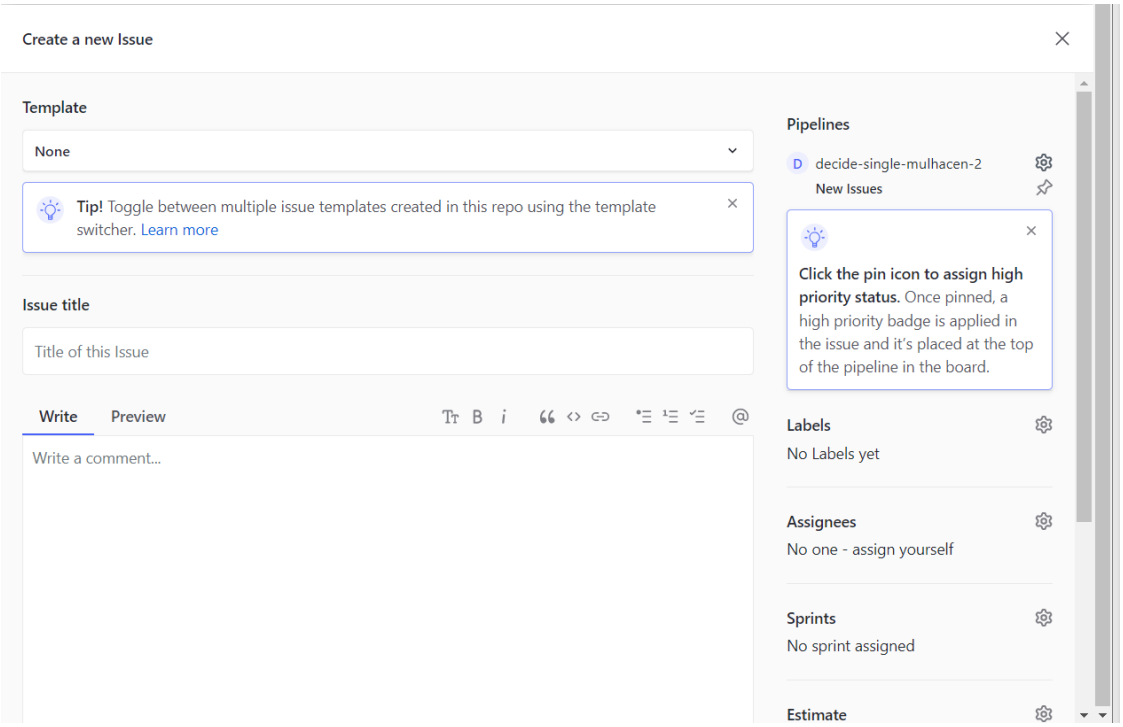
  Para poder desplegarlo ha habido primero que crear un contenedor con la aplicación, creando un *dockerfile* en el que se instala todas las dependencias necesarias para el proyecto así como un *docker-settings.py* en el que se especifica la configuración para el despliegue en Azure, y también para poder hacer run del contenedor. Además, se crea un *docker-compose.yml* donde se especifica un segundo contenedor para la base de datos en postgres y se establece en el contenedor de decide una dependencia con el contenedor de la base de datos.

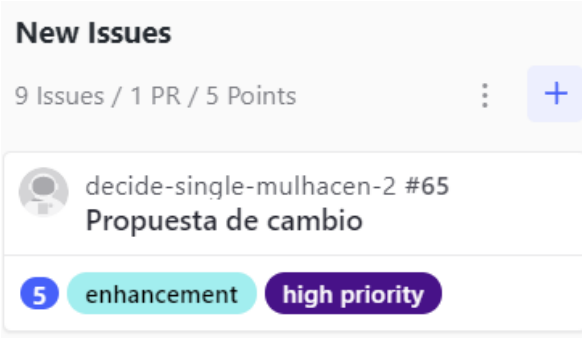
  El contenedor de la aplicación se sube a Azure para pueda descargarlo la máquina a la que se le proporciona el *docker-compose.yml* y poder así desplegar la aplicación. Para poder hacer funcionar el dominio asignado por Azure ha habido que añadir los enlaces correspondientes a los proyectos de Google y Twitter para permitir el uso de la API desde la dirección asignada por Azure.

Ejercicio de propuesta de cambio

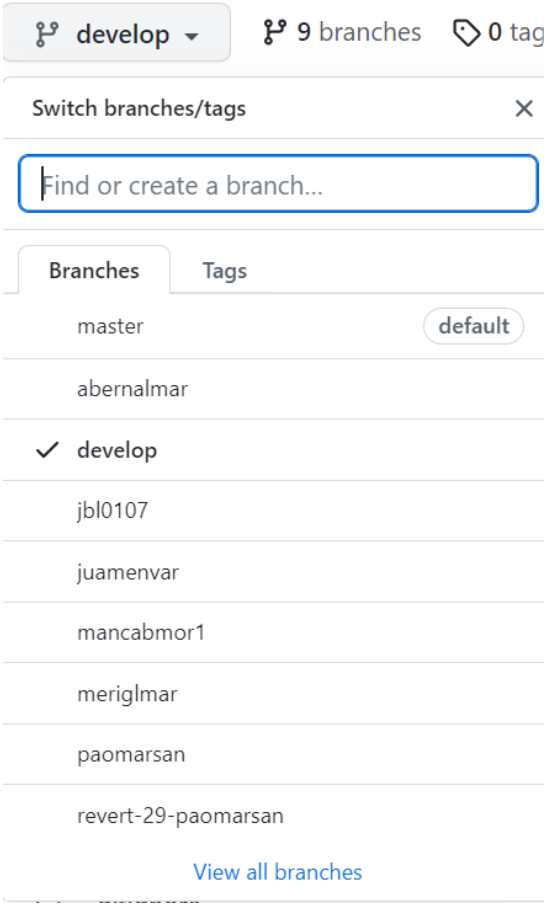
Para hacer cualquier cambio deberemos seguir los siguientes pasos:

1. El primer paso a realizar es la creación de incidencia en Zenhub. Debemos detallar en qué consiste en la propia descripción , asignar a los miembros responsables de la tarea y añadir las etiquetas pertinentes.





2. Tras esto, la/s persona/s asignada/a la incidencia creará o crearán la rama en la que trabajar a raíz de la rama develop. Se puede crear la rama desde la consola con el comando *git checkout -b nombreRama* o desde la propia página de GitHub de la siguiente manera, dándole el nombre pertinente a la rama relacionada con el cambio:



3. El siguiente paso a realizar es el desarrollo de la implementación. Para ello se trabajará en el entorno hasta que se comprueben que los cambios realizados han sido satisfactorios. Para trabajar en la rama, realizamos un pull para tenerla actualizada a la última versión del repositorio. Una vez terminada la implementación, se hará un commit y un push, para observar los cambios aplicados en el repositorio remoto. Ya podemos pasar nuestra incidencia a la columna Done.

4. Una vez subido, la persona encargada del respectivo cambio ejecutará automáticamente todos los test y deberá comprobar que los tests han pasado correctamente.

5. Ahora procedemos a fusionar nuestra rama con la rama develop mediante la función merge. Se comprueba que no hayan surgido conflictos y en el caso de que los hubiera solventamos el error.

6. Se comprueba que la persona encargada no haya detectado errores en la rama develop al actualizarla con los nuevos datos añadidos.

7. Nuestro siguiente paso es comprobar la calidad de código con Codacy. Aquí nos cercioramos de que se cumplen los valores mostrados. En el caso de no cumplir con los criterios establecidos, debemos remediar los errores y comprobar nuevamente la cobertura de código.

8. Tras esto se procede a realizar la pull-request a la rama master donde está todo el código funcional y potencialmente desplegable.

9. Por último, se comprueba que se haya desplegado en Azure adecuadamente.

10. Si todo esto se ha cumplimentado de manera correcta, se traslada la issue del cambio a la columna del tablero “Done”.

Conclusiones y trabajo futuro

Tras finalizar el proyecto podemos decir que nos ha alegrado haber elegido realizar el proyecto de tipo “single”, sin tener que llevar el trabajo en conjunto con otros grupos, ya que esto supone una mayor y difícil organización con muchas más personas.
Como grupo hemos tenido una comunicación activa y trabajo constante, en general ha ido todo de forma correcta sin muchos problemas.

También ha influido al principio del desarrollo del proyecto, el desconocimiento del funcionamiento de decide y el uso de nuevas tecnologías como herramientas.
Como punto a destacar de manera positiva es la forma correcta de trabajar que hemos tenido con Github, organizando así las issues, sus estados y responsables de gestionarlas e implementarlas, además del tablero de ZenHub creado en el propio GitHub.

Como conclusión, podemos decir que se ha requerido sobre todo al principio más tiempo para comprender, implementar y probar, pero que finalmente se ha alcanzado y llevado a cabo de forma correcta el proyecto, terminando su implementación en los plazos acordados cumpliendo con los objetivos marcados.