

Decide-Single-Majaceite



Escuela Técnica Superior de
Ingeniería Informática

Grupo 1

ID ópera: No proporcionado

Curso escolar: 2020/2021

Asignatura: Evolución y gestión de la configuración

Milestone en el que se entrega la documentación: M3

Miembros del equipo

Alejandro Barba Moreno

Alejandro Carrillo Martín

Francisco Javier García Roales

Angel Pomares Luza

Juan Pablo Portero Montaña

Fernando Wals Rodríguez

Enlaces de interés

[Repositorio de código](#)

[Sistema desplegado en heroku](#)

1.Resumen ejecutivo

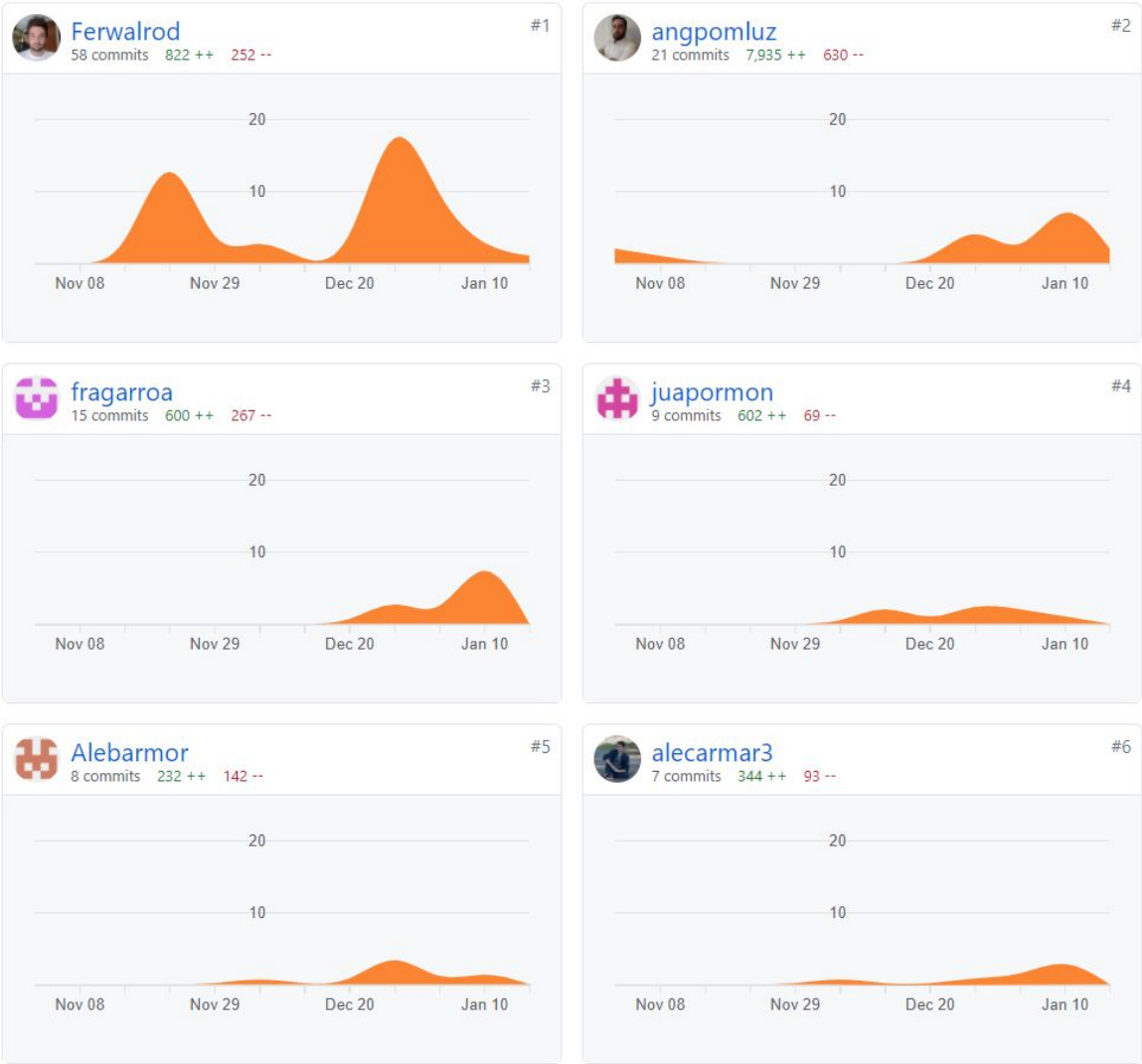
1.1 Indicadores del proyecto

Miembro del equipo	Horas	Commits	LoC	Test	Issues	Incremento
Alejandro Barba Moreno	29	8	232	6	7	Implementación de gráficas con estudio de datos, Creación de test de test para la implementación de las gráficas con estudio de datos, Testing de cobertura.
Alejandro Carrillo Martín	24	7	344	6	7	Implementación descarga de votaciones en formato PDF, Creación de test para la descarga de votaciones en formato PDF, Testing de cobertura.
Francisco Javier García Roales	53	15	600	6	8	Integración de decide con Telegram, Testing de la integración de decide con telegram, Configuración y despliegue en docker, vídeo de las funcionalidades cambiadas en la aplicación.
Angel Pomares Luza	69	21	731	6	9	Implementación de gráficas con procesamiento de información relevante de los votantes, Creación de test para el procesamiento de datos de votantes en visualizer, Mejora del visualizer creando template para las gráficas, Implementación de lector de CSV, Configuración y despliegue en docker.
Juan Pablo Portero Montaña	51	9	602	6	8	Integración de decide con slack, Testing de la integración de decide con slack, Testing de algunas funciones de utilidad desarrolladas durante el M2, Configuración y despliegue en docker.
Fernando Wals Rodríguez	60	57	822	8	6	Implementación descarga de votaciones en formatos JSON, XML, CSV, Creación de test para la descarga de

						votaciones en distintos formatos, Integración con Travis, Despliegue en heroku, Integración con codacy.
--	--	--	--	--	--	---

1.2 Evidencias de indicadores

https://github.com/enderfews/EGC_Majaceite/graphs/contributors



(Fig 1.2.a)

- [Reporte de horas Angel Pomares](#)
- [Reporte de horas Fernando Wals](#)
- [Reporte de horas Francisco Javier García](#)
- [Reporte de horas Juan Pablo Portero](#)
- [Reporte de horas Alejandro Barba](#)
- [Reporte de horas Alejandro Carrillo](#)

2.Integración con otros equipos

Nuestro equipo eligió la modalidad Single por lo que no hay integración con otros equipos

3.Descripción del sistema

Nuestro proyecto busca contribuir a la mejora de Decide una plataforma de voto electrónico desarrollado por el departamento de Lenguajes y Sistemas Informáticos de la ETSII para la asignatura Evolución y Gestión de la Configuración.

Los incrementos funcionales llevados a cabo por nuestro equipo giran entorno a la visualización de resultados de decide exclusivamente ya que la modalidad de nuestro proyecto es Single, estos incrementos son los siguientes:

- Pintado de gráficas y estudio de datos.
- Descarga de resultados de votaciones en formato JSON, XML, CSV y PDF.
- Mostrar información relevante en tiempo real, como el número de votos, porcentaje del censo, estadísticas de votantes, según perfiles, etc.
- Implementar visualizaciones para diferentes plataformas como Telegram y Slack.

3.1 Pintado de gráficas y estudio de datos

En la sección de pintado de gráficas y estudio de datos, se han hecho una serie de modificaciones, las cuales nos permitirán poder, no solo ver con más claridad y lucidez los resultados de unas votaciones, sino además poder observar y comprobar otra serie de datos relacionados con los votantes que nos ayudarán a poder estudiar datos más concretos relacionados con los usuarios del sistema.

Para ello, se ha añadido un tipo de gráfica nueva: la gráfica tarta; en la que se muestra los datos de una manera más sencilla y mucho más entendible. Para ello, se ha implementado el siguiente código html:

```
graficas.html • visualizer.html
decide > visualizer > templates > visualizer > graficas.html > script > onload
1 <h3>Número de votos totales</h3>
2 <div id="container-piechart-total" class="d-inline-flex border">
3   <canvas id="piechart-total" class="w-100 h-100"></canvas>
4 </div>
5
6 <script>
7
8   var configTotal = {
9     type: 'pie',
10    data: {
11      datasets: [{
12        data: {{data|safe}},
13        backgroundColor: [
14          '#FF0000', '#00FF00', '#00FFFF', '#0000FF', '#000080', '#FFFF00'
15        ],
16        label: 'Population'
17      }],
18      labels: {{labels|safe}}
19    },
20    options: {
21      responsive: true
22    }
23  };
24
25
26  window.onload = function() {
27    var ctx_total = document.getElementById('piechart-total').getContext('2d');
28    window.myPie = new Chart(ctx_total, configTotal);
29  };
30
31 </script>
```

(Fig 3.1.a)

Lo que hacemos es inicializar un script en el que creamos un elemento por nombre “configTotal” y lo ponemos de tipo “pie” (tarta). Lo más importante sería pasarle, desde el views.py, el “data”, que va a ser los resultados de las votaciones, y el “labels”, que serán las opciones que tengamos disponibles como elección en la votación. Con “backgroundColor” añadimos los colores que queramos que tenga la gráfica, por orden de opciones (labels) y, cuando hayan más opciones que colores, los colores que aparecerán serán de la escala de grises. Con “{% include './graficas.html' %}” en “visualizer.html” conseguimos que las gráficas sean mostradas en el html del módulo visualizer.

```
def get_context_data(self, **kwargs):
    context = super().get_context_data(**kwargs)
    vid = kwargs.get('voting_id', 0)
```

(Fig 3.1.b)

```

try:
    r = mods.get('voting', params={'id': vid})
    voting = json.dumps(r[0])

    # Aquí hacemos del json un diccionario
    voting_information = json.loads(voting)
    postproc = voting_information["postproc"]

    # Metemos en data y labels la información necesaria
    a = 0; b = 0
    labels = []; data = []

    while a < len(postproc):
        option = postproc[a]
        labels.append(option["option"])
        a += 1

    while b < len(postproc):
        option = postproc[b]
        data.append(option["votes"])
        b += 1

    # La añadimos al context
    context['voting'] = voting
    context['data'] = data
    context['labels'] = labels
    context['VotId'] = vid
except:
    raise Http404

return context

```

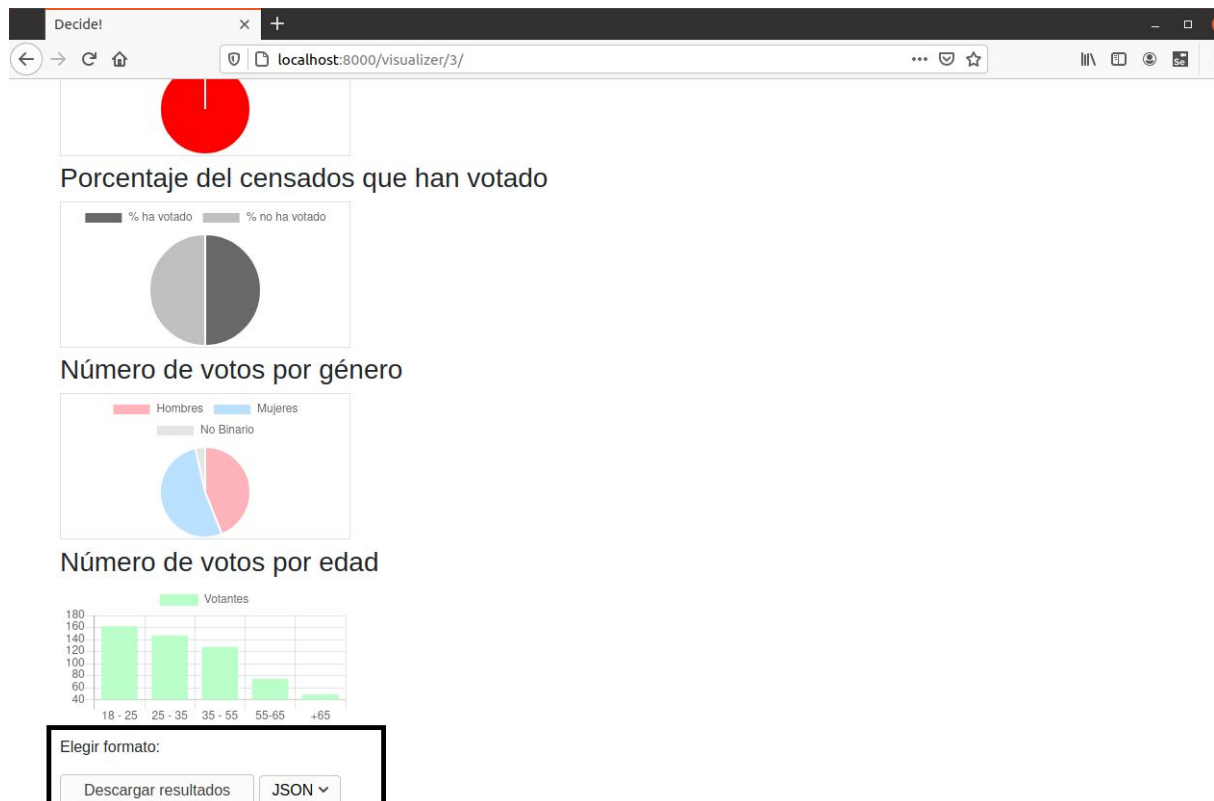
(Fig 3.1.c)

Mediante el código expuesto aquí, creamos los objetos de tipo diccionario “data” y “labels”, que enviaremos, por medio del “context”, a la página html para que pueda ser utilizada por la gráfica. Los datos necesarios para “labels” y “data” son conseguidos al transformar un objeto de tipo “json” y transformarlo en un diccionario, con el cual rellenamos “labels” y “data”.

3.2 Descarga de los resultados en diferentes formatos(CSV, PDF,JSON,XML)

La descarga de resultados permite que una vez que se ha terminado una votación y se han contado los resultados, los usuarios puedan descargar los resultados en diferentes formatos de los que se disponen.

El sistema funciona de la siguiente manera: Cuando se accede al visualizer para ver una votación, podemos ver un botón que nos permite descargar el resultado. Junto con un “select” con distintas opciones. Esas opciones son: CSV,JSON,PDF y XML.



(Fig 3.2.a)

```

<!-- Código añadido por Fernando-->
form action="/downloadResults/" method="GET">
  {% csrf_token %}
  <p>Elegir formato:</p><input type="submit" value="Descargar resultados" />
  <input type="hidden" name="VotID" value={{VotID}} />
  <select name="Formato">
    <option value="json">JSON</option>
    <option value="csv">CSV</option>
    <option value="xml">XML</option>
    <option value="pdf">PDF</option>
  </select>
</form>
<!-- Fin del código añadido por Fernando-->

```

(Fig 3.2.b)

Cuando se pulsa en “descargar resultados” se envían los datos al view de visualizer. La URL a la que te manda sería `../downloadResults/`. Esta url se encuentra en el `urls.py` del módulo principal de decide.

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('doc/', schema_view),
    path('gateway/', include('gateway.urls')),
    path('downloadResults/', VisualizerView.downloadResults),
    path('', include('django_telegrambot.urls')),
    path('bot/', include('bot.urls')),
    path('actions/', include('actions.urls')),
]
```

(Fig 3.2.c)

Se le pasan dos datos: “VotId”; que recoge el Id de la votación y “Formato”; donde se guarda el tipo de formato. El view del visualizer se encarga de gestionar esa petición mediante el método de downloadResults. Este método recogerá los dos datos que se han nombrado. Mediante el “VotId”, buscará en votaciones la votación con ese id. Y de ahí sacará la propiedad de “postproc”, que contiene todos los resultados de esa votación. El dato “Formato” decidirá en qué formato se devolverán los resultados. Si elegimos “csv” se ejecutará el siguiente código:

```
elif request.GET["Formato"]=="csv":
    listed_values=[]
    for d in Vote.postproc:
        values=[]
        for v in d.values():
            values.append(v)
        listed_values.append(values)
    response=HttpResponse(content_type='text/csv')
    writer=csv.writer(response)
    writer.writerow(['Votes','Number','Option','postproc'])
    for value in listed_values:
        writer.writerow(value)
    response['Content-Disposition']= 'attachment; filename="votingResults.csv"'
    return response
```

(Fig 3.2.d)

Primero extraemos los votos y sus distintos datos. Mediante un for anidado sacamos el número de votos, número de opción, nombre de la opción y el postproc de cada opción de la votación. El resultado es un array de arrays, donde cada array tiene esos datos. Luego, mediante la librería “csv” creamos un writer para escribir en la HttpResponse. Primero escribimos la primera fila, que tendrá los nombres de cada columna y finalmente con un bucle for, vamos escribiendo los datos. Antes del return, la penúltima línea se encarga de guardar el resultado en ‘Content-Disposition’ con el nombre de “votingResults.csv”. Django lo utiliza para devolver archivos.

Si elegimos Json se ejecuta el siguiente código:


```
elif request.GET["Formato"]=="json":
    response=JsonResponse({'results':Vote.postproc})
    response['Content-Disposition']= 'attachment; filename="votingResults.json"'
    return response
```

(Fig 3.2.e)

Como el postproc es un atributo que ya viene en formato JSON, solo hay que guardarlo en un JsonResponse y utilizar 'Content-Disposition' para que el ordenador del usuario descargue el resultado en ese formato.

Si elegimos xml se ejecutará el siguiente código:

```
elif request.GET["Formato"]=="xml":

    data= XT.Element('Results')
    votes=XT.SubElement(data,'votes')
    numbers=XT.SubElement(data,'numbers')
    opts=XT.SubElement(data,'options')
    postprocs=XT.SubElement(data,'postprocs')
    listed_values=[]
    for d in Vote.postproc:
        Values=[]
        for v in d.values():
            Values.append(v)
        listed_values.append(Values)
    for i in range(0,len(listed_values)):
        v=XT.SubElement(votes,'vote')
        v.set('name','v' + str(i))
        v.text=str(listed_values[i][0])
        n=XT.SubElement(numbers,'number')
        n.set('name','n'+str(i))
        n.text=str(listed_values[i][1])
        o=XT.SubElement(opts,'option')
        o.set('name','o'+str(i))
        o.text=str(listed_values[i][2])
        p=XT.SubElement(postprocs,'postproc')
        p.set('name','p'+str(i))
        p.text=str(listed_values[i][3])

    stringData=XT.tostring(data)
    response=HttpResponse(stringData,content_type='text/xml')
    response['Content-Disposition']= 'attachment; filename="votingResults.xml"'
    return response
```

(Fig 3.2.f)

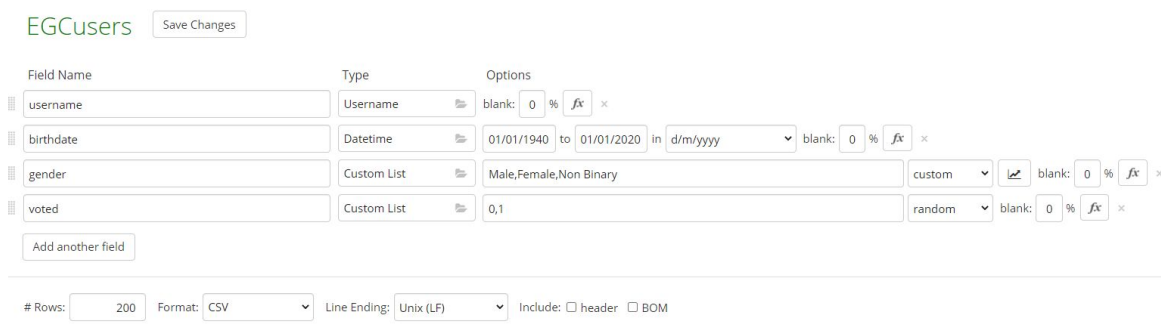
Primero se crea una etiqueta raíz llamada "Results", partiendo de ahí se crean las raíces "votes", "numbers", "options" y "postproc". Mediante un for anidado sacamos el número de votos, número de opción, nombre de la opción y el postproc de cada opción de la votación. El resultado es un array de arrays, donde cada array tiene esos datos. Luego, con un bucle for se van añadiendo en cada etiqueta los valores de cada opción de la votación. Por ejemplo, las tres primeras líneas de código del bucle, crea un nuevo elemento que cuelga de "votes", el nombre que se le da a esa etiqueta es 'v' seguido del número de la iteración y

el valor es el número de votos de esa opción. De esta manera para el resto de atributos. Finalmente se pasa a string y se envía de la misma forma que para el resto de formatos.

3.3 Mostrar estadísticas de los votantes

Esta mejora del módulo visualizer permite calcular estadísticas de los votantes de una votación como su edad, su género o si están censados y mostrar estos datos en gráficas para mejorar su visualización.

Lo primero que se ha hecho es utilizar la plataforma [Mockaroo](#) para generar “dummies” de usuario que con nombres, fechas de nacimiento, género e información sobre si han votado o no y descargarlos en csv.



(Fig 3.3.a)

Este paso es necesario ya que nuestra modalidad de desarrollo es single y no tiene una integración con el módulo de *autenticación* que tenga usuarios con los atributos género o fecha de nacimiento, así que se ha optado por cargar estos usuarios desde un csv.

Este archivo se almacena en la carpeta *visualizer/resources* y se carga utilizando la función *readCSV* definida en *visualizer/utlis.py*

```
#Read a csv file from a given path
def readCSV(filepath):

    result = []

    with open(filepath,'r') as csvfile:
        reader = csv.reader(csvfile,delimiter=',')
        for row in reader:
            result.append({'username':row[0], 'birthdate':row[1], 'gender':row[2], 'voted':row[3]})

    return result
```

(Fig 3.3.b)

Una vez extraídos estos datos del csv se necesitan dos métodos *get_votes_by_age* y *calculate_age* que procesan estos datos, *get_votes_by_age* recibe una lista de fechas de nacimiento de tipo string y un rango de edades convirtiendo estas fechas de nacimiento a edades y clasificándolas en un map donde que cada par clave/valor indica una edad y el

número de personas de esas edad que han votado. Por otro lado *calculate_age* calcula la edad de una persona a partir de una fecha de nacimiento en formato String.

```
# Receives a range of ages and a list of birthdates and returns a list containing
# the number of people in each age range
def get_votes_by_age(age_range, birthdates):

    ages=[]
    #Mapping date strings to ages
    for bs in birthdates:
        ages.append(calculate_age(bs, True))

    #filling up the return list with zeros
    res = dict(zip(age_range, [0]*len(age_range)))

    #iteriting the list and clisifiing the values
    for age in ages:
        for ar in reversed(age_range):
            if age >= ar:
                res[ar] += 1

    return res

def calculate_age(born, is_string=False):

    if is_string:
        born = datetime.strptime(born, "%d/%m/%Y").date()

    today = datetime.today()
    return today.year - born.year - ((today.month, today.day) < (born.month, born.day))
```

(Fig 3.3.c)

Una vez extraídos los datos y procesados con las 2 funciones anteriores se cargan los datos en el contexto para que las reciba la vista *views.py* donde se cargarán las graficas

```

54
55     var configGenero = {
56       type: 'pie',
57       data: {
58         datasets: [{
59           data: {{gender_votes}},
60           backgroundColor: [
61             '#ffb3ba', '#bae1ff'
62           ],
63           label: 'Population'
64         }],
65         labels: ["Hombres", "Mujeres", "No Binario"]
66       },
67       options: {
68         responsive: true
69       }
70     };
71

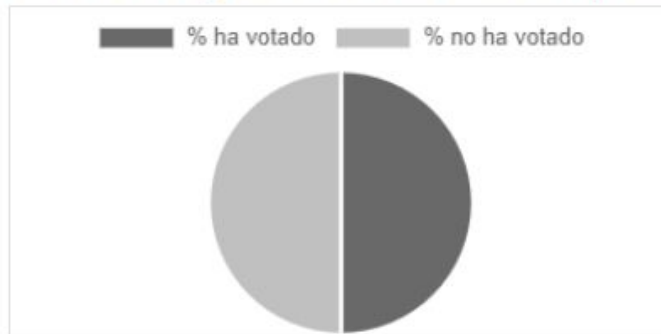
```

(Fig 3.3.d)

Las gráficas(ubicadas en el archivo *templates/graficas.html*) se incrustan en el template *views.py* a través de la función *include*.

El resultado final al renderizar las gráficas es este:

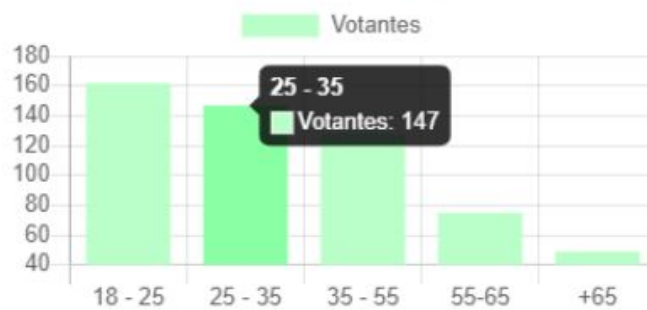
Porcentaje del censados que han votado



Número de votos por género



Número de votos por edad



(Fig 3.3.e)

3.4 Integración con Telegram y Slack

Bot de Telegram

Es importante destacar que el código relacionado con el bot de Telegram se encuentra en un nuevo módulo llamado “bot”.

Requisito previo:

- pip install django-telegrambot
- sudo apt install snapd
- sudo snap install ngrok

Pasos a seguir:

La comunicación con el bot no se hace desde Decide sino desde el propio Telegram, aunque para que funcione es importante ejecutar la aplicación siguiendo los siguientes pasos:

- Primero tendremos que ejecutar la aplicación de decide con el entorno virtual ya activado:
`/decide$ python3 ./manage.py runserver`
- Una vez ejecutada la aplicación, en otra pestaña de la terminal, ejecutamos ngrok con el siguiente comando:

```
ngrok http 8000
```

IMPORTANTE: Se puede añadir un parámetro al comando para que la conexión sea a Europa en lugar de a Estados Unidos. Para este ejemplo no lo hemos usado pero sería recomendable hacerlo si se va a probar la aplicación mucho porque las respuestas serían más rápidas. Dicho comando sería así:

```
ngrok http -region=eu 8000
```

Al ejecutar el comando de ngrok se nos abrirá una pestaña como esta:

```
ngrok by @inconshreveable

Session Status      online
Session Expires    1 hour, 59 minutes
Version            2.3.35
Region             United States (us)
Web Interface       http://127.0.0.1:4040
Forwarding          http://3dbc771ec750.ngrok.io -> http://localhost:8000
Forwarding          https://3dbc771ec750.ngrok.io -> http://localhost:8000

Connections      ttl    opn    rt1    rt5    p50    p90
0                0      0.00   0.00   0.00   0.00
```

(Fig 3.4.a)

En dicha pestaña, hacemos click derecho en la dirección que contiene el “https” y le damos a copiar la dirección del enlace. En mi caso:

```
https://3dbc771ec750.ngrok.io
```

- Una vez hecho esto debemos irnos al archivo settings.py de la carpeta decide y modificar el valor del “BASEURL” a la dirección que hemos copiado y guardamos los cambios:

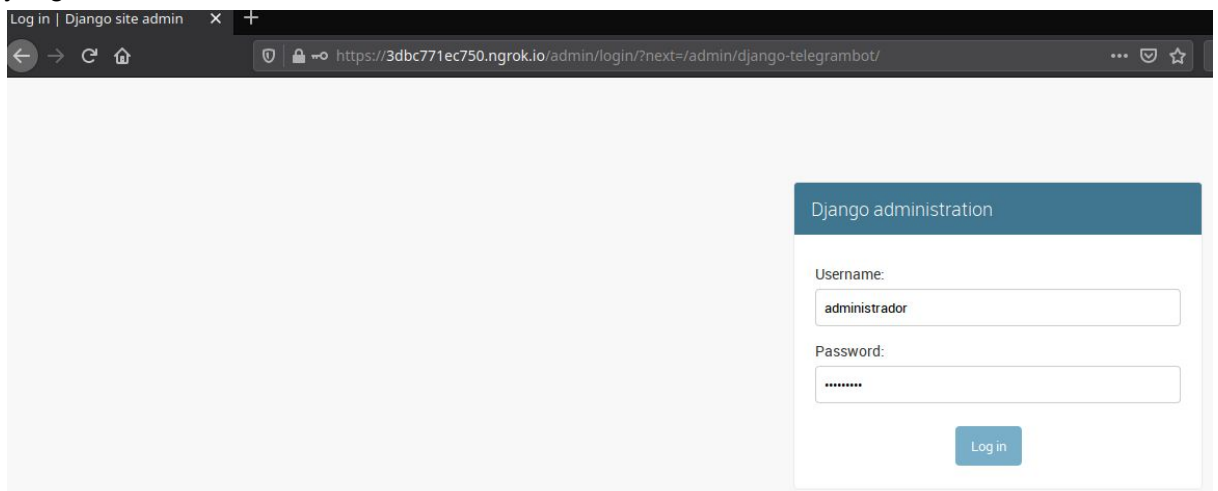
```
settings.py X
decide > decide > settings.py > BASEURL
75 | BASEURL = 'https://3dbc771ec750.ngrok.io'
```

(Fig 3.4.b)

- Cuando hagamos esto ya podemos dirigirnos a la misma dirección que hemos copiado, que es donde está desplegada nuestra aplicación ahora, añadiendo al final un “/bot/” para que se nos abra la siguiente pestaña:



- El siguiente paso será darle al enlace que pone “**Django-Telegrambot Dashboard**” y loguearnos con un usuario administrador en Decide:



(Fig 3.4.c)

- Una vez realizado esto se nos mostrará una pestaña en la que hay una lista con los bots disponibles, que en nuestro caso sólo es uno:

Django-TelegramBot

The full documentation is at <https://django-telegrambot.readthedocs.org>.

Bot update mode: **WEBHOOK**

Bot List:

 **EGC_Majaceite_Telegram_Bot** @EGCTestBot

Add to group/channel

Recent actions

My actions

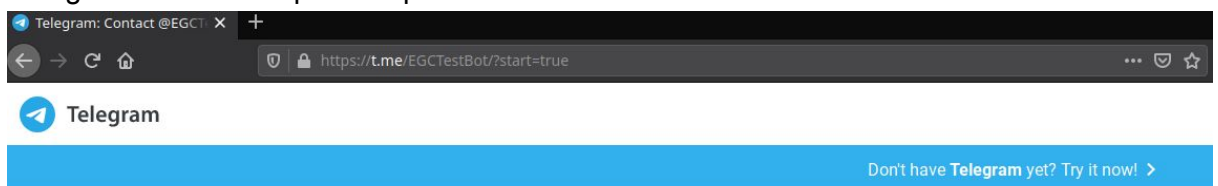
- + 1: 1
Vote
- + Census object (1)
Census
- + test
Voting
- + http://localhost:8000
Auth
- + test
Question
- + administrador
User

(Fig 3.4.d)

- Para poder comunicarnos con el bot, tendremos que darle al botón que pone: “@EGCTestBot”

@EGCTestBot

- Una vez hecho esto nos redirigirá a una página de Telegram donde te permite escoger entre varias opciones para conversar con el bot:



EGC_Majaceite_Telegram...
@EGCTestBot

SEND MESSAGE

OPEN IN WEB

(Fig 3.4.e)

- En mi caso elegiré abrirlo en la web:

OPEN IN WEB

(Fig 3.4.f)

- Cuando se nos abre la siguiente página de Telegram ya tendremos el contacto del bot abierto y con un comando abajo del todo que nos permite empezar a conversar con él:

START

(Fig 3.4.g)

- Si hemos realizado todos los pasos bien y en orden, el bot debe respondernos a dicho comando cuando lo pulsemos. La respuesta que nos da es la siguiente:



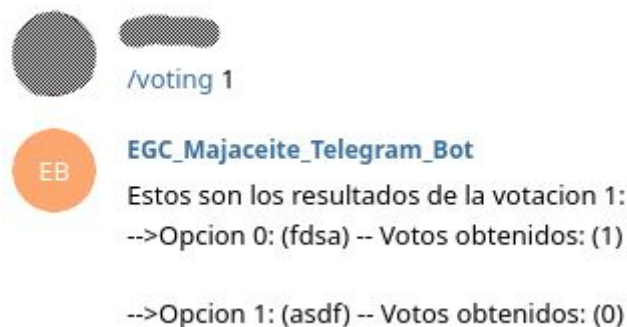
(Fig 3.4.h)

- Como nos dice el propio bot, podemos mandarle el comando "/help" escribiéndolo en el chat y dándole a enviar. Si lo hacemos, nos responde con la lista de todos los comandos disponibles y una breve descripción:



(Fig 3.4.i)

- El comando que nos interesa para saber las votaciones es el de "/voting *votingID*". Es importante destacar que como bien dice en la descripción del comando éste no funcionará si la votación no ha finalizado y ha ejecutado el tally correctamente. Vamos entonces a poner el comando "/voting 1" pues yo tengo ya una votación con ese ID:



(Fig 3.4.j)

Cómo iniciar bot en slack:

Antes de nada, señalar que todo el código de slack se encuentra en un nuevo módulo “actions”

Cuenta en slack para las pruebas y modificaciones (hay que loguearse para entrar a ciertas urls de este documento):

usuario: PruebasEGC99

contraseña: paraprobar99

Prerequisitos:

-pip install slack

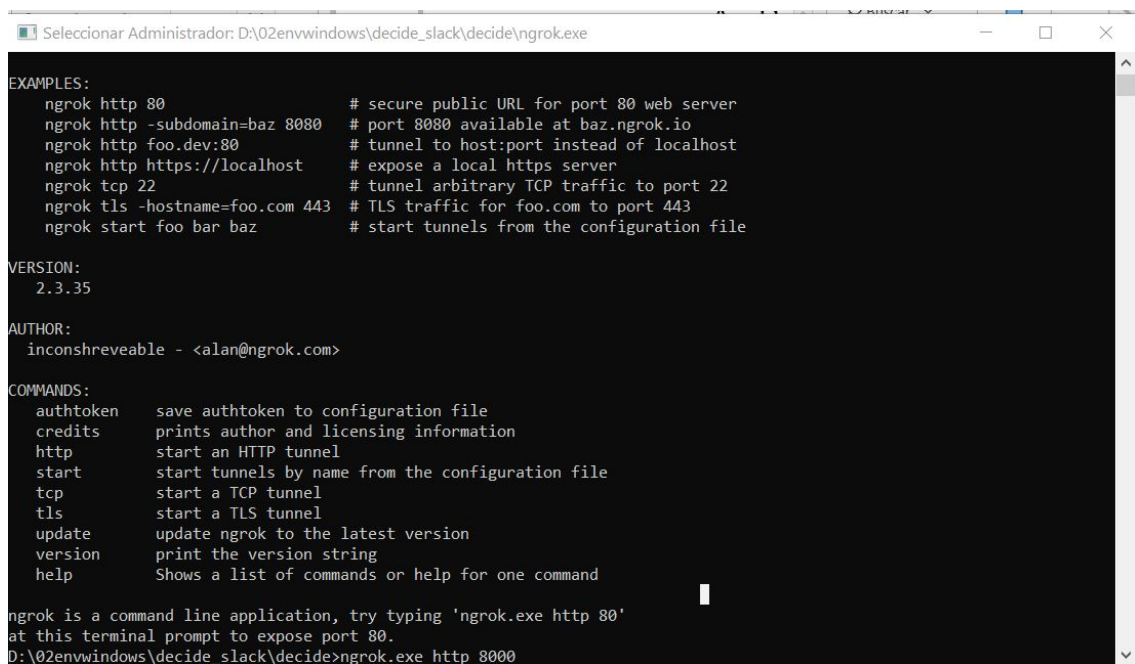
-pip install slackclient

Para iniciar el bot, lo primero que debes de hacer es ejecutar la aplicación accediendo a la ruta del proyecto y ejecutando el comando siguiente:

```
/decide$ python manage.py runserver 0:8000
```

Trás hacer esto, tendremos el proyecto corriendo de forma local, pero para usar el bot slack necesitamos una url pública, para esto buscaremos el ejecutable “ngrok.exe” en la ruta del proyecto, y lo ejecutamos.

Una vez dentro pondremos el siguiente comando:



```
Seleccionar Administrador: D:\02envwindows\decide_slack\decide\ngrok.exe

EXAMPLES:
  ngrok http 80                        # secure public URL for port 80 web server
  ngrok http -subdomain=baz 8080      # port 8080 available at baz.ngrok.io
  ngrok http foo.dev:80              # tunnel to host:port instead of localhost
  ngrok http https://localhost       # expose a local https server
  ngrok tcp 22                       # tunnel arbitrary TCP traffic to port 22
  ngrok tls -hostname=foo.com 443    # TLS traffic for foo.com to port 443
  ngrok start foo bar baz            # start tunnels from the configuration file

VERSION:
  2.3.35

AUTHOR:
  inconshreveable - <alan@ngrok.com>

COMMANDS:
  authtoken  save authtoken to configuration file
  credits    prints author and licensing information
  http       start an HTTP tunnel
  start      start tunnels by name from the configuration file
  tcp        start a TCP tunnel
  tls        start a TLS tunnel
  update     update ngrok to the latest version
  version    print the version string
  help       Shows a list of commands or help for one command

ngrok is a command line application, try typing 'ngrok.exe http 80'
at this terminal prompt to expose port 80.
D:\02envwindows\decide_slack\decide>ngrok.exe http 8000
```

(Fig 3.4.k)

y obtendremos lo siguiente:

```
Administrador: D:\02env\windows\decide_slack\decide\ngrok.exe - ngrok.exe http 8000
ngrok by @inconshreveable
Session Status      online
Session Expires    7 hours, 59 minutes
Version            2.3.35
Region             United States (us)
Web Interface       http://127.0.0.1:4040
Forwarding          http://3a2f6e1be22a.ngrok.io -> http://localhost:8000
                   https://3a2f6e1be22a.ngrok.io -> http://localhost:8000
Connections
  ttl    opn    rt1    rt5    p50    p90
   0      0     0.00   0.00   0.00   0.00
```

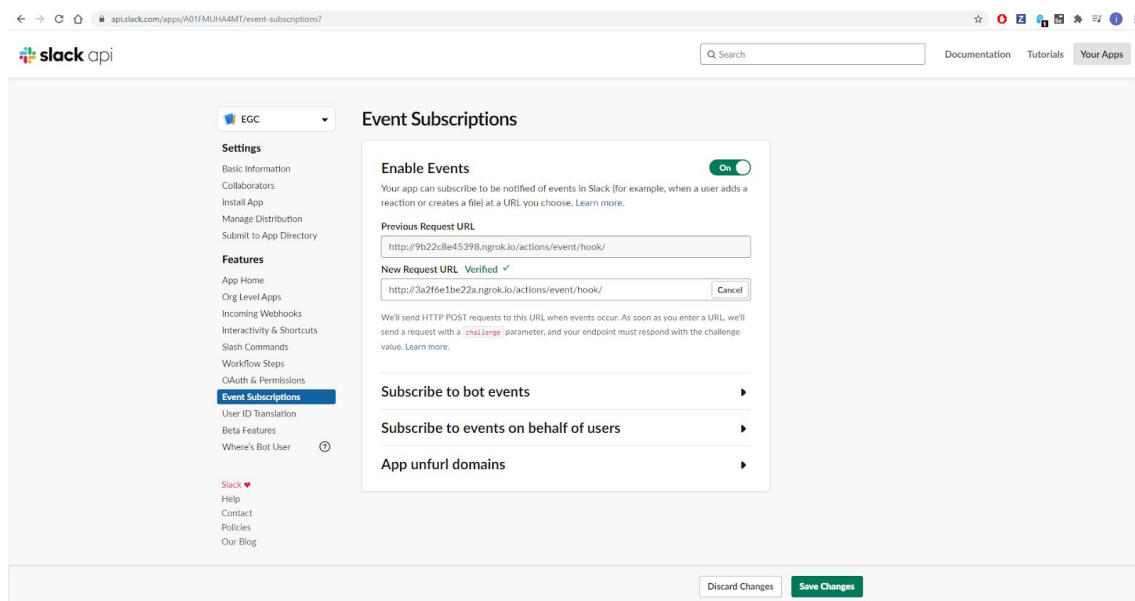
(Fig 3.4.l)

Donde el campo “forwarding” contiene la dirección que utilizaremos de ahora en adelante

Para usar esa dirección en slack, debemos ir a la dirección

<https://api.slack.com/apps/A01FMUHA4MT/event-subscriptions?>

y una vez allí, en la configuración de eventos ponemos la siguiente ruta que contiene el gancho del proyecto con el que el bot quedará conectado a la web:



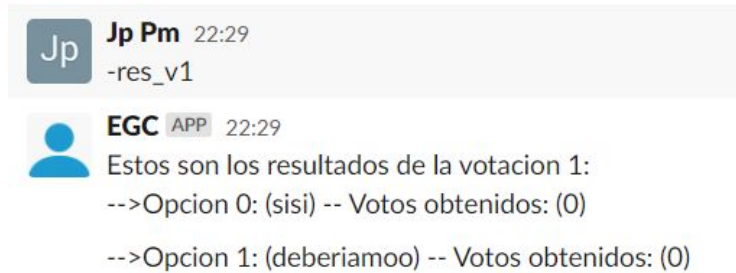
(Fig 3.4.m)

Si lo hemos hecho correctamente, veremos que la dirección queda verificada, esto es que slack ha hecho una petición a nuestra aplicación y esta ha respondido. Debemos guardar los cambios para que la configuración quede realizada.

Una vez realizada esta configuración previa, para probar que funciona solo tenemos que acceder a <https://resultadosegc.slack.com/> y entrar en el canal de ingeniería del software, donde el bot está activo.

Se tienen disponibles el comando “-help” con el que se obtienen los comandos disponibles y el más importante “-res_vX” siendo X el id de la votación de la que queremos ver los resultados.

Esta será la respuesta del bot al pedirle una votación concreta:



(Fig 3.4.n)

4. Visión global del proceso de desarrollo

A continuación se describirán la serie de pasos a seguir desde que se publica una issue hasta el paso por las fases en las que esta se desarrolla, se testea, y acaba en producción:

1. El primer paso es detectar una necesidad de desarrollo en el proyecto ya sea una Bug, Una mejora, o cualquier otro tipo de desarrollo y crear una incidencia desde la sección issues de GitHub tal y como se explica en la guía *Gestión de incidencias* que hay en este mismo documento, para este ejemplo la issue se llamará *issueEjemplo*.
2. Arrancaremos clockify para poder registrar el tiempo que tardamos en resolver la issue
3. Arrancamos la máquina virtual de decide o la distribución ubuntu donde tengamos el sistema instalado
4. Activamos el entorno de python3 donde lo tengamos definido utilizando el comando `source carpetaEnvironment/bin/activate`
5. Entramos en la carpeta donde tengamos el proyecto decide, si no lo tenemos descargado necesitaremos tener instalado git y utilizar el comando `git clone https://github.com/Ferwalrod/EGC_Majaceite.git`
6. Entramos en la carpeta decide y hacemos un `git fetch` para tener actualizado el índice de ramas del proyecto en remoto
7. Si la incidencia consiste en crear una nueva funcionalidad o mejora nos cambiamos a la rama *develop* y descargamos los últimos cambios que haya en esta en remoto con `git pull origin develop`.
8. Entonces creamos una nueva rama a partir de la última versión de develop con el comando `git branch -b issueEjemplo` y ya podríamos empezar a desarrollar el código de la issue

9. Para ello utilizaremos el editor de Código que prefiramos compatible con Django, la mayoría de nuestro equipo ha utilizado *VisualStudioCode* y escribiremos el código de la mejora dentro del módulo correspondiente de decide
10. Una vez terminado de implementar el código pasaremos a hacer una prueba rápida para comprobar que todo funciona, para ello tenemos que arrancar el servidor situándonos con el terminal dentro de la carpeta de decide y ejecutar el comando `python3 ./manage.py runserver 0:8000`, si estamos ejecutándolo en una partición ubuntu podemos omitir el `0:8000`
11. Tras comprobar rápidamente que el cambio funciona, pasamos a añadir los cambios a la staging area y a commitarlos en el repositorio local, todo esto lo podemos hacer con el siguiente comando: `git commit -am "Descripción del cambio hecho"`
12. Con el commit hecho volvemos a cambiar a la rama develop para comprobar si ha sido actualizada por algún otro compañero y traer los últimos cambios con `git pull origin develop`
13. Con develop actualizada volvemos a cambiar a la rama de nuestra issue con `git checkout issueEjemplo` y mergeamos los cambios de develop en nuestra rama con `git merge develop`
14. Solucionamos conflictos de mergeo si los hubiese, una opción muy buena para solucionar conflictos es utilizar la herramienta gráfica *gitkraken* para git que aparte de ser un gestor de repositorios excepcional permite solucionar conflictos en 2 archivos comparándolos de una manera muy sencilla e intuitiva.
15. Tras solucionar los conflictos hacemos un commit como se ha explicado en el paso 11 y pusheamos los cambios a remoto utilizando el comando `git push origin issueEjemplo`.
16. Y ahora volvemos a mergear a develop para que estén en develop los cambios de nuestra rama *issueEjemplo* podemos hacerlo con los comandos `git checkout develop` y `git merge issueEjemplo` respectivamente.
17. Ahora pusheamos a origen los cambios de *develop* con la rama *issueEjemplo* integrada con `git push origin develop`
18. Con develop en remoto actualizado con los cambios de nuestra feature es momento de hacer los test de nuestra feature, para ello tenemos la rama QA, esta rama está pensada para funcionar como una versión de develop desde la que hacer el testing de lo implementado en el proyecto.
19. Lo primero que haremos sera mergear en QA los últimos cambios que hemos subido a develop, para ello nos cambiaremos a la rama QA en local con `git checkout QA` y con `git pull origin QA` obtenemos los últimos cambios de QA. Por último hacemos un `git merge develop` para mergear develop a QA
20. A continuación sacamos una rama de testeo para nuestra feature desde QA con `git checkout -b issueEjemplo-Test`
21. Implementamos los test de nuestra feature, si son test de unitarios o de controlador lo haremos en el archivo `tests.py` y si son test de selenium en el archivo `testsSelenium.py` del modulo correspondiente.
22. Cuando terminemos los test tendremos que ejecutar todos los tests para comprobar que los que hemos añadido no han roto otros que ya funcionaban, para ello utilizaremos el comando `python3 ./manage.py test --exe -v 2`, El `--exe` podemos omitirlo si estamos lanzando los tet en un sistema linux nativo.
23. Tras comprobar que pasan todos los test pasaremos a mergear la rama *issueEjemplo-Test* en QA utilizando el proceso descrito anteriormente

24. Una vez mergeados tenemos que volver a bajarnos en local la última versión de develop y mergearla en QA para resolver conflictos con nuestros tests recién añadidos, si hay conflictos los resolvemos, hacemos commit y mergeamos a *develop*
25. A continuación pusheamos develop a remoto, si hemos llegado hasta aquí tendremos ya en el repositorio remoto la feature implementada con los test hechos y comprobados.
26. Tras cada push a develop Travis se encarga de lanzar un proceso por el que hace una build de la ram develop y pasa los tests del sistema para comprobar que todo funciona correctamente, si funciona todo actualiza el *README.md* de la rama donde indicará que la build ha pasado.
27. El siguiente paso es mergear develop en el resto de ramas relevantes del repositorio, primero haremos un mergeo a la rama *deployment*. Travis está configurado para que al detectar cambios en la rama *deployment* se genere una nueva build, se pasen los test y si todos pasan se desplieguen los cambios en heroku.
28. Tras desplegar en heroku comprobaremos que todo funciona entrando en nuestro sistema desplegado desde la url <https://egc-majaceite.herokuapp.com>
29. El siguiente paso es actualizar la rama *docker-integration* que es la rama desde la que docker está configurada para descargarse los archivos del repositorio al levantar el contenedor, la url de esta rama está configurada en el *docker-compose.yml*. Para actualizar esta rama haremos un merge de *develop* a *docker-integration* con el proceso descrito anteriormente.
30. Cuando hagamos el merge debemos asegurarnos de tener todas las dependencias que requiere el proyecto en docker, si hemos incluido alguna nueva deberemos incluirla o bien en el archivo *requirements.txt* dentro de la carpeta *decide* o bien actualizar el archivo *dockerfile* dentro de la carpeta *docker* y añadir la dependencia nueva con *RUN pip install nombredependencia* si es una dependencia de pip o con el comando *RUN apk add --no-cache nombredependencia* si es una dependencia tipo apt
31. Tras añadir la dependencia hacemos un commit con los cambios tal como se ha descrito en pasos anteriores.
32. Por ultimo mergearemos el cambio a main si el cambio ha sido debido a un hotfix de un problema que estuviese en main y se ha arreglado.
33. Para ello nos cambiaremos a la rama main con *git checkout main*, hacemos un *git pull origin main* para actualizar nuestra rama main local con los últimos cambios del remoto y mergeamos develop a main con *git merge develop*.
34. Cuando travis detecte cambios en main lanzará el proceso de construcción de build y actualizará el *README.md* indicando si la build y los test de *main* han pasado o no.
35. Por último sacaremos una nueva release de main con el hotfix hecho y las versionariamos como se explica en el apartado *Creación de releases* de este mismo documento.
36. Hecho todo lo anterior solo nos queda cerrar la incidencia en github para ello podemos consultar la sección *Creación de releases* de este documento

4. Entorno de desarrollo

El sistema elegido por el equipo se ha sido alguna distribución basada en linux, esto ha variado de compañero a compañero pero la mayoría a utilizado ubuntu 20.04, ya sea instalando el sistema operativo desde cero en una partición independiente o a través de una máquina virtual como se indicaba en la práctica 2 de EGC.

La guía ofrecida por el profesorado que hemos seguido para hacer la configuración básica de decide puede encontrarse en el siguiente enlace:

https://1984.lsi.us.es/wiki-egc/images/egc/c/c2/01-Entorno_en_VirtualBox.pdf

Los comandos para realizar la instalación el resto de la configuración pueden encontrarse en este video proporcionado por el profesorado de la asignatura:

<https://ascinema.org/a/o46wrY8hAungZe7ZaxSaSggEV>

Los pasos para la configuración de decide son los siguientes:

1. Arrancar la máquina y loguearnos con el usuario que hayamos creado
2. Ejecutar el siguiente comando para actualizar los repositorios de código de nuestra distribución de linux
`sudo apt-get update`
3. Instalamos librerías y dependencias necesarias para el proyecto
`sudo apt-get install python3 python3-venv python3-pip postgresql libpq-dev`
4. Creamos un entorno virtual de python para nuestro proyecto
`python3 -m venv python3EnvDec`
5. Activamos el entorno virtual que acabamos de crear
`source python3EnvDec/bin/activate`
6. Instalamos git con el siguiente comando
`sudo apt-get install git-all`
7. Realizamos la configuración previa de git desde la siguiente guía:
<https://1984.lsi.us.es/wiki-egc/index.php/ConfPreviasPractica3>
8. Nos vamos a la carpeta compartida que hemos creado previamente
`cd CarpetaCompartida`
9. Creamos una carpeta para clonar nuestro repositorio y lo clonamos
`mkdir decide`
`cd decide/`
`git clone https://github.com/Ferwalrod/EGC_Majaceite.git`
10. Instalamos wheel
`pip install wheel`
11. Entramos en la carpeta decide y ejecutamos el comando pip install para que instale todas las dependencias del proyecto en nuestro sistema
`pip install -r requirements.txt`
12. Entramos en la base de datos de postgre y creamos un usuario que para operar la base de datos
`sudo su - postgres`
`psql -c "create user decide with password 'complexpassword'"`
`psql -c "create database decidedb owner decide"`
`exit`
13. Crearemos un superusuario en nuestro sistema para poder entrar al panel de administración
`./manage.py migrate`


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

14. Ahora descargamos un configuraremos los drivers de chrome y firefox para que se ejecuten correctamente, con los siguientes comandos vistos en la práctica de Pruebas Software

```
wget https://github.com/mozilla/geckodriver/releases/download/v0.27.0/geckodriver-v0.27.0-linux64.tar.gz
tar -xzf geckodriver-v0.27.0-linux64.tar.gz
chmod +x geckodriver
sudo cp geckodriver /usr/bin/
rm geckodriver-v0.27.0-linux64.tar.gz
```

15. Seguimos la siguiente lista de instrucciones, encontrada en la práctica [Despliegue de contenedor en Docker](#)

```
#Desinstalamos versiones antiguas:
2 sudo apt-get remove docker docker-engine docker.io containerd runc
3
4 #Instalamos dependencias
5 sudo apt-get update
6
7 sudo apt-get install \
8     apt-transport-https \
9     ca-certificates \
10    curl \
11    gnupg-agent \
12    software-properties-common
13
14 # Instalamos llave de cifrado
15 curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo
apt-key add -
16
17 # Añadimos el repositorio
18 sudo add-apt-repository \
19     "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
20     $(lsb_release -cs) \
21     stable"
22
23 # Instalamos docker
24 sudo apt-get update
25 sudo apt-get install docker-ce docker-ce-cli containerd.io
26
27 # Instalamos docker-compose
28 sudo apt-get install docker-compose
29
30 # Añadimos nuestro usuario al grupo docker
31 sudo usermod -aG docker $USER
32
33 # Tenemos que salir de la sesión y volver a entrar para que los
cambios tomen efecto
34
35 # Probamos que funciona con:
36 docker run hello-world
```

16. Llegado a este punto hemos terminado la configuración del entorno

Las versiones de las dependencias en el archivo requirements.txt de nuestro proyecto son las siguientes:

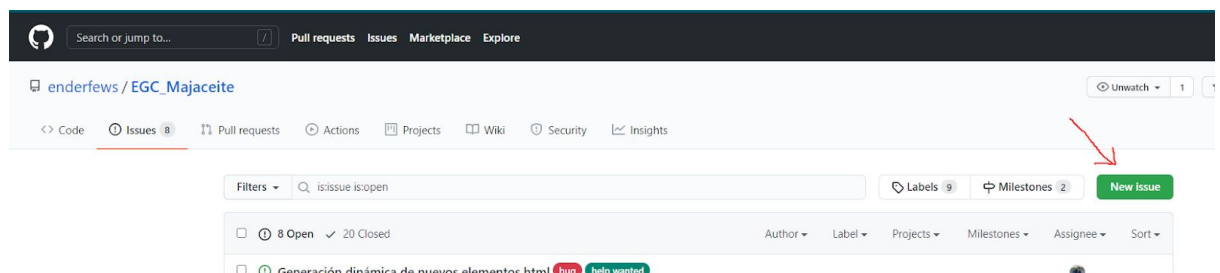
```
Django==2.0.8
pycryptodome==3.6.6
djangorestframework==3.7.7
django-cors-headers==2.1.0
requests==2.20.0
django-filter==1.1.0
psycopg2==2.7.5
django-rest-swagger==2.2.0
Pillow==5.3.0
xhtml2pdf==0.2.3
django-heroku
gunicorn
psycopg2-binary==2.7.6.1
python-gettext==4.0
django-makemessages-xgettext==0.1.1
social-auth-app-django
django-nocaptcha-recaptcha
```

5.Gestión de incidencias

5.1 Gestión de incidencias internas

La gestión de incidencias llevada a cabo por el equipo se hace a través de la plataforma github mediante la pestaña issues. Para ello el equipo sigue los siguientes pasos:

1. Detectamos una incidencia del tipo Bug/ERROR, Mejora, Documentación, Desarrollo, Ayuda, Bloqueo o Duda
2. Nos vamos al repositorio de github EGC_Majaceite > Pestaña issues y le damos al botón *New Issue*



(Fig 5.1.a)

3. Le damos un título a la incidencia que describa la cuestión de manera breve y concisa
4. Rellenamos la descripción siguiendo la siguiente plantilla en formato Markdown:

(Descripción detallada del problema)

![image](url de la imagen)

Pasos para replicar el error:

1. (Paso 1)
2. (Paso 2)
3. (Paso 3)

Soluciones propuestas:

- (Solución propuesta 1)
- (Solución propuesta 2)

****Autor incidencia**:** (Nombre del miembro del equipo que pone la incidencia)

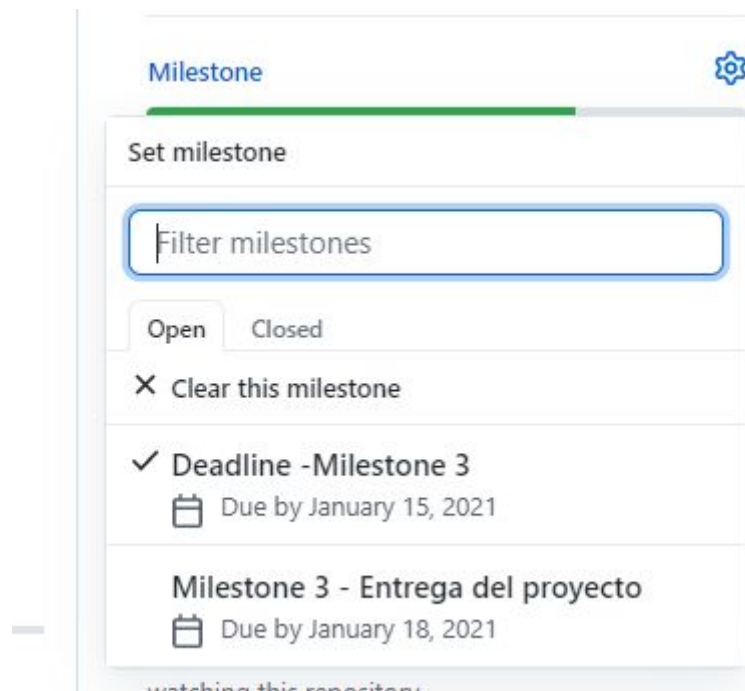
****Estado**:** (Se pone uno de los siguientes: New, Accepted, Started, Fixed, Verified)

****Prioridad**:** (Se pone uno de los siguientes: Priority-Critical, High, Medium, Low)

****Tipo**:** (Se pone uno de los siguientes: Bug/ERROR, Mejora, Documentación, Desarrollo, Ayuda, Bloqueo, Duda)

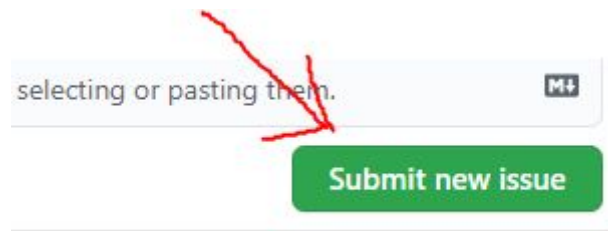
****Rol de encargado**:** (Se pone uno de los siguientes: Coordinador, Programador, Tester, DevOps)

5. En la parte derecha añadimos pulsando *Assignees* a la persona/s encargada/s de solucionar la incidencia.
6. Pulsamos en Label para etiquetar la incidencia y elegimos una de las propuestas por github por defecto
7. Hacemos click Milestone y añadimos la milestone a la que pertenece la incidencia



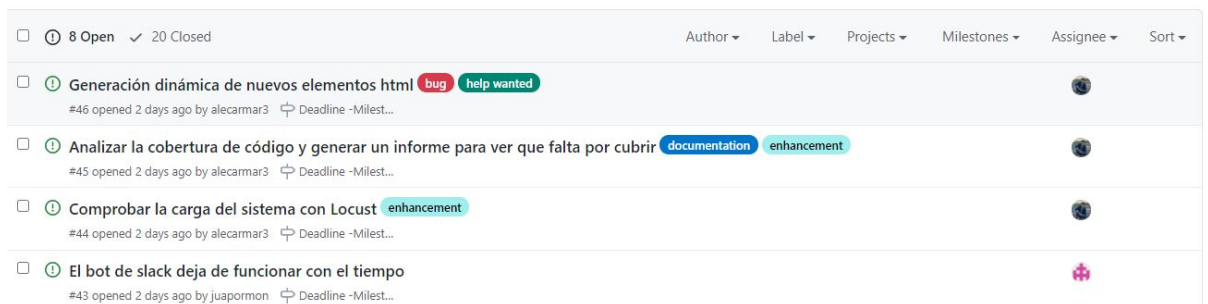
(Fig 5.1.b)

8. A continuación hacemos click en *Submit New Issue* para publicar la incidencia



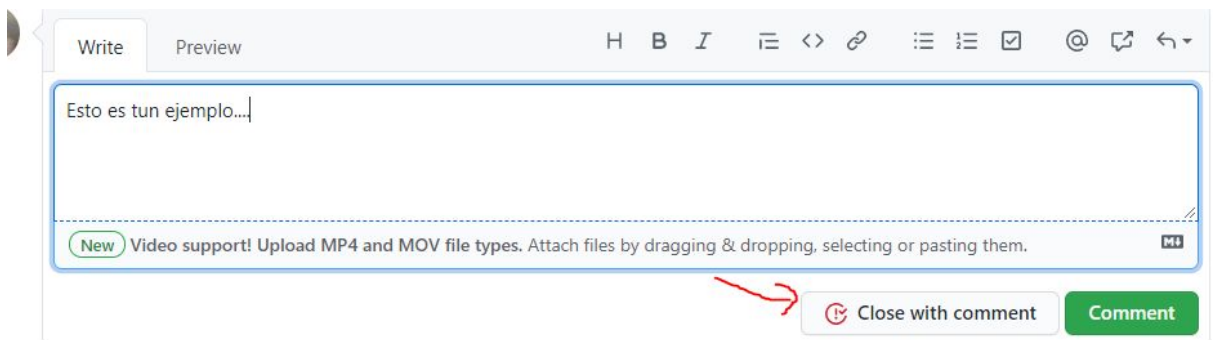
(Fig 5.1.c)

9. Cuando hayamos solucionado la cuestión que ocupa la incidencia podemos cerrarla yéndonos a la pestaña Issues de github, y haciendo click en la issue que queramos cerrar dentro de la pestaña open.



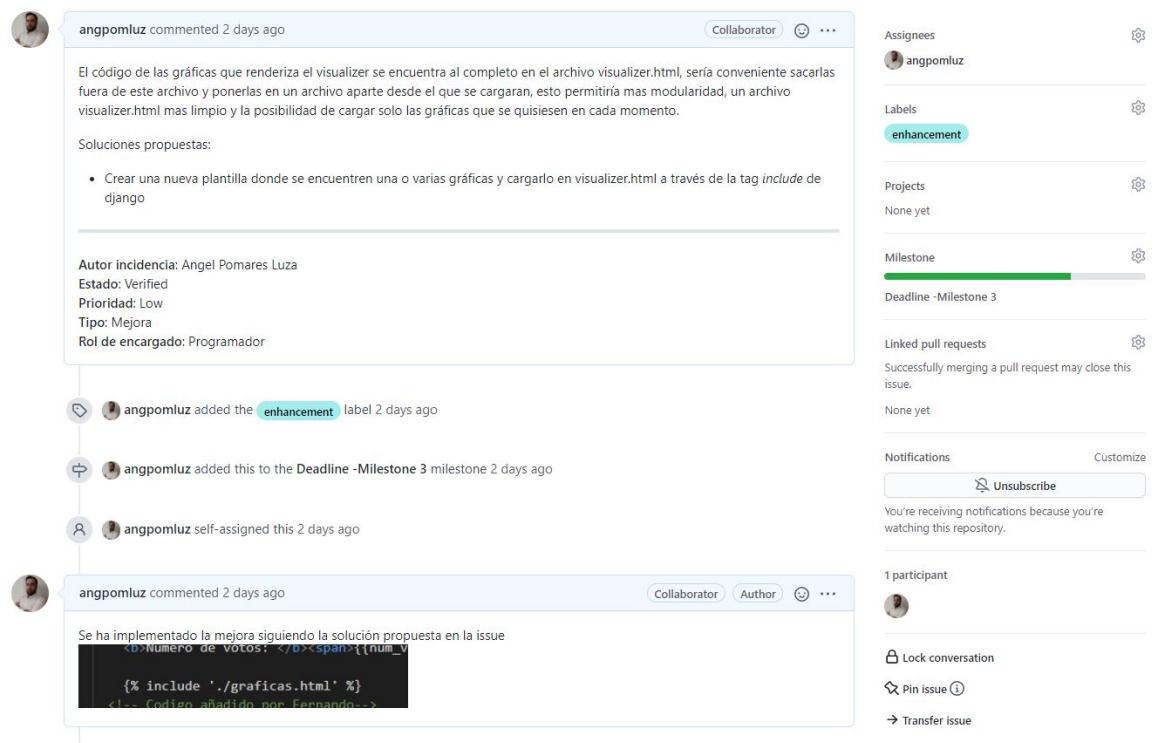
(Fig 5.1.d)

10. Por último rellenamos el comentario de resolución de la incidencia y hacemos click en *Close Issue*



(Fig 5.1.e)

Aquí un ejemplo de una issue solucionada:



(Fig 5.1.f)

Para finalizar indicaremos cómo rellenar los atributos relevantes de la incidencia del apartado 4 de la lista anterior:

Estado:

- New: La incidencia se ha creado pero aún no se ha asignado
- Accepted: Se ha asignado la incidencia a uno o varios miembros del equipo para que la resuelvan
- Started: La incidencia ha comenzado a resolverse
- Fixed: La incidencia está resuelta
- Verified: La incidencia aparte de arreglarse se ha probado y verificado que el cambio funciona y es estable

Prioridad:

- Priority-Critical: Deben resolverse de manera inmediata ya que es una incidencia que provoca un retraso en otras tareas y resulta bloqueante
- High: Es una incidencia importante con alta prioridad y debe resolverse antes que las de prioridad medium o low
- Medium: Es una incidencia que debe arreglarse pero no es prioritaria
- Low: Es una incidencia que poca o ninguna relevancia, normalmente una pequeña mejora que no afecta al desarrollo o la estabilidad del proyecto

Tipo:

- Bug/ERROR: Indica que el reporte de la incidencia se debe a un fallo en el sistema o en el código
- Mejora: Indica una mejora a implementar en el sistema

- Documentación: Reporta que se debe crear o hacer alguna modificación sobre en los documentos del proyecto
- Desarrollo: Indica que hay alguna funcionalidad que se debe implementar y se ha pasado por alto.
- Ayuda: Indica una petición de ayuda de un compañero al resto del equipo con una tarea o resolución de un fallo
- Bloqueo: Indica que el autor de la incidencia ha encontrado un bloqueo en la implementación que no le deja avanzar o un error que no es capaz de resolver.
- Duda: Indica un tipo de incidencia para preguntar una duda al equipo de una manera más formal y extensa

Rol de encargado:

- Coordinador: Encargado de organizar las reuniones y gestionar la comunicación del equipo
- Programador: Encargado del desarrollo de los incrementos del proyecto y resolución de errores en el código
- Tester: Encargado de realizar testear lo implementado por los programadores y resolver los errores en los test
- DevOps: Figura encargada de realizar las tareas de configuración relacionadas con la gestión del repositorio, integración continua y despliegue en remoto

5.2 Gestión de incidencias externas

Dado que nuestro proyecto es tipo Single y solo trabaja en él nuestro equipo de desarrollo no se producen incidencias externas con otros equipos.

Si la incidencia externa se produce en alguna de las tecnologías ligadas a nuestra aplicación como Travis o Heroku, y esta provoca por ejemplo que el servicio de Travis deje de estar disponible por un tiempo, se debe crear una incidencia en la tabla issues y notificar a los compañeros para que estén informados

6. Gestión del código fuente

En el proyecto decide-single-Majaceite hemos decidido utilizar github como plataforma para la gestión del repositorio remoto, para ello se hizo un fork de <https://github.com/EGCETSII/decide.git> y se creó un manual de uso del repositorio al comienzo del proyecto para que los miembros del equipo tuviesen una metodología para trabajar con él, el manual se expone a continuación:

6.1 Estructura de ramas del repositorio

- **Main:** En esta rama irán las versiones del producto que estén preparadas para ir a producción p.e.: v0.0, v1.0, v2.0, etc. Las que tengan un incremento importante en features, también irán las versiones intermedias en las que en las que se haya realizado una hotfix.
- **Develop:** En esta rama irán todos los cambios del proyecto que se vayan completando día a día, nuevas features, cambios en las features que ya hay, cambios en la configuración del proyecto etc. Será la rama base desde la que saldrá cada nueva rama de feature en la que se vaya a trabajar
- **QA:** En esta rama se realizarán las features que tengan que ver con el testing de la aplicación y el control de la calidad.
- **Deployment:** Esta rama se llevarán a cabo los despliegues de la aplicación en heroku

6.2 Nomenclatura de ramas, commits y tags

Ramas de nuevas features:

Se nombran como: **MXFXX - nombreFeature**

X : el número de la milestone a la que pertenece la feature

XX : el número de la feature que irá desde 01 hasta 99

nombreFeature: un nombre en camelcase que describa brevemente la feature, por ejemplo visualizaciónTelegram

Ramas para arreglar fallos en features:

Se nombrarán como: **MXFXX - Fix - nombreArreglo**

MXFXX: El identificador de la feature

nombreArreglo: un nombre en camelcase que describa brevemente el fallo que se va a arreglar, p.e.: metodo1ReturnFormatoIncorrecto

Ramas de test:

Se nombran como: **MXFXX - Test**

MXFXX: El identificador de la feature

En estas ramas se harán todos los tipos de test tanto los unitarios como de controlador y selenium

Commits:

Título: Se describe brevemente lo que se está desarrollando, arreglando o testeando, p.e.: Avance desarrollo integración con slack

Descripción: Se describe al nivel que se quiera lo que se está haciendo, si es el último commit de una feature, arreglo o test antes de mergear a develop tiene que venir bien descrito todo lo que se ha hecho

Tags:

El versionado de etiquetas se hará para las ramas main, develop y QA siguiendo la siguiente estructura:

vX.X.X

X : versión mayoritaria de la aplicación a la que se incrementará con cada milestone, comenzará en la 0 y cuando se entregue la milestone 2 avanzará a la 1

X : versión incremental de la aplicación correspondiente a nueva funcionalidad añadida, empezará en 0 y se irá incrementando en uno con cada feature añadida.

X : versión correspondiente a un incremento, cambio o arreglo hecho en el proyecto, por ejemplo la etiqueta v1.1.3 corresponde al tercer cambio hecho en la primera milestone.

6.3 Crear una nueva feature

Modo consola

1. Abrimos la consola
2. Nos situamos en el repositorio del proyecto
3. Cambiamos a la rama develop

```
$ git checkout develop
```

4. Hacemos pull de la última versión de develop para asegurarnos de trabajar con los últimos cambios

```
$ git pull origin develop
```

5. Nos vamos a trello y copiamos el código MXFFX de la feature que vamos a implementar y le damos un nombre siguiendo la nomenclatura indicada en *Nomenclatura de ramas, commits y tags*
6. Creamos la rama y nos cambiamos a ella

```
$ git checkout -b 'M2F01-Fix-R4m4T4c0Dr4r4'
```

7. Pusheamos la rama a github para que aparezca en el remoto

```
$ git push origin 'M2F01-Fix-R4m4T4c0Dr4r4'
```

8. Ya tenemos lista la rama para trabajar sobre ella

6.4 Modificar una feature existente

Modo consola

1. Abrimos la consola
2. Nos situamos en el repositorio del proyecto
3. Nos situamos en develop

```
$ git checkout 'develop'
```

4. Hacemos un pull request de develop desde el repositorio remoto para asegurarnos de tener la última versión de la rama

```
$ git pull origin 'develop'
```

5. Cambiamos a la rama a modificar

```
$ git checkout 'M2F01-Fix-R4m4T4c0Dr4r4'
```

6. Hacemos pull de la última versión de la rama a modificar para asegurarnos de trabajar con los últimos cambios

```
$ git pull origin 'M2F01-Fix-R4m4T4c0Dr4r4'
```

7. Mergeamos develop en nuestra rama de feature para asegurarnos de que tenemos los últimos cambios de develop y que no hay conflictos.
8. Si hay conflictos los tenemos que arreglar primero y hacer un commit.
9. Hacemos las modificaciones que queramos en la feature
10. Realizamos commit y push de los cambios realizados

```
$ git add .  
$ git commit -m "CommitMessage"  
$ git push origin 'M2F01-Fix-R4m4T4c0Dr4r4'
```

6.5 Mergear una feature acabada

Modo consola

1. Abrimos la consola
2. Nos situamos en el repositorio del proyecto
3. Cambiamos a la rama develop

```
$ git checkout develop
```

4. Hacemos un pull request de develop desde el repositorio remoto para asegurarnos de tener la última versión de la rama

```
$ git pull origin 'develop'
```


5. Cambiamos a la rama de la feature que queremos mergear a develop

```
$ git checkout 'M2F01-Fix-R4m4T4c0Dr4r4'
```
6. Mergeamos develop a nuestra rama de feature(si hay conflictos los solucionamos y hacemos un commit con los conflictos arreglados)

```
$ git merge 'develop'
```
7. Pusheamos a remoto la última versión de la rama de feature ya integrada con develop

```
git push origin 'M2F01-Fix-R4m4T4c0Dr4r4'
```
8. Volvemos a cambiarnos a la rama develop

```
$ git checkout develop
```
9. Mergeamos la feature acabada a develop

```
$ git merge --no-ff 'M2F01-Fix-R4m4T4c0Dr4r4'
```
10. Pusheamos a remoto develop

```
$ git push origin develop
```

6.6 Corregir fallos en una feature

Supongamos que dicho fallo se encuentra en la rama master y necesitamos un hotfix

Modo consola

1. Abrimos la consola
2. Nos situamos en el repositorio del proyecto
3. Creamos una rama para realizar el hotfix

```
$ git checkout master  
$ git checkout -b hotfix_branch
```
4. Realizamos los cambios necesarios
5. Realizamos commit de los cambios realizados

```
$ git add .  
$ git commit -m "CommitMessage"
```
6. Con los cambios del hotfix realizados, mergeamos tanto a master como a develop

```
$ git checkout master  
$ git merge hotfix_branch  
$ git checkout develop  
$ git merge hotfix_branch
```
7. Borramos la rama del hotfix

```
$ git branch -d hotfix_branch
```

6.7 Borrar una rama creada por error

Modo consola

Si hemos creado una rama por error y queremos eliminarla del remoto escribiremos:

```
$ git push origin --delete ramaPocha
```

Luego cambiaremos a la rama develop

```
$ git checkout develop
```

Y la borrarémos en el repositorio local

```
$ git branch -d ramaPocha
```

6.8 Testear una feature

1. Para testear una feature lo primero tiene que estar integrada en develop como explica la sección *Mergear una feature acabada* de este mismo manual
2. Lo segundo que tenemos que hacer es asegurarnos de que la rama QA tiene la última versión de develop, Para ello haremos un pull de la última versión de develop, y de la última versión de QA
3. Hacemos pull de la última versión de develop

```
$ git checkout develop
```

```
$ git pull origin develop
```

4. A continuación nos cambiamos a la rama QA y nos traemos la última versión

```
$ git checkout QA
```

```
$ git pull origin QA
```

5. Comprobamos que la última versión de ambas ramas apunte al mismo commit:

```
$ git log -n 2
```

Al ejecutar el comando podemos ver como las ramas QA y develop de remoto están en el mismo commit



```
administrador@egcubuntuserver ~/CarpetaCompartidaVM/EGC_Majaceite/decide (QA)$ git log -n 2
commit 5250e163f5945e8e829c74379363cb91e55c2753 (HEAD -> QA, origin/develop, origin/QA, develop, M2F02-Test)
Author: Angel <angelpomaresus@gmail.com>
Date: Tue Dec 29 19:49:48 2020 +0000

    Merge M2F03-infoTiempoReal into develop

commit 33401ac0a81ef297b758c4e638c0113aa4e90d39 (tag: v0.0, origin/main, origin/HEAD, main, M2F03-MostrarInfoTiempoReal, M2F01-Fix-R4m4T4c0Dr4r4)
Author: Angel Pomares <angelpomaresus@gmail.com>
Date: Mon Nov 9 08:53:07 2020 +0100

    decide files included
```

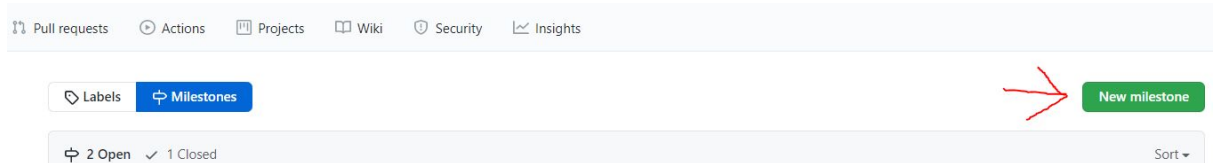
(Fig 6.8.a)

6. Comprobado lo anterior lo único que tenemos que hacer es sacar la rama de testeo de nuestra feature a partir de QA y empezar a testear
7. Cuando hayamos acabado el testeo tenemos que mergear la rama de test en QA otra vez, para ello tendremos que hacer el mismo proceso que el explicado en la sección *Mergear una feature acabada*, los pasos son los mismos solo que esta vez el mergeo tenemos que hacerlo en QA en vez de en develop

6.9 Creación de milestones

El principal responsable de la gestión del repositorio debe crear las milestone oficiales en las fechas de entrega marcadas por la asignatura así como las milestones de deadlines internas del equipo. Para ello se deben seguir los siguientes pasos

1. Abrir el repositorio en github, navegar hasta la pestaña Issues > Milestones
2. Hacer clic en *New Milestone*



(Fig 6.9.a)

3. Introducimos un título breve y conciso que permita identificar la milestone, la fecha de esta y una descripción. A continuación pulsamos en create milestone

New milestone

Create a new milestone to help organize your issues and pull requests. Learn more about [milestones and issues](#).

Title

Entrega de ejemplo

Due date (optional)

10/01/2021

Description

Descripción de la entrega de ejemplo

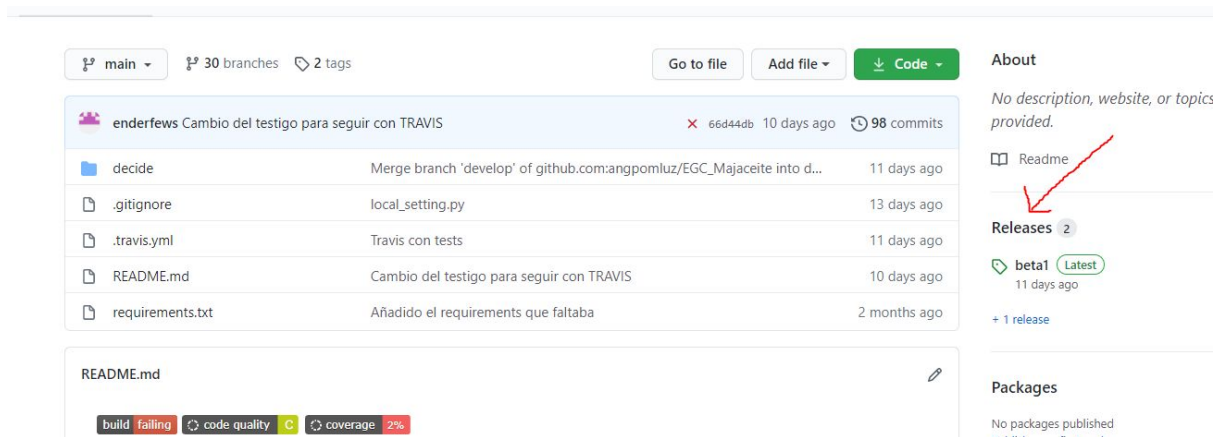
Create milestone

(Fig 6.9.b)

6.10 Creación de releases

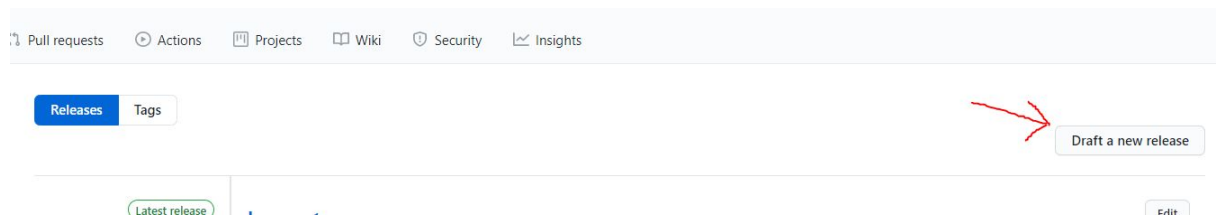
El principal responsable de la gestión del repositorio debe crear una release cada vez que se llegue a una de las milestone de entrega definidas por el calendario de la asignatura, esta release se sacará de la última versión de la rama main(master). Pasos a seguir para crear una release:

1. Abrimos la página principal de nuestro repositorio den github y hacemos click en el enlace Releases de la parte derecha



(Fig 6.10.a)

2. Hacemos click en *Draft a new release*



(Fig 6.10.b)


3. Escribimos la versión de la última etiqueta que haya en la rama main de la aplicación, le damos de título decide-majaceite si el proyecto ha llegado a la versión 1.0 o mayor o beta-decide-majaceite en caso contrario. Escribimos una descripción de la release y le damos a *Publish Release*

Releases

Tags

v1.0

@

 Target: main ▾

×

Duplicate tag name

This tag already has release notes. Would you like to [edit them?](#)


decide-majaceite

Write

Preview

Primera versión funcional de Decide Majaceite

Attach files by dragging & dropping, selecting or pasting them.

 Attach binaries by dropping them here

☐ This is a pre-release

We'll point out that this release is identified as non-production ready.

Publish release

Save draft

(Fig 6.10.c)

Con esto ya tendremos publicada la release de nuestro proyecto

6.11 Modo de uso del repositorio

A continuación se describe el modo de uso y los permisos del equipo de desarrollo sobre el repositorio

- El equipo encargado del desarrollo de incrementos y testeo sólo tendrán permitido trabajar sobre la rama *develop* y *QA* pero no deberán hacer commits sobre ellas directamente, en lugar de ello se debe seguir el siguiente procedimiento:
 - a. Los desarrolladores que quieran integrar o modificar una feature deben sacar una rama a partir de la última versión de *develop*, acabar su trabajo en la rama de la feature, actualizar *develop*, volver a mergear *develop* en su rama de feature por si hubiese conflictos solucionarlos en su propia rama y por último mergear su rama en *develop*.
 - b. Los testers que quieran realizar test sobre una feature deberán mergear primero la última versión de *develop* sobre *QA* y a partir de aquí seguir el procedimiento del paso anterior pero sacando ramas y mergenado en *QA* en vez de en *develop*
- El miembro o miembros del equipo encargados de la integración con travis trabajarán con la rama *deployment* y deben configurar travis para que trabaje con esta rama también, para ello cada vez que *develop* avance se debe hacer un merge de *develop* a *deployment* y encargarse de que travis lance los test y cree la build en *deployment* comprobando que no hay errores, hecho esto travis deberá subir los cambios a heroku.
- El miembro encargado de la gestión del repositorio debe encargarse de etiquetar correctamente cada rama que lo requiera cuando se alcance una milestone, así como de crear las milestones del proyecto, se debe crear como mínimo las milestones para las deadlines internas del equipo y la milestone de entrega del M2, M3 y M4 indicadas en la asignatura. También debe ser el encargado de mergear a la rama *main* la última versión estable de *develop* el día antes de la entrega de un milestone y de sacar una release a partir de esa versión de *main*.
- En cuanto a la gestión de incidencias todos los miembros del equipo tendrán permiso para crear y modificar incidencias llevando a cabo el siguiente procedimiento:
 - a. Si se quiere crear una nueva incidencia se seguirá el proceso descrito en la sección Gestión de incidencias de este mismo documento, agregando la incidencia con el atributo estado *Nueva*.
 - b. Cualquier miembro del equipo que quiera aportar en la resolución de la incidencia puede hacerlo utilizando la sección de comentarios de la incidencia
 - c. La persona encargada de resolver la incidencia será también la encargada de cerrar poniendo a su vez un comentario de la solución. Nadie más que la persona que ha solucionado la incidencia o quien la abrió en su defecto debe cerrarla

7.Gestión de construcción e integración continua

En nuestro proyecto decide-single-Majaceite se han utilizado las herramientas:

Hemos utilizado *Codacy* para el análisis de cobertura de los tests, de esta manera podemos ver los “malos olores” del código, ver que no se está probando y mejorarlo. Para la integración continua y despliegue continuo se ha utilizado Travis-CI y Heroku, cada vez que pasa una build de Travis, se despliega en Heroku. Y para la gestión del código fuente y todas las issues, se ha utilizado Github.

El proceso de integración continua es el siguiente:

Se realiza un commit, Travis realiza una build automática de esa rama y comprueba que todo está correctamente. Si se realiza un commit o un merge en la rama de despliegue “*deployment*”, Travis comenzará con el proceso, si tiene éxito entonces desplegará los cambios de la rama hacia Heroku de forma automática. Se ha tenido que añadir nuevos elementos en las dependencias tanto del proyecto como en Travis para su correcto funcionamiento, debido a las mejoras que se han hecho.

En el requirements se ha tenido que añadir nuevos elementos a instalar, debido a que hay que decirle a heroku que dependencias debe instalar en el despliegue. Si no, no puede desplegar. A partir de “*jsonnet*” se han puesto todas las dependencias de nuestro proyecto(y para el despliegue).

```
Django==2.0
pycryptodome==3.6.6
django-rest-framework==3.7.7
django-cors-headers==2.1.0
requests==2.18.4
django-filter==1.1.0
psycpg2-binary==2.7.4
django-rest-swagger==2.2.0
coverage==4.5.2
django-nose==1.4.6
jsonnet==0.12.1
django_heroku
gunicorn
xhtml2pdf
django-telegrambot
slack
slackclient
```

(Fig 7.a)

De este modo, cuando Travis realice la build y lea *requirements.txt*, instalará todas las dependencias que se han usado para la realización del proyecto.

Para selenium tuvimos que configurar Travis para que se descargase todas las dependencias, con ello los drivers necesarios. Esos drivers tuvieron que guardarse en una carpeta del sistema de Travis conocida como “share”.

```
install:
- pip install -r requirements.txt
- pip install codacy-coverage
- pip install selenium
- pip install xhtml2pdf
- pip install django-telegrambot
- pip install slack
- pip install slackclient
- wget -N https://chromedriver.storage.googleapis.com/87.0.4280.88/chromedriver_linux64.zip
  -P ~/
- unzip ~/chromedriver_linux64.zip -d ~/
- rm ~/chromedriver_linux64.zip
- sudo mv -f ~/chromedriver /usr/local/share/
- sudo chmod +x /usr/local/share/chromedriver
- sudo ln -s /usr/local/share/chromedriver /usr/local/bin/chromedriver
- wget -N https://github.com/mozilla/geckodriver/releases/download/v0.28.0/geckodriver-v0.28.0-linux64.tar.gz
  -P ~/
- tar -xzf ~/geckodriver-v0.28.0-linux64.tar.gz -C ~/
- rm ~/geckodriver-v0.28.0-linux64.tar.gz
- sudo mv -f ~/geckodriver /usr/local/share
- sudo chmod +x /usr/local/share/geckodriver
- sudo ln -s /usr/local/share/geckodriver /usr/local/bin/geckodriver
```

(Fig 7.b)

Para poder realizar todo esto se tuvieron que pedir permisos para acceder a esa carpeta con el comando *chmod*.

Se han seguido los siguientes indicadores de calidad de los builds que ofrece *Codacy*:

- Grade
- Issues
- Duplication
- Coverage
- Complexity

8.Gestión de liberaciones, despliegue y entregas

8.1 Licencia utilizada en el proyecto

La licencia elegida para nuestro proyecto es *GNU Affero General Public License v3.0* es la misma licencia que tiene decide originalmente ya que hemos querido preservar los derecho y libertades que se daban.

En la siguiente imagen(Fig 8.1.a) puede verse las restricciones y libertades de la licencia ([fuente imagen](#))

Permissions	Limitations
✓ Commercial use	✗ Liability
✓ Modification	✗ Warranty
✓ Distribution	
✓ Patent use	
✓ Private use	

(Fig 8.1.a)

8.2 Proceso definido para el despliegue

Para el proceso de despliegue se han utilizado dos tecnologías principales impartidas por la asignatura: Travis-CI y Heroku. Ambas se utilizan en el proceso de despliegue.

Para empezar, se ha definido una rama de despliegue llamada “deployment”, donde se encuentra una versión del proyecto preparada para desplegar. Solo los cambios que se hagan en esa rama se verán en la aplicación desplegada. El resto de cambios que se hagan en las demás ramas no se desplegarán. En el fichero de configuración de Travis.yml se define la configuración donde se especifica qué rama se utilizará para el despliegue, qué tecnología usará y la url de despliegue de heroku.

Cada vez que se realice un commit a esa rama o un merge hacia esa rama, empezará con el proceso de integración continua, el cual se encargará Travis-CI. Si el proceso de build es un éxito, Travis-CI realizará el despliegue con las configuraciones definidas en el settings del proyecto. Si fracasa, no realizará el despliegue..

Cada vez que se quiere añadir una nueva issue al proyecto desplegado, se saca una rama de “develop” y se hacen los cambios. Una vez hechos y comprobados, se mergea la rama a develop y esta hacia la rama “QA”, donde se realizarán los tests. Se sacará una rama de QA para realizar los tests de esa nueva issue y una vez comprobados se mergea a “QA”. Finalmente se mergea “QA” hacia “develop” y este se mergea hacia “main” y “deployment”. De esta manera se realiza la inclusión de una nueva issue en nuestra aplicación desplegada.

8.3 Proceso definido para las entregas

Primero se crearán las milestones del proyecto, en nuestro caso serán la M2 y la M3 que son las que implican desarrollo de incrementos sobre el código base de decide, la guía de como crear estas milestones se puede encontrar en la sección *Gestión del código fuente* subsección *Creación de milestones*.

Una vez se hayan creado las milestones y llegue el día de una de ellas se creará una release se mergeará la última versión estable de develop a main que es la rama principal del proyecto, acto seguido travis actualiza automáticamente el indicador del *README.md* para indicar si la build ha pasado o a fallado, y codacy hará lo mismo, cuando el readme esté actualizado con los indicadores tendremos que crea una release cómo se explica en en la sección *Gestión del código fuente* subsección *Creación de releases*.

8.4 Política de nombrado e identificación de los entregables

La política de nombrado que hemos elegido para este proyecto es una muy utilizada en la industria de la ingeniería informática: X.Y.Z. Estos números indican la versión del proyecto, donde cada número representa un cambio mayor o menor. Esta política de nombrado es similar a la sección de gestión de código fuente, subsección tags, donde se define lo siguiente:

El versionado de etiquetas se hará para las ramas main, develop y QA siguiendo la siguiente estructura:

vX.X.X

X : versión mayoritaria de la aplicación a la que se incrementará con cada milestone, comenzará en la 0 y cuando se entregue la milestone 2 avanzará a la 1

X : versión incremental de la aplicación correspondiente a nueva funcionalidad añadida, empezará en 0 y se irá incrementando en uno con cada feature añadida.

X : versión correspondiente a un incremento, cambio o arreglo hecho en el proyecto, por ejemplo la etiqueta v1.1.3 corresponde al tercer cambio hecho en la primera milestone.

9.Ejercicio propuesta de cambio

Vamos a ejemplificar cómo se realizaría la issue #38 - *Las funciones de utilidad creadas no están comentadas*. Seguiremos los siguientes pasos

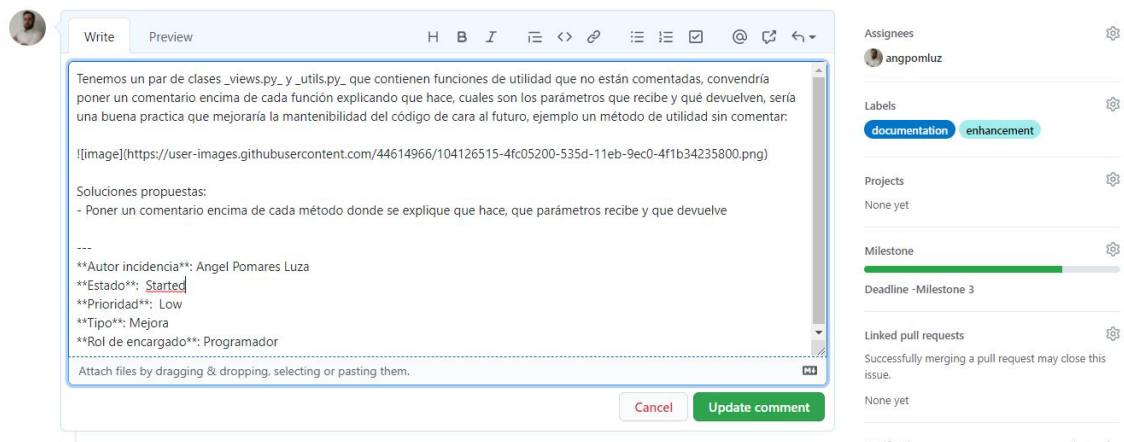
1. Entramos en el tablero de issues y vemos que está abierta la issue #38 - *Las funciones de utilidad creadas no están comentadas* . A continuación hacemos click en ella
2. Leemos la descripción y la solución propuesta, como vemos que podemos hacerla ponemos el clockify para contabilizar el tiempo que vamos a tardar en hacerla

The screenshot displays the Lockify mobile application interface. At the top, there is a dark teal header bar with icons for a star, a folder, a back arrow, and a settings gear. Below this, the app's logo 'lockify' is visible in the top left of the main content area, and a 'Back' button is in the top right. The main content area is a light blue-grey color and contains several white rectangular boxes. The first box at the top shows 'Start: 8:08 PM' with a calendar icon and a timer '00:00:19'. The second box contains the text 'Trabajo en incidencia #38'. The third box shows 'EGC Majaceite' with a downward arrow. The fourth box shows 'Desarrollo' with an upward arrow. Below these is a 'Filter tags' section with three checkboxes: 'Configuración' (unchecked), 'Desarrollo' (checked with a blue checkmark), and 'Documentación' (unchecked). At the bottom of this section is a button with a plus icon and the text 'Create new tag'.

(Fig 9.a)

3. Editamos la descripción de la incidencia y cambiamos el Estado a Started, también nos asignaremos si no estábamos ya asignados en la parte derecha para indicar que vamos a resolver esta incidencia nosotros

Open angpomluz opened this issue 3 days ago · 0 comments



(Fig 9.b)

4. Abrimos una consola y activamos virtual environment para arrancar decide

```
Last login: Wed Jan 13 19:20:17 2021
administrador@egcubuntuser ~ $ ls
CarpetaCompartidaVM decideCode python3EnvDec snap
administrador@egcubuntuser ~ $ source python3EnvDec/bin/activate
(python3EnvDec) administrador@egcubuntuser ~ $
```

(Fig 9.c)

5. Nos vamos a la carpeta donde tenemos el repositorio de git en local, nos cambiamos a la rama develop y hacemos un pull request para tener los últimos cambios de remoto en local

```
(python3EnvDec) administrador@egcubuntuser ~ $ cd CarpetaCompartidaVM/EGC_Maj
aceite/decide
(python3EnvDec) administrador@egcubuntuser ~/CarpetaCompartidaVM/EGC_Majaceit
e/decide (develop)$ git checkout develop
Already on 'develop'
Your branch is up to date with 'origin/develop'.
(python3EnvDec) administrador@egcubuntuser ~/CarpetaCompartidaVM/EGC_Majaceite/decide (develop)$
(python3EnvDec) administrador@egcubuntuser ~/CarpetaCompartidaVM/EGC_Majaceite/decide (develop)$ git pull origin develop
remote: Enumerating objects: 12, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (12/12), done.
remote: Total 12 (delta 2), reused 10 (delta 0), pack-reused 0
Unpacking objects: 100% (12/12), 7.94 KiB | 142.00 KiB/s, done.
From github.com:enderfews/EGC_Majaceite
* branch develop -> FETCH_HEAD
e409294..8dd6e74 develop -> origin/develop
Updating e409294..8dd6e74
Fast-forward
 decide/bot/tests.py | 4 ++--
 decide/decide/settings.py | 4 ++--
 decide/local_settings.py | 26 ++++++++
 requirements.txt | 4 ++--
 4 files changed, 19 insertions(+), 19 deletions(-)
(python3EnvDec) administrador@egcubuntuser ~/CarpetaCompartidaVM/EGC_Majaceite/decide (develop)$
```

(Fig 9.d)

6. Creamos una rama a partir de develop que contenga la incidencia 38, la nombraremos como issue38-ComentarCodigo

```
/CarpetaCompartidaVM/EGC_Majaceite/decide (develop)$ git checkout -b issue38-ComentarCodigo
```

(Fig 9.e)

7. Abrimos el editor de código que utilizemos y procedemos a acometer la issue que consiste en comentar el código de los métodos de utilidad para mejorar la mantenibilidad, aquí un ejemplo de una sección en la que se ha trabajado:

```

17
18 # Read a csv file from a local given path
19 #
20 # Parameters: filepath -> Path to the csv file
21 # Returns: returns a list of dictionaries, each dictionary represents a row of the csv
22 # containing as pairs key/value the column name and his val
23 def readCSV(filepath):
24
25     result = []
26
27     with open(filepath, 'r') as csvfile:
28         reader = csv.reader(csvfile, delimiter=',')
29         for row in reader:
30             result.append({'username': row[0], 'birthdate': row[1], 'gender': row[2], 'voted': row[3]})
31
32     return result

```

(Fig 9.f)

8. Una vez hemos terminado de implementar los cambios añadimos los archivos modificados a la staging area de git y hacemos un commit de los cambios

```
git commit -am "Añadidos comentarios a los métodos de utilidad del visualizer"
```

(Fig 9.g)

9. Cambiamos a la rama develop y hacemos un pull request para comprobar si hay cambios nuevos pusheados por otros compañeros

```

(python3EnvDec) administrador@egcubuntuserver ~/CarpetaCompartidaVM/EGC_Majaceite/decide (issue38-ComentarCodigo)$ git checkout develop
Switched to branch 'develop'
Your branch is up to date with 'origin/develop'.
(python3EnvDec) administrador@egcubuntuserver ~/CarpetaCompartidaVM/EGC_Majaceite/decide (develop)$ git pull origin develop
fatal: 'origin' does not appear to be a git repository
fatal: Could not read from remote repository.

```

(Fig 9.h)

10. Volvemos a cambiar a la rama issue38-ComentarCodigo y mergeamos develop en ella

```

jaceite/decide (develop)$ git checkout issue38-ComentarCodigo
jaceite/decide (issue38-ComentarCodigo)$ git merge develop

```

(Fig 9.i)

11. En este caso no hay conflictos pero si los hubiese los resolvemos, hacemos un commit y a continuación volvemos a mergear a develop

```

(issue38-ComentarCodigo)$ git checkout develop

(develop)$ git merge issue38-ComentarCodigo

```

(Fig 9.j)

12. Ahora pusheamos los cambios al repositorio remoto

```
(develop)$ git push origin develop
```

(Fig 9.k)

13. Con el cambio implementado nos vamos a la incidencia abierta en github y hacemos los cambios en el estado donde indicaremos que está en estado Verified, y por último ponemos un comentario con la solución y cerramos la incidencia pulsando el botón *Close with comment*.

The screenshot shows a GitHub interface. At the top is a comment form with a text area containing the following text:

Autor incidencia: Angel Pomares Luza
Estado: Verified
Prioridad: Low
Tipo: Mejora
Rol de encargado: Programador
Below the text area is a file upload instruction: "Attach files by dragging & dropping, selecting or pasting them." To the right of the text area are two buttons: "Cancel" and "Update comment".
Below the form is an activity feed with three items:
1. angpomluz added **documentation** **enhancement** labels 3 days ago
2. angpomluz added this to the **Deadline -Milestone 3** milestone 3 days ago
3. angpomluz self-assigned this 3 days ago
Below the activity feed is another comment form. It has a "Write" tab and a "Preview" tab. The "Write" tab is active, showing a text area with the text: "Se han comentado los métodos de utilidad del modulo visualizer". Below the text area is the same file upload instruction. To the right of the text area are two buttons: "Close with comment" and "Comment".

(Fig 9.l)

14. Con la incidencia cerrada paramos el clockify y hemos terminado



(Fig 9.m)

10.Conclusiones y trabajo futuro

Esta experiencia de desarrollo y configuración nos ha servido para darnos cuenta la cantidad de trabajo y complejidad que hay detrás de un plan de configuración correcto para un proyecto y lo susceptible que es cualquier cambio afectar a otros puntos del proyecto, a su vez también hemos visto la utilidad del despliegue continuo y la cantidad de tiempo que ahorra. También nos ha servido para conocer mejor el framework Django que se ha demostrado muy simple y rápido a la hora de montar proyectos web pequeños y medianos.

Como mejora para futuros años proponemos desde la parte del visualizer que es en la que hemos trabajado la implementación de un sistema automatizado para enviar las gráficas de resultados de las votaciones por correo en formato imagen a los votantes.