CABINA DE VOTACIÓN

DOCUMENTACIÓN DEL PROYECTO

EVOLUCIÓN Y GESTIÓN DE LA CONFIGURACIÓN

21/12/2014

AGUADERO GARCÍA, FRANCISCO JAVIER

ALCÁNTARA LÓPEZ, ALFONSO

ALGARÍN RODRÍGUEZ, JOSÉ IGNACIO

BORJA GARCÍA – BAQUERO, CARLOS

DELGADO CUDER, JOSÉ JAVIER

JIMÉNEZ VARGAS, DAVID

MAIRELES OSUNA, JUAN ELÍAS

MIÑÓN SABORIDO, DAVID

RODRÍGUEZ TERNERO, LARA

ÍNDICE:

1. RESUMEN DEL PROYECTO:

Cabina de votación es un subsistema que forma parte del proyecto de la asignatura de EGC llamado Ágora US. Este proyecto a su vez está basado en un software real llamado Ágora Voting con el que básicamente podemos realizar y administrar votaciones de forma online.

Ágora US está dividido en 11 subsistemas: Autenticación, Creación y administración de votaciones, Sistema de modificación de resultados, Almacenamiento de votos, Deliberaciones, Recuento, Creación y administración de censos, Frontend de Resultados, Visualización de resultados, Verificación y, por último, Cabina de votación que es el subsistema que hemos desarrollado. Cabina de votación se encarga de crear un espacio donde los usuarios puedan realizar sus votaciones de forma confidencial, por lo que todos los datos, cuando salen de nuestro subsistema, están correctamente cifrados.

El subsistema cabina de votación se ha desarrollado utilizando Python 2.7 y framework Django 1.4.7, además se han añadidos los paquetes del repositorio PIP para poder implementar todas las funcionalidades:

- RSA 3.1.4
- requests 2.4.3
- djangorestframework 2.4.3
- pycrypto 2.6.1

A lo largo de todo el cuatrimestre y a medida que se ha avanzado temario hemos tenido que ir tomando decisiones adaptando nuestro trabajo a los nuevos conocimientos que hemos adquirido en la asignatura, las decisiones que el grupo ha tomado internamente son:

- Telegram como herramienta de comunicación.
- GitLab como repositorio.

GitHub? en todo caso no es un REPOSITORIO!

Las decisiones tomadas por todos los subsistemas son:

- WhatsApp como herramienta de comunicación.
- Projetsii para comunicación.
- GitHub como repositorio común.
- El apartado “issues” de GitHub para las incidencias.

2. INTRODUCCIÓN:

Nuestro proyecto se engloba dentro de un proyecto mayor basado en el software real Ágora Voting: Ágora US. En particular el proyecto que nos ocupa se encarga del desarrollo del subsistema "Cabina de Votación". En él mostrará una interfaz con las opciones de voto que el usuario podrá seleccionar en función de una votación seleccionada. El voto seleccionado por el usuario será cifrado y posteriormente enviado al subsistema de "Almacenamiento de votos".

En este punto hay que mencionar a los subsistemas de "Autenticación", "Almacenamiento de votos", "Creación y administración de votaciones", “Creación y administración de censos” y “Verificación” para poder entender de forma adecuada el funcionamiento de nuestro subsistema. Las relaciones de nuestro subsistema con los mencionados es:

- Autenticación: nos provee de dos cookies (token y user) que identifican al usuario autenticado en el sistema y además, mediante a su API, nos proporciona toda la información almacenada del mismo y si es válido, es decir, si coinciden el token y user.
- Creación y administración de votaciones: nos envía la información correspondiente a la votación seleccionada por el usuario dado el identificador de ésta.
- Creación y administración de censos: nos comunicamos con ellos para saber si un usuario puede realmente votar en una votación. Además, cuando el usuario vota, actualizamos el estado de la votación del usuario (le indicamos que ya ha votado para que no pueda volver a hacerlo).
- Verificación: de este subsistema obtenemos la clave pública para cifrar el voto.
- Almacenamiento de votos: a este subsistema le enviamos el voto cifrado para que lo guarde.

3. GESTIÓN DEL CÓDIGO FUENTE:

3.1. CÓDIGO FUENTE:

Para la gestión del código fuente se utilizará un repositorio de Git, concretamente GitLab. Esta elección se ha realizado en detrimento de SVN debido a:

- Es distribuido.
- Es Software Libre.
- Es más eficiente en la gestión de ramas y archivos.

El repositorio contará con una sola rama master. Durante el desarrollo del subsistema se crearán ramas hijas para el desarrollo de código experimental. Tras la aceptación del código experimental se realizará un merge con la rama master.

Por cada cambio lógico en el sistema, es decir, el cambio de alguna funcionalidad, se realizará un commit al repositorio, sin seguir una política definida en éstos a la hora de su realización.

Cuando el código experimental de una subrama se encuentre finalizado, se realizará un parche y tras la aceptación de dicho parche se aplicará a la rama master.

El rol único y principal en dicha gestión es el de Administrador.

3.2. ENUNCIADO EJERCICIO:

A los alumnos del cuarto curso del Grado en Ingeniería Informática – Ingeniería del Software, de la Escuela Técnica Superior de Ingeniería Informática (ETSII) de la Universidad de Sevilla (US), en la asignatura Evolución y Gestión de la Configuración (EGC) le han mandado hacer una sistema de votación online, Ágora Voting, el cual está compuesto por 11 subsistemas.

Este proyecto tiene un subsistema llamado “Cabina de Votación”, una interfaz que muestra una votación elegida por el votante entre las disponibles, el cual tiene un repositorio de código en GitLab en la dirección https://gitlab.com/cabina-votacion/cabina-agora-us.git. A partir de esa dirección, realice los siguientes cambios en dicho repositorio:

- Clone el repositorio de “Cabina de Votación”.
- Dentro de view.py, modifique uno de sus métodos.
- Cree la rama "prueba" y sitúese en ella.
- Modifique las mismas líneas que antes en el archivo views.py y realice el primer commit.
- Vuelva a la rama master y realice el segundo commit.
- Haga un merge de la rama "prueba" con la rama "master".

- Resuelva los conflictos resultantes para dejar el repositorio en un estado concordante.

No olvide realizar los cambios pertinentes para que queden bien identificados los cambios que realice.

3.3. SOLUCIÓN:

Para realizar el ejercicio utilizaremos la cmd para Windows y un terminal para Linux y ejecutaremos las siguientes instrucciones.

git config --global user.name "Example User"

git config --global user.email example@example.com

git clone git@gitlab.com:cabina-votacion/cabina-agora-us.git

cd cabina-agora-us

En unos sitios dice github y en otros gitlab?

gedit cabina_app/views.py

git branch prueba

git checkout prueba

gedit cabina_app/views.py

git add cabina_app/views.py

hay un par de add que no termino de entender...

git commit

git checkout master

git add cabina_app/views.py

git commit

git merge prueba

4. GESTIÓN DE LA CONSTRUCCIÓN E INTEGRACIÓN CONTINUA:

4.1. CONSTRUCCIÓN:

Para llevar a cabo la construcción del proyecto se ha usado el archivo django-admin.py, proporcionado por el propio framework, que crea la estructura básica del proyecto.

La codificación del subsistema se ha realizado mediante la ayuda del IDE Pycharm y Eclipse con el plugin PyDev, con la versión del lenguaje de programación Python 2.7.

4.2. INTEGRACIÓN:

Los mecanismos para la integración continua no se han llevado a cabo debido a que:

- La integración se ha realizado en local, en nuestros propios equipos de trabajo.
- La coordinación de un grupo tan extenso de personas, y con el uso de tecnologías distintas, lo ha imposibilitado.
- Los conocimientos necesarios para que se llevase a cabo esta integración ha llegado de manera tardía.

Los problemas encontrados durante la integración han sido:

- Una integración a mano, sin automatización, es una dificultad añadida.
- Una mala comunicación entre los distintos subsistemas, debido al uso de varios medios informales en cierto modo (WhatsApp).
- Una cantidad de Software abundante a instalar, en la máquina dónde se realiza, debido a la diversidad de tecnologías utilizadas.
- El desconocimiento de las herramientas usadas por parte de algunos subsistemas (grupos que no sabían que había foros en ProjETSII para discutir).
- Los continuos cambios en el diseño de clases y formas de transmitir datos por parte de algunos grupos.

Las facilidades por parte de nuestro grupo para llevarla a cabo han sido:

- Un primer despliegue en la nube.
- En Linux, un script para instalar las dependencias del subsistema.
- En Windows, un manual de usuario con los pasos para instalar todas las dependencias.
- Se ha dado soporte personal para la ayuda a la integración de la cabina con otros subsistemas.

La integración de todos los subsistemas la hemos realizado en Ubuntu con Tomcat 7 y Xampp 5.6.3 para los subsistemas en Java y PHP, y el servidor de desarrollo de django para el nuestro.

4.3. ENUNCIADO EJERCICIO:

Las tecnologías que se describen a continuación, se utilizan de forma conjunta (ya que son compatibles) para desplegar un proyecto Phyton.

- Django es un framework web, especializado en código Open Source, escrito en el lenguaje de programación Python y que sirve de utilidad para crear aplicaciones web de manera ágil y con menos código.

A partir de esta información, realice los siguientes apartados:

- Instale Django 1.4.7 mediante PIP (Python Package Index) en su máquina, previamente tiene que haber instalado la versión de 2.7 de Python.
- Cree proyecto por defecto en Django llamado “prueba”.

4.4. SOLUCIÓN:

Para realizar el ejercicio utilizaremos la cmd para Windows y un terminal para Linux y ejecutaremos las siguientes instrucciones.

django-admin.py startproject prueba

cd prueba

python manage.py runserver

se habla poquísimo de integración

5. GESTIÓN DEL CAMBIO, INCIDENCIAS Y DEPURACIÓN:

Para la publicación de incidencias, primeramente se empezó a usar el foro de ProjETSII, donde se creaba un post para los fallos detectados o cambios necesarios.

Más tarde, se decidió dejar de usar ProjETSII a favor de GitHub, para tener todo en un mismo sitio y para un mejor seguimiento, ya que tiene un apartado para las tareas más completo, con tipos de tareas "bug" específicas para los fallos detectados.

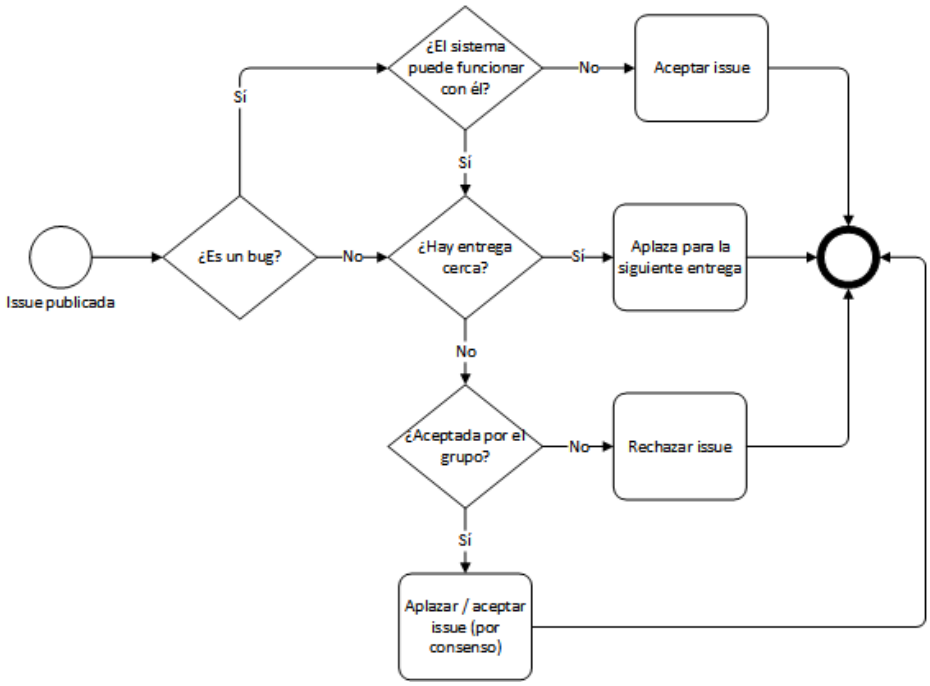
A los tipos de tareas predefinidas por GitHub (bug, duplicate, enhancement, help wanted, invalid, question y wontfix) se les ha añadido las dos siguientes:

- Agreement: para discusiones de consensos de todos los grupos
- Conflict: para conflictos entre grupos

Los únicos estados posibles definidos por GitHub son: abierto y cerrado. Así, se asume que, cuando un miembro del equipo responsable de la issue comenta, ya está en progreso, a no ser que lo deje explícito en el propio comentario.

Para publicar bugs se da una descripción de los pasos que ha seguido para llegar a donde le dio el error, y opcionalmente se adjuntan capturas y configuración de la máquina que se estaba usando. Además, se notificaba a los responsables mediante WhatsApp.

El flujo para decidir qué hacer con una issue ha sido el siguiente:



es esto propio? está sacado de algún sitio?

Para decidir si es aceptada o no por el grupo, el análisis que se realiza internamente es:

- Si es una petición de cambio: se analiza el impacto temporal y tecnológico de dicho cambio y su necesidad.
- Si es un bug: se buscan posibles soluciones y causas del fallo.

En el momento en que se acepte, rechace o aplace, se notificará en el issue correspondiente qué se ha decidido y por qué mediante un comentario.

El mecanismo seguido para la resolución de una incidencia es:

1. Con la información proporcionada en la issue, se reproduce el error y se comunica al grupo.
2. Se discuten las posibles causas del error y también las posibles soluciones que se le puede dar, y se elige una de ellas.
3. Un miembro del grupo (generalmente, el que haya hecho esa parte en concreto) será el encargado de resolver dicha incidencia. Para ello, se usa el debugger que viene integrado en el entorno de desarrollo (pycharm o eclipse).
4. Se comunica de la resolución de la incidencia y de cómo se puede evitar que vuelva a ocurrir algo similar al resto del grupo.

5.1. ENUNCIADO Y SOLUCIÓN EJERCICIO 1:

ENUNCIADO:

Realizando unas pruebas de integración se ha detectado que hay un error con el sistema de autenticación: al intentar obtener la información de un usuario del sistema mediante una llamada a su API, el sistema de cabina muestra un error relacionado con el JSON devuelto por autenticación.

Suponiendo que se tiene acceso al repositorio compartido, se pide: realizar los pasos necesarios según lo descrito anteriormente para resolver dicha incidencia.

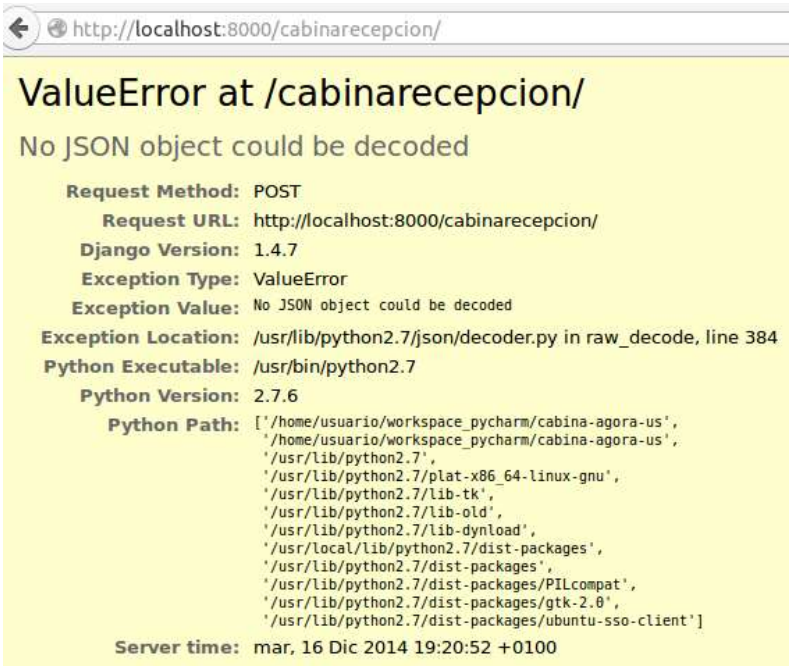
Datos proporcionados para resolver el problema:

- Captura de RESClient sobre el fallo

- API proporcionada por el subsitema de autenticación

Recurso	Descripción	Parámetros	Respuesta	Ejemplo de respuesta
getUser	Obtiene un usuario del sistema, incluyendo sus datos. Este método solo puede ser usado por subsistemas locales (acceso mediante localhost).	• user: nombre del usuario	JSON con la lista de usuarios. Cada usuario incluye los campos "username", "password" y "email", "genre" y "autonomous_community".	{ "username" : "johndoe", "password" : "foo_bar", "email" : "mail@example.com", "genre" : "Masculino", "autonomous_community" : "Madrid" }

- Error del sistema de cabina de recepción



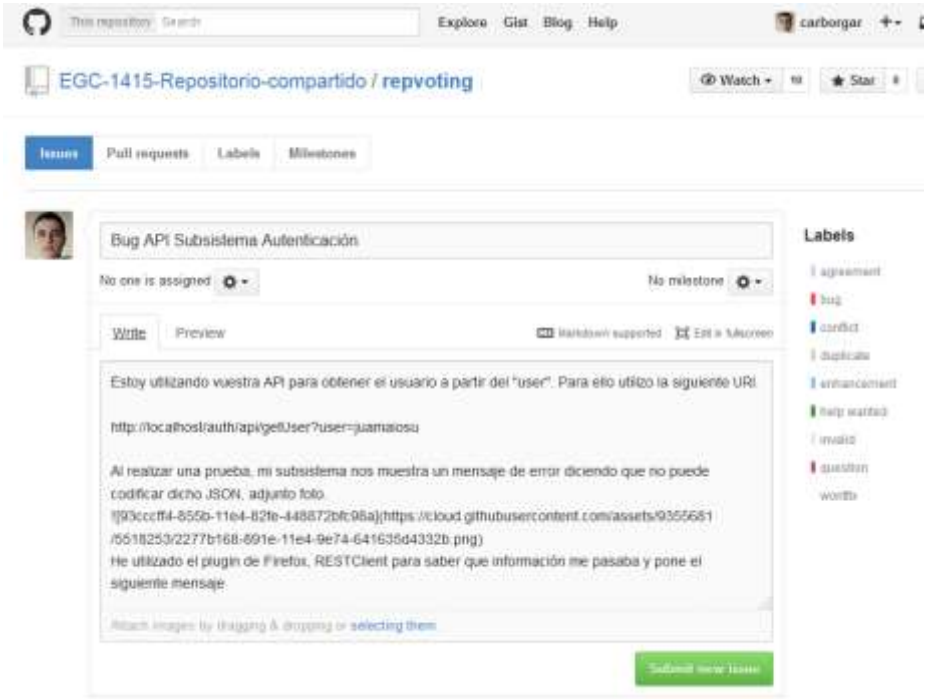
SOLUCIÓN:

finalmente usáis gitlab o github, los dos??

1. Iniciar sesión en GitHub y entrar al apartado Issues

2. Hacer click en “new issue” y rellenar título y cuerpo (adjuntando las imágenes proporcionadas y una descripción de qué pasa.

3. Agregar a la tarea la etiqueta (label) “bug”



4. Publicar la issue y esperar a la respuesta del grupo encargado de resolverla

5.2. ENUNCIADO Y SOLUCIÓN EJERCICIO 2:

ENUNCIADO:

Se ha notificado en GitHub un error relacionado con el subsistema de cabina de votación (ver imagen).

Suponiendo que se tiene acceso al repositorio compartido, se pide: realizar los pasos necesarios según lo descrito anteriormente para resolver dicha incidencia.

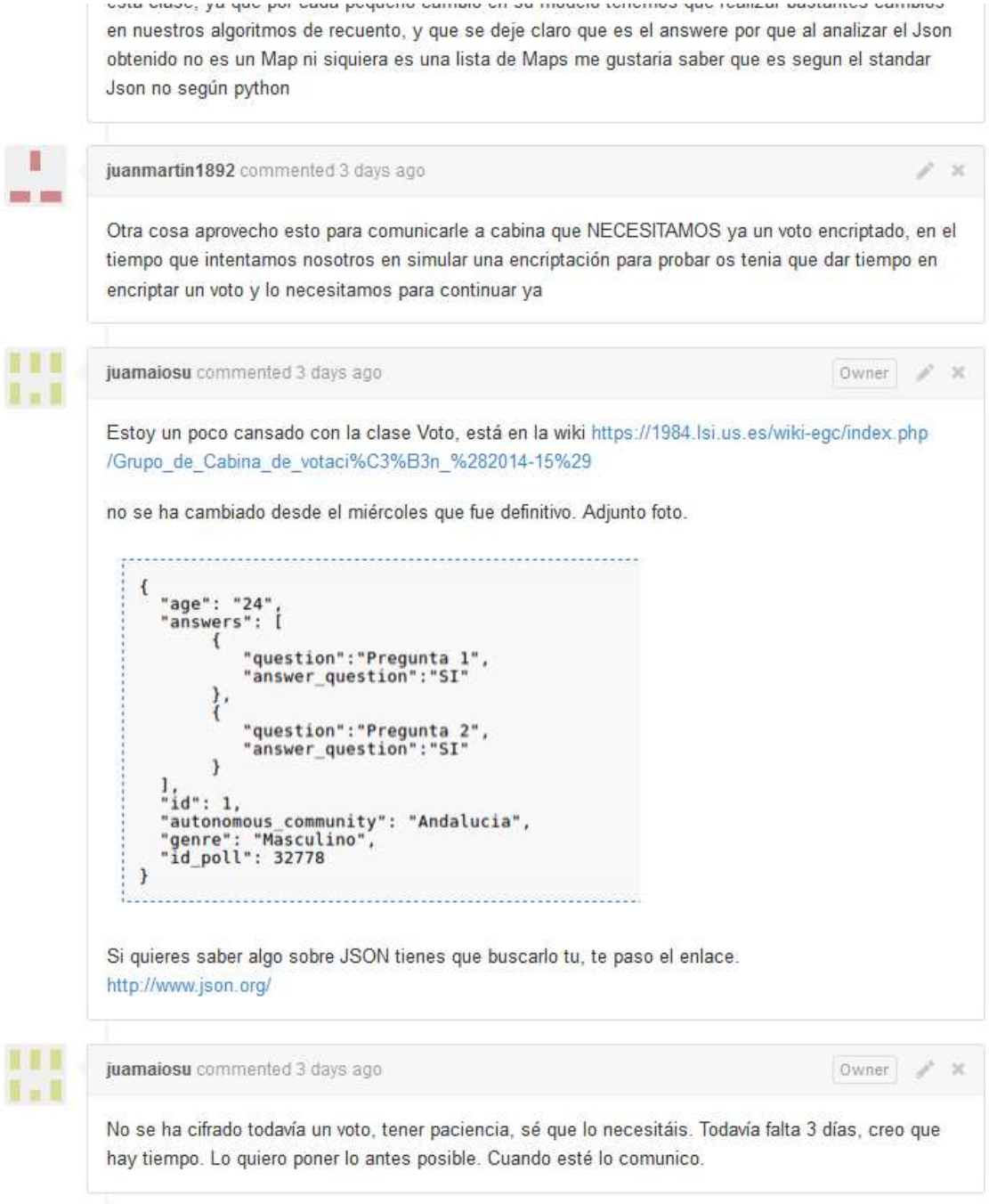
Datos:

- Incidencia en GitHub

SOLUCIÓN:

1. Identificar el tipo de problema: se trata de una petición de cambio (piden que se dé una versión definitiva de la clase)
2. Ver qué tiempo queda hasta la entrega: en este caso, unos 3 días (se considera entrega lejana)
3. Comunicar dicha incidencia al grupo y evaluarla

4. Comentar en la propia incidencia la respuesta del grupo (en este caso, indicar dónde puede ver la clase que pedía y explicar los motivos. En este caso no es necesario hacer debug ni cambios en nada, sólo informar.



Si hubiera que hacer debug, simplemente habría que asignar a algún miembro para que revise y comunicar en GitHub el proceso.

6. GESTIÓN DE LIBERACIONES, DESPLIEGUE Y ENTREGAS:

Los elementos del proyecto entregables son el código del subsistema y las distintas versiones del script de despliegue.

No se sigue ninguna política de identificación de ítems.

La publicación liberación y entrega, a través del github compartido actualmente y antes se realizaba a mano a través de pendrives, etc.

No hay roles en la entrega.

6.1. ENUNCIADO EJERCICIO:

Seguir los pasos del manual que podemos visualizar aquí:

https://1984.lsi.us.es/wiki-egc/index.php/Grupo_de_Cabina_de_votaci%C3%B3n_(2014-15)#Instalaci.C3.B3n_del_subsistema

y ejecute el proyecto.

6.2. SOLUCIÓN:

INSTALACIÓN DEL SUBSISTEMA EN WINDOWS

1. Instalar Python 2.7.7 (link)

2. Instalar Setuptools (link) teniendo en cuenta python 2.7.7 y 64 o 32 bits

3. Ir a la ruta de instalación de scripts python (por defecto: C:\Python27\Scripts) desde el cmd y ejecutar:

```
easy_install pip

pip install Django==1.4.7

pip install djangorestframework==2.4.3

pip install rsa==3.1.4

pip install django-cors-headers==0.13

pip install requests==2.4.3

pip install pycrypto==2.6.1
```

4. Ir a la carpeta donde esté el proyecto por cmd y ejecutar: manage.py runserver

5. Abrir el navegador en 127.0.0.1:8000

INSTALACIÓN DEL SUBSISTEMA EN UBUNTU

1.- Crear carpeta llamada cabina-integracion

mkdir cabina-integracion

2.- Descargar los scripts para Linux:

install.sh run.sh

3.- Mover los scripts descargados a la carpeta cabina-integracion, creada en el punto 1

4.- Dar permisos de ejecución a los scripts:

chmod +x *.sh

5.- Copiamos dentro de la carpeta cabina-integracion la última versión de cabina cabina-agora-us. La estructura que va a tener la carpeta cabina-integracion es la siguiente:

cabina-integracion:

- cabina-agora-us
- install.sh
- run.sh

6.- Ejecutar scripts:

./install.sh

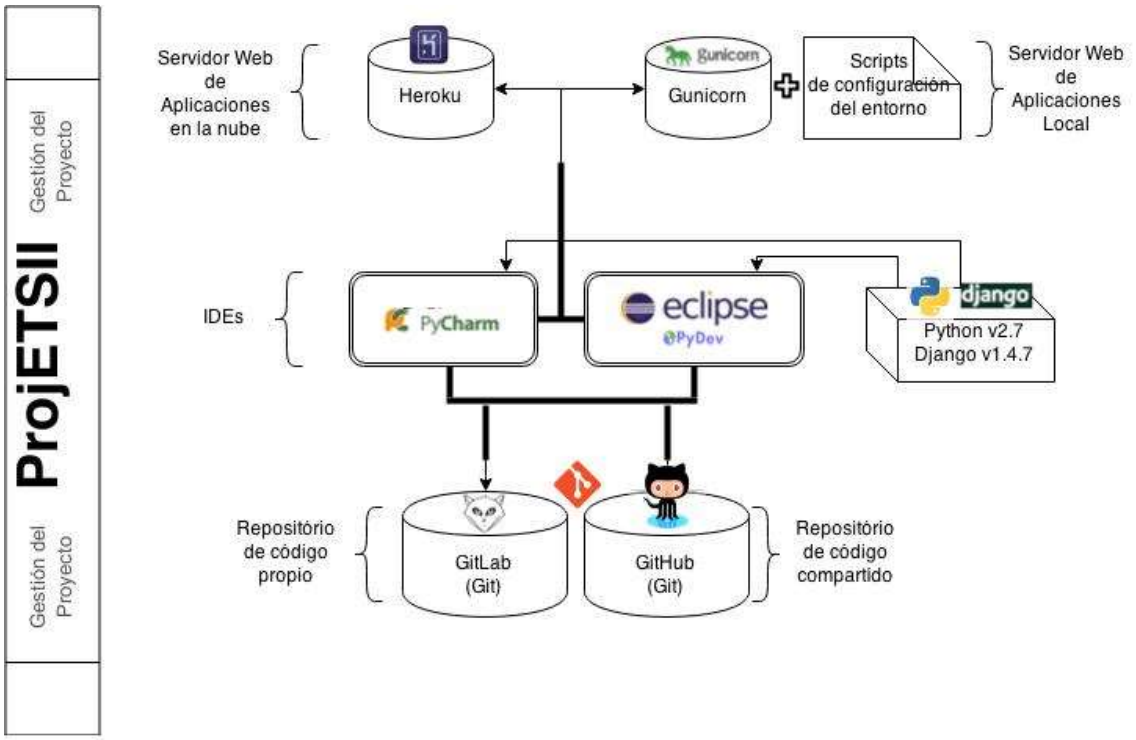
./run.sh

El primer script (install.sh) instala y arranca el proyecto. El segundo (run.sh) solo lo arranca. Por tanto la primera vez que ejecutéis el proyecto después de descargarlo ejecutad el script install.sh, y el resto de las veces el run.sh

Tras la ejecución de cada script, saldrá el siguiente mensaje: ================= OK ================== y el terminal se quedara "esperando". Llegados a este punto tendremos el servidor arrancado. Para parar el servidor presionamos Ctrl + C.

"Texto extraído de la siguiente URL: https://1984.lsi.us.es/wiki-egc/index.php/Grupo_de_Cabina_de_votaci%C3%B3n_(2014-15)#Instalaci.C3.B3n_del_subsistema"

7. MAPA DE HERRAMIENTAS:



Herramienta	Descripción
Python v2.7	Lenguaje de desarrollo
Django v1.4.7	Web Framework basado en Python
PyCharm	Entorno de Desarrollo
Eclipse Luna (PyDev)	Entorno de Desarrollo con el plugin de PyDev
GitLab	Repositorio Git
Projetsii	Gestión del proyecto
Gunicorn	Python WSGI HTTP Server for UNIX
Heroku	Plataforma Paas en la nube que soporta diferentes lenguajes de programación

En nuestro subsistema Cabina de votación decidimos trabajar con el framework Django en su versión 1.4.7 ejecutado sobre Python 2.7 a través de los IDEs PyCharm y/o Eclipse junto a su plugin para Python PyDev.

Para todo lo relacionado con la gestión y seguimiento del proyecto de nuestro subsistema, decidimos crearnos un repositorio Git a través de la plataforma de GitLab, la cual fue la que más se adecuó a nuestras necesidades, y Telegram para la comunicación.

Para desplegar el proyecto hemos usado Gunicorn para hacerlo en local junto con un par de scripts, para sistemas Windows o Unix, para configurar el entorno de trabajo. Pero hemos tenido problemas con Gunicorn y el CSS de le aplicación, por lo que se ha optado por desactivarlo para este entregable.

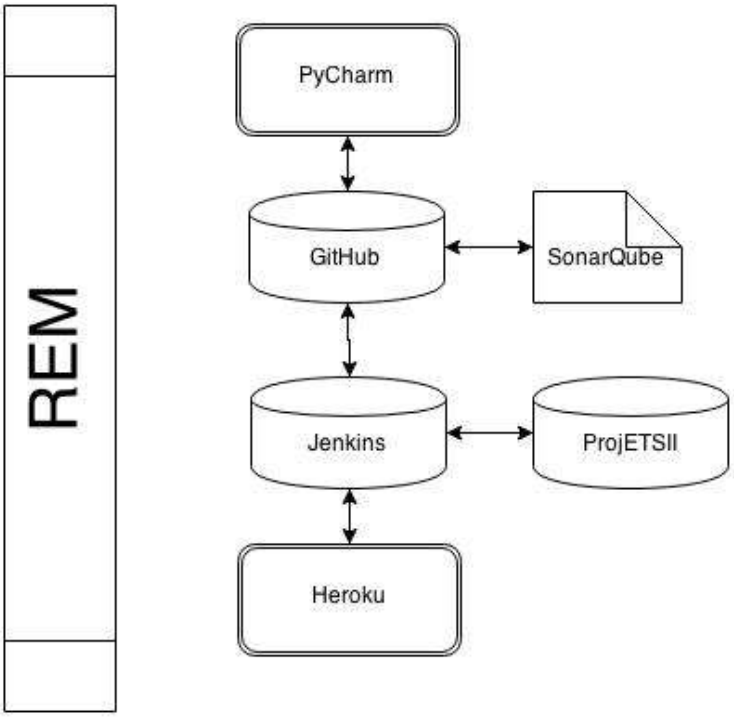
De cara a la integración con los demás subsistemas nos vimos atados finalmente a crear un repositorio Git común en GitHub en el que cada subsistema tendría una rama donde desarrollar su código, además de usar Heroku como servidor de aplicaciones web para desplegar nuestro subsistema y agilizar el proceso de integración que debemos llevar a cabo. Finalmente, Heroku no se ha ido actualizando, ya que con el script no tenía sentido.

7.1. ENUNCIADO EJERCICIO:

Cree un mapa de herramientas para un sistema en el que se tiene:

PyCharm (IDE), GitHub (Repositorio de código), SonarQube (Calidad del código), Jenkins (Servidor de integración continua), Heroku (Construcción de proyectos), ProjETSII (Gestor de incidencias) y REM (Gestión de Requisitos).

7.2. SOLUCIÓN:



8. CONCLUSIONES:

A la vista de todo lo ocurrido durante el transcurso de la asignatura y el desarrollo de Ágora US, y más concretamente, el subsistema "Cabina de votación", podemos obtener varias conclusiones.

En primer lugar, analizando el nefasto resultado de la organización de los 11 grupos encargados de los 11 subsistemas que componen Ágora US, queda de manifiesto que es necesario establecer una adecuada política de gestión de grupos y tareas para que un proyecto pueda concluir de manera exitosa. Por tanto, a la hora de afrontar un proyecto, más allá de su magnitud, es de vital importancia utilizar de forma eficiente herramientas de gestión de proyectos (como Redmine) y nombrar a un Jefe de Proyecto que se encargue de coordinar cada grupo personalmente y que esté al tanto de todos los cambios.

En segundo lugar, si se pretende abordar un proyecto compuesto por varios subsistemas, es importante dedicar el tiempo necesario a definir la comunicación entre cada subsistema. Si esto no se lleva a cabo, el momento de la integración será totalmente caótico (hecho que ha ocurrido en la asignatura) y se deberá realizar varias veces el mismo trabajo. Es importante en este punto la figura del Jefe de Proyecto, citada en la primera conclusión, para que informe a cada subsistema de los cambios que han ido ocurriendo y cómo les afecta.

En tercer lugar, es necesaria una política de gestión del código fuente eficiente, clara y **compartida** por todos los grupos. Todos los grupos (especialmente los que, por necesidades de la aplicación, deben comunicarse directamente) deben acceder a todo el código fuente disponible. Esto nos obliga a definir una política de aceptación de "commits" cuando en el proyecto intervienen un gran número de personas, como es el caso de EGC (más de 80 personas). Este punto enlaza directamente con la siguiente conclusión: la elección de la tecnología.

En cuarto lugar, le elección de la tecnología es una decisión importante que no debe tomarse de forma precipitada y que marcará nuestro proyecto durante todo el desarrollo. Aunque es cierto que no es necesario utilizar la misma tecnología para desarrollar todos los subsistemas, sí es conveniente por los siguientes motivos:

- Facilita enormemente la integración.
- Mejora la gestión del código fuente, en caso de que se opte por un repositorio común.

Por tanto, en la medida de lo posible, se deben escoger tecnologías similares.

En quinto lugar, y enlazando con todo lo anterior, se hace indispensable una gestión de la integración. El no tener un servidor de integración continua como Jenkins (en parte porque todo se ha hecho en local) ha provocado que la integración de los subsistemas no haya dado buenos resultados. Todas las integraciones que se han realizado durante la asignatura han sido manuales, mediante pendrives y similares. Esto, unido a todo lo anteriormente mencionado, ha provocado que un proyecto aparentemente simple, como es Ágora US, se convierta en proyecto muy difícil de gestionar y de llevar a cabo.

Me gustan las conclusiones que denotan aprendizaje pero la documentación es mas bien pobre y podría mejorarse mucho más