MEMORIA EGC

SUBSISTEMA RECUENTO

Indice

Índice de imágenes

1.Miembros del grupo

MIEMBRO	ROL
José Cerro Daza	Gestor de la Configuración
Adrián Lagomazzini Mellado	Secretario
Manuel Humanes Ángel	Gestor de la Configuración
Pedro Grau Cabrera	Gestor de la Configuración
Jorge Romero Iglesias	Gestor de la Configuración
Antonio Toro Valle	Gestor de la Configuración
Ramón Ponce	Gestor de la Configuración
Juan Martín Pérez	Secretario
Nacho Baena	Gestor de la Configuración
Mª Concepción Gimeno Pastor	Jefa de Proyecto

2.Resumen

En el presente documento se hará una breve introducción sobre la gestión de la configuración adoptada para la realización del proyecto AGORA US, un proyecto de software libre desarrollado por alumnos de la asignatura “Evolución y Gestión de la Configuración”, de la titulación “Ingeniería del Software” de la Universidad de Sevilla.

Para la realización de este proyecto se ha procedido a la subdivisión del sistema en 11 subsistemas, estos son “Autenticación”, “Creación/administración de votaciones”, “Sistema de modificación de resultados”, “Almacenamiento de votos”, “Deliberaciones”, “Recuento”, “Creación/administración de censos”, “Frontend de resultados”, “Visualización de resultados”, “Verificación” y “Cabina de votación”

En este documento se hará la explicación de la gestión del grupo de trabajo encargado del subsistema “Recuento”.

La integración de los distintos subsistemas se ha coordinado mediante diversas reuniones realizadas a lo largo del cuatrimestre.

El siguiente documento se estructura en las siguientes secciones:

Introducción: En la introducción se explicará de manera más extendida las funciones del proyecto ÁGORA US, el lugar que ocupa el subsistema recuento en el marco de la aplicación y una breve descripción de cómo nos relacionamos con los subsistemas adyacentes.

Gestión del código fuente: En esta sección explicaremos los sistemas de gestión del código fuente, políticas adoptadas y las herramientas utilizadas

Gestión de la construcción e integración continua: En esta sección explicamos las herramientas y formas usadas en el proyecto para gestionar la construcción del proyecto y realizar integración continua del proyecto

Gestión del cambio, incidencias y depuración: En esta sección desarrollamos las políticas adoptadas a la hora de gestionar incidencias o cambio en el proyecto

Gestión de liberaciones, despliegue y entregas: En este apartado se hablará del protocolo definido a la hora de la liberación, despliegue y realización de entregas en nuestro proyecto

Mapa de herramientas: El mapa de herramientas describe las herramientas que se han utilizado para la realización del proyecto y cómo interactúan entre ellas

Conclusiones: En las conclusiones se reflejan las ideas finales extraídas del documento

3.Introducción

Este proyecto consiste en el desarrollo de un programa que permita a cualquier organización realizar procesos electorales de forma segura, flexible y transparente, basado en el sistema existente Agora Voting

El sistema gestiona las votaciones desde el proceso de gestión de los censos, realización de las votación y visualización de los resultados de cada votación, las votaciones se harán de forma segura con uso de encriptación para los votos basados en autoridades que serán las encargadas de salvaguardar las claves para la encriptación y des encriptación de los votos, el sistema está limitado a votaciones en formato referéndum, de respuestas 'si' o 'no' en múltiples preguntas, el proyecto se ha realizado bajo unas políticas de software libre.

El subsistema "Recuento" se encargará de contabilizar los votos de una votación determinada.
Para realizar el recuento tendrá que pedir los votos al subsistema "Almacenamiento de votos" y deberá lanzar la tarea de recuento sincronizando las diferentes autoridades, devolviendo finalmente un Json con los resultados al subsistema de frontEnd de resultados.

Ilustración 1 Subsistemas integrados

Por lo tanto, nuestro subsistema se comunicará con el subsistema de "Almacenamiento de votos" para extraer los votos almacenados en la Base de Datos. Dichos votos estarán cifrados, para evitar la manipulación y modificación de las votaciones, por lo que nuestro subsistema tendrá que descifrar esos votos y confirmar la integridad de los votos usando la librería proporcionada por el subsistema de "Verificación" para posteriormente realizar el recuento.

Una vez realizado el recuento proporcionaremos el resultado obtenido al subsistema "Frontend" que se encargará de la visualización de los resultados obtenidos.

Concretamente, el subsistema "Recuento" funciona de la siguiente manera:

La persona responsable de la votación, una vez determina que ésta ha terminado,

procede a mostrar los resultados de la votación, para ello hará uso de un botón en la interfaz de administración de AgoraUS.

Dicho botón tiene la función de llamar a nuestro subsistema, con la ID de la votación de la que se quiere hacer el recuento, de forma que con esa ID nuestro subsistema hace llamada al subsistema de "Almacenamiento de votos", haciendo que este subsistema mande una lista cifrada de votos, que nosotros, gracias a la librería del subsistema "Verificación", desciframos y comprobamos la integridad de estos votos.

Una vez descifrados, recorremos la lista de estos votos y vamos contando el número de resultados positivos y negativos de cada pregunta.

Tras esto nuestro subsistema creará un JSON que está basado en una entidad creada por nosotros, llamada "Resultado", cuyos atributos son "pregunta", "numeroSi" y "numeroNo", como se ve en el siguiente ejemplo:

[{"pregunta":"¿es útil?","numeroSi":1,"numeroNo":2},{"pregunta":"¿llegará tarde?","numeroSi":2,"numeroNo":0},{"pregunta":"¿cambiarías algo?","numeroSi":2,"numeroNo":1}]

El resto del proceso a partir de este punto hasta la visualización del proyecto no compete a nuestro subsistema por lo que no procederemos a su explicación.

4.Gestión de código fuente

qué es STS??

4.1. Funcionamiento del cliente Git dentro del STS

Para la realización del presente trabajo, en el cual estamos involucrados diez personas, se hace necesario el uso de una herramienta de gestión de código fuente.

Esto proporcionará al proyecto:

- Un mecanismo de almacenamiento de los elementos que deben gestionar.
- Un registro histórico de las acciones realizadas con cada elemento o conjunto de elementos.
- La posibilidad de realizar cambios sobre los elementos almacenados.

En nuestro caso, se ha utilizado para dicha gestión el sistema de control de versiones Git, concretamente la herramienta seleccionada ha sido GitHub, ya que esta nos permite crear cuentas de forma fácil y gratuita, tiene una interfaz bastante intuitiva y posee funcionalidades como si de una red social se tratase, por lo tanto es la que mejor se adapta a las necesidades del proyecto.

Una vez elegida la herramienta, se hace estrictamente necesaria la adjudicación de una persona dentro del grupo, que en este caso es Jorge Romero, como responsable de la gestión del código fuente.

Politicas de Gestion del código:

Se definen principalmente tres políticas básicas a la hora de trabajar con el sistema de gestión de versiones git:

Política de realización de commits:

Solo se realizarán push de segmento de código que reflejen un cambio lógico en la funcionalidad del código, definimos cambio lógico como un cambio que afecte a la funcionalidad del algoritmo, y que pueda ser justificado como cambio relevante para el algoritmo.

Cada commit debe estar justificado de manera completa y detallada en el comentario de dicho commit, el título de dicho comentario debe reflejar un breve resumen del cambio realizada y el cuerpo del comentario debe de responder a las preguntas ¿Por que se ha realizado el cambio?¿Como se ha realizado dicho cambio? y ¿Donde se ha realizado dicho cambio?

Política de Ramas:

usar mejor estructuración de las partes (e.g. subseción)

Se harán uso de tres tipos distintos de ramas, una rama master encargada de contener el código más estable de la aplicacion, una rama que nace de la rama master, para contener el código a desarrollar llamada develop, y unas ramas work,

en estas ramas se guardará el código que un desarrollador está modificando en cada momento.

Solo cuando el código se considere lo suficientemente estable se podrá subir a la rama master y el encargado de esta actividad será Jorge Romero debido a su rol de Gestor del Código

Cada vez que un desarrollador quiera realizar un cambio en el algoritmo debe de crear una rama desde la rama desarrollo y al finalizar su trabajo deber unirla a la rama desarrollo

OK, buena ilustración.

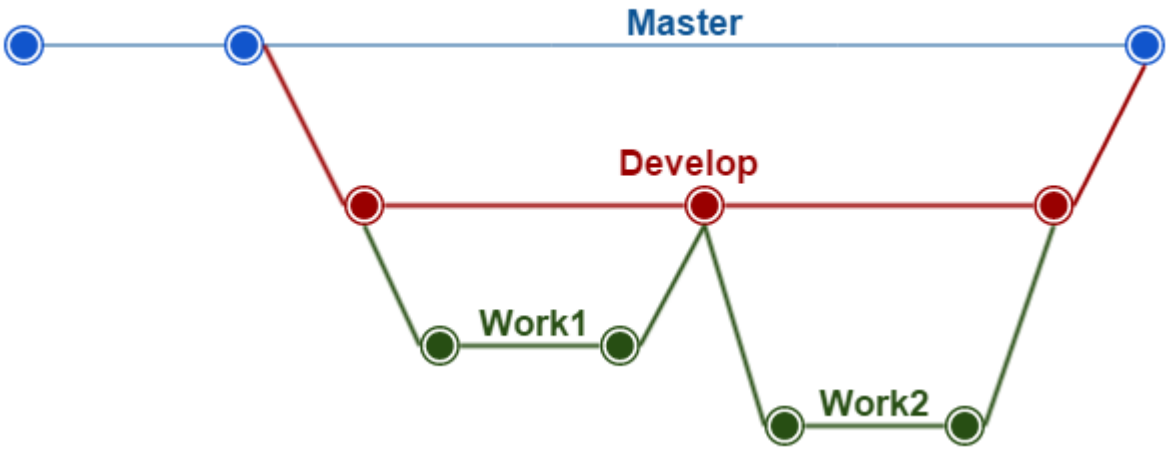


Ilustración 2 Ramas

Por problemas de tiempo y debido a la poco volumen de código del proyecto finalmente no se a aplicado la política de ramas definidas y se decidió desarrollar directamente sobre la rama master, entendiendo que esta no es una buena política de gestión de las ramas.

(no es una buena escusa... podría haberse más volumen de código pues eso siempre es posible..;-)

Política de resolución de conflictos:

En caso de darse un conflicto la responsabilidad de resolver este conflicto recae en el desarrollador que halla causado el conflicto con la ayuda de Jorge Romero responsable de la gestión del código

no es muy frecuente tener a una persona siempre encargada de la gestión del código fuente. Suele ser más descentralizado

Cliente git:

En nuestro caso se ha decido hacer uso del cliente de Git que contiene la herramienta Spring Tool Suite (STS), en primer lugar para aprovechar la compatibilidad con el resto de las herramientas del propio STS y por que nos parecio un sistema más simple e intuitivo debido a que se realiza de manera gráfica en lugar del uso de comandos:

Se debería usar en línea de comandos primero y luego, si fuera del caso, usar la herramienta gráfica

Uso del cliente git del sts.

Pasa importar el proyecto desde el sts haciendo uso del cliente git asociado deberemos seguir los siguientes pasos:

1º Importar proyecto, "file --> import".

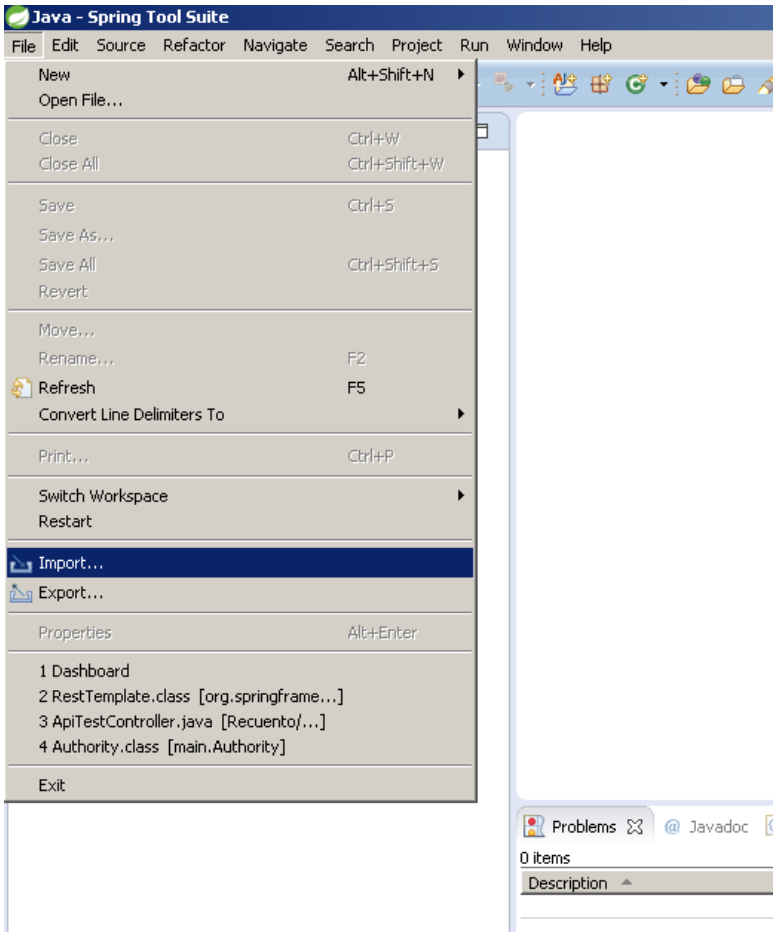


Ilustración 3 Importar proyecto

2º Indicar que vamos a hacer uso de git "Git/ Projects from Git"

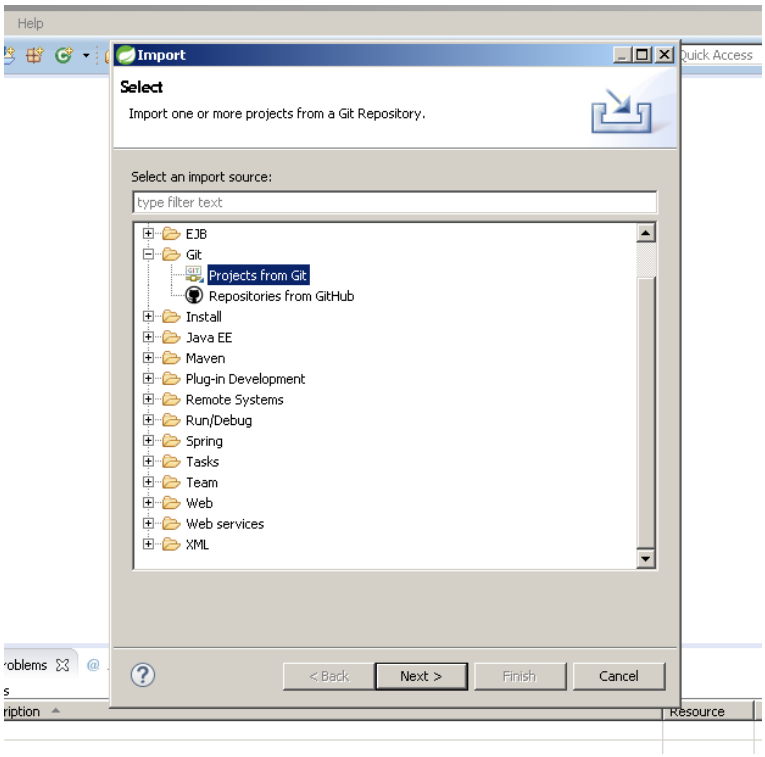


Ilustración 4 Seleccionar Proyecto

podríamos hacer uso directamente de la herramienta de gitHub ya que usamos como repositorio GitHub pero para hacerlo de una manera más general hemos hecho uso del cliente git

3º Indicar que vamos a importarlo mediante una uri

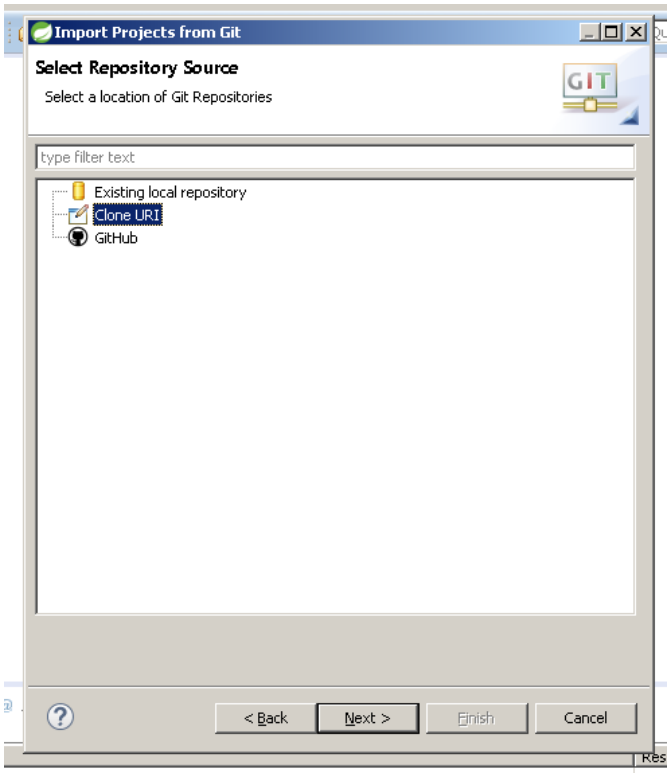


Ilustración 5 Clone URI

4º Introducir los datos necesarios para la conexion con el servidor que son, la uri desde la que deseamos importar y los datos de autenticación de nuestro usuario en git en gitHub.

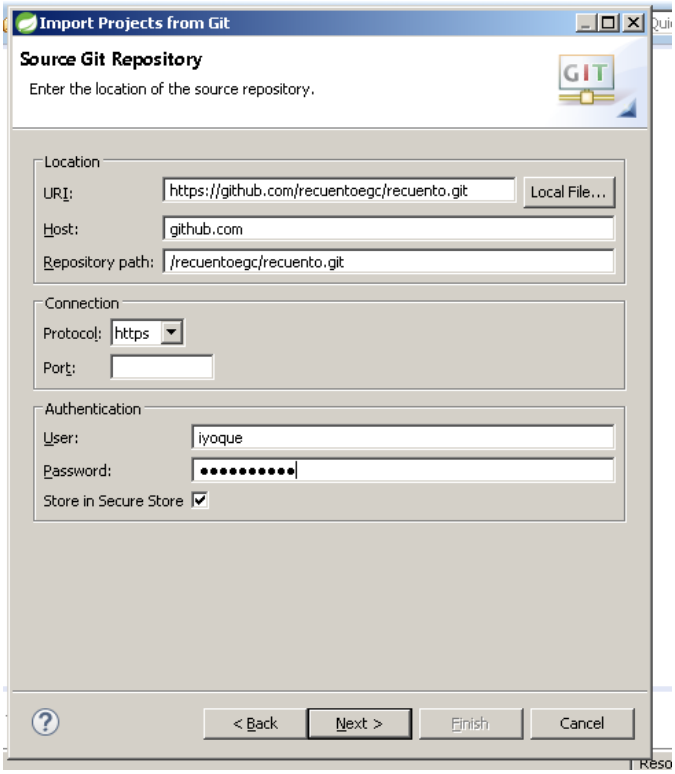


Ilustración 6 Autenticar

5º Indicamos desde que rama deseamos importar

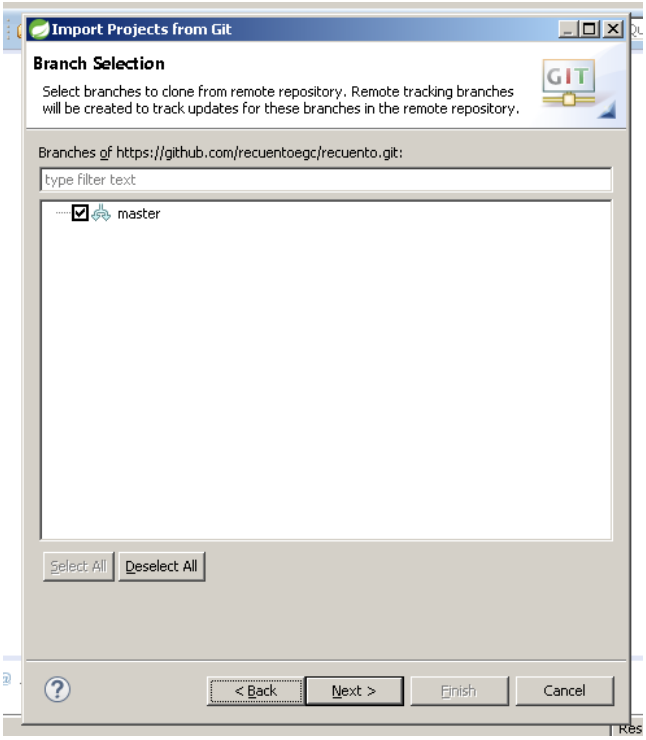


Ilustración 7 Importar Rama

6º Y por último indicamos donde se encuentra nuestro repositorio local

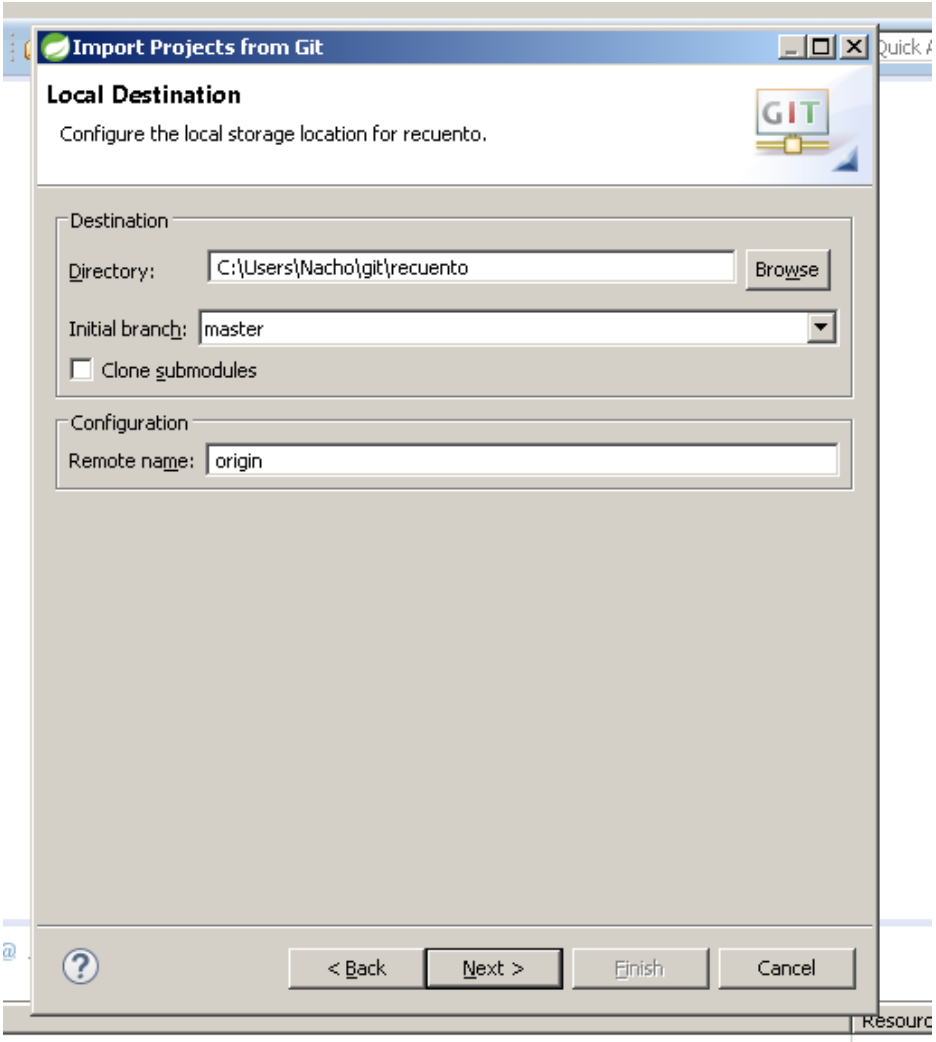


Ilustración 8 Local Destination

Ya tendríamos importado nuestro proyecto en nuestro ordenador.

4.2. Ejercicio repositorio Git

Con lo que habéis planteado de las ramas sería mucho mejor plantear un ejercicio que haga realmente lo que habéis descrito arriba

Se pretende crear un repositorio git usando Github, en el cual se debe crear al menos un archivo llamado "Ejercicio.txt", dicho archivo se introducirá en el repositorio en el primer "commit" conteniendo el texto inicial: "Este es un ejercicio sobre git". Seguidamente se realizará un "push" en dicho repositorio y ver reflejado los cambios en la plataforma GitHub.

Una vez iniciado el repositorio y realizado ese primer "commit" y "push", se debe crear una rama "desarrollo", en esta rama se debe modificar el archivo "ejercicio.txt" editando el texto que contenga por este otro: "Este es un ejercicio sobre git en el cual se ha creado una rama y se modificó este archivo". Luego se hará un "push" y se observará el efecto en Github. Obviamente solo debe reflejarse dicho cambio en la rama "desarrollo" y en la rama "master" no debe haberse modificado el archivo.

Finalmente, se visualizarán los commits realizados con el comando pertinente, y seguidamente, se debe hacer "merge" de la rama "desarrollo" a la rama "master" y comprobar que el archivo "ejercicio.txt" corresponde con la última modificación.

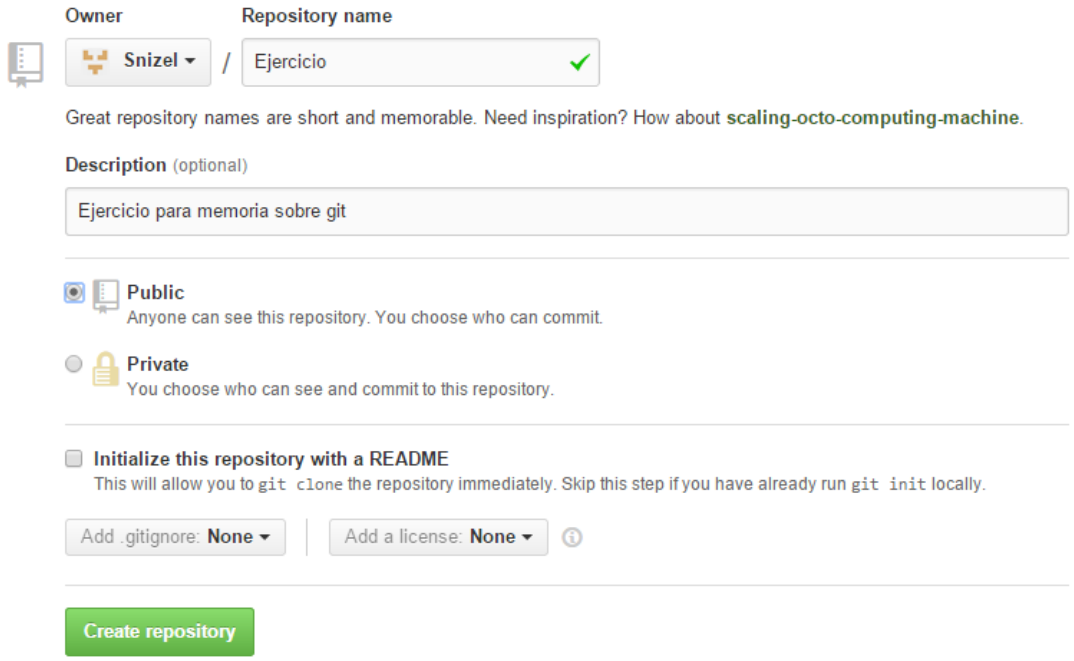


Ilustración 9 Commits

Una vez creado e iniciado correctamente el repositorio, estableceremos la configuración global con los siguientes comandos, para que así git sepa el nombre y el email del autor del cambio.

```
ubuntu@ubuntu-VirtualBox:~$ git config --global user.name "Ramon"
ubuntu@ubuntu-VirtualBox:~$ git config --global user.email demonsnizel@gmail.com
```

Ilustración 10 Configuración global

Una vez toda la configuración esta lista, pasamos a clonar el repositorio que creamos en Github, la URL es ofrecida por Github, la cual es:

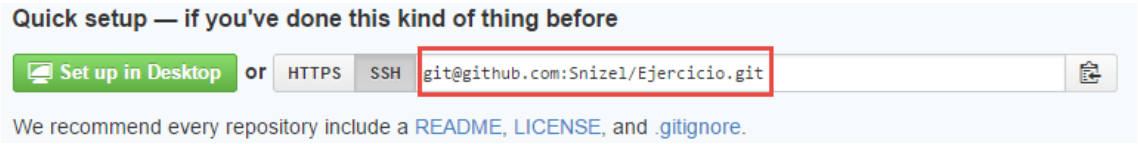


Ilustración 11 Clonar repositorio

Para ello usamos el comando "git clone":

```
ubuntu@ubuntu-VirtualBox:~/Escritorio$ git clone git@github.com:Snizel/Ejercicio
.git
Cloning into 'Ejercicio'...
warning: You appear to have cloned an empty repository.
```

Ilustración 12 Git Clone

Como se puede ver nos ha creado una carpeta Ejercicio en el escritorio, y es en dicha carpeta donde se encuentra el contenido del repositorio, en este momento, debe estar vacío.

Ilustración 13 Carpeta

Entramos a dicho directorio por terminal, y usando el comando "git status" podemos conocer el estado del repositorio, y como podemos observar, nos indica que es necesario hacer un "commit" inicial, como debe ser.

```
ubuntu@ubuntu-VirtualBox:~/Escritorio$ ls
Ejercicio
ubuntu@ubuntu-VirtualBox:~/Escritorio$ cd Ejercicio/
ubuntu@ubuntu-VirtualBox:~/Escritorio/Ejercicio$ git status
# On branch master
#
# Initial commit
#
nothing to commit (create/copy files and use "git add" to track)
ubuntu@ubuntu-VirtualBox:~/Escritorio/Ejercicio$
```

Ilustración 14 Git status

Como se puede apreciar nos indicar que el repositorio clonado esta vació, ya que lo acabamos de crear sin añadir ningún archivo.

Por tanto, haremos nuestro primer commit y push creando primero el archivo requerido así como su contenido.

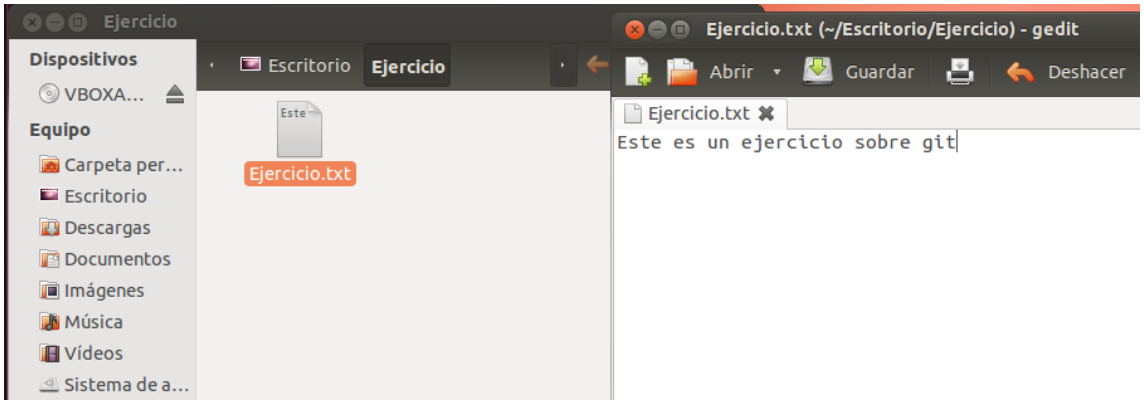


Ilustración 15 Commit-Push

Podemos observar que git ha detectado el cambio, y nos comunica que hay un archivo que no está siguiendo:

```
ubuntu@ubuntu-VirtualBox:~/Escritorio/Ejercicio$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       Ejercicio.txt
#       Ejercicio.txt~
nothing added to commit but untracked files present (use "git add" to track)
ubuntu@ubuntu-VirtualBox:~/Escritorio/Ejercicio$
```

Ilustración 16 Cambios

Para que siga dicho archivo usaremos el comando “git add”:

```
ubuntu@ubuntu-VirtualBox:~/Escritorio/Ejercicio$ git add Ejercicio.txt
ubuntu@ubuntu-VirtualBox:~/Escritorio/Ejercicio$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   Ejercicio.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       Ejercicio.txt~
ubuntu@ubuntu-VirtualBox:~/Escritorio/Ejercicio$
```

Ilustración 17 Git add

Ahora podemos hacer nuestro primer commit con dicho archivo:

```
ubuntu@ubuntu-VirtualBox:~/Escritorio/Ejercicio$ git commit -a
[master (root-commit) 77f9f94] Creando arhivo Ejercicio.txt pedido en el ejercic
io
 1 file changed, 1 insertion(+)
 create mode 100644 Ejercicio.txt
```

Ilustración 18 Primer Commit

Ahí se observa el título usado en el commit, el ID del commit, y que se realizó a la rama master.

Finalmente hacemos push al repositorio y observaremos los cambios.

```
ubuntu@ubuntu-VirtualBox:~/Escritorio/Ejercicio$ git push origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 325 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
```

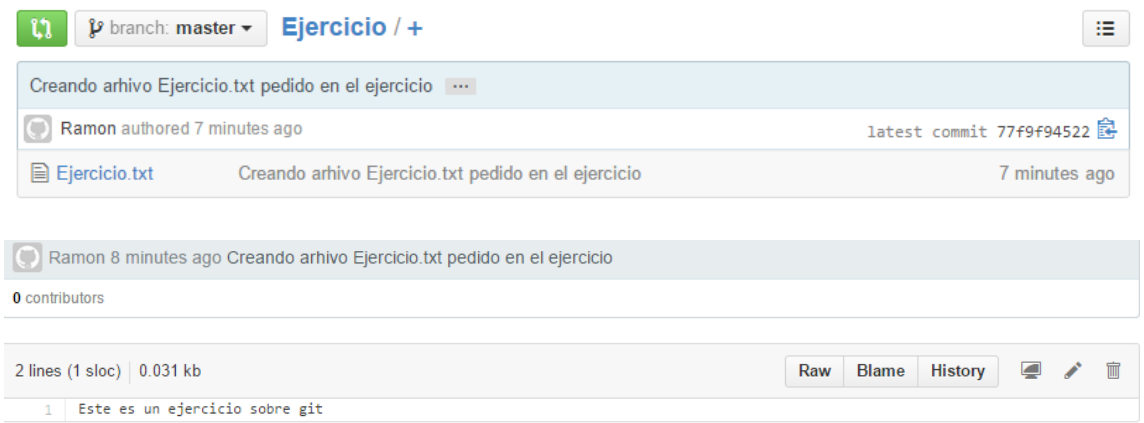


Ilustración 19 Push Repositorio

Seguidamente, creamos la rama “desarrollo”, para ello usamos el comando “git branch desarrollo”:

```
ubuntu@ubuntu-VirtualBox:~/Escritorio/Ejercicio$ git branch desarrollo
ubuntu@ubuntu-VirtualBox:~/Escritorio/Ejercicio$ git branch
  desarrollo
* master
ubuntu@ubuntu-VirtualBox:~/Escritorio/Ejercicio$ git checkout desarrollo
Switched to branch 'desarrollo'
ubuntu@ubuntu-VirtualBox:~/Escritorio/Ejercicio$ git branch
* desarrollo
  master
```

Ilustración 20 Git brach desarrollo

Cambiamos a dicha rama, y para enviar la rama al servidor usamos el comando “git push origin desarrollo” y observamos que se realizó correctamente.

```
ubuntu@ubuntu-VirtualBox:~/Escritorio/Ejercicio$ git push origin desarrollo
Total 0 (delta 0), reused 0 (delta 0)
To git@github.com:Snizel/Ejercicio.git
 * [new branch]      desarrollo -> desarrollo
ubuntu@ubuntu-VirtualBox:~/Escritorio/Ejercicio$ git branch -a
* desarrollo
  master
  remotes/origin/desarrollo
  remotes/origin/master
```

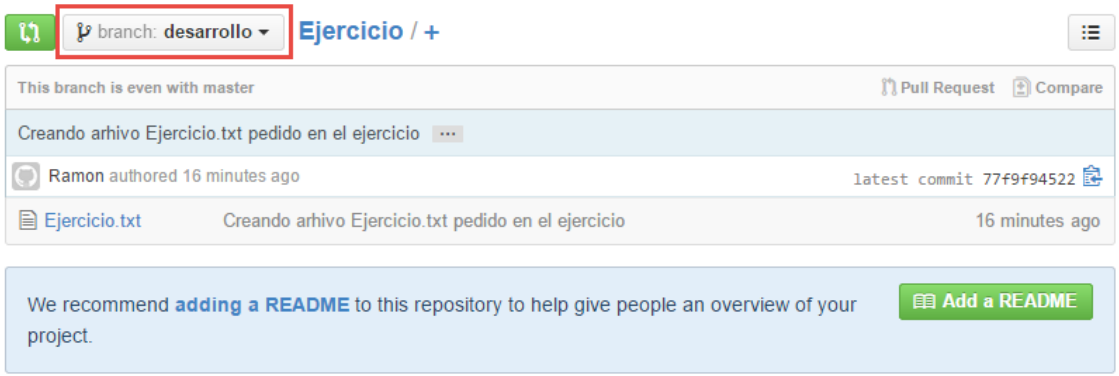


Ilustración 21 Git push origin desarrollo

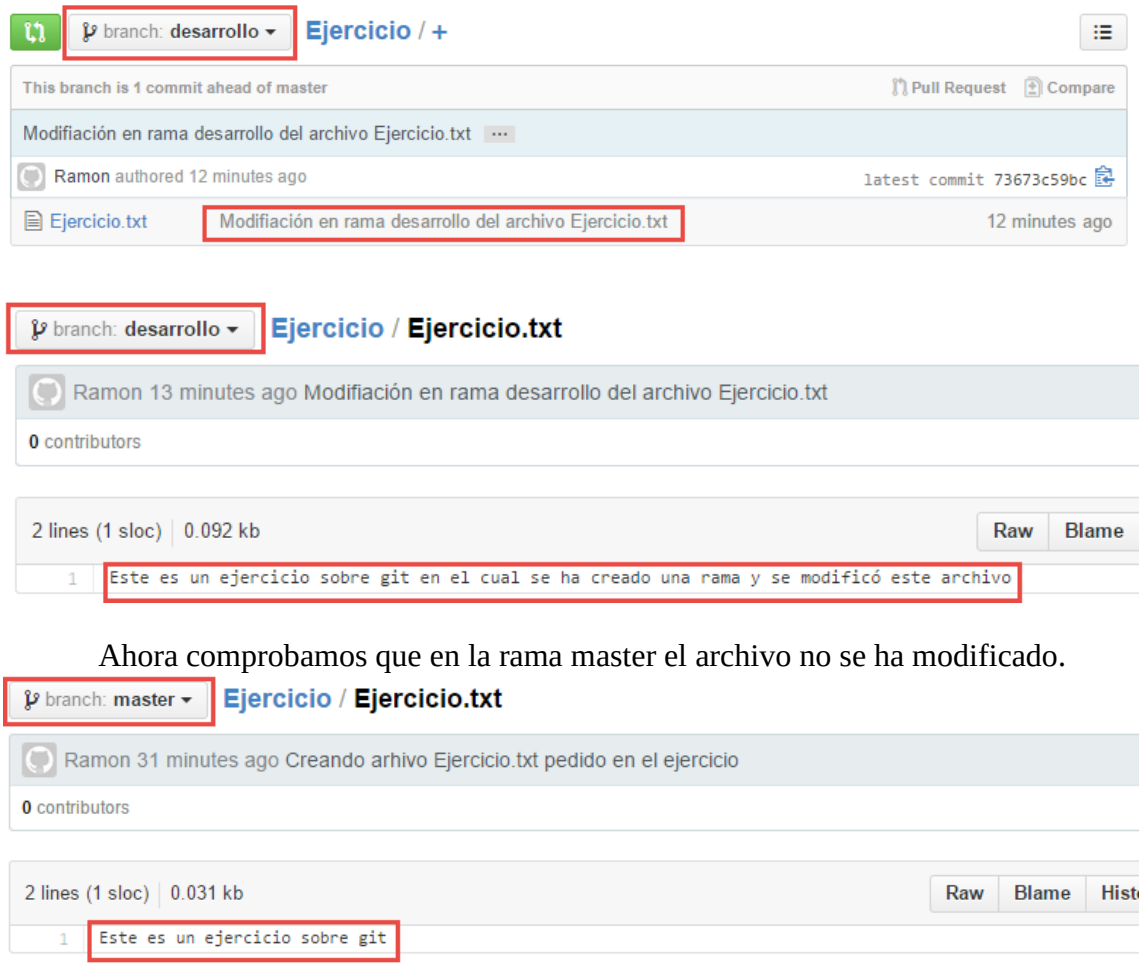
Seguidamente nos disponemos a realizar la modificación del archivo Ejercicio.txt en esa nueva rama, y haremos commit y push.

```
ubuntu@ubuntu-VirtualBox:~/Escritorio/Ejercicio$ git commit -a
[desarrollo 73673c5] Modifiación en rama desarrollo del archivo Ejercicio.txt
 1 file changed, 1 insertion(+), 1 deletion(-)
```

```
ubuntu@ubuntu-VirtualBox:~/Escritorio/Ejercicio$ git push origin desarrollo
Counting objects: 5, done.
Delta compression using up to 3 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 441 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:Snizel/Ejercicio.git
   77f9f94..73673c5  desarrollo -> desarrollo
```

Ilustración 22 Modificacion del archivo

Ahora observamos los cambios en Github.



Ahora comprobamos que en la rama master el archivo no se ha modificado.

Ilustración 23 Observando cambios

Seguidamente, como se pide en el enunciado, se visualizarán los commits realizados, esto se puede hacer con el comando "git log":

```
ubuntu@ubuntu-VirtualBox:~/Escritorio/Ejercicio$ git log
commit 73673c59bc1b3708a801cc75e38b1f5c4f8306ac
Author: Ramon <demonsnizel@gmail.com>
Date:   Thu Dec 18 17:52:31 2014 +0100

    Modifiación en rama desarrollo del archivo Ejercicio.txt

    Se ha modificado el archivo Ejercicio.txt con el nuevo contenido:
    Este es un ejercicio sobre git en el cual se ha creado una rama y se modificó este archivo

commit 77f9f945225153f796bf75d49b032b8120377161
Author: Ramon <demonsnizel@gmail.com>
Date:   Thu Dec 18 17:34:48 2014 +0100

    Creando arhivo Ejercicio.txt pedido en el ejercicio

    Se ha añadido el archivo requerido con el contenido:Este es un ejercicio sobre git.
```

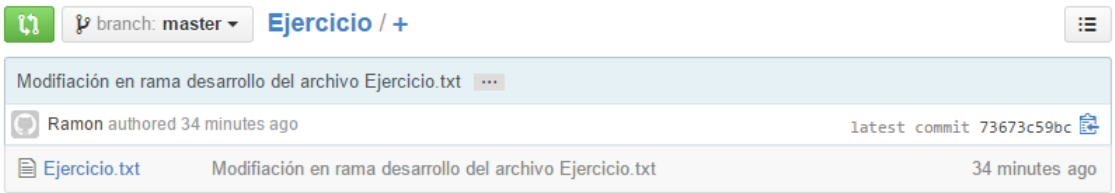
Ilustración 24 Git log

Aquí se puede observar como el autor es el definido anteriormente así como el email, el "id" de los commits, la fecha en la que se realizaron, el título y descripción de cada commit.

Finalmente, procederemos a realizar el "merge" de la rama desarrollo a la rama "master", para ellos nos situamos en la rama master y usamos el comando "git merge desarrollo".

```
ubuntu@ubuntu-VirtualBox:~/Escritorio/Ejercicio$ git checkout master
Switched to branch 'master'
ubuntu@ubuntu-VirtualBox:~/Escritorio/Ejercicio$ git branch
  desarrollo
* master
ubuntu@ubuntu-VirtualBox:~/Escritorio/Ejercicio$ git merge desarrollo
Updating 77f9f94..73673c5
Fast-forward
 Ejercicio.txt |    2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

```
ubuntu@ubuntu-VirtualBox:~/Escritorio/Ejercicio$ git push origin master
Total 0 (delta 0), reused 0 (delta 0)
To git@github.com:Snizel/Ejercicio.git
   77f9f94..73673c5  master -> master
```



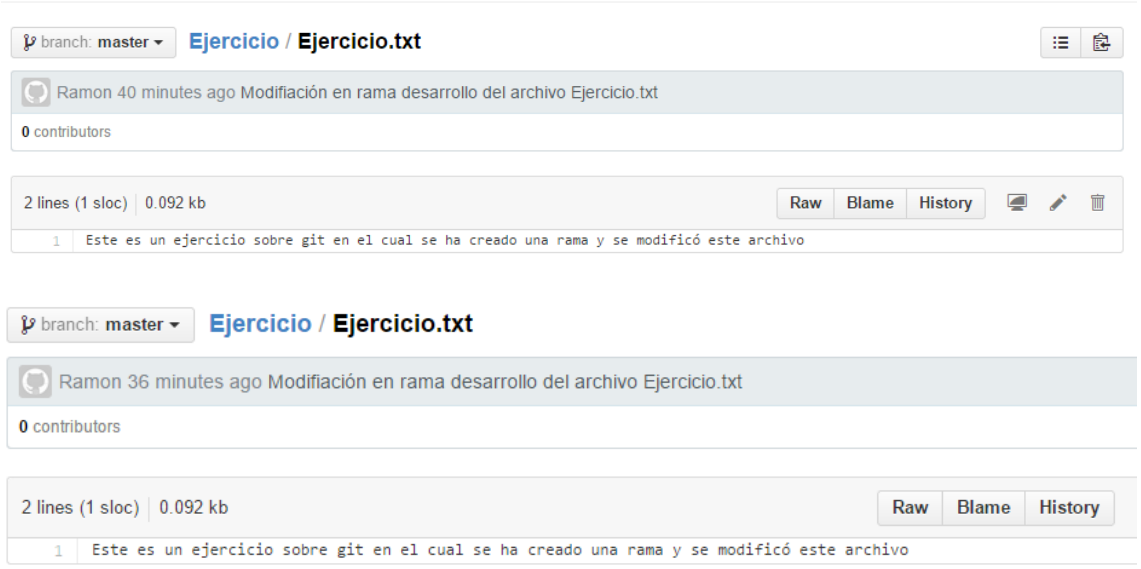


Ilustración 25 Git Merget desarrollo

Como se puede observar ahora la rama master contiene ahora el archivo con su nuevo contenido, por tanto ya se han cumplido todos los objetivos del ejercicio.

Está bien el ejercicio. Quedaría tal vez mejor poner también lo de la rama work (descrito antes) y hacer los merge correspondientes.

5.Gestión de la construcción e integración continua

La gestión de la construcción consiste en decidir cómo se va a construir/formar el software a partir del código. El software, como sabemos, es complejo y durante su desarrollo se han de tener en cuenta complicaciones: equipo de trabajo distribuido, varias construcciones y entregas, y uso de librerías externas al proyecto.

Todos los demás subsistemas son libres de tener siempre instalada la última versión del código en su máquina haciendo pulls diarios de nuestro repositorio en GitHub. Se nos avisa, eso sí, de los riesgos que conlleva esto (posibles fallos de retro compatibilidad o pequeños bugs que se cuelan en el desarrollo normal de cualquier software).

Para la comunicación con los tres subsistemas con los que estamos relacionando hemos hecho lo siguiente:

Almacenamiento: De ellos consumimos su api a partir de una url que va asociada a una id de una votación en concreto y que nos devuelve una lista de votos cifrados.

Verificación: De ellos obtenemos una librería, que importamos en nuestro proyecto, y que utilizamos para llamar a los métodos que descifran los votos.

Frontend: Este subsistema consume nuestro api rest a partir de una url para obtener los resultados de las votaciones mediante un Json.

Buena ilustración.

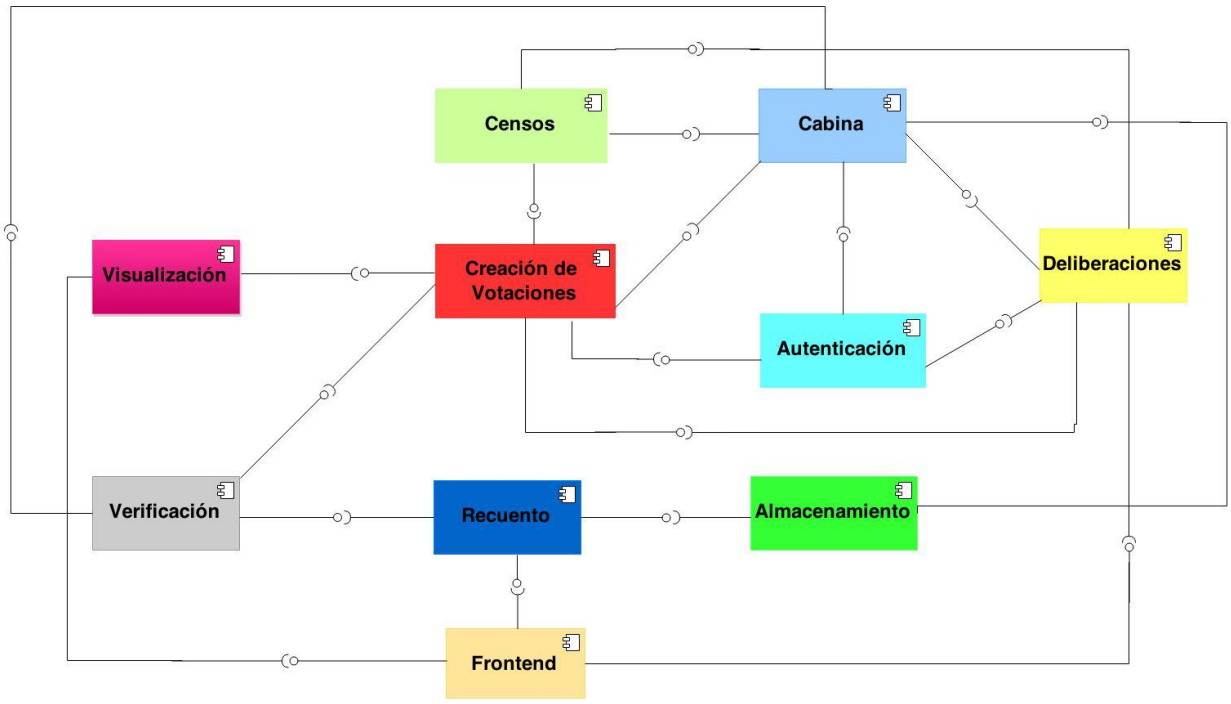


Ilustración 26 Comunicación Subsistemas

Para la realización de la integración, el subsistema "Recuento" se comunica con los subsistemas:

Almacenamiento de votos, subsistema encargado de la gestión de la base de datos para el almacén de votaciones.

Verificación, subsistema que aporta una librería para el cifrado, descifrado, y prueba de la veracidad de los votos almacenados.

Front-end de resultados, encargado de recibir el resultado del tratamiento de votos por parte del subsistema "Recuento", almacenarlos en una base de datos independiente y transformarlos para, posteriormente, ofrecérselo al subsistema de "Visualización de resultados" cuando éste los solicite.

La integración de subsistemas se ha realizado de la siguiente manera:

Una vez realizada la votación, y existiendo en el subsistema de "*Almacenamiento*" una lista de votos cifrados con un identificador de la votación, el subsistema de "*Front-end de resultados*" lanzará una petición al subsistema "*Recuento*", que incluirá el identificador de la votación ya existente, de forma que este subsistema, con ese identificador, hará llamada al subsistema "Almacenamiento de votos", para que éste devuelva la lista de votos cifrados pertenecientes a dicha votación.

Habiendo obtenido dicha lista de votos cifrados, se procede a verificar la veracidad de dichos votos, usando el método de la librería del subsistema "Verificación" "checkVote()"

Durante el desarrollo de esta sección ha habido una sucesión de conflictos debido a la mala comunicación entre subsistemas, a la falta de conocimiento de los restantes subsistemas y a la diversidad de lenguajes usados.
En este caso, el principal problema ha venido a costa de que el subsistema de "Cabina de votación" ha cambiado repetidas veces el modelo de dominio del problema, de forma que a medida que ellos han realizado cambios, sin consultarlos con el resto de subsistemas, el subsistema "Recuento" ha tenido que realizar varias veces su algoritmo, teniendo que adaptarlo a su gusto.
Otro de los conflictos encontrados ha sido que no se ha podido comprobar el cifrado y descifrado de votos debido a que aún no hay ninguno almacenado en el subsistema de almacenamiento. Esto se ha debido a que el subsistema de "Verificación" no ha compartido su librería hasta los últimos momentos del desarrollo del sistema, de forma que ni el subsistema "Cabina de votación", ni el subsistema "Recuento" han tenido acceso a ella.

Integración continua.

Es una metodología informática propuesta inicialmente por Martin Fowler que consiste en hacer integraciones automáticas de un proyecto lo más a menudo posible para así poder detectar fallos cuanto antes.

Se entiende por integración la compilación y ejecución de tests de todo un proyecto.

En el proyecto la herramienta que se utilizará es Jenkins, el cual es un software de Integración continua open source escrito en Java. Jenkins corre en un servidor de aplicaciones y soporta herramientas de control de versiones como CVS, Subversion, Git, Mercurial, Perforce y Clearcase.

Puede ejecutar proyectos basados en Apache Ant y Apache Maven, así como scripts de shell y programas batch de Windows.

Se encuentra liberado bajo licencia MIT.

Instalación de Jenkins

Se ha instalado Jenkins en el sistema operativo Ubuntu y para ello en el terminal se ejecutará lo siguiente:

wget -q -O - https://jenkins-ci.org/debian/jenkins-ci.org.key | sudo apt-key add -
sudo sh -c 'echo deb http://pkg.jenkins-ci.org/debian binary/ > /etc/apt/sources.list.d/jenkins.list'
sudo apt-get update
sudo apt-get install Jenkins

Una vez hecho, lanza un servidor web en el puerto 8080, accesible desde la URI http://localhost:8080.

Configuración básica de Jenkins

En la parte izquierda del panel de inicio de Jenkins pinchamos sobre Administrar Jenkins:

Ilustración 27 Jenkins

Una vez dentro pinchamos sobre Configuración Global de Seguridad y configuramos todo como en la imagen siguiente:

Ilustración 28 Configuración Global

Ahora procedemos a configurar el JDK, el cual podremos utilizar uno que hayamos descargado previamente en nuestro sistema como se ve en la imagen o podremos decir que se descargue e instale automáticamente introduciendo el nombre de usuario y contraseña de Oracle:

Ilustración 29Configurar JDk

A continuación se configura Maven y como en el caso anterior, podremos utilizar una versión descargada con anterioridad o elegir la opción de que se instale automáticamente:

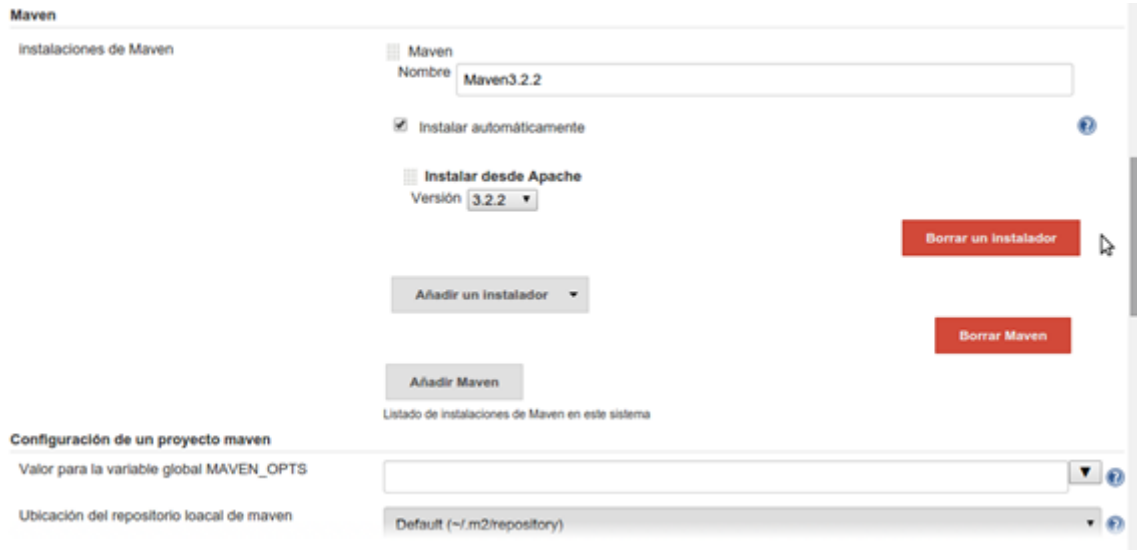


Ilustración 30 Configurar Maven

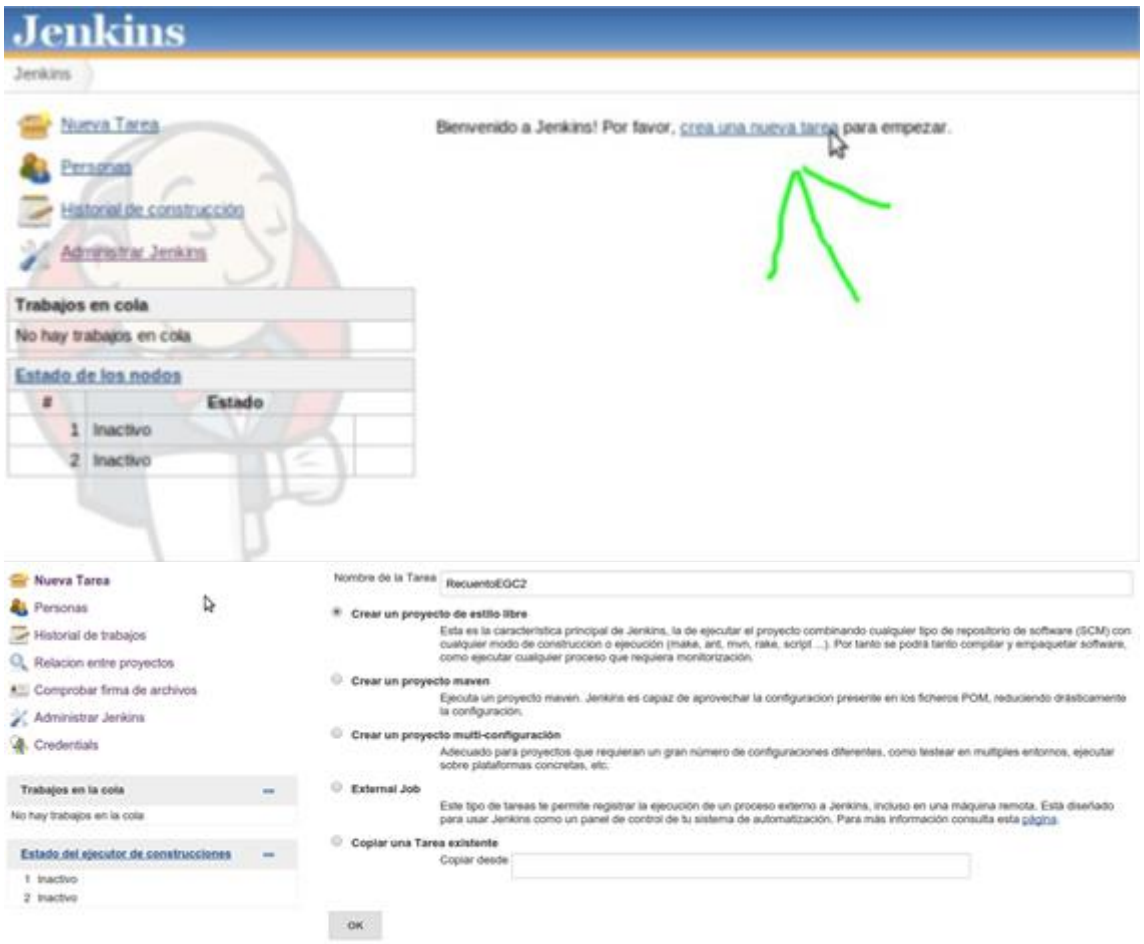
Como se va a utilizar un proyecto que se encuentra en GitHub, hay que instalar el plugin de git, y para ello se pincha en Administrar Jenkins – Administrar Plugins, y se realiza una búsqueda avanzada hasta encontrar Git Plugin:

Ilustración 31 Plugin de git

Se configura el plugin de git como se ve en la imagen:

Ilustración 32 Configuración plugin

En las siguientes imágenes se va a mostrar como subir un proyecto que se encuentra en Git a Jenkins:



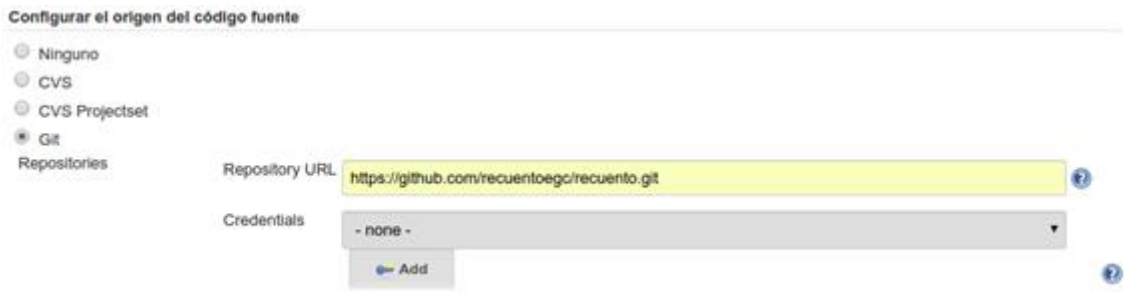


Ilustración 33 Subir proyecto git a jenkins

Importancia del Testing en integración continua

Sin testing en nuestro proyecto, un sistema de integración continua únicamente probará que nuestro proyecto compila.

A medida que añadimos pruebas unitarias/integración a nuestro proyecto, en cada compilación desde el sistema de integración continua se ejecutan la pruebas.

Cuanto más pruebas unitarias/integración existan en nuestro proyecto, más eficaz es un sistema de integración continua, ya que será más sencillo que se detecten errores que se ha subido al repositorio.

Demasiado descriptivo en cuánto a la herramienta y se olvida un poco de hacer más trabajo en describir conceptualmente qué se ha hecho en la integración continua.

6.Gestión de cambios e incidencias

Dado el sistema de trabajo y comunicación llevado por los miembros de los distintos subsistemas del sistema Agora Us, los cambios y las incidencias han sido algo muy común en el proyecto.

Lamentablemente, la forma de resolver estas incidencias no ha sido la más conveniente, ya que no se tenía conocimiento de las técnicas expuestas en el tema 4 de la asignatura "Gestión de incidencias y depuración" puesto que las incidencias llegaron antes que la lección.

Explicado esto, se expone a continuación la metodología realizada por el grupo recuento para la gestión de Cambio, Incidencias y Depuración.

Debido a los numerosos cambios de formato del tipo Voto, se tuvo que reescribir el algoritmo reiteradas veces. La modificación del código la han realizado solo una parte del grupo, ya que 10 personas trabajando sobre un código tan escueto produciría muchos fallos. Para solucionar la incidencia, diagnosticamos la misma y pasamos a repararla. Estos cambios se deciden en las reuniones de equipo y queda reflejado en las actas y en los commits realizados en Github.

A continuación, la gestión de incidencias y cambios que se debería haber llevado a cabo.

La gestión anterior es demasiado básica, ya que no se tiene un control de las incidencias que han llevado al cambio del código para resolverlas. Lo ideal habría sido crear una tabla en la que aparezcan varias columnas con los campos: Identificador de la incidencia, título de la incidencia, descripción de la incidencia, prioridad, solución propuesta e identificadora del commit en el que aparece dicha solución. Esta tabla se va rellenando a la vez que se resuelve la incidencia.

Para notificar las incidencias, se debe usar la herramienta que provee el sitio de GitHub "Issues" en el repositorio de EGC compartido. En ella, los propios miembros internos del subsistema o de otros subsistemas, exponen el problema encontrado y de esta forma se ponen en conocimiento de todos. Aquí un ejemplo del uso de esta herramienta:

Ilustración 34 Issues

El subsistema de verificación informa de un fallo con la longitud de la clave RSA

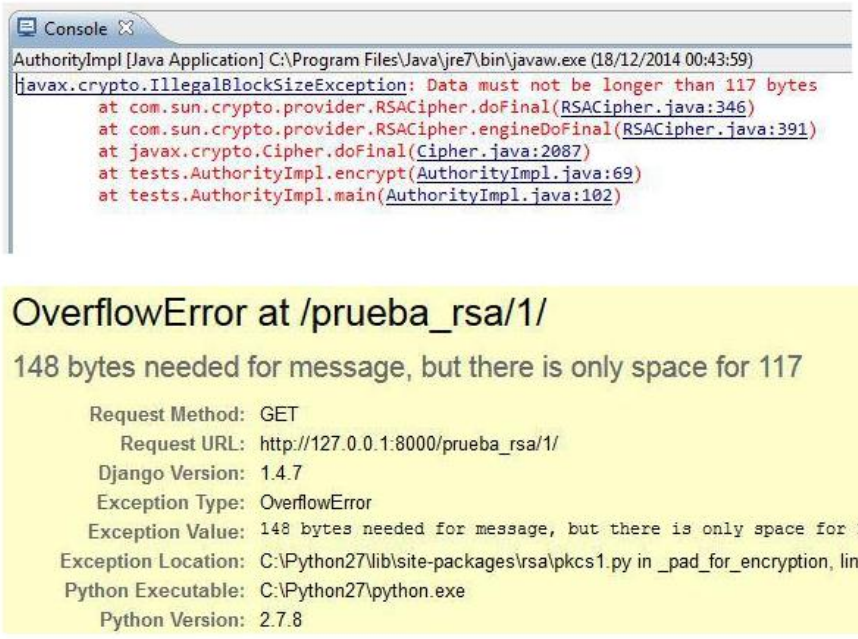
Verificación: longitud de la clave RSA #10

Open **juamaiosu** opened this issue 17 hours ago · 4 comments

juamaiosu commented 17 hours ago — Owner

Abro esta incidencia para especificar que la longitud de la clave RSA generada por verificación es demasiado pequeña, ya que con 1024 bits de clave RSA, solo se puede cifrar 117bytes y la longitud del voto es de 147 bytes. Propongo subir la longitud de la clave a 2048 o pasarnos a otro algoritmo de cifrado como seria "AES".

Adjunta la foto del fallo de verificación y la nuestra.



juamaiosu added the **bug** label 17 hours ago

Ilustración 35 Clave RSA

El problema más detallado y las propuestas de soluciones.

Ilustración 36 Detalles

Y aquí la conversación que mantienen los miembros del proyecto para llegar a una solución.

La manera correcta de tratar una incidencia una vez ya informada es la siguiente:

Reproducir: Se reproduce la incidencia

Para ello:

Se prueba a reproducirla en el entorno de desarrollo.
Se prueba en otros escenarios como preproducción, producción y otros con distintas variantes de tecnologías.

Diagnosticar: Una vez encontrada la manera de reproducir la incidencia, se pasa a diagnosticar la misma. Para ello:

Se sacan hipótesis propias.
Se prueban experimentos de esas hipótesis.
Ensayo/Error

Reparar: Ya diagnosticada la incidencia, se pasa a repararla. Para ello:

Se añaden pruebas para el fallo
Ver que la prueba falla
Repararla
Demostrar correcto funcionamiento
Realizar pruebas de regresión

Analizar: Una vez finalizado todos los procesos anteriores, se pasa a la realización de un documento con las lecciones aprendidas.

6.1. Ejercicio GitHub

Como propuesta de ejercicio, vamos a suponer que el subsistema de Front-End solicita un Json distinto al que el subsistema Recuento le ofrece.

Aqui la notificación que envía el subsistema Front-End a Recuento mediante la herramienta Issues de GitHub:

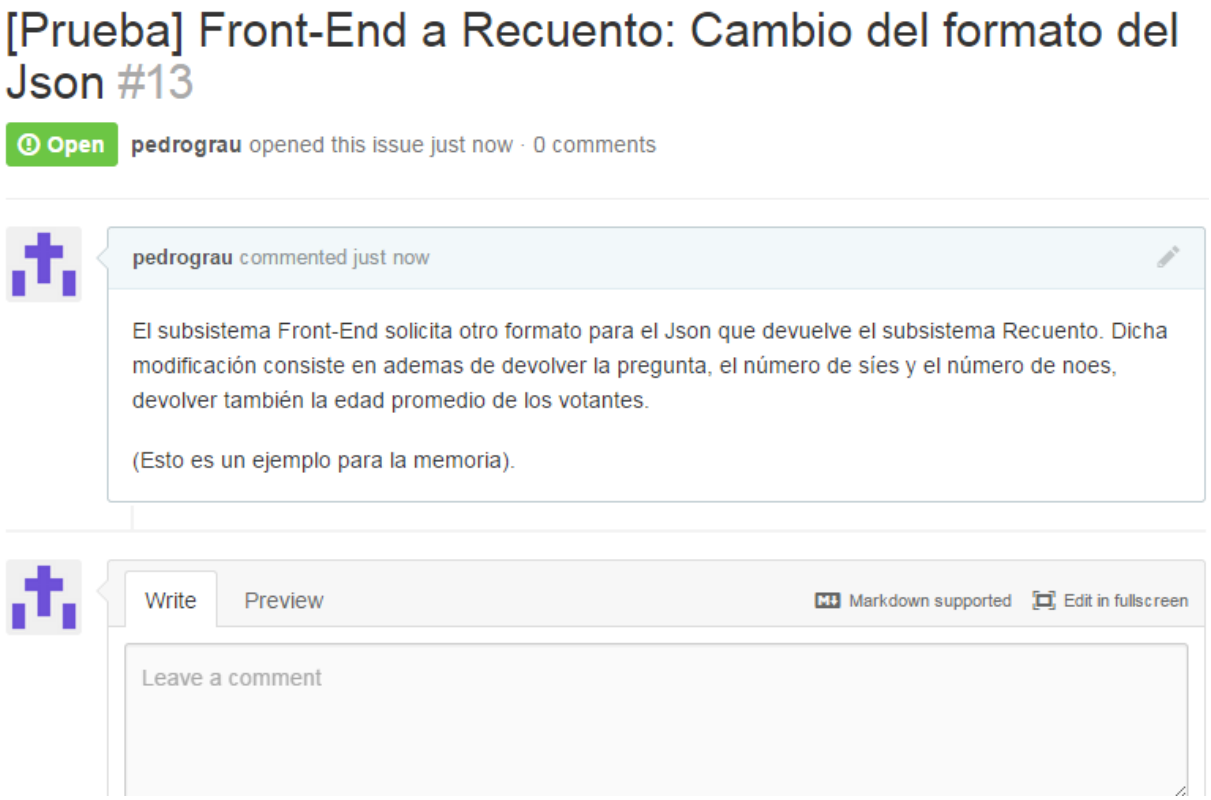


Ilustración 37 Notificacion Front-End

Una vez conocida la petición, se registra la incidencia en la tabla de incidencias, anotando para ello:

Identificador de la incidencia que será un número.
Título de la incidencia que será: “Cambio del formato del Json”.

Descripción de la incidencia que será:"El subsistema Front-End solicita otro formato para el Json que devuelve nuestro subsistema. Dicha modificación consiste en además de devolver la pregunta, el número de síes y el número de noes, devolver también la edad promedio de los votantes".

- Prioridad: Se ha decidido darle prioridad Alta ya que es un cambio de urgencia para la integración.
-

Los campos Solución propuesta e id del commit se rellenan más adelante.

	A	B	C	D	E	F
1	ID	TITULO	DESCRIPCION	PRIORIDAD	SOLUCIÓN PROPUESTA	ID COMMIT
2	1	Cambio del formato del Json	El subsistema Front-End solicita otro formato para el Json que devuelve nuestro subsistema. Dicha modificación consiste en ademas de devolver la pregunta, el número de síes y el número de noes, devolver también la edad promedio de los votantes	ALTA		

Ilustración 38 incidencia 1

Una vez registrada la incidencia correctamente, se procede a la resolución de la misma que no es más que sumar todas las edades del atributo Edad del tipo Voto para luego dividir el número resultante entre el número de votos.

Tras implementar la nueva funcionalidad, se procede a realizar el commit pertinente y ya tenemos los datos para rellenar los dos apartados que quedaban en la tabla.

	A	B	C	D	E	F
1	ID	TITULO	DESCRIPCION	PRIORIDAD	SOLUCIÓN PROPUESTA	ID COMMIT
2	1	Cambio del formato del Json	El subsistema Front-End solicita otro formato para el Json que devuelve nuestro subsistema. Dicha modificación consiste en ademas de devolver la pregunta, el número de síes y el número de noes, devolver también la edad promedio de los votantes	ALTA	Sumar todas las edades del atributo Edad del tipo Voto en una variable, para luego dividirla entre el numero de votos.	40aa61e2

Ilustración 39 incidencia 2

Tras esto, se prueba la nueva funcionalidad en distintos entornos y se informa a el subsistema de Front-End de que ya está aplicada la nueva funcionalidad para que la testen y comprueben que todo funciona correctamente.

7.Mapa de herramientas

Debido a la experiencia recientemente adquirida por parte de todos los miembros del grupo del framework "Spring", el lenguaje elegido para el desarrollo de nuestro proyecto ha sido Java, junto con este framework.

Para la realización de este proyecto, ya que cada miembro del grupo está usando un sistema operativo diferente, hemos decidido montar una máquina virtual donde se instalarán las herramientas que vamos a utilizar para el desarrollo de nuestra parte del código.

El Software de virtualización que se ha elegido es Oracle Virtual Box (Versión 4.2.6 r82870), que se puede descargar de forma gratuita y en la que se montará el siguiente conjunto de herramientas.

Una de las que hemos elegido, "Spring Tool Suite" (versión 3.6.1 RELEASE), que está disponible para Ubuntu y para Windows; que nos proporciona una distribución basada en all-in-one de un Eclipse personalizado que hace que el desarrollo de aplicaciones sea más fácil. Esta herramienta nos facilita una plantilla que se configura fácilmente y nos da un amplio abanico de tecnologías para poner en nuestra plantilla personalizada, además "Spring Tool Suite" nos proporciona un cliente para el uso de GitHub y así poder tener nuestro proyecto en el repositorio y que la gestión del mismo se haga desde el propio entorno de desarrollo.

Todos los miembros del grupo han utilizado la misma máquina virtual con la siguiente configuración:

- Sistema operativo: Windows 7 Enterprise (64 bits)
- IDE: Spring Tool Suite, un mod de Eclipse que nos proporciona una distribución "all-in-one" del framework "Spring" y que utiliza el lenguaje Java.

Al tratarse del corazón de nuestro proyecto nos gustaría añadir los siguientes enlaces para que pueda ampliar información sobre la herramienta:

Página principal: http://spring.io/tools

Guía de inicio al STS: http://spring.io/guides/gs/sts/

Gestor de incidencias del proyecto STS: https://issuetracker.springsource.com/browse/STS#selectedTab=com.atlassian.jira.plugin.system.project%3Asummary-panel

Metodo de resolucion de dudas del proyecto STS: https://stackoverflow.com/questions/ask?tags=spring-tool-suite

- Apache Maven (Versión 2.9): Tiene infinidad de plugins. Su característica más útil, es que te descarga y añade a tu classpath las librerías que usa tu proyecto. Solo tienes que definirlas en un fichero xml. La mayoría de IDE´s lo soportan.
- Servidor de Aplicaciones: Se ha utilizado Apache Tomcat versión 7.0, el cual nos permite arrancar en local nuestro código para comprobar que funciona correctamente.
- Servidor de Integración Continua: Se utilizará Jenkins que es un software de open source escrito en Java. Es un sistema que funciona como Apache Tomcat(contenedor de Servlet) y que soporta herramientas como GitHub.
- Repositorio de Código: Hemos usado GitHub porque Spring Tool Suite, como se ha dicho antes, nos proporciona un cliente para que todo lo referido a la gestión de código (commit, push, pull,...) lo podamos hacer desde el propio entorno de desarrollo además es el que se ha estudiado en la asignatura durante el curso.
- Navegador Web: Hemos utilizado Google Chrome para visualizar el json con el resultado del recuento, accedemos a esta mediante la dirección en local.

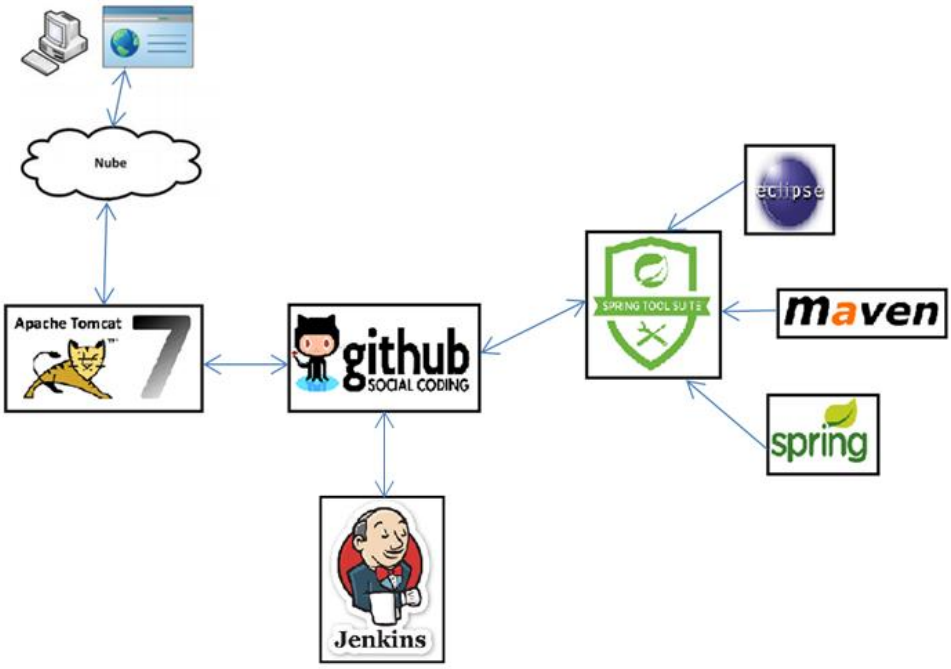


Ilustración 40 Mapa de herramientas

8.Despliegue de todo el sistema

En cuanto al despliegue, todos los portavoces de todos los subsistemas entablaron conversación para llegar a un acuerdo respecto al despliegue.

En principio, se acordó que usaríamos una máquina virtual, donde se desplegarían todos los subsistemas excepto el subsistema “Almacenamiento”, que ya está desplegado en Internet.

En los próximos días, salieron nuevas ideas como desplegarlo en “Openshift”, que es un servicio online, por ello, esta sección sigue pendiente por completar y sentenciar una vía de despliegue entre todos los subsistemas, ya que es probable que incluso Openshift se descarte debido a que quizás no funcionen todos los subsistemas debidamente en ese entorno, por tanto todo lo descrito en este texto, es meramente provisional.

Información sobre Openshift: https://www.openshift.com/

9.Conclusiones

Los componentes de este grupo han llegado a las siguientes conclusiones:

No hemos tenido una buena comunicación con los demás grupos. Existe un caso aislado en el que un grupo ha creado un proyecto en projETSII en el cual no están invitados todos los miembros de todos los grupos y esto hace que no todos los grupos vean los posibles cambios realizados.

Por otro lado, la forma de gestionar los recursos de algunos grupos ha sido caótica, al realizar cambios sin consultar al resto de grupos a los que su proyecto afecta ha provocado la duplicidad en cuanto a la dedicación y trabajo del proyecto.

El grupo de cabina de votación cambió tres veces el tipo de votos provocando la imposibilidad de realizar pruebas de funcionamiento del sistema, al no existir votos encriptados.

10.Lecciones aprendidas

Como lecciones aprendidas tenemos lo siguiente:

- La forma de distribuir los grupos de trabajo no ha sido la más correcta posible, al no existir un grupo que gestionara al resto de grupos, se ha necesitado una persona o un grupo de personas encargadas de gestionar la comunicación entre grupos.

- Por consiguiente, para futuros proyectos realizados por grandes grupos de trabajo se hace necesario una participación activa del profesor en la organización siendo este el gestor de grupos, o bien, que el propio profesorado nombre a personas gestoras de todos los grupos de trabajo, asegurando así la correcta comunicación entre ellos, pudiendo marcar fechas tope para la realización de los proyectos, de forma que se haga partícipes a todos los grupos durante todo el recorrido de la asignatura.

- En cuanto a la integración se refiere, hemos llegado a la conclusión de que es necesario recibir unas clases prácticas de integración, al igual que en el proyecto en el que nos involucramos que debiera existir un grupo encargado de la gestión de las arquitecturas. Luego, ha sido muy laboriosa la tarea de integración debido a que no ha habido un consenso a la hora de elegir un entorno de trabajo común, y al no saber qué entorno ha elegido cada grupo ha habido muchos problemas con las versiones de los sistemas (por ejemplo, la versión de Django usada por el grupo de Cabina de Votación).

- Otro punto de vista a tener en cuenta es la necesidad de autogestión dentro de un mismo grupo, ya que al elegirse la aplicación Whatsapp para la comunicación entre los miembros del grupo (en el caso del subsistema Recuento), no pudiendo estar siempre conectados y por consiguiente no leer los mensajes o problemas que hayan surgido a lo largo del proyecto, ha retrasado el trabajo del grupo, perdiendo horas de trabajo y por lo tanto, productividad. A pesar de que todas las semanas como mínimo se reunía el grupo una vez, estas reuniones no han sido todo lo productivas como se hubiera deseado cuando se planificó al principio, y quizás el grupo debería de haberse visto con mayor frecuencia.

- En cuanto al desarrollo de la memoria, las herramientas de edición colaborativa (como Google drive) para editar simultáneamente los documentos del trabajo han facilitado mucho la paralización a la hora de documentar.

paralelización?

- Al no ser trabajadores a tiempo completo en el trabajo, la dedicación a la asignatura se ha visto condicionada por numerosos factores externos, como realización de actividades relacionadas con otras asignaturas. Esto ha producido una desviación, realizando gran parte del trabajo los días antes de la fecha de

entrega. Inicial mente se intentó establecer un margen de seguridad, para que todo el trabajo estuviese realizado una semana antes de la entrega, pero en entornos como este, es difícil evitar que se prioricen actividades sin tener en cuenta el margen.

11.Referencias

Nombre: ÁgoraVoting
Url: https://agoravoting.com/

Nombre:Spring framework
Url:http://spring.io/

Nombre: Maven
Url: http://maven.apache.org/

Nombre: GitHub
Url: https://www.github.com/

Nombre: Integración continua(Jenkins)
Url: http://es.slideshare.net/paradigmatecnologico/git-y-jenkins-el-futuro-en-la-gestin-del-ciclo-de-vida-de-aplicaciones
https://1984.lsi.us.es/wiki-egc/index.php/Pr%C3%A1ctica_4_14-15

12.Glosario de términos

A

- API: Conjunto de métodos que ofrece un determinado subsistema para ser utilizado por otro subsistema como una capa de abstracción.

B

- Base de datos: “Almacén” usado por los subsistemas de “Almacenamiento de votos”, “Frontend”, “Verificación” y “Autenticación” para guardar de una forma organizada la información necesaria en cada caso y utilizarla fácilmente.
- Bug: Aquello que provoca el Fallo.
- Baseline: Cada uno de los estados importantes por los que pasa nuestro software.
- Branch/Rama: Línea de desarrollo independiente que puede compartir parte de historia común con otras.

C

- Cifrar: Ocultar el contenido de los votos para que las personas no autorizadas no puedan obtener la información de las votaciones.
- Classpath: Opción para indicar a la Máquina Virtual de Java donde buscar paquetes y clases definidas por el usuario a la hora de ejecutar el programa.
- Cliente: Software encargado de realizar peticiones al servidor de GitHub.

D

- Descifrado: Procedimiento usado por el subsistema “Recuento” para obtener el contenido de las votaciones y poder efectuar el recuento.
- Distribución “All-in-one”: Herramienta que integra todo el software necesario para la creación y gestión de nuestro código fuente.
- Django: Framework basado en Python usado por el subsistema “Cabina de votación”.

E

- Error: Estado interno de un sistema que contiene un Bug.

F

- Fallo: Incapacidad del sistema para realizar una función.

J

- JSON: JSON, acrónimo de *JavaScript Object Notation*, es un formato ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.