Agora@us

Creación y Administración de votaciones

Dirigido a:

Prf. David Benavides Cuevas
Prf. Pablo Neira Ayuso

Proyecto creado para la asignatura EGC (Evolución y Gestión de la Configuración) 4º -Curso 2014/15- Grado en Software de la ETS de Ing. Informática (Universidad de Sevilla)

Versión: 2.0
Fecha: 21 de Diciembre de 2014

Hoja de control de versiones

Título	Documento del Proyecto
Versión	2.0
Fecha de Creación	24/10/2014
Fecha de Cierre	21/12/2014

Historial de versiones

Versión	Descripción	Realizado Por	Fecha
1.0	Creación del documento + Punto 1 y 2	María Pérez	24/11/2014
1.1	Punto 3	Fernando García	24/11/2014
1.2	Punto 5	Jaime Guerrero	9/12/2014
1.3	Punto 4	Francisco Pérez	11/12/2014
1.4	Modificación Punto 2	María Pérez	15/12/2014
1.5	Punto 8	Mercedes Vallejo	15/12/2014
1.6	Punto 10	Daniel De Tena	15/12/2014
1.7	Punto 7 + Diagramas	Olga Moreno	15/12/2014
1.8	Punto 6	Daniel De Tena	17/12/2014
1.9	Ampliación Punto 4	Francisco Pérez	17/12/2014
1.10	Diseño para la versión final	Olga Moreno	17/12/2014
1.11	Revisión + Mejoras	María Pérez	18/12/2014
1.12	Modificación Punto 10	Daniel De Tena	18/12/2014
1.13	Ampliación Punto 5	Jaime Guerrero	18/12/2014
1.14	Ampliación Punto 4	Francisco Pérez	18/12/2014
1.15	Ampliación Punto 5	Jaime Guerrero	19/12/2014
1.16	Ampliación Puntos 3,4 y 5. (Añadidos diagramas) + corrección.	Olga M. Moreno	19/12/2014

	Retoque del diseño del documento.	Martín	
1.17	Ampliación Punto 3	Fernando García	19/12/2014
1.18	Ampliación Punto 8	Mercedes Vallejo	19/12/2014
2.0	Documento en versión 2.0 con respecto a diseño y contenido.	Olga M. Moreno Martín	20/12/14

ÍNDICE

1. Resumen del Proyecto

El objetivo general del trabajo es que el grupo ponga en práctica en un proyecto real todos los conceptos teórico-prácticos que se vean en la asignatura y profundice en ellos todo lo que su motivación lo lleve.

Buena frase!

En concreto, los alumnos de la asignatura "Evolución y Gestión de la Configuración" desarrollarán una aplicación para realizar votaciones, semejante a "Agora Voting". El proyecto general se ha descompuesto en diferentes subsistemas que serán desarrollados por cada grupo y en nuestro caso trabajamos en la creación y administración de votaciones.

El objetivo final del proyecto "Creación y Administración de Votaciones" es proporcionar un conjunto de entregables evaluables compuestos por:

- El presente Documento del Proyecto donde se documentan los pasos a seguir a lo largo del proyecto, el estudio realizado, el aprendizaje obtenido, y las conclusiones a las que se han llegado.
- La máquina virtual con el entorno necesario para realizar los ejemplos descritos en el Documento del Proyecto.
- El diario de grupo donde se muestran las actas de las reuniones, la plantilla de tiempo del equipo y las decisiones importantes que se han tomado.

Para ello se han llevado a cabo las siguientes actividades:

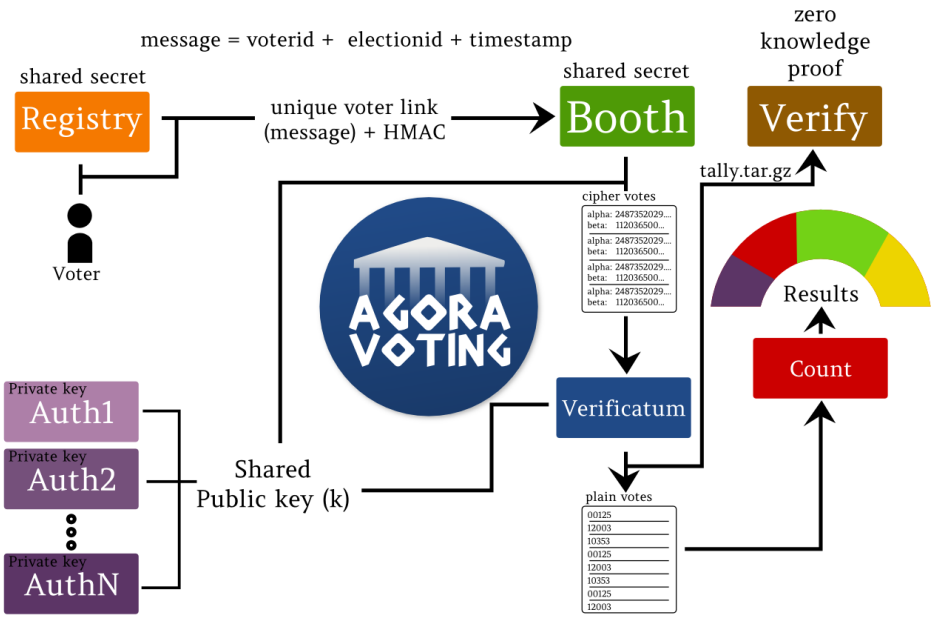
Podría ir en la intro pero no en el resumen

- Sesiones teóricas: Sesiones teóricas donde el profesor explica los distintos módulos de teoría que se ponen en práctica durante la elaboración de la aplicación.
- Creación de la aplicación: Conjunto de actividades que se realizan con el objetivo de desarrollar la aplicación:
  - Conformación del grupo y elección del subsistema a desarrollar.
  - Reuniones del grupo e intergrupales. El objetivo puede ser desarrollar código o tomar decisiones importantes. Estarán reflejadas en las Actas.
  - Iteraciones de Código: Implementación de la aplicación.
  - Integración intergrupales.
- Redacción de Documentación: Se incluye el diario del Grupo, las Actas, el Documento del Proyecto.
- Entrega: Consiste en empaquetar toda la entrega y enviar el contenido a través de Opera.

No hay un resumen del trabajo en sí...

2. Introducción

El proyecto elegido a desarrollar es, cómo se ha introducido en el Resumen del proyecto, el subsistema "Creación y Administración de Votaciones" de la aplicación "Agora@us", inspirada en la conocida aplicación "Agora Voting".



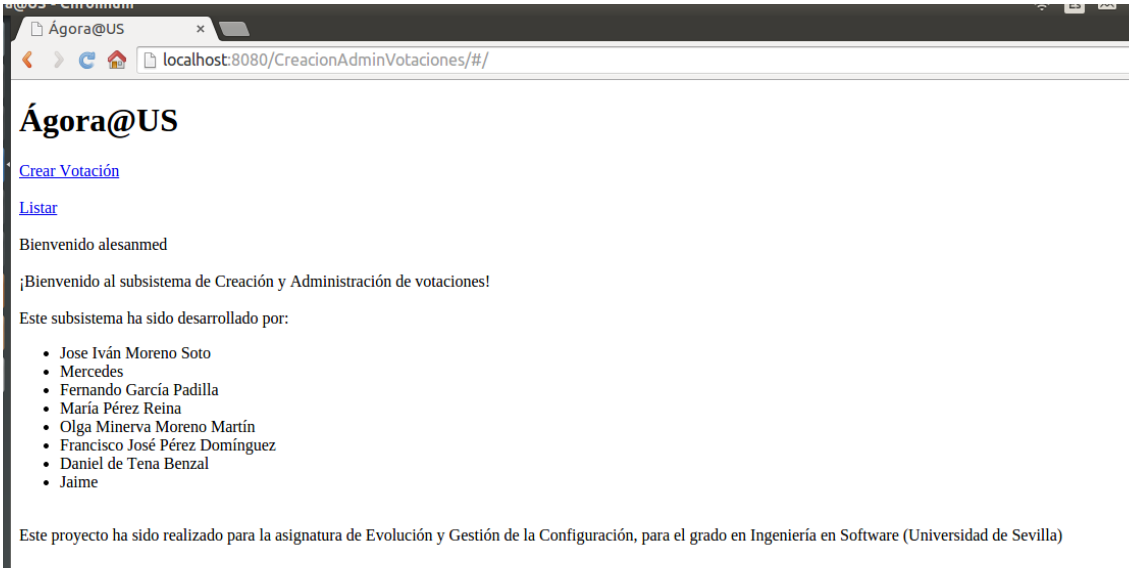
Poner leyendas a todas las figuras del documento

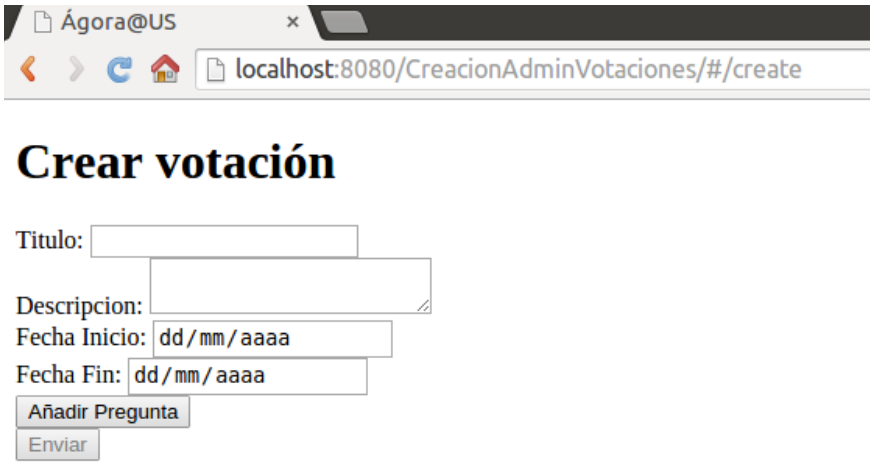
Este subsistema se encarga de proporcionar un formulario para crear una Votación, gestionando las votaciones en una base de datos. Además se ofrece la interfaz donde un usuario visualiza la lista de las votaciones creadas, con opción a borrarlas si no se ha votado todavía, y a editar el censo relacionado.

Además, hay que siempre referirse a esas figuras cnd se elebore el texto.

La parte en la que se describe el sistema debería ir en una sección o al menos subsección a parte.

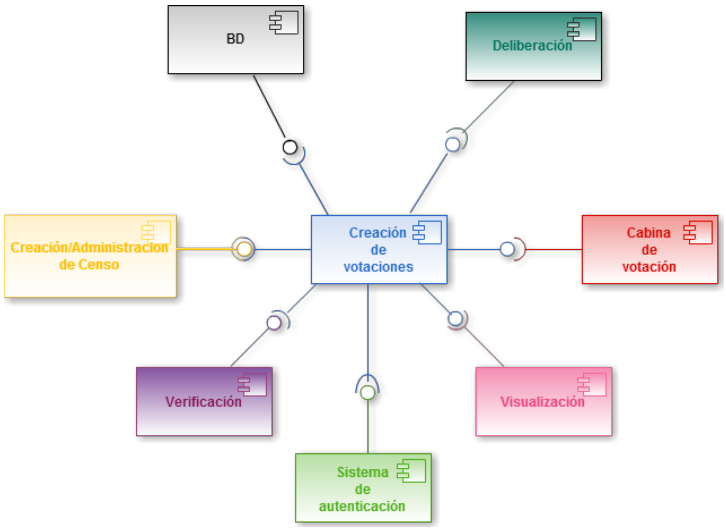
Estos pantallazos deberían ir más ordenados y explicados.



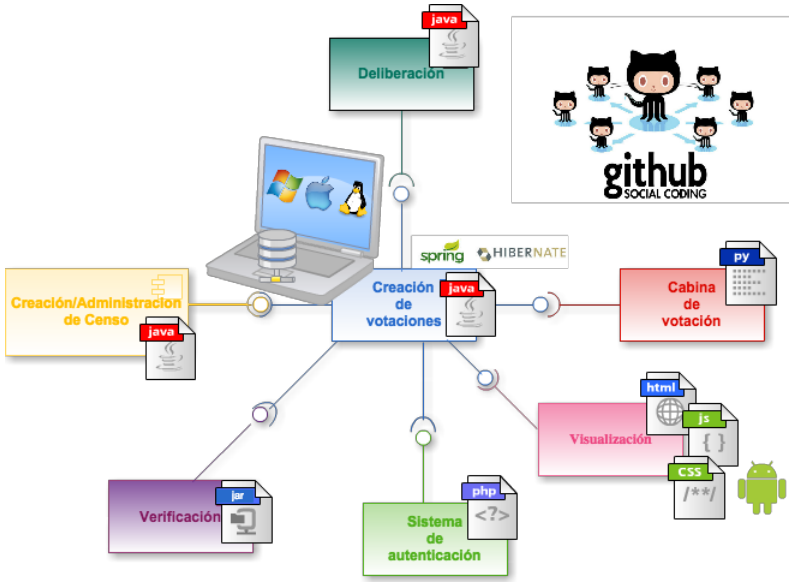


Además, se puede ofrecer como servicio o API la información de las votaciones para que los diferentes subsistemas tengan acceso, como se detalla en el punto "4.Gestión de la Construcción e integración continua".

Para lograr estos objetivos se llevará a cabo la gestión del código fuente entre los integrantes del grupo, la gestión de la integración continua con los subsistemas con los que "Creación y Administración de Votaciones" se relaciona, entre otros, elaborando documentación que explique los procesos llevados a cabo, las técnicas y los problemas encontrados.



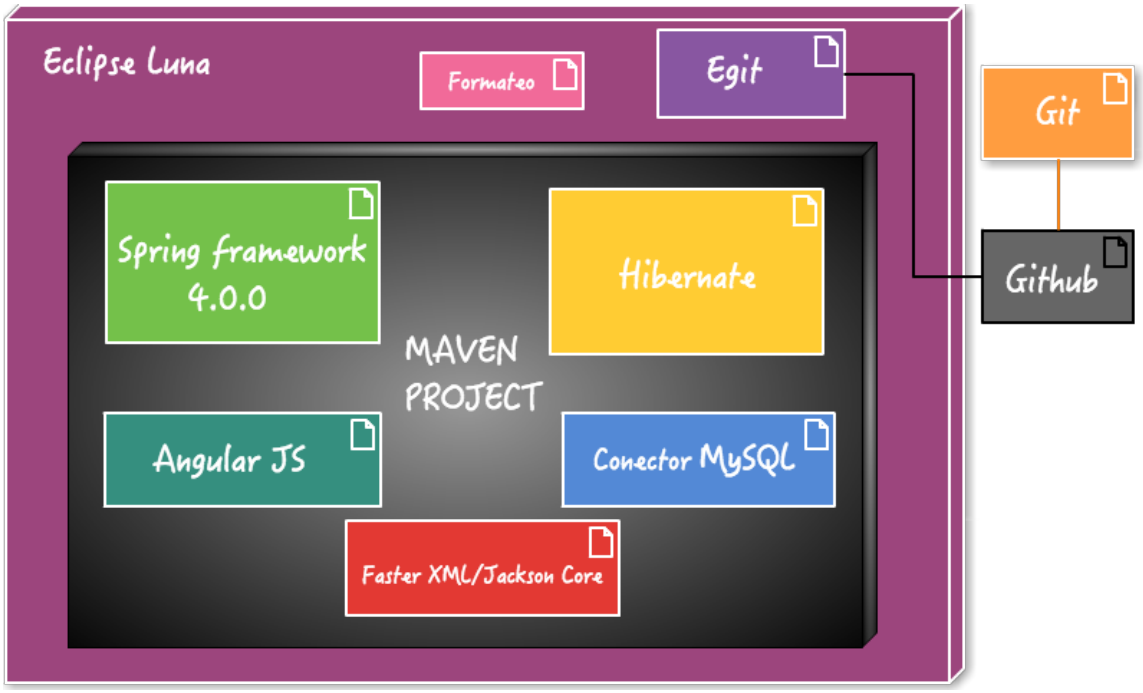
A continuación se muestra el mismo diagrama desde la perspectiva de la funcionalidad para aclarar las herramientas utilizadas y las relaciones desglosadas:



3. Gestión del código fuente

Para hablar de la gestión de código, primero, debemos situarnos.

Esta es la estructura de nuestro proyecto con respecto a la gestión del código:



En nuestro subsistema hemos pasado por diferentes etapas a lo largo del desarrollo que debemos mencionar antes de continuar, en concreto dichas etapas han sido tres:

Este tipo de informes son interesantes pues denotan aprendizaje.

En primer lugar, y de forma errónea, empezamos utilizando un repositorio 'SVN' alojado en ProjETSII en el cual íbamos subiendo los cambios de nuestra parte de la aplicación sin apenas control o protocolo a la hora de hacer commits.

Más tarde, conforme avanzábamos en la asignatura y descubrimos la ventaja de los repositorios 'Git', creamos uno alojado en Github en el que íbamos subiendo nuestro proyecto y los cambios en el mismo pero sin integración en dicho repositorio con ningún otro subsistema de la aplicación 'Agora@us' con los que nos debemos integrar en el desarrollo de la asignatura.

Y por último, cuando se comenzó con la fase de integración se encontró la necesidad de compartir entre todos los subsistemas un único repositorio por lo que, de mutuo acuerdo con el resto de grupos, se creó un repositorio 'Git'

compartido donde participan todos los subsistemas de la aplicación 'Agora@us', para facilitar la integración y su seguimiento continuado.

Cabe destacar, que en las primeras sesiones de integración en clase realizamos la misma transfiriendo el código mediante una memoria USB (algunos grupos realizaban esto mismo vía e-mail) y de ahí surgió la necesidad de crear el repositorio compartido entre los subsistemas de la aplicación que lo necesitaban para simplificar todo el proceso de integración.

Lo mismo, señal de avance

Por tanto, una vez puestos en contexto y ahora centrándonos en la gestión de código usada finalmente, podemos decir que para la gestión de nuestro código fuente hemos utilizado un repositorio Git, que es nuestro sistema de control de versiones, alojado en el popular servicio 'GitHub' lo que nos permite trabajar haciendo commits localmente cuando una nueva funcionalidad esté totalmente implementada, así como hacer un push al repositorio para añadir los cambios al mismo cuando se crea necesario (tendremos que atender los conflictos en caso de resultar necesario).

Sería bueno añadir en qué momento se tomó la decisión.

Como comentábamos anteriormente, este repositorio es compartido por nuestro grupo (Creación y administración de Votaciones) y por diversos subsistemas más, en el cual dichos subsistemas están ya integrados de partida. La URL de dicho repositorio es: https://github.com/EGC-1415-Repositorio-compartido

Respecto a la gestión de ramas, por lo general cada grupo trabajará en la suya dentro del repositorio compartido, salvo algunos casos específicos como el del subsistema de autenticación. En el caso de este último subsistema tienen seis ramas. La principal llamada 'auth' y cinco más llamadas 'auth_api', 'auth_test', 'auth_register', 'auth_database' y 'auth_authentication'.

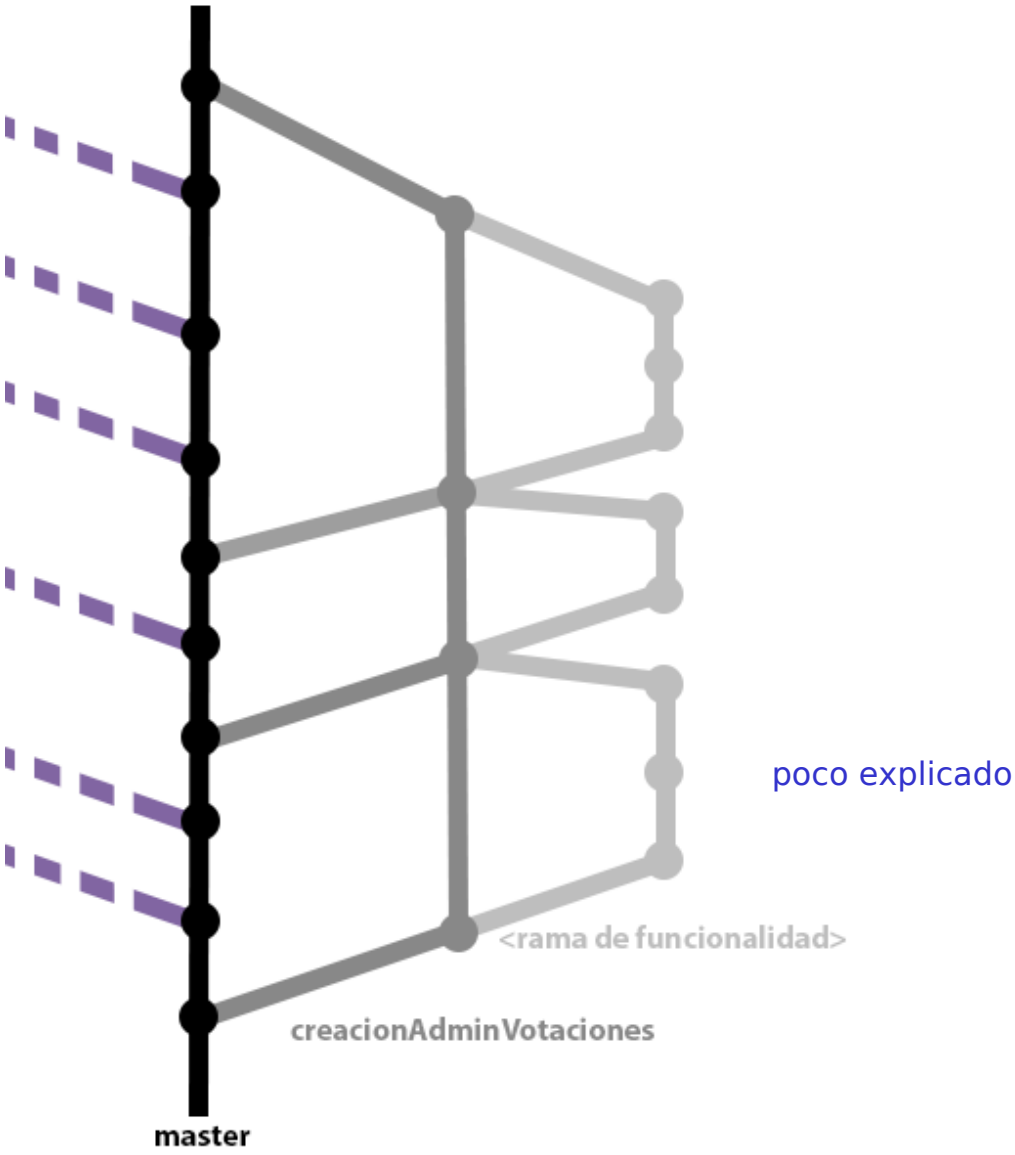
No haría falta describir lo de otros subsistemas.

Para tener una visión global, los grupos que hacen uso de las ramas en el repositorio compartido son los siguientes (grupo / rama principal):

- **Autentificación** / 'auth'
- **Creación y administración de votaciones** / 'creacionVotaciones'
- **Modificación de resultados** / 'modificacion'
- **Deriberaciones** / 'Deriberations'
- **Recuento** / 'counting'
- **Creación y administración de censos** / 'adminCensos'
- **Front-end de resultados** / 'Frontend-Resultados'
- **Visualización de resultados** / results_view''
- **Verificación** / 'votes_verification'

Atendiendo ahora a nuestro caso en concreto, nuestra rama de desarrollo principal se denomina 'creacionVotaciones', crearemos una rama más partiendo de ésta para trabajar en nuevas funcionalidades. Cuando una funcionalidad esté acabada haremos un merge con nuestra rama principal y otro con la rama principal del repositorio compartido ('master') para que los cambios se apliquen tanto a nuestra rama, la específica de nuestro subsistema, como a la global.

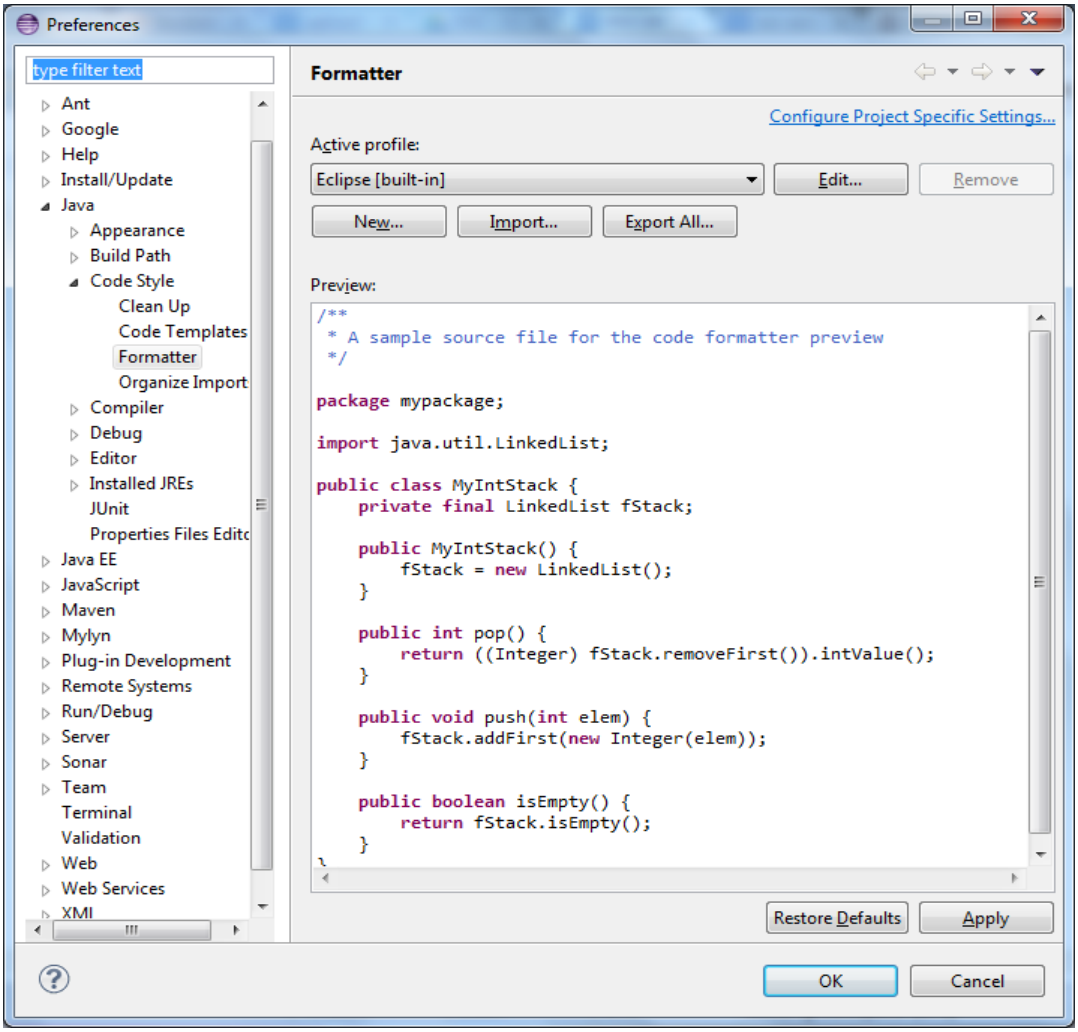
Por tanto, el funcionamiento de la parte del repositorio de la que nuestro grupo hace uso para la realización del proyecto será muy parecido a este ejemplo:



poco explicado

Sobre la aprobación de cambios, aprovechando que desde el inicio del proyecto realizamos, al menos, una reunión semanal los lunes de 15:30 a 17:30, se debaten en la misma o en posibles reuniones extraordinarias que se convocan siguiendo los mecanismos de comunicación aprobados en el grupo, como se describe más detalladamente en el punto "*5. Gestión del cambio, incidencias y depuración*". En dichas reuniones discutiremos los cambios realizados con el resto del grupo y tomaremos una decisión en consecuencia.

Nuestra política de nombre es sencilla: Todos nuestros nombres de variables y constantes están en inglés salvo en los comentarios de código, que sí estarán en español pero sólo están presentes en los métodos que explican las relaciones con otros subsistemas. En cuanto al estilo o formato de código utilizamos el mismo formato que tiene por defecto de Eclipse. Se puede consultar en 'Windows > Preferences' y una vez en el menú que se nos abrirá en 'Java > Code Style > Formatter' (si lo hemos modificado entonces debemos usar el botón de 'Restore default'). El formato en concreto sería el siguiente:



Ejercicio de creación de repositorio y creación / unión de subramas con Git

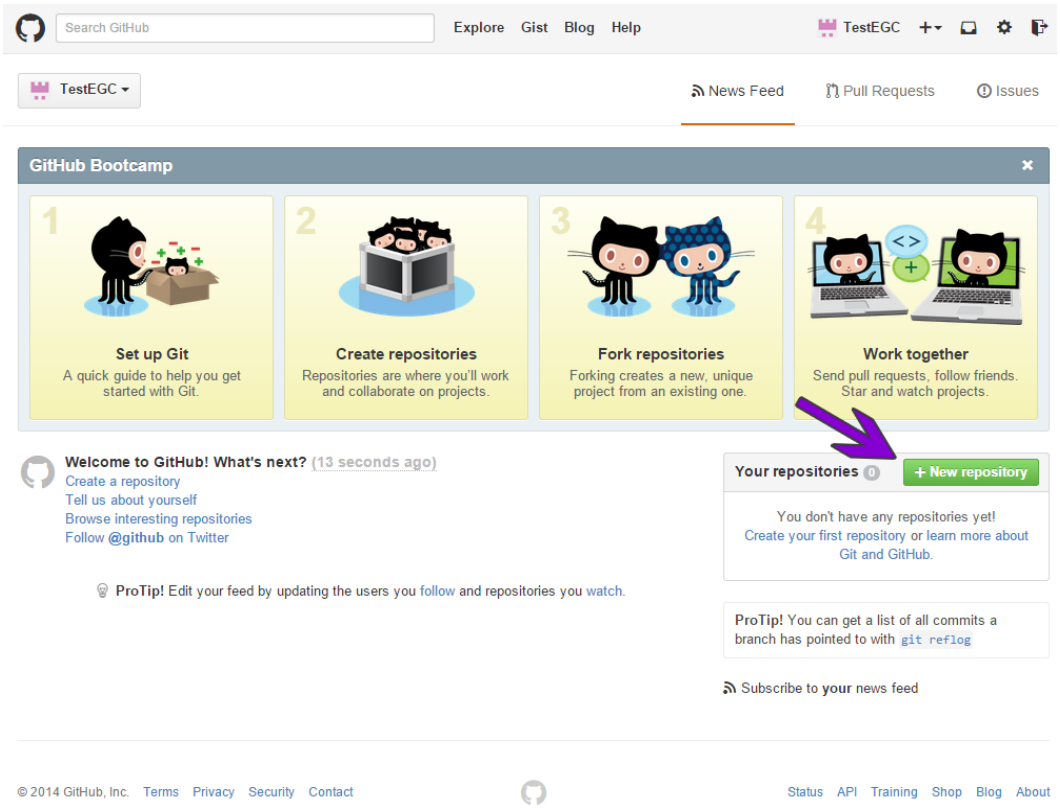
Como hemos explicado en este apartado, tenemos un repositorio compartido en el cual cada grupo tiene una rama para su subsistema y, centrándonos en nuestro caso, otra que emerge de ésta para trabajar en torno a las nuevas funcionalidades. Este ejercicio pretende recrear el modo de trabajar que hemos empleado en nuestro grupo para en el desarrollo de la aplicación.

El propósito del ejercicio es, por tanto, la creación de un repositorio en el servicio 'GitHub', la posterior creación de una rama partiendo de la 'master' y, una vez hecho esto, la creación de otra provisional para la funcionalidad. Siguiendo con el ejercicio, una vez montada esta estructura, tendremos que añadir funcionalidades a nuestra aplicación y realizar los correspondientes merge y cierres de rama.

Ejercicio más bien simple.

Paso 1: Creación del repositorio en 'GitHub'.

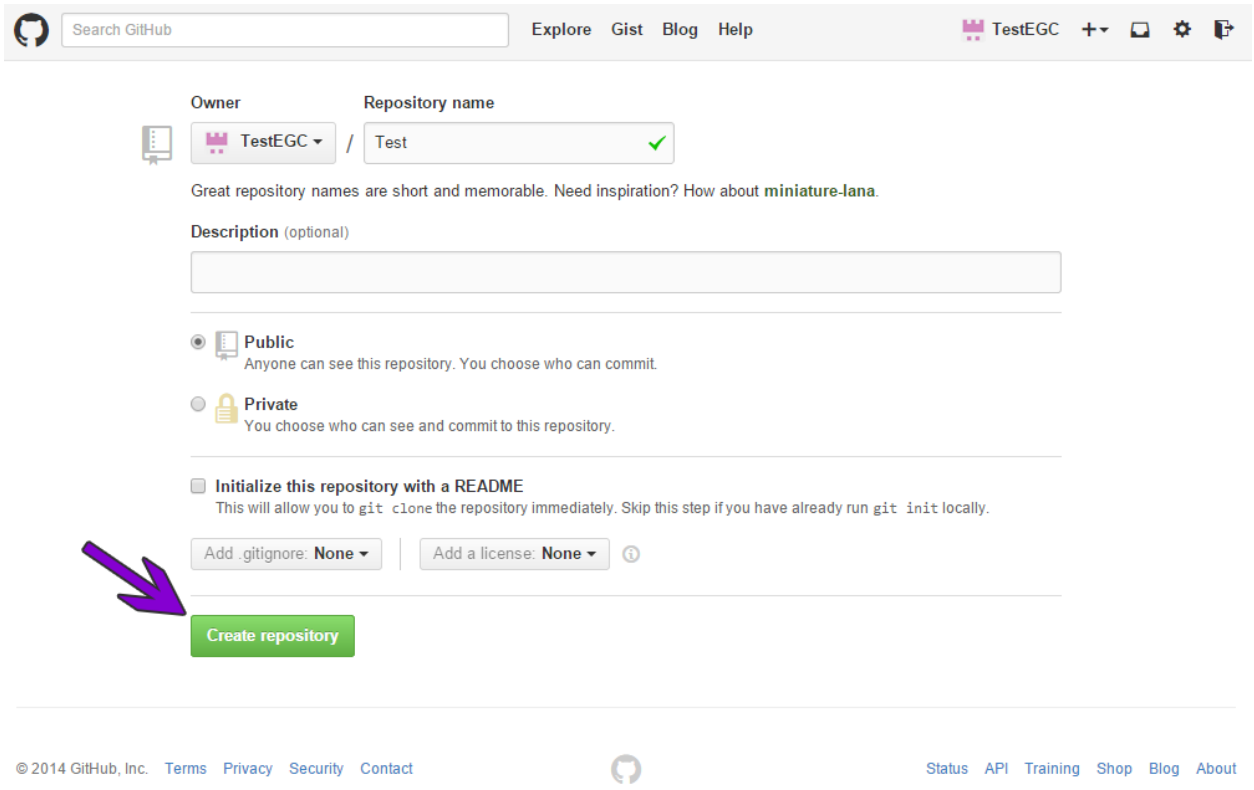
El primer paso del ejercicio es muy sencillo e intuitivo, pero de igual forma vamos a resolverlo: Primeramente nos registramos o identificamos en la web de 'Github' (https://github.com/) y, una vez hecho esto, hacemos click en el botón verde llamado '+ New Repository' que nos aparecerá en la vista principal de la aplicación.



Ahora la aplicación web nos llevará a la siguiente vista, en la que debemos elegir el nombre de nuestro repositorio, que para este ejemplo será 'Test'. Además, podemos añadir una descripción y seleccionar si nuestro repositorio es público o privado, aunque en el caso de que quisiéramos que fuese de este último tipo deberíamos 'upgradear' nuestra cuenta abonando la cantidad indicada en la web ya que no es una opción gratuita.

Otra opción muy importante y que no debemos pasar por alto es la de añadir una licencia a nuestro repositorio. Esto estará en función de nuestra forma de pensar en cuanto al software, lo que sí tenemos que tener presente es que si no queremos añadir ninguna licencia y dejamos el campo tal como viene por defecto ('None') se asume que el repositorio tendrá todos los derechos reservados.

Por último en esta vista debemos seleccionar si iniciar el repositorio con un archivo 'Readme'. Ésta es la opción más sencilla, pero en este ejemplo veremos, a continuación, ambas.



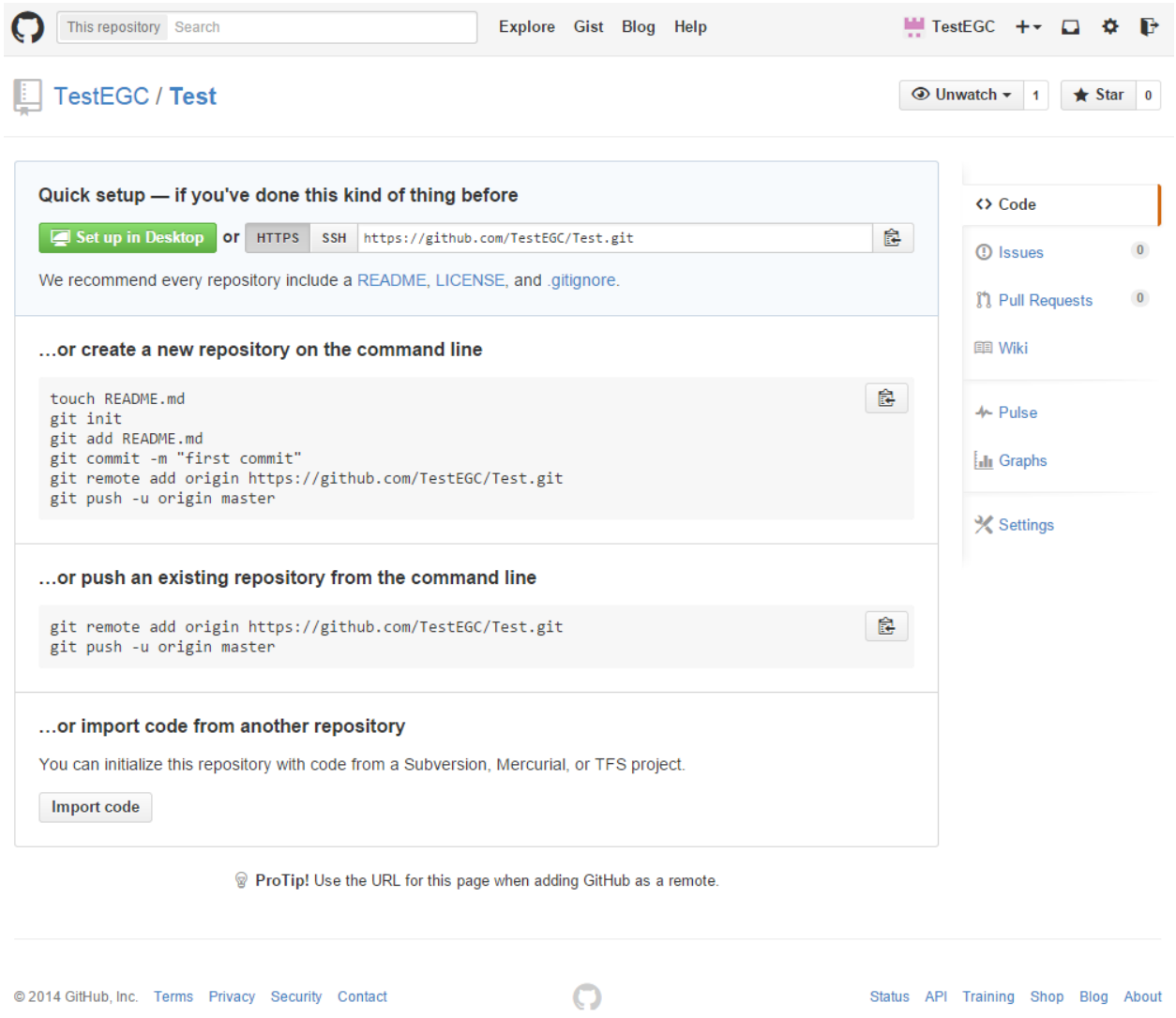
Una vez que hayamos terminado de seleccionar el nombre, descripción y características de nuestro repositorio hacemos click en el botón verde 'Create repository', señalado en la imagen de arriba.

La parte de creación del repo podría sobrar pero si ya está no pasa nada

Paso 2: Inicializar el repositorio.

Como se indicaba arriba, tenemos la opción de inicializar el repositorio con un archivo 'Readme', o 'Léeme' en castellano, veamos ahora cada uno de los casos en función de si marcamos esa opción a la hora de crear el repositorio o no:

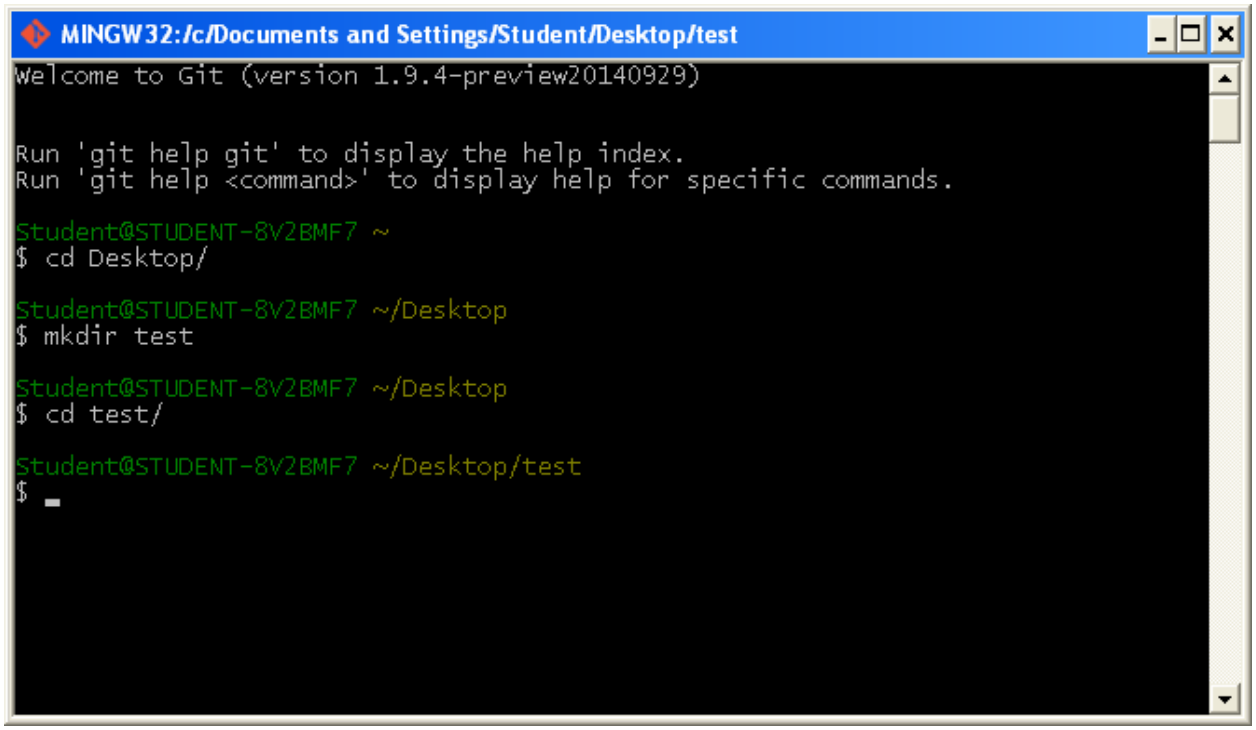
Paso 2 / Caso A: No inicializamos el repositorio con un archivo 'Readme':
En el caso de optar por esta opción la aplicación nos redirigirá a la siguiente vista:



En ella, la aplicación nos da diferentes opciones con las que inicializar nuestro repositorio, en la realización de este ejemplo usaremos la primera.

Para ello vamos a usar en todo momento, y a partir de ahora, 'Git Bash' con el editor de texto 'VIM'.

Primeramente vamos a crear una carpeta en la que inicializar nuestro repositorio Git, por ejemplo en nuestro escritorio. Lo podemos realizar fácilmente con las siguientes instrucciones:

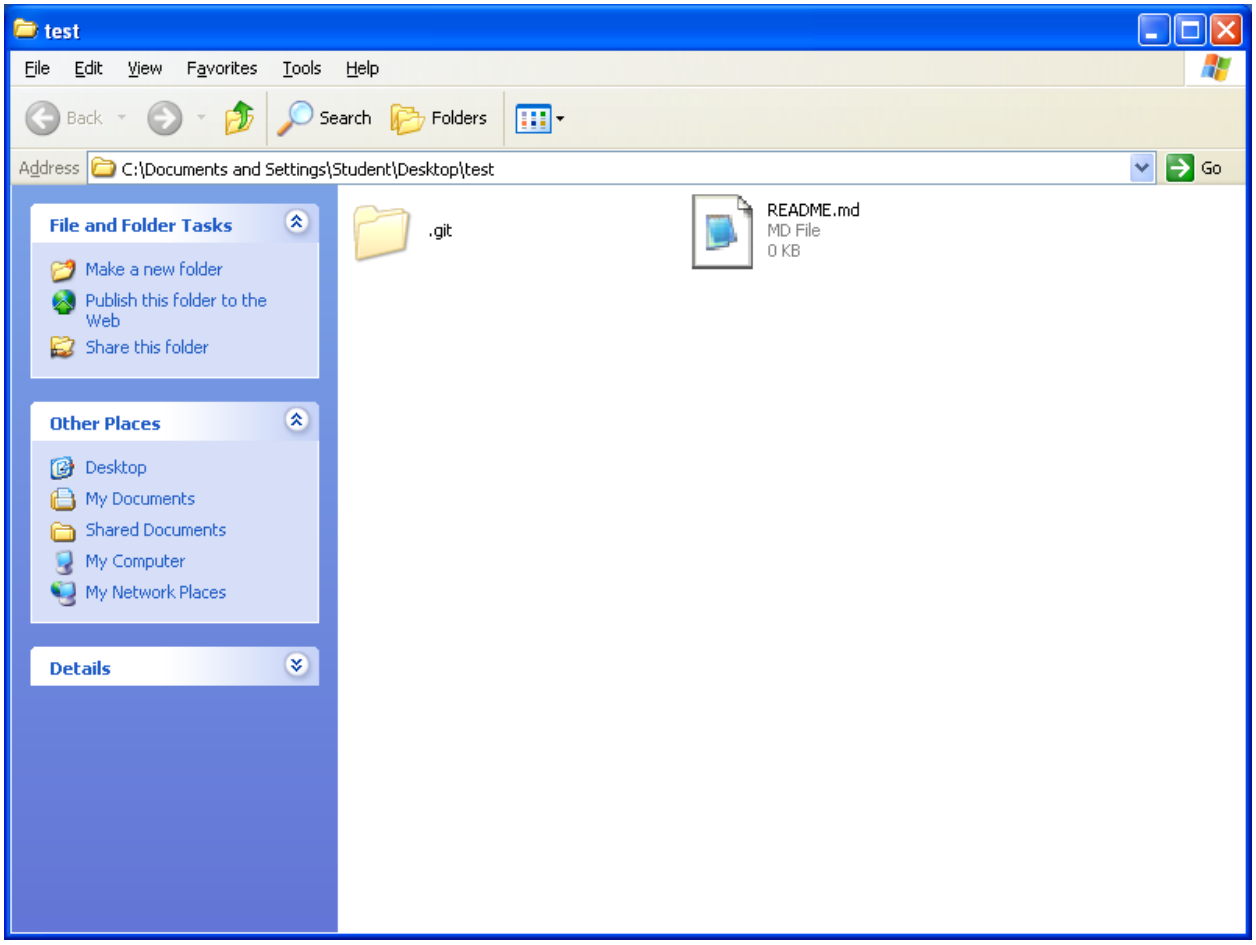


La carpeta 'test', alojada en nuestro escritorio, que acabamos de crear va a ser la que contenga nuestro repositorio Git.

Creamos un archivo 'README.md' con la instrucción 'touch README.md', podremos acceder a él y editarlo de la manera que creamos oportuna, añadiendo toda la información conveniente sobre nuestro proyecto alojado en el repositorio y los aspectos relacionados con el mismo, ya que en los siguientes pasos haremos commit (recordemos que estos se realizan en local) y un push al repositorio y dicho archivo será fácilmente accesible por todas las personas a las que le demos permisos en nuestro nuevo repositorio.

A continuación creamos un repositorio Git con la instrucción '**git init**'. Se nos mostrará en pantalla el mensaje de que se ha inicializado un repositorio Git vacío en la carpeta en la que nos encontremos actualmente.

Ahora, si vamos a nuestra carpeta y activamos la opción de mostrar carpetas ocultas, veremos que con esta instrucción se ha creado una carpeta oculta denominada '.git'. Dicha carpeta es muy importante ya que almacenará toda la información de nuestro repositorio. Además podemos ver el archivo 'README.md' que creamos unos pasos atrás.



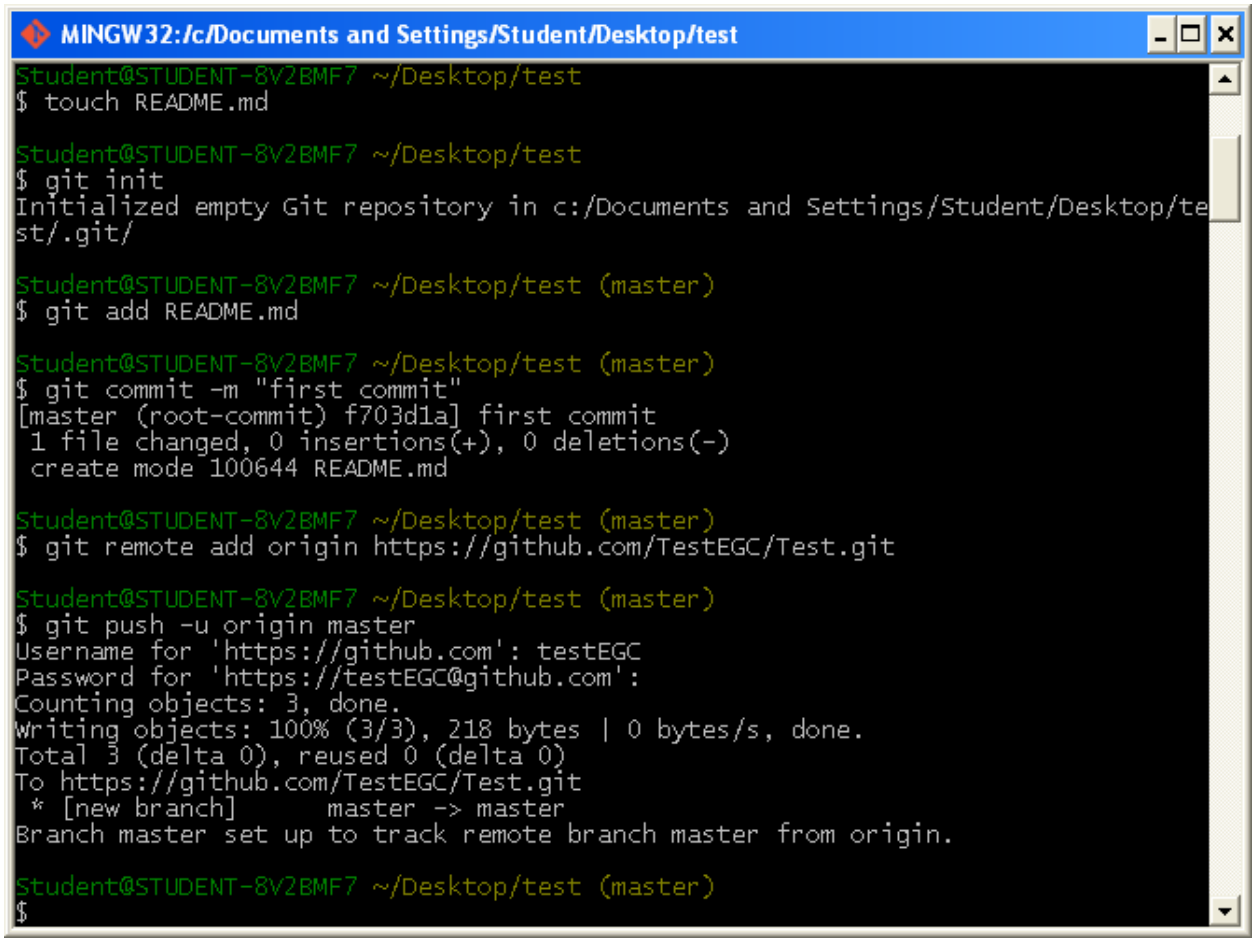
Ahora indicamos que el archivo que hemos creado previamente (README.md) forma parte del repositorio con la instrucción '**git add README.md**'.

Si queremos simplificar nuestra tarea, y siempre y cuando quisiéramos añadir todos los archivos que se encuentren en la carpeta que hemos creado a nuestro repositorio Git, podemos usar la instrucción '**git add .**'.

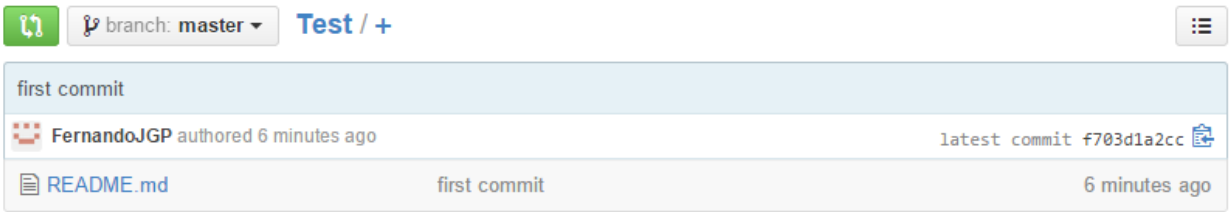
También podemos editar el archivo '**.gitignore**' para, aunque un archivo esté en la carpeta y hayamos indicado subir todos los de la misma, no subir dichos archivos al repositorio. En dicho archivo debemos añadir un archivo o patrón que ignorar por línea.

Con el resto de comandos que se nos indican en la web simplemente hacemos un commit y un push al servidor y, de esta manera, nuestro archivo ya estará en nuestro repositorio Git de 'GitHub'.

Si todo ha ido correctamente, ejecutando cada una de las instrucciones indicadas en la página de 'GitHub', obtendremos algo parecido a lo obtenido en esta captura:

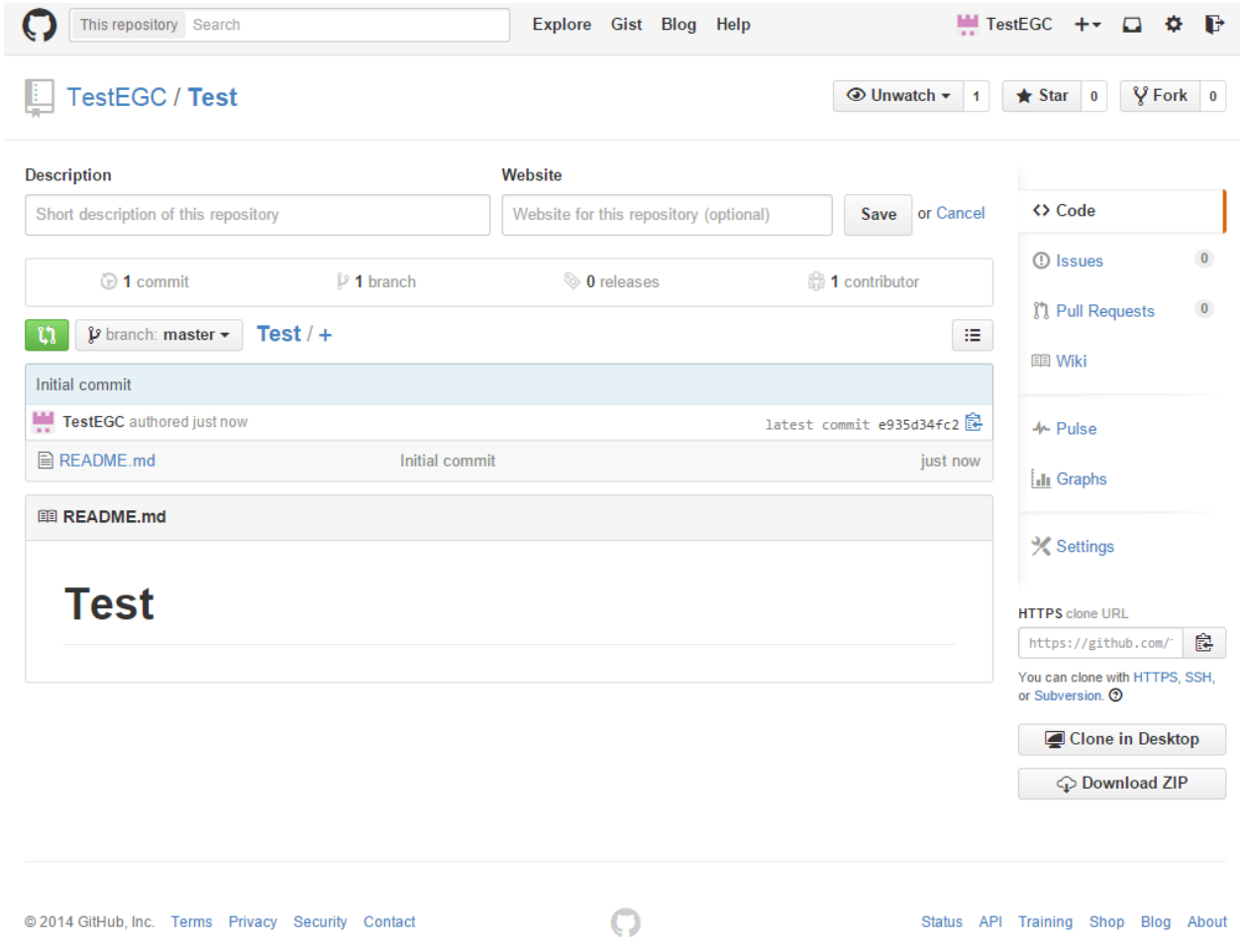


Ahora, si refrescamos la página en la que previamente 'GitHub' nos daba indicaciones de cómo inicializar nuestro repositorio, veremos que ha cambiado y ya aparece nuestro archivo (y todos los que hayamos decidido subir).



Paso 2 / Caso B: Inicializamos el repositorio con un archivo 'Readme':
Es la opción más sencilla, y al contrario que en la anterior no tendremos que realizar ningún paso más, por lo que habremos finalizado.

El aspecto que tendrá nuestro repositorio visto desde 'GitHub' será, por tanto, el siguiente:



Como hemos podido comprobar, la opción de inicializar el repositorio con un archivo 'README.md' tal y como nos permite 'GitHub' es mucho más fácil y rápida que la de iniciarlo manualmente por nosotros mismos, pero podemos tener muchas razones por la que querer que esto no sea así, por eso debemos indicarlo en este ejercicio.

Paso 3: Creación de una rama para el subsistema.

Una vez inicializado el repositorio vamos a proceder a lo que sería la creación de la rama para nuestro grupo.

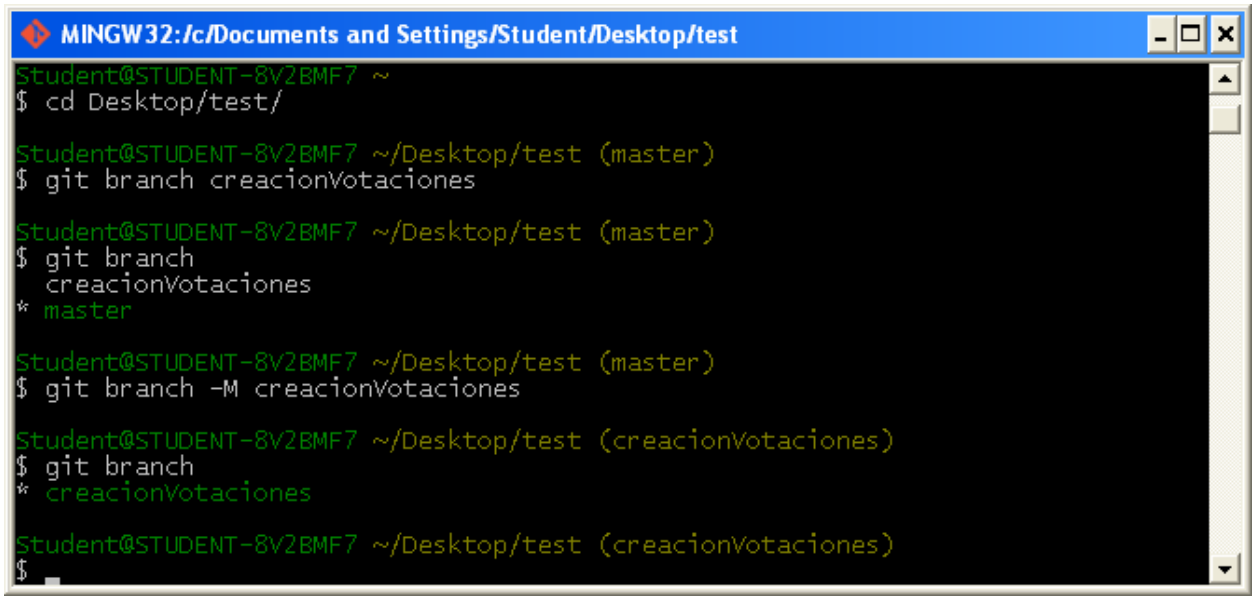
Si hemos realizado el paso 2 según el 'caso A' ya tendremos una carpeta creada en la ubicación que consideremos oportuna para nuestro repositorio, si no es así y hemos realizado el 'caso B', símplemente, creamos una carpeta, accedemos a ella y realizamos un '**git clone <url del repositorio>**' para traernos el contenido de nuestro repositorio y, lo que es más importante, la carpeta '.git'.

Ahora tendremos que utilizar la opción '**git branch <nombre>**' para crear una rama partiendo de la principal ('master').

En nuestro caso la rama se llamará '**creacionVotaciones**' que es el nombre de la rama de nuestro subsistema en el repositorio compartido utilizado para la realización de este proyecto.

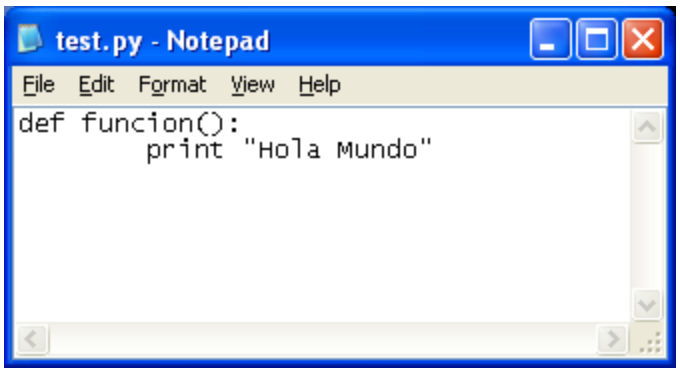
Ahora, si hacemos un '**git branch**' podremos ver las ramas presentes en el repositorio:

Finalmente nos movemos a ella con '**git branch -M creacionVotaciones**'



Podemos ver cómo en verde en el momento que ejecutamos la instrucción '**git branch**' se nos muestra la rama que tenemos seleccionada actualmente.

Antes de continuar, vamos a añadir un fichero a esta rama que es el que usaremos para la realización del ejercicio. Lo hacemos de la misma forma vista en el 'caso A' del apartado anterior, y lo denominaremos 'test.py', por tanto ejecutamos la instrucción '**touch test.py**' y lo modificamos. Para este ejemplo lo hemos añadido el siguiente contenido:

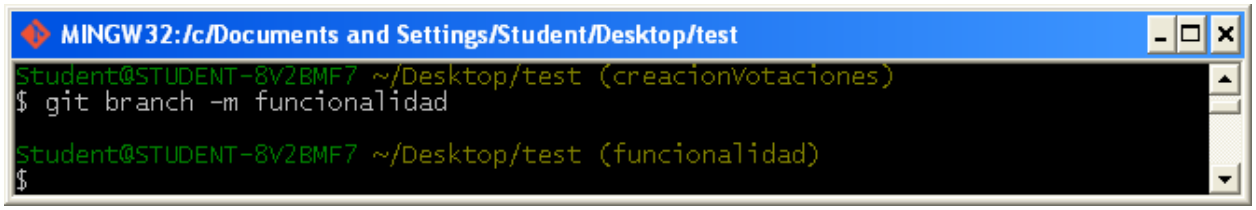


Creo que el ejercicio está muy detalladamente descrito pero es muy simple
Sería más interesante hacer un ejemplo real de cambio y que tuviera más complejidad

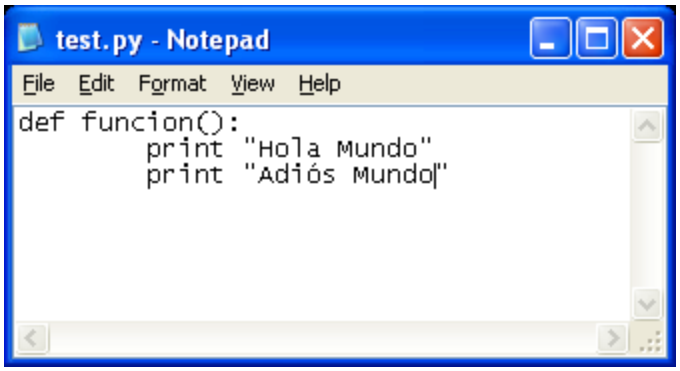
Ahora le tenemos que decir al repositorio que le siga la pista con '**git add test.py**' y, seguidamente, hacemos un commit, por ejemplo: '**git commit -m "añadido fichero test.py"**'.

Paso 4: Creación de una rama de funcionalidad partiendo de la de subsistema.

Esta vez vamos a crear y a movernos a la rama a la vez. Para eso tenemos que usar el comando '**git branch -m <nombre>**'. En el caso de este ejemplo, la nueva rama se llamará 'funcionalidad', por tanto ejecutaremos '**git branch -m funcionalidad**'.



Ahora ya podemos empezar a añadir funcionalidades a la pequeña función en python que hemos diseñado antes para este ejemplo. Por ejemplo, añadiremos que la función imprima también "Adiós mundo" por pantalla:



Como ya hemos terminado de añadir nuestra funcionalidad, hacemos un commit en esta rama (funcionalidad): '**git commit -m "Añadida la funcionalidad de mostrar Adiós mundo"**

Paso 5: Merge a la rama del subsistema y cerrado de la de funcionalidad.

Para este paso nos situamos en la rama a la cual le queremos hacer el merge, en nuestro caso a 'creacionVotaciones' ('**git checkout creacionVotaciones**') y ejecutamos la instrucción '**git merge <nombre de la rama>**'. En nuestro caso, el nombre de la rama será 'funcionalidad'.

Ahora, podemos borrar la rama de funcionalidad con la instrucción '**git branch -d funcionalidad**'

Paso 5: Merge con la rama global del proyecto.

Este último paso debe resultarnos muy sencillo, ya que no es nada que no hayamos hecho en el anterior: Debemos situarnos en la rama master y ejecutar la instrucción de merge allí.

Tan fácil como ejecutar las siguientes instrucciones:

- '**git checkout master**', con la cual nos moveríamos a la rama 'master'.
- '**git merge creacionVotaciones**' con la que realizaríamos la unión de las ramas.

Tras esto, ya tendríamos los cambios realizados en la rama de nuestro subsistema en la rama principal donde se encuentran todos los subsistemas de la aplicación y por tanto cumplido el objetivo de este ejercicio.

El ejercicio está bastante comple, detallado y eso es de agradecer pero en términos "teóricos" y de procesos éste capítulo está bastante pobre y debería ampliarse y trabajarse.

4. Gestión de la construcción e integración continua

La gestión de la construcción consiste en decidir qué procesos se van a seguir a la hora de compilar/construir el código.

Nuestro grupo hará uso del Entorno de Desarrollo Integrado (IDE) Eclipse Luna, el cual facilita una herramienta de construcción automática de proyectos codificados en Java. Esta herramienta de construcción ejecuta el archivo javac.exe proporcionado por el Kit de Desarrollo Java 7 (JDK 7). Durante el proceso de compilación Java, el código fuente de alto nivel de los archivos con extensión .java se transforma en archivos de código intermedio con extensión .class que puede ser interpretado por la máquina virtual java o desplegado en un servidor de aplicaciones. A este código intermedio se le conoce como bytecode.

En nuestro caso usamos Maven 3.1.0, una herramienta de software para la gestión y construcción de proyectos Java que además facilita enormemente la gestión de dependencias. Para la integración de Maven con Eclipse, se necesita la instalación del plugin M2E-WTP, que está disponible a través del Eclipse Marketplace.[1]

Se mezcla demasiado la descripción de herramientas en este punto. Eso debería en "mapa"

Además de las citadas herramientas y dado que nuestro proyecto se encuentra alojado en un repositorio GitHub, tenemos instalado un plugin de integración de Git con Eclipse llamado EGit[2]. Como en el caso de M2E-WTP, también está disponible en el Eclipse Marketplace.

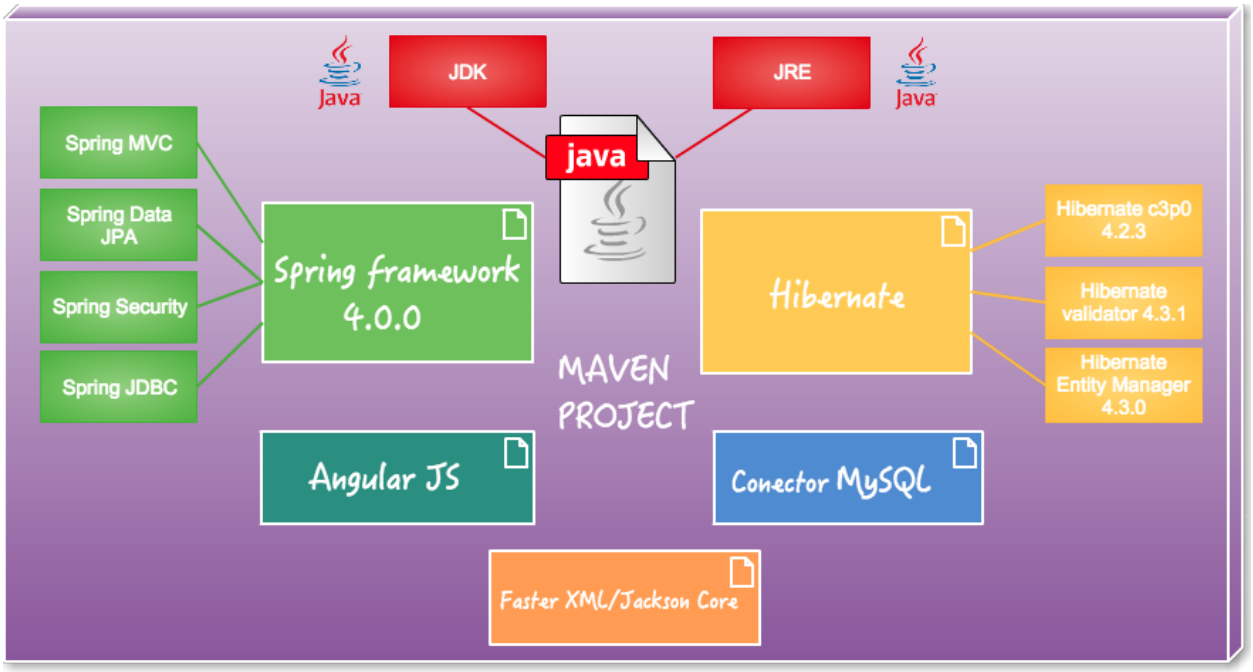
Con respecto al servidor de base de datos, emplearemos MySQL Community Server 5.5, que incluye un buen número de conectores Java y un marco de trabajo en el que gestionar nuestra base de datos.

El servidor de aplicaciones elegido para el despliegue de la aplicación es Tomcat 7.0, un servidor de sobras conocido y del que existe una cantidad ingente de documentación en internet. El entorno de desarrollo Eclipse también proporciona plugins para el despliegue de la aplicación en Tomcat.

[1] Si quiere obtener más información sobre Maven o sobre el plugin de integración con Eclipse, puede visitar los siguientes enlaces: *http://es.wikipedia.org/wiki/Maven* *https://www.eclipse.org/m2e-wtp/*

[2] Si desea obtener más información sobre este plugin puede acceder a http://eclipse.org/egit/.

Entre las dependencias más importantes de nuestro proyecto se encuentran:



1. Spring framework 4.0.0

Spring es un framework de código abierto para el desarrollo de aplicaciones Java. Entre sus principales características está el uso de la inversión del control a través de la inyección de dependencias y la programación orientada a aspectos, aunque dispone de gran variedad de módulos que proveen un buen rango de servicios[3]. Entre los más importantes de los que hacemos uso se encuentran:

- Spring MVC(Modelo-Vista-Controlador). Gestiona el flujo de información web.
- Spring Data JPA: Proporciona soporte con la capa de acceso a datos implementada con JPA.
- Spring Security: Gestiona el control de acceso y la autentificación.
- Spring JDBC: "*Se encarga de los detalle de bajo nivel, empezando por abrir la conexión, preparar y ejecutar las sentencias SQL, procesar las excepciones, manejar las transacciones y finalmente cerrar la conexión.*"[4]

[3] Información obtenida de http://es.wikipedia.org/wiki/Spring_Framework
[4] Esta definición está tomada de http://www.tutorialspoint.com/spring/spring_jdbc_framework.htm

todo ésto no viene a cuento en este apartado

2. Hibernate.

- Hibernate c3p0 4.2.3: Es un Data Source que gestiona las múltiples conexiones concurrentes a la base de datos.
- Hibernate validator 4.3.1: Framework de implementación de validaciones a entidades basado en anotaciones.
- Hibernate Entity Manager 4.3.0: La definición de la documentación oficial de la comunidad nos dice que Hibernate Entity Manager "*Implementa las interfaces de programación y las reglas de ciclo de vida tal y cómo se definen en la especificación JPA 2.0*". [5]

3. Conector MySQL 5.1.26

MySQL proporciona este componente, que permite a nuestra aplicación conectarse con el servidor de base de datos.

4. Angular JS

Según la página web oficial de Angular:

"*AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. Angular's data binding and dependency injection eliminate much of the code you would otherwise have to write. And it all happens within the browser, making it an ideal partner with any server technology*"[6]

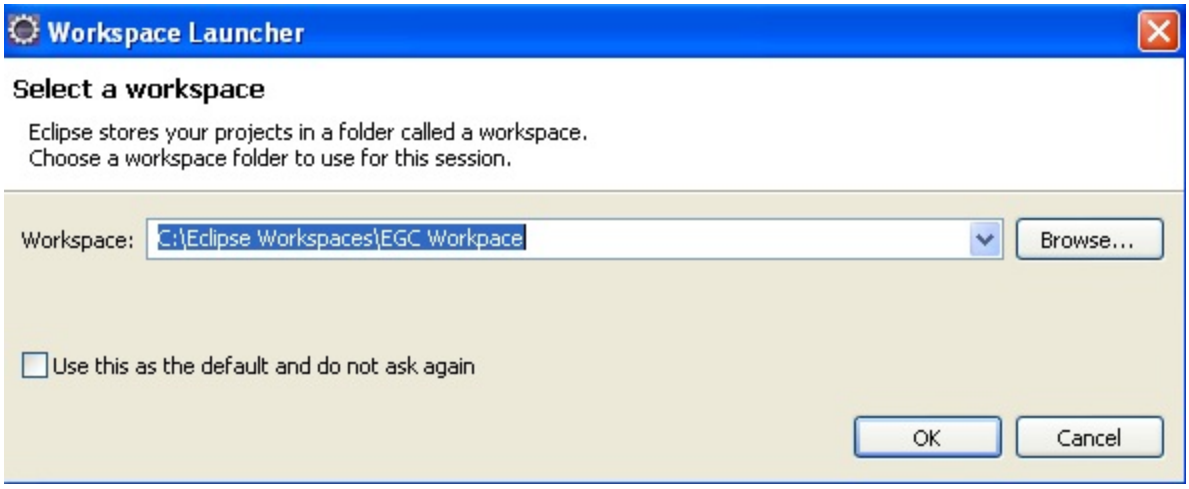
5. Faster XML/ Jackson core

Es una librería Java multipropósito para el procesado del formato de datos JSON.[7]

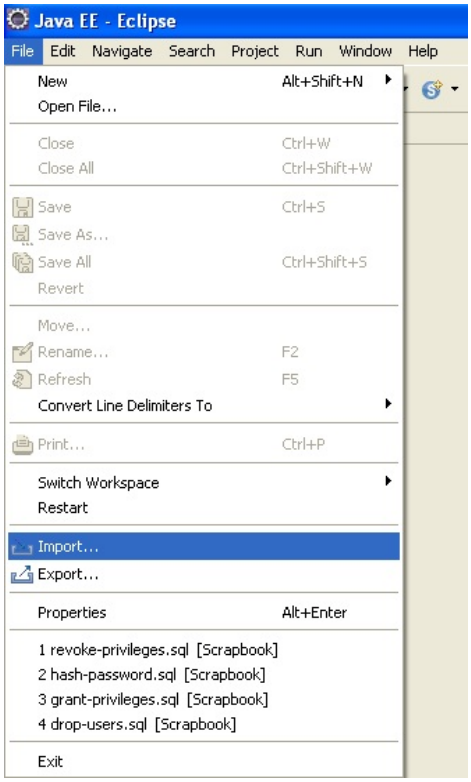
[5] Tomada de https://docs.jboss.org/hibernate/entitymanager/3.6/reference/en/html_single/#d0e61
[6] Definición obtenida de https://docs.angularjs.org/guide/introduction
[7] Si desea obtener más información, visite la página web http://wiki.fasterxml.com/JacksonHome

Ejercicio de construcción del proyecto

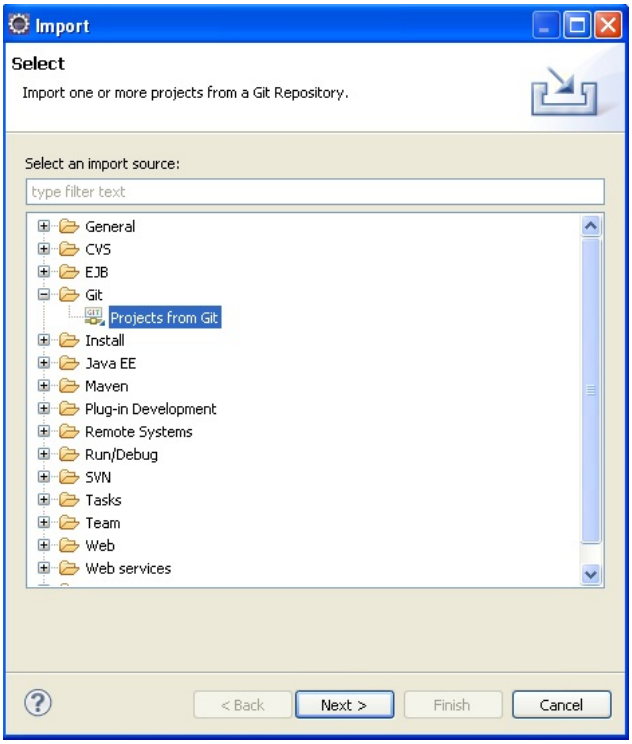
Paso 1: Ejecutamos Eclipse y elegimos el Workspace donde vamos a alojar el proyecto.



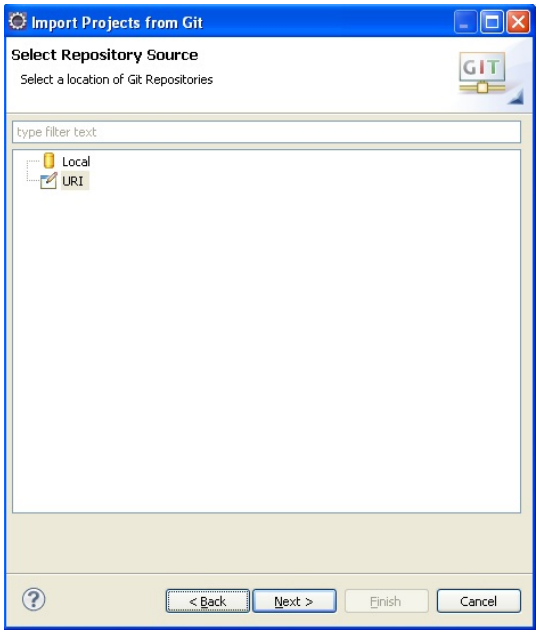
Paso 2: Seleccionamos File - Import



Paso 3: Seleccionamos que queremos importar un proyecto desde Git. Pulsamos Next.

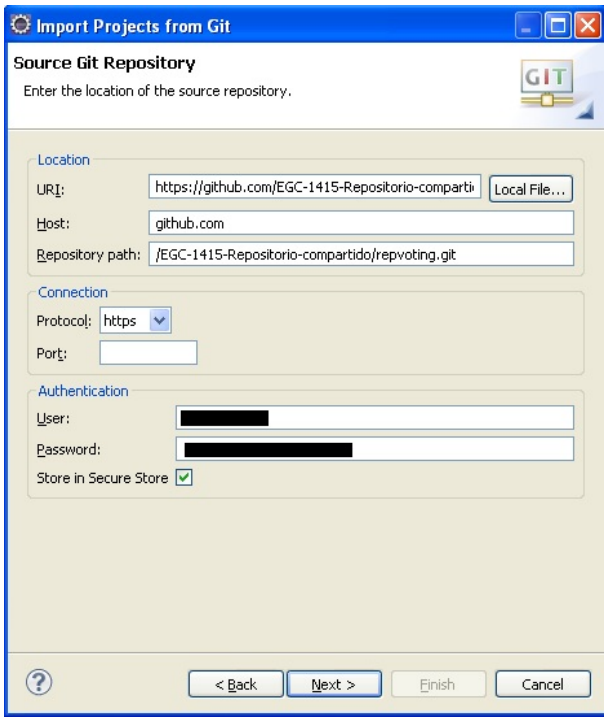


Paso 4: Elegimos "URI" como fuente del repositorio. Pulsamos Next.

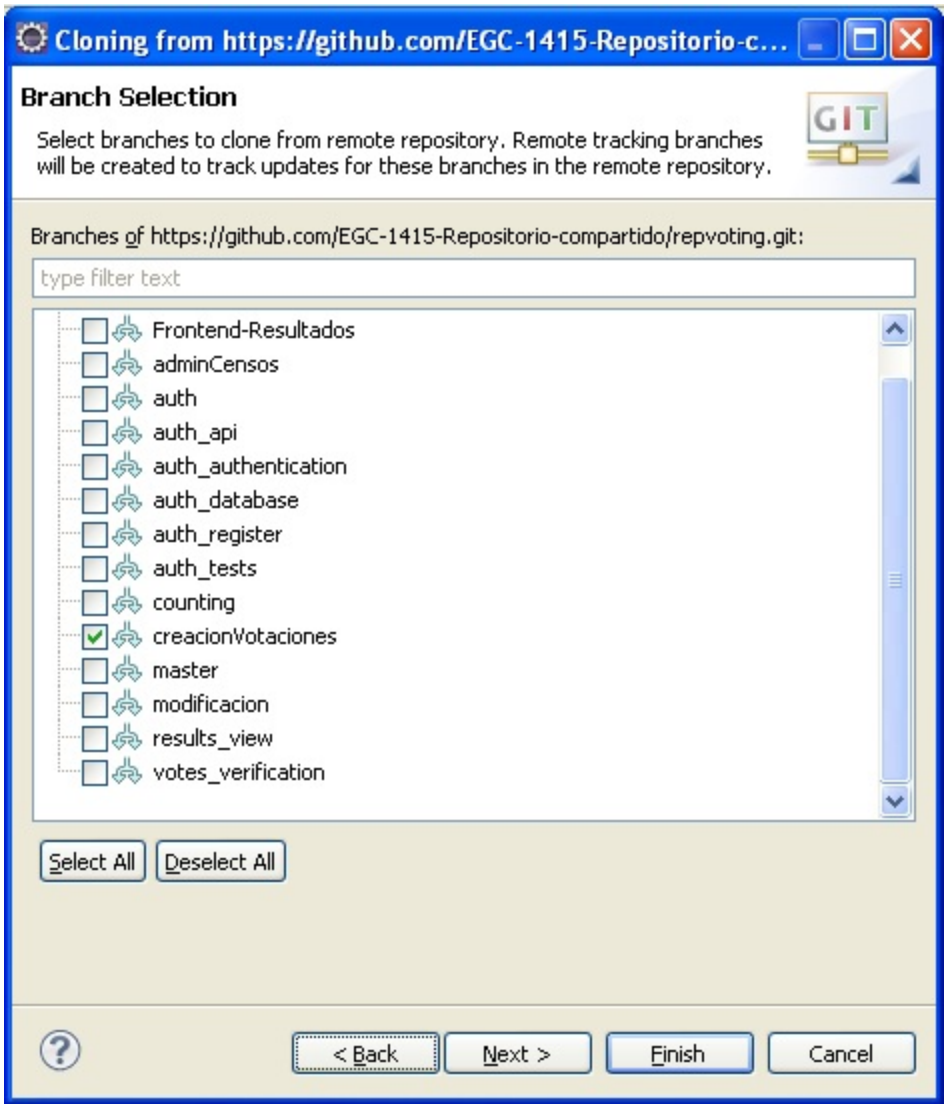


Paso 5: En la ventana de localización del repositorio, introducimos la URI del repositorio compartido por los grupos en la asignatura, así como nuestro usuario y contraseña de GitHub.

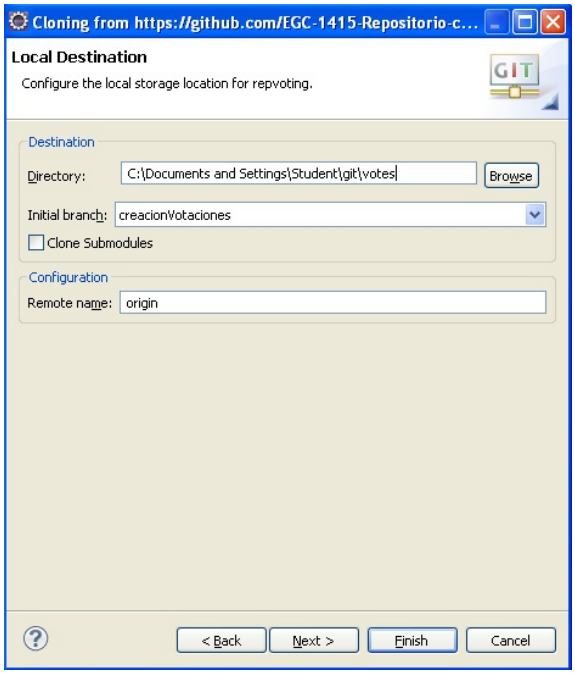
URI: https://github.com/EGC-1415-Repositorio-compartido/repvoting.git



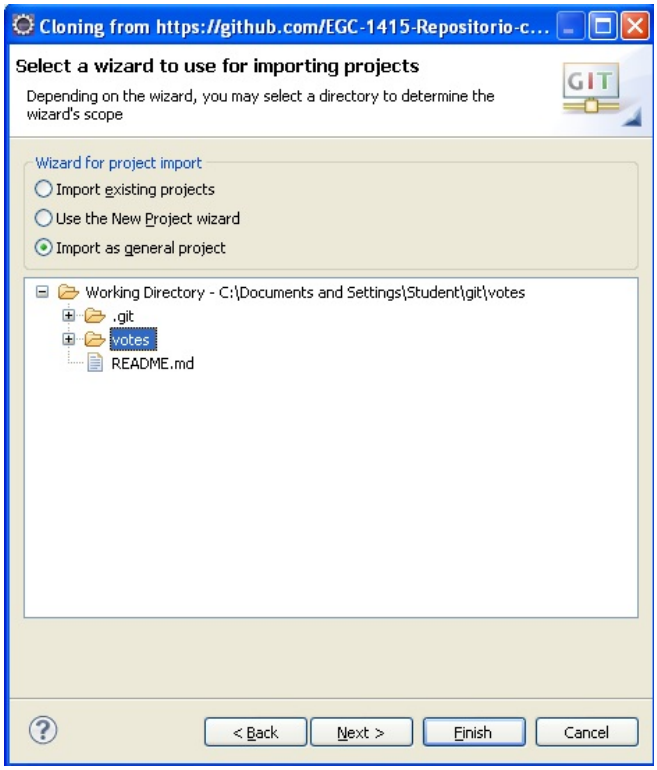
Paso 6: En la pantalla de selección de rama, seleccionamos *creacionVotaciones* y pulsamos Next.



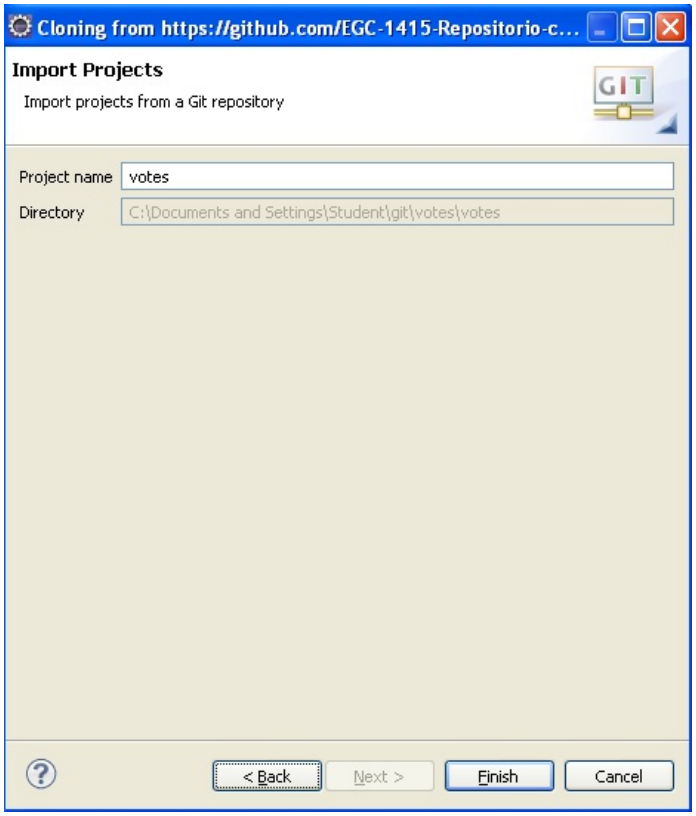
Paso 7: En la pantalla de destino local, elegimos el directorio donde será almacenado el proyecto Git. Además, indicaremos que nuestra rama inicial será *creacionVotaciones* y como nombre remoto *origin*.



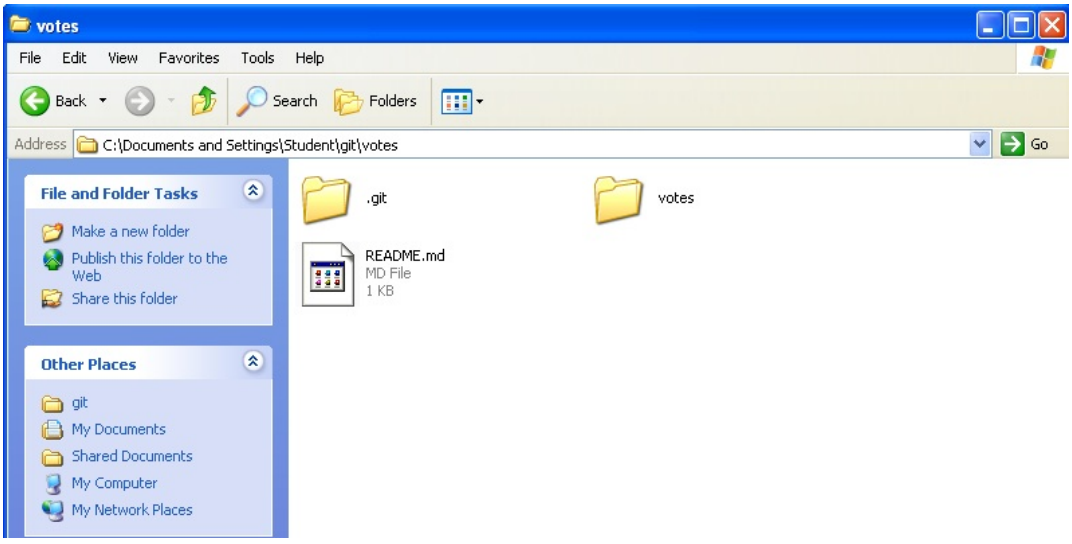
Paso 8: En esta pantalla elegiremos *Import as General Project* y seleccionamos la carpeta *votes*. Pulsamos Next.



Paso 9: En esta pantalla nombramos el proyecto como *votes* y pulsamos Finish.

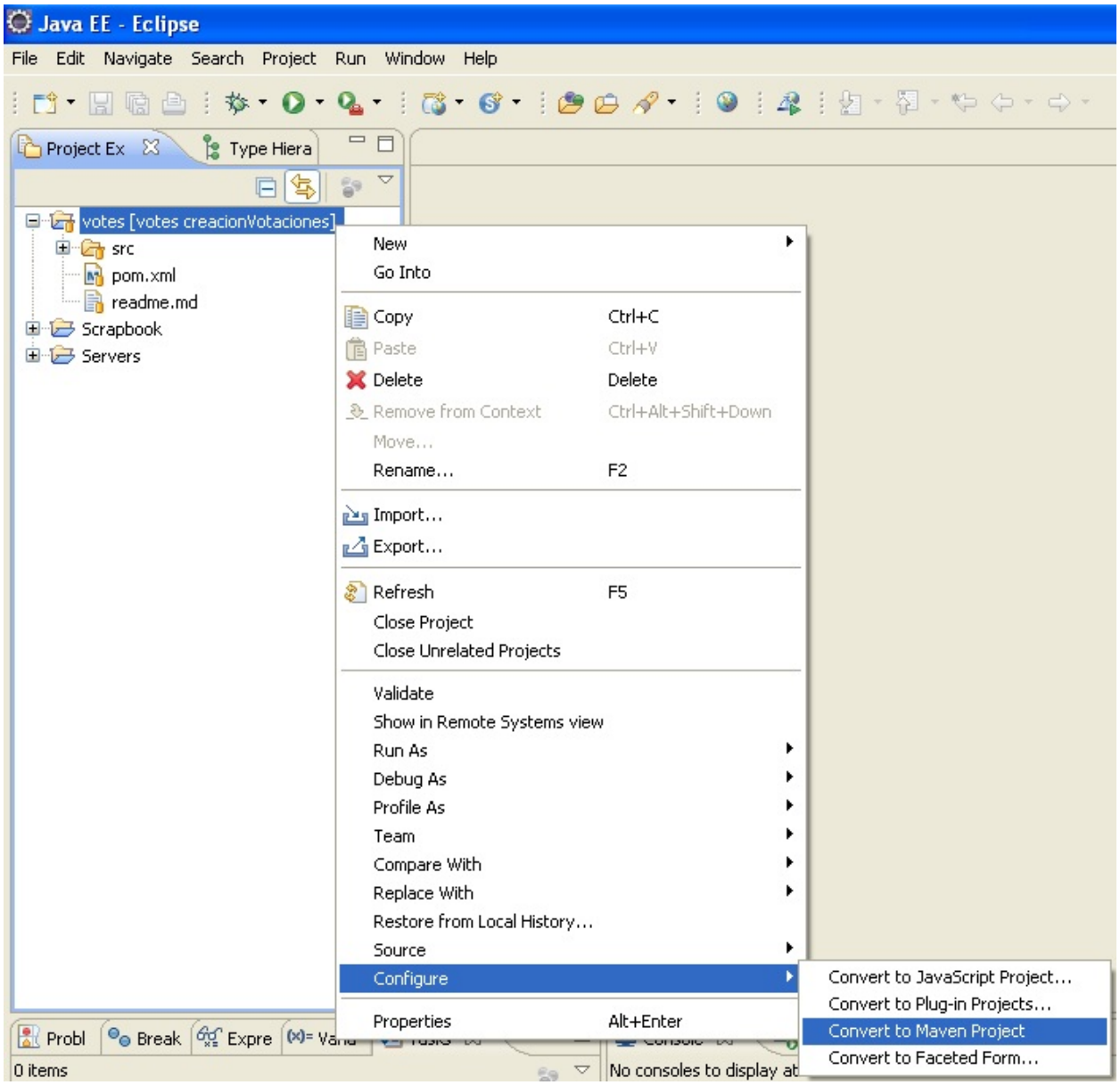


Paso 10: Una vez que se clonado el proyecto, podemos irnos al directorio local que hemos seleccionado y ver tanto el proyecto clonado como los archivos de configuración inicial de Git.



Paso 11: Volvemos a Eclipse y pulsamos con el botón derecho el proyecto votes-Configure-Convert to Maven Project. Esto convertirá nuestro proyecto en un proyecto Maven, y descargará las librerías necesarias especificadas en nuestro archivo pom.xml, y se almacenarán en el repositorio local. Por defecto, este repositorio local es C:/Documents and settings/Usuario/.m2/repository.

Tras la actualización de dependencias, Eclipse construirá el proyecto automáticamente.



La integración se ha realizado por partes, comenzando por los subsistemas más cercanos, como pudieron ser Censos o Autenticación, una vez integrados estos comenzamos la integración con el resto, dejando para el final los subsistemas que presentaban un código menos estable o tecnologías diferentes.

La integración con el resto de subsistemas, al principio se realizó de manera manual, actualizando git y manteniendo todos los proyectos posibles en Eclipse. El mayor problema de este "método" de integración es que era habitual olvidar actualizar algunos proyectos lo que provocaba diferentes comportamientos de nuestros subsistemas dependiendo de quien lo ejecutase. Finalmente, se ha optado por utilizar Jenkins, esta herramienta nos ha permitido realizar la integración por partes y de manera mucho más cómoda. En nuestro caso hemos optado por una tarea que se situase en el master del repositorio compartido, que lo mantuviese actualizado y que desplegase cada uno de los subsistemas que se deben integrar con el fin de que se pueda lanzar una batería de pruebas sobre esos subsistemas. Además se han automatizado todas las tareas, desde desplegar los subsistemas en tomcat hasta generar la librería de "Verificación" a partir de su código fuente.

Para facilitar la integración del resto de subsistemas con nosotros, hemos implementado una API muy sencilla:

Esto no debería ir aquí. Una sección a parte en la que se hable de es.

1. Guardar una votación:

METHOD: POST -> .../vote/save.do

```
{"type":"survey",
 "usenarmeCreator":"marperrei1",
 "title":"survey1",
 "description":"survey1Sample",
 "startDate":"24/11/2014",
 "endDate":"27/11/2014",
 "questions":[
    {"type":"question",
     "text":"question1"},
    {"type":"question",
     "text":"question2"}
 ]}
```

2. Recuperar una votación:

METHOD: GET ~> ..../vote/survey.do?id= *
{"type":"survey", **"id":"*",** **"usenarmeCreator":"marperrei1",** **"title":"survey1",** **"description":"survey1Sample",** **"startDate":"24/11/2014",** **"endDate":"27/11/2014",** **"questions":[** **{"id":"X",** **"type":"question",** **"text":"question1"},** **{"id":"Y",** **"type":"question",** **"text":"question2"}** **]}**

3. Recuperar todas las votaciones:

METHOD: GET ~> ..../vote/allSurveys.do?id= *
{{"type":"survey", **"id": "XXX",** **"version": "YYY",** **"usenarmeCreator":"marperrei1",** **"title":"survey1",** **"description":"survey1Sample",** **"startDate":"24/11/2014",** **"endDate":"27/11/2014",** **"questions":[** **{"type":"question",** **"id": "X0X0X0",** **"version": "Y0Y0Y0",** **"text":"question1"},** **{"type":"question",** **"id": "X1X1X1",** **"version": "Y1Y1Y1",** **"text":"question2"}** **]** **},** **{"type":"survey",**

```
"id": "XXX",
"version": "YYY",
"usenarmeCreator":"marperrei1",
"title":"survey1",
"description":"survey1Sample",
"startDate":"24/11/2014",
"endDate":"27/11/2014",
"questions":[
   {"type":"question",
    "id": "X0X0X0",
    "version": "Y0Y0Y0",
    "text":"question1"},
   {"type":"question",
    "id": "X1X1X1",
    "version": "Y1Y1Y1",
    "text":"question2"}
]
}}
```


4. Recuperar todas las votaciones finalizadas

METHOD: GET -> ..../vote/finishedSurveys.do?id=*


```
{{"type":"survey",
"id": "XXX",
"version": "YYY",
"usenarmeCreator":"marperrei1",
"title":"survey1",
"description":"survey1Sample",
"startDate":"24/11/2014",
"endDate":"27/11/2014",
"questions":[
   {"type":"question",
    "id": "X0X0X0",
    "version": "Y0Y0Y0",
    "text":"question1"},
   {"type":"question",
    "id": "X1X1X1",
    "version": "Y1Y1Y1",
    "text":"question2"}
]
},
{"type":"survey",
"id": "XXX",
```

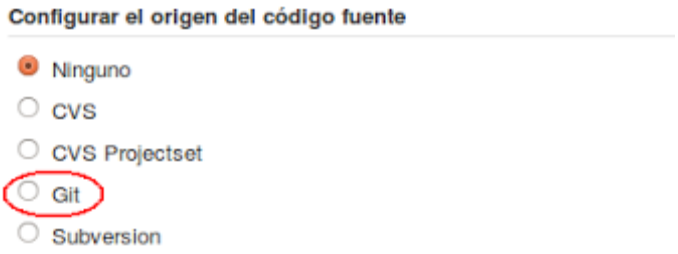
```
"version": "YYY",
"usenarmeCreator":"marperrei1",
"title":"survey1",
"description":"survey1Sample",
"startDate":"24/11/2014",
"endDate":"27/11/2014",
"questions":[
   {"type":"question",
    "id": "X0X0X0",
    "version": "Y0Y0Y0",
    "text":"question1"},
   {"type":"question",
    "id": "X1X1X1",
    "version": "Y1Y1Y1",
    "text":"question2"}
]
}}
```

Ejercicio de integración con Jenkins Antes no se ha hablado de eso.

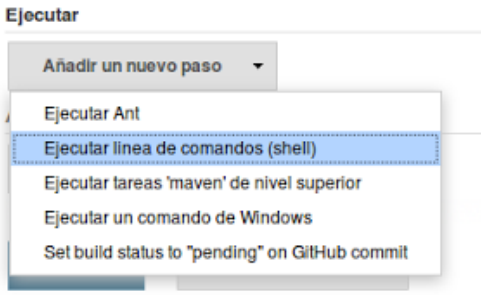
Para la realización de este ejercicio vamos a suponer que nuestro equipo tiene instalado Jenkin junto con el plugin para GitHub. Si se utiliza la máquina virtual que hemos aportado no existirá ningún tipo de problema, para acceder a Jenkins basta con escribir esta dirección: "http://localhost:8090/"

Una vez hayamos entrado pulsamos "Nueva Tarea" y le ponemos como nombre, por ejemplo, "ejercicio_jenkins" y tipo de tarea "Estilo libre".

Una vez que hayamos hecho esto, veremos el panel de configuración de la tarea. Como los subsistemas que tenemos que integrar se encuentran en GitHub, vamos a importar los proyectos desde un repositorio Git. Para ello, seleccionamos como origen del código fuente Git e indicamos la URL de nuestro repositorio: **https://github.com/EGC-1415-Repositorio-compartido/repvoting.git**



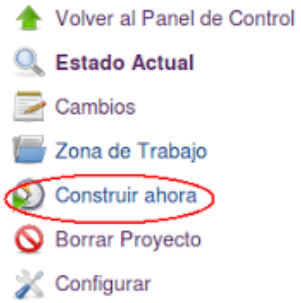
Cuando hayamos hecho esto, podremos añadir scripts de ejecución que interactúen con los archivos que acabamos de descargar. De esta forma podremos desplegar los proyectos, compilarlos, etc.



Ahora, añadiremos al cuadro de texto el siguiente script:

```
#Ve a Master y actualiza
git checkout master
git pull

#Copia el subsistema de autenticacion genera su BD
cp -Rf auth /opt/lampp/htdocs
mysql -uroot < /opt/lampp/htdocs/auth/auth_DB_script.sql

#Compila el codigo de verificacion y crea el jar
javac -encoding latin1 -g verification/verificacion/src/main/*.java
cd verification/verificacion/src
jar cvf verificacion.jar main/*.class
cd ../../../

#Instala la dependecia en maven
mvn install:install-file -Dfile=verification/verificacion/src/verificacion.jar
-DgroupId=verificacion -DartifactId=main -Dversion=1.0 -Dpackaging=jar

#Compila y despliega censos
cd census/ADMCensus/
mvn compile war:war
cd target
rm -f /var/lib/tomcat7/webapps/ADMCensus.war
mv *.war /var/lib/tomcat7/webapps/ADMCensus.war
cd ../../../

#Compila y despliega votaciones
cd votes
mvn compile war:war
cd target
rm -f /var/lib/tomcat7/webapps/CreacionAdminVotaciones.war
mv *.war /var/lib/tomcat7/webapps/CreacionAdminVotaciones.war
cd ../../
```

Este script, es un fragmento del que se ha utilizado para nuestro proyecto, gracias a él, podemos desplegar estos proyectos para más tarde lanzar las pruebas que comprueben el estado de la aplicación.



Cuando pulsemos en "Construir ahora", Jenkins ejecutará el script, si todo ha ido bien tendremos junto al identificador del "build" un icono azul, si no este icono será rojo.

Se podría mejorar bastante esta parte también añadiendo más cosas a Jenkins y realmente describiendo el proceso de integración continua, por ejemplo, en tema de notificaciones y demás cosas que están en las transparencias de clases.

5. Gestión del cambio, incidencias y depuración

Para la gestión del cambio, nuestro subgrupo tiene establecido un protocolo bien definido. Cada Lunes tenemos una reunión todos los miembros del grupo. Una parte de este tiempo es invertido en identificar las posibles mejoras, ampliaciones, cambios en el proyecto que sean necesarios, incidencias, etc. Si detectamos cambios urgentes en los que se precisa estar el grupo completo, o al menos gran parte, nos comunicamos principalmente a través de un grupo de whatsapp para acordar día y hora para reunirnos. Hay excepciones en las que hemos usado el correo y algún sms debido a que no todos los integrantes del grupo tienen conexión a Internet desde sus móviles sin Wi-fi.

Este ha sido el protocolo aplicado durante todo el desarrollo del subsistema de "Creación y Administración de Votaciones". Consideramos que ha tenido éxito la forma en la que nos hemos comunicado y los resultados obtenidos, a pesar de la dificultad de compatibilizar los horarios de todos los integrantes del grupo.

Este protocolo no ha sido utilizado sólo para los realizados sobre el código, también lo hemos aprovechado para cualquier tipo de cambio en el proyecto en general.

Los roles que hemos tenido han sido variados entre ciertos componentes del equipo, pero dos han sido fijos durante todo el desarrollo. Hemos tenido dos desarrolladores principales durante el desarrollo del proyecto, ayudados por algunos de los componentes del equipo para temas como CSS, Drivers JSON, etc. En cada reunión, un miembro del equipo comprobaba cada iteración del proyecto y junto con otro miembro, que simulaba ser cliente, y se realizaban los casos de uso correspondientes. Dichos casos de uso, deberían haber sido escritos oficialmente en algún documento, pero utilizando las anotaciones de clase y los pocos casos de uso que hay y que son muy intuitivos, dado que todos conocemos el proceso de votación, no los hemos documentado como tales.

A veces, esta comprobación de casos de uso se realizaba con otra pareja más, realizando las mismas tareas, con el fin de realizar una depuración más a fondo. Además, dado que somos muchos componentes en el grupo para un proyecto pequeño, a veces hemos duplicado tareas. Esto no sería rentable para una empresa, pero en un ámbito didáctico nos ha venido bien para encontrar algunos bugs y/o mejoras que no habían detectado otros.

La depuración se realiza con el debug de Eclipse y la consola de desarrollador. En caso de dudas complejas, se consultaba a los desarrolladores principales.

El resultado de estas comprobaciones se comunicaba a otro miembro del grupo, el cual transmitía los resultados a todo el grupo. De esta manera, el grupo está informado de los posibles bugs y cosas que mejorar para posteriormente realizar un debate con dicha información. De esta reunión, salían los cambios a realizar, su prioridad y la necesidad de crear nuevas ramas en el proyecto, crear una línea base con la parte estable, etc. En caso de no estar de acuerdo en algo, se realizaba una votación y la mayoría se imponía. De esta manera, establecemos el estado actual del proyecto y podíamos hacernos una idea global de la situación.

Los cambios aprobados, si no eran urgentes, teníamos toda la semana para llevarlos a cabo. Si eran urgentes debido a que terceras personas dependían de nosotros o porque teníamos que tenerlos terminados para mitad de semana, teníamos que buscar los huecos entre clases necesarios para realizarlos o bien hacerlos desde cada.

Las políticas para descartar los cambios, a pesar de ser un proyecto pequeño hemos tenido ciertos problemas. Hemos dado primero prioridad a problemas de funcionalidad, ya sea por errores nuestros, cambios en los requisitos o que éstos estuviesen poco claros. Los siguientes cambios eran los relacionados con la integración con otros grupos y la forma de ofrecer o recoger los datos. Y por último, los cambios menores como por ejemplo las CSS u orden en que deben aparecer los elementos en la pantalla tenían una prioridad menor. Los problemas nos han llegado principalmente por causas ajenas a nosotros, ya que desde un principio los requisitos no han estado totalmente claros, ha habido cambios casi semanalmente y ha reinado un poco el caos. Aún así, hemos actuado bien y hemos podido lograr que todo funcione en los plazos correspondientes.

Para la gestión del cambio a nivel del sistema completo (con todos los subsistemas), tras varias reuniones se llegó al acuerdo de estar todos juntos en un mismo repositorio compartido en github. De esta manera, cuando surgen errores o faltan cosas creamos un nuevo issue en github y lo asociamos al responsable del grupo en cuestión. Además, desde la última reunión que tuvieron los responsables de cada grupo la penúltima semana de clase, se acordó que cualquier cambio se avisará con un issue que incluya a todos los miembros de la organización. Esto se acordó así debido a que ya están todos los subsistemas casi terminados y cualquier cambio en uno de ellos podría afectar al resto, lo que podría tener un efecto muy negativo en el desarrollo del proyecto. Las peticiones de cambio por parte de algún subsistema a otro, también se llevarán a cabo siguiendo el mismo método.

Todas estas políticas comentadas en este apartado no se han podido llevar a cabo durante las primeras semanas de clases, ya que tuvimos que pensar la

Hay muchas partes en las que se explican cosas demasiado y hacen el texto más complejo

forma más óptima de hacerlo entre todo el caos que reinaba en la asignatura al principio.

En un principio consideramos la posibilidad de hacer la gestión del proyecto con JIRA[8], una aplicación web destinada a el seguimiento de errores, de incidentes y la gestión operativa de proyectos. Al igual que muchas otras herramientas de gestión de proyectos, JIRA también se utiliza para la administración de tareas.

Esta herramienta fue desarrollada por la empresa australiana Atlassian. Inicialmente, JIRA se utilizó para el desarrollo de software, sirviendo de soporte para la gestión de requisitos, seguimiento del estado y posteriormente para la gestión de errores.

JIRA es una aplicación extremadamente flexible[9] que permite coordinar y controlar procesos semi estructurados. Una vez que el equipo de trabajo está familiarizado con el sistema y a medida que va definiendo procesos de trabajo, JIRA puede transformarse en un motor de procesos modelable de acuerdo a dichos procesos. Es decir, JIRA permite comenzar con una solución simple y flexible, para luego evolucionar a un sistema de procesos modelable y estructurado.

Nos decantamos por JIRA dado que todas las referencias a esta herramienta eran positivas. Además, nos atraía su interfaz, intuitiva y familiar. No obstante, ya en los primeros pasos con la herramienta comenzaron a surgir inconvenientes. La versión en la nube es de pago y la prueba tiene una duración de sólo siete días, tiempo a todas luces insuficiente. La versión en servidor tiene una prueba gratuita de tiempo indefinido. Decidimos descargarla para darle una oportunidad probando en local, en lo que buscábamos la forma de usarlo de forma compartida entre los miembros del grupo.

La instalación de JIRA no entraña dificultad, tampoco el uso de las características básicas, como se ha dicho es bastante intuitiva. No es necesario invertir un tiempo excesivo en familiarizarse con JIRA, si bien a veces resulta complicado encontrar ciertas funcionalidades dado el número de posibilidades que ofrece.

Nuestra intención era integrar JIRA con un repositorio Git. Llegados a este punto volvieron a surgir inconvenientes, ya que asumimos características que no eran propias de JIRA sino de Stash, que se integra con JIRA a través de un plugin. Los plugins en JIRA representan el grueso de su funcionalidad. Se instalan desde el

[8] Información obtenida de http://es.wikipedia.org/wiki/JIRA

[9] Si desea obtener más información sobre JIRA puede acceder a http://spanishpmo.com/index.php/jira-la-herramienta-perfecta-para-la-administracion-de-proyectos/

Marketplace, y la mayoría son de pago. Buscamos una alternativa a Stash, y conseguimos integrar JIRA con Git a través de otro plugin, llamado Git Integration Plugin for JIRA, que era gratuito. Sin embargo, lo único que este plugin ofrecía era una visualización de los archivos, las ramas, commits y merges de un repositorio Git asociado al proyecto. A diferencia de Stash, no permitía trabajar en el repositorio Git desde la interfaz de JIRA, que era uno de los atractivos de la herramienta. Con JIRA integrado con Stash es posible añadir plugins que proveen a los desarrolladores de herramientas a través de las cuales trabajar sobre el repositorio Git asociado al proyecto de forma gráfica e interactiva. Consideramos que realizar un estudio de Stash y configurarlo todo debidamente se salía de contexto puesto que aún no estaba presente el hecho de no haber encontrado una forma viable de usar JIRA de forma compartida entre los miembros del grupo. Dados estos inconvenientes, acabamos por descartar JIRA como herramienta para la gestión del proyecto, puesto que no hacía más que retrasarnos y la gestión ya la estábamos llevando a cabo por otros medios más convencionales.

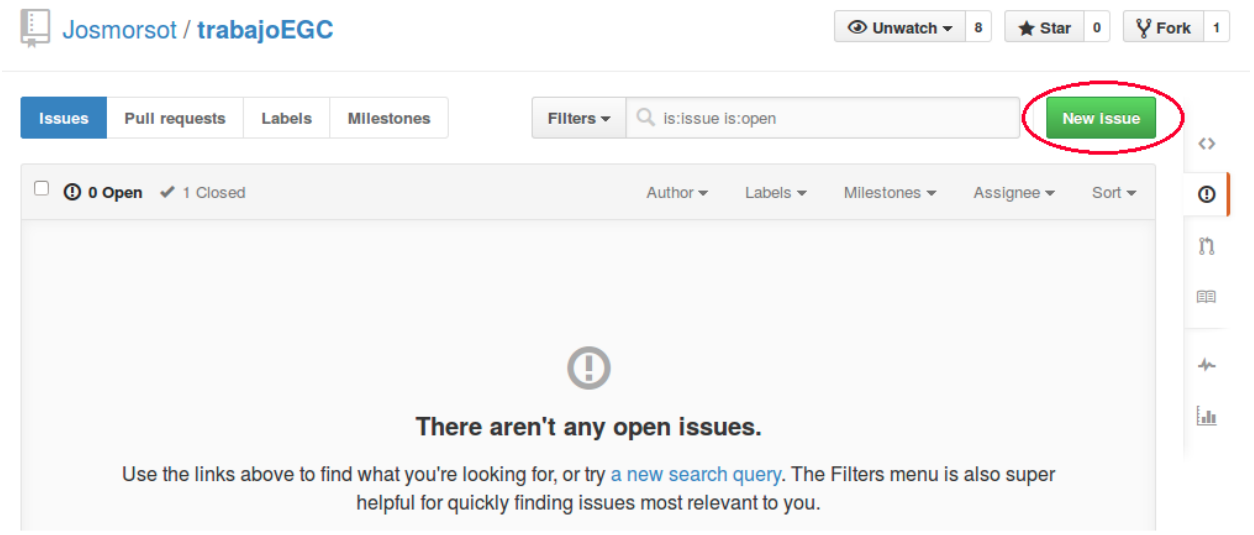
Al descartar JIRA como gestor de nuestro proyecto nos vimos obligados a estudiar otra serie de herramientas de gestión, como OpenProject o RedMine. Tras esto llegamos a la conclusión que teníamos a nuestra disposición Projetsii, que como siempre, nos brinda la posibilidad de usarla gratuitamente; además de eso en la asignatura hemos visto el funcionamiento de RedMine, prácticamente idéntico al de Projetsii, ya que ésta es tan solo una versión adaptada a la ETSII de la propia RedMine.

Además de Projetsii, como sistema de almacenamiento de archivos en la nube, hemos usado Google Drive *(la URL de la carpeta compartida se encuentra en la wiki de nuestro grupo),* debido a que nos permite editar los documentos del proyecto online, así que los nueve integrantes podemos modificar a la vez el mismo documento y hablar entre nosotros usando el chat del que dispone este tipo de ediciones.

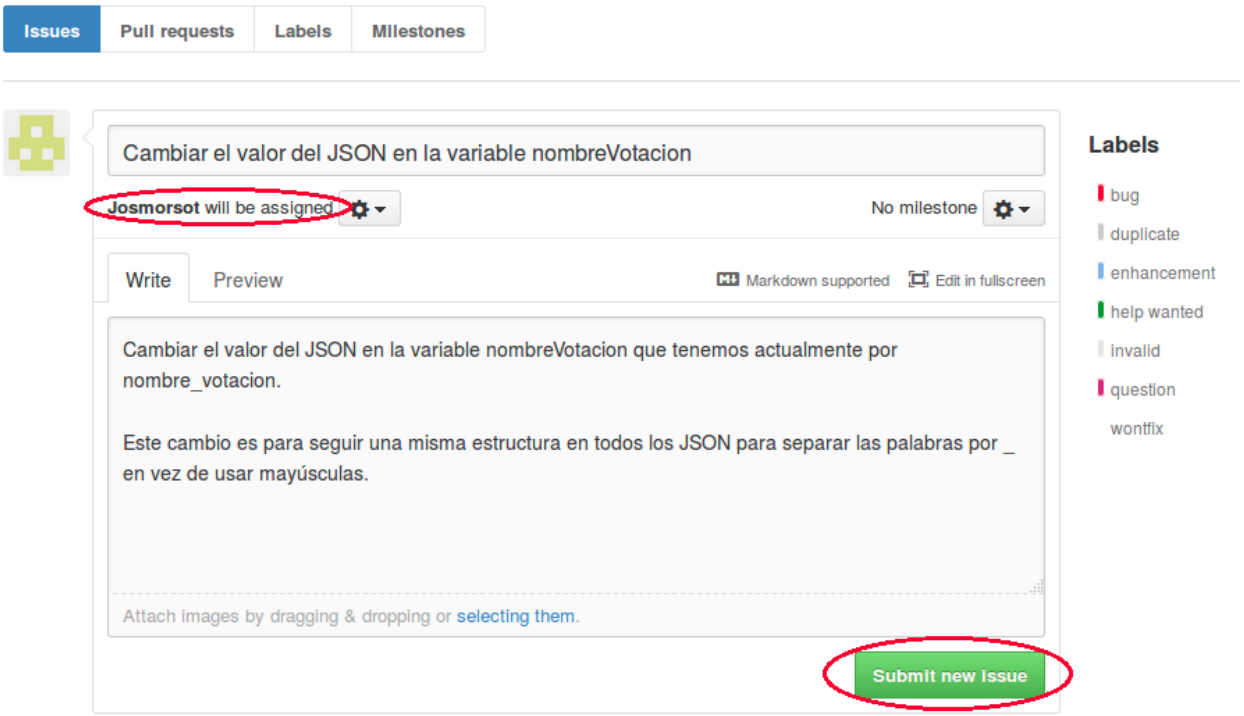
Ejercicio de gestión de una incidencia

En el siguiente ejercicio simularemos los pasos a seguir para la gestión de una incidencia ideal, en la que no surja ningún problema para resolverla. Esta será la forma a seguir para incidencias tanto a nivel de nuestro subsistema como del sistema completo, en el que se incluyen el resto de subsistemas. En este ejemplo en concreto, se contempla el desarrollo completo de una incidencia dentro de nuestro propio subsistema. Al final, veremos un pequeño ejemplo real con otro subsistema.

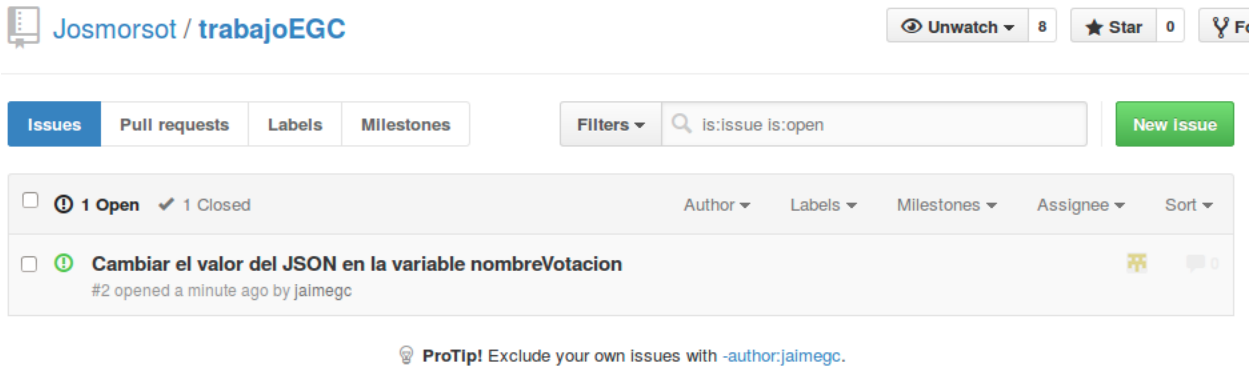
Paso 1: Accedemos a http://www.github.com e introducimos nuestro usuario y contraseña. A continuación pulsamos dentro de nuestro subproyecto y aparecerá una ventana como la siguiente y pulsamos sobre "Issues". Pulsamos sobre "New Issue" para rellenar la nueva incidencia detectada.



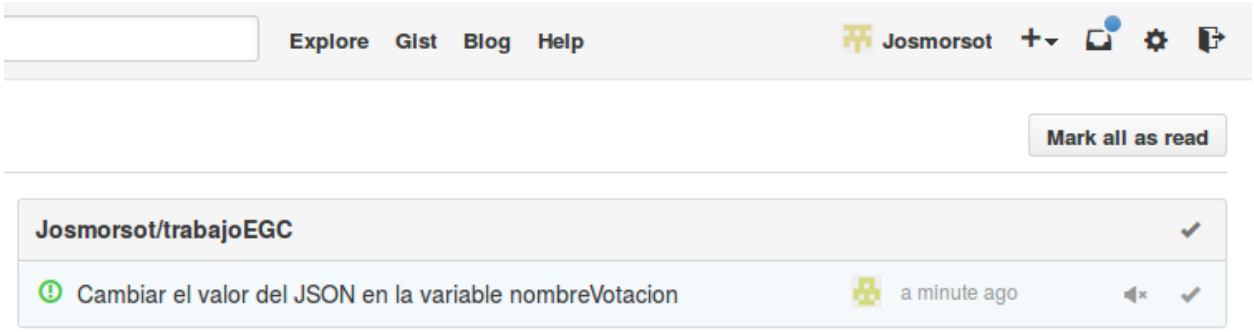
Paso 2: Rellenamos un título y una descripción para la incidencia y la asignamos a algún miembro del grupo. En este caso, se ha añadido al desarrollador principal Josmorsot. Finalmente, pulsamos en "Submit new Issue" para generar la incidencia.



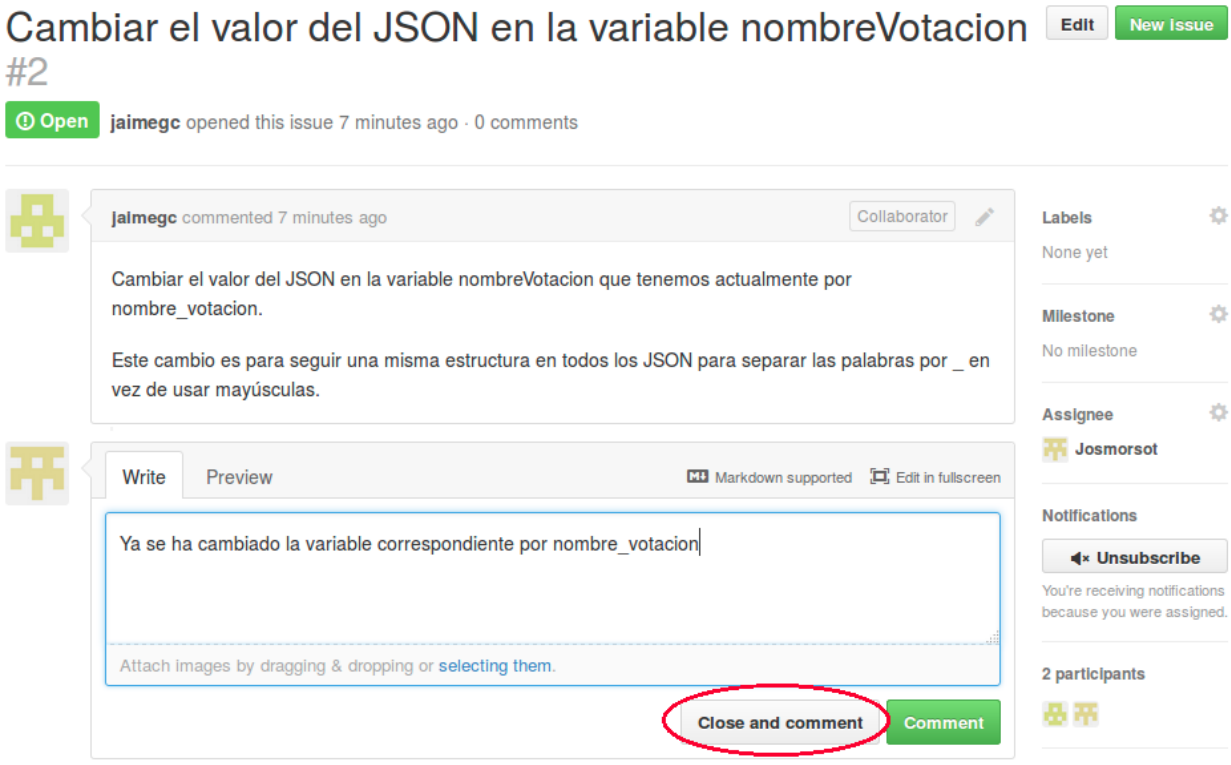
Paso 3: Si volvemos al menú de las Issues, podemos ver que aparece registrada como incidencia abierta (Open). Por tanto, está pendiente de ser resuelta



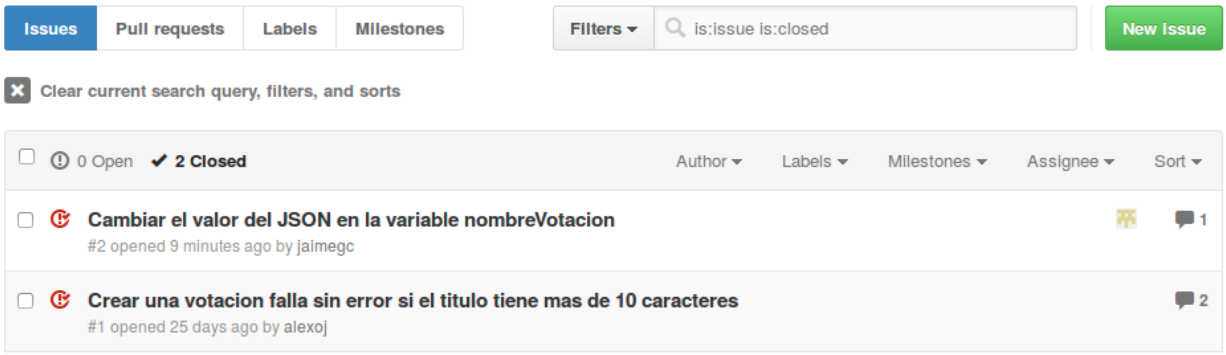
Paso 4: Cuando creamos la incidencia y la asignamos, automáticamente el usuario al que le hemos asignado la incidencia, recibe un mensaje en el correo y en github.



Paso 5: El usuario asignado una vez que resuelve la incidencia le da al menú de "Issues" y la edita, poniendo una descripción lo más detallada posible. Una vez rellena, pulsa sobre "Close and comment"

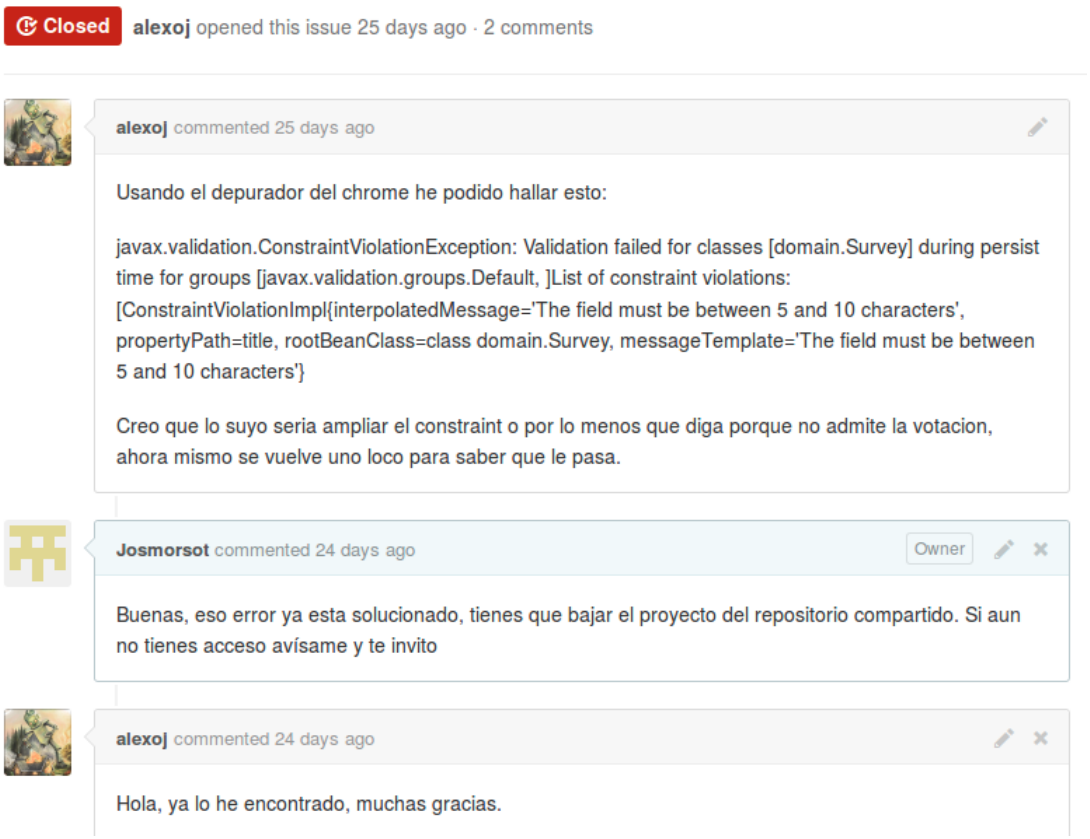


Paso 6: En el menú Issues podemos ver como la incidencia ha pasado a estar cerrada (Closed). En este caso, es la incidencia que aparece arriba en la imagen.



Para finalizar, aprovechando la imagen de arriba, vemos una segunda incidencia llamada "Crear una votación, falla sin error si el título tiene más de 10 caracteres". Esta incidencia fue enviada por otro subsistema al nuestro, siguiendo los pasos similares a los descritos arriba y pertenece a un caso real.

Aquí podemos ver una imagen que lo acredita:



Está mejor aunque no habéis dicho nada de depuración

6. Gestión de liberaciones, despliegue y entregas

La Gestión de liberaciones no se ha realizado de forma controlada, dado que no se ha visto en clase. No obstante, se ha investigado sobre este proceso de cara a conocer la forma correcta de llevarlo a cabo.

La Gestión de liberación[10] se ocupa de los cambios sobre los servicios de TI ya definidos. Según ITIL, un servicio de TI[11] es un medio por el cual los clientes pueden obtener resultados sin que éstos incurran en costos y riesgos específicos. La anatomía de un Servicio de TI incluye, entre otras cosas:

- Aplicaciones de software que necesita para funcionar
- Ambiente de operación
- Datos
- Procesos de Gestión del Servicio (ej. Gestión Incidentes)
- Acuerdos de Nivel de Servicio

La Gestión de liberación trabaja estrechamente con la Gestión de cambios y con la Gestión de configuración, de modo que se asegure que la Base de Datos de la Gestión de Configuración (CMDB) esté actualizada, que los cambios se manejen de manera adecuada y que todas las nuevas liberaciones de software sean almacenadas en la Biblioteca de Software Definitiva (DSL).

La Gestión de liberación toma una visión integral de un cambio a un servicio TI y debe asegurar que todos los aspectos de una liberación, ya sean técnicos o no, sean considerados juntos.

Un objetivo importante de la Gestión de liberación es proteger el ambiente de producción garantizando que existan los procedimientos y los controles correctos para liberar el software y el hardware. La Gestión de liberación permite a la organización manejar las liberaciones de software y hardware frecuentes sin que esto impacte en la calidad del servicio brindado.

La implementación de la Gestión de liberación ayuda a la disminución de la interrupción del servicio a través de la sincronización de las liberaciones, asegura que el hardware y el software provisto en el ambiente de producción son de

[10] Información obtenida de http://www.processlibrary.biz/index.php?main_page=document_product_info&cPath=69_115&products_id=539

[11] Si desea obtener más información sobre servicios de TI puede acceder a https://www.linkedin.com/groups/Qu%C3%A9-es-Servicio-TI-Es-4449624.S.126516834

buena calidad y facilitan la detección de versiones erróneas y de copias de software no autorizado.

Más concretamente, la Gestión de liberaciones[12] se encarga de generar una logística adecuada. Este proceso es el que realmente ejecuta los cambios, el proceso de Gestión de cambios no es el encargado de cambiar nada. Desde una perspectiva de cambios, los cambios se identifican, se registran, se categorizan, se priorizan, se autorizan, se coordina la construcción, se coordinan las pruebas y se coordina la implementación, pero quien ejecuta la construcción, las pruebas y la implementación es el proceso de Gestión de liberaciones. Para lograrlo, usa lo que se conoce como unidades de liberación. Esto es, toda aquella parte de la infraestructura que normalmente vamos a liberar de manera conjunta. Por ejemplo, implementaciones a un cierto nivel se habrán de seleccionar muy claramente y de manera particular como una unidad de liberación, ya que si lo hacemos desde un módulo que suponga una parte esencial niveles más bajos pueden verse afectados. Esto nos sirve para optimizar los recursos. Otro concepto importante es la política de liberación, que establece clara y específicamente qué parte de la infraestructura se puede modificar, qué parte no se puede modificar, y en qué períodos de tiempo. Se espera un RFC autorizado para disparar todos los mecanismos de logística mencionados, siendo necesario generar ambientes controlados para las pruebas, disminuyendo la probabilidad de impacto en la organización. RFC[13] (Request for Comments) son una serie de publicaciones de IIETF que describen diversos aspectos del funcionamiento de Internet y otras redes de computadoras, como protocolos, procedimientos, etc. y comentarios e ideas sobre estos. IIETF[14] (Internet Engineering Task Force) es una organización internacional abierta de normalización, que tiene como objetivos el contribuir a la ingeniería de Internet, actuando en diversas áreas, como transporte, encaminamiento, seguridad.

En definitiva, los principales propósitos de la Gestión de liberaciones son:

- Hacer una planificación y programación adecuada con todos los componentes que se requieren para cumplir con los criterios de aceptación del servicio.
- Controlar la construcción. Por ejemplo, en el caso de hardware parte desde la compra de equipos, si es software parte desde comprar la licencia o comenzar a desarrollar.

No se trata de que contéis a qué se refiere la GL sino aplicarlo en vuestro caso concreto

[12] Información obtenida de http://youtu.be/J6PrH4GvOHs

[13] Si desea obtener más información sobre RFC puede acceder a http://es.wikipedia.org/wiki/Request_for_Comments

[14] Si desea obtener más información sobre IETF puede acceder a http://es.wikipedia.org/wiki/Internet_Engineering_Task_Force

- Probar y desplegar las liberaciones. Incluyendo todos los roles y tipos de prueba según la metodología seguida.
- Entregar nuevas funcionalidades requeridas por el negocio.

Todo esto, mientras se protege la integridad de los servicios existentes.

Una vez que todos los subsistemas han finalizado su código y se ha probado la integración con todos en distintos entornos de desarrollo, al no encontrar ningún fallo, se pasa a realizar el despliegue. Esto consiste en poner las aplicaciones en un entorno, no de desarrollo, sino con lo justo para que se pueda consumir, es decir, los servidores para que puedan correr las aplicaciones.

El despliegue de todos los subsistemas que forman Agora US se hará sobre una máquina Debian 7. Para que el despliegue general se realice sin más problemas debemos tener instalados un conjunto de programas:

- Git
- Maven
- Tomcat 7
- Java openjdk7
- Mysqlserver
- Mysqlclient
- Python
- Xampp (debemos cambiar el puerto de mysql para evitar problemas)

Se parte de la base de que todo el código está en el repositorio compartido de Github y se pondrán distinguir entre tres tipos de proyectos según su tecnología:

- Proyectos Java (creación/administración de votaciones, deliberaciones, recuento, creación/administración de censos, almacenamiento)
- Proyectos Django (cabina de votación)
- Proyectos PHP (auth, frontend de resultados, visualización de resultados)

En el caso de los proyectos java se debe crear la base de datos según el nombre usado en el proyecto y para su construcción, se hará uso de maven y tomcat. Del primero, se utiliza la directiva mvn clean install (que limpia archivos temporales e inicializa el contexto de spring para su funcionamiento), que es necesario para crear los war (Web Application Archive). Este directiva hay que realizarlo dentro de cada una de las ramas, donde su ubica el fichero pom.xml que es el que contiene todas las dependencias del proyecto. Una vez generados los war, debemos desplegarlos con tomcat para que estén accesibles desde una URL, en concreto, copiar el war a var/lib/webapps/tomcat7

Para los proyectos PHP, por ejemplo auth, se copia a /opt/lampp/htdocs/auth sus ficheros, creamos la base de datos egcdb y su tabla con el script proporcionado. Una vez realizado, se debe crear un usuario "usuario" con password "passwrod"

Otros proyectos, como por ejemplo result_view, al no tener una base de dato, solo tendríamos que copiar su código a /opt/lamp/htdocs/ y al arrancar el tomcat, ya podríamos acceder.

El proyecto Python es el de cabina de votación. Para este proyecto, se debe crear una estructura de carpetas como la siguiente:

- Carpeta raíz: *cabina-integracion*
- cabina-agora-us: dentro la la raíz (será la que contenga el código del subsistema)
- Dos scripts (dentro de la razi): *install.sh* y *run.sh*

La primera vez que ejecutamos el proyecto de cabina, debemos ejecutar el install.sh (instala paquetes necesarios) y después run.sh (ejecuta el proyecto). Las veces siguientes que arranquemos este proyecto, solo será necesario ejecutar el segundo script.

Almacenamiento no corresponde a ningún grupo de los proyectos descritos a integrar debido a que es un subsistema externo, no sería necesaria su integración en la máquina Debian.

Otro caso de proyecto que no encaja en ninguna categoría, pese a ser un proyecto Java, es verificación puesto que este lo utilizan otros subsistemas para la encriptación de votos, por lo que este grupo genera una librería, la cual es consumida por otros subsistemas.

Lo más cómodo sería instalar esta librería en el repositorio maven y que cada proyecto que la necesite añada las dependencias a esta, por lo que ya no tendríamos que preocuparnos de las rutas de las librerías.

Cuando todo se haya integrado, tras realizar unas últimas pruebas de ejecución, por ejemplo con JMeter, el sistema Agora Us estaría listo para ser consumido por usuarios.

El despliegue se ha realizado entre los grupos Administración de censos, Deliberaciones y Creación de votaciones.

7. Gestión de la variabilidad

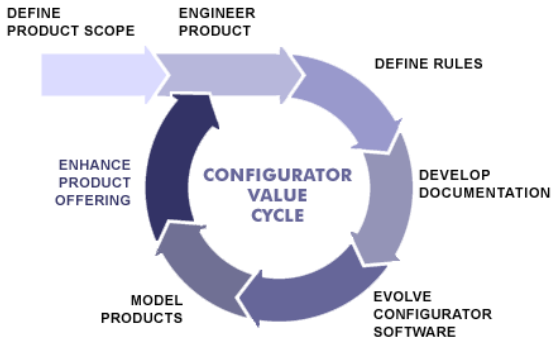
El interés por el estudio de la variabilidad (entendida como la habilidad de cambio o de personalización de un sistema) en el desarrollo del software ha aumentado significativamente durante los últimos años bajo la demanda, cada vez mayor, de los usuarios.

Esto se debe a su interés en diversos campos, desde la personalización del software a las líneas de productos. La forma más común de gestionar la variabilidad en líneas de productos software es mediante modelos de características o features que permiten seleccionar la configuración de cada aplicación concreta dentro de una línea de productos. Sin embargo la trazabilidad entre los modelos de características y los modelos de diseño (generalmente basados en UML) no es sencilla. Durante la fase de diseño y desarrollo de la arquitectura del producto, sea cuál sea el mecanismo de gestión de la variabilidad que se haya escogido debe seguirse lo más fielmente posible, según desarrolladores experimentados.

El problema tiene dos aspectos principales: por un lado, la definición del modelo que represente la línea de productos y sus variaciones (y permita configurar el conjunto óptimo de las mismas para cada aplicación concreta) y, por otro, el registro de la trazabilidad entre las variaciones del modelo y el reflejo de las mismas en la arquitectura que implementa la línea de productos (para facilitar la derivación de cada aplicación concreta).

Otro mecanismo para gestionarla, que está siendo estudiado e implantado, es:

- *El mecanismo de combinación de paquetes o package merge de UML 2.* Esta solución permite representar de forma ortogonal las variaciones arquitectónicas de la línea de productos, su relación directa con las características opcionales e incluso, utilizando paquetes de clases parciales, con la estructura del código que implementa el diseño.



8. Mapa de herramientas

Las herramientas utilizadas para este proyecto han sido las siguientes:

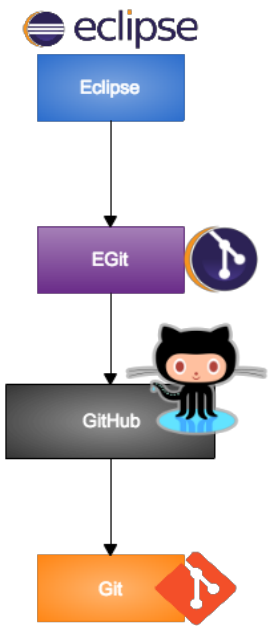
Tipo	Nombre	Versión
Máquina virtual	Oracle VM VirtualBox	4.2.6
IDE	Eclipse Luna	4.4.1
Servidor web	Apache en conjunto con Tomcat	7(Tomcat)
Sistema de control de versiones	Git	
Repositorio	GitHub	
Herramienta para el uso de GitHub	Plugin para Eclipse EGit	
Servidor de base de datos	MySQL Community Server	5.5
Conector	MySQL	5.1.26
Software	Java JRE	1.7.0
Software	Java JDK	7
Software de construcción de proyectos	Maven	3.1.0
Software para la gestión de incidencias	GitHub	
Framework	Spring	4.0.0
Framework	Angular JS	1.3
Herramienta de mapeo objeto-relacional	Hibernate	4.1.3
Librería	Faster XML/ Jackson core	

A continuación se detalla el uso de dichas herramientas y como se conectan entre ellas en función del objetivo para el que son utilizadas en nuestro subsistema "Creación y Administración de Votaciones".

En primer lugar, hemos utilizado la maquina virtual Oracle VM VirtualBox y en ella se han instalado todas las herramientas y software necesario para nuestro proyecto.

El primer objetivo es la gestión del código fuente, para ello se han utilizado las siguientes herramientas:

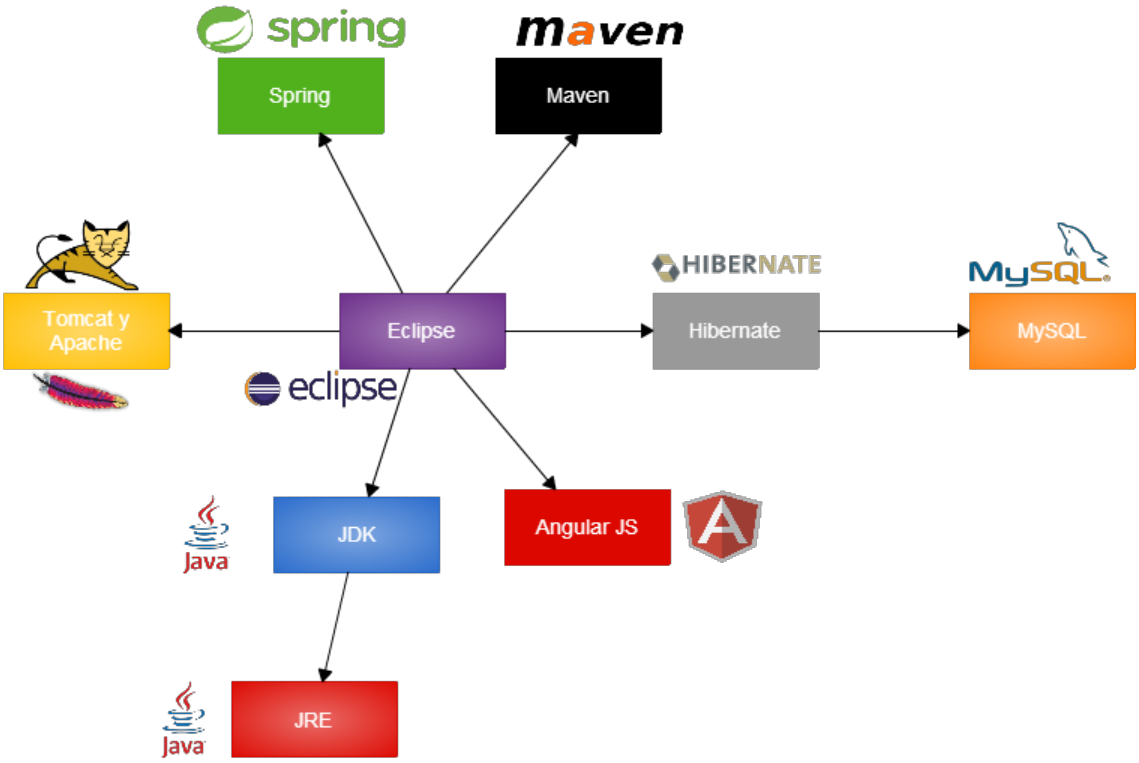
Gestión del código:



Hemos usado un repositorio alojado en GitHub que utiliza el sistema de control de versiones de Git. Para usar las funcionalidades de GitHub hemos utilizado el plugin para Eclipse EGit que proporciona todas las funcionalidades básicas de GitHub de una manera fácil e intuitiva.
En el punto 3 de este documento se ha hablado de una forma más detallada de la gestión del código.

El segundo objetivo es la realización del código fuente, para ello se han utilizado las siguientes herramientas:

Realización del código:



Para la realización del código hemos usado el IDE Eclipse Luna ya que hemos optado por el lenguaje JAVA y además, éste permite la instalación de muchas de las herramientas necesarias para nuestro proyecto. Eclipse utiliza el software JDK(que a su vez engloba al JRE) que provee herramientas de desarrollo para la creación de programas en Java, como por ejemplo un intérprete y un compilador.

Para la creación y gestión del proyecto hemos utilizado el software Maven, que permite describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Lo hemos integrado en el proyecto con el uso del plugin para eclipse M2E-WTP.

Nuestro proyecto utiliza el patrón de diseño Modelo-vista-controlador, para ello nos ayudamos de algunos Frameworks que facilitan la tarea. En concreto hacemos uso de Spring y Angular JS.

Como servidor web utilizamos Apache en conjunto con el contenedor web Tomcat , que permite el soporte de servlets y JSPs. Tomcat y Apache pueden utilizarse como servidores web en sí mismos, pero nosotros hemos decidido usarlos en conjunto para aprovechar las ventajas de ambos.

El servidor de base de Base de Datos que hemos utilizado es MySQL. En concreto, hemos utilizado el conector MySQL 5.1.26 para que nuestra aplicación pueda conectarse al servidor de base de datos . Además, hemos utilizado la herramienta Hibernate que permite el mapeo objeto-relación para poder transformar las entidades de base datos en MySQL a los objetos JAVA que utilizamos en nuestro proyecto y viceversa.

A continuación se muestra el script de creación de la Base de Datos utilizado en nuestro caso, para lo cual se ha ejecutado en una consola:

```
drop database if exists `CreacionAdminVotaciones`;
create database `CreacionAdminVotaciones`;

grant select, insert, update, delete
on `CreacionAdminVotaciones`.* to 'acme-user'@'%';

grant select, insert, update, delete, create, drop, references, index, alter,
create temporary tables, lock tables, create view, create routine,
alter routine, execute, trigger, show view
on `CreacionAdminVotaciones`.* to 'acme-manager'@'%';
```

En el punto 4 de este documento se ha tratado de una manera detallada de todas las dependencias y herramientas software utilizadas, como plugins y frameworks.

El tercer objetivo es la gestión de incidencias. Para ello hemos utilizado las funcionalidades que ofrece GitHub.

En el punto 5 de este documento se ha tratado de una manera detallada la gestión de incidencias.

Por último señalar que dada la necesidad de integración con otros subsistemas, hemos utilizado la librería Jackson para proveer a los demás de un JSON con el que poder hacer uso de nuestros servicios.

9. Lecciones aprendidas

LA001: Herramienta de Gestión

Área / Categoría: Gestión del Proyecto

Amenaza / Oportunidad: La versión de JIRA en la nube, la herramienta elegida para trabajar, es de pago.

Descripción de la situación: Se ha realizado un estudio de las características y funcionalidades que nos proporcionaba la herramienta y se había elegido como la mejor.

Descripción del Impacto en los objetivos del proyecto: Se ha retrasado el uso de una herramienta para la gestión.

Acciones correctivas y preventivas implementadas: se ha utilizado otra herramienta de las disponibles.

Lección Aprendida / Recomendaciones: Se debe realizar un estudio de las características y la disponibilidad de las herramientas previa al comienzo de la implementación del código.

LA002: Espacio común de trabajo

Área / Categoría: Integración

Amenaza / Oportunidad: Cambios en subsistemas producen errores en otros.

Descripción de la situación: Surgen continuos problemas y cambios por la falta de entendimiento

Descripción del Impacto en los objetivos del proyecto: La integración se demora y no se puede comprobar si todo funciona correctamente.

Acciones correctivas y preventivas implementadas: Se ha definido un espacio de comunicación para todo el equipo.

Lección Aprendida / Recomendaciones: Se debe establecer un espacio común para reflejar el trabajo de cada uno y realizar consultas o discutir sobre las implementaciones.

LA003: Repositorio Común

Área / Categoría: Integración

Amenaza / Oportunidad: La integración se demora.

Descripción de la situación: La integración en las sesiones dedicadas a ella no se realiza correctamente por no tener acceso al código actualizado de los demás subsistemas.

Descripción del Impacto en los objetivos del proyecto: No se pueden realizar los cambios necesarios para la correcta integración en el momento oportuno.

Acciones correctivas y preventivas implementadas: Se ha creado un repositorio común en "Github" para tener acceso continuamente al código actualizado de todos los subsistemas.

Lección Aprendida / Recomendaciones: Se debe comenzar a desarrollar el código haciendo uso de una plataforma común para todo el equipo.

LA004: Planificación del Trabajo

Área / Categoría: Planificación

Amenaza / Oportunidad: Mala división del trabajo.

Descripción de la situación: Se ha realizado investigación y trabajo por parte de integrantes del grupo que en el documento no han desarrollado el apartado de ese ámbito.

Descripción del Impacto en los objetivos del proyecto: Varios trabajadores han investigado sobre lo mismo, con lo que se ha realizado el doble de trabajo.

Acciones correctivas y preventivas implementadas: Se han realizado varias iteraciones de revisiones del documento para aportar más información.

Lección Aprendida / Recomendaciones: Se debe dividir el proyecto en tareas para asignar a cada integrante una funcionalidad concreta desde el principio.

LA005: Gestión de cambios

Área / Categoría: Gestión del Código/Cambios

Amenaza / Oportunidad: Se deben tomar decisiones que influyen en el trabajo de otros subsistemas.

Descripción de la situación: Hay que realizar cambios que afectan a otros subsistemas y puede causarles problemas.

Descripción del Impacto en los objetivos del proyecto: Han aparecido problemas en la integración.

Acciones correctivas y preventivas implementadas: Se ha establecido una comunicación con los subsistemas relacionados previa al cambio.

Lección Aprendida / Recomendaciones: Se debe mantener una comunicación continua con los subsistemas relacionados para mantener informado de todo cambio que deba realizarse y agilizar el trabajo.

LA006: Priorizar el trabajo

Área / Categoría: Planificación

Amenaza / Oportunidad: El retraso en nuestras tareas puede retrasar el trabajo de otros subsistemas relacionados.

Descripción de la situación: Se deben realizar varias tareas y algunas influyen en el trabajo de otros subsistemas.

Descripción del Impacto en los objetivos del proyecto: Se ha producido un retraso en la integración.

Acciones correctivas y preventivas implementadas: Se han priorizado las tareas que influyen a otros subsistemas frente a las que sólo afectan a nuestro subsistema.

Lección Aprendida / Recomendaciones: Se debe llevar una planificación actualizada de las tareas pendientes y priorizar la que afecta a más subsistemas.

10. Conclusiones

El proyecto se ha desarrollado de forma satisfactoria, si bien hay aspectos que se podrían optimizar de cara a futuros proyectos de perfil similar. Teniendo en cuenta que es la primera vez que trabajamos poniéndonos de acuerdo con un equipo de trabajo tan amplio, el resultado es positivo. Contamos con un buen número de lecciones aprendidas para no volver a cometer los mismos errores y trabajar de forma más efectiva desde un principio, detalladas en el apartado anterior.

Nuestra gestión del código fuente ha ido evolucionando desde el uso de 'Projetsii' hasta el uso de un repositorio Git alojado en 'GitHub', en el cual nuestro sistema está integrado con el resto. Al haber definido correctamente los requisitos desde un principio, el código no ha sufrido demasiados cambios. No obstante, las necesidades de los otros sistemas relacionados sí lo han hecho, por lo que se han realizado varias iteraciones en el código necesario para integrar. En cuanto a nuestra gestión de la construcción, hemos mantenido Eclipse Indigo SR2 y Maven dado que no ha habido inconveniente alguno a la hora de la integración con otros subsistemas.

A la hora de enfrentarnos a proyectos de esta complejidad, la comunicación juega un papel esencial, a la par que un máximo responsable que tome decisiones. Así, se establece de forma clara cual es el trabajo a realizar por cada subgrupo, evitando conflictos. Se ha de informar claramente los cambios y decisiones tomadas con el mayor nivel de detalle posible para no dar lugar a confusión y entender a nivel de código y realización de tareas el porqué de esas decisiones. Consideramos que aprovechando la ventaja de poder elegir desde un principio junto a los compañeros el entorno de trabajo, la mejor opción en un proyecto de este tamaño hubiera sido que todos los subgrupos trabajáramos desde un principio con el mismo lenguaje de programación, sobre el mismo proyecto, sobre un mismo repositorio, y cada uno sobre su rama. Sin embargo, la inexperiencia e incertidumbre iniciales dieron lugar a que cada subgrupo decidiera trabajar en el entorno en el que se sentían más cómodos, lo que más tarde repercutió en que la integración no fuera tan sencilla. Asumimos que dada la libertad que se nos ha otorgado, uno de los objetivos era que nos diéramos cuenta por nosotros mismos de estos aspectos, además de que aprendiéramos a desenvolvernos sin una guía.