Universidad de Sevilla

Memoria del proyecto agora@US

Grupo de Autenticación

Daniel Ayala Hernández
Daniel de los Reyes Leal
Fidel Mazo Delgado
Juan Carlos Roldán Salvador
Alejandro Sánchez Medina

22-12-2014

Contenido

Los dos puntos

1. Historial de cambios

Las versiones se controlan de la siguiente manera:

Dada una versión X.Y.Z

Habrá un incremento en X cuando se haya realizado una versión o mejora completa de todo el documento. Y y Z se pondrán a 0.

Habrá un incremento en Y cuando se introduzca una sección nueva. Z se pondrá a 0.

Habrá un incremento en Z cuando se produzca una modificación y mejora de secciones ya existentes.

Lo idea habría sido poner también
los miembros que se hicieron cargo de los cambios

Versión	Fecha	Modificaciones
0.1.0	27/10/2014	▪ Añadida estructura básica del documento. ▪ Añadida gestión del código fuente.
0.1.1	06/11/2014	▪ Añadida mención del repositorio compartido.
0.1.2	13/11/2014	▪ Añadida norma de nombrado de ramas del subsistema.
0.2.0	20/11/2014	▪ Añadido contenido de la sección de gestión de la construcción e integración continua.
0.3.0	03/12/2014	▪ Añadida introducción.
0.4.0	04/12/2014	▪ Añadido resumen. ▪ Expansión de la introducción. ▪ Añadidas capturas de pantalla a la gestión de código fuente.
0.4.1	08/12/2014	▪ Expansión de gestión de la construcción e integración continua.
0.5.0	11/12/2014	▪ Añadida depuración.
0.5.1	13/12/2014	▪ Añadidos diagramas a la sección de gestión del código fuente.
0.6.0	14/12/2014	▪ Añadido mapa de herramientas.
0.7.0	17/12/2014	▪ Añadido ejercicio y solución a la sección de construcción e integración continua. ▪ Añadida sección de despliegue del sistema.
0.7.1	18/12/2014	▪ Añadida gestión de peticiones de cambio e incidencias a la sección de gestión del cambio, incidencias y depuración.
0.7.2	18/12/2014	▪ Añadido ejercicio y solución a la sección de gestión de gestión del cambio, incidencias y depuración.
0.8.0	19/12/2014	▪ Añadidas las conclusiones

Versión	Fecha	Modificaciones
0.8.1	20/12/2014	▪ Añadido diagrama de flujo a la sección de peticiones de cambio e incidencias.
1.0.0	20/12/2014	▪ Añadidas menciones de la intención de ampliar secciones.

2. Resumen

En esta memoria se detallan las decisiones tomadas durante el desarrollo del proyecto correspondiente a la asignatura Evolución y Gestión de la Configuración (EGC), consistente en el desarrollo de un subsistema que deba integrarse con otros subsistemas creados por el resto de grupos.

Durante este desarrollo e integración, nuestro equipo se ha enfrentado a la toma de decisiones relacionadas con la gestión de un proyecto y la organización de este, tanto dentro del grupo de desarrollo como de cara a los otros subsistemas, teniendo que establecer pautas y procesos a seguir durante el desarrollo y llegar a acuerdos. Son estas decisiones en las que se ponen en práctica los conocimientos adquiridos durante la asignatura, y no las de carácter técnico, las que se detallan en este documento.

Además de los contenidos de la asignatura, se buscó información sobre herramientas que pudieran ser usadas durante la gestión del proyecto para acercarlo lo máximo posible a una experiencia real.

Cada sección de este corresponde a un aspecto a gestionar, y se incluyen ejercicios a modo de ejemplo, así como sus soluciones.

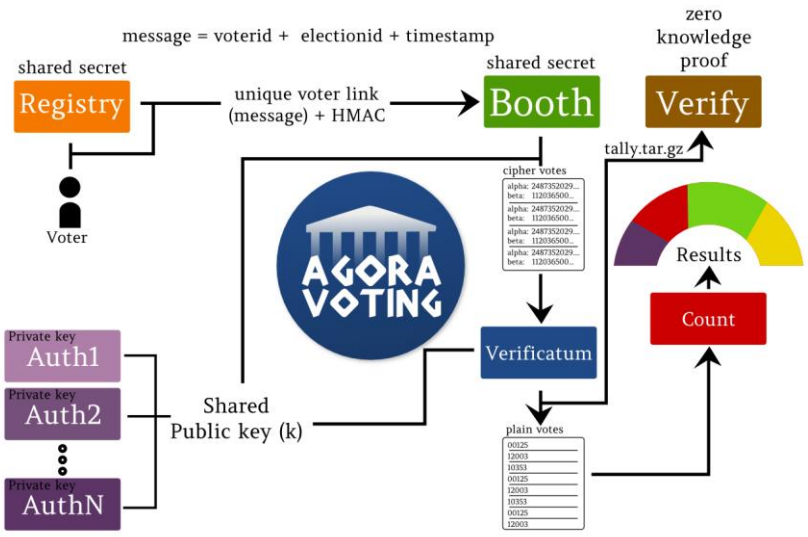
Debería ser una combinación de ambas

El resumen no da un resumen del documento. Debería haberse extendido más y ser más concreto. Para la versión final hay que trabajarlo más.

3. Introducción

En la actualidad, el sistema Agoravoting permite realizar votaciones a través de Internet. En este, se implementan protocolos que aseguran el anonimato y la seguridad a la hora de realizar votaciones.

Basándonos en Agoravoting, como parte de la asignatura EGC se pretende crear un sistema de votaciones online.



Las figuras siempre tienen una leyenda que las identifica no deben ponerse, en un doc técnico figuras sin leyenda e.g.
Fig 1. Estructura de Agora Voting

El sistema se llamará AGORA@US, y estará formado por varios subsistemas que se comunicarán entre sí para ofrecer el servicio completo:

- Autenticación
- Creación/Administración de votaciones
- Modificación de resultados
- Almacenamiento de votos
- Deliberaciones
- Recuento
- Creación/Administración de censos
- Frontend de resultados
- Visualización de resultados
- Verificación
- Cabina de votación

Cada grupo de la asignatura tiene asignado un subsistema que debe desarrollar, siendo el de nuestro grupo el subsistema de **Autenticación**.

El sistema de autenticación se encargará de realizar la autenticación del usuario antes de que este acceda al resto de la aplicación. Para que quede constancia de su estado como autenticado, se almacenarán en su navegador 2 cookies:

- Una con identificador "user", en la que se almacenará el nombre de usuario autenticado.
- Una con identificador "token", en la que se almacenará un token generado a partir de su nombre de usuario y contraseña.

Siempre que se desee constatar que el usuario accediendo a una funcionalidad está correctamente autenticado, se debe comprobar que el token almacenado en la cookie es el correcto. Para ello se ofrece una API Rest en la que las peticiones tendrán el siguiente formato, siendo todas del tipo GET:

mayúsculas

http://[host]/auth/api/[recurso]?param1=value1&param2=value2...

[host] hace referencia al host en el que se encuentre el sistema (localhost)

[recurso] hace referencia al nombre del recurso de la API.

Las tablas, al igual que la figura, deben también tener leyenda. Esta figura aquí queda mal

Recurso	Descripción	Parámetros	Respuesta	Ejemplo de respuesta
getUser	Obtiene un usuario del sistema, incluyendo sus datos. Este método solo puede ser usado por subsistemas locales (acceso mediante localhost).	**user**: nombre del usuario	JSON con el usuario pedido. Los datos del usuario son los siguientes: "username", "password" y "email", "genre" y "autonomous_community".	{ "username" : "johndoe", "password" : "foo_bar", "email" : "mail@example.com", "genre" : "Masculino", "autonomous_community" : "Madrid" }
getUsers	Obtiene todos los usuarios del sistema, incluyendo sus datos. Este método solo puede ser usado por subsistemas locales (acceso mediante localhost).		JSON con un array con los datos de cada usuario. Los datos de cada usuario son los siguientes: "username", "password" y "email", "genre" y "autonomous_community".	[{ "username" : "johndoe", "password" : "foo_bar", "email" : "mail@example.com", "genre" : "Masculino", "autonomous_community" : "Madrid", "age" : "18" }, { "username" : "Marta", "password" : "12345", "email" : "mail2@example.com", "genre" : "Femenino", "autonomous_community" : "Madrid", "age" : "26" }]

Recurso	Descripción	Parámetros	Respuesta	Ejemplo de respuesta
checkToken	Comprueba si un token es válido. Para ello, se obtiene el usuario correspondiente al token (indicado al comienzo del token), se genera el token del usuario y se comprueba si es igual que el pasado como parámetro.	**token**: token a validar.	JSON con el campo "valid" indicando la validez del token (true/false).	{ "valid"=true }
checkToken User	Comprueba si un token es válido para un usuario. Para ello, se obtiene el usuario pasado como parámetro, se genera el token del usuario y se comprueba si es igual que el pasado como parámetro.	**user**: nombre del usuario cuyo token se va a comprobar. **token**: token a validar.	JSON con el campo "valid" indicando la validez del token (true/false).	{ "valid"=true }

La sección de introducción tampoco está bien enfocada. Está bien las cosas que se ponen pero debemos estructurarla de forma distinta. Se explicará en clases cómo hacerlo.

En todo caso el apartado de la funcionalidad de la aplicación debería ser otra sección a parte y debiera también contener los problemas actuales identificados en el sistema.

No haría falta hacer un análisis completo de requisitos del sistema pero sí, al menos, algo un poco más ordenado y elaborado.

4. Gestión del código fuente

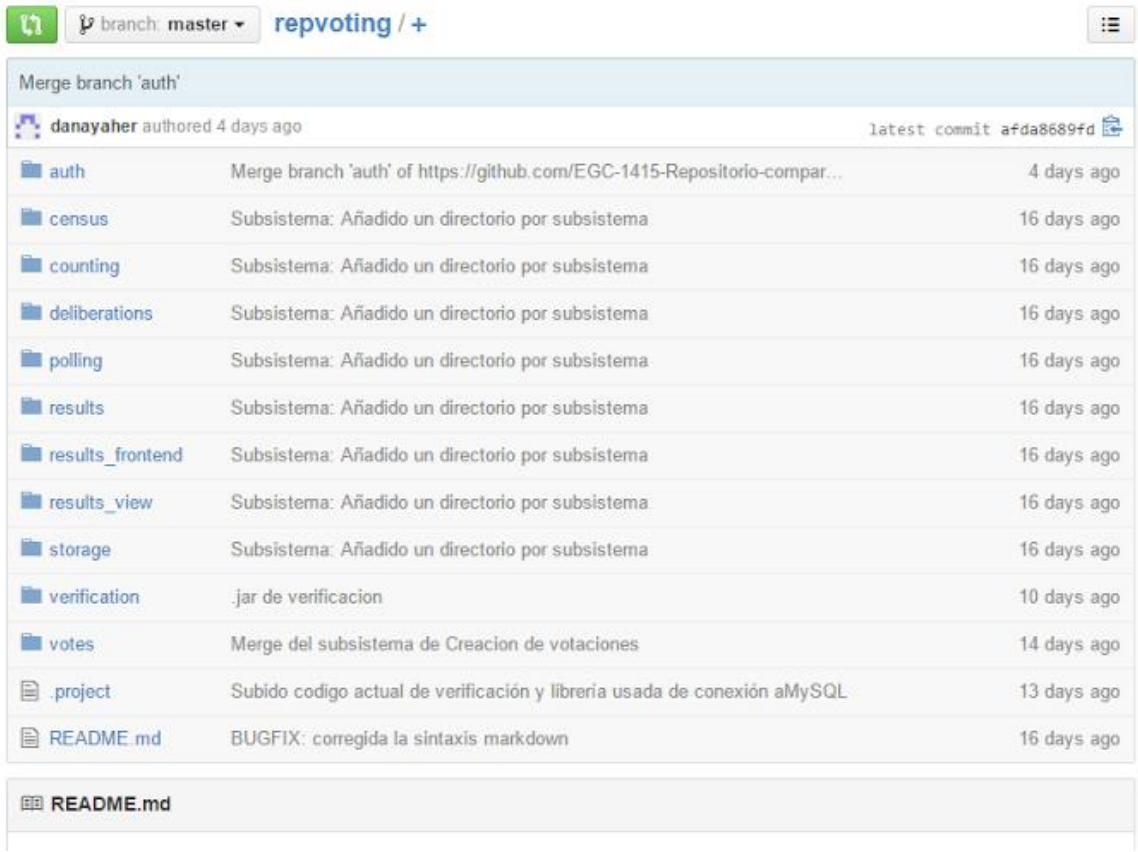
Para administrar el código fuente del subsistema se usará un repositorio Git. Este repositorio será usado por varios subsistemas para disponer del código fuente de todos los subsistemas, pudiendo usarlo cómodamente en la integración. El nombre del repositorio es Repvoting, y estará alojado en el servicio de alojamiento de repositorios Git Github.

Esta es una propuesta concreta y debería justificarse ésta decisión. Sus pros/cons

En este, cada subsistema se desarrollará en una rama diferente, a partir de la cual podrán crearse nuevas ramas. La rama master se usará para "recopilar" el trabajo estable de todos los grupos, haciéndose merge de las otras ramas cuando sea necesario integrar varios subsistemas.

El código fuente de cada subsistema se guardará en una carpeta diferente para que no haya conflictos.

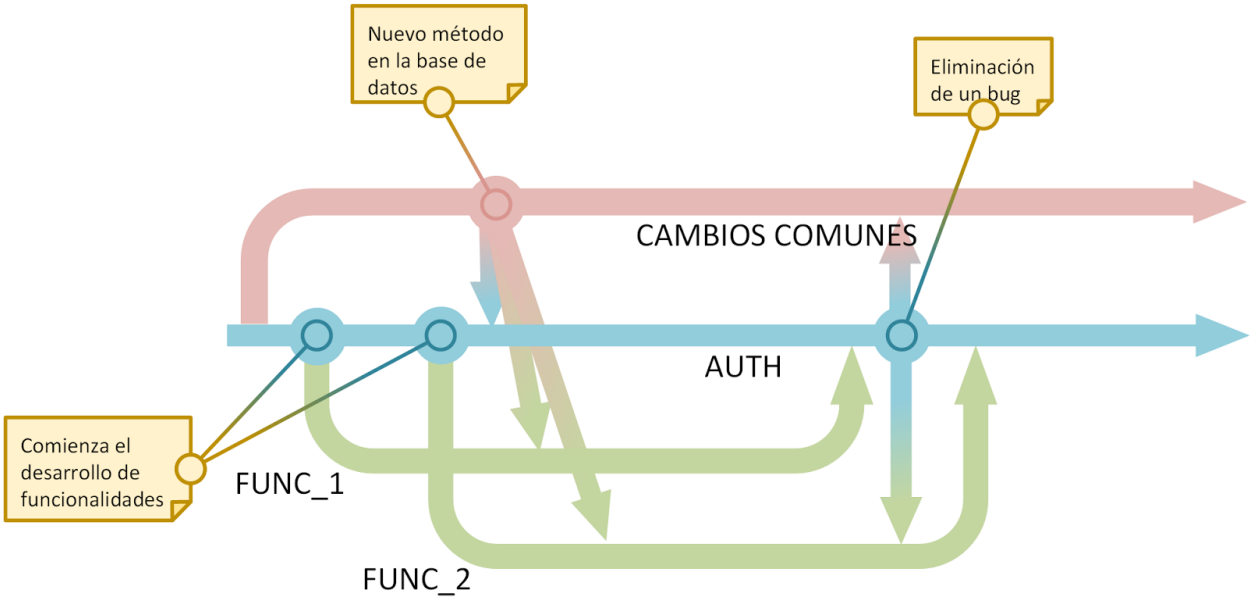
Ésto también es importante y hay que argumentarlo.



La gestión de ramas en el subsistema de autenticación se realizará de la manera indicada a continuación. Estas normas han sido creadas de manera que sean independientes del programa de control de versiones usado:

- Se crearán ramas según las funcionalidades que haya que implementar, creando una rama por cada nueva funcionalidad. No se ha escogido una división en ramas basada

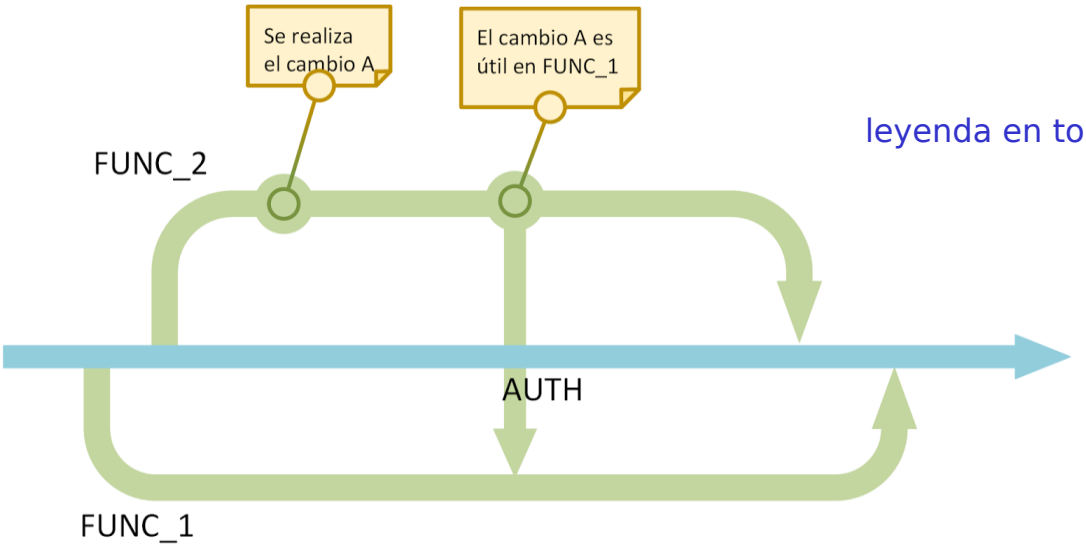
en la arquitectura debido al reducido tamaño del subsistema, que hace que esta sea muy simple. Sin embargo, la división en funcionalidades plantea un problema: la repetición de trabajo. Por ejemplo: dos funcionalidades necesitan acceder a una lista de usuarios, por lo que los dos miembros encargados de implementar la funcionalidad realizan el mismo trabajo por duplicado. Para solucionar esto, se pretende crear una rama de cambios globales dedicada a cambios que puedan ser útiles para todos los miembros, de manera parecida a una rama de parches pero sin enfocarla a la resolución de fallos, sino al desarrollo. Esta rama se usará principalmente para métodos de acceso a la base de datos. También se usará la rama maestra del subsistema (auth) de la misma manera, pero sólo para realizar cambios relacionados con la corrección de errores.



leyenda de la figura y reconocer si es una figura tomada de algún sitio y en ese caso de dnd?

- Si en algún momento en una rama es de utilidad el código perteneciente a otra rama, se hará un merge de esta última en la rama que puede utilizar el código (sin borrar ninguna de las dos). Si hay algún conflicto durante el merge no relacionado con el código que se quiere pasar de una rama a otra, permanecerá siempre el código de la rama original. De esta manera, se pretende evitar la repetición de trabajo cuando es menos obvio la utilidad de este para todo el grupo, de manera que no se haya usado la branch de cambios globales. Para que esto sea posible, se requiere que los miembros del equipo puedan identificar el trabajo que es posible que ya haya realizado otro miembro del grupo.

Parece que todo esto que se describe no es algo que se haya "vivido" sino una propuesta en caso de que se hiciera así. Aclararlo. Debería explicarse un poco más y mejor con ejemplos concretos y más asequibles.



leyenda en todas las figuras

- Se realizará un merge que incluya los cambios de una rama en la rama principal cuando se haya terminado la funcionalidad a la que está dedicada cada rama, cerrándose la rama dedicada a la funcionalidad. Con la excepción de la rama de cambios comunes relacionados con la base de datos, que nunca se cerrará, pero cuyos cambios se reflejarán siempre en las otras ramas.

Falta algo así como una "arquitectura" de vuestro sistema para saber en qué unidades funcionales lo dividís

- Todos los miembros del equipo tendrán derechos de escritura y lectura de todas las ramas. Teniendo en cuenta el tamaño reducido del equipo y el proyecto, se ha llegado a la conclusión de que administrar los permisos que corresponden a cada miembro sería poco útil (es poco probable que un miembro del equipo realice un cambio innecesario e inesperado en una rama que no le corresponde) y consumiría más tiempo que cualquier pequeño problema que pueda causar la falta de restricciones. Además, de esta manera, un miembro del grupo puede revisar, y fácilmente corregir, un error en una rama correspondiente a una funcionalidad asignada a otro miembro (siempre y cuando este error sea obvio y la corrección no pueda causar problemas). En general, se facilita la colaboración a la hora de desarrollar funcionalidades.

qué es esto? algo tangible?

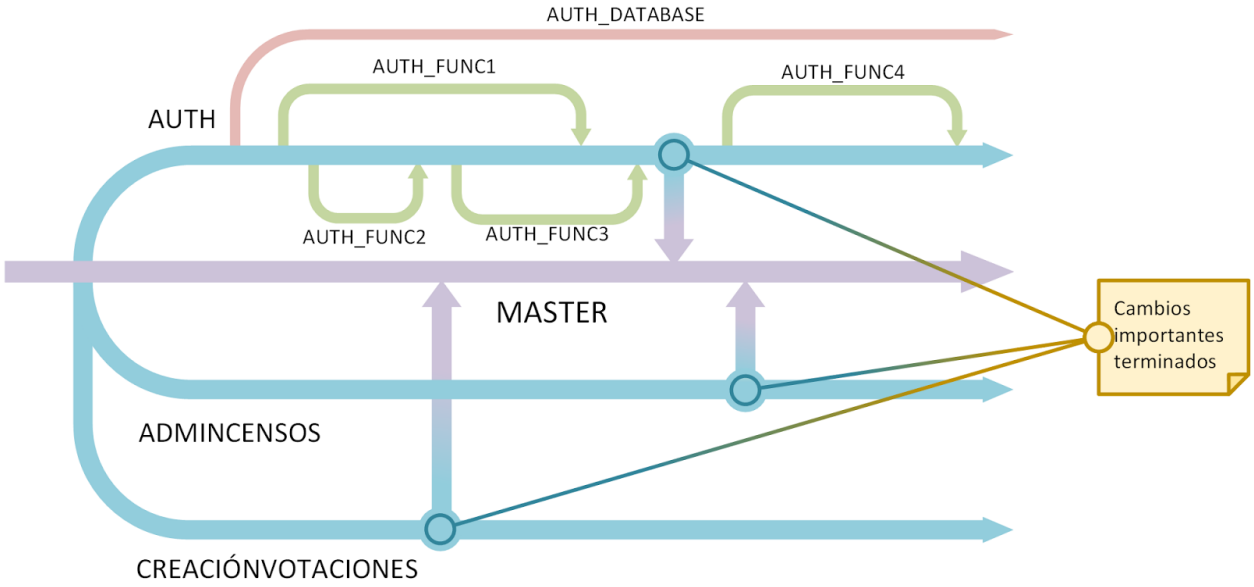
- Siempre que se haga un cambio considerable se hará pasar este al repositorio remoto central, para que todos los miembros del equipo dispongan lo antes posible del código actualizado.

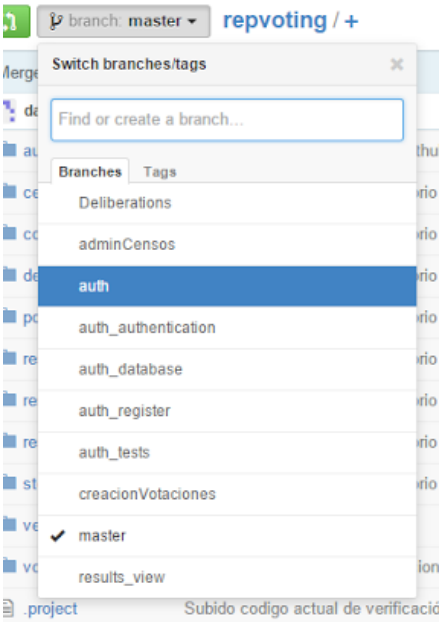
- Siempre que se realice un parche sobre el código (una modificación que no añada nuevas funcionalidades), la descripción corta de esta será BUGFIX: seguido del bug que se pretende corregir, y la plataforma sobre la que se produzca dicho bug, en caso de que haya una concreta.

- Las ramas creadas para desarrollar el subsistema de autenticación en el repositorio compartido tendrán siempre un nombre que comience por “auth_”. La rama original correspondiente al subsistema se llamará “auth”. Los otros subsistemas usarán ramas

esto es interesante siempre que se hay apuesto en práctica. Si se ha puesto en práctica habría que aclarar si se ha inspirado en algún sitio. Se debería también argumentar cuáles son los posibles riesgos de ésta "arquitectura de ramas"

independientes. Cuando se implementen funcionalidades importantes, se debe hacer merge de las ramas de cada subsistema a “master”.





A esta gestión se le añade lo siguiente fuera del ámbito de la gestión de ramas:

- Los cambios serán, fundamentalmente, nuevas funcionalidades. Estas funcionalidades deberán ser discutidas por el grupo, y aceptadas por todos. Es entonces cuando se pueden implementar. Mientras se implementa una funcionalidad, un miembro del equipo

puede proponer un cambio o adición a esta para que los otros miembros den el visto bueno y se lleve a cabo. Los cambios dedicados a la corrección de errores no deben de ser consultados con el resto del grupo.

Ejercicio

Herramientas necesarias:

- Git.

Por qué en este repo!! debería ser un ejercicio que describa lo de arriba.

Enunciado:

En este ejercicio se mezcla el enunciado con el ejercicio.

- Obtener una copia local del repositorio **https://github.com/alesanmed/repcloning.git**.
- Crear la rama auth_changes y cambiar a esta, en la que se realizarán los cambios, acorde a nuestra gestión de la configuración.
- Editar el archivo **main.py** y, sobre la línea ***print("Hola a todos");*** escribir **if(True):**
- Hacer commit del archivo
- Hacer merge de la rama auth_changes con auth, ya que se han terminado de realizar los cambios
- Debido a que el archivo falla (se puede comprobar ejecutándolo) se va a realizar un bugfix.
- Acorde a nuestra gestión del código fuente, el bugfix se realizará en la rama principal de nuestro subsistema, por lo que se deberá cambiar a la rama **auth**, si no se está en ella.
- Cambiar el archivo **main.py** y añadir una tabulación a la línea **print("Hola a todos");**
- Hacer un commit de los cambios, cuyo título comience con la etiqueta **BUGFIX**.
- Debido a que ya hemos terminado de implementar nuestro subsistema, se debe hacer un merge con la rama **master** y dejar el repositorio en un estado consistente.

Solución:

poner los comandos en un tipo de letra "courier new" p.e

- En primer lugar se debe ejecutar el comando **git clone git@github.com:alesanmed/repcloning.git**
- Una vez tengamos el repositorio en local, debe cambiarse a la rama **auth**, ya que será nuestra rama principal. Para hacer esto, trayendo los cambios del repositorio se debe ejecutar el comando:
  - **git checkout --track origin/auth**
- Acto seguido, se creará la nueva rama, para lo cual se puede realizar cualquiera de las dos siguientes opciones:
  - Ejecutar:
    - **git branck auth_changes**
    - **git checkout auth_changes**
  - Ejecutar:

        - **git checkout -b auth_changes**

- Se modificará el archivo y para hacer commit del mismo:
    - **git add auth/main.py** en caso de estar en la carpeta raíz. En cualquier otro caso, se escribirá la ruta hasta el archivo o **“.”**.
    - **git commit -m “Mensaje de commit”**
- Para fusionar las ramas se deben ejecutar los siguientes comandos:
    - **git branch auth**
    - **git merge auth_changes**
- Ahora se realizará el bugfix, para lo cual se debe realizar el cambio en el archivo **main.py** y ejecutar:
    - **git add auth/main.py** en caso de estar en la carpeta raíz. En cualquier otro caso, se escribirá la ruta hasta el archivo o **“.”**.
    - **git commit -m “Mensaje de commit comenzando con BUGFIX”**
- Por último se fusionarán las ramas realizando:
    - **git checkout master**
    - **git merge auth**
    - Resolver el conflicto generado aplicando el criterio personal
    - **git add auth/main.py** en caso de estar en la carpeta raíz. En cualquier otro caso, se escribirá la ruta hasta el archivo o **“.”**.
    - **git commit -m “Mensaje de commit”**

        - **git checkout -b auth_changes**
- Se modificará el archivo y para hacer commit del mismo:
    - **git add auth/main.py** en caso de estar en la carpeta raíz. En cualquier otro caso, se escribirá la ruta hasta el archivo o **“.”**.
    - **git commit -m “Mensaje de commit”**
- Para fusionar las ramas se deben ejecutar los siguientes comandos:
    - **git branch auth**
    - **git merge auth_changes**
- Ahora se realizará el bugfix, para lo cual se debe realizar el cambio en el archivo **main.py** y ejecutar:
    - **git add auth/main.py** en caso de estar en la carpeta raíz. En cualquier otro caso, se escribirá la ruta hasta el archivo o **“.”**.
    - **git commit -m “Mensaje de commit comenzando con BUGFIX”**
- Por último se fusionarán las ramas realizando:
    - **git checkout master**
    - **git merge auth**
    - Resolver el conflicto generado aplicando el criterio personal
    - **git add auth/main.py** en caso de estar en la carpeta raíz. En cualquier otro caso, se escribirá la ruta hasta el archivo o **“.”**.
    - **git commit -m “Mensaje de commit”**

Faltan algunas cosas en el capítulo de "código". Por ejemplo: ¿Qué roles tienen los desarrolladores? ¿tienen todos los mismos permisos? ¿qué ventajas e inconvenientes tiene la solución que aportáis para la gestión de ramas? También hecho en falta alguna mención a los "tags". No usáis ninguna forma de tagging? Se habla poquísimo también de la resolución de conflictos y qué problemas enfrentamos en dicha resolución de conflictos.

También sería interesante reportar las cosas que hayáis aprendido en el proceso y por qué habéis llegado a la conclusión de que la propuesta que hacéis es la mejor. Reportar la evolución de lo que habéis ido haciendo sería también interesante.

5. Gestión de la construcción e integración continua

Para integrar los diferentes componentes del subsistema, se llevará a cabo una integración “de abajo a arriba”. Con esta metodología se pretende trabajar siempre sobre una base real y probada.

Primero se desarrollarán e integrarán los componentes que no dependen de otros de más bajo nivel: funciones dedicadas a realizar cálculos de un token, crear cookies, obtener un listado de usuarios o introducirlos en la base de datos, etc.

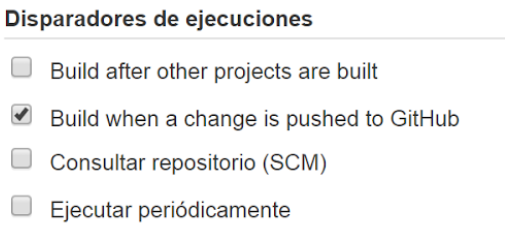
A continuación, se desarrollarán e integrarán las funciones que hagan uso de alguna de las funciones anteriores. Por ejemplo, comprobación de la validez de una cookie, registro de usuarios incluyendo comprobaciones de los campos, etc.

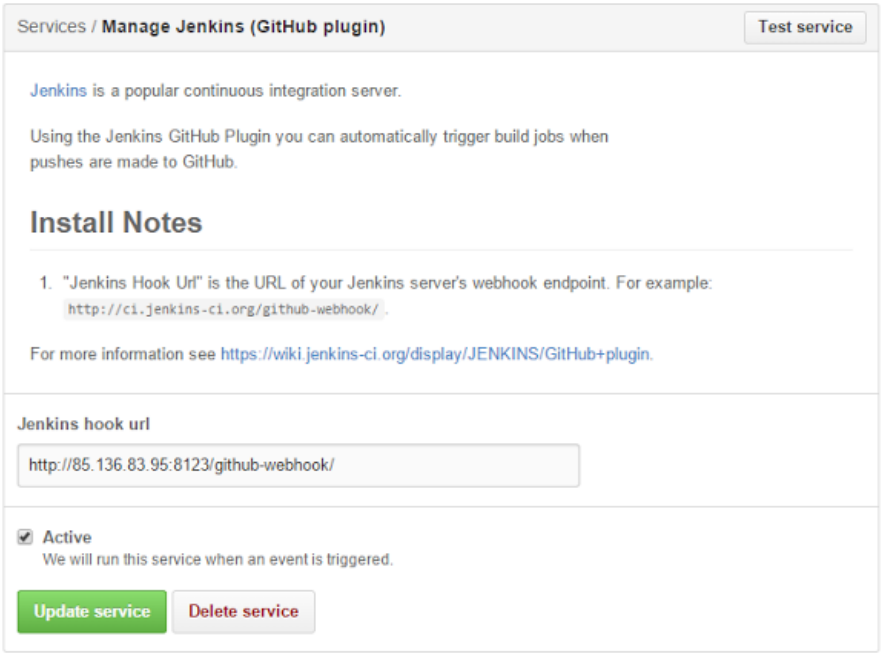
Por último, se desarrollará la interfaz que permite acceder a las diferentes funcionalidades, y una vez desarrolladas, se unirán para completar el subsistema.

Para facilitar la integración con los otros subsistemas, se usará un repositorio compartido que permita acceder fácilmente al código de los otros subsistemas. De esta manera todo el código está fácilmente accesible a todos los desarrolladores del sistema y se pueden usar técnicas de integración continua que afecten a varios subsistemas.

Para mantener la calidad, evitar cambios con efectos negativos que pasen desapercibidos y en general automatizar tantos procesos como sea posible, se usará un servidor de Jenkins en el que se realicen actividades de manera automática cada vez que se realice un push en el repositorio compartido. Esto se ha logrado mediante un plugin de Github para Jenkins que permite hacer que se lleve a cabo una tarea cuando se realice un push en un repositorio de Github y configurando el servicio de Jenkins en Github:

De git o de github?





El servidor se encuentra accesible en http://85.136.83.95/ y consiste simplemente en una pequeña interfaz desde la que acceder a Jenkins (con contraseña.

Las actividades realizadas mediante Jenkins serán las siguientes:

Análisis estático del código: se ha usado la herramienta php codesniffer. Esta permite analizar un conjunto de archivos php y detectar malas prácticas según un estándar. Este se indica en el comando de realización del análisis, y puede ser uno de los siguientes estándares disponibles: Zend, PEAR, PHPCS, Squiz y MySource.

Se ha usado el estándar Zend al ser uno de los más frecuentemente usados (junto con PEAR, estando este más enfocado a la creación de paquetes para PEAR). Algunas (pero no todas) de las malas prácticas detectadas por Zend son:

- Uso de tabulaciones en lugar de espacios.
- Uso de la etiqueta de cierre “?>” en archivos que contienen exclusivamente código php.
- Líneas de código de más de 120 caracteres.
- Uso del operador “if (...) {“ sin espacios entre “if” y “(“ o entre “)” y “{“.
- Ausencia de espacios después de la coma al separar los argumentos de una función.

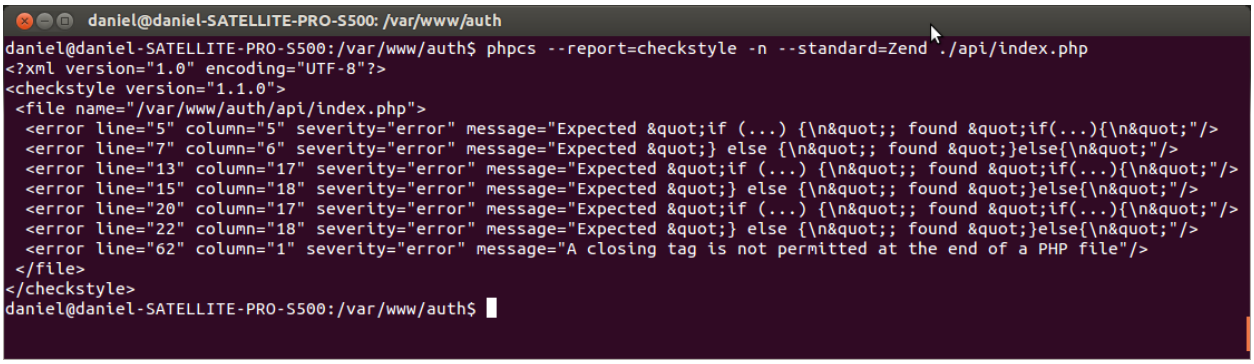
Al ejecutarse php codesniffer se genera un informe. Se puede indicar el formato. Se ha escogido el formato checkstyle, que es el usado por el plugin de Jenkins que crea informes en las tareas automatizadas. Este, además, incluye por cada error el archivo, la línea y la columna en la que ha tenido lugar (al contrario que otros formatos, como el usado por defecto).

El comando de consola usado para realizar el análisis es, por tanto:

phpcs --report=checkstyle --standard=Zend -n ./auth > checkstyle.xml

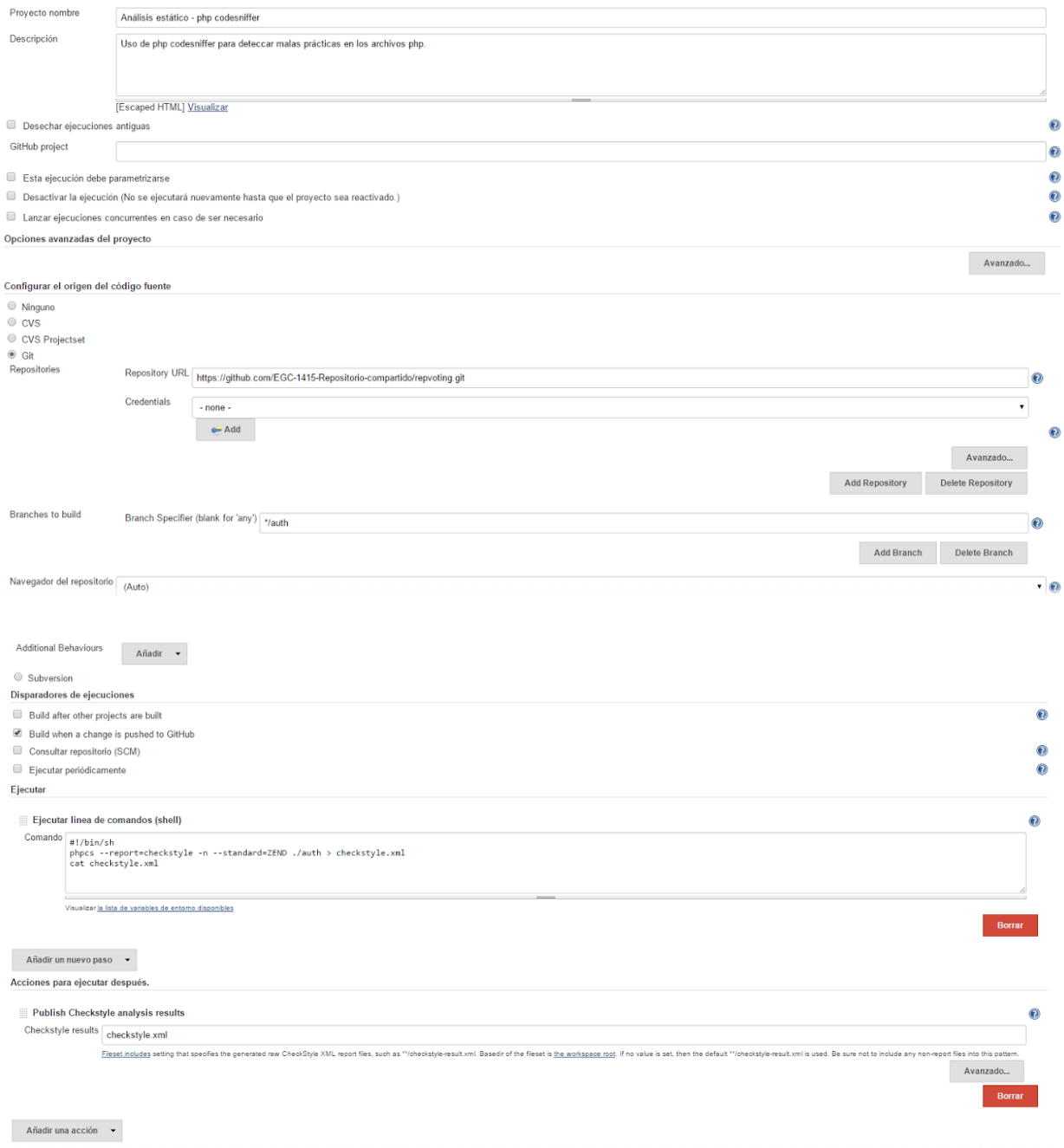
Donde:

- **--report=checkstyle** indica que el informe se debe crear en formato checkstyle.
- **--standard=Zend** indica que el estándar a usar será Zend.
- **-n** indica que solo se tendrán en cuenta los “error”, no los “warning”, que tienen menor importancia.
- **./auth** indica que el análisis debe realizarse de manera recursiva en todos los archivos del fichero “auth” (en el que se encuentra el código fuente de nuestro subsistema).
- **> checkstyle.xml** hace que el resultado se escriba en un archivo llamado checkstyle.xml.



Como se ha indicado, este análisis no se realizará manualmente, sino mediante Jenkins. El plugins Checkstyle permite crear informes a partir de los resultados de php codesniffer. Mediante una tarea automatizada, se analizan todos los archivos en el repositorio de Git. Si se detecta un error, la tarea es marcada como fallida. Esta tarea se realiza cada vez que se haga un push al repositorio. Es decir, siempre que haya un cambio no local.

La configuración final de la tarea es la siguiente:



Realización de los tests de PHPUnit Se ha usado la herramienta PHPUnit. Esta herramienta permite ejecutar pruebas unitarias de software PHP. Esta herramienta puede usarse de manera manual mediante una línea de comandos, pero la automatizaremos usando Jenkins.

Debemos preparar unos ficheros de prueba con unas clases que ejecuten unas baterías de pruebas de diferentes áreas de la aplicación. En este caso, nosotros hemos creado:

- databaseTest.php: Fichero de prueba de funcionalidades relacionadas con las operaciones con la base de datos de usuario.
- tokenTest.php: Fichero de prueba de funcionalidades relacionadas con las cookies generadas y su integridad.

Un fichero de prueba se compone de una clase que extienda de **PHPUnit_Framework_TestCase.** En el interior de esa clase están todos los métodos de cada prueba, además de otros métodos especiales:

- setUp(): Método de inicialización de prueba. Se ejecuta antes de cada prueba.
- tearDown(): Método de finalización de prueba. Se ejecuta después de cada prueba.

De esta manera, si tenemos una clase con 3 pruebas, se ejecutará el método setUp() 3 veces y el método tearDown() otras 3 veces.

Usaremos la consola de Jenkins para ejecutar el siguiente comando:

phpunit nombre_dir > nombre_res.txt

Donde:

- **nombre_dir:** Ruta relativa al directorio de entrada de los ficheros de prueba.
- **nombre_res:** Ruta relativa al fichero de texto que guardará la salida de consola.

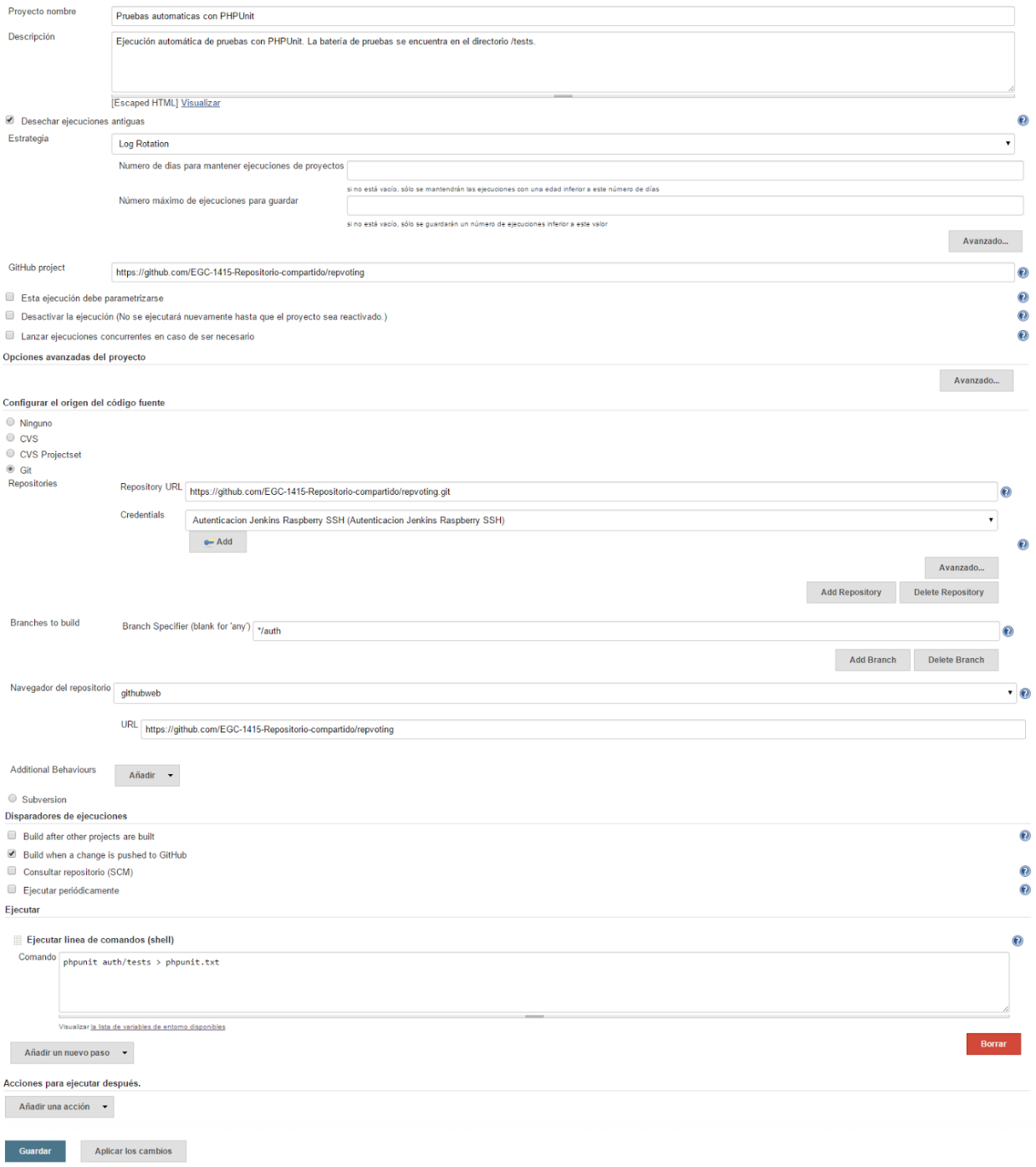
Como se ha indicado, este análisis no se realizará manualmente, sino mediante Jenkins. Usaremos una tarea automatizada que ejecute las pruebas de nuestro subsistema y genere un informe de resultados.

[En un futuro nos gustaría ampliar esta sección configurando la tarea para que envíe notificaciones automáticas informando de un posible fallo en las pruebas]

La interfaz de nuestra tarea tendrá el siguiente aspecto:

Algo sin duda deseable

esta figura se ve muy mal



Generación automática de documentación: Se ha usado la herramienta DoxyGen. Esta permite generar documentación de código C, Objective-C, C#, PHP, Java, Python, IDL, Fortran, VHDL y Tcl y exportarlo a formato web HTML, documentación LaTeX, RTF y XML. Usaremos un plugin para Jenkins el cual, cada vez que haya un cambio en la rama *auth* del servidor o por petición del usuario, cogerá los archivos mediante Git y ejecutará el análisis de sintaxis de

comentarios DoxyGen, el cual luego publicará como documentación HTML accesible desde la herramienta Jenkins.

La sintaxis de comentarios DoxyGen se resumen como:

- Utilizar comentarios de varias líneas para cabeceras de clases, funciones y variables, además de para archivos.
- Uso de palabras reservadas para interpretar una línea de comentario:
    - **@file:** Indicador de que el elemento documentado es el archivo actual.
    - **\brief:** Descripción breve.
    - **\details:** Descripción larga, extensible a varias líneas sin necesidad de repetir la palabra.
    - **\author:** Autor. Puede añadirse los símbolos “<>” para añadir un correo electrónico de contacto.
    - **\var:** Descripción de una variable.
    - **\param nombre_de_parametro:** Descripción del parámetro de entrada de un elemento.
    - **\return:** Descripción del valor de vuelta de una función.

La primera vez que se ejecuta doxygen genera un archivo de configuración **archivo.cfg** que describe todos los parámetros del proceso de generación de documentación. El comando por consola es

doxygen -g <config-filename>

Donde:

-g: Indica que debe crear el archivo <*config-filename*>

Entre los parámetros más importantes destacan:

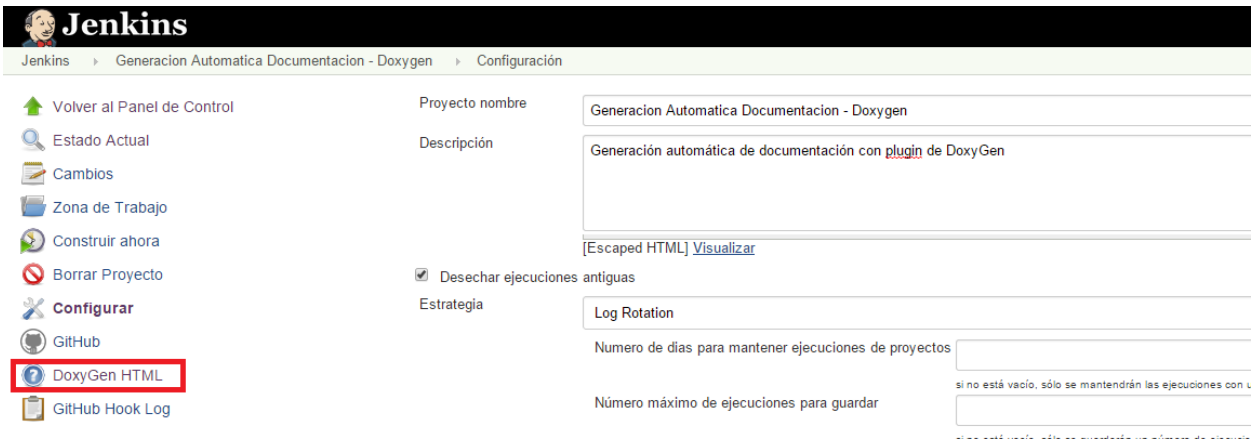
- **DOXYFILE_ENCODING:** Codificación de salida.
- **PROJECT_NAME:** Nombre de cabecera de la documentación.
- **OUTPUT_DIRECTORY:** Directorio de salida de la documentación.
- **OUTPUT_LANGUAGE:** Idioma de la documentación.
- **INPUT:** Directorio de código de entrada.
- RECURSIVE. Etiqueta para especificar que los archivos y/o directorios contenidos en el directorio de entrada serán analizados.
- **FILTER_SOURCE_FILES:** Patrón de filtrado para sólo analizar determinados formatos contenidos en el directorio de entrada.
- **GENERATE_HTML:** Etiqueta para indicar que queremos generar documentación en HTML.

- **HTML_OUTPUT:** subdirectorio de la documentación en HTML dentro del directorio de salida.
- **GENERATE_LATEX:** Etiqueta para indicar que queremos generar documentación en LaTeX.

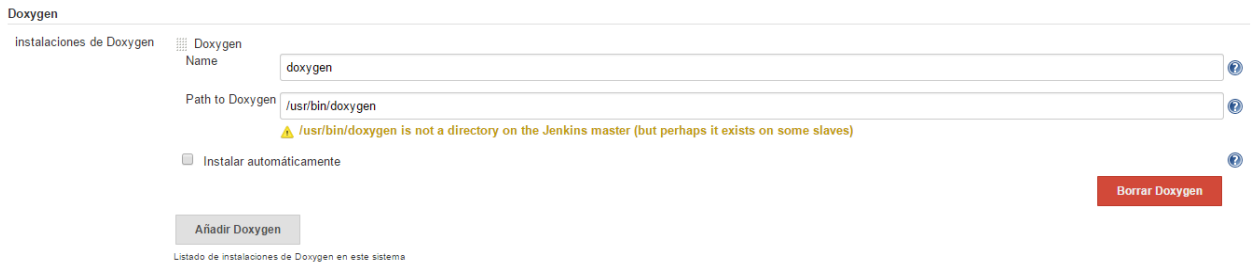
Para ejecutar la generación de documentación una vez editado el fichero de configuración se emplea el comando:

doxygen <config-filename>

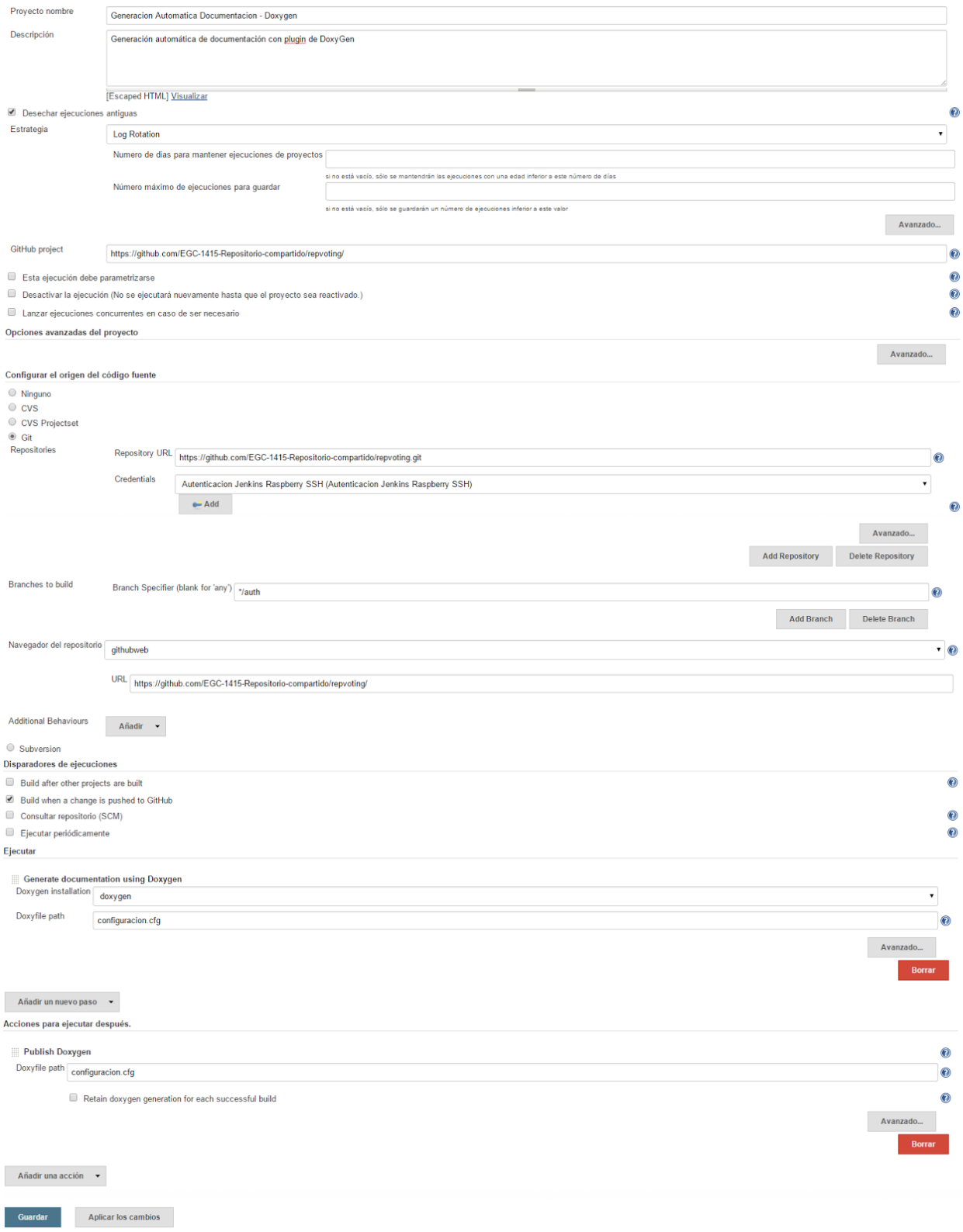
Como antes hemos mencionado, este proceso puede automatizarse mediante el plugin para Jenkins aunque sólo este último paso de ejecución. Debemos encargarnos de rellenar el archivo de configuración nosotros mismos para futuras ejecuciones. Por último, el plugin puede hacer que nuestra documentación se despliegue junto a Jenkins y poder enlazar a ella. Para acceder a la última documentación actualizada hay que hacer click en:



Un paso previo a programar la tarea es añadir la ruta de instalación al administrador de Jenkins:



Por último, la configuración de un proyecto libre de Jenkins que use estas funcionalidades de DoxyGen tiene el siguiente aspecto:



La programación de esta tarea hace que se ejecute cada vez que exista un cambio en el servidor de la rama “auth”.

Construcción e integración con otros subsistemas

Al desarrollarse el subsistema en php, este no debe ser construido mediante una compilación o la generación de un archivo (war, gem, etc.). Solo son necesarios los archivos con el código fuente y un servidor php. Se recomienda usar xampp, ya que es una forma cómoda de crear un servidor incluyendo todas las herramientas necesarias.

El despliegue, que permite la integración con el resto de subsistemas en la máquina que los contenga, se llevaría a cabo de la siguiente manera:

- Instalar Xampp.
- Iniciar el servicio Apache de Xampp, a ser posible en el puerto 80.
- Poblar la base de datos a usar con la estructura y los datos de los otros subsistemas.
- Ejecutar el archivo .sql del sistema de autenticación, que creará una tabla de usuarios con los datos necesario e introducirá datos de prueba.
- Dentro del directorio “htdocs” de xampp se copian los archivos correspondientes al subsistema de autenticación, que se encuentran en la carpeta auth del repositorio compartido repvoting.
- Asegurar que, en el archivo logAttempt.php se redirige a la página adecuada después de la autenticación.

Si no se usa xampp, simplemente se deben copiar los archivos del sistema en el directorio en el que se encuentren los recursos del servidor, que debe admitir php.

Terminados estos pasos el sistema de autenticación estará listo para su uso, encargándose de que se generan cookies identificando a los usuarios y ofreciendo una API detallada en la sección de Introducción.

El sistema de autenticación será el punto inicial del sistema completo. Después de la autenticación, se accede al resto de subsistemas. Estos se alojarán en servidores separados

Ejercicio 1

Herramientas necesarias:

- Jenkins con los siguientes plugins:
  - Git.
  - GitHub.
  - Checkstyle.
  - Static Code Analysis Plug-ins.
  - DoxyGen.
- PHP codesniffer 1.1.0

Enunciado:

- Crear una tarea en Jenkins.
- Indicar que debe usarse el código del repositorio https://github.com/EGC-1415-Repositorio-compartido/repvoting.git. Concretamente, la rama “auth”.
- Configurar la tarea para que mediante comandos de terminal que se realice un análisis de todos los archivos .php en la carpeta /auth y se cree un archivo con el informe que sea utilizable por el plugin checkstyle.
- Configurar la tarea para que se genere un informe del plugin checkstyle a partir del archivo del paso anterior.
- Configurar la tarea para que esta se ejecute siempre que se realicen cambios en el contenido del repositorio (la configuración en el lado del repositorio ya ha sido llevada a cabo).
- Forzar la ejecución de la tarea una vez.

Solución:

- Una vez en el menú de Jenkins, seleccionar “Nueva Tarea”.
- Se indica su nombre y que se trata de un “proyecto de estilo libre”. A continuación se selecciona “Ok”.
- Rellenar los campos “Nombre” y “Descripción”.
- En el campo “Configurar el origen del código fuente” seleccionar Git.
- En el campo “Repository URL” introducir la URL del repositorio proporcionado.
- En el campo “Branches to build” introducir “./auth”.
- En el campo “Disparadores de ejecuciones” seleccionar únicamente “Build when a change is pushed to GitHub.
- Añadir el paso “Ejecutar línea de comandos (shell)”.
- En el campo “Comando” introducir:
  #!/bin/sh

  phpcs --report=checkstyle -n --standard=ZEND ./auth > checkstyle.xml

  cat checkstyle.xml

- En el campo “Acciones para ejecutar después”, introducir la acción “Publish Checkstyle analysis result”.
- En el campo “Checkstyle results” introducir “checkstyle.xml”.
- Seleccionar “Guardar”.
- Seleccionar “Construir ahora”.

Estaría bien poner una pantalla del resultado de la ejecución del ejercicio

Ejercicio 2

Herramientas necesarias:

- Jenkins con los siguientes plugins:
  - Git.
  - GitHub.
  - DoxyGen.
- DoxyGen.

Enunciado:

- Crear una tarea en Jenkins.
- Indicar que debe usarse el código del repositorio https://github.com/EGC-1415-Repositorio-compartido/repvoting.git. Concretamente, la rama "auth".
- Configurar la tarea para que analice el código en la carpeta "/auth" y genere documentación actualizada cada vez que haya un cambio en el servidor de la rama.
- Forzar la ejecución de la tarea una vez.

Solución:

- En el menú de Jenkins, seleccionar "Administrador de Jenkins".
- Seleccionar "Configurar el Sistema".
- Seleccionar "Instalaciones de Doxygen...".
- En el campo "Nombre" introducir "doxygen".
- En el campo "Path to Doxygen" hay que indicar dónde está instalado DoxyGen. Introducir la ruta absoluta del ejecutable de este, que por defecto es "/usr/local/bin/doxygen". Es posible que Jenkins muestre una advertencia, pero debe ser ignorada.
- Seleccionar "Guardar".
- Una vez en el menú de Jenkins, seleccionar "Nueva Tarea".
- Se indica su nombre y que se trata de un "proyecto de estilo libre". A continuación se selecciona "Ok".
- Rellenar los campos "Nombre" y "Descripción".
- Seleccionar "Desechar ejecuciones antiguas".
- Seleccionar "Aplicar los cambios".
- Mediante la consola del sistema situarnos en la carpeta de nuestra tarea. ("var/lib/jenkins/jobs/TAREA/workspace").
- Ejecutar el comando de creación del fichero de configuración.
  doxygen -g <configuration-filename.cfg>
- Editar el fichero de configuración con la guía anterior y guardar.

- Volver al panel de configuración de la tarea.
- En el campo “Configurar el origen del código fuente” seleccionar Git.
- En el campo “Repository URL” introducir la URL del repositorio proporcionado.
- Introducir credenciales o nuestra clave SSH previamente aceptada en el servidor Git.
- En el campo “Branches to build” introducir “./auth”.
- En el campo “Navegador del repositorio” introducir “githubweb” con URL del repositorio proporcionado.
- En el campo “Disparadores de ejecuciones” seleccionar únicamente “Build when a change is pushed to GitHub.
- Añadir el paso “Generate documentation using Doxygen”.
- En el campo “Doxygen Installation” seleccionar “doxygen”.
- En el campo “Doxyfile path” introducir el nombre del fichero de configuración.
- Añadir como acción post-build “Publish Doxygen”.
- En el campo “Doxyfile path” introducir el nombre del fichero de configuración.
- Seleccionar “Guardar”.
- Seleccionar “Contruir ahora”.
- Seleccionar el enlace “DoxyGen HTML”.

Ejercicio 3

Herramientas necesarias:

en los ejercicios pasa de nuevo lo mismo: se mezcla enunciado con solución

- Jenkins con los siguientes plugins:
  - Git.
  - GitHub.
- PHPUnit.

Enunciado:

- Crear una tarea en Jenkins.
- Indicar que debe usarse el código del repositorio https://github.com/EGC-1415-Repositorio-compartido/repvoting.git. Concretamente, la rama “auth”.
- Configurar la tarea para que analice el código y ejecute la batería de pruebas en el directorio “/auth/tests”.
- Forzar la ejecución de la tarea una vez.

Solución:

- Una vez en el menú de Jenkins, seleccionar “Nueva Tarea”.
- Se indica su nombre y que se trata de un “proyecto de estilo libre”. A continuación se selecciona “Ok”.

- Rellenar los campos “Nombre” y “Descripción”.
- Seleccionar “Desechar ejecuciones antiguas”.
- En el campo “Configurar el origen del código fuente” seleccionar Git.
- En el campo “Repository URL” introducir la URL del repositorio proporcionado.
- Introducir credenciales o nuestra clave SSH previamente aceptada en el servidor Git.
- En el campo “Branches to build” introducir “./auth”.
- En el campo “Navegador del repositorio” introducir “githubweb” con URL del repositorio proporcionado.
- En el campo “Disparadores de ejecuciones” seleccionar únicamente “Build when a change is pushed to GitHub.
- Añadir el paso “Ejecutar línea de comandos (shell)”.
- En el campo “Comando” introducir el comando

  phpunit nombre_dir > nombre_res.txt

  con la ruta a las pruebas y la ruta de salida de los resultados deseada.

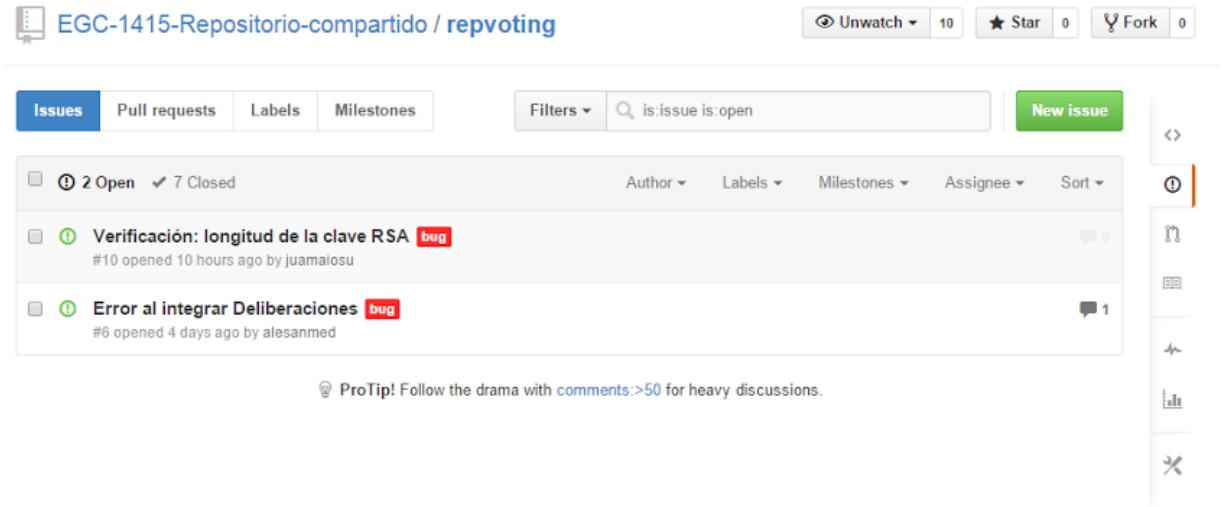
- Seleccionar “Guardar”.
- Seleccionar “Construir ahora”.

6. Gestión del cambio, incidencias y depuración

Peticiones de cambio e incidencias

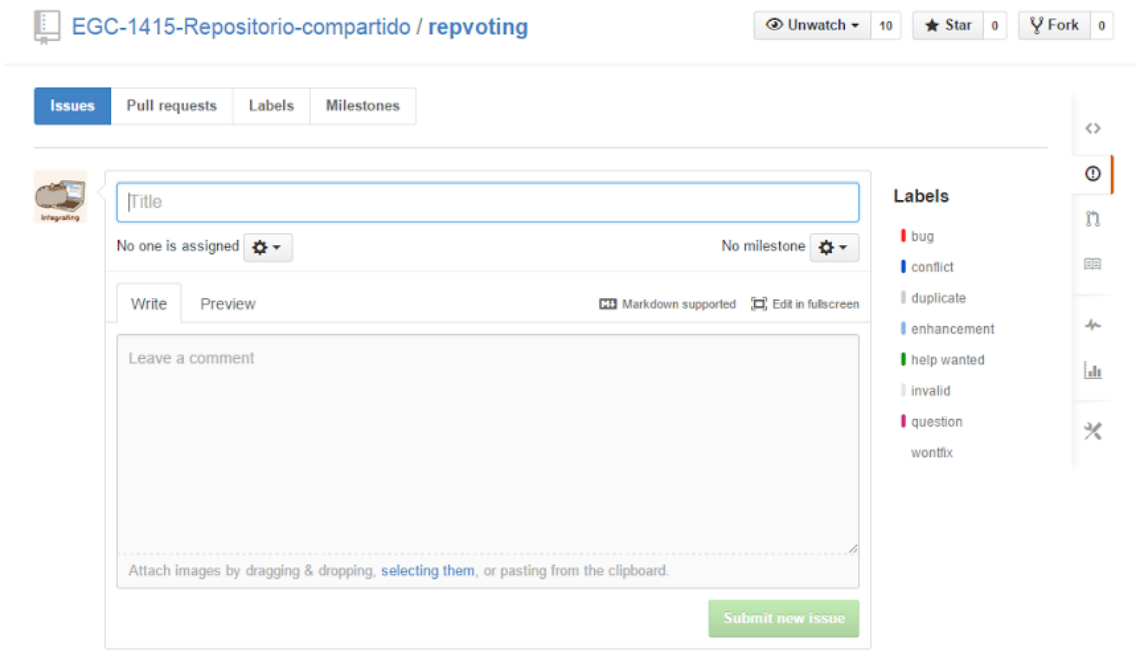
Se ha usado el mismo procedimiento para comunicar peticiones de cambio e incidencias: los issues de GitHub.

Estos son una funcionalidad ofrecida por GitHub por la cual cualquier usuario puede enviar un "aviso" a los desarrolladores de un proyecto. Desde la pestaña de "Issues", el usuario puede leer los issues, tanto abiertos como cerrados (estos estados se explicarán más adelante) y crear un nuevo issue.

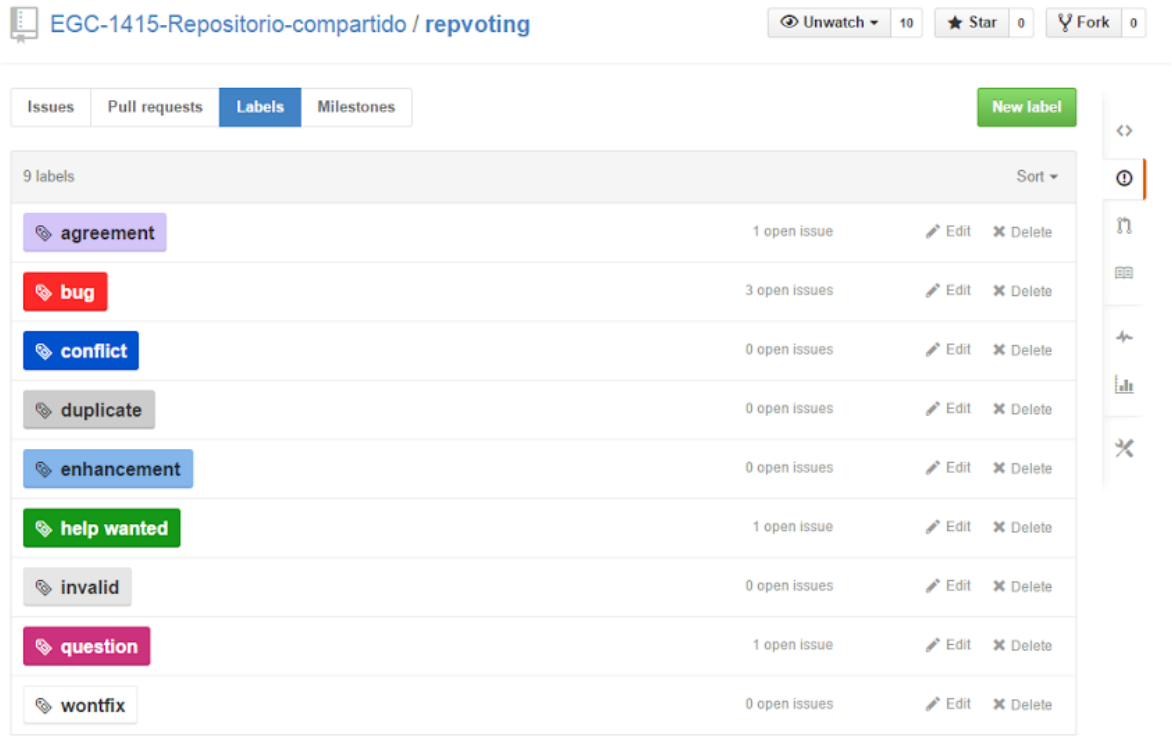


Dentro de la pantalla de creación de issue, el usuario puede:

- Darle un título al issue que lo describa brevemente.
- Asignarlo a un responsable, es decir, designar la persona que en teoría se encargaría de atender el issue.
- Escribir una descripción, con la posibilidad de incluir imágenes.
- Etiquetar el issue. Por ejemplo, se puede marcar una incidencia como "bug", una propuesta de cambio como "enhancement" o una duda sobre el funcionamiento del sistema como "question".

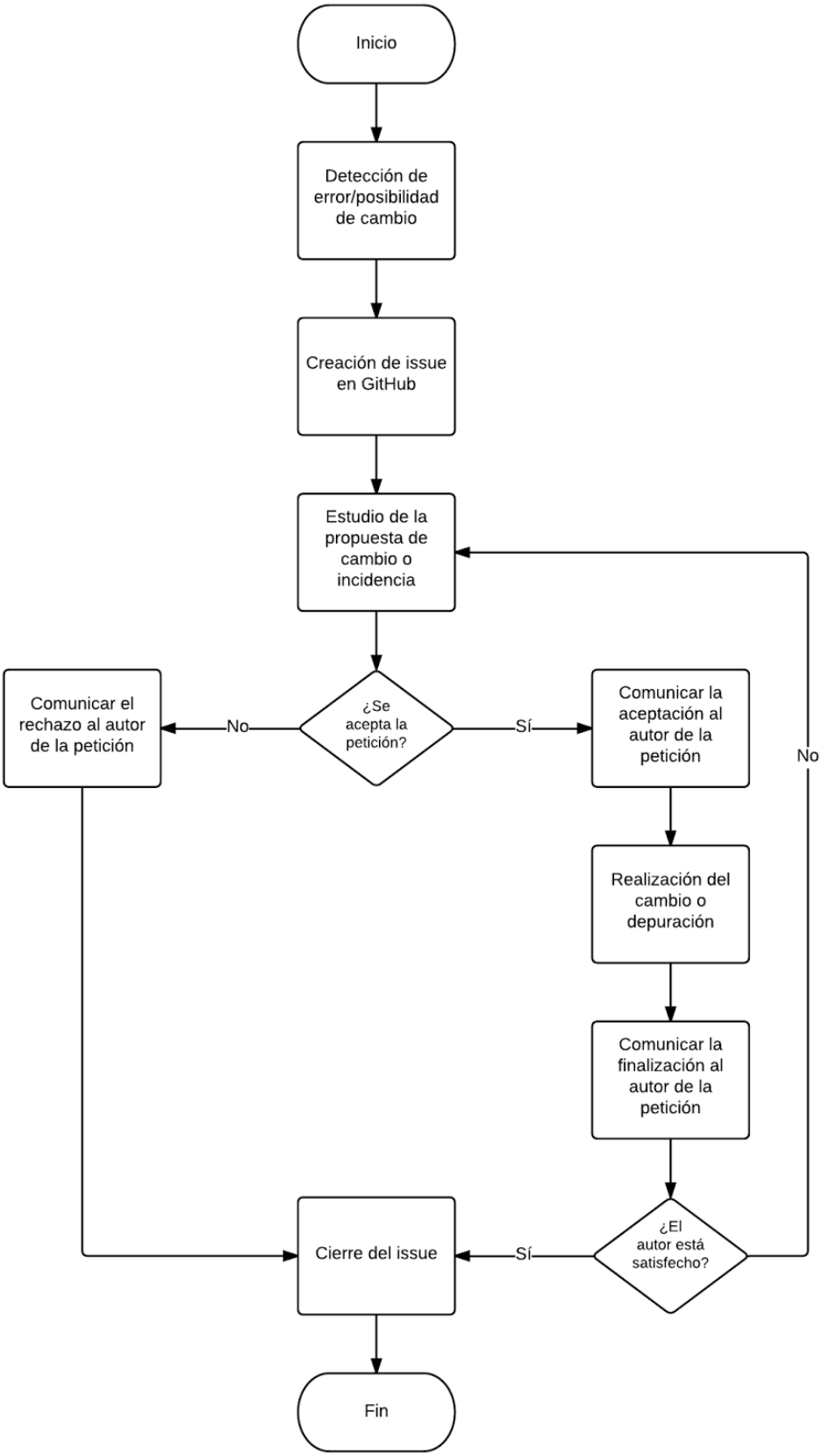


En la pestaña “Labels”, los desarrolladores del proyecto pueden crear y editar etiquetas. Nuestro grupo ha añadido las etiquetas personalizadas”conflict” y “agreement”, para aquellos issues relacionados con un conflicto entre subsistemas y las situaciones en las que hay que llegar a un acuerdo de forma colectiva.



Por esta facilidad de uso, posibilidades de personalización y el hecho de que los diferentes grupos desarrolladores de subsistemas ya usaran un repositorio en GitHub, se acordó usarlos como medio de comunicación en el tratamiento de peticiones de cambios e incidencias. En el caso de nuestro grupo, el proceso será el siguiente:

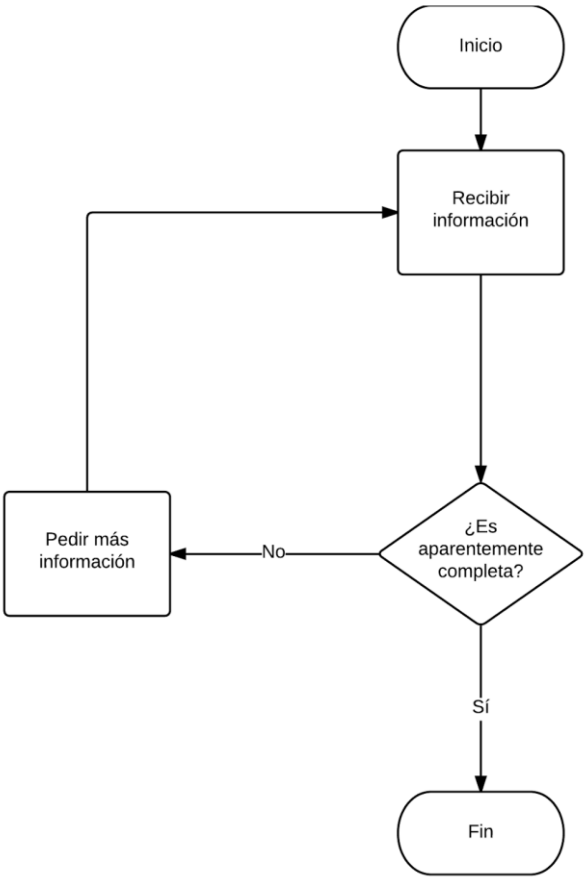
1. Un miembro del equipo de desarrollo de un subsistema detecta un error propone un cambio en nuestro subsistema.
2. Este crea un issue en GitHub detallando el error o cambio propuesto.
3. Nuestro equipo recibe una notificación avisando de que se ha creado un nuevo issue.
4. Nuestro equipo estudia la incidencia o el cambio.
5. Si se considera que el cambio no es necesario o que la incidencia no puede estar causada por un fallo en nuestro subsistema, se rechazará la petición. Se comentará el issue explicando la causa del rechazo y se cerrará el issue.
6. Si no es así, se avisará al autor del issue mediante un comentario en este de que el grupo ha sido informado y se trabajará en el cambio o incidencia.
7. Se trabaja en el cambio. Si se trata de una incidencia debe seguirse el procedimiento indicado en la sección de **Depuración**.
8. Una vez terminados los cambios se avisará al autor del issue mediante otro comentario.
9. El autor del cambio puede cerrar el issue si está satisfecho o mantenerlo abierto en caso de que no considere correcto cómo se ha realizado el cambio, volviéndose al paso 4.


Inicio
Detección de error/posibilidad de cambio
Creación de issue en GitHub
Estudio de la propuesta de cambio o incidencia
¿Se acepta la petición?
No
Comunicar el rechazo al autor de la petición
Sí
Comunicar la aceptación al autor de la petición
Realización del cambio o depuración
Comunicar la finalización al autor de la petición
¿El autor está satisfecho?
No
Sí
Cierre del issue
Fin

Depuración

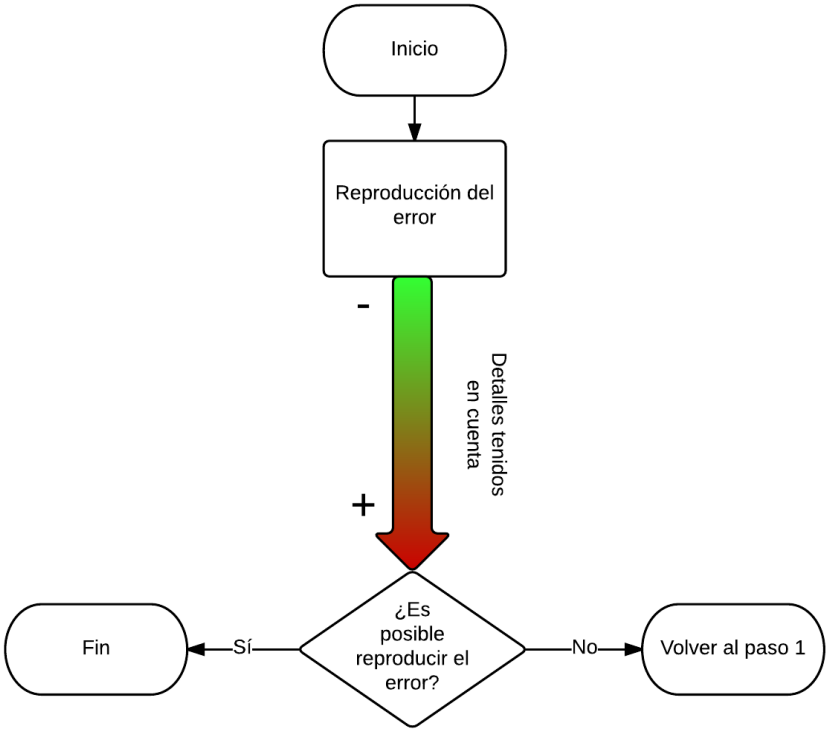
Una vez detectada una incidencia, la depuración de esta se llevará a cabo de la siguiente forma:

1. Se analizará la información proporcionada de la manera detallada en la sección de Peticiones de cambio e incidencias. Si esta se considera incompleta y poco detallada, se le pedirá al informador que amplíe la información y se volverá a analizar. Si se



considera correcta, se procederá a reproducir el error.

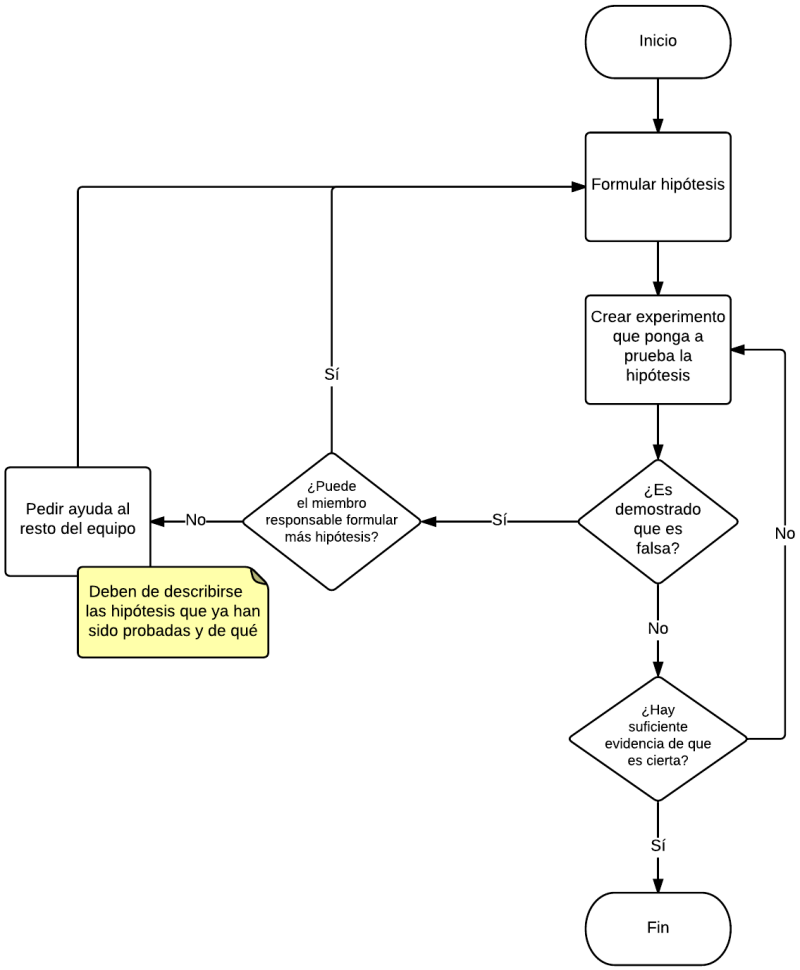
2. Se reproducirá el error a partir de la información proporcionada. Se empezará reproduciendo el error de manera general y sin tener en cuenta los detalles (por ejemplo, si la información proporcionada es “el sistema no responde al realizar la acción X con los datos Y”, primero se realizará la acción X sin usar los datos Y), y si de esta manera no se produce el error, se tendrán en cuenta más detalles progresivamente. Si es imposible reproducir el error usando toda la información proporcionada, se pedirá al informador que amplíe la información proporcionada, volviendo al paso 1.

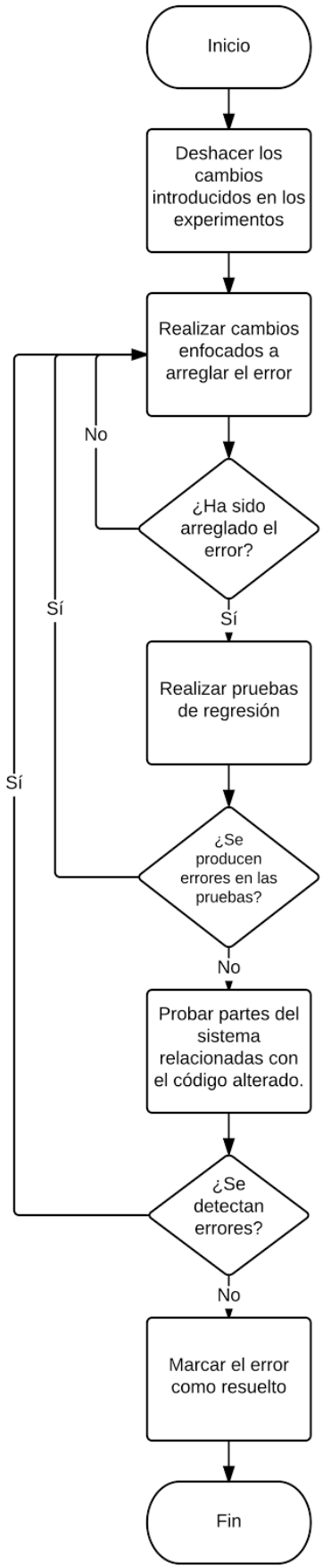


3. Una vez reproducido el error, se procederá a diagnosticar la causa. Para ello, el miembro del equipo encargado de resolver el error deberá formular hipótesis que lo expliquen. Para probar una hipótesis, se realizará un experimento (o varios si hace falta) en el que se introduzca un cambio que ponga a prueba la hipótesis. Si el experimento muestra que la hipótesis era cierta, se conocerá la causa exacta del error y se continuará al siguiente paso. Si el experimento demuestra que no era cierta, se deberá formular una nueva hipótesis y repetir el procedimiento. Si ninguna hipótesis demuestra ser cierta, el miembro encargado de resolver el error se pondrá en contacto con el resto del equipo y les informará del error para que puedan ayudar con nuevas hipótesis. Es importante que se comuniquen las hipótesis que ya han demostrado ser falsas.

4. Detectado el error, este debe ser resuelto. Este procedimiento puede variar mucho dependiendo de la naturaleza del error, pero siempre se deben seguir las siguientes indicaciones:
    - Realizar las correcciones a partir del sistema original, deshaciendo cualquier cambio realizado como parte de un experimento al realizar las hipótesis.
    - Es recomendable que se creen pruebas diseñadas específicamente para probar que el error se ha resuelto.
    - Después de comprobarse que no se produce el error, deben ejecutarse las pruebas en PHPUnit del sistema para comprobar que todo está en orden.

- Incluso si el resultado de las pruebas es el correcto, es posible que haya habido algún efecto no detectado por estas. Deberían probarse las partes del subsistema relacionadas con el código modificado.




Inicio
Deshacer los cambios introducidos en los experimentos
Realizar cambios enfocados a arreglar el error
No
¿Ha sido arreglado el error?
Sí
Sí
Realizar pruebas de regresión
Sí
¿Se producen errores en las pruebas?
No
Probar partes del sistema relacionadas con el código alterado.
¿Se detectan errores?
No
Marcar el error como resuelto
Fin

5. Una vez que está arreglado el problema y se ha comunicado el error no termina el proceso. Se puede aprender del error. Estas son algunas maneras en las que se puede reflejar el aprendizaje:
    - Es posible que el error no haya sido detectado debido a la falta de pruebas. En ese caso, se deben añadir nuevas pruebas al conjunto de pruebas con PHPUnit.
    - Es posible que el fallo haga patente la necesidad de mejora de algún proceso definido. En este caso, se debe cambiar el proceso.
    - El error puede mostrar falta de interés o atención al realizar una tarea. En ese caso, el causante debe comprometerse a intentar evitar que vuelva a ocurrir lo mismo.
    - Arreglar un error puede hacer que el grupo cuestione el correcto funcionamiento de otras partes del sistema cuyo funcionamiento se consideraba correcto. La resolución del error puede ayudar a detectar otros errores de los que no se ha informado.

Tras investigar herramientas y técnicas de depuración en PHP (lenguaje usado en nuestro subsistema), se ha visto que todas las herramientas requieren el uso de instrucciones específicas para la depuración del código, por lo que se ha decidido no usar dichas herramientas, sino técnicas usadas por otros usuarios como es el uso de funciones propias de php como **var_dump()** o **die()**.

Ejercicio

Herramientas necesarias:

- Ninguna

Enunciado:

- Crear un issue en el proyecto de GitHub https://github.com/alesanmed/repcloning.
- Darle un nombre y una descripción básicos.
- Incluir una captura de pantalla.
- Escribir un comentario sobre el issue indicando que ha sido resuelto.
- Terminar el issue de manera apropiada.

Solución:

- En el menú principal del proyecto, seleccionar “Issues”.
- Seleccionar “New issue”.
- Escribir el título en el campo “Title” y la descripción en el campo “Write”.
- Para incluir capturas, realizar la captura con la tecla de capturas de pantalla y pegarla en el campo de descripción (mediante clic derecho -> pegar o ctrl+v).
- Seleccionar “Submit new issue”.

- Para enviar un comentario, escribir el comentario en el campo “Write” al ver los detalles del issue y seleccionar “Comment”.
- Seleccionar “Close issue”.

El apartado de gestión de incidencias podría mejorarse bastante. Está muy descriptivo y poco concreto. No se justifican los usos de las etiquetas, no se deja claro si el proceso propuesto ha sido o no ha sido aplicado en algún caso concreto. Sería bueno poner ejercicios que no sean solo de reportar incidencias sino que sean casos reales de gestión de la incidencia y, en su caso, de depuración de la misma.

7. Gestión de liberaciones, despliegue y entregas

Se ha acordado desplegar la mayor parte de los subsistemas en una máquina virtual de Ubuntu. En esta, nuestro subsistema se desplegará mediante xampp como se indica en la sección 5.

El sistema de almacenamiento de votos se desplegará en la nube, en un servidor gratuito de openshift.

La máquina virtual anteriormente mencionada posee todas las herramientas necesarias para desplegar el sistema: xampp, Git, Eclipse (adaptado para poder usar proyectos de Maven) y Jenkins.

Los subsistemas deben haber sido creados de tal forma que al desplegarse de manera independiente, importando las librerías adecuadas, el sistema entero funcione correctamente, comunicándose los subsistemas relacionados entre sí.

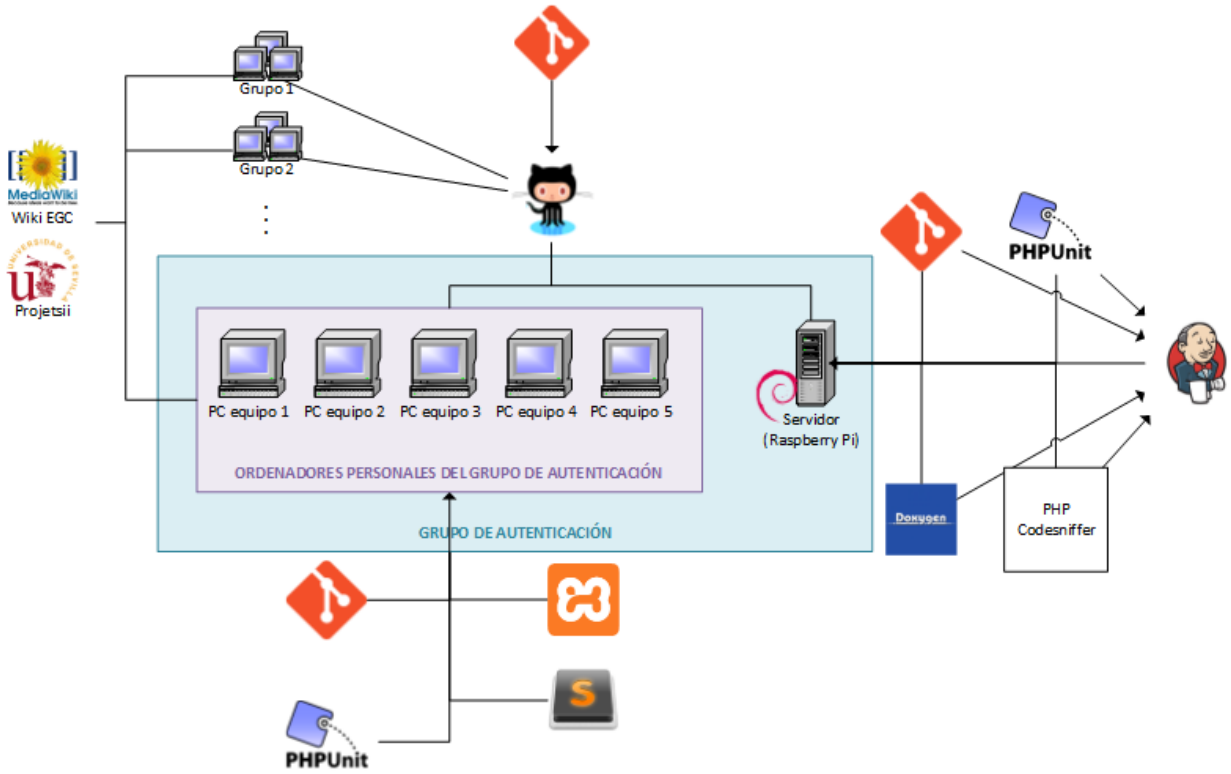
Para facilitar la configuración de estos, se decidió usar los mismos datos de autenticación en todas las bases de datos.

[Esta sección será ampliada después del taller de integración final. El método final de despliegue todavía no ha sido decidido, pero se ha documentado la opción aparentemente más probable en caso de que no se posible desplegar todos los subsistemas en un servidor en internet]

OK

8. Mapa de herramientas

Representación gráfica del mapa de herramientas



Descripción del mapa de herramientas

- Todos los grupos han usado la Wiki de EGC para redactar información sobre su grupo, hacer preguntas a los desarrolladores de AgoraVoting, etc.
- Todos los grupos han usado projetsii, concretamente los foros de este, para comunicarse entre ellos y tomar decisiones que afecten a todos los grupos, como el uso de los issues de Github.
- Para gestionar y almacenar el código fuente se ha usado un repositorio Git.
- El repositorio central está alojado en Github. Github también será usado para gestionar cambios e incidencias mediante el uso de issues.
- Todos los integrantes de los grupos tendrán acceso al repositorio, al pertenecer a la asociación registrada en este.
- Los ordenadores de nuestro grupo usan el siguiente software:
    - **Git**, para administrar el código fuente y compartirlo entre los miembros de nuestro grupo y los demás grupos.
    - **Xampp**, para disponer de un servidor Apache y MySQL de manera cómoda.

    - **PHPUnit**, para ejecutar los tests del sistema en php.
    - **Sublime Text**, para editar el código fuente.
- Nuestro grupo ha usado un servidor desplegado en una Raspberry Pi disponible en casa de uno de los miembros del grupo 24 horas al día. El sistema operativo que corren es Raspbian 3.12, una versión de Debian Wheezy.
- En esta se aloja un servidor de Jenkins en el que se realizan de forma automática tareas de integración continua. Para realizarlas obtiene el código fuente mediante Git. Son las siguientes:
    - Generación de documentación a partir del código fuente mediante Doxygen.
    - Análisis del código para detectar malas prácticas mediante php-codesniffer.
    - Ejecución de pruebas mediante PHPUnit.
- Jenkins también ha sido usado para desplegar varios subsistemas de forma automática.

9. Conclusiones

Desarrollando este sistema, nuestro grupo ha llegado a las siguientes conclusiones.

Acerca de las herramientas:

- **Git** es una potente herramienta de Control de Versiones, por lo que se decidió su uso frente a Subversion antes de que fuera impuesto en clase. Su uso ha permitido una gestión cómoda del código de la aplicación. Durante las jornadas, se nos enseñó un modelo de gestión de ramas (Git flow) que hemos adaptado a la magnitud y las características de agora@US.
- **Jenkins** es un servidor de integración continua altamente modular, que nos ha permitido realizar un seguimiento del código. Es importante que todos los miembros del grupo estén atentos a los resultados de este servidor durante el desarrollo para que sea de utilidad real. La ejecución automática de tareas hace que no haya que ejecutar tediosamente todo tipo de pruebas cuando se realiza un cambio.
- Tal vez habría sido adecuado, si el proyecto hubiera tenido mayor tamaño, disponer de un canal de **IRC** donde Jenkins notificara de los problemas de desarrollo para estimular la atención de los miembros del grupo. El desconocimiento de la aplicación de IRC en entornos empresariales nos ha llevado al uso de otros canales de información más conocidos entre el usuario medio (Telegram en este caso).
- El uso de **PHPUnit, CodeSniffer** nos ha ayudado a seguir ciertas reglas de estilo y buenas prácticas de las que no habíamos oído hablar durante el grado, para tener un código más legible.
- La generación documentación del código mediante **DoxyGen** ha facilitado disponer de documentación actualizada de este en todo momento.
- El disponer de un servidor de integración continua **disponible 24 horas**, ha facilitado llevar un seguimiento real del proyecto.
- El disponer de un entorno de **preproducción común** al resto de grupos contribuye a evitar que se produzcan conflictos provocados por la diferencia entre entornos.

Acerca del trabajo en grupo:

- El clima de colaboración entre subsistemas no ha sido tan sano como podría haber sido en muchos casos, en los que la discusión no ha sido especialmente constructiva. No obstante, conforme hemos ido mejorando nuestra comunicación y nuestro protocolo, se han ido dejando a un lado las diferencias, estableciendo relaciones mucho más **profesionales**.
- En un contexto tan variado, con tantas tareas que realizar, dejar **evidencia escrita** de los acuerdos y decisiones es fundamental para solucionar futuros conflictos.

Interesantes y buenas conclusiones