Agora@US

Se nota que habéis trabajado mucho y el trabajo se ve pero el documento es un poco caótico y lioso. Podría mejorarse su orden y lógica lo que haría incrementar la nota.

Creación y Administración de Censos

Grupo 9

Evolución y Gestión de la configuración

2014-2015

INTEGRANTES

- **David Álvarez Silva**: Jefe de proyecto
- **Antonio Juan Amador Salmerón**: Desarrollador
- **Francisco Javier Delgado Vallano**: Gestor de la configuración
- **Guiomar Fernandez de Bobadilla Brioso**: Desarrollador
- **Jose Luis García Mora**: Gestor de documentación
- **Sebastián Garrocho Capacete**: Gestor de la documentación
- **Javier Guisado Torres**: Gestor de la configuración
- **Rafael Quesada García**: Gestor de pruebas

HISTORIAL DE VERSIONES

Versión	Autor	Cambio	Fecha
0.1	• David Álvarez Silva	Creación de la estructura del documento	28/10/2014
0.2	• Javier Guisado Torres	Integración de las diferentes partes de la memoria	02/12/2014
0.3	• David Álvarez Silva	Corrección de formatos	12/12/2014
0.4	• TODOS	Revisión y corrección	18/12/2014
1.0	• David Álvarez Silva	Envío para revisión	21/12/2014

1. INDICE

INTEGRANTES **1**

HISTORIAL DE VERSIONES **2**

1. INDICE **3**

2. INDICE DE FIGURAS **5**

3. RESUMEN **8**

4. INTRODUCCIÓN **9**

5. GESTIÓN DEL CÓDIGO FUENTE **11**

5.1. GESTIÓN DE RAMAS 11
5.2. CREACCIÓN Y APLICACIÓN DE PATCH 19
5.3. ROLES 22
5.4. POLÍTICA DE NOMBRE Y ESTILOS UTILIZADOS EN EL CÓDIGO FUENTE 23
5.5. EJERCICIO 25

6. GESTIÓN DE LA CONSTRUCCIÓN E INTEGRACIÓN CONTINUA **29**

6.1. GESTIÓN DE LA CONSTRUCCIÓN 29
6.1.1. CONSTRUCCIÓN DEL PROYECTO *30*
6.1.2. DESCARGA DEL CÓDIGO FUENTE EN ENTORNO *32*
6.1.3. CONSTRUCCÍON Y EJECUCIÓN *35*
6.2. INTRODUCCIÓN A LA INTEGRACIÓN 39
6.2.1. INTEGRACIÓN INTERNA *41*
6.2.2. INTEGRACIÓN EXTERNA *59*
6.3. EJERCICIO 1 65
6.4. EJERCICIO 2 74

7. GESTIÓN DEL CAMBIO, INCIDENCIA Y DEPURACIÓN **78**

7.1. GESTIÓN DE CAMBIOS 78
7.2. GESTIÓN DE INCIDENCIAS 80
7.3. DEPURACIÓN 90
7.4. EJERCICIO 1 94
7.5. EJERCICIO 2 98

8. GESTIÓN DE DESPLIEGUE **102**

8.1. PROPUESTA DE DESPLIEGUE 102

9. MAPAS DE HERRAMIENTAS 105

9.1. MAPA DE HERRAMIENTAS PROPIO 105

9.1.1. SISTEMA OPERATIVO 105

9.1.2. ENTORNOS 105

9.1.3. LENGUAJES 105

9.1.4. SERVIDORES 106

9.1.5. BASE DE DATOS 106

9.1.6. REPOSITORIOS 106

9.1.7. FRAMEWORKS 107

9.1.8. HERRAMIENTAS DE ENTORNOS 107

9.1.9. REPRESENTACIÓN GRÁFICA DE LA RELACIÓN ENTRE LAS HERRAMIENTAS 108

9.2. MAPA DE HERRAMIENTAS GENERAL 109

9.2.1. SISTEMAS OPERATIVOS 109

9.2.2. ENTORNOS 109

9.2.3. LENGUAJES 109

9.2.4. SERVIDORES 110

9.2.5. BASE DE DATOS 110

9.2.6. REPOSITORIOS 110

9.2.7. FRAMEWORKS 111

9.2.8. HERRAMIENTAS DE ENTORNO 112

9.2.9. INTEGRACIÓN CONTINUA 112

9.2.10. REPRESENTACIÓN GRÁFICA DE LA RELACIÓN ENTRE LAS HERRAMIENTAS 113

10. CONCLUSIONES 114

BIBLIOGRAFÍA 115

GLOSARIO DE TÉRMINOS 116

ANEXOS 118

INSTRUCCIONES MÁQUINA VIRTUAL 118

2. INDICE DE FIGURAS

FIGURA 1: VISTA DE TADOS LOS TAGS DEL REPOSITORIO 12
FIGURA 2: VENTANA DE COMMIT 13
FIGURA 3: MENÚ TEAM DESPLEGADO 14
FIGURA 4: REALIZANDO UN MERGE 15
FIGURA 5: CREACIÓN DE UN NUEVO TAG 16
FIGURA 6: REVISIÓN DE CONFLICTO 17
FIGURA 7: COMMIT DEL CAMBIO REALIZADO 18
FIGURA 8: SITUARSE EN LA RAMA DE CAMBIOS 19
FIGURA 9: CAMBIO EN EL CÓDIGO 20
FIGURA 10: FICHERO PATCH 21
FIGURA 11: EJEMPLO DE CÓDIGO FUENTE 24
FIGURA 12: ESTRUCTURA DEL REPOSITORIO 25
FIGURA 13: MENÚ TEAM 26
FIGURA 14: RAMA CON LA QUE REALIZAR EL MERGE 26
FIGURA 15: MENÚ TEAM 27
FIGURA 16: CREACIÓN DE TAG 28
FIGURA 17: ESTRUCTURA DEL REPOSITORIO 32
FIGURA 18: ESTRUCTURA DEL REPOSITORIO 41
FIGURA 19: VERIFICACIÓN DEL PROYECTO 44
FIGURA 20: RESULTADO DE LA VERIFICACIÓN DEL PROYECTO 45
FIGURA 21: EMPAQUETADO DEL PROYECTO 45
FIGURA 22: RESULTADO DEL EMPAQUETAMIENTO 46
FIGURA 23: WAR GENERADO 46
FIGURA 24: CREACIÓN DEL PROYECTO EN JENKINS 47
FIGURA 25: CONFIGURACIÓN DE REPOSITORIO GIT EN JENKINS 48
FIGURA 26: OPERACIONES DEL PROYECTO EN JENKINS 48
FIGURA 27: PANEL DE CONTROL DEL PROYECTO 49
FIGURA 28: INFORME DE JENKINS DE VALIDACIÓN DEL PROYECTO 49
FIGURA 29: INFORME DE JENKINS DE GENERACIÓN DEL WAR 50
FIGURA 30: DIRECTORIO EN EL QUE SE HA GENERADO EL WAR 50
FIGURA 31: NOTIFICACIONES DE CORREO 51
FIGURA 32: COMANDO PARA ARRANCAR XAMPP 51
FIGURA 33: COMANDO PARA INTEGRAR AUTENTICACIÓN Y CENSOS 52
FIGURA 34: VISTA DEL REPOSITORIO EN PROJETSII 53
FIGURA 35: CAMBIOS REALIZADOS EN EL REPOSITORIO (PROJETSII) 53

FIGURA 36: MAPA DE INTEGRACIÓN INTERNA 55
FIGURA 37: MAVEN DESCARGANDO LIBRERÍAS DEL PROYECTO 57
FIGURA 38: COMMIT EN LOCAL 63
FIGURA 39: COMMIT AND PUSH 64
FIGURA 40: INICIAR SERVICIO XAMPP 65
FIGURA 41: MENÚ CONEXTUAL DE LA VENTANA PROJECT EXPLORER 66
FIGURA 42: VENTANA IMPORT 66
FIGURA 43: VENTANA GIT 67
FIGURA 44: MPORTAR PROYECTO GIT 67
FIGURA 45: VISTA DE TODAS LAS RAMAS DE UN REPOSITORIO 68
FIGURA 46: SELECCIÓN DE LA RAMA A IMPORTAR 69
FIGURA 47: IMPORTAR PROYECTO EXISTENTE 70
FIGURA 48: CONFIRMAR PROYECTO A IMPORTAR 70
FIGURA 49: VISTA DEL WORKSPACE 71
FIGURA 50: PHPMYADMIN 72
FIGURA 51: DATA.XML 72
FIGURA 52: PERSISTENCE.XML 73
FIGURA 53: IMPORTAR PROYECTO GENERAL 75
FIGURA 54: MENÚ CONFIGURE DE UN PROYECTO 76
FIGURA 55: CREAR FICHERO POM.XML 77
FIGURA 56: NUEVA INCIDENCIA EN BUGTRACKER 83
FIGURA 57: NOTIFICACIONES EN BUGTRACKER 85
FIGURA 58: LISTADO DE ERRORES EN BUGTRACKER 85
FIGURA 59: ESTADO DE UNA INCIDENCIA EN BUCGTRACKER 86
FIGURA 60: NOTIFICACIÓN DE ERROR SOLUCIONADO 87
FIGURA 61: INFORME DE ERRORES 88
FIGURA 62: EJEMPLO DE EXCEPCIÓN 91
FIGURA 63: MODO DEPURACIÓN DE ECLIPSE 92
FIGURA 64: FICHERO CON INCIDENCIA POR RESOLVER 94
FIGURA 65: RESOLUCIÓN ESPERADA DE LA INCIDENCIA 94
FIGURA 66: INCIDENCIA POR RESOLVER 95
FIGURA 67: DETALLES DE LA INCIDENCIA 95
FIGURA 68: CAMBIO DE ESTADO EN UNA INCIDENCIA 96
FIGURA 69: HISTORIAL DE CAMBIOS EN UNA INCIDENCIA 97
FIGURA 70: CREACIÓN DE ISSUE EN GITHUB 98
FIGURA 71: DETALLES DE INCIDENCIA 99
FIGURA 72: NOTIFICACIÓN EN GITHUB DE RESOLUCIÓN 99
FIGURA 73: NOTIFICACIÓN POR EMAIL DE RESOLUCIÓN 100

FIGURA 74: REGISTRO DE INCIDENCIAS ABIERTAS 101
FIGURA 75: MAPA DE HERRAMIENTAS PROPIO 108
FIGURA 76: MAPA DE HERRAMIENTAS GENERAL 113

3. RESUMEN

En este proyecto realizamos el subsistema de creación y administración de censos del sistema Agora@US. A lo largo de las siguientes páginas se detalla la evolución y gestión de la configuración del nombrado subsistema.

Se abordan temas como la gestión del código fuente en el que utilizamos dos herramientas distintas de control de versiones como son Subversion y Github (para integración interna y externa respectivamente).

En la gestión de la construcción y la integración continua abordamos la construcción del proyecto a partir del código fuente y la integración interna del subsistema (haciendo uso de la herramienta Jenkins, explicada en el desarrollo de la asignatura, y Maven) así como la integración externa con el resto de subsistemas del proyecto general.

Para la gestión de cambio tratamos la gestión de impactos en el proyecto que a su vez está íntimamente unida a la depuración para minimizar o eliminar los diversos problemas que estos cambios puedan traer al proyecto que no son más que las incidencias o reportes de estos errores.

A la hora del despliegue, se ha elaborado una propuesta en común con otros grupos, para realizarlo con el resto de subsistemas.

En lo referente a las herramientas utilizadas, tanto interna como externamente, se detalla la relación entre estas en el apartado de mapa de herramientas.

Finalmente se aportan una serie de conclusiones en cuanto al desarrollo y evolución del trabajo en la asignatura de EGC.

4. INTRODUCCIÓN

Agora Voting es una herramienta que nos permite realizar votaciones de manera segura y fiable a través de internet. Dicha aplicación será recreada en la medida de lo posible por la asignatura de Evaluación y Gestión de la Configuración (EGC) en el curso 2014/2015. Debido a la complejidad del sistema, se optó por dividir en módulos funcionales, los cuales se asignaron a distintos subgrupos con un máximo de ocho componentes. Los módulos en los cuales se acordó dividir el proyecto son:

- Autenticación
- Creación/administración de votaciones
- Sistema de modificación de resultados
- Almacenamiento de votos
- Deliberaciones
- Recuento
- Creación/Administración de censos
- Frontend de Resultados
- Visualización de resultados
- Verificación
- Cabina de votación

Nuestro grupo, se encarga de la Creación/Administración de censos.

Dicho módulo se ocupa de asegurar cuales son los posibles votantes y que estos estarán recogidos en un censo asociado a una votación. Los votantes podrán ver que su voto ha sido almacenado correctamente y si ha votado con anterioridad.

De manera técnica se trata de realizar una interfaz del sistema, donde un usuario, en este caso el administrador/creador de la votación puede indicar cuales son los participantes/votantes de dicha votación. De estos participantes se llevará un control para evitar que un votante pueda votar en más de una ocasión, impidiendo así

problemas a la hora de relacionar a un votante con su voto, ya que si esto fuera posible se podrían modificar los votos y por tanto no sería fiable dicho sistema. Además de proporcionar dicha información a los módulos que así lo requieran.

El objetivo general del trabajo es que el grupo ponga en práctica y observe cómo se ponen en funcionamiento 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.

En nuestro caso, tenemos una relación directa con los módulos de *Autenticación*, *Creación/Administración de votaciones, Cabina de votación y Deliberaciones*. Por tanto, se determinará en cada apartado los problemas y soluciones que hemos encontrado al relacionarnos con dichos módulos.

5. GESTIÓN DEL CÓDIGO FUENTE

5.1. GESTIÓN DE RAMAS

La gestión de ramas se ha dividido en dos gestores de versiones, SVN con ProjETSII y Github.

SVN se utilizó para la integración interna, es decir, el desarrollo del subsistema en el equipo mientras que en Github se iban subiendo versiones estables de funcionalidad para tener al resto de subsistemas con la versión más actualizada de nuestra parte del proyecto.

Cuando se consiguió una primera versión final, ya se dejó de utilizar SVN y se empezaron a realizar pequeños cambios sobre Github para solventar pequeños fallos reportados por otros grupos o detectados por el nuestro.

Si se detecta un error con varias horas para su resolución, se volvería a crear una rama en SVN y se empezaría de nuevo el desarrollo.

Con SVN, todas las acciones se realizan con el repositorio remoto.

Tenemos que diferenciar dos tipos de ramas en las que se ha ido desarrollando el sistema, una que sería el *trunk*, rama común a todos los desarrolladores, en la que se almacena una versión estable del sistema, es decir, la última versión hasta el momento que ha sido testeada y por tanto aprobada; y otra rama destinada al desarrollo de nuevas funcionalidades o modificaciones de la versión almacenada en el *trunk* ya sea por cambios de requisitos, cambios en funcionalidades, etc.; que parte del *trunk* para pasar a ser posteriormente una nueva versión del mismo.

Además de esto, en la gestión de ramas hay que tener en cuenta otro elemento que es el *tag*. Éste mantiene *snapshots* o copias de seguridad del proyecto, es decir, del código almacenado en *trunk.*

En la siguiente imagen se muestra un ejemplo:

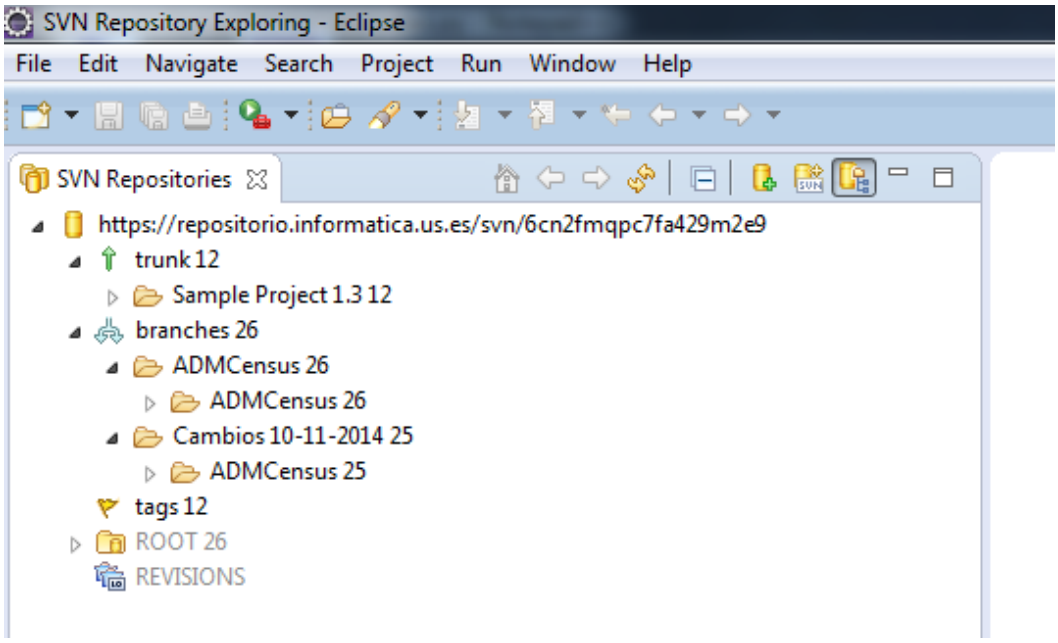


Figura 1: Vista de tados los Tags del repositorio

Como *baseline* inicial se estableció un sistema que sólo contenía la clase de dominio a manejar, que en éste caso se denomina "Census", la cual contiene los atributos y los datos necesarios para la manipulación de datos del sistema.

Por tanto, una vez establecido el *trunk*, el administrador, el cual es el creador del repositorio y el gestor del mismo, tiene la opción de abrir uno o varios *branches* o ramas, en las que se llevarán a cabo las tareas de desarrollo paralelamente.

Estos *branches* son repartidos a distintos integrantes del grupo a los que se les asignarán los permisos correspondientes, ya sea de lectura, escritura o ambos.

En el desarrollo del sistema, se abrieron dos *branches*, ya que debido a la envergadura del sistema, no fueron necesaria más.

El desarrollo realizado en un *branch* determinado queda salvado por los ***commits*** en los que antes de realizar ésta acción hay que describir las tareas realizadas con el fin de llevar un control de las versiones de los mismos.

En cuanto al sistema de *commit* se decidió no realizar uno "por cada línea de código realizada", ya que genera dificultades a la hora de comparar versiones. Al

mismo tiempo, no se vio factible la idea de hacer commit el último día de la entrega por dos razones:

- Posibilidad de pérdida de cambios en local por error o rotura de equipo
- Puede que una funcionalidad dependa de la otra por lo que podemos atrasar el desarrollo de uno de los desarrolladores.

Cada desarrollador trabaja en una funcionalidad determinada en local hasta que este decide que su tarea esta lista y puede hacer un commit en la rama.

En la siguiente imagen podemos comprobar un ejemplo de ello.

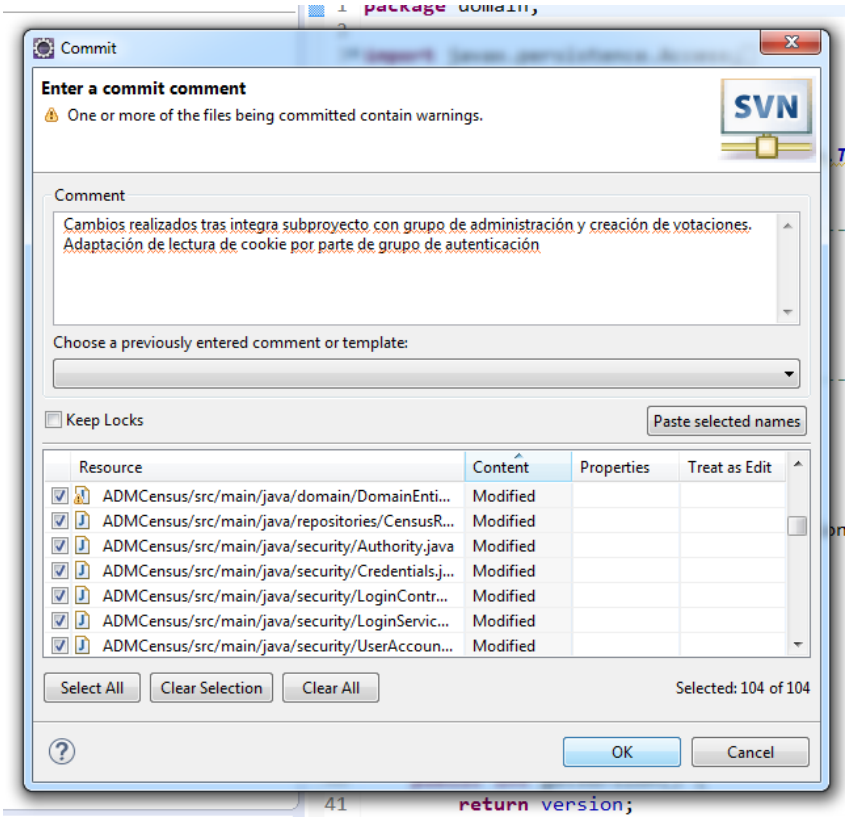


Figura 2: Ventana de commit

Las ramas fueron creadas con una fecha límite, la cual se correspondía con la fecha establecida previamente, por lo que antes de la finalización de dicho plazo el desarrollo a llevar a cabo debía ser completado y por tanto probado para realizar posteriormente un *merge*.

Los ***merges***, son la integración de las nuevas funcionalidades con la rama principal o *trunk*, de modo que dicha rama pasaría a tener una nueva versión estable y el código con la máxima funcionalidad posible. Éstos fueron realizados por el administrador del repositorio, tras obtener la aprobación.

En las siguientes imágenes se puede observar ejemplos de ello:

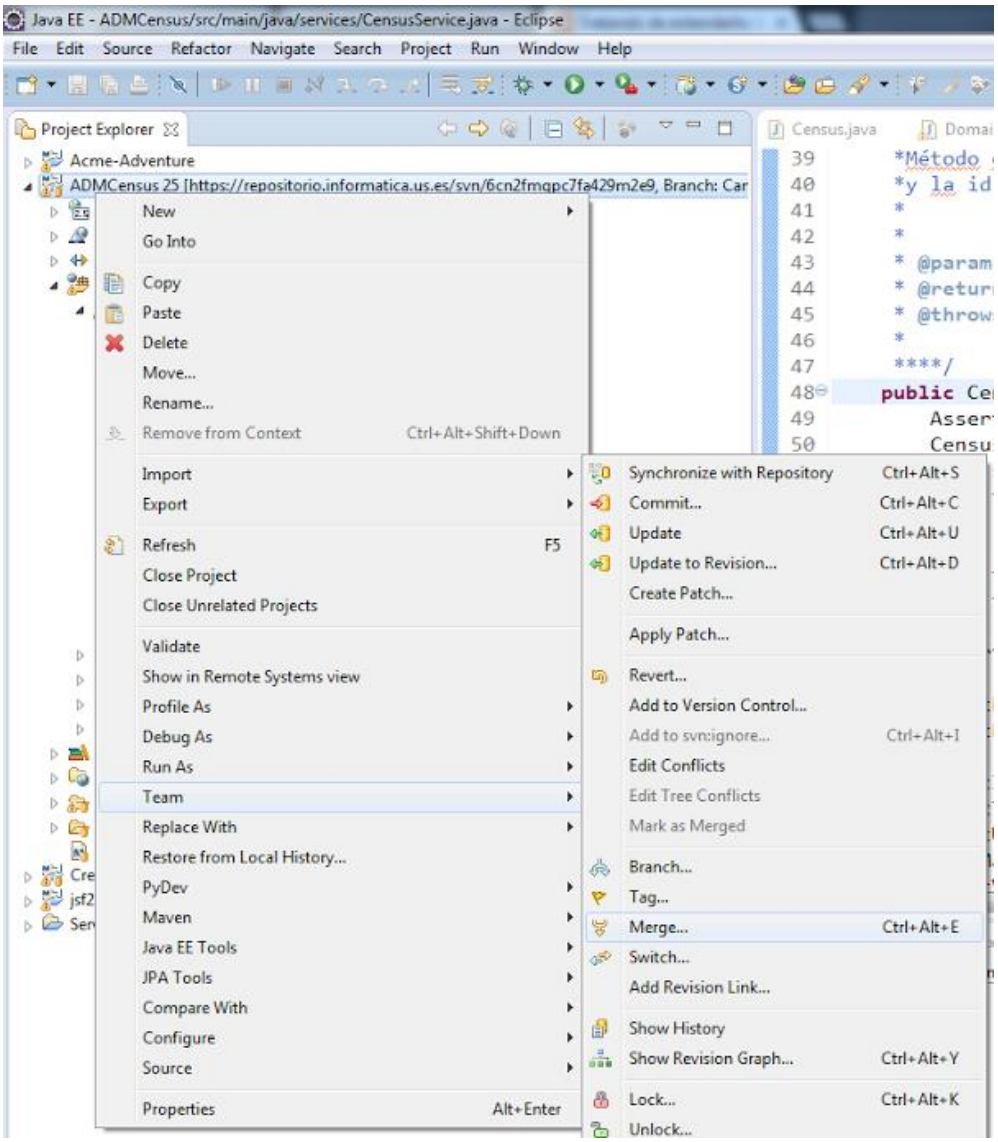


Figura 3: Menú Team desplegado

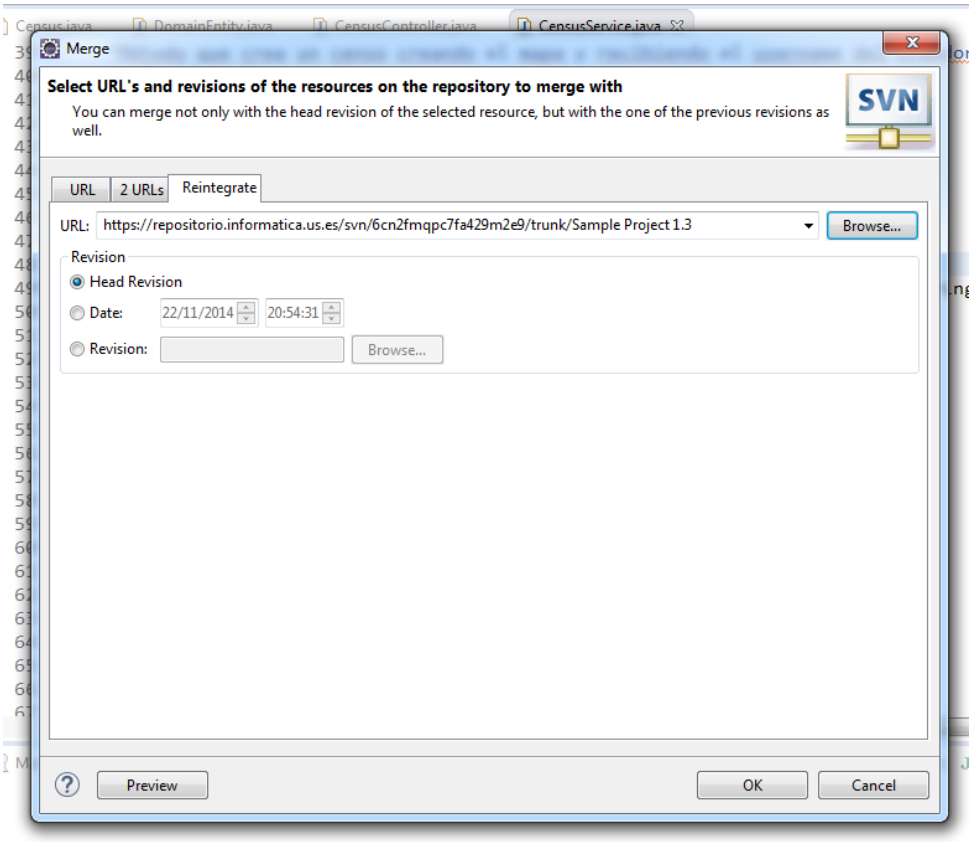


Figura 4: Realizando un Merge

Una vez realizado el *merge* de una determinada rama, ésta se considera cerrada y por tanto acaba siendo cerrada por el equipo de desarrollo. Exceptuando los casos en los cuales se requiera modificar la funcionalidad tratada en dicha rama cerrada.

Como nota, cabe destacar que en el caso de la integración del código con otros subsistemas, solo se realizarán en el caso de que dichos subsistemas sean totalmente estables para evitar el retroceso en cuanto al progreso del desarrollo.

Entonces, una vez alcanzado este punto en el que la versión del *trunk* ha cambiado se crea un nuevo *tag*, ya que se ha alcanzado una nueva versión estable.

En la próxima imagen muestra la creación de un nuevo *tag*:

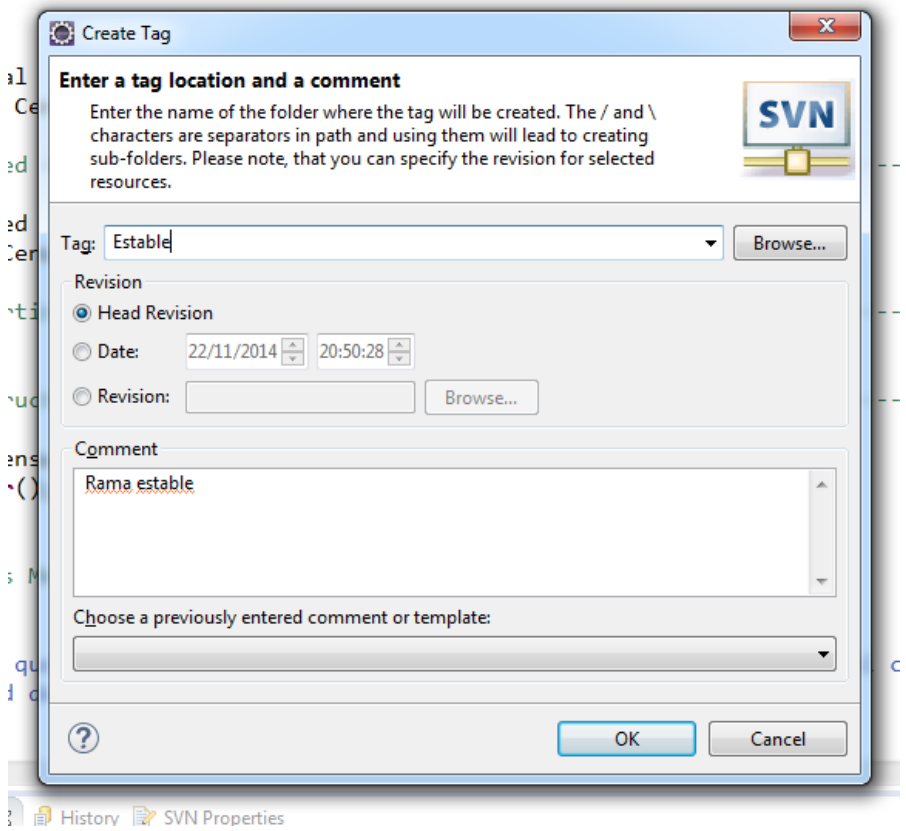


Figura 5: Creación de un nuevo Tag

Por tanto, como podemos comprobar la ventaja de esto, reside en que si al desarrollar la tarea asignada en un determinado *branch* se encuentra con un error o no puede continuar con el desarrollo debido a algún problema como puede ser que dicha funcionalidad no haga falta desarrollar, se puede volver a la versión almacenada en el *tag*, es decir, al *baseline*.

Finalmente, cabe destacar, las situaciones en las que se producen **conflictos**, situaciones en las que un determinado archivo que se pretende subir al repositorio tiene distinta versión a la que tiene el remoto o bien se ha modificado una línea existente por lo que se generan conflictos.

Dichas situaciones son difíciles de solventar, por lo que para llevar a cabo el desarrollo del sistema se han mantenido buenas prácticas de desarrollo con el fin de evitar esto mismo, aunque es complicado evitar los conflictos en el desarrollo software.

Aun así, en el caso de que surja algún conflicto, el encargado de resolverlo será el mismo desarrollador que lo haya generado, debiendo asegurarse de qué termina subiéndose al repositorio y qué es lo que se desecha.

En la siguiente imagen se puede comprobar lo anteriormente descrito de cómo se revisa un conflicto para ver qué cambios se persisten. En este caso, vemos que tenemos en local una variable “fecha_inicio” y en el repositorio “fecha_comienzo”, en este caso, vamos a llevar el cambio del repositorio al local para quitar el conflicto:

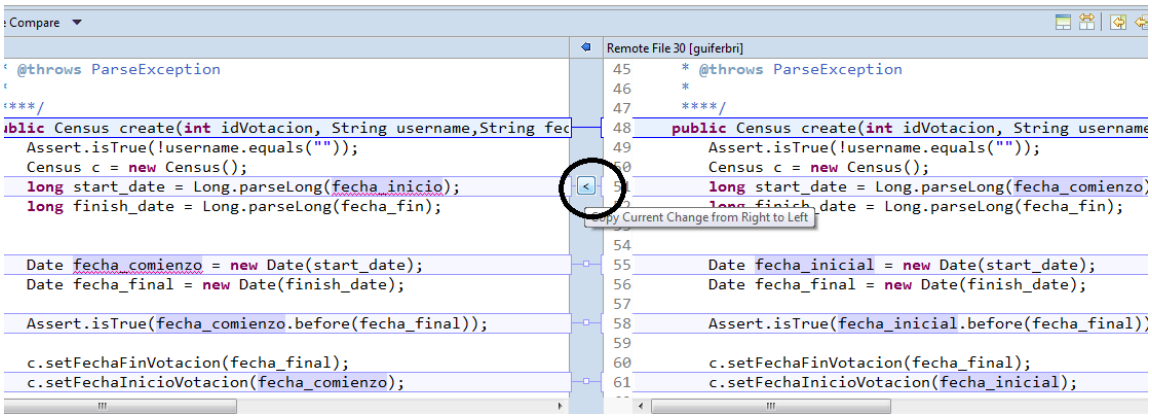


Figura 6: Revisión de conflicto

Por seguridad, el encargado de resolver los conflictos, si ve que es un cambio en la funcionalidad considerable, realizará un par de test funcionales con la herramienta JUnit para asegurar el correcto funcionamiento del proyecto.

Una vez asegurado dicho funcionamiento, en la siguiente imagen, vamos a realizar un commit para llevar este cambio al repositorio.

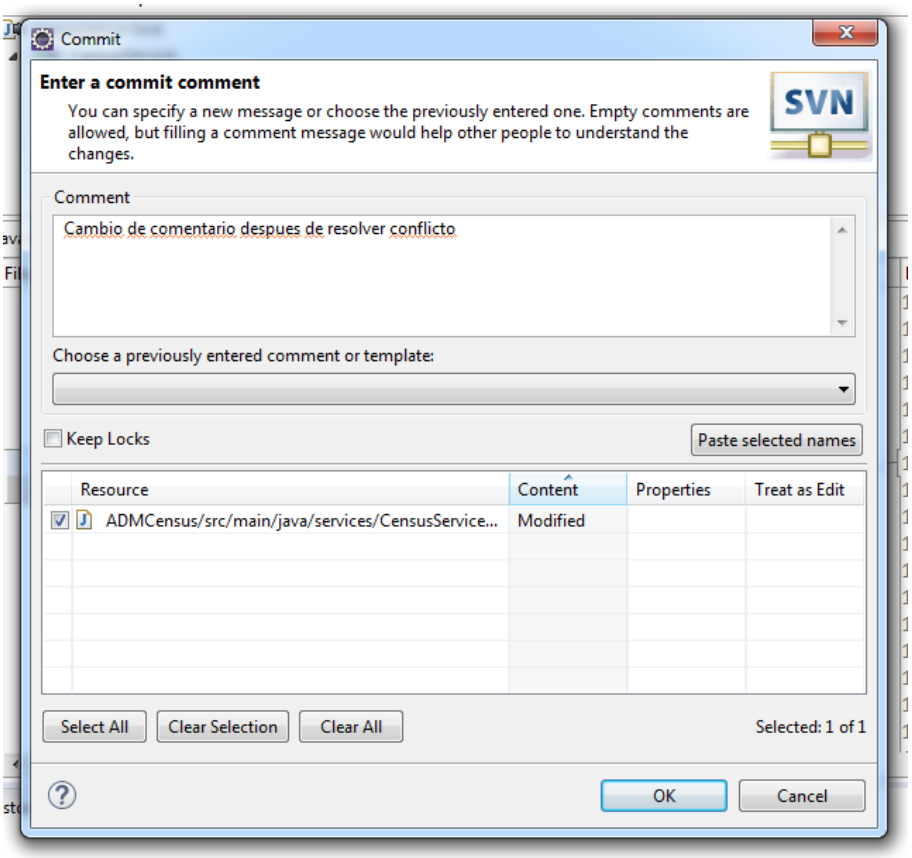


Figura 7: Commit del cambio realizado

5.2. CREACCIÓN Y APLICACIÓN DE PATCH

Para **generar un patch** con la herramienta Subversión es muy sencillo realizarlo mediante una ventana de comandos. En primer lugar hay que dirigirse hasta el directorio en el cual se encuentra el WorkSpace con el proyecto enlazado al repositorio mediante SVN. Si el fichero del cual se desea generar un *patch* se encuentra en una rama, como es el caso de la siguiente imagen, habrá que dirigirse a dicha la rama:

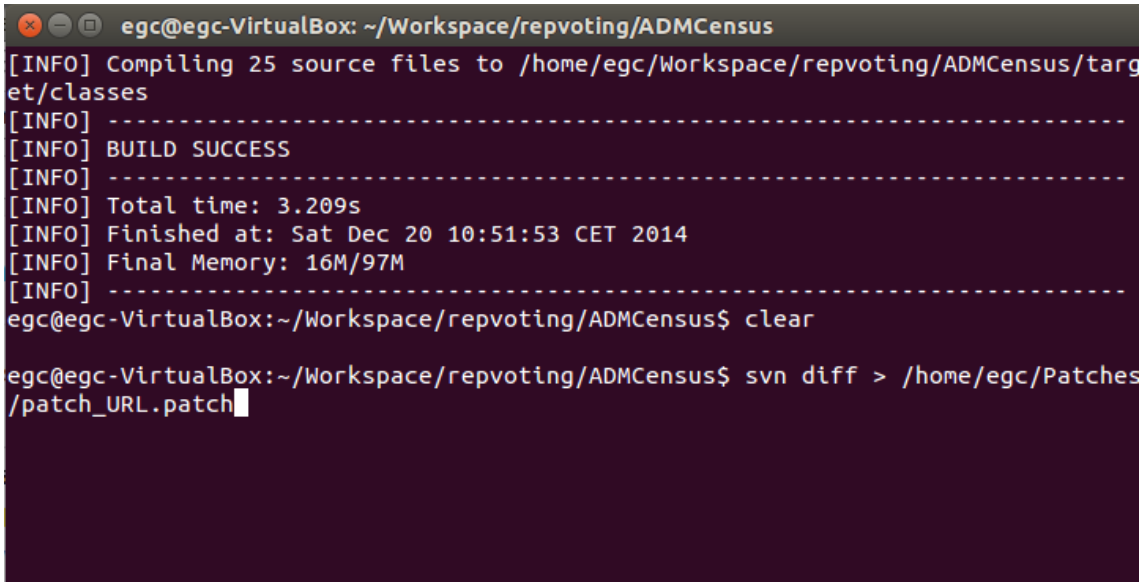


Figura 8: Situarse en la rama de cambios

En el ejemplo de la imagen nos hemos situado en el WorkSpace "WorkSpaceEGC" y el directorio llamado "Cambios 10-11-2014" se trata de una rama del repositorio.

Una vez situados en dicha rama, añadimos el parche (modificaciones en el código fuente sin haber realizado todavía *commit*) y ejecutamos la siguiente línea de comando:

svn diff *> rutaDondeGuardarFichero/nombrePatch.****patch***

Tras el símbolo > se añade la ruta, en la cual se generará el fichero *patch*, y el nombre indicado para dicho fichero. En la siguiente imagen se puede ver la modificación realizada, concretamente se ha cambiado la URL:

http://localhost:8080/ADMCensus/census/prueba.do

por:

http://http://localhost/auth/index.php.

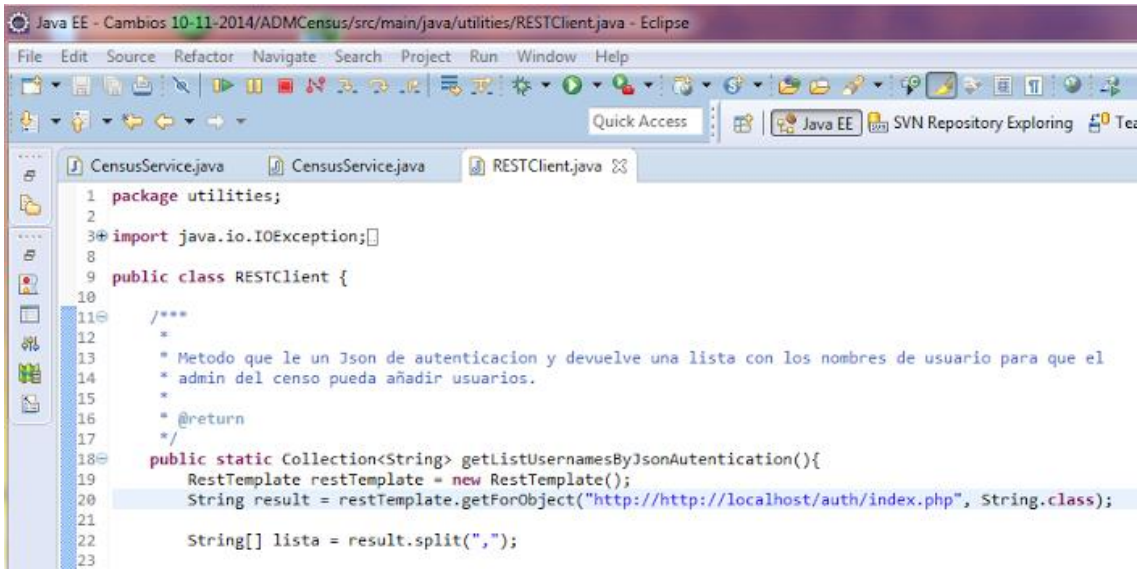


Figura 9: Cambio en el código

En esta otra imagen se muestra el formato con el cual se ha generado el patch en el fichero indicado en el comando ejecutado.

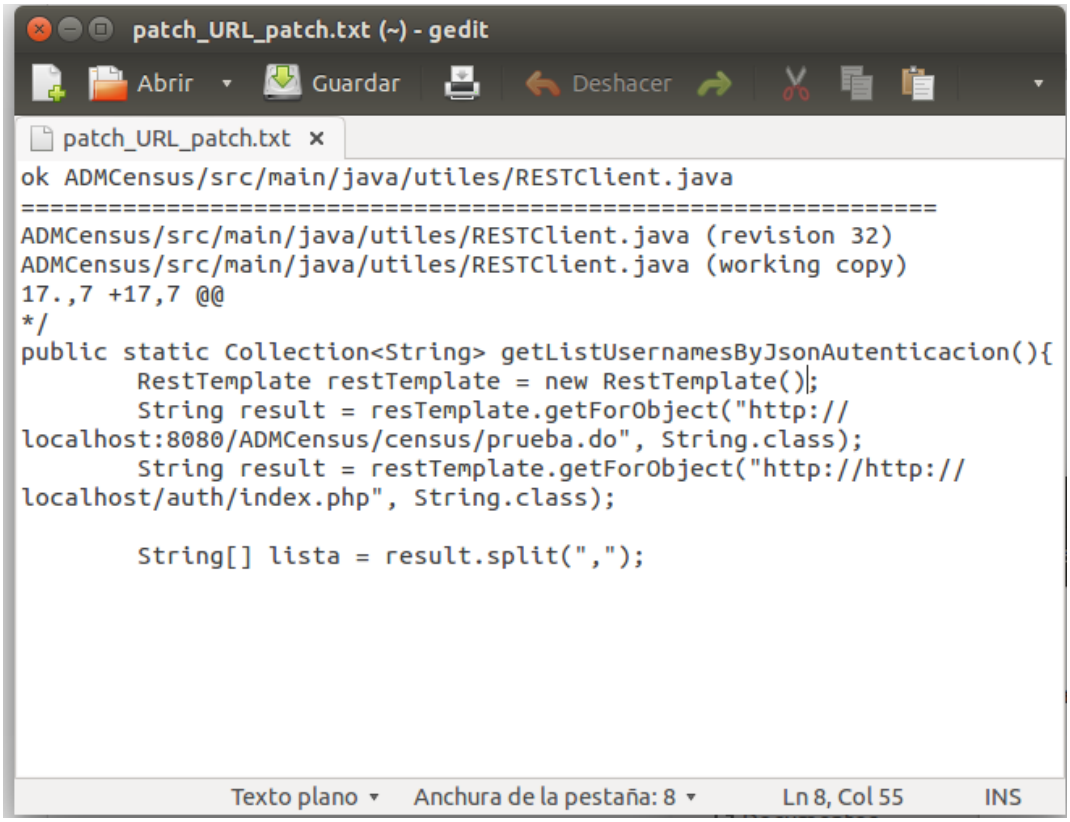


Figura 10: Fichero patch

Para aplicar un *patch* recibido hay que situarse en el directorio en el que se encuentre el proyecto, o como en el caso anterior, en la rama donde se desee aplicar el *patch*. Una vez situados ejecutar el siguiente comando en el terminal:

patch -p0 -i *rutaDondeGuardarFichero/nombrePatch.****patch***

5.3. ROLES

A pesar de la existencia de sólo dos desarrolladores del subsistema, los roles principales para gestionar el código serían los siguientes:

- **Administrador**: Es la persona encargada de aceptar los cambios que han producido conflictos. Tras hablar con la persona que ha realizado el cambio en la funcionalidad y como consecuencia generar dicho conflicto, será el administrador quién realice la subida final al repositorio. También es el encargado de realizar los *merge* que se requieran. Además desarrolla y realiza subidas como un desarrollador normal.
- **Desarrollador**: Persona/s cuya función es realizar el código asignado y subirlo a la rama de trabajo correspondiente. En caso de haber conflicto se pondrá en contacto con el administrador para dar parte de ello y resolverlo conjuntamente.

5.4. POLÍTICA DE NOMBRE Y ESTILOS UTILIZADOS EN EL CÓDIGO FUENTE

- Realizando commit: en la medida de lo posible, se ha intentado seguir el siguiente patrón para que el desarrollador que haga *update*, pueda identificar rápidamente los cambios que se está actualizando.
  - Diferenciar commit de arreglar fallos, y commit de nueva funcionalidad (interna o externa)
  - Tipo de mensaje en el commit: p.e: NombreRama - FAIL - LUGAR (Master - Fallo - Método crear censo); p.e: Master - FUNC - LUGAR (Funcionalidad - API Puede borrar)
- Estilo:
  - Las variables se definen con el nombre del objeto a devolver y empezando con minúscula.
  - El nombre de la clase debe empezar siempre con mayúscula.
  - Los servicios deben llamarse con la siguiente estructura: NombreEntidadService.
  - Los repositorios deben llamarse con la siguiente estructura: NombreEntidadRepository.
  - Los métodos se llamarán como la función que realicen en inglés, empezando con minúscula. Si el nombre es más de una palabra irán todas juntas siendo la primera en mayúsculas (*lowerCamelCase*). Ej:

    public boolean ***updateUser****(int censusId, String username){...}*

  - El nombre del proyecto y el de la base de datos deben ser el mismo.
  - En las clases de dominio primero se definirán los atributos, a continuación los getter y setter de estos. En el caso de haber relaciones con otras entidades, se definirán dichas relaciones después de los getter y setter de los atributos de la propia clase.
  - El código debe estar bien tabulado, como se muestra en la siguiente imagen:

```java
/***
 * Metodo utilizado por cabina para actualizar el estado de voto de un usuario
 *
 * @param censusId
 * @param token
 */
public boolean updateUser(int censusId, String username) {
    boolean res = false;
    Assert.isTrue(!username.equals(""));
    Census c = findOne(censusId);
    HashMap<String, Boolean> vpo = c.getVoto_por_usuario();

    if (vpo.containsKey(username) && !vpo.get(username)){

        vpo.remove(username);
        vpo.put(username, true);
        res = true;
    }

    c.setVoto_por_usuario(vpo);
    save(c);

    return res;
}
```

Figura 11: Ejemplo de código fuente

- Estilo de las vistas: Para mostrar contenido en una página jsp, siempre usamos el comando jstl:out de la librería jstl. Otra buena práctica es, para no repetir código innecesariamente, crear tags (etiquetas) para que las llamadas sean más sencillas y las vistas estén menos sobrecargadas.
- Se debe añadir un comentario sobre los métodos donde se explique una breve descripción de la funcionalidad que realizan.

5.5. EJERCICIO

"Realizar un Merge de una rama con el trunk, asignando posteriormente un tag."

Resolución

Para poder dar solución al ejercicio propuesto, debemos tener en nuestro repositorio al menos una rama más aparte del *trunk* para que esto pueda ser posible.

Comentar, que esta solución se ha realizado haciendo uso de SVN, con una estructuración de dos ramas, la principal o *trunk* y la rama "Rama1" que será usada para realizar la acción de *merge*.

En primer lugar, comencemos por comprobar que lo desarrollado en la rama "Rama1" es totalmente correcto y por tanto no contendrá errores, ya que si ese es el caso daría ocasión a un conflicto, cosa que hay que intentar evitar para llevar a cabo unas buenas prácticas de la gestión del código fuente.

Una vez comprobado esto, procedemos a actualizar tanto el *trunk* como la rama en cuestión, además de asegurarnos de que nadie trabaja sobre dicha rama.

Tal y como se ha comentado antes, el proyecto como mínimo debe estar de la siguiente manera:

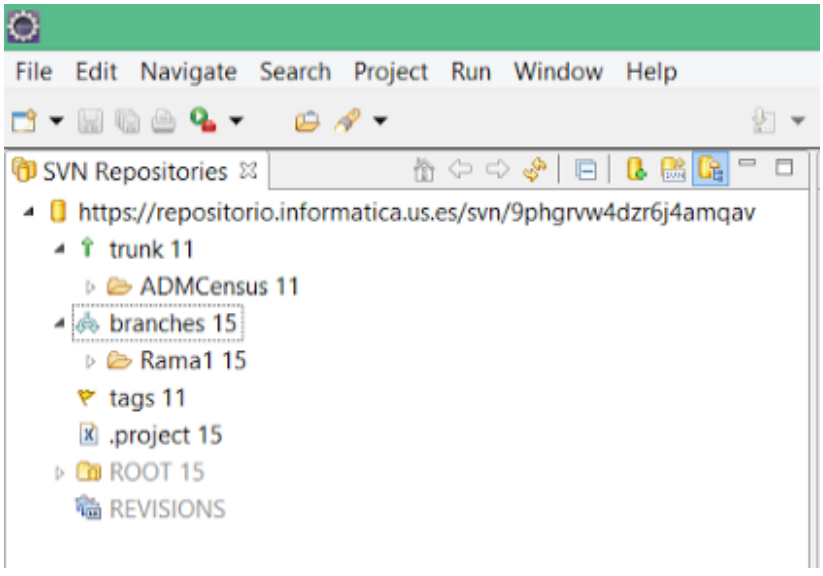


Figura 12: Estructura del repositorio

Ya teniendo la última versión de ambas ramas, procedemos a seleccionar la rama *trunk* para posteriormente realizar el *merge* en cuestión, mediante la opción de *merge* (*team/merge*).

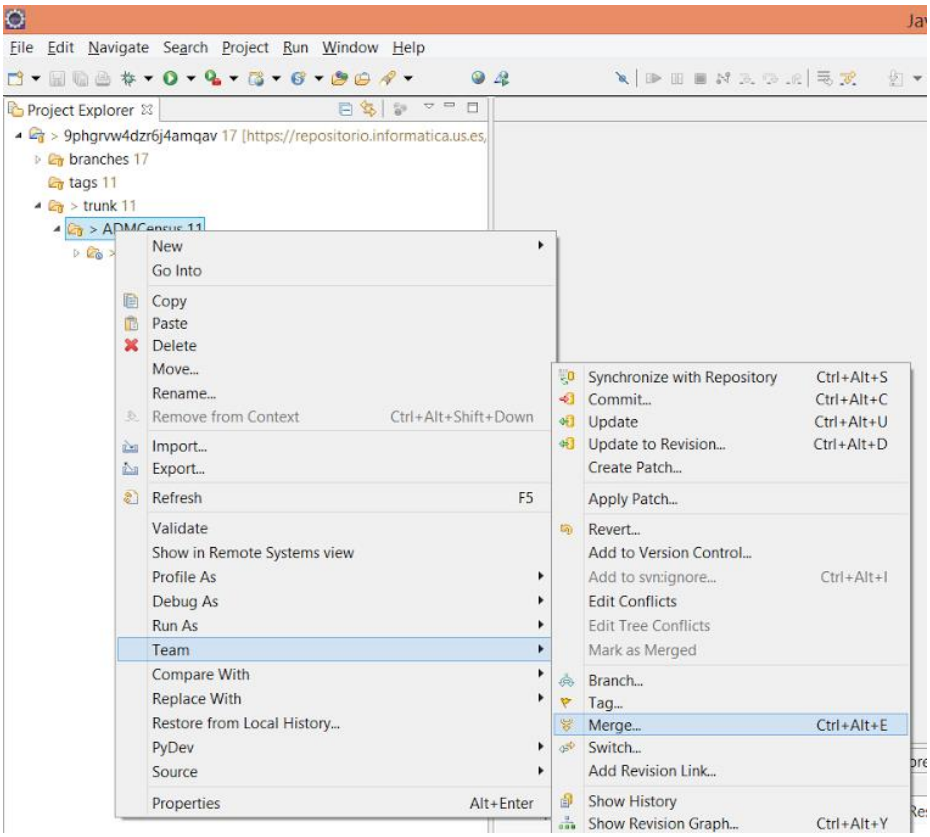


Figura 13: Menú team

A continuación, se muestra una ventana semejante a la que podemos ver en la siguiente imagen, la cual consiste en las distintitas configuraciones para ajustar valores tales como pueden ser en éste caso las ramas que van a verse involucradas.

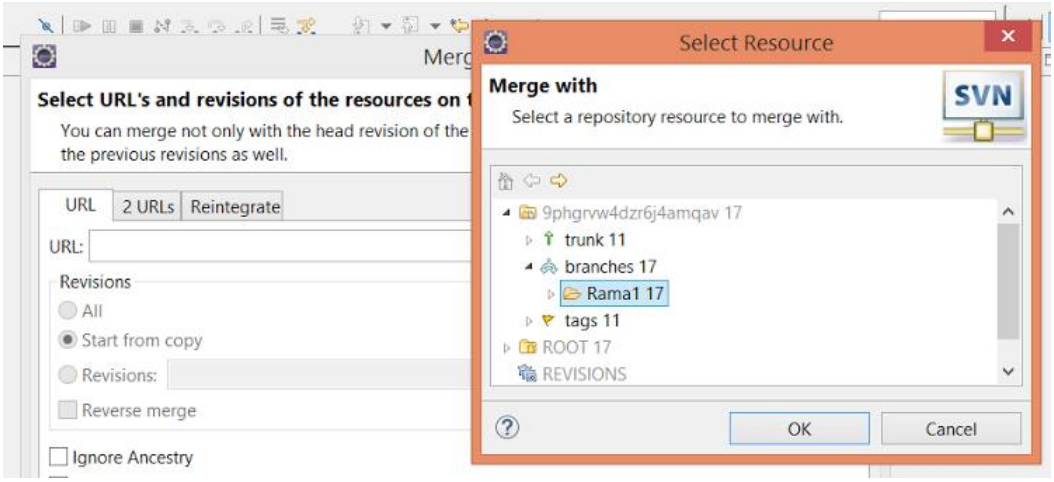


Figura 14: Rama con la que realizar el merge

Llegados a este punto, sólo nos queda realizar dicha acción, tras la cual comprobaremos, en el caso de no haber realizado una buena gestión, si surgen conflictos.

Esto sería el final del ejercicio propuesto ya que al hacer *Merge* se realiza la acción de *commit* automáticamente.

Ahora, procedemos a la creación de un *tag*, que permite el cierre de una versión estable de nuestro proyecto. Esto nos va a ayudar a poder volver a una versión anterior a la actual.

Primero procedemos a crear un *tag*, seleccionando la opción de crear Tag tal y como podemos comprobar en la imagen mostrada a continuación (Team → Tag).

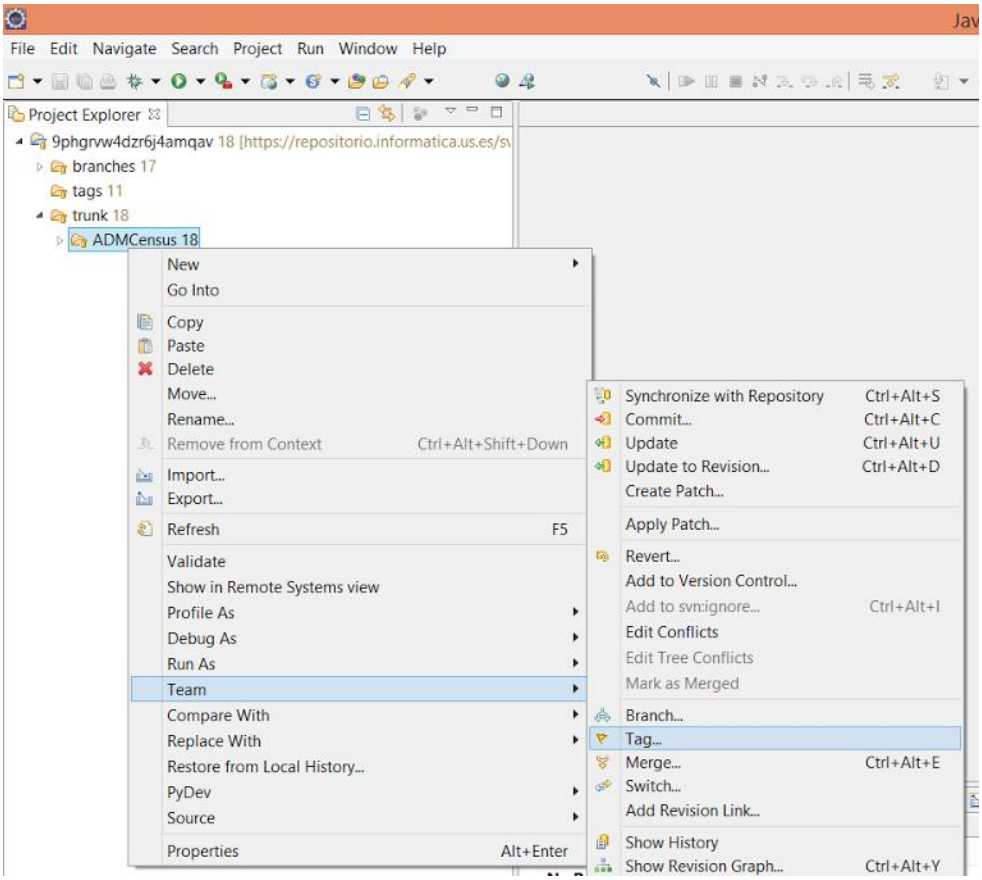


Figura 15: Menú team

Llegados a este punto, visualizamos una ventana semejante a la mostrada a continuación, en la que debemos elegir el *tag* en el que queremos almacenar dicha versión de nuestro proyecto.

Además de esto, conviene dejar un comentario en el que se describa lo desarrollado hasta el momento, de modo que si en un futuro buscamos algo en concreto en nuestro software esto nos lo facilite.

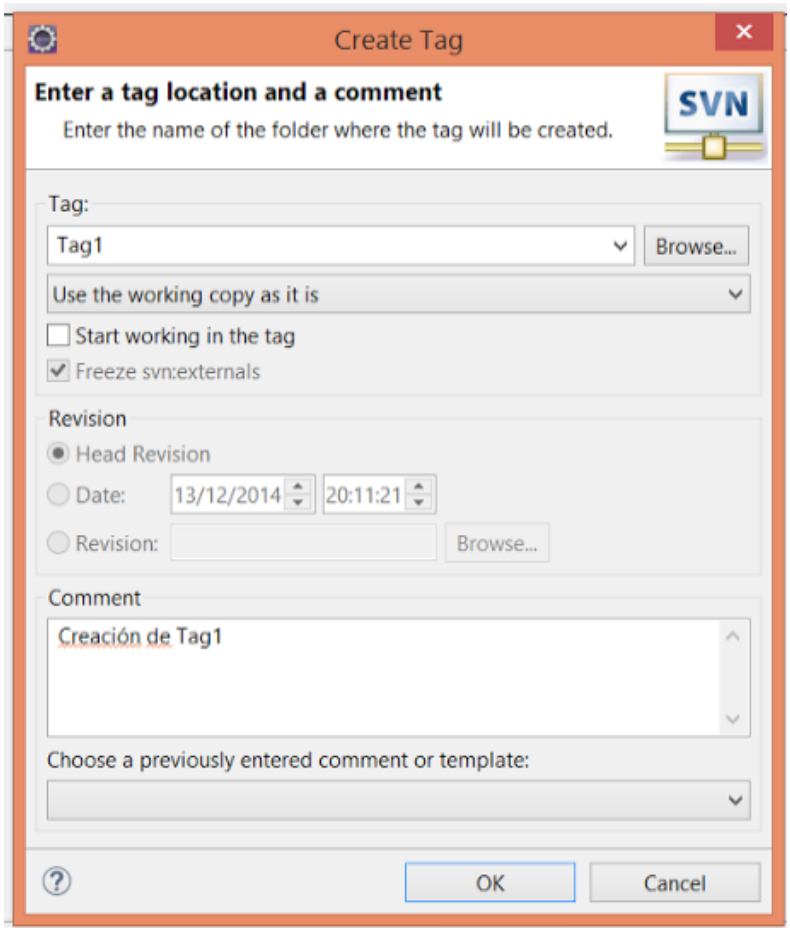


Figura 16: Creación de Tag

Finalmente, una vez realizado lo descrito anteriormente, la realización tanto del *merge* como del *tag*, ya habríamos concluido el ejercicio propuesto de manera satisfactoria.

6. GESTIÓN DE LA CONSTRUCCIÓN E INTEGRACIÓN CONTINUA

6.1. GESTIÓN DE LA CONSTRUCCIÓN

La gestión de la construcción consiste en convertir/construir software a partir del código fuente. La relación entre el código fuente y la construcción es muy estrecha, ya que el código se encuentra alojado en el repositorio de ProjETSII, como se ha explicado anteriormente, y es muy importante una buena organización de ramas y el establecimiento de buenas prácticas para resolver conflictos, realizar *commits*, etc. ya que dicha organización facilitará la construcción del producto final, o al menos en nuestro caso, del subsistema Creación/Administración de censos.

En general Agora@US es un proyecto complejo de realizar debido a la existencia de varios subsistemas y una fuerte relación entre la mayoría de ellos. Los cuales a la hora de construirlos e integrarlos, si no se han seguido unas políticas de organización adecuadas, pueden resultar trabajoso llegando al punto de no conseguir el funcionamiento total de la aplicación y por tanto el fracaso del proyecto.

Para la construcción de nuestro subsistema se ha dividido en dos partes: las funcionalidades internas y las externas. Las primeras consisten en los métodos CRUD: creación, modificación, eliminación y listado de censos. Para las funcionalidades externas se ha generado una API con todos los métodos necesarios para obtener una comunicación exitosa con los subsistemas con los cuales nos comunicamos, o bien requieran de nuestro servicio.

Pasos a seguir para realizar la construcción del proyecto Creación/Administración de censos.

6.1.1. CONSTRUCCIÓN DEL PROYECTO

Para la construcción del subsistema Creación y Administración de censos se ha basado en la tecnología JEE (Open Source).

Se ha desarrollado sobre un sistema operativo Ubuntu 14.04 de 64bits con una versión de Java 7, ya que al estudiar la compatibilidad de los frameworks utilizados, no se encontró ninguna incompatibilidad, y con el IDE de desarrollo Eclipse Luna (4.4.1).

El equipo de desarrollo vio conveniente seguir el modelo de programación MVC (Modelo-Vista-Controlador) para el subsistema y para ello se apoyó en el framework Spring-MVC versión 3.2.

Como Sistema de Gestión de Base de Datos (SGBD) se utilizó MySQL 5.5 y la conexión con este lo hacía con Hibernate 4.

Para hacer el ORM entre Java y el SGBD se aprovechó las características de JPA 2.1 y su integración con Hibernate.

Además, aprovechando la versión de Eclipse Luna ya tendríamos todos los plugins necesarios instalados, como por ejemplo el de SVN.

Como los proyectos software actuales tiene una gran dependencia de terceros (librerías, APIs, etc.), se utiliza Maven 3 para obtener las librerías necesarias para la construcción a través de los archivos Pom.xml de los proyectos, y así poder importar las dependencias de una forma más sencilla.

Todas estas herramientas se describen con más detalle en el apartado Mapa de Herramientas.

Las dependencias del proyecto se pueden clasificar en dos grandes grupos:

- **Internas:** son librerías que el subsistema necesita pero no son competencia de ningún otro subsistema, sino de terceros. Por ejemplo, Administración y Creación de censos, necesita dar datos en formato Json (previamente acordado con los demás subsistemas de Agora@us) y para ello utilizó dependencia de *Codehaus*, en concreto de una librería que proporciona llamada *Jackson*, la cual provee de las llamadas para convertir un objeto Java a formato Json.

- **Externas:** son las dependencias de otros subsistemas para poder cumplir una funcionalidad concreta. Dentro de las externas podemos distinguir otros dos grupos:
  - Consumidas: son funcionalidades, que por sus características, desarrolladas por otros subsistemas, pero que nuestro subsistema las ha necesitado para alguna determinada tarea (por ejemplo el registro de usuarios).
  - Producidas: son las funcionalidades que brindamos a otros grupos que necesitan de datos o acciones que se han desarrollado internamente (por ejemplo, el registro de usuarios que han votado o no)

6.1.2. DESCARGA DEL CÓDIGO FUENTE EN ENTORNO

Como prerrequisitos de este paso, se debe tener un equipo con todas las tecnologías nombradas anteriormente en el punto construcción del proyecto, para así, poder descargar el proyecto del repositorio y comenzar a trabajar sin más problema. Estas tecnologías necesarias se proporcionan en la máquina virtual entregada junto con esta memoria.

A continuación, vamos a describir los pasos para poder descargar en nuestro IDE eclipse ADMCensus:

- Con eclipse arrancado, en un Workspace cualquiera, pulsamos Window -> Open Perspective -> Svn Repository Exploring. Aquí está listo para decirle que lea lo que hay en el repositorio.
- Ahora le damos a botón derecho new -> Repository Location… he introducimos nuestro usuario y contraseña que nos han proporcionado y la ruta del repositorio, en este caso es:

https://repositorio.informatica.us.es/svn/6cn2fmqpc7fa429m2e9

- Acto seguido, tendremos la estructura del repositorio en nuestro equipo:

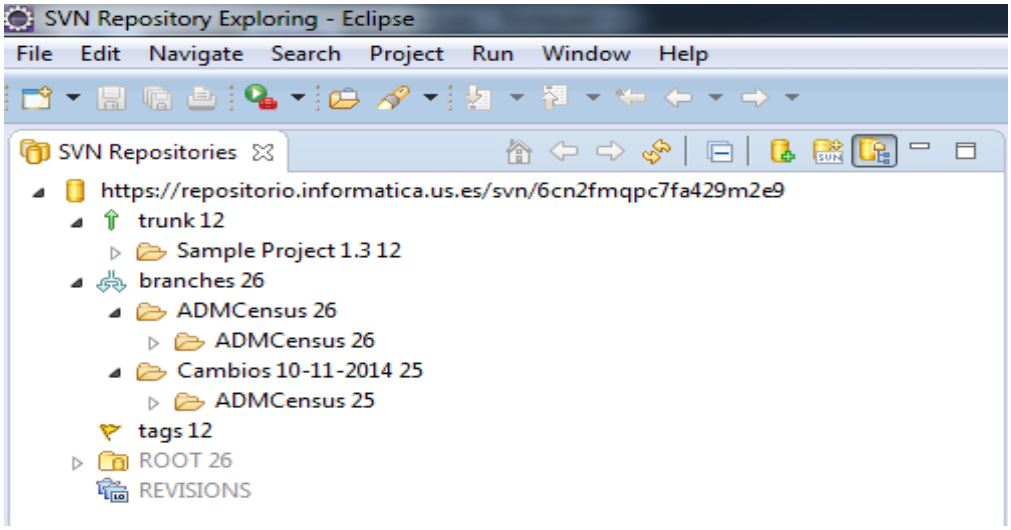


Figura 17: Estructura del repositorio

Aquí, podremos descargar la versión de rama que queramos, o bien si queremos una versión final para realizar pruebas, podremos descargar el trunk o el tag.

Importarlo mediante botón derecho sobre lo que queramos descargar (*trunk*, *branch* o *tags*) y *Check out*. Con esto, ya tendremos el proyecto en nuestro Workspace a falta de configurar la base de datos.

- La base de datos del proyecto se llama ADMCensus y para que el proyecto funcione, debemos crearla e insertar dos usuarios necesarios, a continuación, ponemos el script completo:

```
    drop database if exists `ADMCensus`;

    create database `ADMCensus`;

    create user 'acme-user'@'%' identified by password
'*4F10007AADA9EE3DBB2CC36575DFC6F4FDE27577';

    create user 'acme-manager'@'%' identified by password
'*FDB8CD304EB2317D10C95D797A4BD7492560F55F';

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

- Por último, ejecutamos dos clases Java para poder tener datos funcionales. Las clases son las siguientes:
  - PopulateDatabase.java -> Crea la estructura de la base de datos. La debemos ejecutar como una clase java normal.
  - CreateCensus.java -> Creará usuarios y censos de pruebas -> La debemos ejecutar como un test JUnit.
- Ya tenemos lista la aplicación para agregar el proyecto al servidor Tomcat con botón derecho sobre Tomcat -> Add and Remove -> agregamos el proyecto -> Finish.

6.1.3. CONSTRUCCÍON Y EJECUCIÓN

Una vez que hayamos acabado de realizar el trabajo propuesto, estamos listos para empaquetar la aplicación para poder ejecutarla en cualquier entorno con Tomcat.

El empaquetado consiste en crear un .war (Web Application Archive) y aquí Eclipse, nos da una gran facilidad, puesto que solo tenemos que pulsar con el botón derecho sobre el proyecto -> Export -> WAR file.

Elegimos el nombre del archivo War y el destino en el que queremos guardarlo. A continuación vamos a explicar con detenimiento qué significan las opciones que nos da Eclipse a la hora de generar el archivo War.

- Optimize for a specific server runtime: si sabemos con seguridad que el proyecto va a ser desplegado sobre un servidor concreto, lo podemos indicar, pero como en este caso, es un subsistema y no sabemos con seguridad la máquina en la que se desplegaran.
- Export source files: si se desea que su código fuente pueda ser leído desde el WAR, se pulsará, en caso contrario, no.
- Overwrite existing file: si hay un archivo con el mismo nombre, se sustituirá.

Como la base de datos tiene una estructura determinada, tendremos que crear un archivo .sql que albergue la estructura de ésta. Para ello copiaremos y ejecutaremos el script siguiente en PHPMyAdmin y ya tendremos la base de datos creada.

```
start transaction;
drop database if exists `ADMCensus`;
create database `ADMCensus`;
use `ADMCensus`;

create user 'acme-user'@'%' identified by password
'*4F10007AADA9EE3DBB2CC36575DFC6F4FDE27577';
create user 'acme-manager'@'%' identified by password
'*FDB8CD304EB2317D10C95D797A4BD7492560F55F';

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

-- Base de datos: `ADMCensus`
-- Estructura de tabla para la tabla `Census`

CREATE TABLE IF NOT EXISTS `Census` (
  `id` int(11) NOT NULL,
  `version` int(11) NOT NULL,
  `fechaFinVotacion` datetime DEFAULT NULL,
  `fechaInicioVotacion` datetime DEFAULT NULL,
  `idVotacion` int(11) DEFAULT NULL,
  `tituloVotacion` varchar(255) DEFAULT NULL,
  `username` varchar(255) DEFAULT NULL,
  `valor` tinyblob
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Estructura de tabla para la tabla `hibernate_sequences`
```

```
CREATE TABLE IF NOT EXISTS `hibernate_sequences` (
  `sequence_name` varchar(255) DEFAULT NULL,
  `sequence_next_hi_value` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Volcado de datos para la tabla `hibernate_sequences`

INSERT INTO `hibernate_sequences` (`sequence_name`, `sequence_next_hi_value`) VALUES
('DomainEntity', 5);

-- Estructura de tabla para la tabla `UserAccount`

CREATE TABLE IF NOT EXISTS `UserAccount` (
  `id` int(11) NOT NULL,
  `version` int(11) NOT NULL,
  `password` varchar(255) DEFAULT NULL,
  `username` varchar(255) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Estructura de tabla para la tabla `UserAccount_authorities`
--

CREATE TABLE IF NOT EXISTS `UserAccount_authorities` (
  `UserAccount_id` int(11) NOT NULL,
  `authority` varchar(255) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Índices para tablas volcadas
-- Indices de la tabla `Census`

ALTER TABLE `Census`
 ADD PRIMARY KEY (`id`), ADD UNIQUE KEY `UK_qx9ataj93wny5sbrfxx6maweq` (`idVotacion`);
```

```
-- Indices de la tabla `UserAccount`

ALTER TABLE `UserAccount`
 ADD PRIMARY KEY (`id`), ADD UNIQUE KEY
`UK_csivo9yqa08nrbkog71ycilh5` (`username`);

-- Indices de la tabla `UserAccount_authorities`
ALTER TABLE `UserAccount_authorities`
 ADD KEY `FK_b63ua47r0u1m7ccc9lte2ui4r`
(`UserAccount_id`);

-- Restricciones para tablas volcadas
-- Filtros para la tabla `UserAccount_authorities`

ALTER TABLE `UserAccount_authorities`
ADD CONSTRAINT `FK_b63ua47r0u1m7ccc9lte2ui4r` FOREIGN KEY
(`UserAccount_id`) REFERENCES `UserAccount` (`id`);

/*!40101 SET
CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET
CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET
COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;

commit;
```

Con esto, ya tenemos nuestra estructura de base de datos lista para que funcione el subsistema.

Con este archivo WAR y el .sql podremos ejecutar la aplicación en cualquier entorno, siempre y cuando se tengan los requisitos tecnológicos necesarios.

6.2. INTRODUCCIÓN A LA INTEGRACIÓN

¿Qué es la integración?

La integración es un término que se utiliza en el ámbito del software para referirse como se van a combinar los componentes individuales del software en un mismo sistema. El resultado de esta combinación es una "versión estable" del producto, al cual se le realizarán las pruebas de validación con el objetivo de obtener el producto final. Dicha labor requiere un gran esfuerzo, debido a que cada componente se integrará tras la aprobación de los componentes mediante las pruebas individuales. El objetivo de la integración es obtener un componente único, producto, el cual se valida mediante comprobaciones de funcionalidad, rendimiento y fiabilidad.

Existen dos estrategias distintas a la hora de realizar la integración: por fase e incremental.

La integración por fase

La integración por fase consiste en combinar todos los componentes a la vez tras finalizar la fase de diseño, a este tipo de estrategia también se le conoce por el nombre de "integración Big Bang". Este tipo de estrategia tiene la desventaja de que los errores encontrados son difíciles de aislar, por tanto su solución es más laboriosa. Podría resultar útil en proyectos pequeños, ya que no se necesita demasiado tiempo a la hora de formar el producto final.

La integración incremental

La integración incremental consiste en combinar los componentes progresivamente, es decir, se integra un componente y tras aprobar dicho ensamblaje se integra otro componente hasta obtener el producto final tras la integración de todos los componentes. La ventaja de esta estrategia reside en la capacidad de detectar, aislar y corregir los errores. La desventaja, si existe, reside en el tiempo que se puede llegar a tardar a la hora de integrar por completo los componentes de un producto, debido a que se lleva un plan sistemático.

Tras analizar las ventajas y desventajas de las distintas estrategias, se optó por realizar integración incremental a pesar de ser más costoso en tiempo, pero más fiable y sistemático en cuanto a la detección y solución de posibles errores.

La integración incremental a su vez hay tres formas distintas de realizarla: ascendente, descendente y sandwich. En los siguientes apartados explicamos qué método hemos seguido para la integración interna del subsistema Creación/Administración de censos, y una propuesta de integración con el resto de subsistemas.

6.2.1. INTEGRACIÓN INTERNA

Forma de trabajar

Debido a que ni el volumen ni la complejidad del subsistema es muy grande, solo tenemos dos desarrolladores:

Amador Salmerón, Antonio Juan.

Fernández de Bobadilla Brioso, Guiomar.

Para desarrollar código internamente se creó una estructura de *branches*, *trunk* y *tags* como se describe en la siguiente imagen.

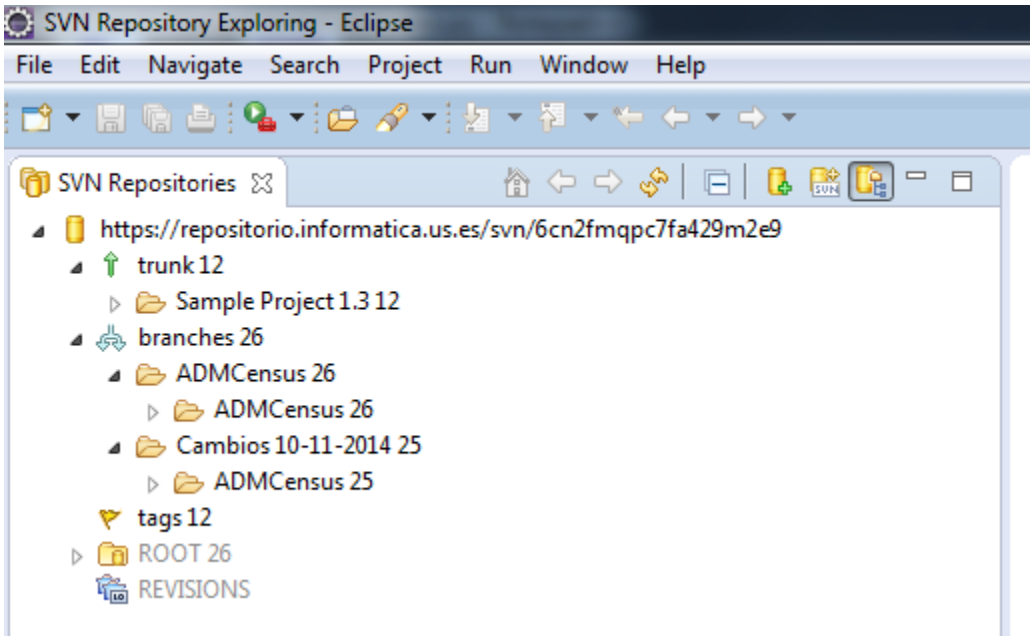


Figura 18: Estructura del repositorio

En el *trunk* se puso la plantilla inicial con las tecnologías ya integradas y se creó una rama donde los desarrolladores comenzaron el desarrollo. (Rama ADMCensus)

En clase se propusieron cambios en la funcionalidad de nuestro subsistema, por lo que el equipo decidió abandonar dicha rama y crear la nueva rama que se llama Cambios.

Cada desarrollador se ocupó de una función distinguiendo dos:

- Métodos internos: Antonio Juan Amador Salmerón.
- API externa: Guiomar Fernández de Bobadilla Brioso

En los casos en los que se ha ido añadiendo nuevas comunicaciones con otros subsistemas, se ha seguido realizando la integración ascendente, de manera similar a un ciclo.

Tanto para los métodos internos (como se ha explicado anteriormente) como para la realización de la API para comunicarnos con los demás subsistemas, se ha seguido una integración incremental ascendente, es decir, comenzando con el modelo de dominio del subsistema asignado, creación de la base de datos, repositorios para acceder a los datos necesarios, los servicios con la funcionalidad interna requerida, las vistas y los controladores.

Los desarrolladores trabajan en local y los *commits* solo se hacen cuando el desarrollador vea que la funcionalidad desarrollada esta lista para integrarlo en la rama incluyendo su respectivo comentario.

Herramientas

La herramienta utilizada para la integración interna ha sido Subversion (SVN) y el plugin de SVN para el entorno de desarrollo Eclipse. Con dicho plugin la resolución de conflictos ha sido más sencilla y eficaz. Subversión está enlazado con el repositorio alojado en la herramienta ProjETSII donde se encuentra nuestro proyecto. Solo tendremos que ir a la vista de eclipse para enlazar el repositorio, poner la url de este, nuestro usuario y contraseña, y ya podremos acceder al contenido.

Como primera herramienta de integración se ha utilizado **Maven**, la cual nos ha ayudado a verificar que cuando hacemos update del repositorio, los cambios realizados no interfieren en el proyecto y podamos seguir trabajando sobre seguro.

Maven se estructura en ciclos de trabajo que son los siguientes:

- compile: Genera los ficheros .class compilando los fuentes .java
- test: Ejecuta los test automáticos de JUnit existentes, abortando el proceso si alguno de ellos falla.
- package: Genera el fichero .jar con los .class compilados
- install: Copia el fichero .jar a un directorio de nuestro ordenador donde maven deja todos los .jar. De esta forma esos .jar pueden utilizarse en otros proyectos maven en el mismo ordenador.
- deploy: Copia el fichero .jar o .war a un servidor remoto, poniéndolo disponible para cualquier proyecto maven con acceso a ese servidor remoto.

También existen algunas metas o directivas de maven que están fuera del ciclo:

- clean: Elimina todos los .class y .jar generados. Después de este comando se puede comenzar un compilado desde cero.
- assembly: Genera un fichero .zip con todo lo necesario para instalar nuestro programa java. Se debe configurar previamente en un fichero xml qué se debe incluir en ese zip.
- site: Genera un sitio web con la información de nuestro proyecto. Dicha información debe escribirse en el fichero pom.xml y ficheros .apt separados.
- site-deploy: Sube el sitio web al servidor que hayamos configurado.

Nota: para ejecutar estas metas, debemos estar en la raíz del proyecto, es decir, en el mismo directorio donde se encuentra el archivo pom.xml

Cada vez que uno de los desarrolladores realiza *update* para actualizar sus cambios o bien está preparado para hacer un *commit*, este ejecuta la meta "compile" para asegurar que los cambios que sube al repositorio no van a dejar inconsistente el proyecto. Semanalmente, se ejecuta la meta test para verificar con JUnit el correcto funcionamiento de esta.

En estas clases de JUnit encontraremos test unitarios para comprobar que unos resultados, predefinidos por el equipo de desarrollo, son correctos y no ha dejado de funcionar nada.

Antes de esto, es conveniente hacer la meta “mvn clean” por si se han bajado algunos .class o archivos que no nos son necesarios para borrarlos y hacer las pruebas en limpio.

A continuación se muestra en la imagen una verificación del proyecto después de hacer *update* y actualizar varios cambios.

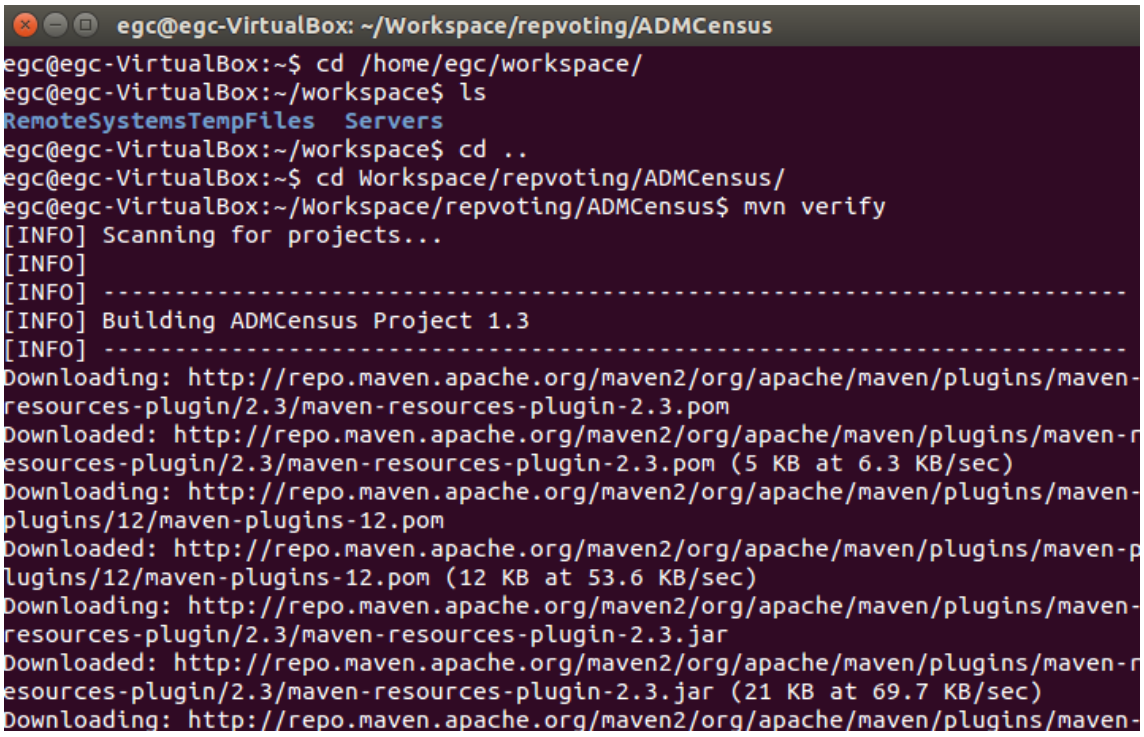


Figura 19: Verificación del proyecto

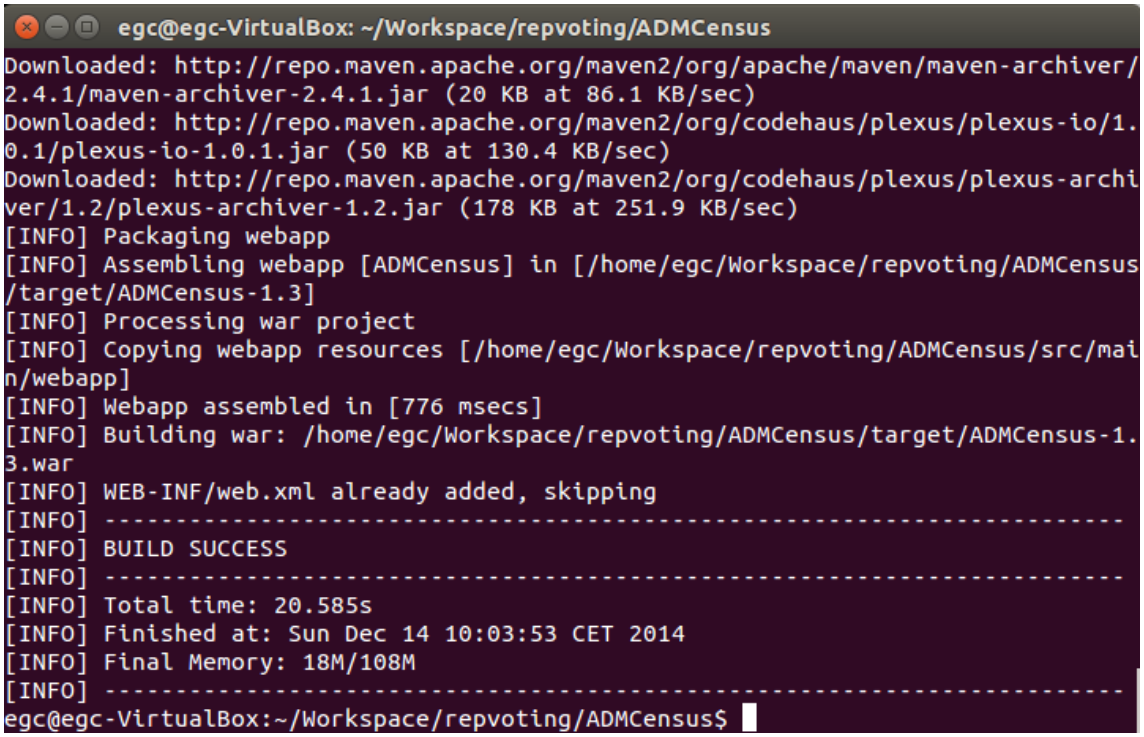


Figura 20: Resultado de la verificación del proyecto

Otro ejemplo de construcción del .war

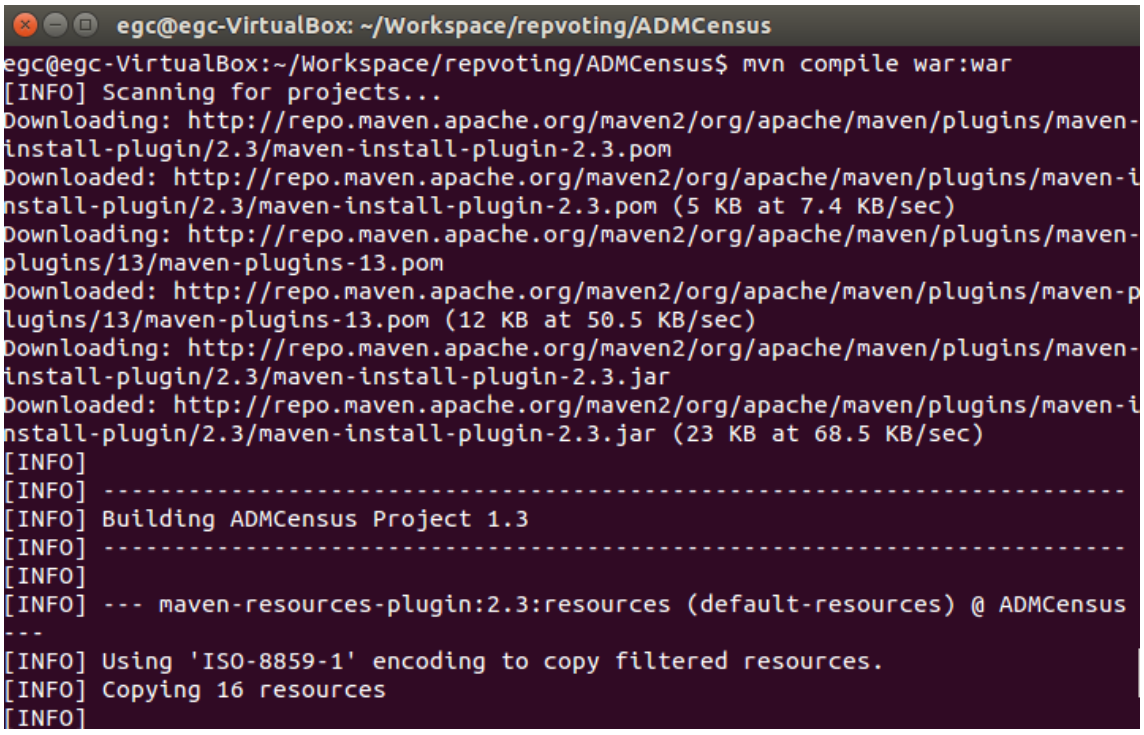


Figura 21: Empaquetado del proyecto

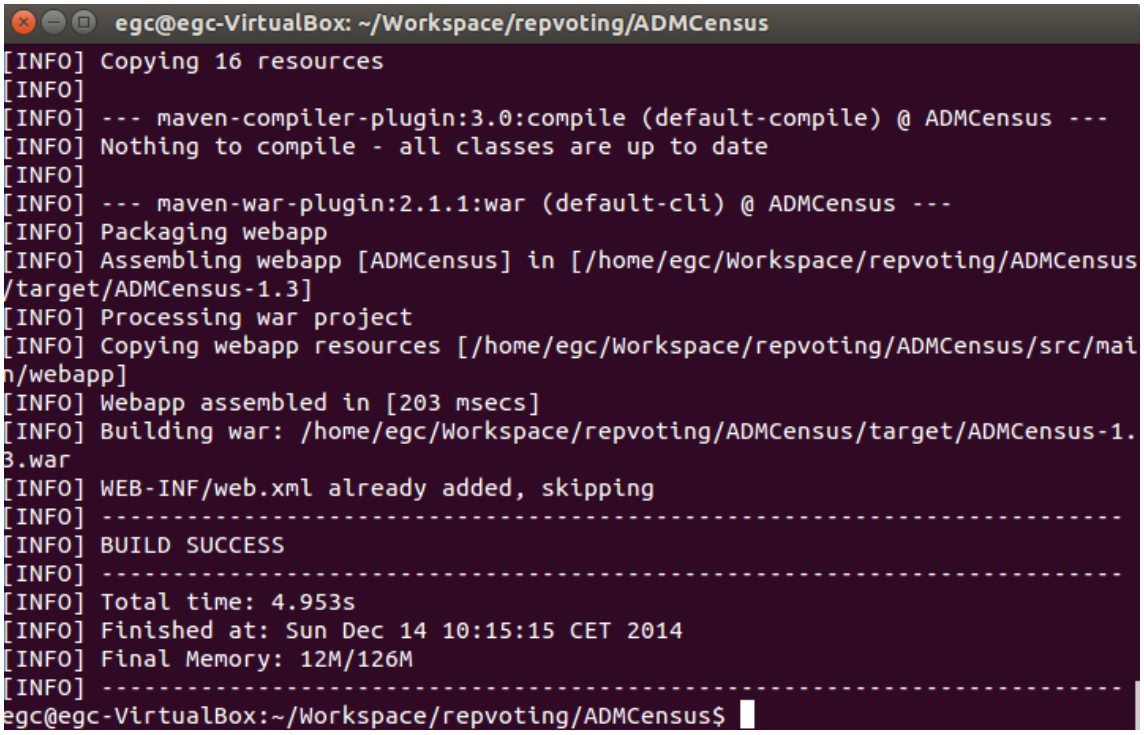


Figura 22: Resultado del empaquetamiento

Como vemos la ejecución se ha realizado con éxito (*BUILD SUCCESS*) y nos ha generado el *war* como vemos en la siguiente imagen:

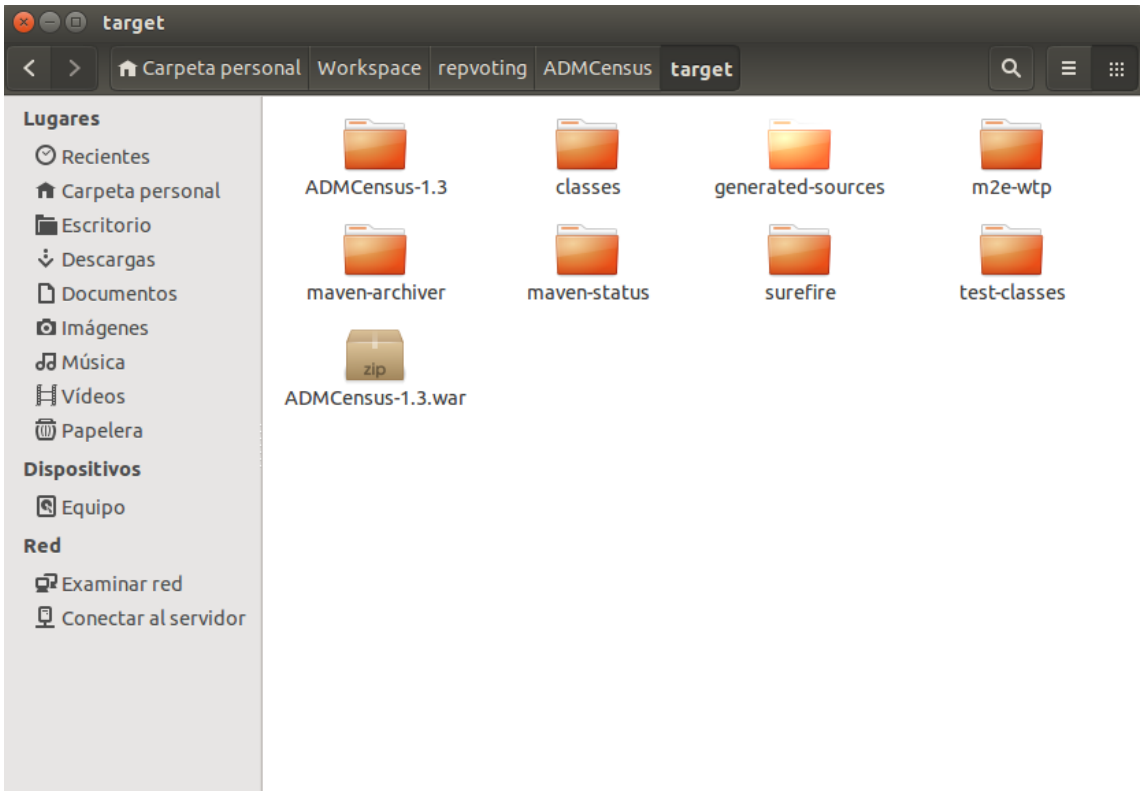


Figura 23: War generado

Para montarlo hacemos uso de la herramienta Jenkins, ya que ésta utiliza Maven para las tareas anteriormente descritas.

Jenkins es una herramienta de integración continua Open Source escrito en Java cuya función principal es la integración continua. La instalación de Jenkins es fácil sobre una distribución Debian, lo único que tuvimos que cambiar en dicha instalación es el puerto, que por defecto utiliza el 8080 y este ya es utilizado por el Tomcat del Eclipse, así que le asignamos el 8090.

Para que en nuestro entorno de desarrollo Jenkins haga sus funciones, debemos tener instalado Java, Maven y Git, ya que este se empezó a utilizar en las últimas fases debido al desconocimiento de la herramienta y porque es el repositorio utilizado para alojar todos los subsistemas.

A continuación vamos a configurar Jenkins para nuestro proyecto en GitHub. Pulsamos en nueva tarea, como un proyecto de estilo libre y le asignamos un nombre, en nuestro caso ADMCensus.

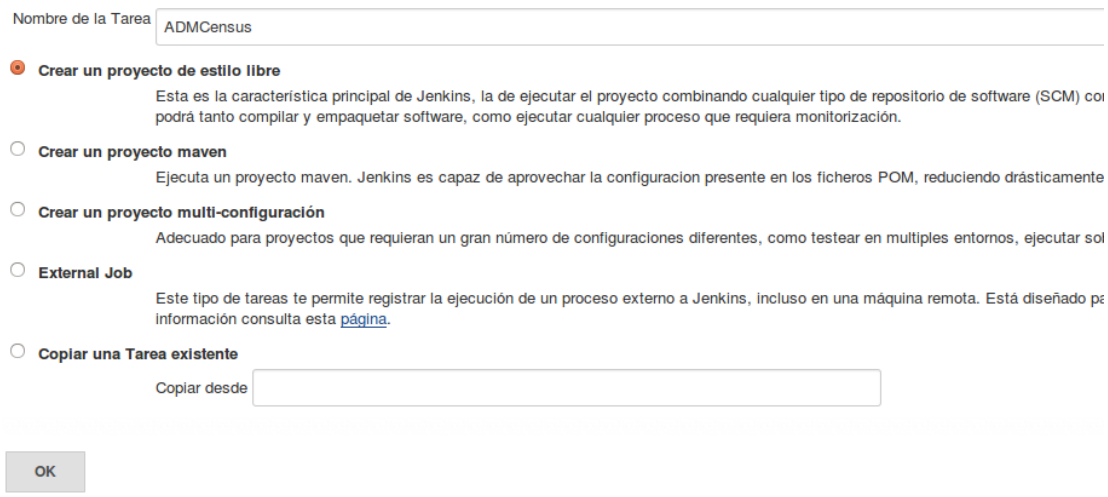


Figura 24: Creación del proyecto en Jenkins

Una vez creado, vamos a configurar el repositorio y nuestra rama:

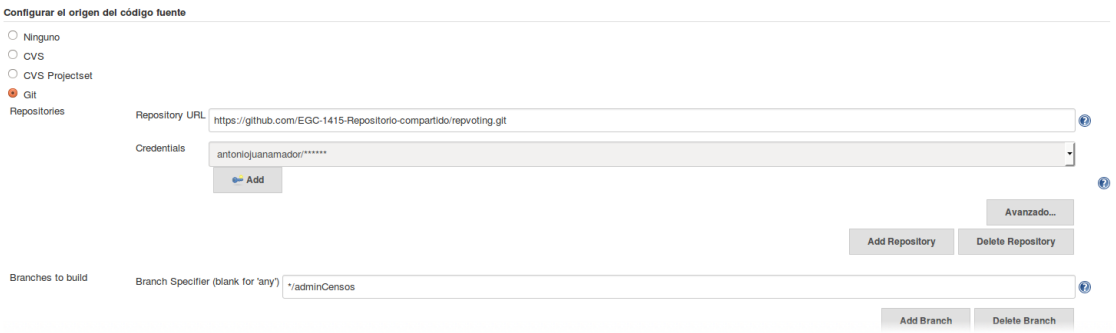


Figura 25: Configuración de repositorio Git en Jenkins

Ahora vamos a hacer uso de Maven. En la siguiente imagen podemos ver cómo le indicamos a Jenkins que descargue el repositorio, se mude a la rama (*git checkout adminCensos)*, se coloque en la raíz del proyecto donde se encuentra el archivo pom.xml (*cd* ADMCensus) y comienza algunas partes del ciclo Maven.

Primero validamos el proyecto y seguidamente podemos decirle que genere el war.

Figura 26: Operaciones del proyecto en Jenkins

Una vez configurado esto pulsamos en “construir ahora” para que haga dichas acciones.

Figura 27: Panel de control del proyecto

Tras haber realizado la construcción, nos dirigimos a la salida de consola y observamos los siguientes resultados. En la siguiente imagen vemos los resultados de las instrucciones realizadas, indicadas anteriormente, por Jenkins.

```
Lanzada por el usuario anonymous
Ejecutando.en el espacio de trabajo /var/lib/jenkins/jobs/ADMCensus/workspace
Cloning the remote Git repository
Cloning repository https://github.com/EGC-1415-Repositorio-compartido/repvoting.git
 > git init /var/lib/jenkins/jobs/ADMCensus/workspace # timeout=10
Fetching upstream changes from https://github.com/EGC-1415-Repositorio-compartido/repvoting.git
 > git --version # timeout=10
using .gitcredentials to set credentials
 > git config --local credential.helper store --file=/tmp/git3302852410850509467.credentials # timeout=10
 > git fetch --tags --progress https://github.com/EGC-1415-Repositorio-compartido/repvoting.git +refs/heads/*:refs/remotes
 > git config --local --remove-section credential # timeout=10
 > git config remote.origin.url https://github.com/EGC-1415-Repositorio-compartido/repvoting.git # timeout=10
 > git config remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git config remote.origin.url https://github.com/EGC-1415-Repositorio-compartido/repvoting.git # timeout=10
Fetching upstream changes from https://github.com/EGC-1415-Repositorio-compartido/repvoting.git
using .gitcredentials to set credentials
 > git config --local credential.helper store --file=/tmp/git3205667567580321016.credentials # timeout=10
 > git fetch --tags --progress https://github.com/EGC-1415-Repositorio-compartido/repvoting.git +refs/heads/*:refs/remotes
 > git config --local --remove-section credential # timeout=10
 > git rev-parse refs/remotes/origin/adminCensos^{commit} # timeout=10
 > git rev-parse refs/remotes/origin/origin/adminCensos^{commit} # timeout=10
Checking out Revision de6d50becd686cc7e49a4b13d81e1bf4927e526a (refs/remotes/origin/adminCensos)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f de6d50becd686cc7e49a4b13d81e1bf4927e526a
First time build. Skipping changelog.
[workspace] $ /bin/sh -xe /tmp/hudson4906062982998643175.sh
+ git checkout adminCensos
Switched to a new branch 'adminCensos'
Branch adminCensos set up to track remote branch adminCensos from origin.
+ cd ADMCensus
+ mvn validate
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building ADMCensus Project 1.3
[INFO] ------------------------------------------------------------------------
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 0.191s
[INFO] Finished at: Sun Dec 14 10:49:30 CET 2014
[INFO] Final Memory: 5M/107M
[INFO] ------------------------------------------------------------------------
```

Figura 28: Informe de Jenkins de validación del proyecto

En esta otra imagen se muestra como Jenkins genera el war automáticamente en la ruta descrita:

```
+ mvn compile war:war
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building ADMCensus Project 1.3
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.3:resources (default-resources) @ ADMCensus ---
[INFO] Using 'ISO-8859-1' encoding to copy filtered resources.
[INFO] Copying 16 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.0:compile (default-compile) @ ADMCensus ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 25 source files to /var/lib/jenkins/jobs/ADMCensus/workspace/ADMCensus/target/classes
[INFO]
[INFO] --- maven-war-plugin:2.1.1:war (default-cli) @ ADMCensus ---
[INFO] Packaging webapp
[INFO] Assembling webapp [ADMCensus] in [/var/lib/jenkins/jobs/ADMCensus/workspace/ADMCensus/target/ADMCensu
[INFO] Processing war project
[INFO] Copying webapp resources [/var/lib/jenkins/jobs/ADMCensus/workspace/ADMCensus/src/main/webapp]
[INFO] Webapp assembled in [594 msecs]
[INFO] Building war: /var/lib/jenkins/jobs/ADMCensus/workspace/ADMCensus/target/ADMCensus-1.3.war
[INFO] WEB-INF/web.xml already added, skipping
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 6.870s
[INFO] Finished at: Sun Dec 14 10:49:39 CET 2014
[INFO] Final Memory: 20M/177M
[INFO] ------------------------------------------------------------------------
Finished: SUCCESS
```

Figura 29: Informe de Jenkins de generación del war

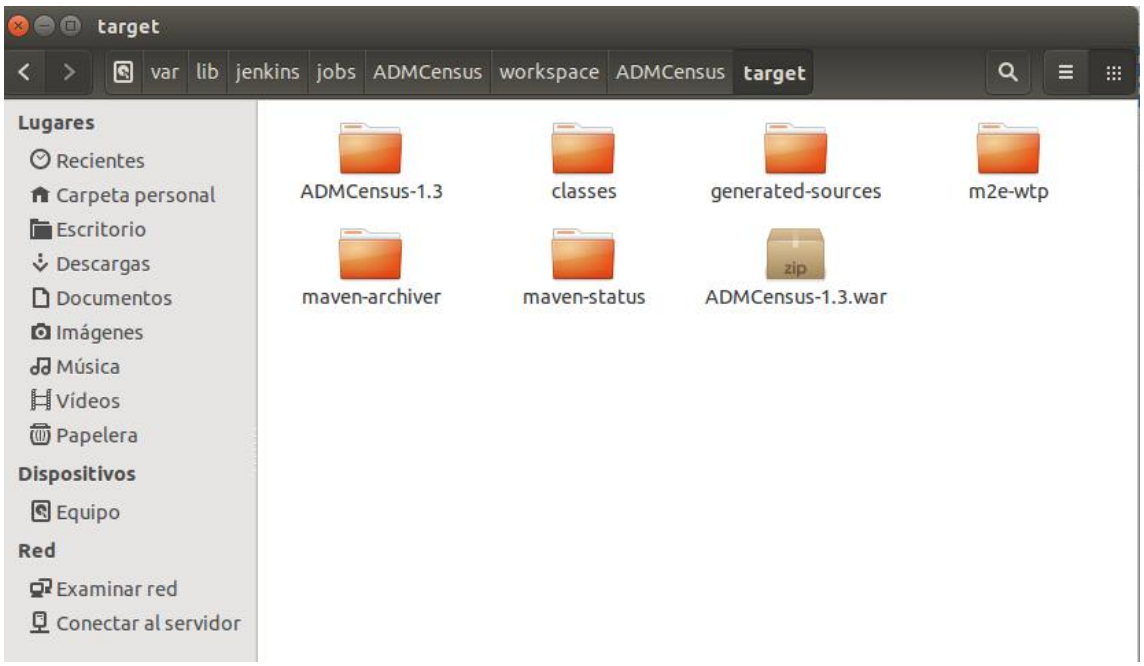


Figura 30: Directorio en el que se ha generado el war

En caso de que falle, podemos automatizar la notificación a los desarrolladores, esto podemos hacerlo en la configuración del proyecto, dentro de otras acciones, añadir notificación. En la siguiente imagen podemos ver la creación de una:

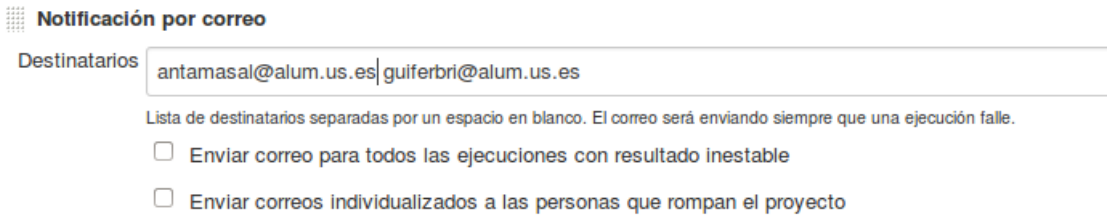


Figura 31: Notificaciones de correo

Aparte de esta acción, también podríamos almacenar resultados, ejecutar otros proyectos, entre otras que Jenkins permite.

Una vez entendido el funcionamiento de Jenkins, vamos a la última parte de su utilización, la automatización del despliegue.

Antes de realizar dicha automatización, debemos tener ejecutado XAMPP para iniciar el servidor de base de datos y poder acceder a las mismas, mediante el comando:

sudo /opt/lampp/lampp start

La contraseña para ejecutar dicha instrucción es: egc

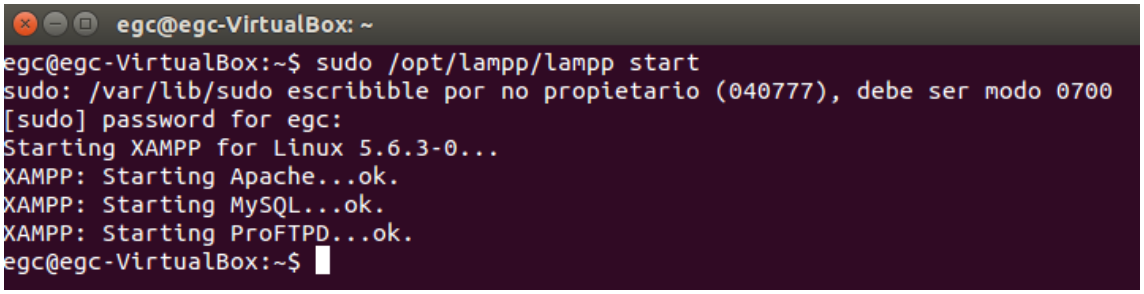


Figura 32: Comando para arrancar XAMPP

A la hora de realizar la automatización, debemos tener en cuenta que tenemos instalado en nuestro sistema Tomcat 7 y que la aplicación se despliega en la ruta http://127.0.0.1:8080/ADMCensus, por tanto debemos indicarle a Jenkins dónde se encuentra la carpeta de Tomcat, en este caso en /var/lib/tomcat7/webapps/, donde almacenará el proyecto a desplegar, quedando de la siguiente manera:

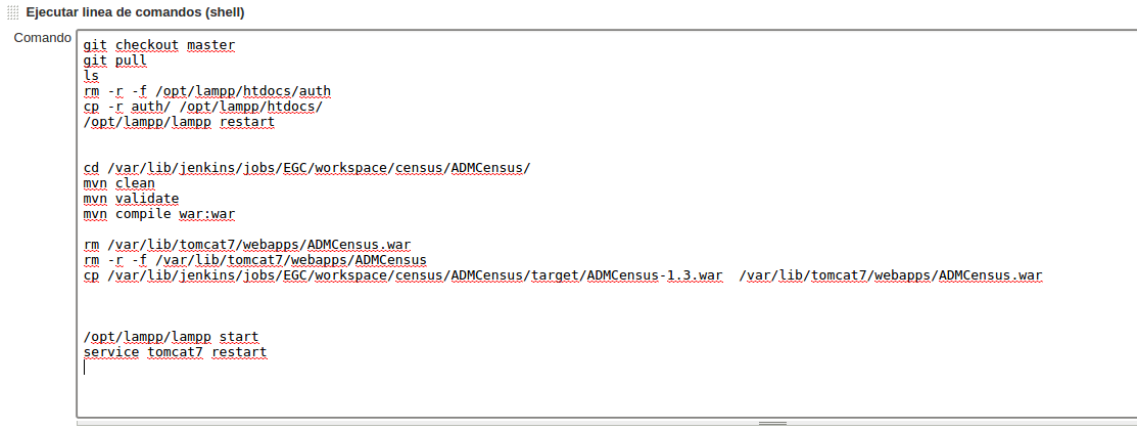


Figura 33: Comando para integrar Autenticación y Censos

Con las anteriores líneas de código, estamos integrando nuestro subsistema y el de autenticación, además debemos tener en cuenta que se debe limpiar la caché de Maven para borrar datos residuales. De esta forma podemos conseguir la automatización del despliegue con todos los subsistemas, sin tener que preocuparnos de los cambios que realicen en sus respectivos repositorios.

Descripción del entorno

En nuestra opinión es importante que a la hora de empezar el desarrollo, es buscar algún servidor que aloje el código fuente, ya que cuando se trata de un equipo de desarrolladores, trabajar en local y después unir todo el código al final del desarrollo aumenta las posibilidades de fracaso a la hora de realizar la integración.

En este caso, el equipo decidió alojar su código fuente en el repositorio de ProjETSII proporcionado por la Universidad de Sevilla.

Este repositorio se utilizará como un gestor de versiones, poder ramificar el desarrollo, tener baselines, etc. Como se puede ver en las siguientes imágenes la gestión del código fuente en primer lugar y el historial de versiones en la siguiente, vemos la gran utilidad que nos proporciona, debido a que si un equipo en el cual se está desarrollando la aplicación en local se estropea, el código queda a salvo en el repositorio.

Además, en el control de cambios, nos es de gran utilidad a la hora de subir archivos que ya han sido modificados, permitiendo comparar lo que se quiere agregar al repositorio con lo existente.

Sobre este tema de resolución de conflictos ya se ha explicado anteriormente en el apartado *Gestión del código fuente* como el grupo de desarrolladores tomó las decisiones de resolución.

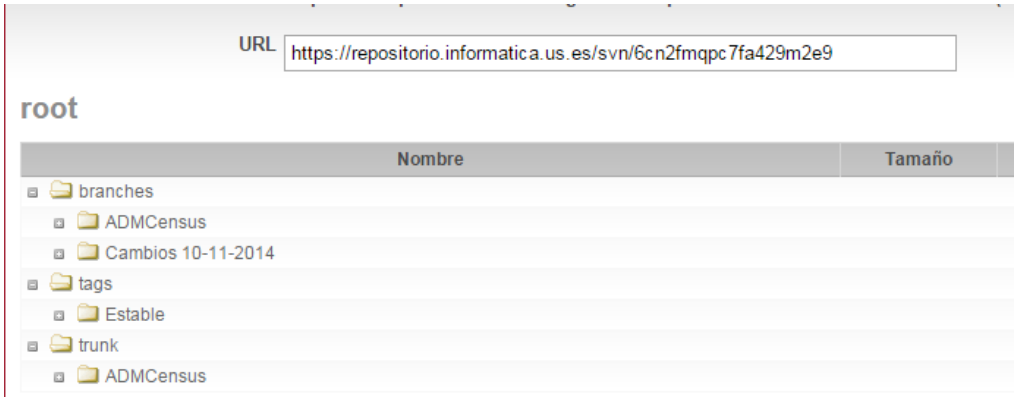


Figura 34: Vista del repositorio en ProjETSII

Figura 35: Cambios realizados en el repositorio (ProjETSII)

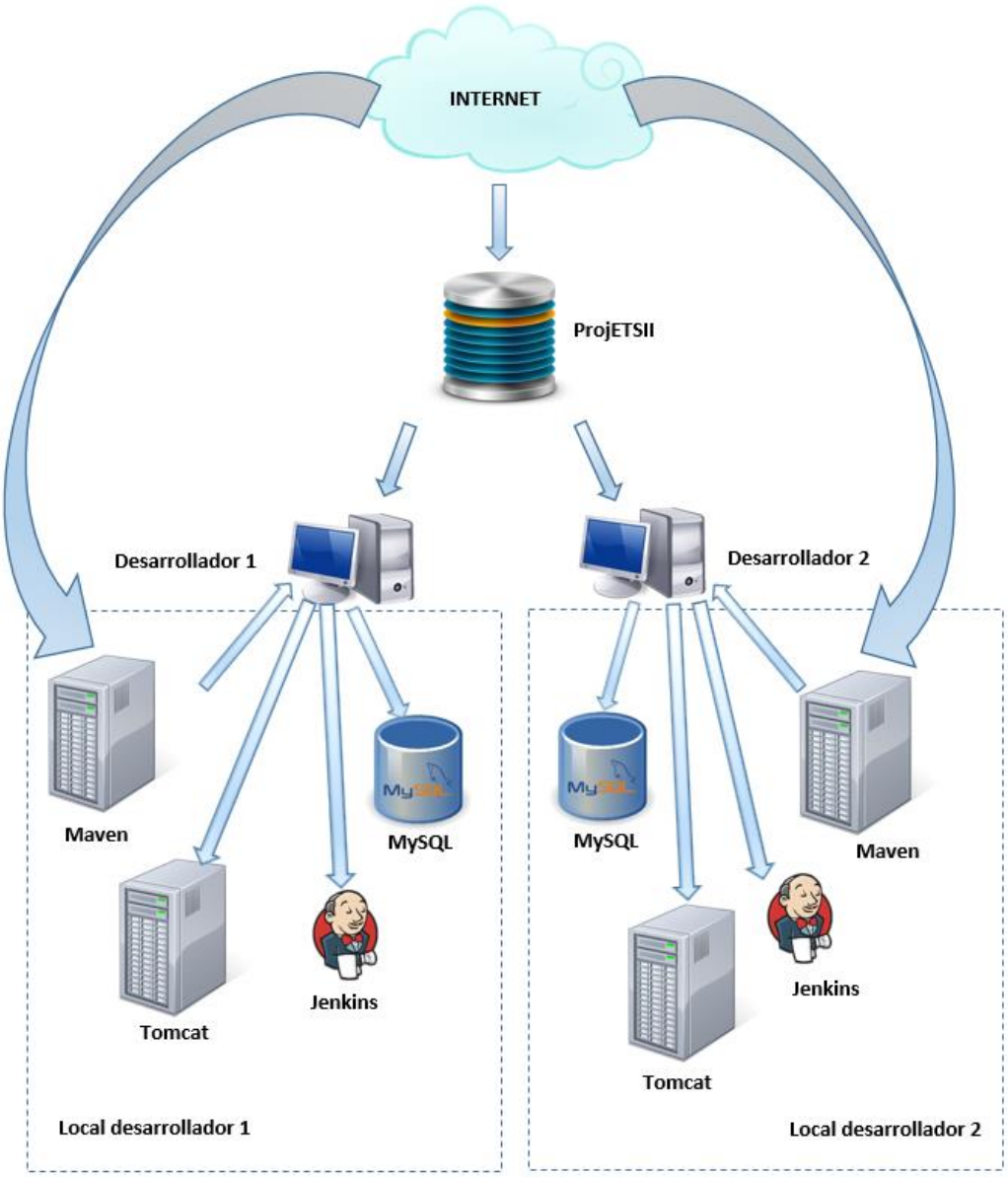


Figura 36: Mapa de integración interna

Este servidor está alojado en la nube por lo cual el equipo de desarrollo se despreocupa de eso.

Una vez descrito el repositorio que se utilizará, describiremos el entorno local de los desarrolladores.

Cada uno tiene un equipo portátil con la máquina virtual la cual contiene las herramientas necesarias para el trabajo, las cuales se detallan a continuación:

Conectividad a internet: es totalmente necesaria debido a Maven, ya que utilizar este sin conexión a Internet, le quita todas las ventajas proporcionadas.

Maven: herramienta para la construcción de software. En este caso, el equipo lo ha utilizado para la gestión de las dependencias y librerías del proyecto y para la creación de la estructura de la aplicación. Aunque tiene muchas funciones más, no han sido necesarias.

Servidor Tomcat 7: servidor de aplicaciones muy extendido para el despliegue de aplicaciones basadas en JEE.

SGBD MySQL 5.5: Sistema de base de datos relacional encargada de persistir los datos necesarios de la aplicación.

Una vez fueron conocidos todos los subsistemas con los que Creación/Administración de censos se comunicaba y se identificaron los servicios que debíamos ofrecer, evaluamos los cambios que se debían realizar. En el caso de que dicha comunicación afectara a nuestro modelo de datos, se volvería a repetir el ciclo ascendente anteriormente descrito. Sin embargo, en los casos en los que solo ha sido necesario realizar una comunicación mediante Json o modificar la lógica de la aplicación, se ha realizado una integración “descendente”, ya que no ha sido necesario realizar el ciclo ascendente completo.

En base a los 10 principios de Martin Fowler para hacer una integración continua perfecta, evaluaremos cuales de éstos cumple la integración de nuestro subsistema:

✓ Mantener un único repositorio de código fuente: sí, todo el código se encuentra en ProjETSII, repositorio anteriormente descrito.

✓Automatizar la construcción del proyecto: sí, con la herramienta de maven, ejecutando el comando *mvn compile* sobre el directorio donde está el proyecto, podemos automatizar la compilación del proyecto para ver que esta todo correcto.

Veremos algo como la siguiente pantalla, signfica que Maven ha encontrado con éxito las dependencias y está consultando con su repositorio (http://repo.maven.apache.org/maven2) para ver las librerías que tiene el proyecto y si nos falta alguna.

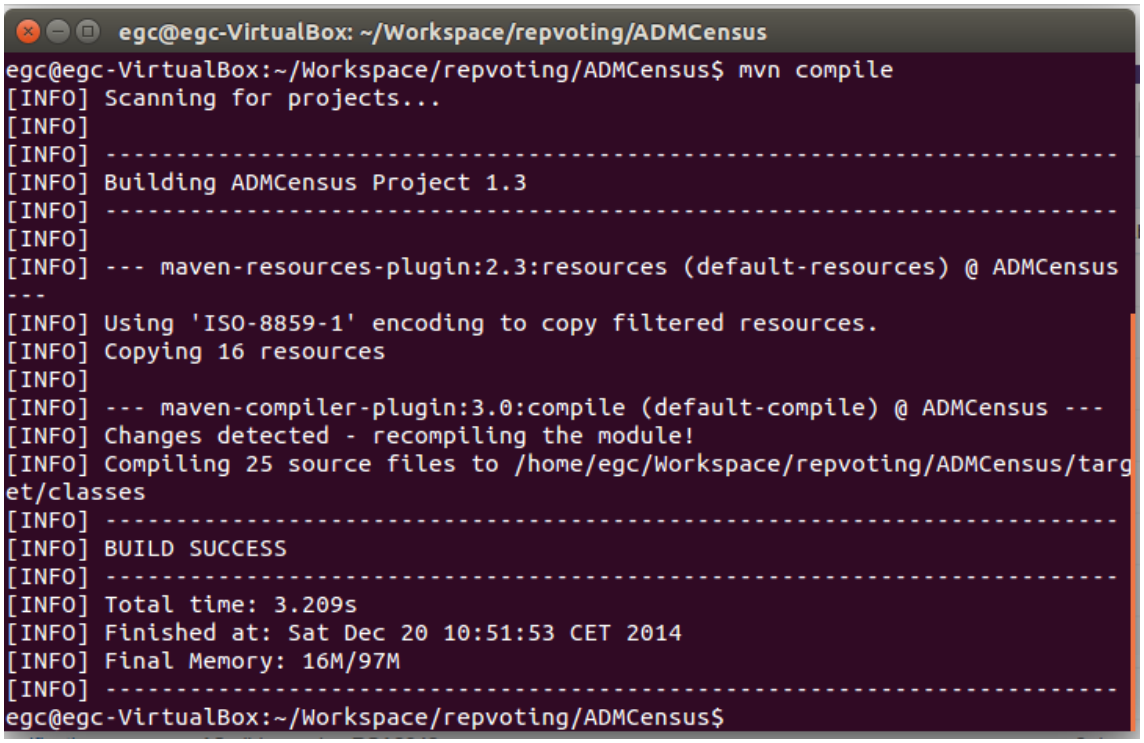


Figura 37: Maven descargando librerías del proyecto

✓ Hacer que la construcción del proyecto ejecute sus propios tests: sí, utilizando la herramienta de Junit, se automatizan los test cada vez que se hacía un *mergeo* para comprobar su correcta funcionalidad.

X Entregar los cambios a la línea principal todos los días: no, puesto que nos parece excesivo dicho *mergeo*. El equipo decidió llevar cambios a la rama principal una vez por semana o al finalizar una funcionalidad completa.

✓ Construir la línea principal en la máquina de integración: sí, el equipo lo primero que realizó fue una línea base del proyecto para que todas las ramas

extendiera de la principal, asegurando así un ramificado con la coherencia de funcionalidad adecuada.

✓ Mantener una ejecución rápida de la construcción del proyecto: sí, ya que tanto el despliegue mediante el fichero .war ó en la máquina de desarrollo, el tiempo máximo para construir el proyecto es de 7-9 minutos.

✓ Probar en una réplica del entorno de producción: sí, el equipo tiene una máquina de pre-producción con las herramientas instaladas simulando las que el cliente tendría.

✓ Hacer que todo el mundo pueda obtener el último ejecutable de forma fácil: sí, pues el equipo proporcionará un fichero .war con el cual solo lo tendrá que desplegar con el servidor de aplicaciones Tomcat 7 y el proyecto estará preparado para su utilización.

✓Publicar qué está pasando: si, el equipo en cada *commit* o *merge* del repositorio, explicaba lo realizado en comentarios

✓ Automatizar el despliegue: aunque al comienzo del desarrollo se comenzó a realizar de una forma manual, tras conocer la herramienta de Jenkins se realizó con esta.

6.2.2. INTEGRACIÓN EXTERNA

Tras la propuesta de creación de un repositorio común por parte de un miembro del grupo de otro subsistema, en concreto Autenticación, nuestra propuesta para integrarnos con los distintos subsistemas con los que estamos relacionados y de forma general para el resto, teniendo en cuenta los principios de integración de Martin Fowler es la siguiente:

✓ Mantener un único repositorio de código fuente → Se cumple ya que se utiliza el repositorio común, en el cual cada subsistema tiene una rama asignada en la que trabajar. Dicho repositorio se encuentra alojado en Git en la siguiente dirección: https://github.com/EGC-1415-Repositorio-compartido/repvoting

X Automatizar la construcción del proyecto → Debido al desconocimiento de las tecnologías usadas por otros subsistemas, no se cumple.

✓ Hacer que la construcción del proyecto ejecute sus propios tests → Cada subsistema realizará test unitarios para comprobar la correcta funcionalidad y detectar posibles errores.

X Entregar los cambios a la línea principal todos los días → No se cumple la realización de "commit" cada día, pero sí siempre que se añada una nueva funcionalidad válida o se realicen cambios.

✓ Construir la línea principal en la máquina de integración → En una máquina, ir haciendo los merge de manera secuencial, es decir, cada rama por separado. De esta forma en dicha máquina se irá probando cada combinación de las ramas hasta completar la construcción, que quedará alojada en la máquina.

X Mantener una ejecución rápida de la construcción del proyecto → Al haber distintas tecnologías, estimamos que la construcción de todos los subsistemas será de 45 minutos. Por tanto, no se considera rápida.

✓ Probar en una réplica del entorno de producción → Mediante una máquina virtual, se simulará en el entorno donde se hará uso de la aplicación Agora@US para

hacer pruebas de rendimiento, fiabilidad y funcionalidad de la aplicación en un estado estable y "final". Entendiendo como final que responde correctamente a la funcionalidad requerida a falta de pruebas en el entorno donde se usará.

X Hacer que se pueda obtener el último ejecutable de forma fácil → No se cumple debido a la multitud de tecnologías distintas con las que está construido cada subsistema, no habrá un único ejecutable.

✓ Publicar qué está pasando → Se cumple ya que todos los desarrolladores tienen acceso a cada subsistema y pueden ver los fallos, comentarios de las subidas, etc., en definitiva, se tiene un estado actualizado del proyecto en general.

✓ Automatizar el despliegue → Cada subsistema generará scripts y artefactos de forma que para realizar la integración en los distintos entornos (desarrollo, pruebas y producción) se realice el despliegue de forma sencilla con Jenkins.

Entorno de integración:

Cada subsistema deberá llevar lo necesario para que su proyecto arranque y permita integrarse con los otros subsistemas con los que esté relacionado. En nuestro caso, Creación/Administración de Censos, será necesario la base de datos MySQL 5.5, servidor Tomcat e IDE de desarrollo Eclipse.

Una vez se haya puesto todo en la máquina, la cual será la utilizada como máquina de integración (5º principio de Martin Fowler: *Construir la línea principal en la máquina de integración*), se procederá a integrar cada subsistema de forma ordenada y coherente. No se podrá integrar un nuevo subsistema hasta que el anterior no haya sido probado y comprobado el correcto funcionamiento.

Orden de integración (general):

El orden de integración propuesto es, en nuestra opinión, según la complejidad de integración, dejando para el final los subsistemas que no requieren integración y en primer lugar el subsistema el cual necesita integrarse o es integrado por una gran cantidad de subsistemas. Es decir, se intentará montar la base funcional de la

aplicación como la autenticación, creación de votaciones y censos y por último se dejará la parte más visual como el subsistema de visualización. Con este orden se pretende dejar la aplicación tras la integración con una mínima funcionalidad.

- Autenticación
- Creación/Administración de censos
- Creación/administración de votaciones
- Cabina de votación
- Almacenamiento de votos
- Sistema de modificación de resultados
- Verificación
- Recuento
- Frontend de Resultados
- Visualización de resultados
- Deliberaciones

Orden de integración para Creación/Administración de censos:

- Autenticación
- Creación/administración de votaciones
- Cabina de votación
- Deliberaciones

Estética de la aplicación:

Como cada grupo tendrá su estilo, podríamos votar el que más guste y adaptar todos los subsistemas a ese. En caso de que no se haya desarrollado ningún estilo, podrán votar igualmente para poder aplicarlo al suyo, de esta forma, los subsistemas no desentonarán.

Forma de integración:

La forma real de trabajar ha sido la siguiente:

Cada subsistema ha creado una rama en el repositorio Git anteriormente indicado. En nuestro caso, al haber estado trabajando con Subversion para la realización del código, dicho repositorio Git solo ha sido utilizado para subir versiones estables y funcionales del proyecto, salvo para cambios realizados a partir de la integración inicial. Cuando ha sido necesario realizar un cambio, se ha realizado, testeado y actualizado en Subversion y a continuación subido a la rama correspondiente (*adminCensos*) en Git. Esta actualización se ha realizado mediante el uso de Eclipse a través del plugin añadido a dicho IDE. En la siguiente imagen se muestra un ejemplo de lo que acabamos de explicar. El *commit* es para la actualización de un menú del proyecto, para la realización del *commit* se siguen los siguientes pasos: hacer clic con el botón derecho sobre el proyecto (o si es un solo archivo el modificado también valdría) → Team → **Commit** y aparecerá la ventana siguiente para indicar el archivo a actualizar y un comentario sobre el cambio realizado.

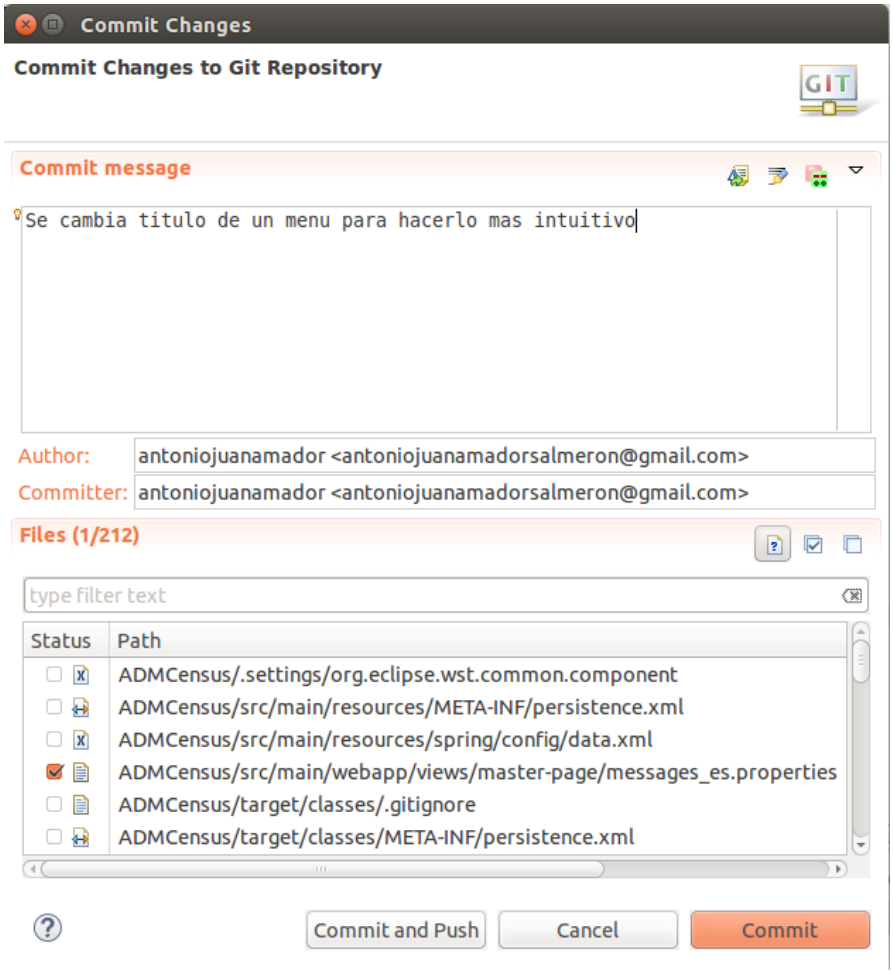


Figura 38: Commit en local

Esto habrá realizado la actualización en el repositorio local. Para llevar los cambios al repositorio común hay que realizar un *push* de manera similar al *commit*: hacer clic con el botón derecho sobre el proyecto (o si es un solo archivo el modificado también valdría) → Team → **Commit and Push.** Hacer clic en "Ok" y si no ha habido ningún problema, como podría ser la generación de un conflicto, se habrá subido el código y estará listo para ser utilizado por otros subsistemas.

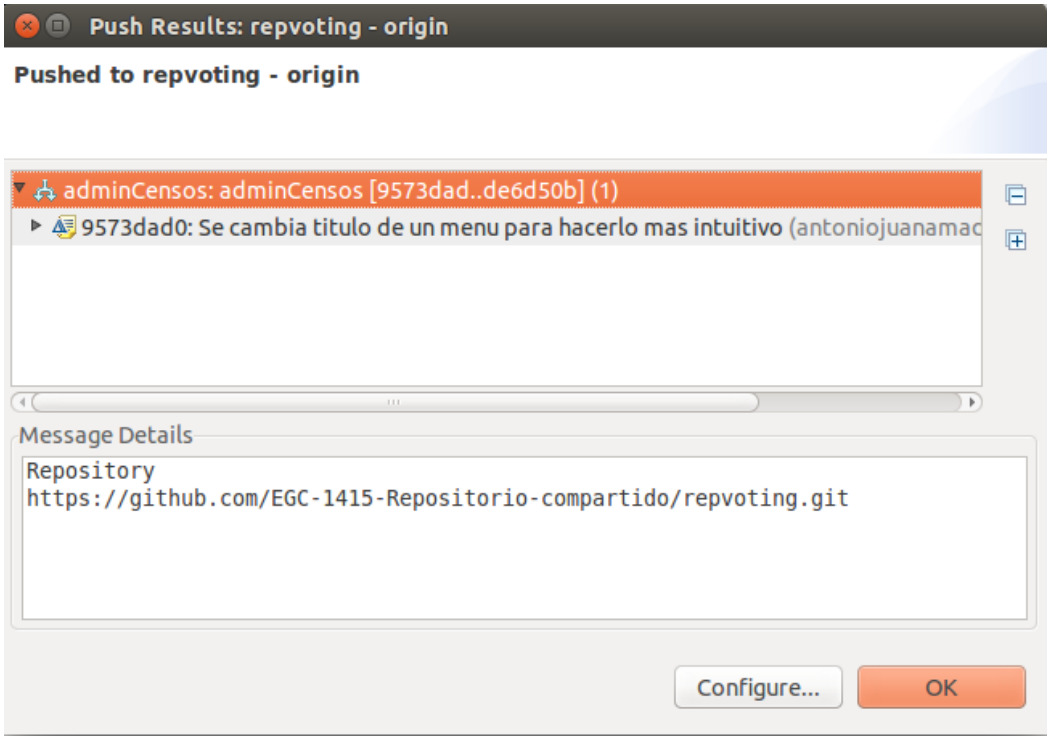


Figura 39: Commit and Push

6.3. EJERCICIO 1

"Realizar la integración con el subsistema Deliberaciones haciendo uso de la máquina virtual proporcionada. Comprobar que se ha realizado correctamente."

Resolución

El primer paso es preparar en el entorno de trabajo como en el apartado anterior (*Gestión del código fuente*) se indica. Es decir, iniciar MySQL (instalado con Xampp) y abrir Eclipse. Una vez realizado esto nos encontraremos algo similar a la siguiente imagen (en el caso de no haber integrado aún ningún subsistema)

Nota: para iniciar el servicio Xampp completo (Apache, Tomcat y MySQL) basta con poner el siguiente comando en un terminal: sudo /opt/lampp/lampp start

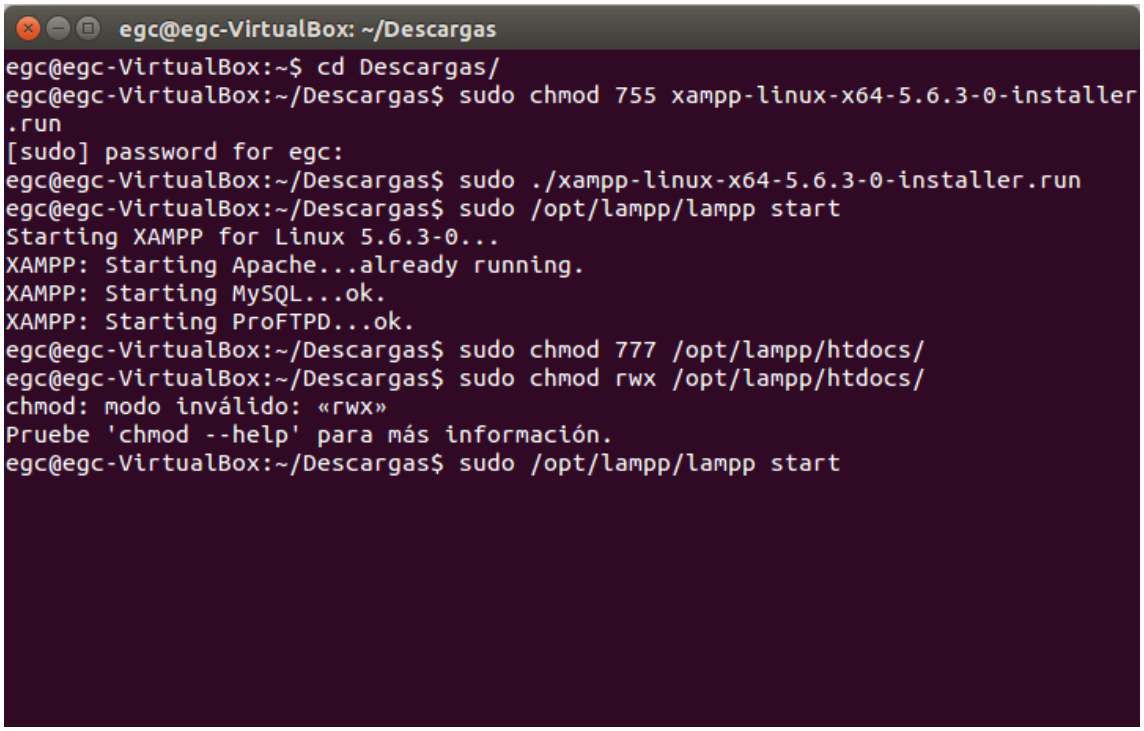


Figura 40: Iniciar servicio Xampp

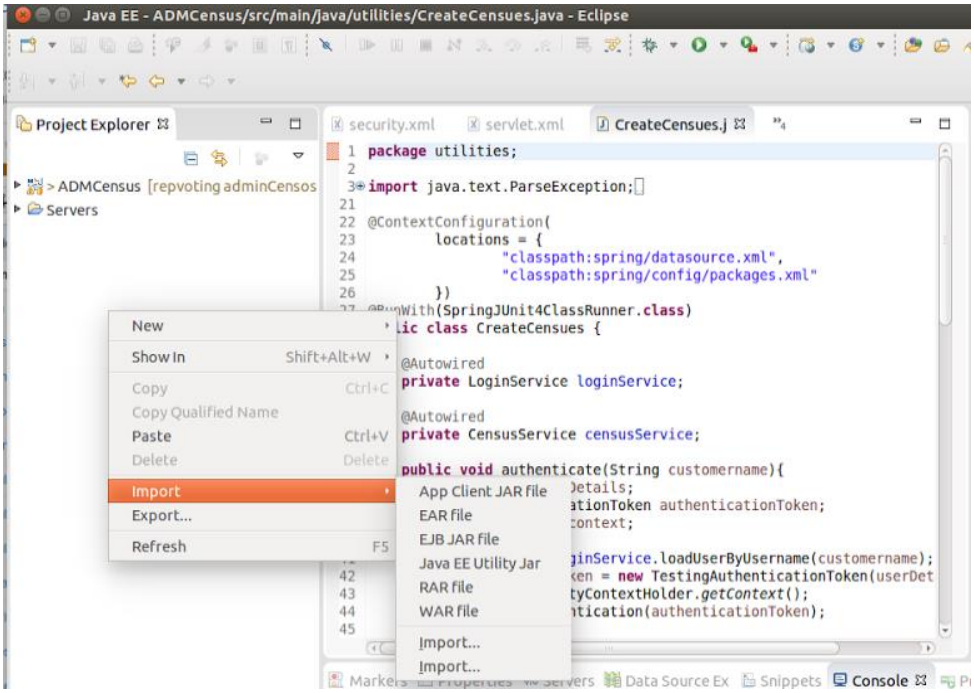


Figura 41: Menú conextual de la ventana Project Explorer

Para integrar el subsistema *Deliberaciones* lo primero que hay que realizar es descargarnos el proyecto del repositorio común donde está alojado. Para ello, en Eclipse, importamos el proyecto desde Git (File → Import → Git → Projects from Git)

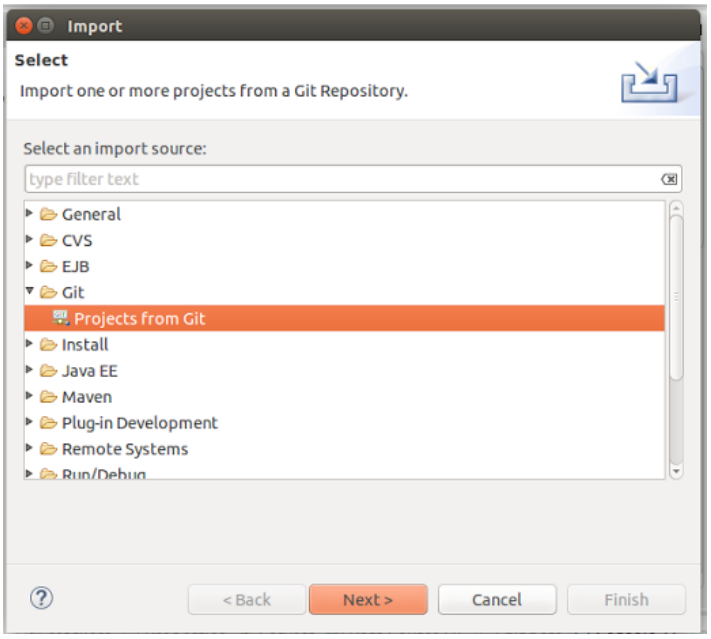


Figura 42: Ventana import

Hacemos clic en “Next”y a continuación en “Clone URI”

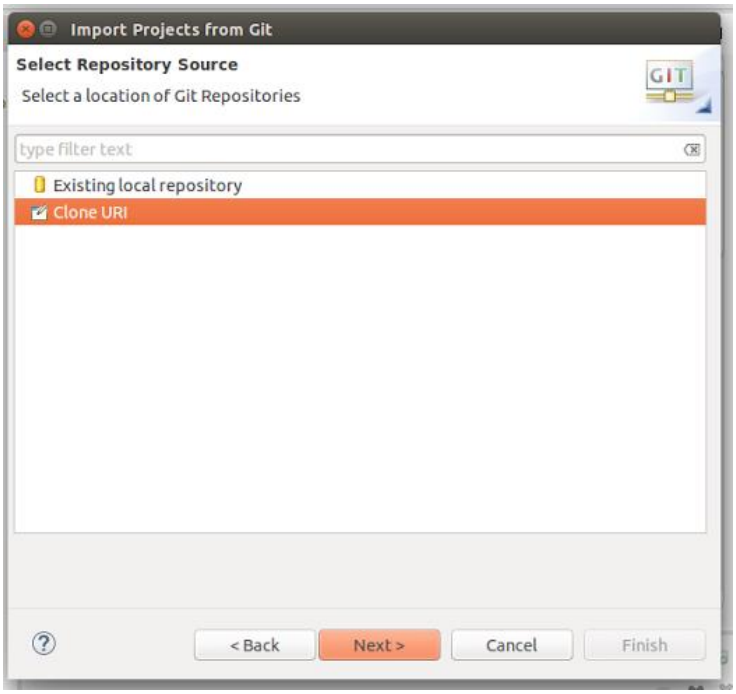


Figura 43: Ventana Git

En el menú que nos aparecerá hay que indicar la URL del repositorio: https://github.com/EGC-1415-Repositorio-compartido/repvoting.git y el Usuario y Contraseña con los cuales estamos registrados en GitHub para autenticarnos.

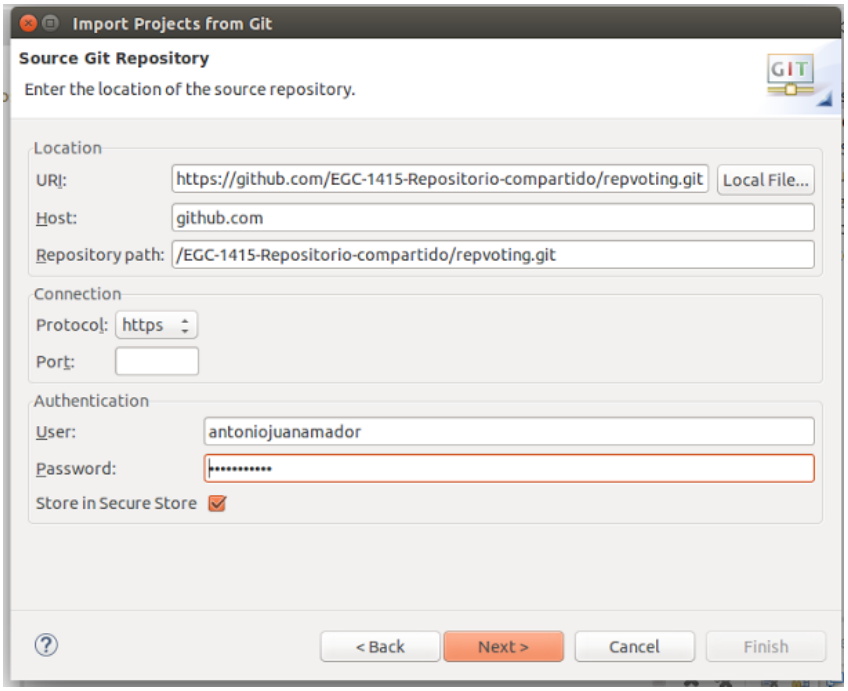


Figura 44: mportar proyecto Git

Hacemos clic en “Next” y aparecerá un nuevo menú similar al siguiente. En dicho menú están todas las ramas que se encuentran en el repositorio indicado. Volvemos a hacer clic en “Next”.

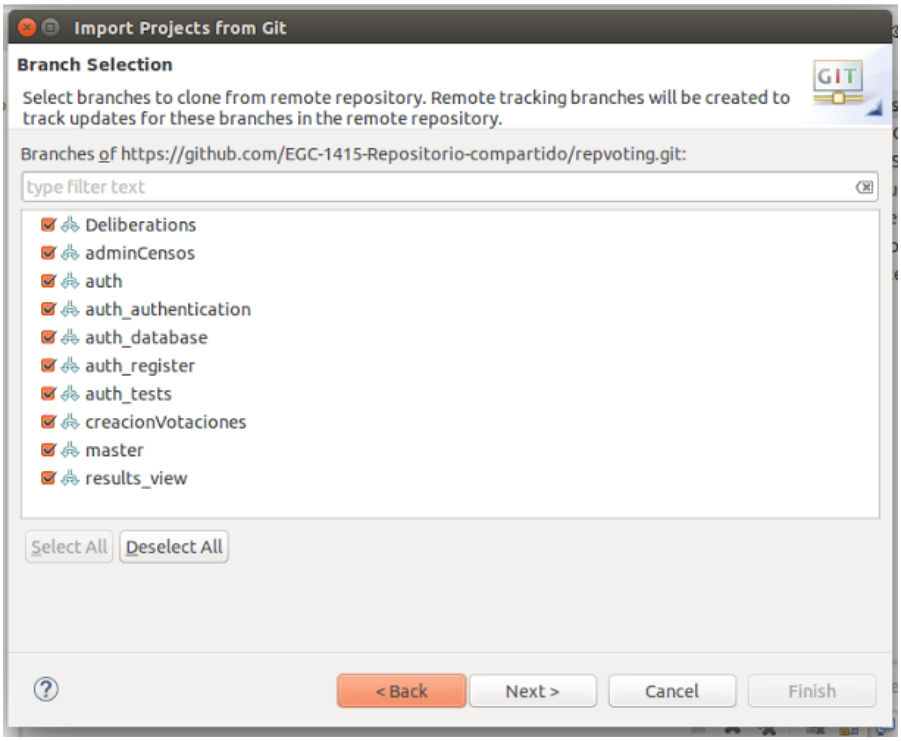


Figura 45: Vista de todas las ramas de un repositorio

En el siguiente menú seleccionamos la rama *Deliberations* en “Initial branch” y “Next”.

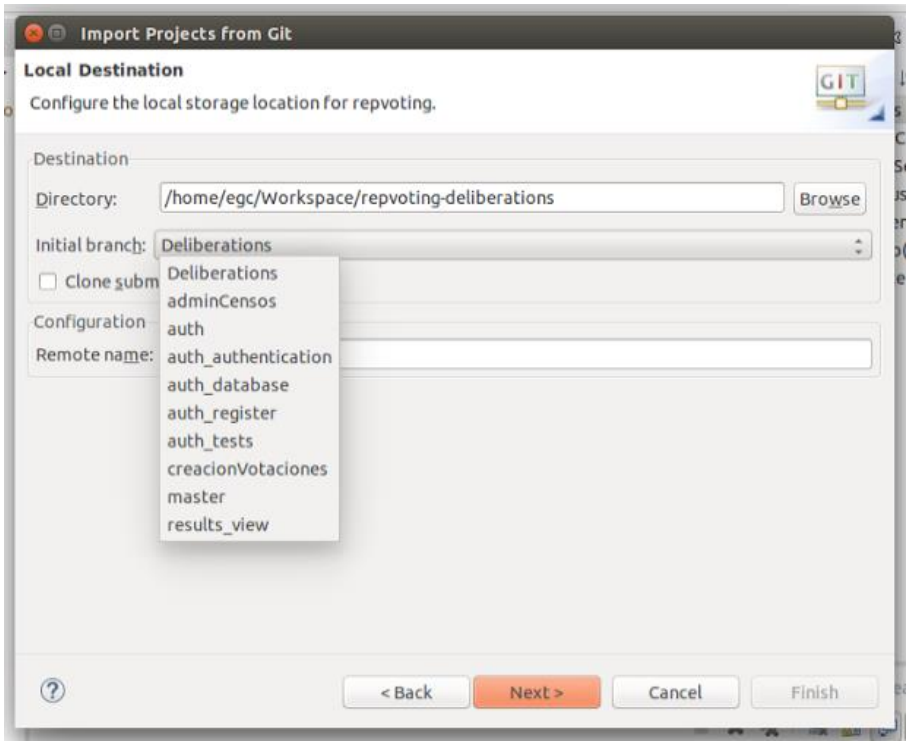


Figura 46: Selección de la rama a importar

Hacemos clic en “Import existing projects” lo cual indica que es un proyecto ya existente, y volvemos a hacer clic en “Next”.

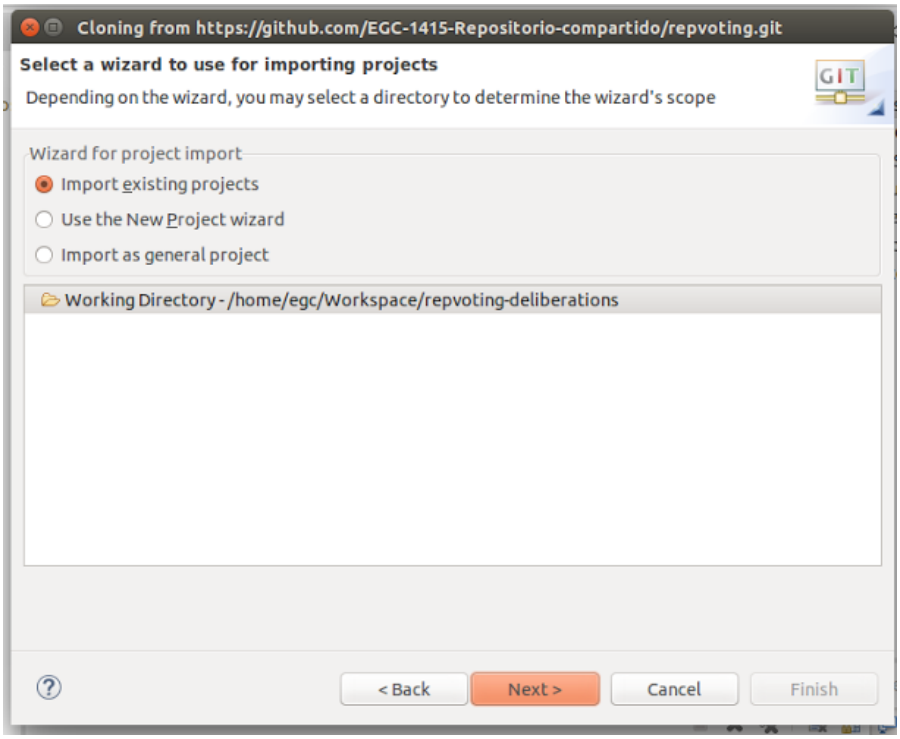


Figura 47: Importar proyecto existente

Aparecerá un nuevo menú para confirmar que el proyecto seleccionado y la rama son las correctas. Hacer clic en "Next" para continuar y ya estará el proyecto en tu WorkSpace.

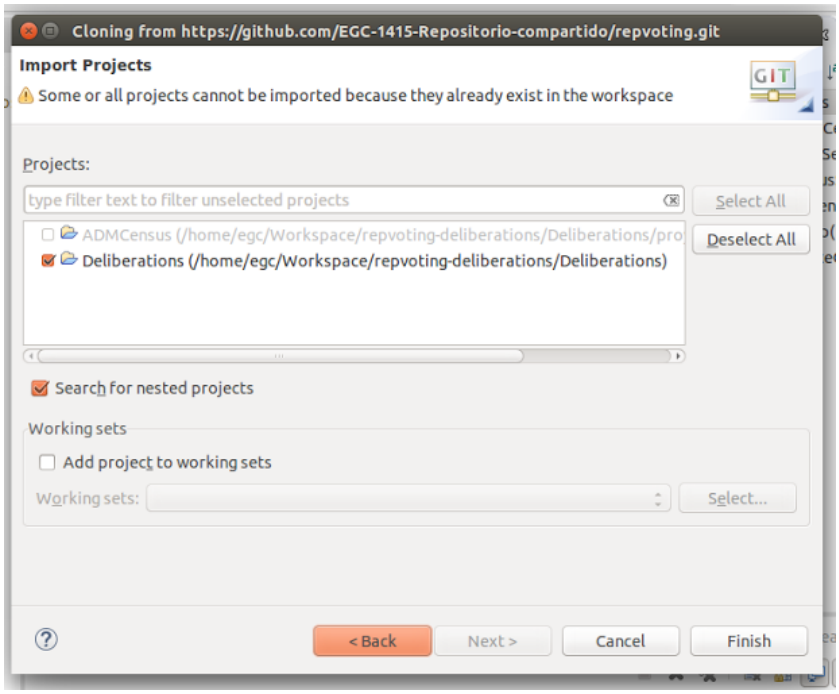


Figura 48: Confirmar proyecto a importar

Tras realizar estos pasos tu WorkSpace debe quedar similar a la siguiente imagen:

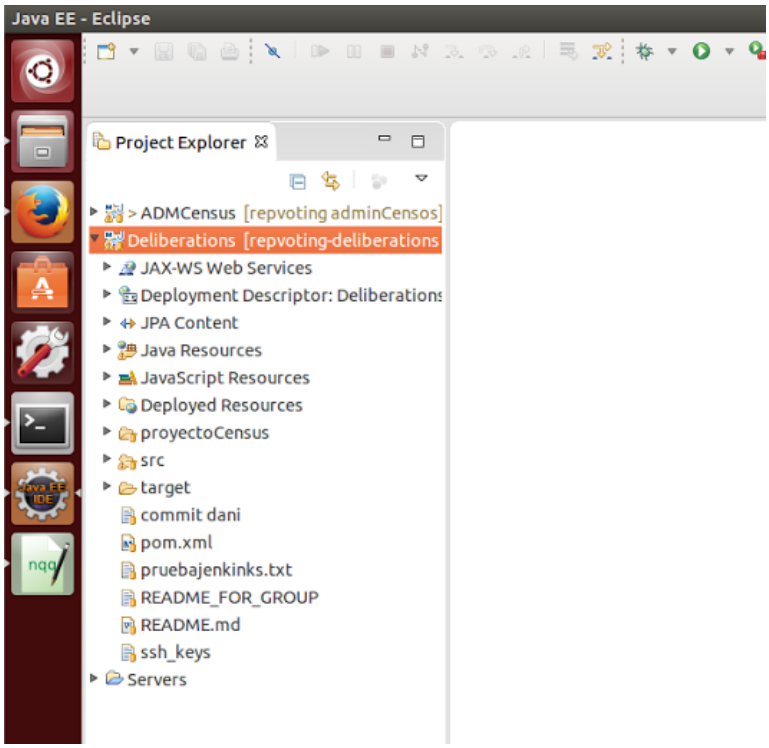


Figura 49: Vista del workspace

Para la segunda parte del ejercicio, comprobar que el subsistema *Deliberaciones* está integrado correctamente con *Administración/Creación de censos* (ADMCensus), habrá que arrancarlo y confirmar que las llamadas entre ambos se realizan de la forma esperada. Es necesario los siguientes pasos previos a arrancar el servidor Tomcat:

1. Crear base de datos llamada *Deliberaciones*
2. Configurar credenciales de la base de datos:
   a. En phpMyAdmin, creamos un usuario. En el menú superior pulsamos Privileges y add user:

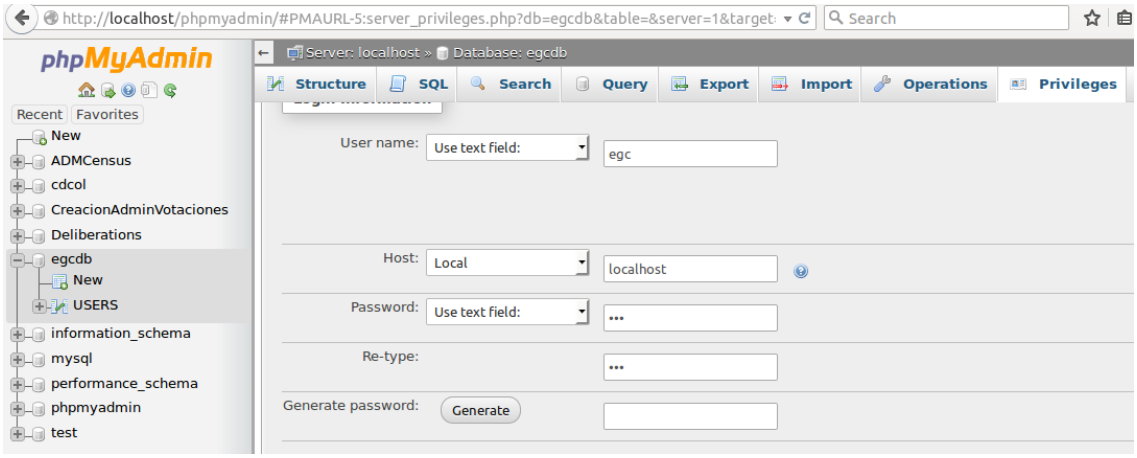


Figura 50: phpMyAdmin

A este usuario (egc:egc) se le van a dar todos los permisos para simplificar la configuración de permisos dentro de los proyectos.

b. En el proyecto importado hay que modificar dos archivos. El primero "*data.xml*" situado en **src/main/resources/spring/config/data.xml**. Debe quedar el bloque llamado "Data source" como se indica en la siguiente imagen:

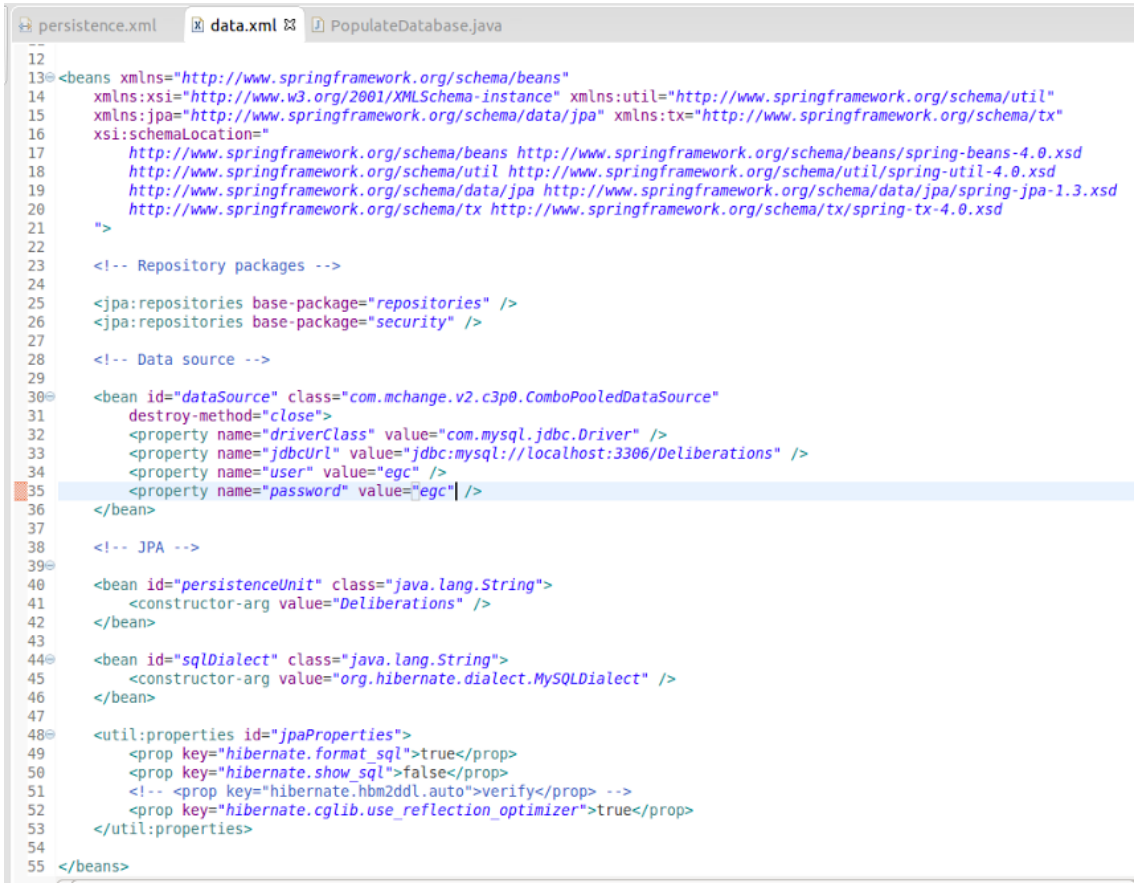


Figura 51: data.xml

El segundo archivo a modificar es *"persistence.xml"* que se encuentra en **src/main/resources/META-INF/persistence.xml.** Habrá que modificar los property y quedar de la siguiente forma:

persistence.xml data.xml PopulateDatabase.java

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <!--
4   * persistence.xml
5   *
6   * Copyright (C) 2014 Universidad de Sevilla
7   *
8   * The use of this project is hereby constrained to the conditions of the
9   * TDG Licence, a copy of which you may download from
10  * http://www.tdg-seville.info/License.html
11  -->
12
13 <persistence version="2.0"
14     xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
15     xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
16
17     <persistence-unit name="Deliberations">
18
19         <!-- <provider>org.hibernate.ejb.HibernatePersistence</provider> -->
20
21         <properties>
22             <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver" />
23             <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/Deliberations" />
24             <property name="javax.persistence.jdbc.user" value="egc" />
25             <property name="javax.persistence.jdbc.password" value="egc" />
26
27             <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect" />
28             <!-- <property name="hibernate.hbm2ddl.auto" value="none" /> -->
29         </properties>
30
31     </persistence-unit>
32
33 </persistence>
34
```

Figura 52: persistence.xml

3. Popular la base de datos: abrir el archivo PopulateDatabase.java (src\main\java\utilities) y ejecutar como Java Application (PopulateDatabase.java → Run as Java Application). Esto creará la estructura de la base de datos del proyecto y persistirá algunos objetos si los tiene.

Si todo ha ido bien, podemos arrancar el servidor Tomcat y realizar las llamadas requeridas entre ambos subsistemas de forma exitosa.

6.4. EJERCICIO 2

"Realizar la integración con el subsistema Creación de votaciones haciendo uso de la máquina virtual proporcionada. Comprobar que se ha realizado correctamente."

Resolución

Este ejercicio es muy similar al anterior. Los pasos hasta importar el proyecto son los mismos, a partir de aquí hay algunas diferencias respecto a la integración con *Deliberaciones*.

Al igual que con *Deliberaciones* descargarnos el proyecto del repositorio común donde está alojado. Importamos el proyecto en Eclipse desde Git (File → Import → Git → Projects from Git). Clic en "Clone URI" para clonar el repositorio https://github.com/EGC-1415-Repositorio-compartido/repvoting.git e indicamos el Usuario y Contraseña con los cuales estamos registrados en GitHub para autenticarnos.

Aparecerá, como en el caso anterior, un menú con todas las ramas existentes en el repositorio. Hacemos clic en "Next". En el siguiente menú, a diferencia del ejercicio 1, seleccionamos la rama *creacionVotaciones* en "Initial branch" y "Next". En el nuevo menú seleccionar "Import as general project" y "Next".

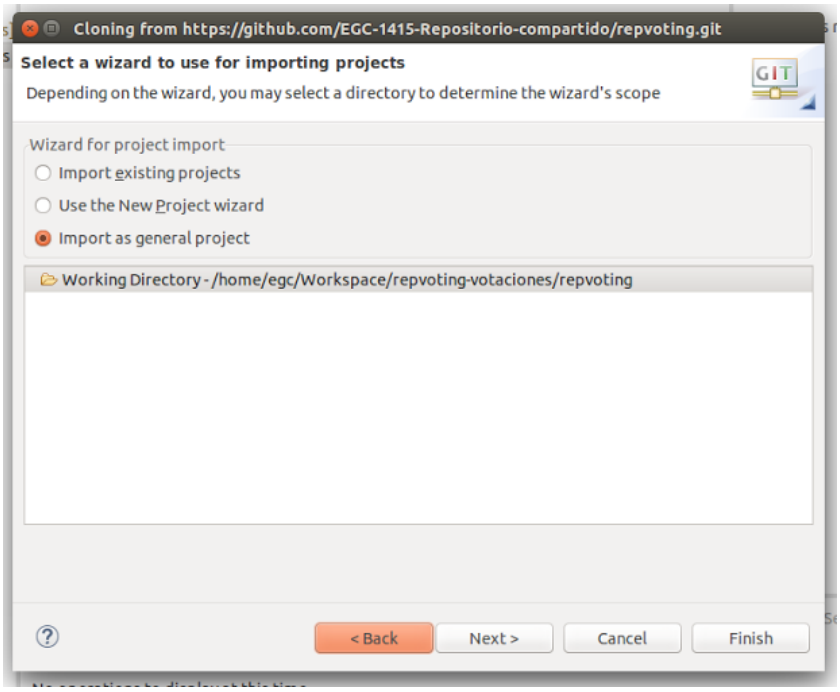


Figura 53: Importar proyecto general

Una vez realizado esto tendrás el proyecto del subsistema *Creación de votaciones* en tu WorkSpace. Hay que realizar unos cambios sobre dicho proyecto para que todo funcione correctamente. Al haber sido importado como proyecto general, hay que convertir a un proyecto Maven. Esto se realiza de la siguiente forma: hacer clic con el botón derecho del ratón sobre el proyecto → Configure → Convert to Maven Project.

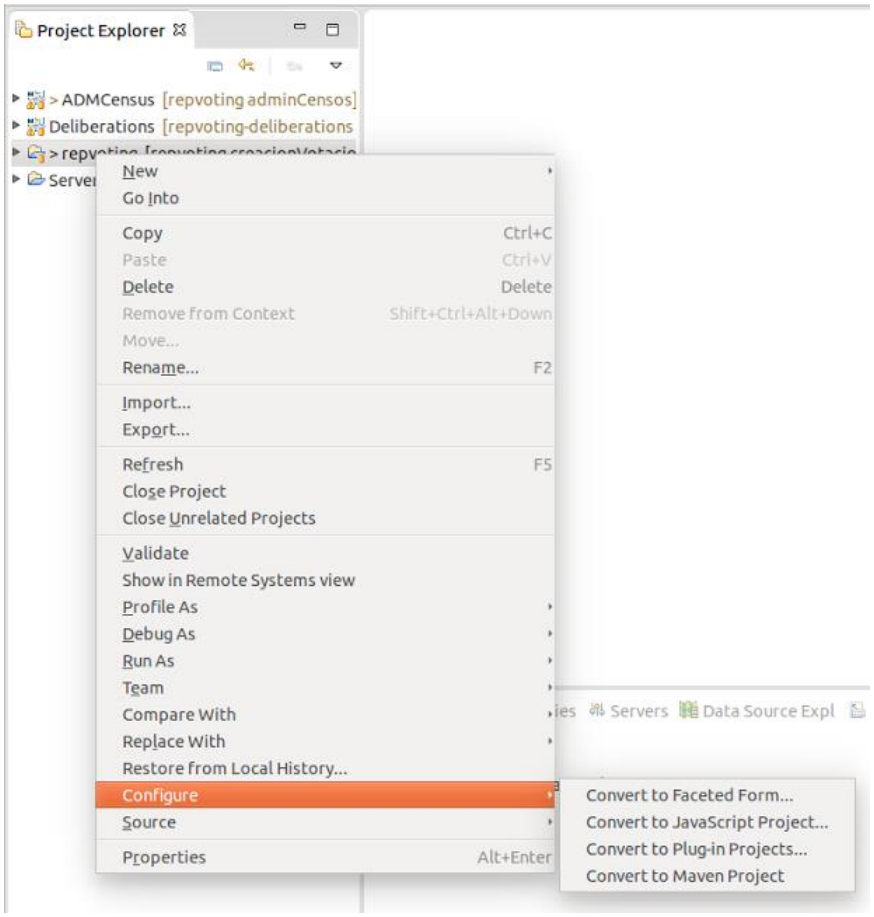


Figura 54: Menú configure de un proyecto

Aparecerá un menú para crear el POM.xml, para poder tener las dependencias del proyecto, hacer clic en “Finish”.

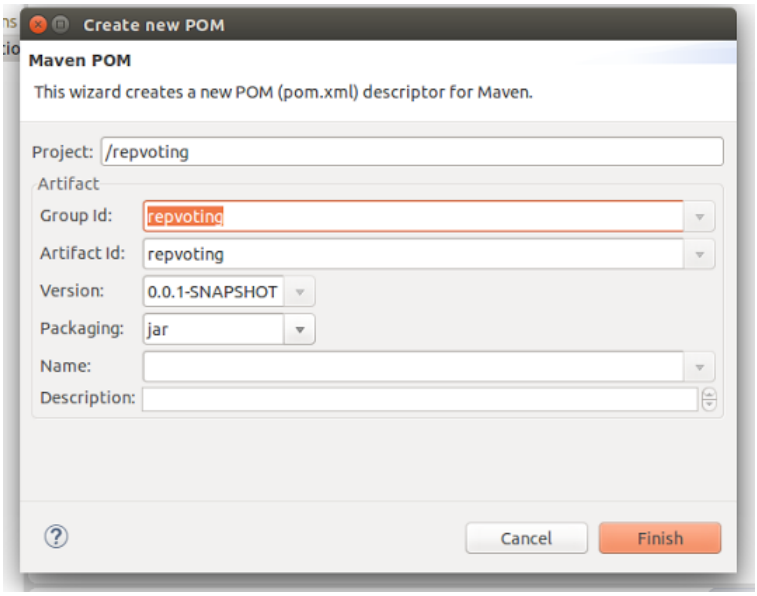


Figura 55: Crear fichero pom.xml

Hasta aquí es la realización de importar el proyecto a tu equipo. Para comprobar que funciona correctamente, habrá que realizar los mismos pasos que en el ejercicio anterior:

1. Crear base de datos llamada *Deliberaciones*
2. Configurar credenciales de la base de datos
3. Popular la base de datos
4. Arrancar el servidor

7. GESTIÓN DEL CAMBIO, INCIDENCIA Y DEPURACIÓN

7.1. GESTIÓN DE CAMBIOS

Cuando hablamos de gestión de cambios, nos referimos a las acciones a llevar a cabo en los momentos en los que surgen cambios que tienen como consecuencia un impacto en el desarrollo de un determinado proyecto.

Esta fase va de la mano de depuración, ya que la depuración consta de la aplicación de cambios para solventar incidencias. En el momento que aparece un cambio a efectuar, la primera acción es analizar dicho cambio para llegar a contemplar los efectos tanto negativos como positivos para el desarrollo. Este análisis será el que determine si los cambios son factibles, dependiendo de su impacto, utilidad y beneficios; además de que esto puede repercutir a fases anteriores al desarrollo, como pueden ser la fase de análisis de requisitos.

Podemos clasificar los cambios en 3 tipos, dependiendo de su impacto:

- Si el impacto de dicho cambio es **crítico**, se realizará con la mayor prioridad buscando la estabilidad en el proyecto. Entendemos cambio crítico aquel que vaya en contra de alguna regla de negocio.
- En cambio, si dicha modificación es de **nivel medio**, la fecha de ejecución dependerá en función de otros cambios con mayor prioridad. Entendemos cambio medio aquel que afecta a una regla de negocio pero no va en contra de ella.
- Por último nos encontramos con los **cambios leves**, es decir, de muy poco impacto. Estos suelen ser los que se llevan a cabo en último lugar. Entendemos como cambios leves aquellos cambios que afectan tanto al diseño como a la navegabilidad del sistema.

Si dichos cambios son factibles, se lleva a cabo la realización de estos. La asignación de cambios se otorga al desarrollador que esté trabajando o se haya encargado del requisito funcional sobre el cual se tiene que efectuar un determinado cambio. Este desarrollador es el responsable de los cambios realizados en el código y es suya la responsabilidad de indicar por medio de los Commit, al repositorio, de los cambios realizados. De esta forma se pueden controlar todos los cambios realizados en el proyecto.

En cuanto a los roles para dicha gestión, tenemos a los propios desarrolladores, los cuales serán los encargados de ejecutar un cambio determinado en el código; al gestor de pruebas que será el encargado de analizar los cambios a efectuar y en último lugar tenemos al jefe de proyecto, cuya función es la de organizar al equipo y analizar junto con dicho gestor estos cambios.

Como conclusión, para mantener una buena práctica de gestión de cambios, es obligado mantener un registro de los cambios realizados durante el ciclo de vida del desarrollo del proyecto, para mantener un control mediante el cual se pueden resolver futuros conflictos.

7.2. GESTIÓN DE INCIDENCIAS

Inicialmente la gestión de incidencias se realiza verbalmente reportando los errores al grupo de trabajo en el cual se encuentra un determinado error. El grupo de trabajo afectado resuelve dicho error y reporta una solución a través de una aplicación de mensajería instantánea. A medida que la integración con los distintos grupos avanza, se utiliza como herramienta de gestión, ProjETSII, en la cual se realizan reportes mediante documentos o imágenes para explicar con un mayor detalle donde se encuentra dicho error. Finalmente, una vez solucionado dicho error, se comunica a través de ProjETSII que dicha incidencia ha sido solucionada y por tanto, donde se va a depositar el código con el error ya solucionado.

La forma correcta de tratar las incidencias se trata a continuación, para ello vamos a indicar cuales serían los pasos a seguir y qué información se debe almacenar:

1. Registro de incidencia:

La admisión y registro de la incidencia es el primer paso necesario y el más importante para poder ejecutar una correcta gestión de la misma.

Las incidencias pueden provenir de diversas fuentes tales como usuarios, gestión de aplicaciones, el Centro de Servicios o el soporte técnico, entre otros.

El proceso de registro debe realizarse inmediatamente, pues resulta mucho más costoso realizarlo a posteriori, además de que se corre el riesgo de que la aparición de nuevas incidencias demore indefinidamente el proceso en cuestión.

Cabe destacar que es muy importante aportar toda la información posible a la descripción de la incidencia ya que es la que va a ayudar a localizar el error reportado y por tanto solventarlo con una mayor brevedad.

Esto consiste en introducir un asunto de la incidencia breve y conciso, y una descripción en la que se detalle todo lo que ha ocurrido para encontrarse con un determinado error.

Se aconseja introducir:

- Los datos personales, tales como nombre, apellidos y correo electrónico.
- El entorno de desarrollo en el que ha surgido dicho error, en el caso de que el sistema se encuentre en desarrollo.
- El sistema bajo el que se está desarrollando o corriendo el sistema.
- El momento en el que esto sucedió.
- Las acciones que hicieron aparecer el error.
- Los mensajes que tanto el sistema como el entorno de desarrollo puede mostrar.
- Y por último, a ser posible, una imagen que refleje lo descrito en este apartado.

2. Clasificación de incidencias:

Al registrar una incidencia, debemos proceder a la clasificación de este. Esta tiene como objetivo principal el recopilar toda la información posible necesaria para que posteriormente puede ser utilizada para la resolución de dicha incidencia.

El proceso a seguir para realizar la clasificación debe implementar, al menos, los siguientes pasos:

- Categorización: se asigna una categoría (que puede estar a su vez subdividida en más niveles) dependiendo del tipo de incidente o del grupo de trabajo responsable de su resolución. Se identifican los servicios afectados por el incidente.
- Establecimiento del nivel de prioridad: dependiendo del impacto y la urgencia se determina, según criterios preestablecidos, un nivel de prioridad. Este nivel será el que determine el tiempo en el que se llevará a cabo la revisión de una determinada incidencia.
- Asignación de responsable: se debe atribuir a un responsable, el cual será el portavoz del grupo de trabajo el cual se encargará de informar y organizar el tratamiento, y por tanto la resolución de una determinada incidencia.

- Monitorización del estado y tiempo de respuesta esperado: se asocia un estado al incidente (por ejemplo: registrado, en progreso, suspendido en prueba, resuelto, cerrado)

BugTracker:

Como propuesta para realizar la gestión de incidencias, de entre varias herramientas que nos pueden llegar a ser útiles, se propone la herramienta *Zoho BugTracker*, la cual podemos encontrar accediendo al siguiente enlace:

https://www.zoho.com/bugtracker/

Esta es una aplicación que ha resultado tanto sencilla como útil e intuitiva, además puede decirse que está diseñada para poder asegurar una correcta gestión de incidencias.

Esto conlleva una mejora en el aseguramiento de la calidad del software y asistencia tanto a programadores como a cualquier otra persona involucrada en el desarrollo de un determinado proyecto. Además de que puede ser de ayuda en cuanto al seguimiento de los defectos de software, incluso en los sistemas informáticos ya desarrollados.

En primer lugar para utilizar dicha herramienta hay que registrarse en el sistema. Una vez creada la cuenta hay que crear un proyecto, en el que reportaremos las incidencias.

Tenemos que agregar a los componentes de dicho proyecto, los cuales también deben de estar registrados en el sistema. Agregar componentes es una tarea muy sencilla ya que la interfaz de dicha herramienta se asemeja mucho a una red social.

Para realizar incidencias en el menú lateral seleccionaremos "*Errores*". Nos aparecerá una ventana para introducir una nueva incidencia. Cuando pinchamos en dicho enlace aparecerá una imagen como la mostrada a continuación, en la cual podemos observar los distintos campos a rellenar para reportar dicha incidencia:

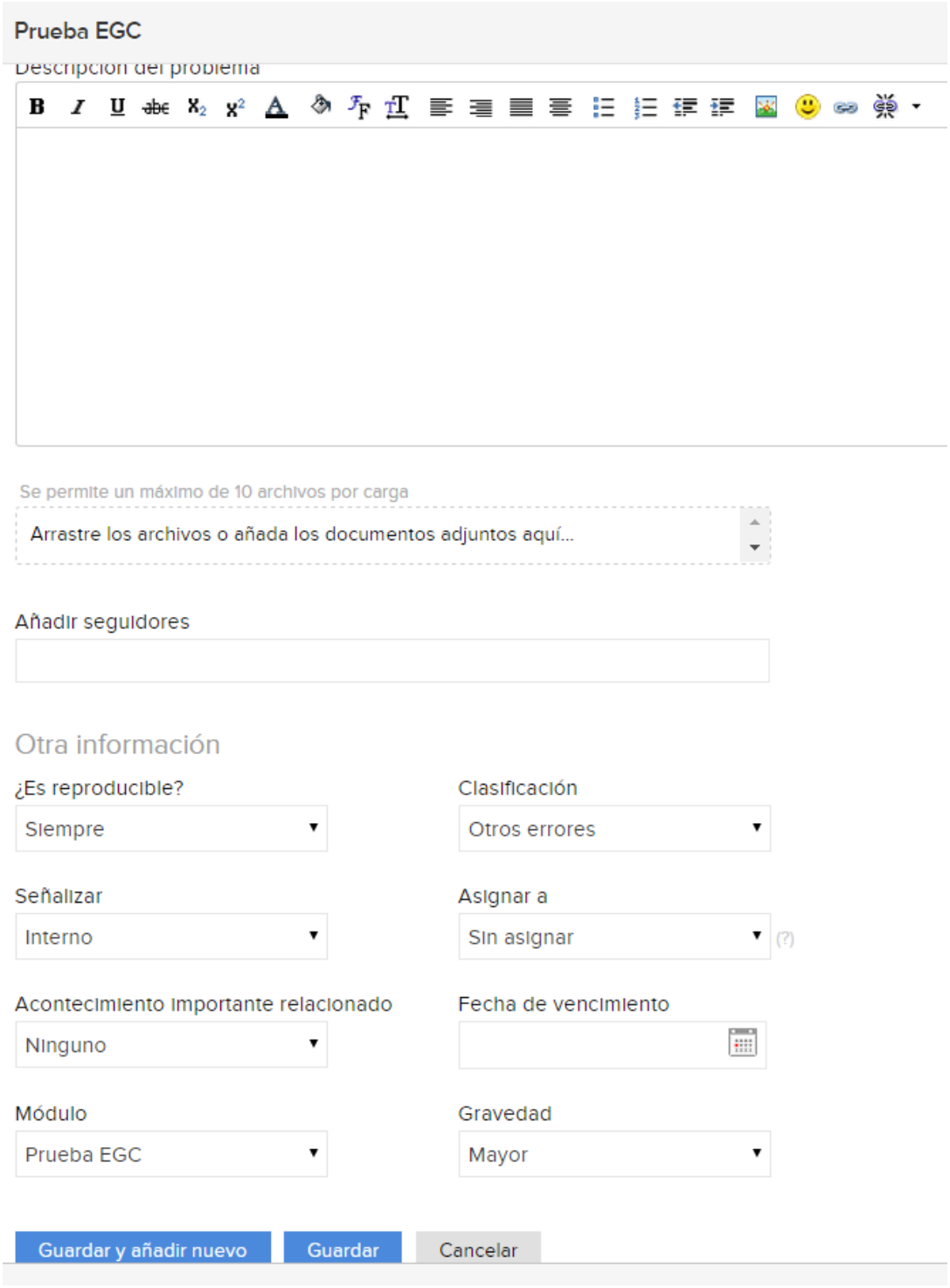


Figura 56: Nueva incidencia en BugTracker

En primer lugar, tenemos un campo para dar una descripción detallada de la incidencia.

En segundo lugar tenemos una opción que nos permite subir los ficheros en los cuales se ha encontrado la incidencia.

Tras esta, nos encontramos la opción mediante la cual podemos asignar dicha incidencia a alguno de los componentes del proyecto. En breves, comentaremos con mayor detalle esta opción.

Por último, vemos una sección en la que podemos introducir información adicional. Como por ejemplo:

Si es reproducible, qué clasificación contiene el error, que gravedad tiene, que modulo se ve afectado, etc.

Como hemos dicho anteriormente, vamos a centrarnos en el procedimiento que debe realizar la persona a la que se le ha asignado la incidencia.

En primer lugar automáticamente se le envía un correo electrónico indicando la asignación de dicha tarea, además, como podemos ver a continuación, dentro de la herramienta aparecerá una notificación.

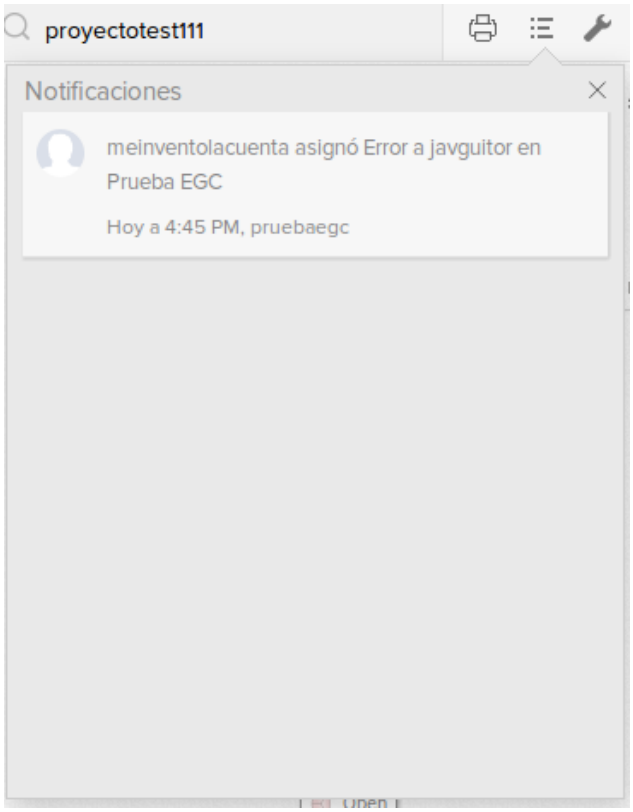


Figura 57: Notificaciones en BugTracker

En la lista de errores aparecerá este nuevo error y nos mostrará en una primera instancia la información de la tarea. Esta información puede ser configurada para que muestre lo que el usuario vea oportuno.

Figura 58: Listado de errores en BugTracker

Esta persona podrá ir modificando el estado de la incidencia para informar al resto de integrantes, el estado en que se encuentra.

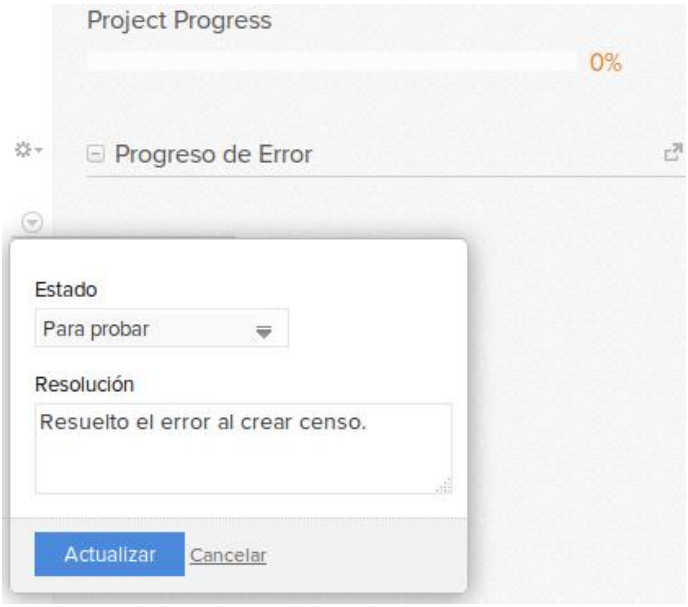


Figura 59: Estado de una incidencia en Bucgtracker

Cuando el resto de integrantes vea la notificación de que el error ha cambiado de estado se le realizarán las pruebas oportunas para comprobar que todo funciona correctamente.

En el caso en que el error no haya sido solucionado completamente se volvería a repetir dicho proceso.

Si por el contrario el error se ha solucionado, se procede al cambio del estado del error a cerrado.

Por tanto, todos los integrantes recibirán una notificación como la siguiente.

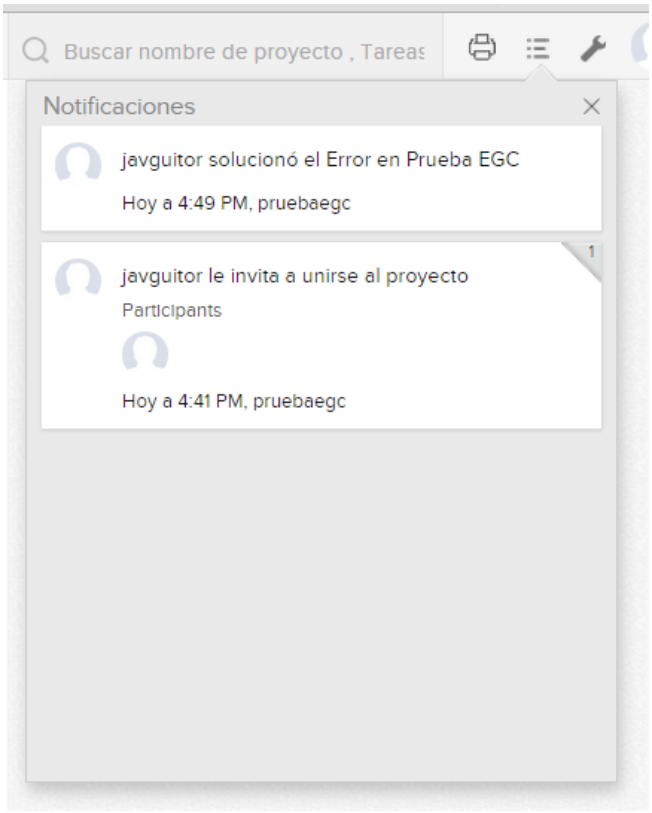


Figura 60: Notificación de error solucionado

Como podemos observar se trata de una herramienta bastante útil y sencilla a la hora de reportar errores, además de aportar una funcionalidad extra que permite realizar un seguimiento del proyecto, y comprobar en tiempo real cuántos errores existen y en qué situación se encuentran.

En esta última imagen se puede ver esta funcionalidad que hemos nombrado.

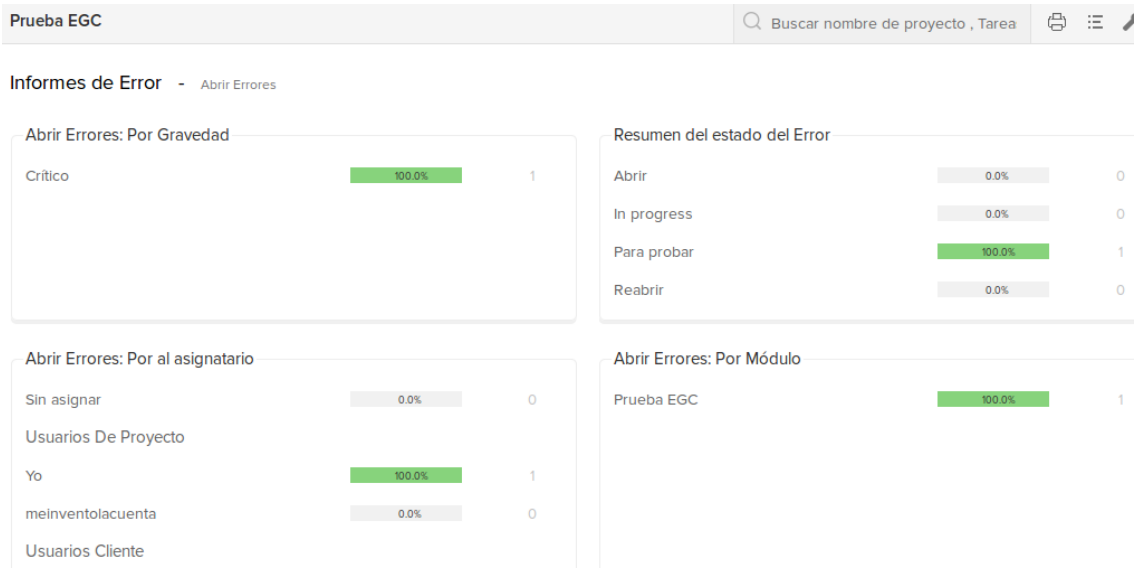


Figura 61: Informe de errores

Github:

Finalmente, se acordó utilizar como herramienta para reportar incidencias la propia de Github, ya que tenemos un repositorio compartido con los distintos subsistemas:

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

La decisión de utilizar esta herramienta se debe a la sencillez de reportar la incidencia. Es decir, los distintos subsistemas van a utilizar la herramienta de Github para poder integrarse con el resto de subsistemas, y poder así, obtener la última versión del código. Por tanto, si durante la integración hubo algún problema es más cómodo reportar dicha incidencia y poder asignarla a la o las personas involucradas en el sistema.

Para crear una nueva incidencia utilizando la funcionalidad de Github, hay que poseer una cuenta de usuario, además de tener un proyecto subido a dicho repositorio. Para ello, dentro de la rama en la cual queramos realizar la incidencia, seleccionamos la opción *Issues* en el menú lateral.

Una vez realizado esto, se nos abrirá un nuevo menú en el cual podremos introducir el nombre de la incidencia, la persona a quien se le va a asignar la incidencia,

un campo donde poner de manera detallada la descripción de la misma y por último podremos etiquetarla con una serie de categorías predefinidas.

A la persona a la que se le ha asignado la incidencia recibirá un correo electrónico. En dicho correo se le indicará donde se ha encontrado el error y qué especificaciones tiene.

Cuando la persona asignada resuelva dicha incidencia reporta su solución a la persona que creó la incidencia. Marcará por tanto la incidencia por cerrada. Cuando esto se realiza, se le envía un correo electrónico a la persona que creó la incidencia, para que en el caso en el que no esté utilizando la aplicación en el momento de la resolución, sepa que se ha resuelto.

La principal ventaja que presenta dicha herramienta para el reporte de incidencias, es que se asemeja mucho con una red social.

7.3. DEPURACIÓN

Cuando hablamos de depuración hacemos referencia al proceso de encontrar e intentar reducir y solventar los errores o defectos que el software pueda contener.

Una vez registrada y clasificada una determinada incidencia, se procede a estudiar y analizar la incidencia en cuestión, para posteriormente realizar la depuración de esta. Se comienza depurando aquellas incidencias de mayor nivel de prioridad, es decir, las que reporten un error de mayor proporción que afecte en el funcionamiento del sistema.

Como primer paso para realizar la depuración, debemos comprobar de qué se trata la incidencia, para conocer con más detalle lo reportado. Esto nos ayudará a proporcionar varias hipótesis en las que basarnos para localizar nuestro objetivo. El tratamiento a llevar a cabo, como en muchas ocasiones, dependen del tipo de error y por tanto de la naturaleza de los mismos.

Por consiguiente, una vez analizadas las posibles hipótesis, el siguiente paso es el de aislar el problema, reduciendo el espacio de búsqueda del error descartando dichas hipótesis.

Dados los diferentes métodos existentes para proceder a la depuración, podemos comenzar con intentar reproducir la incidencia en distintos escenarios, para comprobar que el problema no es del entorno de desarrollo y por tanto poder descartar esa opción.

En caso de que eso no resulte ser satisfactorio, procedemos a comprobar la traza mostrada por el sistema al ejecutar dicha excepción. La siguiente imagen muestra un ejemplo de ello:

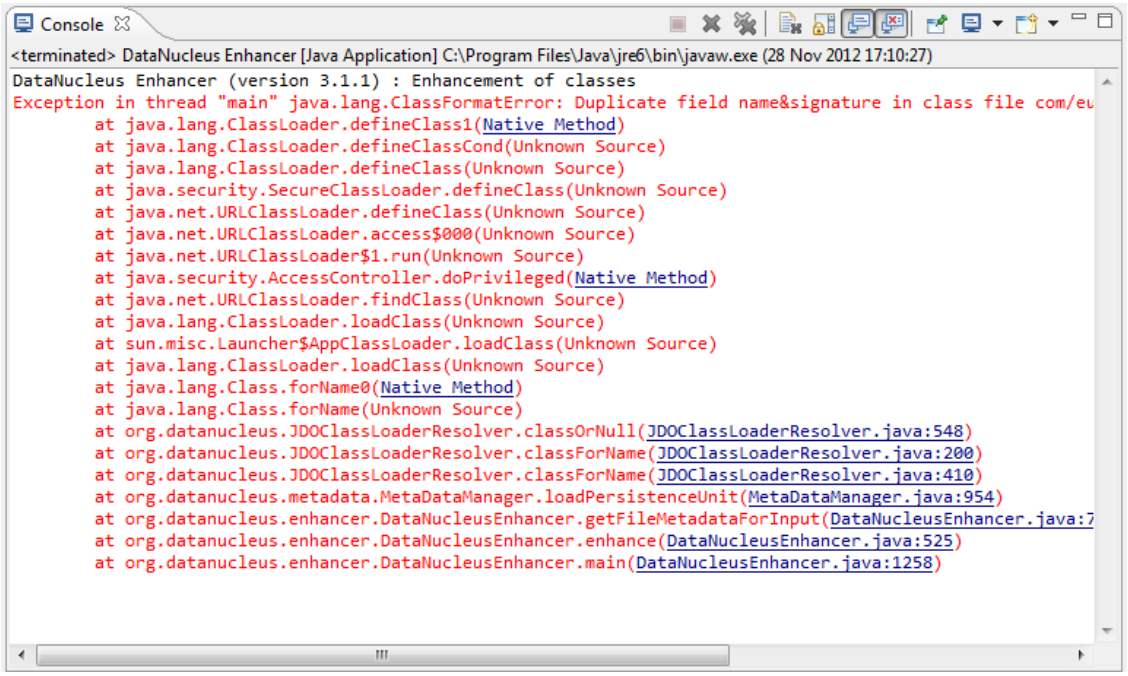


Figura 62: Ejemplo de excepción

Para ello, existen herramientas integradas en la mayoría de los entornos de desarrollo, las cuales reciben el nombre de depuradores. Los depuradores también ofrecen funciones tales como correr un programa paso a paso, es decir, pausar la ejecución del programa para examinar el estado actual en cierta instrucción especificada por medio de un punto de ruptura (*Breakpoint)* y poder visualizar los valores de las variables implicadas.

En la siguiente imagen muestra esto:

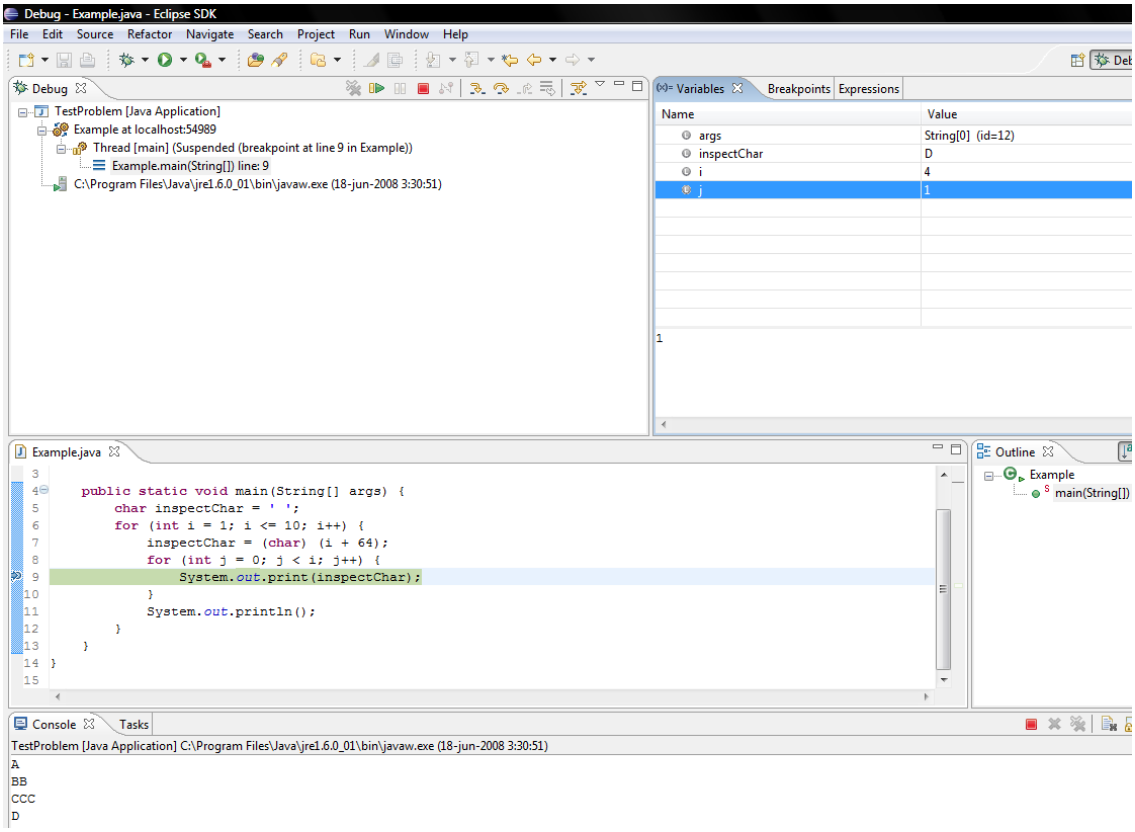


Figura 63: Modo depuración de Eclipse

De esta forma, si con una traza no es posible localizar el error, con este tipo de herramientas se puede indagar dentro del código y poder así detectar donde se encuentra el error observando los valores que van tomando las variables a lo largo de la ejecución.

Otro método de depuración es la comparación con ejercicios similares realizados de manera correcta. Es por ello que si se tiene una memoria de buenas prácticas o simplemente un proyecto similar, es posible encontrar la solución a dicho error comparando dichos proyectos.

No siempre que el propio desarrollador que ha elaborado el código, lo inspecciona en búsqueda de errores es capaz de encontrar el error, debido a que está predispuesto a que su código sea óptimo, por lo que hay casos en los que resulta de gran ayuda que sean localizados y posteriormente se proceda a solucionar los errores reportados, por desarrolladores ajenos al código en cuestión.

Una vez localizado el error, se procede al análisis de este y por consiguiente a la resolución del mismo.

Llegados a este punto, la fase de depuración queda casi concluida, ya que antes de darlo por finalizado se debería realizar una comprobación de si existen posibles errores que sean causados como consecuencia de la resolución de este, además de errores o fallos que sean similares a este que se puedan dar en un futuro.

7.4. EJERCICIO 1

"Realización de una incidencia por parte del "usuario 1" al "usuario 2" debido a un error en el fichero holamundo.py"

Resolución

Para realizar el ejercicio el "usuario 1" será javguitor quien envía la incidencia al "usuario 2", en este caso guio.fbb, adjuntándole el fichero holamundo.py, el cual contiene lo siguiente:

holamundo.py

```
1  """Este archivo tiene el fin de ser ilustrativo para la asignatura de EGC
2  12/12/2014"""
3
4  def holaMundo():
5      print ("Adios mundo")
```

Figura 64: Fichero con incidencia por resolver

El usuario guio.fbb, tras resolver la incidencia, el archivo debe quedar de la siguiente forma:

holamundo.py holamundo.py

```
1  """Este archivo tiene el fin de ser ilustrativo para la asignatura de EGC
2  12/12/2014"""
3
4  def holaMundo(nombre):
5      print ("Hola mundo. Realizado por " + nombre)
```

Figura 65: Resolución esperada de la incidencia

Javguitor realiza la incidencia en el menú lateral seleccionando "*Errores*", como se ha descrito anteriormente, este menú posee una serie de opciones las cuales nos permitirán especificar la incidencia.

La incidencia creada por javguitor le llega a guio.fbb mediante una notificación. Este usuario abre dicha notificación y comprueba el error y sus detalles como se muestra en las siguientes imágenes. Ambas aparecen en la misma vista.

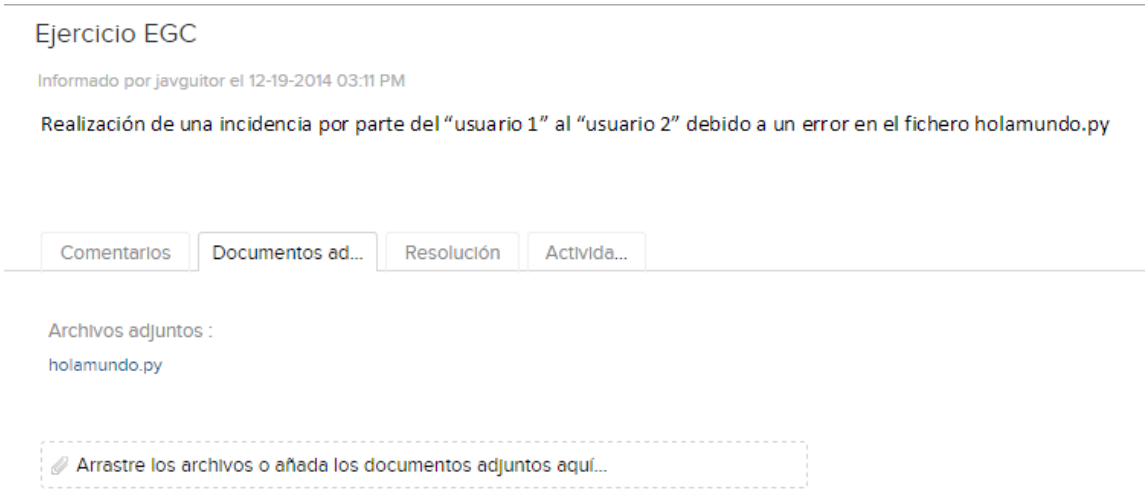


Figura 66: Incidencia por resolver

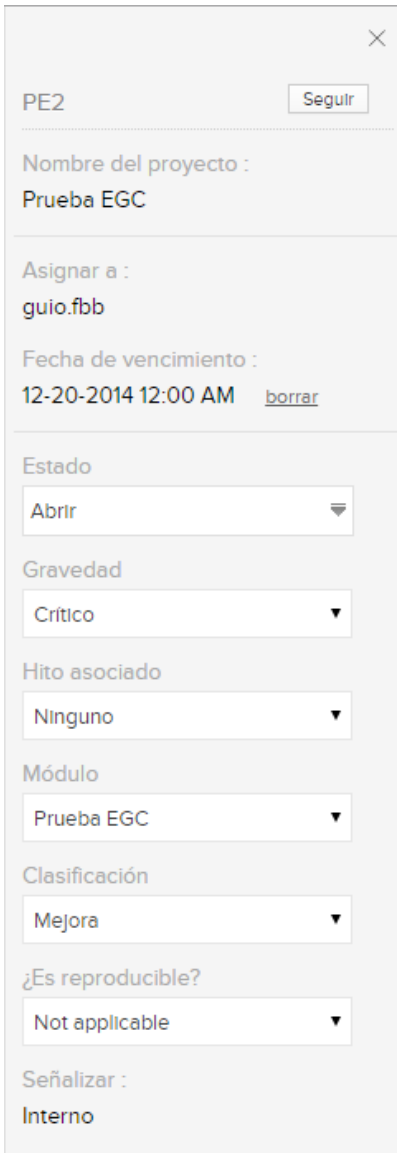


Figura 67: Detalles de la incidencia

En la imagen situada en segundo lugar se ven las especificaciones de la incidencia creada por javguitor.

El usuario guio.fbb descarga el fichero adjuntado en la incidencia para resolverla. Realiza los cambios pertinentes y actualiza el estado de la incidencia adjuntando el fichero modificado.

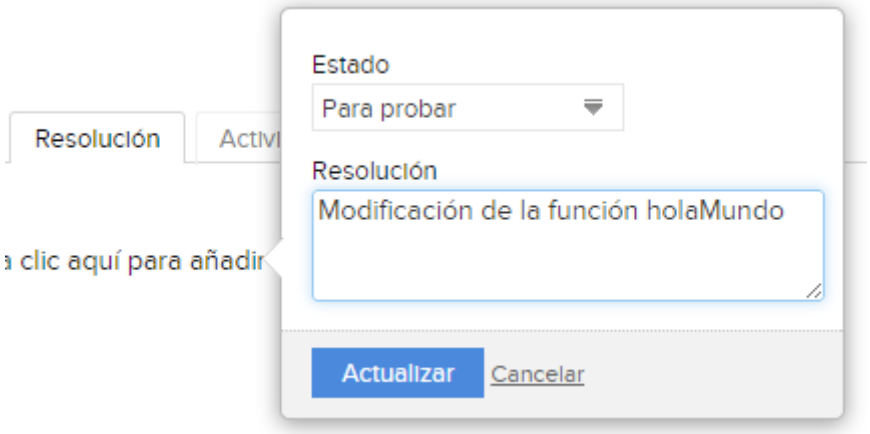


Figura 68: Cambio de estado en una incidencia

En este caso guio.fbb cambia el estado a “Para probar”. Una vez haya sido probado el nuevo fichero modificado, procederá a volver a cambiar el estado a “Cerrado”. Ya estaría resuelta la incidencia.

Los usuarios podrán comprobar el historial de cambios de la incidencia en el apartado “Actividades”. En la siguiente imagen se muestra el proceso completo realizado en el ejercicio desde que se creó la incidencia hasta que ha sido resuelta.

Figura 69: Historial de cambios en una incidencia

7.5. EJERCICIO 2

"Realización de una incidencia por parte del "usuario 1" al "usuario 2" haciendo uso de GitHub."

Resolución

La resolución de este ejercicio es un caso real. A continuación se explica cómo el usuario davalvsil crea una incidencia debido a un problema en la integración con uno de los subsistemas de Agora@US, en concreto con el subsistema Creación de votaciones.

La creación de dicha incidencia se muestra en la siguiente imagen:

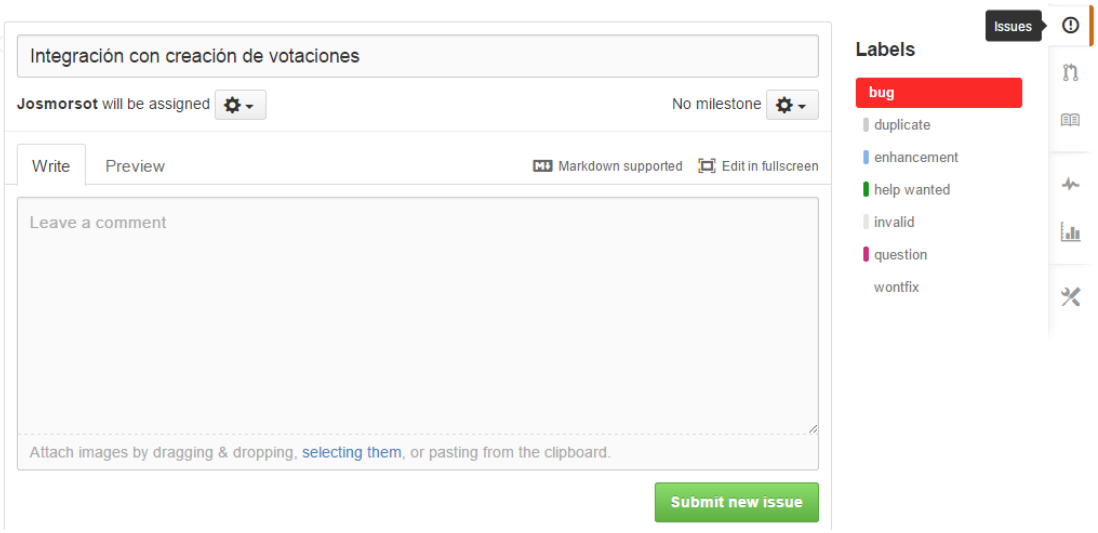


Figura 70: Creación de Issue en GitHub

Como podemos ver el título de la incidencia es "Integración con creación de votaciones", ha sido asignada al usuario Josmorsot para que la resuelva y etiquetada como "bug". Una vez creada debe quedar similar a esta imagen donde se puede ver que el estado de la incidencia es Abierto (Open). Además con un simple vistazo podemos observar los detalles de ésta.

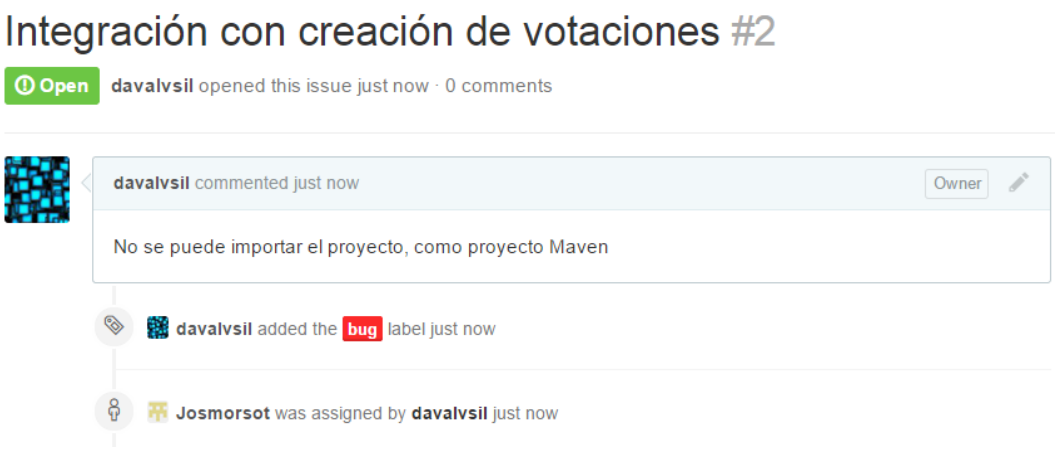


Figura 71: Detalles de incidencia

Una vez la persona asignada haya resuelto la incidencia será notificado al usuario creador (davalvsil) mediante 2 formas:

1. Notificación en la propia herramienta → Al abrirla se verá la resolución y el cambio de estado de la incidencia. En este caso al haber sido solventada completamente el estado ha pasado a Closed.

Figura 72: Notificación en GitHub de resolución

2. Notificación por correo electrónico → El usuario davalvsil ha recibido el siguiente correo electrónico con la resolución de la incidencia reportada.

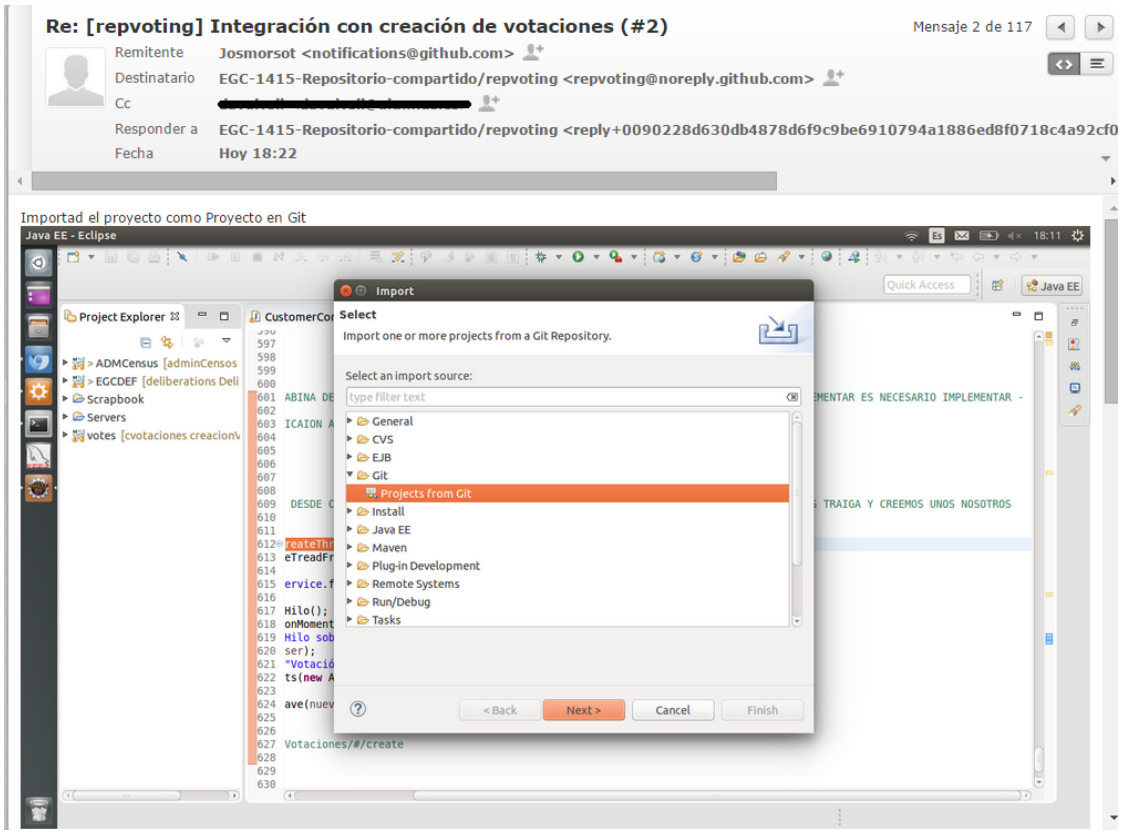


Figura 73: Notificación por email de resolución

Ya está la incidencia solventada.

Otra opción de GitHub es ver todas las incidencias registradas en el proyecto aunque no hayan sido asignadas al usuario registrado. En el caso del proyecto Agora@US estas son algunas de ellas:

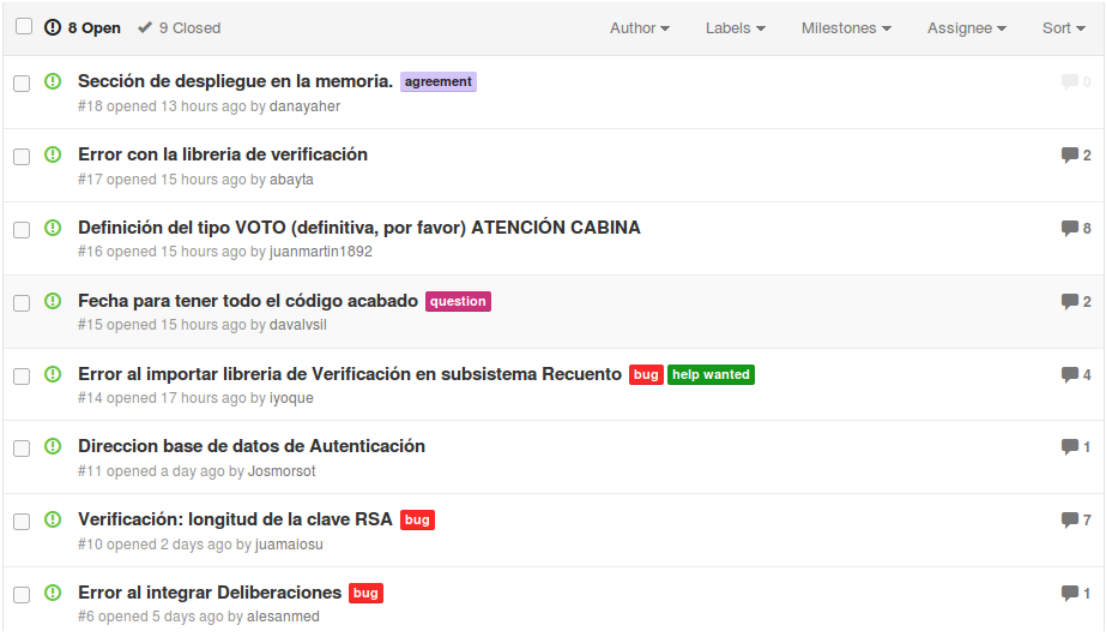


Figura 74: Registro de incidencias abiertas

8. GESTIÓN DE DESPLIEGUE

8.1. PROPUESTA DE DESPLIEGUE

La siguiente propuesta de despliegue se ha realizado en común con los miembros de los grupos de Deliberaciones, Administración de votaciones y nuestro subsistema.

Una vez que todos los subsistemas hayan finalizado su código y se hayan realizado las pruebas pertinentes de integración con todos ellos, en distintos entornos de desarrollo, al no encontrar ningún fallo, se pasa a realizar el despliegue. 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 podrán 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 la carpeta var/lib/webapps/tomcat7

Para los proyectos PHP, como Auth, se copia a la carpeta /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 Visualización de Resultados (Result_view), al no tener una base de datos, solo tendríamos que copiar su código a /opt/lamp/htdocs/ y al arrancar el Tomcat, ya podríamos acceder.

Para el proyecto Python realizado por Cabina de Votación se debe crear una estructura de carpetas como la siguiente:

- Carpeta raíz: cabina-integracion
- cabina-agora-us: dentro la raíz (será la que contenga el código del subsistema)
- Dos scripts (dentro de la raíz): 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 seria 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.

9. MAPAS DE HERRAMIENTAS

9.1. MAPA DE HERRAMIENTAS PROPIO

9.1.1. SISTEMA OPERATIVO

UBUNTU

En un principio el desarrollo iba a realizarse en Windows, pero tras hablar con los miembros de otro subsistema que realizarían el desarrollo en Ubuntu utilizando unos frameworks de este sistema operativo, se acordó desarrollar desde un primer momento en Ubuntu para evitar futuros problemas.

9.1.2. ENTORNOS

ECLIPSE

Utilizado para el desarrollo en lenguaje java con soporte para los servidores y frameworks escogidos para la realización del subsistema. Como en el anterior caso, el equipo se beneficia de una gran familiarización con el entorno y por lo tanto un mejor aprovechamiento y utilización de este.

9.1.3. LENGUAJES

JAVA

Al principio se planteó la posibilidad de hacer el desarrollo en Python, pero cuando el grupo fue consciente de los problemas que podíamos encontrarnos a la hora de realizar la integración, a la vez que de los problemas de aprendizaje de dicho lenguaje, ya que ninguno ha realizado desarrollos con cierto peso en este lenguaje, se optó por descartarlo.

La opción que se eligió fue Java, por la familiaridad que tenemos con dicho lenguaje y el conocimiento que tenemos sobre las posibilidades que tiene para poder realizar una integración. Este lenguaje es la base de la codificación y uso de los distintos frameworks, permitiéndonos el aprovechamiento de varios de estos.

9.1.4. SERVIDORES

APACHE

Servidor base para la utilización del motor de servlets de Tomcat, necesario puesto que ambos se presentan en combinación.

TOMCAT

Servidor utilizado para el despliegue del subsistema. Este despliegue se realiza en local y no en la web.

Ambos servidores se instalan bajo el sistema operativo Ubuntu y se integran con el entorno de desarrollo Eclipse.

9.1.5. BASE DE DATOS

MySQL

Este gestor de base de datos se instala sobre el sistema operativo Ubuntu y permite la creación y administración de una base de datos relacional para nuestro subsistema gracias a los frameworks utilizados.

Se decidió utilizar MySQL por la familiaridad que tenemos con dicho gestor de base de datos, a la vez que pensamos que habría menos problemas para integrar el resto de subsistemas ya que estos también utilizaban MySQL.

9.1.6. REPOSITORIOS

SVN

Esta herramienta de control de versiones utilizada en conjunto con Eclipse nos permite una sencilla gestión del código fuente de nuestro subsistema.

El servidor utilizado para alojar el repositorio es el que nos proporciona la propia escuela, ProjETSII.

9.1.7. FRAMEWORKS

SPRING

Este framework hace de soporte para el resto de herramientas utilizadas, instalado sobre el entorno de desarrollo Eclipse.

HIBERNATE

Herramienta de Mapeo objeto-relacional (ORM) utilizada para la implementación de la siguiente herramienta a describir.

JPA 2.1

Esta API nos proporciona ventajas en su utilización, como el uso del lenguaje propio JPQL, para la interacción con nuestra base de datos MySQL. Para su implementación se utiliza el framework anteriormente citado.

JSP

Basado en Java nos permite la creación de páginas web dinámicas y un mejor aprovechamiento de las herramientas anteriores. Instalado en conjunto con Spring sobre el entorno Eclipse al igual que el resto de frameworks.

9.1.8. HERRAMIENTAS DE ENTORNOS

APACHE-MAVEN

Herramienta del entorno de trabajo Eclipse que nos permite una gestión sencilla del proyecto Java ofreciéndonos un entorno unificado mediante un conjunto de plugins. En definitiva mejora el entorno de trabajo facilitando la instalación y uso de los frameworks elegidos para la realización de este proyecto.

9.1.9. REPRESENTACIÓN GRÁFICA DE LA RELACIÓN ENTRE LAS HERRAMIENTAS

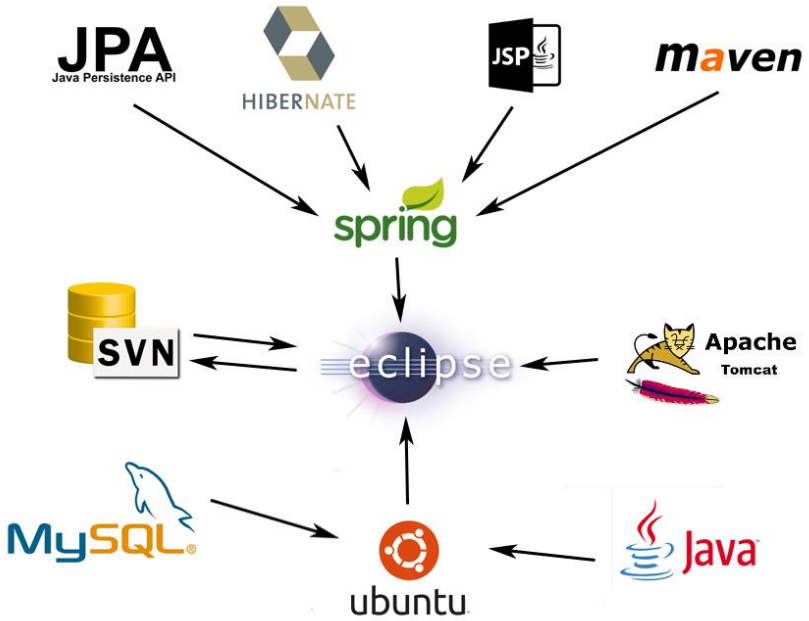


Figura 75: Mapa de herramientas propio

9.2. MAPA DE HERRAMIENTAS GENERAL

9.2.1. SISTEMAS OPERATIVOS

LINUX

Toda la integración se ha realizado bajo sistemas operativos basados en Linux, ya que soporta todas las herramientas utilizadas en sus diferentes versiones además de beneficiarnos con el uso de software libre.

9.2.2. ENTORNOS

ECLIPSE

Utilizado para el desarrollo en lenguaje java de los distintos subsistemas con soporte para los servidores y frameworks escogidos por estos.

PYTHON VIRTUALENV

Herramienta para crear entornos de trabajo independientes. Sobre este funcionan todos los frameworks relacionados con el lenguaje Python.

9.2.3. LENGUAJES

JAVA

Este lenguaje es la base de la codificación y uso de los distintos frameworks, permitiéndonos el aprovechamiento de varios de estos.

HTML/PHP

Utilizados por varios subsistemas para la codificación. Se hacen uso de APIs y json para la comunicación con el resto de lenguajes.

PYTHON

Utilizado por un subsistema junto a un framework específico para este lenguaje.

9.2.4. SERVIDORES

APACHE

Servidor base para la utilización del motor de servlets de Tomcat, necesario puesto que ambos se presentan en combinación.

TOMCAT

Servidor utilizado para el despliegue del subsistema. La razón de que haya dos es que el que está relacionado con XAMPP se utiliza para el despliegue local, el instalado sobre el SO se utiliza para despliegue en la web.

XAMPP

Servidor utilizado para el despliegue de los subsistemas que utilizan PHP. Despliegue en local.

GUNICORN

Servidor para el despliegue local de los subsistemas basados en Python. Compatible con todos los frameworks utilizados y ligero en recursos.

9.2.5. BASE DE DATOS

MySQL

Este gestor de base de datos se instala sobre el sistema operativo y se relaciona directamente con el servidor XAMPP y permite la creación y administración de una base de datos relacional para nuestro subsistema gracias a los frameworks utilizados.

9.2.6. REPOSITORIOS

SVN

Esta herramienta de control de versiones utilizada como plugin en conjunto con Eclipse nos permite una sencilla gestión del código fuente de nuestro subsistema.

El servidor utilizado para alojar el repositorio es el que nos proporciona la propia escuela, ProjETSII.

GIT/GIT HUB

Herramienta de control de versiones utilizada por distintos subsistemas para la gestión del código fuente. Además de instalarse sobre el sistema operativo se utiliza un plugin instalado sobre el entorno de desarrollo eclipse.

9.2.7. FRAMEWORKS

SPRING

Este framework hace de soporte para el resto de herramientas utilizadas, instalado sobre el entorno de desarrollo Eclipse.

HIBERNATE

Herramienta de Mapeo objeto-relacional (ORM) utilizada para la implementación de la siguiente herramienta a describir.

JPA 2.1

Esta API nos proporciona ventajas en su utilización, como el uso del lenguaje propio JPQL, para la interacción con nuestra base de datos MySQL. Para su implementación se utiliza el framework anteriormente citado.

JSP

Basado en Java nos permite la creación de páginas web dinámicas y un mejor aprovechamiento de las herramientas anteriores. Instalado en conjunto con Spring sobre el entorno Eclipse al igual que el resto de frameworks.

DJANGO

Framework utilizado en los subsistemas desarrollados en Python que permite un desarrollo rápido y limpio, no hay una comunicación directa con el resto de Frameworks.

REQUESTS

Librería HTTP licenciada de Apache2, simplifica y mejora la librería HTTP propia de Python.

DJANGO REST FRAMEWORK

Conjunto de herramientas para la creación de web API's de forma sencilla y rápida.

RSA

Framework para la encriptación y desencriptación, login y verificación de firmas digitales; y generación de claves.

9.2.8. HERRAMIENTAS DE ENTORNO

APACHE-MAVEN

Herramienta del entorno de trabajo Eclipse que nos permite una gestión sencilla del proyecto Java ofreciéndonos un entorno unificado mediante un conjunto de plugins. En definitiva mejora el entorno de trabajo facilitando enormemente la instalación y uso de los frameworks (salvo Django) elegidos para la realización de este proyecto.

9.2.9. INTEGRACIÓN CONTINUA

JENKINS

Servidor de integración continua utilizado para la integración de los diversos subsistemas del proyecto, entre algunas de las funcionalidades que se han utilizado cabe destacar el uso del plugin de Git, ejecución de scripts y despliegue automatizado. Dicha herramienta ha sido instalada sobre el sistema operativo Ubuntu.

9.2.10. REPRESENTACIÓN GRÁFICA DE LA RELACIÓN ENTRE LAS HERRAMIENTAS

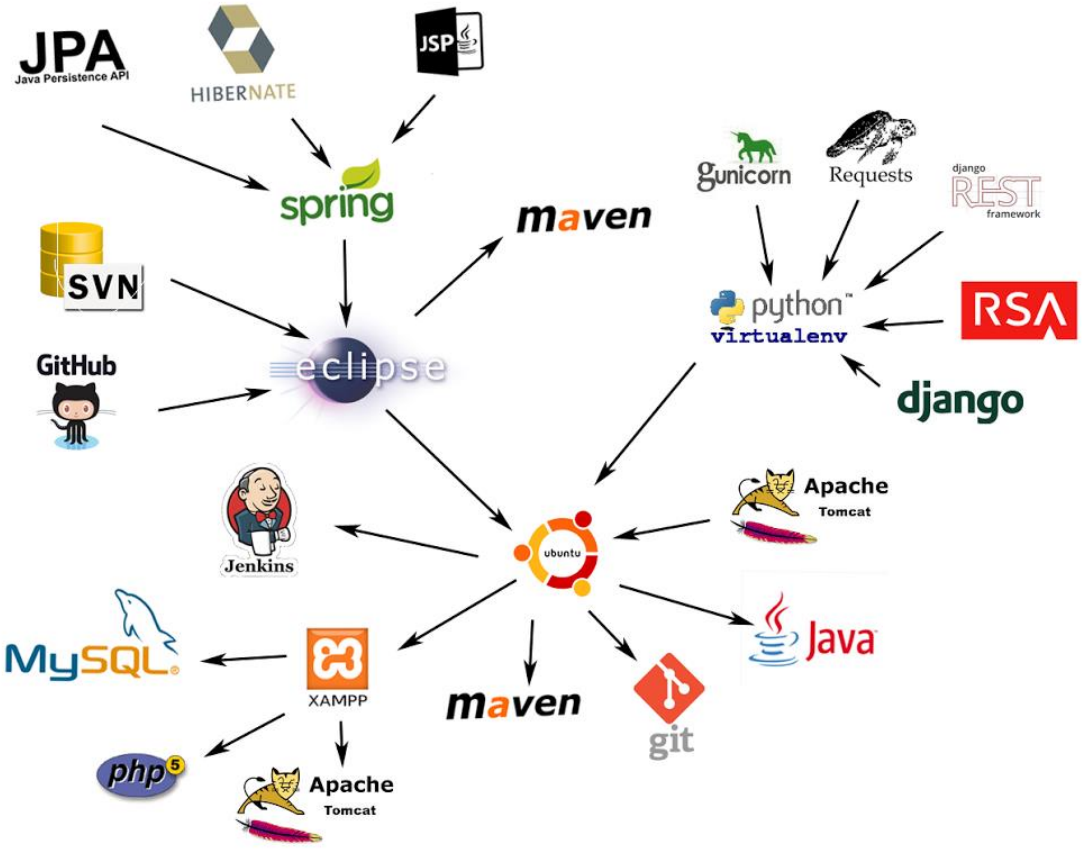


Figura 76: Mapa de herramientas general

10. CONCLUSIONES

A través del proyecto propuesto en la asignatura hemos podido llevar a la práctica tanto los conceptos vistos en la teoría de Evolución y Gestión de la Configuración, como de asignaturas anteriores. Enfrentándonos a problemas reales y dándoles solución por nuestros propios medios, en la medida de lo posible.

Al tener que hacer un proyecto, en el que distintos subsistemas debían integrarse y estando cada uno de ellos realizados de formas distintas, hemos tenido que aprender a usar herramientas, que para algunos de nosotros eran desconocidas, como Git para poder manejar un repositorio compartido o Jenkins para poder realizar la integración.

Por último decir que aunque la comunicación interna de nuestro grupo se ha realizado de una forma fluida, han surgido muchos problemas a la hora de comunicarnos y poner en común con otros grupos algunas soluciones o problemas. No solo debido a la falta de comunicación, sino también a la falta de participación o interés de algunos miembros de otros grupos, debido a estos problemas ha habido cambios hasta el último momento repercutiendo en otros grupos que sí ponían de su parte.

BIBLIOGRAFÍA

- **Integración**
  - http://tratandodeentenderlo.blogspot.com.es/2009/09/integracion-continua.html
  - http://emanchado.github.io/camino-mejor-programador/html/ch08.html

- **Gestión del código fuente**
  - https://ariejan.net/2007/07/03/how-to-create-and-apply-a-patch-with-subversion/

- **Gestión del cambio, incidencias y depuración**
  - https://www.zoho.com/bugtracker/
  - http://es.wikipedia.org/wiki/Depurador
  - http://itilv3.osiatis.es/operacion_servicios_TI/gestion_incidencias/registro_clasificacion.php
  - http://www.fing.edu.uy/inco/cursos/ingsoft/pis/proceso/MUM/treebrowses/disciplinas/gestionconf/indexGestionConf.htm

- **En todos los apartados**
  - Diapositivas de la asignatura

- **Trabajos consultados de otros años**
  - Grupo 1 - Chromium OS
  - Grupo 5 - Django Framework
  - Grupo 4 - Gestión de la configuración de AOSP

GLOSARIO DE TÉRMINOS

TÉRMINO	*DEFINICIÓN*
API	Conjunto de rutinas, protocolos y herramientas para la construcción de aplicaciones de software.
AUTOMATIZACIÓN	Ejecución de procesos con la mínima (o ninguna) intervención.
BASE DE DATOS	Conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.
BASELINE	Línea base para la construcción de una aplicación software.
BRANCH	División del repositorio para la realización de cambios en el código sin afectar a la rama principal o Trunk.
CENSO	Recuento de la población que tiene derecho a voto en unas determinadas elecciones.
COMMIT	Acción en un sistema de control de versiones para enviar cambios realizados en el código (ya sea en local o a servidor).
DESPLIEGUE	Consiste en poner el software en un entorno accesible, de manera que el cliente pueda acceder.
FRAMEWORK	Entorno software que proporciona una funcionalidad particular como parte de una plataforma para el desarrollo de aplicaciones, productos y soluciones software.
INCIDENCIA	Error detectado y documentado en el desarrollo de una aplicación software.
LIBRERÍA	Conjunto de soluciones software utilizadas para el desarrollo simplificado de aplicaciones.

PLUGIN	Aplicación que, en un programa informático, añade una funcionalidad adicional o una nueva característica al software.
PUSH	Comando utilizado para subir cambios a un repositorio remoto.
MÁQUINA VIRTUAL	Software para ejecutar un SO externo con distintas herramientas en una máquina con un SO anfitrión.
REPOSITORIO	Depósito o archivo centralizado donde se almacena y mantiene información digital, habitualmente bases de datos o archivos informáticos.
SCRIPT	Programa simple usualmente almacenado en un archivo de texto plano para la realización de distintas tareas pequeñas dentro del contexto de una aplicación software.
TAG	Puntos de guardado en momentos importantes del desarrollo de un proyecto en un sistema de control de versiones.
TRUNK	Rama principal del sistema de control de versiones donde se encuentra el código principal de una aplicación.
WAR	JAR utilizado para distribuir una colección de JavaServer Pages, servlets, clases Java, archivos XML, librerías de tags y páginas web estáticas (HTML y archivos relacionados) que juntos constituyen una aplicación web.

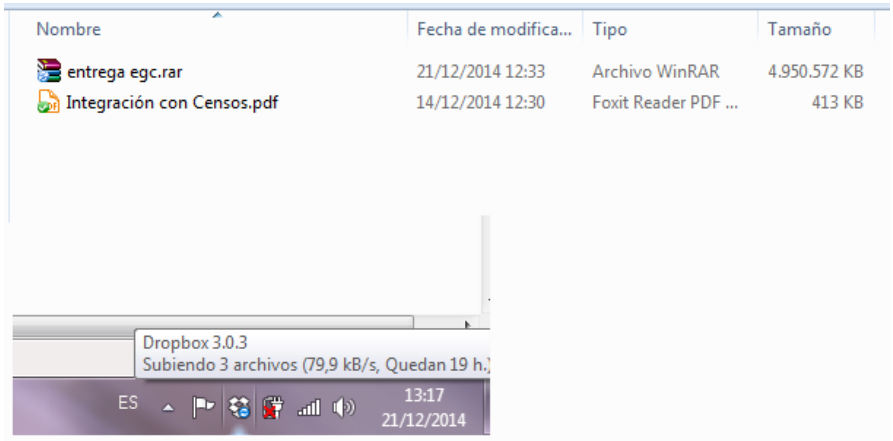
ANEXOS

INSTRUCCIONES MÁQUINA VIRTUAL

Descarga de la máquina virtual

La máquina virtual aportada es Ubuntu 14.04.

Debido al peso que tiene dicha máquina (cerca de 5GB) a la hora de elaborar este documento no se ha conseguido subir a Internet, como demuestra la siguiente imagen.



El enlace en el que se está subiendo dicha máquina es el siguiente:

https://www.dropbox.com/sh/bu2wibsjbchbmi6/AADWvu3TgjnUUl0HkXw1A6FYa?dl=0

Usuario y contraseña del sistema

Usuario: egc

Contraseña: egc

Importante (antes de desarrollo)

Hay dos instancias de Tomcat en la misma máquina, uno para desarrollo y otro para despliegue, ambos usan el mismo puerto, por tanto antes de arrancar el eclipse para desarrollar, debemos iniciar un terminal y poner el siguiente comando:

sudo service tomcat7 stop

Contraseña: egc

Una vez hecho esto, abrimos eclipse y realizamos las modificaciones oportunas en el código.

Probar desarrollo

Para poder probar el desarrollo realizado en nuestro subsistema, debemos iniciar XAMPP con el comando que se indica a continuación en un terminal:

sudo /opt/lampp/lampp start

Tras arrancar XAMPP abrimos eclipse (si no lo tenemos abierto) y arrancamos Tomcat.

Una vez arrancados ambos servidores (XAMPP y Tomcat), abrimos Chrome y pulsamos sobre el marcador "agora@us" que nos llevará a la página de login.

Usuario: danayaher

Contraseña: danayaher1

Ruta de carpetas

Ruta del Eclipse: /home/egc/eclipse/

Ruta de Cabina de votación: /home/egc/

Ruta de proyectos PHP: /opt/lampp/htdocs/

Más información

Para más información consultar el archivo .txt situado en el escritorio de la máquina virtual.