AGORA@US

Grupo 9: Creación y Administración de Censos

6 DE ENERO DE 2017

Pablo Romero Vázquez

Santiago Fraga Martín-Arroyo

Nicolás Lorenz Rosado

Simón Egea Guerrero

Rubén Barrientos Mohedano

Enlaces de interés

Repositorio de Github del grupo “Creación y Administración de Censos”:

https://github.com/Trabajo-EGC/Censo

Repositorio de Github de incidencias común:

https://github.com/Trabajo-EGC/Censo/issues

Wiki de la asignatura:

https://1984.lsi.us.es/wiki-egc/index.php/P%C3%A1gina_Principal

Espacio del grupo “Creación y Administración de Censos” en la wiki:

https://1984.lsi.us.es/wiki-egc/index.php/CreacionAdministracionCensos

Espacio del grupo “Creación y Administración de Censos” en Opera:

http://opera.eii.us.es/egc/public/trabajo/ver/id/52

Servidor ftp donde se puede descargar la máquina virtual utilizada (DP):

ftp://postgrado.lsi.us.es/DT/

Integrantes

Santiago Fraga Martín-Arroyo	Pablo Romero Vázquez	Simón Egea Guerrero
Ingeniero Software	Ingeniero Software	Coordinador
		
Nicolás Lorenz Rosado	**Rubén Barrientos Mohedano**	
Ingeniero Software	Ingeniero Software	
		

Historial de versiones

Versión	Autor	Descripción	Fecha
1.0	Pablo Romero Vázquez	Esquema de la documentación y realización de la misma.	08/01/2017
1.1	Santiago Fraga Martín-Arroyo	Realización y corrección de la documentación.	10/01/2017
1.1	Santiago Fraga Martín-Arroyo	Revisión	10/01/2017

Contenido

Resumen

En este proyecto, realizado para la asignatura “Evolución y Gestión de la Configuración”, se va trabajar con código heredado del curso anterior.

La finalidad de este proyecto es mostrar la dificultad que conlleva trabajar con código heredado y los problemas que trae cuando se intenta complementar con otros proyectos independientes.

El coordinador del equipo será Simón Egea Guerrero y el subsistema a desarrollar o mejorar es el de “Creación y Administración de Censos”. Dicho subsistema almacena la información correspondiente a los votantes pertenecientes a un censo asociado a una determinada votación. Para ello, el sistema debe complementarse con otros subsistemas como el de “Autenticación” o el de “Creación y Administración de Votaciones”.

Para el despliegue, los coordinadores de los diferentes equipos se reunieron y llegaron a un acuerdo en la forma en la que se iba a trabajar.

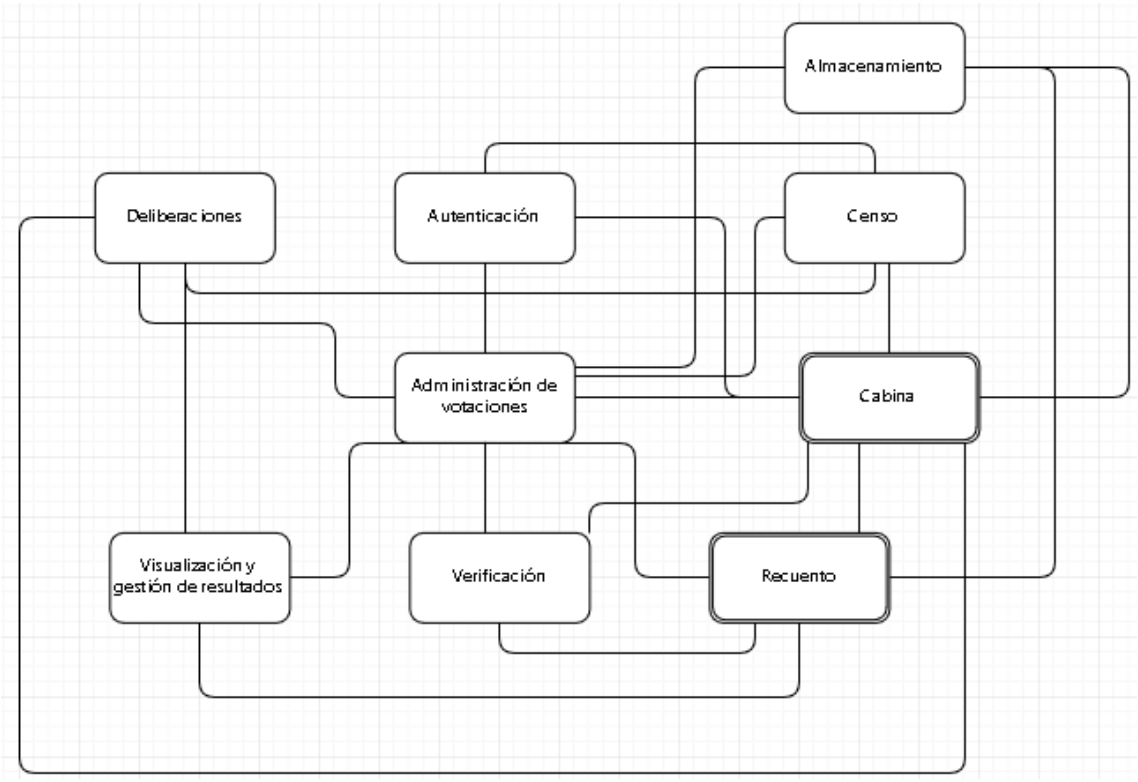
Introducción

El proyecto a desarrollar es “Agora@us”, una herramienta software que gestiona las votaciones realizadas por internet. Este proyecto está dividido en varios subsistemas debido a la complejidad del mismo. Los subsistemas son:

- Almacenamiento de votos: contiene los votos de una votación.
- Autenticación: registra a los usuarios del sistema.
- Cabina de votación: permite el voto de los usuarios.
- Creación/Administración de censos: asocia la votación con un censo.
- Creación/Administración de votaciones: crea una votación.
- Deliberaciones: Almacena comentarios emitidos por los votantes.
- Recuento y modificación de resultados: cuenta los votos de una votación.
- Verificación: verifica la autenticidad de los votos.
- Visualización y Gestión de resultados.

Hay que señalar que en el grupo de tarde no existen tantos grupos para cubrir todos los subsistemas, por lo que los subsistemas de “Autenticación” y “Deliberaciones” no han podido ser desarrollados o mejorados por ningún grupo de este año y se han tenido que usar los del año anterior.

La estructura del sistema Agora@us quedaría reflejada en la siguiente figura:



Nuestro equipo será el encargado de hacer el subsistema de “Creación y Administración de Censos”. En esta nueva versión, se han añadido nuevas funcionalidades:

Para poner en marcha el subsistema hemos hecho uso de una máquina virtual que incorpora Java, Spring, Git, Jenkins, MySql y Tomcat 7.

Descripción del sistema.

Componentes y relación con el resto de subsistemas.

Esta sección se ha mantenido igual que en el documento del grupo anterior al no haber modificaciones en el diseño del año pasado, solo mejoras.

El subsistema de Creación y Administración de Censos tiene como funcionalidad principal recoger información de una serie de votaciones que nos llegarán a través del subsistema de Creación y Administración de Votaciones para poder asignarle a cada votación un determinado censo que recoja a los participantes/votantes de la misma.

Para conseguir el correcto funcionamiento, nuestro subsistema cuenta con varios métodos auxiliares que permiten a su vez enviar los censos creados a otros subsistemas que consumen nuestra información.

Nuestro proyecto está compuesto por una serie de paquetes y carpetas que se detallan a continuación:

1. **src/main/java**
   Es una carpeta compuesta de una serie de paquetes que serán el foco de la programación, desde donde se creará un sistema de información de abajo a arriba.

   Los paquetes que nos encontramos son los siguientes:
   - controllers, donde crearemos las clases destinadas a realizar las funciones CRUD definidas en el paquete Services y otros métodos o reglas de negocio.
   - converters, compuesto de clases donde especificaremos cómo transformar los objetos java a string y viceversa.
   - domain, donde definiremos nuestro modelo de dominio creando tantas clases como entidades tenga el mismo.
   - repositories, formado por clases en las cuales se definirán las distintas consultas a realizar a la Base de Datos de nuestro sistema.
   - security, donde definiremos las clases que gestionarán el login del sistema.
   - services, compuesto de clases donde se programarán los métodos CRUD y otros métodos definidos por las reglas de negocio oportunas haciendo uso del paquete Repositories y que serán llamados en las clases del paquete Controllers.
   - utilities, es un paquete formado por algunas clases cuya función es gestionar la conexión y la configuración con la Base de Datos.
   - utilities.intenal, ídem que el paquete definido arriba con alguna funcionalidad distinta.

2. **src/main/resources**
   Esta carpeta está compuesta de algunas subcarpetas y clases destinadas a la configuración de nuestro sistema.
   Destacamos algunas de mayor importancia dado que deben ser gestionadas correctamente a la hora de crear nuestro subsistema:

   - converters.xml, donde introduciremos un listado de todos los convertidores que creemos para que Spring los localice.
   - i18n-l10n.xml, clase destinada a la internacionalización donde deberemos indicar la respectiva traducción de las entidades que creemos.
   - security.xml, en esta clase especificaremos quién tiene permisos a los distintos recursos que creemos.
   - tiles.xml, clase donde se especificará la existencia de enlaces en una determinada vista.
   - PopulateDatabase.xml, documento destinado a la creación de entidades en un sistema y la relación entre las mismas, persistidas en la Base de Datos.

3. **src/test/java**
   Carpeta destinada a incluir clases con funciones para realizar tests unitarios a los métodos realizados dentro del paquete Service y comprobar su correcto funcionamiento.

   También se incluyen librerías externas al proyecto. Estas librerías son un conjunto de clases con métodos y atributos que podremos utilizar en las distintas clases de nuestro proyecto. Además, también existen librerías, como Maven Dependencies, que se encargan de la gestión de las dependencias dentro del proyecto.

4. **bin**
   Carpeta que contiene las mismas carpetas definidas arriba (exceptuando las carpetas de tests) y además, una nueva carpeta llamada webapp donde especificaremos y crearemos las distintas imágenes, scripts, estilos, y vistas de nuestro proyecto.

5. **src**
   Comprende a las carpetas main y tests descritas anteriormente.

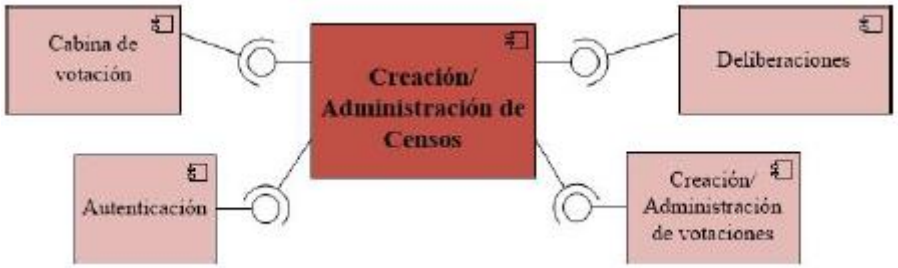
6. **target**
   Es una carpeta que contiene un archivo denominado MANIFEST.MF. Se usa para definir datos relativos a la extensión y al paquete.

   Por último, encontramos el archivo pom.xml. Este fichero es la herramienta principal de un Proyecto Maven. Contiene información acerca del proyecto, de los plugins que instalemos, de versiones, de dependencias… Además, antes de ejecutarse el proyecto, siempre se visita este fichero.

Cuando heredamos este proyecto verificamos la existencia de todas las carpetas y paquetes mencionados anteriormente. No obstante, comprobamos que algunas clases no se encontraban en los paquetes y carpetas recomendados para un proyecto Maven. Por ejemplo, la clase de tests unitarios no tenía el nombre correcto, ni los métodos correctos, ni la ubicación correcta. Se ha tenido en cuenta como mejora la reordenación del código.

Por otra parte, este proyecto puede resultar difícil de entender cuando se hereda dado que no dispone de comentarios suficientes en los métodos realizados. Esto se ha tenido en cuenta como mejora, con el fin de facilitar la comprensión del código a los alumnos que hereden este Proyecto el próximo curso.

En el siguiente diagrama se muestran las relaciones que hay entre nuestro subsistema y el resto de subsistemas.



Como se puede comprobar nuestro subsistema consume de autenticación, ya que necesitamos los usuarios autenticados/logados en el sistema para poder introducirlos en los censos. También consume del subsistema Creación y Administración de votaciones, ya que aparte de los usuarios autenticados también necesitamos tener las votaciones a las que añadir los participantes/votantes. Estos componentes que acabamos de nombrar son precisamente los que se meten por parámetros en la mayoría de métodos de nuestro sistema para devolver el censo deseado.

Por otra parte, de nuestro subsistema consumen los módulos de Cabina de Votación y Deliberaciones. Cabina de votación una vez tenga las votaciones y los censos desarrolla el espacio donde el cliente pueda realizar la votación. Por su parte, el módulo de Deliberaciones consiste en una base de datos que almacena comentarios con información relevante sobre los votantes.

Cambios

El proyecto del año pasado está desarrollado usando el IDE Eclipse y usa las tecnologías de la asignatura Diseño y Pruebas (Spring, MySql, Maven, Hibernate y Tomcat 7). Para la integración continua se usó Jenkins.

En nuestro caso, hemos trabajo en el subsistema usando la máquina virtual de Diseño y Pruebas (que contiene las tecnologías mencionadas anteriormente) junto con Git para acceder al repositorio el cuál se encuentra en los servidores de Github.

Además de las tecnologías anteriormente nombradas se ha utilizado Jenkins para la realización de la integración continua, con el fin de automatizar tareas.

Los cambios que se han introducido en el sub-sistema heredado son los siguientes:

1. Mostrar en una tabla las últimas votaciones que se han realizado en el sub-sistema.
2. Modificación de la fecha a la hora de guardar los censos en la base de datos a que la mayoría habían expirado.
3. Se ha añadido una nueva dependencia en el archivo "pom.xml" con WireMock.

   WireMock es un simulador de API basado en HTTP. (http://wiremock.org/)

4. Se ha añadido los plugins de pmd y doap.
   PMD te da consejos de como mejorar tu codigo con buenas practicas. Este plugin se ejecutara cada vez que se inicia la fase test de nuestro proyecto.
   https://maven.apache.org/plugins/maven-pmd-plugin/
   Doap te genera un documento a partir del "pom.xml" que tiene el proyecto.
   https://maven.apache.org/plugins/maven-doap-plugin/
5. Se han realizado los servicios y las queries necesarias para que se puedan obtener los censos más participativos.
6. Se ha creado una nueva funcionalidad que permite conocer el porcentaje de usuarios que no votan para cada censo.
7. Se ha mejorado el css ya que se ha considerado que era pobre.
8. Se ha añadido un plugin al pom.xml del proyecto para generar el empaquetado del proyecto. Además, se ha añadido varias líneas de código en el pom.xml para la realización del deploy.
9. Se ha añadido un pequeño documento de ayuda que explica detalladamente todos los métodos de los que dispone el sub-sistema.
10. Se han realizado Tests para comprobar que todas las nuevas funcionalidades que han sido creadas funcionan correctamente.
11. Se ha implementado un buscador de censos, el cuál buscara mediante una palabra clave todos los censos cuyo título coincida total o parcialmente con dicha palabra.

Entorno de desarrollo.

La máquina virtual usada por el grupo es la misma que se usó para cursar la asignatura Diseño y Pruebas. Dicha máquina cuenta con el siguiente software:

- Java 7
- MySQL
- Spring Tool Suite 3.7.1
- Tomcat 7
- Hibernate

Instalación del entorno de desarrollo

Para la instalación de todos los componentes necesarios para la ejecución correcta del proyecto hemos seguido los siguientes pasos:

1. Lo primero que debemos hacer es descargar nuestro software de virtualización (VirtualBox) y la máquina virtual. Para ello, debemos de acceder a la siguiente dirección: ftp://postgrado.lsi.us.es/DT/ (Actualmente la página esta fuera de servicio). La máquina virtual que los profesores de la asignatura Diseño y Pruebas nos proporciona tiene todo lo necesario para una ejecución mínima del sub-sistema heredado.
2. Una vez ejecutada la máquina virtual debemos de usar el Workspace que una vez más nos proporciona los profesores de la asignatura Diseño y Pruebas. Dicho Workspace viene con una configuración inicial que es suficiente para ejecutar el sub-sistema heredado.
3. Antes de poder ejecutar el sub-sistema debemos de configurar el JRE (JDK) que va a usar nuestro eclipse, para ello nos vamos a la pestaña de “Window” -> ”Preferences” -> “Installed JREs”.
4. Ahora debemos de configurar nuestro servidor de Tomcat, para ello nos iremos a la pestaña de “Window” -> ”Preferences” -> “Server” -> “Runtime Environmente”, desde esta ventana seleccionaremos “Apache Tomcat v7.0” y dejaremos todo lo demás tal y como esta. Una vez hecho esto le daremos doble click a “Tomcat v7.0” que está situado en la ventana de “Servers” y cambiaremos el puerto de 80 a 8080.
5. Ahora importaremos el proyecto heredado, para ello iremos a la pestaña de “Import” -> “Maven” -> “Existing Maven Projects” una vez en esta ventana seleccionaremos el proyecto y lo importaremos.
6. Ahora iniciaremos la base de datos de Mysql que es donde almacenaremos los datos necesarios. Lo primero de todo es iniciar el Mysql Workbench como “Boss” (la contraseña de dicho usuario viene junto con la maquina virtual). Posteriormente debemos de ejecutar los siguientes comandos:

```
drop database if exists `Sample`;
create database `Sample`;


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

Donde pone “Sample” se debe de poner el nombre que tiene nuestro proyecto.

7. Ahora ejecutaremos la clase “PopulateDatabase.java” que se encontrará en el proyecto que hemos importado.
8. A continuación, añadimos nuestro proyecto al Tomcat de Eclipse con el botón derecho en “Tomcat v7.0” situado en la pestaña de “Servers” y seleccionamos “Add and Remove… ”. Por último, pasamos nuestro proyecto (que debe de estar en “Available”) a “Configured” y le daremos a “Finish”.
9. Una vez hecho esto, damos click derecho sobre “Tomcat v7.0” situado en la pestaña “Servers” y seleccionamos la opción “Start”.
10. Ahora solo queda escribir en nuestro navegador http://localhost:8080/Sample/ (en simple escribiremos el nombre de nuestro proyecto).

Gestión del código fuente.

Para gestionar el código y las versiones se ha usado Git junto con el repositorio de Github, al ser la herramienta sobre la que se asienta gran parte de la asignatura. El repositorio de nuestro proyecto es el siguiente: https://github.com/Trabajo-EGC/Censo

Los comandos usados han sido los siguientes. Cabe destacar que han sido los mismos que usaron el grupo del año pasado, como viene recogido en su documentación:

- Para hacer un commit: “git commit -a”.
- Para hacer un pull: “git pull”.
- Cambiar a una nueva rama: “git branch nombre_rama”.
- Cambiar a una rama ya creada: “git checkout rama_objetivo”.
- Merge: git add “nombre_del_archivo”.
- Mostrar el historial: “git log”.

Adicionalmente se usarón los siguientes comandos:

- Para unir ramas: “git merge nombre_rama”
- Ver los commits de la rama: “git branch -v”
- Para iniciar un repositorio local: “git init”
- Para añadir los cambios a nuestro repositorio local: “git add *”
- Para traernos todo el repositorio de github: “git clone ruta_github”
- Para realizar un commit en nuestro repositorio local y añadirle un comentario: “git commit -m ”Comentario” ”

Sintaxis del código fuente.

Sobre la sintaxis, no hemos variado la metodología del grupo anterior, por lo que seguimos usando los estándares de Java, usando el estilo “UpperCamelCase” para los nombres de las clases, y “lowerCamelCase” para los nombres de los métodos y las variables. Los nombres de los repositorios, servicios y controladores seguirán el patrón “NombreEntidad [Repository | Service | Controller]” respectivamente.

Resolución de conflictos.

Cuando se produce algún tipo de conflicto a la hora de lleva a cabo alguna tarea los pasos que se deben de seguir para que dicho problema sea resuelto satisfactoriamente son los siguientes:

1. Se debe de crear un “issue” con un título descriptivo y una pequeña explicación del problema que ha surgido. Si se ve adecuado se debe de poner los pasos que se deben de seguir para reproducir el error en otra máquina distinta.
2. El “issue” debe de ser etiquetado con un “Label” adecuado, dependiendo de si es un “bug” o un una “propuesta” para la realización de algún tipo de cambio que pueda mejorar el sistema.
3. Cuando algún miembro del equipo comience a trabajar en esta incidencia esta se deberá de etiquetar con el “Label” “En curso” para así llevar un control de las incidencias en las que se están trabajando.
4. Cuando se ha solucionado la incidencia el “issue” es etiquetado como “Revisar” para que otro miembro (distinto al que la ha solucionado) revise si efectivamente se ha corregido correctamente la incidencia. Si la incidencia no estuviera correctamente solucionada se regresaría al paso 2 y se dejaría solamente las etiquetas de “bug”.
5. Si la incidencia esta solucionada correctamente el “issue” es etiquetado como “Cerrada” y pasaría a estado de “Closed”.

Lecciones aprendidas.

Durante el desarrollo de este sistema las lecciones aprendidas generadas han sido las siguientes:

- Se ha aprendido a utilizar Git junto con Github ya que ningún miembro del grupo sabía usar estas tecnologías antes de cursar la asignatura.
- Se han obtenido los conocimientos necesarios para usar la herramienta Maven para realizar un despliegue del sistema.
- Hemos mejorado en la coordinación a la hora de realizar trabajos relacionados con el sub-sistema desarrollado.
- Se ha aprendido a usar la herramienta Jenkins para la realización de la integración continua.
- Hemos obtenido experiencia a la hora de trabajar con un código heredado y nos hemos dados cuenta que es una muy buena práctica tener todo documentado y un código limpio y ordenado, ya que facilitaría muchísimo las cosas a la hora de trabajar en el mismo sub-sistema en el futuro.