Memoria EGC

Subsistema Frontend de Resultados

Documento corto, sin ejercicios... bastante mejorable para el número de personas que han integrado el equipo. Por cierto, no viene firmado por las personas

¿quién ha entregado el proyecto?

Grupo Frontend de Resultados

ÍNDICE

1. Introducción

Este documento contiene toda la información relativa al desarrollo del subsistema Frontend de Resultados de la aplicación web Agora@US. Agora@US es un sistema software que ofrece la posibilidad de realizar votaciones por Internet, y surge como objetivo del trabajo cuatrimestral de la asignatura Evolución y Gestión de la Configuración, perteneciente a la titulación de Grado en Ingeniería del Software.

Dicho trabajo consiste en el desarrollo de la aplicación Agora@US por parte de los alumnos. La organización de dicho trabajo consiste en la división del sistema Agora@US en subsistemas más pequeños, cuyo desarrollo es asignado a distintos grupos de trabajo conformados por los alumnos.

En nuestro caso, el grupo se encarga de desarrollar el subsistema de Frontend de Resultados, el cual se encarga de comunicarse con los subsistemas de Recuento y Modificación de Resultados para obtener datos y se los envía al subsistema de Visualización de Resultados.

Debemos tener en cuenta que la aplicación funciona de manera que un usuario se autentica, accede a las votaciones de las que vaya a ser partícipe y vota. Por último,cuando el proceso de la votación haya terminado, el usuario podrá acceder a la aplicación para consultar los resultados de las votaciones.

Internamente, donde el subsistema de Frontend de Resultados entra en acción es en el momento en que un usuario selecciona en la interfaz la consulta de una votación concreta. Es entonces cuando el subsistema de Visualización de Resultados manda la petición a Frontend de Resultados, que realiza una consulta en su base de datos y, en caso de que la información se encuentre en dicha base de datos, se la devuelve directamente a Visualización. Enel caso de que dicha información no esté disponible en la base de datos, Frontend de Resultados se comunicará con el subsistema de Recuento de Resultados para obtener dichos datos, persistiéndolos en la base de datos y pasándoselos a Visualización de Resultados.

2. Control de versiones del documento

En esta sección del documento se recogen los cambios sufridos en el mismo desde su creación hasta la versión final, tal y como se muestra en la siguiente tabla.

Versión	Fecha	Contenido	Autor
1.0	17/12/2014	Formato y estructura del documento.	Adrián González Castro
1.1	17/12/2014	Redacción de introducción y resumen.	Adrián González Castro
1.2	17/12/2014	Redacción de gestión de la construcción y la integración continua.	Daniel Fernández Romero, Raquel María Cumplido Díaz
1.3	18/12/2014	Redacción de gestión del cambio, incidencias y depuración.	José Antonio Fernández Bueno
1.4	18/12/2014	Redacción de la gestión del código fuente.	Álvaro Fernández García
1.5	20/12/2014	Redacción del mapa de herramientas.	Manuel Cabrera Coronilla
2.0	20/12/2014	Redacción de las conclusiones.	Raquel María Cumplido Díaz

3. Miembros del grupo

A continuación se muestran listados en una tabla los componentes del grupo y el papel que ejercen en el desarrollo del proyecto.

Nombre y apellidos	Rol
Cumplido Díaz, Raquel María	Gestor de la Configuración
Cabrera Coronilla, Manuel	Gestor de la Configuración
Ocaña Almagro, José Pablo	Gestor de la Configuración
Fernández Bueno, José Antonio	Jefe de Proyecto
Fernández Romero, Daniel	Gestor de la Configuración
Rodríguez Palomar, Alejandro	Gestor de la Configuración
Fernández García, Álvaro	Gestor de la Configuración
González Castro, Adrián	Gestor de la Configuración

El resumen siempre va antes que la introducción.

4. Resumen

En esta sección se describe de manera resumida el funcionamiento del subsistema Frontend de Resultados y las distintas etapas por las que ha pasado el desarrollo del proyecto en general.

Vamos a comenzar definiendo con un poco de detalle cuál es el propósito principal de este subsistema. El subsistema Frontend de Resultados se encarga de recibir los recuentos de los votos de las distintas votaciones o referéndums para almacenarlos en una base de datos propia y, posteriormente, atender las peticiones del subsistema de Visualización de resultados. Dichos recuentosserán recibidos de los subsistemas Modificación de Resultados y Recuento de Votos, que se ponen de acuerdo entre ellos para ofrecernos la información correcta a través de Recuento de Votos.

Desde las primeras etapas del proyecto, los grupos de los subsistemas de Recuento, Modificación, Frontend y Visualización de Resultados llegamos al acuerdo de comunicarnos entre nosotros mediante el uso de APIs REST, por lo que los datos serían mandados en forma de JSON con formatos distintos, dependiendo de la API en cuestión.

De esta manera, en primera instancia tomamos la decisión de utilizar el framework de Google, GWT (Google Web Toolkit), para desarrollar el subsistema y crear nuestra API. Sin embargo, tras investigar y desarrollar la parte de la aplicación relacionada con la comunicación con la base de datos, en este caso HSQLDB (HyperSQL DataBase) y después de la nefasta experiencia sufrida en la primera sesión de integración, se decidió migrar el desarrollo para utilizar el framework Spring e Hibernate para comunicarnos con una base de datos MySQL. Los motivos de esta migración fueron principalmente la incapacidad de HSQLDB de soportar operaciones concurrentes y el hecho de que gran parte de los otros grupos de trabajo de la asignatura estaban trabajando con el mismo framework.

A partir de este momento, se desarrolla un segundo prototipo de la aplicación que consume la API del subsistema de Recuento de Resultados, instanciando los tipos Voto y Votación a partir del JSON emitido por dicha API, persistiéndolos en la base de datos. Por otro lado, estos datos son exportados a través de una API al subsistema de Visualización de Resultados adecuándose al formato utilizado por estos últimos. El funcionamiento global es el antes descrito: si la votación que necesita Visualización está en la base de datos es devuelta, y en caso contrario se realiza la petición a la API de Recuento de Votos.

Este segundo prototipo fue el que utilizamos durante la segunda sesión de integración y logramos integrarnos con el subsistema de Visualización de Resultados con éxito.

Tras esta segunda sesión de integración se refina este mismo prototipo y se consigue establecer la correcta comunicación con el subsistema de Recuento de Votos y con el

subsistema de Modificación de Resultados. En este momento, el subsistema realiza las consultas al subsistema de Recuento de Votos o al de Modificación de Resultados, dependiendo de si el subsistema de Visualización de Resultados necesita el recuento de votos original o el modificado.

Sin embargo, poco antes de la fecha de la tercera sesión de integración el formato de los JSON que estamos consumiendo y el de los JSON que necesita el subsistema de Visualización fue cambiado, por lo que hubo que adecuar la aplicación a estas nuevas circunstancias.

Llegamos así a la versión final del subsistema, de la cual adjuntamos la documentación a continuación.

En primer lugar, las clases del dominio de nuestra aplicación son la clase Propuesta y la clase ReferendumRecuento. La clase Propuesta contiene tres atributos que modelan una pregunta, el número de votos que han votado "SÍ" para esa propuesta y el número de votos que han votado "NO" para dicha propuesta.

Por otro lado, la clase ReferendumRecuento tiene tres atributos:**idVotaciónRecuento**, que es el identificador de la votación de la cual queremos obtener la información contenida en el subsistema de Recuento de Votos;**idVotaciónModificación**, que juega el mismo papel pero para la información que se quiere obtener del subsistema de Modificación de Resultados; y, por último, una **colección de propuestas**.

La verdadera funcionalidad del subsistema la encontramos en la única clase del controlador de la aplicación, definido en el archivo PropuestaController.java. Dicho controlador consta de los siguientes métodos:

- Object[] getVotación(@RequestParam Integer idVotación)

Este método se encarga de recibir el identificador de una votación que quiera ser consultada desde el subsistema de Visualización de Resultados para, posteriormente, devolver un JSON para que lo pueda consumir dicho subsistema. En este caso, la consulta se realiza al subsistema de Recuento de Votos.

- Object[] getModificacion(@RequestParam Integer idVotación)

Este método se encarga de recibir el identificador de una votación que quiera ser consultada desde el subsistema de Visualización con el fin de devolver un JSON para que lo pueda consumir dicho subsistema. En este caso, la consulta se realiza al subsistema de Modificación de Resultados.

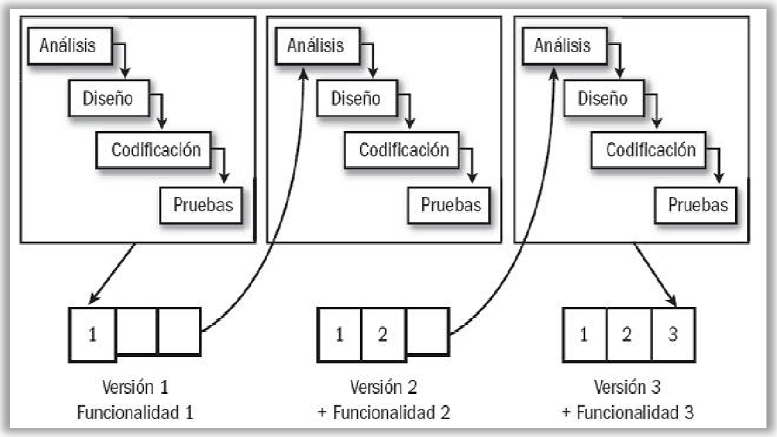
En ambos casos se trata de métodos bastante sencillos que lo que realizan es comunicarse con la base de datos para ver si pueden devolver las votaciones directamente o deben pedírselas a los subsistemas correspondientes.

5. Gestión del código fuente

En primer lugar, cabe destacar que hemos aplicado un ciclo de vida iterativo en nuestro subsistema, puesto que al principio no teníamos claro todos los requisitos que nos serían necesarios.

Este ciclo de vida nos ha permitido iterar sobre un ciclo de vida en cascada, dejando así el software perfectamente funcional al fin de cada iteración.

Para los cambios significativos en el código, como puede ser añadir una nueva funcionalidad, se definió un cambio de versión y los cambios dentro de esa versión se establecieron como revisiones.



Dentro de este ciclo de vida hemos tenido varias versiones, donde hay que diferenciar dos momentos cruciales del proyecto:

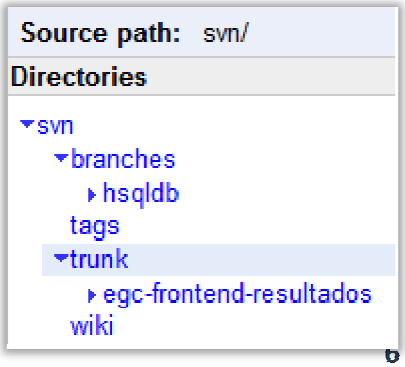
- Desarrollo de la aplicación con AppEngine y GWT
- Desarrollo de la aplicación con Spring y Hibernate.

Nuestra idea un principio era realizar el proyecto sobre AppEngine y GWT basándonos en que era una tecnología que todos conocíamos (asignatura *AISS*) y que nos permitía utilizar MVC (Modelo Vista Controlador). Un software realizado sobre esta tecnología nos daría capacidad de ofrecer una API Rest y persistencia.

Durante esta etapa se definió como repositorio de código Google Code.Para ello utilizábamos un plug-in de Eclipse que nos permitía tener integrado el sistema de control de versiones SVN en el propio IDE.

La estructura del repositorio elegida fue:

- *Trunk*: rama de desarrollo principal
- *Branches*: rama para evolución paralela para añadir determinadas *features*.
- *Tags*: rama para las versiones cerradas.



En este proyecto, todos los integrantes tienen el mismo rol de desarrollador y todos los permisos sobre el repositorio, por lo que había que definir un procedimiento que se tomase como base para poder trabajar sobre el repositorio.

El procedimiento seguido fue:

1- Realizar un *checkout* sobre el repositorio. (sólo 1ª vez)
2- Modificar los ficheros necesarios.
3- Realizar un *update* para actualizar los cambios en local.
4- Resolver conflictos si había alguno.
5- Realizar un *commit* añadiendo título y descripción para actualizar la versión en el repositorio.

Puesto que había muchos desarrolladores para un procedimiento tan básico, se pasó a definir un procedimiento más avanzado donde se desarrolla toda la casuística que se puede generar durante el desarrollo del proyecto.

Los cambios sobre el código son comunicados por WhatsApp al equipo para ponerlos en conocimiento si alguno estuviese trabajando sobre el mismo fichero.

Los *checkouts* serán del tipo no reservado, puesto que en ocasiones necesitamos trabajar sobre el mismo fichero varias personas.

Se estableció que se debía informar de la creación de ramas al equipo, y asegurarse que todo el mundo tuviese conocimiento de que se estaba desarrollando en esa rama. Una rama se creará siempre que se intente mejorar o añadir funcionalidad a la aplicación, no siendo obligatorio su *merge* debido a que nos encontremos en un callejón sin salida.

En nuestro caso, sólo creamos un *branch* para desarrollar la persistencia HSQLDB. Este branch no siguió adelante, ya que nos encontramos con la imposibilidad de desplegarlo en AppEngine.

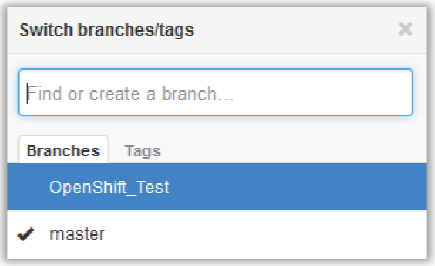
La gestión de parches, por otro lado, no fue necesaria puesto que se definió que cada desarrollador tenía que resolver los conflictos línea a línea de forma que se conservase la integridad y funcionalidad de la aplicación.

Se determinó que los *commits* deberían hacerse cada 20 minutos, aunque era más bien un consejo, no algo obligatorio. También se definió el formato a seguir en los *commits* para que fuera más sencilla la gestión y visualización de los mismos.

La segunda etapa del desarrollo de la aplicación tuvo lugar al darnos cuenta de que el desarrollo de la aplicación en AppEngine y GWT era casi imposible con los requisitos nuevos que se nos habían comunicado, así que cambiamos de tecnología y utilizamos Spring MVC, Hibernate y MySQL. Esta tecnología era conocida por todos los integrantes del grupo puesto que pertenece a la asignatura de Diseño y Pruebas de 3º curso de nuestro grado.

Hubo un cambio más debido a que vimos en clase de prácticas GIT, y el hosting más conocido para repositorios GIT, que es GitHub. Así que pasamos de un repositorio centralizado gestionado con SVN a un repositorio distribuido gestionado por GIT.En este repositorio se han seguido las mismas directrices en cuanto a gestión que con SVN, salvo algunas funcionalidades añadidas ofrecidas por GIT.

La estructura del repositorio en GitHub es la siguiente: la rama de desarrollo principal es la rama **master**, y se podían crear tantas ramas como fuera necesario para *features* siguiendo el mismo procedimiento que en SVN.



El procedimiento base para la interacción con el repositorio se vio alterado, puesto que pasamos de un servidor centralizado a uno distribuido, por lo que ahora era necesario seguir este procedimiento:

1- Realizar un ***git clone*** sobre nuestro proyecto en GitHub.
2- Modificar los ficheros oportunos.
3- Realizar un ***git pull*** para actualizar la versión local.
4- Realizar un ***git add*** con todos los ficheros que queramos pasar al área de preparación.
5- Realizar un ***git commit*** añadiendo título y descripción para hacer un *commit* al repositorio local.
6- Realizar un ***git push*** a la rama sobre la que estemos trabajando.

Durante el desarrollo de la aplicación bajo Spring, Hibernate y MySQL se llegó a un punto en el cual veíamos necesario desplegar la aplicación en un servidor externo, por lo que se creó la rama **OpenShift_Test**, donde se está trabajando para conseguir esta funcionalidad. También se han puesto a disposición de los demás grupos el fichero **.war** del proyecto y un*script* de creación de la base de datos. Esto se ha hecho de manera temporal para que los demás grupos puedan utilizar nuestra aplicación sin tener que realizar todo el proceso de despliegue para generar tanto el fichero **.war** como el *script* de la base de datos.

En resumen, nuestro proyecto ha sufrido muchos cambios, dando lugar a que se hayan generado tres versiones correspondientes a las 3 iteraciones realizadas siguiendo nuestro ciclo de vida. Cada una de ellas corresponde a la integración con uno de los subsistemas.

- Primera versión: Integración con Visualización de Resultados
- Segunda versión: Integración con Recuento.
- Tercera versión: Integración con Modificación.

6. Gestión de la construcción y la integración continua

Con respecto a los aspectos de construcción, siempre se ha llevado a cabo de manera local. A pesar de que el equipo está distribuido y que cada uno hace sus propias construcciones, el proyecto global se encuentra en un repositorio común a todos los integrantes, de manera que siempre haya una versión estable y capaz de ser construida por cualquiera de los miembros del equipo. Es decir, el proyecto debe poder ser construido en cualquier sistema operativo, siempre que la configuración de desarrollo sea similar a aquella en la que se desarrolló la versión de manera original. Afortunadamente, no tuvimos que lidiar con este tipo de obstáculos, pues cada uno de los miembros del equipo utiliza Windows como sistema operativo y la misma configuración en términos de desarrollo.

No obstante, el equipo de trabajo no seguía un control de versiones propiamente dicho. En su lugar, las modificaciones sobre el proyecto eran llevadas a cabo y, si procedía, se aplicaban los cambios al proyecto y la nueva versión era subida al repositorio común. Es decir, no existía una periodicidad a la hora de hacer la subida del proyecto con cambios aplicados (una versión en sí), sino que se realizaba la subida conforme surgía la necesidad.

La gestión de dependencias se realizaba de manera automática mediante el uso de la herramienta Maven. Así, cada vez que el proyecto era construido, se utilizaba el fichero de configuración pertinente (pom.xml) y Maven se encargaba de acceder a los módulos y componentes externos y de definir en qué orden se utilizaban dichos elementos en el proceso de construcción.

En cuanto a la integración continua, en un principio no se utilizaba ningún tipo de mecanismo para llevarla a cabo. Es decir, el equipo subía el proyecto con nuevas funcionalidades cuando era necesario, sin tener en cuenta aspectos de integración siquiera.

Si bien es cierto que a nivel de equipo no existía integración alguna, a nivel global sí que se llevaba a cabo una integración por fases. Dichas fases fueron los denominados talleres de integración, en el que se pretendía unir todos los subsistemas de Agora@US e intentar que la integración diese lugar a un sistema estable. Téngase en cuenta que cada subsistema concernía a un grupo de trabajo diferente, por lo que los métodos de desarrollo eran muy diversos y era complicado llevar a cabo una integración de otro tipo.

Finalmente, y tras estudiar en las clases de prácticas y apreciar la utilidad de la herramienta de integración continua Jenkins durante las jornadas de la asignatura, se decidió adoptar esta herramienta como mecanismo de integración continua de cara a

automatizar el proceso de integración mediante el uso de baterías de pruebas. Estas pruebas contemplan aspectos como pruebas funcionales, de rendimiento, de concurrencia, de inserción de datos en BBDD, consistencia del subsistema tras realizar la construcción, etc.

7. Gestión del cambio, incidencias y depuración

7.1 Mecanismos de depuración

Con respecto a la depuración de errores en nuestro subsistema, usaremos el depurador del IDE Eclipse, puesto que nuestro proyecto está realizado con dicho IDE. Los pasos a seguir son los siguientes:

- Reportar el error.
- Diagnosticar la causa.
- Aplicar las correcciones correspondientes.
- Realizar el *commit* para aplicar los cambios.

7.2Gestión de cambios

La gestión de cambios se realiza mediante GIT y la aplicación web Github, en la que podemos ver todos los cambios realizados por los integrantes del subsistema.

Las pautas que se han establecido para homogeneizar los *commits* realizados en GIT siguen el siguiente formato:

- En primer lugar, el título comenzará con una palabra clave en mayúsculas y en español. Esta palabra hará referencia al tipo de cambio del que se trata. Ejemplos:

  CORRECCIÓN: [título del commit]
  APIGET: [título del commit]
  APIPOST: [título del commit]
  PERSISTENCIA: [título del commit]
  CONFIGURACIÓN: [título del commit]
  DESPLIEGUE: [título del commit]

- Una vez especificado el tipo de cambio, en la parte [título del commit] se expondrá con un poco más de detalle el cambio lógico que supone dicho *commit*, sin superar los 80 caracteres.

Por último, se añadirá una descripción detallada que responda al porqué del cambio y explique en qué consiste.

7.3Gestión de incidencias

Al tratarse éste de un proyecto de pequeñas dimensiones, no se contempló el hecho de gestionar las posibles incidencias producidas más allá del informe de incidencias rellenado durante los talleres de integración realizados.

8. Mapa de herramientas

8.1 Lista de herramientas

- **Java**

Será el lenguaje de programación que utilizaremos para implementar nuestro subsistema, ya que está bastante extendido y tenemos bastante experiencia con él.

Es un lenguaje de programación de propósito general orientado a objetos, derivado en gran medida de C y C++.

- **JDK 1.7**

Java Development Kit es el conjunto de herramientas que nos provee todo lo necesario para crear, compilar e interpretar nuestro código programado en Java.

- **Eclipse**

Será el IDE o marco de trabajo en el cual desarrollaremos nuestro subsistema. Entre sus características destaca que es de código abierto, y dispone de multitud de *plug-ins* y permite la integración con Hibernate.

- **Maven**

Maven es una herramienta de software para la gestión y construcción, ya que está diseñada para Java.Ofrece un modelo de construcción bastante simple basado en XML.

Maven utiliza un Project Object Model (POM) para describir el proyecto, y está listo para usarse en red, y descargarse las librerías de un repositorio de forma automática.

- **Spring**

Este framework de código abierto está bastante extendido en la plataforma Java.

Sus características principales son la inyección de dependencias y la programación orientada a aspectos.

- **Hibernate**

Es una herramienta de software libre que permite el mapeo objeto-relación. Para su configuración se usan

archivos XML y anotaciones en los objetos que permiten establecer relaciones.

- **GIT**

Diseñado por Linus Torvard, es una herramienta de control de versiones.

Entre sus características, cabe destacar la gestión distribuida, fuente de apoyo al desarrollo no lineal y gestión eficiente de grandes proyectos.

- **Jenkins**

Software escrito en Java. De código abierto, gestor de integración continua y permite la integración con GIT. Esto facilita el uso de baterías de pruebas para cada *push* realizado sobre el repositorio en GitHub.

- **JSON**

Es un formato ligero para el intercambio de datos. Usaremos una librería Java para codificar nuestros datos e intercambiarlos con los subsistemas correspondientes.

- **MySQL**

Como sistema gestor de base de datos usaremos MySQL para almacenar los datos de nuestro subsistema. Usaremos la versión GNU GPL.

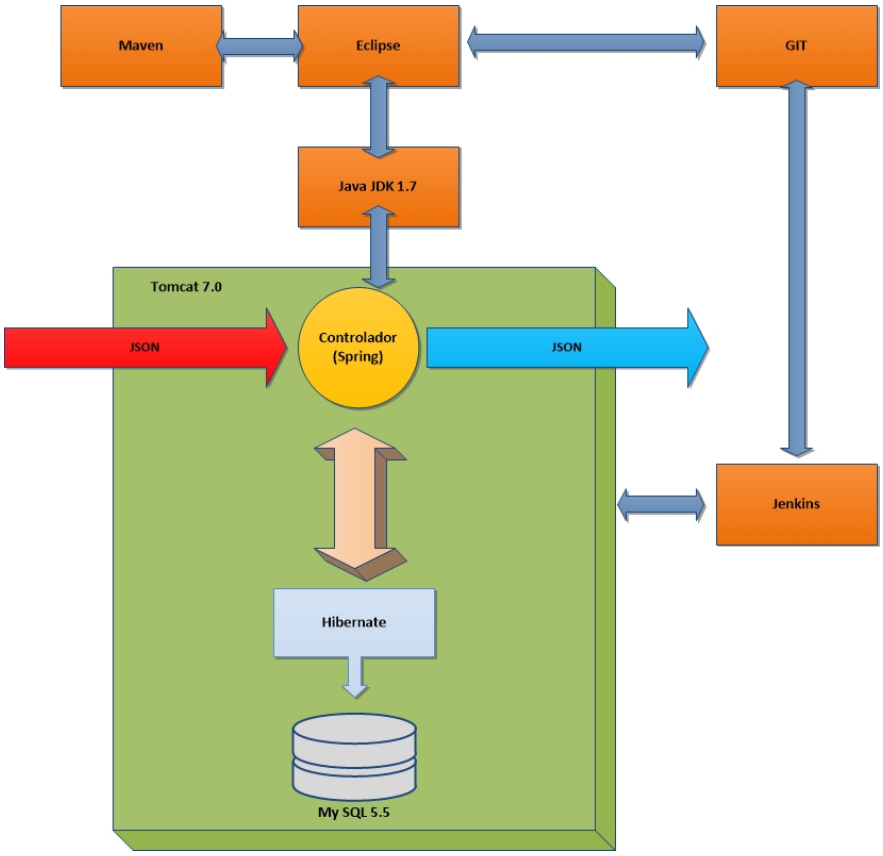
- **Apache Tomcat**

Para la ejecución de nuestro subsistema necesitaremos un servidor Tomcat, puesto que ofrece soporte como contenedor web, de servlets y JSP. En definitiva, es un servidor web adaptado.

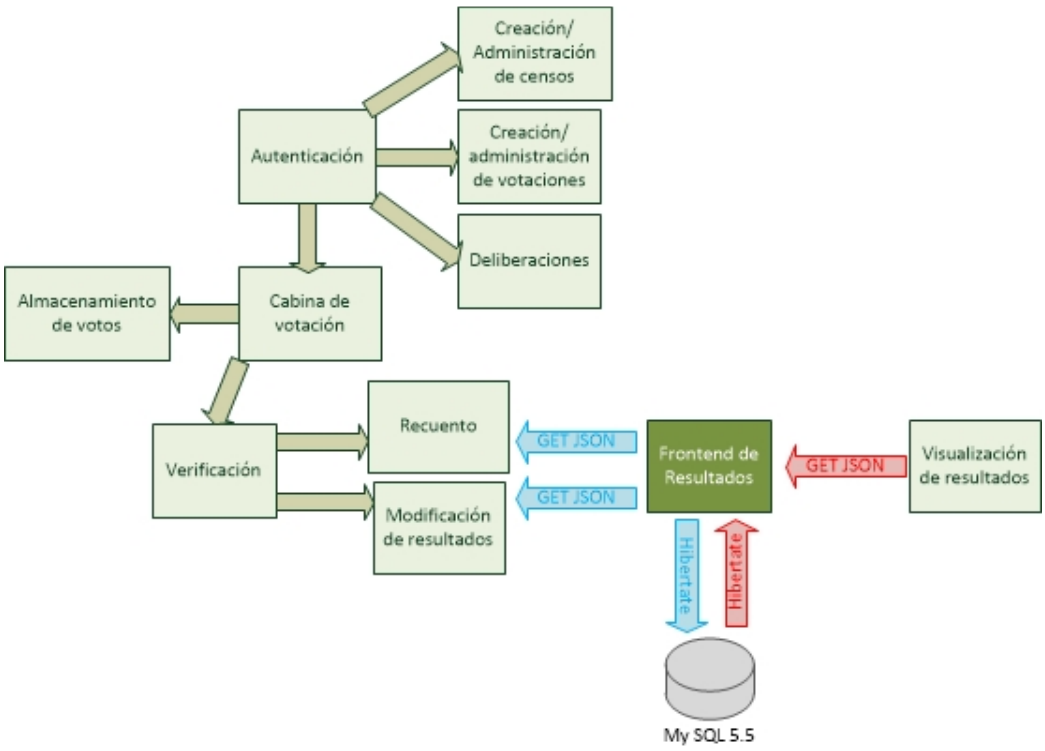
- **Python**

Otros subsistemas usaran el lenguaje de programación Python para el desarrollo de su subsistema.

8.2 Diagrama de herramientas visión local



8.3 Diagrama de herramientas visión global



9. Conclusiones

Durante el desarrollo de este proyecto hemos podido comprobar de primera mano la importancia que juegan en la gestión de la configuración aspectos como la gestión de la integración continua o la gestión del código fuente.

Si no se especifican este tipo de cuestiones muy probablemente un proyecto informático esté abocado al fracaso de manera irremediable, como hemos podido comprobar en las primeras sesiones de integración de código.

Por último, ha resultado bastante para útil poner en práctica estos conceptos durante el desarrollo del proyecto, puesto que se ha aprendido a trabajar utilizando herramientas como GIT que actualmente son usadas por equipos de desarrollo profesionales.