**EGC**

Frontend de Resultados

Grupo 8

Cabrera Coronilla, Manuel

Cumplido Díaz, Raquel Mª

Fernández Bueno, José Antonio

Fernández García, Álvaro

Fernández Romero, Daniel

González Castro, Adrián

Ocaña Almagro, José Pablo

Rodríguez Palomar, Alejandro

# Resumen

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 de información que ofrece la posibilidad de realizar votaciones por Internet. Ésta se utilizará para aprender a llevar a cabo la gestión de la configuración en un proyecto, y constituye el trabajo cuatrimestral de la asignatura Evolución y Gestión de la Configuración, perteneciente a la titulación de Grado en Ingeniería del Software.

La organización del 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 que posteriormente son enviados al subsistema de Visualización de Resultados.

Para realizar el trabajo se han usado, entre otras, las siguientes herramientas:

* Gestión del proyecto: *Redmine*
* Gestión del código fuente: *Git*
* Gestión de la construcción: *Maven*
* Gestión de la integración continua: *Jenkins*
* Gestión del despliegue: *Openshift*
* Gestión de la calidad: *SonarQube*

# Historial de versiones

|  |  |  |  |
| --- | --- | --- | --- |
| **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 Cumplido Díaz |
| 2.1 | Fecha de mentira | Bla | Bla |
| 2.2 | Fecha de mentira | Bla | Bla |

1. Índice

[Resumen 3](#_Toc410738408)

[Historial de versiones 3](#_Toc410738409)

[1. Índice 5](#_Toc410738410)

[2. Índice de figuras 6](#_Toc410738411)

[3. Roles 7](#_Toc410738412)

[4. Introducción 7](#_Toc410738413)

[5. Gestión del código fuente 9](#_Toc410738414)

[**5.1.** **Sintaxis del código fuente** 9](#_Toc410738415)

[**5.2.** **Gestión de ramas** 10](#_Toc410738416)

[**5.3.** **Creación y aplicación de patch** 10](#_Toc410738417)

[**5.4.** **Roles** 10](#_Toc410738418)

[**5.5.** **Conflictos** 10](#_Toc410738419)

[**5.6.** **Migración de SVN a Git** 10](#_Toc410738420)

[**5.7.** **Herramientas similares** 10](#_Toc410738421)

[**5.8.** **Ejercicios** 10](#_Toc410738422)

[6. Gestión de la construcción y despliegue 10](#_Toc410738423)

[**6.1.** **Maven** 10](#_Toc410738424)

[**6.2.** **Openshift** 10](#_Toc410738425)

[**6.3.** **Ejercicios** 11](#_Toc410738426)

[7. Gestión de la construcción e integración continua 11](#_Toc410738427)

[**7.1.** **Jenkins** 12](#_Toc410738428)

[**7.2.** **Ejercicios** 12](#_Toc410738429)

[8. Gestión de la calidad 12](#_Toc410738430)

[9. Gestión del cambio, incidencias y depuración 12](#_Toc410738431)

[**9.1.** **Gestión del cambio** 12](#_Toc410738432)

[**9.2.** **Gestión de las incidencias** 12](#_Toc410738433)

[**9.3.** **Depuración** 12](#_Toc410738434)

[**9.4.** **Ejercicios** 12](#_Toc410738435)

[10. Mapa de herramientas 12](#_Toc410738436)

[**10.1.** **Lista de herramientas** 12](#_Toc410738437)

[**10.2.** **Visión interna del mapa de herramientas** 14](#_Toc410738438)

[**10.3.** **Visión global del mapa de herramientas** 15](#_Toc410738439)

[11. Integración de todos los subsistemas 16](#_Toc410738440)

[**11.1.** **Gestión del código** 16](#_Toc410738441)

[**11.2.** **Estrategia de integración** 16](#_Toc410738442)

[**11.3.** **Herramientas** 16](#_Toc410738443)

[**11.4.** **Automatización de la construcción** 16](#_Toc410738444)

[12. Conclusiones 17](#_Toc410738445)

[Bibliografía 17](#_Toc410738446)

[Glosario de términos 17](#_Toc410738447)

[Anexos 17](#_Toc410738448)

1. Índice de figuras

Figura 1: Arquitectura de Agora@US 6

Figura X: Visión interna del mapa de herramientas 5

Figura X: Visión global del mapa de herramientas 3

Figura 4: [Nombre] 5

Figura 5: [Nombre] 3

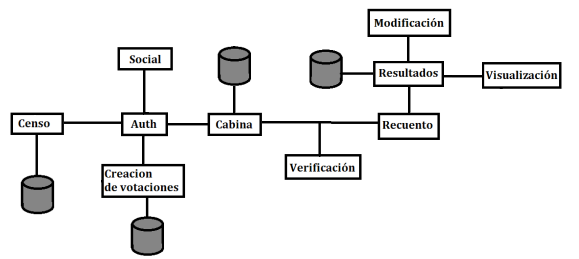
Figura 6: [Nombre] 5

# Roles

|  |  |
| --- | --- |
| **Nombre y apellidos** | **Rol** |
| Manuel Cabrera Coronilla | Gestor de la configuración |
| Raquel Cumplido Díaz | Gestor de la configuración |
| José Antonio Fernández Bueno | Jefe de proyecto |
| Álvaro Fernández García | Gestor de la configuración |
| Daniel Fernández Romero | Gestor de la configuración |
| Adrián González Castro | Gestor de la configuración |
| José Pablo Ocaña Almagro | Gestor de la configuración |
| Alejandro Rodríguez Palomar | Gestor de la configuración |

# Introducción

En esta sección se presenta la arquitectura de la aplicación en su totalidad y se describe, en resumidas líneas, el funcionamiento del subsistema Frontend de Resultados, así como las distintas etapas por las que ha pasado el desarrollo del proyecto en general.



*Figura 1: Arquitectura de Agora@US*

Para comenzar, se define el propósito principal del subsistema Frontend de Resultados. Éste 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 recuentos serán recibidos de los subsistemas Modificación de Resultados y Recuento de Votos, que se ponen de acuerdo entre ellos para ofrecer la información correcta a través del subsistema de Recuento de Votos.

Desde las primeras etapas del proyecto, entre los grupos de Recuento, Modificación, Frontend y Visualización de Resultados se llegó al acuerdo de que la comunicación entre estos subsistemas se realizaría mediante el uso de APIs REST, por lo que los datos serían enviados como un JSON con formatos distintos, dependiendo de la API en cuestión.

Por otro lado, en lo referente a la tecnología utilizada, en primer lugar se tomó la decisión de utilizar el *framework* GWT (*Google Web Toolkit*) para desarrollar el subsistema y crear la API. No obstante, tras investigar y desarrollar la parte de la aplicación relacionada con la comunicación con la base de datos, en este caso HSQLDB (*HyperSQL DataBase*), unido a la nefasta experiencia sufrida en la primera sesión de integración, se decidió migrar el desarrollo para utilizar el framework *Spring* e *Hibernate* para que la comunicación se realizase con una base de datos MySQL. Una causa más de esta migración fue la incapacidad de HSQLDB para soportar operaciones concurrentes.

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 se utilizó durante la segunda sesión de integración, logrando llevar a cabo la integración con el subsistema de Visualización de Resultados.

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 se consumían 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. Se llega así a la versión final del subsistema, cuya estructura se expone a continuación.

En primer lugar, las clases del dominio de la 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 se desea 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 se encuentra en el único 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 realizan la comunicación con la base de datos para comprobar si pueden devolver las votaciones directamente o deben pedírselas a los subsistemas correspondientes.

# Gestión del código fuente

[Waly: Explicación como la del grupo 9 incluyendo ventajas y desventajas de Redmine]

* 1. **Sintaxis del código fuente**

En este apartado se detalla la política que se ha seguido a la hora de escribir el código fuente, con el fin de homogeneizar la sintaxis utilizada.

En primer lugar, los nombres de todas las clases empiezan por mayúscula, y en caso de tener más de una palabra, cada una de las que siguen a la primera también empezarán con mayúscula. En el caso de las variables, por el contrario, no se empieza por mayúsculas.

Por otro lado, los servicios y los repositorios relacionados con las entidades del dominio se denominarán siguiendo el esquema NombreEntidadService y NombreEntidadRepository respectivamente.

En lo que respecta a los métodos de cada clase, éstos comienzan con minúscula, pero en el caso de que tengan más de una palabra, de la segunda en adelante comienzan con mayúscula.

Por último, para la estructura general del código se ha seguido el esquema de programación indentada usando tabulaciones.

* 1. **Gestión de ramas**

[Raquel]

* 1. **Creación y aplicación de patch**

[Raquel]

* 1. **Roles**

[Raquel]

* 1. **Conflictos**

[Raquel]

* 1. **Migración de SVN a Git**

[Álvaro]

* 1. **Herramientas similares**

[Adrián]

* 1. **Ejercicios**

[Adrián]

# Gestión de la construcción y despliegue

[JP]

* 1. **Maven**

[Álvaro: mirar grupos 1 y 9]

* 1. **Openshift**

[Waly: recuerda explicar primero en general para qué sirve y qué ofrece, y luego en nuestro caso los pasos que se han realizado para conseguir desplegarlo]

* 1. **Ejercicios**

[Álvaro: mirar grupos 9 y 1 (ejercicio de Maven, no de OpenShift)]

# Gestión de la construcción e integración continua

A la hora de llevar a cabo la compilación y ejecución de pruebas del proyecto, es recomendable que esto se realice de manera automática y periódica, para detectar posibles errores introducidos en el código fuente lo antes posible. A este proceso de descargar el código fuente desde el control de versiones, compilarlo, ejecutar pruebas y generar informes se le denomina integración continua.

Para ayudar en la pronta detección de errores y en el ahorro de una gran cantidad de tiempo, las herramientas de integración continua utilizan el correo electrónico para notificar a los desarrolladores cuando se producen errores en alguna prueba o proceso.

Algunos autores propugnan una serie de buenas prácticas para conseguir realizar la integración continua. Entre ellas encontramos:

* Tener un único repositorio de código.
* Automatizar la construcción, es decir, que un solo comando sea capaz de construir el sistema.
* Después de cada construcción se debe comprobar que ésta se ha realizado correctamente.
* Toda persona que esté trabajando en el sistema debe hacer *commit* a la rama *master* cada día, con el fin de minimizar los conflictos.
* Después de cada *commit* se debe construir el sistema verificando la correcta integración del cambio.
* Automatizar el despliegue, consiguiendo así probar el sistema en un servidor de prueba. El siguiente paso de esta práctica sería el despliegue continuo, que despliega de manera periódica el sistema detectando errores.

Aunque para este proyecto se ha usado Jenkins para llevar a cabo la integración continua, existen otras opciones, como Bamboo, Continuum, o Hudson.

Para el caso concreto del subsistema Frontend de Resultados, se observa que la envergadura del mismo es bastante reducida, lo cual tienta a que su desarrollo se realice de un modo más rudimentario y sin control. Sin embargo, se han seguido todas las buenas prácticas anteriormente expuestas, como se podrá observar a continuación, en la sección específica sobre el uso de Jenkins.

* 1. **Jenkins**

[Xory y yo]

* 1. **Ejercicios**

[Xory y yo]

# Gestión de la calidad

[Daniel: un parrafito y luego las secciones que quieras (al final recuerda ejercicios)]

# Gestión del cambio, incidencias y depuración

[Ale]

* 1. **Gestión del cambio**

[Ale]

* 1. **Gestión de las incidencias**

[Ale]

* 1. **Depuración**

[Ale]

* 1. **Ejercicios**

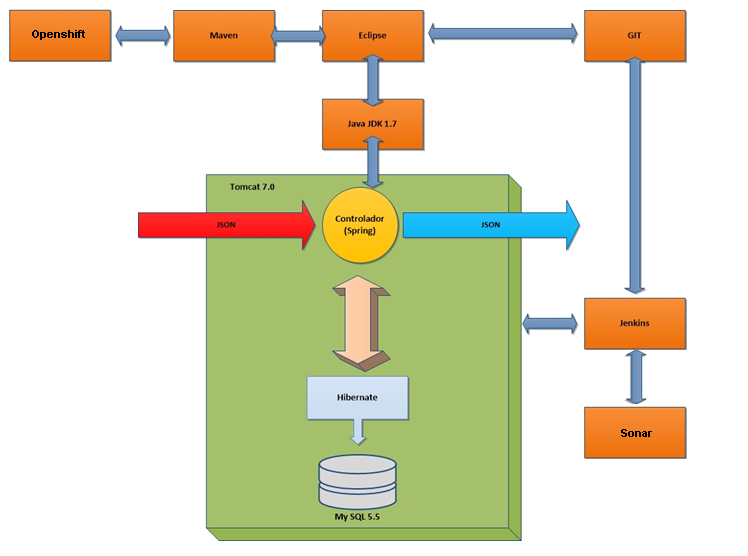
[Ale: un ejercicio que incluya un ejemplo de las tres gestiones (añadiendo funcionalidad de que las votaciones puedas ser de otra forma)]

# Mapa de herramientas

* 1. **Lista de herramientas**

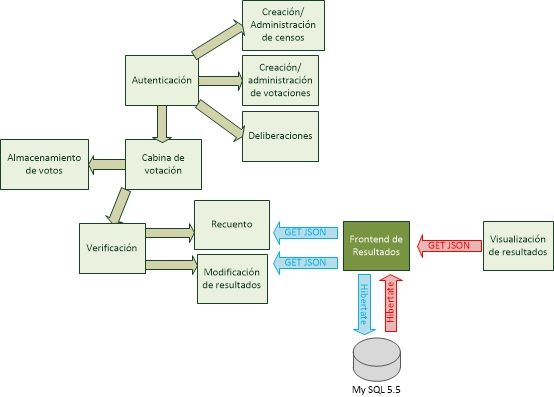
|  |  |
| --- | --- |
| * **Java**   Será el lenguaje de programación que utilizaremos para implementar nuestro subsistema, ya que está bastante extendido y tenemos bastante experiencia con él.  Es un lenguaje de programación de propósito general orientado a objetos, derivado en gran medida de C y C++. | **C:\Users\Manuel\AppData\Local\Microsoft\Windows\INetCache\Content.Word\java_tech.jpg** |
| * **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. | C:\Users\Manuel\AppData\Local\Microsoft\Windows\INetCache\Content.Word\jdk-java.png |
| * **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. | **C:\Users\Manuel\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Eclipse-logo.png** |
| * **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. | **C:\Users\Manuel\AppData\Local\Microsoft\Windows\INetCache\Content.Word\maven.png** |
| 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. | C:\Users\Manuel\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Spring.png |
| * **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. | C:\Users\Manuel\AppData\Local\Microsoft\Windows\INetCache\Content.Word\hibernate.png |
| * **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. | **C:\Users\Manuel\AppData\Local\Microsoft\Windows\INetCache\Content.Word\git.jpg** |
| * **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. | C:\Users\Manuel\AppData\Local\Microsoft\Windows\INetCache\Content.Word\jenkins.png |
| * **JSON**   Es un formato ligero para el intercambio de datos. Usaremos una librería Java para codificar nuestros datos e intercambiarlos con los subsistemas correspondientes. | C:\Users\Manuel\AppData\Local\Microsoft\Windows\INetCache\Content.Word\json_logo-555px.png |
| * **MySQL**   Como sistema gestor de base de datos usaremos MySQL para almacenar los datos de nuestro subsistema. Usaremos la versión GNU GPL. | C:\Users\Manuel\AppData\Local\Microsoft\Windows\INetCache\Content.Word\mysql_logo.png |
| * **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. | C:\Users\Manuel\AppData\Local\Microsoft\Windows\INetCache\Content.Word\tomcat.jpg |
| * **Python**   Otros subsistemas usaran el lenguaje de programación Python para el desarrollo de su subsistema. | C:\Users\Manuel\AppData\Local\Microsoft\Windows\INetCache\Content.Word\python-logo-master-v3-TM.PNG |
| * **SonarQube**   Se trata de una plataforma *open source* para la comprobación continua de la calidad del código fuente. | http://javierac.biz/wp-content/uploads/2014/05/SonarQube.png |
| * **OpenShift**   Para conseguir realizar el despliegue del subsistema en la nube se utiliza este producto, que ofrece un servicio de *hosting* gratuito, junto a otros, como la integración con Jenkins. | http://upload.wikimedia.org/wikipedia/en/thumb/3/3a/OpenShift-LogoType.svg/1024px-OpenShift-LogoType.svg.png |

* 1. **Visión interna del mapa de herramientas**

****

*Figura X: Visión interna del mapa de herramientas*

* 1. **Visión global del mapa de herramientas**



*Figura X: Visión global del mapa de herramientas*

# Integración de todos los subsistemas

[Intro: Xory y JP]

* 1. **Gestión del código**

**[Xory y JP]**

* 1. **Estrategia de integración**

**[Xory y JP]**

* 1. **Herramientas**

**[Xory y JP]**

* 1. **Automatización de la construcción**

**[Xory y JP]**

# Conclusiones

[JP: rehacerlas enteras y con más cosas]

# Bibliografía

[Daniel: revisar y completar si es necesario]

# Glosario de términos

[Álvaro: mirar el grupo 7 (**pero no copypaste**)]

# Anexos

[Xory y JP: Instalación de las máquinas virtuales]