|  |
| --- |
| Autenticación |
| Trabajo de Evolución y Gestión de la Configuración |
| Grupo 3 (turno de mañana)  Jorge Rodríguez Fuerte  Alejandro Tortolero Martín |
| Sergio Carrascosa Oliva  Pablo José Carrillo García  Julián Alexander Mayer Álvarez  José Manuel Navarro Márquez |
| 4ºcurso de Ingeniería Informática-Ingeniería del Software |
|  |

Índice

[Control de versiones 4](#_Toc442792851)

[Enlaces importantes 6](#_Toc442792852)

[Índice de figuras 7](#_Toc442792853)

[Índice de tablas 8](#_Toc442792854)

[Resumen 9](#_Toc442792855)

[Miembros del grupo 10](#_Toc442792856)

[1. Introducción 11](#_Toc442792857)

[2. Descripción del sistema 14](#_Toc442792858)

[2.1 Descripción funcional 14](#_Toc442792859)

[2.2 Descripción técnica 17](#_Toc442792860)

[3. Elementos de control 18](#_Toc442792861)

[4. Entorno de desarrollo 20](#_Toc442792862)

[5. Gestión del código fuente 24](#_Toc442792863)

[5.1 Arquitectura y creación de documentos 25](#_Toc442792864)

[5.2 Permisos 27](#_Toc442792865)

[5.3 Gestión del código en relación al conjunto de subsistemas y nomenclatura 28](#_Toc442792866)

[5.5 Aprobación de cambios: 29](#_Toc442792867)

[5.6 Gestión de las ramas del código 30](#_Toc442792868)

[5.7 Roles 32](#_Toc442792869)

[5.8 Aplicación de un parche (patch) 32](#_Toc442792870)

[5.9 Políticas y nombre de estilos 33](#_Toc442792871)

[5.10 Usage model 33](#_Toc442792872)

[6. Gestión de la construcción e integración continua 34](#_Toc442792873)

[6.1 Gestión de la construcción 34](#_Toc442792874)

[6.2 Gestión de la integración continua 34](#_Toc442792875)

[6.2.1 Herramientas 35](#_Toc442792876)

[6.2.2 Uso de herramientas 35](#_Toc442792877)

[6.2.3 Tiempo entre construcción de los proyectos 36](#_Toc442792878)

[6.2.4 Ejemplo propuesto 36](#_Toc442792879)

[7. Gestión del cambio, incidencias y depuración 37](#_Toc442792880)

[7.1 Peticiones de cambio e incidencias 37](#_Toc442792881)

[7.2 Depuración 39](#_Toc442792882)

[8. Gestión de las liberaciones, despliegue y entregas 41](#_Toc442792883)

[8.1 Entregables 41](#_Toc442792884)

[8.2 Entrega 41](#_Toc442792885)

[8.3 Roles 42](#_Toc442792886)

[8.4 Mecanismo de despliegue 42](#_Toc442792887)

[9. Mapa de herramientas 43](#_Toc442792888)

[9.1 Descripción del mapa de herramientas 43](#_Toc442792889)

[10. Conclusiones 46](#_Toc442792890)

[11. Bibliografía 47](#_Toc442792891)

# Control de versiones

|  |  |  |  |
| --- | --- | --- | --- |
| Versión | Fecha | Modificaciones | Miembro del grupo |
| 0.1.0 | 30/11/2015 | Añadir estructura básica del documento | José Manuel Navarro Márquez |
| 0.1.1 | 03/12/2015 | Añadir estilo al documento | Jorge Rodríguez Fuerte |
| 0.1.2 | 05/12/2015 | Realización de los apartados introducción y resumen | José Manuel Navarro Márquez |
| 0.1.3 | 08/12/2015 | Realización del apartado descripción del sistema | José Manuel Navarro Márquez |
| 0.1.4 | 10/12/2015 | Realización del apartado elementos de control | Sergio Carrascosa Oliva |
| 0.1.5 | 14/12/2015 | Realización del apartado de entorno de desarrollo | Sergio Carrascosa Oliva |
| 0.1.6 | 18/12/2015 | Realización del apartado gestión del código | Jorge Rodríguez Fuerte |
| 0.1.7 | 20/12/2015 | Realización del apartado gestión del cambio, incidencias y depuración | Jorge Rodríguez Fuerte |
| 0.1.8 | 21/12/2015 | Realización del apartado gestión de las liberaciones y mapa de herramientas | Alejandro Tortolero Martín |
| 0.1.9 | 22/12/2015 | Expansión de ejercicios en el apartado de gestión del código | Sergio Carrascosa Oliva |
| 1.0.0 | 26/12/2015 | Expansión de ejercicios en el apartado de resumen e introducción.  Corrección de errores | Alejandro Tortolero Martín |
| 1.0.1 | 27/01/2016 | Reestructuración del documento  Añadido nuevos apartados: control de versiones, enlaces, índice de figuras e índice de tablas, gestión de la construcción e integración continua, conclusiones y bibliografía. | Sergio Carrascosa Oliva |
| 1.0.2 | 28/01/2016 | Extensión del apartado de introducción | José Manuel Navarro Márquez |
| 1.0.3 |  | Extensión del apartado de gestión del código | Jorge Rodríguez Fuerte |
| 1.0.4 | 28/01/2016 | Corrección de errores en los apartados de elementos de control y entorno de desarrollo  Añadido número de páginas | Sergio Carrascosa Oliva |
| 1.0.5 |  | Subsanación de errores del apartado de gestión del código y ampliación de sub-apartados | Jorge Rodríguez Fuerte |
| 1.0.6 | 29/01/2016 | Realización del apartado de integración continua y bibliografía | Julián Alexander Mayer Álvarez |
| 1.0.7 | 31/01/2016 | Corrección y mejora de la sección gestión de las liberaciones, despliegue y entregas | Pablo José Carrillo García |
| 1.0.8 |  | Ampliación del apartado de gestión de las liberaciones, despliegue y entregas | Pablo José Carrillo García |
| 1.0.9 | 31/01/2016 | Modificación del apartado descripción del mapa de herramientas | Julián Alexander Mayer Álvarez |
| 2.0.0 | 02/02/2016 | Extensión del apartado de introducción y descripción del sistema | José Manuel Navarro Márquez |
| 2.0.1 |  | Extensión del documento | Sergio Carrascosa Oliva |
| 2.0.2 | 08/02/2016 | Apartado Miembros del documento | Alejandro Tortolero Martín |
| 2.0.1 | 09/02/2016 | Apartados 6.1, 6.2.4 y 8.4 | Pablo José Carrillo García |
| 2.0.2 | 09/02/2016 | Realización del apartado conclusión | Julián Alexander Mayer Álvarez |

Tabla 1. Control de versiones.

# Enlaces importantes

* El repositorio se puede encontrar aquí: <https://github.com/AgoraUS1516/G03>.
* El repositorio central de AgoraUs se puede encontrar aquí:

<https://github.com/AgoraUS1516>

* Espacio de grupo del subsistema Autenticación: <https://1984.lsi.us.es/wiki-egc/index.php/Autenticación>
* Gestión de incidencias mediante GitHub: <https://github.com/AgoraUS1516/G03/issues>
* Subsistema en remoto: <http://agora-us.260mb.net>

# Índice de figuras

[Figura 1. Arquitectura de Agora Voting 10](#_Toc442621300)

[Figura 2. Index del sistema de AGORA@US 13](#_Toc442621301)

[Figura 3. Peticiones GET sacado de la API del proyecto de EGC del año anterior. 14](#_Toc442621302)

[Figura 4. Formulario de registro del sistema AGORA@US 14](#_Toc442621303)

[Figura 5. Diagrama de entornos 16](#_Toc442621304)

[Figura 6. Descarga de Aptana Studio 3.6.1 18](#_Toc442621305)

[Figura 7. Aceptar las condiciones e instalar 19](#_Toc442621306)

[Figura 8. Instalación de Aptana Studio 3.6.1 19](#_Toc442621307)

[Figura 9. Selección de carpeta para workspace 20](#_Toc442621308)

[Figura 10. Proyecto exportado en Aptana 21](#_Toc442621309)

[Figura 11 23](#_Toc442621310)

[Figura 12 34](#_Toc442621311)

[Figura 13 35](#_Toc442621312)

[Figura 14. . Mapa de herramientas 40](file:///C:\Users\raul-sergio\Desktop\RepositorioEgc\G03\Documentacion\MemoriaActualizada2.docx#_Toc442621313)

# Índice de tablas

[Tabla 1. Control de versiones. 5](#_Toc442621291)

[Tabla 2. API REST ofrecida del proyecto de EGC del año anterior 15](file:///C:\Users\raul-sergio\Desktop\RepositorioEgc\G03\Documentacion\MemoriaActualizada2.docx#_Toc442621292)

# Resumen

En la presente memoria describimos como se ha desarrollado el proyecto de la asignatura de Evolución y Gestión de la Configuración (EGC), que consiste principalmente en integrar nuestro proyecto que corresponde a un subsistema, con otros subsistemas para ofrecer una herramienta de voto online.

En las primeras semanas de curso se propusieron una serie de proyectos o subsistemas que engloban una aplicación o sistema completo, cuya funcionalidad es la de hacer votaciones online.

Estos proyectos, son proyectos creados de años anteriores, basado en el proyecto real y ya en funcionamiento Agora Voting. De estos proyectos, se nos proporcionan su desarrollo para poder realizar lo que será este año nuestro trabajo, que consiste en modificar y corregir posibles conflictos, incidencias o simplemente proporcionar mejoras para cumplir el objetivo principal, integrarnos con los demás subsistemas y hacer funcionar el sistema en toda su totalidad.

Los distintos subsistemas proporcionados se reparten en subgrupos hechos en el aula, donde cada uno de estos subgrupos tendrá que coordinarse unos con otros para llevar a cabo el objetivo principal antes mencionado.

Nuestro grupo de trabajo ha tenido que valorar y tomar decisiones tanto internamente cómo con los demás grupos de los demás subsistemas teniendo que establecer procesos y herramientas ya constituidas para crear políticas de trabajo y seguir unas pautas para el desarrollo de nuestro subsistema y para la integración con los demás subsistemas.

Además de la temática que se imparte en la asignatura, se aportaron una serie de herramientas de gestión de código, gestión de incidencias y depuración para poder desarrollar nuestro subsistema y poder realizar los procesos de integración continua con los demás subsistemas así como el uso de un servidor donde se irá actualizando el desarrollo de cada subsistema y hacer más fácil la integración completa de todos los subsistemas.

# Miembros del grupo //FALTA LEYENDA DE LA TABLA

|  |  |  |
| --- | --- | --- |
| Nombre del participante | Rol | Foto de cada miembro del grupo |
| Sergio Carrascosa Oliva | C:\Users\raul-sergio\Desktop\Scan0003.jpgC:\Users\raul-sergio\Desktop\Sin título2.pngGestor de la configuración |  |
| Pablo José Carrillo García | Gestor de la configuración |  |
| Julián Alexander Mayer Álvarez | C:\Users\raul-sergio\Desktop\Sin título4.pngGestor de la configuración |  |
| José Manuel Navarro Márquez | C:\Users\raul-sergio\Desktop\Sin título.pngC:\Users\raul-sergio\Desktop\Sin título (2).pngGestor de la configuración |  |
| Jorge Rodríguez Fuerte | C:\Users\raul-sergio\Desktop\Sin título3.pngGestor de la configuración |  |
| Alejandro Tortolero Martín | Gestor de la configuración |  |

# 1. Introducción

Trabajaremos sobre un proyecto desarrollado el año pasado en la asignatura de EGC llamado AGORA@US que consiste en crear un sistema de votaciones online basado en el proyecto profesional ya desarrollado y en funcionamiento Agora Voting.



Figura 1. Arquitectura de Agora Voting

Referencia: <https://1984.lsi.us.es/wiki-egc/index.php/Lista_de_proyectos_a_realizar_14-15>

En este documento especificamos cada punto acordado para el desarrollo del proyecto de nuestro subsistema y basándonos en la documentación a modo guía proporcionada por los profesores. La dividimos en varias áreas, subdivididas en procesos, en concreto:

* **Elementos de control:** se enumeran los elementos de gestión de la configuración y cómo se va a gestionar la configuración de cada uno de ellos.
* **Entorno de desarrollo:** se describe cuál es el entorno de desarrollo y la versión que se ha usado y los pasos para instalar nuestro sistema como los subsistemas relacionados para hacer funcionar el sistema completo.
* **Gestión del código fuente:** se describe cómo hacemos la gestión del código fuente durante el desarrollo de nuestro proyecto. En principio se creó un repositorio en Git para la gestión del código, pero hablando con los demás grupos de los demás subsistemas se acordó crear un repositorio en común con subcarpetas, una por cada grupo o subsistema para facilitar la integración o el trabajo de cada grupo en común con los demás subsistemas.
* **Gestión de la construcción y de la integración:** se describen las técnicas de integración continua, principalmente usamos Jenkins para automatizar el trabajo de las tareas cuando se produzcan cambios en el repositorio, se especificará con más detalle en el apartado correspondiente.
* **Gestión del cambio, incidencias y depuración:** se describen las herramientas y los procesos que se usan y como se usan.
* **Gestión de liberaciones, despliegue y entrega:** se describe el procedimiento usado para desplegar en una máquina los subsistemas de forma que estén integrados.
* **Gestión de la variabilidad:** se describen los mecanismos usados y los niveles en los que se gestiona la variabilidad.
* **Mapa de herramientas:** se describe las herramientas usadas en el nuestro proyecto.

El sistema a desarrollar consta de varios subsistemas que se dividen por cada grupo de clase formado en el aula, siendo estos:

* Verificación.
* Artefactos comunes.
* Cabina de votación.
* Autenticación.
* Recuento y modificación de resultados.
* Almacenamiento de votos.
* Creación y administración de censos.
* Deliberaciones.
* Creación y administración de votaciones
* Frontend y visualización de resultados.

Nuestro grupo se centrará en el desarrollo del proyecto del subsistema de **Autenticación**, que se encargará de la autenticación de los usuarios en el sistema antes de proceder a votar dentro del sistema, así como un formulario de registro para los usuarios que no están registrados en el sistema.

Para el desarrollo del trabajo, han sido necesarias las siguientes herramientas y tecnologías:

* **Virtual Box:** gestor de máquinas virtuales donde poder montar una máquina sobre la que desarrollar y poner en marcha el proyecto. Se tomó ésta decisión con la idea de conseguir un entorno homogéneo entre todos los miembros del equipo, evitando así incoherencias durante el desarrollo, y también para facilitar la integración con los demás subsistemas.
* **Windows XP 64 bits:** sistema operativo instalado en la máquina virtual. Es una distribución de Microsoft ya que los miembros del equipo están más familiarizado con este tipo de sistemas operativos.
* **GitHub**: plataforma de desarrollo escogida donde se creó el repositorio para alojar el proyecto. Fue escogido debido a que parecía el más adecuado por su facilidad de uso, y porque AGORA@US también estaba alojada en la plataforma GitHub.
* **Git**: software de control de versiones seleccionado para gestionar el código. Debido a su fácil integración en todo tipo de entornos, sin necesidad de entornos de desarrollo, y a que el equipo de desarrollo estaba familiarizado con el mismo, pareció la decisión más lógica.
* **Jenkins**: servidor de integración continua elegido para llevar la tarea de construcción e integración automática. El equipo es la herramienta con la que estaba más familiarizado para este tipo de tareas.
* **Aptana**: [entorno de desarrollo integrado](https://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado) de software libre basado en [eclipse](https://es.wikipedia.org/wiki/Eclipse_%28software%29) y provee soporte para [lenguajes](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) que utilizaremos en el proyecto. El equipo está más familiarizado con este entorno y por ello su elección.
* **FileZilla:** es un [cliente FTP](https://es.wikipedia.org/wiki/Cliente_FTP) [multiplataforma](https://es.wikipedia.org/wiki/Multiplataforma) de [código abierto](https://es.wikipedia.org/wiki/C%C3%B3digo_abierto) y [software libre](https://es.wikipedia.org/wiki/Software_libre), usado por ser gratuito y por su uso extendido.
* **Bitnami**: [entorno de pre-producción que utilizamos en nuestro proyecto](https://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado), integrado con Apache y MySQL, usado por ser gratuito y por su uso extendido.
* **Slack**: herramienta de comunicación para la organización y sincronización del equipo usada por su facilidad de uso en entornos empresariales.

# 2. Descripción del sistema

Como ya hemos comentado antes, nos encargaremos del subsistema de AGORA@US llamado **Autenticación**.

## 2.1 Descripción funcional

Como ya hemos comentado antes, nos encargaremos del subsistema de AGORA@US llamado **Autenticación**.

Nuestro subsistema se encargará de los primeros pasos cuando un usuario accede a AGOR@US, que consiste en decidir sobre una serie de opciones dependiendo de si se ha registrado antes en el sistema o no, o si hace uso de un servicio u otro para acceder a AGOR@US en definitiva cuatro opciones:

* Página de Bienvenida o index del sistema AGOR@US.
* Identificarse como usuario sin DNIe.
* Identificarse como usuario con DNIe.
* Registrarse en el sistema si aún no está registrado.
  + Página de Bienvenida o index del sistema agor@us

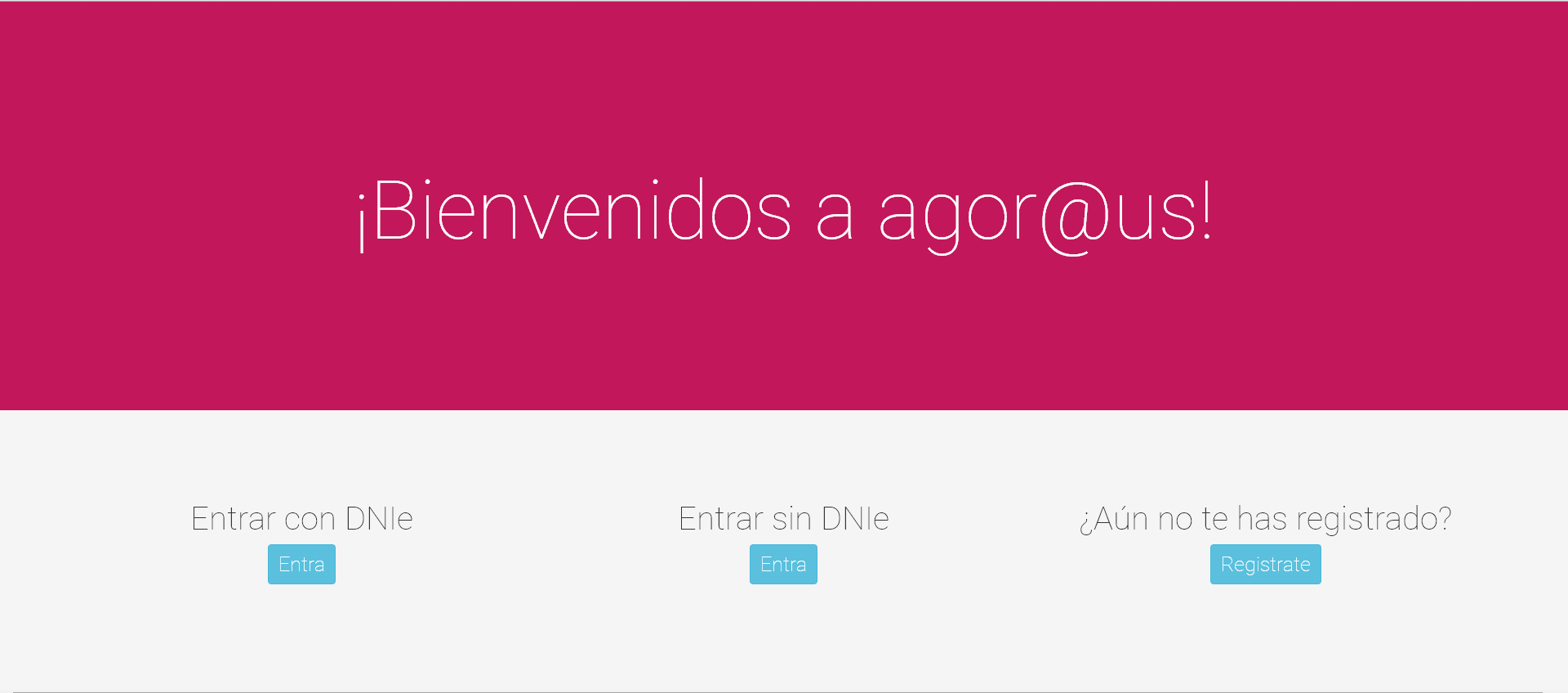


Figura 2. Index del sistema de AGORA@US

Cuando abrimos la aplicación AGOR@US lo primero que nos encontramos es esta página de bienvenida donde nos podemos encontrar con tres opciones: “Entrar con DNIe”, “Entrar sin DNIe” y “¿Aún no te has registrado?”, las cuáles iremos describiendo a continuación una a una de una manera más detallada.

* Identificándose como usuario sin DNIe **//FALTA LEYENDA DE LA TABLA**



El sistema de autenticación pide un nombre de usuario y una contraseña que tendrá que coincidir con algún usuario guardado en la base de datos, con ello el usuario podrá entrar en el sistema y realizar las distintas funcionalidades de los demás subsistemas. *Una vez autenticado se guardan dos cookies en el sistema, una con identificador “user”, donde guarda el nombre de usuario autenticado y otra con el identificador “token” en la que se almacenará un token generado a partir de su nombre de usuario y contraseña.(Texto recogido de la memoria del año anterior).*

El subsistema recogido de años anteriores también ofrece una API REST con peticiones de tipo GET para que se verifique que el usuario esta autenticado correctamente cada vez que se acceda a distintas funcionalidades del sistema, las peticiones tendrá el siguiente formato:

.

Figura 3. Peticiones GET sacado de la API del proyecto de EGC del año anterior.

* Identificándose como usuario con DNIe

//FALTA CAPTURA Y EXPLICACIÓN DE ESTA PARTE A ESPERAS DE CREACIÓN DEL MISMO (JOSE)

* Registro en el sistema

El subsistema también ofrece un registro para usuarios no registrados, donde solicita una serie de datos (Nombre de usuario, correo elctrónico, contraseña y confirmación de la misma, género, edad y comunidad autónoma) al usuario antes de poder entrar en el sistema, así como un generador de código captcha cuyo objetivo es distinguir a un ordenador de un ser humano, y de este modo, impedir que los robots (también llamados bots) realicen un uso indebido de nuestros servicios.

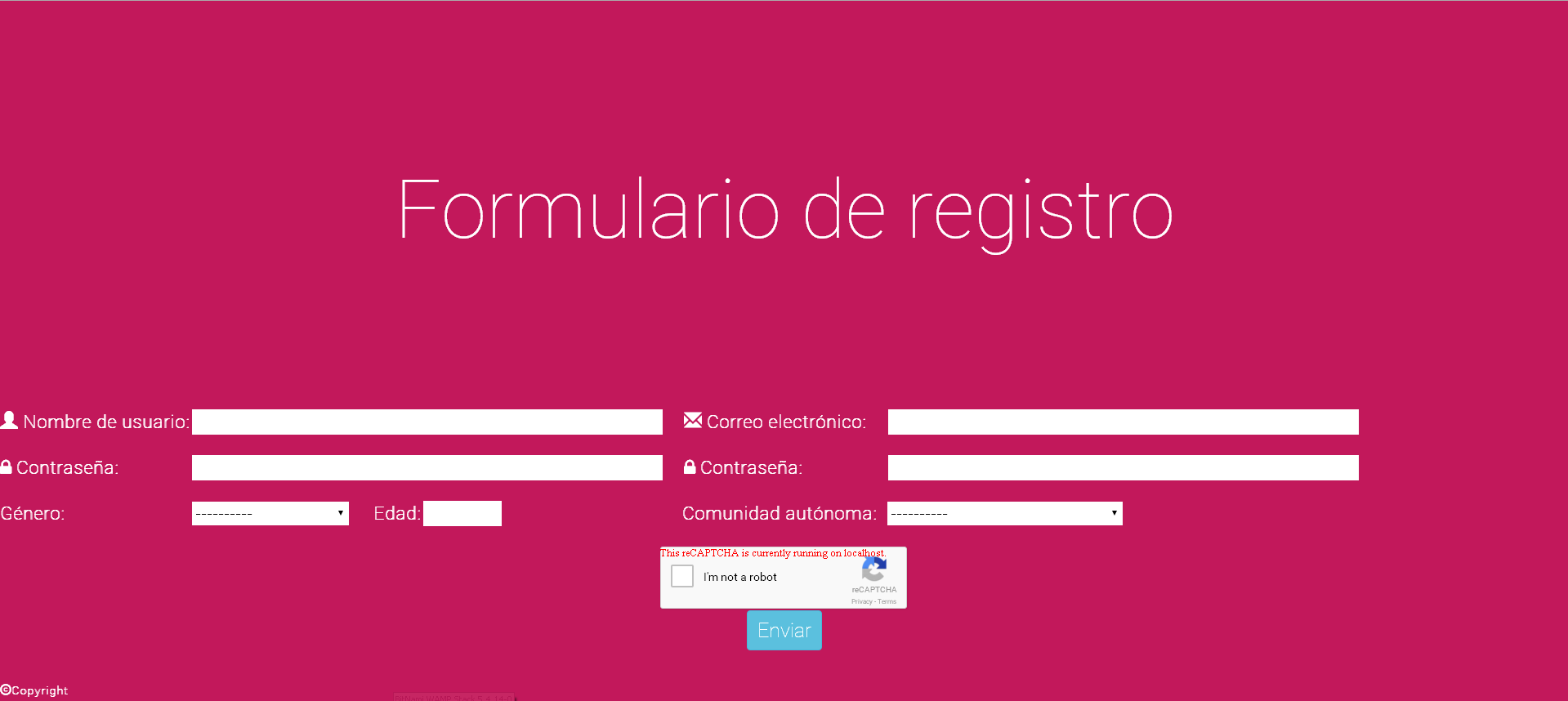


Figura 4. Formulario de registro del sistema AGORA@US

## 2.2 Descripción técnica

Nuestro proyecto está basado en código PHP donde solicita a los métodos de la API que se muestran en la siguiente tabla y mediante peticiones GET anteriormente mostradas, las distintas funcionalidades del aspecto técnico de nuestro subsistema.





Tabla . API REST ofrecida del proyecto de EGC del año anterior

**//Habría que poner que es del grupo de autenticación no?**

# 3. Elementos de control

Se enumerarán cuáles son los elementos que estarán bajo gestión de la configuración y cómo se va a gestionar la configuración de cada uno de ellos. Es decir, se explicitará la primera tarea de identificación de los Configuration Items (CI) que definan la base line.

- Los elementos de configuración (configuration ítems, CI) pueden ser productos finales o intermedios, productos entregables al cliente o productos internos.

Por ello, es preciso controlar los elementos de configuración que definen la baseline para conocer en qué situación se encuentran y disponer de una versión de cada elemento para toda persona involucrada en el desarrollo del proyecto.

- Los elementos de configuración que se encuentran bajo gestión de la configuración son los siguientes:

* Código fuente.
* Casos de prueba, realizados con PHP Unit para ejecutar los tests del sistema en PHP.
* Código objeto y ejecutable.
* Datos del proyecto.
* Documentos técnicos.
* Entorno: es uno de los elementos que más recursos y control requiere. Dentro de éste, se distinguen tipos de entornos.
* Entorno de desarrollo: Aptana
* Entorno de integración: Jenkins
* Entorno de pre-producción: Bitnami
* Entorno de producción: Servidor Apache



Figura 5. Diagrama de entornos

Se debe de indicar que el número de entornos depende de las diferentes organizaciones. Incluso dentro de una misma organización, los entornos pueden variar dependiendo del proyecto que se trate.

- Para gestionar la configuración de cada uno de los elementos mencionados anteriormente hay que definir directrices y procedimientos para:

* Gestión de licencias.
* Gestión de cambios.
* Gestión de versiones.
* Control de accesos.
* Control de construcciones.
* Despliegue.
* Instalación.
* Gestión de la integridad de configuraciones de línea base.

# 4. Entorno de desarrollo

- El entorno de desarrollo que se ha usado es Aptana Studio, versión 3.6.1.

Aptana es entorno de desarrollo integrado (IDE) de código abierto para el desarrollo de aplicaciones web basado en Eclipse. Aptana incluye soporte para HTML, XHTML, CSS, JavaScript, DOM, PHP y otros lenguajes a través de plugins.

- A continuación, se muestra los pasos que hay que seguir para instalar Aptana tanto en nuestro sistema como los subsistemas relacionados para hacer funcionar el sistema al completo.

1. En primer lugar, debemos descargar Aptana a través de<http://aptana.com>. En nuestro caso, hemos usado la versión 3.6.1.



Figura 6. Descarga de Aptana Studio 3.6.1

.

1. Una vez descargado, ejecutamos Aptana\_Studio\_3\_Setup\_3.6.1 y nos aparecerá una ventana tal que así:



Figura 7. Aceptar las condiciones e instalar

Tan solo debemos pulsar en aceptar e instalar y esperar unos minutos.

. 

Figura 8. Instalación de Aptana Studio 3.6.1

1. Una vez finalizado, aparecerá en nuestra computadora un nuevo icono denominado Aptana Studio. Hacemos doble click y nos mostrará una nueva ventana donde debemos seleccionar una carpeta que nos sirva como ‘workspace’.



Figura 9. Selección de carpeta para workspace

1. Al realizar el paso anterior, en Aptana existe el concepto de Proyecto. Un proyecto es un contenedor usado por el IDE para agrupar ficheros y carpetas relacionadas. Típicamente los proyectos agrupan los ficheros .html, .css, .js y.php de un único sitio web completo.

Para crear un nuevo proyecto en Aptana selección Archivo -> Nuevo -> Proyecto, y aparecerán un asistente que le guiará durante el proceso.

En nuestro caso, tan sólo debemos navegar hasta la carpeta BitNami/wampstack-5.4.14-0/apache2/htdocs, cuya referencia es precisamente donde tenemos clonado nuestro repositorio.

A continuación mostramos una captura de pantalla con la navegación realizada anteriormente:



Figura 10. Proyecto exportado en Aptana

# 5. Gestión del código fuente

Usaremos un repositorio Git llamado AgoraUS1516 para administrar el código fuente del sistema.

En este repositorio, que se ubica en el servicio de alojamiento de repositorios Git Github, cada subsistema se desarrollará en un repositorio diferente dentro del repositorio global. En el repositorio AgoaUS se recogerá el trabajo realizado y funcional de todos los grupos y su correspondiente integración.

Usar un repositorio global facilita el acceso al código por parte de cada grupo. Podemos obtener el código de otros grupos clonando sus repositorios y haciendo pull, obteniendo así el código más reciente del subsistema en cuestión.

Usar un único repositorio nos facilitaría aún más el acceso al código ya que solo tendríamos que hacer pull de la rama master para obtener el código de los demás subsistemas pero nos encontraríamos con el inconveniente de que cualquier persona podría modificar lo que quisiera, independientemente del grupo en el que se halle.

Este inconveniente no se nos presenta porque sólo tenemos acceso de lectura a los repositorios a los que no pertenecemos.

Tampoco se nos presentará problemas de conflictos con los demás grupos al alojarnos en repositorios diferentes.

Si usásemos un único repositorio tendríamos que crear una carpeta o una rama para cada subsistema con el fin de evitar numerosos conflictos que se presentarían al compartir muchos subsistemas la misma estructura de archivos.



Figura 11

La gestión del código fuente del repositorio de Autenticación se detallará a continuación:

## 5.1 Arquitectura y creación de documentos

**Introducción:**

En el directorio principal, en la rama master, encontramos la carpeta "auth", en la que se encuentra el contenido del subsistema. Hemos trabajado con una rama por funcionalidad. Así tenemos una rama llamada desarrolloDNIe para la funcionalidad del DNIe y una rama llamada Desarrollo-Captcha para la funcionalidad de verificación mediante captcha. Debido al reducido tamaño del subsistema todos los miembros del equipo trabajaremos en ambas ramas por lo que todas las modificaciones afectarán a toda la rama cada vez que hagamos push de la rama en cuestión. Es responsabilidad de cada miembro asegurarse de que las modificaciones realizadas no impacten negativamente en el funcionamiento del subsistema.

Mediante esta URL se puede acceder al repositorio: <https://github.com/AgoraUS1516/G03>

**Implementación:**

Utilizamos la carpeta local que aloja el repositorio en nuestro equipo para la creación y modificación de los ficheros. De este modo para que los cambios sean definitivos se debe realizar commit y luego push en el caso de una rama. Para que sean permanentes en la master tendríamos que hacer merge. Es responsabilidad individual de cada componente del proyecto asegurarse de que los cambios se hayan realizado correctamente antes de hacer push y de gestionar de forma correcta los conflictos generados.

**Lecciones aprendidas:**

Trabajar con una rama por funcionalidad nos permitió centrarnos en cada una por separado sin problemas de conflictos entre ambas. El hecho de no tener una rama por miembro nos permitió tener acceso a todos los cambios constantemente y poder trabajar sobre ellos de la forma más rápida y eficaz al tratarse de un subsistema de tamaño reducido al igual que el tamaño del equipo. La desventaja de emplear esta forma de trabajo es que si no se emplea una comunicación activa y compromiso puede generar problemas.

**Ejercicio:**

* Enunciado: El subsistema de autenticación se encuentra en un repositorio en Git y se quiere modificar para introducir un captcha. Desde que la web se creó no se había realizado ninguna modificación y se van a realizar algunas modificaciones más de la mencionada anteriormente. Además descubrimos que tenemos que modificar un archivo en el que está trabajando un compañero para realizar nuestra modificación. ¿Qué debemos hacer para realizar la modificación correctamente?
* Solución: En primer lugar tendríamos que clonar el repositorio en nuestro equipo. Para ello abrimos Git Bash como administrador y nos dirigimos en la consola a la carpeta donde queramos clonar nuestro repositorio. Una vez alli, hacemos clone y se nos creará una carpeta en el directorio escogido con todo el contenido del repositorio en la última versión. Una vez hecho esto tendremos que hacer checkout a la rama Desarrollo-Captcha. Posteriormente realizamos las modificaciones correspondientes y nos comunicamos con el compañero mencionado en el enunciado para avisar de la modificación que se va a realizar. Estas modificaciones tienen lugar en nuestro equipo por lo que cuando terminemos tendremos que hacer un commit para que los cambios se apliquen a nuestro repositorio local. Una vez todo está en orden hacemos push para que los cambios se apliquen al repositorio remoto y para que todos los miembros del equipo tengan acceso a las modificaciones realizadas.

## 5.2 Permisos

**Introducción:** Todos los miembros del grupo tendrán permisos de lectura y escritura en todo el repositorio.Al ser el equipo y el proyecto reducido vemos innecesario administrar los permisos pues tomaría más tiempo que cualquier problema que pueda ocasionar nuestro método de trabajo.

De esta forma el subsistema también es accesible a todos los miembros del grupo y si hay colaboración, la detección y corrección de errores puede ser más ágil.

**Implementación:** Hemos utilizado GitHub, que nos proporciona repositorios públicos y privados de manera que solo los miembros de un repositorio puedan realizar cambios en este. Pero carece de medidas de seguridad o permisos más allá de las mencionadas. De este modo hemos utilizado Git de manera habitual intensificando las comunicaciones entre los miembros del grupo.

**Lecciones aprendidas:** El hecho de no tomar en cuenta los permisos nos ha dado libertad en el acceso al proyecto por lo que no se han dado situaciones en las que un miembro del grupo no pueda modificar ciertas partes del proyecto. Esto tiene la desventaja de que cualquiera puede ocasionar un error grave en el proyecto, ya sea de forma maliciosa o no, pero es complicado que suceda esta situación. Al tener sólo los miembros del grupo acceso a nuestro subsistema este riesgo se minimiza notablemente.

**Ejercicio:**

* Enunciado: Un miembro del grupo comprueba que en nuestro subsistema, alojado en un repositorio en Git, falta una modificación menor que debería haber realizado otro miembro del grupo. ¿Cómo debería actuar?
* Solución: Ya que no tenemos restricción de permisos, al ser una modificación menor, en primer lugar, el miembro que ha comprobado que falta dicha modificación debería comunicarse con el miembro del equipo encargado de realizarla por ejemplo, mediante una incidencia en Git (Issue).

Posteriormente, debería realizar él mismo la modificación y hacer push para que los cambios se apliquen en el repositorio remoto. En el apartado de gestión de incidencias detallaremos como trabajamos con las Issues.

## 5.3 Gestión del código en relación al conjunto de subsistemas y nomenclatura

**Introducción:** En el repositorio común de integración para todos los subsistemas (AgoraUS) se introducirán todos los subsistemas para realizar finalmente su integración. Si los demás grupos necesitan acceso a otros subsistemas, al tener permisos de lectura sobre los repositorios de estos, podrán utilizarlos sin problema. Cuando se implemente la totalidad de la modificación del subsistema se debe hacer push al repositorio AgoraUS. El nombre de cada repositorio será GXX donde XX es el número de grupo. En la descripción de cada repositorio podemos observar el subsistema que aloja.

**Implementación:** Se creó un repositorio GXX por subsistema de manera que todos los grupos tienen permiso de lectura sobre todos ellos pero sólo el grupo encargado de un subsistema tiene permisos de escritura sobre este.

También se creó un repositorio de integración común llamado AgoraUS encargado de alojar todos los subsistemas para su integración. Se puede acceder a él mediante este enlace <https://github.com/AgoraUS1516/AgoraUS>.

**Lecciones aprendidas:** Debido a esta organización siempre hemos tenido acceso a todo el proyecto y con la seguridad de que sólo el grupo encargado de un subsistema en concreto puede realizar modificaciones sobre este. También la nomenclatura elegida ha hecho más ágil el acceso a las distintas partes del proyecto.

**Ejercicio:**

* Enunciado: Un miembro del grupo es el encargado de revisar el subsistema de autenticación ya modificado y pasarlo al entorno de integración. Además le falta una última tarea para la que necesita acceder al subsistema del grupo 5. ¿A qué repositorio debe acceder para obtener el subsistema que necesita? Cómo debe actuar para realizar todo esto?
* Solución: En primer lugar tendría que obtener el subsistema que necesita del repositorio G05 para realizar la última tarea y la revisión final del subsistema. Para ello tendrá que clonar en su equipo el repositorio G05 (haciendo clone) para tener acceso a la última versión del subsistema. Posteriormente realizará la tarea correctamente gracias a la información que necesitaba. Una vez hecho todo esto tendrá que introducir el proyecto en el repositorio de integración AgoraUS.

Para ello clonará el repositorio AgoraUS en su equipo, posteriormente incluirá el subsistema de Autenticación en la rama máster del repositorio. Cuando hayamos realizado todo esto correctamente deberá hacer commit y posteriormente push para que el subsistema quede correctamente alojado en el repositorio remoto AgoraUS.

**5.4 Gestión de conflictos**

**Introducción:** En el caso de que se genere un conflicto, será el miembro del grupo que lo divise primero el encargado de resolverlo. Para esto tendrá que ponerse en contacto con los demás miembros o con alguno en concreto si fuere necesario para acordar la versión escogida, etc.

**Implementación:** Utilizamos la consola y el sistema de resolución de conflictos de Git. Este sistema impide un commit o un push en el caso de que haya conflicto además de mostrar un aviso. Para resolver el conflicto, la persona que intentó la operación debe editar manualmente los archivos para quedarnos con el contenido adecuado. Una vez hecho esto, se realiza la operación de nuevo pero sin conflicto.

**Lecciones aprendidas:** La resolución de conflictos ha sido intuitiva y rápida, limitándose a conocimiento del proyecto y comunicación con los demás miembros del grupo.

**Ejercicio:**

* Enunciado: Durante el desarrollo del proyecto, dos componentes del grupo revisan el código buscando errores. Ambos modifican diversos archivos en común de manera que el segundo en subir los cambios provoca un conflicto. ¿Cómo debería actuar dicho miembro?
* Solución: En primer lugar tendría que localizar los conflictos y comprobar si puede resolverlos el sólo en el caso de que sean conflictos menores. Si no fuera ésta la situación, tendría que ponerse en contacto con el otro miembro para debatir cual sería la mejor versión final. Una vez hecho esto, el miembro del grupo resuelve el conflicto en función de lo que haya comentado con su compañero y sube los cambios al repositorio.

## 5.5 Aprobación de cambios:

Todos los cambios deben ser discutidos y aceptados por los miembros del grupo. Una vez haya quedado clara la forma de proceder, se implementa. La corrección de errores no se comunica a los demás miembros del grupo a no ser que la corrección traiga consecuencias mayores para el proyecto.

Todos los cambios son discutidos por los miembros del grupo. Casi siempre se llega a un acuerdo rápidamente pero hemos notado que, en ocasiones, tarde o temprano hay que volver a debatir el asunto.

Esta forma de trabajar nos ha funcionado bien debido a una buena comunicación entre todos los miembros del grupo. No ha supuesto mucho tiempo debatir sobre la mayoría de los cambios exceptuando algunos casos.

**Ejercicio:**

* Enunciado: Un miembro del grupo, revisando el proyecto, se da cuenta de que la realización del captcha estaría mejor de otra forma. Además ha encontrado algunos errores en ciertas partes del proyecto. ¿Cómo debería actuar?
* Solución: En primer lugar, debería ponerse en contacto con sus compañeros para debatir el por qué del cambio, el impacto que tendría y la mejor forma de realizarlo en caso de aprobarse su realización. Una vez llegado a un acuerdo, se organiza la implementación del cambio (si se decidió abordarlo) y se lleva a cabo. Con respecto a los errores, en caso de no ser errores de gran envergadura, deberán ser resueltos por el propio miembro del grupo que está revisando el subsistema. Si se tratase de un error mayor sería conveniente que se reuniera con los demás miembros para debatir su solución.

## 5.6 Gestión de las ramas del código

**//Seguramente este apartado vaya al principio**

Inicialmente trabajamos en la rama master para cargar el subsistema sobre el que trabajamos y para crear la jerarquía de directorios. Una vez hecho esto, creamos dos ramas. Usaremos una rama para cada funcionalidad que añadamos al subsistema, en nuestro caso, una rama para introducir el acceso mediante DNIe (llamada desarrollo DNIe) y otra para introducir la verificación mediante captcha (llamada Desarrollo-Captcha). Además, las mejoras de estilo se realizarán en la rama master.**//Esto estaría bien?**

Todos los miembros del equipo trabajaremos sobre todas las ramas ya que todos tenemos tareas correspondientes a las dos nuevas funcionalidades que implementamos.

Una vez que cada funcionalidad esté completa y operativa se une la rama correspondiente a la master.

Nos referiremos a los comandos de git como detallamos a continuación a lo largo de todo el documento. Los comandos de Git que utilizaremos serán los siguientes:

1. clone:
   1. git clone https://github.com/AgoraUS1516/G03.git (en el caso de nuestro repositorio): para clonar el repositorio remoto en nuestro repositorio local.
2. commit:
   1. git add -A: para añadir todos los nuevos cambios a la lista de commit.
   2. git commit -a: para guardar los cambios añadidos mediante el comando anterior en el repositorio local.
3. push:
   1. git push origin nombreDeLaRama: para subir los cambios guardados en la rama en cuestión en el repositorio local al repositorio remoto.
4. pull:
   1. git pull: para actualizar el estado del repositorio local con el estado del repositorio remoto.
5. branch de creación:
   1. git branch nombreDeLaRama: crea una nueva rama.
6. merge:
   1. git merge nombreDeLaRama: desde la rama master se ejecuta este comando para unir la rama en cuestión con la rama master.
7. branch de consulta:
   1. git branch: consulta las ramas existentes.
8. checkout:
   1. git checkout nombreDeLaRama: cambia de la rama actual a la rama en cuestión.
9. branch -D:
   1. git branch -D nombreDeLaRama: borra la rama en cuestión.

**Ejercicio:**

* Enunciado: Un miembro del equipo se encarga de crear las ramas correspondientes a las dos funcionalidades que implementaremos. Además comienza a trabajar una vez ha hecho esto en una de las funcionalidades. ¿Qué tiene que hacer para crear ambas ramas? ¿Qué tiene que hacer una vez deje de trabajar en la tarea que comenzó? ¿En el futuro, cuando una funcionalidad esté lista, ¿cómo tiene que proceder el miembro del equipo que se encargue de esto?
* Solución: En primer lugar, una vez esté lista la arquitectura del proyecto, tendrá que crear una rama por funcionalidad haciendo un branch de creación. Una vez hecho esto, tendrá que hacer un checkout a la rama de la funcionalidad en la que trabaje y comenzar a desarrollar su tarea. Cuando haya terminado, tendrá que hacer commit y push para que los cambios se apliquen en el repositorio remoto. Cuando finalice el desarrollo de una funcionalidad y está lista para ser aplicada definitivamente al subsistema, tendrá que hacer merge de la rama que corresponda para unirla a la master.

## 5.7 Roles

El único rol que existe es el de desarrollador. No tenemos jefe de proyecto y ningún miembro del equipo tiene más poder o más responsabilidad que otro. Cada miembro se encargará de la tarea que le corresponde según el acuerdo al que ha llegado el equipo y todas las decisiones que conciernan al proyecto serán tomadas por la totalidad del grupo.

## 5.8 Aplicación de un parche (patch)

No utilizaremos un parche de manera habitual. De hecho es muy probable que no lleguemos a utilizarlos. Esto sería en el caso de que necesitemos cambiar un archivo que se encuentre en todas las ramas. Esto lo haríamos para evitar tener que unir las ramas a la master para poder aplicar el cambio antes de terminar con las funcionalidades.

En el caso de ser necesario, el miembro del equipo encargado de realizar el cambio, una vez lo haya realizado, tendrá que generar el parche y distribuírselo a los demás miembros.

Para generar un parche haremos lo siguiente:

* git format -patch -1 HEAD: Esto genera un fichero que contiene la diferencia incremental (también conocido de manera popular como *parche* o *patch* en inglés). El primer parámetro indica el número de cambios a exportar, el segundo indica el commit a partir del cual quieres exportar.
* git format -patch -N COMMIT -ID: Donde 'N' se reemplaza por el número de cambios a exportar. El *COMMIT-ID* indica el primer cambio a exportar, es decir, el cambio de referencia.

*\*El contenido está sacado de la web de la asignatura:* [*https://1984.lsi.us.es/wiki-egc/index.php/Exportando\_cambios:\_Creando\_una\_diferencia\_incremental*](https://1984.lsi.us.es/wiki-egc/index.php/Exportando_cambios:_Creando_una_diferencia_incremental)

## 5.9 Políticas y nombre de estilos

**//En este apartado no entiendo bien que debo incluir :S**

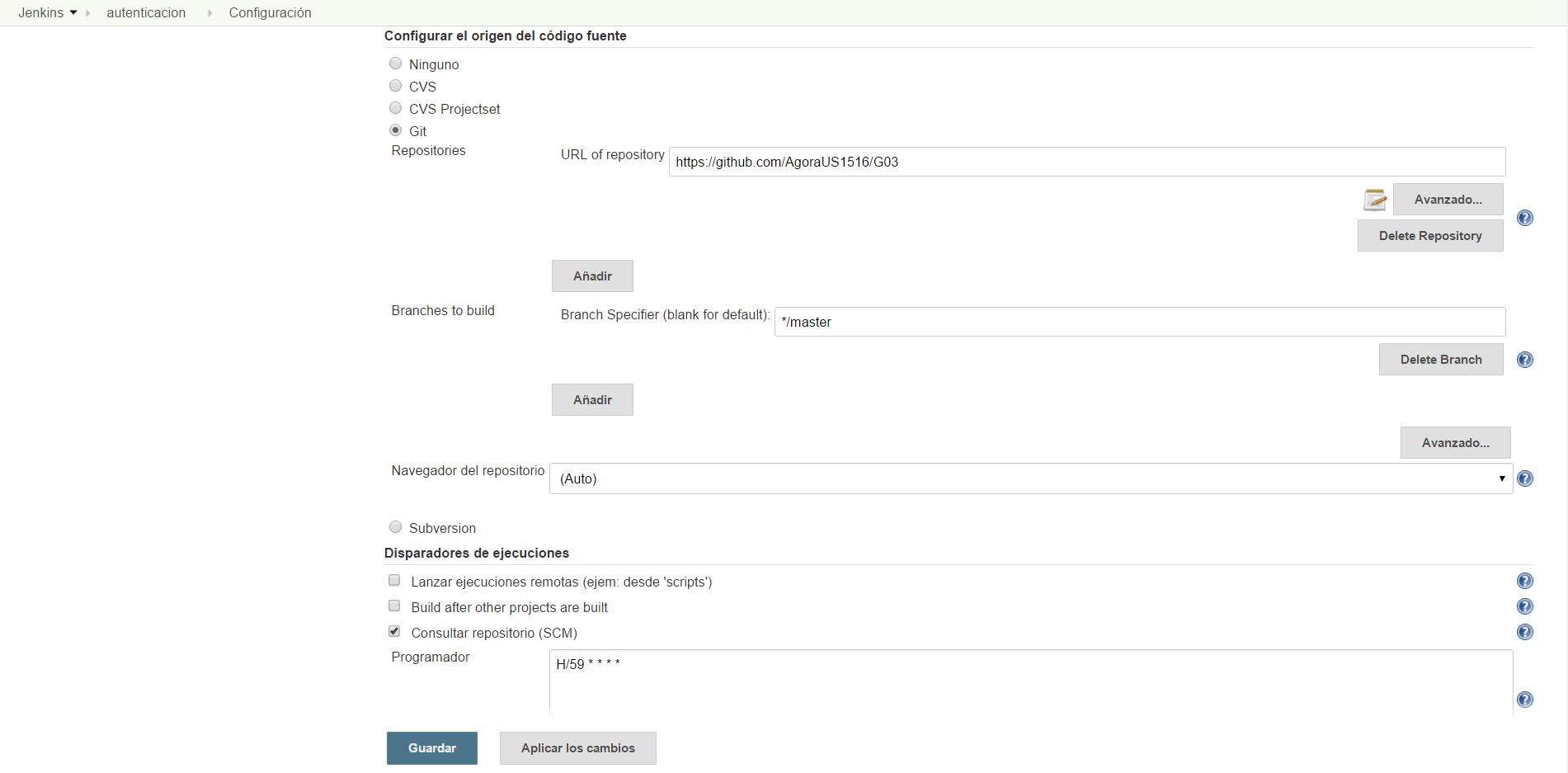
## 5.10 Usage model

**Aún por pulir**

# 6. Gestión de la construcción e integración continua

## 6.1 Gestión de la construcción

## Para la integración de los distintos componentes del subsistema se ha optado por usar un método de integración “bottom-up”, es decir, las partes individuales se diseñan con detalle y luego se enlazan para formar componentes más grandes, que a su vez se enlazan hasta que se forma el sistema completo. Para facilitar esta integración se ha optado por usar un servidor de Jenkins y un plugin de Github para Jenkins que automatice los procesos de integración. La comprobación de si se ha realizado alguna modificación en el repositorio será cada hora.



**//FALTA LEYENDA DE LA TABLA**

## 6.2 Gestión de la integración continua

El proceso de integración continua tiene como objetivo principal comprobar que cada actualización del código fuente desarrollado por el equipo de trabajo no genere problemas en una aplicación que se está desarrollando.

Por lo tanto, la integración continua se usa como práctica de desarrollo software donde todos los componentes del equipo integran a lo largo del tiempo de vida del proyecto su trabajo, con el fin de conseguir un software de mayor calidad.

No hay que pensar que la integración continua es tener instalado el servidor de integración continua (por ejemplo Jenkins, el cual será mencionado más adelante) y que éste compile el código periódicamente o tener automatizados los despliegues dándole a un botón. Estaríamos cometiendo un grave error.

- Las ventajas que ofrece la integración continua son las siguientes:

* Ejecución automática de pruebas unitarias, conociendo a su vez los resultados de éstos.
* Ofrece información de cualquier incidencia tras un período de codificación, dándonos la posibilidad de saber quién y cuándo ha realizado un cambio de impacto negativo en el proyecto.
* Permite detectar los errores lo más pronto posible, en fases tempranas de desarrollo, dando la posibilidad de encontrar solución cuanto antes.
* Monitorización continua de las métricas de calidad del proyecto.

## 6.2.1 Herramientas

Para la integración continua de nuestro proyecto, se ha decidido usar la herramienta Jenkins, ya que ha formado parte del contenido aprendido de la asignatura Evolución y Gestión de la configuración, concretamente en las sesiones de prácticas.

Jenkins es un software de integración continua. Se trata de un instrumento de carácter open-source, el cual podemos instalar dicho servidor tanto en local como en la nube.

## 6.2.2 Uso de herramientas

Gracias a Jenkins, tenemos la oportunidad de gestionar cualquier tipo de proyecto software. La base de Jenkins es la creación de tareas bajo las cuales podemos configurar las comprobaciones del repositorio de código cada cierto periodo de tiempo y que dado cambio en dicho repositorio, éste se compile y se ejecuten todas las pruebas que tengamos definidas.

Es imprescindible tener un repositorio de control de versiones, como por ejemplo Git, donde almacenamos todo el código necesario para realizar la construcción de éste, sacando así partido y provecho a la herramienta ya que es capaz de monitorizar el control de versiones y actuar ante cualquier cambio.

Jenkins tiene la capacidad de notificar, mediante vía email o cualquier otro medio al responsable del equipo de trabajo, cualquier incidencia producida. De ser lo contrario, si el resultado ha sido todo un éxito, la aplicación estará lista para el despliegue, gracias a los archivos generados.

Además, Jenkins ofrece la posibilidad de lanzar métricas de calidad e incluso visualizar los resultados dentro de la propia herramienta siendo éste una función interesante a destacar.

NOTA:para instalar la herramienta basta con seleccionar el instalador deseado (<https://jenkins-ci.org/>).

## 

## 6.2.3 Tiempo entre construcción de los proyectos

**Ni idea**

## 6.2.4 Ejemplo propuesto

Una vez explicado en qué consiste la gestión de la integración, mencionado en la sección 6.2, así como la herramienta que usaremos a continuación para llevar a cabo dicho proceso, se propone un enunciado con la finalidad de poner a prueba y en su conjunto todo lo relacionado con la gestión de la integración.

* Enunciado:

1. Crear una tarea en Jenkins e indicar que debe usarse el código del repositorio <https://github.com/AgoraUS1516/G03.git>. Además, forzar la comprobación de si se ha realizado alguna modificación en el repositorio para automatizar en caso afirmativo los procesos de integración.
2. Forzar la ejecución de la tarea una vez.

* Solución:

1. Accedemos al servidor de Jenkins mediante la ruta en el navegador: localhost:8080/. Una vez en la pantalla inicial, seleccionamos “Nueva tarea” y escribimos nombre y marcamos “Proyecto de estilo libre”.

A continuación, en la pantalla de configuración de la tarea en las opciones de “Configurar el origen del código fuente” seleccionamos Git y escribimos el repositorio indicado en el enunciado y dejamos la rama \*/master. Por último, en la sección de “Disparadores de ejecuciones” seleccionamos solo “Consultar repositorio (SCM)”, escribimos “H/59 \* \* \* \*” y guardamos la configuración.

1. Seleccionar construir ahora.

# 7. Gestión del cambio, incidencias y depuración

## 7.1 Peticiones de cambio e incidencias

**Abstracción:** Es necesaria una comunicación entre todos los grupos para ver cómo va el desarrollo del proyecto y para generar incidencias. De esta forma establecimos varios canales de comunicación. En primer lugar tenemos un representante de cada grupo y todos los representantes están en un grupo de WhatsApp para comunicar todo lo que sea necesario. Por otro lado utilizamos también para la gestión de incidencias los issues de Git

**Implementación:** Mediante WhatsApp, los coordinadores del proyecto se ponen en contacto para conocer el estado de los diferentes subsistemas y para mantenerse siempre informados y operativos tanto para preguntar cualquier duda sobre algún subsistema como para responderla, etc.

Por otro lado, para la gestión de incidencias utilizamos los issues de GitHub. Esto es una funcionalidad de GitHub que facilita la comunicación entre todos los desarrolladores ya que permite generar una incidencia, leer las diferentes incidencias que se encuentren abiertas o cerradas, etc. Es una forma de informar de forma rápida y sencilla a todos los desarrolladores.



Figura 12

.

Como tenemos un repositorio por subsistema, las incidencias que repercutan al proyecto global serán recogidasen el repositorio de AgoraUS, dejando las incidencias del repositorio de cada subsistema para la comunicación de cada grupo.

Durante la creación de una issue podemos darle un título, asignarle un responsable de encargarse de gestionar la issue e introducir una descripción que tiene la posibilidad de incluir imágenes.



Figura 13

.

El proceso que emplearemos será el siguiente:

1. Un miembro de un grupo necesita algo relacionado con nuestro subsistema.
2. Dicho miembro crea un issue detallando su incidencia en AgoraUS.
3. Recibimos una notificación de que se ha generado una issue.
4. Estudiamos el issue.
5. Si podemos se llega a la conclusión de que no podemos atender la issue, se rechaza la petición comentando en la issue el motivo de rechazo.
6. Si se llegara a la conclusión contraria se notifica al autor de la issue que estamos trabajando en su incidencia.
7. se trabaja en el cambio.
8. Una vez atendida la petición se comunica al autor del issue
9. El autor del cambio puede cerrar el issue si está satisfecho o mantenerlo abierto en caso de que no esté satisfecho con nuestra respuesta.

**Lecciones aprendidas:** Los issue han facilitado mucho la comunicación entre los grupos y entre los miembros de cada grupo. Como todos los grupos usamos Git, nos aseguramos de que la información llegue a nuestros compañeros. No obstante no se ha utilizado tanto como WhatsApp al ser este proyecto mucho más centrado en cada subsistema que en el sistema completo. Sobre todo se ha utilizado por problemas de integración.

**Ejercicio:**

* Enunciado: Tratando de integrarnos con otro subsistema hemos descubierto un error en su código que impide la integración. ¿Cómo debemos actuar?
* Solución: En primer lugar debemos generar una issue en el repositorio de AgoraUS. Debemos completar todos los campos así como el título, el responsable encargado de gestionar la issue y la descripción.

La descripción cuanto más detallada sea mejor. Sería conveniente incluir imágenes y capturas. Hay que tener en cuenta que el otro grupo tiene que entender el problema de la mejor manera posible para su mejor resolución.

## 7.2 Depuración

**Abstracción:** Si recibimos una incidencia, la depuración se realizará de la siguiente forma:

1. Se analizará la issue recibida. Analizamos la información añadida, en caso de que esté poco detallada, pediremos que se amplíe la información y volveremos a analizar la issue. Si la información es correcta, pasamos al siguiente paso.
2. Estudiaremos el contenido de la issue e intentaremos localizar el foco del problema si se tratara de un error.
3. Buscaremos la causa del error e intentaremos su resolución si se tratara de un error.
4. Por último diagnosticamos el resultado de la issue al que la redactó.

**Implementación:** Si se trata de una petición simple, el primer miembro que vea la issue será el encargado de resolverla. Si es un error de código, en primera instancia el responsable de resolver la issue será el que codificó esa parte y si se trata de un error mayor, nos reuniríamos los miembros del grupo para tratar su solución.

**Lecciones aprendidas:** Al trartarse de un proyecto pequeño y que solo tenemos que modificar una aplicación ya existente, no surgieron muchos problemas de código. Los problemas principales fueron de integración. En la mayoría de las issues se piden datos de configuración y de hosting.

**Ejercicio:**

* Enunciado: Recibimos una issue de que no es posible integrar con nuestro sistema. ¿Qué procedimiento empleamos para resolver la issue?
* Solución: En primer lugar recibimos la issue e intentamos resolverla porque hemos sido los primeros en recibirla. Primero reproducimos el error y a partir de ahí buscamos la causa. Posteriormente evaluamos el impacto y en función del resultado decidimos si resolverla o comunicárselo al equipo para decidir sobre su resolución. Una vez tengamos un resultado, favorable o no, enviamos una respuesta a la issue recivida.

# 8. Gestión de las liberaciones, despliegue y entregas

En esta sección, abarca toda la información referente a los distintos elementos y/o entregables necesarios para la entrega, así como toda aquella documentación que lo compone, proyecto desarrollado por todo el equipo de trabajo y la correspondiente máquina virtual para el despliegue.

## 8.1 Entregables

Nuestro trabajo contendrá una serie de entregables a desarrollar por los distintos componentes del grupo, donde se detallará a continuación cada uno de ellos.

* Documentación del proyecto: documento formal, técnico y profesional que abarca todos y cada uno de los apartados requeridos por el documento TrabajoEGC\_15\_16, ubicado en el siguiente enlace: <https://1984.lsi.us.es/wiki-egc/index.php/Archivo:TrabajoEGC_15_16.pdf>
* Máquina virtual: software que contiene todos los elementos y herramientas necesarias para realizar el proyecto en su totalidad y su correspondiente despliegue.
* Diario de grupo/equipo: documento que refleja todas las actividades desempeñadas durante la realización del proyecto, así como las decisiones tomadas para alcanzar el objetivo de éste. Incluye actas de reuniones y tareas asignadas a cada miembro del grupo.

## 8.2 Entrega

Una vez finalizado todos y cada uno de los entregables necesarios (indicados en la sección 8.1) para el desarrollo del proyecto, subimos los archivos pertinentes al portal web: <https://opera.eii.us.es/egc/public/>

En nuestro caso, el grupo creado se denomina “Autenticación” (Grupo 3) cuyo enlace correspondiente es: <https://opera.eii.us.es/egc/public/default/grupo/ver/id/30>

## 8.3 Roles

En el momento de entrega, se creado cierta distinción entre los distintos miembros del grupo.

En nuestro caso, un miembro del grupo será el encargado de preparar la máquina y subirlo donde corresponda, mientras que otro miembro del grupo se encargará de finalizar el documento de diario de grupo.

A continuación, otro compañero de grupo se encarga de unificar los documentos y realizar los últimos retoques tanto del formato como de la propia documentación.

Por último, otro compañero de grupo se encargará de la subida del comprimido a Ópera.

## 8.4 Mecanismo de despliegue

**TEXTO RECOGIDO DEL GRUPO 4. CREO QUE PUEDE SERVIRNOS. PROHIBIDO EL PLAGIO DE ÉSTE**

A la hora de desplegar, se debe ejecutar la máquina virtual proporcionada a tal efecto. La ventaja de utilizar una máquina virtual es que no hay que preocuparse por disponer una a una de todas las herramientas en la versión utilizada, sino que en un mismo entorno se puede disponer de todo ello directamente.

Como herramientas, dado que se hace uso de un servidor Apache y de una base de datos relacional MySQL, se emplea un servidor XAMPP; además, es necesario un servidor Tomcat para desplegar y acceder a la aplicación.

Dicha máquina está elaborada mediante VirtualBox, como se especificó en clase.

# 9. Mapa de herramientasMapa de herramientas

Figura . . Mapa de herramientas

\*Modelo de imagen obtenido del entregable final del grupo de “Autenticación” del curso anterior.

## 9.1 Descripción del mapa de herramientas

En esta sección, se van a definir cada una de las herramientas que dan soporte y sirven de base para el desarrollo del código, así como el despliegue del mismo. Para ello, describiremos poco a poco cada uno de los elementos representados en la figura 14, expuesta en el apartado anterior.

En primer lugar, indicar que todos los grupos han utilizado Wiki de la asignatura Evolución y Gestión de la Configuración ([EGC](http://1984.lsi.us.es/wiki-egc)), por lo que aparece como nexo común a todos ellos. Gracias a ella, descargamos la aplicación desarrollada por los alumnos que cursaron la asignatura el año pasado y en ella, a su vez, se fueron documentando el desarrollo del proyecto (actas de reunión, repartos de tareas, subsistemas relacionados, etc).

Para la gestión del código fuente se ha decidido utilizar git como gestor y GitHub como servidor donde alojar el código. Todos los grupos participan en un repositorio común llamado [AgoraUS1516](https://github.com/AgoraUS1516/), en el que sólo los jefes de cada grupo tienen permiso de escritura. Ese repositorio central almacena en su interior un repositorio individual para cada grupo, cuya nomenclatura es “G” + número de grupo (por ejemplo, “[G03](https://github.com/AgoraUS1516/G03)”), representando así cada uno de los subsistemas que componen AgoraUS. A su vez, se ha decidido utilizar GitHub para las incidencias, mediante el módulo “issues”.

Se ha preparado una máquina virtual con la configuración necesaria para hacer funcionar el proyecto. Las herramientas principales son:

* **Aptana:** entorno de desarrollo integrado (IDE) para el desarrollo de nuestro código fuente (HTML, CSS, PHP, etc) usando concretamente la versión 3.6.1. Su definición e instalación se encuentra en la sección 4, detallado anteriormente.
* **Bitnami:** es un instalador multiplataforma de aplicaciones web de software libre cuyo objetivo es facilitar la integración y configuración de gran cantidad de aplicaciones. Además, instala todos los elementos que requiere el funcionamiento de la aplicación como puede ser un servidor HTTP Apache, o una base de datos MySQL (más detallado en la sección 8.4).

Hay que indicar que BitNami BitNami crea paquetes, que llama stacks o pilas, que contienen todo lo necesario (programas, scripts, bases de datos, dependencias de librerías resueltas,...) para la instalación de la aplicación.

NOTA: los stacks que hay que descargar se encuentran en el siguiente enlace [**http://bitnami.org/stacks**](http://bitnami.org/stacks)

* **MySQL:** es un sistema de gestión de base de datos relacional o SGBD que permite ser utilizado por varias personas al mismo tiempo. Archiva datos en tablas separadas, dando lugar a una mayor velocidad y flexibilidad. Las tablas están conectadas por relaciones definidas que hacen posible combinar datos.

La aplicación va a ser desplegada en un servidor Apache en el que Jenkins se encargará de la gestión de la integración continua, tal y como se explicó en la sección 6.2 de este documento. En concreto, accederá al repositorio central y extraerá todo el código de la aplicación, lo compilará y le realizará una serie de tests para comprobar que todo funciona adecuadamente. Asimismo, la base de datos será desplegada en un servidor distinto.

# 10. Conclusiones

Como resultado tras la ejecución de este proyecto a lo largo del cuatrimestre, tenemos que mencionar en primer lugar la dificultad de mejorar un código desarrollado por alumnos que cursaron la asignatura de Evolución y Gestión de la Configuración el curso pasado, así como la interpretación de éste tanto su estructura como su funcionamiento. Por lo tanto, nos enfrentamos ante un primer muro para la realización de este proyecto ya que no nos habíamos enfrentado a este tipo de situación anteriormente, dedicando así bastante tiempo para el entendimiento de éste.

En segundo lugar, cabe destacar la necesidad de re-factorizar el código heredado ya que la estructura del código era un poco caótica dificultando así su comprensión. A su vez, indicar los problemas que han ocurrido con los demás subsistemas durante la ejecución del proyecto ya que no nos proporcionaron la información necesaria para integrar nuestro subsistema con los demás.

Finalmente, mencionar que este proyecto nos ha servido para enfrentarnos ante la complejidad de mejorar un código heredado, en nuestro caso, mejorar el subsistema de autenticación y la integridad de éste con los demás subsistemas pertenecientes a AgoraUs ya que es necesario una buena gestión de comunicación entre los subsistemas que contienen dependencias y gestión de incidencias para conseguir que todo funcione correctamente en su conjunto y poder así alcanzar los objetivos marcados por dicha asignatura.

# 11. Bibliografía

La documentación que nos ha servido como base para desarrollar este proyecto y que nos ha servido de guía para documentar ciertas secciones de la presente memoria es la siguiente:

* <https://1984.lsi.us.es/wiki-egc/index.php/Teor%C3%ADa_-_15/16>
* <https://1984.lsi.us.es/wiki-egc/index.php/Hoja_de_tiempos_grupo_autenticaci%C3%B3n_2014-2015>
* <https://1984.lsi.us.es/wiki-egc/index.php/Pr%C3%A1cticas_-_15/16>
* <https://1984.lsi.us.es/wiki-egc/index.php/Archivo:TrabajoEGC_15_16.pdf>
* <https://es.wikipedia.org/wiki/Integraci%C3%B3n_continua>
* <http://recursostic.educacion.es/observatorio/web/en/software/software-general/767-bitnami-instalador-de-aplicaciones-web>