|  |
| --- |
| 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 3](#_Toc441705007)

[Enlaces 5](#_Toc441705008)

[Índice de figuras 6](#_Toc441705009)

[Índice de tablas 7](#_Toc441705010)

[1. Resumen 8](#_Toc441705011)

[2. Introducción 9](#_Toc441705012)

[3. Descripción del sistema 11](#_Toc441705013)

[4. Elementos de control 13](#_Toc441705014)

[5. Entorno de desarrollo 15](#_Toc441705015)

[6. Gestión del código fuente 19](#_Toc441705016)

[6.1 Arquitectura y creación de documentos 20](#_Toc441705017)

[6.2 Permisos 22](#_Toc441705018)

[6.3 Gestión del código en relación al conjunto de subsistemas y nomenclatura 23](#_Toc441705019)

[6.4 Gestión de conflictos 24](#_Toc441705020)

[6.5 Decisión de cambios: 25](#_Toc441705021)

[7. Gestión de la construcción e integración de la construcción 26](#_Toc441705022)

[8. Gestión del cambio, incidencias y depuración 27](#_Toc441705023)

[8.1 Peticiones de cambio e incidencias 27](#_Toc441705024)

[8.2 Depuración 29](#_Toc441705025)

[9. Gestión de las liberaciones 31](#_Toc441705026)

[10. Mapa de herramientas 32](#_Toc441705027)

[10.1 Descripción del mapa de herramientas 32](#_Toc441705029)

# Control de versiones

|  |  |  |  |
| --- | --- | --- | --- |
| Versión | Fecha | Modificaciones | Miembro del grupo |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Enlaces

# Índice de figuras

[Figura 1. Arquitectura de Agora Voting. 8](#_Toc441686417)

[Figura 2. Index del sistema de AGORA@US 10](#_Toc441686418)

[Figura 3.Formulario de registro del sistema AGORA@US 10](#_Toc441686419)

# Índice de tablas

[Tabla 1. API REST ofrecida del año anterior. 11](file:///C:\Users\AWE\Desktop\Memoria.docx#_Toc441686431)

# 1. 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 proyecto 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 comentado.

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 va actualizando el desarrollo de cada subsistema y hacer más fácil la integración completa de todos los subsistemas.

# 2. 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 más 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.

# 3. Descripción del sistema

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 dos opciones dependiendo de si se ha registrado antes en el sistema o no, en definitiva dos opciones:

* Identificarse como usuario.
* Registrarse en el sistema si aún no está registrado.
* Identificándose como usuario

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).*



Figura 2. Index del sistema de AGORA@US

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:

.

Registro en el sistema

El subsistema también ofrece un registro para usuarios no registrados, donde solicita una serie de datos al usuario antes de poder entrar en el sistema.



Figura 3.Formulario de registro del sistema AGORA@US





Tabla 1. API REST ofrecida del año anterior.

# 4. 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: Jenkings
* Entorno de pre-producción: Bitnami
* Entorno de producción: Servidor Apache



Figura X. 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. (Se ha omitido el asterisco de este párrafo)

- 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.

# 5. 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. Se descarga desde *http://aptana.com*

- 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 X. 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 X. Aceptar las condiciones e instalar

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



Figura X. 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 x. 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 X. Proyecto exportado en Aptana

# 6. Gestión del código fuente

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

En este repositorio, que se aloja 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 aun 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 X.

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

## 6.1 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. No hemos trabajado con ramas debido al reducido tamaño del subsistema por lo que todas las modificaciones afectarán a todo el subsistema cada vez que hagamos push. 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. 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:**

Usar un directorio único para alojar el subsistema nos permitió trabajar de forma más cómoda con la totalidad del proyecto. Si el proyecto estuviese en fase de creación, posiblemente hubiese sido más conveniente utilizar una subdivisión en ramas para poder tener el proyecto actualizado y evitar conflictos pero, debido a que los cambios los realizaremos en partes concretas del proyecto, no lo vemos necesario ya que este hecho minimiza la generación de conflictos y la necesidad de acceder a funcionalidades no realizadas por otros miembros del grupo al encontrarse el proyecto operativo. La desventaja de emplear esta forma de trabajo es que si no hay una comunicación activa y una buena organización 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, escribimos el comando git clone https://github.com/AgoraUS1516/G03.git y se nos creará una carpeta en el directorio escogido con todo el contenido del repositorio en la última versión. 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. Para ello nos dirigimos a la carpeta que contiene a la rama master (dentro del clon que realizamos inicialmente) mediante la consola de Git Bash y ejecutamos el comando git add -A para añadir todos los cambios que se commitearán. Posteriormente escribimos el comando git commit -a para enviar todos los cambios al repositorio local. Una vez todo está en orden ejecutamos el comando git push origin master para que los cambios se apliquen al repositorio remoto y para que todos los miembros del equipo tengan acceso a las modificaciones realizadas.

## 6.2 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 mas allá de las mencionadas. De este modo hemos utilizado Git de manera habitual intensificando las comunicación 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 (en el apartado anterior se detalla el proceso) para que los cambios se apliquen en el repositorio remoto. En el apartado de gestión de incidencias detallaremos como trabajamos con las Issues.

## 6.3 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 alojara todos los subsistemas para su integración.

**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 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. Una vez hecho todo esto tendrá que introducir el proyecto en el repositorio de integración AgoraUS.

## 

## Gestión de conflictos

**Abstracción:** 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 esta 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.

## 6.5 Decisión de cambios:

**Abstracción:** 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.

**Implementación:** 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.

**Lecciones aprendidas:** 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 una funcionalidad del subsistema estaría mejor de otra forma. Además ha encontrado algunos errores en ciertas partes del proyecto. ¿Cómo debería actuar?
* Solución: Debería ponerse en contacto con sus compañeros para debatir el por qué del cambio y la mejor forma de realizarlo. Una vez llegado a un acuerdo, se organiza el cambio para su posterior realización. 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. Si se tratase de un error mayor sería conveniente tratar el asunto con los demás miembros.

# 7. Gestión de la construcción e integración de la construcción

# 8. Gestión del cambio, incidencias y depuración

## 8.1 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 X. Diagrama de entornos

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 X. Diagrama de entornos

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.

## 8.2 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.

# 9. Gestión de las liberaciones

Nuestros entregables son varios archivos PHP, CSS y JavaScript que, en conjunto, forman la estructura de autenticación de la plataforma. Existe, a su vez, un fichero con la estructura básica de la base de datos que incorporará la aplicación para la gestión de los datos de usuarios. Cada versión, etiqueta los archivos con “”, donde. Estas etiquetas se colocan utilizando git, de manera que sea fácil localizar las versiones exactas de cada entregable.

Los archivos estarán subidos al servidor por medio de la herramienta Jenkins, que accederá al repositorio de nuestra parte del sistema, extraerá los ficheros y les pasará unas pruebas. La base de datos estará desplegada en el servidor.

# 10. Mapa de herramientas

# Mapa de herramientas

Figura X. Mapa de herramientas

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

## 10.1 Descripción del mapa de herramientas

Todos los grupos han utilizado la wiki de [EGC](http://1984.lsi.us.es/wiki-egc), por lo que aparece como nexo común a todos ellos. Desde ella descargamos la aplicación del año pasado y en ella fuimos documentando el desarrollo (actas de reunión, 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 tiene en su interior un repositorio individual para cada grupo, cuya nomenclatura es “G” + número de grupo (por ejemplo, “[G03](https://github.com/AgoraUS1516/G03)”). 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**. Utilizado para la escritura del código fuente.
* **Bitnami**. Proporciona un servidor Apache donde desplegar la aplicación en local.
* **MySQL**. Base de datos.

La aplicación va a ser desplegada en un servidor Apache en el que Jenkins se encargará de la integración continua. 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.

**11.** Conclusiones

**12.** Bibliografía