Cabina de Telegram

Grupo 2

ID de opera: 75

Miembros del grupo (en orden alfabético):

- Bujalance Muñoz, Alberto
- Gamero Monge, Alejandro
- Iglesias Pérez, Daniel
- Rosa Serrano, Victor
- Troncoso Correa, Julio

Enlaces de interés:

- repositorio de código
- sistema desplegado (En este caso, nuestro sistema desplegado permite usar el bot @CabinaDevBot en Telegram)
- wiki de la asignatura

Índice

1 - Resumen

Como proyecto para la asignatura de Evolución y Gestión de la Configuración (EGC), se quiere desarrollar un portal de jornadas que permita tener un sistema de participación en el cual poder realizar votaciones. Para acometer dicho proyecto, se llevo a cabo una división del mismo en subsistemas qué permitirá la interaccionan entre ellos.

Estos subsistemas se han repartido entre los distintos subgrupos de trabajo que se han establecido entre los alumnos de la asignatura. En nuestro caso, el subsistema elegido se llama "**Cabina de Telegram**", el cual consiste en la creación de un bot para la aplicación de Telegram que nos permita realizar votaciones desde él.
Nuestro subsistema permitirá la interacción con el portal de jornadas desde dispositivos móviles con un funcionamiento basado en comandos sencillos y por pasos muy desglosados que dotarán al usuario de una gran comodidad y facilidad para votar en el sistema.

Un bot de Telegram es una herramienta creada por Telegram que nos permite "llamarlos" en cualquier momento para conseguir una serie de funcionalidades diferentes. La comunicación con él es muy intuitiva ya que es muy similar o casi idéntico a una conversación cualquiera con una persona en Telegram, por lo que su uso es apto para todos.

Este trabajo se podría realizar haciendo cambios sobre el código heredado de años anteriores, como puede ser: añadir nuevas funcionalidades, trasladarlo a otro lenguaje, etc. También era posible realizar los distintos subsistemas desde cero, que es lo que hemos decidido en nuestro caso.

En principio nuestro sistema, al tratarse de un bot, no iba a proporcionar ninguna API a otros subsistemas, pero sí que necesita consumirlas como puede ser el caso del login en el bot, que consumirá la API del equipo de registro, o para poder votar, que necesitará la API del equipo de votaciones, entre otros.

Nota: Dado que para la fecha de entrega, las APIs que necesitamos de los distintos equipos aún no las tienen disponibles, hemos decidido cambiar el diseño de las funcionalidades para que simulasen el comportamiento esperado con datos de prueba. Por lo que no consumimos APIs de otros subsistemas en esta versión entregada.

2 - Introducción y contexto

Para la utilización del bot hay que ir a la aplicación de Telegram, ya sea en un dispositivo móvil o en Telegram web, pulsar en el botón de búsqueda y poner @CabinaDevBot, tras lo cual se empieza una conversación con el bot.
Para interactuar con él, basta con escribir en la conversación con éste los comandos que tiene disponibles, en nuestro caso son: /login, /votar, /exit y /commands. Igualmente, para ver los comandos disponibles en nuestro bot y una breve descripción de los mismos hay que teclear /commands.

Para el caso de /login, por ejemplo, el flujo de la conversación es la siguiente:

1. El usuario introduce el comando a utilizar en el chat, /login para este caso.
2. El bot le contestará y le pedirá datos al usuario, su nombre de usuario concretamente.
3. Tras la respuesta del usuario, el bot puede ejecutar la acción destinada o iniciar de nuevo una nueva pregunta o propuesta, la contraseña en este caso.
4. Tras esto, el bot ejecutará la acción de autenticar al usuario al sistema si no hay ningún problema con los datos introducidos por el usuario. Para finalizar, el bot dará un mensaje de bienvenida al usuario.

Cuando se manda un mensaje al bot, este mensaje es recibido por Telegram, quien lo redirige al lugar donde tengamos ejecutando la aplicación, de este modo es posible ejecutar la aplicación en local y que funcione en cualquier sitio, y que sin estar el bot en ejecución los mensajes salgan marcados como recibidos, ya que es Telegram quien se encarga de las comunicaciones entre los bots.

3 - Descripción del sistema

Como se ha dicho anteriormente, hemos decidido realizar desde cero todo el código de la aplicación, por lo que no ha habido cambios respecto al proyecto anterior. Básicamente hemos ido añadiendo las distintas funcionalidades que hemos necesitado para su funcionamiento.

El proyecto está dividido en tres carpetas, la primera llamada basicBot, en la que se alberga el script para iniciar el bot. Las distintas funciones del bot están en otro archivo, en el cual también están los datos identificativos del bot en Telegram, que son el nombre y el token.
La segunda carpeta es la de functionality, y en ésta hay varios archivos con funciones de apoyo para los distintos métodos que usamos en el bot.
Por último tenemos la carpeta objetos, en la que hay clases java de apoyo.

Las funciones o comandos que tiene el bot son las siguientes:

- **/start:** Inicia una conversación con el bot, es un comando por defecto del bot y es modificable a partir del @BotFather.

- **/commands:** Muestra los distintos comandos que tiene implementado el bot y da una breve descripción de los mismos. Es modificable a partir del @BotFather.
- **/login:** Conecta al usuario al sistema a través de un nombre de usuario y una contraseña, en caso de introducirse datos incorrectos se cancela la operación.
- **/votar:** Seleccionamos una votación a través de su id y el bot nos empezará a hacer las preguntas correspondientes a dicha votación y el usuario tiene que ir contestando a ella una tras una. También se a decidido que al principio de este método te mostrará las votaciones abiertas del momento.
- **/exit:** Desloguea al usuario en caso de estar logueado.

4 - Planificación del proyecto

El proyecto se ha dividido en funcionalidades, que se han repartido entre los miembros del grupo. Además cuando se recibe un issue y se considera necesario, se encarga a distintos miembros del grupo cada vez. Previamente, ha sido necesario el desarrollo de unas versiones iniciales para crear una estructura inicial sobre la que poder trabajar en las distintas funcionalidades.

Las funcionalidades principales en las que se ha dividido el trabajo son:

Funcionalidad	**Encargado**
/login	Victor Rosa
/votar	Alejandro Gamero y Julio Troncoso
/exit	Daniel Iglesias
/commands	Alberto Bujalance

Esta división inicial del trabajo no implica que personas que no sean encargadas de alguna funcionalidad no hayan participado también en el desarrollo o corrección de alguna de ellas.

Como las funcionalidades necesitan de APIs de otros subsistemas, al principio se realizarón las distintas funcionalidades de modo que cuando estuviesen disponibles estas APIs se completarían. Tras acercarse la fecha de entrega final, y dado que las APIs de los otros subsistemas no se han terminado, decidimos cambiar las distintas funcionalidades para que hagan una simulación con datos de prueba introducidos por nosotros.

Debido a esto, las distintas funcionalidades han pasado por tres etapas: una primera en la que se obtiene el esqueleto o la forma básica en la que va a funcionar, una segunda en la que se deja lista para completar cuando se tuvieran las APIs de los otros subsistemas, y una última en la que, a falta de dichas APIs, se han modificado para que simulen el comportamiento esperado con unos datos de prueba específicos.

5 - Entorno de desarrollo

Para el desarrollo del proyecto se ha usado la herramienta Eclipse Oxygen (4.7) como entorno de desarrollo integrado (IDE) en el que se ha trabajado con Java 8 y Maven, haciendo uso de la librería de TelegramBots 3.5.

Para la instalación del sistema hemos instalado Java 8, y Eclipse Oxygen, tras lo cual creamos un nuevo proyecto Maven en el que cambiamos el pom.xml para añadir las dependencias a la librería de TelegramBots.
Luego hay crear un bot en Telegram, esto se hace mediante el uso de otro bot llamado @BotFather, que permite la creación y edición de bots y que nos proporciona el token que necesitaremos para utilizar el bot.

Si se quiere usar el sistema que proporcionamos sólo sería necesario tener instalado Java 8 y Eclipse, e importar el proyecto. Para ejecutarlo directamente en eclipse solo habría que ejecutar Application.java.

Hay que tener en cuenta que la aplicación funciona de la misma manera una vez ejecutada en local o totalmente desplegada, es decir, ejecutando en local te permite la interacción con el bot en Telegram como si se tratara de la aplicación desplegada en un servidor.
Ambas usan un usuario y token especial para el bot, por lo que si ejecutamos la aplicación en dos sitios a la vez no funcionará el bot ya que hay conflicto entre las comunicaciones, por lo que en el caso de que esté la aplicación desplegada, ejecutarla en eclipse por ejemplo conllevará a que deje de funcionar.

Esto se soluciona usando otro token y nombre, es decir usar otro bot distinto, por lo que en el archivo principal se encuentran tanto el bot username como el bot token del bot principal, y comentados los de un bot secundarios que se creó de prueba y que puede usarse para ejecutar dos versiones distintas a la vez.

6 - Gestión del cambio, incidencias y depuración

Pasamos ahora a explicar la gestión del cambio, incidencias y depuración que hemos seguido durante el desarrollo del proyecto.
Se describen a continuación una serie de protocolos o procesos que tienen como finalidad detectar, informar y corregir errores o producir mejoras en el código de nuestro sistema. Para ello estos procedimientos deben ser sistemáticos, estrictos y detallados.

Con estos objetivos en mente el equipo ha utilizado el sistema de issues de Github. Dentro del repositorio se pueden establecer issues con una visibilidad total tanto para los miembros

del equipo como para los componentes de los demás grupos. Atendiendo a la ley de linux: “dado un número suficientemente elevado de ojos, todos lo errores se convierten en obvios”.
Los issues de Github nos permiten exponer un problema para que podamos recibir ayuda o informar de un error. Posee además la posibilidad de establecer etiquetas que permitan una mayor claridad a la hora de encarar un issue, asignar a un compañero para que sepa que debe encargarse él de esa tarea y lo más importante, poder responder y aportar información sobre un problema que ya hemos solucionado.

Si atendemos a la división de incidencias según la procedencia de la información sobre el error, internas (denunciadas por miembros de nuestro propio equipo) o externas (denunciadas por miembros de los demás grupos) podemos definir el proceso de resolución con el siguiente protocolo de actuación:

6.1 - Incidencias Internas

1. Un miembro del equipo detecta la necesidad de hacer un cambio.
2. El desarrollador abre un nuevo issue en Github.
3. El issue debe tener un título fácilmente entendible y explicativo. Aunque lo más importante es que se concrete lo más posible el cambio.
4. En la descripción del issue debe explicarse de forma detallada qué es lo que ocurre y los pasos que otro miembro del equipo debe seguir para para obtener el mismo error e intentar solucionarlo.
5. En este punto se decide si se aprueba el cambio.
6. El issue debe estar asignado a una persona para su implementación y en el caso de que sea necesario ir acompañado de una etiqueta que lo clasifique.
7. Cuando se haya logrado corregir el error o realizado la mejora, el desarrollador que ha implementado el cambio debe responder al issue con una descripción de la solución.
8. Se recomienda que la persona que solucione el issue no lo cierre, sino que sea el desarrollador que lo abrió el encargado de cerrarlo una vez que haya comprobado que el error ha desaparecido.
9. Es necesario aludir al issue en el commit que lo resuelva.

Pasamos un enlace a los issues resueltos por el equipo issues resueltos

6.2 - Incidencias Externas

Se sigue el procedimiento anterior con la única excepción de que se debe identificar nuestro grupo en el título cuando el issue recae sobre otros grupos diferentes al nuestro.

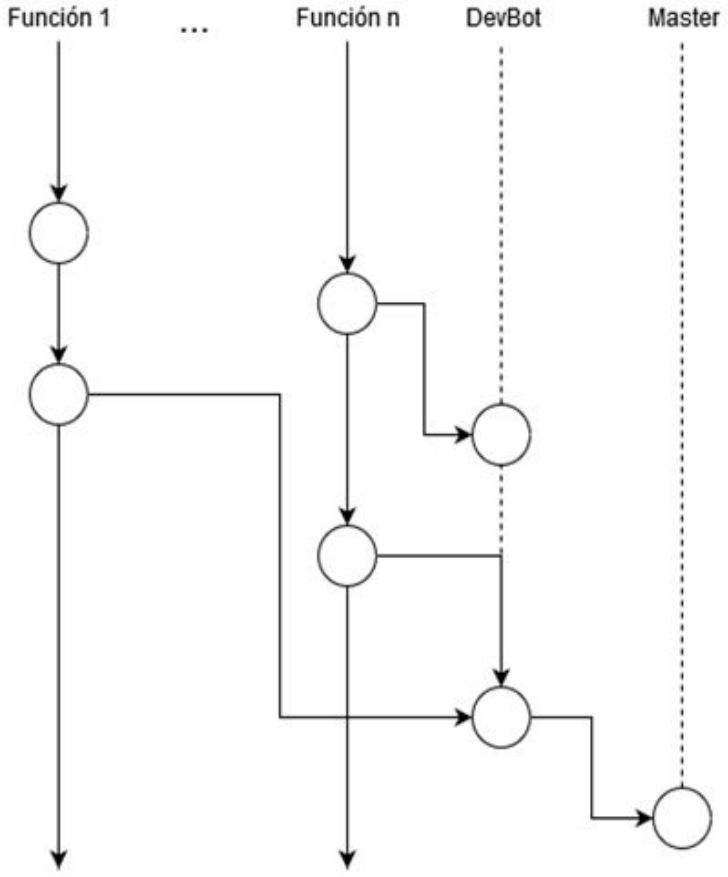
7 - Gestión del código fuente

Se realizan commits cada vez que se complete una funcionalidad, como es el caso de añadida funcionalidad de prueba de votacionesAbiertas o de add: funcionalidad de exit, o cuando se corrija un issue, como en closes #5 esqueleto de exit.
En caso de que no se termine una funcionalidad y se deje de trabajar ese día, también se realizará commit.

Se sigue el modelo para commits propuesto por el equipo de Integración.

El repositorio dispondrá de varias ramas:

- **Master:** se trata de la rama principal y es donde se alojarán las distintas versiones funcionales del proyecto.
- **Dev:** contendrá las funcionalidades o issues solucionados una vez estén acabados.
- **Ramas de funciones:** habrá tantas ramas de funciones como funcionalidades haya que ir añadiendo al sistema, así como issues a solucionar.



Ramas del proyecto

El proceso para añadir una funcionalidad consistirá en lo siguiente:

1. Se creará una nueva rama con nombre igual a la funcionalidad a desarrollar y se seguirá trabajando en esa rama hasta acabarla.
2. Una vez acabada la funcionalidad se llevará a la rama de Dev, donde se alojará hasta que se termine de comprobar que se trata de una versión estable y que no hay problemas.
3. Por último se moverá a Master la versión de Dev cuando se haya testeado debidamente.

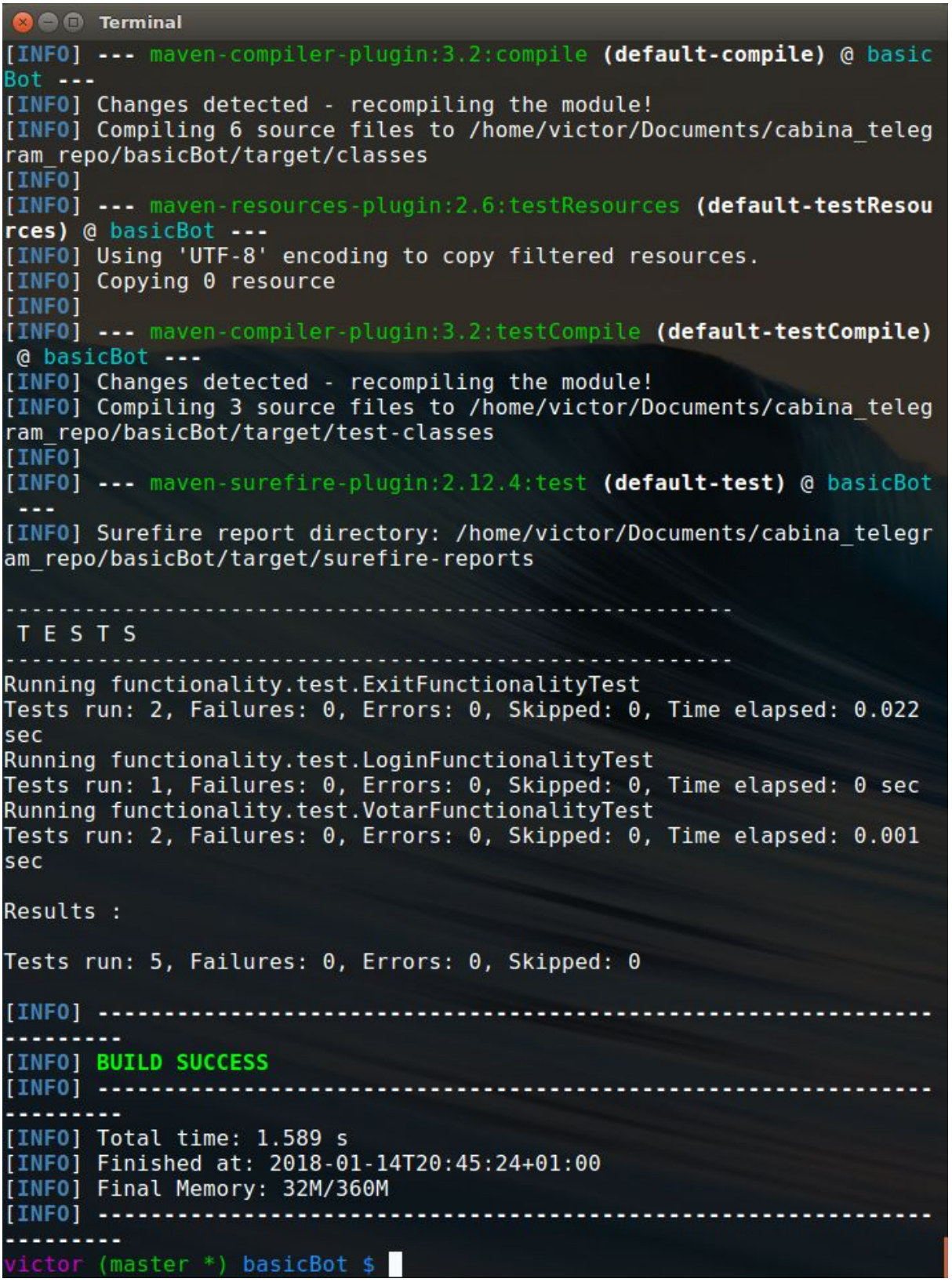
Como se ha podido observar en los distintos enlaces puestos en este apartado, hemos utilizado GitHub como herramienta de desarrollo para la gestión del código fuente.

8 - Gestión de la construcción e integración continua

8.1 - Gestión de la construcción

Nuestro proyecto está realizado en Java, y por ello utilizaremos Maven para la construcción y todo lo relacionado con el proceso de testing.

En primer lugar se realizan los test, para ello utilizamos jUnit:



Luego se compila el proyecto:

```
Terminal
victor (master) basicBot $ mvn compile
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building basicBot 0.0.1-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ basicBot -
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ basicBot ---
[INFO] Nothing to compile - all classes are up to date
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 0.540 s
[INFO] Finished at: 2018-01-13T21:49:50+01:00
[INFO] Final Memory: 12M/303M
[INFO] ------------------------------------------------------------------------
victor (master) basicBot $
```

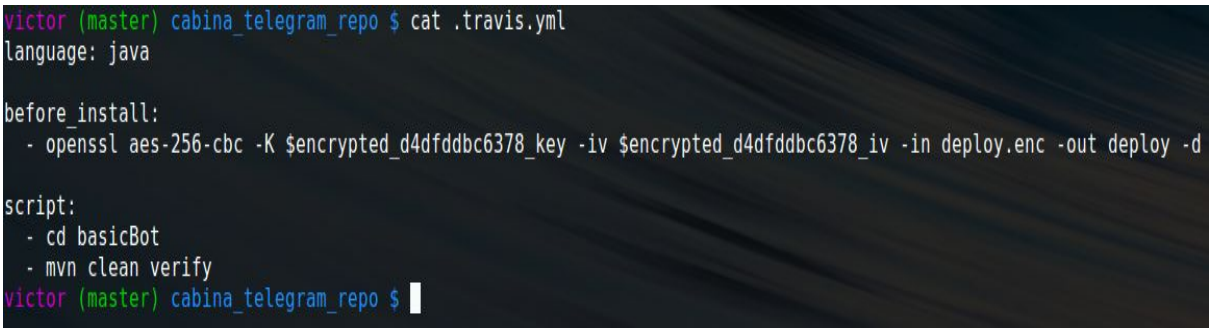
Maven también se encarga de la gestión de dependencias. En nuestro sistema, además de las dependencias de jUnit, utilizamos una librería de terceros para la construcción de bots en Java que se puede consultar aquí. Para ello utilizamos el archivo pom.xml y Maven se encarga de todo lo demás.
La librería se puede encontrar en el Maven Central Repository en el siguiente link.
El archivo pom.xml queda de la siguiente manera:

```
Open
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
2   <modelVersion>4.0.0</modelVersion>
3   <groupId>com.egc.telegram</groupId>
4   <artifactId>basicBot</artifactId>
5   <version>0.0.1-SNAPSHOT</version>
6   <name>basicBot</name>
7
8   <properties>
9     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
10    <maven.compiler.source>1.8</maven.compiler.source>
11    <maven.compiler.target>1.8</maven.compiler.target>
12  </properties>
13
14  <dependencies>
15    <dependency>
16        <groupId>org.telegram</groupId>
17        <artifactId>telegrambots-abilities</artifactId>
18        <version>3.5</version>
19    </dependency>
20
21    <dependency>
22        <groupId>junit</groupId>
23        <artifactId>junit</artifactId>
24        <version>4.12</version>
25    </dependency>
26  </dependencies>
27
28 </project>
```

8.2 - Integración Continua

Para la integración continua se ha usado la herramienta vista en la asignatura, Travis. Mediante el fichero llamado .travis.yml especificaremos el lenguaje, el nombre del archivo y la ruta de dicho archivo que debe ejecutar cada vez que se realice un “push” así como un

comando Maven para limpiar los target de posibles compilaciones anteriores. De esta manera aseguramos que los cambios introducidos no producen una recesión en el proyecto. Dejamos un enlace a la página del proyecto en Travis aquí.



9 - Gestión de liberaciones, despliegue y entregas

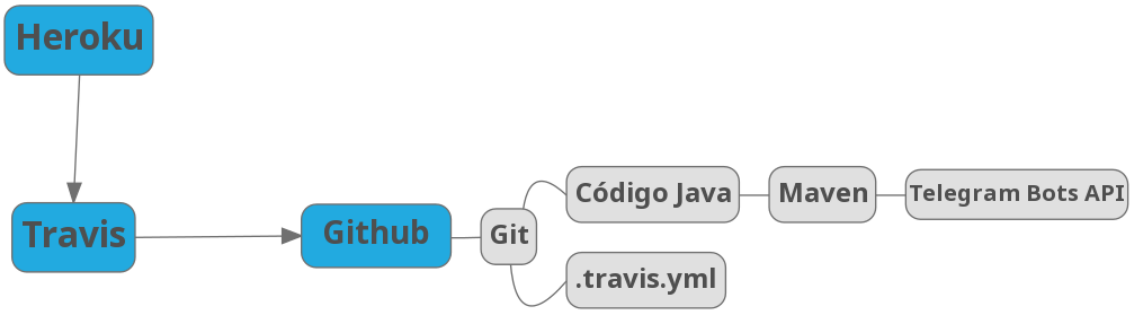
Consideramos la versión actual del sistema aquella que se encuentra en la rama “Master” de nuestro repositorio. Una vez que la rama “Dev” es estable y todas las funciones están correctamente integradas, se suben todos los datos a master para tener una nueva versión.

Travis, junto con maven, se encarga de realizar los tests apropiados al subsistema y de comprobar que la configuración es correcta para su despliegue.

Una vez se da el visto bueno a la configuración, se procede al despliegue automático con Heroku, que mantendrá el bot encendido para que cualquiera pueda usarlo desde un dispositivo con Telegram.

La entrega de la documentación y el software se hará mediante la página del grupo en Ópera, siguiendo las instrucciones del profesorado de la asignatura.

10 - Mapa de herramientas



Heroku es utilizado junto con Travis para desplegar automáticamente el bot una vez se alcanza una nueva versión en la rama “master”

Travis CI es una herramienta de integración continua que usamos para conectar github con heroku cada vez que se añade una nueva versión y además para pasar automáticamente una serie de tests de maven.

Usamos Github como repositorio gratuito y Git para gestionar este repositorio.

Utilizamos Java como lenguaje de programación y Maven para gestionar las librerías y los tests automáticos. La librería principal que utilizamos para los tests es Junit y para el bot, la [Telegram Bots API de rubenlagus]

11 - Ejercicio de propuesta de cambio

Como ejercicio de propuesta de cambio hemos pensando realizar una modificación en nuestro bot para el método **/exit**. Dicho cambio consistiría en añadir una comprobación para que no podamos salir del sistema si no estamos autenticados en él.
Los pasos que tomaremos para dicho proceso son los siguientes:

1. Creamos un issue en GitHub especificando el problema.
2. La persona asignada en ese issue deberá crear una rama con el formato issue#<número del issue>.
3. Modificamos los cambios pertinentes en el código.
4. Se realizan distintas pruebas para la comprobación del funcionamiento del mismo.
5. Commiteamos a la rama del issue creada anteriormente, agregando un “closes <número del issue>” al final del mismo.
6. Tras esto, se añade la rama a Dev.
7. Se comprueba en el sistema, estando en la rama Dev, que todo funciona correctamente.
8. Una vez que se confirma que Dev tiene una versión estable, se sube a la rama Master y el issue se cerrará automaticamente, dando por finalizado el proceso de cambio.

12 - Conclusiones y trabajo futuro

Como conclusiones aportadas podemos establecer las siguientes:

Cosas que han ido bien:

- Desde primera hora la comunicación del equipo ha sido buena y se han establecido reuniones periódicas que han ayudado a que todo el equipo esté informado de la situación del proyecto.
- Junto con el equipo de integración se decidieron rápidamente varios protocolos como el de gestión de los commits, lo que ha proporcionado desde el principio ideas claras de cómo gestionar el trabajo de forma homogénea

Cosas que han ido mal:

- Debido a que todos los grupos hemos ido dejando el trabajo para el final, nos hemos encontrado con apenas unos días para terminar el proyecto y sin poder interactuar con otros subsistemas como se tenía pensado hacer en la asignatura. Lo que ha supuesto un cambio en la funcionalidad del sistema respecto a lo que se tenía pensado hacer desde el principio.
- Muchas veces el uso tan específico de las ramas del repositorio han provocado más problemas que el orden que pretenden aportar en un proyecto sencillo con un grupo de desarrollo pequeño como es el caso.

Mejoras:

- Suponiendo que no fuese necesaria ir al M5, una mejora clara sería la integración con otros subsistemas.
- Otra mejora sería la internacionalización del bot que permitiese cambiar entre inglés y español.
- Se podría realizar un menú con botones en lugar de comandos escritos por chat.