Documentación del proyecto

Cabina de votación Telegram

**Índice de contenido**

[1. Resumen 2](#_Toc471896492)

[2. Introducción y contexto 3](#_Toc471896493)

[3. Descripción del sistema 4](#_Toc471896494)

[3.1. Planificación del trabajo 4](#_Toc471896495)

[3.2. Cambios realizados 6](#_Toc471896496)

[3.3. Diagrama de componentes 6](#_Toc471896497)

[4. Elementos de control 7](#_Toc471896498)

[5. Entorno de desarrollo 8](#_Toc471896499)

[6. Gestión del código fuente 9](#_Toc471896500)

[6.1. Gestión de las ramas en el código 9](#_Toc471896501)

[6.2. Gestión de parches 10](#_Toc471896502)

[6.3. Aprobación de cambios 10](#_Toc471896503)

[6.4. Roles en la gestión del código 11](#_Toc471896504)

[6.5. Políticas de nombre y estilo en el código fuente 12](#_Toc471896505)

[7. Gestión de la construcción e integración continua 12](#_Toc471896506)

[8. Gestión del cambio, incidencias y depuración 13](#_Toc471896507)

[9. Gestión de liberaciones, despliegue y entregas 14](#_Toc471896508)

[9.1. Entregables 14](#_Toc471896509)

[9.2. Roles 14](#_Toc471896510)

[10. Mapa de herramientas 15](#_Toc471896511)

[11. Conclusiones y trabajo futuro 17](#_Toc471896512)

# **Resumen**

Es cierto que, a la hora de decidir un plan entre amigos, comprar un determinado producto o decidir que se realiza para cenar se generan conflictos debido a que dichas decisiones se realizan de manera rudimentaria y no podemos poner de acuerdo a un cierto número de personas en un asunto. La desinformación y la gestión de las opiniones de cada uno a veces resulta ser una tarea tediosa de llevar a cabo, ya sea por el número de integrantes que deben de decidir sobre un hecho. Muchas personas para evitar dichos conflictos optan por una actitud pasiva y llegan a conformarse con lo que los demás integrantes han decidido.

Al principio del inicio del proyecto, en el equipo teníamos diversas temáticas a la hora de orientar nuestro proyecto, hasta que se nos ocurrió la idea de programar una herramienta que nos permitiera solventar dichas decisiones a través de votaciones que podían llevarse a cabo a través de una aplicación gratuita accesible a todos llamada Telegram.

Nuestra intención ha sido crear un sistema capaz de llegar a gran escala, cuidando tanto la privacidad como su seguridad encriptando los votos. Para ello, hemos usado de una de las herramientas que nos ofrece esta aplicación, los bots. Se pueden iniciar los bots como si de otra conversación de Telegram se tratase, solamente que en vez de recibir los mensajes otra persona lo recibe un programa, el cual nos aporta ciertas funcionalidades según lo que se haya programado.

Es una buena solución debido a que puede llegarse a generar votaciones de manera realmente sencilla, compartirlas con cualquier contacto de Telegram y observar los resultados. También pensamos que es la mejor solución ya que Telegram es una aplicación de mensajería instantánea gratuita, la cual cualquier persona puede hacer uso de ella desde cualquier terminal móvil u ordenador.

Como cosas a destacar después de haber realizado el proyecto, podemos llegar a decir que estamos realmente orgullosos de haber realizado dicho trabajo, ya que hemos llegado a disfrutar colaborando con otros equipos (haciendo uso de sus APIs) y aprendiendo los conceptos básicos para la realización del bot. Estamos seguro que no será el único bot que realizaremos.

# **Introducción y contexto**

Como se ha explicado con anterioridad, hemos desarrollado un bot para la aplicación gratuita de mensajería instantánea Telegram. Con dicho bot tendremos la posibilidad de crear, compartir, gestionar y seguir votaciones, e incluso emitir un voto para cualquier votación externa. Dicha idea surgió a raíz de la falta de motivación para realizar los posibles trabajos restantes de los que disponía el equipo a la hora de asignar los trabajos mediante la herramienta Opera.

Nuestra aplicación nos permite controlar un sistema de votaciones complejo, integrado con Agora@US. En concreto, nos hemos centrado en la parte de cabina de votación, permitiendo al usuario mostrar las votaciones disponibles y emitir votos en ellas, pudiendo además modificar y/o eliminar dichos votos.

Como funcionalidad adicional, brindamos la posibilidad al usuario de crear votaciones desde la propia cabina Telegram y también de compartir las votaciones que desee con otros usuarios.

Para votar, un usuario debe estar registrado y autenticado en la aplicación; el registro se realiza a través de una interfaz web mantenida por nosotros en la que el usuario debe introducir sus datos correctamente para darse de alta en el sistema. Una vez ha hecho esto, la aplicación se comunica con la API del módulo de Autenticación B para que el usuario sea registrado y pueda votar.

Cuando un usuario accede a una votación, la aplicación coge los datos de sesión del usuario y comprueba si el usuario ha votado o no en dicha votación comunicándose con la API del módulo de Censo. Si el usuario ha votado previamente en dicha votación, el sistema lo detecta y se le permite modificar o eliminar su voto. Si no había votado, se le permite emitir un voto nuevo.

Una vez que el usuario emite un voto, la aplicación usa un ejecutable del módulo de Verificación para encriptar el voto y mandarlo de forma segura por la red para guardarlo en la base de datos.

Para crear votaciones en el sistema, al usuario se le pedirán los datos relacionados con dicha votación, como son:

* Título
* CP
* Preguntas (Título, tipo de pregunta y respuesta)

Cuando el usuario considere que ha introducido toda la información necesaria para crear la votación, la aplicación se comunica con la API del módulo de Recuento y Modificación, para que la nueva votación se haga persistente y accesible a los demás usuarios para que puedan votar en ella.

Las principales ventajas de usar la herramienta de bots de Telegram son varias, entre ellas:

* Es gratuita
* Es open-source
* Permite que nuestro servicio sea escalable (la aplicación funcionará de igual forma sin importar el número de peticiones que haga, de una forma transparente para nosotros).

Además de dichas ventajas, usar un bot nos permite realizar cualquier proceso en tiempo real y sin la necesidad de la presencia de ninguna persona para revisar el funcionamiento de la aplicación aparte del usuario que lo esté utilizando, todo ello unido a lo simple y lo intuitivo que resulta su uso.

En adición supone una mejoría en cuanto a experiencia de uso puesto que tendríamos la posibilidad de emitir votos desde cualquier lugar utilizando nuestro teléfono móvil o tableta, sin tener que estar delante de un PC en un momento concreto.

# **Descripción del sistema**

## **Planificación del trabajo**

Para la gestión de tareas dentro del proyecto hemos hecho uso del método de tableros kanban. Waffle.io es una herramienta que nos permite el uso de dicho tablero a través de un navegador, donde todos los integrantes del equipo pueden acceder y asignar tareas a realizar.

Waffle.io del equipo: <https://waffle.io/AgoraUS-G1-1617/CabinaTelegram>

Dicho tablero se encuentra vinculado a nuestro repositorio de GitHub, en donde cada tarea generada en Waffle.io serán instancias de cada issue de GitHub.

La metodología seguida a la hora de asignar las tareas ha sido libre, en donde cada miembro del equipo se ha asignado tareas de la lista de tareas *Ready* de Waffle.io.

Las listas de tareas de nuestro tablero kanban son las siguientes:

* *Backlog*: Preparado, pila de tareas o bugs que son descubiertos o añadidos, pero todavía no están listos para empezar a trabajar con ellos.
* *Ready*: Son las tareas ya listas para empezar a trabajar con ellas.
* *In progress*: Son las tareas que ya están en proceso por el usuario asignado.
* *Done*: Son las tareas que se ha completado. Desde aquí puede ser archivada para que no se muestre en el tablero.

Como se ha dicho en la descripción de *Done*, las tareas en dicha tabla han sido archivadas y por lo tantos no son visibles en Waffle.io, pero podemos ver cuáles son en GitHub si filtramos las issues cómo *Closed*.

A continuación, presentamos un listado de las tareas que se han realizado en el proyecto junto a qué miembro del equipo se encuentra asignada:

|  |  |
| --- | --- |
| TAREA | ASIGNADO A |
| Crear votación por bot | Juan Rodríguez Dueñas |
| Compartir votación | José Luis Salazar González |
| Crear base de datos y scripts | José Manuel Gavira González |
| Cambiar ruta base de datos | José Manuel Gavira González |
| Crear base de datos al iniciar al bot | Jorge Puente Zaro |
| Crear scripts de automatización | José Luis Salazar González |
| Obtener resultado mediante API | José Manuel Gavira González |
| Test de creación de votaciones | José Luis Salazar González |
| Integración con recuento y modificación | José Luis Salazar González  Jorge Puente Zaro  José Manuel Gavira González  Juan Rodríguez Dueñas |
| Votar a través del bot | Juan Rodríguez Dueñas |
| Añadir voto a la base de datos | José Manuel Gavira González |
| Test de la base de datos | Jorge Puente Zaro |
| Encriptar votos | José Luis Salazar González |
| Obtener votación de API | Juan Rodríguez Dueñas |
| Enviar voto mediante API | Jorge Puente Zaro |
| Enviar votación creada por API | José Luis Salazar González |
| Ver todas las votaciones | José Manuel Gavira González |
| Crear formulario web para login | José Luis Salazar González |
| Test de obtener votaciones | José Manuel Gavira González |
| Añadir test a Travis | José Luis Salazar González |
| Restricción de voto | Juan Rodríguez Dueñas |
| Permitir modificar votos | José Luis Salazar González |
| Añadir panel a info\_votacion | José Luis Salazar González |
| Auto-descargar .jar de verificación | José Luis Salazar González |
| Test de obtener votación | Juan Rodríguez Dueñas |
| Test de enviar voto | Jorge Puente Zaro |
| Test de obtener recuento | José Manuel Gavira González |
| Test de creación de votación API | José Luis Salazar González |
| Eliminar participación | Juan Rodríguez Dueñas |
| Integrar con censo | José Luis Salazar González |
| Documentación del proyecto | José Luis Salazar González  Jorge Puente Zaro  José Manuel Gavira González  Juan Rodríguez Dueñas |

Tabla 1: Planificación de equipo

## **Cambios realizados**

Desde el principio, nuestro sistema iba a guardar las votaciones y los votos en una base de datos, ya que no existían ninguna API a usar y debíamos demostrar que nuestro sistema funciona. Este sería uno de los primeros cambios realizados en nuestro sistema, ya que posteriormente obtuvimos la API de recuento y modificación.

Otro cambio destacable sufrido fue a la hora de organizar las votaciones. Nuestra idea principal fue organizar las votaciones por usuario, pero recuento y modificación no proporcionaban esa opción, teniendo que modificar nuestro modelo.

## **Diagrama de componentes**

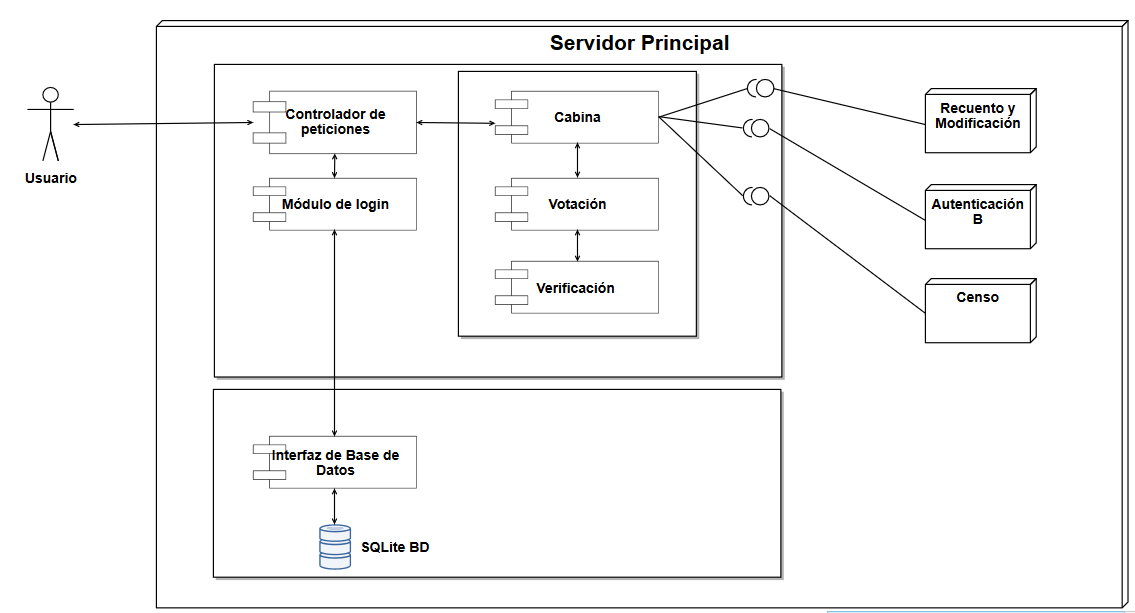


Ilustración 1: Diagrama de componentes

Como se puede observar en el diagrama, el usuario únicamente “accede” al módulo de control de peticiones (con las que se comunica con el bot). Antes de realizar cualquier acción con el bot, el usuario deberá registrarse en una página de login que está conectada con una base de datos que almacena los tokens necesarios para el proceso de login.

El módulo de control de peticiones se comunica a su vez se comunica con la cabina de Telegram. Dicha cabina recibirá las peticiones de voto, creación de votación o de gestión de votaciones del usuario, por ello para poder ejecutar dichas peticiones está comunicado con el módulo de votación.

El módulo de cabina integra los subsistemas de ”Recuento y modificación” (hacemos uso de su sistema de bases de datos), “Autenticación B”(para comprobar los credenciales del usuario en el proceso de login) y “Censo”(para comprobar si un usuario ya ha votado o no).

Nuestro proyecto también integra al subsistema de “Verificación” para el cifrado de los votos, pero como dicho subsistema suministra un .jar para el proceso de cifrado que se encuentra en nuestro entorno, lo consideramos un módulo más. Como se puede observar el módulo de verificación se comunica con el de votación.

# **Elementos de control**

Nuestro base line inicial se encuentra definido por los siguientes elementos de configuración:

* JetBrains PyCharm (versión 2016.3.1) - <https://www.jetbrains.com/pycharm/>
* Git (versión 2.9.3) - <https://git-scm.com/>
* Python (versión 3.5.2) - <https://www.python.org/downloads/release/python-352/>
* pyTelegramBotAPI (versión 2.2.3) - <https://github.com/eternnoir/pyTelegramBotAPI>
* pyTest (versión 3.0.5) - <http://docs.pytest.org/en/latest/>
* Requests (versión 2.9.1) - <http://docs.python-requests.org/en/master/>
* Flask (versión 0.12) - <http://flask.pocoo.org/>
* Telegram Desktop (versión 0.10.20) - <https://telegram.org/>
* Lubuntu (versión 16.10) - <http://lubuntu.net/>

# **Entorno de desarrollo**

## **Descripción del entorno de desarrollo**

Para el desarrollo de este proyecto se ha utilizado el entorno “JetBrains Pycharm Community Edition” (versión 2016.3.1).

La instalación de dicho entorno es sencilla, simplemente consiste en acceder al sitio web <https://www.jetbrains.com/pycharm/download/#section=windows> y seleccionamos la opción “Download” para la versión “Community”. Tras esto sólo quedará seguir los pasos del instalador.

Dicho entorno funciona con el lenguaje de programación Python, concretamente con la versión 2.7. Para instalar dicho lenguaje accedemos al sitio web <https://www.python.org/download/releases/2.7/>, seleccionamos el instalador adecuado para nuestro sistema operativo y seguimos los pasos del instalador.

En el proyecto se han utilizado las siguientes librerías de Python (además de las instaladas por defecto):

|  |  |
| --- | --- |
| LIBRERÍA | VERSIÓN |
| flask | 0.12 |
| pyTelegramBotAPI | 2.3.0 |
| requests | 2.7.0 |
| configparser | 3.5.0 |

Tabla 2: Librerías

Para la integración continua del proyecto se ha utilizado “Jenkins”. Para instalarlo en nuestro equipo accederemos al sitio web https://jenkins.io/, haremos click en “Download Jenkins”, seleccionaremos la versión que prefiramos para nuestro sistema operativo, descomprimimos el archivo que descargaremos e instalaremos el archivo extraído.

En adición a todos estos softwares se han utilizado GitHub para el control de versiones y la gestión de código, Travis CI para la automatización de las pruebas, y Waffle.io para la gestión de incidencias.

## **Instalación del entorno de desarrollo**

Lo primero de todo que debemos hacer es proceder a descargar *Oracle VM VirtualBox*, programa que nos permitirá trabajar con la máquina virtual que hemos aportado.

*Oracle VM VirtualBox*: <https://www.virtualbox.org/wiki/Downloads>

Procederemos a realizar la instalación simplemente haciendo clic en “Siguiente” en cada ventana de instalación (posiblemente se requiera reiniciar el sistema).

Es necesario que la opción de “Virtualización de nucleos” que se encuentra dentro de la BIOS de nuestro ordenador se encuentre activada para que nuestro programa pueda ejecutar máquinas virtuales.

Luego de haber realizado la instalación y configuración de los elementos necesarios para ejecutar *Oracle VM VirtualBox*, procedemos a descargar la máquina virtual que contendrá todo el entorno de desarrollo del proyecto.

Máquina virtual: <https://alumuses-my.sharepoint.com/personal/jossalgon5_alum_us_es/_layouts/15/guestaccess.aspx?guestaccesstoken=hPH5yzkkdAf2jI%2F4MuHO9%2F1yGYG3RArcdrtd4rB%2Btek%3D&docid=1ed5f054ec6a14c9e98f7749e62da6219&rev=1>

Una vez descarga procederemos a descomprimir los archivos mediante un software de descompresión de archivos .zip (nuestra recomendación: 7zip).

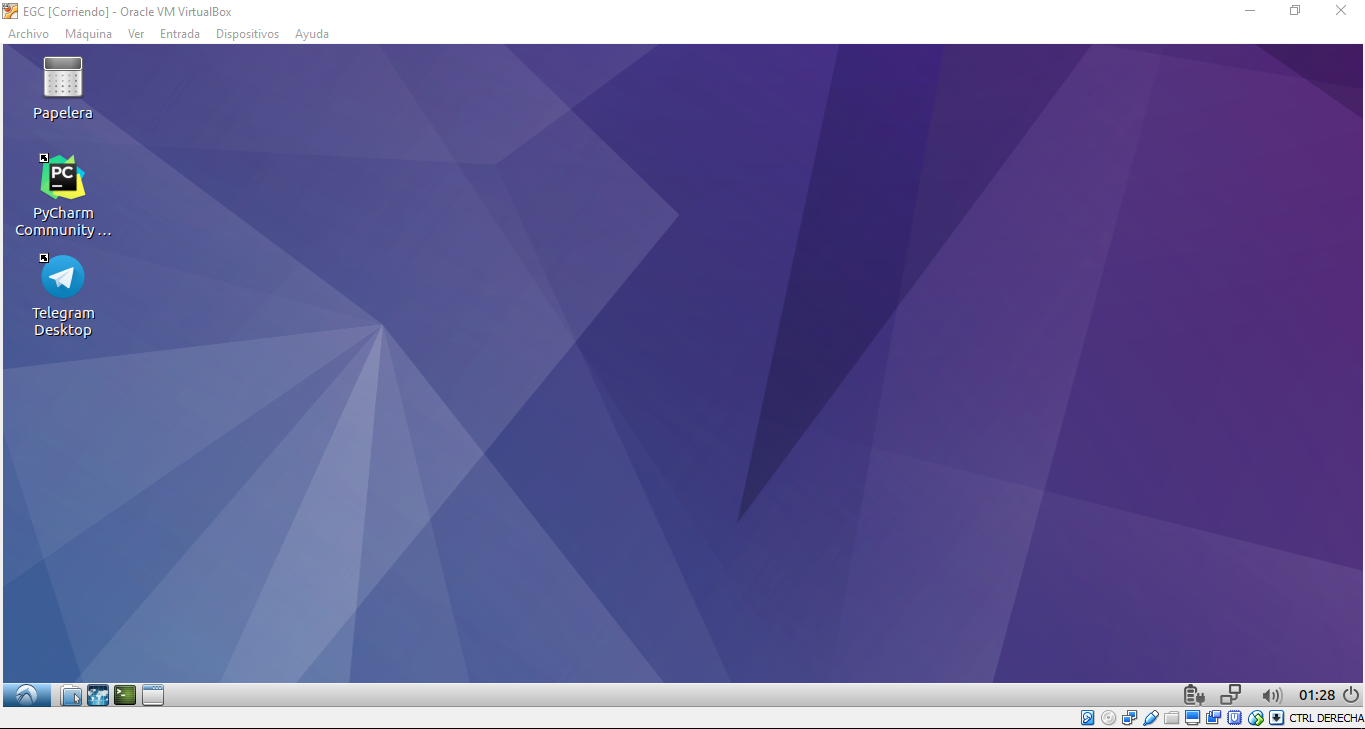
Por último, para ejecutar la máquina simplemente debemos hacer doble clic en el archivo “EGC.vbox”, el cual abrirá el programa *Oracle VM VirtualBox*. Una vez hecho esto, solo debemos hacer clic en “Run” para iniciar la máquina virtual. Si todo lo anterior se ha realizado de manera correcta, deberemos de ver el escritorio del sistema operativo Lubuntu: 

Ilustración 2: Escritorio Lubuntu

La máquina virtual dispondrá del software necesario para desarrollar el proyecto.

Una vez dentro, procederemos a descargar/clonar el proyecto subido a GitHub. Para ello simplemente deberemos de abrir el terminar del sistema operativo en el escritorio y ejecutar el siguiente comando:

*git clone https://github.com/AgoraUS-G1-1617/CabinaTelegram*

Si todo va como lo esperado, veremos en la consola algo como esto.

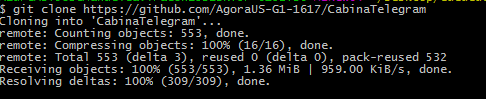


Ilustración 3: Clonar proyecto GitHub

Después de esto, se generará una carpeta en el escritorio con el nombre “CabinaTelegram” la cual contendrá todos los archivos del proyecto.

Procederemos a abrir ahora nuestro software de entorno de desarrollo *JetBains PyCharm*. La primera vez que se inicia, nos dará la posibilidad de crear un nuevo proyecto o de importarlo. Para importar nuestro proyecto previamente descargado de GitHub sólo debemos de hacer los siguientes pasos:

1. Hacer clic en “Open”.

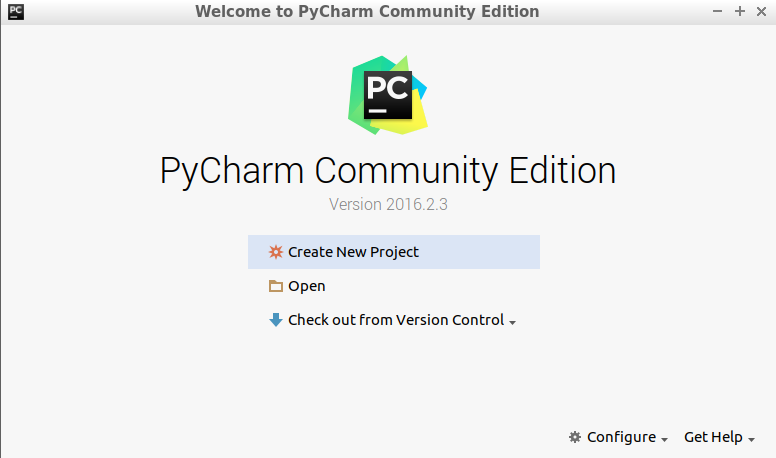


Ilustración 4: Inicio JetBrains Pycharm

1. Seleccionamos nuestra carpeta “CabinaTelegram” ubicada en el escritorio.

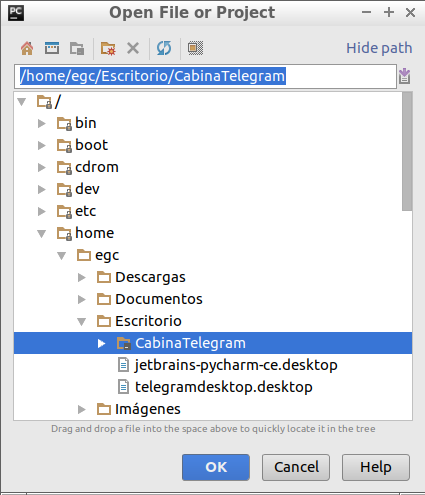


Ilustración 5: Importar proyecto

1. Al hacer clic en “OK”, ya tendremos nuestro proyecto importado.

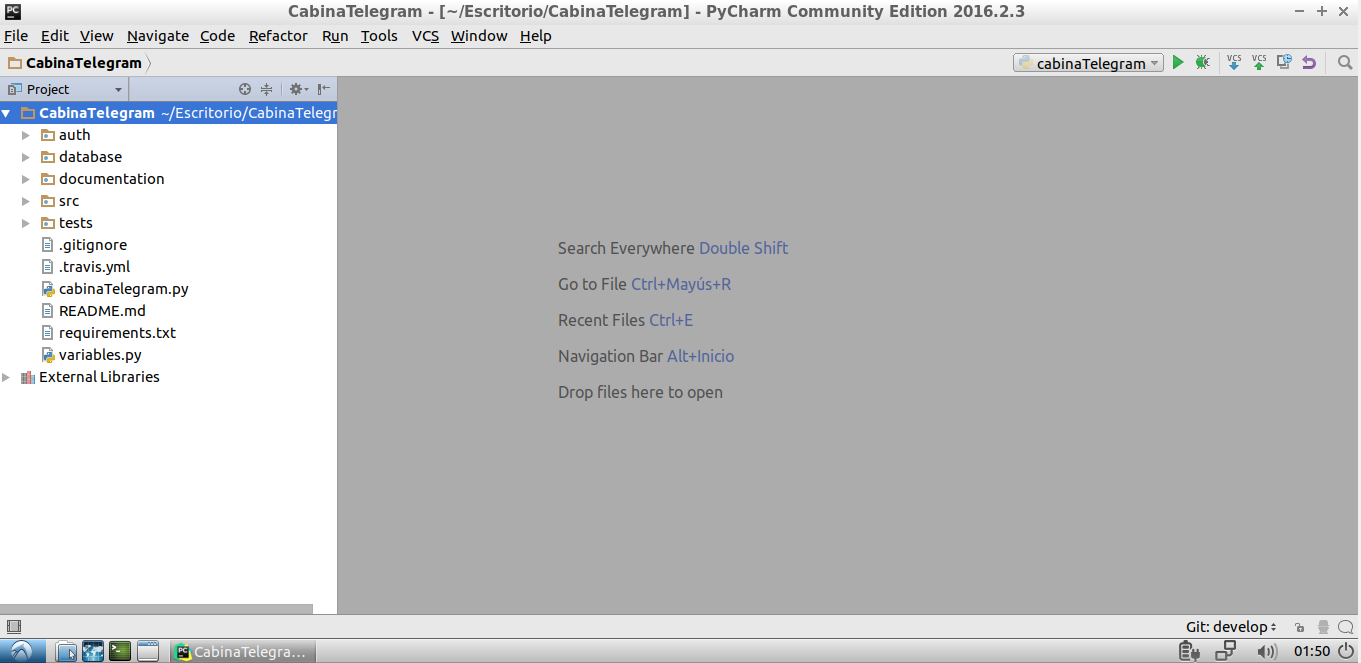


Ilustración 6: Proyecto importado

Una vez realizado esto, necesitamos de un token para generar un bot de pruebas para poder comprobar las modificaciones realizadas en el proyecto. Para ello, accedemos a la aplicación *Telegram Desktop* y realizamos los pasos necesarios para poder usarlo. Luego, en el buscador escribimos @BotFather para iniciar un bot aportado por Telegram el cual se encarga de ofrecer tokens para los bots que creemos. Para iniciar el bot, únicamente escribimos “/start”.

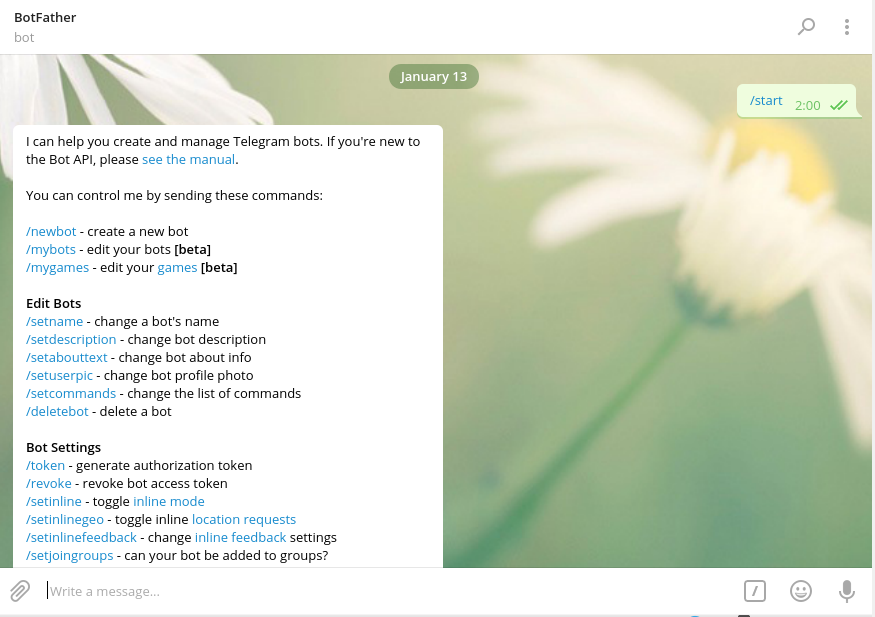


Ilustración 7: @BotFather

Luego para proceder a crear el bot, escribimos “/newbot”. A continuación, procederemos a indicar un nombre para nuestro bot.

Una vez hecho esto, escribimos “/mybots” y seleccionamos nuestro bot recién creado. Por último, hacer clic en “API Token” para obtener el token de dicho bot el cual nos usaremos para realizar nuestras pruebas. Deberemos de copiar dicho token para incluirlo en un archivo *config.ini* dentro de la capeta del proyecto. Dicho archivo tendrá la siguiente estructura:

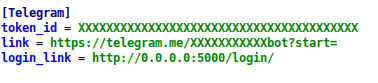


Ilustración 8: Estructura config.ini

El valor de la variable *token\_id* deberá de contener el token del bot aportado por el @BotFather, y en la valiable link, donde existan X, deberá cambiarse por el nombre completo que se le ha dado al bot.

Ya creado el proyecto, si quisiéramos realizar una primera prueba de ejecución, deberemos de arrancar previamente el servidor *Flask* para poder acceder de manera local a nuestra interfaz de autenticación. Para ello, deberemos de abrir un terminal dentro de la carpeta del proyecto, y ejecutar el siguiente comando:

*python3 src/main/python/auth/app.py*

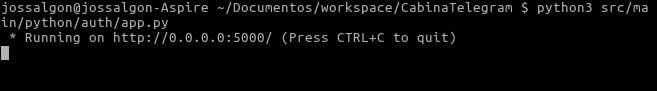


Ilustración 9: Iniciar Flask

Si todo ha ido bien, en el terminar observaremos la línea de comando mostrado en la anterior ilustración.

Por último, para ejecutar los que viene siendo el propio bot de prueba que hemos creado previamente, sólo deberemos de iniciar el archivo *cabinaTelegram.py* haciendo clic en PyCharm al botón de “Play” ubicado en la esquina superior.

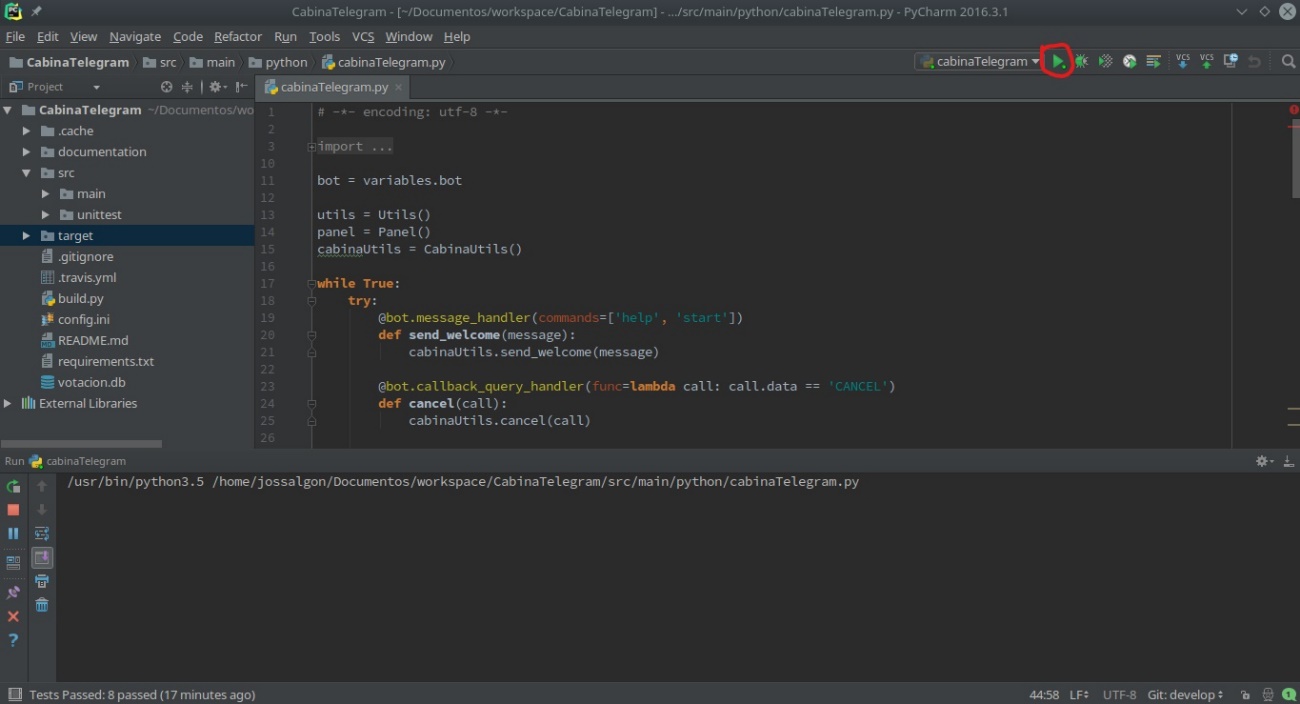


Ilustración 10: Bot iniciado

Ya únicamente debemos de buscar nuestro bot en la aplicación *Telegram Desktop* e iniciarlo.

# **Gestión del código fuente**

## **Gestión de las ramas en el código**

En la planificación inicial del proyecto se decidió el uso de dos ramas para la gestión de código:

* **develop:** Es la rama que cuenta con los últimos cambios añadidos y a la que se suben éstos. El código en ésta rama es funcional, pero podría presentar errores, aunque es útil para presentar la última versión del proyecto en una revisión técnica.
* **master:** Es la rama en la que se encuentra el código final y estable, preparado para incluirlo en el entregable.

Para la gestión del código (y por tanto de las ramas) se utiliza Git. Un resumen de cómo utilizar ramas con Git se puede ver en el siguiente enlace:

<https://git-scm.com/book/es/v1/Ramificaciones-en-Git-Procedimientos-b%C3%A1sicos-para-ramificar-y-fusionar>

Como repositorio remoto, se usa GitHub. En el siguiente enlace se puede ver un sumario de cómo utilizar GitHub con Git (en inglés)

<https://github.com/Kunena/Kunena-Forum/wiki/Create-a-new-branch-with-git-and-manage-branches>

## **Gestión de parches**

Para aplicar un parche se deben seguir los siguientes pasos:

1. Un desarrollador detecta un error o quiere corregir una funcionalidad
2. Lo comunica al resto del equipo y se acuerda realizar el parche
3. Un desarrollador crea una issue en GitHub y la pasa en Waffle.io al estado *Ready*
4. Se empieza a desarrollar el parche, pasando la issue al estado *In* *Progress*
5. Una vez terminado el desarrollo del parche, un desarrollador debe probarla en su equipo local. Si la prueba falla, se vuelve a repetir el punto 5 hasta que la del código funcione como es debido, si no, se pasa al punto 6
6. El desarrollador comunica al resto del equipo que el parche está listo para desplegarse. Una vez informados todos los miembros del equipo, se sube al repositorio de GitHub
7. Una vez subido, las herramientas de gestión, pruebas e integración se ejecutan e informan de que el nuevo código está desplegado satisfactoriamente y ha pasado las pruebas necesarias. Si hubiese algún error, hay que volver al punto 5. En caso contrario, se avanza al punto 8
8. Se cierra la issue.

## **Aprobación de cambios**

Todos los miembros del equipo son libres de proponer mejoras, las cuales serán llevadas a cabo después de llegar a un consenso con todos los integrantes del equipo.

En caso de incorporar nuevas funcionalidades que dependen de las APIs de otros equipos, en necesario notificar al otro equipo y recabar la información necesaria.

## **Roles en la gestión del código**

|  |  |  |
| --- | --- | --- |
| https://lh5.googleusercontent.com/Lxl7J-cePo5CeeYZPCUSmXsUV1609H70QfB1dbG4_STtz4P8Irx3F0cwbdxGuQfC-3xYPjOPqtzJTIrY4oUxqzMDcfcDGhVmYgrIRXEWYTNSGEn7p51W9ZWuHPJhgPIYtclIBGUs | **José Luis Salazar González** | Jefe de proyecto, programador |
| https://lh3.googleusercontent.com/ZPfyzvD7-XNeTzolByypIRuQFvaJm8yLIPHqyB3jYFYgYlVf7AuXRcBsU9Gvt485mnSOACIBHjW_QaCiFI2TG2o0qZiEShTF6f3BwG9u_uaoSAIWnJESZJlo6rjWU8J_XvvDRqJU | **Jorge Puente Zaro** | Programador |
| https://lh3.googleusercontent.com/eYPvJlrzwweRi9LNHW85dP42xMl3EYd9b7h3v_vD6acKSDXBfzChXx3D-ozZ-lsjlbQklsbbmeXlgLqDZwuxmKtrlJ-4bQ9PWTKvqbWCErT-_T-8M-JQ3WKoZ78A6e0sdZhBqiu6 | **José Manuel Gavira González** | Programador |
| https://lh3.googleusercontent.com/g6I1kyNCFl4LweUrwTNnmit6rF3DtlcQ6tU2szmXwPqLacsWbxC6AKja4vFL8QM1_YcAyHR8RGtt287uw-q2CnPzNhfb9jY52BYNQEtlFUR_yza_D9VSdN_hBz9lKWBPdsLZ2YD5 | **Juan rodríguez Dueñas** | Programador |

Tabla 3: Tabla de roles

## **Políticas de nombre y estilo en el código fuente**

A la hora de programar, el equipo ha seguido los siguientes patrones:

* A la hora de definir clases, siempre la primera letra será únicamente la que sea mayúscula. Ej: *Votacion*.
* No se hará uso de caracteres especiales a la hora de definir clases, atributos, métodos o variables. Ej: *titulo* (sin tilde en la ‘i’).
* Cuando queremos almacenar un identificador, si procedemos a poner el nombre de la variable en español, la palabra ‘id’ irá precediendo a esta misma. En el caso de que la variable sea en inglés, la palabra ‘id’ ira después de esta misma.
* Cuando especificamos una lista o pila, el nombre de las variables siempre estará expresado en plural. Ej: *preguntas\_respuestas*.
* Debemos de hacer uso de variables con dos palabras para no equivocarnos en que vamos a almacenar si fuera necesario. Se hará con ‘\_’ entre dichas palabras. Ej: *fecha\_cierre*.

Se pretenderá llevar un estilo ‘*clean code*’ para todo el código generado.

# **Gestión de la construcción e integración continua**

La construcción se gestiona desplegando el código en un contenedor docker mediante Jenkins. No se necesita construir ni compilar el programa puesto que Python es un lenguaje interpretado.

El despliegue de la aplicación se gestiona de diferente forma según la rama a la que se ha subido:

* Cada vez que se hace un push a la rama develop en GitHub, Jenkins descarga el código desde GitHub y lo pone en una carpeta local. Una vez descargado, pone el código en un contenedor docker y realiza las configuraciones necesarias. Una vez configurado el contenedor, instala automáticamente en él las dependencias establecidas en el archivo “requirements.txt”.  
  Por último, se lanza el bot en Python y la página web de login con Flask.
* Para la rama master, se ha optado por hacer el despliegue cada vez que el equipo de desarrollo así lo considere, por ejemplo, cuando se ha decidido que hay una nueva funcionalidad lista para poder integrarla a en la versión estable.  
  Para ello, se debe ejecutar el script “master.sh” de forma manual, que tiene la misma funcionalidad que el script que genera el docker con la rama develop, pero con la rama master.

# **Gestión del cambio, incidencias y depuración**

Los cambios se controlan mediante el sistema de control de versiones GitHub junto con la herramienta waffle.io. Mediante la técnica del tablero Kanban se reparten las tareas en distintas columnas como se explica con anterioridad (en dichas columnas se pueden apreciar los cambios que se pretenden realizar). Para los cambios ya realizados basta con acceder a la sección issues de GitHub y acceder a ellos en la sección *Closed*. Para un nivel mayor de detalle de los cambios realizados se puede acceder al historial de commits al repositorio.

Como se acaba de explicar, los cambios tienen tres estados:

* Preparado (a espera de iniciarse).
* En proceso.
* Finalizado.

Los dos primeros estados se pueden comprobar en Waffle.io y en GitHub, mientras que los finalizados solo se pueden comprobar en GitHub.

En cuanto a roles no hay una diferenciación de privilegios a la hora de gestionar los cambios, cualquier miembro puede realizar cualquier acción sobre los cambios.

Por lo general, los cambios se extraen todos de los requisitos definidos en un inicio, luego cualquier cambio que entre al estado “preparado” será realizado tarde o temprano. Tampoco se diferencian prioridades para seleccionar un cambio. El único factor que puede determinar el desarrollo más o menos tardío de un cambio podría ser el tamaño que se asigna a cada tarea (priorizando los que más tiempo puedan consumir).

La depuración se realiza de la misma forma para incidencias internas como para aquellas que procedan de otro subsistema. El equipo analizará el origen de la issue, buscando su causa y las posibles soluciones que existan a esta. De todas las posibles soluciones el equipo selecciona la más convincente y la aplica. Tras esto se notifica al resto del equipo o al equipo del subsistema externo de que la incidencia ha sido solucionada. En caso de tratarse de un error de código, el encargado de solucionarlo será el que desarrolló dicho módulo.

# **Gestión de liberaciones, despliegue y entregas**

## **Entregables**

Al final de la asignatura deberemos aportar una serie de entregables realizada por el equipo, los cuales se detallarán a continuación:

* **Documentación del proyecto:** documento profesional donde recoge todos los elementos citados en el documento TrabajoEGC\_16\_17 (<http://1984.lsi.us.es/wiki-egc/index.php/Archivo:TrabajoEGC_16_17.pdf>)
* **Diario de equipo:** documento donde se recogen todas las actividades realizadas durante el equipo a lo largo del desarrollo del proyecto, así como los cambios realizados.
* **Máquina virtual:** contiene las herramientas y elementos indispensables para la realización del proyecto y su despliegue.

Una vez finalizado todos los entregables deberán ser subido al portal Opera, en nuestro caso, en el siguiente enlace: <http://opera.eii.us.es/egc/public/trabajo/ver/id/68>.

## **Roles**

Durante la entrega del proyecto, cada uno de los integrantes del grupo revisará cada entregable de manera grupal.

En esta situación, un miembro será el encargado de preparar la máquina virtual, comprobando que todo funciona correctamente. Luego de esto, el mismo miembro se encargará de subirla donde corresponda.

Los demás miembros del equipo se centrarán en revisar la documentación del proyecto y el diario de equipo, cuidando el formato y procurando darle un toque profesional y técnico. Cuando todo esté revisado, se subirá al portal Opera un archivo comprimido con ambos entregables.

# **Mapa de herramientas**

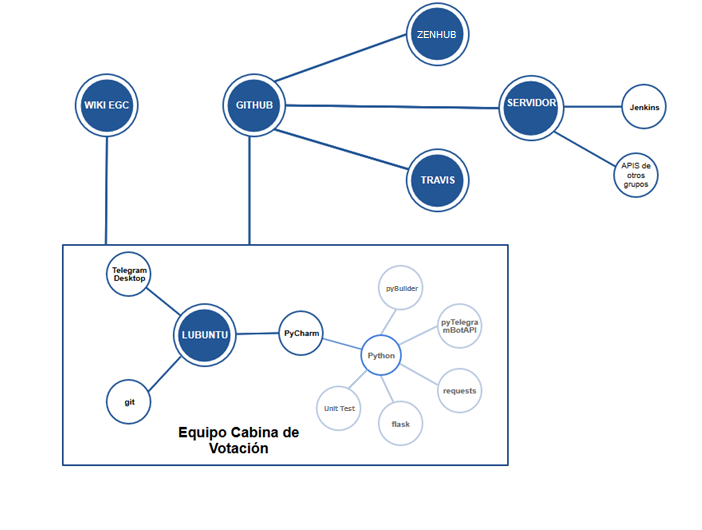


Ilustración 11: Mapa de herramientas

A continuación, se describe el entorno de desarrollo utilizado para el proyecto:

El sistema operativo utilizado ha sido **Lubuntu 16.10**. Lubuntu es una modificación de Ubuntu enfocada a consumir menos recursos y ser más rápida. Para ello tiene diversas modificaciones respecto a Ubuntu, por ejemplo, usa el gestor de escritorio LXDE, en vez de Unity.

Nos hemos decantado por éste SO puesto que hemos utilizado un entorno en una máquina virtual, por tanto, era conveniente tener un sistema que consumiera la menor cantidad de recursos posible y no era necesario un sistema de escritorio con múltiples efectos visuales.

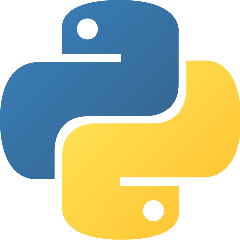
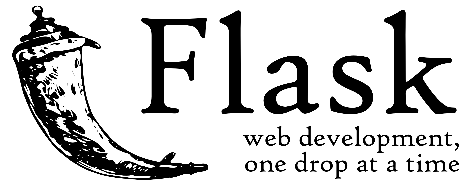


Para comprobar el funcionamiento del bot, hemos utilizado el cliente de escritorio de **Telegram**. Podríamos haber utilizado también la versión web, pero ésta es menos rápida y estable, además de tener menos opciones.



Para la gestión del código fuente local y la comunicación con el repositorio remoto hemos usado **Git**, no sólo por ser un sistema excelente, sino también por la integración que tiene con GitHub.

El entorno de desarrollo escogido ha sido **PyCharm** puesto que es el más sencillo de usar -en nuestra opinión- para desarrollar en Python.

Por su parte, hemos escogido **Python** como lenguaje de programación porque la implementación más completa para el uso que le hemos dado está escrita en ese lenguaje. Dicha implementación se recoge en la librería pyTelegramBotAPI, que nos permite comunicarnos con la API de bots de Telegram mediante el lenguaje Python.

Para la creación de la página de inicio de sesión hemos utilizado **Flask**, que es un microframework que permite la creación de páginas web con Python de una forma muy rápida y directa. Además, incluye un servidor web para poder ejecutar la página sin necesidad de instalar uno aparte.

Los tests están implementados con ayuda de la librería **pytest**, que nos permite crear pruebas para probar el funcionamiento del programa y además ejecutar dichas pruebas.

**Requests** es una librería de Python que permite trabajar con peticiones y respuestas HTTP; en nuestro caso, la usamos para comunicarnos con las APIS de las que consumimos servicios (tanto enviar peticiones como recibir e interpretar los datos recibidos).

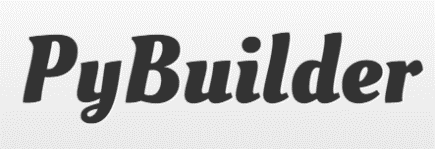
Las demás herramientas utilizadas son las siguientes:

La **Wiki de la asignatura** es la fuente de información principal para los alumnos, puesto que en ella están la planificación de la asignatura y del proyecto, la norma del trabajo, las diapositivas de teoría, etc.

**GitHub** es el servidor remoto de código usado por nuestro grupo y también por todos los demás, nos permite tener el código accesible por Travis y compartir las issues con Waffle.

**Travis** **CI** es el servicio utilizado para incorporar Integración Continua con el código subido a GitHub. Se encarga de correr automáticamente las pruebas definidas en el código.

**ZenHub** es la herramienta que nos permite tener un seguimiento detallado de las incidencias que se generen en GitHub. Se trata de una extensión de Google Chrome que nos permite asignarles estados a las incidencias mediante la técnica de tableros Kanban, que facilita la gestión al mostrar las incidencias y sus estados de una forma más visual. Además, permite asignar peso a las tareas de forma que se pueda priorizar unas tareas frente a otras. Por último, una de las propiedades más características de esta herramienta es la posibilidad de crear “Epic issues”. Estas incidencias permiten englobar a un grupo de tareas, siendo la “épica” la tarea madre y el resto las subtareas que la conforman.

**PyBuilder** es una herramienta para la gestión y construcción de proyectos Python escrita íntegramente en dicho lenguaje. Además cuenta con un plugin que permite crear ciclos de vida de una forma muy similar al funcionamiento de Apache Maven.

Por último, el servidor principal dedicado a los proyectos de la asignatura es en el que se ejecutará el bot propiamente dicho y a la vez, los proyectos de los demás grupos, con los que nos comunicaremos mediante sus APIS públicas.

El despliegue del bot en este servidor está automatizado mediante Jenkins, tal y como se ha explicado en el punto 7.

# **Conclusiones y trabajo futuro**

Tras la realización del proyecto, realmente estamos muy orgullosos de los objetivos cumplidos en el mismo, debido a que el equipo de Cabina de Votación del año pasado no aportó nada y partimos desde 0. Por otro lado, la creación de un bot de Telegram ha sido un acontecimiento realmente bueno, ya que hemos aprendido a integrar una idea nuestra con una herramienta que solemos usar todos nosotros como son las aplicaciones de mensajería instantánea.

Como ya hemos comentado antes, no disponíamos de código heredado el cual pudiéramos basarnos e incrementar su funcionalidad, por lo cual nos enfrentamos a un gran reto.

También cabe destacar que hemos tenido muchos obstáculos durante el de desarrollo del proyecto debido a era un proyecto el cual era imprescindible el avance de otros proyectos para que siguiera adelante. Ya que los demás equipos no avanzaban, hemos tenido que crear una base de datos para realizar la autenticación y otra para almacenar las votaciones y así poder realizar nuestras propias pruebas, ya que hemos tardado en disponer de la API de los demás grupos.

Finalmente, mencionar que nunca nos hemos llegado a enfrentar a un proyecto de esta envergadura y que no hubiera sido posible si no hubiera existido sin la comunicación constante con otros equipos, ya que hemos descubierto que es un elemento esencial para el éxito de cualquier proyecto software.

Como propuesta de mejora para un futuro sería añadir la funcionalidad de visualizar las votaciones de cada usuario y mejorar la eficiencia a la hora de enviar los votos a recuento.