UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Desarrollo de un bot de Discord para el apoyo a la docencia en línea

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Guardino Alonso, Ramón

Tutor/a: Castro Bleda, María José

CURSO ACADÉMICO: 2023/2024

Resumen

La pandemia del Covid-19 forzó a muchos centros educativos a adoptar medidas para permitir que las clases y otras actividades docentes pudieran llevarse a cabo de forma remota. Aun tras el fin del confinamiento, muchos de estos centros han seguido usando estos recursos por su gran utilidad y conveniencia. Durante este periodo, jóvenes en todo el mundo tuvieron que aprender a adaptarse a programas como Zoom y Microsoft Teams, los cuales resultaron poco familiares para personas de estas edades al estar enfocados a empresas y reuniones más formales.

Discord es un programa de comunicación de texto y audio utilizado principalmente por personas jóvenes para comunicarse con amigos. Al ser una aplicación más conocida y utilizada por personas de este demográfico, su uso para la docencia podría suponer una más fácil adaptación y mayor satisfacción del usuario. Otro aspecto de Discord que lo hace más deseable a sus alternativas es su amplio soporte para bots, permitiendo a usuarios desarrollar sus propios programas que cumplan distintas funciones dentro de la propia aplicación.

La idea del proyecto es desarrollar un bot que permita que la mayor parte de actividades docentes puedan llevarse a cabo dentro de un único servidor de Discord, para la fácil organización y centralización de diferentes tareas. El bot contará con diversas funciones para proporcionar un ambiente cómodo y eficiente tanto para alumnos como para profesores.

Palabras clave: Bot, Bot educativo, Discord, Python, Docencia, Docencia en remoto, Plataformas comunicación, Gestión actividades docentes.

Resum

La pandèmia del COVID-19 va forçar a molts centres educatius a adoptar mesures per a permetre que les classes i altres activitats docents pogueren dur-se a terme de manera remota. Fins i tot després de la fi del confinament, molts d'estos centres han continuat usant estos recursos per la seua gran utilitat i conveniència. Durant este període, joves a tot el món van haver d'aprendre a adaptar-se a programes com a Zoom i Microsoft Teams, els quals van resultar poc familiars per a persones d'estes edats en estar enfocats a empreses i reunions més formals.

Discord és un programa de comunicació de text i àudio utilitzat principalment per persones joves per a comunicar-se amb amics. A l'ésser una aplicació més coneguda i utilitzada per persones d'este demogràfic, el seu ús per a la docència podria suposar una més fàcil adaptació i major satisfacció de l'usuari. Un altre aspecte de Discord que ho fa més desitjable a les seues alternatives és el seu ampli suport per a bots, permetent a usuaris desenvolupar els seus propis programes que complisquen diferents funcions dins de la pròpia aplicació.

La idea del projecte és desenvolupar un bot que permeta que la major part d'activitats docents puguen dur-se a terme dins d'un únic servidor de Discord, per a la fàcil organització i centralització de diferents tasques. El bot comptarà amb diverses funcions per a proporcionar un ambient còmode i eficient tant per a alumnes com per a professors.

Paraules clau: Bot, Bot educatiu, Discord, Python, Docència, Docència en remot, Plataformes comunicació, Gestió activitats docents.

Abstract

The Covid-19 pandemic forced many educational centers to adopt measures to allow classes and other teaching activities to be carried out remotely. Even after the end of confinement, many of these centers have continued to use these resources due to their great usefulness and convenience. During this period, young people around the world had to learn to adapt to programs such as Zoom and Microsoft Teams, which were unfamiliar to people of these ages as they were focused on companies and more formal meetings.

Discord is a text and audio communication program used mainly by young people to communicate with friends. As it is an application that is better known and used by people of this demographic, its use for teaching could mean easier adaptation and greater user satisfaction. Another aspect of Discord that makes it more desirable than its alternatives is its extensive support for bots, allowing users to develop their own programs that fulfill different functions within the application itself.

The idea of the project is to develop a bot that allows most teaching activities to be carried out within a single Discord server, for the easy organization and centralization of different tasks. The bot will have various functions to provide a comfortable and efficient environment for both students and teachers.

Keywords: Bot, Educational bot, Discord, Python, Teaching, Remote teaching, Communication platforms, Management of teaching activities.

Índice general

Índice de figuras

Índice de tablas

1. Introducción

1.1 Motivación

La motivación para el desarrollo de este proyecto surge principalmente a raíz de la experiencia personal vivida tras realizar clases en línea en la universidad durante la época del confinamiento causado por el Covid-19. Todos los alumnos se vieron forzados a instalar y aprender a usar Microsoft Teams para poder reanudar las clases, y muchos expresaron frustración a la hora de usarlo.

Muchos estudiantes habían utilizado previamente la aplicación de mensajería Discord para comunicarse entre ellos y era una opinión común que muchos hubieran preferido realizar las clases en línea a través de este programa en lugar de Microsoft Teams porque lo consideraban mucho más cómodo y fácil de usar. En efecto, a lo largo de la pandemia, Discord resultó ser el programa preferido por la mayoría para reunirse y realizar actividades fuera de los horarios de clase.

Gracias a las herramientas que proporciona Discord para la creación de bots y las habilidades de programación aprendidas durante la carrera de Ingeniería Informática, se presenta la oportunidad de crear una herramienta que sirva como complemento a Discord, ayudando a convertirlo en una alternativa más atractiva para los docentes y alumnos con la esperanza de mejorar la experiencia en las aulas remotas.

1.2 Objetivos

El objetivo del trabajo será desarrollar un bot que pueda ser utilizado en Discord para proporcionar diferentes funcionalidades tanto a profesores como a alumnos durante las clases en línea. Para ello, se estudiarán diferentes aplicaciones comúnmente utilizadas por estudiantes para extraer ideas y decidir cuáles de ellas podrían ser implementadas en el bot.

Una vez decididas las funciones, se deberá planear detenidamente una interacción entre usuario y bot que sea intuitiva y cómoda de usar. Al finalizar las fases de planificación, se programarán las ideas y realizarán pruebas para asegurarse de que todo funciona correctamente.

1.3 Estructura de la memoria

Para desarrollar los objetivos planteados, la memoria se dividirá en los siguientes apartados:

1. Introducción: El apartado actual, donde se presenta la idea a desarrollar y los objetivos del proyecto.
2. Estado del arte: Se presenta una selección de aplicaciones y servicios relacionados con la idea a desarrollar, analizando sus características y explicando cómo pueden servir de inspiración.
3. Requisitos y funcionalidad: Se desarrollan las diferentes funciones que deberá cumplir el bot, explorando lo que un usuario podría encontrar al interactuar con él.
4. Desarrollo del bot: Se detalla el proceso de creación del bot desde el punto de vista del desarrollador mediante el uso de diferentes herramientas de programación.
5. Conclusiones: Se detallan las conclusiones obtenidas durante el desarrollo y en qué medida se han cumplido los objetivos planteados.
6. Bibliografía: Se incluyen las referencias que han sido utilizadas durante el desarrollo del trabajo.
7. Anexo: Se detalla cómo la idea puede contribuir a los Objetivos de Desarrollo Sostenible.

1.4 Introducción a Discord

Discord [1] es una plataforma de comunicación digital que ha ganado gran popularidad en los últimos años, especialmente entre jóvenes amantes de los videojuegos, aunque su uso se ha expandido mucho más allá de ese ámbito inicial.

Discord permite a los usuarios crear "servidores" [2], que son espacios virtuales dedicados a temas específicos. Dentro de estos servidores, se pueden crear múltiples canales de texto y voz para diferentes temas o propósitos. Para poner un ejemplo relacionado con la docencia, un servidor podría contener a todos los alumnos y profesores de un curso, y cada asignatura podría ser un canal diferente.

Una de las características más atractivas de Discord es su flexibilidad. Se puede usar para charlar con amigos mientras se juega, organizar un club de lectura, coordinar proyectos de trabajo, o incluso crear una comunidad en torno a un interés compartido.

A diferencia de las redes sociales tradicionales, Discord no se base en seguir a personas y ver sus publicaciones, sino que se centra en la comunicación en tiempo real y en la creación de comunidades cerradas. Un usuario puede unirse a servidores que le interesan o creas los suyos propios e invitar a otros.

Una persona también puede explorar servidores públicos a través de la web o dentro de la propia aplicación y unirse a ellos. Para personas interesadas en aprender nuevas

habilidades, pueden buscar dentro de la sección de “educación” del explorador [3] y allí encontrarán comunidades dedicadas a todo tipo de actividades, como aprender idiomas o programar. En estos servidores suelen haber recursos muy útiles para el aprendizaje, así como usuarios con mucho conocimiento del tema que pueden ayudar a los más inexpertos. También puede ser útil para conocer a personas con gustos similares y aprender junto a ellos.

La plataforma es gratuita en su versión básica, que incluye la mayoría de las funciones que un usuario promedio necesitaría. También ofrece una versión de pago con algunas características adicionales [4], principalmente con fines estéticos y que no crearían ninguna diferencia en un entorno docente entre un usuario de pago y uno gratuito.

Varios estudios han comprobado la eficacia de Discord en entornos docentes. Uno de ellos afirmó que los estudiantes “tienden a sentirse felices porque la atmósfera relajada que sienten al jugar juegos o chatear con sus amigos se transmite a las actividades de aula” [5]. Otro estudio obtuvo resultados positivos implementando Discord para la docencia inversa, donde los alumnos debieron utilizar Discord para ver los videos subidos y responder ciertas preguntas para más adelante discutirlas en persona dentro de un aula presencial [6].

A pesar de esto, Discord tiene algunos aspectos que podrían considerarse negativos a la hora de realizar clases. Uno de ellos es la facilidad con que los usuarios pueden cambiar de servidor, ya que puede causar distracciones si los alumnos están leyendo o escribiendo en otro servidor durante la clase [7]. En un estudio donde participaron 45 estudiantes de universidad, 11 de ellos indicaron que su mayor problema utilizando Discord para una clase fue la dificultad de aprender a navegar diferentes canales en el servidor, y 9 de ellos expresaron descontento por tener que prestar atención a las notificaciones de una nueva aplicación [8].

Finalmente, los usuarios pueden crear sus propios bots o instalar bots creados por otras personas [9]. Estos bots actúan como un usuario de Discord que sigue una programación y al que se le pueden dar comandos para realizar funciones que normalmente no serían posibles o resultarían demasiado tediosas para un usuario humano.

2. Estado del arte

En este capítulo se va a explicar en detalle algunas de las aplicaciones y programas que, de forma similar al bot de Discord, se usan habitualmente como herramientas para la docencia en aulas ya sean presenciales o en línea. Analizar estos productos existentes es importante ya que puede ayudar a comprender qué ideas han resultado un éxito y cuáles podrían mejorarse, y aplicar este conocimiento al programa que se desea desarrollar.

2.1 Herramientas de aprendizaje similares

2.1.1 Microsoft Teams y Zoom

Microsoft Teams [10] es una aplicación lanzada en 2017 que combina chat de texto y de vídeo y otorga la capacidad de almacenar y compartir archivos con otros usuarios. Como su nombre indica, está dedicada al uso por equipos, por lo que permite a sus usuarios crear salas donde múltiples personas puedan enviar mensajes simultáneamente y realizar llamadas conjuntas, con la posibilidad de compartir su pantalla y utilizar pizarras virtuales. En este aspecto, es muy similar a Discord, y la principal diferencia entre ambas aplicaciones es en su marketing.

Mientras Microsoft Teams está enfocado a empresas y grupos profesionales, Discord está diseñado para personas jóvenes que busquen realizar actividades juntos, principalmente jugar videojuegos. Esto significa que, si se quisiera introducir a alumnos jóvenes a una herramienta para la docencia en línea, Discord podría resultar la mejor opción, ya que es probable que muchos de ellos lo hayan utilizado antes.

A pesar de esto, muchos jóvenes se han visto obligados a utilizar Microsoft Teams en algún momento de sus vidas debido al confinamiento causado por el Covid-19. Esto periodo se puede apreciar en el crecimiento de usuarios anuales de la aplicación. Desde su salida en 2017 a 2019, había 20 millones de usuarios anuales. En 2020, el primer año del confinamiento, esta cifra aumentó a 75 millones y este gran crecimiento siguió a los 2 años siguientes, con 145 y 270 millones de usuarios respetivamente. En 2023, cuando ya se terminaron de levantar todas las restricciones, el crecimiento fue mucho menor, con un total de 300 millones de usuarios [11].

A pesar del fin de la pandemia, Microsoft Teams se ha convertido en una herramienta permanente para muchas aulas por su gran utilidad tanto dentro de clase como fuera, permitiendo a los estudiantes hacer preguntas, recibir respuestas y continuar trabajando en proyectos grupales fuera del horario lectivo. En la figura 1 se muestra una guía rápida para el uso docente de Microsoft Teams.

Otra herramienta muy popular usada durante la pandemia fue Zoom [12], la cual puede ser usada de una forma muy similar a Teams. El uso de una u otra plataforma depende

en parte de las preferencias de cada entidad. Aquellos centros que suelan utilizar productos de Microsoft como Word, Excel o PowerPoint probablemente elijan utilizar Teams debido a su integración con estos programas y la capacidad de utilizar la misma cuenta. El resto podrían escoger Zoom ya que podría considerarse más accesible y fácil de usar, aun a falta de algunas características en su versión gratuita [13].

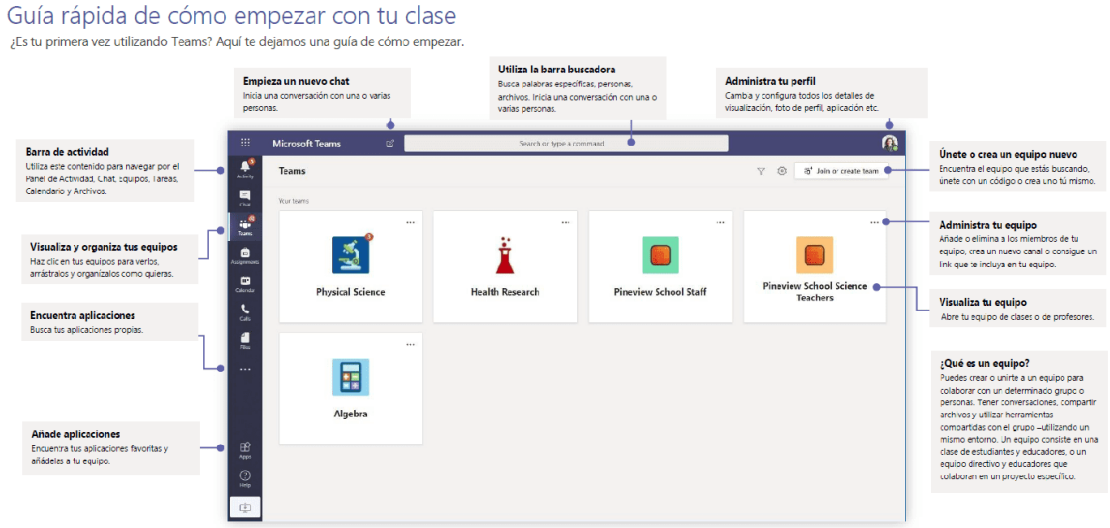

Figura 1: Guía de uso de Microsoft Teams [14].

2.1.2 Kahoot

Kahoot [15] es una plataforma que sirve como un juego educativo. El profesor prepara de antemano preguntas y múltiples respuestas posibles, donde una o más pueden ser la correcta. Estas se proyectan en el aula de forma que todos los alumnos puedan verlas, y ellos pueden elegir las respuestas que consideren correctas desde sus ordenadores o móviles dentro de un límite de tiempo. El sistema otorga puntos en función de la velocidad de respuesta correcta y al final del juego aparece una tabla con las puntuaciones de todos los jugadores, ordenada de mayor a menor [16].

Además de ser una forma divertida de aprender, el sistema genera un ambiente competitivo amistoso, fomentando el entusiasmo y participación de los alumnos al estar motivados a ganar a sus compañeros. Además, puede servir como herramienta para el profesor para saber qué alumnos están al día con los estudios o, por el contrario, cuáles podrían ir más rezagados y necesitar ayuda.

Un estudio en una universidad de Nueva Zelanda concluyó “Observamos que Kahoot brindó a los estudiantes más oportunidades de interactuar con el profesor, sus compañeros y el contenido de la conferencia. También ayudó a crear una experiencia de aprendizaje que se describió como “divertida”, lo que contribuyó a una dinámica útil de participación en el aula.”[17].

Esta plataforma ha ganado una gran popularidad, siendo usada desde escuelas primarias hasta clases de universidad e incluso en empresas para crear eventos de

formación de equipos, permitiendo a empleados conocerse mejor en un ambiente más casual y divertido.

Existen otras herramientas similares a Kahoot [18]. Por ejemplo, Poll Everywhere permite crear presentaciones más interactivas integrando encuestas en tiempo real y diferentes pruebas para los alumnos. Otras herramientas como Sildo son muy útiles para eventos y reuniones porque permiten obtener diferentes preguntas del público y determinar las más relevantes para ser respondidas.

Viendo la efectividad de esta plataforma, se ha considerado buena idea implementar en el bot de Discord un sistema muy similar, donde los profesores puedan crear preguntas que luego los alumnos podrán responder y obtener puntos, todo esto sin necesidad de salir de la aplicación de Discord. En la figura 2 se ilustra un ejemplo de una ronda de Kahoot.

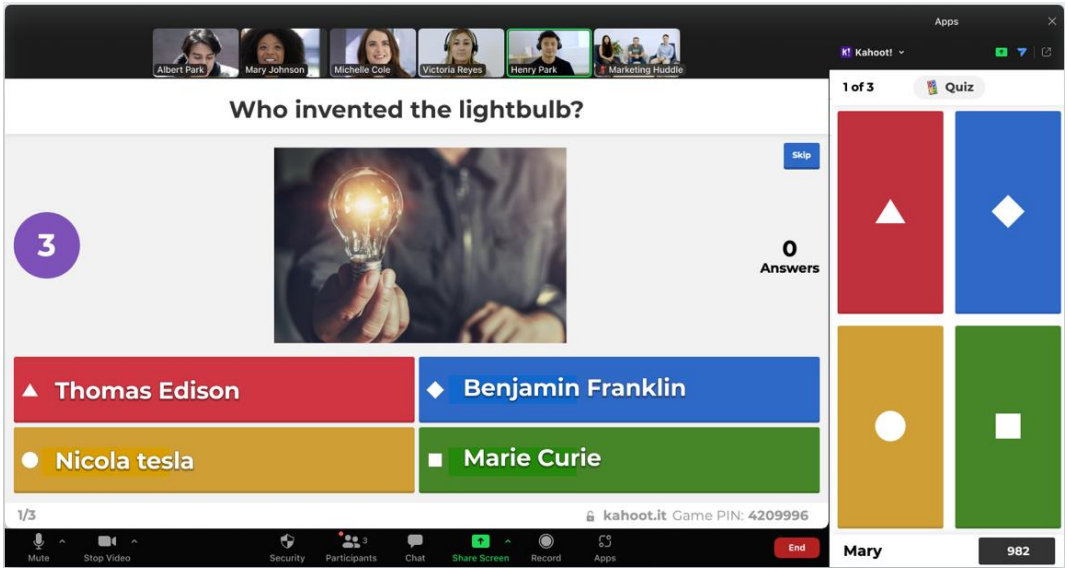

Figura 2: Ejemplo de una ronda de Kahoot [19].

2.1.3 Plataformas en línea propias de entidades

Las plataformas en línea de entidades son espacios web que habilita cada una y que habitualmente sólo pueden acceder personas relacionadas con la entidad como alumnos y profesores. Un ejemplo de una plataforma de este tipo sería Poliformat, perteneciente a la Universidad Politécnica de Valencia [20]. En la figura 3 se puede observar su página principal.

En estas plataformas los usuarios pueden acceder a diferentes espacios por cada asignatura que estén cursando o impartiendo. En estos, los profesores pueden publicar diferentes recursos a los que los alumnos tendrán acceso, por ejemplo, diapositivas utilizadas en clase o programas informáticos relevantes. Los profesores también pueden programar tareas o incluso exámenes que el alumno deberá realizar antes de una fecha y hora indicada, o con un límite de tiempo.

Además, se proporciona un acceso rápido a la aplicación de mensajería de la entidad, de manera que los usuarios puedan contactar unos con otros de manera cómoda y veloz y obtengan notificaciones cuando obtengan un mensaje o haya algún evento importante.

Las funciones de cada una de estas plataformas dependerán de la entidad que las desarrolle, pero un fin común entre todas es intentar tener un sitio web donde centralizar la mayoría de las actividades que se podrían realizar en un centro docente para la comodidad y facilidad de organización de todos sus usuarios.

Estas plataformas suelen estar construidas utilizando programas de e-learning que proporcionan herramientas diferentes para personalizar la propia web. Poliformat está construida utilizando Sakai Project [21], software utilizado también por grandes universidades como la Universidad de Cambridge y la Universidad de Yale. Otros programas similares son .LRN [22], utilizado en algunas importantes universidades españolas como la Universidad de Valencia y la Universidad Carlos III de Madrid, y también Moodle [23], utilizado por la Universidad Jaume I para su plataforma AulaVirtual. Dado que todas estas plataformas necesitan cumplir funciones muy similares, hay estudios que han concluido que no hay grandes diferencias entre ellas, que harían una preferible a las otras [24].

Un estudio reciente ha descubierto que casi un 50% de los estudiantes prefieren realizar cursos en línea, debido en parte a la capacidad de compaginar los estudios con otras responsabilidades como el trabajo o la familia [25]. Esto ha causado que muchas compañías lancen sus propias plataformas de aprendizaje, como por ejemplo Google Classroom, que es gratuita y además ofrece integración con otros productos y servicios de Google. La plataforma Brightspace, de D2L, es un ejemplo de una que facilita la educación basada en competencias, teniendo diferentes herramientas para determinar las fortalezas y debilidades de cada alumno y usando esta información para crear una experiencia de aprendizaje personalizada.

Un servidor de Discord podría organizarse de manera similar a una de estas plataformas, usando diferentes canales para separar cada asignatura y dar acceso a los recursos necesarios. El bot podría ayudar en funciones más complicadas como entregar tareas o publicar anuncios que sólo reciban las personas relevantes.

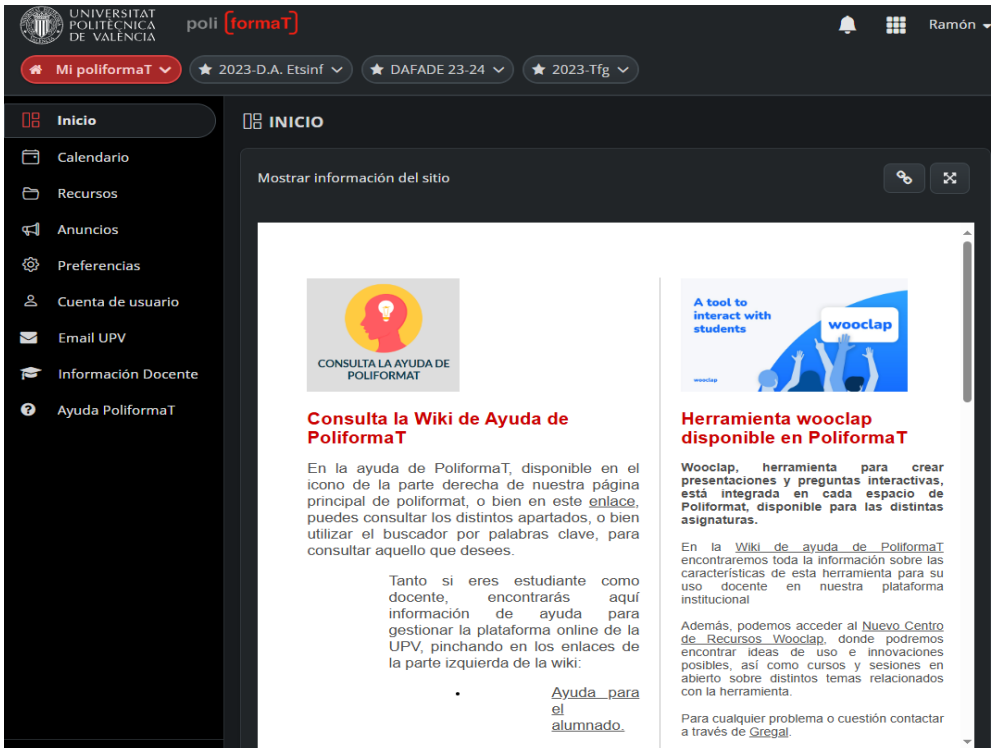

Figura 3: Página principal de Poliformat [20].

2.1.4 Aplicaciones para la planificación y organización

Los estudiantes recurren cada vez más a aplicaciones de organización porque les resulta difícil gestionar la alta cantidad de tareas que deben realizar cada día. Desde estudiar para exámenes a escribir trabajos o preparar presentaciones, todo tiene una fecha límite y un tiempo que se le debe dedicar.

Esta gran cantidad de tareas puede generar en el estudiante estrés por diversos motivos. La habilidad de organizarse eficientemente puede conllevar a diferentes beneficios para los alumnos como el ahorro de tiempo en diferentes actividades, no perderse eventos importantes, reconocer la relevancia de unas tareas sobre otras y mantenerse al día con todos los estudios [26]. Diversas aplicaciones pueden ayudar a planificar bien el tiempo que se le dedica a cada tarea y documentar todo lo importante de manera que no se olvide, pudiendo en muchos casos habilitar notificaciones que aseguren llamar la atención del usuario en el momento indicado.

Aplicaciones como Google Calendar [27] permiten dar al usuario una visualización de sus tareas de manera efectiva. Al crear una tarea en Google Calendar, se le puede asignar una fecha y hora específicas. Estas aparecen como bloques de color en el calendario, lo que facilita identificar rápidamente qué hay que hacer y cuándo. Se pueden personalizar los colores para diferentes tipos de tareas, creando un sistema visual que permita distinguir entre actividades de distinto tipo o de distintas asignaturas. En la figura 4 se ilustra un ejemplo de una semana planificada con diferentes actividades.

Sería conveniente incorporar al bot de Discord una función que permita a los usuarios programar sus propios recordatorios dentro de la aplicación, de manera que puedan tener la seguridad de que no van a olvidar tareas y puedan recibir todas las notificaciones relevantes sin tener que salir de Discord.

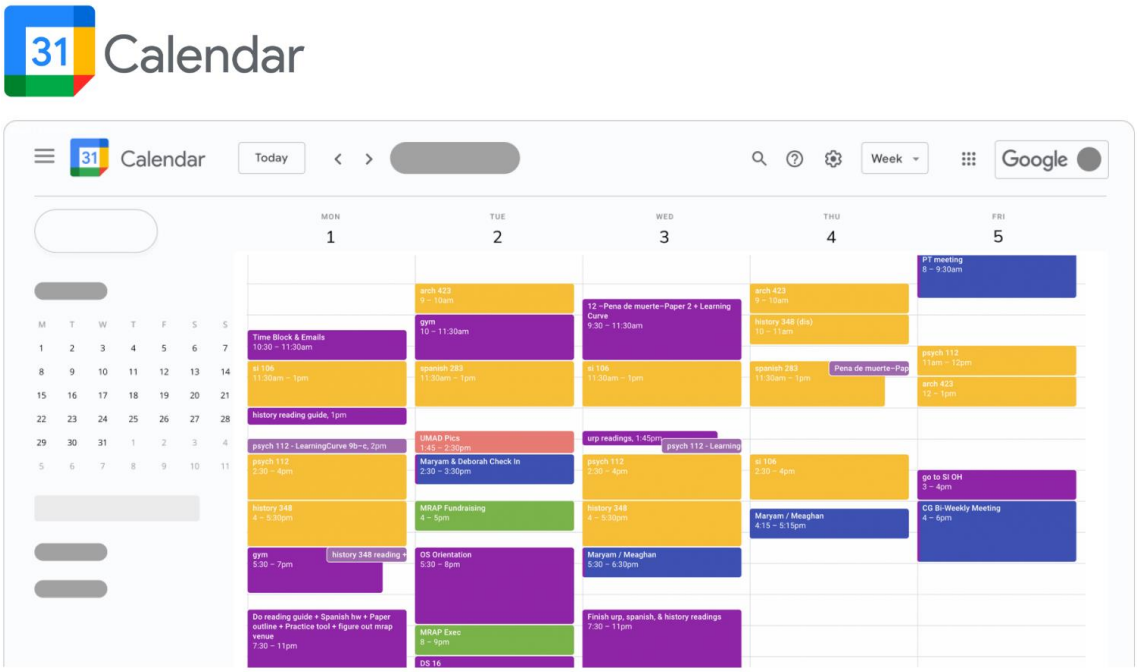

Figura 4: Planificación semanal en Google Calendar [28].

3. Requisitos y funcionalidad

En este apartado se van a detallar los requisitos y la funcionalidad del bot que se pretende desarrollar. Se especificarán las necesidades que se deberán satisfacer, tanto desde el punto de vista del usuario final como desde la perspectiva técnica. Se abordarán los requisitos funcionales, que describen lo que el sistema debe hacer, y los no funcionales, que definen cómo debe hacerlo. Además, se desglosará la funcionalidad prevista, explicando diferentes casos de uso resultantes de la interacción entre distintos tipos de usuarios y el bot.

3.1 Análisis de requisitos

Los requisitos funcionales describen lo que un sistema o programa debe hacer. Especifican las funciones concretas que un programa debe realizar para cumplir con su propósito y se centran en las acciones que los usuarios podrán realizar, los datos que el sistema debe procesar y las respuestas que se deberán proporcionar ante diferentes entradas o situaciones.

Tras analizar diferentes aplicaciones en la sección anterior, se ha decidido que los requisitos funcionales serán los siguientes:

- Asignación de roles a alumnos y profesores para que tengan acceso a los canales del servidor relacionados a las asignaturas que cursan.
- Responder a usuarios con diferentes mensajes de ayuda en función del comando que utilicen. Algunos de estos comandos sólo estarán disponibles para profesores.
- Organización de una actividad similar a Kahoot que los alumnos podrán realizar dentro del servidor.
- Mantener un sistema persistente de puntuación que fomente la competitividad entre alumnos y ayude al profesor a valorar el rendimiento de estos durante el curso.
- Permitir a los alumnos exponer dudas o feedback sobre la clase, con la posibilidad de hacerlo anónimamente.
- Habilidad de programar recordatorios periódicos automáticos de entregas a realizar o exámenes próximos.

Los requisitos no funcionales se refieren a las características y restricciones del programa que no están directamente relacionadas con las funciones específicas que realiza. Definen la calidad y las limitaciones del sistema y son cruciales para el éxito del proyecto, ya que afectan el rendimiento y la experiencia del usuario, aunque estos no tengan conocimiento de los procesos que tienen lugar para hacer funcionar a la aplicación.

Para el bot, algunos de sus requisitos no funcionales serían los siguientes:

- Tiempo de respuesta: El bot debe responder a las consultas de los usuarios en menos de 2 segundos en condiciones normales de uso.
- Compatibilidad: El bot debe funcionar correctamente en todas las plataformas donde Discord está disponible (Windows, macOS, iOS, Android, navegadores web).
- Mantenibilidad: El código del bot debe estar bien documentado y estructurado para facilitar futuras actualizaciones y correcciones.
- Disponibilidad: El bot debe estar operativo al menos el 99.9% del tiempo, especialmente durante las horas lectivas habituales.
- Usabilidad: La interacción con el bot debe ser intuitiva, con comandos fáciles de recordar y usar, incluso para usuarios con poca experiencia tecnológica.
- Seguridad: El bot no extraerá ningún tipo de información personal de los usuarios, y la información académica será compartida únicamente con profesores y compañeros.

3.2 Casos de uso

Los casos de uso se centran en el usuario y cómo interactúa con el sistema, mientras que los requisitos funcionales se enfocan en lo que el sistema debe hacer para permitir estas interacciones. Los casos de uso se pueden utilizar para validar los requisitos funcionales, asegurando que todas las funcionalidades necesarias para soportar las interacciones del usuario estén incluidas.

A continuación, se van a detallar todos los casos de uso que un usuario podrá realizar mediante el bot:

1. Obtener el rol de “Profesor” (Tabla 1).
2. Obtener una guía de uso del bot (Tabla 2).
3. Obtener una guía de introducción al servidor (Tabla 3).
4. Obtener el rol de la clase que se cursa (Tabla 4).
5. Eliminar el rol de una clase (Tabla 5).
6. Enviar tareas de una asignatura (Tabla 6).
7. Publicar dudas y consultas anónimamente (Tabla 7).
8. Responder a consultas de otros alumnos (Tabla 8).
9. Consultar los puntos de los alumnos (Tabla 9).
10. Añadir manualmente puntos a un alumno (Tabla 10).
11. Restar manualmente puntos a un alumno (Tabla 11).
12. Crear un recordatorio (Tabla 12).
13. Creación de preguntas para el juego (Tabla 13).
14. Comenzar una ronda del juego (Tabla 14).
15. Responder a una pregunta en el juego (Tabla 15).

Tabla 1: Caso de uso "Obtener el rol de 'Profesor'".

Caso de uso	Obtener el rol de "Profesor"
Objetivo	Añade el rol de "profesor" a un usuario, identificándole como personal docente y otorgando permisos administrativos no disponibles para alumnos, como la capacidad de ver todos los canales del servidor o borrar mensajes de otros usuarios. Para esto se utilizará una contraseña que sólo los profesores conocerán. En caso de que se filtre la contraseña, el responsable del código del bot podrá cambiarla.
Actor	Profesor que no disponga del rol en el servidor.
Flujo principal	Se la hace conocer al profesor la contraseña que debe utilizar. El profesor envía un mensaje directo al bot conteniendo la contraseña.
Final exitoso	El profesor recibe el rol en el servidor y un mensaje de confirmación del bot en sus mensajes directos para confirmar la operación.
Final fallido	El profesor escribe la contraseña en el servidor, causando que el bot borre su mensaje para evitar que se filtre el comando. El bot también enviará un mensaje recordando que el comando solo puede ser usado en mensajes directos.

Tabla 2: Caso de uso "Obtener una guía de uso del bot".

Caso de uso	Obtener una guía de uso del bot
Objetivo	Se le ofrecen al usuario instrucciones sobre cómo utilizar el bot.
Actor	Cualquier usuario.
Flujo principal	El usuario escribe el comando !ayuda.
Final exitoso	El bot envía un mensaje conteniendo todos los comandos disponibles tanto para usuarios como profesores, indicando también que en caso de duda pueden contactar con un docente.
Final fallido	No procede.

Tabla 3: Caso de uso "Obtener una guía de introducción al servidor".

Caso de uso	Obtener una guía de introducción al servidor
Objetivo	Similar al caso anterior, se le ofrece a un nuevo usuario instrucciones sobre cómo utilizar el bot y una pequeña guía de cómo empezar a usar el servidor.
Actor	Nuevos usuarios.
Flujo principal	El usuario se une al servidor por primera vez.
Final exitoso	El bot envía automáticamente un mensaje de ayuda al nuevo usuario mediante mensajes directos, incluyendo todos los comandos disponibles e indicaciones de uso del servidor.
Final fallido	No procede.

Tabla 4: Caso de uso "Obtener el rol de la clase que se cursa".

Caso de uso	Obtener el rol de la clase que se cursa
Objetivo	Permitir que el alumno tenga acceso a los canales del servidor de las asignaturas que curse.
Actor	Alumnos.
Flujo principal	El usuario hace click en un emoticono adjunto a un mensaje en el canal de introducción para indicar las clases que cursa (ejemplo: elegir el emoticono del tubo de ensayo indica que cursan química.)
Final exitoso	El usuario obtendrá los roles de las clases que curse, obteniendo acceso a los canales del servidor pertinentes.
Final fallido	No procede.

Tabla 5: Caso de uso "Eliminar el rol de una clase".

Caso de uso	Eliminar el rol de una clase.
Objetivo	Al contrario del caso anterior, este elimina un rol en caso de que un alumno lo haya escogido por error o deje de cursar la asignatura.
Actor	Alumnos.
Flujo principal	Exactamente igual que el caso anterior, el alumno deberá volver a hacer click en el emoticono.
Final exitoso	El usuario pierde el rol y con ello acceso a los canales de esa asignatura.
Final fallido	No procede.

Tabla 6: Caso de uso "Enviar tareas de una asignatura".

Caso de uso	Enviar tareas de una asignatura
Objetivo	Permitir a los alumnos realizar entregas de trabajos de una asignatura y que el profesor las reciba en un único lugar.
Actor	Alumnos.
Flujo principal	El usuario utiliza el comando !entregar en los mensajes directos del bot (para evitar que otros alumnos tengan acceso), seguido de la asignatura a la que pertenece y el archivo conteniendo su trabajo.
Final exitoso	Se envía un mensaje de confirmación al alumno. El bot reenvía la tarea junto con el nombre del alumno a un canal privado del servidor que solo el profesor de la asignatura puede ver. En este canal se almacenarán las entregas de todos los alumnos de la asignatura para facilitar la organización del profesor.
Final fallido	A. El alumno utiliza el comando en el servidor, causando que el bot borre su mensaje y envíe un mensaje recordando que el comando solo puede ser usado en mensajes directos. B. El alumno no indica una asignatura válida o no adjunta un archivo, causando que el bot publique un mensaje indicando su error.

Tabla 7: Caso de uso "Publicar dudas y consultas anónimamente".

Caso de uso	Publicar dudas y consultas anónimamente.
Objetivo	El servidor contará con canales para cada asignatura donde alumnos podrán publicar feedback y dudas sobre la clase. El bot permitirá hacer esto anónimamente si lo desean.
Actor	Alumnos.
Flujo principal	El usuario utiliza el comando !consulta seguido de la asignatura a la que pertenece y su mensaje.
Final exitoso	Se envía un mensaje de confirmación al alumno y el bot publica la consulta en el canal de dudas de la asignatura indicada.
Final fallido	El alumno no indica una asignatura válida o no adjunta un archivo, causando que el bot publique un mensaje indicando su error.

Tabla 8: Caso de uso "Responder a consultas de otros alumnos".

Caso de uso	Responder a consultas de otros alumnos.
Objetivo	En relación con el caso anterior, alumnos o profesores podrán responder a las consultas y el alumno que la publicó recibirá una notificación por cada respuesta.
Actor	Cualquier usuario.
Flujo principal	El usuario utiliza la función de Discord de responder a un mensaje específico para contestar a una consulta.
Final exitoso	El alumno que publicó la consulta recibirá una notificación con la respuesta y el nombre del usuario que le ha respondido en sus mensajes directos con el bot.
Final fallido	No procede.

Tabla 9: Caso de uso "Consultar los puntos de los alumnos".

Caso de uso	Consultar los puntos de los alumnos.
Objetivo	El sistema de puntos sirve como recompensa en diversas actividades en el aula para cuantificar el rendimiento de cada alumno. Este comando ofrece una rápida y sencilla visualización de los puntos de cada uno en comparación con los demás compañeros.
Actor	Cualquier usuario.
Flujo principal	El usuario utiliza el comando !top en cualquier canal del servidor.
Final exitoso	El bot publica una lista de cada usuario con sus respectivos puntos en orden de mayor a menor cantidad.
Final fallido	No procede.

Tabla 10: Caso de uso "Añadir manualmente puntos a un alumno".

Caso de uso	Añadir manualmente puntos a un alumno.
Objetivo	Si el profesor lo cree conveniente, puede otorgar a un alumno una cantidad especificada de puntos para premiar su rendimiento en el aula.
Actor	Profesores.
Flujo principal	El profesor utiliza el comando !sumar seguido del nombre del alumno y la cantidad de puntos que desee añadir.
Final exitoso	El bot envía un mensaje de confirmación y se suman los puntos indicados al alumno.
Final fallido	Si un alumno intenta utilizar este comando, no funcionará y el bot enviará un mensaje indicando que solamente puede ser utilizado por profesores.

Tabla 11: Caso de uso "Restar manualmente puntos a un alumno".

Caso de uso	Restar manualmente puntos a un alumno.
Objetivo	Al contrario del caso anterior, un profesor puede restar puntos a un alumno en casos de comportamiento o rendimiento inadecuados.
Actor	Profesores.
Flujo principal	El profesor utiliza el comando !restar seguido del nombre del alumno y la cantidad de puntos que desee restar.
Final exitoso	El bot envía un mensaje de confirmación y se restan los puntos indicados al alumno.
Final fallido	Si un alumno intenta utilizar este comando, no funcionará y el bot enviará un mensaje indicando que solamente puede ser utilizado por profesores.

Tabla 12: Caso de uso "Crear un recordatorio".

Caso de uso	Crear un recordatorio.
Objetivo	Permitir que profesores puedan programar recordatorios para que el bot ayude a los alumnos recordar eventos próximos como entregas de tareas o exámenes.
Actor	Profesores.
Flujo principal	El profesor utiliza el comando !recordatorio seguido de un número y una letra indicando la unidad de tiempo que debe esperar (s,m,h,d = segundos, minutos, horas, días). Seguido de esto se incluye el mensaje del recordatorio.
Final exitoso	El bot envía un mensaje de confirmación. Transcurrido el tiempo de espera, el bot notificará a todos los usuarios con el rol de la clase donde se utilizó el comando junto con el mensaje del recordatorio.
Final fallido	A. Si un alumno intenta utilizar este comando, no funcionará y el bot enviará un mensaje indicando que solamente puede ser utilizado por profesores. B. Si no se indica una unidad de tiempo aceptable, no funcionará y el bot enviará un mensaje explicando el error. C. Si la cantidad de tiempo no es un número entero, no funcionará y el bot enviará un mensaje explicando el error.

Tabla 13: Caso de uso "Creación de preguntas para el juego".

Caso de uso	Creación de preguntas para el juego.
Objetivo	Permite a los profesores añadir preguntas junto con sus respuestas a la base de datos que utilizará el juego inspirado en Kahoot.
Actor	Profesores.
Flujo principal	El profesor utiliza el comando !crear seguido de la cantidad de puntos que se otorgarán automáticamente a un alumno si responde correctamente. Seguido de esto escribe el título de la pregunta. Una vez enviado el mensaje, el bot pide que introduzca una respuesta a esta pregunta. El profesor la introduce y el bot pregunta si es correcta, indicando al profesor que responda con s o n (sí o no). Tras introducir al menos una respuesta, el profesor puede seguir añadiendo tantas respuestas como desee o puede finalizar el proceso en cualquier momento escribiendo 'fin'.
Final exitoso	El bot envía un mensaje de confirmación y la pregunta es añadida a la base de datos.
Final fallido	Si un alumno intenta utilizar este comando, no funcionará y el bot enviará un mensaje indicando que solamente puede ser utilizado por profesores.

Tabla 14: Caso de uso "Comenzar una ronda del juego.".

Caso de uso	Comenzar una ronda del juego.
Objetivo	Un profesor puede dar comienzo al juego inspirado en Kahoot, mostrando una pregunta y permitiendo que los alumnos respondan.
Actor	Profesores.
Flujo principal	El profesor utiliza el comando !pregunta.
Final exitoso	El bot publica en el canal la pregunta más antigua creada por el profesor en la base de datos junto con sus respuestas. El bot espera 10 segundos mientras los alumnos responden y transcurrido el tiempo anuncia a los alumnos que han acertado, otorgándoles los puntos indicados, o anuncia que nadie ha acertado la pregunta. La pregunta y sus respuestas son borradas de la base de datos para que no vuelvan a aparecer en el juego.
Final fallido	Si un alumno intenta utilizar este comando, no funcionará y el bot enviará un mensaje indicando que solamente puede ser utilizado por profesores.

Tabla 15: Caso de uso "Responder a una pregunta en el juego".

Caso de uso	Responder a una pregunta en el juego.
Objetivo	Permitir a los alumnos responder a las preguntas en el juego inspirado en Kahoot.
Actor	Alumnos.
Flujo principal	El alumno publica un mensaje en el canal con un número correspondiente a la respuesta que considera correcta. En caso de que publique varios mensajes dentro del límite de tiempo, solo se aceptará la primera respuesta.
Final exitoso	Al final de los 10 segundos, el bot publica un mensaje confirmando los usuarios que han acertado la pregunta, otorgándoles los puntos indicados.
Final fallido	No procede.

4. Desarrollo del bot

Una vez decididas las funcionalidades de las que se desea disponer, en este apartado se va a detallar el proceso de cómo se va a desarrollar el bot. Primero, se van a explicar las tecnologías que se van a utilizar en esta fase y a continuación se verán diferentes ejemplos de la programación de las funciones que deberá poder realizar el bot.

4.1 Tecnologías utilizadas

4.1.1 Visual studio code

Visual Studio Code [29] es un editor de código gratuito y de código abierto desarrollado por Microsoft. Es un entorno de desarrollo integrado (IDE) diseñado para hacer que la escritura y ejecución de código sea eficiente y agradable. También incluye características como control de versiones a través de Git, una terminal integrada y herramientas avanzadas de depuración de código, ofreciendo en una sola aplicación casi cualquier herramienta que un programador podría necesitar.

Además, soporta una gran cantidad de lenguajes de programación. Los más populares como Java, Python, C++ o Javascript vienen integrados de base en el programa, pero se puede descargar soporte para muchos otros lenguajes a través de extensiones que pueden ser instaladas dentro de la aplicación [30]. La figura 5 muestra el menú para ver las extensiones actuales o instalar otras nuevas. Esto lo hace ideal tanto para principiantes que están aprendiendo su primer lenguaje como para desarrolladores experimentados que trabajan en proyectos complejos. Las extensiones también pueden cumplir diferentes funciones como herramientas nuevas de depuración, de control de versiones o incluso temas visuales para personalizar la interfaz a gusto propio.

Visual Studio Code se ha escogido como programa para el desarrollo gracias a todas estas características y además porque se tiene experiencia previa utilizándolo, evitando así un tiempo de aprendizaje y errores que podrían darse a raíz de la falta de experiencia.

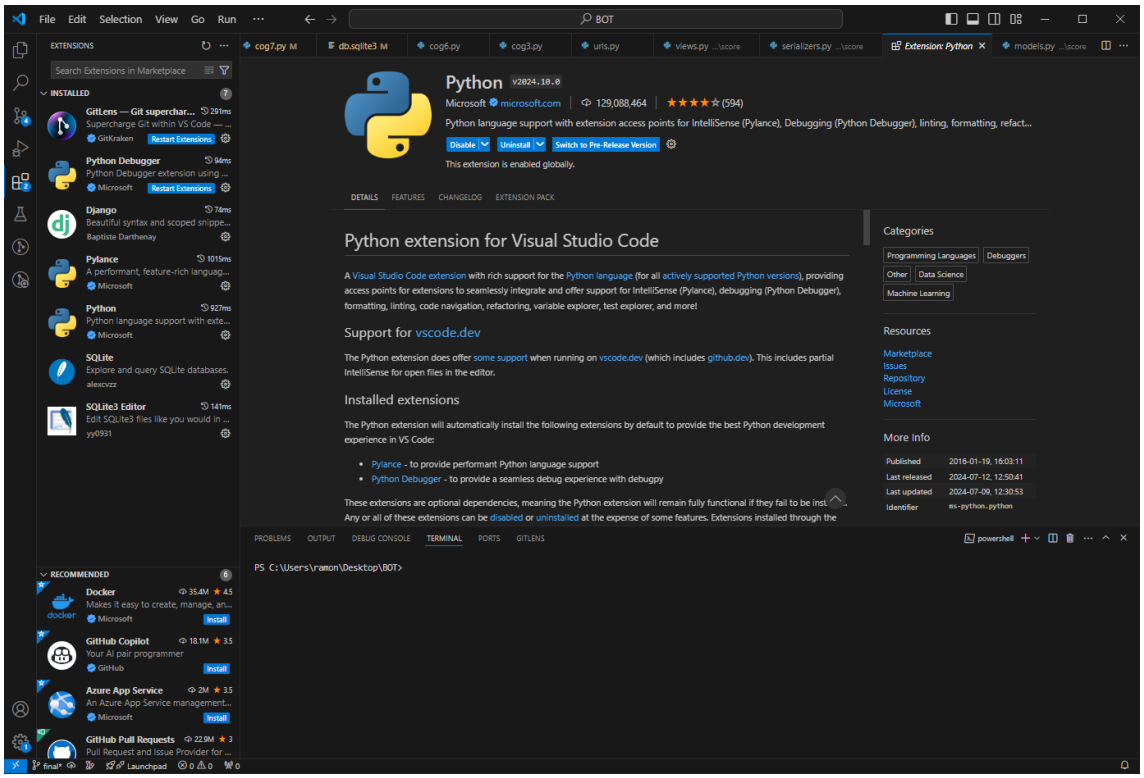

Figura 5: Menú de extensiones en Visual Studio Code.

4.1.2 Python

Python [31] es un lenguaje de programación de alto nivel, creado por Guido van Rossum y lanzado por primera vez en 1991. Se ha convertido en uno de los lenguajes más populares del mundo de la programación, conocido por su simplicidad y legibilidad.

Python enfatiza la legibilidad del código, utilizando indentación (espacios en blanco) para separar bloques de código, lo que fuerza a los programadores a escribir de manera ordenada y consistente. Esta característica, junto con su sintaxis clara y concisa, hace que Python sea especialmente amigable para principiantes.

Python es un lenguaje interpretado, lo que significa que no necesita ser compilado antes de ejecutarse. Esto permite un desarrollo más rápido ya que probar código consume menos tiempo. Además, se puede utilizar para una amplia variedad de aplicaciones, desde desarrollo web y análisis de datos hasta inteligencia artificial y automatización de tareas.

Una de las grandes fortalezas de Python es su extensa biblioteca estándar y la gran cantidad de paquetes de terceros disponibles. Esto permite a los desarrolladores acceder a una gran cantidad de funcionalidades preexistentes, ahorrando tiempo y esfuerzo en el desarrollo.

Una de estas bibliotecas es discord.py [32], la cual actúa como envoltorio de API (Interfaz de Programación de Aplicaciones) entre Discord y el programador y está diseñada específicamente para el desarrollo de bots, con una extensa documentación

para guiar a los diseñadores y ayudarles a aprender las diferentes funciones. La figura 6 muestra la introducción a esta biblioteca en la web.

Python es conocido por su manejo de tareas asíncronas, lo cual es muy necesario para los bots de Discord ya que deben poder manejar múltiples interacciones simultáneas sin bloquear el hilo principal de la ejecución.

Python posee una comunidad muy grande y activa, lo que significa que hay abundantes recursos, tutoriales y soporte disponibles para los desarrolladores que trabajan en bots de Discord o cualquier otro tipo de proyecto. Además, es un lenguaje multiplataforma, lo que facilita el desarrollo y despliegue del bot en sistemas operativos diferentes como Windows, Linux o MacOS.

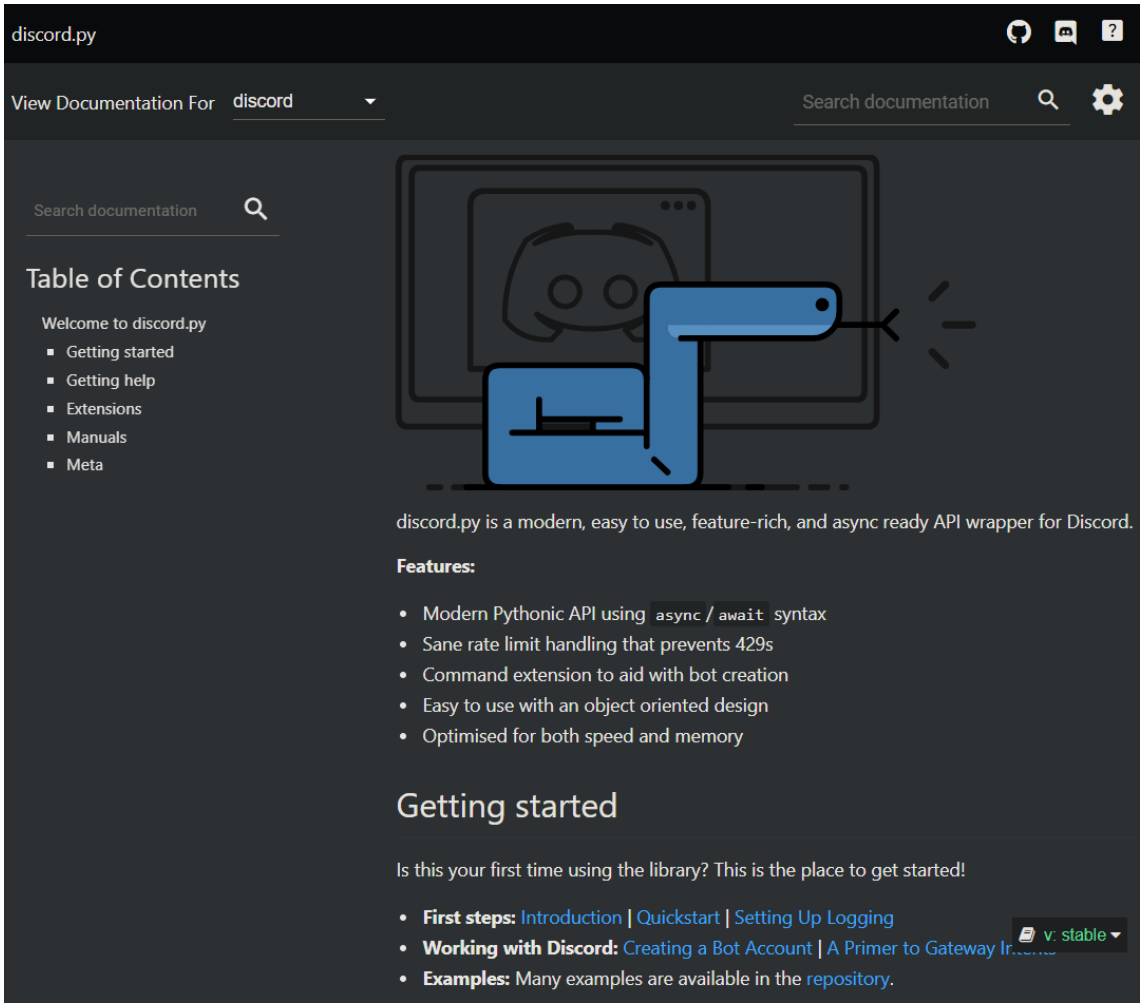

Figura 6: Web de la biblioteca Discord.py [32].

4.1.3 GIT

Git [33] es un sistema de control de versiones, creado por Linus Torvalds en 2005. Es una herramienta muy popular en el desarrollo de software moderno, diseñada para rastrear y gestionar cambios en el código fuente a lo largo del desarrollo de un proyecto.

Git permite a los programadores trabajar en proyectos de forma colaborativa, manteniendo un registro detallado de cada modificación. Cada desarrollador puede tener una copia completa del proyecto (comúnmente llamado repositorio) en su dispoistivo, hacer cambios y luego sincronizar estos cambios con los de otros desarrolladores. Aun así, también es una herramienta muy útil para proyectos en solitario.

Git utiliza "commits" para guardar el estado del proyecto en un momento dado. Cada commit es una copia exacta del proyecto en ese instante, con un mensaje que describe los cambios realizados, facilitando rastrear quién hizo qué cambios y cuándo. Esto permite guardar diferentes versiones del código a las que se puede regresar en cualquier momento si se desea revertir algún cambio o buscar la causa de un problema que en una versión anterior no ocurría.

Git también permite la creación de diferentes ramas. Las ramas permiten a los desarrolladores trabajar en diferentes versiones del proyecto simultáneamente, lo que es útil para desarrollar nuevas características o experimentar sin afectar la versión principal del código. Las ramas secundarias pueden combinarse con la principal para implementar los cambios realizados en ellas. La figura 7 muestra un simple ejemplo del manejo de ramas en Git.

Git también facilita la colaboración a través de plataformas en línea como GitHub [34], que proporcionan interfaces web para gestionar repositorios Git y colaborar con otros desarrolladores. En estas webs los usuarios pueden publicar sus propios proyectos, permitiendo si lo desean que cualquier persona pueda descargar el código y crear su propia rama de desarrollo.

Como se ha mencionado anteriormente, Visual Studio Code proporciona un soporte para Git, haciendo muy sencilla la implementación de esta poderosa herramienta para el control de versiones del proyecto.

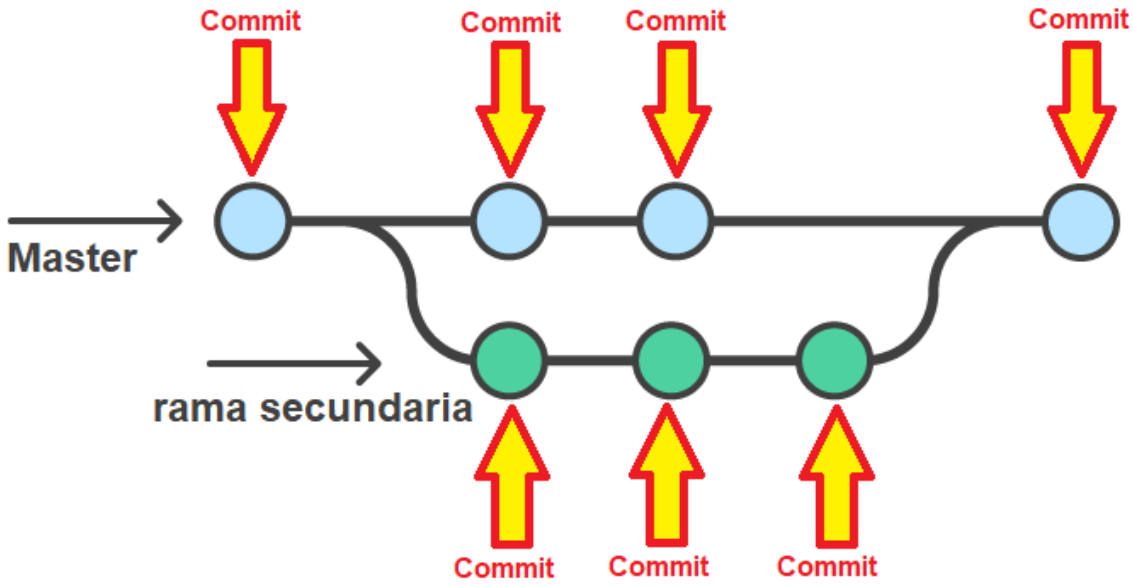

Figura 7: Ejemplo de diagrama de ramas de Git [35].

4.1.4 Django

Django [36] es un esquema de trabajo (framework) orientado a la ayuda para el desarrollo web. Está escrito en Python, permitiendo que pueda ser descargado como biblioteca para añadir funcionalidades a un proyecto escrito en este lenguaje.

Django sigue el patrón Modelo-Vista-Plantilla. El modelo representa la estructura de datos de la aplicación. Las vistas contienen la lógica del programa, recibiendo solicitudes web e interactuando con los modelos para recuperar o modificar datos. Estos datos son finalmente recibidos en la plantilla, que define cómo se van a presentar al usuario. Esta estructura permite una separación clara de cada función, facilitando el desarrollo y mantenimiento de aplicaciones web complejas. La figura 8 ilustra este patrón de una manera más visual para facilitar su comprensión.

Django también ofrece la herramienta Django Rest Framework [37] para la construcción de APIs (Interfaces de programación de aplicaciones) web. Las APIs RESTful típicamente utilizan métodos HTTP estándar (GET, POST, PUT, DELETE, etc.) para realizar operaciones en los recursos, y usan códigos de estado HTTP para indicar el resultado de las solicitudes. Este enfoque facilita la creación de servicios web que son fáciles de entender, mantener y escalar.

El uso de Django es necesario principalmente para el sistema de puntos de cada alumno y el juego inspirado en Kahoot. Para cada uno de estos, habrá una base de datos alojada en un ordenador remoto. Para poder posibilitar que el bot pueda comunicarse con la base de datos, será necesario crear una API utilizando Django, de manera que sirva como intermediaria entre ambos servicios.

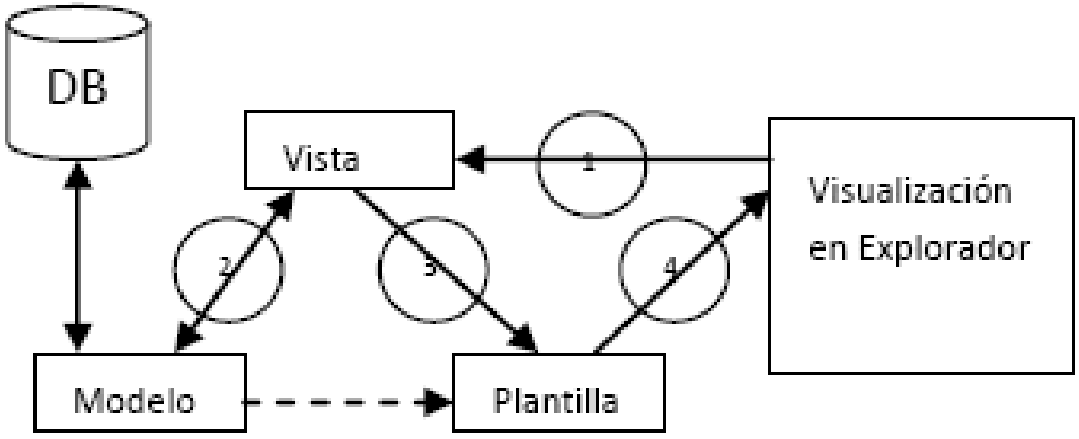

Figura 8: Ejemplo del patrón Modelo-Vista-Plantilla [38].

4.1.5 SQL

SQL (Lenguaje de Consulta Estructurada) es un lenguaje de programación diseñado para gestionar y manipular bases de datos relacionales. Desarrollado en la década de 1970 por IBM, SQL se ha convertido en el estándar para interactuar con bases de datos en una gran variedad de aplicaciones y sistemas.

Una base de datos relacional es un sistema de gestión de datos que organiza la información en tablas interconectadas. Estas tablas están compuestas por filas (también llamadas registros) y columnas (conocidas como campos). Cada tabla representa una entidad (en el caso del bot, habría tablas diferentes para los alumnos y las preguntas del juego) y las relaciones entre estas entidades se establecen mediante claves que conectan los datos entre diferentes tablas. La figura 9 muestra un ejemplo de diferentes tablas relacionadas entre si.

Una de las fortalezas de SQL es su capacidad para realizar consultas complejas, permitiendo extraer información precisa de grandes volúmenes de datos. Pudiendo filtrar, ordenar, agrupar y combinar datos de múltiples tablas usando instrucciones concisas.

SQLite [39] es un sistema de gestión de bases de datos relacional ligero y rápido que utiliza el lenguaje SQL. A diferencia de otros sistemas de bases de datos más grandes como MySQL o PostgreSQL, SQLite no requiere un servidor separado o configuración compleja, pudiendo ser integrado directamente en las aplicaciones. Esto lo hace más limitado a la hora de gestionar grandes cantidades de datos, pero para un pequeño proyecto como el bot esto no supone un problema.

SQLite también es el sistema de datos utilizado por Django, viniendo integrado con Python, lo cual permite su utilización sin necesidad de instalar y configurar un sistema de datos separado.

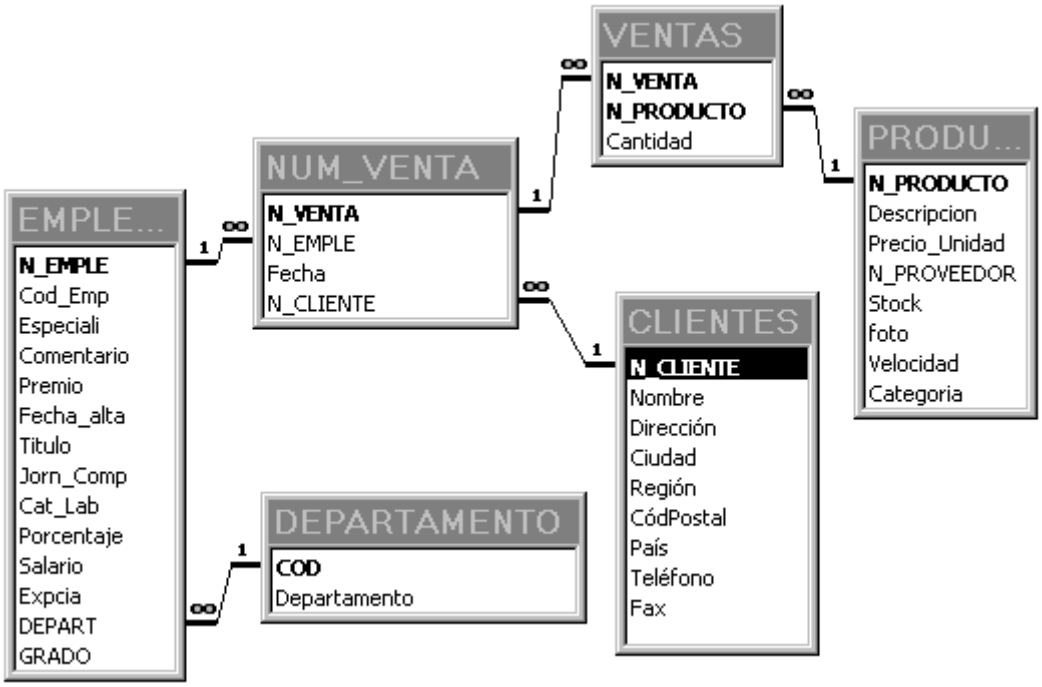

Figura 9: Ejemplo de tablas relacionales [40].

4.2 Proceso de desarrollo

4.2.1 Creación del servidor

El primer paso será crear el servidor de Discord que servirá como espacio para la clase en línea y al que se añadirá el bot. Este proceso es muy sencillo y cualquier persona con una cuenta de Discord puede realizarlo.

El creador tiene todos los permisos de administración del servidor, otorgándole poderes como invitar o expulsar a cualquier usuario que desee, borrar mensajes de otros usuarios, crear y eliminar diferentes canales o cambiar la imagen y nombre del servidor.

Los usuarios normales, en este caso los alumnos, tendrán permisos mucho más limitados para evitar que puedan causar problemas en el servidor. En general, solamente podrán enviar mensajes de texto en los canales permitidos y participar en los canales de voz donde se llevarán a cabo las clases.

Los profesores tendrán algunos permisos adicionales, como la habilidad de borrar mensajes de otros usuarios o silenciarles durante las llamadas para evitar posibles problemas. Para evitar tener que asignar permisos a cada usuario, se puede usar un sistema de roles que incorporan los servidores. Por ejemplo, se puede crear el rol “profesor” y configurarlo para que todos los usuarios con este rol tengan permisos adicionales. Luego, los profesores podrán obtener este rol mediante una contraseña usando el bot.

El bot deberá tener permisos de administración muy similares al dueño del servidor, ya que si no los tuvieran no sería posible desempeñar muchos de los casos de uso planeados. Esto se podrá configurar en la fase de creación del bot.

Una vez creado el servidor, se deberán crear diferentes canales, los cuales serán separados en diferentes grupos. El primer grupo será el General, al que todo usuario tendrá acceso, contando con un canal de texto y otro de voz en el que cualquier persona podrá participar sin importar qué clases estudie o qué rol tenga.

Este grupo también contará con un canal de texto llamado “Introducción”, que servirá para que los alumnos puedan elegir las clases que cursan, obteniendo los roles necesarios para acceder a diferentes grupos de canales usando el bot. En este canal no estará permitido que nadie más envíe mensajes para que siempre sea visible el mensaje de introducción.

Para los demás grupos, habrá uno por cada asignatura disponible. Se deberá crear un rol por cada una, y configurarlos para que otorguen permiso para poder ver y utilizar los canales del grupo correspondiente a la asignatura.

Cada uno de estos grupos tendrá cuatro canales. Uno será un canal de voz donde tendrán lugar las clases, otro será un canal de texto donde los alumnos y profesores podrán hablar entre ellos. Otro será un canal dedicado a dudas y consultas de la asignatura, donde alumnos podrán hacer preguntas o proponer ideas para mejorar la experiencia docente. El canal restante será uno de texto al que solo tendrá acceso el profesor. Aquí podrá utilizar diferentes comandos del bot de forma privada y también será donde reciba las tareas de los alumnos que las envíen a través del bot. El aspecto del servidor se puede apreciar en la figura 10.

Figura 10: Aspecto del servidor de Discord.

4.2.2 Instalación del bot

El segundo paso será la creación del bot y su integración en el servidor. Para ello, se debe acceder al portal de desarrolladores de Discord [41] y seleccionar crear una nueva aplicación.

En la pestaña de Información General , ilustrada en la figura 11, se podrá elegir un nombre y foto de perfil para el bot. Se ha decidido llamarlo EduBot, por su propósito de apoyo a la educación, así como una foto de perfil de un robot profesor para que nuevos miembros del servidor puedan comprender de inmediato que es una aplicación y no un usuario real. En esta página también se puede encontrar información como la cantidad de servidores en los que está instalado el bot.

En la pestaña de Bot se pueden elegir los permisos que se le otorgarán al bot una vez dentro del servidor, en este caso de Administrador para que pueda desempeñar todas las funciones sin límites. También se puede elegir si se desea que el bot sea público o no, lo cual permitiría a cualquier persona instalarlo en su servidor. Una vez configurados los permisos se generará un código que será muy importante guardar y no compartir con nadie, ya que este permite al usuario operar y programar el bot desde su ordenador.

También se podrá generar un enlace que permitirá instalar el bot en un servidor. Al pegar este enlace en el navegador, la aplicación de Discord lo detectará y preguntará a qué servidor se desea añadir. Tras elegir el servidor creado en el apartado anterior, el bot pedirá autorización para poder usar los permisos de administración que se le han otorgado, y una vez aceptado el bot podrá convertirse en un nuevo miembro del servidor.

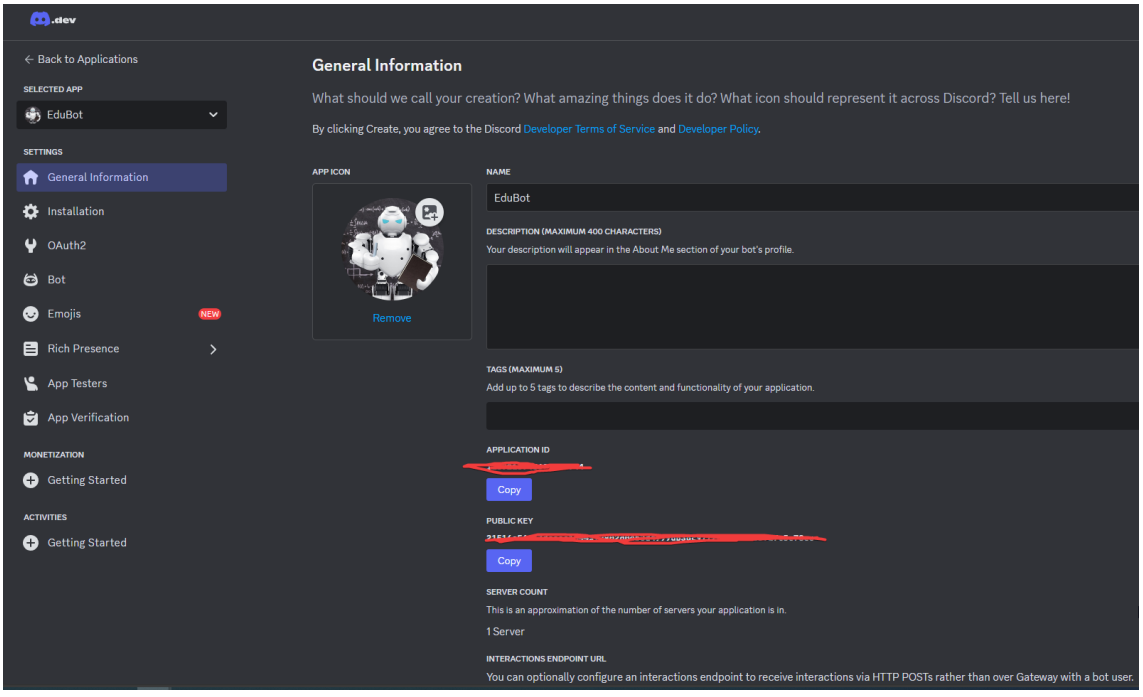

Figura 11: Pestaña general del menú de aplicaciones [41].

4.2.3 Programación inicial

Una vez añadido el bot se deberá comenzar con su programación. Desde Visual Studio Code se creará un nuevo proyecto, y en este un archivo llamado boy.py. Este archivo contendrá el código principal del bot y la extensión .py sirve para que el programa detecte que se está utilizando el lenguaje de programación de Python.

Desde la terminal integrada en Visual Studio Code, se utilizará el comando “pip install discord.py” para instalar la biblioteca de Discord para Python, la cual dará acceso a muchas funciones esenciales para el desarrollo del bot. También se deberá escribir una línea de código al inicio del fichero y en todos los que se creen a continuación diciendo “import discord”, para que el archivo permita utilizar las funciones de la biblioteca.

En esta biblioteca se encuentra el paquete “Client”, el cual sirve para crear una variable que permita una conexión con Discord. Al crear esta variable también se deberán declarar los “Intents” del bot, es decir, los permisos que tendrá dentro del servidor. También se podrá indicar el prefijo que indicará el uso de un comando, que en este caso será “!”. Una vez creada esta variable, será posible poner en marcha el bot utilizando la línea “client.run(token)”, donde token es una variable de texto en la cual se ha introducido el código obtenido en el apartado anterior. Con esto, será posible activar el bot escribiendo en la terminal “py bot.py”, comando que interpretará el archivo y ejecutará su código, manteniendo el bot activo hasta que se interrumpa el comando.

Para poder ver de forma más precisa cuándo se ha activado el bot tras escribir el comando, se puede utilizar un evento asíncrono de nombre “on_ready”, el cual realizará una acción cuando el bot esté preparado. Escribiendo “print(“Bot activado”)” en este evento, la terminal publicará un mensaje de confirmación cuando el bot esté listo para usarse, aunque este proceso no suele tardar más de 5 segundos. Este mensaje no lo verán los usuarios del servidor, ya que su intención es solamente como referencia para el programador.

Para la creación de diferentes comandos se van a preparar diferentes archivos “Cogs” [42]. Estos son módulos o extensiones que permiten definir comandos que pueden ser cargados al fichero de código principal. Los Cogs no son estrictamente necesarios, pero facilitan la organización del trabajo al tener cada función separada en un archivo distinto al principal, lo cual permite que este esté más limpio y que los comandos sean más fáciles de encontrar. La figura 12 muestra la implementación de estas funciones.

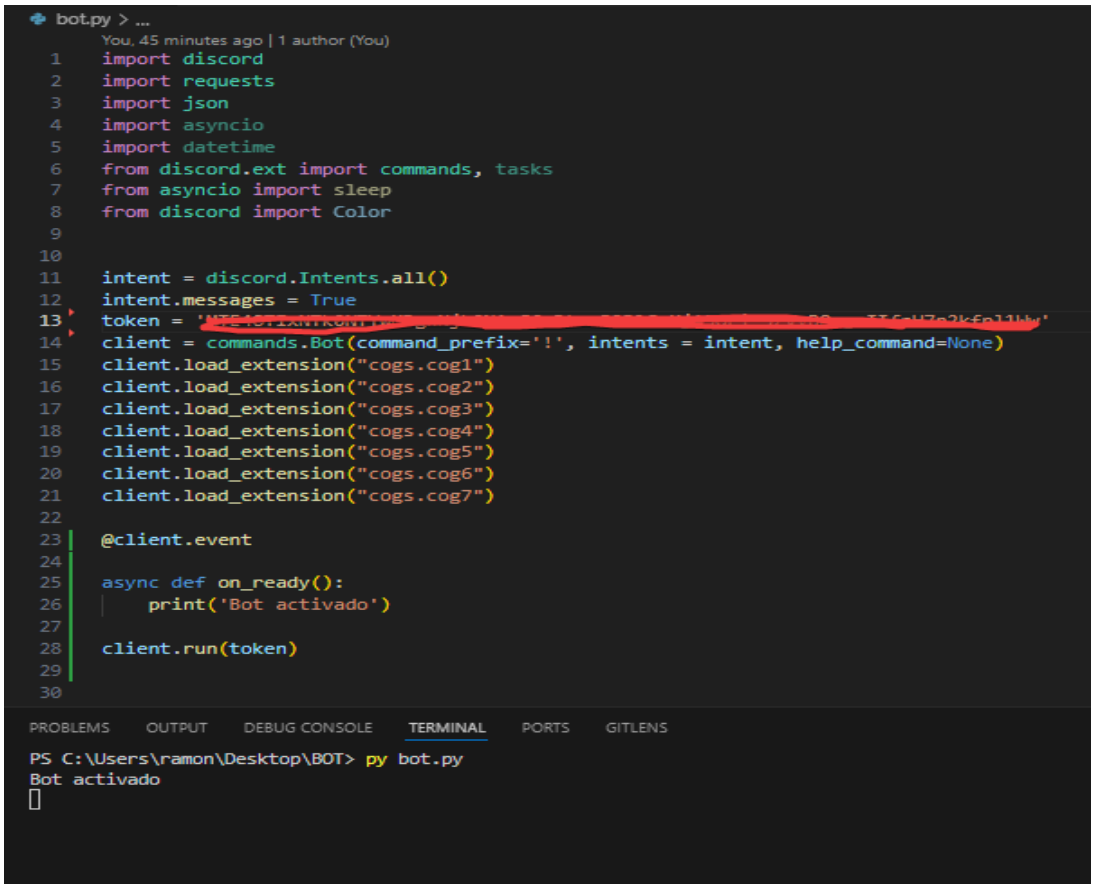

Figura 12: Programación inicial del bot.

4.2.4 Programación de los casos de uso

4.2.4.1 Mensajes de ayuda

Lo primero que se programará será el mensaje de introducción al servidor que el bot enviará a los nuevos usuarios, ya que esto será la primero que estos vean al entrar. Para esto, se puede usar el evento asíncrono “on_member_join”, el cual realiza una acción cuando detecte que se ha unido un nuevo miembro al servidor.

Ya que el mensaje contendrá mucha información, es deseable darle un formato más organizado y visualmente agradable. Para ello, la biblioteca de Discord posee una función llamada “Embed”, que permite personalizar el formato del mensaje. Se puede elegir un título, una descripción, el color de la caja de texto e incluso añadir una imagen que lo acompañe. También es posible separar diferentes cuadros de texto en columnas para distinguir información diferente de manera más intuitiva.

Una vez creado el Embed, simplemente hay que escribir una línea de código para que el bot lo envíe a los mensajes directos del usuario que se acaba de unir. En la figura 13 se muestra toda la información que se incluirá dentro del mensaje.

```python
async def on_member_join(member):

    embed = discord.Embed(
        title = 'Bienvenido al servidor',
        colour = discord.Color.dark_teal(),
        description = 'Aquí encontrarás todos los comandos disponibles de EduBot. Para acceder a tus clases, comprueba el canal #introducción.
        un comando especial para obtener su rol en el servidor.'
    )

    embed.set_thumbnail(url="https://i.imgur.com/k2J2F9T.png")
    embed.add_field(name='Comandos para todos', value='!ayuda: Consulta todos los comandos disponibles en el servidor.\n \n !top: Consulta una
                    !entregar: Envía un documento al profesor de una asignatura. Sólo se puede usar en los mensajes directos con el bot. Ejempl
                    \n \n !consulta: Publica anónimamente una consulta en el canal de dudas de la asignatura indicada. El usuario será avisado
                    Ejemplo: !consulta Química ¿Cuántos átomos tiene una molécula de agua? \n', inline=True)
    embed.add_field(name='Comandos para profesores', value=' !recordatorio: Programa un mensaje que alertará a los miembros de una clase sobre
                    Ejemplo: !recordatorio 2h entregar tarea \n \n !sumar: Otorga cierta cantidad de puntos a un usuario. Ejemplo: !sumar jose
                    cantidad de puntos a un usuario. \n \n !crear: Abre el editor de preguntas, en el comando se debe especificar los puntos qu
                    !crear 5 ¿Cuántos átomos tiene una molécula de agua? \n \n !pregunta: Publica la pregunta más antigua creada por el profeso
                    que respondan correctamente y posteriormente la borra de la base de datos.' )
    embed.set_footer(text='Para cualquier duda, contacta a un profesor')



    await member.send(embed=embed)
```

Figura 13: Código para el envío del mensaje de introducción.

Para el comando de ayuda, se puede reutilizar la mayoría de este código. La diferencia será que en lugar de utilizar el evento “on_member_join”, se utilizará el evento “on_message”, el cual detecta si un usuario ha enviado un mensaje específico. En este caso comprobará si alguien ha escrito el mensaje “!ayuda”, y a continuación enviará el mensaje programado. La figura 14 muestra el aspecto del mensaje de introducción dentro de Discord.

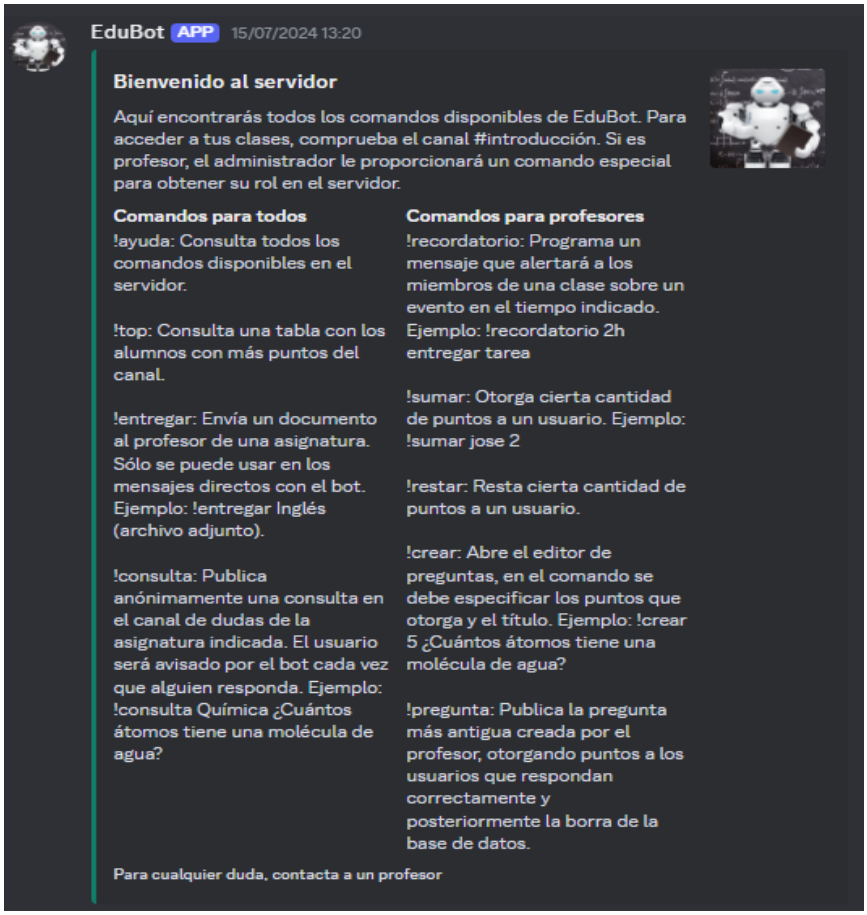

Figura 14: Mensaje de introducción al servidor.

4.2.4.2 Asignación de roles a usuarios

Para que los alumnos puedan asignarse a si mismos los roles de las asignaturas que estudian, se utilizará un sistema de reacciones con emoticonos en el mensaje de introducción. Para esto, se usará un nuevo evento llamado “on_raw_reaction_add”, el cual realiza una acción cuando detecta una reacción a un mensaje.

Primero se necesitará obtener el número identificador del mensaje al que se desea reaccionar, el cual se puede obtener desde Discord en las opciones del mensaje. Para evitar que el código funcione en otros mensajes, se incluirá una línea que compare el identificador del mensaje reaccionado con el mensaje de introducción, y si no son iguales terminará el evento.

A continuación, se deberá obtener el identificador del servidor de Discord, de nuevo siendo fácil de encontrar haciendo click derecho en el icono del servidor. Con este código se podrán acceder a los diferentes roles del servidor y convertirlos en variables dentro del código de Python. Finalmente, se comprobará el nombre del emoticono usado como reacción y se otorgará el rol de la asignatura acorde al usuario. La figura 15 muestra la implementación de este proceso.

```python
#ROLES
@client.event

async def on_raw_reaction_add(payload):

    reactors = 1192654435437002772

    server = client.get_guild(1086775885476139080)

    role1 = discord.utils.get(server.roles, name="Inglés")
    role2 = discord.utils.get(server.roles, name="Matemáticas")
    role3 = discord.utils.get(server.roles, name="Lengua")
    role4 = discord.utils.get(server.roles, name="Física")
    role5 = discord.utils.get(server.roles, name="Química")
    role6 = discord.utils.get(server.roles, name="Biología")

    if payload.message_id != reactors:
        return

    if payload.emoji.name == 'us':
        await payload.member.add_roles(role1)
    elif payload.emoji.name == '':
        await payload.member.add_roles(role2)
    elif payload.emoji.name == 'es':
        await payload.member.add_roles(role3)
    elif payload.emoji.name == '':
        await payload.member.add_roles(role4)
    elif payload.emoji.name == '':
        await payload.member.add_roles(role5)
    elif payload.emoji.name == '':
        await payload.member.add_roles(role6)
```

Figura 15: Código de asignación de roles de asignaturas.

Para permitir deshacerse del rol, se utilizará el evento “on_raw_reaction_remove”, que funciona al contrario que el evento anterior, realizando una acción cuando el usuario elimina su reacción. Esto se puede hacer simplemente volviendo a pinchar en el emoticono que se haya utilizado como reacción previamente. El código será igual que antes, pero cambiando la función “add_roles” a “remove_roles”.

Para asignar el rol de Profesor se desea usar un sistema de contraseñas, por lo que se utilizará un evento “on_message” que detecte si el mensaje enviado contiene la contraseña.

Para que el profesor no filtre la contraseña por error, el bot comprobará si el mensaje ha sido enviado en un servidor. Si se da este caso, el bot borrará el mensaje con la contraseña, lanzando una advertencia de que el código sólo puede ser utilizado en mensajes directos con el bot, como se puede apreciar en la figura 16.

Si se envía la contraseña correctamente, se le asignará el rol de Profesor. El bot será capaz de encontrar al usuario dentro del servidor aunque el comando no se haya realizado allí porque extraerá el identificador del autor del mensaje y procederá a buscarlo dentro del servidor.

```
if message.content.startswith('!profesorcontraseña'):
    server = client.get_guild(1086775885476139080)
    role = discord.utils.get(server.roles, name="Profesor")
    member = server.get_member(message.author.id)
    guild = message.guild
    if guild is None:
        await member.add_roles(role)
        await message.channel.send('Se le ha otorgado el rol de Profesor')
    else:
        await message.delete()
        await message.channel.send('Este comando sólo se puede usar en mensajes directos con el bot')
```

Figura 16: Código de asignación del rol “Profesor”.

4.2.4.3 Entregar una tarea

Para los siguientes comandos, se empezará a hacer uso de los Cogs. Cuando se crea una clase de este tipo, se debe inicializar incluyendo la variable cliente para poder utilizarla dentro de los comandos.

Todos los Cogs tienen 2 partes, la primera es la clase que incluye el código que cumplirá el Cog, y la segunda es una función “setup”, que suele ser una línea de código muy simple que sirve para añadir el Cog al código principal. Para la primera parte, si se busca crear un comando, se deberá utilizar el decorador “@commands.command()” y el nombre del comando será derivado automáticamente del nombre de la clase.

Al crear la clase, siempre se deberán incluir por lo menos 2 parámetros. El primero es “self”, el cual se refiere al propio bot, y el segundo es “ctx” (contexto), que hace referencia al mensaje que ha invocado al comando. Se podrán incluir más parámetros dependiendo de qué función vaya a cumplir el comando, pero en este caso solamente se añadirá la variable “clase”, que indica la asignatura a la que se va a entregar la tarea.

Al utilizar el comando, el alumno deberá escribir “!entregar” seguido del nombre de la asignatura y adjuntar el archivo de la tarea en el mismo mensaje. El bot utilizará el nombre de la asignatura para encontrar el canal del profesor correspondiente a ella mediante su código identificador y enviará allí el nombre del autor del mensaje junto con el archivo, al que accederá mediante la función “ctx.message.attachments[0].url”. También borrará el mensaje y mandará un error si el alumno lo utiliza dentro del servidor para evitar que otros alumnos pueden ver la entrega y prevenir posibles plagios. El código de este proceso aparece en la figura 17.

```
10
    You, last week | 1 author (You)
11  class cog2(commands.Cog):
12      def __init__(self, client):
13          self.client = client
14
15
16
17      @commands.command(name='entregar')
18
19      async def entregar(self, ctx, clase):
20
21
22
23          for guild in self.client.guilds:
24
25
26              if clase == 'Inglés':
27                  channel = guild.get_channel(1194962712552296448)
28              elif clase == 'Matemáticas':
29                  channel = guild.get_channel(1194962765966737418)
30              elif clase == 'Lengua':
31                  channel = guild.get_channel(1194962796606148708)
32              elif clase == 'Física':
33                  channel = guild.get_channel(1194962828654821466)
34              elif clase == 'Química':
35                  channel = guild.get_channel(1194962866302877747)
36              elif clase == 'Biología':
37                  channel = guild.get_channel(1194962894123708459)
38
39              if ctx.guild is None:
40                  file = ctx.message.attachments[0].url
41                  await ctx.channel.send('Se ha enviado la tarea.')
42                  await channel.send(f"{ctx.author} {file}")
43
44              else:
45                  await ctx.message.delete()
46                  await ctx.channel.send('Este comando sólo se puede usar en mensajes directos con el bot')
47
48
49
50
51  def setup(client):
52      client.add_cog(cog2(client))
```

Figura 17: Código del comando “entregar”.

4.2.4.4 Publicar y responder consultas

Para este comando se utilizarán 2 parámetros adicionales. El primero es un asterisco, el cual no es una variable, sino que indica que todo el texto restante en el mensaje formará parte del siguiente parámetro, al que se le llamará “text”. De este modo, el usuario utilizará el comando seguido del nombre de la asignatura, que al igual que el comando anterior formará la variable “clase”, y todo el texto siguiente formará la variable “text” que será el mensaje que se enviará al canal de consultas de la asignatura indicada.

El principio del código será muy similar al comando “entregar”, con la diferencia de que buscará el canal de consultas en lugar de el del profesor para enviar allí el mensaje. En este caso, el bot no incluirá el nombre del usuario que ha usado el comando, permitiendo realizar consultas de forma anónima. El bot guardará también una variable “id”, que hará referencia al mensaje que se ha enviado.

En la segunda parte del código se creará una función “check”, que comprobará 3 condiciones para determinar si un mensaje es una respuesta a la consulta. En primer lugar, comprobará si el canal al que se ha enviado el mensaje es igual al canal donde el bot envió la consulta. A continuación se asegurará de que se ha usado la función de respuesta a un mensaje específico, lo cual es posible consultando si el mensaje tiene una referencia utilizando la línea “m.reference is not None”. Finalmente verificará si el identificador del mensaje de referencia corresponde a la variable “id” guardada anteriormente.

Si se cumplen las 3 condiciones, el bot enviará la respuesta a los mensajes directos del autor de la consulta, indicando el nombre del usuario que le ha respondido y el contenido de su mensaje. Este fragmento del código se ubicará dentro de un bucle “while(True)” de manera que se pueda repetir cada vez que un usuario responda a la consulta. La figura 18 ilustra el código de este proceso y la figura 19 muestra un ejemplo de uso del comando dentro de Discord, con un alumno publicando una consulta y un profesor respondiéndola.

```python
class cog5(commands.Cog):
    def __init__(self, client):
        self.client = client


    @commands.command(name='consulta')

    async def consulta(self, ctx, clase,*,text):


        for guild in self.client.guilds:


            if clase == 'Inglés':
                channel = guild.get_channel(1197591355065643008)
            elif clase == 'Matemáticas':
                channel = guild.get_channel(1197591407477665942)
            elif clase == 'Lengua':
                channel = guild.get_channel(1197591500494741616)
            elif clase == 'Física':
                channel = guild.get_channel(1197591547093454908)
            elif clase == 'Química':
                channel = guild.get_channel(1197591602156290190)
            elif clase == 'Biología':
                channel = guild.get_channel(1197591647932911697)


            id = await channel.send(f"{text}")


            def check(m):
                return m.channel == channel and m.reference is not None and m.reference.message_id ==id.id

            while(True):

                msg = await self.client.wait_for(event='message', check=check)

                await ctx.author.send(f'Tu consulta "{text}" ha sido respondida por {msg.author.nick} con el mensaje "{msg.content}"')




def setup(client):
    client.add_cog(cog5(client))
```

Figura 18: Código del comando “consulta”.

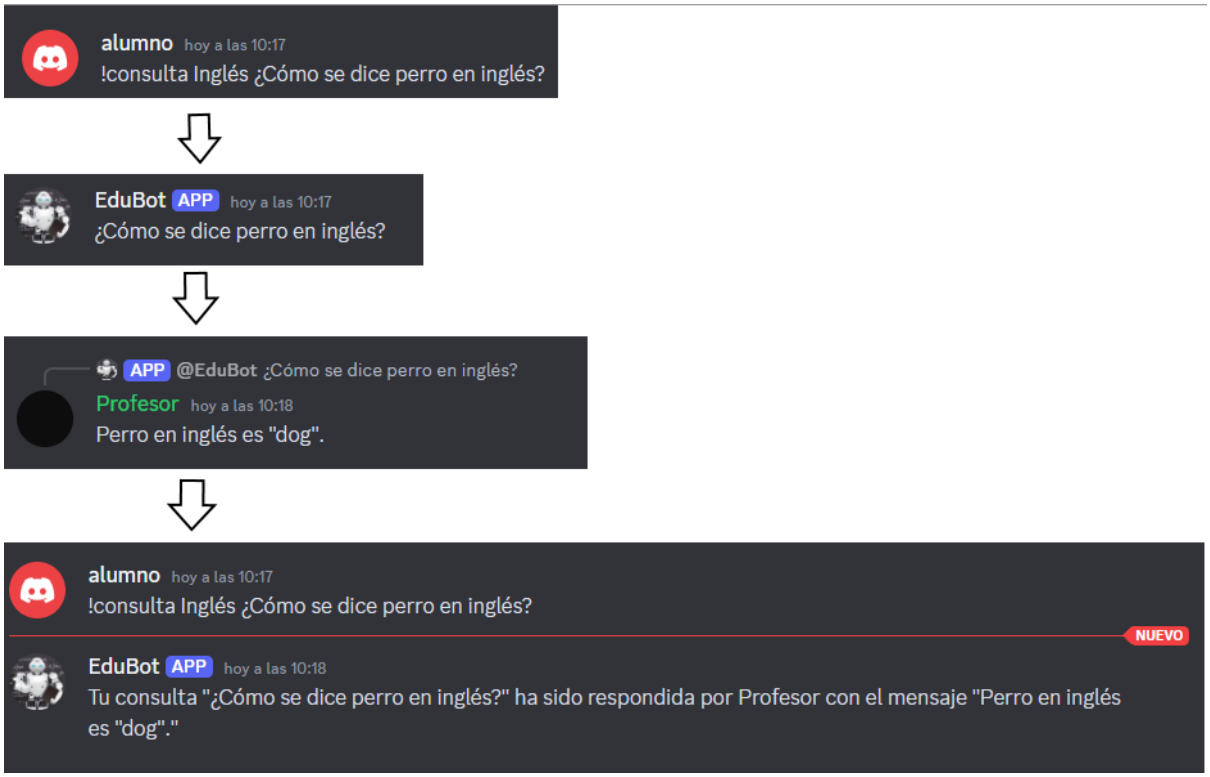

Figura 19: Proceso de publicación y respuesta de consultas.

4.2.4.5 Creación de recordatorios

En este comando se utilizará el parámetro “time”, que indicará el tiempo que deberá esperar el bot para publicar el recordatorio. El usuario deberá incluir después del comando un número seguido sin espacios de una letra para indicar la unidad de tiempo a esperar. Habrá 4 letras posibles, “s” para segundos, “m” para minutos, “h” para horas y “d” para días. Por ejemplo, si el usuario escribe “4h”, el bot esperará 4 horas para publicar el recordatorio. A este parámetro le seguirá un asterisco y después el parámetro “task”, el cual representará el mensaje que contendrá la alerta.

Como este será un comando exclusivo para profesores, el primer paso será comprobar si el autor del mensaje posee o no el rol de Profesor. En caso de que no lo tenga, el bot simplemente enviará un mensaje de error.

Este comando será usado por los profesores en los canales generales de las asignaturas, para que todos los alumnos de esa clase puedan verlo. El bot utilizará el código identificador del canal de texto para extraer el rol de la asignatura al que corresponde y, una vez transcurrido el tiempo de espera, notificará a todos los alumnos con el rol incluyendo el mensaje contenido en la variable “task”.

Como a Python solamente se le puede indicar que espere en unidades de segundos, se añadirá una función que convertirá el tiempo indicado a segundos. Por ejemplo, si el usuario ha indicado “5m” como tiempo de espera, la función detectará que al usar minutos deberá multiplicar el tiempo por 60, obteniendo un valor de 300 que se pasará a la función “await asyncio.sleep(converted_time)”.

La función también dispondrá de comprobaciones para diferentes fallos, como el uso de un número no entero o de una unidad de tiempo diferente a las 4 disponibles, proporcionando el mensaje de error adecuado para cada uno, visible en la figura 20.

```python
class cog1(commands.Cog):
    def __init__(self, client):
        self.client = client

    @commands.command(name='recordatorio')

    async def recordatorio(self, ctx, time, *, task):


        role = discord.utils.get(ctx.guild.roles, name="Profesor")



        if role in ctx.author.roles:

            if ctx.message.channel.mention == '<#1192261052713423089>':
                rol = discord.utils.get(ctx.guild.roles, name="Inglés")
            elif ctx.message.channel.mention == '<#1192260060051681402>':
                rol = discord.utils.get(ctx.guild.roles, name="Matemáticas")
            elif ctx.message.channel.mention == '<#1192261294791856240>':
                rol = discord.utils.get(ctx.guild.roles, name="Lengua")
            elif ctx.message.channel.mention == '<#1192261378669559878>':
                rol = discord.utils.get(ctx.guild.roles, name="Física")
            elif ctx.message.channel.mention == '<#1192261442779484311>':
                rol = discord.utils.get(ctx.guild.roles, name="Química")
            elif ctx.message.channel.mention == '<#1192261498916057108>':
                rol = discord.utils.get(ctx.guild.roles, name="Biología")


            def convert(time):
                pos = ['s', 'm', 'h', 'd']

                time_dict = {"s": 1, "m": 60, "h": 3600, "d": 360024}

                unit = time[-1]

                if unit not in pos:
                    return -1
                try:
                    val = int(time[:-1])
                except:
                    return -2

                return val * time_dict[unit]

            converted_time = convert(time)



            if converted_time == -1:
                await ctx.send("No has puesto el tiempo correctamente.")
                return

            if converted_time ==-2:
                await ctx.send("El tiempo debe ser un número entero.")
                return

            await ctx.send(f"Se ha programado un recordatorio para {task} en {time}.")

            await asyncio.sleep(converted_time)
            await ctx.send(f"{rol.mention} Esto es un recordatorio para {task} ")

        else:
            await ctx.send('No tienes permiso para utilizar este comando.')
```

Figura 20: Código del comando “recordatorio”.

4.2.4.6 Sistema de puntos para alumnos

Para almacenar los puntos de cada alumno será necesario crear una base de datos. Para ello, al crear un proyecto en Django se creará automáticamente un archivo llamado "models.py" [43], facilitando la creación de tablas de datos. La tabla creada contendrá el nombre del usuario y la cantidad de puntos que posea.

Para cada atributo, se deberá indicar de qué tipo de valor se trata. Para el nombre se usará un "CharField", el cual es un valor comúnmente usado para almacenar cadenas de texto (strings) cortas [44]. Para los puntos se usará un "IntegerField", que representa números enteros en un rango entre -2147483648 y 2147483648 [45]. La creación de esta tabla puede verse en la figura 21.

Una vez creada la clase, el comando "py manage.py makemigrations" creará la tabla indicada y a continuación "py manage.py migrate" aplicará esta tabla a la base de datos. Manage.py es un archivo que se crea automáticamente al instalar Django en un proyecto y permite realizar diferentes acciones de administración [46]. Al no haber indicado un atributo como clave primaria Django otorgará automáticamente un valor numérico como identificador para cada alumno.

```
...
from django.db import models


...
class Score(models.Model):

    name = models.CharField(_("nombre"), max_length=255)
    points = models.IntegerField(_("puntos"))

    def __str__(self):
        return self.name

```

Figura 21: Archivo models.py de la tabla Score.

El siguiente paso será crear un super usuario, ilustrado en la figura 22, el cual es un usuario que actuará como administrador. Para ello, se usará el comando "py manage.py createsuperuser", indicando también un nombre de usuario y contraseña. Después, se accederá al archivo "admin.py" [47], el cual es otro fichero creado automáticamente por Django, y se importará el modelo creado, indicando también los atributos que se desean poder consultar o modificar en el área de administrador dentro del servidor que se lanzará a continuación.

```python
from django.contrib import admin

from . import models
...

@admin.register(models.Score)

class ScoreAdmin(admin.ModelAdmin):
    list_display = [
        'name',
        'points',
    ]

```

Figura 22: Archivo admin.py de la tabla Score.

Para poner en marcha el servidor se usará el comando “py manage.py runserver”, y se podrá acceder a este a través del navegador utilizando la dirección ip de la máquina local, que siempre será “127.0.0.1:8000”. Si a esta dirección se la añade al final “/admin” se podrá acceder a la zona del administrador, y una vez indicados el nombre y la contraseña del super usuario creado anteriormente se podrán manipular las diferentes tablas de la base de datos desde el navegador. Por ejemplo, pudiendo añadir o eliminar usuarios y modificar sus puntos.

A continuación, se necesitará un método para que el bot pueda enviar y recibir datos al servidor. Primero, se creará una vista para la API en “views.py” [48]. Aquí se crearán 2 clases, la primera modificará los puntos de cada alumno mediante un método “post”, creando una nueva entrada en la tabla si es un alumno que no tenía puntos anteriormente o actualizando los puntos del alumno si ya existía en la base de datos. La segunda clase usará un método “get” para obtener información de la tabla y poder mostrar los puntos de cada alumno dentro del servidor en orden descendiente. La figura 23 ilustra esta vista.

También se deberá crear un fichero serializador [49]. Este es una clase que comprueba los datos que se reciben a través de la API y los compara con los descritos en la clase para verificar que son válidos para luego convertirlos en un formato que pueda ser guardado en la base de datos.

```python
from django.shortcuts import render
from rest_framework.views import APIView
from .serializers import ScoreSerializer
from rest_framework.response import Response
from rest_framework import status
from django.db.models import F

from .models import Score

You, 7 months ago | 1 author (You)
class UpdateScores(APIView):

    def post(self, request, format=None):
        serializer = ScoreSerializer(data=request.data)
        if serializer.is_valid():

            name = serializer.validated_data['name']
            points = serializer.validated_data['points']

            if Score.objects.filter(name=name).exists():
                serializer = Score.objects.get(name=name)
                serializer.points = F('points') + points

            serializer.save()

            return Response(None, status=status.HTTP_201_CREATED)

        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

You, 6 months ago | 1 author (You)
class Leaderboard(APIView):

    def get(self, request, formate=None, **kwargs):
        scores = Score.objects.all().order_by('-points')[:10]
        serializer = ScoreSerializer(scores, many=True)
        return Response(serializer.data)
```

Figura 23: Archivo views.py de la tabla Score.

Una vez creada la vista, se deberá conectar al servidor a través de una dirección url. Para ello, se usará el archivo “urls.py” [50] creado por Django. Aquí se detallará la dirección que se desea usar y la clase de la vista que realizará funciones en ella. Por ejemplo, la dirección “http://127.0.0.1:8000/api/score/leaderboard/” utilizará la clase de la vista “Leaderboard”, y una vez accedida a ella a través del navegador se mostrarán los puntos de cada alumno en la base de datos. La figura 24 muestra la conexión de la vista anterior con las urls y en la figura 25 se puede apreciar cómo se vería la clase “Leaderboard” de la vista en el navegador.

```python
from django.contrib import admin
from django.urls import path
from .quiz.views import ChooseOldestQuestion, ChooseNewestAnswer, UpdateQuestion, UpdateAnswer, DeleteOldestQuestion
from .score.views import UpdateScores, Leaderboard
from .quiz.models import Question

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/score/update/', UpdateScores.as_view(), name='score_update'),
    path('api/score/leaderboard/', Leaderboard.as_view(), name='leaderboard'),



]
```

Figura 24: Archivo urls.py de la tabla Score.

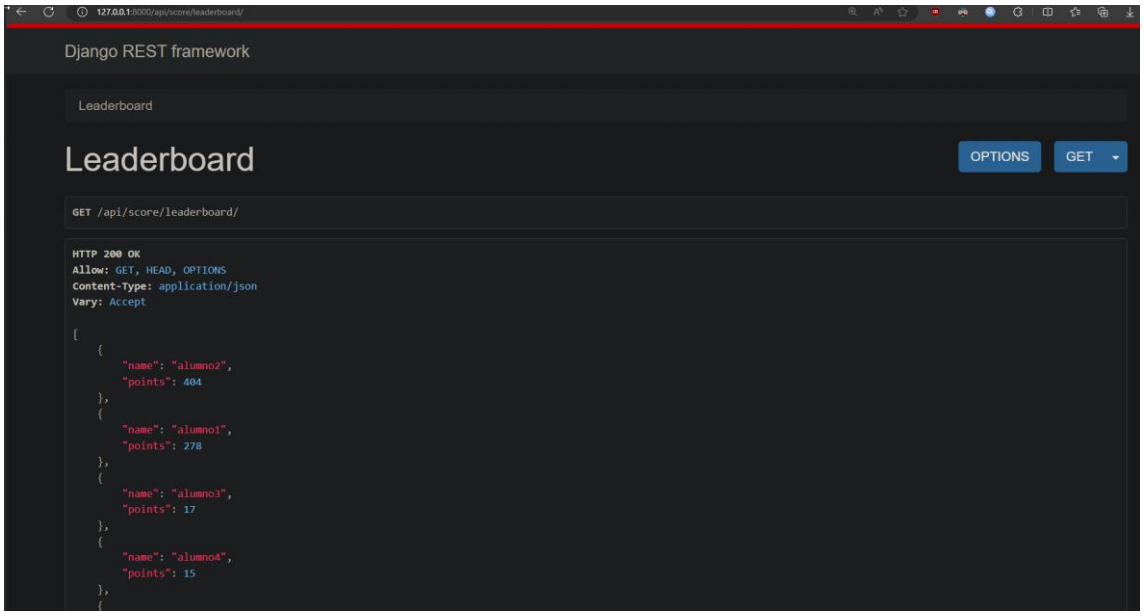

Figura 25: Acceso a la url de la clase “Leaderboard” desde el navegador.

Para que el bot pueda mostrar esta información dentro de Discord se deberá crear un método dentro de bot.py que acceda a la url cuando el servidor esté en marcha. Los datos obtenidos se podrán guardar en una variable que actuará como un “array” conteniendo todas las filas de la tabla obtenida. Accediendo a cada elemento de esta variable a través de un bucle “for” se podrá conseguir que el bot muestre toda la información necesaria. Una vez descrita esta función, simplemente habrá que invocarla cuando el usuario escriba “!top” para que el bot muestre los puntos de los usuarios.

Este comando podrá ser usado por cualquier usuario sin necesidad de estar conectado a la dirección ip de la máquina donde se aloja el servidor, siempre y cuando este esté conectado. La figura 26 muestra un uso del comando dentro de Discord, usando usuarios de prueba.

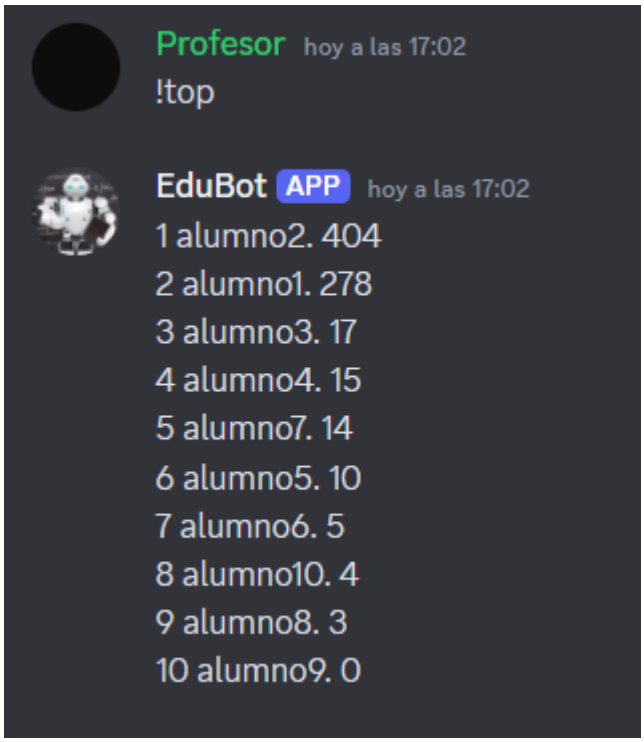

Figura 26: Uso del comando !top.

Para actualizar los puntos se creará una función que se invocará cuando un profesor utilice los comandos !sumar y !restar seguido del nombre del alumno y los puntos que se desean modificar. La función accederá a la url de la vista “UpdateScores” y utilizará la información proporcionada como argumentos para buscar si el alumno existe o no y otorgarle o quitarle los puntos indicados. El código de esta función aparece en la figura 27.

```
def get_score():
    leaderboard = ''
    id = 1
    response = requests.get("http://127.0.0.1:8000/api/score/leaderboard/")
    json_data = json.loads(response.text)

    for item in json_data:
        leaderboard += str(id) + ' ' + item['name'] + ". " + str(item['points']) + "\n"
        id += 1

    return(leaderboard)

def update_score(user, points):

    url = 'http://127.0.0.1:8000/api/score/update/'
    new_score = {'name': user, 'points': points}
    x = requests.post(url, data = new_score)

    return
```

Figura 27: Funciones para la obtención y modificación de los puntos de los alumnos.

4.2.4.7 Creación del juego de Kahoot

Para almacenar las preguntas y sus respuestas se deberán crear 2 tablas nuevas en la base de datos. La tabla Question tendrá 3 atributos: el título de la pregunta, el autor y los puntos que otorgará responderla correctamente. Los primeros 2 atributos serán CharFields y el tercero un SmallIntegerField [51], el cual sirve para almacenar valores numéricos más pequeños que in IntegerField.

La tabla Answer tendrá también 3 atributos. El título de la respuesta, si es correcta o no y la pregunta a la que pertenece. El primero será un CharField, el segundo un BooleanField [52], el cual es un campo que solamente puede almacenar 2 valores: verdadero o falso. Finalmente, el tercer parámetro será una clave ajena [53] (ForeignKey) a la tabla Question, permitiendo una relación de una a muchas, lo cual significa que una pregunta podrá tener varias respuestas asociadas. También se indicará en este atributo un modelo de borrado en cascada, permitiendo que al borrar una pregunta so eliminen también de la base de datos todas sus respuestas. La figura 28 muestra la implementación de estas 2 tablas.

```python
from django.db import models
from django.utils.translation import ugettext as _

You, 6 months ago | 1 author (You)
class Question(models.Model):


    title = models.CharField(_("título"),max_length=255)
    points = models.SmallIntegerField(_("puntos"))
    author = models.CharField(_("author"),max_length=255)

    def __str__(self):
        return self.title


You, 6 months ago | 1 author (You)
class Answer(models.Model):

    question = models.ForeignKey(Question, related_name='answer', verbose_name=_("Pregunta"), on_delete=models.CASCADE)
    answer = models.CharField(_("Respuesta"), max_length=255)
    is_correct = models.BooleanField(_("correcta?"), default=False)


    def __str__(self):
        return self.answer
```

Figura 28: Archivo models.py de la tablas Question y Answer.

A continuación, se modificarán los archivos admin.py y serializer.py de forma similar al apartado anterior. En el archivo views.py se crearán 5 clases: 2 de ellas servirán para añadir preguntas y respuestas nuevas a la base de datos, las siguientes 2 servirán para obtener la pregunta más antigua y la más nueva de la base de datos creadas por el usuario del comando y la final servirá para borrar la pregunta más antigua una vez haya sido utilizada en el juego. Finalmente, en el archivo urls.py se creará una url en el servidor para cada una de estas clases.

Para crear preguntas y respuestas nuevas desde Discord, los profesores podrán utilizar el comando !crear seguido de los puntos y el título de la pregunta. Esto comenzará un proceso donde el bot pedirá al usuario que introduzca una respuesta a la pregunta. El bot guardará la respuesta en una variable y procederá a preguntar si esta es correcta o

no, indicando que responda con las letras s o n (sí o no). Una vez respondido, se tendrá toda la información necesaria para crear una pregunta y una respuesta, y procederá a añadir ambas a la base de datos. El bot a continuación indicará al usuario que puede terminar el proceso escribiendo “fin” o puede seguir añadiendo respuestas para la misma pregunta. Hasta que el usuario decida terminar, el bot repetirá el proceso mediante un bucle while, publicando respuestas nuevas con cada iteración. La figura 29 muestra el código del comando mientras que la figura 30 ilustra un ejemplo del proceso de creación dentro de Discord.

```
import json
import asyncio
import datetime
from discord.ext import commands
from discord.ext.commands import Bot
from asyncio import sleep


You, 6 months ago | 1 author (You)
class cog6(commands.Cog):
    def __init__(self, client):
        self.client = client



    @commands.command(name='crear')

    async def crear(self, ctx,puntos,*,titulo):

        role = discord.utils.get(ctx.guild.roles, name="Profesor")

        if role in ctx.author.roles:

            url = 'http://127.0.0.1:8000/api/question/update/'
            url2 = 'http://127.0.0.1:8000/api/answer/update/'
            await ctx.channel.send(f"Introduzca una respuesta")
            fin = False
            def check(m):
                    return True



            answer = await self.client.wait_for(event='message', check=check)
            await ctx.channel.send(f"¿Es la respuesta correcta? (s/n)")
            correct = await self.client.wait_for(event='message', check=check)
            if correct.content.startswith('s'):
                correcto = True
            else:
                correcto = False

            new_question = {'title': titulo, 'points': puntos, 'author':ctx.author.name}
            x = requests.post(url, data = new_question)

            response = requests.get("http://127.0.0.1:8000/api/newest/")
            json_data = json.loads(response.text)
            ident=json_data[0]['question']
            ident=ident+1

            new_answer = {'question':ident,'answer':answer.content,'is_correct':correcto}
            y = requests.post(url2, data = new_answer)
            await ctx.channel.send(f"Introduzca otra respuesta o escriba 'fin' para terminar.")
            answer = await self.client.wait_for(event='message', check=check)
            if answer.content.startswith('fin'):
                fin=True

            while fin==False:

                await ctx.channel.send(f"¿Es la respuesta correcta? (s/n)")
                correct = await self.client.wait_for(event='message', check=check)
                if correct.content.startswith('s'):
                    correcto = True
                else:
                    correcto = False
                new_answer = {'question':ident,'answer':answer.content,'is_correct':correcto}
                y = requests.post(url2, data = new_answer)
                await ctx.channel.send(f"Introduzca otra respuesta o escriba 'fin' para terminar.")
                answer = await self.client.wait_for(event='message', check=check)
                if answer.content.startswith('fin'):
                    fin=True

            await ctx.channel.send(f"Se ha publicado su pregunta.")


        else:
            await ctx.send('No tienes permiso para utilizar este comando.')



def setup(client):
    client.add_cog(cog6(client))
```

Figura 29: Código del comando “crear”.

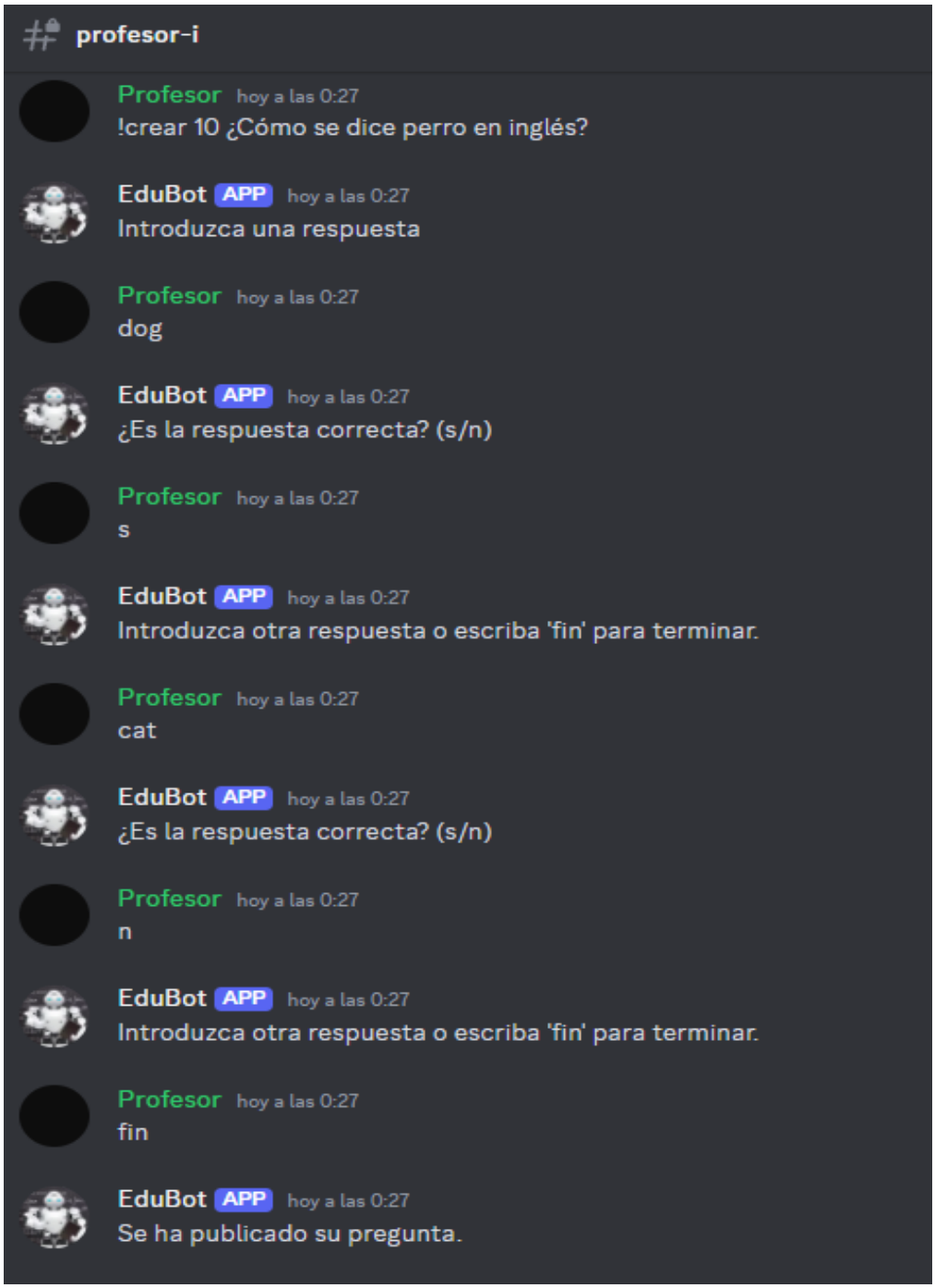

Figura 30: Ejemplo de uso del comando “crear”.

El último comando que se creará será !pregunta, el cual iniciará el juego de Kahoot. El primer paso será importar la función update_score creada en el apartado anterior, para actualizar los puntos de los alumnos en caso de que acierten la pregunta.

A continuación, se creará la función get_question, la cual accederá a una url para obtener la pregunta más antigua de la base de datos que haya sido creada por el autor del comando. La función también obtendrá todas las respuestas asociadas y les dará

un formato organizado de manera que en Discord aparezcan todas en una línea separada para ser fácilmente legibles y tengan un número asociado que el alumno podrá escribir para elegir la respuesta correcta.

La función principal llamará a esta función para permitir que el bot envíe un mensaje que dará comienzo al juego. El bot entonces esperará 10 segundos mientras los alumnos envían sus respuestas. Se comprobará si los mensajes recibidos después de la publicación de la pregunta son números y si dichos números corresponden a respuestas correctas. En caso de que lo sean, el bot invocará la función update_score para añadir los puntos indicados a los usuarios.

El bot creará una lista que guardará los nombres de los alumnos que hayan respondido para asegurar que no puedan contestar otra vez a la misma pregunta. También creará otra lista para guardar los nombres de los alumnos que hayan respondido correctamente para anunciar a los ganadores en un mensaje que enviará al final del juego. Si esta lista está vacía, el bot anunciará que nadie ha acertado la pregunta.

También se premiará a los alumnos que respondan más rápido que otros. Cada vez que alguien responda correctamente, se restará en 1 la cantidad de puntos obtenidos al acertar, pero esta cantidad nunca podrá ser inferior a 1.

Finalmente, el bot accederá a una de las url creadas para borrar la pregunta de la base de datos, asegurando que no pueda volver a aparecer en el juego. La figura 31 muestra el código de la función y la figura 32 un ejemplo de una ronda del juego, con 3 alumnos diferentes como participantes.

```python
class cog7(commands.Cog):
    async def pregunta(self, ctx):
        def get_question(auth):
            qs = ''
            id = 1
            answer = 0
            points = 0
            response = requests.get("http://127.0.0.1:8000/api/oldest/")
            json_data = json.loads(response.text)


            for item in json_data:
                if 'author' in item and item['author']==auth:
                    qs += "\n"
                    qs += item['title']+"\n"

                    for subitem in item['answer']:
                        qs += str(id) + ". " + subitem['answer'] + "\n"

                        if subitem['is_correct']:
                            answer=id

                        id += 1

                    points = item['points']

                    return (qs,answer,points)


        role = discord.utils.get(ctx.guild.roles, name="Profesor")

        if role in ctx.author.roles:

            qs, answer, points = get_question(ctx.author.name)
            question_message = await ctx.channel.send(f"({points} puntos): {qs}")
            url = 'http://127.0.0.1:8000/api/delete/auth/'
            def check(m):
                return m.content.isdigit() == True

            await asyncio.sleep(10)

            messages = await ctx.channel.history(after=question_message.created_at).flatten()
            correct_answers = []
            answered_users = {}

            for message in messages:
                if int(message.content) == answer and  message.author.id not in answered_users:
                    user = message.author
                    correct_answers.append(message.author.nick)
                    if points >1:
                        points = points - 1
                    update_score(user.name, points)

                answered_users[message.author.id] = True

            if correct_answers:
                await ctx.channel.send(f"La pregunta ha sido acertada por: {', '.join(correct_answers)}")
            else:
                await ctx.channel.send("Nadie ha acertado la pregunta")
            answered_users.clear()


            requests.delete(url)


        else:
            await ctx.channel.send('No tienes permiso para utilizar este comando')


def setup(client):
    client.add_cog(cog7(client))
```

Figura 31: Código del comando “pregunta”.

Figura 32: Ejemplo de uso del comando “pregunta”.

5. Conclusiones

5.1 Conclusiones

Una vez terminada la programación y realizadas diferentes pruebas en la aplicación de Discord para comprobar que todo funciona correctamente, se puede afirmar que se han cumplido todos los objetivos planteados al principio del proyecto.

Durante el desarrollo, la mayoría de los problemas que han surgido han sido resultado de fallos de código, donde el programa no realizaba la función deseada o lanzaba un error. Por suerte, Visual Studio Code dispone de sistemas muy eficientes para detectar fallos e indicar al usuario qué parte del código está resultando en un error y la razón. En casos donde esto no era suficiente, ha sido necesario analizar a fondo la respuesta del bot y compararla con el código escrito, para encontrar qué parte del proceso no funciona como debería.

En conclusión, ha sido posible planear y realizar la creación de un bot con una gran variedad de funciones que podrían servir de apoyo para la docencia en línea a través de la aplicación Discord.

5.2 Trabajo futuro

Considerando un posible trabajo futuro, el bot podría expandirse de diferentes formas. La más simple sería añadir nuevas funciones, para lo cual habría que realizar en primer lugar otro análisis del estado del arte para comprender qué beneficios los usuarios buscan en otras aplicaciones y que podrían incorporarse al bot. Otra mejora sería hacer el bot más flexible, ya que actualmente está diseñado para que funcione en un servidor con la estructura específica que se ha creado y sería más atractivo si pudiera incorporarse en cualquier entorno.

5.3 Conocimientos necesarios

En cuanto a las asignaturas de la carrera relevantes para el proyecto, las más importantes han sido Introducción a la Informática y a la Programación y Programación, las cuales fueron realizadas en los primeros 2 cuatrimestres y ayudaron a aprender los fundamentos del desarrollo de código. La asignatura Bases de datos y sistemas de información ayudó a aprender acerca de sistemas y lenguajes de almacenamiento de datos, lo cual ha sido fundamental para el sistema de puntos de los alumnos y el juego de Kahoot. Finalmente, Ingeniería del Software sirvió para poner en práctica todos estos conocimientos y obtener experiencia creando un programa desde cero.

Bibliografía

[1] Página principal de Discord. <Discord - Group Chat That's All Fun & Games> [Consulta: 20 de abril de 2024]

[2] Explorador de servidores de Discord. <Discord Servers - Home> [Consulta: 20 de abril de 2024]

[3] Servidores de Discord relacionados a la Educación <Discord Servers - Education> [Consulta:19 de julio de 2024]

[4] Versiones premium de Discord <Nitro Benefits and Features | Discord> [Consulta: 20 de abril de 2024]

[5] Buzzacco, O.C. *The Potential for Discord in the Higher Education Classroom* (2024) <(PDF) Examining the Benefits and Challenges of Using Discord in Online Higher Education Classrooms (researchgate.net)> [Consulta:19 de julio de 2024]

[6] Akmar, R., Mawardi, M., Ulianas, A. et al. *Effectiveness of Discord Instructional Media Integrated with Flipped Classroom and Guided Inquiry Learning on Reaction Rates on Students Learning Outcomes* (2024) <(PDF) Effectiveness of Discord Instructional Media Integrated with Flipped Classroom and Guided Inquiry Learning on Reaction Rates on Students Learning Outcomes (researchgate.net)> [Consulta:19 de julio de 2024]

[7] Lauricella, S., Holding, R., Craig, C. D. *Examining the Benefits and Challenges of Using Discord in Online Higher Education Classrooms* (2024) <(PDF) Examining the Benefits and Challenges of Using Discord in Online Higher Education Classrooms (researchgate.net)> [Consulta:19 de julio de 2024]]

[8] Tenney School. *The Pros and Cons of Using Discord in the Classroom (2024).* <The Pros and Cons of Using Discord in the Classroom - Tenney School> [Consulta:19 de julio de 2024]

[9] Portal de desarrollo de aplicaciones de Discord <Discord Developer Portal — Documentation — Intro> [Consulta: 20 de abril de 2024]

[10] Página principal de Microsoft Teams <Sign In | Microsoft Teams> [Consulta: 3 de mayo de 2024]

[11] Curry, D. *Microsoft Teams Revenue and Usage Statistics (2024).* <Microsoft Teams Revenue and Usage Statistics (2024) - Business of Apps> [Consulta: 3 de mayo de 2024]

[12] Página principal de Zoom. <Una plataforma para conectarse | Zoom> [Consulta: 3 de Septiembre de 2024]

[13] Watts, R., Novak, J. *Microsoft Teams Vs. Zoom (2024 Comparison)* (2024) <Microsoft Teams vs. Zoom – Forbes Advisor> [Consulta: 30 de agosto de 2024]

[14] Alejandra, A. *Microsoft Teams: Guía de iniciación y tutorial* (2020) <Microsoft Teams: Guía de iniciación y tutorial 【 PASO A PASO 】 ☆ (profesionalreview.com)> [Consulta: 30 de agosto de 2024]

[15] Página principal de Kahoot! <Kahoot! | Learning games | Make learning awesome!> [Consulta: 7 de mayo de 2024]

[16] Instrucciones de juego de Kahoot! <Kahoot! for schools: how it works | Feature overview> [Consulta: 7 de mayo de 2024]

[17] Licorish, S.A., Owen, H.E., Daniel, B. et al. *Students' perception of Kahoot!'s influence on teaching and learning*. RPTEL 13, 9 (2018). <Students' perception of Kahoot!'s influence on teaching and learning | Research and Practice in Technology Enhanced Learning | Full Text (springeropen.com)> [Consulta: 7 de mayo de 2024]

[18] G2. *Top 10 Kahoot! Alternatives & Competitors* (2024). <Top 10 Kahoot! Alternatives & Competitors in 2024 | G2> [Consulta: 30 de agosto de 2024]

[19] Rosman, S. *Online Learning Platforms: Analyzing Pros & Cons* (2022). <Online Learning Platforms: Analyzing Pros & Cons (spatial.chat)> [Consulta: 30 de agosto de 2024]

[20] Página principal de Poliformat <PoliformaT : Castellano : Inicio (upv.es)> [Consulta: 20 de mayo de 2024]

[21] Página principal de Sakai Project <Sakai Learning Management System | Sakai LMS> [Consulta: 22 de julio de 2024]

[22] Página principal de .LRN <.LRN Home (dotlrn.org)> [Consulta: 22 de julio de 2024]

[23] Página principal de Moodle <Página Principal | Moodle.org> [Consulta: 22 de julio de 2024]

[24] I. Castro, M.J. Castro-Bleda, P. Aibar (2013) *LEARNING MANAGEMENT SYSTEMS FOR NEW PRACTICES IN TEACHING, EDULEARN13 Proceedings, pp. 2982-2985.*

[25] Bouchrika, I. *List of Learning Management Systems for Schools and Universities in 2024* (2024) <List of Learning Management Systems for Schools and Universities in 2024 | Research.com> [Consulta: 30 de agosto de 2024]

[26] Chron. *Importance of Organization for Students* (2021). < Importance of Organization for Students (chron.com)> [Consulta: 22 de mayo de 2024]

[27] Página principal de Google Calendar <Google Calendar - Semana del 7 de julio de 2024> [Consulta: 22 de mayo de 2024]

[28] reclaimai. *Top 5 Free Apps for Students: Time Management + Productivity* (2022) <Top 5 Free Apps for Students: Time Management + Productivity | Reclaim> [Consulta: 30 de agosto de 2024]

[29] Página principal de Visual Studio Code <Visual Studio Code - Code Editing. Redefined> [Consulta: 3 de junio de 2024]

[30] Extensiones de Visual Studio Code <Managing Extensions in Visual Studio Code> [Consulta: 3 de junio de 2024]

[31] Página principal de Python <Welcome to Python.org> [Consulta: 20 de junio de 2024]

[32] Librería de Discord para Python <Welcome to discord.py (discordpy.readthedocs.io)> [Consulta: 20 de junio de 2024]

[33] Página principal de Git <Git (git-scm.com)> [Consulta: 27 de junio de 2024]

[34] Página principal de GitHub <GitHub: Let's build from here · GitHub> [Consulta: 27 de junio de 2024]

[35] Wrov. *Git, GitHub, y Visual FoxPro (4)* (2023) <Git, GitHub, y Visual FoxPro (4) | Visual FoxPro. Técnicas avanzadas (wordpress.com)> [Consulta: 30 de agosto de 2024]

[36] Página principal de Django <The web framework for perfectionists with deadlines | Django (djangoproject.com)> [Consulta: 27 de junio de 2024]

[37] Página principal de Django Rest Framework <Home - Django REST framework (django-rest-framework.org)> [Consulta: 27 de junio de 2024]

[38] Condori, J. *Phython - DjangoFramework de desarrollo web para perfeccionistasBasado en el Modelo MTV* (2012) <Revista de Información, Tecnología y Sociedad - Phython - DjangoFramework de desarrollo web para perfeccionistasBasado en el Modelo MTV (umsa.bo)> [Consulta: 30 de agosto de 2024]

[39] Página principal de SQLite < SQLite Home Page> [Consulta: 30 de junio de 2024]

[40] Aulapc. *CONCEPTOS SOBRE TABLAS RELACIONALES:* <Tablas relacionales (aulapc.es)> [Consulta: 30 de agosto de 2024]

[41] Portal de desarrolladores de Discord <Discord Developer Portal — My Applications> [Consulta:19 de julio de 2024]

[42] Documentación de Cogs de discord.py <Cogs (discordpy.readthedocs.io)> [Consulta:19 de julio de 2024]

[43] Documentación de models.py <Models | Django documentation | Django (djangoproject.com)> [Consulta: 2 de agosto de 2024]

[44] Documentación de CharFields <CharField - Django Models - GeeksforGeeks> [Consulta: 2 de agosto de 2024]

[45] Documentación de IntegerFields <IntegerField - Django Models - GeeksforGeeks> [Consulta: 2 de agosto de 2024]

[46] Documentación de manage.py <django-admin and manage.py | Django documentation | Django (djangoproject.com)> [Consulta: 2 de agosto de 2024]

[47] Documentación de admin.py <The Django admin site | Django documentation | Django (djangoproject.com)> [Consulta: 2 de agosto de 2024]

[48] Documentación de views.py <Writing views | Django documentation | Django (djangoproject.com)> [Consulta: 3 de agosto de 2024]

[49] Documentación de serializadores en Django <1 - Serialization - Django REST framework (django-rest-framework.org)> [Consulta: 3 de agosto de 2024]

[50] Documentación de urls.py <URL dispatcher | Django documentation | Django (djangoproject.com)> [Consulta: 3 de agosto de 2024]

[51] Documentación de SmallIntegerFields <SmallIntegerField - Django Models - GeeksforGeeks> [Consulta: 5 de agosto de 2024]

[52] Documentación de BooleanFields <BooleanField - Django Models - GeeksforGeeks> [Consulta: 5 de agosto de 2024]

[53] Documentación de ForeignKey <Many-to-one relationships | Django documentation | Django (djangoproject.com)> [Consulta: 5 de agosto de 2024]

Anexo

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No Procede**
ODS 1. **Fin de la pobreza.**				**X**
ODS 2. **Hambre cero.**				**X**
ODS 3. **Salud y bienestar.**	**X**			
ODS 4. **Educación de calidad.**	**X**			
ODS 5. **Igualdad de género.**		**X**		
ODS 6. **Agua limpia y saneamiento.**				**X**
ODS 7. **Energía asequible y no contaminante.**				**X**
ODS 8. **Trabajo decente y crecimiento económico.**				**X**
ODS 9. **Industria, innovación e infraestructuras.**				**X**
ODS 10. **Reducción de las desigualdades**.	**X**			
ODS 11. **Ciudades y comunidades sostenibles**.				**X**
ODS 12. **Producción y consumo responsables.**				**X**
ODS 13. **Acción por el clima.**				**X**
ODS 14. **Vida submarina.**				**X**
ODS 15. **Vida de ecosistemas terrestres.**				**X**
ODS 16. **Paz, justicia e instituciones sólidas.**				**X**
ODS 17. **Alianzas para lograr objetivos.**	**X**			

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

-ODS 3: Salud y bienestar: Los estudiantes tienen que lidiar con mucho estrés en sus vidas diarias debido a la gran carga de trabajo que tienen y dificultad para organizar todas sus tareas. El bot puede ayudar a mejorar su experiencia académica, lo cual conllevaría menor estrés y menos problemas de salud mental para sus usuarios.

-ODS 4: Educación de calidad: El bot puede facilitar el acceso a la educación al proporcionar un punto de entrada fácil y familiar para los estudiantes, especialmente en situaciones donde la asistencia presencial es difícil o imposible. La facilidad de acceso y organización que los estudiantes puedan aprender en un ambiente más cómodo.

-ODS 5: Igualdad de género: El bot proporciona funciones que pueden ser utilizadas de forma equitativa sin importar el género del usuario. Esto garantiza que cualquier persona pueda sentirse bienvenida y participar en clase sin sentirse discriminada.

-ODS 10: Reducción de las desigualdades: Las plataformas en línea permiten que cualquier persona pueda participar en las clases sin importar su origen o capacidades. El único requisito es acceso a un ordenador y todos los usuarios serán tratados por igual.

-ODS 17: Alianzas para lograr objetivos: Los alumnos pueden utilizar las plataformas en línea para reunirse tanto dentro como fuera de las horas lectivas. Es importante facilitar espacios así para crear un sentimiento de comunidad en los usuarios y darles acceso a diferentes herramientas que permitan que puedan colaborar y aprender juntos.