Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

Depósito y almacenamiento de archivos en Telegram

Autor:
Mario Benito Rodríguez
Tutor:
César Llamas Bello

RESUMEN

El propósito de este proyecto es desarrollar una aplicación de depósito y almacenamiento de archivos para diferentes grupos de usuarios mediante la API de la plataforma de Telegram. El objetivo principal es el mantenimiento de los archivos en Telegram mediante la clasificación y etiquetación de los mismos ordenados en sistemas de ficheros con directorios.

El programa se denominará *FileBot* y vendrá identificado como una aplicación interactiva en Telegram con el siguiente nombre: @FileDataBot. Se trata de un *bot* que gestiona sistemas de ficheros, permitiendo así la creación de carpetas, directorios, archivos de cualquier tipo y formato a los usuarios de las diferentes conversaciones en Telegram.

El proyecto ha sido desarrollado empleando la API de programación de Telegram de Python, junto con las herramientas de desarrollo que se proporcionan en el propio lenguaje de programación como tal. Además, se han empleado un conjunto de API's necesarias para la subida de archivos a plataformas externas como Bitbucket y Github mediante mecanismos de almacenamiento remotos.

ABSTRACT

The purpose of this project is to develop an application about saving and storing files in different groups of users using the Telegram Bot API. The main goal is the storage of files and data in the Telegram platform by sorting out them in file systems with directories.

The program is called *FileBot* and it is identified as an interactive application in Telegram with the following name: @FileDataBot. This is a bot that manage file systems, allowing users of Telegram chats the creation of folders, data files with several formats and types.

The project has been developed using the API framework of Telegram in Python, with other option tools that this programming language offers. Moreover, this application has used API's for uploading file data in other extern platforms like Bitbucket or Github, setting the possibility of creating remote repositories for the Telegram groups.

ÍNDICE DE CONTENIDO

ÍNDICE DE ILUSTRACIONES

PARTE PRIMERA - INTRODUCCIÓN Y CONTEXTO

CAPÍTULO 1. INTRODUCCIÓN Y OBJETIVOS.

1.1 Introducción

Las aplicaciones de mensajería instantánea en Internet han conseguido que multitud de usuarios se comuniquen en tiempo real, sin necesidad de esperas innecesarias que se producían en el envío de los mensajes de correo electrónico y demás aplicaciones de mensajería.

Esto ha provocado una revolución muy grande en las plataformas de comunicación de usuarios, tanto es así que muchas de las aplicaciones que surgen para dispositivos móviles han evolucionado e incorporado esta tecnología instantánea de gestión de mensajes. Aplicaciones como Telegram o Whatsapp se utilizan prácticamente en todos los dispositivos móviles o en versiones de Escritorio por lo rápido que es comunicarse con los usuarios sin necesidad de realizar llamadas o esperas de conexión.

Telegram ofrece una interfaz de programación de aplicaciones, también conocidos como “bots”, que permiten garantizar ciertos servicios y funcionalidades a los grupos o usuarios de la plataforma en cualquier momento, esto nos lleva a pensar en multitud de ideas que se pueden implementar como aplicaciones dentro de la propia plataforma de comunicación y surge una explosión de combinaciones posibles.

1.2 Motivación

Este proyecto surge precisamente debido a la inexistencia de clasificaciones de archivos en Telegram, puesto que las últimas actualizaciones de la aplicación permitían acceder de manera simple a los archivos que se habían enviado en las conversaciones de usuarios, ya sean públicas y privadas, pero no existía una etiquetación determinada ni permitían realizar modificaciones de esos archivos como lo haría un sistema de ficheros real que controla un sistema operativo.

La compartición de archivos siempre ha sido un tema importante para grupos de usuarios que comparten un trabajo común, en el que tienen que manejar archivos de distintos tipos establecidos en jerarquías de directorios, por ello mediante la API de programación se puede implementar una aplicación que permita la administración de sistemas de ficheros para los usuarios junto con la posibilidad de establecer sincronizaciones remotas con otras aplicaciones externas que proporcionen los mecanismos de almacenamiento necesarios.

Esta idea fue inicialmente presentada por el profesor César Llamas Bello de la Universidad de la Escuela Técnica de Ingeniería Informática como una oferta de proyecto con la intención de expandirla en lo máximo posible con mejoras durante su desarrollo en todo momento, acepté su oferta y él se comprometió a tutorizar el TFG.

1.3 Objetivos

El objetivo principal de este Trabajo de Fin de Grado es la fabricación de una aplicación interactiva (Bot de Telegram) para el depósito y almacenamiento de archivos de manera persistente con la posibilidad de establecer sincronizaciones de repositorios en diferentes plataformas en cuestión de los gustos de los usuarios del grupo o del canal correspondiente de Telegram.

Para ello nos basaríamos en la forma de organización de los sistemas de ficheros del sistema operativo GNU/Linux con las tablas de inodos representando a los archivos con su contenido en concreto, dando así una clasificación en tablas de archivos para los grupos de usuarios en la plataforma de

Telegram. Será necesario además determinar una implementación eficiente del modelo de dominio adecuada y un servidor de alojamiento con suficiente espacio de almacenamiento para albergar los sistemas de ficheros de grupos y usuarios concretos que interactúen con nuestro programa.

El administrador tendrá funcionalidades especiales para establecer toda la configuración del bot de almacenamiento de archivos, como la elección de plataformas externas implicadas, la activación de botones compartidos para la navegación del sistema de ficheros u otras opciones de ajustes de grupo disponibles a incorporar que sean consideradas interesantes durante el proceso de desarrollo software del proyecto. Sin embargo, cualquier miembro del grupo podrá comunicarse con el bot mediante comandos para crear, borrar o modificar carpetas o ficheros del propio sistema de archivos compartido, puesto que se trata de repositorios compartidos elaborados para cada grupo.

La idea es impregnar la compartición de archivos en cada uno de los chats de Telegram donde los usuarios necesiten clasificar y etiquetar los archivos en un único sistema de ficheros asociado. Esto se podría implementar tanto en conversaciones privadas como grupales (con multitud de usuarios en la conversación), puesto que las aplicaciones existentes de alojamiento de archivos en la nube (Dropbox, Google Drive…) nos permiten crear espacios de archivos individuales privados o compartidos para múltiples usuarios invitados.

Por tanto, como objetivos generales se consideran las siguientes pautas a seguir:

- Planificación de un proyecto software mediante un Proceso de Desarrollo Software Unificado (Ingeniería del Software).
- Desarrollo de esquemas, modelos y pautas para el diseño de la aplicación.
- Implementación de la aplicación empleando Python como lenguaje de programación con un conjunto de librerías y API's para el mismo que nombraremos a lo largo de esta memoria.

1.4 Organización de la memoria

Esta memoria se organiza en cuatro partes principales, una primera parte introductoria para establecer el contexto y el marco de trabajo, una segunda parte donde se exponen los detalles del diseño del proyecto y su planificación, una tercera parte para la administración y organización de la aplicación y finalmente una última parte donde se extraen conclusiones y posibles mejoras a integrar en el proyecto:

- **Marco conceptual:** Se describen brevemente los conceptos, herramientas y tecnologías necesarias para planificar y desarrollar el proyecto.
- **Proyecto software:** Se detalla la planificación, el modelado de desarrollo, fases de análisis y diseño del proyecto.
- **Mantenimiento y organización:** Se informa de los servidores posibles para el mantenimiento de la aplicación y las instrucciones necesarias para ejecutar la aplicación y ejercer los servicios de la misma.
- **Conclusiones y mejoras futuras:** Se extraen conclusiones sobre el proyecto y se plantea una línea de posibles mejoras futuras para la aplicación.
- **Bibliografía:** Referencias bibliográficas consultadas durante el proyecto.
- **Anexos:** Capturas de la aplicación.

CAPÍTULO 2. MARCO CONCEPTUAL.

2.1. Telegram Messenger

Telegram Messenger es un servicio de mensajería por Internet desarrollado desde el año 2013 por los hermanos Nikolai y Pavel Durov, basándose en la gestión de mensajes de texto y contenido multimedia entre los usuarios. Inicialmente fue empleado para teléfonos móviles, pero en el año siguiente de su lanzamiento se diseñó para multiplataforma.

El servicio de Telegram es administrado por una organización sin ánimo de lucro cuya sede opera en Berlín. Una de las ventajas de Telegram es que no ha sido creado para usos comerciales, por lo tanto, garantiza que siempre será gratuita. Esto influye en la competitividad de las aplicaciones de mensajería haciendo que modifiquen los precios o incluso permanezcan como gratuitas (el caso de Whatsapp a partir de 2016 del retiro del saldo anual).

La plataforma de Telegram utiliza el protocolo MTProto (Protocolo de Transporte Móvil), que soporta documentos, datos multimedia (animaciones gráficas, gifs) o archivos de alta duración. Otras características principales de este servicio de mensajería son la posibilidad de establecer conversaciones con el propio usuario, mantener la persistencia de la información de los chats, búsqueda de contactos, llamadas, canales de difusión, supergrupos y el establecimiento de alias/usernames.

Quizá lo más atractivo de Telegram sea la introducción de bots en las conversaciones de grupos y usuarios de la aplicación, estos bots o programas informáticos interactivos permiten incorporar funcionalidades muy diversas dentro de la plataforma de mensajería, como por ejemplo proporcionar fotografías, vídeos o cualquier tipo de contenido de Telegram por medio de comandos de manera muy fácil y sencilla. Para ello, Telegram ofrece una API y una plataforma de bots que puede utilizar cualquier usuario para fabricar su propio programa inteligente y realizar multitud de servicios como los mencionados anteriormente.

Telegram ofrece chats con cifrado end-to-end, es decir, los mensajes nunca pasan por los servidores de Telegram, sino que se cifran y descifran en los dispositivos de cada usuario. Además de ofrecer este alto nivel de protección, puedes iniciar chats secretos que se autodestruyen, lo que quiere decir que podrás establecer un límite de tiempo para que la otra persona pueda leer los mensajes antes de que estos desaparezcan para siempre de ambos dispositivos.

Las aplicaciones están disponibles para más de 10 sistemas operativos: Android, iOS, macOS, Windows, GNU/Linux, Firefox OS, navegadores web, entre otros. La API permite la creación de clientes externos, algunos de ellos bajo software libre —excepto el lado del servidor— que pueden ser modificados por la comunidad. De acuerdo con el lema de proveer mayor privacidad y seguridad, las aplicaciones sufrieron cambios de diseño y facilidad de manejo.

Fue traducido al español para los móviles a principios de febrero de 2014 y para computadoras a finales del mismo año. A partir de ese año, Telegram aumentó sorpresivamente su cuota de uso en España y otros países, llegando a ser uno de los competidores más influyentes de la mensajería instantánea, compitiendo así con el servicio de mensajería más popular conocido por todo el mundo: Whatsapp.

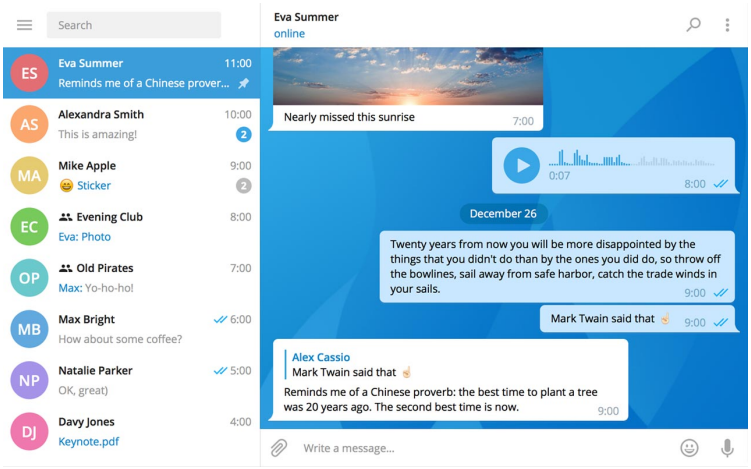


ILUSTRACIÓN 1 – TELEGRAM DE ESCRITORIO PARA ORDENADOR

Telegram tiene las funcionalidades básicas de todas las redes sociales y servicios de mensajería: conversaciones grupales, individuales o privadas, encriptación, ajustes de usuario y preferencias (notificaciones, sonidos, privacidad, datos, almacenamiento, fondos de pantalla de aplicación…) y los diferentes contactos con los que puedes comunicarte.

Después tiene otra serie de funcionalidades peculiares a mayores, como por ejemplo la capacidad de anclar mensajes en una conversación grupal que sea un supergrupo y solamente lo pueden hacer los administradores del propio supergrupo. El concepto de supergrupo en Telegram es nuevo con respecto a otros servicios de mensajería, puesto que un supergrupo puede albergar hasta 5000 miembros en una sola conversación y permitir funcionalidades a mayores como el anclaje de mensajes. Anclar mensajes suele ser útil para cuando quieres mantener un mensaje importante en la conversación independientemente de la cantidad de mensajes que expongan los miembros del grupo.

Además, un grupo no tiene por qué tener un límite de miembros para poder ser un supergrupo, el traspaso de uno a otro se puede realizar en cualquier conversación grupal, esto se debe a que básicamente un supergrupo es un grupo con adiciones y con pequeñas funcionalidades agregadas o simplemente con una mayor capacidad de usuarios que pueden pertenecer al mismo.

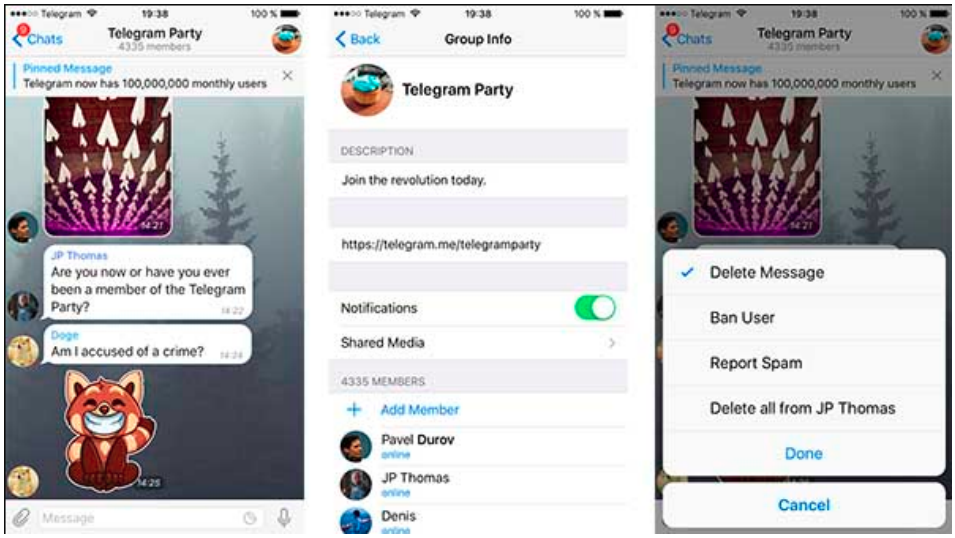


ILUSTRACIÓN 2 – TELEGRAM DE DISPOSITIVO MÓVIL

Otra funcionalidad curiosa es la posibilidad de editar o incluso borrar los mensajes propios de usuario para todo el grupo, normalmente en este tipo de aplicaciones de mensajería instantánea solo se permite la eliminación de mensajes para el propio usuario y no para el resto, pero Telegram otorga la capacidad de borrar los mensajes que hayas enviado independientemente del tipo de conversación (privada o grupal) o de si eres administrador o no. En cuanto a la edición en tiempo real de mensajes propios tampoco es necesario ser administrador del chat correspondiente para elaborar la modificación deseada por el usuario.

Los administradores ejercen un papel mucho más importante que el de añadir o eliminar participantes como en la aplicación Whatsapp Messenger, en Telegram pueden eliminar todo tipo de mensajes inapropiados para el grupo o conversación, tanto mensajes propios como de cualquier usuario, reportar spam de un usuario determinado en la plataforma, anclar mensajes en el caso de que se necesiten, entre otras muchas opciones de chats disponibles.

Una característica diferente de Telegram con respecto a Whatsapp y otras aplicaciones del estilo similar, es que cualquier miembro del grupo puede agregar otras personas al grupo sin necesidad de ser administrador. Sin embargo, solamente un administrador y no un usuario cualquiera puede eliminar miembros de los grupos, debido a los permisos especiales que hemos especificado antes.

Por último, la funcionalidad más destacada y usada de Telegram que otras plataformas no disponen, es el almacenamiento interno de stickers o gifs que se pueden guardar en la cuenta de usuario de la aplicación. Este sistema permite identificar contenido dentro de Telegram (gifs, stickers…) sin necesidad de realizar búsquedas fuera de la plataforma de servicio (ya sea en la galería del dispositivo móvil o bien mediante la búsqueda en Internet). Además, el propio sistema de almacenamiento interno de la aplicación permite establecer nuevos contenidos que han sido proporcionados por otros usuarios de la plataforma.

2.2. Telegram Bot API

La API (Interfaz de Programación de Aplicaciones) de Bots de Telegram es el mecanismo de creación de programas interactivos de esta aplicación de mensajería instantánea y sirve para la programación de bots que permitan ejercer diferentes servicios dependiendo de los objetivos de diseño que se proponen para la aplicación a desarrollar.

Los bots son usuarios artificiales controlados y automatizados por botones, comandos o secuencias. Destacan por su libertad en el desarrollo, mayoritariamente por terceros, para realizar servicios y funcionalidades de distintos tipos. Debido a la existencia de esta API, los medios catalogaron a Telegram como pionero en la masificación de bots conversacionales.

Para ejercer el derecho de un bot de programación, se necesita contactar con un bot principal llamado @BotFather desde la propia plataforma de Telegram. Este bot está diseñado para ejercer el servicio de creación de bots para los distintos programadores que desean programar aplicaciones artificiales y pretenden desarrollar funcionalidades para los usuarios de Telegram mediante estos programas interactivos.

ILUSTRACIÓN 3 – BOTFATHER

BotFather guarda y almacena los tokens, alias y permisos de todos los bots ejercidos a los miembros de la aplicación con los que ha contactado en la creación de los mismos. Los tokens son claves necesarias para activar a los bots en programas de aplicación que desarrollen los dueños de los mismos y comprobar que no existen varios procesos a la vez utilizando el mismo token de bot ejecutándose al mismo tiempo.

La primera versión de Telegram Bot API se incorporó en la plataforma en la versión 3.0 de Telegram para Android y clientes afines en Junio de 2015. Conforme se iba actualizando la API, se fueron incluyendo otras funcionalidades como botones de interfaz para activar callbacks, la creación de encuestas y su uso en chats grupales, desarrollo de juegos para Telegram en HTML5, incorporación de teclados de distintos tipos para la interacción comunicativa y otras muchas opciones de plataforma.

Las actualizaciones más recientes permiten realizar bots con una gran cantidad de funcionalidades y servicios de usuarios, además junto con las herramientas del kit de desarrollador que permiten los lenguajes de programación, se pueden subir archivos a distintas aplicaciones externas (Google Drive, Dropbox, Bitbucket, Github…), descargar archivos y carpetas de cualquier página web de internet o leer los contenidos de ficheros de otros computadores situados en diferentes puntos del planeta.

Los lenguajes de programación nos permiten desarrollar aplicaciones de diferentes tipos y definir el alcance de nuestro programa interactivo. La cantidad de API's existentes de estos lenguajes adaptadas y la combinación de Telegram Bot API producen una explosión de combinaciones de servicios para los usuarios. Lenguajes como Java, Python ó Php han sido adaptados para la API de Telegram por la comunidad y se han implementado sus propias librerías de programación para ser usadas de manera sencilla para el conjunto total de operaciones de la misma.

Una vez que tenemos nuestro bot operativo con una clave o token proporcionado por BotFather, podemos comenzar a ejecutar peticiones HTTP sobre un navegador web permitiendo realizar las diferentes funcionalidades de la API con nuestro programa interactivo en la plataforma de Telegram. Un ejemplo de petición HTTP es la siguiente:

```
https://api.telegram.org/bot123456:ABC-DEF1234ghIkl-zyx57W2v1u123ew11/getMe
```

ILUSTRACIÓN 4 – PETICIÓN HTTP DE OPERACIÓN DETERMINADA

En este ejemplo, el token de autorización del bot es: 123456:ABC-DEF1234ghIkl-zyx57W2v1u123ew11 y además en la petición HTTP se está realizando la función o operación getMe de la API oficial para obtener el objeto que representa al bot del usuario de Telegram.

Mediante peticiones HTTP se permite realizar llamadas a cada una de las operaciones y métodos definidos en la API oficial de Telegram, todas estas operaciones se encuentran documentadas y especificadas con sus parámetros adecuados en la página web oficial de Telegram dentro del apartado de la API de desarrollo de bots y programas interactivos.

El funcionamiento básico de la API es el siguiente: cualquier usuario que disponga de un token de autorización de un bot asociado puede ejecutar una petición HTTP correspondiente a una operación o método de la API, esta petición HTTP será procesada por los servidores de Telegram y el programa interactivo al token adecuado elaborará las acciones que sean necesarias para satisfacer a la llamada de la API que ha realizado el usuario.

La comunicación del bot interactivo con los usuarios de la conversación o chat adecuado de Telegram mediante el envío de mensajes y transferencia de archivos es realizada por parte de la distribución correspondiente en los servidores de Telegram que contienen la implementación de las opciones de funcionalidad de la API que vamos a explicar a continuación.

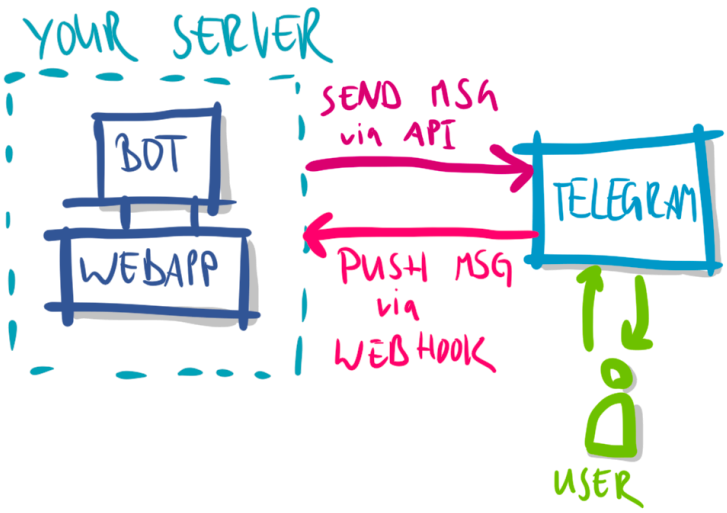


ILUSTRACIÓN 5 – FUNCIONAMIENTO INTERNO DE TELEGRAM BOT API

Existen distintas formas de pasar parámetros en una petición HTTP y Telegram permite tanto peticiones POST como GET y todas las maneras de proporcionar parámetros se llevan a cabo de la misma manera que en cualquier petición web normal. Además, la respuesta de cada una de las peticiones HTTP que se realicen al servidor de Telegram contiene un objeto JSON con diferentes campos: 'ok' para determinar si se ha ejecutado la petición correctamente, 'description' para una descripción del resultado de manera opcional y 'result' con el contenido de la funcionalidad del método u operación correspondiente.

Hay varias formas de obtener las actualizaciones que vayan surgiendo en un bot de Telegram, se puede emplear el método getUpdates o bien se utiliza setWebhook para especificar una petición HTTP cada vez que se genere una actualización. Nuestra aplicación será mucho mas sofisticada y no necesitará el trackeo de actualizaciones ni estudio de las mismas, ya que utilizaremos funciones programadas para determinados eventos producidos por los usuarios en Telegram.

getUpdates

Use this method to receive incoming updates using long polling (wiki). An Array of Update objects is returned.

Parameters	Type	Required	Description
offset	Integer	Optional	Identifier of the first update to be returned. Must be greater by one than the highest among the identifiers of previously received updates. By default, updates starting with the earliest unconfirmed update are returned. An update is considered confirmed as soon as getUpdates is called with an *offset* higher than its *update_id*. The negative offset can be specified to retrieve updates starting from *-offset* update from the end of the updates queue. All previous updates will forgotten.
limit	Integer	Optional	Limits the number of updates to be retrieved. Values between 1—100 are accepted. Defaults to 100.
timeout	Integer	Optional	Timeout in seconds for long polling. Defaults to 0, i.e. usual short polling. Should be positive, short polling should be used for testing purposes only.
allowed_updates	Array of String	Optional	List the types of updates you want your bot to receive. For example, specify [“message”, “edited_channel_post”, “callback_query”] to only receive updates of these types. See Update for a complete list of available update types. Specify an empty list to receive all updates regardless of type (default). If not specified, the previous setting will be used. Please note that this parameter doesn't affect updates created before the call to the getUpdates, so unwanted updates may be received for a short period of time.

setWebhook

Use this method to specify a url and receive incoming updates via an outgoing webhook. Whenever there is an update for the bot, we will send an HTTPS POST request to the specified url, containing a JSON-serialized Update. In case of an unsuccessful request, we will give up after a reasonable amount of attempts. Returns true.

If you'd like to make sure that the Webhook request comes from Telegram, we recommend using a secret path in the URL, e.g. `https://www.example.com/<token>`. Since nobody else knows your bot's token, you can be pretty sure it's us.

Parameters	Type	Required	Description
url	String	Yes	HTTPS url to send updates to. Use an empty string to remove webhook integration
certificate	InputFile	Optional	Upload your public key certificate so that the root certificate in use can be checked. See our self-signed guide for details.
max_connections	Integer	Optional	Maximum allowed number of simultaneous HTTPS connections to the webhook for update delivery, 1-100. Defaults to *40*. Use lower values to limit the load on your bot's server, and higher values to increase your bot's throughput.
allowed_updates	Array of String	Optional	List the types of updates you want your bot to receive. For example, specify [“message”, “edited_channel_post”, “callback_query”] to only receive updates of these types. See Update for a complete list of available update types. Specify an empty list to receive all updates regardless of type (default). If not specified, the previous setting will be used. Please note that this parameter doesn't affect updates created before the call to the setWebhook, so unwanted updates may be received for a short period of time.

ILUSTRACIÓN 6 – OPERACIONES DE OBTENCIÓN DE ACTUALIZACIONES

En ambas operaciones de la API se consiguen objetos del tipo “Update”, este tipo de objetos encapsulan todos los datos e información relacionada con las actualizaciones producidas en Telegram por parte del bot en cuestión. En el caso de getUpdates se proporciona un array de “Updates” que se hayan realizado hasta ese momento de manera directa, por el contrario, con webhooks se irán paulatinamente consiguiendo updates de manera lenta y progresiva hasta cancelar el proceso de espera de actualizaciones de esta operación.

Existen multitud de operaciones y métodos en Telegram Bot API, también se exponen los diferentes tipos y clases propias que se emplean como parámetros de entrada y salida en las mismas operaciones de la API. Como se ha dicho anteriormente, Telegram ha definido una interfaz de operaciones para los bots en la plataforma, determinando así un estándar de desarrollo para los diferentes usuarios que quieran implementar estas operaciones mediante los diferentes lenguajes de programación.

2.2.1. Clases de Telegram Bot API

Comenzaremos con la explicación de los tipos y clases más importantes de la API que vamos a emplear en nuestro proyecto software de depósito y almacenamiento de archivos.

User

This object represents a Telegram user or bot.

Field	Type	Description
id	Integer	Unique identifier for this user or bot
first_name	String	User's or bot's first name
last_name	String	*Optional.* User's or bot's last name
username	String	*Optional.* User's or bot's username
language_code	String	*Optional.* IETF language tag of the user's language

ILUSTRACIÓN 7 – TIPO DE USUARIO

Los usuarios en Telegram tienen los siguientes atributos: un identificador de usuario que es diferente para cada usuario y permite identificar a cada usuario de Telegram con un número entero distinto, el nombre de usuario (first_name), apellidos del usuario (last_name) y el alias o también denominado nick de identificación de usuario (username), esta última propiedad se trata de un nombre único que permite identificar al usuario de manera exclusiva, permitiendo así las búsquedas de usuarios en Telegram sin necesidad de números de contacto o agregaciones de contactos al móvil antes de comunicarse con los usuarios como tal.

El último atributo de la clase usuario en la API que no hemos explicado (language_code) se asocia con el código del lenguaje de etiquetado de Internet asignado al usuario en concreto, de esto no nos tenemos que preocupar, puesto que para nuestra aplicación no se va a utilizar de ninguna forma, sin embargo, está bien conocer todos los atributos y características de las instancias con las que vamos a trabajar en este proyecto software.

Hay que destacar que el username o nickname privado de identificación del usuario es una propiedad opcional de la plataforma y por tanto no todos los usuarios van a disponer de alias en Telegram, de manera que tendremos que tener en cuenta todas estas opcionalidades que no son obligatorias en la propia plataforma para el diseño de nuestra aplicación en sus funcionalidades. Cabe destacar que a pesar de ser una propiedad opcional es principalmente exclusiva de esta aplicación de comunicación de usuarios, puesto que permite realizar búsquedas rápidamente como si de un navegador se tratase.

Chat

This object represents a chat.

Field	Type	Description
id	Integer	Unique identifier for this chat. This number may be greater than 32 bits and some programming languages may have difficulty/silent defects in interpreting it. But it is smaller than 52 bits, so a signed 64 bit integer or double-precision float type are safe for storing this identifier.
type	String	Type of chat, can be either “private”, “group”, “supergroup” or “channel”
title	String	*Optional*. Title, for supergroups, channels and group chats
username	String	*Optional*. Username, for private chats, supergroups and channels if available
first_name	String	*Optional*. First name of the other party in a private chat
last_name	String	*Optional*. Last name of the other party in a private chat
all_members_are_administrators	Boolean	*Optional*. True if a group has ‘All Members Are Admins’ enabled.
photo	ChatPhoto	*Optional*. Chat photo. Returned only in getChat.
description	String	*Optional*. Description, for supergroups and channel chats. Returned only in getChat.
invite_link	String	*Optional*. Chat invite link, for supergroups and channel chats. Returned only in getChat.

ILUSTRACIÓN 8 – TIPO DE CHAT

Las conversaciones de Telegram también van a ser representadas como objetos e instancias para trabajar con ellas en esta API oficial, es bastante lógico que una aplicación como Telegram guarde o almacene las conversaciones como cualquier otra plataforma lo haría, de manera que cada conversación pública o privada, ya sea grupo o no, se asociará con una instancia chat que contiene los atributos mostrados en la imagen anterior: *id*: identificador de chat, un número entero diferenciado entre grupos o conversaciones privadas, type: tipo de chat (privado, grupo, supergrupo o canal), *title*: título de la conversación, puede no ser único y exclusivo de la conversación, *username*: alias identificativo del chat, *first_name*, *last_name*: nombre y apellidos del usuario si se trata de una conversación privada, *all_member_are_administrators*: opcionalidad de todos los miembros administradores en función de si está activada o no, *photo*: foto del chat, *description*: descripción del chat y *invite_link*: enlace de invitación de la conversación.

Gran parte de estos atributos son ya muy conocidos por otras plataformas de comunicación, solo explicaremos los atributos más relevantes y nuevos que se incorporan en Telegram. El enlace de invitación es una opcionalidad exclusiva de la plataforma que permite incorporar un enlace público para incorporar miembros al chat que quieran unirse por sus propios medios. Esto es muy útil cuando quieres que cualquier usuario se pueda unir al grupo sin necesidad de que los miembros o administradores del grupo vayan invitando a los usuarios cada vez que alguien quiera unirse, simplemente se crea un enlace con el que compartes con los usuarios o bien especificas en algún lugar público para que se puedan unir cuando quieran y en el momento que deseen.

El identificador de chat para grupos es diferente con respecto a las conversaciones privadas, puesto que los grupos, supergrupos o canales son representados con números enteros negativos (ejemplo: -3895653), mientras que para representar las conversaciones privadas se utilizan los números enteros positivos (ejemplo: 3895653).

Message

This object represents a message.

Field	Type	Description
message_id	Integer	Unique message identifier inside this chat
from	User	*Optional.* Sender, can be empty for messages sent to channels
date	Integer	Date the message was sent in Unix time
chat	Chat	Conversation the message belongs to
forward_from	User	*Optional.* For forwarded messages, sender of the original message
forward_from_chat	Chat	*Optional.* For messages forwarded from a channel, information about the original channel
forward_from_message_id	Integer	*Optional.* For forwarded channel posts, identifier of the original message in the channel
forward_date	Integer	*Optional.* For forwarded messages, date the original message was sent in Unix time
reply_to_message	Message	*Optional.* For replies, the original message. Note that the Message object in this field will not contain further *reply_to_message* fields even if it itself is a reply.

ILUSTRACIÓN 9 – TIPO DE MENSAJE I

El objeto mensaje de la API tiene muchísimos atributos, de los cuales, solamente explicaremos los más importantes, puesto que la API dispone de mucha funcionalidad y la cantidad de servicios que se podrían ofrecer en un bot de Telegram son de multitud de tipos. Los atributos principales de esta imagen y de los cuales nos preocuparemos en nuestro proyecto software son: *message_id*: identificador del mensaje de un chat, *from*: usuario que ha escrito el mensaje (de donde proviene el mensaje) y *chat*: la instancia chat de donde procede el mensaje o la conversación en donde se ha escrito ese mensaje por un usuario.

El resto de atributos dependen de funcionalidades exclusivas y concretas, como las respuestas a los mensajes de otros usuarios que cuentan como mensajes, reenvíos de mensajes de otras conversaciones y cosas muy concretas que no consideramos que sean imprescindibles en el desarrollo de nuestro bot de depósito y almacenamiento de archivos.

edit_date	Integer	*Optional.* Date the message was last edited in Unix time
text	String	*Optional.* For text messages, the actual UTF-8 text of the message, 0-4096 characters.
entities	Array of MessageEntity	*Optional.* For text messages, special entities like usernames, URLs, bot commands, etc. that appear in the text
audio	Audio	*Optional.* Message is an audio file, information about the file
document	Document	*Optional.* Message is a general file, information about the file
game	Game	*Optional.* Message is a game, information about the game. More about games »
photo	Array of PhotoSize	*Optional.* Message is a photo, available sizes of the photo
sticker	Sticker	*Optional.* Message is a sticker, information about the sticker
video	Video	*Optional.* Message is a video, information about the video
voice	Voice	*Optional.* Message is a voice message, information about the file
video_note	VideoNote	*Optional.* Message is a video note, information about the video message
new_chat_members	Array of User	*Optional.* New members that were added to the group or supergroup and information about them (the bot itself may be one of these members)

ILUSTRACIÓN 10 – TIPO DE MENSAJE II

caption	String	*Optional.* Caption for the document, photo or video, 0-200 characters
contact	Contact	*Optional.* Message is a shared contact, information about the contact
location	Location	*Optional.* Message is a shared location, information about the location
venue	Venue	*Optional.* Message is a venue, information about the venue
new_chat_member	User	*Optional.* A new member was added to the group, information about them (this member may be the bot itself)
left_chat_member	User	*Optional.* A member was removed from the group, information about them (this member may be the bot itself)
new_chat_title	String	*Optional.* A chat title was changed to this value
new_chat_photo	Array of PhotoSize	*Optional.* A chat photo was change to this value
delete_chat_photo	True	*Optional.* Service message: the chat photo was deleted
group_chat_created	True	*Optional.* Service message: the group has been created
supergroup_chat_created	True	*Optional.* Service message: the supergroup has been created. This field can't be received in a message coming through updates, because bot can't be a member of a supergroup when it is created. It can only be found in reply_to_message if someone replies to a very first message in a directly created supergroup.
channel_chat_created	True	*Optional.* Service message: the channel has been created. This field can't be received in a message coming through updates, because bot can't be a member of a channel when it is created. It can only be found in reply_to_message if someone replies to a very first message in a channel.
migrate_to_chat_id	Integer	*Optional.* The group has been migrated to a supergroup with the specified identifier. This number may be greater than 32 bits and some programming languages may have difficulty/silent defects in interpreting it. But it is smaller than 52 bits, so a signed 64 bit integer or double-precision float type are safe for storing this identifier.
migrate_from_chat_id	Integer	*Optional.* The supergroup has been migrated from a group with the specified identifier. This number may be greater than 32 bits and some programming languages may have difficulty/silent defects in interpreting it. But it is smaller than 52 bits, so a signed 64 bit integer or double-precision float type are safe for storing this identifier.
pinned_message	Message	*Optional.* Specified message was pinned. Note that the Message object in this field will not contain further *reply_to_message* fields even if it is itself a reply.

ILUSTRACIÓN 11 – TIPO DE MENSAJE III

Los atributos *photo, sticker, video, voice, document y audio* van a desempeñar un rol muy importante en nuestra aplicación, puesto que necesitaremos identificar si los mensajes enviados contienen archivos y estas propiedades nos proporcionan la respuesta, de hecho, nos dan los objetos que necesitaremos para extraer los datos de los archivos e instanciarlos en nuestro sistema de ficheros de cada repositorio.

Tendremos que tener en cuenta cuando la aplicación se encuentra en un procedimiento de espera de archivos y estudiar estos mensajes para verificar si lo que ha pasado el usuario en el chat son archivos a almacenar o no. La manera de diferenciar mensajes que contienen archivos viene proporcionada por estos atributos en concreto y mediante ellos podemos diferenciar si se trata de un mensaje con solo texto o bien con cierto tipo de archivo.

El atributo *text* hace referencia al contenido del mensaje de texto siempre y cuando no contenga archivos dentro el mismo, esto nos permitirá investigar el contenido de los mensajes que mande el usuario por las conversaciones en donde se encuentre el bot y verificar si se ha enviado cierta respuesta que se estaba esperando para realizar cierto tipo de servicio o funcionalidad.

La aplicación de Telegram que vamos a elaborar no estudiará todos los mensajes que envíen los usuarios por todas las conversaciones, solamente cuando exista un procedimiento de espera de transacción o de respuesta por parte del usuario, por ejemplo: cuando el usuario esté realizando el comando de creación de carpeta o directorio en el sistema de archivos y el programa le pida que introduzca un nombre para la nueva carpeta a crear.

La propiedad *caption* se traduce como comentario de archivo, puesto que en esta y otras muchas aplicaciones de comunicación, se permite mandar ficheros con un texto asociado en el envío, esto nos puede permitir realizar varios pasos a la vez en nuestras operaciones de creación de archivos (envío de archivo y asignación de nombre para el archivo).

No obstante, existen excepciones para los diferentes archivos, de los cuales solo algunos permiten establecer un comentario de texto en los mismos durante su envío en Telegram, a pesar de esto tendremos que ocuparnos de realizar las transacciones y procedimientos necesarios en la aplicación para ofrecer los servicios de la mejor manera posible al usuario.

PhotoSize

This object represents one size of a photo or a file / sticker thumbnail.

Field	Type	Description
file_id	String	Unique identifier for this file
width	Integer	Photo width
height	Integer	Photo height
file_size	Integer	*Optional*. File size

ILUSTRACIÓN 12 – TIPO DE FOTO

Document

This object represents a general file (as opposed to photos, voice messages and audio files).

Field	Type	Description
file_id	String	Unique file identifier
thumb	PhotoSize	*Optional*. Document thumbnail as defined by sender
file_name	String	*Optional*. Original filename as defined by sender
mime_type	String	*Optional*. MIME type of the file as defined by sender
file_size	Integer	*Optional*. File size

ILUSTRACIÓN 13 – TIPO DE DOCUMENTO

Video

This object represents a video file.

Field	Type	Description
file_id	String	Unique identifier for this file
width	Integer	Video width as defined by sender
height	Integer	Video height as defined by sender
duration	Integer	Duration of the video in seconds as defined by sender
thumb	PhotoSize	*Optional*. Video thumbnail
mime_type	String	*Optional*. Mime type of a file as defined by sender
file_size	Integer	*Optional*. File size

ILUSTRACIÓN 14 – TIPO DE VIDEO

Audio

This object represents an audio file to be treated as music by the Telegram clients.

Field	Type	Description
file_id	String	Unique identifier for this file
duration	Integer	Duration of the audio in seconds as defined by sender
performer	String	*Optional.* Performer of the audio as defined by sender or by audio tags
title	String	*Optional.* Title of the audio as defined by sender or by audio tags
mime_type	String	*Optional.* MIME type of the file as defined by sender
file_size	Integer	*Optional.* File size

ILUSTRACIÓN 15 – TIPO DE AUDIO

La parte característica de las instancias que representan a los archivos de diferentes formatos es que todos ellos poseen un identificador de fichero como atributo principal (file_id) que permite clasificar a los archivos en el servidor de Telegram de manera directa e individual.

Dependiendo de cada tipo de fichero, tienen una serie de propiedades que permiten definir su duración, tipo de documento, extensión, tamaño y otras características que serán exclusivas de cada formato de archivo. Todo esto será tenido en cuenta en nuestro proyecto de aplicación y en sus funcionalidades que desarrollaremos en los siguientes apartados de esta memoria.

File

This object represents a file ready to be downloaded. The file can be downloaded via the link `https://api.telegram.org/file/bot<token>/<file_path>`. It is guaranteed that the link will be valid for at least 1 hour. When the link expires, a new one can be requested by calling getFile.

> Maximum file size to download is 20 MB

Field	Type	Description
file_id	String	Unique identifier for this file
file_size	Integer	*Optional.* File size, if known
file_path	String	*Optional.* File path. Use `https://api.telegram.org/file/bot<token>/<file_path>` to get the file.

ILUSTRACIÓN 16 – TIPO DE ARCHIVO

Probablemente la clase más importante de todas: File (Fichero), permite representar los archivos que se almacenan en el espacio de asignación correspondiente al bot dentro de los servidores de Telegram. Cada archivo tendrá un identificador diferente para cada bot independientemente del contenido y tipo, puesto que cada bot tiene su zona de almacenamiento reservada para sus archivos, además en todo momento cualquier aplicación de Telegram o la plataforma en sí se asegura de que las repeticiones de archivos o bien el spam de documentos por los chats no se tengan en cuenta siempre y cuando ya exista un archivo igual en los servidores, por tanto si un archivo enviado es repetido y existe en el propio servidor de Telegram, no se vuelve a almacenar dentro de su clasificación como un archivo nuevo, simplemente se ignora ese envío.

De esta manera, el servidor evita almacenamientos innecesarios que no sirven para nada, puesto que, si se está enviando el mismo tipo de fichero todo el rato por muchos chats de la plataforma, la funcionalidad es casi instantánea y muy eficiente en cuanto a términos de persistencia de archivos, puesto que en todas esas conversaciones de Telegram se ha referenciado al mismo archivo almacenándolo solamente una sola vez.

Ahora procederemos a hablar de los atributos de la clase fichero: *file_id*: identificador de archivo, *file_size*: tamaño del fichero en bytes y *file_path*: ruta del fichero dentro de la zona de asignación del bot en concreto, este último atributo es una ruta asociada al token del bot identificando un archivo concreto del servidor de Telegram, esto funciona de manera independiente para cada uno de los bots existentes en Telegram. Cada bot es capaz de acceder a sus propios ficheros que le han mandado o ha leído en diferentes chats en los que ha estado y por tanto tiene sus correspondientes rutas para acceder a ellos y reenviarlos o almacenarlos de distintas formas.

Existen una serie de restricciones en cuanto al manejo de archivos por parte del bot: los archivos no deben sobrepasar un tamaño de archivo de 20 MegaBytes, si los sobrepasa el archivo no se puede descargar a pesar de que en la plataforma de Telegram como tal los usuarios pueden enviar archivos de hasta 1,5 GigaBytes y la otra restricción es la de que los archivos que se intenten obtener con petición HTTP por medio del programa interactivo que elaboraremos solo tienen un máximo de una hora para ser descargados en la aplicación de la plataforma, si se quiere volver a descargar ese archivo se tendrá que volver a llamar al método de obtención del archivo correspondiente (getFile).

ReplyKeyboardMarkup

This object represents a custom keyboard with reply options (see Introduction to bots for details and examples).

Field	Type	Description
keyboard	Array of Array of KeyboardButton	Array of button rows, each represented by an Array of KeyboardButton objects
resize_keyboard	Boolean	*Optional*. Requests clients to resize the keyboard vertically for optimal fit (e.g., make the keyboard smaller if there are just two rows of buttons). Defaults to *false*, in which case the custom keyboard is always of the same height as the app's standard keyboard.
one_time_keyboard	Boolean	*Optional*. Requests clients to hide the keyboard as soon as it's been used. The keyboard will still be available, but clients will automatically display the usual letter-keyboard in the chat – the user can press a special button in the input field to see the custom keyboard again. Defaults to *false*.
selective	Boolean	*Optional*. Use this parameter if you want to show the keyboard to specific users only. Targets: 1) users that are @mentioned in the *text* of the Message object; 2) if the bot's message is a reply (has *reply_to_message_id*), sender of the original message. *Example:* A user requests to change the bot's language, bot replies to the request with a keyboard to select the new language. Other users in the group don't see the keyboard.

ILUSTRACIÓN 17 – TIPO DE CREACIÓN DE TECLADOS CUSTOM

Mediante este tipo concreto podemos generar teclados custom, elaborados de cierta manera para la realización de una transacción en concreto de una funcionalidad determinada o simplemente cuando queremos esperar la respuesta de un usuario y le ayudamos exponiendo un teclado elaborado de cierta forma con las respuestas que puede elegir para continuar con el caso de uso en específico.

KeyboardButton

This object represents one button of the reply keyboard. For simple text buttons *String* can be used instead of this object to specify text of the button. Optional fields are mutually exclusive.

Field	Type	Description
text	String	Text of the button. If none of the optional fields are used, it will be sent to the bot as a message when the button is pressed
request_contact	Boolean	*Optional*. If *True*, the user's phone number will be sent as a contact when the button is pressed. Available in private chats only
request_location	Boolean	*Optional*. If *True*, the user's current location will be sent when the button is pressed. Available in private chats only

Note: *request_contact* and *request_location* options will only work in Telegram versions released after 9 April, 2016. Older clients will ignore them.

ILUSTRACIÓN 18 – TIPO DE BOTÓN DE TECLADO

Aquí está la clase necesaria para elaborar botones de teclado que se pueden incorporar al teclado "custom" que se realiza con la clase anterior explicada. El atributo principal que vamos a usar es: *text*, que permite determinar la cadena de caracteres que representan el contenido del botón de teclado, además, este campo se va a pasar al bot cada vez que se haga click sobre el botón si no se han utilizado las opciones secundarias que explicaremos a continuación.

Los atributos opcionales son: *request_contact* y *request_location*, para enviar el número de teléfono o la localización del usuario al realizar click sobre el botón siempre y cuando estas opciones estén activadas (si los atributos tienen el valor booleano *True*). Estas opciones secundarias solo servirán y se aplicarán cuando se utilicen en conversaciones privadas, de manera que no nos interesará para nuestro bot, puesto que trabajará con grupos y además no necesitaremos estos datos de información para exponer nuestros servicios.

ReplyKeyboardRemove

Upon receiving a message with this object, Telegram clients will remove the current custom keyboard and display the default letter-keyboard. By default, custom keyboards are displayed until a new keyboard is sent by a bot. An exception is made for one-time keyboards that are hidden immediately after the user presses a button (see ReplyKeyboardMarkup).

Field	Type	Description
remove_keyboard	True	Requests clients to remove the custom keyboard (user will not be able to summon this keyboard; if you want to hide the keyboard from sight but keep it accessible, use *one_time_keyboard* in ReplyKeyboardMarkup)
selective	Boolean	*Optional*. Use this parameter if you want to remove the keyboard for specific users only. Targets: 1) users that are @mentioned in the *text* of the Message object; 2) if the bot's message is a reply (has *reply_to_message_id*), sender of the original message. *Example:* A user votes in a poll, bot returns confirmation message in reply to the vote and removes the keyboard for that user, while still showing the keyboard with poll options to users who haven't voted yet.

ILUSTRACIÓN 19 – PROCEDIMIENTO PARA BORRAR TECLADOS CUSTOM

Esta funcionalidad también es incorporada en la API para eliminar los teclados asociados a los mensajes o esperas de respuestas de los usuarios en un determinado servicio del bot. Nos servirá para eliminar cualquier tipo de teclado "custom" elaborado para cierto escenario dentro de los casos de uso que vamos a desarrollar en nuestro proyecto.

InlineKeyboardMarkup

This object represents an inline keyboard that appears right next to the message it belongs to.

Field	Type	Description
inline_keyboard	Array of Array of InlineKeyboardButton	Array of button rows, each represented by an Array of InlineKeyboardButton objects

Note: This will only work in Telegram versions released after 9 April, 2016. Older clients will display *unsupported message*.

ILUSTRACIÓN 20 – INTERFAZ GRÁFICA DE BOTONES

Esta clase va a ser fundamental en el tema de la visualización de la interfaz gráfica dentro de cada uno de los sistemas de archivos de cada chat en Telegram, puesto que nos permite incrustar una serie de botones de distintas formas posibles y según los gustos del programador para simular una visualización con un enfoque gráfico decente como las aplicaciones con interfaz gráfica como tal.

Se trata de un contenedor que se puede insertar en los mensajes y contiene listas de filas y columnas a modo de matriz bidimensional de botones asociados a la clase KeyboardButton, también contiene otras opciones secundarias para recolocar los tamaños de los botones de la visualización en función de si son grandes o pequeños, la desaparición de los botones tras clickarlos en la interfaz gráfica o la selección de usuarios que pueden utilizar esta interfaz gráfica de botones.

De las opciones secundarias que tiene esta clase, solamente nos interesa la de selección de usuarios para la activación de privacidad como mecanismo de control en la interfaz gráfica, el resto no creo que las utilicemos para nuestra aplicación de Telegram para el desarrollo del programa interactivo de sistemas de archivos, simplemente nos limitaremos a la visualización e implementación de esos botones que contienen los ficheros de los grupos.

InlineKeyboardButton

This object represents one button of an inline keyboard. You **must** use exactly one of the optional fields.

Field	Type	Description
text	String	Label text on the button
url	String	*Optional.* HTTP url to be opened when button is pressed
callback_data	String	*Optional.* Data to be sent in a callback query to the bot when button is pressed, 1-64 bytes
switch_inline_query	String	*Optional.* If set, pressing the button will prompt the user to select one of their chats, open that chat and insert the bot's username and the specified inline query in the input field. Can be empty, in which case just the bot's username will be inserted. **Note**: This offers an easy way for users to start using your bot in inline mode when they are currently in a private chat with it. Especially useful when combined with *switch_pm...* actions – in this case the user will be automatically returned to the chat they switched from, skipping the chat selection screen.
switch_inline_query_current_chat	String	*Optional.* If set, pressing the button will insert the bot's username and the specified inline query in the current chat's input field. Can be empty, in which case only the bot's username will be inserted. This offers a quick way for the user to open your bot in inline mode in the same chat – good for selecting something from multiple options.
callback_game	CallbackGame	*Optional.* Description of the game that will be launched when the user presses the button. **NOTE**: This type of button **must** always be the first button in the first row.
pay	Boolean	*Optional.* Specify True, to send a Pay button. **NOTE**: This type of button **must** always be the first button in the first row.

ILUSTRACIÓN 21 – TIPO DE BOTÓN DE VISUALIZACIÓN

En la ilustración anterior, incorporamos la clase básica para crear botones de la visualización del comando de navegación del programa interactivo que vamos a diseñar, estos botones son los que integraremos en el contenedor que hemos explicado en la anterior ilustración (InlineKeyboardMarkup). Las capacidades de desarrollo que permite este tipo predefinido de la API son muy grandes, debido a la cantidad de utilidad que proporcionan sus atributos base que explicaremos a continuación:

- Texto del botón: Etiqueta de cadena de caracteres que permite definir el contenido del botón de visualización. Esto será empleado para diferenciar los botones de manera correspondiente a sus archivos o carpetas del sistema de ficheros del repositorio en concreto.

- Url del enlace de funcionalidad del botón: Se trata de la activación del botón a una url o ruta de enlace de localización de un recurso que va a ser ejecutada y abierta cuando se haga click sobre el botón determinado. Nosotros no utilizaremos en principio esta opcionalidad, debido a que referenciaremos los archivos de nuestros repositorios de manera local y no de manera remota, aunque la posibilidad sigue estando ahí para futuros cambios o incorporaciones dentro del mantenimiento del bot.

- Datos de la función de retrollamada: Información expresada en cadena de caracteres que se va a proporcionar a la devolución de la llamada o también denominada como “callback” en términos de programación de computadoras. La API funciona de la siguiente manera, cuando se crea un botón de visualización (InlineKeyboardButton), se le debe proporcionar datos de texto que se incoporarán como argumento a una función determinada cuando un usuario de Telegram haga click y por tanto una consulta en el mismo botón. Esta función o callback que se va a ejecutar cuando se realice una consulta determinada, debe ser implementada por el programador del bot en concreto y proporcionar la funcionalidad que desee que ocurra dependiendo de la aplicación que se esté desarrollando. Aquí es donde deberemos adecuar una implementación de la navegación del sistema de ficheros en función de si los botones corresponden a carpetas o directorios, o bien mostrar el contenido de los archivos si se trata de ficheros simples con extensión y formato.

- Atributos como *switch_inline_query* y *switch_inline_query_current_chat* permiten la comunicación con el bot desde conversaciones privadas para grupos determinados u otros chats en concreto que seleccione el usuario. Es posible que se establezca algún tipo de funcionalidad extra en la aplicación en un futuro con respecto a estos temas u opciones.

- La función de referencia de llamada de juego: Solamente sirven cuando hayamos creado un juego nuevo para la plataforma de Telegram y queremos incorporarlo a la visualización propuesta en nuestra interfaz gráfica de botones. Sin embargo, nuestro proyecto no va de diseñar y desarrollar videojuegos o minijuegos, por tanto, no utilizaremos esta opcionalidad en ningún momento.

- La opción de pagar nueva que se ha incorporado recientemente en las últimas actualizaciones, si se activa este parámetro opcional se establecerá un botón de pago en la primera fila de la estructura de botones de visualización. Telegram incorpora esta nueva funcionalidad porque gran parte de las aplicaciones no serán gratuitas y necesitarán un coste de mantenimiento para la supervivencia de las aplicaciones en servicio.

Ahora comenzaremos a explicar más en detalle la función de retrollamada o más conocida como “callback” que hemos introducido anteriormente. Esta función nos permite controlar a los clientes y usuarios que seleccionen los botones de la interfaz gráfica y determinar qué tipo de botón se ha seleccionado, así como los datos del botón asociados que serán necesarios para ejercer un servicio o funcionalidad correspondiente.

La API oficial de Telegram Bot ofrece una implementación de una clase denominada CallbackQuery que representa a las consultas realizadas por los usuarios que llaman a la función de retroalimentación “callback” en cada uno de los botones establecidos en la interfaz gráfica de las conversaciones de la aplicación que pretendemos diseñar.

CallbackQuery

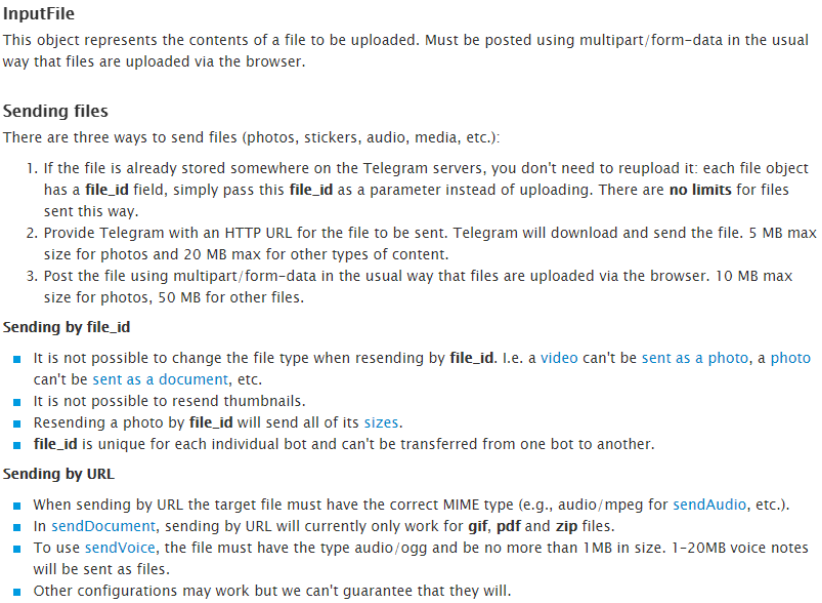
This object represents an incoming callback query from a callback button in an inline keyboard. If the button that originated the query was attached to a message sent by the bot, the field *message* will be present. If the button was attached to a message sent via the bot (in inline mode), the field *inline_message_id* will be present. Exactly one of the fields *data* or *game_short_name* will be present.

Field	Type	Description
id	String	Unique identifier for this query
from	User	Sender
message	Message	*Optional.* Message with the callback button that originated the query. Note that message content and message date will not be available if the message is too old
inline_message_id	String	*Optional.* Identifier of the message sent via the bot in inline mode, that originated the query.
chat_instance	String	Global identifier, uniquely corresponding to the chat to which the message with the callback button was sent. Useful for high scores in games.
data	String	*Optional.* Data associated with the callback button. Be aware that a bad client can send arbitrary data in this field.
game_short_name	String	*Optional.* Short name of a Game to be returned, serves as the unique identifier for the game

ILUSTRACIÓN 22 – CONSULTA DE FUNCIÓN DE RETROLLAMADA

Mediante esta instancia podemos acceder a los atributos adecuados que nos permiten diferenciar quién ha presionado ese botón, el identificador único de la consulta realizada por el usuario, la instancia de chat en la cual se ha realizado la consulta, los datos que se han especificado en la función del callback del botón correspondiente o bien el nombre del juego como identificador único (en el caso de que se tengan juegos realizados).

La API especifica que cuando se haga click en los botones con una función de callback asociada, se tiene que finalizar la acción llamando a answerCallbackQuery para responder a esa consulta de usuario específica. Es algo que no es obligatorio para su servicio, pero si se debería tener en cuenta si se quiere programar bien una aplicación de Telegram capaz de utilizar todas estas funcionalidades de consultas en botones y demás de opciones.



InputFile

This object represents the contents of a file to be uploaded. Must be posted using multipart/form-data in the usual way that files are uploaded via the browser.

Sending files

There are three ways to send files (photos, stickers, audio, media, etc.):

1. If the file is already stored somewhere on the Telegram servers, you don't need to reupload it: each file object has a **file_id** field, simply pass this **file_id** as a parameter instead of uploading. There are **no limits** for files sent this way.
2. Provide Telegram with an HTTP URL for the file to be sent. Telegram will download and send the file. 5 MB max size for photos and 20 MB max for other types of content.
3. Post the file using multipart/form-data in the usual way that files are uploaded via the browser. 10 MB max size for photos, 50 MB for other files.

Sending by file_id

- It is not possible to change the file type when resending by **file_id**. I.e. a video can't be sent as a photo, a photo can't be sent as a document, etc.
- It is not possible to resend thumbnails.
- Resending a photo by **file_id** will send all of its sizes.
- **file_id** is unique for each individual bot and can't be transferred from one bot to another.

Sending by URL

- When sending by URL the target file must have the correct MIME type (e.g., audio/mpeg for sendAudio, etc.).
- In sendDocument, sending by URL will currently only work for **gif**, **pdf** and **zip** files.
- To use sendVoice, the file must have the type audio/ogg and be no more than 1MB in size. 1-20MB voice notes will be sent as files.
- Other configurations may work but we can't guarantee that they will.

ILUSTRACIÓN 23 – ENVÍO Y TRANSFERENCIA DE FICHEROS

Hablaremos ahora de las diferentes formas de realizar envíos de archivos con la API. Como bien viene determinado en la página oficial de Telegram, se pueden enviar ficheros de tres maneras posibles: mediante el identificador de archivo que tienen en los servidores de Telegram, con una url (Uniform Resource Location) que permite identificar al archivo de manera remota o bien cargando de manera local el archivo correspondiente a partir de un ordenador específico (multipart/form-data).

Cada uno de los tipos de envío tiene sus restricciones, ya que nos encontramos ante una plataforma de comunicación que posee restricciones de envíos de archivos y traspaso de ficheros, este tipo de operaciones de transferencia de ficheros también tendrá sus limitaciones dependiendo de la forma de transferencia elegida.

Si se utiliza la forma de envío con el identificador de archivo de Telegram (file_id de archivo), se tiene que tener en cuenta que se tendrá que utilizar el método de enviar archivo por chat adecuado según el tipo del archivo (photo, video, document…) correspondiente. Se lanzará una excepción en caso de que se esté utilizando operaciones incorrectas con el tipo del archivo a enviar por parte de la API de Telegram.

Otra información útil a tener en cuenta sobre el envío de archivos por medio del identificador de ficheros (file_id) es que este identificador es único para cada archivo dentro de la zona de espacio de almacenamiento asignada para el bot de Telegram determinado. Lo que queremos decir con esto es que si se pasa un mismo archivo a varios bots de Telegram, éstas instancias de archivo vendrán identificadas con un file_id diferente a pesar de ser archivos iguales y de almacenarse solamente un archivo en los servidores de Telegram.

Esto es en parte lógico, ya que los bots que fabriquen los desarrolladores o los usuarios de Telegram no deberían acceder a todos los archivos de los servidores de Telegram en global, se les asigna una zona de almacenamiento o asignación de archivos mediante indexaciones específicas (file_id) donde pueden realizar cualquier tipo de operación para esos ficheros suyos concretos. De esta manera, el verdadero funcionamiento se lleva a cabo con tablas de asignaciones a cada uno de los bots para establecer los identificadores de ficheros para los respectivos ficheros únicos almacenados en los servidores de Telegram.

En cuanto al envío por medio de peticiones HTTP y rutas de localización de recursos, depende de cada configuración del archivo en concreto (zip, pdf…), pero en principio si se utiliza el envío de cada

formato de archivo con una correcta asociación al método adecuado, no debería haber problema en la operación de descarga y transmisión de archivo con url.

Sin embargo, nosotros no utilizaremos este tipo de envíos, si no que los transmitiremos de manera local cargándolos a partir del almacenamiento existente en el servidor. Esto permitirá que, mediante el depósito de los archivos y clasificación de los mismos en repositorios y grupos, enviemos directamente los archivos del sistema de ficheros determinado por la conversación de Telegram sin necesidad de saber el identificador de archivo.

2.2.2. Operaciones de Telegram Bot API

En este apartado explicaremos las operaciones y métodos más importantes que necesitaremos saber para la implementación de nuestra aplicación de control y gestión de archivos, para ello comenzaremos con la operación más básica de todas: el método de transferencia de mensajes por parte del bot para un determinado chat o conversación de Telegram.

sendMessage

Use this method to send text messages. On success, the sent Message is returned.

Parameters	Type	Required	Description
chat_id	Integer or String	Yes	Unique identifier for the target chat or username of the target channel (in the format `@channelusername`)
text	String	Yes	Text of the message to be sent
parse_mode	String	Optional	Send *Markdown* or *HTML*, if you want Telegram apps to show bold, italic, fixed-width text or inline URLs in your bot's message.
disable_web_page_preview	Boolean	Optional	Disables link previews for links in this message
disable_notification	Boolean	Optional	Sends the message silently. Users will receive a notification with no sound.
reply_to_message_id	Integer	Optional	If the message is a reply, ID of the original message
reply_markup	InlineKeyboardMarkup or ReplyKeyboardMarkup or ReplyKeyboardRemove or ForceReply	Optional	Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

ILUSTRACIÓN 24 – OPERACIÓN DE ENVÍO DE MENSAJE

Probablemente la principal operación de la API que permite la comunicación e interacción entre los usuarios y la propia aplicación de Telegram. Esté método será utilizado en todos los comandos e implementaciones de servicios que desarrollemos en el programa interactivo software que vamos a diseñar en la plataforma.

La funcionalidad de esta operación es la de mandar un mensaje de texto determinado a una conversación de Telegram en concreto (ya sea privada o grupal). También permite incorporar el establecimiento de la estructura de botones proporcionada por la clase InlineKeyboardMarkup mediante el parámetro *reply_markup* de la propia operación de envío de mensaje.

Hay que decir que este método de la API solamente requiere dos parámetros obligatorios como argumentos para la llamada del mismo: el identificador del chat de Telegram (chat_id) y el texto del mensaje a enviar por la conversación (text). El resto de parámetros son opcionales y no es necesario enviar estos datos durante la llamada del método o bien asignarles un valor en específico dependiendo del lenguaje de programación que implemente estas operaciones.

sendPhoto

Use this method to send photos. On success, the sent Message is returned.

Parameters	Type	Required	Description
chat_id	Integer or String	Yes	Unique identifier for the target chat or username of the target channel (in the format `@channelusername`)
photo	InputFile or String	Yes	Photo to send. Pass a file_id as String to send a photo that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a photo from the Internet, or upload a new photo using multipart/form-data. More info on Sending Files »
caption	String	Optional	Photo caption (may also be used when resending photos by *file_id*), 0-200 characters
disable_notification	Boolean	Optional	Sends the message silently. Users will receive a notification with no sound.
reply_to_message_id	Integer	Optional	If the message is a reply, ID of the original message
reply_markup	InlineKeyboardMarkup or ReplyKeyboardMarkup or ReplyKeyboardRemove or ForceReply	Optional	Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

sendDocument

Use this method to send general files. On success, the sent Message is returned. Bots can currently send files of any type of up to 50 MB in size, this limit may be changed in the future.

Parameters	Type	Required	Description
chat_id	Integer or String	Yes	Unique identifier for the target chat or username of the target channel (in the format `@channelusername`)
document	InputFile or String	Yes	File to send. Pass a file_id as String to send a file that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a file from the Internet, or upload a new one using multipart/form-data. More info on Sending Files »
caption	String	Optional	Document caption (may also be used when resending documents by *file_id*), 0-200 characters
disable_notification	Boolean	Optional	Sends the message silently. Users will receive a notification with no sound.
reply_to_message_id	Integer	Optional	If the message is a reply, ID of the original message
reply_markup	InlineKeyboardMarkup or ReplyKeyboardMarkup or ReplyKeyboardRemove or ForceReply	Optional	Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

sendVideo

Use this method to send video files, Telegram clients support mp4 videos (other formats may be sent as Document). On success, the sent Message is returned. Bots can currently send video files of up to 50 MB in size, this limit may be changed in the future.

Parameters	Type	Required	Description
chat_id	Integer or String	Yes	Unique identifier for the target chat or username of the target channel (in the format `@channelusername`)
video	InputFile or String	Yes	Video to send. Pass a file_id as String to send a video that exists on the Telegram servers (recommended), pass an HTTP URL as a String for Telegram to get a video from the Internet, or upload a new video using multipart/form-data. More info on Sending Files »
duration	Integer	Optional	Duration of sent video in seconds
width	Integer	Optional	Video width
height	Integer	Optional	Video height
caption	String	Optional	Video caption (may also be used when resending videos by *file_id*), 0-200 characters
disable_notification	Boolean	Optional	Sends the message silently. Users will receive a notification with no sound.
reply_to_message_id	Integer	Optional	If the message is a reply, ID of the original message
reply_markup	InlineKeyboardMarkup or ReplyKeyboardMarkup or ReplyKeyboardRemove or ForceReply	Optional	Additional interface options. A JSON-serialized object for an inline keyboard, custom reply keyboard, instructions to remove reply keyboard or to force a reply from the user.

ILUSTRACIÓN 25 – OPERACIONES DE ENVÍO DE FICHEROS

Las ilustraciones anteriores corresponden a los métodos y operaciones de envío y trasmisión de fotos, documentos y vídeos en una conversación concreta de Telegram. De la misma forma que la operación de transferencia de mensajes, estos métodos van a ser de lo más útil para el desarrollo de nuestro proyecto software, puesto que necesitaremos enviar los archivos de los sistemas de ficheros y repositorios en los chats de grupo de Telegram.

El control de los ficheros en cada una de las aplicaciones externas remotas será mantenido de otras formas diferentes que no se hacen referencia dentro de este apartado de explicación de las operaciones de la API, sin embargo, cuando necesitemos transferir los archivos o determinar el contenido de los mismos por Telegram, utilizaremos estas operaciones de transmisión a partir del almacenamiento local de archivos en el servidor del bot.

getFile

Use this method to get basic info about a file and prepare it for downloading. For the moment, bots can download files of up to 20MB in size. On success, a File object is returned. The file can then be downloaded via the link `https://api.telegram.org/file/bot<token>/<file_path>`, where `<file_path>` is taken from the response. It is guaranteed that the link will be valid for at least 1 hour. When the link expires, a new one can be requested by calling getFile again.

Parameters	Type	Required	Description
file_id	String	Yes	File identifier to get info about

Note: This function may not preserve the original file name and MIME type. You should save the file's MIME type and name (if available) when the File object is received.

ILUSTRACIÓN 26 – OPERACIÓN DE OBTENCIÓN DE FICHERO

Aquí tenemos otra operación muy importante que va a resultar clave en nuestra aplicación de sistemas de ficheros, este método de la API corresponde con la manera de obtener los archivos del servidor de Telegram y descargarlos para almacenarlos en nuestro servidor dedicado a nuestro programa interactivo.

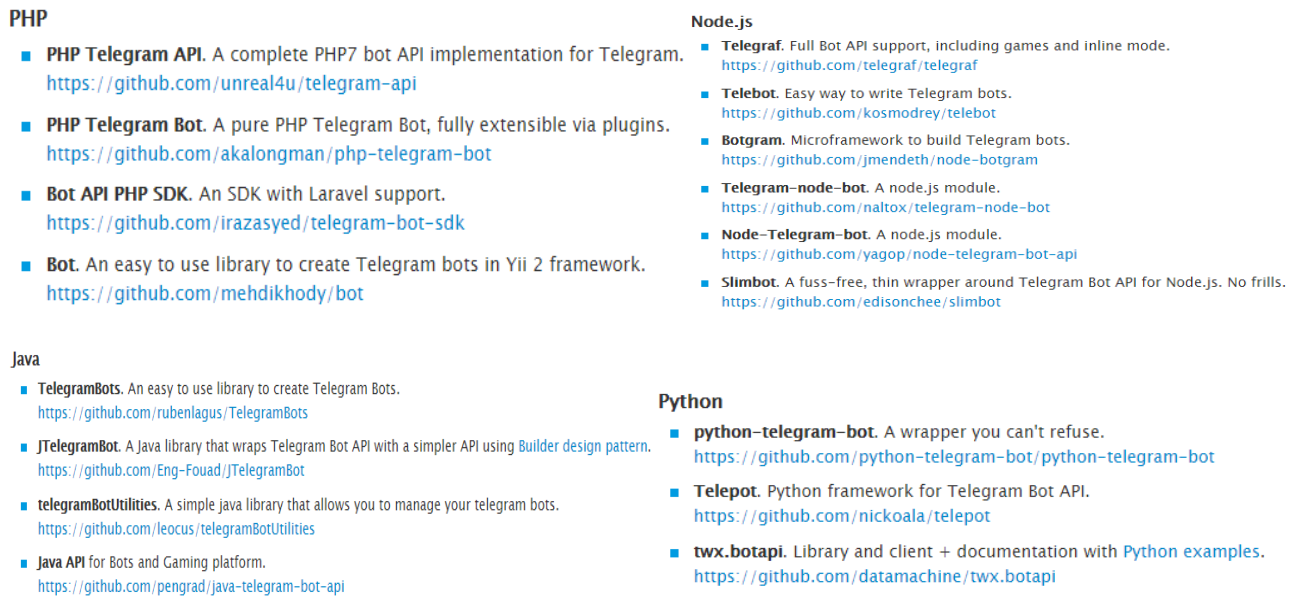
El funcionamiento consiste en descargarse los ficheros que vayan pasando los usuarios en las conversaciones privadas o grupales de Telegram y almacenarlos de manera local para sus posteriores modificaciones que puedan resultar de los servicios proporcionados por la propia aplicación, como por ejemplo el renombrado, movimiento o copia de archivos dentro de sus repositorios del servidor.

Con esto queda explicado todo lo referente al tema de la API de Bots de Telegram con sus tipos de datos y operaciones especificadas que son fundamentalmente importantes para el desarrollo del proyecto software, junto con el control de los sistemas de repositorios locales y remotos

2.2.3. Librerías de Telegram Bot API

La página web oficial de Telegram ofrece diferentes recomendaciones basadas en las diferentes librerías realizadas por la comunidad de la plataforma, este tipo de librerías son de código abierto localizadas en GitHub y cualquier programador interesado en la misma puede contactar con los dueños del repositorio para incrementar la extensibilidad de la API en el lenguaje concreto.

De manera que tenemos diferentes lenguajes de programación como Java, Node.js, Python o PHP. Existen más lenguajes de implementación de la API, sin embargo, el resto de los lenguajes mencionados son simplemente mejores y permiten desarrollar las funcionalidades del bot más cómodamente. A continuación, dejaremos una ilustración para mostrar todos los lenguajes de programación y sus librerías correspondientes que implementan la API de Telegram:



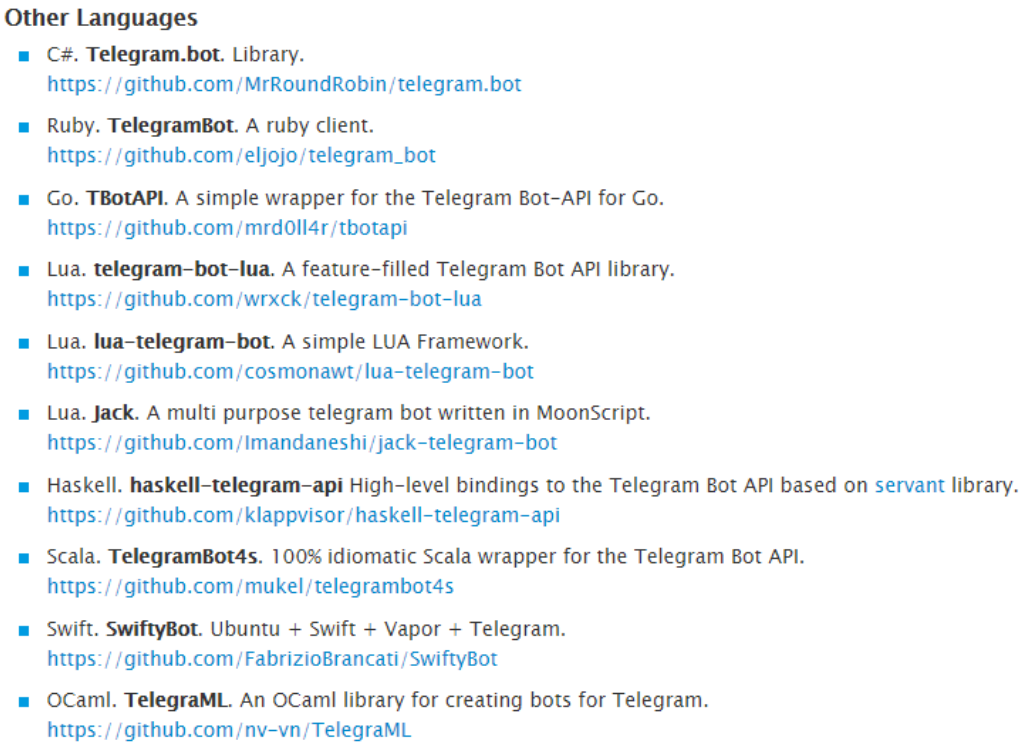


ILUSTRACIÓN 27 – LIBRERÍAS DE TELEGRAM BOT API

El lenguaje Javascript permite desarrollar aplicaciones web dinámicas (frontend) y además junto con su ampliación en Node.js puede implementar funcionalidad del lado del servidor (backend). Otro de los lenguajes de programación interpretados es Python, por el contrario de Javascript, principalmente se utiliza para realizar scripts para diversos campos que se ejecutan en el servidor, esto no quiere decir que Python no permita el desarrollo de aplicaciones web, sin embargo, para ello ya existen herramientas mejores que incorporan mejoras en cuanto a facilidad y funcionalidad como Javascript mediante el uso de HTML5.

El otro lenguaje de programación por excelencia sería Java, este tipo de lenguaje siempre adopta diferentes posturas independientemente de la funcionalidad que requiera la aplicación o la normativa en concreto. Este tipo de lenguaje a diferencia de los anteriores no es interpretado si no estático, mediante una compilación de los archivos de código se generan los ejecutables de programas. Java permite la implementación de conexiones cliente-servidor (sockets), así como el diseño de servicios web RESTful con la ayuda de un servidor de aplicaciones específico.

Estos tres lenguajes han sido los pilares de programación hasta la actualidad y cada uno se sigue utilizando en el contexto adecuado. Nosotros utilizaremos Python para la implementación de nuestro programa interactivo de almacenamiento de archivos. La razón principal de por qué usamos Python como lenguaje de programación, está en la potencia y gran capacidad del mismo para ejercer multitud de funcionalidades y objetivos para el diseño e implementación de la aplicación: comunicación con plataformas remotas y aplicaciones externas, extracción de información con web scraping, instrucciones sencillas para generar llamadas al sistema operativo y otras muchas opciones que ofrece este lenguaje. Además, no necesitaremos realizar aplicaciones web para desarrollar el bot de Telegram como tal, puesto que la API ya posee de los métodos y operaciones necesarias para la funcionalidad básica del programa dentro de Telegram. Esto nos hace orientar más nuestra aplicación hacia Python, pudiendo desarrollar scripts más elaborados para nuestro contexto de sistemas de archivos a tratar.

Otra de las razones para usar Python es que es un lenguaje dinámico y eso nos da ventajas a la hora de elaborar estructuras de tipos de datos sin determinar el tipo en tiempo de compilación del programa, de manera que al no establecer el tipo de los objetos y variables que usaremos en nuestra aplicación, lo hace bastante útil siempre y cuando se tenga en cuenta todas las posibilidades en tiempo de ejecución. Python puede resultar un lenguaje de programación difícil y fácil a su vez, dependiendo de cómo se lleve a cabo en cada momento y del conocimiento asociado al propio lenguaje por parte del programador que lo está usando.

2.3. Bots publicados oficialmente

Existen multitud de bots publicados de manera oficial en Telegram, muchos de ellos se referencian y recomiendan en la propia página web de la plataforma o bien se pueden realizar búsquedas variadas de los bots en su aplicación de visualización (storebot). StoreBot es un bot de Telegram que permite categorizar al resto de bots realizados de manera oficial y expuestos a los usuarios para su utilización en la plataforma de Telegram de diferentes maneras. Las recomendaciones principales de StoreBot son para bots que realmente han sido considerados como útiles para la comunidad y que garantizan una funcionalidad de servicio en todo momento.

StoreBot no solo se encuentra en la página web oficial de Telegram, si no que también se puede acceder y comunicar con el propio bot dentro de la plataforma como tal, lo único que hay que hacer para interactuar con él es realizar la siguiente búsqueda en el propio buscador de Telegram: Storebot. A continuación, adjunto una imagen mostrando el procedimiento necesario para ello:

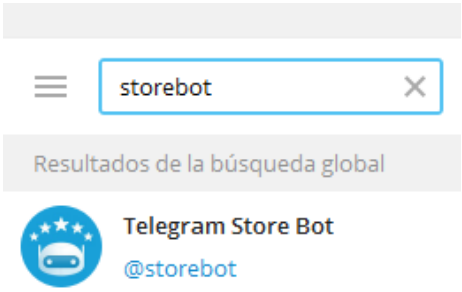


ILUSTRACIÓN 28 – BÚSQUEDA DE TELEGRAM STORE BOT

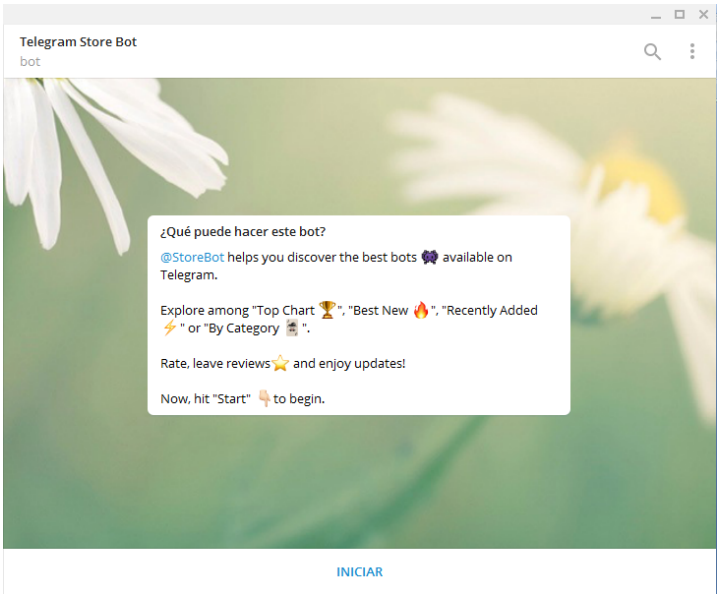


ILUSTRACIÓN 29 – TELEGRAM STORE BOT EN LA APLICACIÓN

Hablaremos en este apartado de algunos de los bots que nos han influido de manera significativa en nuestro proyecto o bien cuya funcionalidad ha sido digna de mención para nuestro programa interactivo.

De manera que el resto de bots que no sean explicados en este apartado no quiere decir que no puedan ser importantes o que no merezcan la pena utilizar, éstos siempre se pueden consultar en la propia página de consultoría de bots (storebot) y se pueden probar sus funcionalidades y servicios en la plataforma de Telegram como tal.

Un buen diseñador de aplicaciones de Telegram necesita estudiar todas las técnicas permitidas en la API y además fijarse en el modo de exposición de los servicios de cada uno de los bots desarrollados por la comunidad para la misma plataforma. Existen multitud de bots creados por la comunidad, sin embargo, solamente unos pocos serán realmente usados por los usuarios o miembros de Telegram y además no siempre se requerirá de los servicios de la aplicación en determinadas situaciones, puesto que hay ocasiones que los usuarios prefieren realizar las funcionalidades manualmente sin necesidad de bots o simplemente porque están acostumbrados a hacerlo de esa manera en concreto.

Por ello, necesitamos pensar bien qué tipo de aplicación queremos diseñar y con qué funcionalidades poblar esos servicios para satisfacer a la mayor parte de usuarios interesados, con el menor número posible de transacciones y esperas de respuesta. La utilidad y manejo sencillo de la aplicación es una característica muy importante para una aplicación a elaborar, ya que queremos que gran parte de los usuarios que utilicen la aplicación, continúen usándola a lo largo del tiempo para el desarrollo de sus diferentes tareas.

El primer bot que vamos a explicar es ImageBot, se trata de un programa que proporciona imágenes o gifs asociados a un patrón concreto que es introducido como argumento de entrada del comando determinado. Dispone de dos comandos concretos: /get para búsquedas de fotos o imágenes y /getgif para búsquedas de gifs, estas dos funcionalidades son muy fáciles de usar debido a la simplicidad de implementación de los comandos con un único argumento de entrada.

A continuación, mostramos un ejemplo de uso con el comando /get para la búsqueda de un coche aleatorio:

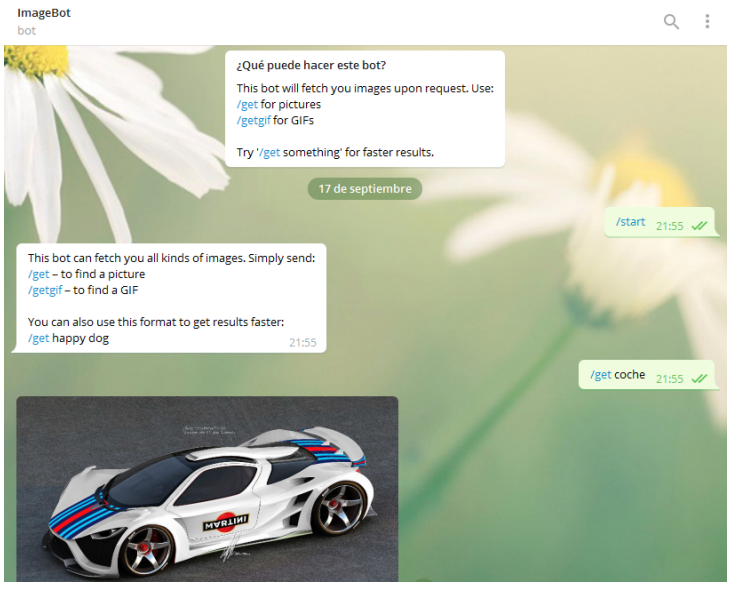


ILUSTRACIÓN 30 – IMAGEBOT

Esta aplicación es bastante aleatoria, puesto que la única funcionalidad que elabora es la de realizar búsquedas en Internet asociadas a imágenes o gifs que corresponden con el parámetro de entrada proporcionado como argumento. De esta manera, el bot puede fallar en sus búsquedas y es posible que no siempre se muestren las imágenes adecuadas a lo que se estaba realmente buscando, pero aun así la funcionalidad del bot es interesante en términos de búsquedas.

Es probable que nuestra aplicación que vamos a desarrollar tenga un comando para realizar búsquedas de archivos de manera eficiente y rápida, ya sea realizando búsquedas de manera local o bien por medio de Internet siguiendo algún patrón específico. Para el diseño de un comando de búsqueda, nos basaríamos básicamente en el comando /get con una entrada como argumento de la misma forma para desarrollar la funcionalidad.

Otro de los bots que realmente nos va a influenciar mucho en nuestro trabajo, es @filetobot cuya funcionalidad se asemeja a nuestra idea de depósito y almacenamiento de archivos, solo que no está muy detallado el almacenamiento del mismo y además no proporciona mecanismos para descargar los archivos como nosotros incorporaremos en nuestra aplicación, junto con la sincronización con aplicaciones externas de almacenamiento de archivos en la nube. Este bot incorpora mecanismos de creación de carpetas, renombrado y eliminación de las mismas, sin embargo, la gestión de archivos y su distribución en carpetas es muy diferente de la idea de un sistema de ficheros que teníamos pensada desde el comienzo de este proyecto.

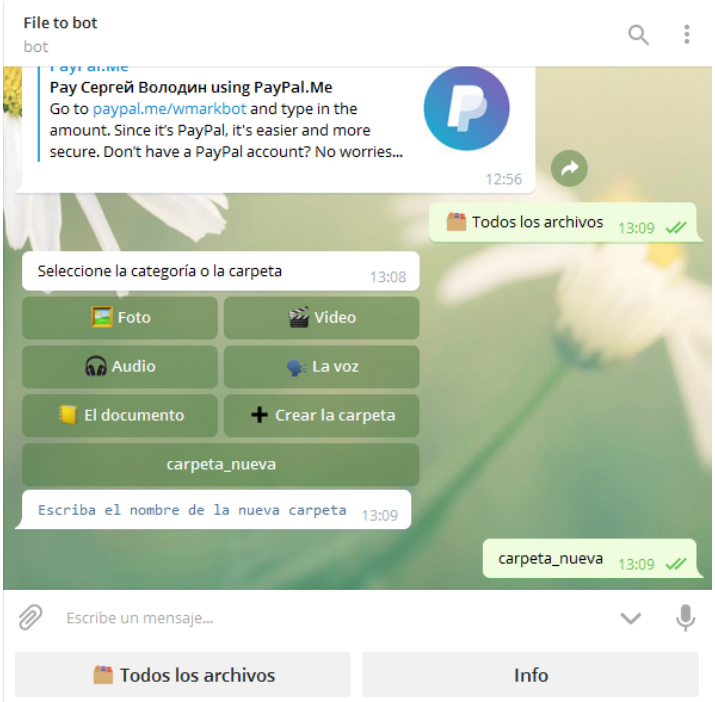


ILUSTRACIÓN 31 – FILE TO BOT

Para la agregación de archivos al bot solamente hace falta pasar archivos por la conversación de Telegram en la cual se encuentra la aplicación, pero esto tiene un inconveniente y es que todos los archivos que se pasen por el chat por cada usuario se añadirán al almacenamiento del bot, por tanto no a todos los usuarios les parecerá bien que un programa esté almacenando todos los archivos que pasen cada vez por la conversación, por ello debería haber una opción para desactivarlo o bien establecer un comando para el almacenamiento de archivo cuando el usuario lo requiera.

Además, este bot no tiene la idea principal de control y administración de sistemas de ficheros para cada repositorio de grupo, de manera que la clasificación y etiquetación de archivos solamente se haría en carpetas simples sin ningún tipo de sistema de organización reiterativo como tal. Nuestra aplicación pulirá otras facetas que este bot no implementa, como por ejemplo la gestión de archivos y la incorporación del bot en grupos, supergrupos o canales diferentes de Telegram. FileToBot solamente permite la interacción de manera privada, es decir por conversaciones directas o individuales y no en grupos de miembros de la aplicación.

Otra opción interesante que tiene este bot es la capacidad de mover archivos de una carpeta a otra en la conversación determinada, este tipo de opcionalidades tendremos que implementarlas en modo comando y en modo interfaz gráfica, puesto que necesitamos hacer que la aplicación sea usable para todo tipo de usuarios, tanto para aquellos que tengan conocimientos de comandos de intérprete como los que desconocen los conceptos principales de los comandos.

PARTE SEGUNDA - PROYECTO SOFTWARE.

CAPÍTULO 3. PLAN DE DESARROLLO SOFTWARE.

3.1. Planificación del Proyecto Software

Teniendo en cuenta que el Trabajo de Fin de Grado está valorado en 12 créditos ECTS (*European Credit Transfer and Accumulation System*), y que un crédito equivale a 25 horas, podemos estimar que la duración del proyecto total será de un total de 300 horas, por lo que se ha adaptado el alcance del proyecto para tratar de ajustarse a dicha duración.

Se han repartido las 300 horas estimadas del proyecto entre 30 semanas, de manera que quedarían en 10 horas semanales. Vamos a elaborar una planificación basada en un plan de fases e iteraciones del desarrollo software, este plan consiste en la estructuración del proyecto en dos fases: fase de análisis y fase de diseño, cada una de ellas vendrán explicadas posteriormente en esta memoria y suponen una etapa de construcción del modelado antes de la implementación de la aplicación, se establecerán las iteraciones que se necesiten en cada una de las fases del plan siempre y cuando sean necesarias para añadir mejoras y colaborar al rendimiento del proyecto software como tal.

Duración estimada: 300 horas
Horas semanales: 10 horas
Número de semanas: 30

3.2. Actividades y tareas

A continuación, expondremos las actividades que se han propuesto para la elaboración y planificación de la aplicación de depósito y almacenamiento de archivos en Telegram que vamos a desarrollar como Trabajo de Fin de Grado de Ingeniería Informática, todas las tareas tienen asociadas una estimación en horas de la duración del trabajo o finalización de la actividad como tal, además se han estructurado de forma secuencial para su realización en el tiempo.

	Desglose de tareas	
Tarea	**Descripción**	**Horas semanales**
T-001	Especificar las actividades del desarrollo software	6 horas
T-002	Investigación de servidores de nube y comparativas de costes	6 horas
T-003	Establecer presupuesto y calendarización	5 horas
T-004	Conocimiento de Telegram Bot API y sus operaciones	8 horas
T-005	Configurar el entorno de desarrollo	8 horas
T-006	Creación de un prototipo base de la aplicación	7 horas
T-007	Creación del diagrama de casos de uso	6 horas

T-008	Creación del modelo de dominio	15 horas
T-009	Creación de los diagramas de secuencia de los casos de uso	25 horas
T-010	Implementar la funcionalidad básica de inicialización de chat (start)	3 horas
T-011	Implementar la funcionalidad de ayuda (help)	5 horas
T-012	Implementar la funcionalidad de creación de sistemas de ficheros	8 horas
T-013	Implementar la funcionalidad del modelado de repositorios	10 horas
T-014	Implementar la funcionalidad de creación de carpetas (mkdir)	6 horas
T-015	Implementar la funcionalidad de creación y subida de archivos (push)	10 horas
T-016	Implementar la funcionalidad de borrado de carpetas (rmdir)	10 horas
T-017	Implementar la funcionalidad de borrado de archivos (rm)	6 horas
T-018	Implementar la funcionalidad de mover ficheros (mv)	10 horas
T-019	Implementar la funcionalidad de copiar ficheros (cp)	10 horas
T-020	Implementar la funcionalidad de renombrado de ficheros	5 horas
T-021	Implementar la funcionalidad de cancelar transacción de servicio (cancel)	5 horas
T-022	Implementar la búsqueda eficiente de ficheros (get)	8 horas
T-023	Desarrollo de interfaz gráfica de navegación de archivos	10 horas
T-024	Implementación de los comandos en interfaz gráfica	15 horas
T-025	Implementar la posibilidad de sincronización con plataformas externas	10 horas
T-026	Implementar la funcionalidad de sincronización con Bitbucket	15 horas
T-027	Implementar la funcionalidad de sincronización con Github	15 horas
T-028	Realizar pruebas de rendimiento del servidor	5 horas
T-029	Realizar pruebas de software de funcionamiento de la aplicación	5 horas
T-030	Crear el manual de instalación del servidor	6 horas
T-031	Crear el manual de instrucciones de usuario	10 horas

T-032	Documentar el proyecto software	10 horas
T-033	Revisar, corregir y mejorar el proyecto con su documentación	15 horas

Como cada actividad puede venir determinada por una duración concreta, vamos a ir asociando cada una de las actividades de nuestra planificación del proyecto software a las semanas correspondientes que hemos establecido para distribuir el trabajo que hay que realizar en las 300 horas totales de duración del proyecto.

Semana	Fechas	Actividades
1	24 Julio - 30 Julio	1,2
2	30 Julio - 5 Agosto	2, 3, 4
3	6 Agosto - 12 Agosto	4, 5
4	13 Agosto - 19 Agosto	5, 6
5	20 Agosto - 26 Agosto	7, 8
6	27 Agosto – 2 Septiembre	8
7	3 Septiembre - 9 Septiembre	8, 9
8	10 Septiembre - 16 Septiembre	9
9	17 Septiembre - 23 Septiembre	9, 10, 11
10	24 Septiembre - 30 Septiembre	11, 12
11	1 Octubre - 7 Octubre	12, 13
12	8 Octubre - 14 Octubre	13, 14, 15
13	15 Octubre - 21 Octubre	15, 16
14	22 Octubre - 28 Octubre	16, 17
15	29 Octubre - 4 Noviembre	17, 18
16	5 Noviembre - 11 Noviembre	18, 19
17	12 Noviembre - 18 Noviembre	19, 20, 21
18	19 Noviembre - 25 Noviembre	21, 22
19	26 Noviembre – 2 Diciembre	22, 23
20	3 Diciembre - 9 Diciembre	23, 24
21	10 Diciembre – 16 Diciembre	24, 25
22	17 Diciembre - 23 Diciembre	25, 26
23	24 Diciembre - 30 Diciembre	26
24	31 Diciembre - 6 Enero	26, 27
25	7 Enero - 13 Enero	27, 28
26	14 Enero - 20 Enero	28, 29, 30
27	21 Enero - 27 Enero	30, 31
28	28 Enero - 3 Febrero	31, 32
29	4 Febrero - 10 Febrero	32, 33
30	11 Febrero - 17 Febrero	33

Esta tabla muestra la planificación general del proyecto a lo largo de las 30 semanas de duración, junto con las actividades y tareas realizadas en cada una de esas horas semanales acordadas en un principio. La fecha de inicio corresponde con el 24 de Julio de 2017, mientras que la fecha de finalización del proyecto está programada para el 17 de Febrero de 2018.

La verdadera razón del por qué no hemos utilizado un desarrollo rápido y ágil basado en SCRUM con Sprints es porque este proyecto requiere de la capacidad cognitiva del plan de fases de la ingeniería del software, puesto que existe una gran cantidad de información a manejar y necesitamos las técnicas y herramientas de modelado para obtener el conocimiento y la comprensión de la aplicación a gran escala que queremos desarrollar.

3.3. Cálculo del presupuesto

Es necesario establecer una estimación del coste de desarrollo antes de llevarlo a cabo, para después poder establecer una comparación con el coste real, observar y analizar las desviaciones.

Para estimar el coste salarial, se ha tomado como referencia el convenio que establece el Ministerio de Trabajo e Inmigración [36], que está en vigor desde 2009. En él se recoge que el salario por convenio de un Analista programador y Diseñador de página web es de 21.555,66 € anuales. Si suponemos que se trabajan 5 días a la semana como horario normal y cada día se dedican 8 horas a la elaboración del proyecto software, tendremos el siguiente cálculo:

21.555,66 € / 12 meses / 4 semanas/mes / 40 horas/semana = 11,22690625 € / hora.

11,22690625 €/hora x 300 horas/hombre = 3368.071875 €

La razón por la que se escoge el perfil de Analista programador y Diseñador de página web se debe a que el proyecto involucra decisiones de análisis y diseño que escapan a las competencias y conocimientos de un Programador Senior, además debido a la metodología de trabajo empleada, no resulta factible diferenciar el coste de trabajadores con roles propios de cada fase, como podría ser el de Jefe de Proyecto o Administrador de test.

En cuanto al hardware utilizado, se emplea un Acer Extensa X2510 Intel Core i3-4030U con las siguientes características: 4 GB de RAM, 500 GB de disco duro y de 15,6”. El coste de adquisición de este ordenador portátil es de 374,51 € con impuestos de IVA y costes de portes incluidos. En relación con la amortización de equipos informáticos, la Agencia Tributaria establece un coeficiente lineal máximo del 25%, y un período máximo de 8 años.

374,51 € * 25 / 100 = 93,6275 €/año
Teniendo en cuenta que el proyecto se va a desarrollar durante medio año:
93,6275 x 0,5 = 46,81375 €

A continuación, se estima el coste total del proyecto, sumando a los dos costes anteriores un nuevo coste relacionado con la cuenta Premium de la plataforma de desarrollo Cloud9 como servidor dedicado para nuestra aplicación de Telegram:

Estimación del coste

3368.071875 € Salario del Analista Programador y Diseñador de página web

46,81375 € Coste amortizado del equipo hardware empleado

15,947625 €/mes x 6 meses = 95,68575 € Coste de cuenta Premium en Cloud9

Coste total: **3510.571375 €**

3.4. Costes reales

En la sección anterior se detallaba la estimación inicial de los costes para el proyecto; a continuación, vamos a comparar los costes reales con los previstos en dicho apartado, y a obtener conclusiones sobre los resultados.

Teniendo en cuenta que se mantiene el coste por hora que se le paga al Analista Programador y Diseñador de página web, dado que se han realizado menos horas de las previstas en un principio (2 horas menos en la última semana), tenemos una pequeña disminución del coste por este lado.

11,22690625 €/hora x 298 horas/hombre = 3345.618063 € Coste salarial
Disminución de 22.4538125 €

El coste del hardware se ha mantenido tal y como se había previsto, ya que no se han producido averías ni problemas que supusiesen modificar el gasto previsto, tampoco se ha reutilizado otro tipo de material informático para realizar el desarrollo de la aplicación ni ningún ordenador alternativo.

46,81375 € Coste amortizado del equipo hardware empleado
Ajustado a la previsión

En cuanto al servidor dedicado para la ejecución de la aplicación de Telegram, utilizaremos las instancias EC2 de los servicios propuestos por Amazon Web Services (AWS), que nos proporcionan un año gratis con el modelo t2.micro con un funcionamiento estable y seguro. Además, el precio de esta instancia al mes es menor que los servicios que ofrece Cloud9 o incluso puede ser mucho más escalable que otras plataformas con un precio similar como Heroku.

0.00995588513 €/hora x 24 horas/día x 7 días/semana x 4 semanas/mes = 6.690354807 €/mes
6.690354807 €/mes x 6 meses = 40.14212884 € Coste de contratar una instancia EC2 t2.micro en AWS

El coste de los servicios de AWS es bastante diferente a cualquier otro tipo de servicio ofrecido por otra plataforma concreta, puesto que todos los servicios de Amazon se cobran por horas y más concretamente por segundos según su política más actual de facturación de instancias. Este tipo de facturación viene especialmente bien en otro tipo de aplicaciones que se benefician de hibernaciones y suspensiones durante un tiempo determinado, sin embargo, en principio en nuestra aplicación de Telegram no afectaría y habría que calcular el precio en función del mes como tal.

El coste final del proyecto ha sido de 3455.027754 €, con una diferencia de 55.543621 € calculada para medio año del desarrollo del programa interactivo, por no decir que Amazon proporciona en su capa gratuita de servicios un año entero gratuito en la instancia EC2 que sirve perfectamente para alojar la aplicación de Telegram y generar un correcto funcionamiento en ese tiempo.

Coste real
3368.071875 € Salario del Analista Programador y Diseñador de página web
46,81375 € Coste amortizado del equipo hardware empleado
40.14212884 € Coste de contratar una instancia EC2 t2.micro en Amazon Web Service
Coste total: **3455.027754 €**

CAPÍTULO 4. INGENIERÍA DEL SOFTWARE

4.1. Requisitos del proyecto

4.1.1. Requisitos funcionales

Los requisitos funcionales de la aplicación corresponden con toda la funcionalidad que debe tener la aplicación de la plataforma de Telegram con los diferentes servicios pensados para el almacenamiento y depósito de archivos, éstos vienen determinados de la siguiente manera:

- El sistema deberá gestionar un sistema de ficheros para cada grupo o conversación privada de Telegram asociado con una ruta identificativa.
- El sistema deberá almacenar los archivos en el sistema de ficheros asociado al grupo, pudiendo incluirse en el propio directorio raíz o bien en una de las carpetas de su interior del propio repositorio.
- El sistema deberá permitir crear una carpeta con una ruta identificativa única en el repositorio local, ya sea en el directorio raíz o dentro de otras carpetas de su interior.
- El sistema deberá permitir eliminar una carpeta o directorio tanto con contenido como vacía del repositorio local mediante una ruta identificativa.
- El sistema deberá permitir subir o almacenar archivos nuevos con el formato adecuado que se hayan proporcionado por el usuario en el directorio raíz o en una de las carpetas del repositorio local.
- El sistema deberá almacenar los nuevos archivos con un nombre concreto en su ruta identificativa, ya sea el del archivo original proporcionado o bien pidiendo uno nuevo al usuario.
- El sistema deberá permitir eliminar un archivo de un formato concreto en el sistema de ficheros de un grupo o conversación privada.
- El sistema deberá permitir renombrar un fichero (archivo o carpeta) en el sistema de ficheros de un grupo o conversación privada.
- El sistema deberá permitir mover un fichero de una ruta concreta a otra correspondiente situada dentro del sistema de ficheros de un grupo o conversación privada.
- El sistema deberá permitir copiar un fichero de una ruta específica en otra asociada al repositorio local en concreto.
- El sistema deberá permitir al usuario consultar la búsqueda de un fichero de la manera más eficiente posible dentro de un repositorio local.
- El sistema deberá permitir cancelar un comando específico que se esté llevando a cabo o que se encuentre en un proceso de espera.
- El sistema deberá incorporar una funcionalidad opcional de sincronización con una plataforma externa de control de repositorios remotos de ficheros (Bitbucket, Github, Google Drive, Dropbox, etc).
- El sistema deberá mantener sincronizados ambos repositorios, tanto el local como el remoto asociado a la plataforma externa elegida, siempre y cuando se haya establecido anteriormente una sincronización concreta.
- El sistema deberá permitir al usuario del grupo o conversación privada obtener un enlace web del repositorio remoto de la plataforma externa sincronizada.
- El sistema deberá proporcionar una opción de desincronización con la plataforma externa para los sistemas de ficheros de grupos conectados de manera remota.

- El sistema deberá proporcionar una interfaz gráfica para la navegación de carpetas y directorios del sistema de ficheros de un grupo o conversación privada en específico.
- El sistema deberá integrar todas y cada una de las opciones de creación de ficheros, borrado de ficheros, renombrado de ficheros, mover ficheros, copiar ficheros o cancelación de transacciones en la interfaz gráfica de la estructura de navegación del sistema de ficheros.
- El sistema deberá mostrar el contenido de la carpeta con el conjunto de ficheros que se encuentran en la misma atendiendo a la estructura de interfaz gráfica especificada anteriormente para la navegación.
- El sistema deberá envíar el archivo por la conversación de Telegram cuando un usuario permitido lo haya seleccionado en la interfaz gráfica del sistema de ficheros de un repositorio.
- El sistema deberá gestionar las opciones de privacidad de cada uno de los grupos y repositorios de Telegram con los que ha interactuado.
- El sistema deberá permitir a los administradores del grupo activar la privacidad en cualquier momento para la propia navegación privada en las interfaces gráficas asociadas al sistema de ficheros del repositorio concreto.
- El sistema deberá proporcionar una información de ayuda para el grupo o conversación privada, especificando el alcance, objetivos y funcionalidades de la aplicación en general.
- El sistema deberá proporcionar una opcionalidad para comprimir un repositorio local del sistema de ficheros correspondiente y enviarlo en la conversación adecuada de la plataforma.

4.1.2. Requisitos no funcionales

Los requisitos no funcionales de la aplicación son los siguientes:

- La interfaz gráfica del sistema de archivos referente a la navegación deberá ser sencilla y fácil de usar para todos los usuarios de Telegram.
- El sistema deberá ser interactivo con el usuario, proporcionando la máxima ayuda posible en todos los comandos posibles.
- El sistema deberá enviar la menor información posible en cada interacción y transacción de comando en la conversación (menor spam posible).
- La plataforma del sistema deberá utilizar un sistema operativo Linux, tener instalado el lenguaje de programación Python con la versión 3 para evitar errores de pérdidas en las peticiones HTTP y los paquetes de la librería de Python para la API de Telegram.
- La plataforma de alojamiento del sistema deberá tener instalado los paquetes de programación de librerías necesarios y API's asociadas a Bitbucket, Github, Google Drive y Dropbox para la subida de archivos con Python.
- La plataforma de alojamiento del sistema deberá disponer de una buena conexión a Internet para poder ejecutar peticiones HTTP al servidor de Telegram o bien realizar subidas y modificaciones de archivos a las plataformas externas sincronizadas de los repositorios desde la propia aplicación.
- El sistema deberá basarse en los comandos de la terminal del sistema operativo Linux/GNU y el resto de sistemas derivados de la familia Unix para la implementación de sus propios servicios que ejercen funcionalidades similares.
- El sistema deberá atender a todos los usuarios en todo momento, así como gestionar un conjunto de hilos base para el posible aumento de conversaciones y grupos existentes en Telegram comunicándose con la aplicación.

4.1.3. Requisitos de información

Los requisitos de información de la aplicación corresponden con todo tipo de información y datos que serán necesarios almacenar de manera persistente para que la aplicación funcione de manera correcta ejerciendo sus servicios, de manera que reunimos los siguientes requisitos de información:

- El sistema deberá almacenar un archivo JSON con los datos referentes a los archivos creados en los sistemas de ficheros de cada repositorio, guardando así la ruta identificativa y el identificador propio para cada archivo.
- El sistema deberá guardar y almacenar los archivos y carpetas que se vayan creando en los repositorios locales de los grupos y conversaciones privadas de Telegram.
- El sistema deberá almacenar un archivo con los datos de usuario y contraseña con seguridad integrada para realizar subidas y modificaciones en los repositorios remotos con el sistema de control de versiones Git por ssh.
- El sistema deberá guardar en la propia carpeta del sistema de ficheros todos los archivos de configuración necesarios (Configuración de claves públicas y privadas, información del sistema de control de versiones y plataforma externa…) para realizar subidas de archivos a las aplicaciones remotas.
- El sistema deberá almacenar y gestionar un archivo para la privacidad de grupos y conversaciones privadas de Telegram, guardando el nombre de los repositorios locales cuya privacidad se encuentre activada.

4.2. Casos de uso de la aplicación

4.2.1. Diagrama de casos de uso

El diagrama de casos de uso se ha diseñado utilizando Astah Professional como herramienta de modelado con la licencia de la universidad que se le proporciona a cada alumno del grado. Los casos de uso vienen determinados por el conjunto de funcionalidades que hemos explicado y definido anteriormente en el anterior apartado de elicitación y análisis de requisitos de la aplicación software.

En nuestro caso a modelar, todos los usuarios del grupo o conversación privada pueden realizar las funcionalidades básicas de la aplicación: creación y borrado ficheros, renombrado de ficheros, cancelación de comandos pendientes, mover ó copiar ficheros de una ruta a otra, búsqueda de archivos con un nombre específico, obtener enlace de sincronización remota o compresión de repositorio en un formato adecuado.

Por otra parte, existe otro tipo específico de usuario denominado administrador, que puede realizar todas las funcionalidades básicas anteriores y además puede utilizar varios comandos a mayores con respecto al resto de usuarios normales. Estos comandos son: Modificar la privacidad de la interfaz gráfica de visualización del sistema de ficheros, sincronizar el repositorio local con una plataforma externa remota y finalmente desincronizar la plataforma y aplicación externa en el caso de haberla.

Explicaremos en detalle estos casos de uso más adelante, pero básicamente podemos afirmar que un administrador de un chat de Telegram puede determinar si la visualización o interfaz gráfica de botones de los ficheros en el grupo es pública o privada, esto afecta a la hora de navegar en el propio sistema de ficheros por parte de los usuarios.

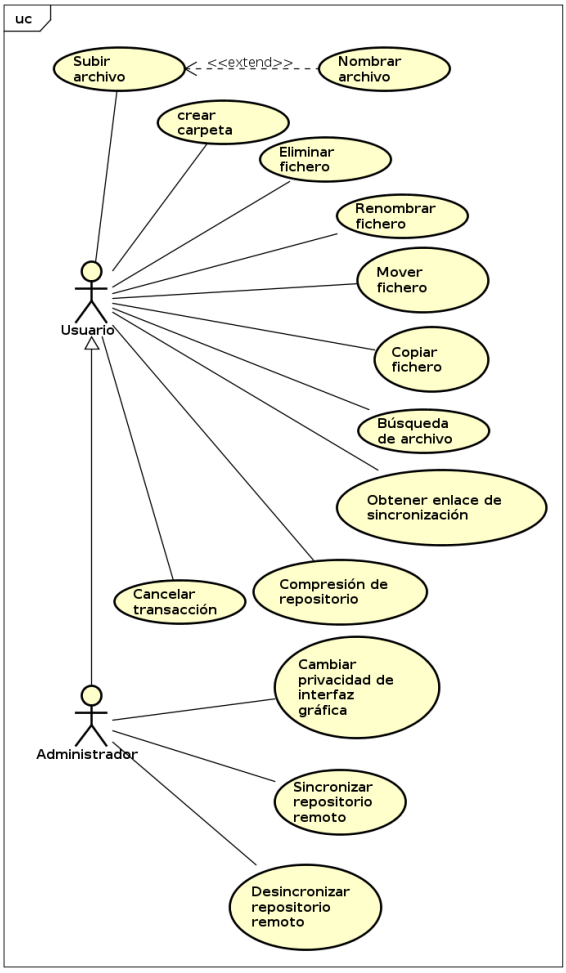


ILUSTRACIÓN 32 – DIAGRAMA DE CASOS DE USO

Si se establece en modo público, la navegación de esa visualización puede ser afectada por todos los usuarios y por tanto todos los miembros del grupo pueden navegar en la interfaz de botones cuando alguien ha utilizado un comando de visualización del contenido de archivos. Si por el contrario, se establece una visualización privada, solamente se permitirá navegar al usuario que ha utilizado ese comando y por tanto las opciones de ajustes de la interfaz de creación, eliminación y modificación de contenido de archivos podrán ser utilizadas solo por ese usuario.

La integración de interfaz privada es una opción muy útil para los usuarios de los grupos, puesto que normalmente el usuario querrá navegar entre los archivos y directorios sin necesidad de que otro usuario pueda interrumpir esa navegación y su privacidad de contenido. Por ello, me ha parecido una opción más que interesante a integrar como mecanismo de preferencia por parte de los administradores del grupo.

Las otras dos opciones de sincronización/desincronización de repositorio remoto con la aplicación requerirán del establecimiento de conexiones con aplicaciones externas y plataformas ajenas que sean acordes durante la ejecución de nuestra aplicación de Telegram. Por ello, en un principio propondremos dos tipos de plataformas de almacenamiento de archivos en la nube que sean gratuitas: Bitbucket y Github mediante el uso del sistema de control de versiones Git para el control de los repositorios remotos asociados.

La razón principal del por qué solamente los administradores pueden utilizar estas opciones reside en la importancia de los servicios que ofrecen estos casos de uso, pretenden realizar conexiones y adaptar la privacidad de cada una de las funcionalidades del grupo para los usuarios. Además,

establecer una serie de opciones de ajustes o preferencias para el grupo y las conversaciones es algo bastante bueno en general para cualquier tipo de aplicación, ya que permite customizar los servicios que se ofrecen y hace a la aplicación atractiva para el conjunto de usuarios.

4.2.2. Descripción de casos de uso

En este apartado describiremos los casos de uso que hemos establecido en el diagrama de casos de uso anterior del proyecto software. Nos basaremos en la programación de una interfaz gráfica mediante la API de Telegram para establecer la interacción entre el usuario y el sistema de la aplicación dentro de la descripción de los mismos, puesto que los comandos que implementaremos en la aplicación serán la mayoría instantáneos y además no corresponden con la mecánica de caso de uso como tal.

CU Nombrar archivo	
Pasos	Escenario general
1	El sistema pide al usuario que introduzca un nuevo nombre para el archivo.
2	El usuario introduce un nombre concreto para el archivo.
3	El sistema verifica que no exista otro fichero igual con ese nombre en la ruta de directorios, lo crea de manera definitiva e informa al usuario de su creación.
Pasos	Excepciones
3.1	El sistema encuentra otro fichero con esa ruta e informa al usuario de que no puede elegir ese nombre para el fichero a crear, el caso de uso continúa por el paso 2.

CU Eliminar fichero	
Pasos	Escenario general
1	El usuario selecciona la opción de ajustes eliminar fichero.
2	El sistema pide al usuario que seleccione un fichero del grupo.
3	El usuario selecciona un fichero del repositorio del grupo.
4	El sistema elimina el fichero seleccionado del sistema de ficheros del grupo e informa al usuario de su borrado.
Pasos	Excepciones
3.1	El usuario selecciona una opción de ajustes diferente y el caso de uso queda sin efecto, se realiza el caso de uso correspondiente a la opción de ajustes seleccionada.
3.2	El usuario selecciona la misma opción de ajustes y se le informa de que ya está llevando a cabo esa funcionalidad, el caso de uso continúa por el paso 3.

CU Subir archivo	
Pasos	Escenario general
1	El usuario selecciona la opción de ajustes subir archivo.
2	El sistema guarda la carpeta actual y pide al usuario que envíe un archivo.

3	El usuario envía un archivo que desea subir al sistema.
4	El sistema descarga el archivo correspondiente, verifica que existe un nombre asociado y lo crea en el sistema de ficheros del grupo con sus características específicas.
Pasos	Excepciones
4.1	El sistema no encuentra un nombre asociado y se realiza el caso de uso nombrar archivo.
4.2	El sistema no puede descargar y crear el archivo debido al exceso de la capacidad máxima (20 MB) que deben tener los archivos en Telegram e informa al usuario de su imposibilidad.

CU Crear carpeta	
Pasos	Escenario general
1	El usuario selecciona la opción de ajustes crear carpeta.
2	El sistema pide al usuario la introducción de un nombre concreto para el nuevo directorio a crear dentro del repositorio.
3	El usuario introduce un nombre determinado para la nueva carpeta.
4	El sistema verifica que no existe otro fichero con esa ruta y por tanto con ese nombre concreto, crea la carpeta e informa al usuario de su creación.
Pasos	Excepciones
4.1	El sistema encuentra otro fichero con esa ruta e informa al usuario de que es imposible crear una carpeta con ese nombre, el caso de uso continúa por el paso 3.

CU Renombrar fichero	
Pasos	Escenario general
1	El usuario selecciona la opción de ajustes renombrar fichero.
2	El sistema indica al usuario que seleccione un fichero a renombrar.
3	El usuario selecciona un fichero para renombrar.
4	El sistema almacena el fichero seleccionado y pide al usuario la introducción de un nombre nuevo para renombrar ese fichero.
5	El usuario introduce un nombre concreto para el fichero.
6	El sistema verifica que ese nombre es posible para el fichero y modifica la ruta del fichero con el nuevo nombre.
Pasos	Excepciones

3.1	El usuario selecciona una opción de ajustes diferente y el caso de uso queda sin efecto, se realiza el caso de uso correspondiente a la opción de ajustes seleccionada.
3.2	El usuario selecciona la misma opción de ajustes y se le informa de que ya está llevando a cabo esa funcionalidad, el caso de uso continúa por el paso 2.
6.1	El sistema encuentra que ya existe un fichero con ese nombre dentro de la ruta de directorios y el caso de uso continúa por el paso 5.

CU Cancelar transacción	
Pasos	Escenario general
1	El usuario selecciona la opción de ajustes cancelar opción.
2	El sistema verifica si existen procesos de espera para ese usuario y cancela esas transacciones finalizando así el caso de uso.
Pasos	Excepciones
2.1	El sistema no encuentra transacciones de ese usuario ni procesos relacionados, informa al usuario y el caso de uso queda sin efecto.

CU Mover fichero	
Pasos	Escenario general
1	El usuario selecciona la opción de ajustes mover fichero.
2	El sistema pide al usuario que seleccione un fichero que desee mover de la carpeta actual.
3	El usuario selecciona un fichero de la carpeta actual.
4	El sistema almacena el fichero seleccionado y pide al usuario que navegue por el sistema de ficheros hasta donde quiera mover el fichero.
5	El usuario indica el directorio a donde quiere mover el fichero seleccionado anteriormente.
6	El sistema comprueba que el movimiento es posible de realizar y mueve el fichero al directorio indicado.
Pasos	Excepciones
3.1, 5.1	El usuario selecciona una opción de ajustes diferente y el caso de uso queda sin efecto, se realiza el caso de uso correspondiente a la opción de ajustes seleccionada.
3.2	El usuario selecciona la misma opción de ajustes y se le informa de que está llevando a cabo esa funcionalidad, el caso de uso continúa por el paso 3.
6.1	El sistema encuentra que no es posible realizar el movimiento ya que existe un fichero con el mismo nombre en la ruta destino, el caso de uso continúa por el paso 5.

CU Copiar fichero	
Pasos	Escenario general
1	El usuario selecciona la opción de ajustes copiar fichero.
2	El sistema pide al usuario que seleccione un fichero que desee copiar.
3	El usuario selecciona un fichero determinado del sistema de ficheros del grupo.
4	El sistema guarda el fichero seleccionado y pide al usuario que navegue por el sistema de ficheros indicando la ruta en donde quiera copiar el fichero.
5	El usuario selecciona el directorio en donde se copiará el fichero.
6	El sistema comprueba que la copia se puede realizar, copia el fichero al directorio indicado y el caso de uso finaliza de manera correcta.
Pasos	Excepciones
3.1, 5.1	El usuario selecciona una opción de ajustes diferente y el caso de uso queda sin efecto, se realiza el caso de uso correspondiente a la opción de ajustes seleccionada.
3.2	El usuario selecciona la misma opción de ajustes y se le informa de que está llevando a cabo esa funcionalidad, el caso de uso continúa por el paso 3.
6.1	El sistema encuentra que no es posible realizar la copia ya que existe un fichero con el mismo nombre en la ruta destino, el caso de uso continúa por el paso 5.

CU Búsqueda de archivo	
Pasos	Escenario general
1	El usuario utiliza el comando de búsqueda de archivo con un nombre determinado.
2	El sistema busca en el sistema de ficheros del grupo asociado todos los archivos referentes con ese nombre especificado como argumento de entrada.
Pasos	Excepciones
2.1	El sistema no encuentra ningún archivo relacionado con ese nombre e informa al usuario.

CU Sincronizar repositorio	
Pasos	Escenario general
1	El usuario utiliza el comando de sincronización del repositorio remoto.
2	El sistema muestra las plataformas externas disponibles para la aplicación y pide al usuario que elija una de ellas.
3	El usuario elige una plataforma externa remota para realizar la sincronización.
4	El sistema sincroniza el repositorio local con la plataforma externa remota subiendo así todos los ficheros que se encuentran en el grupo.
Pasos	Excepciones

2.1	El sistema encuentra una sincronización anteriormente realizada y la recupera finalizando así el caso de uso.
2.2	El sistema obtiene que el repositorio ya esta sincronizado, informa al usuario al respecto y el caso de uso finaliza.
2.3	El sistema detecta que el usuario no es un administrador y el caso de uso finaliza.

CU desincronizar repositorio	
Pasos	Escenario general
1	El usuario utiliza el comando de desincronización del repositorio remoto.
2	El sistema verifica que exista una sincronización por parte del grupo y desincroniza el repositorio con la plataforma externa correspondiente.
Pasos	Excepciones
2.1	El sistema no encuentra una sincronización para el grupo y el caso de uso finaliza.
2.2	El sistema detecta que el usuario no es un administrador y el caso de uso finaliza.

CU Compresión de repositorio	
Pasos	Escenario general
1	El usuario utiliza el comando de obtención de un archivo comprimido en zip del repositorio.
2	El sistema comprime el directorio asociado al sistema de ficheros del repositorio y lo devuelve en un formato zip.
Pasos	Excepciones
2.1	El sistema no encuentra ningún archivo ni carpeta asociada al grupo y el caso de uso finaliza.

CU Obtener enlace de sincronización	
Pasos	Escenario general
1	El usuario utiliza el comando de obtención del enlace de sincronización del repositorio remoto.
2	El sistema verifica que existe una sincronización adecuada y proporciona el enlace de sincronización adecuado.
Pasos	Excepciones
2.1	El sistema no encuentra una sincronización activa para el repositorio local y el caso de uso finaliza.

CU Cambiar privacidad de interfaz gráfica	
Pasos	Escenario general
1	El usuario utiliza el comando de modificación de privacidad de la navegación del repositorio.

2	El sistema muestra las opciones de privacidad existentes y pide al usuario que elija una de ellas.
3	El usuario introduce una opción de privacidad determinada.
4	El sistema modifica la privacidad asociada al grupo correspondiente.
Pasos	Excepciones
4.1	El sistema no detecta la opción de privacidad introducida como válida y el caso de uso continúa por el paso 3.

4.3. Modelo de dominio

El diagrama de clases de la etapa de análisis del proyecto software a elaborar también se ha desarrollado con la herramienta de modelado Astah Professional, puesto que nos sirve para todo tipo de diagrama o modelo de diseño para una mejor implementación de la aplicación y viene definido por las siguientes clases mostradas a continuación:

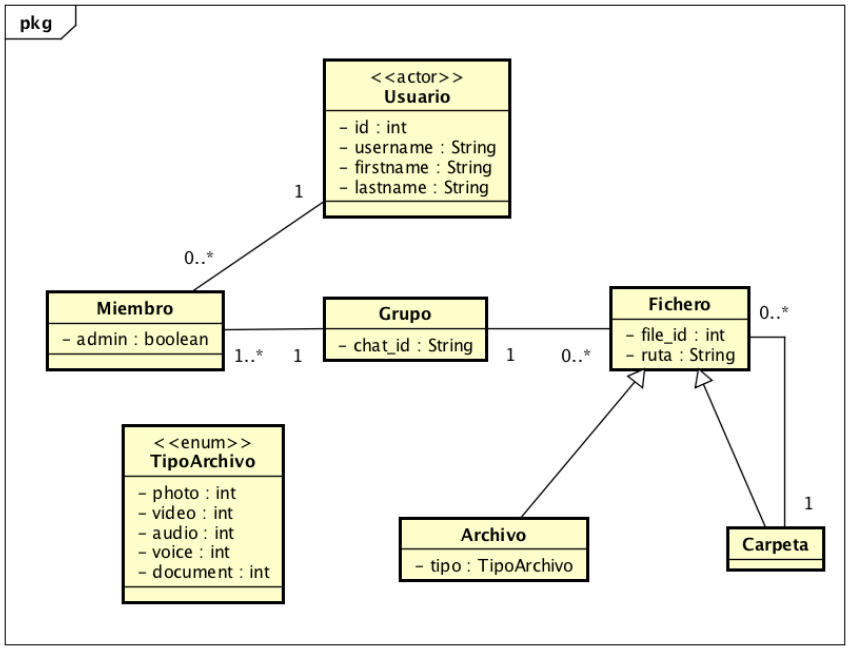


ILUSTRACIÓN 33 – MODELO DE DOMINIO DE ANÁLISIS

En la aplicación de Telegram, los grupos pueden contener muchos usuarios y cada uno de los usuarios pueden pertenecer a diferentes grupos o conversaciones, debido a esta interpretación en nuestro modelo estableceremos una nueva clase asociada para determinar relaciones coherentes en cuanto a una correcta técnica de modelado del dominio.

Nótese la herencia mostrada en la clase Fichero, en la cual se determina que tanto archivos como carpetas son ficheros como tal y la única diferencia entre ellos es que una carpeta o directorio posee a su vez carpetas o archivos dentro de la misma, mientras que un archivo solamente posee contenido de un cierto tipo (documento pdf, mp4, txt y otros formatos de archivo). A pesar de esto, debemos tener en cuenta la herencia en todo momento para verificar si un fichero es una carpeta o un archivo simple de cierto formato.

En Telegram se pueden enviar todo tipo de documentos y formatos de archivo, por ello necesitaremos preocuparnos por los tipos de archivos existentes y verificar que la aplicación funcionará cuando los usuarios nos proporcionen diferentes archivos para almacenarlos en los repositorios locales con sus sincronizaciones remotas concretas.

Hay que fijarse que realmente los archivos como tal no hacen falta ser implementados con su contenido, aunque si es necesario instanciarlos con sus rutas identificativas y sus identificadores de archivo asociados a los servidores de Telegram. Esto se debe a que las operaciones relacionadas con la aplicación que vamos a diseñar necesitarán interactuar con las rutas de los ficheros de los repositorios y ejercer las funcionalidades adecuadas tanto en el modelo de dominio como en la propia plataforma de alojamiento del bot.

Por otro lado, existe cierta normativa en cuanto a las caracterizaciones de los sistemas de ficheros, puesto que al estarnos basando en los sistemas Linux/GNU tendremos que tener en cuenta que en todo momento de creación o modificación de ficheros en un directorio nunca puede haber dos ficheros con el mismo nombre, independientemente de si el fichero es una carpeta o un archivo.

4.4. Diagramas de secuencia del sistema

A continuación, propondremos los diagramas de secuencia para cada uno de los casos de uso ya explicados anteriormente en su descripción de los anteriores apartados. Los diagramas de secuencia nos permiten especificar una implementación mediante las operaciones necesarias para los objetos y clases que se tendrán que utilizar en nuestro proyecto software.

El modelo de dominio expuesto anteriormente será modificado para la determinación de las operaciones de los diagramas de secuencias que elaboraremos en este apartado, puesto que algunas de las clases no serán utilizadas o bien serán implementadas de otra manera mucho más eficiente para la ejecución de los servicios del programa interactivo de Telegram.

Estas operaciones integradas en el diagrama de clases han sido diseñadas a lo largo de la elaboración de cada uno de los diagramas de secuencia del proyecto software, conforme a medida que hemos ido pensando los casos de uso y su modelización de clases y operaciones, nos hemos dado cuenta de nuevas formas y algoritmos eficientes para realizar de manera correcta toda esta funcionalidad cambiando los esquemas ya establecidos, pues al fin y al cabo la adaptación es una parte importante dentro del proceso iterativo del desarrollo software.

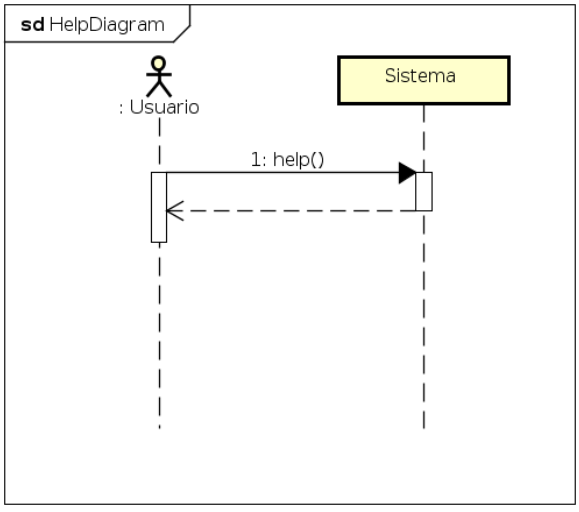


ILUSTRACIÓN 34 – DIAGRAMA DE SECUENCIA DEL COMANDO AYUDA

Comenzamos con la funcionalidad más simple de todas: la ayuda del bot (/help). Tanto el comando *help* como el comando *start* del programa realizan exactamente lo mismo y sirve para introducir el bot con toda la ayuda referente a la información de los comandos y de los términos más concretos que serán necesarios conocer para almacenar archivos en un sistema de ficheros.

En este caso de uso no existe ninguna llamada a otra operación de objeto de otra clase que no sea al controlador como tal del sistema, la funcionalidad del comando es simplemente envíar un mensaje a la conversación correspondiente de Telegram con todos los comandos del bot y una breve explicación de cada uno referente a su funcionalidad junto con los parámetros de entrada como argumentos.

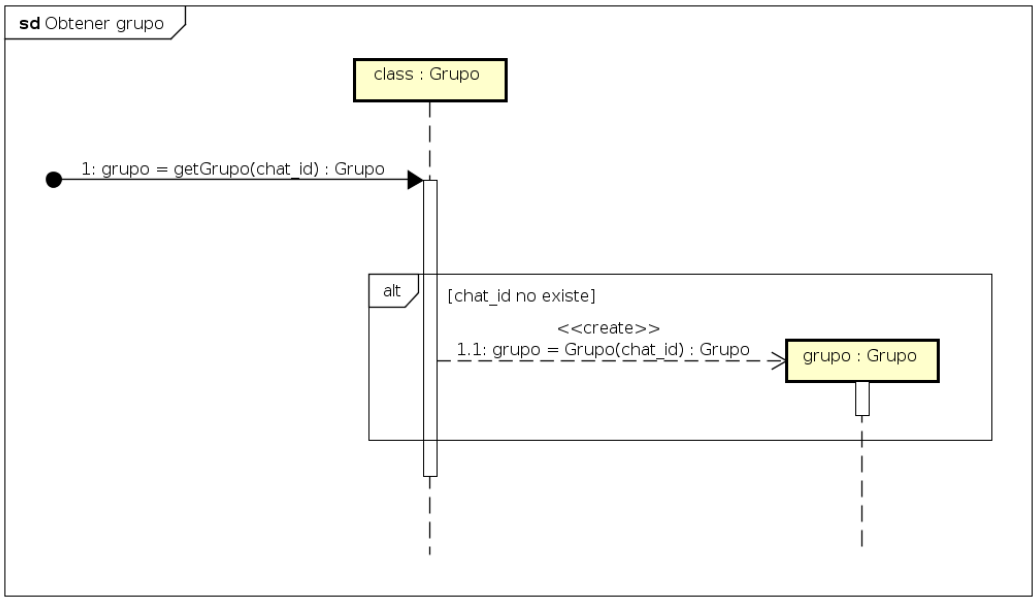


ILUSTRACIÓN 35 – DIAGRAMA DE SECUENCIA DE OBTENCIÓN DE GRUPO

Este diagrama de secuencia corresponde al conjunto de operaciones para la obtención del grupo de Telegram, lo hemos separado en un diagrama diferente debido a la reutilización que va a tener en el resto de casos de uso o servicios expuestos por la aplicación interactiva. Básicamente se trata de la búsqueda de un identificador de chat de grupo dentro del propio sistema para verificar si existe o no dado de alta el grupo en nuestro servidor de almacenamiento.

En el caso de que no existiera el grupo, se crearía de la manera correspondiente, pero no se daría de alta en el servidor como tal, es decir, solamente permanecería en el modelo de dominio de la aplicación sin necesidad de una persistencia en la propia plataforma donde se encuentra alojado el programa en ejecución. Esto se debe a que no necesitaremos persistir una carpeta asociada al repositorio hasta que no se hayan creado ficheros (archivos o directorios) en el propio sistema de ficheros, pero esto tiene que ver con otros comandos de creación de ficheros, así que este tema lo veremos más adelante durante la explicación del resto de diagramas de secuencia.

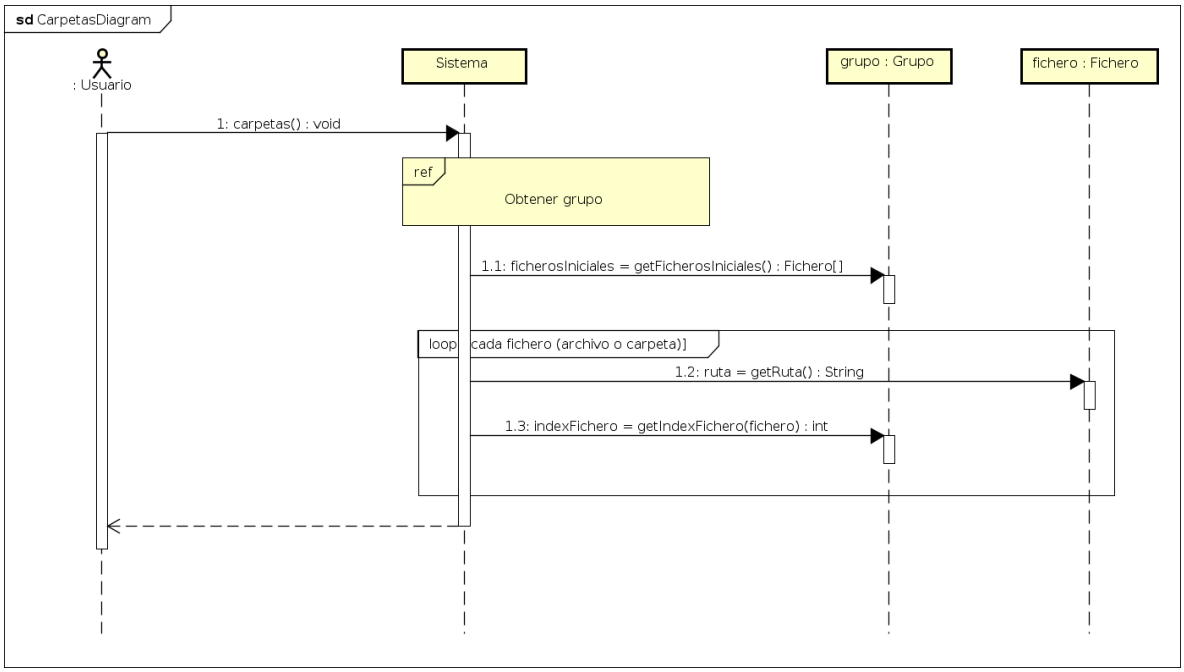


ILUSTRACIÓN 36 – DIAGRAMA DE SECUENCIA DEL COMANDO FILES

El siguiente diagrama de secuencia que vamos a exponer será para el comando de navegación (/files) o bien el caso de uso que muestra el repositorio del grupo con sus ficheros iniciales, ya sean archivos o directorios. Se establecerá una estructura de botones ordenados en filas dando a la posibilidad de navegar al usuario por la interfaz gráfica definida por el comando.

Este comando representa la navegación en una estructura de botones definida gracias a la API de Telegram, permitiendo así la implementación de una interfaz gráfica dentro de la propia aplicación para cada conversación concreta (privada o grupal). Nosotros aprovecharemos esto para mostrar los ficheros iniciales existentes en el directorio raíz del repositorio asociado a cada grupo.

Para ello, siempre que el usuario llame a este comando, se debe buscar el grupo concreto desde el cual está realizando el comando, esto lo conseguiremos con una llamada a la ejecución del diagrama de secuencia explicado anteriormente: Obtener Grupo. Una vez realizado esto y modelizado el grupo mediante la funcionalidad adecuada, se obtienen los ficheros iniciales del directorio raíz para mostrarlos en la estructura de botones que crearemos con la API.

El comando de navegación tiene dos escenarios de finalización posibles: uno para cuando el grupo no contiene ningún fichero en su repositorio asociado y otro para cuando si existe un sistema de ficheros almacenado. En ambos casos se determinará la estructura de visualización con las opciones de ajustes y de operación del sistema de ficheros, solo que cuando exista contenido de ficheros en el repositorio se mostrarán primero los botones asociados a esos ficheros de contenido.

La creación de botones y especificación de interfaz gráfica para su visualización no viene especificada dentro del diagrama de secuencia por dos razones: la primera es que son clases ya modeladas por la API de Telegram para Python (Python Telegram Bot) y la segunda se debe a que no corresponde con nuestro modelo de dominio en absoluto el conocer la implementación de esas clases de Telegram, ya que los diagramas de secuencia solamente están asociados a las clases y objetos de nuestro modelo de dominio de la aplicación que estamos elaborando.

No se producirá una sincronización remota de manera directa con ninguna aplicación para cada repositorio tras la realización de cualquier comando, esto se debe a que vamos a incorporar dos comandos a mayores para sincronizar y desincronizar los repositorios de los grupos

correspondientes donde decidan hacerlo. Se establecerá la opción de añadirlo o quitarlo en cualquier momento para cada conversación de Telegram que desee utilizar esta utilidad como servicio a mayores.

Más adelante, explicaremos más en detalle estos dos comandos de sincronización/desincronización con las plataformas remotas que pretendamos añadir a la aplicación que estamos construyendo en este proyecto, hasta entonces ningún comando establecerá una sincronización al finalizar a excepción de los relacionados con esa conexión remota.

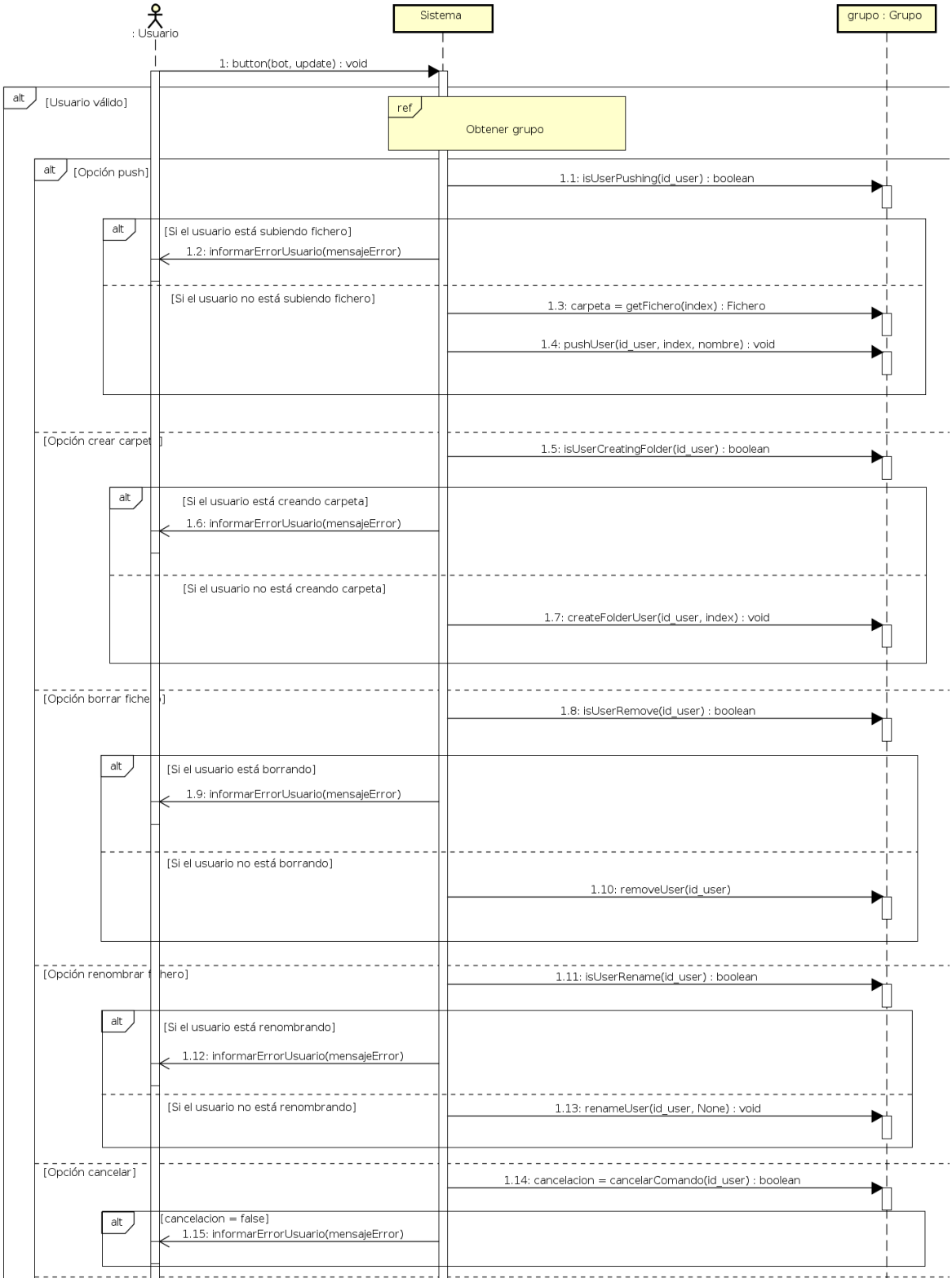


ILUSTRACIÓN 37 – DIAGRAMA DE SECUENCIA DE LA INTERFAZ GRÁFICA DE NAVEGACIÓN DE BOTONES I

Una vez ya explicado el comando de visualización de la interfaz gráfica del sistema de ficheros de un grupo (/files), solo queda establecer la funcionalidad de cada uno de los botones de esa estructura que se muestra en el comando de navegación de repositorio mencionado anteriormente.

Esto se llevará a cabo mediante un callback de retroalimentación para cada pulsación que lleve a cabo el usuario en cada botón de la navegación.

En la ilustración se puede observar la comprobación de cuál de todas las posibles opciones de ajustes ha seleccionado el usuario en la función de realimentación: subida de archivo, creación de carpeta, eliminación de fichero, renombrado de fichero o la cancelación de transacción de comando. Todos estos botones de servicio han sido implementados para establecer una nueva transacción en el usuario dependiendo del tipo de opción seleccionada por el mismo.

El botón de subida de archivo realiza exactamente la misma funcionalidad que el comando de subida de archivos (/push), solo que sin argumento de entrada y además la ruta en donde se va a almacenar el nuevo fichero a subir vendrá especificada por el directorio en donde se encuentre la interfaz gráfica de navegación de usuario.

Si el usuario pulsa en el botón de creación de nueva carpeta, el sistema pide al usuario la introducción de un nombre adecuado para la creación de una carpeta en el directorio en donde se encuentra la interfaz gráfica de navegación del repositorio. Esta es una alternativa que será considerada más sencilla que el comando de creación de carpetas (/mkdir), aunque no tan directa puesto que será necesario navegar al directorio en donde se quiere crear la carpeta antes de seleccionar esta opción de ajustes.

El botón de eliminar o borrar fichero pide al usuario en un mensaje de alerta privado que seleccione un fichero de la interfaz gráfica del repositorio, estableciendo así una nueva transacción de espera de selección de fichero para el usuario.

Los mensajes de alerta privados es una opción bastante interesante que nos proporciona la API de Telegram para cuando necesitamos avisar al usuario indicado sin necesidad de molestar al resto de miembros del grupo, se denominan de alerta porque pretenden captar la atención del usuario lo máximo posible mediante el mensaje privado.

En cuanto al botón asociado a la opción de renombrado de fichero, el sistema automáticamente informa al usuario en un mensaje privado que tiene que seleccionar un fichero (archivo o carpeta) para continuar con la transacción del comando.

La opción de cancelación de comando ofrece la misma funcionalidad que el comando de cancelación de transacciones (/cancel). Si el usuario posee varias transacciones acumulativas, la orden del comando de cancelación eliminará todo tipo de transacción que tenga pendiente la aplicación con respecto al mismo usuario de Telegram.

Las transacciones acumulativas son aquellas referentes cada una a diferentes comandos y casos de uso pero que pueden existir al mismo tiempo y en el mismo momento de ejecución de la aplicación para un usuario determinado. Para entender esto de manera sencilla, pondremos un ejemplo de transacciones acumulativas: creación de carpeta y movimiento de fichero a otra ruta, ambas transacciones se pueden ejecutar a la vez sin ningún inconveniente.

El problema vendría cuando existan dos transacciones de espera de mensaje por el usuario, puesto que no se podría distinguir que tipo de funcionalidad o caso de uso realizar primero y tampoco concretar a cuál de las transacciones corresponde ese mensaje de texto introducido por el usuario.

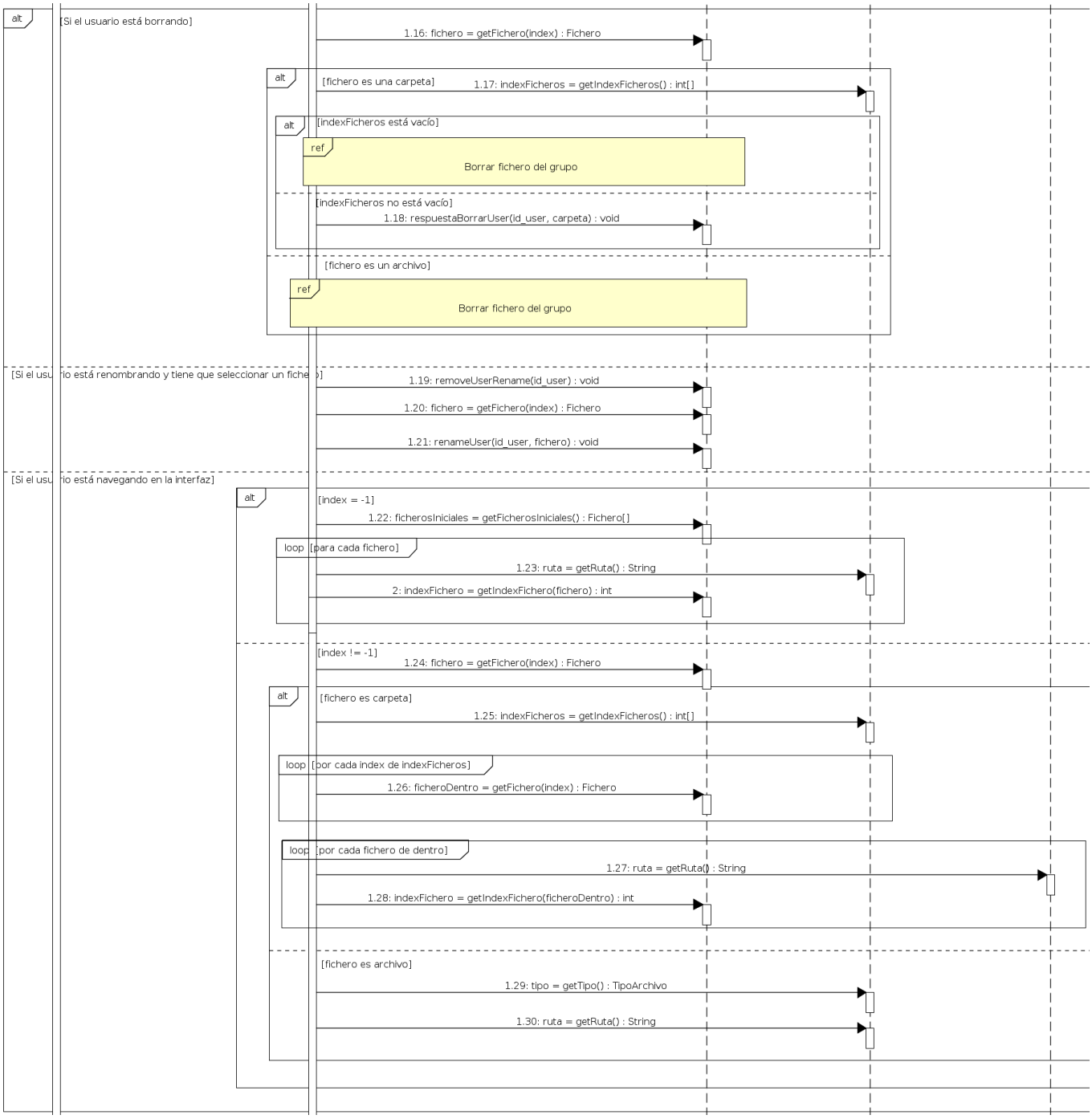


ILUSTRACIÓN 38 – DIAGRAMA DE SECUENCIA DE LA INTERFAZ GRÁFICA DE NAVEGACIÓN DE BOTONES II

Este diagrama de secuencia corresponde a las selecciones de ficheros por parte de los usuarios en la estructura de navegación del repositorio, de manera que se verificará antes si nos encontramos en alguna transacción de selección de fichero (borrado, renombrado, movimiento o copia) o bien simplemente el usuario ha hecho click en un fichero sin haber pulsado en una opción de ajustes anteriormente.

Si la opción de ajustes de borrado de fichero se encuentra activa y el usuario ha seleccionado un fichero concreto, el sistema comprueba si el fichero es un archivo o una carpeta, si es un archivo el sistema eliminará el archivo de manera directa, en caso contrario, se comprobará si la carpeta está vacía o no. De manera que, si la carpeta no contiene ningún tipo de fichero en su interior, el sistema elimina directamente el directorio sin ningún inconveniente e informa al usuario, de lo contrario si hubiera contenido en la carpeta, se iniciaría una transacción de espera de elección de respuesta de borrado para el usuario determinado.

Si la opción de ajustes de renombrado de fichero se encuentra activa y el usuario ha seleccionado un fichero a renombrar, el sistema pide al usuario la introducción de un nombre nuevo para el fichero seleccionado y de nuevo se inicia una transacción de espera de mensaje de usuario.

Si por el contrario no se tiene ningún tipo de transacción activa, significa que el usuario no ha seleccionado ninguna opción de ajuste antes y por tanto se debe mostrar el contenido del fichero, si éste es una carpeta, la interfaz gráfica de botones se modificará para mostrar los ficheros contenidos dentro de la misma, en el caso contrario, el sistema deberá enviar el archivo por la conversación de Telegram adecuada.

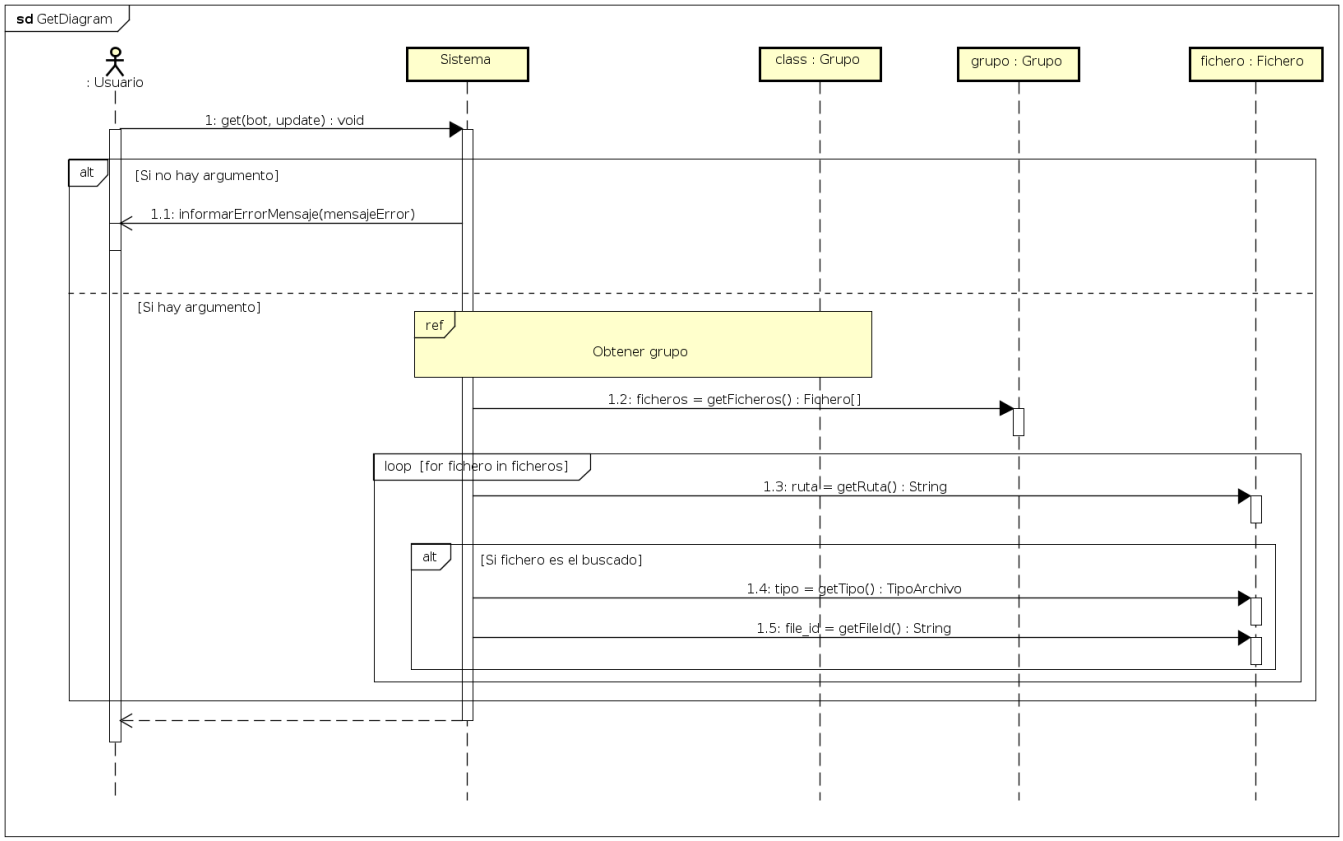


ILUSTRACIÓN 39 – DIAGRAMA DE SECUENCIA DEL COMANDO DE BÚSQUEDA DE ARCHIVOS

Pasamos al caso de uso de búsqueda de archivos (/get) dentro del repositorio, sirve principalmente para realizar búsquedas rápidas correspondientes a un patrón determinado sin necesidad de navegabilidad dentro del sistema de ficheros. La funcionalidad principal del comando es buscar de entre todos los archivos del repositorio del grupo, aquellos que concuerdan con un nombre especificado en el argumento del propio comando por el usuario.

El comando necesita un argumento de entrada como patrón de búsqueda para poder finalizar el caso de uso, en caso contrario el sistema proporcionará un mensaje de error al usuario por la conversación. En el caso de determinarse un nombre como patrón de búsqueda, el sistema accede a los ficheros que contiene el grupo o conversación modelada para realizar la búsqueda por medio de su ruta, quedándose así con los ficheros que sean coherentes o contengan ese nombre proporcionado como argumento.

Además, la API de Telegram permite la incorporación de ficheros locales en las conversaciones privadas y grupales de manera sencilla y sin necesidad de identificadores de archivos (file_id) para la localización de los mismos en la plataforma remota de los servidores de Telegram. Por lo que, la forma de envíar los mismos por medio del sistema de nuestra aplicación, será mediante la subida de

los mismos archivos que se encuentran en el servidor local donde almacenaremos los sistemas de ficheros asociados a los grupos que gestiona la aplicación.

A continuación, explicaremos dos diagramas de secuencia bastante importantes para nuestros casos de uso, son los asociados con la agregación/eliminación de ficheros dentro de los grupos de los chats de Telegram. Cada grupo contiene una tabla de ficheros para guardar en memoria los archivos y carpetas que se vayan creando mediante los comandos de creación de ficheros.

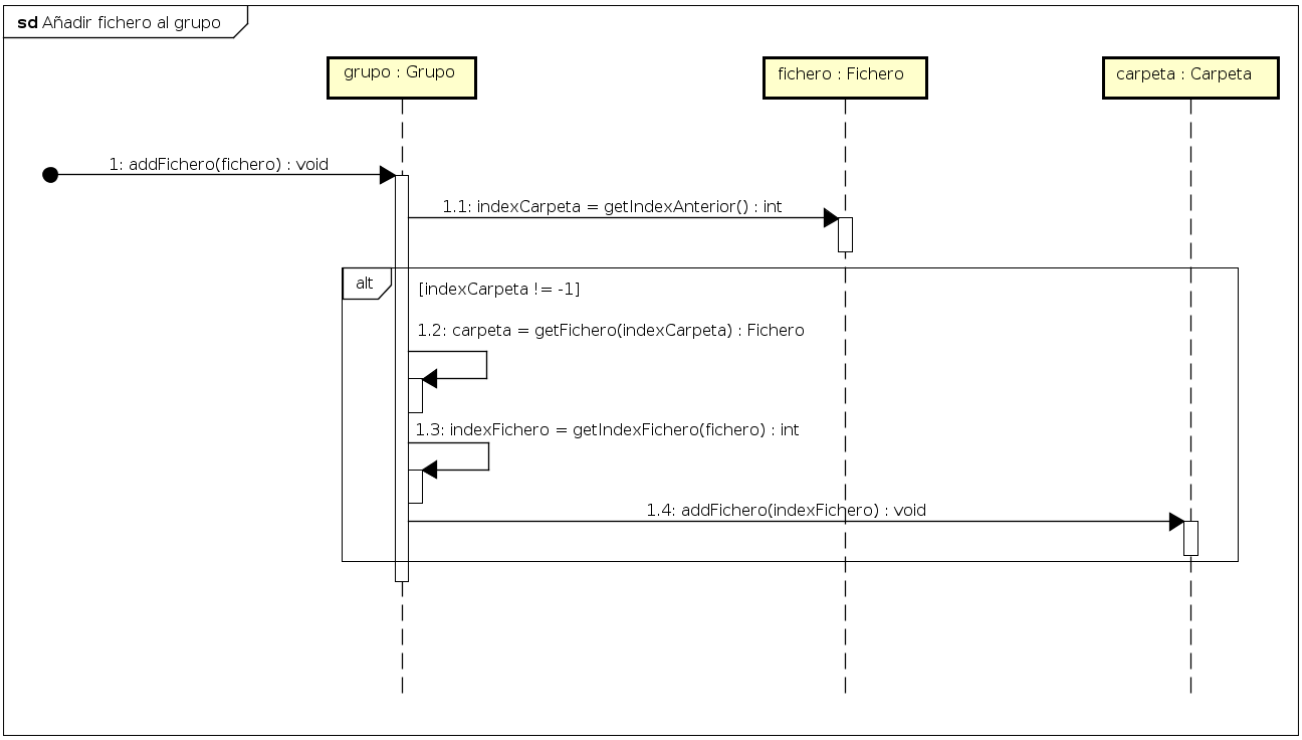


ILUSTRACIÓN 40 – DIAGRAMA DE SECUENCIA DE AGREGACIÓN DE FICHERO

En esta etapa de diseño de diagramas de secuencia para las funcionalidades, elaboraremos ciertos cambios en el modelo de dominio que hemos establecido anteriormente, puesto que el proceso iterativo mejora la estructura y organización del proyecto software a medida que nos acercamos a la implementación de la aplicación.

Uno de los cambios nuevos que se van a incorporar es la modificación de la relación de carpeta y el conjunto de ficheros en su contenido. No vamos a almacenar en cada carpeta una referencia al conjunto de ficheros como instancias, esto incrementaría la memoria del servidor de la aplicación en ejecución y la eficiencia en las búsquedas de ficheros no sería totalmente eficaz. Será mejor si solo almacenamos los índices asociados a sus lugares correspondientes de donde se encuentran esos ficheros de contenido en la tabla de archivos del grupo o conversación.

Lo mismo haremos con la referencia a la carpeta anterior donde se encuentra cada fichero procedente de las tablas de archivos de los grupos instanciados en el servidor, puesto que con el índice podemos acceder perfectamente al fichero determinado con la matriz de ficheros de manera directa sin necesidad de utilizar memoria extra.

En la agregación de un nuevo fichero al sistema de ficheros de un grupo o conversación privada, se creará primero en la tabla de archivos del repositorio adecuado y posteriormente, mediante el índice de la carpeta almacenado en el fichero nuevo añadido, accedemos al directorio específico y almacenamos en la lista de contenido el índice de referencia del fichero que acabamos de agregar al sistema de ficheros correspondiente.

Antes de agregar cualquier fichero con este método, es necesario crear el fichero e instanciarlo con los datos correspondientes como la ruta identificativa, el tipo de fichero o el índice de la carpeta en la cual se encuentra. De manera que la agregación del fichero a la carpeta y a la tabla de ficheros del repositorio se realiza cuando se produzca una llamada desde cualquier caso de uso a esta operación en concreto.

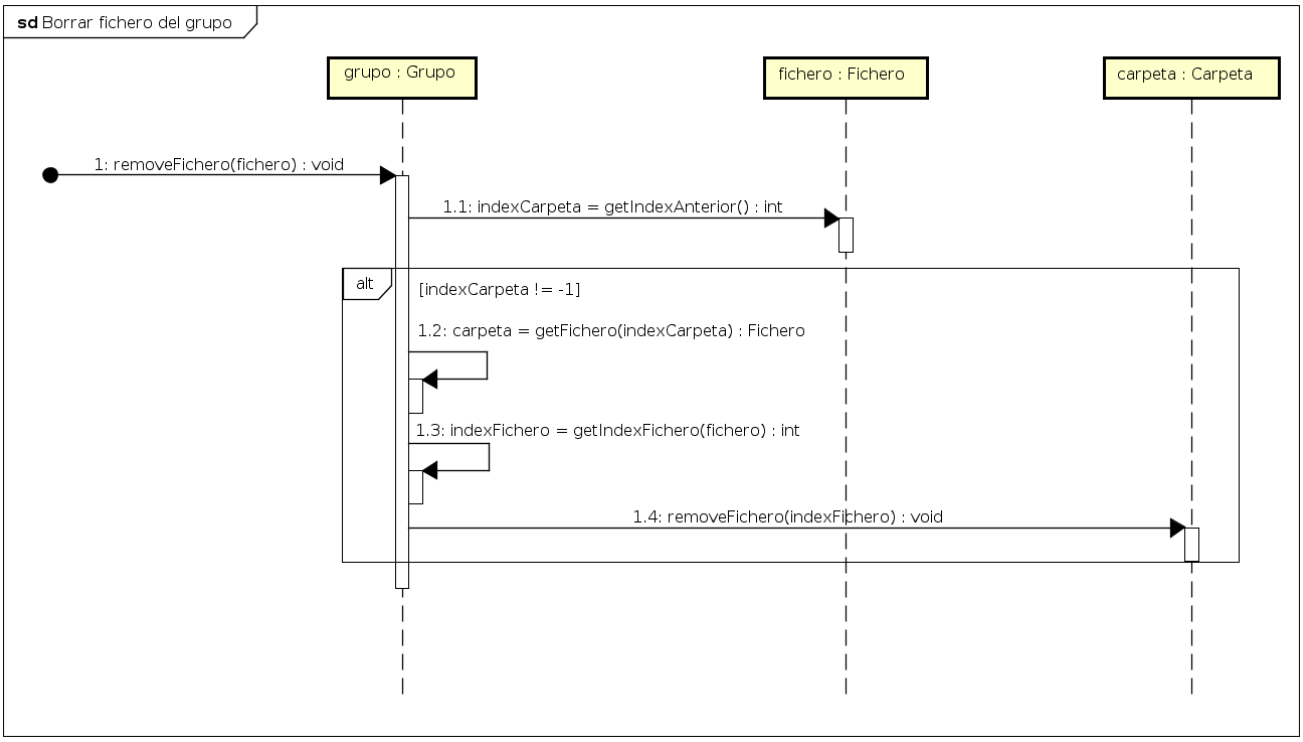


ILUSTRACIÓN 41 – DIAGRAMA DE SECUENCIA DE ELIMINACIÓN DE FICHERO

En cuanto a la eliminación de un fichero de un sistema de ficheros de un grupo o conversación privada de Telegram, se aplicará de la misma manera que en la agregación solo que eliminando el fichero de la tabla de archivos del sistema y también quitando el índice de referencia al fichero de la lista de contenido de la carpeta o directorio del mismo fichero.

Pero hay un factor muy importante a tener en cuenta en la eliminación de fichero, esto es que si existen carpetas con índices de contenido que referencien a ficheros por delante en la tabla de archivos del fichero a eliminar, habrá que ajustar todos estos índices puesto que, eliminar un fichero de una lista produce una disminución en uno de cada uno de los índices situados por delante del mismo.

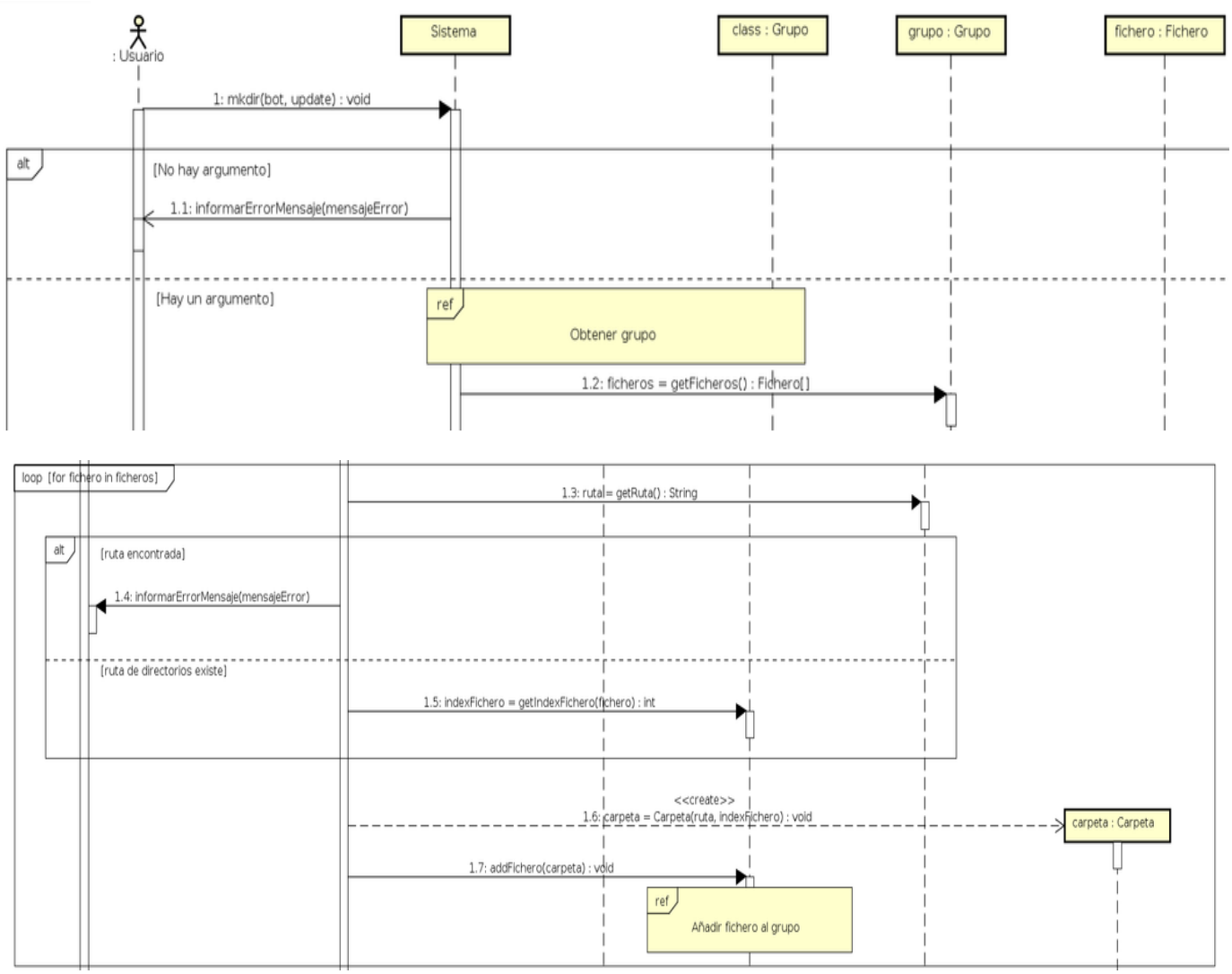


ILUSTRACIÓN 42 – DIAGRAMA DE SECUENCIA DEL COMANDO DE CREACIÓN DE CARPETA

Este comando (/mkdir) sirve para crear carpetas nuevas y como el comando de búsqueda de archivos, requiere de un argumento para poder ser ejecutado. Está basado en el comando *mkdir* de un sistema GNU/Linux en consola de intérprete, de ahí que necesite un argumento para indicar la ruta de creación de la carpeta o directorio.

El escenario que permite finalizar el caso de uso de manera correcta es cuando la ruta que se proporciona como argumento del comando no existe y por tanto el sistema puede proceder a la creación del directorio nuevo en esa ruta correspondiente. Por el contrario, si la ruta ya existe y se encuentra otro fichero que posea esa ruta acordada, el caso de uso quedará sin efecto y no se creará ninguna carpeta, informando así de la existencia de otro fichero con ese nombre.

En esta ilustración se puede visualizar la implementación mediante el recorrido de los ficheros del grupo para verificar que no existe ningún fichero con esa ruta independientemente de si es archivo o carpeta, puesto que no puede haber dos ficheros iguales dentro de una carpeta con la misma ruta. Sin embargo, si existe la posibilidad de crear ficheros con el mismo nombre pero en diferentes directorios, es decir, localizados en rutas diferentes del repositorio.

Si el escenario permite la creación de la carpeta o directorio nuevo en la ruta determinada, el sistema deberá crear una carpeta en el repositorio del grupo modelizado y del cual está llevando a cabo su gestión de almacenamiento de archivos.

Esto es algo que solo se realiza cuando se crean y almacenan nuevos ficheros (directorios o archivos) por medio de los servicios de la aplicación, ya que no es necesario persistir carpetas vacías de grupos que no quieren subir archivos al servidor dedicado.

Además, durante la búsqueda de que no exista otro fichero con la ruta definida para la nueva carpeta, se verifica a su vez que la ruta de directorios en donde se pretende crear la nueva carpeta exista dentro de la tabla de ficheros del repositorio en concreto. Si la ruta de directorios existe y además no existe ningún otro fichero con esa ruta proporcionada como argumento de entrada, el caso de uso finaliza de manera correcta y el sistema da de alta una nueva carpeta dentro del grupo.

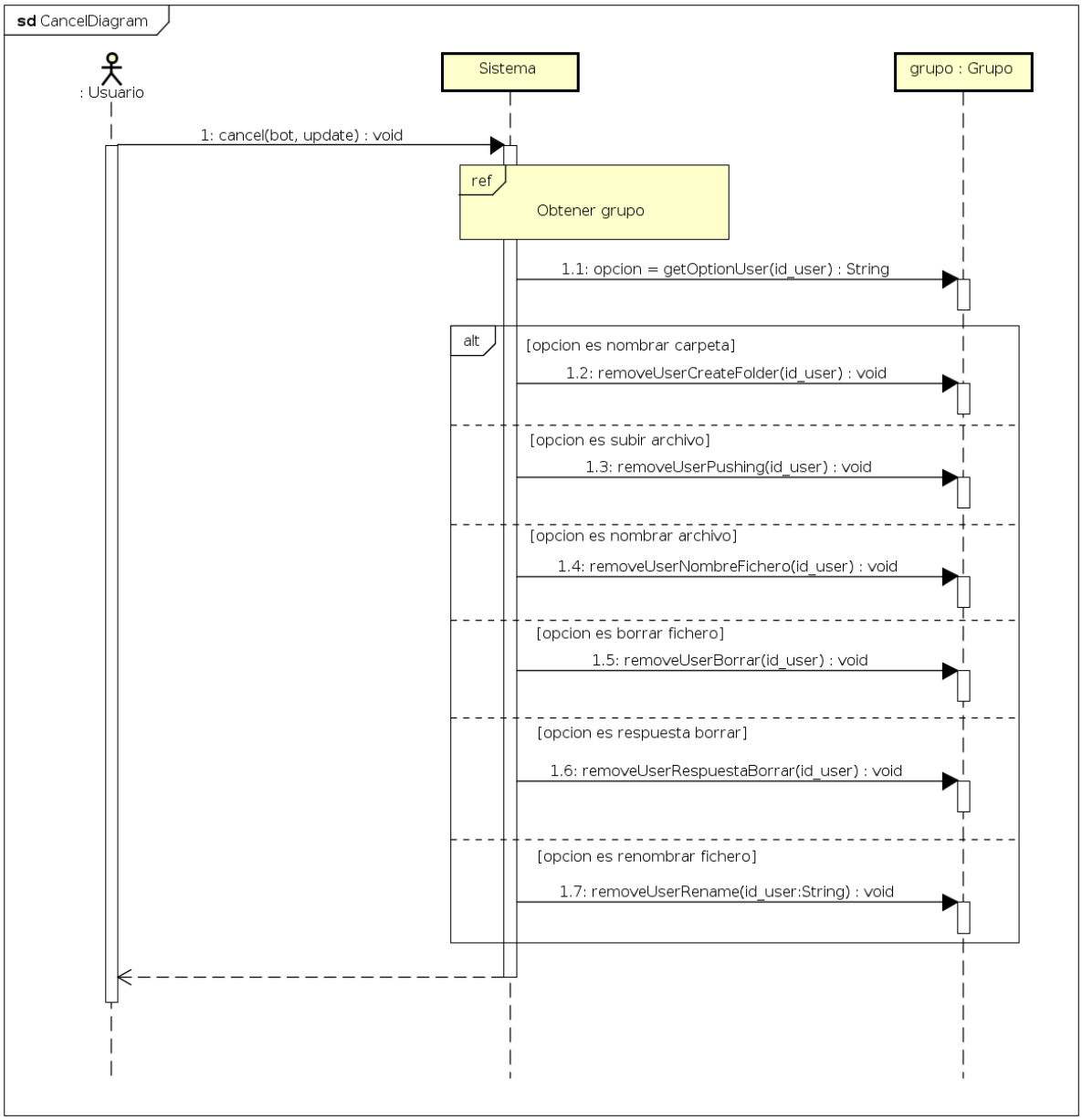


ILUSTRACIÓN 43 – DIAGRAMA DE SECUENCIA DEL COMANDO DE CANCELACIÓN

El diagrama de secuencia del comando de cancelación (/cancel) es muy simple, ya que el programa se dedica a verificar si el usuario está dentro de una transacción o procedimiento de espera para la ejecución de otro comando del bot. Las transacciones son esperas realizadas por el sistema para proseguir la ejecución de los servicios de la aplicación, pudiendo así finalizar el caso de uso por completo, ya sea de manera correcta o incorrecta.

Estos procesos de transacción simulan mecanismos de espera de respuesta por parte del usuario y se van a dar en los casos de uso siempre y cuando no se pueda realizar el comando en una sola interacción, por ello es necesario un comando de cancelación para todas estas transacciones pendientes y por tanto de los comandos de creación de carpetas, archivos, renombrado y borrado de los mismos.

En el caso de que las opciones de comandos no contengan al usuario como transacción de espera, el comando de cancelación informa al usuario de que no está realizando ninguna funcionalidad en concreto. Si por otro lado, el comando encuentra que si está en una de las opciones y transacciones de comandos, se anulará la funcionalidad del caso de uso de ese usuario en ese grupo, quedando así sin efecto el servicio específico.

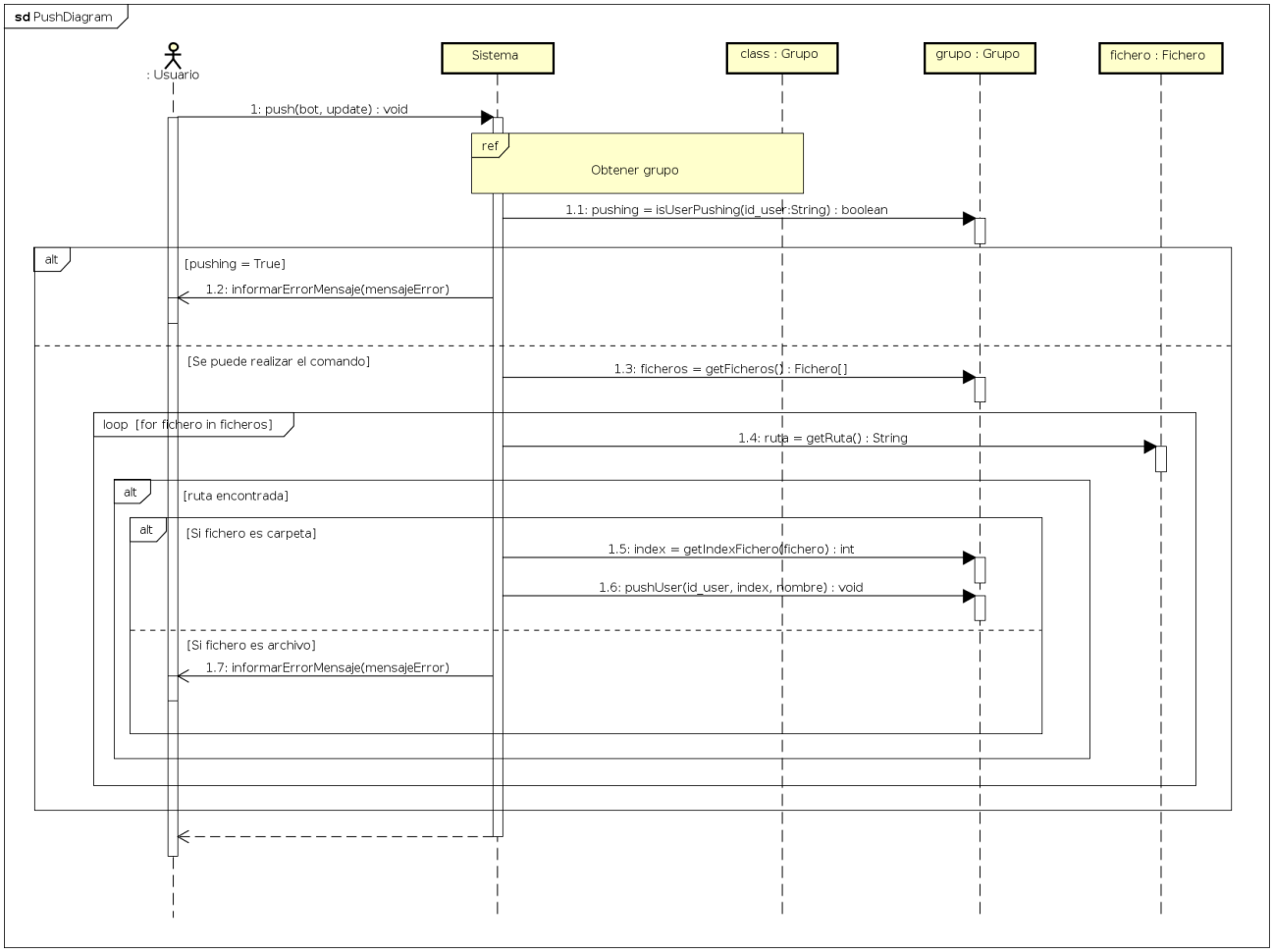


ILUSTRACIÓN 44 – DIAGRAMA DE SECUENCIA DEL COMANDO DE SUBIDA DE ARCHIVO

El comando de subida de archivo es un caso de uso algo especial, puesto que el usuario puede introducir un parámetro de entrada como argumento o no, dependiendo de si quiere especificar una ruta para almacenar el archivo o simplemente subirlo en el directorio raíz del repositorio concreto en donde aplica el comando. Lo hemos establecido de esta manera para darle más utilidad al comando y que así el usuario pueda elegir diferentes formas de uso.

Tras verificar y determinar el grupo de la conversación en la que se ha realizado el comando de usuario, el sistema comprueba si el usuario como tal está en una transacción de subida de archivo o bien si ya ha utilizado este comando anteriormente, en el caso de ser así el caso de uso finaliza y se informa con un mensaje de error al usuario en la conversación adecuada.

El sistema comienza por recorrer los ficheros del grupo pudiendo así ocurrir varias alternativas: Si la ruta corresponde con una carpeta o directorio, el sistema inicia el procedimiento de transacción con ese directorio; si se encuentra un archivo que corresponde con esa ruta, el sistema informa con un mensaje de error de creación de un archivo en esa ruta y el caso de uso queda sin efecto; si la ruta de directorios existe y no se encuentra ningún fichero con el nombre proporcionado, se inicia el proceso de transacción de espera para el archivo de usuario con esa ruta concreta; si la ruta de directorios no existe, el sistema informa de la inexistencia de la ruta de directorios especificada y el caso de uso queda sin efecto.

Al final nos importa construir las rutas en las cuales se van a almacenar los ficheros que se vayan a ir creando con los servicios que se ofrecerán en la aplicación, de manera que no importa la forma de entrada del argumento del comando, simplemente se requerirá de un procesamiento de ruta y de nombre determinado para asignárselo al fichero cuando el usuario lo pase por la conversación de chat.

Una vez que se haya realizado la comprobación y recorrido de los ficheros del grupo, se establecerá una transacción de espera de envío de archivo para el usuario que ha utilizado el comando, el sistema se quedará a la espera de un archivo que tiene que envíar el usuario para finalizar el caso de uso y poder crear el archivo en la ruta correspondiente del repositorio.

La transacción de subida de archivo contendrá los siguientes datos: el índice de la tabla de archivos de la carpeta en donde se creará el archivo enviado y el nombre que puede ser nulo o no, dependiendo de si se ha proporcionado en el argumento del comando o no. Esta información será necesaria para efectuar la subida y creación del archivo con los atributos correctos durante la misma transacción de espera de usuario.

El procesamiento de envío de archivo por parte del usuario no se lleva a cabo en este diagrama de secuencia y por tanto en este servicio de la aplicación, más adelante explicaremos este procedimiento asociado a la transacción de espera con su correspondiente implementación y funcionalidad de nuestro modelo de dominio.

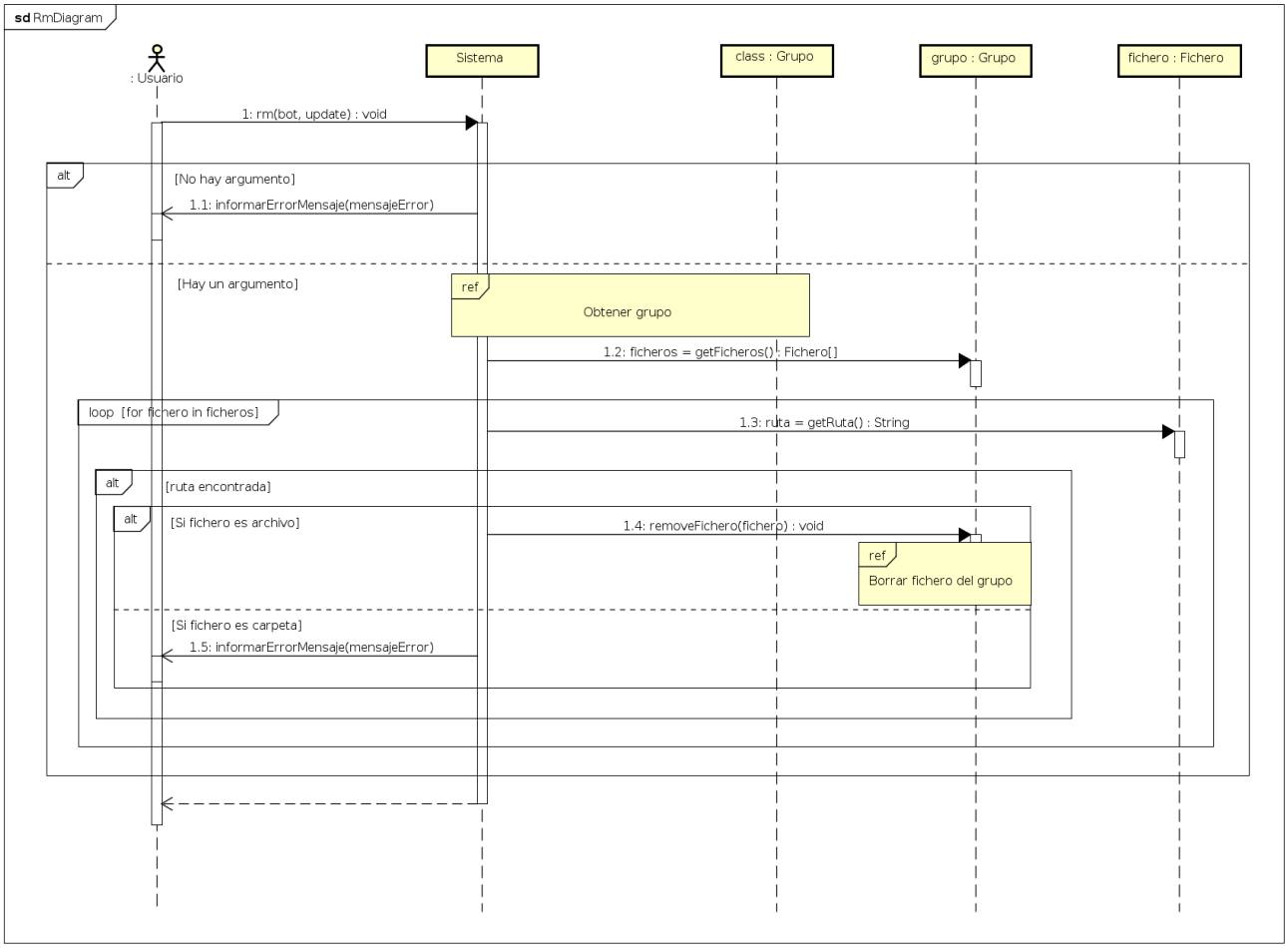


ILUSTRACIÓN 45 – DIAGRAMA DE SECUENCIA DEL COMANDO DE ELIMINACIÓN DE ARCHIVO

El comando de borrado de archivos (/rm) sirve solamente para borrar archivos y no directorios o carpetas dentro de los sistemas de ficheros, los hemos separado con dos casos de usos diferentes para hacer la funcionalidad semejante a la interfaz de línea de comandos de los sistemas operativos Unix que se encuentran separados de la misma manera.

En este comando, si es necesario indicar un argumento de entrada para indicar la ruta que el propio sistema debe eliminar dentro de los repositorios. La localización de rutas es un tema importante dentro de los repositorios, puesto que nos permite identificar los archivos que pretendemos aplicar operaciones en ellos con nuestros servicios.

La funcionalidad de este comando es bastante simple, el sistema comprueba que existe un archivo dentro del grupo cuya ruta identificativa sea igual a la proporcionada como argumento y en el caso de encontrarlo elimina el archivo del repositorio tanto físicamente (archivo persistente del servidor) como virtualmente (modelo del dominio). En el caso de que el sistema no encuentre el archivo que se pretende borrar del grupo o bien la ruta corresponde a una carpeta o directorio, el caso de uso quedará sin efecto informando con un mensaje de error correspondiente.

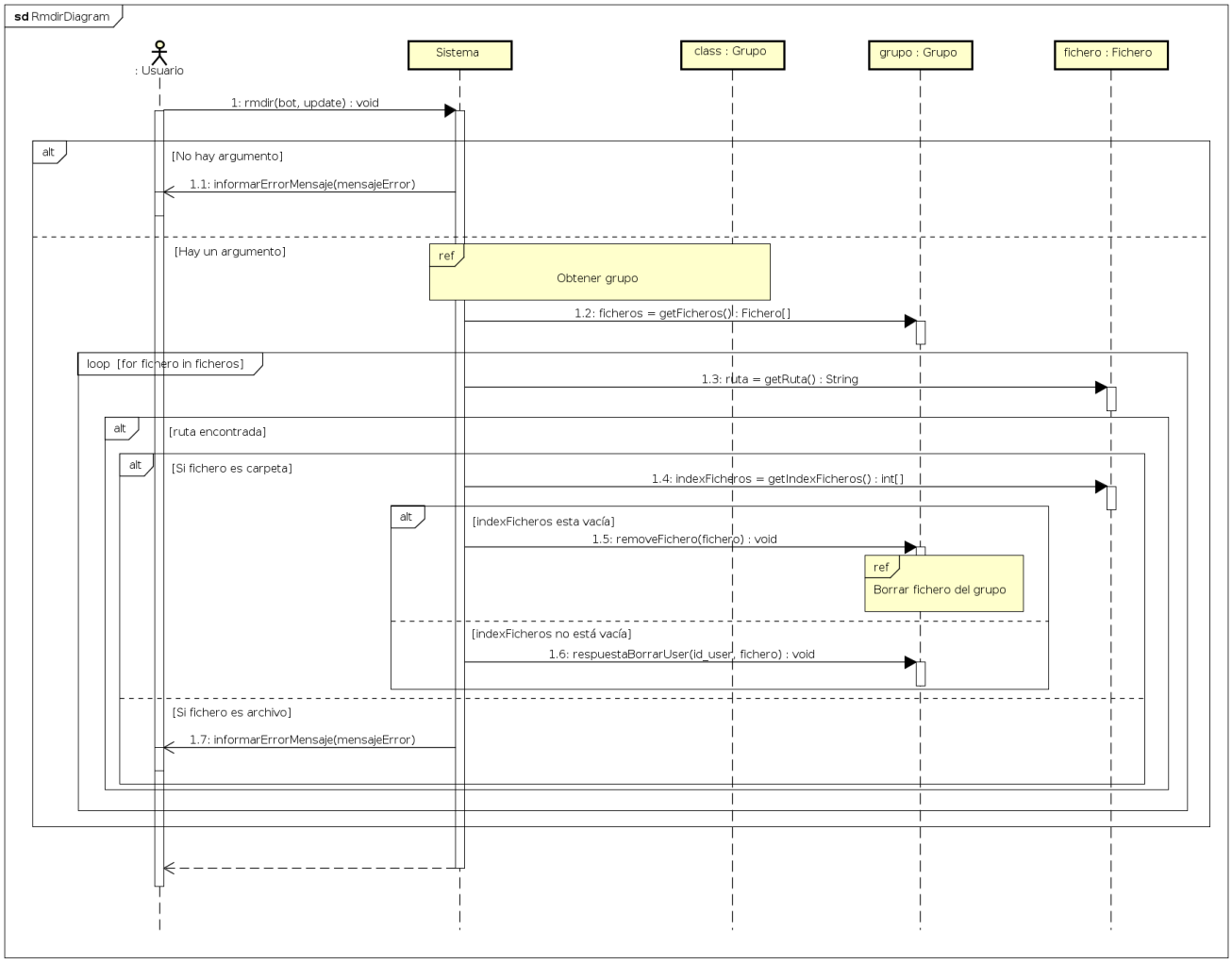


ILUSTRACIÓN 46 – DIAGRAMA DE SECUENCIA DEL COMANDO DE ELIMINACIÓN DE CARPETA

El comando de borrado de carpeta o directorio (/rmdir) sirve para eliminar carpetas del sistema de ficheros, de la misma manera que el comando de borrado de archivos que solamente servía para borrar archivos. Requiere de un argumento para localizar el directorio a eliminar y además se necesita acceder al grupo en el modelo de dominio para realizar las operaciones de comprobación de borrado.

Al encontrar el fichero determinado del grupo que identifica a la ruta proporcionada como directorio a borrar en el argumento del comando, se mira si esa carpeta posee contenido dentro de la misma, es decir, si existen ficheros contenidos dentro de esa carpeta, ya sean archivos u otras carpetas. La verificación del contenido de una carpeta se puede llevar a cabo mediante la lista de índices de la carpeta que corresponden a la posición de almacenamiento de los ficheros dentro de la tabla de ficheros del repositorio.

De manera que, si la lista de índices de ficheros que posee el contenido de la carpeta está vacía, entonces podemos eliminar la carpeta sin ningún inconveniente, sin embargo, si la lista no está vacía y por tanto el directorio contiene ficheros, tendremos que preguntar al usuario para ver si desea o no eliminar la carpeta con todos sus ficheros de manera recursiva.

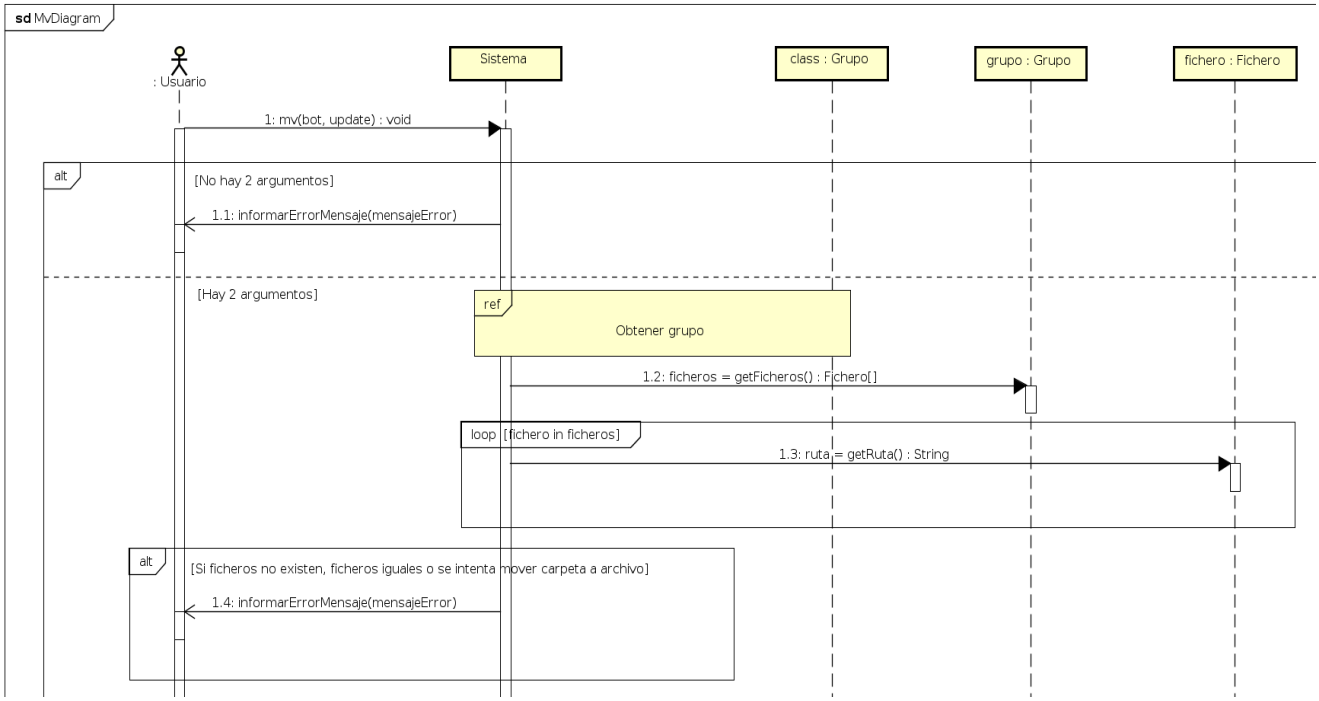


ILUSTRACIÓN 47 – DIAGRAMA DE SECUENCIA DEL COMANDO PARA MOVER FICHERO I

Comenzaremos a explicar de manera detallada el funcionamiento del comando mover fichero (/mv), en un principio el comando debe verificar si el usuario ha introducido dos argumentos de entrada en su llamada, en el caso de que no se dispongan dos parámetros como entrada del comando, el sistema informará con un mensaje de error al respecto y se dispondrá de la ayuda que necesite el usuario para su comprensión correcta del uso del comando.

Tras la comprobación de los dos argumentos de entrada, el sistema obtiene el grupo específico y se realiza un recorrido por los ficheros correspondientes de ese grupo para localizar las rutas que se han proporcionado como parámetros de argumento, almacenando así en memoria los ficheros de origen y de destino que necesitaremos para ejecutar las siguientes operaciónes del comando.

En el caso de que el fichero origen y el fichero destino no existan, los ficheros origen y destino son iguales o bien el fichero origen es una carpeta y el fichero destino es un archivo, el sistema finaliza el caso de uso con un mensaje de error al respecto. Cualquier sistema operativo y aplicación no debe permitir que un directorio intente moverse a un archivo como tal, puesto que un archivo no es una carpeta y no puede albergar otros ficheros como contenido.

Tampoco tendría sentido mover un mismo fichero de una ruta determinada a la misma como tal, puesto que entonces no existiría ningún movimiento de fichero dentro del propio sistema de ficheros y el comando no ejercería ningún tipo de servicio en la aplicación.

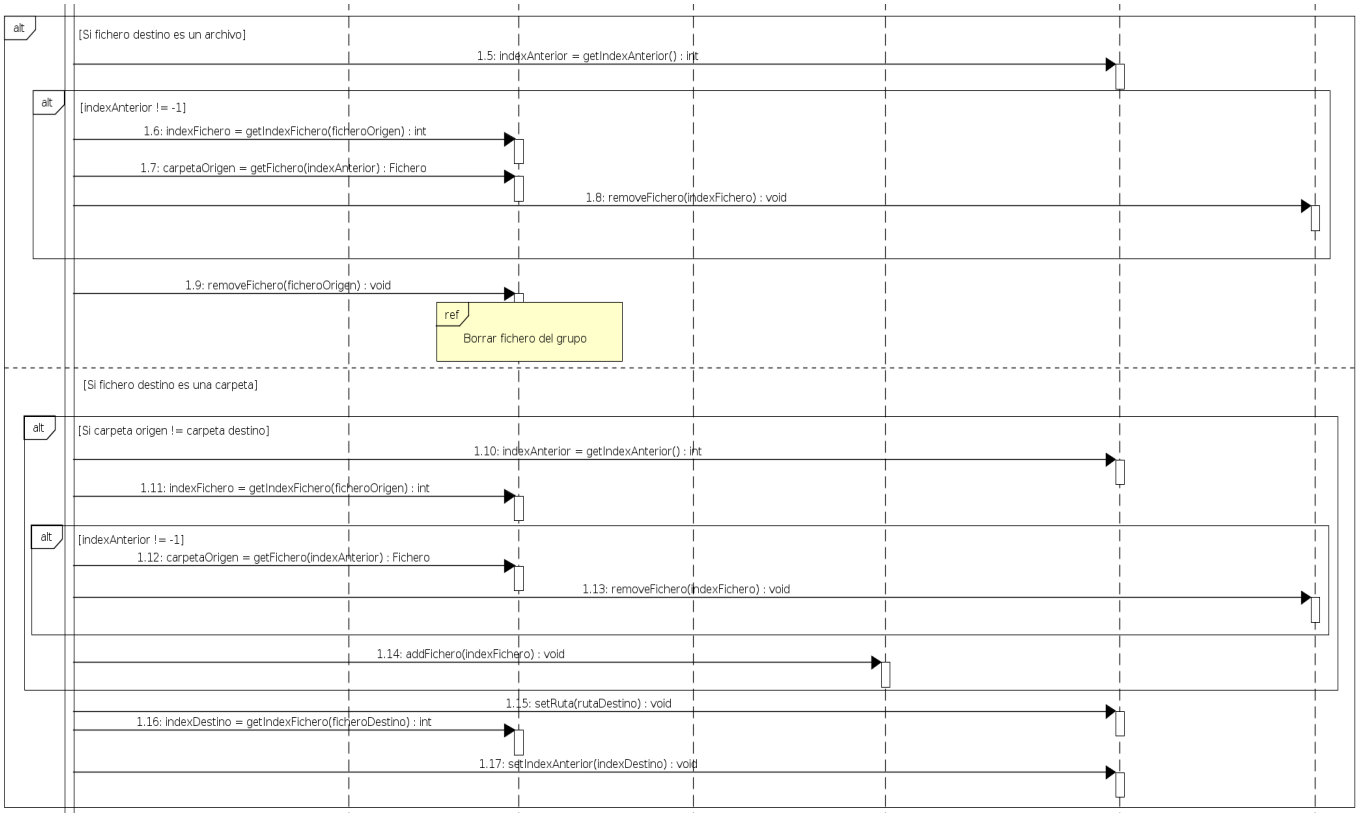


ILUSTRACIÓN 48 – DIAGRAMA DE SECUENCIA DEL COMANDO PARA MOVER FICHERO II

Tras la ejecución del conjunto de operaciones de comprobación de errores, el sistema ejercerá la verdadera funcionalidad del comando mediante el movimiento del fichero a su ruta correspondiente. Un dato importante a tener en cuenta es la de que con este comando se pueden renombrar los ficheros incluso aunque no se produzca ningún movimiento de ruta.

Esto se debe a que estamos desarrollando el comando para que se asemeje al comando *mv* del sistema operativo GNU/Linux o de cualquier otro referente a la familia de este sistema operativo. Por tanto, el comando permitirá tanto renombrado de ficheros como movimiento de archivos y carpetas.

De manera que aquí tenemos dos posibilidades con respecto a la modificación del modelo de dominio de la aplicación, si el fichero destino es un archivo, se va a borrar la instancia correspondiente al fichero origen almacenado y su referencia a la carpeta en donde se encontraba quedándonos así solamente con el fichero destino, si por el contrario el fichero destino es una carpeta, habrá que ver si el fichero origen se intenta mover a otra carpeta o bien pretende ser simplemente renombrado, así como modificar las referencias del fichero origen para que se encuentre en la carpeta nueva indicada y modificar su ruta con el nombre que se le haya proporcionado.

Este comando sin duda es el más complejo de todos los existentes en la aplicación debido a la libertad de utilización que se ofrece dentro del mismo, puesto que sirve para multitud de funcionalidades: renombrar ficheros moviéndolos a otras carpetas, sobreescribir archivos situados en la misma o en diferentes carpetas, movimiento de carpetas dentro de otras carpetas, etc…

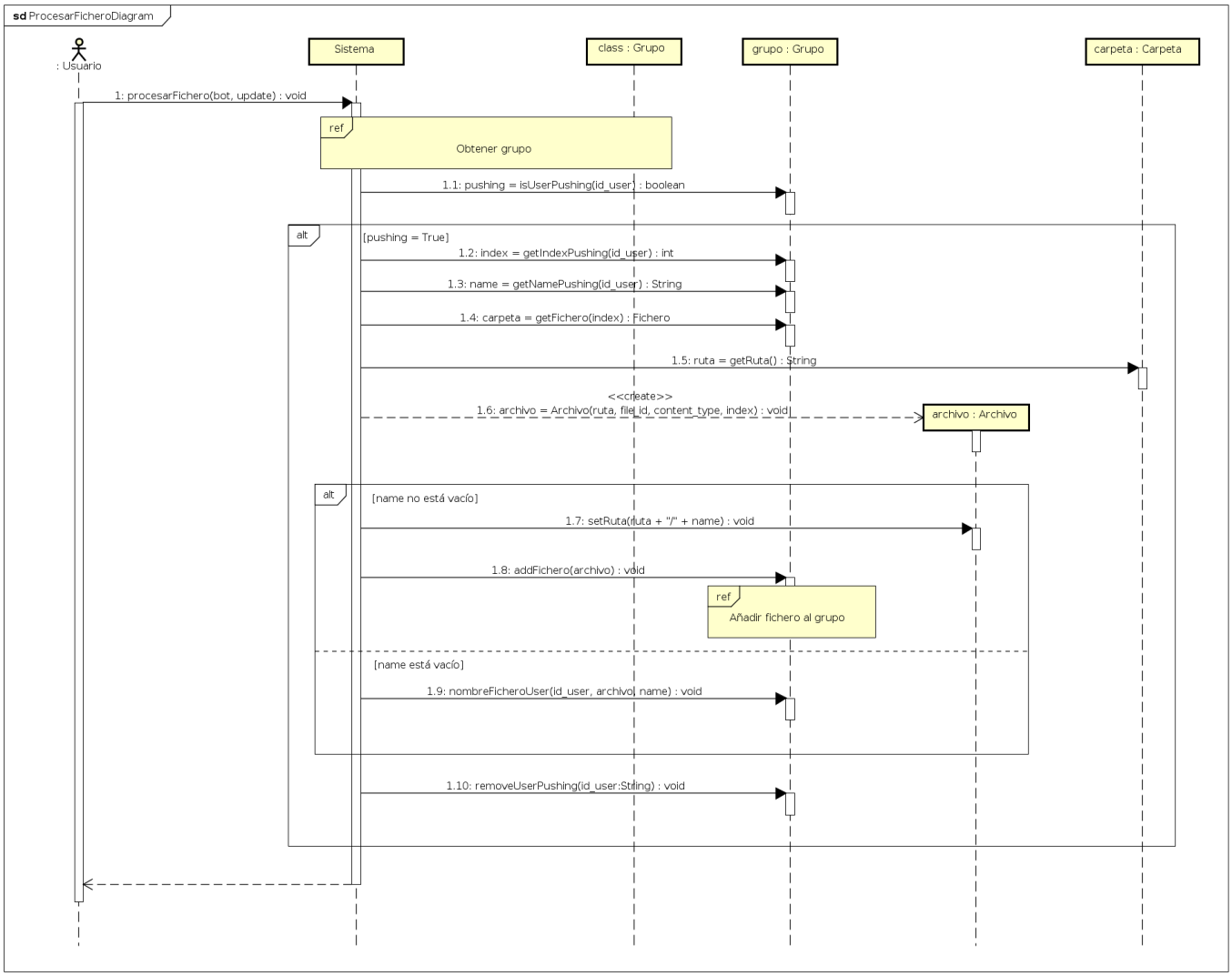


ILUSTRACIÓN 49 – DIAGRAMA DE SECUENCIA DE TRANSACCIÓN DE SUBIDA DE ARCHIVO

Este diagrama de secuencia corresponde a la implementación de las operaciones asociadas al procesamiento de ficheros que se envían por la conversación de Telegram siempre y cuando los usuarios se encuentren en transacciones de subida de archivos. Para que se produzcan transacciones de espera de envío de ficheros solamente se pueden dar con el comando de subida de archivos (/push) o bien con el botón de la interfaz gráfica de subida de archivos que se proporciona con el comando de visualización del repositorio (/files).

El procesamiento de archivos por parte de la aplicación solo se realizará cuando el usuario tenga una transacción pendiente, es decir, si el usuario envía cualquier tipo de documento u fichero sin comunicarse antes con el bot para su subida al sistema de ficheros, la aplicación no tendrá en cuenta ningún tipo de archivo o documento que envíe por el chat.

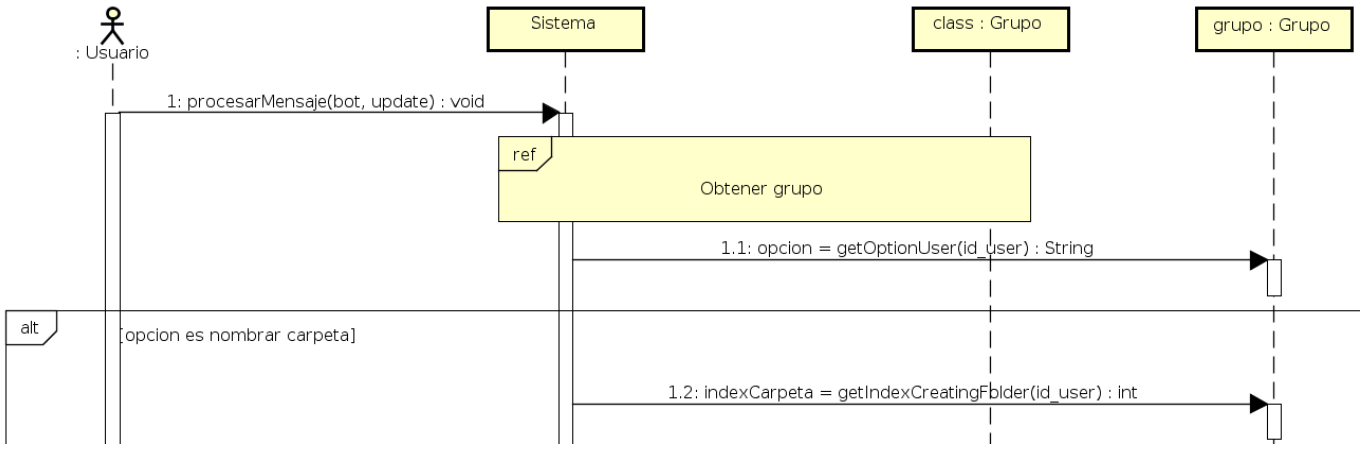
Cuando el usuario envía un archivo por el chat, se distinguirá primero el tipo de archivo (documento, foto, vídeo, audio o nota de voz) para su posterior almacenamiento en el modelo de dominio y su persistencia de descarga mediante la API de Telegram en el repositorio del grupo en concreto, posteriormente se verificará si ya existe un nombre especificado para el archivo, ya sea proporcionado antes como argumento del comando de subida de archivo o bien con la transferencia del archivo en la conversación de Telegram y se comprobará que no existe otro fichero igual en esa ruta de directorios concreta.

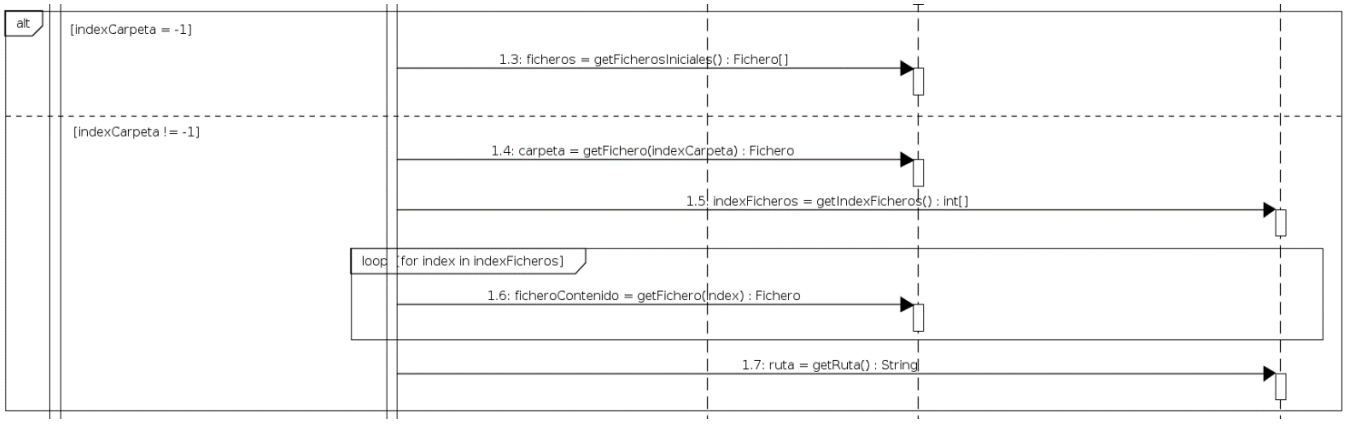
En el caso de que el archivo no contenga un nombre asignado o bien el nombre elegido no es posible, la aplicación dará comienzo a una nueva transacción de espera de introducción de un nombre por parte del usuario, este procedimiento de espera será explicado más adelante en el diagrama de secuencia asociado al tipo de transacción adecuada.

En algunos ficheros es posible agregar un comentario durante el envío en Telegram, esto nos permitirá cambiar el nombre en el caso de que nos hayamos equivocado antes o bien se haya decidido cambiar el nombre en el momento del envío después de ejecutar el comando de subida (/push).

Conviene destacar en el diagrama de secuencia de esta operación de espera de transferencia de archivo, que no se descargará y almacenará el archivo que se ha pasado por la conversación de Telegram hasta que no se haya determinado un nombre para el nuevo archivo, el nombre puede determinarse de muchas maneras como ya hemos explicado antes, pero no se añadirá ni al sistema de ficheros del grupo virtual ni físico hasta identificar al archivo con una ruta específica.

Ahora procedemos a explicar los diagramas de secuencia correspondientes a los procedimientos de espera de mensajes de texto por parte del usuario, también denominadas transacciones de procesamiento de mensajes que se encuentran en varios casos de uso que hemos explicado anteriormente en el modelado de nuestra aplicación.





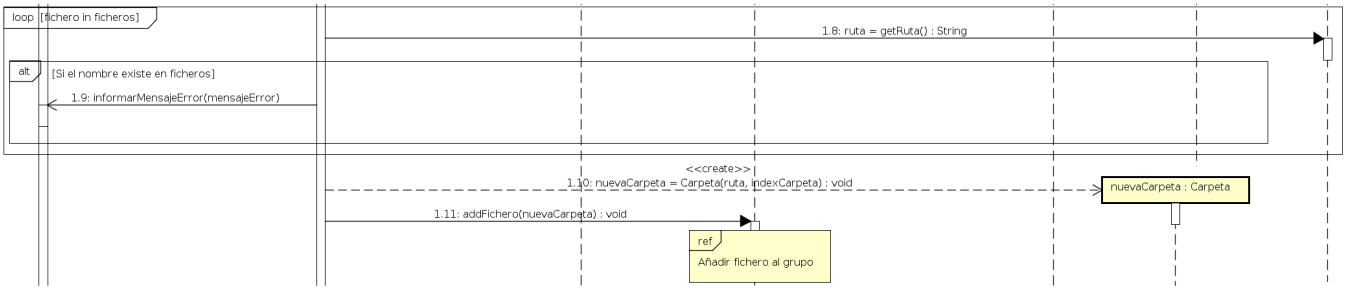


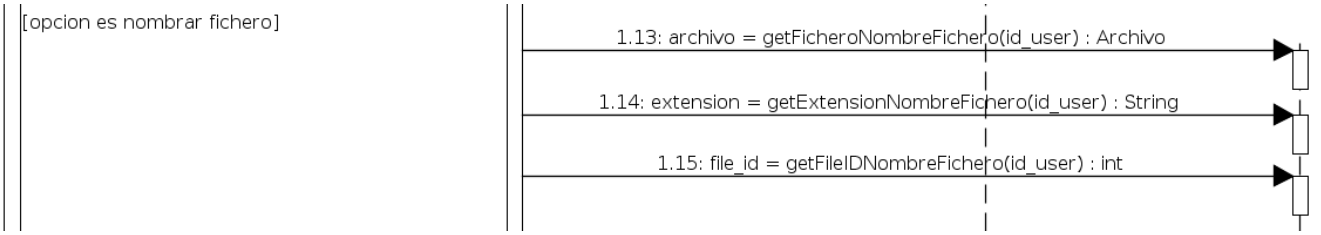
ILUSTRACIÓN 50 – DIAGRAMA DE SECUENCIA DE TRANSACCIÓN DE CREACIÓN DE CARPETA

Esta ilustración corresponde al diagrama de secuencia inicial de procesamiento de mensajes de usuario del sistema, el cual solamente ofrece funcionalidad cuando existan transacciones de espera para los usuarios que hayan ejecutado los comandos de la aplicación. El sistema ejercerá una funcionalidad u otra en función de la opción de transacción en la cual se encuentre el usuario que envíe el mensaje de texto.

Si existe una opción de transacción, se estudia qué tipo de opción se está llevando a cabo como procedimiento de espera y se ejecutan las operaciones necesarias para ofrecer el servicio con el máximo rendimiento posible. La primera transacción de mensaje de usuario que vamos a explicar será la procedente del botón de creación de carpeta dentro de la interfaz gráfica de navegación (/files).

En esta transacción, el usuario introduce un nombre para la nueva carpeta que se quiere crear, de manera que el sistema comprueba que en el directorio donde se pretende crear la carpeta no exista un fichero con ese nombre proporcionado y tras realizar la comprobación, se da de alta una nueva carpeta finalizándose así el caso de uso de creación de carpeta. En caso de que se encuentre un fichero dentro de la carpeta que contenga la misma ruta y por tanto el mismo nombre determinado, se informará con un mensaje de error al respecto de la imposibilidad de la creación del directorio en ese lugar.

Sin embargo, el comando de creación de carpetas (/mkdir) es un comando directo que no emplea ningún tipo de transacción de espera de mensaje ni ningún procedimiento de comunicación con el usuario, esto se debe a que, en solo una única interacción, la aplicación puede crear la carpeta y darla de alta en el sistema puesto que se ha proporcionado la ruta de creación como argumento del comando.



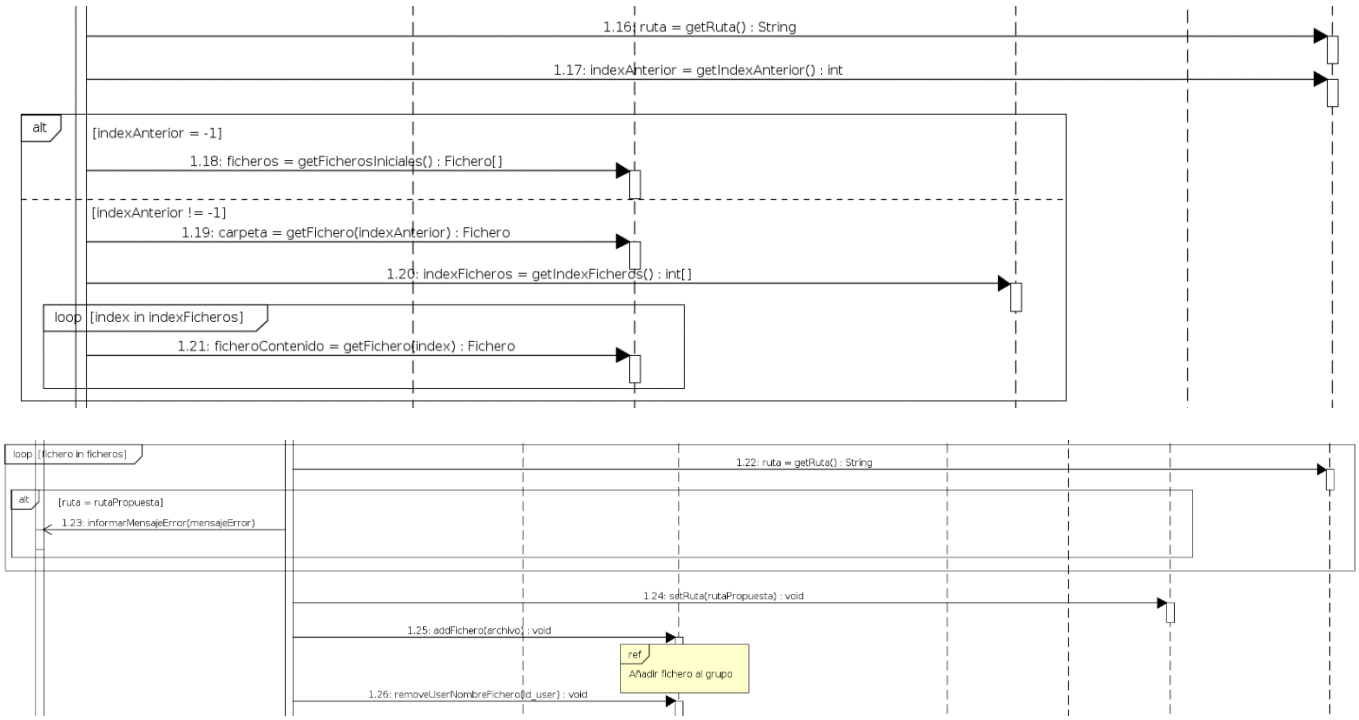


ILUSTRACIÓN 51 – DIAGRAMA DE SECUENCIA DE TRANSACCIÓN DE ASIGNACIÓN DE NOMBRE DE ARCHIVO

En esta ilustración se muestra la opción de transacción de asignación de nombre a un nuevo archivo cuando no se ha proporcionado un nombre anteriormente, esto es cuando hemos utilizado el comando de subida de archivos (/push) y no hemos especificado un nombre tras enviar un fichero por la conversación de Telegram.

Por ello existe esta transacción de nombramiento de archivo, ya que necesitaremos proporcionar un nombre para identificar una ruta única para el archivo nuevo en nuestro sistema. Las operaciones que se muestran en el diagrama de secuencia son las básicas asociadas a la comprobación de nombre dentro de la tabla de ficheros que existe dentro del grupo determinado. De la misma manera que si se encuentra un fichero con el nombre proporcionado por el usuario en esta transacción, la aplicación manda un mensaje de error hasta que el usuario introduzca un nombre disponible para el archivo.

Un dato importante a tener en cuenta aquí es la finalización del diagrama de secuencia en cuanto al establecimiento de la nueva ruta con el nombre proporcionado por el usuario para el archivo que ya venía incluido en la transacción actual a partir de la transacción anterior de espera de archivo. El archivo no se crea cuando se proporcione el nombre, ya estaba creado de antes cuando se ha enviado el formato de archivo por la conversación de Telegram, de manera que no hace falta volverlo a crear ni nada por el estilo. Simplemente se almacena dentro del grupo para su utilización y su correcto establecimiento en esta última transacción de creación de archivo.

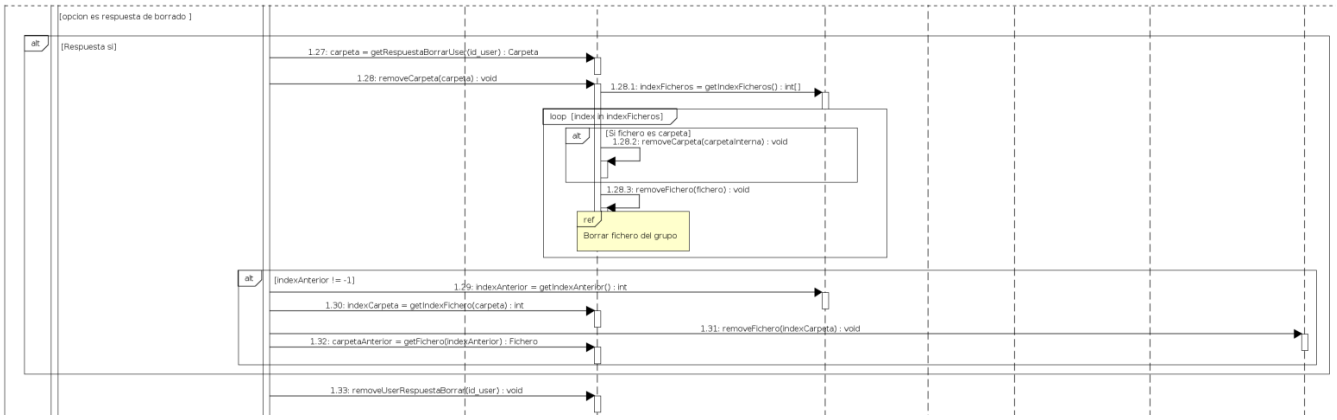


ILUSTRACIÓN 52 – DIAGRAMA DE SECUENCIA DE TRANSACCIÓN DE BORRADO DE CARPETA CON CONTENIDO

Esta transacción representa el procedimiento de borrado de una carpeta o directorio con contenido de ficheros en su interior, recordamos que al intentar eliminar una carpeta con contenido dentro, la aplicación preguntaba al usuario si deseaba borrar la carpeta y su contenido entero. De manera que dependiendo de la respuesta que eligiera el usuario, se va a eliminar o no la carpeta con su contenido tal y como expresa el diagrama de secuencia en esta ilustración.

Si el usuario indica que sí, el sistema utilizará un método del grupo de eliminación recursiva de la carpeta borrando así todo su contenido, la recusividad afecta a las carpetas que se encuentran dentro de esta carpeta y así sucesivamente hasta llegar a los directorios que solamente contienen archivos simples o bien se encuentran vacíos sin contenido alguno.

Además, no solo hay que eliminar de manera virtual los archivos y carpetas que se encuentran en ese directorio, si no que también tendremos que borrar del sistema de almacenamiento físico del servidor esos archivos y por tanto el contenido mismo de la carpeta del repositorio. También hace falta eliminar la referencia de la carpeta que estamos borrando como tal de su directorio anterior, es decir, el índice de la lista de ficheros del directorio o carpeta a eliminar que se encontraba como contenido dentro de su carpeta anterior.

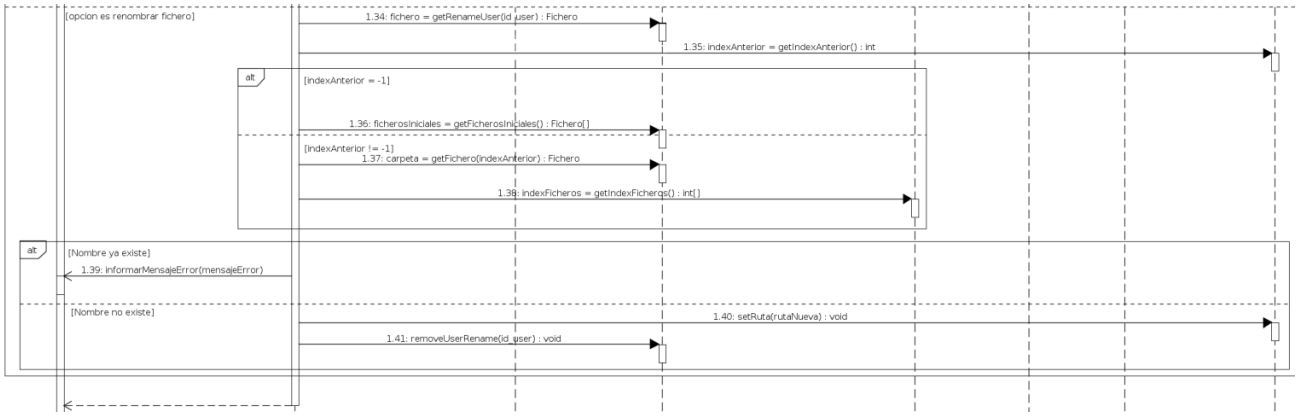


ILUSTRACIÓN 53 – DIAGRAMA DE SECUENCIA DE TRANSACCIÓN DE RENOMBRADO DE FICHERO

La funcionalidad de la espera de transacción de renombrado de fichero es sencilla, puesto que solo queremos cambiar el nombre de un fichero dentro del mismo directorio, simplemente habrá que comprobar si existe otro fichero con el nuevo nombre introducido por el usuario dentro del directorio específico y en el caso de ser así, se informa con un mensaje de error concreto para que el usuario elija otro nombre posible.

Si no hay ningún fichero con ese nombre propuesto, se modifica la ruta del fichero determinado (ya sea archivo o carpeta) utilizando los “setters” específicos para la modelización del nuevo fichero con el renombrado aplicado.

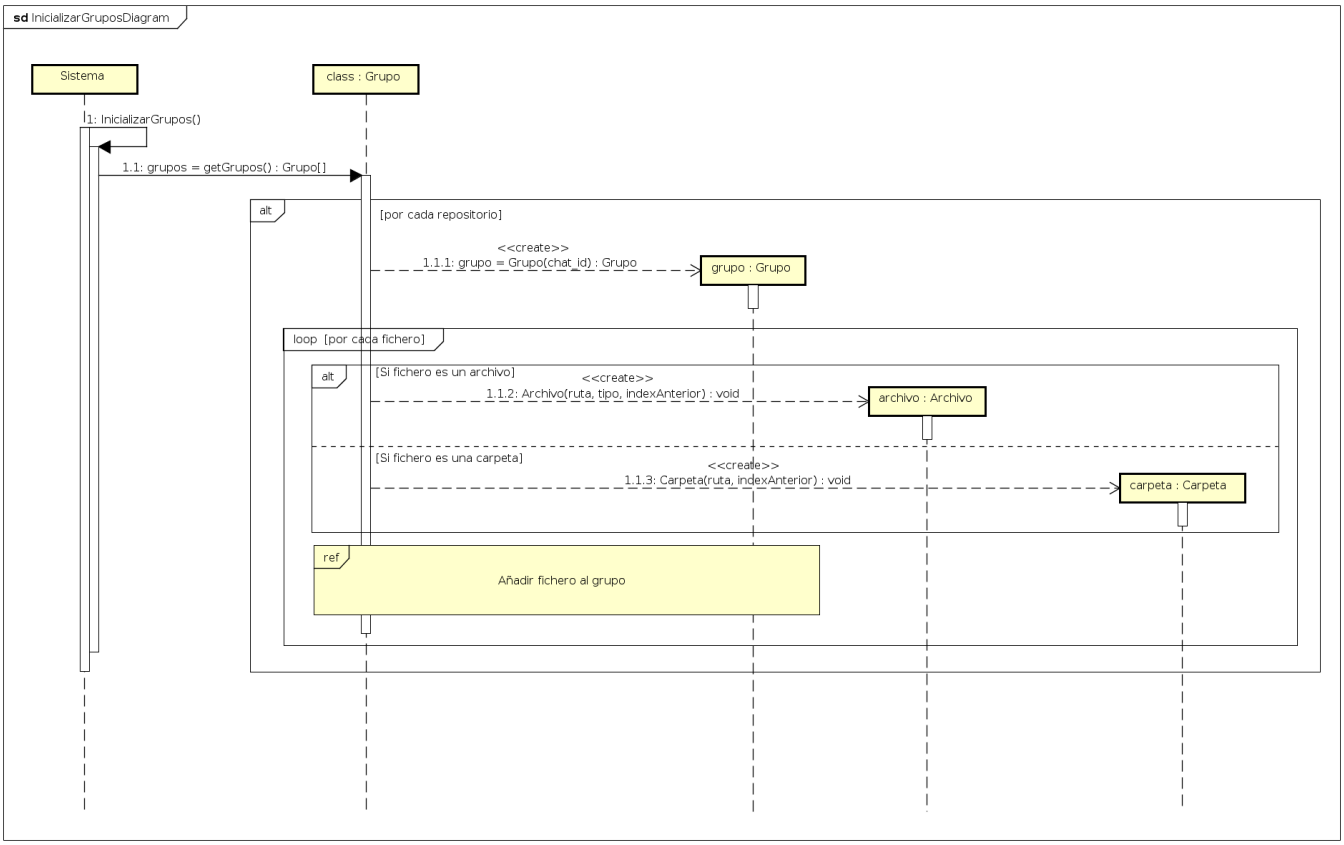


ILUSTRACIÓN 54 – DIAGRAMA DE SECUENCIA DE INICIALIZACIÓN DE GRUPOS

La aplicación se mantendrá en ejecución las 24 horas de cada uno de los 7 días de la semana en cualquier momento, pero es posible que se puedan dar momentos de desconexión o caídas de Internet que produzcan el reinicio del programa de Telegram y por tanto será necesario modelar de nuevo los grupos con sus directorios raíces existentes con todos sus ficheros en tablas de archivos.

Además, en nuestro caso no necesitamos más tipos de datos persistentes que los propios archivos, pero en el caso de tener que almacenarlos para posibles desconexiones y reinicios de la aplicación, utilizaríamos ficheros de almacenamiento en formato JSON o algún tipo de base de datos en el caso de que la información que pretendamos usar sea acorde a las tablas de datos relacionales y sus mecanismos de consulta nos proporcionen ayudas en nuestro sistema software.

Sin embargo, como de momento no tendremos que almacenar nada persistentemente exceptuando los propios ficheros con su clasificación en repositorios de directorios y carpetas, no habrá ningún tipo de problema en los reinicios del servidor ni de la aplicación de Telegram. Simplemente habrá que modelar todos los grupos y sus ficheros persistentes apropiados al diagrama de clases que estamos empleando en este plan de desarrollo de la implementación del programa.

Como vamos a utilizar el lenguaje de programación Python, el recorrido iterativo de una carpeta o directorio lo aportan las librerías necesarias para realizar esto de manera sencilla y eficaz, por tanto, solamente tendremos que nombrar a los repositorios con el nombre asociado al identificador de sus conversaciones de chat de Telegram, de esta manera podemos crear carpetas diferentes y por tanto repositorios locales clasificados.

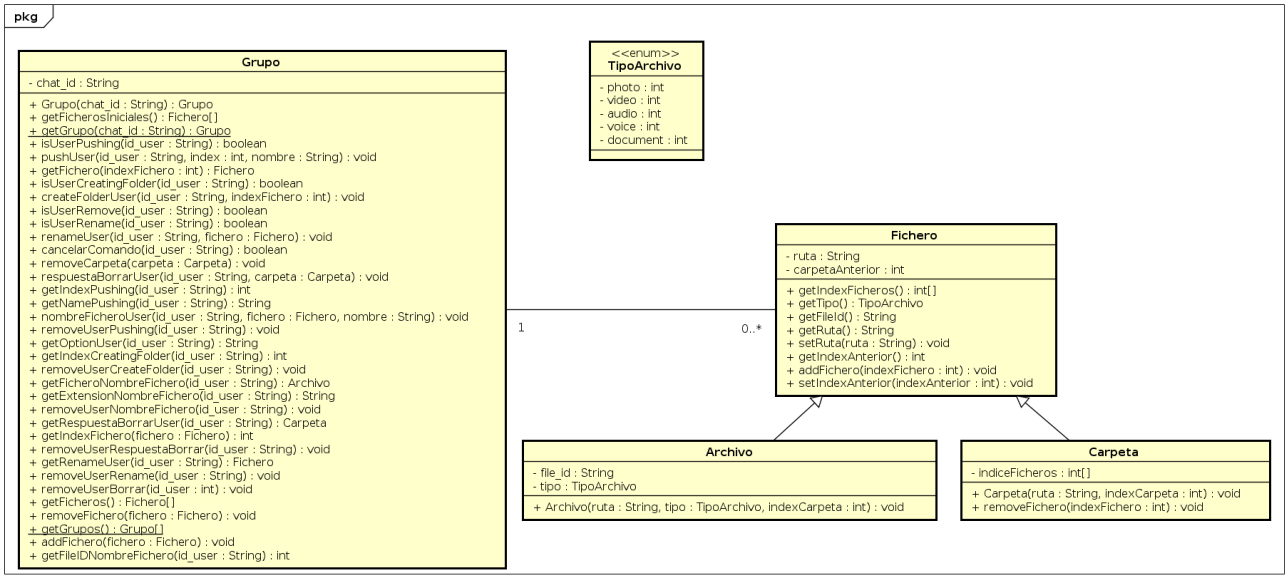


ILUSTRACIÓN 55 – MODELO DE DOMINIO DE DISEÑO

Aquí mostramos el modelo de dominio con las operaciones añadidas a las clases que hemos utilizado para la creación de los diagramas de secuencia de cada una de las funcionalidades expresadas en los casos de uso que se han determinado para la aplicación de Telegram.

Algunas clases se han modificado y eliminado con respecto al prototipo de modelo de dominio desarrollado en el apartado anterior, sin embargo, es normal que ocurran cambios en cuanto a diseño de desarrollo software con respecto a la fase de análisis. Estos cambios influyen en la eficiencia con la interacción de las clases y por tanto al programa en ejecución de procesamiento.

Cabe destacar que en esta aplicación no necesitaremos una gran cantidad de clases ni base de datos como en otros sistemas software que podrían llegar a ser muy robustos, con estas clases son suficientes para elaborar un modelo de dominio de la aplicación de gestión de clasificación de ficheros. Además, es posible que en un futuro se propongan mejoras con respecto a nuevas funcionalidades y servicios, y por tanto haya que elaborar más clases y ampliar la extensibilidad del diagrama de clases que se emplee en la aplicación.

Pero esto lo dejaremos para más adelante, cuando ya tengamos implementada la aplicación y en correcto funcionamiento todos los servicios que hemos diseñado mediante una interfaz gráfica y comandos para las conversaciones de Telegram de los diferentes grupos que deseen albergar un sistema de ficheros clasificado.

PARTE TERCERA - MANTENIMIENTO Y ORGANIZACIÓN.

CAPÍTULO 5 – ADMINISTRACIÓN Y GESTIÓN DEL SERVIDOR.

5.1. Alojamiento de aplicación y servidor de plataforma

El alojamiento de una aplicación de Telegram o programa interactivo como puede ser un bot diseñado para el almacenamiento de archivos necesita de un disco o partición secundaria con una alta capacidad para encargarse de guardar los archivos que pertenecerán a cada uno de los grupos de usuarios con los que interactúe.

Hay que pensar que multitud de grupos o conversaciones de usuarios privadas estarán contactando con nuestro bot de Telegram en distintos momentos del día y además tendremos que tratar los sistemas de ficheros por separado, puesto que cada grupo es diferente y a pesar de que debería tener archivos diferentes, es posible que en ocasiones se almacenen ficheros iguales en los mismos grupos.

Existen bastantes servicios web que ofrecen el alojamiento de programas mediante servidores exclusivos o compartidos, ya sea de manera gratuita con restricciones o bien de pago mediante una tasa de facturación de espacios de trabajo dependiendo de cada aplicación externa utilizada. Ahora veremos varias plataformas externas dedicadas al mantenimiento de aplicaciones de usuarios y una comparativa entre sus ventajas e inconvenientes.

5.1.1. Cloud9

Cloud9 es una IDE (Entorno de Desarrollo Integrado) para soportar multitud de lenguajes de programación como C, C++, PHP, Ruby, Python, Perl, JavaScript, Node.js, etc. Permite a los desarrolladores de código integrar espacios de trabajo (workspaces) en los que permiten depositar sus aplicaciones y programas para un equipo de desarrollo en concreto, permitiendo así el fomento de la programación en parejas (peers programming) junto con la caracterización de diferentes herramientas de desarrollo como el testing de pruebas software en aplicaciones web.

Básicamente este editor se establece en la nube de manera online para los usuarios con sus cuentas en concreto y permite crear diferentes repositorios o espacios de trabajo para almacenar sus ficheros de aplicación dependiendo del lenguaje de programación elegido. Cloud9 es una página web que ofrece bastantes servicios en cuanto a aplicaciones de desarrollo se refiere, permite además desarrollar de manera gratuita con bastantes restricciones en usuarios que no disponen de una cuenta Premium, estas limitaciones pueden ser desde un número concreto de espacios de trabajo privados a crear por cuenta o bien la hibernación de programas en ejecución durante un cierto tiempo pasado.

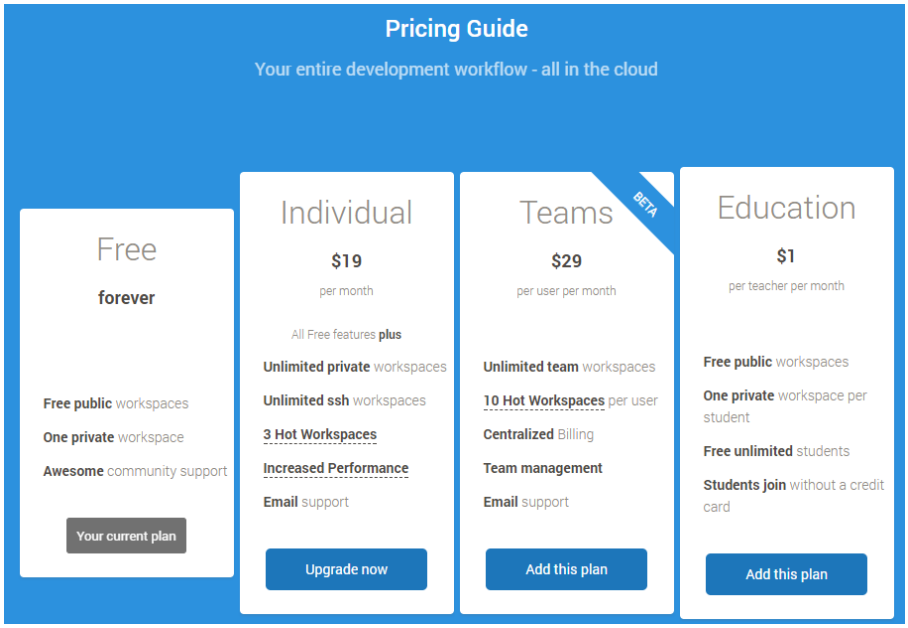


ILUSTRACIÓN 56 – INFORMACIÓN DE PRECIOS EN LA PLATAFORMA CLOUD9

Para nuestro programa interactivo de Telegram, Cloud9 solamente nos serviría si tuviéramos una cuenta Premium en la que hay que pagar una tasa de 19 dólares por mes, que equivaldría a 16,567841 euros por mes. Sería contratar un servidor de aplicaciones por una tasa de facturación mensual para albergar a nuestro bot de Telegram en ejecución sin necesidad de hibernaciones o apagados en cierto tiempo límite. Sin embargo, la plataforma Cloud9 tiene una desventaja, y es la escasa capacidad de almacenamiento existente en donde albergaremos a nuestra aplicación software: 10 Gigabytes (GB) de espacio de disco por espacio de trabajo.

En un principio nos podría servir para los primeros grupos de usuarios que contacten con la aplicación, pero no es conveniente que se tenga disponible tan poco espacio de almacenamiento para un posible incremento de usuarios que se pueda producir en cualquier momento. De manera que será una decisión que habrá que tomar con respecto a un futuro desarrollo completo de nuestro programa de depósito y almacenamiento de archivos, pues siempre existirá la posibilidad de utilizar otras aplicaciones de hosting en Internet para ejercer un servicio constante en Telegram sin necesidad de montarnos nuestro propio servidor con las dificultades de hacerte con uno de ellos y configurarlo previamente para ello.

Una posible alternativa para albergar más espacio es crear otros espacios de trabajo de 10 GB para incrementar así el conjunto total de sistemas de ficheros que se pueden crear por conversación grupal en Telegram, además junto con las características de privilegios que se ofrecen en la cuenta Premium se permitirán crear todos los workspaces que se necesiten de manera ilimitada. Aquí habría que tener en cuenta que existirán diferentes repositorios en los cuales se repartirán los diferentes sistemas clasificados de archivos, además de la gestión que se ha de desarrollar para la aplicación para el control de cada uno de esos sistemas de ficheros.

5.1.2. Heroku

Heroku es otra plataforma como servicio de computación en la nube que fue desarrollada en junio de 2007 con el objetivo de soportar solamente el lenguaje de programación Ruby, pero posteriormente se ha extendido el soporte a Java, Node.js, Scala, Clojure y Python y PHP. De la misma manera que Cloud9, Heroku es un servicio que puede ser gratuito o profesional, dependiendo de las restricciones y ventajas que se quieran añadir a la cuenta de usuario como desarrollador.

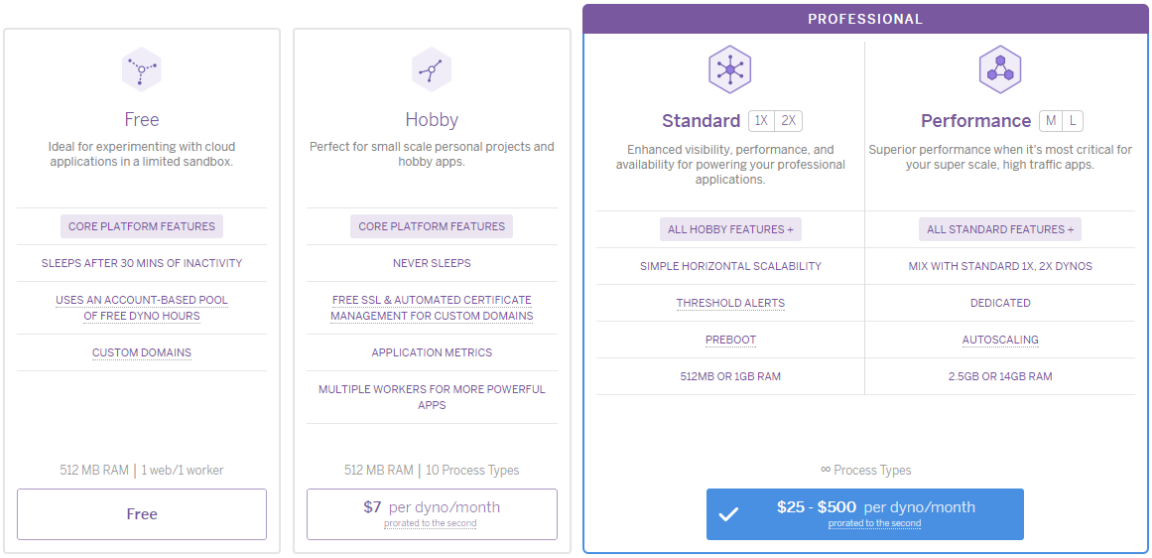


ILUSTRACIÓN 57 – INFORMACIÓN DE PRECIOS EN LA PLATAFORMA HEROKU

Los Dynos son piezas fundamentales del modelo de arquitectura de Heroku, son las unidades que proveen capacidad de cómputo dentro de la plataforma. Están basados en Contenedores Linux, cada uno de ellos está aislado del resto, por lo que los comandos que se ejecutan y los archivos que se almacenan en un Dyno, no afectan a los otros y provee el ambiente requerido por las aplicaciones para ser ejecutadas.

Los posibles comandos que ejecutan los dynos incluyen procesos web o cualquier otro tipo de proceso definido en el archivo *Procfile* de la aplicación. Este es un archivo de texto ubicado en el directorio raíz de la aplicación, y es el mecanismo provisto para la declaración de comandos que luego ejecutarán los dynos. Básicamente, consiste en una lista de tipos de procesos de la aplicación y cada tipo de procesos constituye una declaración de un comando.

Heroku parece una opción más estable que Cloud9, aunque no se especifica el tamaño de disco asignado a cada uno de los dynos preparados para las cuentas de usuarios free o hobby ni tampoco podemos confiar en que nos proporcionen un gran espacio de almacenamiento para los archivos de todos los usuarios. Además, se establece un mecanismo de almacenamiento temporal en cada uno de los dynos asignados a los usuarios, de manera que en el peor caso no nos serviría a menos que contratemos sus servicios de ejecución indefinida, puesto que los sistemas de ficheros de los usuarios de Telegram se perderían tras la hibernación producida en el dyno correspondiente de nuestra cuenta.

Como conclusión a la que podemos llegar tras haber probado ambas plataformas de aplicaciones, es que Heroku permite ejecutar programas de una manera más eficiente y con un menor coste mensual para nuestro uso continuo de la aplicación de Telegram, pero Cloud9 ofrece espacios de trabajo con una mayor capacidad de asignación de memoria de procesamiento y un espacio mayor de disco de almacenamiento a pesar del incremento del coste de mantenimiento de la aplicación mensualmente.

5.1.3. AWS - Amazon Web Services

Finalmente, tras varias búsquedas e investigaciones de sistemas de almacenamiento con un rendimiento eficiente para nuestra aplicación software para el depósito y almacenamiento de archivos, nos encontramos con que los Servicios Web de Amazon que nos ofrece en la nube (AWS – Amazon Web Services) nos vienen perfectos para el alojamiento del programa interactivo que

estamos desarrollando. Las principales razones de la elección de los servicios de Amazon Web residen en la existencia de una gran escalabilidad y alta capacidad de almacenamiento por medio de servidores virtuales (EC2). Además, el precio del almacenamiento y la tasa de ejecución de cada uno de los servicios contratados en Amazon vienen determinados por el tiempo de procesamiento que el servidor virtual ha estado encendido y por tanto ejecutando aplicaciones o procesos, esto también ocurre con el espacio de almacenamiento y sus incrementos de espacio en disco para esas máquinas virtuales asociadas a la cuenta de Amazon.

Un dato interesante de Amazon es que nada más crearte la cuenta en su página web, te ofrecen 12 meses de ofrecimiento de servicios gratuitos siempre y cuando se cumplan una serie de condiciones y limitaciones que se establecen en una capa de servicios gratuita por la propia empresa. Los servicios que ofrece Amazon comprenden una amplia gama del mundo de la informática dentro de los lenguajes de programación y las API's de desarrollo que necesitan los ingenieros informáticos para la implementación de sus aplicaciones en servidores dedicados.

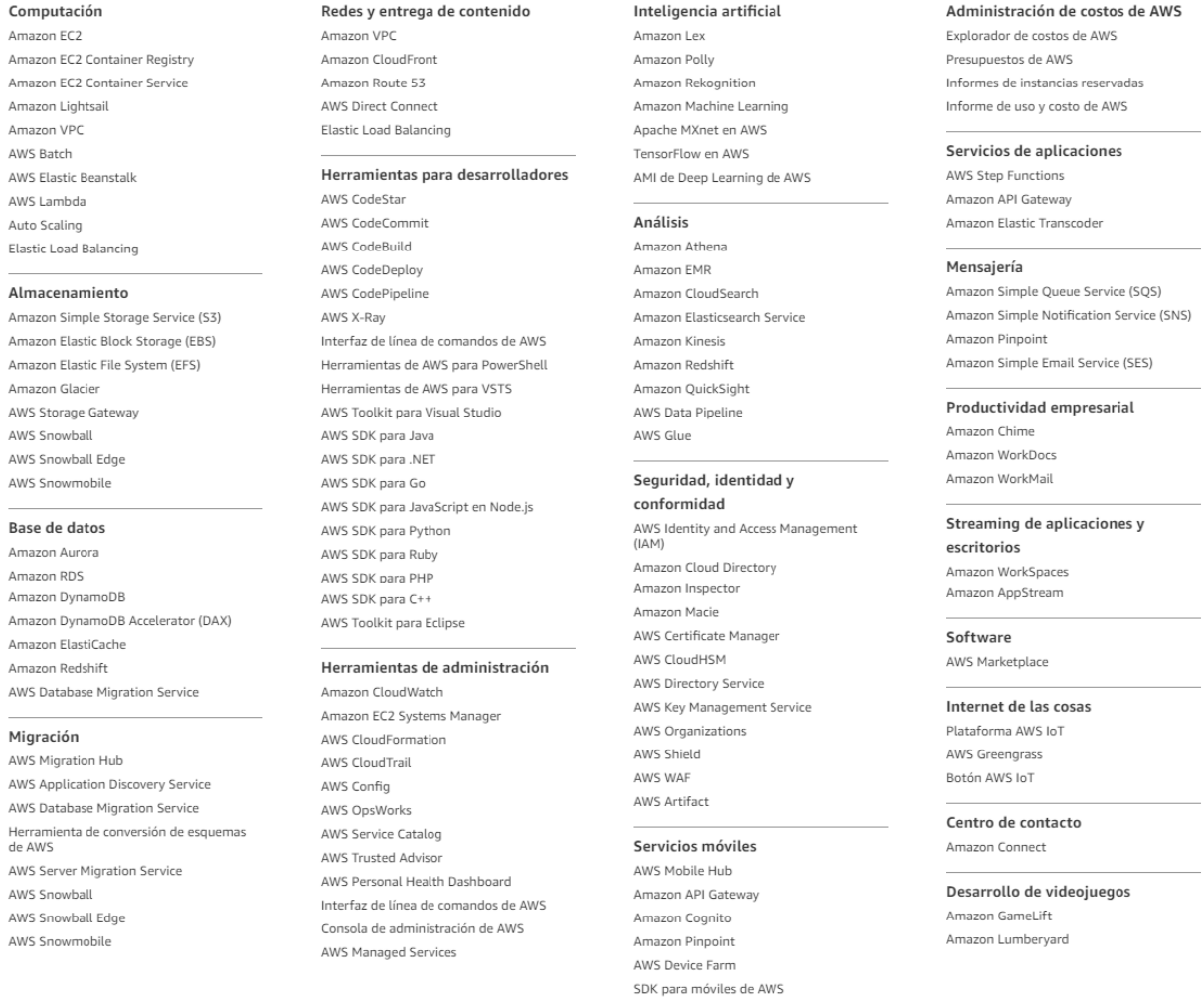


ILUSTRACIÓN 58 – SERVICIOS DE LA PLATAFORMA WEB DE AMAZON

Como se puede observar, en Amazon Web Services se puede realizar cualquier tipo de aplicación que el usuario o equipo de desarrollo quiera desarrollar e implementar. Amazon permite el hosting para aplicaciones web de la misma manera que otras compañías de hosting y además incorpora multitud de funcionalidades a mayores como el acceso a la consola de comandos mediante SSH y la configuración de tu propio servidor virtual con una personalización concreta.

Evidentemente el precio y coste de cada uno de estos servicios dependerá del tipo de proyecto que se esté elaborando en el momento adecuado, del servidor que se está empleando, su tasa de ejecución temporal y del tipo de servicio que se utilice para el desarrollo de los proyectos software con alojamiento en esta plataforma.

No todos los servicios que ofrece la plataforma costarán igual y cada uno ofrecerá unas capacidades de desarrollo mayores o menores dependiendo del tipo de servicio, así como si se utiliza un servidor dedicado o bien compartido para los clientes específicos.

La plataforma web de Amazon posee demasiadas funcionalidades como para rechazarlas a primera vista, además es posible que en un futuro sea necesario realizar aplicaciones web de conexión con la aplicación software de Telegram y se incorpore un hosting mediante los propios servicios de Amazon. Las herramientas que se venden en AWS poseen las características de extensibilidad, escalabilidad y rendimiento para el desarrollo de aplicaciones con integración de seguridad por medio de la plataforma.

Con los 12 meses de prueba gratis que nos han proporcionado pretendemos probar la calidad y capacidad de cómputo que ofrecen los servidores de Amazon para nuestro bot conversacional en una máquina virtual Linux con la instalación de las librerías y bibliotecas apropiadas para su funcionamiento mediante el servicio EC2 con la ejecución de comandos en consola SSH.

Uso general – Generación actual

t2.nano	1	Variable	0.5	Solo EBS	$0.0063 por hora
t2.micro	1	Variable	1	Solo EBS	$0.0126 por hora
t2.small	1	Variable	2	Solo EBS	$0.025 por hora
t2.medium	2	Variable	4	Solo EBS	$0.05 por hora
t2.large	2	Variable	8	Solo EBS	$0.1008 por hora
t2.xlarge	4	Variable	16	Solo EBS	$0.2016 por hora
t2.2xlarge	8	Variable	32	Solo EBS	$0.4032 por hora
m4.large	2	6.5	8	Solo EBS	$0.111 por hora
m4.xlarge	4	13	16	Solo EBS	$0.222 por hora
m4.2xlarge	8	26	32	Solo EBS	$0.444 por hora
m4.4xlarge	16	53.5	64	Solo EBS	$0.888 por hora
m4.10xlarge	40	124.5	160	Solo EBS	$2.22 por hora
m4.16xlarge	64	188	256	Solo EBS	$3.552 por hora
m3.medium	1	3	3.75	1 x 4 SSD	$0.073 por hora
m3.large	2	6.5	7.5	1 x 32 SSD	$0.146 por hora
m3.xlarge	4	13	15	2 x 40 SSD	$0.293 por hora
m3.2xlarge	8	26	30	2 x 80 SSD	$0.585 por hora

ILUSTRACIÓN 59 – TASA DE FACTURACIÓN DE EJECUCIÓN EN INSTANCIAS EC2 EN AMAZON

En esta ilustración establecemos las tasas de facturación del servidor procedente de Irlandia en la Unión Europea. Podríamos ilustrar el conjunto de tasas existentes de los 15 servidores totales distribuidos en todo el mundo que posee la plataforma AWS, pero no creo que sea necesario puesto que las variaciones de precios no son muy relevantes entre sí, de manera que con este servidor nos

sirve para ver que el precio por hora es realmente barato para mantener cualquier programa en ejecución en una instancia EC2 de este tipo.

No necesitamos utilizar tampoco servidores muy potentes y con el t2.micro nos sobraría realmente para una gran cantidad de utilidades y funcionalidades que pretendamos establecer en las aplicaciones de nuestra instancia, por tanto la conclusión definitiva es que los servicios de la plataforma web de Amazon son muy útiles y nos permiten escalar de la manera que nosotros elijamos siempre y cuando nos atengamos al precio y coste de facturación que haya que pagar por su utilización.

	Almacenamiento estándar	Almacenamiento estándar – acceso poco frecuente †	Almacenamiento en Glacier
Primeros 50 TB/mes	$0.023 por GB	$0.0125 por GB	$0.004 por GB
Siguientes 450 TB/mes	$0.022 por GB	$0.0125 por GB	$0.004 por GB
Más de 500 TB/mes	$0.021 por GB	$0.0125 por GB	$0.004 por GB

ILUSTRACIÓN 60 – TASA DE FACTURACIÓN DE ALMACENAMIENTO EN INSTANCIAS EC2 EN AMAZON

En cuanto a espacio de almacenamiento con respecto a otras plataformas externas, Amazon también funciona de una manera peculiar y diferente con tasas de facturación por GigaByte consumido en el servidor, de esta manera si una instancia EC2 no utiliza nada de espacio de almacenamiento o ha usado muy pocos Bytes del disco asignado, no se le aplicará ningún tipo de tasa de facturación.

Esto es increíblemente bueno para cualquier aplicación en general, puesto que se facturará exactamente lo que usa la aplicación y no se basa en una tasa de facturación fija que se cobra al mes como realizan otras plataformas y proveedores de servicio en la nube, de esta manera si en unos meses una aplicación no utiliza absolutamente nada de espacio en disco, el titular de la cuenta no tendrá que pagar ninguna tasa de facturación de espacio.

Además, es perfectamente posible agregar volúmenes de discos a las instancias EC2 de Amazon, incrementando así el espacio de almacenamiento en el sistema operativo adecuado. Esto es perfecto para nuestra aplicación puesto que a medida que necesitemos más espacio podemos ir incrementándolo a gusto del consumidor y grupos de Telegram.

CAPÍTULO 6 – MANUAL DE INSTRUCCIONES DEL PROYECTO.

6.1. Manual de instalación de la plataforma

La instalación de las librerías, bibliotecas y lenguajes de programación necesarios para la ejecución de la aplicación de manera correcta se dará lugar en este apartado, concretamente se proporcionarán los comandos utilizados en un sistema operativo Linux para albergar el control de los sistemas de archivos y de la aplicación como tal.

Esto no quiere decir que otro sistema operativo como por ejemplo Windows no pudiera ejecutar el programa interactivo que estamos elaborando en Python ni ninguna de las librerías asociadas simplemente por no tener la consola de intérprete de comandos que posee el sistema operativo Linux. De hecho, los lenguajes de programación permiten la abstracción del código de implementación de sus funciones para poder ser utilizadas en cualquier tipo de sistema operativo independientemente del tipo de versión o formato.

El único inconveniente que existiría en utilizar Windows como sistema operativo para ejecutar el bot, sería que por defecto no vendría instalado Git como en el sistema operativo Linux ni tampoco vendría Python instalado, de manera que habría que instalar esos dos componentes de manera separada y mediante los instaladores que se proporcionan en las páginas web oficiales de los mismos programas.

Una de las malas implementaciones que se podrían haber integrado en nuestro proyecto software, era la de realizar cada una de las llamadas al sistema operativo por medio de la consola de comandos del sistema operativo Linux, pues esto es considerado un error de diseño y de implementación, ya que otros sistemas operativos (Windows) que no disponen de la misma consola ni los mismos comandos e interrupciones software fallarían en la ejecución de las funcionalidades básicas de nuestros servicios del programa de Telegram.

Por ello se debe integrar las librerías necesarias en el lenguaje de programación determinado para realizar estas operaciones software mediante las instrucciones correspondientes del sistema operativo concreto. A continuación, veremos como instalar la versión del lenguaje de programación adecuado con sus correspondientes bibliotecas que serán imprescindibles para la ejecución del bot de Telegram.

6.1.1. Instalación de Python y sus librerías asociadas

El principal problema del mantenimiento del programa en ejecución que controla el bot de Telegram es la inseguridad de las peticiones HTTP que se realizan en Python 2.7.6 o en versiones más bajas que vienen instaladas por defecto en los sistemas operativos. Esto provoca warnings y fallos en las peticiones como los que vienen mostrados en la siguiente ilustración:

```
/home/leandrotoledo/workspace/fisl-17-telegram-bot-com-python/python/env/local/lib/python2.7/site-packages/urllib3/util/ssl_.py:318: SNIMissingWarning: An HTTPS request has been made, but the SNI (Subject Name Indication) extension to TLS is not available on this platform. This may cause the server to present an incorrect TLS certificate, which can cause validation failures. You can upgrade to a newer version of Python to solve this. For more information, see https://urllib3.readthedocs.io/en/latest/security.html#snimissingwarning.

/home/leandrotoledo/workspace/fisl-17-telegram-bot-com-python/python/env/local/lib/python2.7/site-packages/urllib3/util/ssl_.py:122: InsecurePlatformWarning: A true SSLContext object is not available. This prevents urllib3 from configuring SSL appropriately and may cause certain SSL connections to fail. You can upgrade to a newer version of Python to solve this. For more information, see https://urllib3.readthedocs.io/en/latest/security.html#insecureplatformwarning.
```

ILUSTRACIÓN 61 – WARNINGS EN LA EJECUCIÓN DE LA APLICACIÓN

Estos warnings se proporcionan al ejecutar la aplicación (SNIMissingWarning y InsecurePlatformWarning) indicando que faltan librerías por instalar para que las peticiones HTTP no resulten fallidas o surjan problemas en la validación de las mismas por temas de seguridad de certificados. Algunas clases y paquetes de programación necesitarán ser instalados para solucionarlo de manera manual o bien podemos instalar otra versión de Python superior (3 en adelante) que contenga estas clases ya directamente.

Además, podemos instalar otra versión de Python 3 sin sobreescribir la versión por defecto (versión 2) que viene instalada en nuestro sistema operativo Linux, la principal razón del por qué no actualizaremos esa versión de Python reside en que muchas de las librerías y dependencias se romperían y habría que arreglar todas esas dependencias correctamente, por lo que cuesta menos instalar una versión nueva de Python y ejecutar nuestra aplicación con ella.

Procedemos a instalar la versión 3 de Python, junto con la herramienta de instalación de paquetes y su administración para Python 3, puesto que el sistema de gestor de paquetes pip está asociado a la versión por defecto de Python 2 y necesitamos establecer un nuevo sistema de instalación de paquetes para nuestra nueva versión de Python 3 recién instalada. Los siguientes comandos que vamos a emplear para realizar la instalación de Python 3 con sus librerías asociadas son los siguientes:

sudo apt-get install python3

sudo apt-get install pip-python3

sudo pip3 install python-telegram-bot –upgrade

sudo pip3 install pybitbucket

sudo pip3 install PyGithub

sudo pip3 install GitPython

Los dos primeros comandos corresponden con la instalación de Python 3 y su paquete de herramientas para la instalación de librerías y bibliotecas para la propia versión nueva de Python instalada, mientras que el resto de comandos se asocian con la utilización de esa herramienta de obtención de librerías para almacenar en la carpeta de instalación de Python 3 los paquetes de programación asociados a la API de Telegram Bot, PyBitbucket, PyGithub y GitPython que necesitaremos para el correcto funcionamiento de la aplicación en ejecución.

python3 filebot.py &

Finalmente, para la ejecución de la aplicación de nuestro servidor, tendremos que utilizar el ejecutable de la versión 3 de Python que acabamos de instalar, puesto que la versión 2 daba errores de peligro de peticiones HTTP con respecto a la integridad y pérdida de las mismas. Esto no quiere decir que existan peticiones HTTP que puedan perderse o que sean fallidas, sin embargo, el número de las mismas es reducido en la versión 3 de Python y nos proporciona una interfaz mucho más segura y accesible para los usuarios.

Nótese la ejecución del programa con el símbolo "&" para evitar mantener a la consola de intérprete de comandos bloqueada o la sesión SSH en el caso de utilizar acceso remoto. La ejecución de un proceso en el fondo o en segundo plano es necesaria para cuando albergamos nuestra aplicación en un servidor remoto o plataforma externa contratada como servicio en la nube.

6.1.2. Establecimiento de clave SSH para servidor Linux

Si se está utilizando un sistema de control de versiones para administrar los ficheros de los repositorios en la aplicación, tendremos que asignar claves ssh en los ficheros de configuración para acceder con estos mecanismos de manera directa a las plataformas remotas sin necesidad de estar esperando una respuesta por parte del usuario gestor del repositorio.

El acceso HTTP es muy bueno para cuando los usuarios gestores de los repositorios son reales y pueden introducir el nombre de usuario y la contraseña de manera directa, sin embargo, en una aplicación de gran alcance para la gestión y mantenimiento de muchos repositorios no se puede permitir esperas de introducción de datos.

La administración de este sistema es muy importante para evitar esperas innecesarias en la aplicación de gestión de archivos y además permite realizar instrucciones de subida o modificación en los sistemas de ficheros mediante la lectura de la clave de usuario y contraseña que se encuentra en un fichero de configuración controlado por el propio sistema operativo.

Por esto mismo, accederemos a nuestros repositorios de Bitbucket o Github mediante consola con el uso de SSH (Secure Shell) para utilizar el control de versiones de ficheros y aplicar las operaciones deseadas. Hay que tener en cuenta que nuestra aplicación utilizará como sistema de control de versiones: Git y no Mercurial u otros sistemas, de manera que la forma de configuración y administración es diferente por cada sistema de control de versiones y por cada sistema operativo en el que se encuentre la aplicación ejecutándose.

De manera que necesitamos un acceso mucho más eficiente y que nos permita tener ficheros pre configurados con el almacenamiento de claves para la lectura de los mismos en el momento en el que se necesite, esto nos lo proporciona el acceso SSH para los repositorios concretos que hablaremos en los siguientes párrafos.

Comenzamos con la configuración del establecimiento de las claves SSH para el acceso a los repositorios de manera remota con Git como sistema de control de versiones. Primero empleamos la consola de línea de comandos para ejecutar un comando de generación de un par de claves públicas y privadas en el servidor en el cual vamos a integrar nuestra aplicación.

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key
(/Users/emmap1/.ssh/id_rsa):
```

ILUSTRACIÓN 62 – COMANDO DE GENERACIÓN DE CLAVES SSH

Nos preguntará además que introduzcamos un fichero en donde guardar las claves de configuración, por defecto se establece en el directorio home del usuario del sistema operativo dentro de una carpeta oculta denominada .ssh/id_rsa.

En nuestro caso, no vamos a determinar una nueva ruta o algo parecido, lo haremos todo por defecto, puesto que la ruta por defecto que nos proporciona el comando nos vale para guardar las claves de configuración de acceso a las plataformas remota y poder leerlas de manera sencilla.

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key
(/Users/emmap1/.ssh/id_rsa):
Created directory '/Users/emmap1/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in
/Users/emmap1/.ssh/id_rsa.
Your public key has been saved in
/Users/emmap1/.ssh/id_rsa.pub.
The key fingerprint is:
4c:80:61:2c:00:3f:9d:dc:08:41:2e:c0:cf:b9:17:69
emmap1@myhost.local
The key's randomart image is:
+--[ RSA 2048]----+
|*o+ooo.          |
|.+.=o+ .         |
|. *.* o .        |
| . = E o         |
|    o . S        |
|   . .           |
|     .           |
|                 |
|                 |
+-----------------+
```

ILUSTRACIÓN 63 – RESULTADOS DEL COMANDO DE GENERACIÓN DE CLAVES SSH

A continuación, se generará un perfil propio como identidad con claves públicas y privadas de acceso por SSH, un factor importante para tener en cuenta es que estas claves nos sirven para cualquier plataforma de almacenamiento de archivos (Bitbucket, Github, Gitlab…) que también permita el acceso por SSH y cuyas operaciones utilicen un sistema de control de versiones Git.

Una vez generados los ficheros de configuración, uno para la clave pública y otro para la privada de la identidad de usuario correspondiente, podemos proceder a la conexión de estas claves con la cuenta asociada al bot correspondiente de las aplicaciones remotas que permitan el control de repositorios.

Aquí mostraremos como enlazar la clave pública con la aplicación de Bitbucket para la sincronización de repositorios en la aplicación de Telegram, este procedimiento ha de realizarse de la misma manera con la plataforma Github.

Para realizar esto, tenemos que abrir el fichero .ssh/id_rsa que contiene la clave pública y copiar esa clave en las opciones de ajustes de configuración de clave SSH de la cuenta de la aplicación web Bitbucket, aquí adjunto una ilustración para que se entienda el procedimiento:

ILUSTRACIÓN 64 – AGREGACIÓN DE CLAVE SSH EN BITBUCKET

Tras realizar esto, la administración y configuración del acceso SSH queda finalizado y el bot ya puede emplear los comandos de Git sin necesidad de feedback ni ningún tipo de espera por parte del usuario (introducción de contraseña, cuenta de usuario, etc…)

6.2. Manual de usuario de la aplicación

Toda aplicación software requiere un manual de instrucciones de usuario para explicar los términos, conceptos o conocimientos técnicos que se encuentran en el proyecto, de manera que mediante este apartado pretendemos hacer comprender a los usuarios qué tipo de estructura o funcionamiento está basada nuestra plataforma y por tanto nuestro proyecto software.

En un principio comenzaremos explicando los conocimientos básicos y esenciales que utiliza la aplicación de Telegram, junto con sus características principales para entender el correcto funcionamiento de los sistemas de ficheros y repositorios con los que trabaja el programa. Posteriormente se dará paso a una descripción de los comandos con ejemplos sencillos para determinar sus opcionalidades de uso.

Una parte importante que debe entender cualquier usuario que pretenda usar la aplicación es el funcionamiento de los bots en Telegram, es decir, debe saber que tendrá que comunicarse con el bot mediante su alias (@FileDataBot) de identificación para encontrar al bot dentro de la propia plataforma de Telegram e incorporarlo en el grupo que sea necesario para comenzar con la creación

de un repositorio común.

El usuario además puede utilizar el bot tanto de manera privada o de manera pública, puesto que está preparado para todo tipo de conversación posible dentro de la plataforma, ya sean grupos, supergrupos o simplemente conversaciones privadas directas con el programa interactivo.

6.2.1. Definiciones y términos generales

En este apartado expondremos una sección de explicación de los términos más característicos que deberán entender los usuarios para poder exprimir el potencial de la aplicación al máximo. Primero comenzaremos con las definiciones básicas de los sistemas de ficheros con respecto a la clasificación de directorios y archivos mediante la especificación de rutas asociadas.

Un sistema de ficheros es considerado un repositorio, o lo que es lo mismo, un directorio cuyo contenido son otros directorios y archivos en su interior. Evidentemente cada carpeta o directorio puede contener ficheros o no, produciéndose así la recursividad de ficheros dando el nombre de sistema de ficheros como tal.

De la misma manera que los sistemas de ficheros que manejan los sistemas operativos, nuestra aplicación software maneja repositorios de la misma manera para cada una de las conversaciones y grupos de Telegram, proporcionando así una clasificación adecuada en directorios y una organización de repositorios como las plataformas de almacenamiento de archivos en la nube.

Esta definición que estamos dando del sistema de ficheros está bastante simplificada y no tiene en cuenta toda la complejidad de gestionar los ficheros (tablas de archivos clasificadas, asignación de espacio de almacenamiento, mantenimiento de los archivos…), aun así, es suficiente para hacer entender al usuario que la aplicación es un sistema coordinador que maneja archivos y los clasifica de la manera correcta para cada grupo y conversación.

Concepto de carpeta: También denominado directorio, se trata de un fichero que almacena otro tipo de ficheros a su vez, ya sean archivos o subcarpetas dentro de la misma. Las carpetas y directorios tienen un papel fundamental en la compartición de archivos y etiquetación de los mismos, al fin y al cabo, son este tipo de ficheros los que permiten realizar una clasificación de contenido y diferenciar los archivos de un propio repositorio.

Concepto de ruta: Una ruta es la forma de referenciar un archivo informático o directorio en un sistema de archivos, señalando la localización exacta del fichero mediante una cadena de caracteres concreta. Ésta puede ser de diversas formas dependiendo del sistema operativo, sin embargo, al estar basando nuestra aplicación en el sistema operativo Linux, se emplearán las barras diagonales "/" para separar los nombres de los archivos y directorios que conforman la ruta.

Hay dos tipos de rutas: absolutas y relativas, dependiendo de si la ruta comienza en el directorio raíz o bien a partir del directorio actual en el cual nos encontramos dentro del sistema de ficheros. La mayor parte de los comandos implementados requieren la introducción de rutas absolutas y no relativas, puesto que trabajaremos a partir del directorio raíz del repositorio.

Estas rutas permiten identificar a los archivos que vayan creando los usuarios mediante el uso de los comandos de la aplicación en Telegram, a continuación, veremos un ejemplo de ruta absoluta para identificar a un archivo llamado "campana.mid" a partir del recorrido de una serie de directorios del propio sistema de archivos:

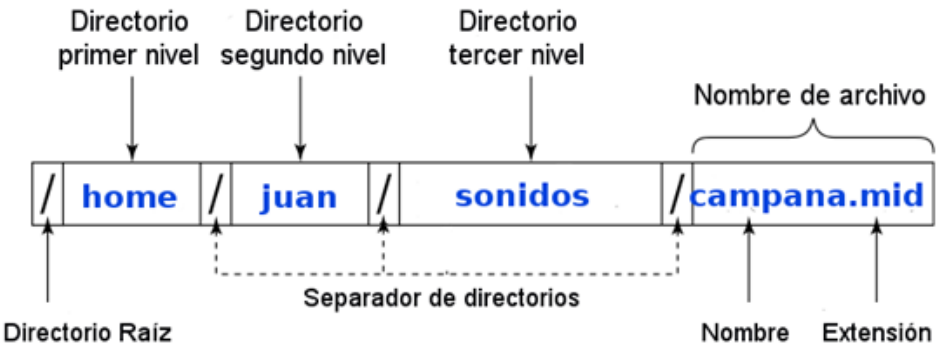


ILUSTRACIÓN 65 – EJEMPLO DE RUTA ABSOLUTA

Los archivos tienen su propia extensión que permite establecer un tipo de formato y permitir al sistema diferenciar si se trata de una fotografía, video o bien un documento normal y corriente. De esta manera nuestra aplicación diferencia cada uno de los formatos y permite clasificar los archivos para utilizar las operaciones de la API de manera correcta en cada caso.

La aplicación no permitirá ningún tipo de cambio para el formato de extensión de un archivo por el momento, puesto que los distintos tipos de formatos que se podrían adaptar a los archivos es una funcionalidad que habría que estudiar y tener en cuenta todo tipo de alternativas para cada tipo de archivo del repositorio.

Ejemplos de rutas de identificación de ficheros:

/carpeta1/carpeta2/archivo => Archivo denominado “archivo” (ruta absoluta, empieza con “/”)

./carpeta1/archivo => Archivo denominado “archivo” (ruta relativa, “.” es directorio actual)

carpeta1/archivo => Archivo denominado “archivo” (ruta relativa expresada de otra forma y equivalente a la anterior)

/carpeta => Carpeta denominada “carpeta” (ruta absoluta, situado en el directorio raíz “/”)

Concepto de directorio raíz: El directorio raíz es el primer directorio o carpeta en una jerarquía de archivos, está considerado por ser el inicial del que “cuelgan” el resto de ficheros dentro del sistema. Contiene todos los subdirectorios y por la propiedad transitiva todos los archivos dependen de él y se encuentran en su contenido.

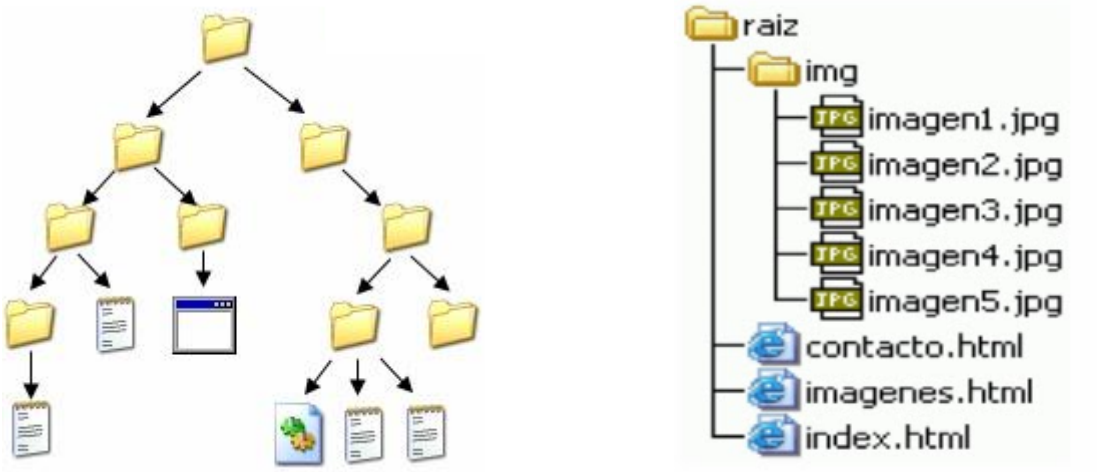


ILUSTRACIÓN 66 – EJEMPLOS DE SISTEMAS DE FICHEROS

Como se puede observar, se pueden almacenar archivos dentro del directorio raíz como si de cualquier otra carpeta o directorio normal se tratara, por ello se permitirá guardar archivos dentro del repositorio como tal e incluso sin la especificación de ruta como tal, es decir, proporcionando un "nombre" asociado como parámetro de comando.

Habiendo hecho esta introducción de conceptos y términos necesarios para entender el funcionamiento de un sistema de ficheros y las capacidades que permite con su etiquetación de directorios y subdirectorios, procederemos a explicar todos y cada uno de los comandos implementados en la aplicación de este proyecto software.

6.2.2. Comandos de la aplicación

Los comandos de usuario propuestos en la aplicación de Telegram han sido determinados de la manera más cómoda y precisa para el usuario y han sido almacenados y guardados por BotFather para el autocompletado de comandos dentro de los chats de Telegram:

- **/start:** Comienza la inicialización del bot explicando la ayuda básica del bot con un mensaje de la ayuda mínima para que los usuarios sepan interactuar con la aplicación. Todos los usuarios que comiencen a comunicarse por primera vez con el bot de Telegram ejecutarán de manera indirecta este comando para inicializar la conversación.

- **/help:** Función básica de ayuda para determinar y explicar de manera breve todos los comandos posibles que se pueden utilizar en la aplicación interactiva elaborada. Se mostrará una ayuda lo más simple posible y efectiva para el usuario, de manera que pueda entender de lo que va a tratar el bot y qué tipo de funcionalidades puede ejecutar.

- **/files:** Muestra una interfaz gráfica con botones asociados al sistema de ficheros del grupo en concreto. Esta funcionalidad depende de si existen o no ficheros dentro del directorio raíz del sistema de ficheros, en el caso de que no haya se mostrará un mensaje por la conversación afirmando que no existe ningún fichero dentro del sistema. Si por el contrario existen ficheros iniciales dentro del directorio raíz, se establecerá una estructura de botones en filas asociando cada uno a esos ficheros iniciales, con un máximo 3 botones por fila.

  Además, se incluirán referencias en cada una de las carpetas o archivos mostrados en esa estructura de visualización de la interfaz, de manera que cuando el usuario correspondiente haga click en los botones, dependiendo de si es un directorio o un archivo

con contenido y formato adecuado, se realizará una funcionalidad u otra controlada por una función callback adecuada.

Si se ha seleccionado una carpeta, se muestra el contenido de la carpeta o directorio, cambiando la interfaz gráfica con el nuevo contenido de ficheros de esa carpeta y además se incluirá un nuevo botón “ATRÁS” siempre y cuando se pueda navegar hacia el directorio anterior dentro del sistema de ficheros.

Si por el contrario se ha realizado click en un archivo, se mostrará un nuevo mensaje de referencia para ese archivo sin modificar la interfaz gráfica, permitiendo así la descarga del mismo por parte de cualquier usuario en la conversación.

Esta funcionalidad en concreto es la parte más interesante de nuestra aplicación, puesto que se permite integrar interfaces gráficas con botones lo más parecidas al formato de aplicaciones web y del navegador para realizar búsquedas eficientes entre los archivos que se hayan almacenado en nuestro servidor. Además, gran parte de los usuarios que intenten utilizar este bot en la plataforma, utilizarán este comando para navegar y realizar las funcionalidades de opciones con botones, puesto que es un entorno visual mucho más simple y agradable que las funcionalidades de comandos en modo consola.

- **/get:** Comando para realizar búsquedas eficientes de archivos que cumplen un cierto patrón en su nombre de identificación. La aplicación necesita un comando u opción de funcionalidad que permita realizar búsquedas de varios archivos que tengan cierta cadena de caracteres en su nombre sin necesidad de navegar por el repositorio.

  No diferenciará entre mayúsculas o minúsculas a la hora de encontrar un fichero cuyo nombre coincida con el patrón de búsqueda del comando, esto significa que el usuario puede proporcionar nombres con las mayúsculas o minúsculas que desee porque el sistema pasará todo argumento de comando a minúscula para realizar las búsquedas.

  Ejemplos de uso:

  /get prueba => Busca todos los archivos cuyo nombre contenga la cadena de caracteres “prueba”.

  /get PrUeBA => Realiza exactamente la misma funcionalidad que el anterior comando, puesto que mayúsculas no tendrán ninguna relevancia.

- **/mkdir:** Comando para crear carpetas o directorios en el sistema de ficheros. Funciona de la misma manera que el comando “mkdir” de consola de un sistema operativo GNU/Linux, se debe especificar una ruta como argumento para especificar el nuevo directorio a crear. El comando controla si se puede crear la carpeta y en el caso de que no se pueda producir la creación, el sistema informa de la razón principal con un mensaje de error.

  Los tres errores posibles que puede haber son: que la ruta asociada pertenezca a un archivo y no a una carpeta o directorio, que no exista la ruta de directorios y por tanto alguno de los directorios especificados en la ruta no existen o bien que el nombre proporcionado para la nueva carpeta ya exista y esté en uso para otro fichero de esa ruta de directorios.

  Ejemplos de uso:

  /mkdir /carpeta1/carpeta2 => Crea una carpeta llamada “carpeta2” dentro del directorio

“carpeta1” del sistema de ficheros.

/mkdir carpeta2 => Crea una carpeta llamada “carpeta2” en el repositorio inicial (directorio raíz del sistema de ficheros).

- **/push:** Funcionalidad para subir archivos al servidor de Telegram y por tanto a los sistemas de ficheros de los grupos que controla la aplicación de este proyecto. Posee un argumento de entrada asociado a la ruta de creación del archivo y ha sido diseñada para un amplio abanico de posibilidades de utilización por parte del usuario.

  Este comando permite la activación de la transacción de espera de archivo de subida para el usuario que lo ejecute, este tipo de procedimientos de espera siempre se pueden cancelar con el comando de cancelación (tanto de la interfaz gráfica como con el comando de chat).

  Además, si en la ruta de argumento no se especifica el nombre del archivo a subir, en el procedimiento de espera de archivo se tendrá en cuenta si se ha proporcionado nombre como comentario en el caso de que el archivo lo permita. En cuanto a los archivos que no permiten la asociación de comentarios se activará de nuevo otra transacción de proceso de espera de introducción de nombre por parte del usuario para la finalización del comando.

  En cuanto a los mensajes de error que puede otorgar este comando encontramos solamente dos que finalicen el comando directamente y no produzcan ninguna transacción de espera: cuando la ruta de directorios no existe y cuando exista ya un fichero dentro de la ruta con el mismo nombre proporcionado.

  Dentro de cada transacción y procesamiento de espera de la aplicación, se realizan comprobaciones con respecto al nombre que se le puede establecer al archivo a subir, ya sea en la transacción de subida de archivo cuando se proporciona el nombre en un comentario del archivo o bien en la espera de introducción de nombre por parte del usuario.

  Ejemplos de uso:

  /push archivo => Crea un archivo en el directorio raíz con el nombre “archivo”

  /push /archivo => Misma funcionalidad que el ejemplo anterior.

  /push / => Crea un archivo en el directorio raíz sin nombre asociado aún.

  /push => Misma funcionalidad que el ejemplo anterior.

  /push /carpeta => Crea un archivo en el directorio “carpeta” sin nombre asociado aún.

  /push /carpeta/archivo => Crea un archivo en el directorio “carpeta” con el nombre “archivo” (siempre y cuando no haya otro fichero con el mismo nombre)

  /push /carpeta1/carpeta2 => Crea un archivo en la ruta de directorios “/carpeta1/carpeta2” sin nombre asociado aún.

- **/rmdir:** Comando para eliminar carpetas o directorios en el sistema de ficheros de un chat concreto. Al igual que existe una operación de creación, debe existir una eliminación de los mismos directorios creados por cualquier usuario.

De la misma manera que el comando de creación de carpetas, este comando requiere un parámetro de entrada y por tanto una ruta como argumento para especificar el directorio que se quiere eliminar. Este comando puede finalizar con tres posibles mensajes de error: la ruta especificada pertenece a un archivo y no a una carpeta, la ruta de directorios no existe o bien el nombre que identifica al fichero no existe.

Cuando se intenta eliminar un directorio que contiene contenido, es decir, archivos o carpetas en su interior, se le preguntará al usuario que ha realizado el comando acerca de si quiere eliminar todo el contenido del directorio como lo haría el sistema operativo GNU/Linux con los privilegios de administrador y sus opcionalidades de borrado recursivas. En el caso de que el usuario no quiera borrar la carpeta con todo su contenido o bien cancele la pregunta, el comando finalizará sin realizar ninguna acción al respecto.

Ejemplos de uso:

/rmdir /carpeta1/carpeta2 => Elimina un directorio llamado “carpeta2” situado dentro del directorio “carpeta1”.

/rmdir /carpeta2 => Elimina un directorio llamado “carpeta2” situado en el directorio raíz del repositorio.

/rmdir /carpeta1/archivo1 => Error como salida del comando, no se pueden eliminar archivos, solamente directorios o carpetas.

- **/rm:** Comando para eliminar archivos (no directorios) de repositorios y sistemas de ficheros. Hemos separado ambos comandos para aclarar mejor que se trabaja con diferentes tipos de ficheros (carpetas y archivos), puesto que intentamos imitar los comandos del sistema operativo GNU/Linux.

  Requiere un argumento de entrada para procesar el borrado de archivo, además, el parámetro de entrada tiene que corresponder a una ruta asociada a un archivo y no a una carpeta o directorio, de lo contrario el sistema finalizaría con un mensaje de error informando al usuario de la incorrecta utilización del comando.

  Ejemplos de uso:

  /rm /carpeta1/carpeta2/archivo1 => Elimina el archivo denominado “archivo1” de la ruta de directorios “/carpeta1/carpeta2”.

  /rm /carpeta1/carpeta2 => Error como salida del comando, solo se pueden eliminar archivos y no carpetas.

- **/mv:** Comando para mover ó renombrar ficheros (carpetas o archivos). Funciona de la misma manera que el comando “mv” del sistema operativo GNU/Linux que se utilizan en multitud de ordenadores.

  El funcionamiento de este comando es bastante complejo y es cierto que puede ser complicado para cierto tipo de usuarios que no estén acostumbrados a trabajar con sistemas de ficheros e incluso con sistemas Linux, pero tiene un increíble potencial para mover archivos y cambiarlos de nombre de manera inmediata.

  Sintaxis de uso: /mv source_route destination_route

El comando requiere dos argumentos para funcionar de manera correcta, en el caso de que solamente se proporcione uno o ninguno, se informará al usuario acerca de que necesita determinar dos rutas concretas del sistema de ficheros. Cuando se hayan proporcionado dos rutas coherentes, el comando ejecutará el movimiento o renombramiento de fichero dependiendo de lo que el usuario quiera realizar.

Las siguientes condiciones de validez se tienen que cumplir para poder finalizar el comando de manera correcta: los ficheros de origen y destino no pueden coincidir, el fichero destino no puede ser un archivo si el fichero origen es una carpeta, la ruta origen que identifica al fichero origen debe existir y además encontrase en el repositorio y la ruta destino no es necesario que exista puesto que se puede estar especificando un nuevo nombre para el fichero.

Ejemplos de uso:

/mv /carpeta1/archivo1 /carpeta1/archivonuevo => Renombrado del archivo “archivo1” a “archivonuevo” dentro de la misma carpeta “carpeta1”.

/mv /carpeta1/archivo1 /carpeta2/archivonuevo => Movimiento y renombrado del archivo “archivo1” a “archivonuevo” situado en la carpeta “carpeta2”.

/mv /carpeta1/archivo1 /carpeta2 => Movimiento de archivo “archivo1” a la carpeta “carpeta2”.

/mv /carpeta1/archivo1 /carpeta2/archivo1 => Mismo resultado que el anterior ejemplo.

/mv /carpeta1/carpeta2 /carpeta1/carpetanueva => Renombrado de carpeta “carpeta2” a “carpetanueva”.

/mv /carpeta1/carpeta2 /carpeta3 => Movimiento de carpeta “carpeta2” a la carpeta “carpeta3”.

/mv /carpeta1/carpeta2 /carpeta3/carpetanueva => Movimiento y renombrado de carpeta “carpeta2” a la carpeta “carpeta3” con el nombre “carpetanueva”.

- **/cp:** Comando para copiar un archivo de una ruta a otra (source-destination) de la misma manera que se utiliza en el comando “mv”, pero copiándolo en lugar de moviendo de un sitio a otro.

  Sintaxis de uso: /cp source_route destination_route

  También se puede renombrar el archivo utilizando el comando si se proporciona un nuevo nombre en la ruta de destino localizada como segundo argumento del comando.

  Como todos los comandos, tienen que cumplir cierta serie de condiciones de validez para poder ejecutarse de manera correcta: la ruta origen no puede ser igual que la ruta destino, puesto que no puede haber dos ficheros iguales en la misma ruta, la ruta de destino no puede corresponder a un archivo si la de origen identifica a una carpeta y finalmente la ruta de origen y de destino tienen que existir para poder producir la copia.

Ejemplos de uso:

/cp /carpeta1/archivo1 /carpeta2/archivonuevo => Copia el archivo “archivo1” de “carpeta1” a “carpeta2” con el nombre “archivonuevo”.

/cp /carpeta1/archivo1 /carpeta2 => Copia el archivo “archivo1” de “carpeta1” a “carpeta2” conservando el mismo nombre.

/cp /carpeta1 /carpeta2 => Copia la carpeta “carpeta1” en el directorio raíz con el nombre “carpeta2”.

/cp /carpeta1 /carpeta2/carpeta3 => Copia la carpeta “carpeta1” en la carpeta “carpeta2” con el nombre “carpeta3”.

/cp /archivo1 /archivo2 => Copia el archivo “archivo1” en el directorio raíz con el nombre “archivo2”.

- **/sync:** Comando para sincronizar el repositorio local con una plataforma externa remota. En un principio solamente se permitirá la conexión con dos aplicaciones web que permiten la incorporación del repositorio mediante el sistema de control de versiones Git: Bitbucket y Github. Ambas aplicaciones están preparadas principalmente para almacenar archivos clasificados de distintos tipos.

  Normalmente estos sistemas de repositorios remotos se utilizan para almacenar archivos referentes a aplicaciones y sistemas empleados por empresas o usuarios que tengan conocimientos de programación al respecto. Pero nosotros las utilizaremos para almacenar cualquier tipo de archivo que quieran guardar los usuarios, puesto que no existe ninguna restricción al respecto en su política de uso y privacidad.

  Solamente puede ser ejecutado por los administradores del grupo de Telegram o bien el usuario si la conversación es privada.

  El comando funciona de la siguiente manera: Si existe una conexión remota ya anteriormente establecida con una plataforma, recupera los archivos necesarios para el sistema de control de versiones y vuelve a sincronizar el repositorio local con el remoto.

  Si por el contrario no existe o no se ha establecido una conexión de plataforma, el usuario debe elegir una plataforma de las propuestas e incluidas en la propia aplicación para comenzar con la sincronización del repositorio.

- **/desync:** Funcionalidad para desincronizar un repositorio con respecto a una plataforma externa, este comando solo puede ser utilizado por administradores de los grupos, puesto que la sincronización/desincronización de los repositorios son opcionalidades especiales que permiten almacenar esos ficheros en aplicaciones externas de manera pública, por tanto la decisión de establecer una sincronización o no recae sobre los administradores, puesto que son los miembros más importantes de todo el grupo (en el caso de grupo y no conversación privada).

- **/link:** Comando que proporciona un enlace de repositorio remoto en el caso de existir una sincronización específica en el grupo o conversación. Los usuarios tienen este servicio para acceder por medio de la web concreta de la plataforma externa a su repositorio remoto y realizar las operaciones de navegación que desee dentro del sistema de archivos.

Por el momento, no se debería poder realizar ninguna modificación de ficheros que no sea por medio del bot de Telegram con respecto al sistema de ficheros remoto. Esto se debe a que, al estar utilizando un sistema de control de versiones, se necesita usar mecanismos como push/pull de ficheros y pueden ocurrir conflictos entre el repositorio remoto y el local.

- **/zip:** Comando para comprimir todo el repositorio local y el sistema de ficheros en un archivo zip (.zip) para proporcionarlo en la conversación de Telegram y que todos los usuarios puedan descargarlo para verificar sus archivos y carpetas de manera sencilla y fácil. La compresión ofrece herramientas muy buenas para este tipo de situaciones en donde nuestros archivos se encuentran etiquetados en repositorios y poder proporcionar un único archivo comprimido con todo el sistema de ficheros completo.

- **/cancel:** Comando para cancelar cualquier transacción de espera o procedimiento de respuesta por parte del usuario para un comando determinado de la aplicación. Sirve para cancelar los comandos de usuario y permitir realizar otro tipo de funcionalidades y servicios que ofrece el proyecto software de Telegram.

  Un procedimiento de espera de respuesta por parte del servidor o transacción de proceso de respuesta existirá en cualquier comando que necesite una interacción con el usuario para ejercer su funcionalidad básica, como por ejemplo cuando se está realizando una subida de archivo con el comando de subida de archivos (/push) o bien cuando se desea crear una carpeta desde la interfaz gráfica de botones que ofrece nuestra aplicación (/files).

  En el caso de que el usuario no disponga de transacciones en esa conversación o chat de grupo asociado, el sistema informará con un mensaje de que no se ha podido llevar a cabo ninguna cancelación de comando.

- **/privacy:** Comando para modificar la privacidad del grupo o conversación de Telegram adecuada. Los ajustes de privacidad sirven para permitir al usuario navegar de manera privada sin que ningún usuario pueda interactuar en su propia interfaz gráfica de visualización del repositorio del comando de navegación.

  La privacidad solamente sirve para las navegaciones de los usuarios en sus respectivas interfaces de visualización de botones que quieran utilizar, sin embargo, solo un administrador puede activar/desactivar la privacidad del grupo en concreto por razones que hemos explicado a lo largo de esta memoria.

  Si el usuario tiene una conversación privada con el bot, este comando no ejecutará ninguna acción al respecto, puesto que la privacidad se encuentra siempre activa y no es necesario realizar ningún cambio para ello.

PARTE CUARTA - CONCLUSIONES.

CAPÍTULO 7 - CONCLUSIONES Y MEJORAS FUTURAS.

7.1 Conclusiones.

Tras la realización del proyecto se puede asegurar que se han conseguido todos los objetivos marcados, a pesar de haberse realizado modificaciones en alguna de las iteraciones de diseño empleadas dentro del procedimiento de la aplicación de la ingeniería del software.

Se ha desarrollado una aplicación para la plataforma de Telegram que permite dar servicio a los usuarios para su gestión y control de sistemas de archivos compartidos mediante clasificaciones especificadas por los mismos, cumple con todos los requisitos especificados y las funcionalidades determinadas en el proceso de desarrollo software planificado.

El procedimiento iterativo del desarrollo del software es muy eficiente para diseñar sistemas complejos a gran escala y que pretenden ofrecer sus servicios con un alto rendimiento. Además, nos permite conocer bien todos los escenarios, posibilidades y opcionalidades que se pueden producir en la aplicación como tal, a parte de la gran capacidad deductiva empleada con un desarrollo de modelos para su amplio entendimiento.

En cuanto a los conocimientos obtenidos tras la elaboración de este proyecto software, podemos destacar que conozco toda la API de Telegram (Python-telegram-bot) junto con toda la funcionalidad de servicios que ofrece cada una de las clases y operaciones, ampliación de nuevas técnicas en el lenguaje de programación Python (tanto la versión 2 como la versión actual 3), librerías variadas de Python (Pybitbucket, PyGithub, GitPython) para la subida de archivos a plataformas remotas y el dominio de almacenamiento de archivos gestionados por el sistema operativo.

El conocimiento de los bots conversacionales en Telegram ha sido bastante interesante para mi futuro como ingeniero informático, puesto que no sólo se trata de elaborar aplicaciones que permiten ofrecer servicios útiles a los distintos usuarios de la plataforma, si no que además hay que preocuparse de métricas importantes como el rendimiento que se va a ofrecer a los usuarios y al conjunto de grupos con los que interactuará la aplicación en sí.

Me ha servido bastante estudiar e investigar las librerías implementadas por la comunidad de Telegram que se han diseñado para realizar programas interactivos siguiendo las operaciones y tipos especificados en Telegram Bot API, así como explorar el conjunto de bots existentes dentro de la plataforma para entender bien el concepto de aplicaciones artificiales como son los bots.

En nuestro caso concreto no ha sido necesario ningún tipo de instalación de base de datos (MySQL, MongoDB…), puesto que no se ha dado el caso de almacenar datos persistentes o a recuperar para la ejecución de la aplicación. La única persistencia que existe son los propios ficheros, tanto de usuarios como los de configuración para la aplicación, que ocupan su capacidad de espacio determinada en el servidor contratado.

7.2 Mejoras futuras.

Aunque el proyecto está terminado, pueden realizarse algunas mejoras y expandir funcionalidades, estas son algunas de ellas:

- **Soporte para múltiples idiomas:** Actualmente la aplicación está implementada en español con comandos en formato inglés, sería conveniente adaptarla para que soporte múltiples idiomas, con el fin de llegar a más usuarios finales procedentes de la comunidad de Telegram. De esta manera, cualquier grupo y por tanto usuario puede comunicarse con la aplicación de manera clara y concisa.

- **Creación de una propia plataforma web:** La idea es desarrollar una aplicación web que permita la visualización de los sistemas de archivos y repositorios de los grupos gestionados por la aplicación de Telegram que hemos elaborado. Se podrán realizar mejoras en la aplicación web, como la incorporación de mecanismos de pago para el incremento de espacio para grupos concretos y sus repositorios de sistemas de archivos. De esta manera, se podría controlar mejor la creación de carpetas compartidas y un espacio de almacenamiento máximo asignado a cada una de ellas para cada repositorio remoto.

- **Gestión del mínimo espacio de almacenamiento:** Una parte importante de las aplicaciones de gestión de sistemas de ficheros es el espacio de almacenamiento existente en el servidor. Por ello, se debe desarrollar una gestión del mínimo espacio de almacenamiento para aquellos grupos en los cuales se encuentran los mismos tipos de ficheros, de esta manera la aplicación solamente almacenaría un solo archivo que sería asignado y accedido por múltiples grupos y repositorios con diferentes sistemas de archivos. Esta es una idea compleja de implementar, pero que habrá que tener en cuenta para no desaprovechar el espacio de almacenamiento que nos ofrece el hosting y por tanto disminuir los costes de espacio del servidor.

- **Adaptabilidad para otras plataformas de almacenamiento externas:** De la misma manera que hemos incorporado las opciones de sincronizar/desincronizar con las aplicaciones externas de almacenamiento Bitbucket y Github, podemos hacerlo con el resto de plataformas externas que nos ofrezcan los mismos servicios (Google Drive, Dropbox, OneDrive, Gitlab…), con las condiciones que sean necesarias, como por ejemplo la distribución del espacio del almacenamiento gestionado por pagos de transacción.

- **Extensibilidad de la aplicación:** Con esto nos referimos tanto a la incorporación de más funcionalidad y servicios en los comandos como a la ampliación del número de casos de uso de nuestra aplicación software de Telegram. Unos ejemplos claros podrían ser la gestión de subida de varios archivos al mismo tiempo, el control de la creación de varias carpetas simultáneamente, búsqueda de carpetas avanzada dentro del sistema de ficheros, la gestión de creación de fichero por medio de un enlace de Internet, el almacenamiento de una carpeta que se encuentra en otras plataformas externas, etc…

BIBLIOGRAFÍA

[1] Colaborativo, «Telegram Messenger,» 2017. [En línea]. Available: https://es.wikipedia.org/wiki/Telegram_Messenger. [Último acceso: 20 Febrero 2018].

[2] Telegram Messenger, «Telegram FAQ,» 2017. [En Línea]. Available: https://telegram.org/faq. [Último acceso: 20 Febrero 2018].

[3] Telegram Messenger, «Bots: An Introduction for developers,» 2017. [En Línea]. Available: https://core.telegram.org/bots. [Último acceso: 20 Febrero 2018].

[4] Telegram Messenger, «Telegram Bot API,» 2017. [En Línea]. Available: https://core.telegram.org/bots/api. [Último acceso: 20 Febrero 2018].

[5] Telegram Messenger, «Bot Code Examples,» 2017. [En Línea]. Available: https://core.telegram.org/bots/samples. [Último acceso: 20 Febrero 2018].

[6] Telegram Messenger, «Telegram Applications,» 2017. [En Línea]. Available: https://telegram.org/apps. [Último acceso: 20 Febrero 2018].

[7] Gabriela González, «Todo lo que tiene Telegram que no tiene Whatsapp,» 2015. [En Línea]. Available: https://hipertextual.com/2015/06/telegram-vs-whastapp. [Último acceso: 20 Febrero 2018].

[8] Alberto Ávila, «Whatsapp vs Telegram… ¿Qué aplicación de mensajería es mejor?,» 2017. [En Línea]. Available: https://www.unocero.com/noticias/apps/whatsapp-vs-telegram-que-aplicacion-de-mensajeria-es-mejor. [Último acceso: 20 Febrero 2018].

[9] Colaborativo, «Telegram Bot API,» 2017. [En Línea]. Available: https://es.wikipedia.org/wiki/Telegram_Bot_API. [Último acceso: 20 Febrero 2018].

[10] Telegram Messenger, «Telegram Bot Platform,» 2017. [En Línea]. Available: https://telegram.org/blog/bot-revolution. [Último acceso: 20 Febrero 2018].

[11] Telegram Messenger, «Telegram Bot Store,» 2017. [En Línea]. Available: https://storebot.me/. [Último acceso: 20 Febrero 2018].

[12] Abhinav Gautam, «Where to host Telegram Bots,» 2017. [En Línea]. Available: https://github.com/python-telegram-bot/python-telegram-bot/wiki/Where-to-host-Telegram-Bots. [Último acceso: 20 Febrero 2018].

[13] Jannes Höke, «Hosting your bot,» 2017. [En Línea]. Available: https://github.com/python-telegram-bot/python-telegram-bot/wiki/Hosting-your-bot. [Último acceso: 20 Febrero 2018].

[14] Cloud9 IDE, «Pricing Guide,» 2017. [En Línea]. Available: https://c9.io/pricing. [Último acceso: 20 Febrero 2018].

[15] Heroku, «Simple, flexing pricing,» 2017. [En Línea]. Available: https://www.heroku.com/pricing. [Último acceso: 20 Febrero 2018].

[16] Roman Gaponov, «How to Create and Deploy a Telegram Bot?,» 2017. [En Línea]. Available: https://djangostars.com/blog/how-to-create-and-deploy-a-telegram-bot. [Último acceso: 20 Febrero 2018].

[17] Amazon Web Services, «Conozca nuestros productos,» 2017. [En Línea]. Available: https://aws.amazon.com/es/. [Último acceso: 20 Febrero 2018].

[18] Amazon Web Services, «Precios de AWS,» 2017. [En Línea]. Available: https://aws.amazon.com/es/pricing/?nc2=h_ql_pr. [Último acceso: 20 Febrero 2018].

[19] Amazon Web Services, «Precios de Amazon EC2,» 2017. [En Línea]. Available: https://aws.amazon.com/es/ec2/pricing/on-demand/. [Último acceso: 20 Febrero 2018].

[20] Bitbucket, «PyBitbucket,» 2016. [En Línea]. Available: https://bitbucket.org/atlassian/python-bitbucket. [Último acceso: 20 Febrero 2018].

[21] GitHub, «PyGitHub,» 2017. [En Línea]. Available: https://github.com/PyGithub/PyGithub. [Último acceso: 20 Febrero 2018].

[22] GitHub, «GitPython,» 2017. [En Línea]. Available: https://github.com/gitpython-developers/GitPython. [Último acceso: 20 Febrero 2018].

[23] Sphinx, «GitPython Tutorial,» 2017. [En Línea]. Available: http://gitpython.readthedocs.io/en/stable/tutorial.html. [Último acceso: 20 Febrero 2018].

[24] Sphinx, «Welcome to Python Telegram Bot's documentation!,» 2017. [En Línea]. Available: https://python-telegram-bot.readthedocs.io/en/stable. [Último acceso: 20 Febrero 2018].

[25] GitHub, «Python-telegram-bot,» 2017. [En Línea]. Available: https://github.com/python-telegram-bot/python-telegram-bot. [Último acceso: 20 Febrero 2018].

[26] Nickoala, «telepot – Python framework for Telegram Bot API,» 2017. [En Línea]. Available: https://github.com/nickoala/telepot. [Último acceso: 20 Febrero 2018].

[27] Sergio Martínez, «Telegram - programando un bot (en Python),» 2017. [En Línea]. Available: https://geekytheory.com/telegram-programando-un-bot-en-python. [Último acceso: 20 Febrero 2018].

[28] Kylmakalle, «Starter pack to host your Python telegram Bot on Heroku for free,» 2017. [En Línea]. Available: https://github.com/Kylmakalle/heroku-telegram-bot. [Último acceso: 20 Febrero 2018].

[29] Michal Zolnieruk, «Host a python Telegram boto n Azure in < 30 minutes,» 2017. [En Línea]. Available: https://hackernoon.com/host-a-python-telegram-bot-using-azure-in-30-minutes-58f246cedf23. [Último acceso: 20 Febrero 2018].

[30] Python Software Foundation, «Miscellaneous operating systsem interfaces,» 2017. [En Línea]. Available: https://docs.python.org/2/library/os.html. [Último acceso: 20 Febrero 2018].

[31] Python Software Foundation, «High-level file operations,» 2017. [En Línea]. Available: https://docs.python.org/2/library/shutil.html. [Último acceso: 20 Febrero 2018].

[32] Adil Khashtamov, «How To Create a Telegram Bot Using Python,» 2017. [En Línea]. Available: https://khashtamov.com/en/how-to-create-a-telegram-bot-using-python. [Último acceso: 20 Febrero 2018].

[33] Adil Khashtamov, «How To Deploy a Telegram Bot,» 2017. [En Línea]. Available: https://khashtamov.com/en/how-to-deploy-telegram-bot-django. [Último acceso: 20 Febrero 2018].

[34] T. -. G. d. Alumno, «Web de la escuela de ingeniería informática,» 2017. [En línea]. Available: https://www.inf.uva.es/wp-content/uploads/2013/01/00-GuiaAlumnoTFG_2017.pdf. [Último acceso: 20 Febrero 2018].

[35] Ministerio de Trabajo e Inmigración, «BOE - XVI Convenio Colectivo Estatal de empresas de consultoría y estudios de mercados y la opinión pública,» 2009. [En línea]. Available: https://www.boe.es/boe/dias/2009/04/04/pdfs/BOE-A-2009-5688.pdf. [Último acceso: 20 Febrero 2018].

[36] Agencia Tributaria, «Tabla de coeficientes de amortización lineal,» 2015. [En línea]. Available: http://www.agenciatributaria.es/AEAT.internet/Inicio/_Segmentos_/Empresas_y_profesionales/Empresas/Impuesto_sobre_Sociedades/Periodos_impositivos_a_partir_de_1_1_2015/Base_imponible/Amortizacion/Tabla_de_coeficientes_de_amortizacion_lineal_.shtml. [Último acceso: 20 Febrero 2018].

ANEXOS

ILUSTRACIÓN 67 – COMANDO DE NAVEGACIÓN SIN FICHEROS EN EL GRUPO

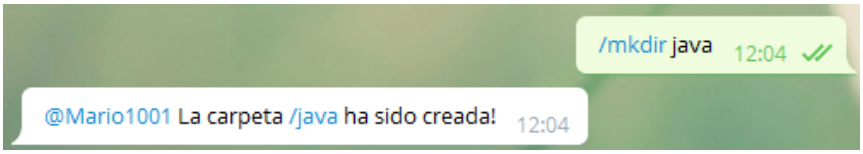


ILUSTRACIÓN 68 – COMANDO DE CREACIÓN DE CARPETA EN EL GRUPO

ILUSTRACIÓN 69 – COMANDO DE NAVEGACIÓN CON CARPETA EN EL GRUPO

ILUSTRACIÓN 70 – SUBIDA DE ARCHIVO EN UNA CARPETA

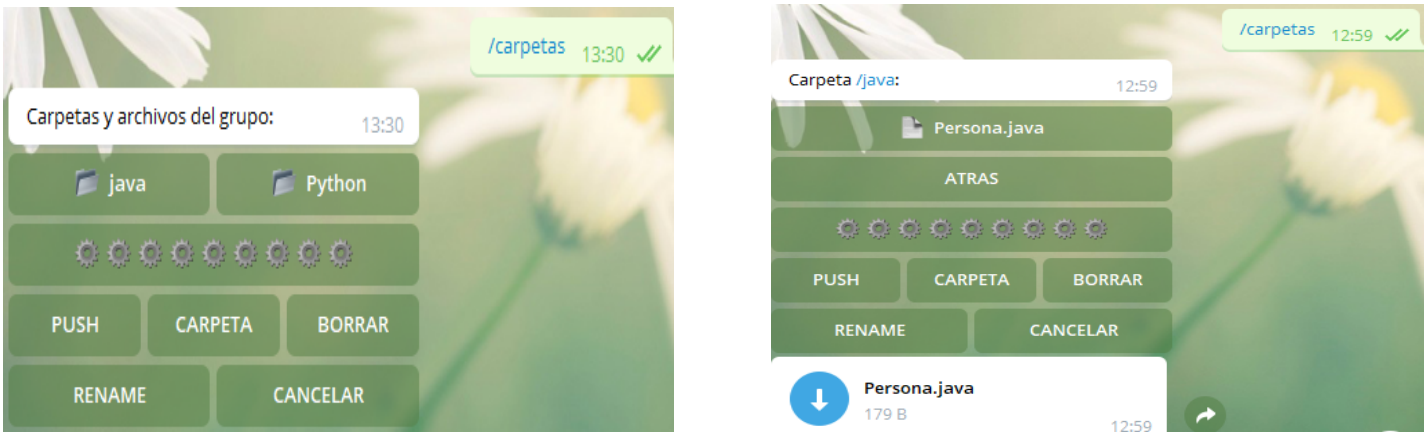


ILUSTRACIÓN 71 – CONTENIDO DE LA CARPETA CON EL ARCHIVO NUEVO SUBIDO

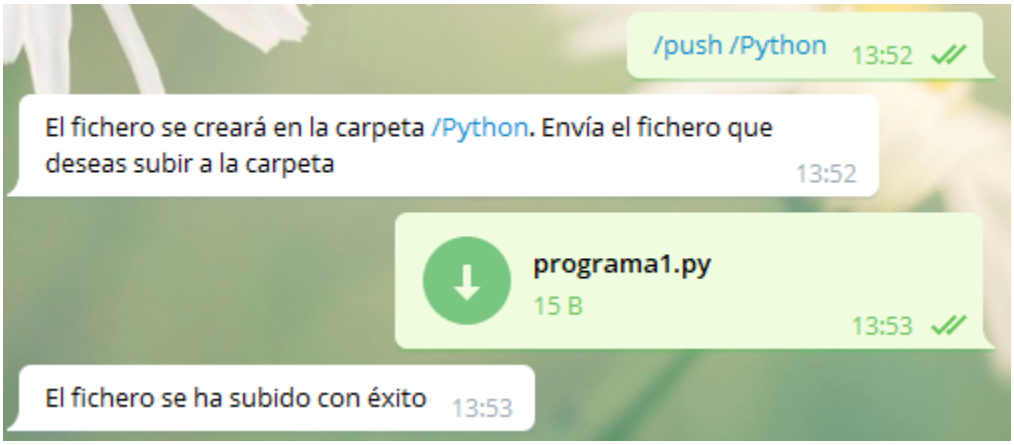


ILUSTRACIÓN 72 – COMANDO DE CREACIÓN DE ARCHIVO

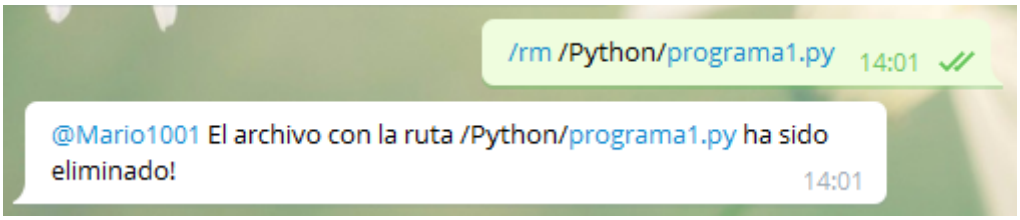


ILUSTRACIÓN 73 – COMANDO DE ELIMINACIÓN DE ARCHIVO

ILUSTRACIÓN 74 – ELIMINACIÓN DE UNA CARPETA CON CONTENIDO

ILUSTRACIÓN 75 – RENOMBRADO DE UNA CARPETA

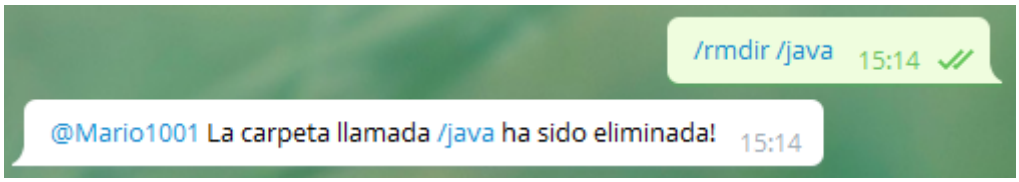


ILUSTRACIÓN 76 – COMANDO DE ELIMINACIÓN DE CARPETA

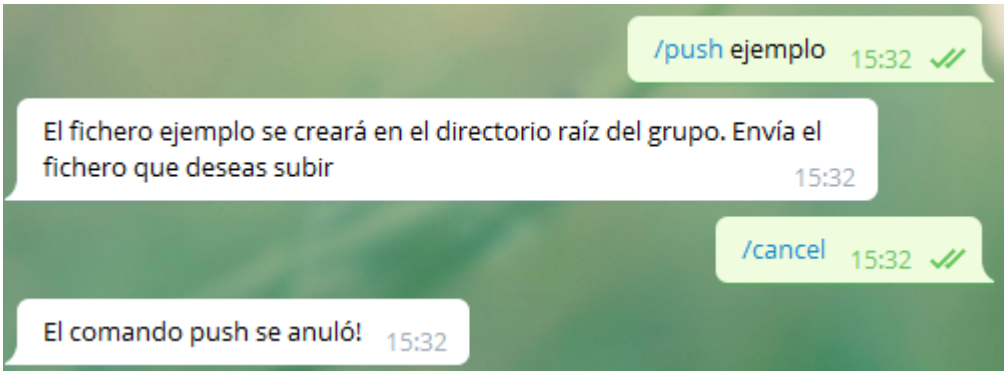


ILUSTRACIÓN 77 – COMANDO DE CANCELACIÓN DE TRANSACCIÓN

ILUSTRACIÓN 78 – COMANDO DE SINCRONIZACIÓN CON LA PLATAFORMA BITBUCKET

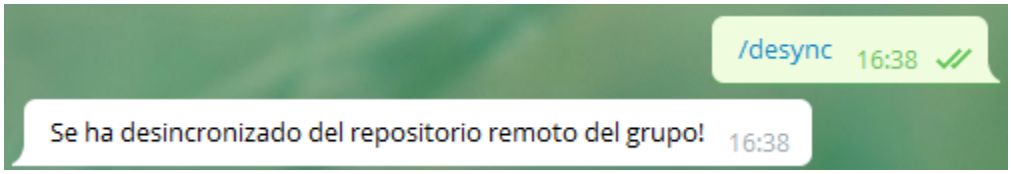


ILUSTRACIÓN 79 – COMANDO DE DESINCRONIZACIÓN

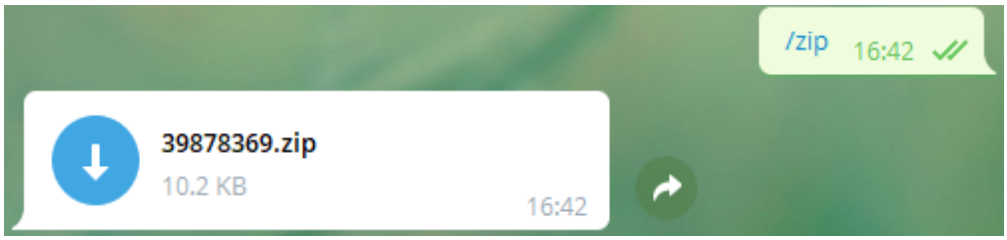


ILUSTRACIÓN 80 – COMANDO DE COMPRESIÓN DEL REPOSITORIO EN ZIP