

PORTADA

BOT PARA LA GESTION DE AFORO DE ESPACIOS

Alumno: Juan Jose Berrio

Tutor: Vicente Martínez

Resumen

El objetivo del proyecto es crear un bot que sea capaz de poder gestionar el aforo de un espacio público (gimnasios, tiendas, supermercados, etc)

El bot estará desarrollado en Python, será desplegado en la red de Telegram y tendrá una comunicación con un backend propio que también será desarrollado en Python.

Se simularán los dispositivos hardware de tornos / cámaras y detección de personas con un programa Python para llevar el conteo del aforo e inundará de información a todo el backend productivo.

Gracias a este backend productivo el bot será entrenado y será capaz de tomar decisiones productivas, resultando de gran utilidad para el usuario.

Abstract

The aim of the project is to create a bot that is capable of managing the capacity of a public space (gyms, shops, supermarkets, etc).

The bot will be developed in Python, will be deployed on the Telegram network and will communicate with its own backend, which will also be developed in Python.

The hardware devices of turnstiles / cameras and people detection will be simulated with a Python program to count the capacity and flood the whole productive backend with information.

Thanks to this productive backend, the bot will be trained and will be able to make productive decisions, which will be very useful for the user.

Tabla de contenidos

1. Introducción

- 1.1 Motivación**
- 1.2 Características y objetivos del sistema**

2. Estado del arte

- 2.1 La gestión de aforo en la actualidad**
- 2.2 ¿Qué es un bot?**
 - 2.2.1 Funcionamiento de un bot**
 - 2.2.2 Bot bueno**
 - 2.2.2.1 Chatbot**
 - 2.2.2.2 Bot Informativo**
 - 2.2.2.3 Crawlers**
 - 2.2.2.4 Game bot**
 - 2.2.3 Bot Malo**
 - 2.2.3.1 Hacker bot**
 - 2.2.3.2 Spam bot**
 - 2.2.3.3 Scrapers bot**
- 2.3 Principales Ventajas del uso de bots y sus principales campos**
 - 2.3.1 Bot en el Marketing**
 - 2.3.2 Bot para la ayuda de personas con discapacidad**
- 2.4 Entorno de programación**
- 2.5 Estructura del desarrollo**

3. Desarrollo simulador de detector de datos

4. Desarrollo del Backend productivo

5. Desarrollo del bot

1.Introducción

1.1 Motivación

Desde la llegada del coronavirus Sars-Cov-2 o también conocido popularmente como Covid-19, el mundo ha cambiado de forma radical. Desde nuestra manera de relacionarnos hasta nuestro día a día. Algo que antes no parecía un problema como era la gestión de aforo en espacios cerrados, ahora es una gran cuestión para grandes y pequeños comercios desde gimnasios, cines, teatros o incluso para poder realizar una pequeña reunión con familiares y amigos. Anteriormente a la llegada de la pandemia, la gestión de aforo no era una medida sanitaria de urgencia y mucho menos un problema para los comercios en general como para sus usuarios, cierto es que ya existía un control de aforo para ciertas actividades de ocio como pueden ser por ejemplo conciertos, o una presentación de teatro, pero no era una cuestión sanitaria como lo es hoy en día, era simplemente un criterio de si se habían vendido todas las localidades o si había disponibilidad de asistir a la actividad.

El Sars-Cov-2 lleva ya 2 años entre nosotros. Desde su aparición en China no han dejado de existir diferentes variantes y mutaciones de este virus el cual ha producido que cada día existan más restricciones a nivel global en todas las sociedades del mundo. Uno de los grandes problemas en la sociedad es el control de aforo en todas las actividades ya que no existe a día de hoy sistemas de recuento que nos ayuden a resolverlo de forma sencilla. Estos sistemas nos ayudaran para el Sars-Cov-2 y otro virus o pandemias que puedan existir en el mundo así para paliar problemas que ocurran por el abarrotamiento de gente dentro de un lugar.

Una solución para poder medir el aforo de un espacio es mediante un Bot informativo o chatbot, el cual nos dará a tiempo real la información necesaria de capacidad de cierto lugar que queramos consultar, quitándole el trabajo al personal del comercio de estar haciendo cuentas tanto positivas como negativas de las personas que estén asistiendo a dicho espacio.

Un Bot es un software informático destinado a la realización de tareas repetitivas con cierta inteligencia, como tareas cotidianas que hacen las personas en su día a día.

Uno de los bots más populares es un ChatBot, implementado en muchísimas empresas y sitios de Internet, se basa en resolver preguntas basándose en inteligencia artificial programada. Destaca por la manera de poder mantener una conversación con una persona humana, como también por la manera de poder ejecutar ciertas ordenes que le enseñemos previamente.

En el caso de la gestión de aforo en ciertos lugares, el ChatBot es una gran herramienta que puede ayudar a facilitar las operaciones de conteo de personas, teniendo así un conteo a tiempo real de la capacidad de aforo de ciertos lugares como puede ser en un gimnasio. Los usuarios que quieran acceder al gimnasio a determinada hora podrán consultar a través del ChatBot la capacidad y tener así una información importante con la cual podrán saber si el gimnasio tiene el aforo completo o aún tiene capacidad para acceder a él, ayudándole al usuario a tener una gestión de su tiempo de ocio en su día a día.

1.2 Características y objetivos del sistema

El objetivo de este trabajo es la creación de un bot que tenga inteligencia artificial para poder ser capaz de dar información al usuario sobre la gestión de aforo de un gimnasio, pero también puede ser utilizado en cualquier espacio público en el que la gestión de aforo sea una necesidad importante.

Los componentes del sistema serán los siguientes:

- ***Simulador de detectores de datos:***

Este programa desarrollado en Python recopilara los datos necesarios que serán almacenados en el backend. Estos datos serán recogidos gracias a la simulación de detectores hardware como cámaras o tornos que estarán localizados en el espacio en donde se quiere llevar el conteo de personas.

- ***Backend***

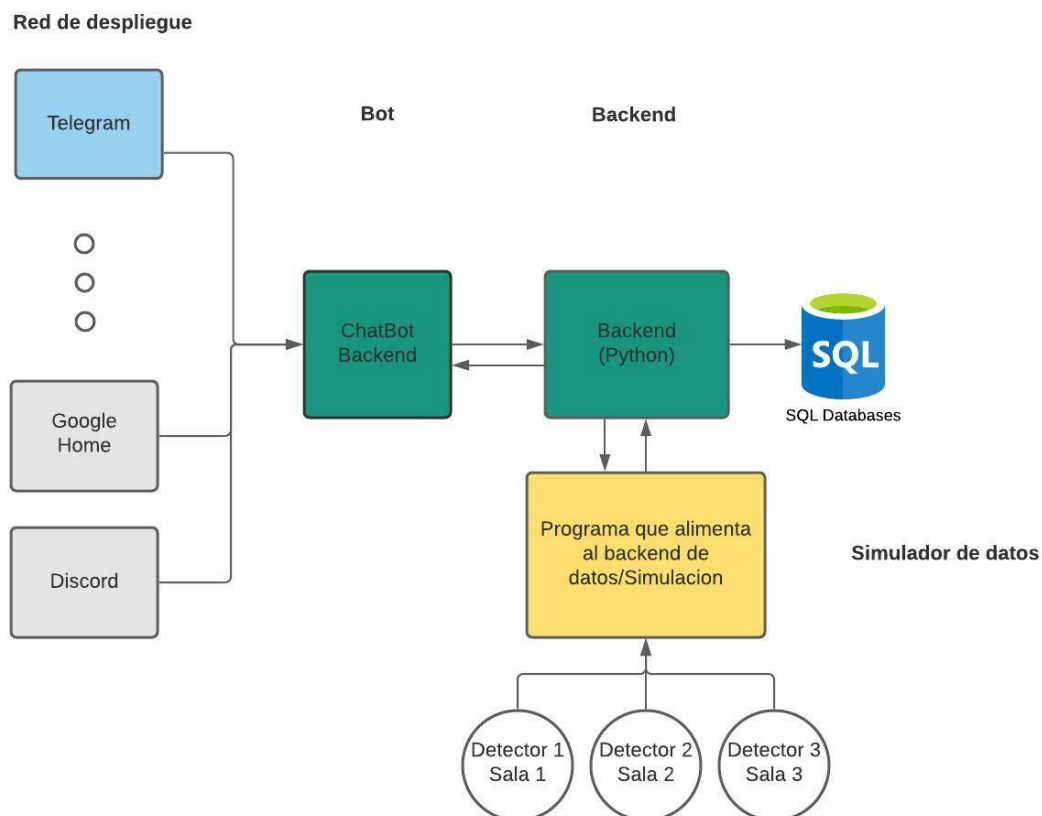
El bot será alimentado con datos simulados para el objetivo de este proyecto con la finalidad de no incurrir en gastos de Hardware innecesarios. Como contadores de personas o dispositivos de presencia que inundan de datos al servidor. Estos datos simulados serán almacenados en un Backend productivo desarrollado en Python.

- ***Bot***

Aplicación software escrita en Python que será encargada de gestionar los mensajes desde una aplicación de mensajería donde este desplegado el bot. Este será capaz de informar al usuario del aforo de un espacio público, gracias a los datos que han sido simulados y almacenados por los sensores.

- **Red de despliegue**

Entorno donde el bot será desplegado y en donde el usuario podrá solicitar en el momento que lo desee el aforo de cierto lugar que quiera saber. Telegram ha sido elegida aplicación de despliegue.



2. Estado del arte

2.1 La gestión de aforo en la actualidad

La gestión de aforo no había sido una necesidad primordial para un espacio público cerrado hasta la llegada del Sars-Cov-2 o también conocido popularmente como Covid-19. Esto cambió radicalmente siendo ahora necesario un conteo de aforo, para poder cumplir las medidas sanitarias que se establezcan según comunidad autónoma o cada país.

Anteriormente en algunos lugares muy esporádicamente se realizaban control de aforo mediante colas de espera, suponiendo así tener un mejor orden para la organización que lo implementaba, pero hoy en día el conteo de aforo se ha ido implementando progresivamente hasta hacerlo una necesidad.

Actualmente el conteo de aforo se realiza estableciendo a una persona en la entrada del establecimiento con un contador en la mano, sumando o restando personas según accedan o salgan del establecimiento. Esta forma de realizarlo conlleva gastos de tiempo, staff y económicos haciendo que esta forma sea bastante ineficiente a largo plazo.

Implementando esta manera de conteo de una forma más digitalizada, supondría un gran avance y ahorro de recursos, dándole así una funcionalidad más eficiente al conteo de personas.

Un Bot junto con una arquitectura tecnológica de conteo sería una de muchas soluciones tecnológicas que harían más eficiente el recuento de personas.

2.2 ¿Qué es un Bot?

Un bot es una aplicación software la cual está programada para realizar tareas de un ser humano, estas tareas en muchas ocasiones son repetitivas o monótonas y pudiendo ser realizadas en un menor tiempo que si las hiciera un ser humano, ahorrando de esta manera unos recursos de tiempos considerables.

Entre las diferentes tareas que un bot puede realizar destacan: Mantener conversaciones interactivas con el usuario, ofrecer información requerida por el usuario llegando incluso a ser motores de búsqueda ayudando a indexar contenido para una búsqueda en el navegador. Por otro lado, también pueden llegar a tener intenciones maliciosas, como causar daños o comprometer datos personales del usuario.

2.2.1 Funcionamiento de un bot

El funcionamiento de un bot se compone principalmente de algoritmos que le permiten realizar las funciones para las cuales ha sido diseñado. Según el diseño del bot puede haber:

Bot conversacional: Bot que funcionan por reglas y son capaces de interactuar con las personas

Bot con independencia intelectual: Bot que, gracias al aprendizaje automático, son capaces de aprender de las conversaciones con los humanos.

Bot con inteligencia artificial: Son una combinación del bot conversacional y del bot con independencia intelectual, usan técnicas y herramientas de procesamiento de lenguaje natural para poder sustentarse.

Un bot funciona en la red con la capacidad de comunicarse con el usuario, para ello utilizan servicios de mensajería como Telegram o Discord.

2.2.2 Bot bueno

Actualmente se puede clasificar un bot según su funcionalidad, pero también si es bueno o malo, teniendo así una perspectiva más general del bot y de su comportamiento.

Un bot por general tiene como funcionalidad realizar tareas automáticas con el fin de ayudar a las personas en ciertas tareas cotidianas, como puede ser reservar un restaurante, ofrecer cierta información que el usuario desea saber sobre un tema en concreto, por ejemplo “¿Qué día hace hoy?”, “¿Cuáles son las últimas noticias?” ofreciendo así respuestas mucho más rápidas que las que un ser humano podría ofrecer y con una gran ventaja diferenciada, que se puede consultar al bot en cualquier momento del día.

Por otro lado, son capaces de aprender de conversaciones con los usuarios pudiendo así tener conversaciones más precisas e interactivas. También son capaces de editar textos de forma automática ayudando así al autocompletado de texto en la búsqueda en un navegador o también en la corrección de faltas de ortografía. Todas estas funcionalidades descritas anteriormente tienen la finalidad de ayudarnos y que los bots sean un complemento de gran utilidad en nuestro día a día.

2.2.2.1 Chatbot

Los chatbots son los bots más comunes y más utilizados hoy en día de todos los tipos de bots que puede haber.

Un chatbot se puede definir como un servicio de mensajería en línea, es decir como una especie de asistente que es capaz de comunicarse con los usuarios a través de mensajes de texto. Se componen en su gran mayoría de algoritmos de inteligencia artificial, utilizan técnicas de lenguaje natural para poder entender a los seres humanos y aprender sus hábitos o gustos.

Entre estas técnicas destaca el procesamiento de lenguaje natural, la comprensión de lenguaje natural y la generación de lenguaje natural.

Procesamiento de lenguaje natural:

Esta técnica se centra en las oraciones y las palabras del usuario. El objetivo es hacer tanto un análisis léxico, como sintáctico de las palabras que el usuario proporciona, para poder corregir errores ortográficos antes de determinar el significado de estas.

Compresión de lenguaje natural:

Esta técnica se centra en que el chatbot determine el significado de la palabra una vez corregida de posibles errores ortográficos. Son utilizados algoritmos para determinar sinónimos de las palabras y construir un dialogo para que el chatbot pueda responder una vez haya entendido el significado de la palabra.

Generación de lenguaje natural:

Esta técnica se centra en que el chatbot puede consultar memorias o repositorios de datos sobre expresiones de lenguaje humano, para así poder ofrecer un diálogo más natural y realista.

La principal función de un chatbot es solucionar dudas o preguntas frecuentes, mediante respuestas automáticas en un breve periodo de tiempo, ofreciendo así una buena experiencia al usuario, pudiendo ser consultado en cualquier momento del día que se desee.

2.2.2.2 Bot informativo

Un bot informativo es un tipo de bot el cual se encarga de dar información que previamente se ha gestionado en canales informativos como pueden ser periódicos digitales o redes sociales, como Instagram o Facebook.

Estos bots recopilan información a tiempo real, ofreciendo una información informativa a los usuarios. La gran diferencia con los chatbots es que estos solo informan como su nombre indican, mas no son capaces de mantener una conversación como si era el caso de los chatbots.

No por eso dejan de ser menos útiles, en lo contrario se adaptan también a las necesidades de los usuarios que quieren tener una información cada cierto tiempo, sin necesidad de ellos mismo solicitarlo.

2.2.2.3 Crawlers

Un bot Crawler se basa en la búsqueda de datos en internet, son conocidos como arañas rastreadoras web. Estos bots funcionan como motor de búsqueda y rastreador, destacan por que son capaces de acceder a un sitio web y poder obtener datos relevantes del mismo para luego darle una finalidad como motor de búsqueda.

Un motor de búsqueda es capaz de proporcionar enlaces según las búsquedas del usuario, generando de esta manera una lista de las páginas web que el usuario más ha visitado o que más relevancia han tenido para él. La finalidad es que el usuario pueda encontrarlas de un forma más rápida y sencilla.

2.2.2.4 Game bot

Un Game bot es un tipo de bot el cual su principal función es comportarse como un jugador más en un videojuego, como si fuera este un ser humano.

Es decir, puede combatir, ser competitivo como un jugador más. Gracias a ellos los jugadores pueden entrenarse y mejorar sus habilidades en el videojuego sin necesidad de contar con un ser humano como rival, ya que estos bots están programados para realizar las funcionalidades del videojuego con un cierto nivel de competitividad. Son muy utilizados en juegos online para que los jugadores puedan tener un primer contacto con el videojuego o también para completar equipos de jugadores que estén incompletos.

2.2.3 Bot malo

No todos los bots tienen una finalidad buena. Existen los bots malos los cuales tienen como finalidad enviar spam, espiar, abrir puertas para la propagación de virus y gusanos en el dispositivo, robar datos personales como contraseñas o datos bancarios mediante ataque Phishing o también llamado suplantación de identidad, ataques DDoS (ataques de denegación de servicio) causando grandes problemas de acceso a un servicio o recurso.

Todas estas finalidades maliciosas son llevadas a cabo por los bot maliciosos gracias a que una de las principales características de estos, es que son muy difíciles de detectar, ya que están muy bien camuflados y adaptados hoy en día, por ejemplo pueden estar en forma de descarga en redes sociales o a través de enlaces que llegan a los correos electrónicos, o también mediante falsas advertencias engañando así al usuario y dando vía libre para que el bot pueda realizar sus funciones maliciosas.

2.2.3.1 Hacker bot

Un hacker bot es un tipo de bot diseñado para la realización de actividades maliciosas como expandir virus para infectar ciertos dispositivos, robar credenciales personales como contraseñas o cuentas bancarias. Estos bots son un gran problema para los usuarios que padecen sus ataques, ya que su forma de actuar es muy discreta y disimulada pudiendo así engañar fácilmente a los usuarios. Actúan mediante acciones que parecen buenas como pueden ser darle un clic a un enlace que ofrece el bot sobre una canción que el usuario ha consultado, así infectando el dispositivo del usuario incluso a revelar datos personales del usuario.

Otra gran característica de los mismo es que son capaces de hacer combinaciones de datos y contraseñas previamente filtradas por personas maliciosas, con el objetivo de encontrar la combinación correcta para el acceso a cuentas personales, y una vez con ellas, los delincuentes pueden realizar compras no autorizadas e incluso llegando a pedir un rescate para la devolución de las cuentas.

Aparte de estos ataques que un hackerbot puede llegar a realizar, hay uno que destaca por gran capacidad de ataque, este es el ataque a servicios web, mediante ataques DoS o DDoS. Los ataques DoS son el resultado de una saturación de la red, llegando a una caída repentina del servicio.

2.2.3.2 Spam bot

Un spam bot, es un bot cuya finalidad es el envío masivo de correo spam, esto suele ser muy molesto para el usuario, ya que es una información sin contenido relevante, incluso llegando a filtrarse algún código malicioso en el propio spam.

Estos bots están programados para rellenar formularios en páginas web o redes sociales para así poder enviar de forma masiva contenido spam o publicidad innecesaria, esto gracias a que tienen unas reglas predefinidas por los atacantes para el envío de Spam a través de correos electrónicos o de publicidad innecesaria en webs o redes sociales.

Redes sociales como ciertas páginas webs utilizan Captcha o desafíos que un ser humano podría realizar fácilmente, pero un bot no, para así evitar el envío spam mediante un bot.

2.2.3.3 Scrapers bot

Los Scrapers bots son bots que están basados en hacer scraping de datos de un sitio web o también conocida como técnica de extracción de datos.

Son capaces de extraer contenido importante de una web como pueden ser, datos de clientes, información sobre productos, precios, ofertas. Todo esto con la finalidad de limitar las ventajas de un sitio web frente a sus competidores, o conocer información confidencial sobre el sitio web.

Actúan gracias a peticiones HTTP GET del sitio que quieren extraer los datos, este les responde generando documentos que luego serán analizados por el creador del bot para así poder extraer los datos mas relevantes que considere.

Al igual que el spam bot, los Scrapers bot pueden ser reducidos gracias a técnicas como Captcha para así reducir su actividad de scraping.

2.3 Principales Ventajas del uso de bots y sus principales campos

En la mayoría de los casos, el uso de bots hoy en día tiene muchos más aspectos positivos como ventajas en su uso con respecto a aspectos negativos. Viendo los tipos de bots anteriormente y poniendo principal atención en cómo prevenir y cuidarnos de las consecuencias de los bots malos, son muchas más las ventajas que desventajas que un bot puede aportar al ser humano en sus tareas diarias e incluso ayudar a potenciar áreas en los negocios y en la atención al cliente.

A continuación, se van a detallar áreas en las cuales en los últimos años los bots han desarrollado su mayor potencial ayudado a descubrir o expandir nuevas funcionalidades en las necesidades de los seres humanos.

2.3.1 Bot en el Marketing

El Marketing siempre ha sido la herramienta principal mediante la cual las empresas pueden generar y promocionar sus valores dentro del mercado. Sabiendo que el objetivo de toda marca empresarial es que su producto sea consumido y genere gran interés en los usuarios, en el marketing es donde las empresas cada año invierten una gran parte de su presupuesto empresarial. Gracias a la digitalización de las cosas y que hoy en día una gran parte de todos los seres humanos pueden y saben acceder a internet mediante un dispositivo, el marketing ha tenido que evolucionar hacia esta nueva tendencia de la digitalización que ya es como una necesidad imprescindible hoy en día.

Es por esto por lo que los bots han evolucionado y han ayudado mucho en esta área, gracias a la digitalización se pueden hacer bots efectivos para que las empresas puedan sacar el máximo provecho de sus productos, así como tener una alta rentabilidad en sus cuentas anuales.

Los bots en el ámbito del marketing digital son capaces de ayudar a la empresa a conocer más a fondo a sus principales clientes, mediante respuestas a sus dudas o necesidades, así como también sabiendo cuáles son sus mayores intereses sobre productos o servicios concretos. Estos bots también ofrecen esta información sobre los productos o servicios a toda hora del día, todos los días del año, ofreciendo de esta manera una alta disponibilidad. Pero sin duda una de sus mayores funcionalidades es la optimización de costes para la empresa, ya que un bot puede servir como un agente comercial, o como un agente de atención al cliente, proporcionando así un ahorro considerable en recursos económicos y ofreciendo ventajas ya mencionadas anteriormente como la alta disponibilidad, aparte que también pueden generar promociones sobre los productos y enviar sus respectivas notificaciones a los usuarios mediante una comunicación previamente programada. Todas estas funciones descritas anteriormente deben de conllevar una previa implementación así como

mantenimiento de este, pero una vez realizado la efectivada de tener un bot en el campo del marketing es muy satisfactorio por todos los resultados positivos que produce.

2.3.2 Bot para la ayuda de personas con discapacidad

Los bots como bien ya se ha comentado tienen como funcionalidad principal ayudar a las personas a realizar tareas y hacer más fácil su día a día. Es donde en la ayuda para las personas discapacitadas los bots más han evolucionado e impactado últimamente de gran manera, siendo esta un área de continua mejora, ya que la comunicación e integración de las personas discapacitadas con la sociedad es una de las mayores prioridades hoy en día para gobiernos e identidades sociales y poder así reducir una brecha digital que aún existe hoy en día con estas personas.

Hoy en día existen asistentes virtuales muy conocidos como Siri de Apple disponible en todo dispositivo de sistemas IOS de Apple, Alexa como asistente virtual de Amazon o Cortana de Microsoft siendo estos los más famosos hoy en día en el mercado, pero hay muchos más menos conocidos, pero también de gran utilidad. Todos estos dispositivos provén ciertas funcionalidades como conocer las últimas noticias del día, conocer la previsión meteorológica, hacer un pedido online e incluso rastrearlo, y son de gran complemento en el día a día, para las personas con discapacidades visuales les sirve como una gran ayuda para poder realizar las tareas previamente dichas. En el caso de Siri de Apple es una de las aplicaciones con más avances últimamente, ya que en cualquier dispositivo Apple que se pueda llevar a mano se puede realizar llamadas a un contacto, transcribir un texto a voz o viceversa, poder crear alarmas o recordatorios, e incluso contar chistes y ser interactivos con los usuarios. En el caso de Cortana esta más enfocado a ordenadores y sus funciones van más enfocadas a usos con el ordenador, como abrir aplicaciones. Todo lo comentando anteriormente son avances que se han realizado en la última década y que han ayudado a las personas con alguna discapacidad bien sea visual, auditiva o cualquiera de esta índole a poder llevar mejor sus tareas diarias, a pesar de todos estos avances, siguen habiendo muchas necesidades que cubrir como por ejemplo la navegación en una página web o una aplicación y poder conocer y utilizar todas sus funcionalidades, es por esto que se sigue investigando para poder que la tecnología mediante bots inteligentes pueda ser de gran uso para estas personas.

Uno de los mayores avances en esta área es SayOBO un chatbot adaptado para personas con problemas de visión el cual tiene como objetivo facilitar el acceso a información o trámites digitales a personas con discapacidades visuales. Hoy en día tecnologías descritas anteriormente como Siri, Alexa o Cortana cuentan con una programación que convierte el contenido web en voz sintetizada, sin embargo, en una web o aplicación no todo es texto y hay funcionalidades como rellenar un formulario o navegación entre secciones, que no llegan a abarcar estas tecnologías. Es por esto que el chatbot SayOBO puede facilitar el acceso y navegación de un sitio web o aplicación,

mediante la creación de un escenario digital sin ninguna barrera y que contenga toda la información necesaria para que la persona con discapacidad visual pueda realizar cualquier función en la web o en la aplicación que este integrado este bot.

2.4 Entorno de programación

En este apartado se comentará las herramientas utilizadas para la realización de este trabajo.

Como entorno de trabajo se ha utilizado PyChart, ya que después de una investigación entre la elección de IDE's Pychart tiene un manejo fácil, proporciona información clara acerca del código y también está basado en la comunidad JetBrains que proporciona soporte así como información sobre módulos y paquetes.

El lenguaje elegido ha sido Python. Se podría haber elegido Java pero por sencillez y aprender un reto de aprender un lenguaje nuevo no visto en la carrera, se ha elegido **Python 3.9**

2.5 Estructura del desarrollo

Se ha identificado, como se vio en el diagrama de arquitectura, 4 componentes:

- **Simulador detector de datos** -> Esto debería de ser la API de los 3 programas, pero en este caso es un programa simulador que simula los detectores de presencia del programa.
- **Backend** -> Es un programa desarrollado en Python donde su objetivo principal es ser un servidor donde tenga diferentes endpoint para obtener los datos que requiere el Bot
- **BBDD** -> En este caso es la BBDD que registra las entradas, salidas y los datos de las salas, no se va a realizar en el proyecto porque se sale del alcance del proyecto. Pero en un entorno productivo debería de integrarse.
- **Bot** -> Es un Bot escrito en Python y desplegado en plataforma e integrado en un cliente de mensajería que es Telegram.

3. Desarrollo simulador de detector de datos

La realización de este proyecto se compone de varias etapas de desarrollo que se van a comentar a continuación.

La primera etapa ha sido la realización de un simulador de detector de datos, simulando el comportamiento de unos dispositivos hardware como cámaras o sensores.

Previamente se hizo un análisis de las diferentes etapas de desarrollo del proyecto y se midieron costes como medidas de tiempo para realizarlo. Analizando la primera etapa del proyecto la cual era la obtención de los datos, que serian el principal componente del bot, se llegó a conclusión que la compra de dispositivos hardware de cámaras como sensores de detector de personas llevaría un coste elevado para un proyecto de final de carrera, no obstante, y con la idea de realizar un buen trabajo se plantea la realización de un simulador programado desde cero, el cual podrá aportar los datos necesarios para el bot. Este simulador se comportará como un espacio cerrado simulado con sus respectivas salas, y cada sala con sus respectivos sensores, los cuales generan datos de forma aleatoria que luego podrán ser consultado por el bot, en otras palabras, gracias al simulador se puede tener un sustituto de un dispositivo hardware de detección como puede ser una cámara o un sensor.

Este simulador al igual que un detector tienen la capacidad hardware de detectar cuando se produce una entrada o una salida de un espacio cerrado controlado. Esto en contexto hardware estaría compuesto por un conjunto de láseres contiguos de tipo entrada y de tipo salida, en los cuales si se produce un corte primero por el de tipo entrada y luego por el de tipo salida daría como resultado una entrada y posteriormente un aumento del recuento en el contador y del mismo modo, pero a la inversa si se produce un corte primero por el de tipo salida y luego en el de tipo entrada se estaría realizando una salida, con su posterior disminución en el contador.

El simulador contiene las siguientes clases que se describirán a continuación.

(Diagrama)*

Dentro de la carpeta Emulator, hay una carpeta llamada configurationEmu. En ella se encuentran las clases **emulatorConfig** y **salasConfig**, las cuales tendrán la configuración inicial del proyecto, como la creación de las salas con sus respectivas características.

En la clase **emulatorConfig** se compone por la configuración inicial, en la cual para poder emular la simulación se ha diseñado un reloj en el cual se configura los ciclos de reloj que tendrán una duración de x segundos, cada x segundos se generara un nuevo evento. También en esta clase se definen el número máximo de sensores como de salas tiene el programa y las acciones de entrada y salida.

En la clase **salasConfig** se define la creación de salas con sus características al igual que la creación de sensores con sus respectivas características.

```
from ..modelEmu.sala import Sala
from ..modelEmu.sensor import Sensor

salas = [
    Sala ("Sala Norte", 0, 8, 0),
    Sala ("Sala Oeste", 1, 7, 0),
    Sala ("Sala Sur", 2, 6, 0),
    Sala ("Sala Sorento", 3, 7, 0),
    Sala ("Sala Espacio", 4, 6, 0),
]

sensores = [
    Sensor(0, "Sala 1 - Sensor 1", salas[0]),
    Sensor(0, "Sala 2 - Sensor 2", salas[1]),
    Sensor(0, "Sala 3 - Sensor 3", salas[2]),
    Sensor(0, "Sala 4 - Sensor 4", salas[3]),
    Sensor(0, "Sala 5 - Sensor 5", salas[4]),
]
```

(salasConfig.py)

Previamente a esta clase salasConfig se han creado la clase **sala** y la clase **sensor** las cuales contienen los atributos que se han considerado como id, nombre, capacidad y ocupación en el caso de la clase **sala** y para la clase **sensor** id, nombre y sala.

También se definen las funciones de entrada que conlleva aumentar ocupación y la función de salida que conlleva disminuir ocupación y funciones de comprobación previa, de si hay o no ocupantes en la sala tanto para poder ejecutar una salida y luego disminuir su ocupación, así como también para poder ejecutar una entrada y actuar en consecuencia.


```

from Backend.emulator.configurationEmu.emulatorConfig
import ConfigEmu

import json

class Sala:

    # init method or constructor
    def __init__(self, nombre, id, capacidad, ocupacion):
        self.nombre = nombre
        self.id = id
        self.capacidad = capacidad
        self.ocupacion = ocupación
        self.sensor = None

    def aumentarOcupacion(self):
        self.ocupacion = self.ocupacion + 1

    def disminuirOcupacion(self):
        self.ocupacion = self.ocupacion - 1

    def __str__(self):
        return self.nombre + "\n      Id: " + str(self.id) +
"      Capacidad: " + str(
        self.capacidad) + "      Ocupacion: " +
str(self.ocupacion)

```

```

def canDoAccion(self, accion):
    if accion == ConfigEmu.ENTER:
        if self.ocupacion < self.capacidad:
            return True
        else:
            return False
    else:
        if self.ocupacion == 0:
            return False
        else:
            return True

def executeAccion(self, accion):
    if (accion == ConfigEmu.ENTER):
        self.aumentarOcupacion()
    else:
        self.disminuirOcupacion()

def toJson(self):
    return "{\"nombre\": \"" + str(self.nombre) +
"\", \"id\": " + str(self.id) + ", \"capacidad\": " +
str(self.capacidad) + ", \"ocupacion\": " +
str(self.ocupacion) + "\"}"

```

(sala.py)

```
import json

class Sensor:

    # init method or constructor
    def __init__( self, id, nombre, sala):
        self.id = id
        self.nombre = nombre
        self.sala = sala

    def __str__(self):
        return self.nombre + "\n      Id: " + str(self.id)
+ "      Sala asociada: " + str(self.sala)

    def entradaPersona(self):
        self.sala.aumentarOcupacion()

    def salidaPersona(self):
        self.sala.disminuirOcupacion()

    def canDoAccion(self, accion):
        return self.sala.canDoAccion(accion)

    def executeaccion(self, accion):
        return self.sala.executeAccion(accion)

    def toJson(self):
        return json.dumps(self.__dict__)
```

(sensor.py)

Hay una clase llamada **salasManager** dentro del paquete **managerEmu**, la cual actúa como gestor de todas las salas del programa. Se ocupa también de realizar las acciones necesarias cuando se detecta un movimiento en un sensor de una sala y en la siguiente fase del proyecto se programará en esta clase las operaciones que realizará el bot.

```
from Backend.emulator.configurationEmu.salasConfig import
salas, sensores

class SalasManager:

    def __init__(self, salasfield):
        self.salas = salasfield
        for index in range(0, len(self.salas) - 1):
            sala = self.salas[index]
            sala.sensor = sensores[index]

    def detectarMovimiento(self, idSala):
        salaAux = self.salas[idSala]
        salaAux.ocupacion = salaAux.ocupacion + 1

    def getPeopleNumber(self, idSala):
        return self.salas[idSala].ocupacion

    def infoSalaJson (self, idSala):
        return str(self.salas[idSala].toJson())

    def printStatus(self):
        for salaAux in self.salas:
            print(salaAux)
```

El programa simulador consta del siguiente código:

```
def job(self):
    sensorNumber = self.numbersensor()
    accion = self.enterorexit()
    if self.candoaccion(sensorNumber, accion):
        self.executeaccion(sensorNumber, accion)

def execute(self):
    while True:
        self.job()
        time.sleep(ConfigEmu.RELOJ)
```

(emulator.py)

El cual se encarga en primer lugar de generar un evento o job por cada pulso del reloj previamente configurado, esta generación del job se realiza en la función **execute**. Mediante la función def **job** se genera un sensor y se genera una acción de entrada o salida, estas generaciones se crearán de forma aleatoria mediante la llamada a **randrange**, seleccionara uno de los sensores previamente configurados y ejecutara una entrada o una salida según el número elegido aleatoriamente.

```
def numbersensor(self):
    salaNumber = randrange(ConfigEmu.MAX_SENSOR)
    return salaNumber

def enterorexit(self):
    typeEnter = randrange(2)
    return typeEnter
```

En el job también se declara comprobaciones de cuándo se pueden ejecutar una acción, esto se realiza con la llamada a **candoaccion**.

En donde se ejecutará una acción de entrada o salida si las configuraciones previas del simulador lo permiten, es decir si no se supera el número máximo de ocupación, o que haya un número mínimo para poder disminuir la ocupación. En cada evento o job que se ejecuta, se saca de cada sensor generado la sala a la cual está asociado y se ejecuta la acción bien sea entrada o salida de la sala. Esto se realiza con la llamada a la función **executeaccion**. Devolviendo un mensaje de error cuando se genera una acción de la llamada randrange que no se pueda realizar, debido a las configuraciones anteriormente expuestas.

```
def executeaccion(self, sensorNumber, accion):
    sensores[sensorNumber].executeaccion(accion)
    self.salasManager.printStatus()

def candoaccion(self, sensorNumber, accion):
    sensor = sensores[sensorNumber]
    if sensor.canDoAccion(accion):
        self.printstatusaccion(accion, sensorNumber)
        return True
    else:
        self.printerroraccion(accion, sensor)
        return False
```

(emulator.py)

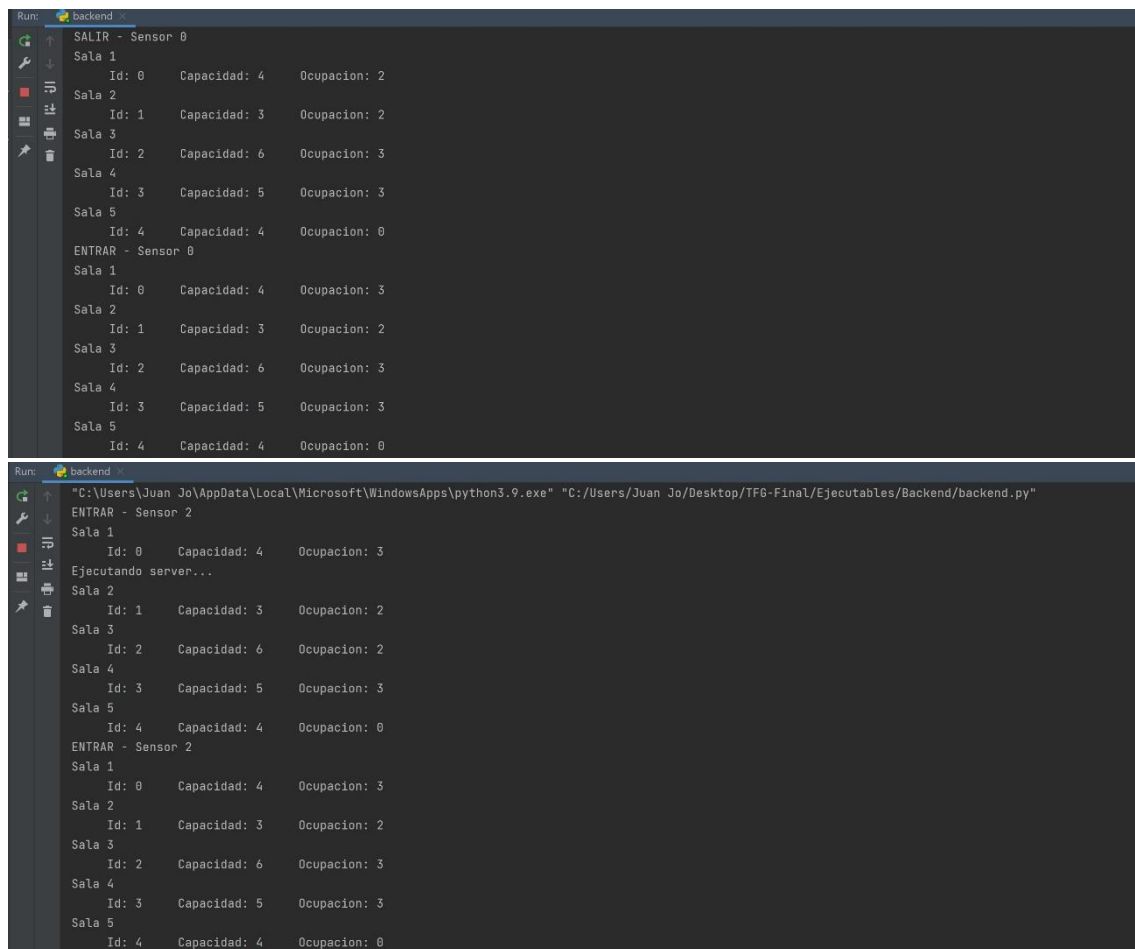
Por otro lado, se ha definido una función warmup, la cual nos devuelve un número de personas, y a partir de esto se realiza la llamada al simulador, todo esto con el objetivo que cuando se llame al simulador ya tenga un numero predefinido y sea más interactiva la inicialización del programa al igual que sucedería en la realidad. Cuando un usuario quiera consultar cuantas personas hay en una sala en cierto momento concreto ya hay personas en dicha sala previamente.

```
@staticmethod
def warmup(salas):
    for index in range(0, len(salas) - 1):
        sala = salas[index]
        sala.ocupacion = randrange(sala.capacidad)

    return salas
```

(emulator.py)

--Ejemplo de simulación--



The image shows two screenshots of a PyCharm terminal window. The top screenshot shows the output of a simulation for 'Sensor 0' with 'SALIR' (Exit) and 'ENTRAR' (Enter) commands. The bottom screenshot shows the output of a simulation for 'Sensor 2' with 'ENTRAR' (Enter) and 'Ejecutando server...' (Running server...) commands. Both screenshots show a list of rooms (Sala 1 to Sala 5) with their respective IDs, capacities, and occupancies.

```
Run: backend
SALIR - Sensor 0
Sala 1
  Id: 0  Capacidad: 4  Ocupacion: 2
Sala 2
  Id: 1  Capacidad: 3  Ocupacion: 2
Sala 3
  Id: 2  Capacidad: 6  Ocupacion: 3
Sala 4
  Id: 3  Capacidad: 5  Ocupacion: 3
Sala 5
  Id: 4  Capacidad: 4  Ocupacion: 0
ENTRAR - Sensor 0
Sala 1
  Id: 0  Capacidad: 4  Ocupacion: 3
Sala 2
  Id: 1  Capacidad: 3  Ocupacion: 2
Sala 3
  Id: 2  Capacidad: 6  Ocupacion: 3
Sala 4
  Id: 3  Capacidad: 5  Ocupacion: 3
Sala 5
  Id: 4  Capacidad: 4  Ocupacion: 0

Run: backend
"C:\Users\Juan Jo\AppData\Local\Microsoft\WindowsApps\python3.9.exe" "C:/Users/Juan Jo/Desktop/TF6-Final/Ejecutables/Backend/backend.py"
ENTRAR - Sensor 2
Sala 1
  Id: 0  Capacidad: 4  Ocupacion: 3
Ejecutando server...
Sala 2
  Id: 1  Capacidad: 3  Ocupacion: 2
Sala 3
  Id: 2  Capacidad: 6  Ocupacion: 2
Sala 4
  Id: 3  Capacidad: 5  Ocupacion: 3
Sala 5
  Id: 4  Capacidad: 4  Ocupacion: 0
ENTRAR - Sensor 2
Sala 1
  Id: 0  Capacidad: 4  Ocupacion: 3
Sala 2
  Id: 1  Capacidad: 3  Ocupacion: 2
Sala 3
  Id: 2  Capacidad: 6  Ocupacion: 3
Sala 4
  Id: 3  Capacidad: 5  Ocupacion: 3
Sala 5
  Id: 4  Capacidad: 4  Ocupacion: 0
```

4. Desarrollo del backend productivo

La siguiente fase de este proyecto es la realización de un backend productivo, en el cual se desarrollarán todas las funciones o acciones que se quiere que el bot realice cuando este vaya a ser desarrollado.

En cualquier proyecto que se vaya a realizar, el backend es una parte fundamental e indispensable para un correcto funcionamiento del proyecto. El backend realiza todos los procesos que el proyecto deba necesitar para su correcto funcionamiento, se encarga de los accesos a las bases de datos, la seguridad, rapidez de carga o conexión de endpoints entre otras muchas funciones.

Para la creación del backend productivo se ha seguido la dinámica como en el desarrollo del simulador de detector de datos y se ha usado PyChar, ya que gracias a su buen manejo en la fase anterior del proyecto se ha decidido mantenerlo como IDE principal de trabajo. Por otro lado, hemos utilizado la herramienta Postman la cual nos permite realizar pruebas, mediante solicitudes de tipo 'HTTP requests', gracias a esta

herramienta se ha podido probar los diferentes endpoints que hemos decidió implementar para cubrir todas las operaciones que tendrá el backend.

Una vez que el desarrollo del simulador de detector de datos se ha finalizado, el siguiente paso a dar fue la elaboración de las operaciones que el bot puede realizar cuando sea consultado. Se realizó una función para poder obtener la salida de la simulación en formato JSON, que es un formato de texto sencillo para el intercambio de datos.

```
def toJson(self):  
    return "{\"nombre\": \"" + str(self.nombre) + "\", \"id\": \"  
    \" + str(self.id) + \", \"capacidad\": \" + str(self.capacidad)  
    + \", \"ocupacion\": \" + str(self.ocupacion) + \"}"
```

Gracias esta función se puede devolver en formato JSON el resultado concreto de un momento de la simulación, esta salida devolverá el nombre de la sala, con su correspondiente id, y con su correspondiente capacidad y ocupación actual.

Para desarrollar el backend se crea una clase Handler dentro de **backend.py** la cual primeramente se configura para que escuche por el puerto 8000 de la maquina local y devolver el resultado de las operaciones y poder recoger la respuesta en Postman.

class Handler(http.server.SimpleHTTPRequestHandler):

```
httpd = socketserver.TCPServer(('', 8000), Handler)  
httpd.serve_forever()
```

Previamente a esto, el simulador desarrollado en la fase anterior se ejecuta al mismo tiempo que el backend a través de un hilo, para poder tener así una conexión.

```
emulator.execute()  
hilo = threading.Thread(target = emulateexecute)  
hilo.start()
```

En la clase Handler se pueden atender peticiones Get o Post, para ello se ha generado una función **respond** la cual genera la respuesta, esta creara un body de respuesta analizando la petición que se ha requerido. En base a la petición recibida generara una repuesta **200 OK**, con un formato text/JSON, un **500 Internal Server Error** en caso de no encontrar la petición indicada o un **302 Error** en caso de error en la operación.

```
class Handler(http.server.SimpleHTTPRequestHandler):
    def do_GET(self):
        self.respond()

    def do_POST(self):
        self.respond()

    def respond(self):
        print("Route ", self.path)
        operation = Operation(self.path)
        bodyResult = self.createBody(operation)
        if bodyResult is None:
            self.send_response(500)
            self.send_header('Content-type', 'text/json')
            self.end_headers()
        elif bodyResult.status:
            self.send_response(200)
            self.send_header('Content-type', 'text/json')
            self.end_headers()
            self.wfile.write(bytes(bodyResult.description, "UTF-8"))
        else:
            self.send_response(302)
            self.send_header('Content-type', 'text/json')
            self.end_headers()
            self.wfile.write(bytes(bodyResult.description, "UTF-8"))
```

(backend.py)

Para las operaciones que tendrá el backend se ha definido una función **createBody** que tiene un selector de operaciones “operation” el cual tendrá las siguientes operaciones:

- Información general de una sala.
- Información general de todas las salas.
- Actual ocupación de todas las salas.
- Sala con menor ocupación.
- Sala con mayor ocupación.
- Si se puede entrar a una sala en concreto
- Información estática del gimnasio en general (hora de apertura, hora de cierre)
- Sala Favorita
- Sala con porcentaje de ocupación.

```
def createBody(self, operation):  
    ## check error  
    if operation.checkIsError():  
        return None  
    body = None  
    if operation.operation == "/sala":  
        body = emulator.salasManager.infoSalaJson(int(operation.id))  
    if operation.operation == "/infoTotalSalas":  
        body = emulator.salasManager.infoTotalSalaJson()  
    elif operation.operation == "/currentOccupation":  
        body = emulator.salasManager.currentOcupacionJson()  
    elif operation.operation == "/lessOccupation":  
        body = emulator.salasManager.lessOcupacionJson()  
    elif operation.operation == "/maxOccupation":  
        body = emulator.salasManager.maxOcupacionJson()  
    elif operation.operation == "/canEnter":  
        body = emulator.salasManager.canEnter(int(operation.id))  
    elif operation.operation == "/info":  
        body = emulator.salasManager.info()  
    elif operation.operation == "/salaFavorita":  
        body = emulator.salasManager.favoritaSalaJson()  
    elif operation.operation == "/porcentajeOcupacion":  
        body = emulator.salasManager.salaPorcentajeOcupacionJson(int(operation.id))  
    return body
```

(backend.py)

Estas operaciones son programadas en la clase **salasManager**, la cual como se dijo en la fase anterior sirve como gestor de todas las salas del programa. Estas operaciones tienen su propio código según la operación y se devuelven según una estructura **Result()** para así de esta manera poder controlar los posibles errores que se puedan generar bien sean por una error de la propia operación **Error 302**, o por errores de no poder encontrar dicha operación o petición **500 Internal Server Error**. Estos errores se comentarán con más detalle más adelante así también la forma de como se han tratado.

```
def infoSalaJson(self, idSala):
    if 0 <= idSala < ConfigEmu.MAX_SALAS:
        return Result(True, str(self.salas[idSala].toJson()))
    else:
        return Result(False, errores[1])

def infoTotalSalaJson(self):
    return Result(True, self.toJsonSalas(self.salas))

def canEnter(self, idSala):
    if 0 <= idSala < ConfigEmu.MAX_SALAS:
        return Result(True, str("{\"canEnterSala\": " + str(not
            self.salas[idSala].isFull()) + "}"))
    else:
        return Result(False, errores[1])

def currentOcupacionJson(self):
    ocupacion = 0
    for salaAux in self.salas:
        ocupacion = ocupacion + salaAux.ocupacion
    return Result(True, str("{\"currentOcupacion\": " + str(ocupacion) +
    "}"))

def lessOcupacionJson(self):
    ocupacion = sys.maxsize
    sala = None
    for salaAux in self.salas:
        if salaAux.ocupacion < ocupacion:
            ocupacion = salaAux.ocupacion
            sala = salaAux

    if sala is not None:
        return Result(True, str(sala.toJson()))
    else:
        return Result(False, errores[0])
```

(salasManager.py)

```

def maxOcupacionJson(self):
    ocupacion = -sys.maxsize
    sala = None
    for salaAux in self.salas:
        if salaAux.ocupacion > ocupacion:
            ocupacion = salaAux.ocupacion
            sala = salaAux
    if sala is not None:
        return Result(True, str(sala.toJson()))
    else:
        return Result(False, errores[0])

def favoritaSalaJson(self):
    total = -sys.maxsize
    sala = None
    for salaAux in self.salas:
        if salaAux.contadorTotal > total:
            total = salaAux.contadorTotal
            sala = salaAux

    if sala is not None:
        return Result(True, str(sala.toJson()))
    else:
        return Result(False, errores[0])

def salaPorcentajeOcupacionJson(self, porcentaje):
    if 0 <= porcentaje <= 100:
        sala = None
        for salaAux in self.salas:
            if ((salaAux.ocupacion / salaAux.capacidad) * 100) <= porcentaje:
                sala = salaAux
                break

        if sala is not None:
            return Result(True, str(sala.toJson()))
        else:
            return Result(False, errores[0])
    else:
        return Result(False, errores[2])

def info(self):
    return Result(True, str(infoConfig))

```

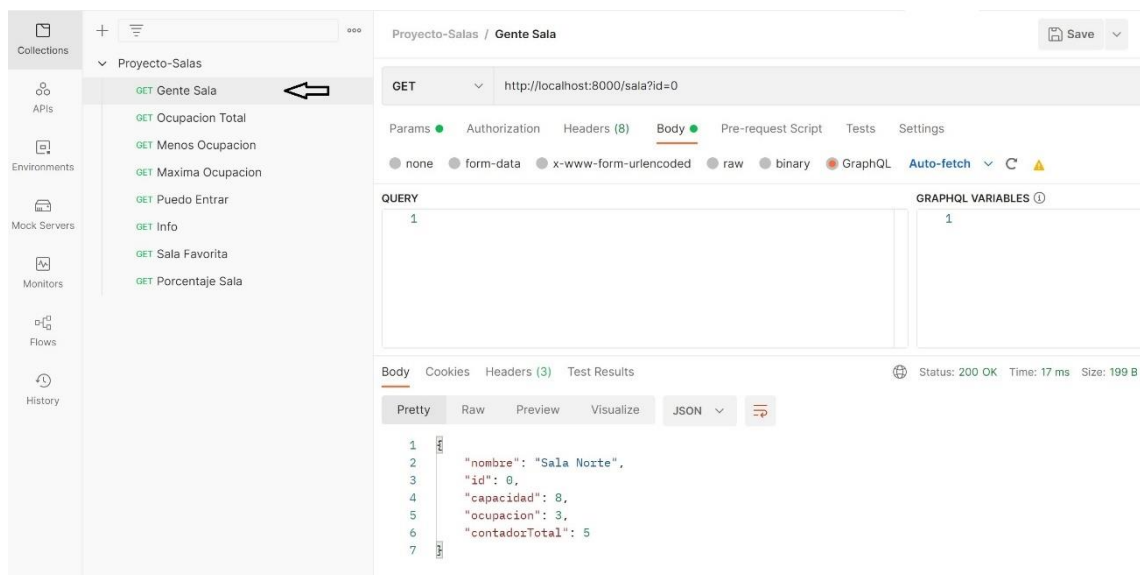
(salasManager.py)

En el siguiente apartado se explicará en detalle el uso de postman en este proyecto, así como la generación de un api, la cual contendrá las solicitudes “http request” de todas las operaciones que se han implementado, este api podrá ser exportada al finalizar el proyecto, para así tener un enlace directo a ella.

4.1 Descripción de API

Para la realización de esta API se ha generado una colección de todos los endpoints que expone el backend, dándole así un toque de simplicidad y sencillas para probarlo.

Con esta colección, se puede acceder de una forma más rápida a las solicitudes “http request” de las operaciones. Por ejemplo, la sección “Gente Sala” contendrá la solicitud en la cual se solicita toda la información general de una sala pasándole en la solicitud el id de la sala que se quiere consultar, es decir esta solicitud devolverá, el nombre de la sala, así como su id, capacidad y ocupación y así con cada una de las secciones del api.



4.1.1 Gente en sala.

Como se comentó anteriormente esta solicitud nos devuelve la información general de la sala. Nombre de la sala, id, capacidad, ocupación actual y un nuevo parámetro llamado contador total, este nuevo parámetro se ha implementado en esta fase del proyecto y se ha programado en el simulador de datos, para que así se pueda tener un contador total de todas las veces que se ha producido una entrada en cierta sala, todo esto con el objetivo de poder darle valor a una solicitud que más adelante veremos llamada sala favorita.

Información general de la sala		
URI	/ localhost:8000/sala	
Método	GET	
Parametros de la petición.	-IdSala (Id de la sala)	Información general de una sala dado un id de sala.
Devuelve	200	OK y text/Json
	302	Error operación
	500	Internal Server error

Petición

```
GET /sala?id=1 HTTP/1.1
Content-Type: application/json
User-Agent: PostmanRuntime/7.29.0
Accept: */*
Postman-Token: 163f931f-9812-4b70-809b-d7e45826c2ef
Host: localhost:8000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 12
```

Respuesta

```
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.9.10
Date: Mon, 21 Mar 2022 18:25:18 GMT
Content-type: text/json
```

```
{
  "nombre": "Sala Oeste",
  "id": 1,
  "capacidad": 7,
  "ocupacion": 6,
  "contadorTotal": 9
}
```

4.1.2 Gente en todas las salas.

Esta petición nos devuelve la información general de todas las salas en el momento que se quiera hacer la consulta.

Información general de la sala		
URI	/ localhost:8000/infoTotalSalas	
Método	GET	
Parametros de la petición.	- Sin parametros en la peticion.	Información general de todas las salas.
Devuelve	200	OK y text/Json
	302	Error operación
	500	Internal Server error

Petición

```
GET /infoTotalSalas HTTP/1.1
User-Agent: PostmanRuntime/7.29.0
Accept: */*
Postman-Token: 5553b03b-6e3d-4dfc-a19b-7afeeb4dc29b
Host: localhost:8000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Respuesta

```
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.9.0
Date: Sat, 26 Mar 2022 23:32:05 GMT
Content-type: text/json
```

```
{
  {
    "nombre": "Sala Norte",
    "id": 0,
    "capacidad": 8,
    "ocupacion": 4,
    "contadorTotal": 6
  },
  {
    "nombre": "Sala Oeste",
    "id": 1,
    "capacidad": 7,
    "ocupacion": 1,
    "contadorTotal": 6
  },
  {
    "nombre": "Sala Sur",
    "id": 2,
    "capacidad": 6,
    "ocupacion": 4,
    "contadorTotal": 7
  },
  {
    "nombre": "Sala Sorento",
    "id": 3,
    "capacidad": 7,
    "ocupacion": 2,
    "contadorTotal": 8
  },
  {
    "nombre": "Sala Espacio",
    "id": 4,
    "capacidad": 6,
    "ocupacion": 0,
    "contadorTotal": 3
  },
}
```


4.1.3 Ocupación total.

Esta petición nos devuelve la ocupación total de todas las salas, es decir en otras palabras la ocupación total del gimnasio o del establecimiento.

Ocupación total de todas las salas		
URI	/ localhost:8000/currentOccupation	
Método	GET	
Parametros de la petición.	Sin parametros en la petición.	Ocupacion total de todas las salas que componen el establecimiento
Devuelve	200	OK
	302	Error operación
	500	Internal Server error

Petición

```
GET /currentOccupation HTTP/1.1
User-Agent: PostmanRuntime/7.29.0
Accept: */*
Postman-Token: 811a7eec-7e0a-47fa-92be-126e47b40d61
Host: localhost:8000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Respuesta

```
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.9.10
Date: Mon, 21 Mar 2022 18:29:06 GMT
Content-type: text/json

{"currentOcupacion": 19}
```

4.1.4 Menor ocupación.

Esta solicitud nos devuelve la sala con menor ocupación, en el momento que se realiza la petición.

Sala con menor ocupación		
URI	/ localhost:8000/lessOccupation	
Método	GET	
Parametros de la petición.	Sin parametros en la petición.	Sala que tiene menor ocupación
Devuelve	200	OK y text/Json
	302	Error operación
	500	Internal Server error

Petición

```
GET /lessOccupation HTTP/1.1
User-Agent: PostmanRuntime/7.29.0
Accept: */*
Postman-Token: e5dfff18-12ce-47de-8d67-4f9ebab75f13
Host: localhost:8000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Respuesta

```
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.9.10
Date: Mon, 21 Mar 2022 18:36:10 GMT
Content-type: text/json
```

```
{
  "nombre": "Sala Sorento",
  "id": 3,
  "capacidad": 7,
  "ocupacion": 1,
  "contadorTotal": 11
}
```

4.1.5 Mayor ocupación.

Esta solicitud nos devuelve la sala con mayor ocupación, en el momento que se realiza la petición.

Sala con mayor ocupación		
URI	/ localhost:8000/maxOccupation	
Método	GET	
Parametros de la petición.	Sin parametros en la petición.	Sala que tiene mayor ocupación
Devuelve	200	OK y text/Json
	302	Error operación
	500	Internal Server error

Petición

```
GET /maxOccupation HTTP/1.1
User-Agent: PostmanRuntime/7.29.0
Accept: */*
Postman-Token: 99953485-4df6-4eaf-b26f-a55eadc2b633
Host: localhost:8000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Respuesta

```
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.9.10
Date: Mon, 21 Mar 2022 18:40:07 GMT
Content-type: text/json
```

```
{
  "nombre": "Sala Espacio",
  "id": 4,
  "capacidad": 6,
  "ocupacion": 6,
  "contadorTotal": 14
}
```

4.1.6 Se puede entrar a una sala.

Esta solicitud nos devuelve un true si alguien puede entrar en la sala, o un false en caso contrario, teniendo en cuenta la ocupación actual de la sala. Se le pasa como parámetro el id de la sala que se quiere saber si puede entrar o no.

Entrar a una sala		
URI	/ localhost:8000/canEnter	
Método	GET	
Parametros de la petición.	-IdSala (Id de la sala)	Se puede entrar o no a una sala según su id.
Devuelve	200	OK
	302	Error operación
	500	Internal Server error

Petición

```
GET /canEnter?id=2 HTTP/1.1
User-Agent: PostmanRuntime/7.29.0
Accept: */*
Postman-Token: 290e71bf-9a31-4b4f-abba-69744e0840ac
Host: localhost:8000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Respuesta

```
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.9.10
Date: Mon, 21 Mar 2022 18:44:14 GMT
Content-type: text/json

{"canEnterSala": False}
```

4.1.7 Información general.

Esta solicitud nos devuelve información general estática sobre el gimnasio. Esta información ha sido previamente configurada en un fichero JSON estático, es decir esta información puede ser cambiada a conveniencia del dueño del programa.

Información general del gimnasio		
URI	/ localhost:8000/info	
Método	GET	
Parametros de la petición.	Sin parametros en la peticion.	Informacion estatica sobre el gimnasio.
Devuelve	200	OK y text/Json
	302	Error operación
	500	Internal Server error

Petición

```
GET /info HTTP/1.1
User-Agent: PostmanRuntime/7.29.0
Accept: */*
Postman-Token: a4e3f585-7865-45cc-82ad-40715bedb961
Host: localhost:8000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Respuesta

```
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.9.10
Date: Mon, 21 Mar 2022 19:18:18 GMT
Content-type: text/json
```

```
{
  "hora-apertura": "8:00",
  "hora-cierre": "20:00",
  "nombre": "Go-fit",
  "direccion": "Calle Gran via",
  "mensualidad": 25,
  "n-salas": 5,
  "ocupacionTotal": 100
}
```

4.1.8 Sala favorita.

Esta solicitud nos devuelve la sala que hasta ese momento tiene el mayor numero de entradas desde que se inicio el programa, esto se debe a que tiene el numero mas alto en el parámetro contador Total. Consecuencia de esto la sala con mayor número de entradas, será la sala favorita.

Sala favorita del gimnasio		
URI	/ localhost:8000/salaFavorita	
Método	GET	
Parametros de la petición.	Sin parametros en la peticion.	Sala en donde se han producido más entradas
Devuelve	200	OK y text/Json
	302	Error operación
	500	Internal Server error

Petición

```
GET /salaFavorita HTTP/1.1
User-Agent: PostmanRuntime/7.29.0
Accept: */*
Postman-Token: a4b7a575-eadb-4609-af9d-04387ed8da5d
Host: localhost:8000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Respuesta

```
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.9.10
Date: Mon, 21 Mar 2022 19:22:50 GMT
Content-type: text/json
```

```
{
  "nombre": "Sala Norte",
  "id": 0,
  "capacidad": 8,
  "ocupacion": 5,
  "contadorTotal": 42
}
```

4.1.9 Sala porcentaje.

Esta solicitud devuelve la sala que tenga un menor porcentaje de ocupación que el indicado en la solicitud. Siendo así esta una de las operaciones más importantes del proyecto, ya que este porcentaje puede ser modificado cuando se quiera y así poner restricciones de aforo según el porcentaje.

Sala con porcentaje de ocupación		
URI	/ localhost:8000/porcentajeOcupacion	
Método	GET	
Parametros de la petición.	-Porcentaje de ocupación	Sala en la que su ocupación actual es menor que la del porcentaje dado.
Devuelve	200	OK y text/Json
	302	Error operación
	500	Internal Server error

Petición

```
GET /porcentajeOcupacion?id=50 HTTP/1.1
User-Agent: PostmanRuntime/7.29.0
Accept: */*
Postman-Token: f293ffe3-283d-4b1f-9d90-34f41fb0a3b1
Host: localhost:8000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Respuesta

```
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.9.10
Date: Mon, 21 Mar 2022 19:30:30 GMT
Content-type: text/json
```

```
{
  "nombre": "Sala Norte",
  "id": 0,
  "capacidad": 8,
  "ocupacion": 2,
  "contadorTotal": 45
}
```

4.2 Errores tratados y controlados.

En este apartado se comentará la implementación y metodología que se ha utilizado para tratar los posibles errores que puedan tener las solicitudes “http requests” de las operaciones.

Como ya se explicó anteriormente cada solicitud generara una respuesta **200 OK** cuando la solicitud tiene una respuesta correcta, un **302 Error** cuando una operación ha producido un error generado por la propia operación, es decir por pasar un id fuera de rango de los id's que puedan tener las salas o por dar un porcentaje fuera de rango entre otros muchos errores que se pueden dar. También se puede producir un **500 Internal Server Error**, en este caso se da cuando no se ha solicitado de manera correcta la operación, es decir se ha escrito mal la uri de la petición.

Se han gestionado y controlado estos tipos de errores de entrada o de salida, con el fin de poder gestionar de mejor manera las entradas que se produzcan en Telegram para el desarrollo del bot, que será la siguiente fase del proyecto.

Para ello se ha creado una tabla con diferentes códigos de error con su correspondiente descripción, y luego se han implementado en las operaciones en las cuales se podrían producir. Para ello se ha realizado una estructura llamada **Result()** la cual devuelve un **true** y devolvería el resultado de la operación satisfactoriamente, o devuelve un **false** con su correspondiente error, ya sea un 302 por error en la operación o un 500 por error al no poder acceder a dicha petición.

4.2.1 Tabla de gestión de errores.

Los errores que se han gestionado quedan reflejados en la siguiente tabla con su correspondiente significado.

ERROR-CODE	SIGNIFICADO
ErrorCode Str(0)	Sala no encontrada.
ErrorCode Str(1)	Identificador de sala no valido.
ErrorCode Str(2)	Parámetro enviado no reconocido.

4.2.2 Ejemplos de errores.

4.2.2.1 Error Identificador de sala no valido

Este error se produce cuando se pide una sala con un identificador fuera de rango, es decir con un identificador que no existe.

The screenshot shows a Postman interface for a GET request to `http://localhost:8000/sala?id=7`. The 'Params' tab is active, showing a parameter 'id' with value '7'. The 'Body' tab is also active, showing a JSON response in 'Pretty' view:

```
1 {
2   "errorCode": 1,
3   "description": "Identificador de sala no válido "
4 }
```

At the bottom, the status bar indicates 'Status: 302 Found', 'Time: 36 ms', and 'Size: 188 B'.

Petición

```
GET /sala?id=7 HTTP/1.1
Content-Type: application/json
User-Agent: PostmanRuntime/7.29.0
Accept: */*
Postman-Token: c8afba4b-a22b-4132-a6b0-00d5bf6b83c8
Host: localhost:8000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 12
```

Respuesta

```
HTTP/1.0 302 Found
Server: SimpleHTTP/0.6 Python/3.9.0
Date: Sun, 27 Mar 2022 00:02:06 GMT
Content-type: text/json

{"errorCode":1, "description": "Identificador de sala no
válido "}
```

4.2.2.2 Error Sala no encontrada

Este error se produce cuando dado un porcentaje, no hay ninguna sala en todo el gimnasio que tenga un porcentaje de ocupación menor que el pasado como parámetro.

GET http://localhost:8000/porcentajeOcupacion?id=0

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
-----	-------	-------------

Body Cookies Headers (3) Test Results

Status: 302 Found Time: 15 ms Size: 173 B

Pretty Raw Preview Visualize JSON

```
1  
2  "errorCode": 0,  
3  "description": "Sala no encontrada"  
4
```

Petición

```
GET /porcentajeOcupacion?id=0 HTTP/1.1  
User-Agent: PostmanRuntime/7.29.0  
Accept: */*  
Postman-Token: ce96be4a-99cb-4b3c-8311-05b0345e194f  
Host: localhost:8000  
Accept-Encoding: gzip, deflate, br  
Connection: keep-alive
```

Respuesta

```
HTTP/1.0 302 Found  
Server: SimpleHTTP/0.6 Python/3.9.0  
Date: Sun, 27 Mar 2022 00:09:07 GMT  
Content-type: text/json  
  
{"errorCode":0, "description": "Sala no encontrada"}
```

4.2.2.3 Error parámetro enviado no reconocido.

Este error se produce cuando se pasa como parámetro un porcentaje fuera de rango es decir un porcentaje no valido como puede ser mayor de 100 o menor de 0.

GET ▼ http://localhost:8000/porcentajeOcupacion?id=-1

Params ● Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
-----	-------	-------------

Body Cookies Headers (3) Test Results 🌐 Status: 302 Found Time: 13 ms Size: 187 B

Pretty Raw Preview Visualize JSON ≡

```
1 {
2   "errorCode": 2,
3   "description": "Parámetro enviado no reconocido"
4 }
```

Petición

```
GET /porcentajeOcupacion?id=-1 HTTP/1.1
User-Agent: PostmanRuntime/7.29.0
Accept: */*
Postman-Token: ab6e3905-4a70-471a-9a3b-45d7ec7bf614
Host: localhost:8000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Respuesta

```
HTTP/1.0 302 Found
Server: SimpleHTTP/0.6 Python/3.9.0
Date: Sun, 27 Mar 2022 00:16:58 GMT
Content-type: text/json

{"errorCode":2, "description": "Parámetro enviado no reconocido"}
```

5. Desarrollo del bot y despliegue en Telegram.

Siguientes pasos en el proyecto:

- Investigación y documentación de la plataforma Telegram para desplegar un bot
- Investigación, desarrollo y posterior documentación del bot.
- Pruebas y despliegue del bot

