Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Bot Para La Gestión De Aforo De Espacios

Autor: Juan José Berrio Arredondo

Tutor(a): Tomas San Feliu Gilabert

Madrid, Junio 2022

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática
Título: Bot Para La Gestión De Aforo De Espacios
Junio 2022

Autor: Juan Jose Berrio Arredondo

Tutor:
Tomas San Feliu Gilabert
Lenguajes y Sistemas Informáticos e Ingeniería de Software.
ETSI Informáticos
Universidad Politécnica de Madrid

A mi familia
y a toda la gente
que me ha apoyado
en este tiempo.
Gracias a todos.

Agradecimientos.

Ante todo, quiero agradecerle a Dios, por la oportunidad de haber tenido una familia maravillosa, amigos, y de haber podido terminar esta etapa de mi vida que hoy culmina con la realización de este trabajo de fin de grado.

A mi madre por ser la persona más maravillosa del mundo y que cada día me inspira a ser mejor persona y no rendirme ante cualquier situación.

A toda mi familia, por ese apoyo infinito e incondicional que me han dado en los momentos en que más lo he necesitado. Muchas Gracias

A ti mi babu por haber estado siempre en buenas y en malas, me has enseñado y demostrado lo que verdaderamente es la palabra amor.

Y a ti mi angel C.F.P que desde que llegaste a mi vida, lo cambiaste todo, y hoy soy un Ingeniero gracias a ti. Lo hemos logrado, ahora, hoy y donde quiera que siempre estes, estaré eternamente agradecido por todo.

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

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.[1] 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 apenas existen hoy en día sistemas de recuento que nos ayuden a resolverlo de forma sencilla. Estos sistemas nos ayudarían para el Sars-Cov-2 y para otros virus o pandemias que pueda existir en el mundo, así como 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 pueda 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 inunden 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 la aplicación de despliegue.

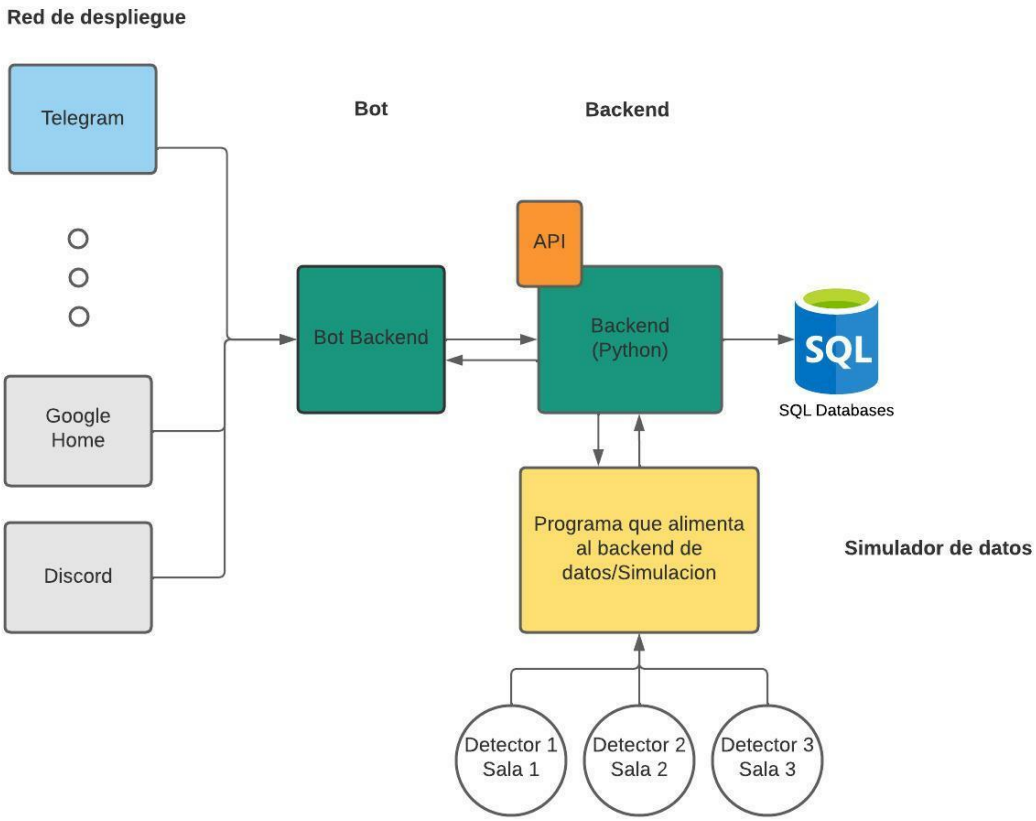


Ilustración 1 *Diseño del proyecto*

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 cambio 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.[2]

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.

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. [3]

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 ultimas noticias?" ofreciendo así respuestas muchos 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 compresión de lenguaje natural y la generación de lenguaje natural.[4]

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.[5]

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.[6]

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.

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 él envió 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 más 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. [7]

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.[8]

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.

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 tramites digitales a personas con discapacidades visuales.[9]

Hoy en día tecnologías 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 abarcar estas tecnologías. Es por esto por lo 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 PyCharm, ya que después de una investigación entre la elección de IDE's PyCharm 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 integrase.
- **Bot** -> Es un Bot escrito en Python y desplegado en plataforma e integrado en un cliente de mensajería que es Telegram.

3 Desarrollo simulador detector de datos

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

3.1 ¿Por qué se necesita un simulador?

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 serían 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 forman 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.

3.2 Desarrollo e implementación del simulador

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

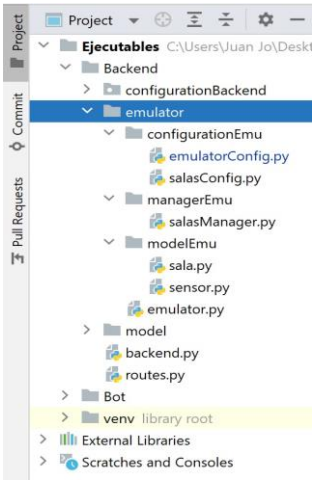


Ilustración 2 Estructura simulador

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.

```
class ConfigEmu:
    RELOJ = 10
    ENTER = 0
    EXIT = 1
    MAX_SALAS = 5
    MAX_SENSOR = 5
```

Ilustración 3 emulator configuración

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

```
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]),
]
```

Ilustración 4 salas configuración

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.

Por otro lado, 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 definirá en esta clase las diferentes operaciones disponibles para la gestión de las salas.

Al final 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)
```

Ilustración 5 job-emulator

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 def **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 en dicho sensor.

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

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

Ilustración 6 elección de sensor y elección de acción

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.

```
28      def executeaccion(self, sensorNumber, accion):
29          sensores[sensorNumber].executeaccion(accion)
30          self.salasManager.printStatus()
31
32      def candoaccion(self, sensorNumber, accion):
33          sensor = sensores[sensorNumber]
34          if sensor.canDoAccion(accion):
35              self.printstatusaccion(accion, sensorNumber)
36              return True
37          else:
38              self.printerroraccion(accion, sensor)
39              return False
40
41      def printstatusaccion(self, accion, sensor):
42          if (accion == ConfigEmu.ENTER):
43              print("ENTRAR - Sensor " + str(sensor))
44          else:
45              print("SALIR - Sensor " + str(sensor))
46
47      def printerroraccion(self, accion, sensor):
48          print("ERROR - Accion " + str(accion) + " - Sensor " + str(sensor.id))
```

Ilustración 7 funciones del emulator

Por otro lado, se ha definido una función warmup, la cual realiza un precalentamiento antes de que el simulador empiece a realizar salidas y entradas según el pulso de reloj. Todo esto con la finalidad de que cuando se ejecute el simulador, ya pueda haber en algunas salas un numero de ocupación y no tenga que empezar todas las salas con el contador a 0.

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

    return salas
```

Ilustración 8 warmup-emulator

3.3 Ejemplo y puesta en producción

En esta simulación se puede ver que se quiere ejecutar en el sensor con id = 4 una entrada y se realiza correctamente. El siguiente evento o job que se genera de forma aleatorio o random es entrar en el sensor con id = 0, se ejecuta correctamente y como se puede ver en comparación a la ejecución anterior la ocupación pasa de ser 3 a ser 4. Otro evento más que ocurre es el de salir en sensor con id = 3, se produce sin ningún problema y la ocupación pasa de ser 5 a ser 4.

```
ENTRAR - Sensor 4
 Sala Norte
      Id: 0     Capacidad: 8     Ocupacion: 3     Contador Total: 13
 Sala Oeste
      Id: 1     Capacidad: 7     Ocupacion: 1     Contador Total: 13
 Sala Sur
      Id: 2     Capacidad: 6     Ocupacion: 2     Contador Total: 14
 Sala Sorento
      Id: 3     Capacidad: 7     Ocupacion: 5     Contador Total: 16
 Sala Espacio
      Id: 4     Capacidad: 6     Ocupacion: 4     Contador Total: 16
ENTRAR - Sensor 0
 Sala Norte
      Id: 0     Capacidad: 8     Ocupacion: 4     Contador Total: 14
 Sala Oeste
      Id: 1     Capacidad: 7     Ocupacion: 1     Contador Total: 13
 Sala Sur
      Id: 2     Capacidad: 6     Ocupacion: 2     Contador Total: 14
 Sala Sorento
      Id: 3     Capacidad: 7     Ocupacion: 5     Contador Total: 16
 Sala Espacio
      Id: 4     Capacidad: 6     Ocupacion: 4     Contador Total: 16
SALIR - Sensor 3
 Sala Norte
      Id: 0     Capacidad: 8     Ocupacion: 4     Contador Total: 14
 Sala Oeste
      Id: 1     Capacidad: 7     Ocupacion: 1     Contador Total: 13
 Sala Sur
      Id: 2     Capacidad: 6     Ocupacion: 2     Contador Total: 14
 Sala Sorento
      Id: 3     Capacidad: 7     Ocupacion: 4     Contador Total: 16
 Sala Espacio
      Id: 4     Capacidad: 6     Ocupacion: 4     Contador Total: 16
```

Ilustración 9 salida-emulator

4 Servidor/Backend

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.

4.1 ¿Por qué se necesita un backend o servidor?

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 PyCharm, 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, se ha utilizado la herramienta Postman la cual permite realizar pruebas, mediante solicitudes de tipo 'HTTP requests', gracias a esta herramienta se han probado los diferentes endpoints que se decidieron implementar para cubrir todas las operaciones que tendrá el backend.

4.2 Definición API y endpoints del backend

Para la realización de esta fase del proyecto se decidió realizar una API donde se definan todos los endpoints necesarios para el sistema.

Una API o también llamada interfaz de programación de aplicaciones es un conjunto de protocolos que se usa para poder diseñar como integrar software en una aplicación.[10] Gracias a la API el backend podrá obtener la ruta de los endpoints de las operaciones que se van a realizar, asi como una conexión con estas operaciones que tienen valor gracias a los datos simulados por el emulator.

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 sencillez para probarlo.

A continuación, vamos a mostrar todos los métodos expuestos por la API así como su definición y su respuesta.

4.2.1 Gente sala

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 operación 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

Tabla 1 Gente en la sala

Petición

```
GET /sala?id=1 HTTP/1.1
Content-Type: application/json
User-Agent: PostmanRuntime/7.29.0
Accept: */*
Postman-Token: 6b8176e5-19c6-476a-a9b9-5c7ade7f080c
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/3.10.4
Date: Sat, 14 May 2022 21:22:46 GMT
Content-type: text/json

{
    "nombre": "Sala Oeste",
    "id": 1,
    "capacidad": 7,
    "ocupacion": 5,
    "contador Total": 8
}
```

Ilustración 10 petición/respuesta-Gente en sala

4.2.2 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 todas las salas		
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

Tabla 2 Información general todas las salas

Petición

```
GET /sala?id=1 HTTP/1.1
Content-Type: application/json
User-Agent: PostmanRuntime/7.29.0
Accept: */*
Postman-Token: 6b8176e5-19c6-476a-a9b9-5c7ade7f080c
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/3.10.4
Date: Sat, 14 May 2022 22:06:08 GMT
Content-type: text/json
{
        "nombre": "Sala Norte",
        "id": 0,
        "capacidad": 8,
        "ocupacion": 2,
        "contadorTotal": 24
    },
    {
        "nombre": "Sala Oeste",
        "id": 1,
        "capacidad": 7,
        "ocupacion": 2,
        "contadorTotal": 28
    },
    {
        "nombre": "Sala Sur",
        "id": 2,
        "capacidad": 6,
        "ocupacion": 1,
        "contadorTotal": 33
    },
    {
        "nombre": "Sala Sorento",
        "id": 3,
        "capacidad": 7,
        "ocupacion": 0,
        "contadorTotal": 28
    },
    {
        "nombre": "Sala Espacio",
        "id": 4,
        "capacidad": 6,
        "ocupacion": 3,
        "contadorTotal": 36
    }
```

Ilustración 11 petición/respuesta gente todas las salas

4.2.3 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/currentOcupation	
Método	GET	
Parametros de la petición.	Sin parametros en la peticion.	Ocupacion total de todas las salas que componen el establecimiento
Devuelve	200	OK
	302	Error operación
	500	Internal Server error

Tabla 3 ocupación total

Petición

```
GET /currentOcupation HTTP/1.1
User-Agent: PostmanRuntime/7.29.0
Accept: */*
Postman-Token: e2bce57a-19ea-4b6f-ab53-98b429579019
Host: localhost:8000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Respuesta

```
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.10.4
Date: Sat, 14 May 2022 22:17:42 GMT
Content-type: text/json

{"currentOcupacion": 19}
```

Ilustración 12 petición/respuesta ocupación total

4.2.4 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/lessOcupation	
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

Tabla 4 sala con menor ocupación

Petición

```
GET /lessOcupation HTTP/1.1
User-Agent: PostmanRuntime/7.29.0
Accept: */*
Postman-Token: d1add2f8-5967-47d3-bee0-5467c870c207
Host: localhost:8000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Respuesta

```
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.10.4
Date: Sat, 14 May 2022 22:36:31 GMT
Content-type: text/json
 {

    "nombre": "Sala Norte",
    "id": 0,
    "capacidad": 8,
    "ocupacion": 0,
    "contadorTotal": 35
}
```

Ilustración 13 petición/respuesta sala con menor ocupación

4.2.5 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/maxOcupation	
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

Tabla 5 sala con mayor ocupación

Petición

```
GET /maxOcupation HTTP/1.1
User-Agent: PostmanRuntime/7.29.0
Accept: */*
Postman-Token: ebe87560-aae8-4c88-bad0-897aa61dfe70
Host: localhost:8000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Respuesta

```
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.10.4
Date: Sat, 14 May 2022 22:40:15 GMT
Content-type: text/json

{
    "nombre": "Sala Espacio",
    "id": 4,
    "capacidad": 6,
    "ocupacion": 5,
    "contadorTotal": 56
}
```

Ilustración 14 petición/respuesta sala con mayor ocupación

4.2.6 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

Tabla 6 Se puede entrar a una sala

Petición

```
GET /canEnter?id=4 HTTP/1.1
User-Agent: PostmanRuntime/7.29.0
Accept: */*
Postman-Token: 4c1b0c48-4665-40a8-a44c-66a0e71ed71d
Host: localhost:8000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Respuesta

```
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.10.4
Date: Sat, 14 May 2022 22:50:23 GMT
Content-type: text/json

{"canEnterSala": False}
```

Ilustración 15 petición/respuesta se puede entrar a una sala

4.2.7 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

Tabla 7 Información general

Petición

```
GET /info HTTP/1.1
User-Agent: PostmanRuntime/7.29.0
Accept: */*
Postman-Token: 4adf1e8c-f04a-403a-a259-596333ef0e5f
Host: localhost:8000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Respuesta

```
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.10.4
Date: Sat, 14 May 2022 22:55:10 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
}
```

Ilustración 16 petición/respuesta información general

4.2.8 Sala favorita

Esta solicitud nos devuelve la sala que hasta ese momento tiene el mayor número de entradas desde que se inició el programa, esto se debe a que tiene el número más 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

Tabla 8 sala favorita

Petición

```
GET /salaFavorita HTTP/1.1
User-Agent: PostmanRuntime/7.29.0
Accept: */*
Postman-Token: 01567d20-184f-45af-b19c-46b2ac78a500
Host: localhost:8000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Respuesta

```
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.10.4
Date: Sat, 14 May 2022 23:00:17 GMT
Content-type: text/json

{
    "nombre": "Sala Espacio",
    "id": 4,
    "capacidad": 6,
    "ocupacion": 4,
    "contadorTotal": 68
}
```

Ilustración 17 petición/respuesta sala favorita

4.2.9 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 50 %.
Devuelve	200	OK y text/Json
	302	Error operación
	500	Internal Server error

Tabla 9 Sala porcentaje

Petición

```
GET /porcentajeOcupacion?id=50 HTTP/1.1
User-Agent: PostmanRuntime/7.29.0
Accept: */*
Postman-Token: 24f8134c-e14c-42d4-8ff5-b915acb75fa9
Host: localhost:8000
Accept-Encoding: gzip, deflate, br
Connection: keep-aliv
```

Respuesta

```
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.10.4
Date: Sat, 14 May 2022 23:10:18 GMT
Content-type: text/json

{
    "nombre": "Sala Norte",
    "id": 0,
    "capacidad": 8,
    "ocupacion": 2,
    "contadorTotal": 50
}
```

Ilustración 18 petición/respuesta sala porcentaje

4.3 Desarrollo e implementación del backend/API

Para el desarrollo del backend se ha realizado primeramente en la clase de sala, 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. Este resultado en formato JSON devolverá de la sala: El nombre, id, capacidad, ocupación y contador total.

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

Ilustración 19 salida formato json

Además, en el **backend.py** se crea una clase Handler la cual primeramente se configura para que escuche por el puerto 8000 de la maquina local y devolver el resultado de las operaciones.

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

Ilustración 20 configuración de backend

Con la clase Handler se pueden atender peticiones de tipo Get/Post/Put/Detele, en este caso solamente hemos generado peticiones GET. Para ello se ha generado una función **respond** para devolver la respuesta, esta creara un body analizando la petición del cliente. 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.

```
def do_GET(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"))
```

Ilustración 21handler-backend

Una vez dada esta estructura al backend, la definición de las operaciones que tendrá la API son las siguientes:

- 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 porcentaje

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 cómo se han tratado.

Por otro lado, para que el backend supiera que operaciones iban a necesitar de un id de sala, es decir que iban a necesitar de tener un parámetro “sala”, para que así la operación tuviera una respuesta correcta, se implementó un método “checkIsError” el cual antes de formar todos los endpoints o rutas de las operaciones en la api, comprueba que las operaciones “CanEnter”, “porcentaje ocupación”, “sala (Gente en Sala)” tengan un id en su ruta o endpoint en la API.

```
def checkIsError(self):
    if (((self.operation == "/sala") or
         (self.operation == "/canEnter") or
         (self.operation == "/porcentajeOcupacion")) and
            self.id is None):
        return True
```

Ilustración 22 función checkIsError (id)

Para finalizar este desarrollo del backend se implementó, una función la cual ejecuta el simulador de datos, al mismo tiempo que se ejecuta el backend, todo esto con la finalidad de tener un único proceso lanzado y no 2 procesos diferentes, así de esta forma comparten el mismo espacio de memoria y hay una conexión directa entre estos componentes del proyecto. Para hacer esto se hizo mediante un hilo Thread.

```
def emulatorexecute():
    emulator.execute()

hilo = threading.Thread(target=emulatorexecute)
hilo.start()
```

Ilustración 23 ejecutar a la vez simulador y backend

4.4 Testing y errores tratados

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.

4.4.1 Tabla de gestión de errores

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

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

Tabla 10 Tabla de errores

4.4.2 Ejemplos de errores

4.4.2.1 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.

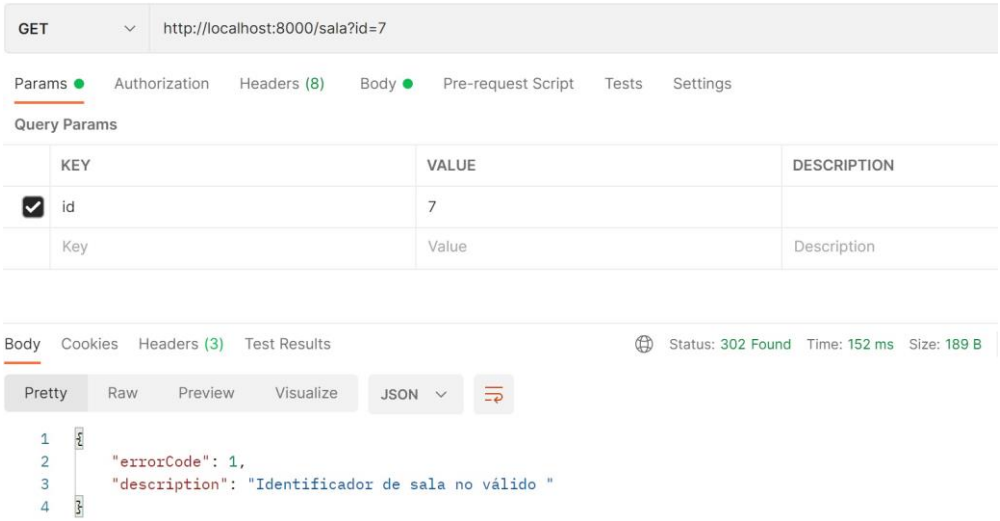


Ilustración 24 Error identificador de sala no valido

Petición

```
GET /sala?id=7 HTTP/1.1
Content-Type: application/json
User-Agent: PostmanRuntime/7.29.0
Accept: */*
Postman-Token: 551a8a7d-cd67-438e-bb7e-69e56bd032e3
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/3.10.4
Date: Sun, 15 May 2022 00:35:34 GMT
Content-type: text/json

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

Ilustración 25 petición/respuesta Error (1)

4.4.2.2 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.

Ilustración 26 error sala no encontrada

Petición

```
GET /porcentajeOcupacion?id=0 HTTP/1.1
User-Agent: PostmanRuntime/7.29.0
Accept: */*
Postman-Token: da853253-c35d-40d5-96fd-0796f2bd2b92
Host: localhost:8000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Respuesta

```
HTTP/1.0 302 Found
Server: SimpleHTTP/0.6 Python/3.10.4
Date: Sun, 15 May 2022 00:50:43 GMT
Content-type: text/json

{"errorCode":0, "description": "Sala no encontrada"}
```

Ilustración 27 petición/respuesta Error (0)

4.4.2.3 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.

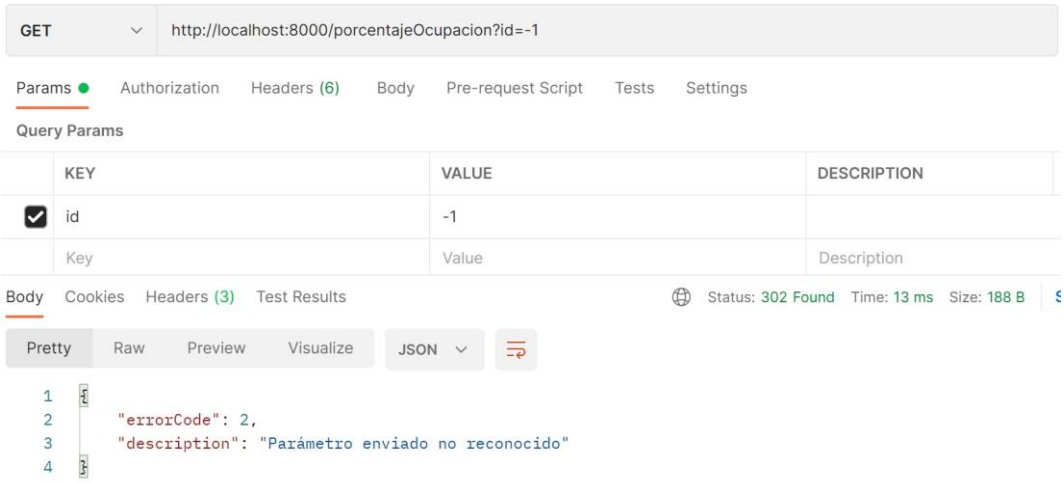


Ilustración 28 error parámetro enviado no reconocido

Petición

```
GET /porcentajeOcupacion?id=-1 HTTP/1.1
User-Agent: PostmanRuntime/7.29.0
Accept: */*
Postman-Token: 586de382-1ea6-4b79-b6c0-f6732452eb00
Host: localhost:8000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Respuesta

```
HTTP/1.0 302 Found
Server: SimpleHTTP/0.6 Python/3.10.4
Date: Sun, 15 May 2022 00:47:02 GMT
Content-type: text/json

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

Ilustración 29 petición/respuesta error (2)

5 Desarrollo del bot y despliegue en Telegram

En este apartado se comentará el desarrollo del bot y su despliegue en la aplicación de mensajería Telegram que ha sido la que se ha elegido para desplegarlo, también se comentara el estudio previo que ha conllevado hacer esta última parte del proyecto.

5.1 Posibles entornos de despliegue del backend y del simulador

Una vez finalizada la fase del desarrollo del simulador de datos (capitulo 3), así como la fase del desarrollo del backend productivo (capitulo 4), el siguiente paso fue pensar en un entorno para poder desplegar estos servicios, que luego se conectaran con el bot.

En los siguientes subapartados se comentarán los posibles candidatos que se analizaron y se estudiaron, así como la decisión final para el despliegue del backend y del simulador.

5.1.1 PythonAnyWhere

¿Qué es?

En primer lugar, se pensó en PythonAnyWhere que es un entorno de despliegue y alojamiento en línea para cualquier tipo de servicio que este hecho en Python. Tiene una suscripción gratuita, pero si se desea tener más potencia o más tiempo de disponibilidad conlleva un coste económico mantenerlo.[11]

¿Utilización?

Para su utilización, se debe de crear una cuenta en PythonAnyWhere.com, el código de nuestro servicio debe ser subido a un repositorio Git, para que luego este puede ser llamado desde la interfaz de PythonAnyWhere, para ello previamente se crea un entorno virtual con su respectiva configuración, es decir con la versión de Python que nuestro servicio ha sido programado. Dentro de este entorno se descomprime el archivo git en donde esta subido nuestro servicio que queremos desplegar. Una vez realizado esto se puede ir a la interfaz de la aplicación PythonAnyWhere y en el botón Reload se puede desplegar el servicio. Es recomendable hacer una serie de configuraciones extras para que el servicio pueda estar siempre actualizado.

5.1.2 Google Cloud/ AWS/Azure

¿Qué es?

El despliegue en la nube bien sea en AWS, Google Cloud o Azure, fue una opción que se pensó en gran medida para desplegar el backend productivo como el simulador de datos. Cualquiera de las tres nubes nos ofrece productos de alta calidad y de gran variedad, así como también nos garantiza que nuestro servicio tendrá una alta calidad con respecto a su rendimiento, así como garantizar su seguridad. Al ser sin duda los 3 principales proveedores de servicios en la nube, el despliegue de un servicio propio en cualquiera de las nubes conlleva de un coste según el tipo de servicio, así como la disponibilidad que queremos que tenga nuestro servicio. Las 3 nubes ofrecen ciertas ofertas o cuentas demo o test, para poder probar sus servicios, pero no suelen tener una gran potencia o

disponibilidad, a menos que se pague un precio según el tipo de servicio que se quiera desplegar en la nube.[12]

¿Utilización?

Para el despliegue de un servicio en cualquiera de las 3 nubes, primeramente, se debe de crear una cuenta y luego configurar la zona en la que queremos levantar nuestro servicio, así como el tipo de servicio que deseamos desplegar, y su disponibilidad de operabilidad. Cada nube tiene una configuración diferente, y unos tipos de pago diferente que en este trabajo no se comentaran, ya que se sale del alcance del proyecto.

5.1.3 NGROK

¿Qué es?

Ngrok es una herramienta muy conocida hoy en día por su bajo coste, así como su fácil manejo, y nos permite desplegar nuestro servicio desde nuestro propio ordenador de forma local, sin necesidad de estar usando alguna plataforma externa como las mencionadas en los apartados anteriores. Todo es posible debido a que esta herramienta se basa en hacer público nuestro localhost, y de esta manera mientras nuestro servicio este levantado, pueda ser consultado desde Internet.[13]

¿Utilización?

Para la utilización de la herramienta Ngrok, se debe primeramente dirigir a la página oficial del proveedor Ngrok y descargar un ejecutable, luego de ello se podrá ejecutar este archivo desde la terminal de nuestra máquina.

5.1.4 Decisión final

Se ha utilizado la herramienta **NGROK** debido a su fácil utilización y su coste gratuito, ya que es solo lanzar un ejecutable de tipo EXEC. Ngrok ha sido muy importante en este proyecto ya que es capaz de generar un VLAN hacia la maquina local, es decir la herramienta genera dos direcciones de conexión una “HTTP” y otra “HTTPS”, por las cuales se pueden tener conexión al localhost de la maquina local en donde esta desplegado el servidor, para que de esta forma Telegram sea capaz de conectarse al programa que se ha diseñado, en este caso a las operaciones que realizara el bot, siendo así posible su despliegue.

5.2 Posibles entornos de despliegue del bot

Para la parte del proyecto del despliegue del bot se han estudiado y analizado diferentes entornos que se comentaran a continuación, así como la decisión final que se ha tomado y su explicación de porque ha sido así.

5.2.1 Google Assistant / DialogFlow

¿Qué es?

En primer lugar, se investigó sobre el asistente de Google, el cual mediante una plataforma llamada Dialogflow nos permite crear chatbots o bots conversacionales.

Dialogflow destaca hoy en día gracias a que puede abarcar en varias interfaces de conversación, como puede ser Google Home, teléfonos, smart watch.

¿Cómo funciona?

El funcionamiento de esta plataforma se centra en un flujo básico de conversación entre usuario y bot. En primer lugar, el usuario proporciona un input, este puede ser un pregunta por escrito o por voz. En el segundo paso es donde aparece el concepto "agente" de Dialogflow, que es un módulo capaz de comprender lenguaje natural y es el encargado de extraer cada parámetro del input, una vez extraídos el agente analizará cada parámetro y responderá con una respuesta programada.[14]

5.2.2 Discord

¿Qué es?

Discord es un servicio de mensajería instantánea que proporciona servidores privados, mediante los cuales se puede comunicar con otros usuarios, bien sea escribiendo en chat o mediante voz. Es una plataforma muy utilizada por los gamers, ya que proporciona una baja latencia y esto ayuda a no tener problemas de caída en el juego que se está jugando mientras se puede tener una conversación de alta calidad.

¿Cómo funciona?

Discord proporciona bots, para poder realizar pequeñas tareas de forma automática, hay una gran variedad de bots según su funcionalidad. Bots que dan la bienvenida cuando entras a un servidor o canal nuevo, bots que funcionan como recordatorios de tareas, e incluso hay bots que integran música o iconos gif. Discord proporciona una función en la cual se pueden añadir bots propios o personales, dándole a esto una gran utilidad a la plataforma, pero eso si el programador debe saber su correcta configuración, como es que el bot debe estar previamente subido a una página de terceros y de allí ser descargado. Todo esto con la finalidad que el bot cumpla con su objetivo.[15]

5.2.3 WhatsApp

¿Qué es?

WhatsApp es la plataforma social con más usuarios en España, y de las primeras en todo el mundo. Es una aplicación de mensajería instantánea en la cual los usuarios pueden tener conversaciones con los contactos que estén registrados en su teléfono móvil.

¿Cómo funciona?

Existe una API de llamada WhatsApp Business, la cual está integrada para cuentas que son verificadas previamente como cuentas empresariales. Este API proporciona una serie de funciones que ayudan a las empresas o pequeños negocios a llevar mejor su actividad en las redes sociales. Son capaces de dejar respuestas automáticas previamente diseñadas.[16]

5.2.4 Telegram

¿Qué es?

Telegram es una plataforma de mensajería que está enfocada en la velocidad y seguridad, con respecto a WhatsApp. Telegram proporciona mayor privacidad a sus usuarios y ciertas funcionalidades como tener gran cantidad de bots sin necesidad de tener una cuenta business.

¿Cómo funciona?

Telegram tiene un propio bot ya integrado llamado "BotFather" que ayuda a cualquier usuario de Telegram a crear su propio bot de una manera rápida y sencilla. El BotFather es capaz de controlar todos los bots y para crear un propio bot bastara con abrir una conversación propia con BotFather y pulsar **/Start** y este nos dará una lista de comandos con los cuales podemos crear y ponerle nombre a nuestro propio bot, aparte BotFather nos proporciona un Token de acceso HTTP, por el cual se puede tener conexión con él y así poder enlazarlo con otros servicios que se deseen.

5.2.5 Decisión final.

Analizando todos los posibles entornos, se ha llegado a la conclusión que Telegram ha sido la plataforma elegida para el despliegue. Esto se debe a su coste gratuito, así como la facilidad y el manejo que proporciona esta plataforma con su BotFather, que gracias a ello se puede realizar un bot propio y darle su debida funcionalidad de manera más sencilla de entender y programar.

5.3 Implementación del bot

En este apartado se va a comentar la realización del bot, así como los pasos que eso ha conllevado.

5.3.1 Creación del bot (BotFather)

Una vez tomada la decisión de desplegar el bot en la plataforma Telegram, se decidió utilizar BotFather que es un bot padre que tiene Telegram con el propósito de crear bots de una manera más rápida y sencilla, el bot para este trabajo de fin de grado se llamara **tfg-gimnasio.**

Para crear o dar de alta a un nuevo bot mediante bofather, es bastante sencillo solo hace falta seguir una serie de pasos **(/start)-(/newbot)- "nombre del bot",** esto nos devolverá un Token el cual es necesario para que se pueda usar la API Bot de Telegram.[17]

5.3.2 Uso de la librería TelegramApi

Una vez dado el nombre al bot, se utilizó una api llamada "**TelegramBotAPI**"[18], que contiene una librería "telebot" que es una librería la cual proporciona una plantilla o BaseCode (Base de código) mediante la cual se puede empezar a programar pequeñas funciones sencillas, como por ejemplo dar un mensaje de bienvenida, así como administrar y personalizar las operaciones que va a tener el bot e incluso poder administrar los errores que mostrara el bot cuando una operación falle.

Para las configuraciones previas se importó tanto la variable (API_TOKEN) que contiene el token que genera el BotFather cuando se crea el bot, una variable

(URL_NGROK) en la cual se guardará la dirección http que nos genera el Ngrok, que es en donde nuestro backend este público y puede ser accesible, y por último también las salas como las operaciones que tendrá el bot.

```
import telebot
import requests
from telebot import types
from Bot.configurationBot.botConfig import salas, operaciones, API_TOKEN, URL_NGROK
from Bot.model.operation import Operation

bot = telebot.TeleBot(API_TOKEN)
request_dict = {}
```

Ilustración 30 import librería telebot

Para las operaciones que tendrá el bot y como se vio en el apartado de la realización del API, para probar el backend productivo en nuestra maquina local, hay algunas operaciones que solicitan de un nombre de sala, hay otras que directamente basta con ponerles el nombre de la operación, es por esto que se ha diseñado una clase del modelo de datos que es "**Operation**" , la cual tiene como primer parámetro "name" que es el nombre de la operación que aparecerá en el teclado de Telegram, el segundo parámetro "oppathlist" se refiere a donde se tiene que llamar al backend según la operación, el tercer parámetro "showkeyboard" se refiere si esa operación necesita una vez seleccionada, utilizar el teclado para referirse sobre que sala se quiere consultar dicha operación, el cuarto parámetro "indexsala" el cual se diseñó para la operación porcentaje sala, la cual devolverá la sala que tenga un porcentaje de ocupación menor que el que se indica en este parámetro(%). Y el ultimo parámetro que tendrá este modelo es "fieldJson" el cual tiene el formato de la salida que se espera que tenga la operación.

```
class Operation:

    # init method or constructor
    def __init__(self,
                 name,
                 oppathlist,
                 showkeyboard,
                 indexsala,
                 fieldJson
                 ):
        self.name = name
        self.oppathlist = oppathlist
        self.indexsala = indexsala
        self.showKeyboard = showkeyboard
        self.fieldjson = fieldJson

operaciones = [
    Operation('Gente en la sala', 'sala', True, -1, ["nombre", "id", "capacidad", "ocupacion", "contadorTotal"]),
    Operation('Ocupación Total', 'currentOcupation', False, -1, ["currentOcupacion"]),
    Operation('Sala menos ocupación', 'lessOcupation', False, -1, ["nombre", "id", "capacidad", "ocupacion", "contadorTotal"]),
    Operation('Sala mayor ocupación', 'maxOcupation', False, -1, ["nombre", "id", "capacidad", "ocupacion", "contadorTotal"]),
    Operation('¿Puedo entrar a una sala?', 'canEnter', True, -1, ["canEnterSala"]),
    Operation('Información general del gimnasio', 'info', False, -1, ["hora-apertura", "hora-cierre", "nombre", "direccion", "mensualidad", "n-salas", "ocupacionTotal"])
    Operation('Sala favorita de la gente', 'salaFavorita', False, -1, ["nombre", "id", "capacidad", "ocupacion", "contadorTotal"]),
    Operation('Porcentaje de sala', 'porcentajeOcupacion', True, 40, ["nombre", "id", "capacidad", "ocupacion", "contadorTotal"]),
    Operation('Información de TODAS las salas', 'infoTotalSalas', False, -1, None),
]
```

Ilustración 31 modelo operation

5.3.3 Operaciones y mensajes del bot

A continuación, se comentará las operaciones y mensajes que tendrá el bot.

5.3.3.1 Hola/Inicio

Esta operación /hola-/inicio se puede definir como un proceso conversacional entre el usuario y el bot. El usuario puede escribir tanto -**/hola** como -**/inicio**

y empezara un proceso conversacional, en el cual el usuario podrá consultar las diferentes operaciones que se han diseñado, y el bot devolverá una respuesta en base a las operaciones programadas y según los datos que el simulador tenga en ese momento.

Para explicar mejor este proceso conversacional de esta operación, se expondrá el siguiente diagrama, y asi se podrá entender mejor todo el proceso o flujo que conlleva la operación /hola-/inicio en el bot.

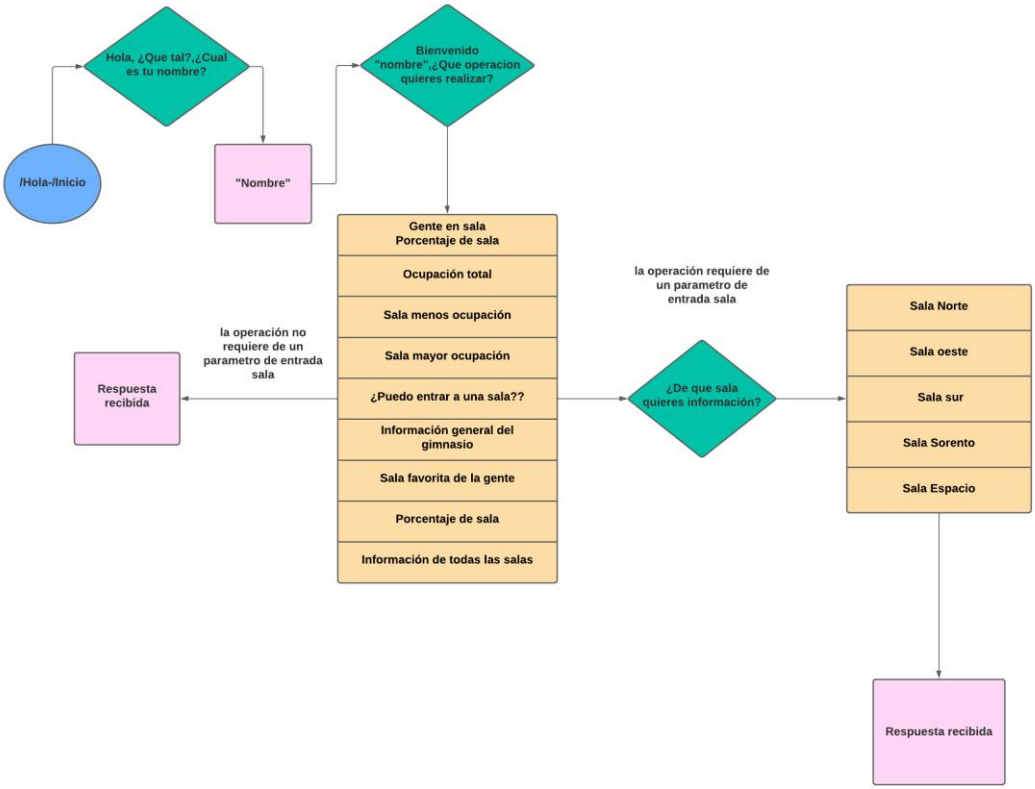


Ilustración 32 Diagrama operación /hola-/inicio

En cuanto a código, este diagrama se puede explicar de la siguiente forma.

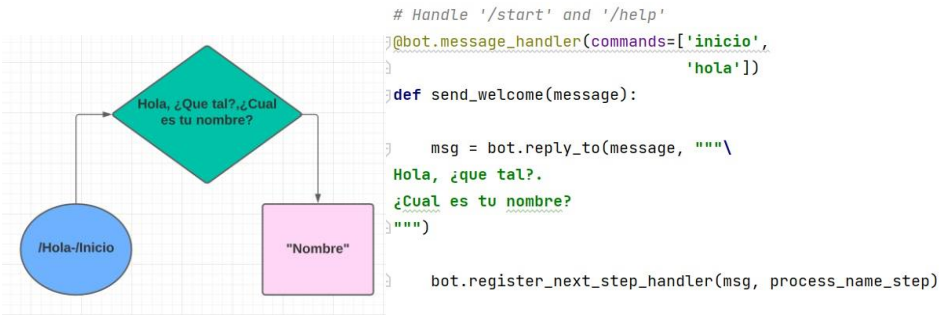


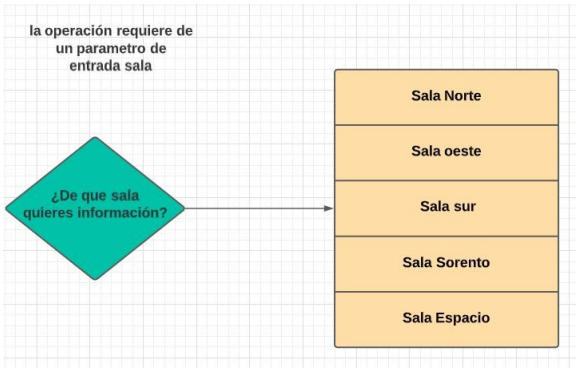
Ilustración 33 función welcome

En esta parte del código se programa la función de bienvenida, en la cual el usuario indicará el comando /hola-/inicio y el bot le responderá pidiéndole su nombre.


```python
def process_name_step(message):
    try:
        chat_id = message.chat.id
        name = message.text
        user = Request(name)
        request_dict[chat_id] = user
        user = request_dict[chat_id]

        markup = types.ReplyKeyboardMarkup(one_time_keyboard=True)
        for operacion in operaciones:
            markup.add(operacion.name)

        msg = bot.reply_to(message, 'Bienvenido ' + user.name + '\n ¿Que operación quieres realizar?', reply_markup=markup)
        bot.register_next_step_handler(msg, process_operation_step)

    except Exception as e:
        bot.reply_to(message, 'Ha habido un error al recogerel nombre de la sala \n' + str(e))
```

Ilustración 34 Función ¿qué operación quieres realizar?

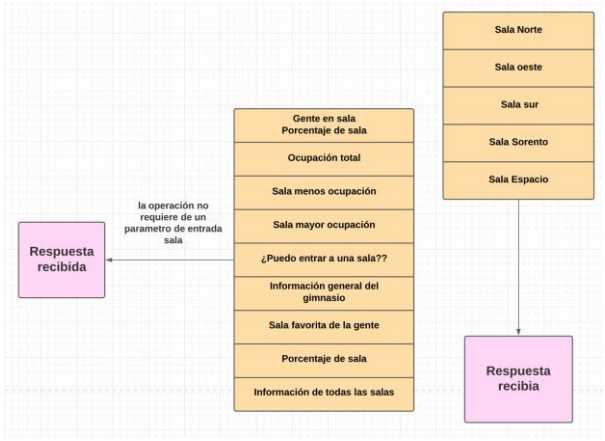
En esta parte, una vez que el usuario ha escrito su nombre el bot le pregunta "¿Qué operación le gustaría realizar?", así el usuario puede elegir qué operación desea realizar. Se controlan errores por si hubiera problemas al recibir la operación a realizar.




```python
def process_operation_step(message):
    try:
        chat_id = message.chat.id
        operation = message.text
        user = request_dict[chat_id]
        index = searchOperation(operation)
        if index != -1:
            user.operation = operaciones[index]

            if (user.operation.showKeyboard):
                if (user.operation.indexsala == -1):
                    markup = types.ReplyKeyboardMarkup(one_time_keyboard=True)
                    for sala in salas:
                        markup.add(sala)

                    msg = bot.reply_to(message, 'Operación seleccionada ' + operation + '\n ¿De que sala quieres información?',
                                       reply_markup=markup)
                    bot.register_next_step_handler(msg, process_sala_step)
                else:
                    user.indexsala = user.operation.indexsala
                    sendReponse(message, user)

            else:
                sendReponse(message, user)

        else:
            raise Exception("Operación NO reconocida")

    except Exception as e:
        bot.reply_to(message, 'Ha habido un error al recoger la operación')
```

Ilustración 35 función de que sala quieres información

En esta parte una vez que se ha seleccionado la operación, se pedirá de que sala se quiere consultar la operación. Esto en los casos que las operaciones requieran de una sala, para así poder generar la respuesta, en otro caso si la operación no requiere de seleccionar una sala, la respuesta se produce una vez seleccionada la operación, sin necesidad de pedir una sala en concreto.



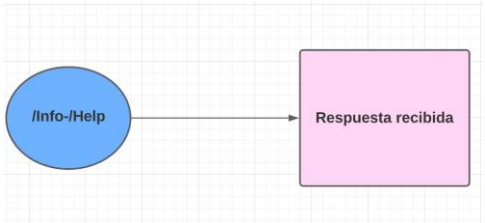

```
def sendReponse(message, user):
    url = ""
    if (user.operation.showKeyboard):
        url = URL_NGROK + "/" + user.operation.oppathlist + "?id=" + str(user.indexsala)
    else:
        url = URL_NGROK + "/" + user.operation.oppathlist

    response = requests.request("GET", url)

    bot.reply_to(message, user.operation.prettyOutput(str(response.text.replace("\'", "\""))))
```

Ilustración 36 función response

En esta última parte del código se genera la respuesta. Para ello se realiza una conexión con el backend gracias a la variable importada “URL_NGROK”, la cual permite tener una conexión con las operaciones programadas, así como con el simulador de datos y de esta manera dar la respuesta correcta.

5.3.3.2 Info/Help

Esta operación /Info-/help se puede definir como un proceso no conversacional, ya que cuando el usuario la consulta, el bot devolverá el resultado de una operación programada. Esta operación es la denominada como “información general del gimnasio” y es una operación que devuelve una información estática o general sobre el gimnasio, asi que su programación no ha sido difícil. A continuación, se expondrá el diagrama de esta operación con su correspondiente código de desarrollo.




```
# Handle '/info' and '/help'
@bot.message_handler(commands=['info',
                               'help'])
def send_info(message):
    chat_id = message.chat.id
    name = message.text
    user = Request(name)
    request_dict[chat_id] = user
    user = request_dict[chat_id]
    user.operation = _Operation('Información general del gimnasio', 'info', False, -1, ["hora-apertura", "hora-cierre", "nombre", "direccion", "mensualidad", "n-salas", "ocupacionTotal"])
    sendReponse(message, user)
```

Ilustración 37 función información general del gimnasio

6 Desarrollo de las pruebas y ejecuciones

6.1 Operación Gente en Sala

Para realizar esta operación el usuario ejecutara el comando **/hola** o **/inicio** (que son los comandos con los cuales se inicia este flujo conversacional), el bot solicitara el nombre. Una vez el usuario escribe el nombre, el bot le permitirá elegir qué operación desea realizar. En este caso se seleccionará "Gente en la sala".

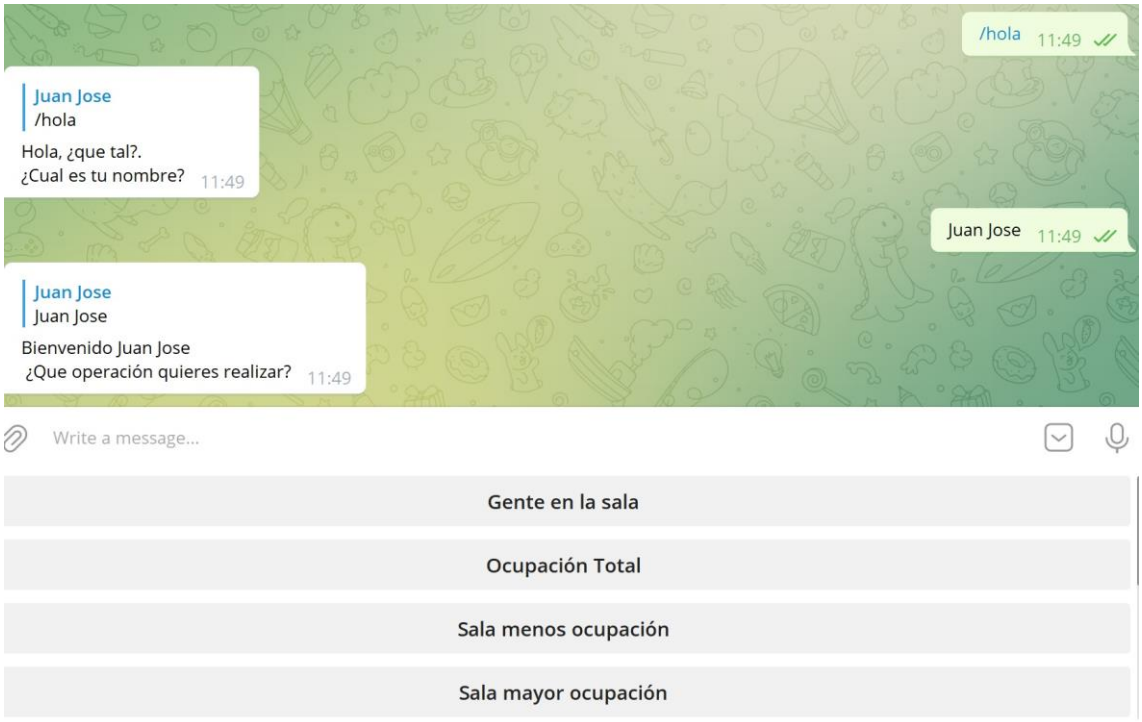


Ilustración 38 Operación Gente en Sala

Una vez seleccionada la operación, el bot pedirá de que sala se quiere esa información, se seleccionará la sala Norte, (o cualquier sala que el usuario quiera).

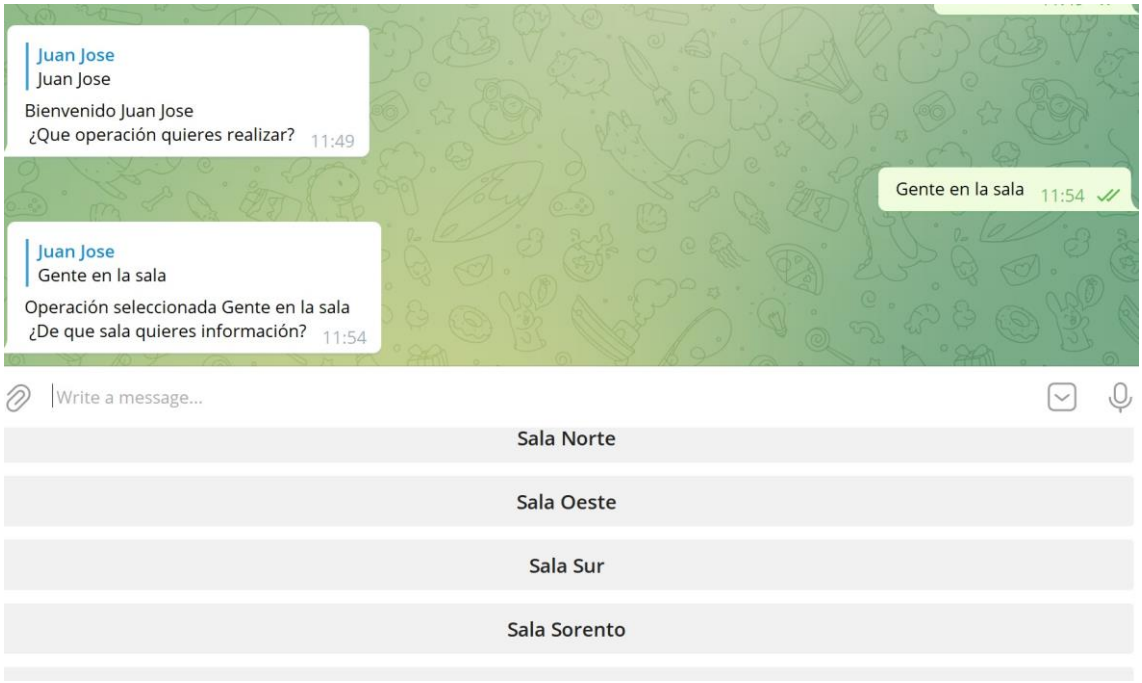


Ilustración 39 Operación gente en la sala (Norte)

En la siguiente imagen se puede ver como una vez seleccionada la sala que se desea consultar, el bot devuelve una respuesta con toda la información sobre la sala, Nombre, id, capacidad, ocupación actual y el contador total haciendo referencia a cuantas personas han entrado en esa sala desde que se ejecutó el simulador.

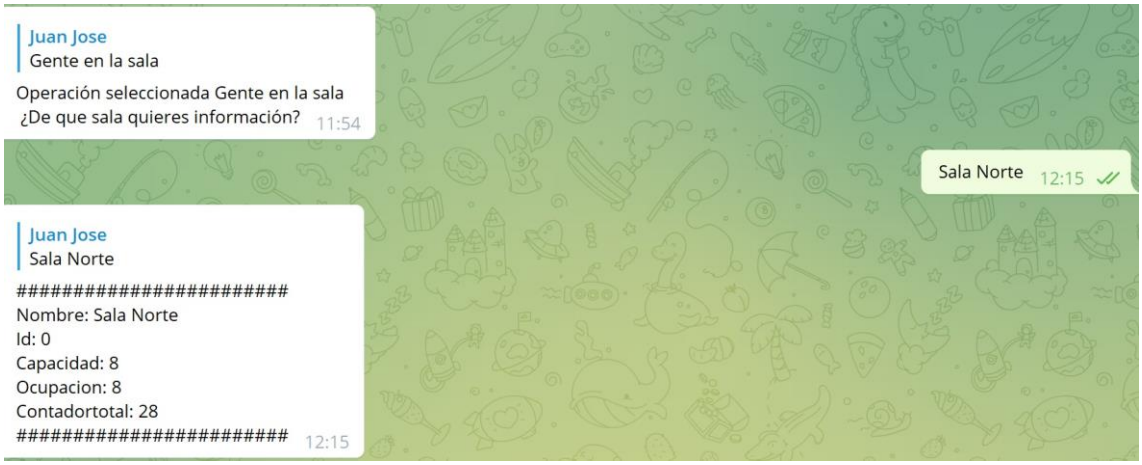


Ilustración 40 Operación gente en la sala (Norte) (1)

6.2 Operación Ocupación total

Para realizar esta operación, el usuario deberá enviarle al bot el comando /hola o /inicio, y siguiendo el flujo de todas las operaciones, el bot permitirá elegir qué operación se desea realizar, se seleccionará “Ocupación Total”, y el bot devolverá como respuesta, la suma de la ocupación de todas las salas, dándole asi una respuesta de ocupación total.

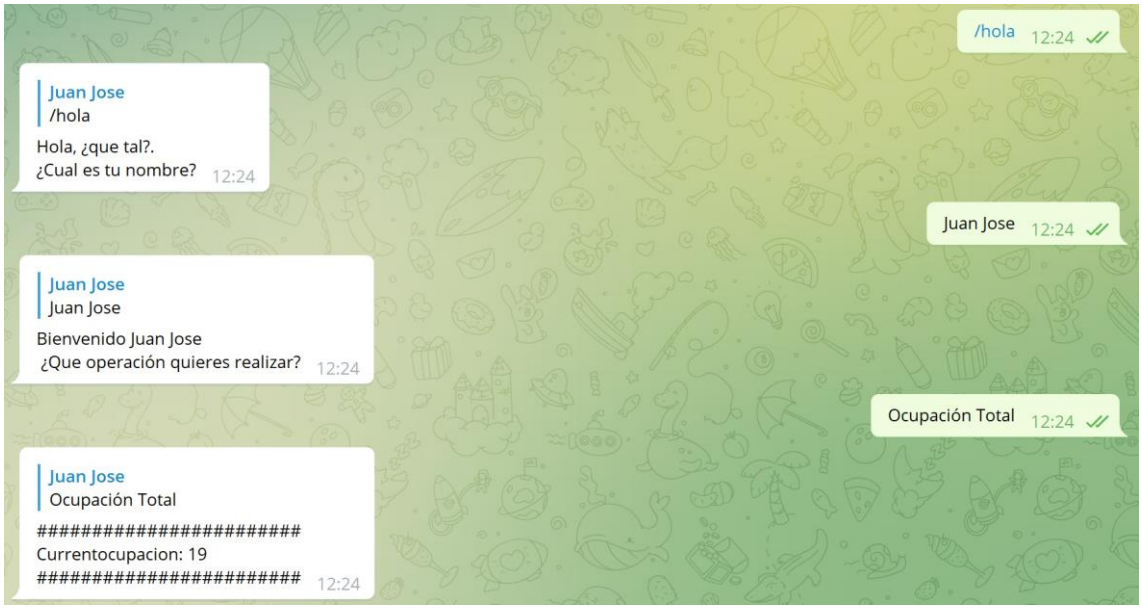


Ilustración 41 Operación Ocupación Total

6.3 Operación Sala menor/mayor ocupación.

Para realizar estas operaciones el flujo conversacional es el mismo que las anteriores conversaciones. El usuario ejecuta los comandos /hola -/inicio, una vez que el bot solicita el nombre y el usuario lo escribe, se podrá elegir estas operaciones. Como el resultado de estas operaciones es el mismo, con la diferencia que una devolverá la sala con mayor ocupación en el momento en que se ejecuta la operación y la otra la sala con menor ocupación en el momento se ha decidido adjuntarlas en este mismo punto (6.3).

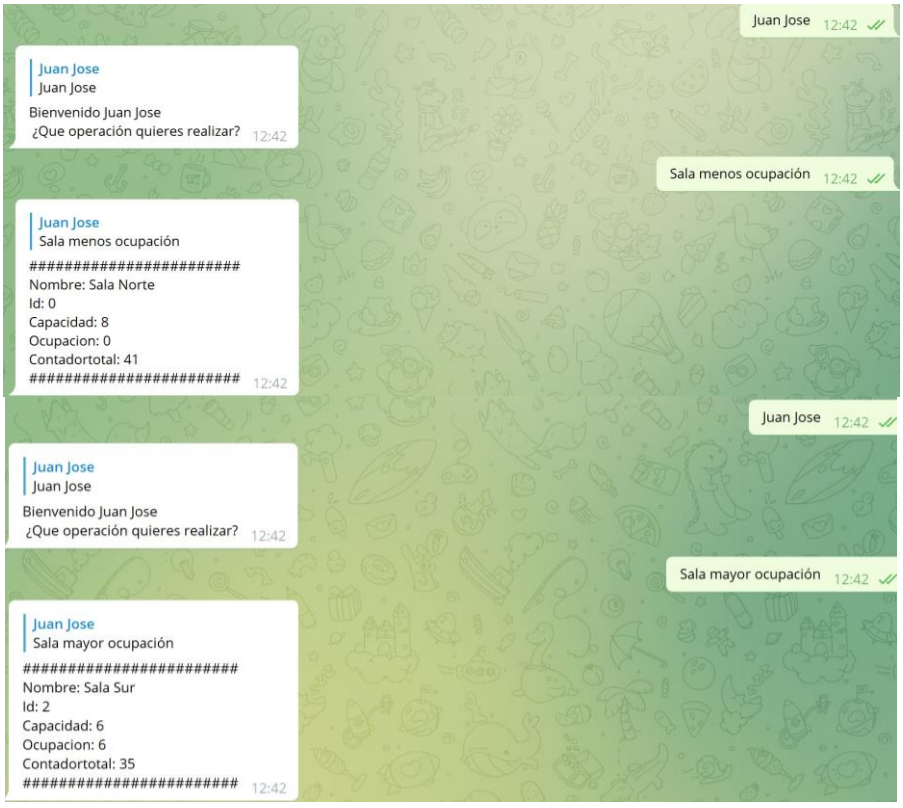


Ilustración 42 Operación Sala menor/mayor ocupación

6.4 Operación ¿Se puede entrar a una sala?

Para realizar esta operación, se sigue el mismo flujo conversacional descritamente anteriormente en las otras operaciones.

Esta operación "¿Se puede entrar a una sala?", tendrá valor cuando una sala tenga su capacidad totalmente llena, entonces en ese caso la respuesta será un "False", en otro caso será un "true". En el siguiente caso como se puede ver la Sala Sorento con id = 3 tiene una capacidad = 7 y una ocupación actual = 7. En este caso si se solicitara entrar en esta sala, el bot daría una respuesta "False".

```
ENTRAR - Sensor 3
Sala Norte
     Id: 0     Capacidad: 8     Ocupacion: 7     Contador Total: 14
Sala Oeste
     Id: 1     Capacidad: 7     Ocupacion: 4     Contador Total: 12
Sala Sur
     Id: 2     Capacidad: 6     Ocupacion: 4     Contador Total: 14
Sala Sorento
     Id: 3     Capacidad: 7     Ocupacion: 7     Contador Total: 15
Sala Espacio
     Id: 4     Capacidad: 6     Ocupacion: 3     Contador Total: 5
```

Ilustración 43 Operación entrar en sala

6.5 Operación Sala favorita.

Para realizar esta operación, se sigue el mismo flujo conversacional descritamente anteriormente en las otras operaciones.

Esta operación devolverá la sala favorita en todo el gimnasio. Es decir, la sala que haya tenido más entradas hasta ese momento (Contador Total más alto), será la sala favorita.

```
SALIR - Sensor 3
Sala Norte
     Id: 0     Capacidad: 8     Ocupacion: 0     Contador Total: 9
Sala Oeste
     Id: 1     Capacidad: 7     Ocupacion: 5     Contador Total: 13
Sala Sur
     Id: 2     Capacidad: 6     Ocupacion: 6     Contador Total: 14
Sala Sorento
     Id: 3     Capacidad: 7     Ocupacion: 2     Contador Total: 10
Sala Espacio
     Id: 4     Capacidad: 6     Ocupacion: 6     Contador Total: 11
```

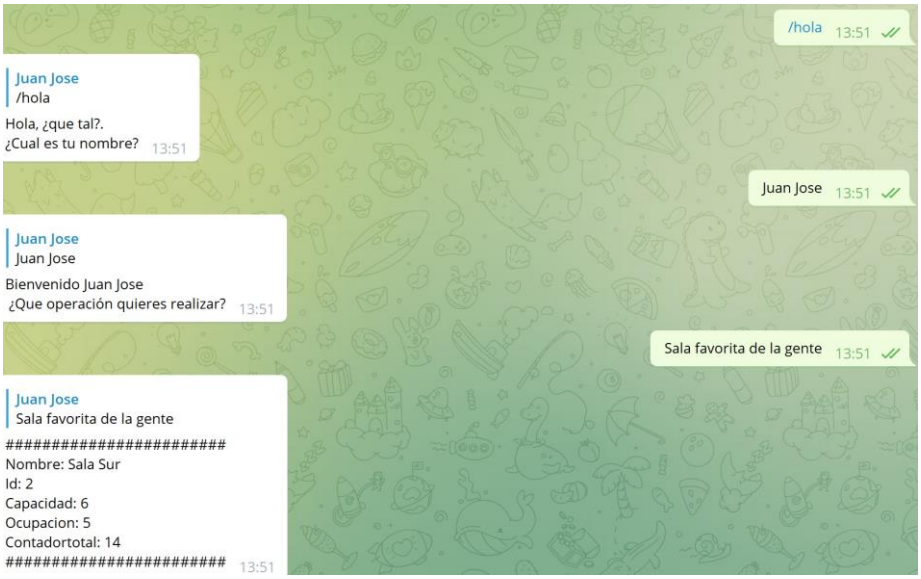


Ilustración 44 Operación sala favorita

6.6 Operación Información General del gimnasio.

Esta operación a diferencia de las anteriores tiene un flujo conversacional diferente. En este caso el usuario para llamar a esta operación deberá ejecutar el comando **/Info - /help.** El bot devolverá la información estática sobre el gimnasio.

Ilustración 45 Operación Información general del gimnasio

6.7 Operación porcentaje de sala.

En esta operación se devolverá la sala que tenga una ocupación menor, que la de un % dado. Este porcentaje se ha programado internamente en la fase de desarrollo del Backend/API. En este caso ha sido 40%.

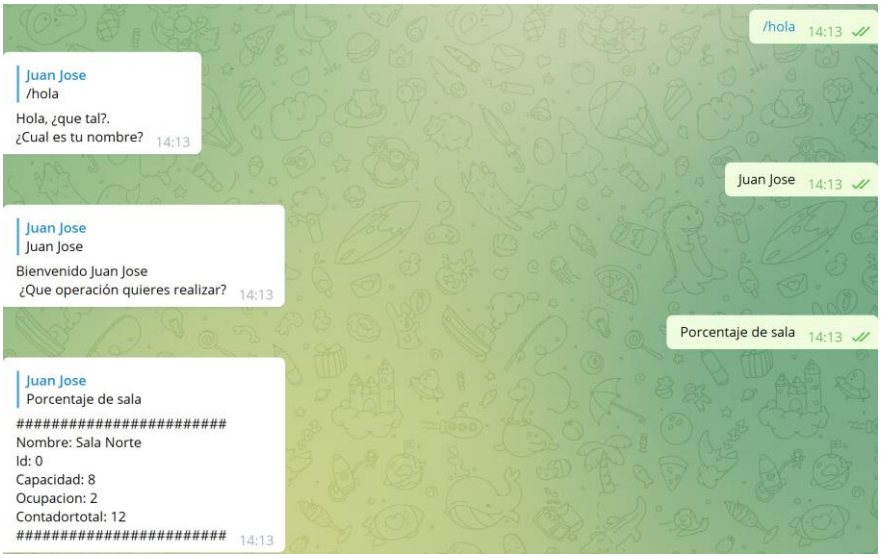


Ilustración 46 Operación porcentaje ocupación

Como se puede ver en esta salida, la sala Norte tiene una ocupación actual de 2 personas, que es menor que su 40% de su capacidad total (8 personas).

7 Resultados y conclusiones

El objetivo de este Trabajo de Fin de Grado fue fundamentalmente el desarrollo de un bot informativo en la red de despliegue de Telegram, asi como su previo desarrollo del backend/API, que es de donde el bot consulta las rutas para obtener las operaciones, como también el desarrollo del simulador de datos, que es la fuente principal de como inundar el backend de datos, para que luego al hacer cada llamada al bot, pueda dar resultados de acuerdo a los datos que el simulador está simulando en ese momento.

Durante el desarrollo de este proyecto, se pensaron y se desarrollaron operaciones que tuvieran lógica hoy en día con el objetivo de gestionar el aforo de un espacio cerrado con salas. Para asi de esta manera llevarlo al desarrollo del bot y hacer interesante este proyecto de fin de grado.

También se desarrolló un API la cual contiene toda la información de cada operación, como sus posibles salidas u errores que pueda generar, siendo asi esta una guía para cualquier persona que desee consultar sobre el contenido de este proyecto de fin de grado.

Los resultados presentados en el capítulo 6, son una clara evidencia que se han cumplido con los objetivos de este proyecto, desde su estudio, pasando por su implementación y desarrollo, llegando hasta las pruebas y despliegue final del bot.

Gracias a este proyecto realizado se ha podido aprender cómo desarrollar un bot, también a poder generar los datos, como desarrollar las operaciones que el bot va a necesitar. Dándole asi a este proyecto un gran valor e impacto, en cuanto al uso para poder gestionar el aforo de un espacio. Además, se ha aprendido sobre el lenguaje de Python, asi como el uso de las diferentes librerías o API'S que hay para poder desarrollar un bot.

En conclusión, el proyecto ha sido un éxito, sin embargo, tiene muchas líneas de mejora y crecimiento hacia futuro, como puede ser la sustitución del simulador de datos por un dispositivo IoT de detección de personas, el despliegue y almacenamiento del backend/API en la nube cloud, en vez del propio dispositivo local del desarrollador, incluso también se podría llegar a incidir en la sustitución de este backend por microservicios en la nube, con el objetivo que el backend tenga un alta disponibilidad y asi si hubiera alguna caída del backend, haya microservicios en la nube que garanticen el funcionamiento del bot.

También por último y como una gran medida de mejora sería la dotación de Inteligencia Artificial al bot, mediante el desarrollo de algoritmos o clasificadores, el bot pudiera ser capaz de interpretar el texto que un usuario le escribiera y dar una respuesta de la operación que el bot ha interpretado, según el texto escrito por el usuario.

8 Análisis de Impacto

En cuanto al análisis de impacto que este proyecto puede desarrollar, se pueden documentar ciertos puntos importantes.

En primer lugar, en el ámbito empresarial sería el que se vería más afectado positivamente por la implementación de este bot, ya que cualquier empresa o comercio, ya sea grande o pequeño, puede implantar este bot para su propio negocio, con el objetivo de poder tener una gestión del aforo a tiempo real. Como se comento en el capítulo 1 de este proyecto, la llegada de la Covid-19 o de futuras pandemias, han resentido mucho a los comercios que han tenido que ver limitado el aforo de sus comercios y esto llevándolos a tomar serias medidas económicas, es por eso por lo que la gestión de aforo hoy en día se ha convertido en un elemento a tener muy en cuenta según las medidas sanitarias en el momento.

Tener un gestor de aforo desde cualquier dispositivo tecnológico, también impactaría en gran medida en lo económico, siendo esta medida tecnológica mucho mas eficiente que tener un propio gestor o contador humano de personas, controlando de manera manual cuantas personas hay en una sala o cuantas personas salen o entran, ahorrando asi muchos gastos innecesarios para los comercios que dispongan de este gestor de aforo.

Por otro lado, y en el ámbito personal, este proyecto puede ayudar a muchas personas a planificarse su día de una manera mas organizada, ya que pueden consultar desde un dispositivo el aforo del lugar que desean visitar, de esta manera pueden tomar mejores decisiones, de si ir o no ir al lugar según el aforo que haya en ese momento.

Con respecto a lo analizado anteriormente, los Objetivos de Desarrollo Sostenible (ODS) de la agenda 2030 en los que podría influir son principalmente: Salud y bienestar (3) - Trabajado decente y crecimiento económico (8).

9 Bibliografía

[1] David Ramos, “silicon.es” 29 Enero 2021. [En línea]. Available:

https://www.silicon.es/a-fondo-tecnologia-gestion-aforo-pandemia-2432399

[2] latam.kaspersky, “kaspersky.com”. Available:

https://latam.kaspersky.com/resource-center/definitions/what-are-bots

[3] JuanJo Santana, “enredia.es” 11 Mayo 2017. Available:

https://www.enredia.es/que-son-los-bots-tipos-usos/

[4] Ramon Peris, “bloo.media”. Available:

https://bloo.media/blog/por-que-implementar-chatbot-en-tu-estrategia-de-marketing/

[5] “cloudflare.com”. Available:

https://www.cloudflare.com/es-es/learning/bots/what-is-a-web-crawler/

[6] “gb-advisoors.com”. Available:

https://www.gb-advisors.com/es/bots-maliciosos/

[7] “inboundmanagerpro.com”. Available:

https://inboundmanagerpro.com/por-que-invertir-en-marketing-es-una-prioridad/

[8] “neilpatel.com”. Available:

https://neilpatel.com/es/blog/bots-estrategia-de-marketing-digital/

[9] “reasonwhy.es”. Available:

https://www.reasonwhy.es/actualidad/chatbot-personas-problemas-vision-espanol-once

[10] “redhat.com”. Available:

https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces

[11] “Alfonsoma”. Available:

https://alfonsoma75.wordpress.com/2018/03/12/pythonanywhere-com-como-publicar-nuestra-webapp-python/

[12] “paradigmadigital.com”. Available:

https://www.paradigmadigital.com/dev/comparativa-servicios-cloud-aws-azure-gcp/

[13] Jesús Adrián Ponce, “sdos.es”. Available:

https://www.sdos.es/blog/ngrok-una-herramienta-con-la-que-hacer-publico-tu-localhost-de-forma-facil-y-rapida

[14] Johnn Hidalgo, “planetachatbot.com”. Available:

https://planetachatbot.com/mi-primer-agente-para-google-assistant/

[15] Yubal Fernández, “xataka.com”. Available:

https://www.xataka.com/basics/que-bots-discord-como-se-usan#:~:text=Los%20bots%20de%20Discord%20son,tareas%20un%20poco%20m%C3%A1s%20complejas.

[16] Tamina Steil, “userlike.com”. Available:

https://www.userlike.com/es/blog/api-de-whatsapp-business

[17] Jose María Lopez, “blogthinkbig.com”. Available:

https://blogthinkbig.com/crear-bot-de-telegram-botfather

[18] “github.com”. Available:

https://github.com/eternnoir/pyTelegramBotAPI