UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Desarrollo de un Sistema Domótico mediante Raspberry Pi Controlado por un Bot

TRABAJO FIN DE ESTUDIOS

GRADO EN INGENIERÍA TELEMÁTICA

Autor: Alejandro Nicolás Franco

Director: Mathieu Kessler

Codirector: Daniel Pérez Berenguer

Cartagena, 8 de diciembre de 2020

ÍNDICE

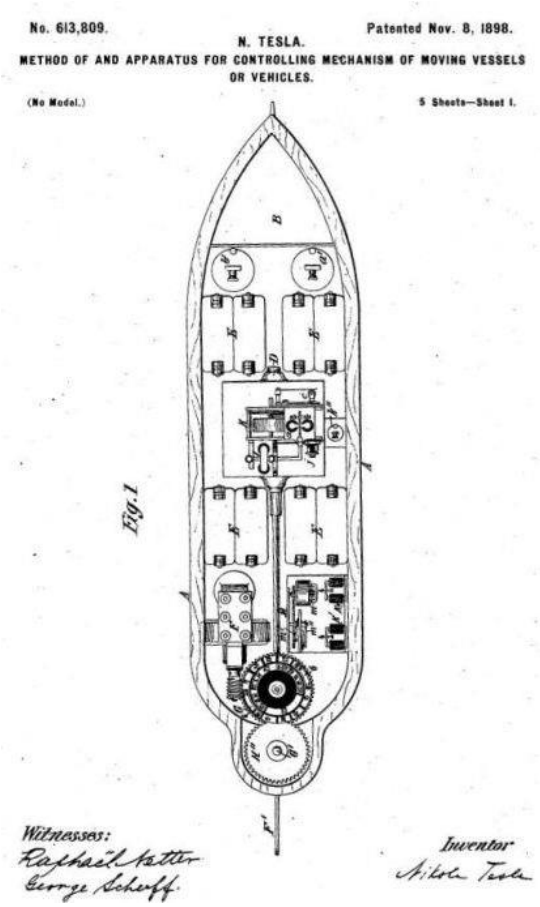
Tabla de ilustraciones

Introducción

Motivos y descripción del proyecto

Este proyecto se basa en desarrollar un sistema domótico mediante Raspberry Pi controlado por un bot. La domótica, por definición, son los sistemas capaces de automatizar una vivienda, mientras que los bots son programas destinados a realizar tareas complejas o repetitivas de manera simple y eficiente. A continuación, se va a analizar algo de la historia que rodea estas dos tecnologías.

En 1898, Nikola Tesla inventó un mando a distancia para controlar un pequeño barco de juguete mediante ondas de radio, el teleautómata [1]. Poco después, el ingeniero español Leonardo Torres-Quevedo utilizó transmisores inalámbricos usados en el telégrafo para controlar primero un triciclo, luego un pequeño bote e incluso lo intentó con torpedos [2].


No. 613,809. Patented Nov. 8, 1898.

N. TESLA.

METHOD OF AND APPARATUS FOR CONTROLLING MECHANISM OF MOVING VESSELS OR VEHICLES.

(No Model.) 5 Sheets—Sheet 1.

Fig. 1.

Witnesses:
Raphael Netter
George Scherff.

Inventor
Nikola Tesla

Figura 1 - Teleautómata de Nikola Tesla

A principios del siglo XX aparecieron los primeros electrodomésticos que facilitaban las tareas del hogar, como la primera aspiradora, inventada en 1901. Durante las siguientes décadas se inventarían las lavadoras, secadoras, tostadoras y demás aparatos eléctricos para el hogar, que suponían una reducción del trabajo realizado dentro de la vivienda [3].

1966 fue un año especialmente interesante, pues nacieron tanto el primer bot, ELIZA, como el primer dispositivo inteligente, ECHO IV. ELIZA fue desarrollada por el Instituto de Tecnología de Massachusetts como un bot que intentaba simular a una psicoterapeuta, así como para enseñar la superficialidad de comunicación entre personas y máquinas [4]. ECHO IV, por otro lado, era un dispositivo que podía realizar listas de la compra, controlar la temperatura de la vivienda, y encender y apagar otros dispositivos, aunque nunca se llegó a comercializar [5].

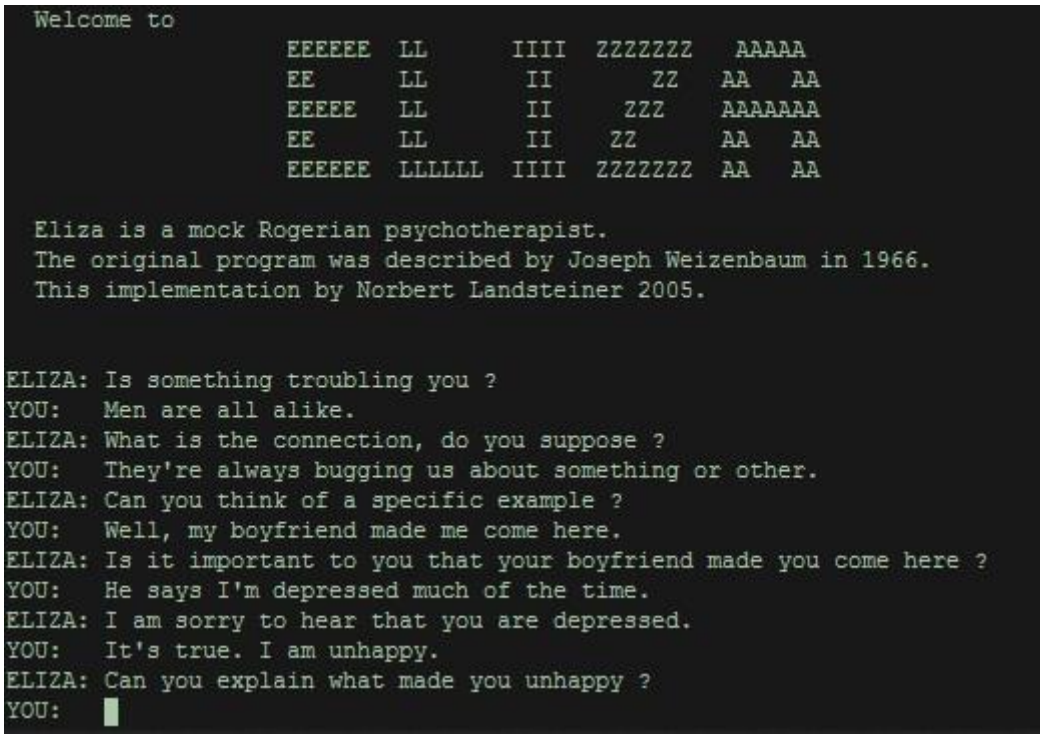


Figura 2 – Eliza

En 1975, en Escocia, Pico Electronics desarrolló X10, el primer protocolo de comunicación entre dispositivos usados para domótica. Los dispositivos X10, trabajaban con señales de radiofrecuencia que representaban información digital para controlar luces y temporizadores [6]. Poco después los años 80 se convirtieron claves para el consumidor común, con la aparición de luces detectoras de movimiento, puertas de garaje que se abren solas, termostatos programables y sistemas de seguridad, que además de ser bastante comunes, eran asequibles. Se acuñó el término Smart Home, en los que dispositivos de una vivienda se conectaban mediante la misma red [5].

Así fue como a finales del siglo XX, las Smart Homes comenzaron a ganar popularidad. Diferentes tecnologías emergieron y fueron lentamente siendo integradas en los hogares. Se volvieron aún más asequibles, y aparecieron más aparatos y dispositivos que trabajaban en conjunto con el resto del sistema [3]. Además, con la llegada de Internet,

aparecieron los bots, quienes se crearon al principio de una manera paródica y como forma de entretenimiento.

Hoy en día la domótica, tal como la conocemos, se encarga de automatizar ciertas tareas en una vivienda, desde controlar la iluminación a realizar trabajos de seguridad, como conectar una alarma a distancia, mientras que los bots nos acompañan día a día para hacer algunas tareas mucho más sencillas, como mirar el tiempo o reservar en un restaurante [7]. Ejemplos de bots actuales son Siri, el bot más famoso de Apple, o el Google Assistant de la empresa homónima [8].

Este proyecto se basa en desarrollar un sistema domótico sobre una Raspberry Pi a la que se añadirán elementos en los que se simularán algunos de estos dispositivos mediante diodos LED y timbres, así como implementar el software para controlarlo mediante el uso de bots en Microsoft Teams y Telegram. Además, se implementará un servicio REST en la Raspberry Pi que permitirá la gestión de los dispositivos simulados.

Objetivos

Se pretende desarrollar un sistema software/hardware para la gestión domótica mediante una Raspberry Pi que pueda ser controlado desde bots.

Para ello los distintos objetivos son:

- Implementar un conjunto de casos de uso para la gestión de dispositivos conectados a una Raspberry Pi. Por ejemplo, sistema de encendido/apagado de luces conectadas al dispositivo, escritura en pantalla de led o gestión de un timbre.

- Implementación de un servicio REST que permita controlar los diferentes dispositivos conectados a la Raspberry Pi, así como un servidor web que lo aloje.

- Desarrollo de un bot que pueda utilizarse a través de diferentes canales de mensajería como Microsoft Teams o Telegram.

- Gestión de las diferentes funcionalidades ofrecidas por el servicio REST a través del bot.

Fases del desarrollo

A continuación, se exponen las distintas fases en las que se ha dividido el trabajo:

1. **Creación de la cuenta de Microsoft Azure**
   Hay que crear una cuenta en el portal de Microsoft Azure, que permitirá gestionar todo lo relacionado con las bases de datos y el bot que se va a implementar.

2. **Preparación del software**
   Hay que elegir, descargar e instalar el software que se va a utilizar para trabajar, entre los que se incluye un IDE para C# y otro para Python, herramientas para comunicarse con la Raspberry Pi, y el software necesario para probar el bot.

3. **Creación del bot**
   Se descargarán las plantillas necesarias para desarrollar el bot utilizando C#, y se utilizará el portal de Azure para generar los recursos necesarios para alojarlo. Además, se programará un despliegue continuo con GitHub.

4. **Preparación de la Raspberry Pi**
   Se descargará e instalará el sistema operativo de la Raspberry Pi, así como actualizar Python e instalar las librerías y frameworks necesarios para montar el servicio REST y el servidor web.

5. **Implementación del servidor web**
   Se implementará un servidor web sobre la Raspberry Pi, sobre el que montaremos el servicio REST utilizando Flask y WSGI.

6. **Casos de uso**
   Utilizando una breadboard, se montarán los circuitos necesarios para los casos de uso que se activarán gracias al bot, entre los que se encuentran el encendido y apagado de las luces, hacer sonar un timbre, o mostrar un mensaje por una pantalla LCD.

7. **Implementación del servicio REST**
   Una vez montados los circuitos de los casos de uso, se testearán con scripts de Python, para ser implementados en el servicio REST más tarde. Se diseñarán los endpoints del servicio REST, y se asociarán a cada uno de los casos de uso.

8. **Desarrollo del bot**
   Se desarrollará el bot en C# utilizando Microsoft Bot Framework, añadiendo comandos que activarán los dispositivos de la Raspberry Pi, simulando el sistema domótico.

9. **Bots en Telegram y Microsoft Teams**
   Por último, se integrará el bot con Telegram y Microsoft Teams para que el bot pueda aparecer en ambas plataformas, y pueda ser utilizado por el público general.

Estructura del Trabajo de Fin de Estudios

El trabajo se va a estructurar de la siguiente manera:

Primero, se presenta la parte teórica, tanto para investigar sobre el estado actual de los bots como para hablar de los casos de uso que se van a implementar, el software que ha sido empleado, y las tecnologías que han hecho posible la realización del trabajo, incluyendo librerías y frameworks.

En segundo lugar, la parte práctica, en la que se describirá el proceso de diseño y desarrollo del trabajo, dividido en tres partes:

- El hardware, y todo lo relacionado con los casos de uso que instalamos sobre la breadboard conectada a la Raspberry Pi.
- La Raspberry Pi en sí, centrándose en el servicio REST y en el servidor web que ha sido implementado sobre ella.
- El bot, tanto su diseño y desarrollo como la implementación en los dos canales objetivo, Telegram y Microsoft Teams.

Por último, se sacarán conclusiones y se sugerirán unas líneas futuras para continuar el trabajo, además de encontrarse los anexos y los recursos empleados para el mismo.

Estado de los bots

Este capítulo se centrará en describir el estado actual de los bots, ofrecer una descripción básica de su funcionamiento, analizar los distintos tipos de bots que podemos encontrar y las cuestiones que pueden surgir por la interacción entre humanos y bots.

Un bot es un tipo de software programado para llevar a cabo unas tareas específicas. Generalmente, se usan para automatizar ciertas tareas, por lo que no es necesario que reciban instrucciones por parte de humanos, aunque no siempre es el caso. Algunas empresas los usan también para realizar tareas repetitivas de manera más rápida [9].

Los bots suelen operar sobre una red, que usan para comunicarse unos con otros. Es así como más del 50% del tráfico existente en internet se compone de bots que interaccionan con páginas web, hablan con usuarios, escanean por contenido o realizan otras tareas [10].

Hoy en día, hay multitud de tipos diferentes de bots, cada uno con sus propias tareas y funciones a realizar [11]. Algunos de los tipos más comunes son:

- **Chatbots**
  Un programa que simula una conversación con un ser humano. Ya se ha hablado antes de ELIZA, que simulaba ser una psicoterapeuta, analizando las entradas de texto que le dabas. Actualmente, existen varios bots que funcionan como asistentes personales.
- **Social Bots**
  Son bots que operan en plataformas y redes sociales. Últimamente su uso se ha extendido, y cada día aparecen más bots en páginas como Reddit, Twitter o Facebook.
- **Shopbots**
  Es un programa que localiza el mejor precio u otra característica de un producto que estés buscando y te manda la información para que lo compres.
- **Spiders o crawlers**
  Se usan para acceder a sitios web y recolectar su contenido para ser indexados por los motores de búsqueda.
- **Bots de monitorización**
  Usados para monitorizar la estabilidad de un sitio web o un sistema, analizando desde tráfico.

Aparte de los bots anteriormente vistos, hay otro tipo de bots maliciosos, que se usan para realizar un ataque automático en otros ordenadores, cometer fraudes o simplemente molestar al usuario de un sitio web social, como un foro o un chat [9].

Algunos de los bots maliciosos que se pueden encontrar en la actualidad son:

- **Spambots**
  Cuya función es recolectar correos electrónicos de páginas de contactos y similares para mandarles spam después.
- **Ataques DDoS**
  Ciber ataque en el que el atacante busca hacer que una red o un ordenador resulte inaccesible a sus usuarios, mediante una inundación de peticiones superfluas realizadas por bots en un intento de sobrecargar los sistemas.
- **Botnets**
  Redes de dispositivos conectados a internet, los cuales cada uno controla uno o más bots. Pueden ser usados para realizar ataques DDoS, robo de datos, o incluso para que el atacante acceda a otro dispositivo.
- **Viewbots**
  Que se usan para inflar las visitas a una web o a un vídeo de manera artificial.
- **Otros tipos**
  Existen otros tipos de bots nocivos, como revendedores de entradas, distribuidores de malware, y bots que aparecen en juegos en línea para recolectar recursos de manera automática.

La técnica que más se usa como medida anti-bots es el CAPTCHA, una forma del test de Turing que permite distinguir entre un usuario humano y un bot menos sofisticado, mediante el uso de imágenes o caracteres que el usuario debe identificar correctamente. Aun así, algunos bots están empezando a sortear el CAPTCHA mediante reconocimiento de caracteres o imágenes, o fallos en la seguridad. Es de esperar que se desarrollen nuevas contramedidas para combatir a los bots nocivos [9].

Figura 3 - Ejemplo de CAPTCHA

La interacción entre humanos y bots cada vez es más común en nuestro día a día, y las ventajas son cada vez más evidentes. Siempre están disponibles, por lo que pueden proporcionar un servicio de atención al cliente rápido y eficaz. También, otras empresas han creado bots para encargarse de gestionar acciones comunes dentro de su negocio, como reservar billetes o lanzar recordatorios para el usuario, lo que resulta en una ventaja clara tanto para el negocio como para los consumidores [9].

Además, se interactúa todos los días con los asistentes personales de nuestros móviles, el asistente de Google, Siri y Alexa, probablemente tres de los chat bots más conocidos del mundo. Los tres permiten al usuario hacer preguntas y obtener respuestas usando un sistema de inteligencia artificial avanzado.

Figura 4 - Logotipos de Alexa, Google Assistant y Siri

Incluso algunos videojuegos están empezando a utilizar bots, incluyéndolos entre sus mecánicas de juego como un elemento más. Un ejemplo reciente es "Interview with the whisperer", un juego desarrollado por la empresa española Deconstructeam, en el que eres un periodista haciendo una entrevista a un hombre que afirma haber inventado una radio que le permite hablar con Dios. Este juego utiliza la tecnología de los chatbots para convertir lo que escribes en algo que los personajes del juego pueden entender, aportando una nueva mecánica [12].

Figura 5 - Interview with the Whisperer, juego de Deconstructeam

Mientras la tecnología de los bots sigue avanzando, surgen dudas y preguntas respecto a ellos, así como preocupación respecto a los problemas que puede causar la comunicación con un robot social. Algunas de estas preocupaciones son cómo de efectivos resultan los mensajes del bot, problemas con la claridad y el entendimiento de los mismos, deshumanización de las respuestas, incomprensión por parte del bot de respuestas que podrían herir los sentimientos de las personas, así como la incapacidad de percibir cualquier tipo de creaciones genuinas del bot [9]. Esto es particularmente relevante cuando los bots interactúan con niños, pues el bot no hace distinción en base a la edad de las personas, y podría llegar a herir la sensibilidad de menores, enviar contenido inapropiado o simplemente utilizar un lenguaje demasiado complejo.

Finalmente, muchos de los avances que hay con los Chatbots son gracias al aprendizaje máquina o machine learning, en el que se entrena a los bots para que comprendan con efectividad el contexto de una frase y a contestar con mayor precisión. Durante los próximos años, se pueden esperar bots inteligentes y que “piensen” por ellos mismos gracias a las redes neuronales y la inteligencia artificial [13].

Casos de uso

En este capítulo, se describen brevemente los casos de uso que se han implementado como ejemplos de aplicaciones domóticas, que se pretenden controlar a través de nuestro bot. Aunque no sean aplicaciones domóticas reales, sirven para ilustrar los conceptos y permitirían, usando los mismos pasos, implementar una solución domótica real.

LED Rojo

El caso más sencillo y básico, que busca simular el encendido o apagado de una bombilla. Para ello, se utilizará un diodo LED de color rojo. Este LED emite una luz roja cuando recibe una señal.

Figura 6 - LED Rojo

LED Board

Este componente utiliza un array de luces LED conectadas en una barra, que permitirá realizar un pequeño juego de luces, así como simular el encendido de varias bombillas simultáneas.

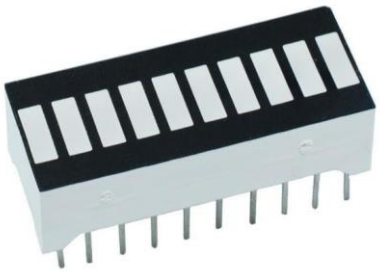


Figura 7 - LED Board

RGB LED

El último de los LED que se utilizará en el proyecto, un LED que permite recibir como entrada 3 señales distintas mediante PWM, que coge como valores de rojo, verde y azul, cambiando así el color del LED para adaptarse a la tonalidad y brillo que le digamos. Se utilizará este LED tanto para simular el cambio de color en una habitación, como para otros comandos del bot, como el de hacer saltar una alarma.

Figura 8 - LED RGB

Timbre

Un buzzer activo instalado en la placa, cuya función será emitir un número de pitidos regulares. Se usará para cualquier aviso sonoro, así como para la simulación de un timbre, el cual se configurará para emitir la cantidad de pitidos que sean requeridos.

Figura 9 - Timbre Activo

LCD Display

Un display LCD 1602 que mostrará tanto mensajes de avisos e información, como los mensajes que nosotros le mandemos usando el bot. Puede representar 2 líneas de 16 caracteres, entre los que se incluyen letras, números, símbolos y código ASCII.

Figura 10 - Display LCD 1602

Tecnologías usadas

En este capítulo, se presentan las diferentes herramientas que se usaron para alcanzar los objetivos planteados. Estas herramientas incluyen lenguajes de programación, frameworks, librerías, arquitecturas y software. Se hará un breve resumen de cada una, incluyendo su utilidad en el proyecto.

Lenguajes

Python

Figura 11 – Logotipo de Python

Python es un lenguaje de programación de alto nivel, orientado a objetos e interpretado, con semántica dinámica. Sus estructuras de datos de alto nivel, combinadas con tipado dinámico, lo hacen muy atractivo tanto para el desarrollo rápido de aplicaciones (RAD), como para lenguaje de scripts [14]. Su sintaxis sencilla y fácil de aprender enfatiza la legibilidad del código, y por lo tanto reduce su coste de mantenimiento. Soporta módulos y paquetes por lo que favorece la modularidad y reutilización del código. Además, su intérprete y su librería están disponible como código abierto [15]. En este proyecto, se utilizará para la parte de la Raspberry Pi, tanto para los casos de uso como para el servicio REST.

C#

Figura 12 - Logotipo de C#

C# es un lenguaje de alto nivel, multiparadigma y de propósito general, que acompasa tipado estático y tipado fuerte y programación imperativa, declarativa, funcional, genérica, orientada a objetos y basada en componentes. Fue desarrollado en el año 2000 por Microsoft como parte de .NET [16]. En el proyecto, se usará en la parte de programación del bot.

MySQL

Figura 13 - Logotipo de MySQL

MySQL es un sistema de gestión de bases de datos relacional, gratuito y de código abierto, utilizado en aplicaciones web que necesitan de bases de datos. En él, se organizan datos en una o más tablas cuyos elementos están relacionados los unos con los otros, estructurándolos de esta manera. Es el lenguaje usado en la base de datos de páginas web populares como Facebook, Twitter y YouTube [17]. En el proyecto, se va a utilizar para recoger datos del servicio REST y mandarlo en formato JSON, mostrándose por el display LCD.

Frameworks y librerías

GPIO Zero

Una librería muy simple que hace de interfaz para los dispositivos GPIO cuando utilizamos una Raspberry Pi [18]. Esta librería proporciona una manera sencilla de comunicarnos con los componentes de los casos de uso en la Raspberry Pi.

Flask

Figura 14 - Logotipo de Flask

Flask es un framework de web escrito en Python. Es considerado como un microframework porque no requiere de herramientas o librerías particulares. No tiene una capa dedicada para las bases de datos, ni validación de formularios, ni otros componentes que existen en otras librerías, pero soporta extensiones que incorporan funcionalidad como si hubiese sido implementada en el propio Flask. Este framework se utiliza de manera profesional en páginas muy conocidas como Pinterest o LinkedIn [19]. En este trabajo ayudará a crear el servicio REST, pues Flask es ideal para este tipo de servicios, ya que no se necesita una página web como tal.

Bot Framework SDK

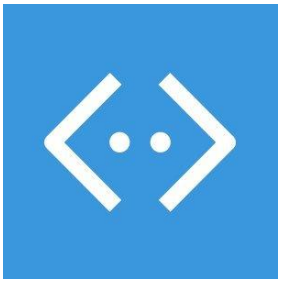


Figura 15 . Logotipo de Microsoft Bot Framework

El Bot Framework SDK de Microsoft Azure, proporciona herramientas para construir, testear, desplegar y gestionar bots inteligentes. Incluye un SDK modular y extensible, así como herramientas, plantillas y servicios relacionados con la inteligencia artificial. Con este framework se pueden crear bots que utilizan diálogos, entienden el lenguaje natural humano, responden preguntas y otros servicios [20]. En el proyecto se utiliza como la base para construir nuestro bot, que es el intermediario entre el servicio REST y el usuario.

Adafruit Python CharLCD

Es la librería de Python que se utiliza para acceder al LCD Display desde la Raspberry Pi. Con ella, se pueden escribir y borrar caracteres, establecer desde dónde se quiere que escriba, y borrar la pantalla [21]. Se utilizará para representar algunos mensajes en el proyecto.

Arquitecturas

REST

REST (Representational state transfer o transferencia de estado representacional) es un estilo de arquitectura software que define una colección de directivas para crear servicios web. Entre estas directivas se incluye un protocolo cliente/servidor sin estado, un conjunto de operaciones bien definidas, una sintaxis universal y el uso de hipermedios. Para definir un servicio web RESTful API, se tienen en cuenta los siguientes aspectos: un URI base, métodos HTTP estándar (GET, POST, PUT, DELETE), y un tipo de media [22]. Se utilizará para crear un servicio REST con endpoints para cada caso de uso.

Herramientas software

Portal Azure

Azure es la plataforma en la nube de Microsoft, que consiste en más de 200 productos y servicios. Con Azure, se puede construir, ejecutar y administrar aplicaciones sobre multitud de nubes. Desde dentro del portal, se puede gestionar la creación y administración de los recursos necesarios para el bot [23]. Además, se utilizará el portal para testear el bot en ciertas fases del desarrollo, así como para conectar con los canales de Telegram y Microsoft Teams.

Figura 16 - Logotipo de Azure

Visual Studio

Uno de los dos IDEs que se va a utilizar para la realización de este proyecto, Visual Studio pertenece a Microsoft. Tiene un editor de código con un componente de autocompletar código, así como de refactorización del código. Entre las múltiples ediciones que tiene, se va a utilizar la Community Edition, que es gratuita, pero cumple todas las necesidades requeridas [24]. En el proyecto, es el IDE que se va a usar para desarrollar el bot en C#.

Figura 17 - Logotipo de Visual Studio

PyCharm

El otro IDE que se va a utilizar, en este caso pertenece a Jetbrains y está especializado en lenguaje Python. Entre sus características, posee análisis de código, un debugger gráfico e integración con sistemas de control de versiones, así como multitud de plugins desarrollados por la comunidad [25]. En este caso va a ser utilizada la Community Edition, y se va a utilizar principalmente para el desarrollo del servicio REST, así como para las pruebas de los casos de uso.

Figura 18 - Logotipo de PyCharm

GitHub

GitHub es una plataforma de alojamiento de código utilizada para control de versiones y para colaboración. En él, se pueden crear repositorios de código en los que almacenar las distintas versiones de los desarrollos que se realicen, para llevar un mejor control de los cambios que se hagan en los proyectos, tanto en el bot como en el servicio REST [26]. Además, se va a utilizar el software GitHub Desktop, que es una aplicación de escritorio muy cómoda para subir los cambios realizados en los proyectos directamente a la web. GitHub será fundamental en el bot especialmente, ya que lo usaremos para desplegar automáticamente el bot gracias al portal de Azure, por lo que nos ahorraremos trabajo al testear el bot. Se han creado dos repositorios en este proyecto, uno para el bot, y otro para el servicio REST y todo lo relacionado con el servidor web.

Figura 19 - Logotipo de GitHub

Bot Framework Emulator

El Bot Framework Emulator es una aplicación de escritorio que permite testear y debuggear bots que utilizan el Bot Framework SDK que se ha explicado anteriormente. Usando esta aplicación, se pueden probar bots que estén ejecutándose localmente, o conectados remotamente a través de un túnel [27]. En este proyecto se va a utilizar para debuggear más efectivamente el bot que se desarrolle, de manera que no haya que estar desplegando el bot continuamente para poder probarlo.

WinSCP

WinSCP es una aplicación de código abierto y libre que hace de cliente FTP, SFTP o SCP. Su función principal es la de transferencia de archivos entre un dispositivo local y un dispositivo remoto, además de administración básica de archivos [28]. Se va a utilizar

para poder intercambiar archivos con la Raspberry Pi fácilmente, de manera que se pueda desarrollar el código en el ordenador, y probar los scripts en la Raspberry Pi.

SSH

SSH o Secure Shell, es un protocolo de red criptográfico para operar servicios de red de manera segura por una red no segura. Sus aplicaciones más típicas incluyen login y ejecución de comandos de manera remota, de manera que podemos acceder a un dispositivo desde el que nos encontramos [29]. Su función en nuestro proyecto será la de acceder a la Raspberry Pi desde el ordenador, así como poder ejecutar comandos de manera remota. De esta manera, se puede tener la Raspberry Pi sin interfaz gráfica, teclado, ratón u otros periféricos.

Postman

Postman es un entorno de desarrollo de APIs completo. Esta aplicación de escritorio permite testear y debuggear el servicio REST que se quiere implementar, ya que, gracias a su interfaz gráfica, se pueden comprobar los endpoints del servicio REST, así como los resultados que se obtendrían en caso de hacer peticiones HTTP [30].

Figura 20 - Logotipo de Postman

Diseño e implementación

Preparativos

El primer paso a llevar a cabo para la realización de este trabajo, es la investigación sobre los pasos a seguir para la creación de un bot, ya que es uno de los dos elementos principales. Se decide que el bot se desarrollará utilizando el Bot Framework SDK del servicio de bots de Microsoft Azure, usando para ello .NET y C#.

A continuación, hay que descargar e instalar las aplicaciones que se vayan a utilizar en el ordenador. Primero se descargan los IDEs, tanto el de C# como el de Python, y después se descargan las herramientas que se vaya a utilizar.

Se elige Visual Studio, pues es el IDE oficial recomendado por Microsoft para programar en C#. Además, ya se ha trabajado anteriormente con este entorno de desarrollo, lo cual ayuda a la hora de trabajar con él. Se puede descargar la Community Edition de manera gratuita desde la página de descargas de Visual Studio [31]. Se instala con las herramientas de .NET.

El segundo IDE que se va a instalar es PyCharm, de la empresa checa JetBrains. Este entorno, aunque no es gratuito de manera normal, se puede descargar e instalar utilizando una dirección de correo de la universidad, pues JetBrains dispone de programas para estudiantes que permiten el acceso a todas las aplicaciones de las que disponen [32]. También se ha utilizado este entorno anteriormente, por lo que no se malgastará tiempo aprendiendo a usarlo.

Se decide que se va a utilizar control de versiones en este proyecto, específicamente se usará GitHub, ya que fue usado en otras ocasiones. Se descarga e instala su aplicación de escritorio GitHub Desktop, y se introduce la cuenta que va a ser utilizada para almacenar el código de este proyecto. El código se dividirá en dos repositorios: uno para el bot, y otro para el servicio REST. Más adelante se crearán los repositorios utilizando los entornos de desarrollo.

Después, se va a crear la subscripción para utilizar el portal de Microsoft Azure. Para ello, entre las muchas opciones que hay disponibles, se va a elegir la cuenta de Azure para estudiantes [33]. Esta cuenta permite crear recursos dentro del portal de Azure, entre los que se encuentran los que hacen falta para la creación del bot, además de un presupuesto de cien dólares de crédito para desplegarlo. Una vez creada la cuenta, se confirma que existe acceso a todas las herramientas necesarias que proporciona Azure. Desde el portal, se puede ir a la documentación y a varios tutoriales para obtener recursos que ayuden a la hora de realizar la implementación.

Raspberry Pi

Primeros pasos

El dispositivo que se va a utilizar es una Raspberry Pi 4 B de 4 GB. Las Raspberry Pi son una serie de ordenadores de placa reducida que, aunque empezaron como un medio para aprender ciencias de la computación y robótica, se han ido desarrollando hasta usarse de muchas maneras distintas, gracias a su poco coste, modularidad y diseño abierto. Este modelo en concreto tiene cuatro núcleos, tiene Wi-Fi y Bluetooth incorporados y se le ha añadido como memoria una tarjeta MicroSD de 64 GB [34].

Figura 21 - Raspberry Pi 4 B

Para operar la Raspberry Pi primero se va a instalar un sistema operativo, en este caso Raspberry Pi OS, antes conocido como Raspbian. Este sistema operativo es el recomendado para un uso normal de la Raspberry Pi, y va a servir para el proyecto. Está basado en Debian y optimizado para el hardware de la Raspberry Pi, e incluye más de 35000 paquetes precompilados para facilitar su instalación. Además, es un proyecto comunitario que se encuentra bajo un desarrollo activo, centrado en mejorar la estabilidad y el rendimiento del mayor número de paquetes de Debian como sea posible [35].

Para instalar Raspberry Pi OS, se va a utilizar un lector de tarjetas SD y la aplicación de escritorio Raspberry Pi Imager, que se puede descargar en la página web oficial de Raspberry Pi. Esta aplicación es un escritor de tarjetas SD con interfaz gráfica que ofrece una manera muy sencilla de instalar un sistema operativo en una Raspberry Pi, ya que descarga la imagen necesaria y la instala automáticamente en la tarjeta SD. Una vez se introduzca la tarjeta en el lector, solo hay que conectarla al ordenador, ejecutar el Raspberry Pi Imager, seleccionar el sistema operativo que se necesita (en este caso Raspberry Pi OS 32-Bit), la tarjeta dónde instalarlo, y darle a escribir.

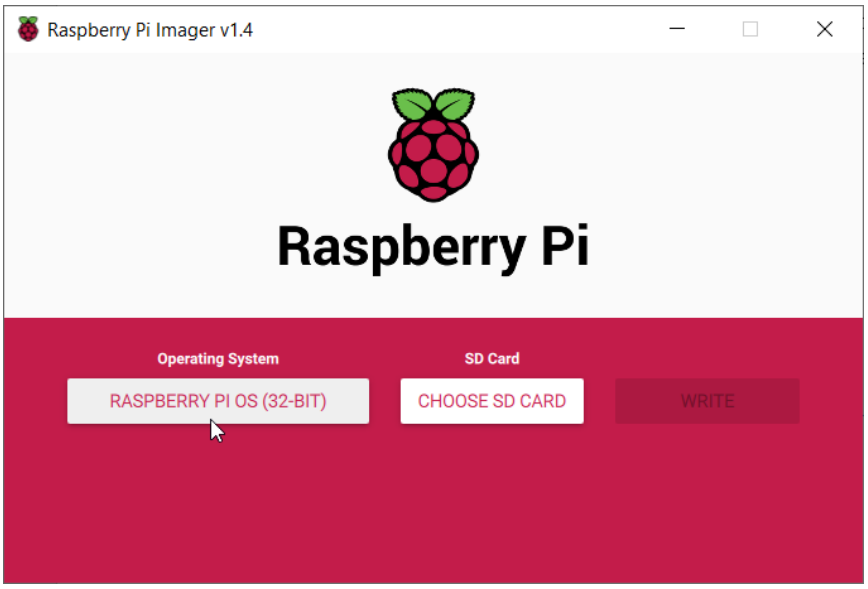


Figura 22 - Raspberry Pi Imager

Una vez instalado, se saca la tarjeta del lector y se inserta en la ranura correspondiente en la Raspberry Pi. Para proceder a configurarla, primero se deben conectarle algunos periféricos, como un ratón, un teclado, y una pantalla o monitor, utilizando para ellos los puertos USB de los que dispone, así como el Micro HDMI [36].

Cuando estén conectados estos periféricos, se enciende la Raspberry Pi y aparecerá la aplicación "Bienvenido a Raspberry Pi", que será la encargada de guiar al usuario por la configuración inicial que se llevará a cabo a continuación. En ella, se cambian ajustes básicos como la localización, el idioma y la configuración del teclado, así como seleccionar una nueva contraseña para el dispositivo. Por defecto, la Raspberry Pi usa el usuario "pi" y la contraseña "raspberry" en todos sus dispositivos, por lo que es recomendable cambiarla.

Una vez se haga esto, pedirá conectarse con una red Wi-Fi para realizar las actualizaciones de software correspondientes, tras las cuales se podrá reiniciar la Raspberry Pi para terminar de configurarla del todo.

El último paso será indicarle a la Raspberry Pi que use Python 3 en lugar de Python 2. Este dispositivo ya tiene instalado ambas versiones por defecto, pero usando el comando "python" ejecuta la versión 2 en lugar de la 3, que es la que se va a usar. Para solucionar esto, se usará un alias para decirle al terminal de Raspberry Pi que cuando se usa este comando, nos referimos a la versión 3 de Python.

Una vez completado este último paso, se habrá dejado la Raspberry Pi configurada y lista para usarse. De aquí en adelante, para conectar con la Raspberry Pi no hará falta tener conectado el monitor y el resto de periféricos, puesto que se accederá a ella con diversos programas de SSH. De esta manera, con tenerla encendida y conectada al Wi-Fi, se podrá comunicar con ella desde el ordenador. El siguiente paso será preparar el servidor web que alojará el servicio REST.

Servidor Web

La siguiente fase del proyecto es la de construir el servidor web que permita alojar el servicio REST. Para ello, se va a utilizar un framework web de Python. Un framework web es un conjunto de librerías, APIs, herramientas y demás componentes que están destinadas a apoyar el desarrollo de aplicaciones web. Por lo tanto, para hacer el servidor web que ejecute el código en Python para el hardware, se necesita un framework web de Python. Se va a decidir entre tres de los frameworks web más populares de Python: Django, TurboGears y Flask [37].

- **Django**
  Full-Stack, gratuito y de código abierto, este framework sigue el patrón de diseño modelo-plantilla-vista (MTV). Django fomenta un desarrollo rápido de aplicaciones, y se centra en automatizar lo máximo posible adhiriéndose al principio de Don't Repeat Yourself (No te repitas) [37].

- **TurboGears**
  Full-Stack y compuesto por varios componentes WSGI, TurboGears está diseñado alrededor de la arquitectura modelo-vista-controlador (MVC). Proporciona una aplicación web extensible y dirigida por base de datos en cuestión de minutos, y cuenta con un modo mínimo con el que puede funcionar como si fuera un microframework [37].

- **Flask**
  Un microframework web basado en herramientas para aplicaciones WSGI y que utiliza plantillas. Una de sus características es el soporte para peticiones del tipo RESTful, además de que es 100% compatible con WSGI. Recomendado para RESTful APIs [37].

Viendo los pros y contras de cada uno de los frameworks, se elige Flask por ser el que parece que mejor se adapta a las necesidades de la aplicación web que se desarrolla.

A continuación, se va a proceder a instalar Flask en la Raspberry Pi. Para ello, se ha decidido utilizar un entorno virtual en el que se instalarán las dependencias necesarias. Un entorno virtual permite administrar las dependencias del proyecto, de manera que se pueden tener varias versiones de la misma librería de Python sin que ello afecte a otros proyectos.

Se crea el entorno virtual y se activa. Una vez activo, se utiliza pip para instalar Flask. Pip es el administrador de paquetes de Python, y viene instalado por defecto. Con él, instalar paquetes se convierte en una tarea trivial, ya que lo único que hay que hacer es escribir una línea en la terminal.

Ya se tiene instalado Flask, pero se necesita un servidor de aplicaciones para almacenarlo. El servidor que se va a utilizar es uWSGI. Este proyecto permite ejecutar scripts de Python, ya que está basado en WSGI (Web Server Gateway Interface), que especifica cómo un servidor web y un script de Python deben comunicarse entre ellos. uWSGI convierte la respuesta de una aplicación de Python, en algo que un navegador es capaz de entender, una respuesta HTTP [38]. Por lo tanto, además de Flask, hay que instalar uWSGI utilizando pip, como se ha hecho anteriormente.

Se tiene la opción de utilizar el servidor nginx en conjunto con uWSGI, ya que nginx por sí solo no soporta el protocolo WSGI, pero para una aplicación de esta envergadura no se considera necesario, por lo que será propuesto para extender la aplicación en el apartado de futuras líneas.

Una vez instalados los paquetes necesarios, se puede empezar a desarrollar el servidor web. Para ello, se va a crear un archivo python tfg.py, en el que se comenzará a escribir el código de lo que será la aplicación web. La aplicación empezará de manera sencilla, con lo básico para funcionar y que así se pueda comprobar que funciona. De esta manera, hay que hacer que la aplicación de momento solo muestre un mensaje en el navegador cuando se va a la dirección establecida. Se ejecuta la aplicación, que está corriendo en el puerto 5000, y se comprueba que efectivamente aparece el mensaje en el navegador. Para ello, se introduce en la URL: localhost:5000. Esto le dice al navegador que la dirección a la que tiene que acceder es la del mismo ordenador que se está usando, y accediendo al puerto 5000. En próximos capítulos, se accederá utilizando la IP de la Raspberry Pi a través de internet [39].

A continuación, se va a crear el archivo que funcionará como punto de entrada de la aplicación, que le dirá al servidor uWSGI cómo interactuar adecuadamente con la misma. Una vez hecho esto, se tendrá la aplicación escrita, y un punto de entrada establecido. Se procede entonces a configurar uWSGI.

Para ello, se crea un archivo .ini, al que se le pasa el módulo que se va a usar en la aplicación, se le indica que debe empezar en el modo maestro con cinco procesos para servir las peticiones entrantes. De esta manera, se termina la configuración de uWSGI.

Una buena opción ahora sería hacer que la aplicación se ejecute directamente al encender la Raspberry Pi. Para conseguir esto, se debe usar un archivo systemd, pero no se va a proceder de esta manera, ya que podría necesitarse usar la Raspberry Pi sin necesidad de activar el servidor.

De esta manera, el servidor web estaría terminado, ejecutando Flask sobre uWSGI. El siguiente paso será empezar a configurar el servicio REST encima de Flask, que se realizará en la siguiente fase de desarrollo.

Empezando el servicio REST

Para empezar con la aplicación, y dado que aún no se tienen los componentes necesarios para implementar los casos de uso objetivos del proyecto, se va a comenzar realizando una API RESTful en Flask. Una API (Application Programming Interface) se refiere a la parte de un programa diseñado para ser manipulado por otro programa. En este caso, se va a crear una API web, la cual permite que esta comunicación entre programas se produzca a través de internet.

Esta API conectará con una base de datos que se creará en el portal de Azure, y devolverá los datos que se le pidan mediante una petición GET. Primero se hará el esqueleto del servicio REST API, y más adelante conectará con la base de datos que también deberá ser desarrollada.

A la hora de hacer el esqueleto del servicio REST API, se debe entender primero cómo funciona exactamente Flask. Flask mapea peticiones HTTP a funciones de Python, de manera que cuando se produce una conexión con el servidor, comprueba si coincide la ruta a la que se quiere acceder y una función definida. Luego, ejecuta el código de la función y muestra el resultado en el navegador. Este mapeo de peticiones a funciones en Flask se llama routing o enrutamiento, y además de indicar qué función se activa con qué ruta, hay que decir tipo de peticiones HTTP están permitidas. En este caso, como sólo se necesitan devolver valores, las peticiones serán de tipo GET.

Se diseñan una serie de rutas que servirán como puntos de acceso a la API. En este caso, se va a crear una ruta para cada una de las tablas que se creen en la base de datos. Estas rutas devolverán la información específica de un elemento contenido en esas tablas o, por defecto, todos los elementos. El siguiente paso será diseñar la base de datos, pues sin tenerla bien definida no se podrán elegir adecuadamente las rutas y las operaciones que tendrá que realizar la función con los datos.

Para diseñar la base de datos hay que crear el recurso en el portal de Azure. Primero, se añaden a los recursos un servidor SQL. A continuación, dentro de ese servidor SQL se añade una nueva base de datos SQL.

Esta base de datos dispondrá de tres tablas:

- **Student**
  Define un estudiante, y contiene las columnas DNI, Nombre, Apellido1, Apellido2 y Correo.
- **Notification**
  Define una notificación, y contiene las columnas Id, SendFrom y Body.
- **Alert**
  Define una alerta, y relaciona las dos tablas anteriores. Contiene las columnas Id, Student y Notification.

Una vez definidas las tablas, y creadas como recursos en el portal de Azure, se añade una fila de datos a Student para futuras comprobaciones. Después, en el código del servicio REST se añade, para cada una de las tablas que se acaban de crear, una ruta predefinida para acceder a los datos. La URI base que vamos a utilizar para acceder al servicio REST es "/api/v1/", sobre la cual se añadirán sustantivos para indicar las rutas de acceso a cada función de Python.

Ahora se necesita el puente de enlace entre el servicio REST y la base de datos de Azure. Para ello se necesitan los drivers ODBC y de Python para el servidor SQL, además de pyodbc, un paquete de Python para conectarnos a bases de datos ODBC (Open Database Connectivity) [40].

Tras múltiples pruebas, y debido a la arquitectura de la Raspberry Pi, los intentos de utilizar los drivers ODBC fallan. Se prueba a utilizar otros paquetes como pymssql, pero también fallan las pruebas que se realizan. Finalmente, se consiguen instalar los drivers necesarios utilizando FreeTDS, una colección de librerías que permite conectar con las bases de datos de los servidores SQL de Microsoft [41]. De esta manera, se utiliza pyodbc con los drivers de FreeTDS para conseguir conectar con las tablas de la base de datos. Se realizan pruebas con cada una de las tablas, y se configuran las funciones en el script de Python para que se devuelvan al navegador los resultados de estas mediante JSON.

Para ayudar con estas pruebas, se utiliza el software Postman, que permite realizar peticiones HTTP a una ruta específica y ver el resultado, así como otros parámetros que pueden resultar útiles en caso de necesitar debuggear la aplicación.

Una vez acabadas las pruebas y confirmando que todo funciona correctamente, se puede dar por concluida esta fase del desarrollo. Finalmente, se va a crear un nuevo repositorio en GitHub que almacenará el código del servicio REST, así como el resto de pruebas que realicemos en Python. De esta manera, se llevará un control de versiones sobre el proyecto, y por lo tanto no se perderá ningún desarrollo.

El siguiente paso es diseñar implementar el circuito en una breadboard, para probar los componentes hardware del proyecto. Una vez esté completada esta parte, se editará el código del servicio REST para que cumpla los objetivos marcados.

Hardware

Introducción

El siguiente punto que fue desarrollado en el proyecto fueron los casos de uso de los que se han hablado, de manera que se diseñaron y se implementaron sobre una breadboard los circuitos necesarios para cumplir con las necesidades de la simulación del sistema domótico.

Primero, se eligieron los sistemas que se querían implementar. Se cuenta con varios componentes para desarrollar el sistema domótico. Entre estos componentes se encuentran luces LED de diferentes tipos, timbres activos y pasivos, motores, teclados numéricos y pantallas LCD. Se probaron estos componentes y se utilizaron para aprender a usar la librería que permite controlar los pines GPIO de la Raspberry Pi.

La Raspberry Pi 4 B cuenta con 40 pines GPIO, que pueden ser designados tanto como entrada como salida. En la siguiente imagen, se puede ver una descripción de cada uno de los 40 pines, junto con la etiqueta que le corresponde.

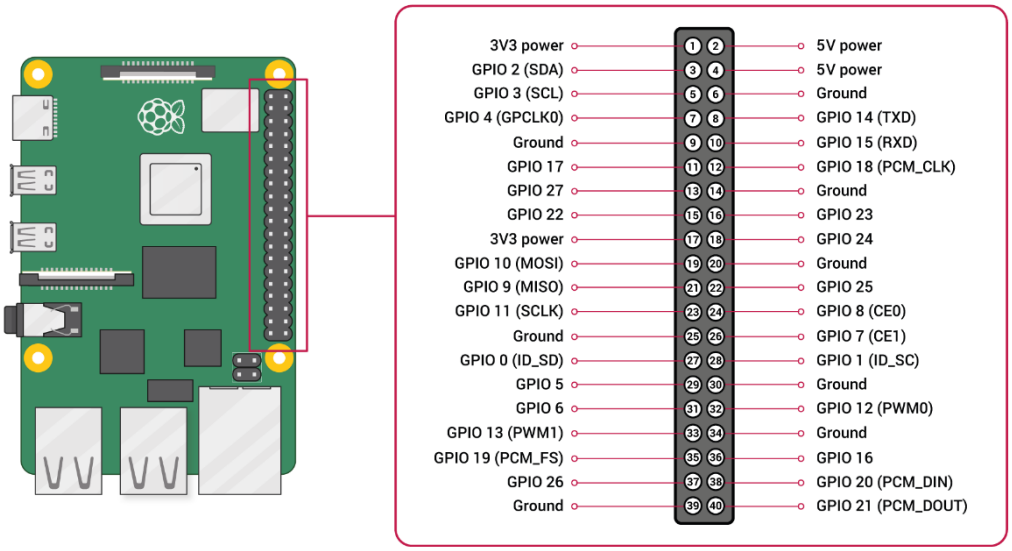


Figura 23 - Pines GPIO de la Raspberry Pi. Imagen descargada de la página de Raspberry Pi.

Hay una manera de utilizar estos pines de forma nativa con Python, ya que la librería para controlarlos viene instalada por defecto en todas las Raspberry Pi, pero se va a utilizar una librería más avanzada que también viene por defecto en el dispositivo. Se trata de gpiozero, una librería creada por Ben Nuttall de la fundación Raspberry Pi. Esta

librería tiene la ventaja de reducir la preparación del pin GPIO a una simple línea de código, además de implementar diferentes métodos para varios componentes diferentes, tanto de entrada como de salida.

Para empezar a trabajar y adaptarse a esta librería, se van a realizar scripts de prueba para algunos de los componentes, de manera que se pueda experimentar de primera mano cómo será el código que hay que hacer en el servicio REST, así como los circuitos que se van a implementar en la breadboard [42].

Para poder trabajar con la breadboard utilizando los pines GPIO de la Raspberry Pi, se va a utilizar una extensión de pines específica para el dispositivo. De esta manera, se pueden probar los circuitos de manera rápida, ya que vienen etiquetados todos los pines.

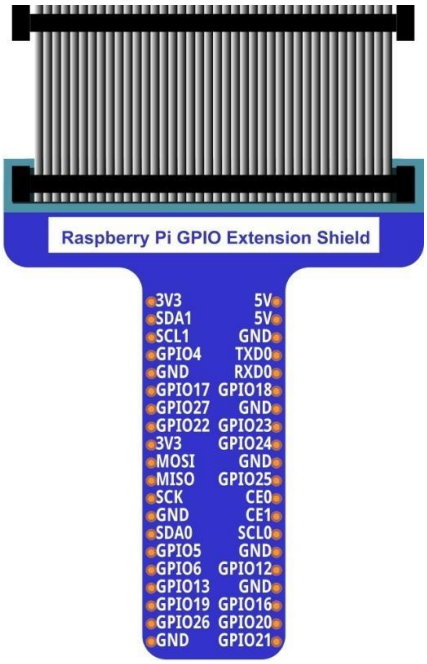


Figura 24 - Raspberry Pi GPIO Extension Shield. Imagen de Freenove.

Circuitos Básicos

LED

El circuito más sencillo, en el que simplemente se encenderá y se apagará un diodo LED que se conectará al pin GPIO 17 de la Raspberry Pi [43].

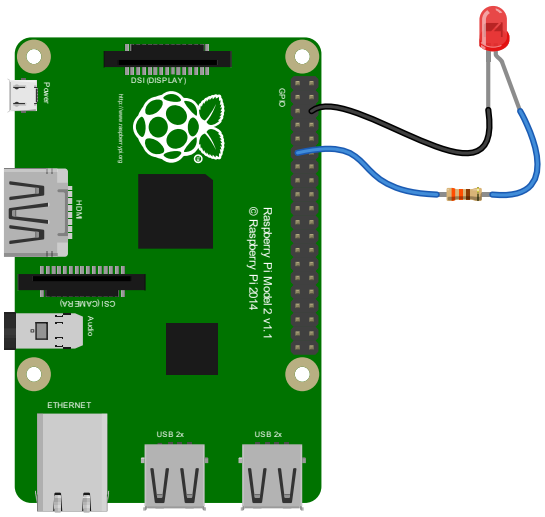


Figura 25 - Circuito LED. Imagen descargada de la documentación de gpiozero.

LEDBoard

Este circuito y su correspondiente clase en esta librería, permiten controlar una colección de LEDs conectados en la Raspberry Pi. De esta manera, es sencillo controlar varios LEDs conectados de forma consecutiva para realizar un juego de luces, o encender o apagar un LED específico [43].

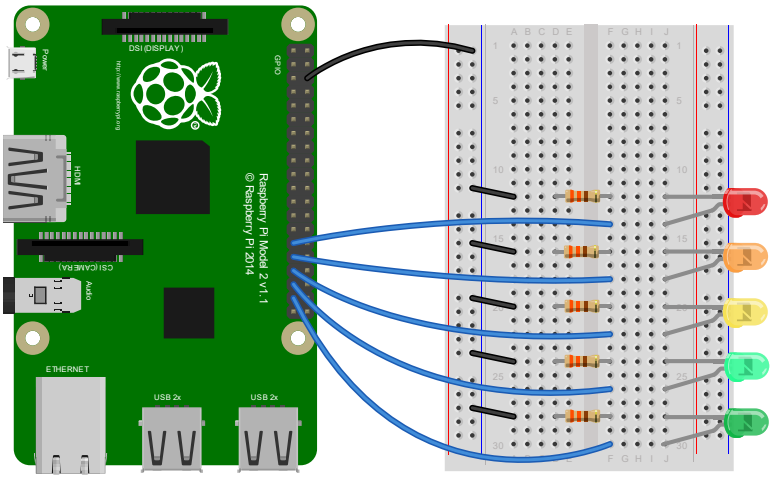


Figura 26 - Cirucito LED Board. Imagen descargada de la documentación de gpiozero.

RGBLED

Este circuito utiliza la clase RGBLED para cambiar los colores de un LED RGB conectado a tres pines distintos de la Raspberry Pi. Esta clase utiliza PWM (Pulse-width-modulation) o modulación por ancho de pulsos para cambiar la intensidad de cada una de las tres patillas del LED RGB. Cada una de estas patillas controla una de las luces de este LED, una roja, una verde y una azul. De esta manera, con la combinación de diferentes intensidades, se pueden crear varios colores en el mismo LED [43].

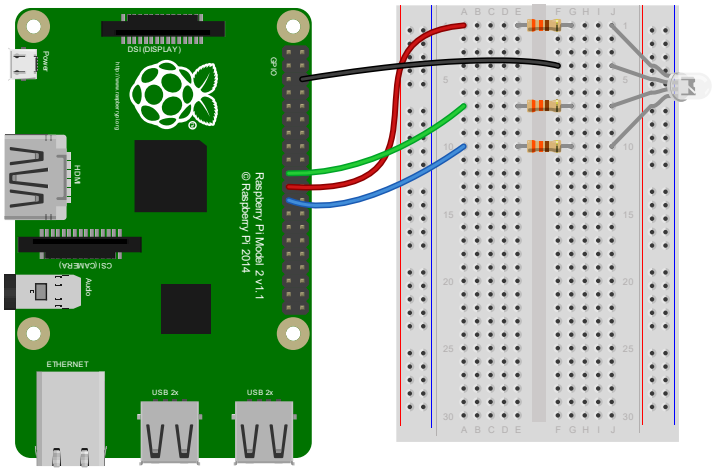


Figura 27 - Circuito LED RGB. Imagen descargada de la documentación de gpiozero.

Timbre

Para este circuito, se utiliza un active buzzer o timbre activo y se aprovechará para comprobar el funcionamiento de las clases para los componentes de entrada, como un botón. Además, para que el timbre funcione correctamente, se utilizará un transistor. De esta manera, el código del script de prueba hace que cuando se pulsa el botón del circuito, suena el pitido del timbre. Para esta prueba se han utilizado las clases de gpiozero Buzzer y Button [44].

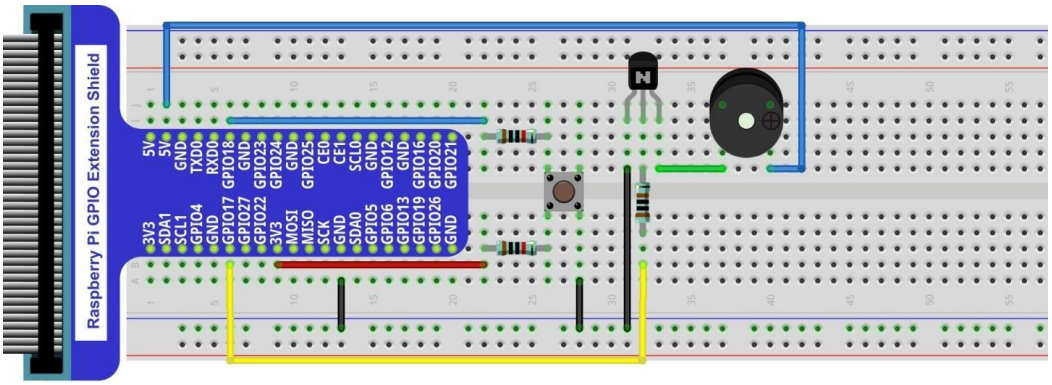


Figura 28 - Circuito Timbre. Imagen descargada de Freenove.

LCD Display

El último circuito básico de prueba que se va a implementar es en el que se probará a utilizar la pantalla LCD. El código de esta prueba es un poco más complicado que el de las demás, ya que la librería gpiozero no cuenta con soporte para una pantalla LCD, y por lo tanto habrá que utilizar la librería básica que controla los pines GPIO de la Raspberry Pi. Además, la pantalla LCD integra una interfaz I2C, lo que permite operarla utilizando sólo cuatro cables. Como resultado de esto, habrá que configurar I2C, ya que por defecto viene desactivada en la Raspberry Pi, además de instalar el paquete smbus. Una vez hecho esto, se comprobará el circuito y el código, que consiste en hacer aparecer en la pantalla LCD la hora actual, y la temperatura de la Raspberry Pi [44].

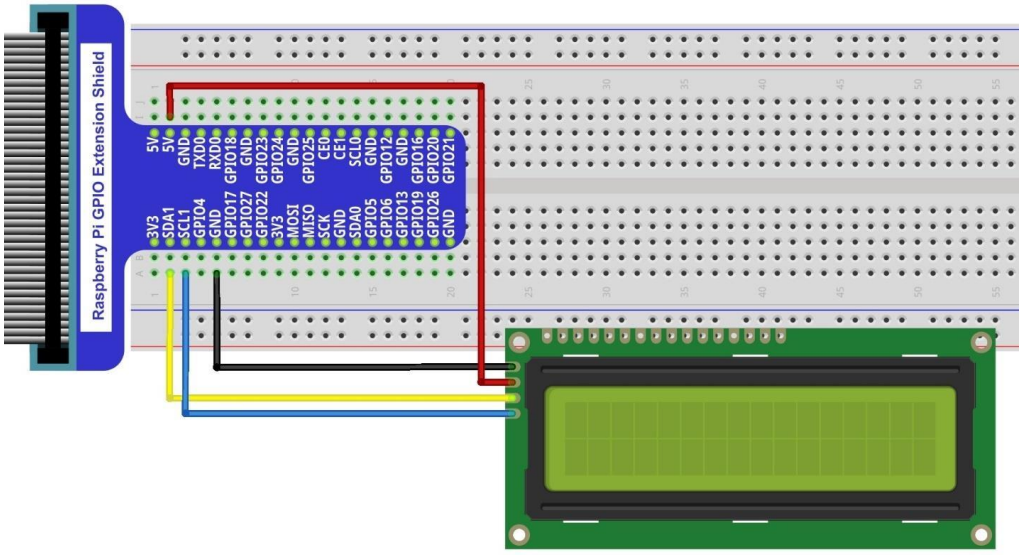


Figura 29 - Circuito LCD. Imagen descargada de Freenove.

Implementación

Una vez comprobado el funcionamiento de la librería gpiozero, la librería para controlar el LCD, y los distintos circuitos que hemos implementado para hacer las pruebas, se puede continuar con el proyecto. En este caso, el objetivo actual es el de diseñar un nuevo circuito que contenga todos los componentes necesarios para conseguir tener los casos de uso sobre la breadboard. De esta manera, se podrá utilizar un solo script para controlar cada dispositivo.

Para las luces, se decide utilizar: el diodo LED rojo, una luz muy básica que simplemente se enciende y se apaga; la barra de LEDs, un array de diez LEDs puestos de manera consecutiva que permitirá crear algún juego de luces; y el LED multicolor, o RGB, que cambia de color eligiendo combinaciones distintas de rojo, verde y azul. Junto a las luces,

se pondrá el circuito del timbre, pero esta vez sin el botón, pues no se consideró necesario.

Finalmente, se añade la pantalla LCD conectada a la Raspberry Pi mediante cuatro cables, sin tener que estar conectada a la breadboard. Así, se tendrá más flexibilidad a la hora de colocarla donde sea necesario.

Ya que se dispone de cables de distintas longitudes y colores, y como hay espacio de sobra en la breadboard, se van a intentar colocar los componentes de la manera más eficiente posible. Poco a poco, se irán añadiendo los dispositivos en diferentes pines, y haciendo las pruebas con los scripts que se han hecho anteriormente, editados para que los nuevos pines coincidan. Así, se asegura de que un componente funciona antes de añadir el resto, y de manera escalonada se irá montando el desarrollo.

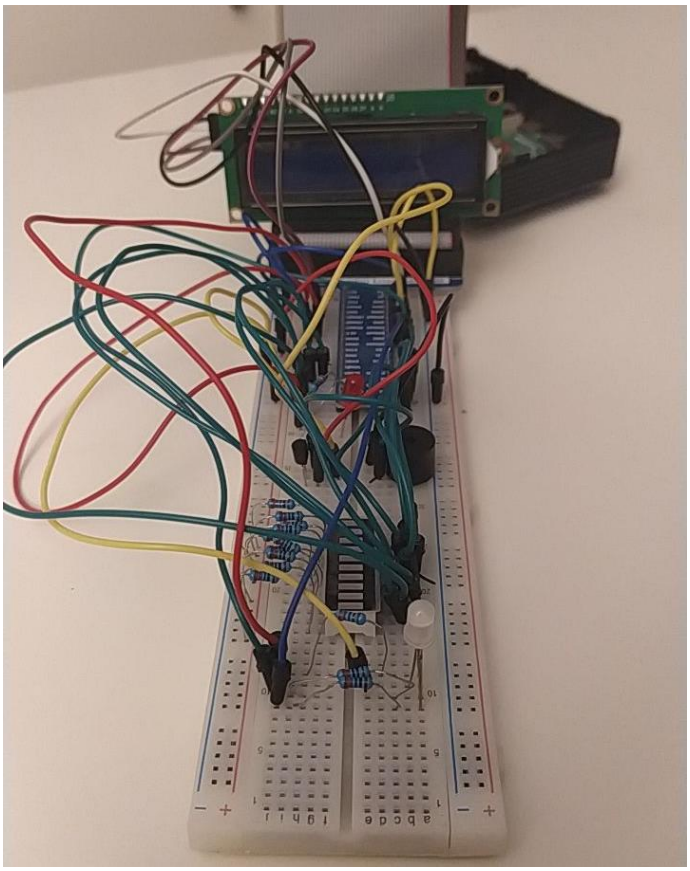


Figura 30 - Foto del montaje total

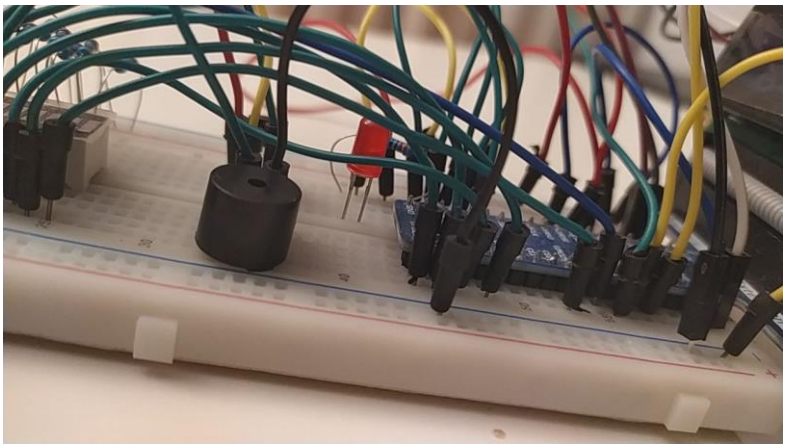


Figura 31 - Foto parcial del montaje 1

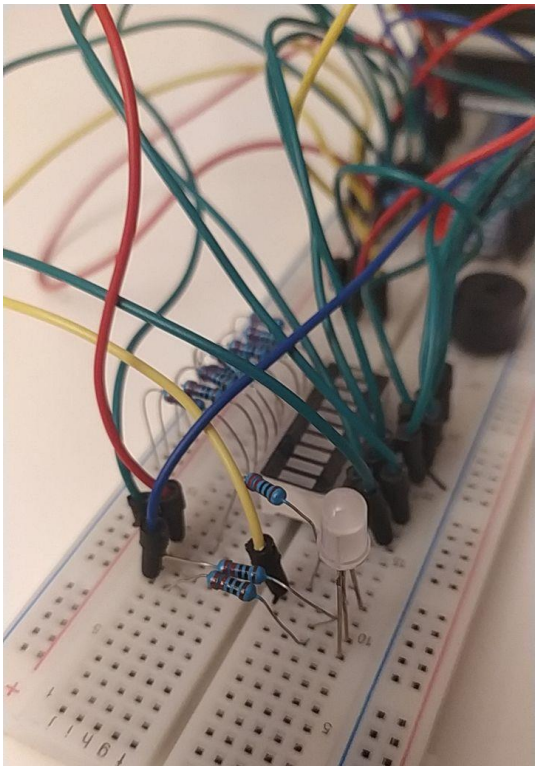


Figura 32 - Foto parcial del montaje 2

Una vez completado el montaje, se creó un nuevo script de Python, que fue el que se usó para probar el circuito final. En este script se hicieron funcionar todos los componentes a la vez, para comprobar que funcionaban correctamente. Finalmente, se subieron los archivos de pruebas y el test final al repositorio en GitHub, para que se quede guardado en caso de tener que arreglar un fallo futuro. En la siguiente fase del proyecto, se utilizará este script para juntarlo al código del servicio REST, acabando así con la parte del proyecto que utiliza la Raspberry Pi.

Servicio REST

En esta fase del proyecto, se incluirá el script que controla los casos de uso que han sido instalados en la breadboard conectada a la Raspberry Pi con el servicio REST que quedó a medias. El objetivo será concluir la parte de programación de Python, dejar terminado el desarrollo que incluye la Raspberry Pi, y hacer tests que puedan verificar el correcto funcionamiento de la implementación.

Para empezar, en el script del servicio REST, se incluyeron inicializaciones para cada uno de los componentes que se pueden controlar utilizando la librería de gpiozero. Estos componentes y sus clases se asocian en la siguiente tabla, junto a los pines que utilizan en la Raspberry Pi y que se usan en la inicialización de dichas clases.

Componente	Clase en software	Pines que utiliza
LED Rojo	LED	17
Barra de LEDs	LEDBoard	25, 12, 16, 20, 21, 5, 6, 13, 19, 26
RGB LED	RGBLED	22, 23, 24
Timbre activo	Buzzer	18
Display LCD	Adafruit_CharLCD	2, 3

Tabla 1 - Relación Hardware / Software

Para la conexión del display LCD, se utiliza como interfaz la clase especial PCF8574_GPIO, que permite controlar la pantalla utilizando menos cables. En este caso, con dos cables se puede gestionar toda la funcionalidad de la pantalla, de manera que primero se inicializa la clase PCF8574_GPIO, pasándole como argumento la dirección I2C del chip, y después se inicializa la clase Adafruit_CharLCD pasándole como argumento las líneas que se van a utilizar, y la interfaz I2C. Además, se configuran las opciones del display para que encienda la luz que tiene por detrás, y se le pasan las líneas y columnas que se van a utilizar, en este caso 16 columnas en dos líneas de caracteres.

Antes de elegir las rutas para los endpoints del servicio REST, se crea la función de conexión con la base de datos. Esta función utiliza pyodbc, la librería para conectar con bases de datos ODBC; los drivers FreeTDS, y se le pasa la base de datos que está alojada en Azure, con el usuario y la contraseña correspondientes. De esta manera, se tiene acceso a la base de datos en todas las rutas que lo requieran, sin necesidad de crear una nueva conexión cada vez.

El último paso a la hora de escribir el código del servicio REST es crear las funciones asociadas a cada ruta de acceso a la aplicación. El objetivo de las rutas de acceso es decretar un endpoint para que el bot pueda acceder a los distintos componentes de la Raspberry Pi, pudiendo seleccionar el estado que desee en cada uno de ellos. Las rutas de acceso deben ser descriptivas de acuerdo a la arquitectura REST, de manera que, sabiendo la ruta, se puede imaginar el resultado de acceder a ella con los parámetros adecuados.

A continuación, se van a enseñar cada una de las rutas de acceso de las que dispone el servicio REST, explicando los parámetros que se le tienen que pasar en la URI, así como el resultado que se obtiene en la Raspberry Pi y en los componentes de la breadboard cuando se accede a ellas. Antes de cada una de ellas, hay que escribir la dirección de la Raspberry Pi y el puerto por el que se accede; en este caso el 5000:

- **/**

  La ruta general, que sería escribir solo la IP y de la Raspberry Pi y añadir el puerto por el que se accede a la aplicación. Utiliza la función *hello* que hace que, al entrar por un navegador, devuelva un mensaje que lee "REST API – Made in FLASK". Se utiliza principalmente en pruebas de conexión, para comprobar que tenemos acceso a las demás rutas sin tener que escribirla entera.

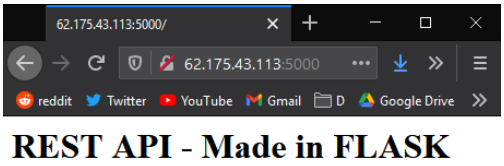


Figura 33 - Ruta general del servicio REST

- **/api/v1/red/**

  Esta es la ruta de acceso para controlar el diodo LED rojo. Utiliza la función *api_red*, que llama al método *toggle* de la clase LED. Este método alterna el estado en el que se encuentra el diodo, por lo que si está encendido se apaga, y si está apagado se enciende. Además, escribe el nuevo estado del LED en el display LCD, informando del cambio.

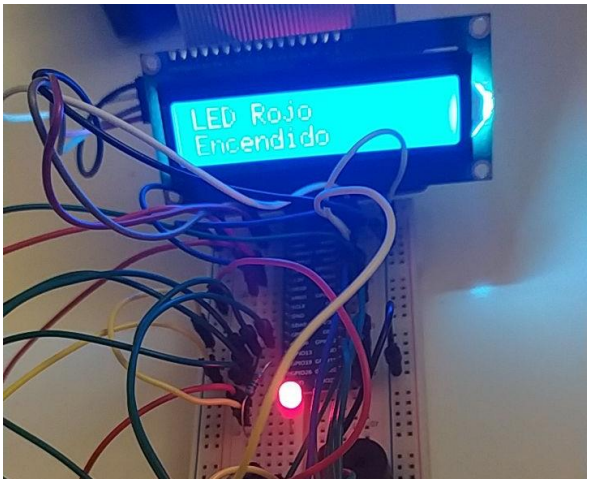


Figura 34 – Ruta del LED

- **/api/v1/buzzer/**
  La ruta de acceso que controla el timbre por defecto, sin pasarle información. Esta ruta utiliza la función *api_buzzer* con un valor por defecto de tres. Esta función hace sonar un timbre un número determinado de veces, en este caso tres, espaciado 500 milisegundos entre pitido y pitido. También deja un mensaje impreso en la pantalla LCD en el que anuncia el número de pitidos que han ocurrido.

- **/api/v1/buzzer/<int:num>/**
  La ruta de acceso alternativa para controlar el timbre, en el que se le pasa en la URI un número entero para indicar el número de veces que debe sonar el timbre. Utiliza también la función *api_buzzer*, pero esta vez utilizando un argumento para indicar las veces que debe sonar.

Figura 35 - Ruta del timbre

- **/api/v1/leds/**
  Esta ruta de acceso se encarga de controlar la barra de LEDs que están instalados en la breadboard. Cuando se accede a esta ruta, se llama a la función *api_leds_all*, que realiza un pequeño juego de luces en el que se alterna el estado uno a uno, con una diferencia de 200 milisegundos, de las luces LED de la barra. Después, muestra un mensaje en la pantalla del display LCD que indica los valores de los LED, con un 0 si están apagados, y con un 1 si están encendidos.

- **/api/v1/leds/<int:num>/**
  La ruta de acceso alternativa para controlar la barra de LEDs. En este caso, tiene un argumento en el que se le pasa un número entre 0 y 9. Esta ruta de acceso utiliza la función *api_leds*, que alterna el estado de un LED indicado por el

número que se ha pasado con el argumento. Además, como en la función anterior, se mostrará por la pantalla LCD los valores de cada uno de los LEDs de la barra.

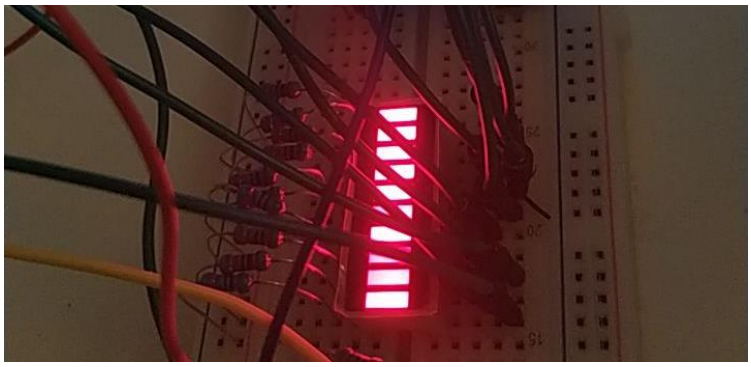


Figura 36 - Ruta del LED Board

- **/api/v1/rgb/test/**
  Esta ruta de acceso es una de las que controla el LED RGB. En este caso, utiliza la función *api_regb_test*, que realiza una prueba de este dispositivo. Para ello, cambia de color cada dos segundos en una secuencia que incluye rojo, verde, azul, rosa, violeta, amarillo y blanco. Después, en la pantalla LCD se muestran los valores de cada uno de los tres LEDs de colores, el rojo, el verde y el azul.

- **/api/v1/rgb/<string:color>/**
  Esta ruta le pasa una palabra a la función que se utiliza, *api_rgb_color.* Esta función actúa de tres maneras distintas dependiendo de lo que se le pase como color. Si es un color en inglés, el RGB LED utiliza la librería *Color* para mostrar ese color. Luego, hay dos palabras especiales que realizan un juego de colores: *police*, que enseña los colores rojo y el azul de manera ininterrumpida durante cuatro segundos simulando una sirena de policía para apagarse después; y *rainbow*, que muestra los siete colores del arcoíris y luego se apaga. Independientemente del juego de colores que realice, después se muestran en la pantalla LCD los valores de los tres LED, igual que en el caso anterior.

- **/api/v1/rgb/<int:r>/<int:g>/<int:b>/**
  La última de las rutas de acceso que controla al LED RGB. En este caso, se utilizan 3 valores, uno para el LED rojo, otro para el LED verde, y otro para el LED azul. La función *api_rgb* coge estos valores, y los asigna a sus respectivos LEDs. Igual que en las dos funciones anteriores, al acabar se muestran dichos valores por la pantalla LCD.

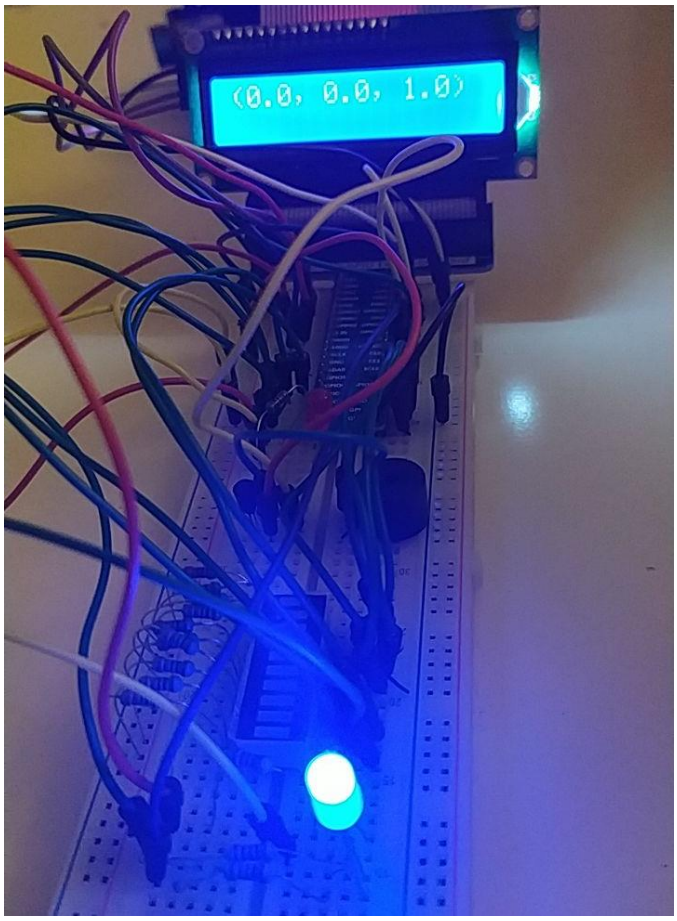


Figura 37 - Ruta del LED RGB

- **/api/v1/lcd/**
  La ruta de acceso para imprimir mensajes en la pantalla LCD. Al acceder, se utiliza la función *api_lcd*, que recoge el parámetro '*msg*' y el string asociado a él, y lo muestra en la pantalla LCD, utilizando un algoritmo que impide que una palabra se corte en la primera línea. En caso de no usar el parámetro, salta un error del tipo *AtributeError*.

Figura 38 - Ruta del LCD Display

- **/api/v1/students**
  Ruta de acceso que conecta con la base de datos de *students*. Para ello, utiliza la función *api_students,* que dado un DNI de un alumno que se encuentre en la base de datos, devuelve en forma de JSON su nombre y apellidos, además de mostrarlo por la pantalla LCD. Esta ruta se utiliza como ejemplo de uso que permite la conexión entre una base de datos, y un dispositivo hardware.

Figura 39 - Ruta de students

Una vez listas las rutas de acceso, se probaron individualmente para corregir errores de funcionamiento y para comprobar que cada una de ellas funcionaba dando el resultado esperado. Para terminar el desarrollo del servicio REST, se decidió buscar un nombre de dominio para evitar el tener que introducir la dirección IP cada vez que se use la aplicación con el navegador web. Una vez adquirido un dominio propio y enlazado a la dirección IP que usamos, se utilizó el redireccionamiento de puertos para que las peticiones al puerto 5000 de la dirección IP que se utiliza redirijan a la dirección IP de la Raspberry PI. Finalmente, se deja el dispositivo conectado a la red para que, cuando se desarrolle el bot, se pueda probar su correcto funcionamiento.

Bot

En esta última fase del proyecto, se va diseñar, desarrollar, desplegar y testear la aplicación bot que se va a construir utilizando el Microsoft Bot Framework. El objetivo final es tener dos bots en plataformas digitales muy conocidas: Telegram y Microsoft Teams. Una vez implementados estos dos bots, se dará por concluido el trabajo, y se procederán a las conclusiones finales.

Diseño

Lo primero que se hace a la hora de desarrollar un bot es elaborar un buen diseño, teniendo claro el objetivo que debe cumplir y a quién está dirigido, pero la principal prioridad que hay que tener es la de conseguir una buena experiencia de usuario. Para, ello se deben tener algunas consideraciones a tener en cuenta cuando se diseña un bot [45]:

- ¿Consigue el bot resolver fácilmente el problema del usuario utilizando para ello el menor número de pasos?
- ¿Consigue el bot resolver el problema del usuario de una manera mejor, más rápida, o más eficiente que alguna de las alternativas?
- ¿El bot está disponible en plataformas que el usuario vaya a utilizar?
- ¿El usuario sabe de manera natural qué hacer mientras utiliza el bot?

Otro de los puntos clave debe ser la primera interacción entre el usuario y el bot. Esto es extremadamente importante, pues esta interacción debe aportar instrucciones al usuario sobre cómo utilizar el bot de manera intuitiva. En este caso, se planeó el utilizar un mensaje de bienvenida que salude al usuario la primera vez que utilice el bot, indicándole el comando que debe utilizar para conocer el resto de comandos que tiene a su disposición para controlar el sistema domótico [46].

Para diseñar el diálogo con el bot, se barajaron diferentes opciones. Entre estas opciones se encuentra el hacer que el bot entienda el lenguaje natural escrito del usuario, utilizando para ello LUIS (Language Understanding) de Microsoft, una poderosa herramienta de procesado de lenguaje natural que utiliza aprendizaje máquina. De esta manera, el bot puede entender una conversación normal con el usuario y contestar como si él mismo fuera humano. Aunque esta idea es atractiva, y de hecho se recomendará más adelante en el apartado de líneas futuras, se decide no emplear esta tecnología, ya que habría que entrenar al bot para que entienda las interacciones necesarias en cada caso de uso.

Por lo tanto, el bot informará de una serie de comandos que el usuario puede utilizar, así como una breve explicación de cada uno, para que este pueda utilizarlos correctamente. En caso de no entender al usuario, o de introducir un comando de manera incorrecta, el bot avisará con un mensaje de advertencia. Tras la correcta introducción de un comando, el bot responderá con un mensaje para que el usuario sepa que la operación ha tenido éxito.

A continuación, se explican con detalle los pasos que se han seguido para la elaboración del bot.

Desarrollo

Una vez acabada la parte de diseño y toma de decisiones para el bot, se empieza con el desarrollo práctico del mismo. Este desarrollo consiste en escribir el código del bot utilizando el Microsoft Bot Framework, de manera que cumpla con el diseño que ha sido planeado anteriormente. El objetivo es tener un bot que pueda comunicarse con la Raspberry Pi, utilizando para ello el servicio REST que ha sido implementado sobre ella.

Para empezar con la creación del bot, se abre un nuevo proyecto utilizando Virtual Studio y se instalan las plantillas del Bot Framework SDK. Estas plantillas son una colección completa de distintos tipos de bots que están disponibles para Virtual Studio, lo que permite comenzar un nuevo proyecto de bot de una forma rápida. Se selecciona una de las plantillas, en este caso la de Echo Bot, de entre los AI Bots en los tipos de proyectos seleccionables en el desplegable de Virtual Studio. Gracias a esta plantilla, ahora el proyecto contiene todo el código necesario para crear un bot funcional, sin necesidad de añadir nada más.

Si se ejecuta este bot sin hacer ningún cambio, Virtual Studio primero compila la aplicación, luego lo despliega en localhost y finalmente abre en el navegador la página de la aplicación por defecto, la que se encuentra en el archivo *default.htm*. Se puede comprobar el funcionamiento utilizando el Bot Framework Emulator, pero se va a utilizar la plataforma de chat con el bot que ofrece el portal de Azure.

Al utilizar el chat de Azure, se aprecia el funcionamiento del Echo Bot que se ha utilizado para la plantilla del bot. En este caso, el bot manda un mensaje cuando un nuevo usuario se une a la conversación, y espera a recibir algún mensaje. Una vez recibe un mensaje en el chat, devuelve este mismo mensaje al usuario, actuando de eco. Este bot, aunque sencillo, permite diferenciar las actividades que puede realizar y entender el código del bot que se ha proporcionado dentro de la plantilla, así como el resto de ficheros del proyecto.

El siguiente paso es utilizar la plantilla para desarrollar el diseño que se había decidido en la fase de planificación del bot. Para ello, se van a editar las funciones que venían con la plantilla, aparte de añadir otras que hagan de puente entre el bot y el servicio REST.

Estas funciones son:

- **SendRaspberryActivity**
  Esta función se encarga de conectar la aplicación con el servicio REST. Como argumentos utiliza dos *strings*: uno para representar el endpoint al que debe acceder, y otro para indicarle al bot lo que debe decir cuando se accede a dicho endpoint. Cuando se llama a esta función, se crea un cliente HTTP que realiza una petición al endpoint que viene en el argumento, y se queda esperando una respuesta. Si el código que devuelve es de éxito, el bot escribirá por el chat la frase indicada por el *string* del argumento utilizando el método

SendActivityAsync. En caso contrario, envía un mensaje de error y manda los comandos disponibles.

- **OnMembersAddedAsync**
  El objetivo de esta función es el de enviar un mensaje cuando un usuario se conecta al bot. En este caso, cuando un usuario interacciona por primera vez, el bot le saludará, y después le mandará el mensaje con los comandos disponibles. Este mensaje ayudará al usuario a saber de lo que es capaz el bot sin necesidad de tener conocimientos previos, respetando así uno de los requerimientos del diseño: que el bot sea intuitivo de usar.

- **OnMessageActivityAsync**
  La función principal del bot, que se encarga de "escuchar" al usuario cuando interacciona con él. Su funcionamiento es el siguiente: una vez llega un mensaje, pasa un algoritmo por el texto que, utilizando REGEX (Regular Expressions o Expresiones Regulares), confirma si el usuario ha utilizado un comando, y de ser así, si ese comando tiene dos partes. El objetivo de este algoritmo es diferenciar entre lo que es el comando en sí, y las opciones que se le quieren pasar al comando. Una vez que se hace esta distinción, se comprueba si está utilizando alguno de los comandos que están programados. En caso afirmativo, utiliza SendRaspberryActivity con los argumentos apropiados, construyendo el URI al que tiene que acceder.

Los comandos que entiende el bot son los siguientes:

- **Hola**
  Si el usuario saluda, el bot le saluda de vuelta.

- **/help**
  Comando para lanzar el mensaje de ayuda, en el que se incluyen todos los comandos que entiende el bot.

- **/led**
  Comando que se encarga de alternar la luz del diodo LED rojo.

- **/leds**
  Comando que se encarga de activar el juego de luces que se produce en la barra de LEDs.

- **/police**
  Comando que simula una sirena de policía, alternando el color rojo y azul en el LED RGB.

- **/test**
  Comando que lanza la prueba del LED RGB, cambiando varias veces de color.

- **/student**
  Comando que devuelve, si lo hay, el nombre completo de un estudiante en la base de datos.

- **/buzzer**
  Si el usuario introduce este comando sin parámetros, se activa el timbre tres veces. De lo contrario, si junto al comando introduce un número, se activa el timbre ese número de veces.

- **/rgb**
  Si el usuario introduce este comando sin parámetros, se enciende el LED RGB y se pone de color rojo. De lo contrario, si junto al comando introduce un color, se pone de ese color.

- **/msg**
  Este comando muestra un mensaje en la pantalla del LCD. El parámetro que viene con el comando es el mensaje que pondrá en la pantalla, escrito de la misma manera que lo ha hecho el usuario.

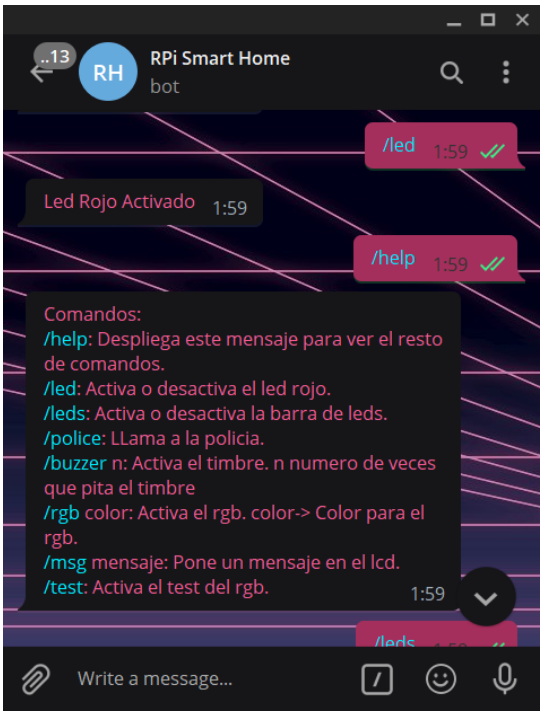


Figura 40 - Ejemplo de uso del bot

Los comandos se han hecho de esta forma para parecerse a los bots que ya se encuentran actualmente en Telegram, de manera que el uso de este bot en la plataforma no difiera del resto de opciones disponibles.

Durante el desarrollo del bot, se utilizó GitHub de dos maneras distintas. La primera, para llevar un control de versiones de la aplicación, en el que se puede guardar el código después de cada edición para que no se pierda nada, o para retomar una versión anterior si se comete un fallo. La segunda, para desplegar el código a Azure cada vez que se hace una petición al repositorio. En el portal de Azure, se encuentra una opción para realizar un despliegue continuo de la aplicación, es decir, una manera de no tener que estar subiendo el código a la plataforma cada vez que se necesite probar el bot. Así, utilizando el centro de despliegue que ofrece Microsoft Azure, y enlazándolo al repositorio de GitHub donde está almacenado el código, el bot se actualiza cada vez que subimos la aplicación al repositorio.

En la siguiente fase del desarrollo, se explicarán los métodos utilizados para testear el bot y el funcionamiento de los medios utilizados para ello.

Testing

En las directrices de Microsoft para la creación de bots, se dicta el testeo o prueba de la aplicación como el siguiente paso a seguir tras su construcción. Esto quiere decir, comprobar que la aplicación hace lo que se le pide de manera correcta y evitar que la conversación entre en un estado inesperado. Para ello, Microsoft Azure ofrece varias opciones, de las cuales nos centraremos en dos: el Web Chat disponible en el portal de Azure, y el Bot Framework Emulator.

El Web Chat con el bot es un chat que permite una conversación directa con el bot desplegado en la plataforma de Azure. Este chat, que se encuentra en la pestaña de administración del bot, permite probar directamente los cambios realizados sin necesidad de ejecutar el bot de manera local. Aunque esta manera es muy cómoda, hay que esperar un tiempo a que se actualice el bot tras la subida del código al repositorio, por lo que es algo lenta. Por este motivo, la mayoría de las pruebas en el proyecto se realizaron en la aplicación de escritorio Bot Framework Emulator.

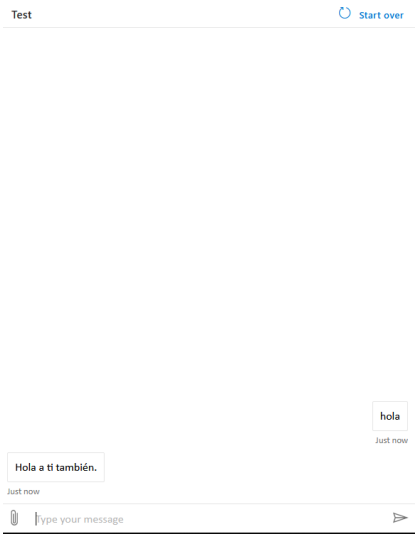


Figura 41 - Debug con Web Chat

Para testear y debugear el bot utilizando Bot Framework Emulator, primero hay que hacer que el bot se ejecute de manera local. Para ello, desde Virtual Studio, se puede compilar y ejecutar la aplicación en un puerto específico, en este caso el 3978. Una vez ha compilado correctamente y se está ejecutando en una ventana del terminal, se abre el programa y se introducen los datos de conexión con el bot, una URI en la que se le indica el puerto donde se está ejecutando. Después, tras abrirse la conexión con el bot, se le puede mandar un mensaje y debería responder. Junto a la ventana de chat, se incluye un log con un resumen de la conversación y de los códigos de éxito y un inspector de mensajes, que permite obtener una versión más detallada de cada mensaje, tanto por parte del bot como del usuario, en formato JSON. Entre esta información se incluyen metadatos como el ID del canal, el tipo de actividad, el ID de la conversación, el mensaje de texto y el endpoint. De esta forma, se puede testear el correcto funcionamiento del bot desarrollado en tan solo unos segundos, sin necesidad de subir el código al repositorio de GitHub para que lo despliegue automáticamente Azure [47].

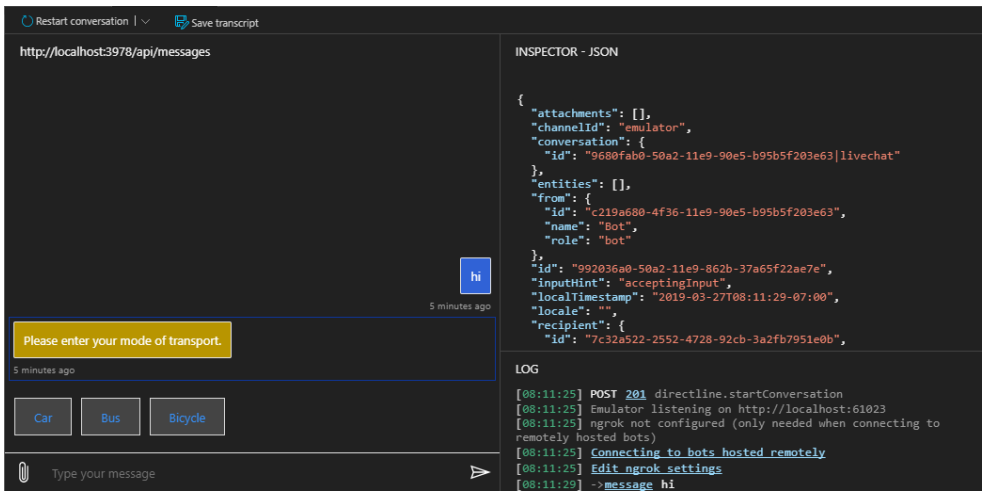


Figura 42 - Ejemplo de uso de Bot Framework Emulator

Una vez comprobado el bot, se pasa a la última fase del desarrollo, que consiste en publicar el bot en dos plataformas digitales: Telegram y Microsoft Teams.

Publicación

Para acabar el desarrollo del proyecto, hay que hacer que el bot pueda ser utilizado de manera remota por un usuario, utilizando para ello las dos plataformas objetivo del proyecto, Telegram y Microsoft Teams.

Telegram es una aplicación multiplataforma de mensajería instantánea, VoIP (Voice Over IP) y llamada de vídeo. Todas las aplicaciones de Telegram son de código abierto, y entre sus características se incluyen las llamadas cifradas extremo a extremo, vista instantánea de contenido multimedia, mensajes en la nube, juegos, y bots. Actualmente, existen más de 400 millones usuarios activos en Telegram [48].

Microsoft Teams es una plataforma de comunicación desarrollada por la empresa Microsoft, como parte de la familia de productos Microsoft 365. Su función principal es la de ofrecer un espacio de trabajo en el que chatear y realizar conferencias, almacenar archivos y administrarlos de manera colaborativa. También incluye aplicaciones como bots para facilitar tareas a los usuarios [49].

El objetivo es que el bot que ha sido desarrollado pueda ser utilizado mediante estas dos plataformas. Para ello, se utilizan los canales que ofrece el portal de Azure, en la pestaña de administración del bot. Un canal es una conexión entre una aplicación de comunicaciones y un bot. El Bot Framework permite desarrollar un bot sin importar el canal, normalizando los mensajes que el bot envía, convirtiendo los mensajes del esquema de Bot Framework al esquema del canal y solventando problemas de soporte si el canal no acepta alguna funcionalidad del esquema.

Una vez dentro de la pestaña, aparecen los distintos canales a los que se puede conectar nuestro bot. Entre estos canales se encuentran Alexa, Cortana, Direct Line, Office 365 Email, Facebook, Kik, LINE, Microsoft Teams, Skype, Slack, Telegram, Twilio, WeChat, Web chat y Webex. Se seleccionan los dos canales que nos interesan, el de Telegram y el de Microsoft Teams. Como cada uno de estos canales una manera diferente de crear un bot, hay que hacerlo por separado.

En Telegram se utiliza un bot llamado *BotFather* para incluir otro bot dentro de la aplicación. Una vez abierta una conversación con este bot, guía con una serie de comandos en los que se añade un nombre para el bot. Cuando lo crea, envía otro mensaje con el token de acceso necesario, que se copia y se pega en la sección de canales correspondiente a Telegram. Tras esto, el bot está completamente configurado y se puede probar que la aplicación funciona [50].

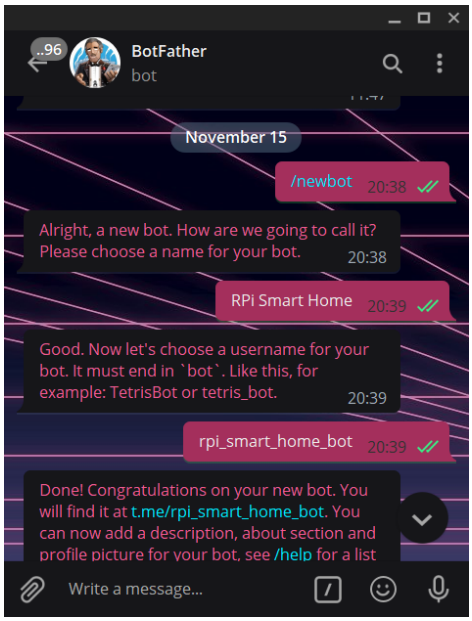


Figura 43 - Creación del bot en Telegram 1

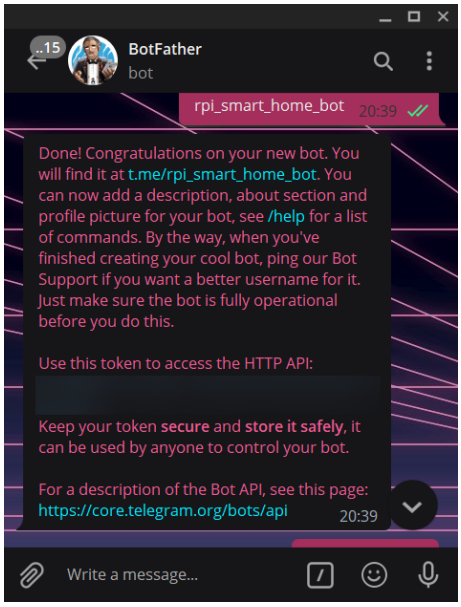


Figura 44 - Creación del bot en Telegram 2

Para conectar el bot con Microsoft Teams, hay que añadirlo a través de una app y un paquete de subida. Este paquete debe contener la información que describe la experiencia con el bot, en forma del manifestó de la aplicación dentro de un archivo *.zip*. Una vez creado el paquete, se añade a Teams, donde se tendrá acceso a la aplicación y al bot [51]. Todo este proceso se puede realizar de manera muy sencilla utilizando una app que se encuentra en Microsoft Teams llamada App Studio. Esta app te guía a través de los pasos que hay que seguir para añadir un bot a una app, así como realizar otras funciones como la del creación del manifestó o poder descargar el zip con el paquete. Si se requiere, se puede publicar el bot para que otros usuarios puedan verlo, pero se deja para el apartado de líneas futuras [52].

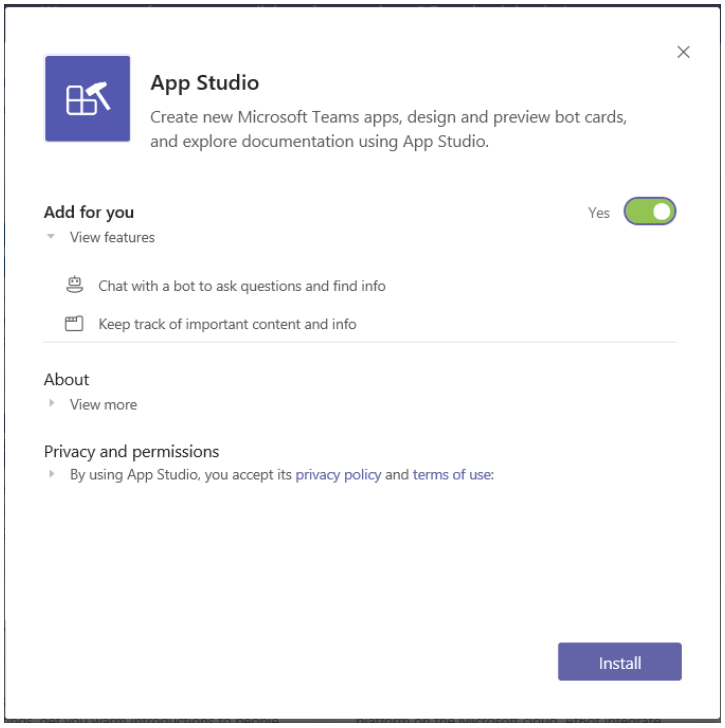


Figura 45 - App Studio

Con esto quedaría terminado el desarrollo completo de la aplicación, con un sistema domótico mediante Raspberry Pi controlado por un bot. En el siguiente apartado, se sacan las conclusiones del trabajo y se explican algunas líneas futuras de investigación recomendadas.

Conclusiones y líneas futuras

En este apartado se explican las conclusiones sacadas del desarrollo de este proyecto y se analiza si se han conseguido los objetivos planteados. Además, se explican unas líneas futuras de investigación, basadas en lo aprendido durante el desarrollo de la aplicación.

Conclusiones

Durante la realización de este proyecto, se ha investigado sobre la historia detrás de los primeros bots y los primeros sistemas domóticos, así como de los ingenieros y empresas que comenzaron a desarrollarlos. También se ha hablado sobre en qué estado se encuentran actualmente los bots y los distintos tipos que se pueden encontrar ahora mismo, tanto los que ayudan como los que dañan. Se ha aprendido sobre el funcionamiento de los bots, las preocupaciones que tienen los usuarios actualmente, y lo que se puede esperar de ellos durante los próximos años.

En cuanto a tecnologías, se ha profundizado la experiencia que se tenía sobre Python y C#, dos de los lenguajes de programación más importantes y más usados en la actualidad. Se han comparado los frameworks web de Python, y se tomó la decisión de utilizar Flask, sobre el que se ha acabado construyendo el servicio REST. Se han adquirido conocimientos en las librerías que utiliza este proyecto, como *gpiozero* y Bot Framework SDK, las cuales han ayudado mucho a hacer que el código tanto del servicio REST como del bot sea más claro, limpio y legible. También se ha aprendido a utilizar los dos IDEs con los que se ha programado, Virtual Studio y PyCharm, aprovechando las características de cada uno de ellos.

Otro aspecto importante del desarrollo que ha sido fundamental a la hora de hacer el trabajo ha sido el uso del control de versiones que ofrece GitHub. Esto ha permitido en varias ocasiones regresar a un estado anterior del proyecto, para solucionar errores que aparecían. El uso de la aplicación de escritorio GitHub Desktop también ha facilitado el subir los archivos al repositorio en pocos segundos, después de cada sesión de trabajo.

En cuanto a los objetivos, se prepararon un conjunto de casos de uso de dispositivos que se podían conectar a la Raspberry Pi. Estos casos de uso fueron diseñados para ofrecer variedad a la cantidad de cosas que puede hacer el sistema domótico, y el resultado obtenido con ellos ha sido satisfactorio, sin llegar a resultar en un problema por espacio en la breadboard que había disponible. El desarrollo del servicio REST y el servidor web también ha salido como se esperaba, consiguiendo una aplicación rápida que va montada sobre la Raspberry Pi.

Se ha conseguido construir un bot que puede utilizarse a través de varios canales de comunicación. Aunque se usa Telegram y Microsoft Teams, al usar Microsoft Bot Framework, se puede llevar el bot a cualquier otro canal soportado, con solo un retoque

mínimo si el canal no admitiera alguna característica del bot. La ventaja de implementar el servicio REST para que funcione en conjunto con el bot, es que el script de activación de los componentes del caso de uso es independiente de la plataforma en la que se encuentra el bot. Por lo tanto, se podría crear de manera efectiva y casi sin esfuerzo un bot adaptado a la necesidad de cada usuario, atendiendo a la plataforma que más utiliza.

Finalmente, este proyecto ha servido para mostrar la compatibilidad entre lenguajes de programación distintos, cada uno con sus librerías y paquetes especializados en desarrollar una tarea concreta. Tener conocimiento previo de ambos lenguajes ha ayudado con la velocidad de implementación de cada parte del proyecto, y ha bajado el nivel de entrada necesario para entender y editar el código que se necesitaba en cada uno de los frameworks que se han utilizado.

Líneas futuras

En este último apartado, se explican rutas alternativas y futuras líneas de investigación y desarrollo que se podrían tomar en relación a este proyecto.

La primera sería aprovechar al máximo las características de Microsoft Bot Framework y el resto de servicios de Azure, integrándolos al desarrollo. En este caso, se podría haber utilizado LUIS, para permitir que el bot entienda el lenguaje natural a la hora de recibir los comandos. También se puede utilizar el servicio QnA Maker para añadir soporte de preguntas y respuestas al bot.

Una alternativa que se podía haber tomado en vez de utilizar solo uWSGI para el servidor web, es la de usarlo junto a nginx. Nginx es un servidor web que puede usarse como un proxy inverso, lo que permite adquirir recursos de otros servidores en nombre del cliente. Usar nginx hace que el servidor sea más robusto y más seguro.

Otro punto que se puede investigar es el de la seguridad en torno al servidor web. Para acceder al servicio REST en este proyecto, se ha desplegado sobre el puerto 5000, cuando se podría hacer sobre el puerto 80. Aunque en un principio se hizo así, al tener la Raspberry Pi conectada a este puerto, llegaban peticiones intentando encontrar vulnerabilidades, por lo que se decidió dejarlo en el 5000.

Una línea futura obvia sería la de implementar el bot en los otros canales que hay disponibles en el portal de Azure, como Skype, Kik, Facebook o Alexa.

Finalmente, se pueden alternar o aumentar el conjunto de casos de uso que se han instalado en la breadboard. Una de las alternativas que se probaron, aunque fue descartada, era la de hacer que suene música por un altavoz conectado a la Raspberry Pi con un comando del bot. Podría ser una alternativa la de hacer una jukebox, o un sistema de sonido controlado por bot.

Bibliografía

Todos los recursos en línea han sido consultados en la fecha 04/12/2020.

[1] "Howstuffworks," [Online]. Available: https://science.howstuffworks.com/innovation/repurposed-inventions/history-of-remote-control.htm. [Accessed 2020].

[2] «Wikipedia - Leonardo Torres Quevedo,» [En línea]. Available: https://es.wikipedia.org/wiki/Leonardo_Torres_Quevedo.

[3] «bccresearch,» [En línea]. Available: https://blog.bccresearch.com/the-evolution-of-smart-home-technology.

[4] «hubspot,» [En línea]. Available: https://blog.hubspot.com/marketing/where-do-bots-come-from.

[5] «zeusintegrated,» [En línea]. Available: https://zeusintegrated.com/blog/item/a-brief-history-of-smart-home-automation.

[6] «Wikipedia - X10,» [En línea]. Available: https://en.wikipedia.org/wiki/X10_(industry_standard).

[7] «iotevolutionworld,» [En línea]. Available: https://www.iotevolutionworld.com/m2m/articles/376816-history-smart-homes.htm.

[8] «conversocial,» [En línea]. Available: https://www.conversocial.com/blog/automation-evolution-the-brief-history-of-chatbots.

[9] «wikipedia - Internet bot,» [En línea]. Available: https://en.wikipedia.org/wiki/Internet_bot.

[10] «cloudflare,» [En línea]. Available: https://www.cloudflare.com/es-es/learning/bots/what-is-a-bot/.

[11] «techtarget,» [En línea]. Available: https://whatis.techtarget.com/definition/bot-robot.

[12] «deconstructeam.itch,» [En línea]. Available: https://deconstructeam.itch.io/interview-with-the-whisperer.

[13] «discover,» [En línea]. Available: https://discover.bot/bot-talk/machine-learning-chatbot/.

[14] «wikipedia - Python,» [En línea]. Available: https://es.wikipedia.org/wiki/Python.

[15] «python,» [En línea]. Available: https://www.python.org/doc/essays/blurb/.

[16] «wikipedia - C Sharp,» [En línea]. Available: https://en.wikipedia.org/wiki/C_Sharp_(programming_language).

[17] «mysql,» [En línea]. Available: https://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html.

[18] «github - gpiozero,» [En línea]. Available: https://github.com/gpiozero/gpiozero.

[19] «wikipedia - flask,» [En línea]. Available: https://en.wikipedia.org/wiki/Flask_(web_framework).

[20] «botframework,» [En línea]. Available: https://dev.botframework.com/.

[21] «github - Adafruit_Python_CharLCD,» [En línea]. Available: https://github.com/adafruit/Adafruit_Python_CharLCD.

[22] «restfulapi,» [En línea]. Available: https://restfulapi.net/.

[23] «azure.microsoft,» [En línea]. Available: https://azure.microsoft.com/es-es/features/azure-portal/.

[24] «visualstudio,» [En línea]. Available: https://visualstudio.microsoft.com/es/vs/.

[25] «jetbrains,» [En línea]. Available: https://www.jetbrains.com/pycharm/features/.

[26] «github,» [En línea]. Available: https://guides.github.com/activities/hello-world/.

[27] «github - BotFramework-Emulator,» [En línea]. Available: https://github.com/Microsoft/BotFramework-Emulator.

[28] «winscp,» [En línea]. Available: https://winscp.net/eng/docs/introduction.

[29] «wikipedia - SSH,» [En línea]. Available: https://en.wikipedia.org/wiki/SSH_(Secure_Shell).

[30] «postman,» [En línea]. Available: https://www.postman.com/api-documentation-tool/.

[31] «visualstudio - downloads,» [En línea]. Available: https://visualstudio.microsoft.com/es/downloads/.

[32] «jetbrains - students,» [En línea]. Available: https://www.jetbrains.com/community/education/#students.

[33] «azure.microsoft - students,» [En línea]. Available: https://azure.microsoft.com/en-us/free/students/.

[34] «raspberrypi - model,» [En línea]. Available: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/.

[35] «raspberrypi raspbian,» [En línea]. Available: https://www.raspberrypi.org/documentation/raspbian/.

[36] «raspberrypi installing raspbian,» [En línea]. Available: https://www.raspberrypi.org/documentation/installation/installing-images/README.md.

[37] «python - wf,» [En línea]. Available: https://wiki.python.org/moin/WebFrameworks.

[38] «uwsgi-docs,» [En línea]. Available: https://uwsgi-docs.readthedocs.io/en/latest/.

[39] «digitalocean,» [En línea]. Available: https://www.digitalocean.com/community/tutorials/how-to-serve-flask-applications-with-uswgi-and-nginx-on-ubuntu-18-04.

[40] «pyodbc,» [En línea]. Available: https://github.com/mkleehammer/pyodbc/wiki.

[41] «freetds,» [En línea]. Available: https://www.freetds.org/.

[42] «raspberrypi gpio,» [En línea]. Available: https://www.raspberrypi.org/documentation/usage/gpio/README.md.

[43] «gpiozero.readthedocs recipes,» [En línea]. Available: https://gpiozero.readthedocs.io/en/stable/recipes.html.

[44] «freenove,» [En línea]. Available: http://www.freenove.com/tutorial.html.

[45] «bot-service-design-principles,» [En línea]. Available: https://docs.microsoft.com/en-us/azure/bot-service/bot-service-design-principles?view=azure-bot-service-4.0.

[46] «bot-service-design-first-interaction,» [En línea]. Available: https://docs.microsoft.com/en-us/azure/bot-service/bot-service-design-first-interaction.

[47] «bot-service-debug-emulator,» [En línea]. Available: https://docs.microsoft.com/en-us/azure/bot-service/bot-service-debug-emulator.

[48] «telegram,» [En línea]. Available: https://telegram.org/apps.

[49] «theverge,» [En línea]. Available: https://www.theverge.com/2016/11/2/13497992/microsoft-teams-slack-competitor-features.

[50] «bot-service-channel-connect-telegram,» [En línea]. Available: https://docs.microsoft.com/en-us/azure/bot-service/bot-service-channel-connect-telegram.

[51] «channel-connect-teams,» [En línea]. Available: https://docs.microsoft.com/en-us/azure/bot-service/channel-connect-teams.

[52] «apps-upload,» [En línea]. Available: https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/deploy-and-publish/apps-upload.

[53] S. Monk, Raspberry Pi cookbook.: Software and hardware problems and solutions, O'Reilly Media, Inc, 2016.

[54] M. Lutz, Learning Python: Powerful Object-Oriented Programming, O'Reilly Media, Inc, 2013.

[55] S. M. &. W. H. E. Tahaghoghi, Learning MySQL: Get a Handle on Your Data, O'Reilly Media, Inc, 2006.

[56] I. Griffiths, Programming C# 8.0, O'Reilly , 2019.

[57] R. S. Miles, The C# Programming Yellow Book, 2015.

Anexos

Instalación de Flask

Creación del entorno:

```
$ python3 -m venv venv
```

Activación:

```
$ . venv/bin/activate
```

Instalación de Flask a través de pip:

```
$ pip install Flask
```