UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Ejecución automática de scripts en sistemas de área local mediante la integración por voz con un dispositivo Alexa y manejo de peticiones con un servidor alojado en una Raspberry Pi

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

AUTOR/A: Martínez Melero, Álvaro

Tutor/a: Vidal Catalá, José Ramón

CURSO ACADÉMICO: 2022/2023

Contenido

Capítulo 1. Introducción

En la actualidad el mundo se rige por la tecnología. Sin duda alguna, estamos inmersos en un mundo interconectado y digital que nos conduce en una clara dirección: el progreso. Pero cuando hablamos de avance tecnológico, ¿A qué nos referimos exactamente?, ¿Cuál es la motivación que nos empuja a encontrar lo que buscamos?

Evidentemente, en la mayor parte de las ocasiones nos mueve la necesidad, y en la sociedad de lo instantáneo, la accesibilidad y la inmediatez se han convertido incuestionablemente, en necesidades.

Hemos dejado atrás tres revoluciones industriales que han servido de trampolín al nacimiento de la industria 4.0, uno de cuyos pilares fundamentales es la automatización. Vivimos en una era digital. Ordenadores, robots e inteligencia artificial, forman parte de nuestro día a día.

La sociedad del siglo XXI es exigente, demanda servicios de internet y telecomunicaciones y la tecnología debe dar respuesta de la forma más personalizada posible a dichas exigencias. Vivimos entre dispositivos conectados a internet, lo que llamamos el internet de las cosas (IoT), que nos permite automatizar muchos procesos diarios y que nos facilita aquellas rutinas diarias, que de otro modo nos serían mucho más costosas. Todo ello sirve de estímulo a la industria tecnológica, que busca responder con eficiencia a la demanda, cada vez más individualizada de los consumidores. Con un solo smartphone, podemos acceder a funcionalidades que de otro modo los dispositivos convencionales jamás podrían brindarnos.

Sin ir más lejos, podemos programar una lavadora desde un móvil indicándole el programa que debe de seguir e incluso decirle a ésta que nos avise una vez haya terminado, al igual que cosas más sencillas como encender una bombilla o apagar un altavoz.

Una herramienta muy importante que nos brinda esta accesibilidad a la hora de la automatización de dispositivos IoT, es la citada con anterioridad: el control por voz. Gracias al boom de la Inteligencia Artificial, grandes empresas como Amazon o Google han lanzado al mercado sus propios dispositivos controlados por voz que nos permiten cómodamente utilizar aquellos aparatos inteligentes del hogar.

Observando todas estas premisas, es lógica la aparición de asistentes de control por voz. Herramientas que, en la última década, están proporcionando un servicio cada vez mayor a la población. Es en esta línea, en la que la industria de estos dispositivos está creciendo, las compañías no deben olvidar que estas facilidades y modos de mejorar nuestras vidas, no solo son exigencias que deban concederse a unos pocos, sino que la verdadera revolución 4.0, reside en facilitar la accesibilidad del producto al mayor número de personas posible.

Dicha accesibilidad puede basarse en principios de coste, aspecto que las empresas ya han abordado, puesto que la tendencia actual nos indica el decrecimiento del precio de los dispositivos.

Por tanto, el objeto de este Trabajo de Fin de Grado plantea una solución que pueda dar respuestas de accesibilidad y control sencillo de dispositivos, en este caso un ordenador, utilizando un dispositivo de voz.

Capítulo 2. Objetivos

Este proyecto de Fin de Grado tiene como objetivo el diseño de una solución accesible y cómoda para la ejecución de scripts en máquinas de una red de área local mediante comandos de voz procesados por el asistente virtual Alexa. Siendo este el objetivo principal, podemos definir una serie de hitos secundarios a lograr durante el desarrollo:

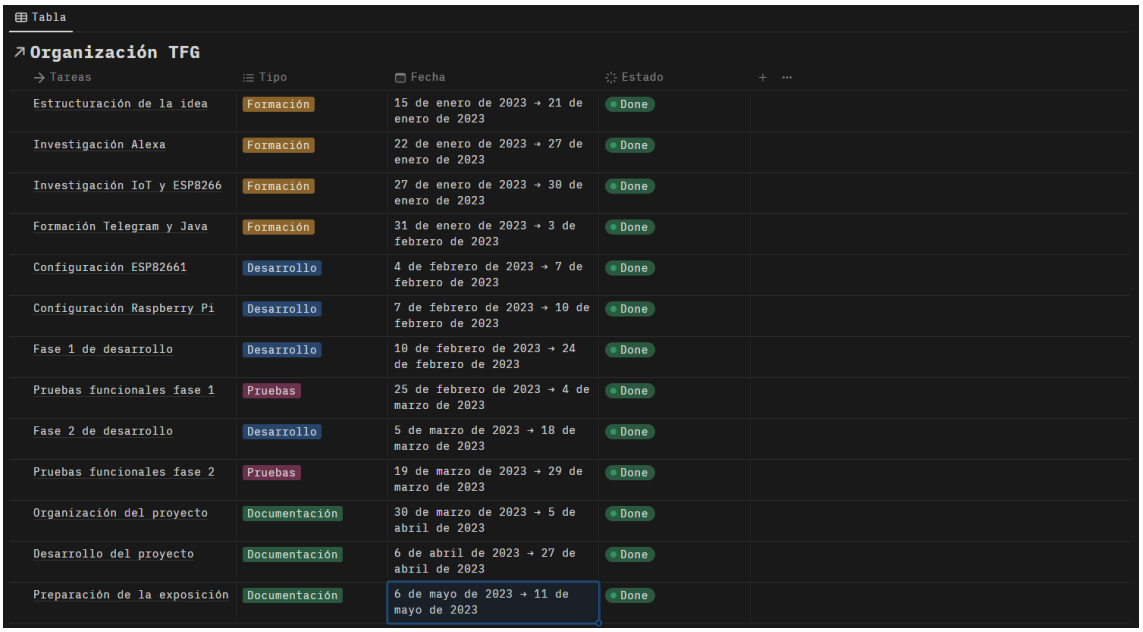
- Comunicación completamente local, sin acceder a los servidores AWS de Amazon para la comunicación entre el dispositivo Echo Dot y el dispositivo donde se ejecutan los scripts.
- Implementación de seguridad a la hora de la ejecución de scripts mediante el protocolo SSL en las conexiones entre el servidor y el dispositivo con un certificado autofirmado, cifrando así el intercambio de datos e información sensible en la LAN.
- Diseño e implementación de un *Bot* con la aplicación Telegram como método de confirmación de las ejecuciones, para evitar la ejecución accidental o malintencionada de un script.
- Conversión del proyecto Java del lado del cliente en un servicio de Windows, para que tenga un funcionamiento transparente al usuario final.

Capítulo 3. Metodología

En este capítulo se detalla la metodología y organización mantenida para el desarrollo del proyecto. Al tratarse de una idea de proyecto personal, he tenido que emplear un sistema de organización de proyectos robusto, marcando así una ruta clara a la hora del diseño y desarrollo de este, comenzando en primer lugar con la selección de un software o herramienta de calidad que permita planificar este tipo de diseños.

En el caso de este proyecto, se ha seleccionado Notion como la herramienta de planificación y organización de todo el proyecto, sus etapas y las estimaciones temporales de cada fase. Para la gestión del software y el manejo de un repositorio para el trabajo desde distintos equipos, se ha optado por Git como gestor de versiones del proyecto.

Notion ha sido una herramienta indispensable en todo el desarrollo del proyecto. Este software de gestión y organización nos permite utilizar su potentísimo, a la vez que sencillo y accesible sistema de gestión multiplataforma, brindándonos así la oportunidad de planificar todo el proyecto y plasmar las ideas que iban apareciendo a lo largo del desarrollo y en su fase inicial. Se ofrecen al usuario miles de plantillas de organización y planificación, a la vez que la libertad para diseñar un esquema propio de trabajo, con herramientas como listas "TODO", documentos donde volcar ideas repentinas o "brainstorming" en inglés, muy útil para anotar pequeños detalles que añadir durante el desarrollo que no estaban contemplados en un principio, o para el propio inicio del proyecto. También nos ofrece calendarios, bases de datos donde añadir tanto ideas como fases de desarrollo y cronogramas donde visualizar el estado del proyecto en el tiempo.



Tabla

Organización TFG

Tareas	Tipo	Fecha	Estado
Estructuración de la idea	Formación	15 de enero de 2023 → 21 de enero de 2023	Done
Investigación Alexa	Formación	22 de enero de 2023 → 27 de enero de 2023	Done
Investigación IoT y ESP8266	Formación	27 de enero de 2023 → 30 de enero de 2023	Done
Formación Telegram y Java	Formación	31 de enero de 2023 → 3 de febrero de 2023	Done
Configuración ESP82661	Desarrollo	4 de febrero de 2023 → 7 de febrero de 2023	Done
Configuración Raspberry Pi	Desarrollo	7 de febrero de 2023 → 10 de febrero de 2023	Done
Fase 1 de desarrollo	Desarrollo	10 de febrero de 2023 → 24 de febrero de 2023	Done
Pruebas funcionales fase 1	Pruebas	25 de febrero de 2023 → 4 de marzo de 2023	Done
Fase 2 de desarrollo	Desarrollo	5 de marzo de 2023 → 18 de marzo de 2023	Done
Pruebas funcionales fase 2	Pruebas	19 de marzo de 2023 → 29 de marzo de 2023	Done
Organización del proyecto	Documentación	30 de marzo de 2023 → 5 de abril de 2023	Done
Desarrollo del proyecto	Documentación	6 de abril de 2023 → 27 de abril de 2023	Done
Preparación de la exposición	Documentación	6 de mayo de 2023 → 11 de mayo de 2023	Done

Ilustración 1. Fases del proyecto en Notion

Para la planificación en Notion, se ha utilizado una base de datos donde se han añadido los objetivos o etapas del proyecto. Se han diferenciado 4 etapas distintas, una primera etapa de formación en la que se incluye todo el proceso de brainstorming inicial sobre las posibles ideas de proyecto, el cómo llevarlo a cabo, y la formación tecnológica necesaria para comenzar con el desarrollo. En segundo lugar, está la etapa de desarrollo, en la que se diseña el software necesario para el proyecto, y se configura el hardware donde este se desplegará. Junto con el desarrollo, aparece la etapa de pruebas en la que se realizan distintos ensayos para comprobar la correcta funcionalidad de la integración hardware y software realizada y se anotan las posibles nuevas funcionalidades a implementar o reparar. Por último, existe la etapa de documentación donde se expondrá todo el proyecto y su presentación.

Notion nos permite también representar todas estas etapas y las distintas actividades de cada una en un diagrama temporal o cronograma. En este se muestra el estado de las tareas y su distribución en el tiempo

Ilustración 2. Cronograma de la planificación del proyecto en Notion

Capítulo 4. Formación y documentación

4.1 Amazon Alexa

4.1.2 Alexa

Alexa es un servicio de voz desarrollado por Amazon y lanzado al mercado en 2014 junto con los dispositivos "Echo". Este servicio es cloud-based , es decir, toda su ejecución se realiza en los servidores de Amazon. Estos servidores se encargan de recoger los comandos de voz que los usuarios envían a los dispositivos que integran Alexa Voice Service (AVS).

Amazon pone a libre disposición la posibilidad de integrar AVS en dispositivos de terceros, más allá de sus productos Echo. Al integrarse, el cliente tiene acceso a Cloud-based Automatic Speech Recognition (ASR), encargado del reconocimiento de la voz.

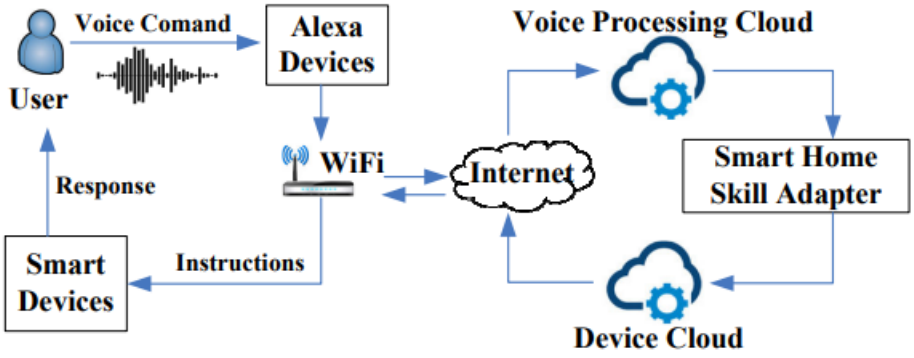


Ilustración 3. Flujo de un comando de voz hasta su ejecución
Fuente: The Insecurity of Home Digital Voice Assistants [1]

4.1.2 Skills

Amazon y sus dispositivos Alexa permiten al desarrollador la creación de *Skills*. Estas *Skills* no son más que funcionalidades o aplicaciones desarrolladas por terceros y publicadas en la plataforma de Amazon Alexa, donde cualquier persona puede instalarlas en sus propios dispositivos y utilizarlas.

Amazon pone a nuestra su disposición una serie de herramientas para el desarrollo de estas aplicaciones, englobadas en su Kit de herramientas de Alexa (ASK). Este kit proporciona al usuario una interfaz de usuario gráfica (GUI) para definir los llamados modelos de interacción en las Skills. Estos modelos de interacción no son más que la relación entre la lógica de negocio de nuestra aplicación, y el interfaz de voz que utiliza el usuario final para comunicarse con el dispositivo Alexa.

Los modelos de interacción utilizan una serie de componentes que el usuario proporciona al hacer la llamada a la *Skill* y definen el comportamiento de esta [2]:

- Los **Intent** son un tipo de input que podemos definir como la "intención" o "propósito" del usuario al llamar a la Skill. Definen por así decirlo la habilidad de la acción y pueden ser invocados mediante frases de activación. Por ejemplo, si le pregunto a un dispositivo con Alexa integrado "Alexa, ¿qué hora es?", el intent podría ser "obtener la hora".
- Los **custom slots** complementan a los intents, son la lista de posibles argumentos que acepta la Skill al llamarla. En el ejemplo anterior, un slot sería añadir a la pregunta el país del que solicitamos saber la hora, "Alexa, ¿Qué hora es en Colombia?". El intent continúa siendo "obtener la hora", pero le añadimos un argumento más, llamado slot, que sería "Colombia".
- También tenemos los **Sample Utterances**, las distintas maneras de llamar a un intent. La existencia de este componente se debe a la variedad de maneras que el cliente puede ejecutar una misma opción, siguiendo con el ejemplo anterior, el cliente podría formular la misma pregunta diciendo "Alexa, ¿Puedes decirme la hora en Alemania?". Por este motivo, han de definirse una serie de distintas llamadas de invocación a la Skill para que el cliente tenga una experiencia sencilla de comunicación con nuestro producto.

Esta serie de componentes son solo el primer paso de lo que define Amazon como el workflow adecuado en el desarrollo de aplicaciones o Skills para Alexa.

Ilustración 4. Skill development workflow
Fuente. Amazon Developer [3]

Forman parte del apartado "Build your skill" (Ilustración 1) y *ASK* nos propone distintas metodologías para llevar a cabo la construcción de la skill, desde Python hasta NodeJS, pasando por otros tantos lenguajes de programación. Sin embargo, en este apartado de desarrollo existe un detalle común a todas estas opciones, y este es Amazon Web Services Lambda (AWS Lambda) como servicio FaaS o Función como Servicio. Esto significa que la lógica Skill se ejecutará en un servidor AWS de Amazon, y por él pasarán todas las peticiones que hagamos con un dispositivo Alexa cada vez que queramos consumirlo.

4.1.3 AWS Lambda

AWS Lambda es un servicio *cloud* de Amazon sin servidor que nos permite la ejecución de código backend sin la necesidad de un servidor propio, y pagando únicamente por el uso del servicio que hemos diseñado [4].

Este servicio ejecuta nuestro código como respuesta a eventos o "triggers". Cada vez que invocamos nuestro Skill de Alexa, el servidor AWS compila el código con su lógica y lo ejecuta, devolviendo al dispositivo la respuesta correspondiente.

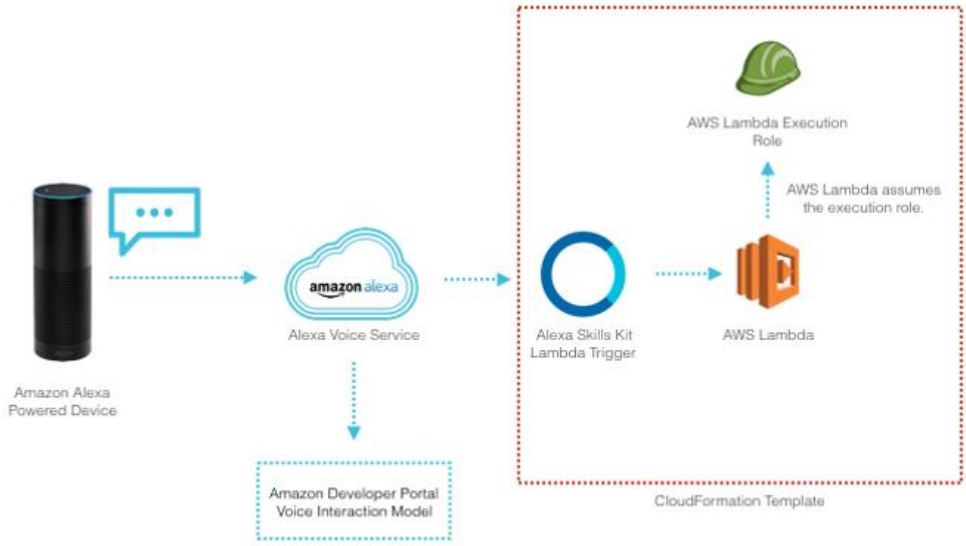


Ilustración 5. Esquema de comunicación entre Alexa y el backend de AWS
Fuente: Alexa Blogs [5]

Este es el esquema de comunicación desde nuestro dispositivo Alexa hasta la ejecución de nuestro código en un servidor de Amazon. Como observamos existe un trigger en el ASK encargado de avisar a AWS Lambda. Este servicio asume la responsabilidad de la ejecución de nuestro código, y al recibir la señal realiza esta ejecución.

Durante la invocación de la skill, los servicios de Amazon en el backend reciben un JSON correspondiente al mensaje del usuario a su dispositivo, y de él extraen el antes nombrado "intent" o intención de la llamada a la skill, correspondiente con una función en el código. Una vez extrae el intent, ejecuta la función en un servidor AWS y envía de vuelta la respuesta en formato JSON que será interpretada por el dispositivo.

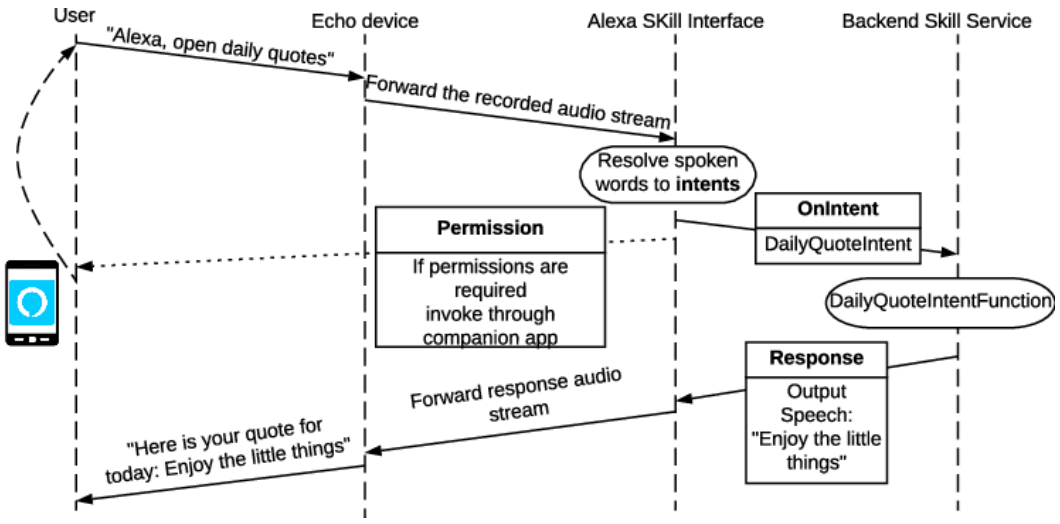


Ilustración 6. Interactive workflow of an Amazon Alexa skill.
Fuente: "Hey Alexa, is this Skill Safe? Taking a Closer Look at the Alexa Skill Ecosystem" [6]

4.2 Dispositivos Echo

4.2.1 Conexión a Cloud

Debemos dejar clara la diferencia entre Alexa como servicio de voz y los dispositivos *Echo*, diseñados por Amazon tal y como Alexa, pero que en este caso son simplemente altavoces inteligentes encargados de recoger nuestra voz en busca de comandos. Estos dispositivos implementan en ellos el servicio Alexa, por lo que al escuchar la palabra clave de activación "Alexa", enseguida captan nuestras órdenes y las envían a Amazon Cloud.

Los dispositivos Echo están obligados a mantener una conexión a internet estable y continua para su funcionamiento, debida a su integración con el servicio Alexa. Todos los comandos que escucha Echo son enviados para su procesamiento en el backend de Amazon Cloud y cuenta con todos los microservicios que ofrece Alexa, tales como ASR o Alexa Skills.

Ilustración 7. Echo Dot 3ª generación

4.3 Philips Hue

4.3.1 Ecosistema Hue

Philips, la gigante empresa tecnológica neerlandesa es toda una referente en el campo del IoT. En 2012 aparece el ecosistema "Philips Hue", una solución para la integración de lámparas y bombillas LED en el hogar, todas gestionadas desde un dispositivo central controlador llamado Philips Hue Bridge, que utilizando el protocolo ZigBee o por HTTP controla estas lámparas o bombillas.

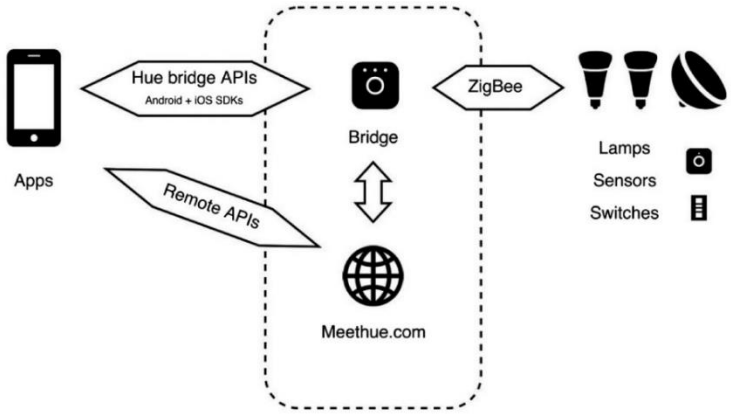


Ilustración 8. Esquema de comunicación del ecosistema Philips Hue.
Fuente. A Case Study of the Philips Hue Ecosystem [7]

4.3.2 Integración Philips Hue con Alexa

Philips y toda su línea de dispositivos Hue tienen una sencilla y directa integración con el servicio Alexa. Amazon define una serie de dispositivos de distintas marcas y les aplica la denominación "Certified for Humans" [8]. Este título acredita a estos dispositivos como excelentes y asegura su calidad, asegurando una muy sencilla integración con Alexa, al estilo "Plug and Play".

Amazon explica que estos dispositivos solo han de ser conectados a la misma red local a la que está conectado nuestro Echo o cualquier otra implementación de Alexa, y decir la frase "Alexa, busca dispositivos". Cualquier dispositivo certificado será localizado e integrado inmediatamente para ser controlado por Alexa. Los dispositivos Philips Hue tienen este tipo de integración con Alexa y pueden ser muy fácilmente controlados.

En este caso, Alexa se comunica con el Philips Hue Bridge mediante un servidor que expone este dispositivo en su puerto 80 y 443. El bridge le indica a Alexa las lámparas o luces LED que controla, y esta los guarda como dispositivos controlables por ella. Realmente, Alexa se comunica únicamente mediante mensajes HTTP con nuestro bridge a través de su API, y este se encarga de comunicarse con las LED.

4.3.3 SSDP Discovery

Simple Service Discovery Protocol es un protocolo creado en 1999 que permite a clientes HTTP de una red de área local descubrir servicios o recursos de la misma red mediante el protocolo UDP en el puerto 1900. Este protocolo funciona sobre la dirección de red multicast 239.255.255.250 y permite a los clientes HTTP, a los que denomina como SSDP Clients, descubrir los servicios que se anuncian en su LAN, los SSDP Services.

SSDP forma parte del conjunto de protocolos Universal Plug and Play (UPnP) y busca brindar a los clientes de manera sencilla y con una mínima configuración, la posibilidad de descubrir dispositivos o servicios en la red. Este es el protocolo utilizado por los asistentes de voz Echo de Amazon para descubrir las bombillas del ecosistema Philips Hue y el resto de dispositivos denominados como "Certified for Humans". Cuando emitimos la orden "Alexa, busca dispositivos", nuestro dispositivo Echo envía unos mensajes SSDP Discovery request en

multicast anunciándose a los posibles servicios SSDP, a los que nuestras bombillas IoT responden con un SSDP Discovery response. Ante la respuesta de los dispositivos, Echo registra las IP de los servicios y nos proporciona control total de ellas, ahora mediante comunicaciones HTTP locales, directamente con los dispositivos encontrados utilizando la API del fabricante.

4.4 Ampliar conocimientos de Java

4.4.1 Spring

Spring es el framework más importante para desarrollo en Java. Spring es un framework que busca la comodidad y flexibilidad a la hora del desarrollo con Java, y está compuesto por varios componentes, como son Spring Boot, Spring Security, Spring Data y un largo etcétera de herramientas que recoge Spring. Para el desarrollo del proyecto se ha hecho uso de varios componentes:

- **Spring Boot**: Se trata de un framework orientado al desarrollo web con Java. Permite iniciar aplicaciones web y desplegarlas de manera sencilla y rápida, con muy pocos archivos XLM de configuración. Existe la web de Spring initializr [9], la cual nos permite generar rápidamente un proyecto de Spring Boot con las dependencias que indicamos, junto con la versión de Java, el tipo de empaquetado, el gestor de dependencias, etc. Esto facilita mucho el trabajo.

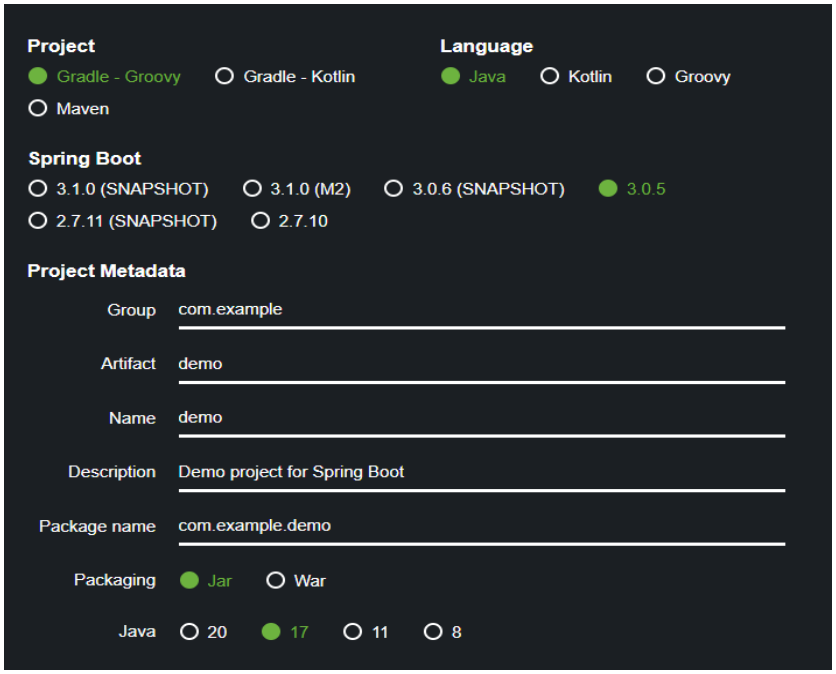


Ilustración 9. Generación de proyecto Spring Boot con Spring Initializr
Fuente. Spring Initializr [9]

- **Spring Data**: Esta es una herramienta para la conexión a bases de datos, incluye los conectores necesarios y facilita la sintaxis de las “queries”, automatizando muchos

procesos. Se basa en la arquitectura de repositorios que acceden a las bases de datos y recuperan los datos para su uso en la capa de servicio.

- **Spring Security**: Framework de Spring encargado de aplicar seguridad a lo largo de toda la aplicación. Es fácilmente configurable mediante el archivo de properties de Spring y permite aplicar seguridad de manera casi inmediata en las aplicaciones web.

Otra de las características más notables son las anotaciones en las clases Java desarrolladas, para aportar más información al framework sobre el funcionamiento o finalidad de estas. Son muy utilizadas para definir clases controladoras, las que manejan las peticiones HTTP de la aplicación web y sus endpoints; clases de servicio, las que alojan la lógica de negocio de la aplicación; clases de repositorio, las encargadas de la comunicación con la base de datos, y muchas otras. Por último, las anotaciones son muy utilizadas para definir la inyección de dependencias, concepto muy importante en Spring.

4.4.2 Arquitectura MVC e inyección de dependencias

Spring nos insta a utilizar la arquitectura MVC junto con la inyección de dependencias. Las aplicaciones Java basadas en el modelo "Model, View and Controller" (MVC) mantienen una arquitectura basada en los tres módulos indicados.

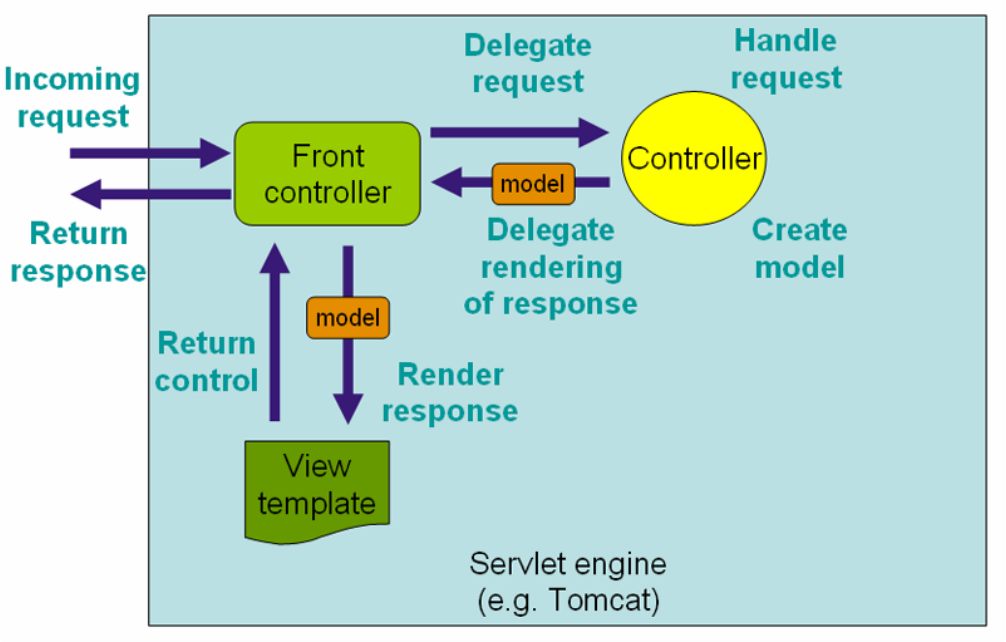


Ilustración 10. Esquema de la arquitectura MVC
Fuente. Web MVC framework [10]

Comenzando por el modelo, este módulo de la aplicación contiene todos los datos que utiliza la información, y es capaz de acceder a ella, crearla, modificarla o eliminarla.

A continuación tenemos la vista, es la capa contenedora de todas las vistas o páginas que el cliente de la aplicación puede observar e interactuar. Se encarga de renderizar los datos del modelo y se actualiza cada vez que estos datos del modelo se ven modificados, para presentar al cliente la correcta información.

Por último, tenemos el módulo controlador. Esta capa es la que comunica al usuario con la aplicación. Recoge todo tipo de interacción que el usuario realiza en la vista, como obtener la página en sí, o pulsar un enlace, un botón, etc. Y comunica al modelo este comportamiento del usuario para que, siguiendo la lógica implementada, realice unas acciones u otras con los datos que envía a la vista [11].

También existe otro concepto clave en el desarrollo Java con el framework Spring y es la inyección de dependencias. Java es un lenguaje de programación orientado a objetos (POO), por lo que es familiar el concepto de dependencias, instancias, objetos y clases. Cuando definimos un objeto en una clase y queremos utilizar ese objeto y sus métodos en otra clase distinta, debemos instanciarlo utilizando su constructor. Esto hace que si un objeto se utiliza mucho en varias clases distintas, deberemos definir muchas instancias de este y controlarlas todas por separado. Spring se encarga de esto mediante la inyección de dependencias, utilizando la arquitectura de inversión de control y su IoC Container.

La inversión de control es una arquitectura en la que, como su nombre indica, se invierte el control a la hora de instanciar objetos en distintas clases. Normalmente, estas instancias las define el programador mediante un constructor del objeto en la clase que se desee. Con esta arquitectura, es el framework quien maneja las instancias y no el programador. Para ello, el programador indicará a Spring qué objetos serán instanciados en otras clases mediante anotaciones. Existen varias anotaciones que nos indican qué tipo de objeto va a ser instanciado, por ejemplo, tenemos la anotación @Service, que indica a Spring que esta clase va a ser utilizada como un servicio y se usarán sus métodos para implementar la lógica de negocio en la capa controladora del programa, o @Repository, que indica que será un objeto o clase repositorio, que se encarga de la comunicación con la base de datos. Estas clases reciben el nombre de inyectables.

Una vez Spring conoce qué clases son inyectables, permite al programador utilizar la anotación @Autowired, la cual recibe como parámetro una de estas clases y la inyecta o instancia en la clase seleccionada. Este proceso funciona gracias al contenedor de inversión de control o IoC Container, un contendor que proporciona Spring en el que todas las clases inyectables son creadas y administradas por el propio framework, y son inyectadas cuando el usuario las requiere mediante la etiqueta antes nombrada. Spring también se encarga de manejar el ciclo de vida de los objetos inyectados, haciendo un uso correcto y óptimo de la memoria del programa.

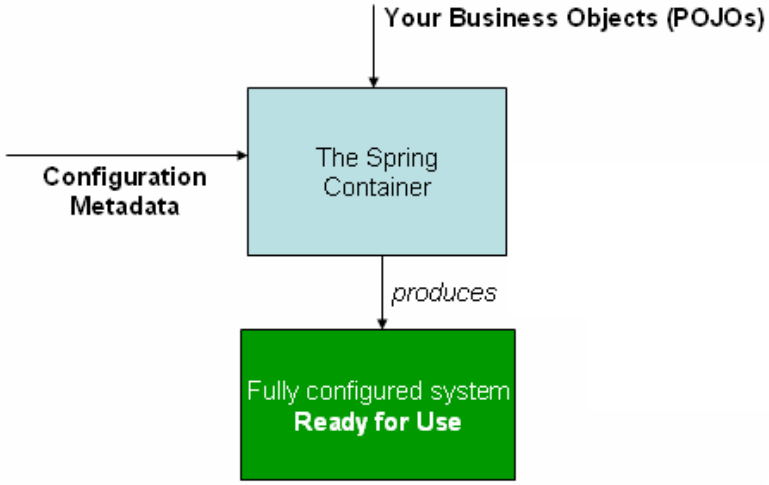


Ilustración 11. Esquema IoC Container de Spring
Fuente. The IoC container [12]

4.5 Telegram

Telegram es una aplicación de mensajería instantánea rusa lanzada en 2013 y utilizada a nivel global. Para este proyecto se utilizará Telegram para la comunicación con el usuario, pedir su confirmación y alertar de las ejecuciones de scripts en su equipo a través de comandos de voz de Alexa.

Telegram es la aplicación perfecta para este proyecto debido a la sencilla implementación de bots para interactuar con el usuario final. Todo esto es gracias a que la aplicación nos pone a libre disposición una API para el desarrollo y control de bots privados, junto con librerías en varios lenguajes de programación que nos ayudan a facilitar todo este proceso.

La creación de bots se divide en varios pasos, comenzando por dar de alta el bot en primer lugar. Para ello se hace uso del @BotFather, un bot creado por Telegram para la creación y gestión de bots personales en la aplicación. Los bots no son más que cuentas especiales, las cuales no necesitan tener un número de teléfono asignado y son totalmente programables por el usuario mediante la API que se nos proporciona. Al crear un bot se nos proporciona una API Key, una clave necesaria para la programación de este. Telegram se encarga de la comunicación entre el usuario y el bot mediante una conexión encriptada HTTPS, el usuario únicamente se encarga del diseño de la lógica del bot mediante las librerías que nos proporciona Telegram.

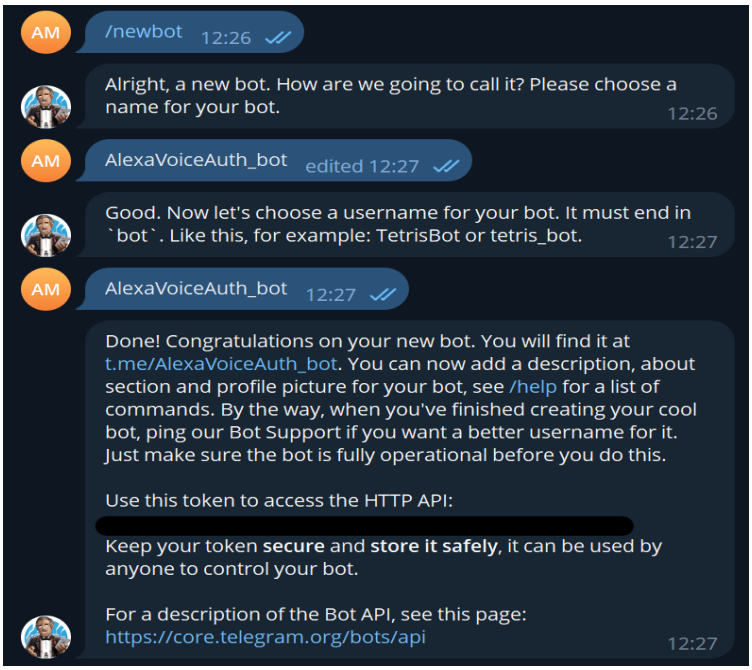


Ilustración 12. Creación de un Bot con Bot Father

En Java, la librería recomendada es la de "TelegramBots" [15], esta librería nos proporciona principalmente tres métodos con los que trabajar, que heredan de la clase "TelegramLongPollingBot". Estos métodos son onUpdateReceived(), getBotUsername() y getBotToken(). El primer método es el más interesante y se ejecutará cada vez que el bot reciba algún tipo de mensaje, como una función callBack(). Puede ser un mensaje de texto, una imagen, un archivo, cualquier tipo de comunicación que establezca el usuario final, y es aquí donde se puede definir toda la lógica del bot.

También aparece un concepto importante como son los Inline Keyboards y los Inline Buttons de Telegram. Estos componentes de la aplicación proporcionan la capacidad al bot de enviar posibles respuestas predefinidas al usuario ante una pregunta que él mismo formula, eliminando así la posibilidad de que el usuario emplee respuestas incorrectas y facilitando mucho el manejo de estas.

Ilustración 13. La herramienta Inline Buttons de Telegram

4.6 Seguridad

4.6.1 SSL/TLS

El protocolo TLS es el utilizado para cifrar las conexiones entre las distintas partes del proyecto, accediendo al implementarse en los servidores mediante HTTPS a los distintos endpoints disponibles. Este cifrado de las comunicaciones elimina las posibilidades de interpretar una posible interceptación de las comunicaciones entre los dispositivos por parte de un tercero.

Este protocolo se puede implementar fácilmente en las aplicaciones Java Web que integran el framework de Spring mediante una sencilla configuración en las propiedades de la aplicación.

Java incorpora por defecto en su instalación base una herramienta llamada "keytool" preparada para su ejecución en línea de comandos. Esta herramienta nos permite generar pares de claves públicas y privadas para el cifrado de la conexión y certificados. Para establecer conexiones seguras por HTTPS necesitamos generar un par de claves pública y privada. Para ello utilizamos el parámetro "-genkeypair" para indicarle a la herramienta que tiene que generar un par de claves, un parámetro "-keypass" indicando la contraseña que utilizaremos para generar las claves, también asignaremos un alias con el parámetro "-alias". Por último, tendremos que seleccionar el tipo de formato del almacén de claves y su clave, con los comandos "-storetype" y "-storepass".

Keytool admite dos formatos para generar el almacén de claves, Java KeyStore (JKS) y PKCS#12. El primero de los dos es el formato propietario que utiliza Java para sus aplicaciones, mientras que PKCS#12 es un formato estandarizado en toda la industria. Para asegurarse de evitar problemas de compatibilidad, es recomendable el uso de este último formato, el marcado como estándar.

Por último, simplemente hay que señalar al framework de Spring que se va a encriptar la comunicación web con el servidor mediante el protocolo TLSv1.2, la ruta del almacén de claves y la clave de esta para acceder. El propio framework se encargará totalmente de la encriptación de toda la comunicación con el servidor.

```
alvaro@HomeServer:~/Escritorio/TFG$ keytool -genkeypair -alias tfgssl -keyalg RSA -keysize 4096 -storetype JKS -keystore
 tfgcertificate.jks -validity 3650 -storepass [redacted]
¿Cuáles son su nombre y su apellido?
  [Unknown]:  Alvaro Martinez
¿Cuál es el nombre de su unidad de organización?
  [Unknown]:  UPV
¿Cuál es el nombre de su organización?
  [Unknown]:  UPV
¿Cuál es el nombre de su ciudad o localidad?
  [Unknown]:  Valencia
¿Cuál es el nombre de su estado o provincia?
  [Unknown]:  Valencia
¿Cuál es el código de país de dos letras de la unidad?
  [Unknown]:  ES
¿Es correcto CN=Alvaro Martinez, OU=UPV, O=UPV, L=Valencia, ST=Valencia, C=ES?
  [no]:  s

Generando par de claves RSA de 4.096 bits para certificado autofirmado (SHA384withRSA) con una validez de 3.650 días
        para: CN=Alvaro Martinez, OU=UPV, O=UPV, L=Valencia, ST=Valencia, C=ES
Introduzca la contraseña de clave para <tfgssl>
        (INTRO si es la misma contraseña que la del almacén de claves):
```

Ilustración 14. Generación de un par de claves con Keytool

4.6.2 Spring Security

Spring security forma parte del framework Spring, y nos permite aplicar distintos filtros de seguridad en todo tipo de aplicaciones Java. Para su instalación, simplemente hay que agregar la librería en el POM.xml del proyecto y realizar una compilación completa con Maven.

En este proyecto, este framework nos permitirá aplicar filtros en las conexiones entrantes y salientes de cada aplicación, logrando así que solo los dispositivos que el usuario desee puedan comunicarse.

Para ello, se define una clase de configuración en el módulo de aplicación del proyecto para Spring Security. Esta clase recibe como parámetro un objeto HttpSecurity el cual nos proporciona un método llamado authorizeRequests(). Este método permite al usuario elegir cuando se autoriza una petición al servidor. Mediante la llamada al método hasIpAddress() podemos indicar qué direcciones tienen permiso para acceder a ciertas URL del servidor.

Al ser un proyecto en una red local, las direcciones IP son fijas y están establecidas por el usuario, así que estas no deberán asignarse dinámicamente, bastará con indicar todas ellas en el fichero de propiedades de la aplicación de Spring Boot.

Capítulo 5. Desarrollo del proyecto

Una vez finalizado el periodo de investigación y estudio sobre las posibles herramientas y conceptos necesarios para la implementación del proyecto, comienza el desarrollo de este.

El proyecto tiene como fin la implementación de una solución hardware y software para lograr la ejecución de comandos en dispositivos Windows de la misma red local, de un modo seguro, sencillo y buscando que la comunicación entre los dispositivos sea local, sin tener que abrir algún puerto en la red.

Todo el desarrollo se ha realizado de forma modular, pivotando sobre las tres aplicaciones software del proyecto que se comunican entre sí para lograr la ejecución de los comandos. El primer objetivo del desarrollo consiste en una implementación funcional, en la que se logre ejecutar Scripts en el terminal destino mediante comandos de voz, y partiendo de la funcionalidad básica se construye una herramienta sólida y que cumpla los objetivos o hitos propuestos para el proyecto.

Primero se abordará el tema del hardware utilizado, seguidamente el software, y por último se explica el funcionamiento y despliegue del proyecto, enlazando ambas partes antes mencionadas y explicadas.

5.1 Hardware

Para lograr una implementación local del proyecto, necesitamos varios dispositivos hardware que nos ayuden en esta tarea. Sin embargo, se ha buscado en todo momento una solución sencilla, segura y económica, que permita un sencillo despliegue del proyecto en cualquier LAN. En este apartado hablaremos de los productos hardware necesarios y el por qué de su elección.

5.1.1 Dispositivo Echo Dot 3ª Generación

El altavoz Echo de Amazon seleccionado para este proyecto será el modelo de 3ª generación de la familia Echo Dot. Este dispositivo aparece en el año 2020, con un tamaño reducido y un bajo coste, actualmente rondando los 25 euros en la web de Amazon y con unas prestaciones más que correctas para el desarrollo del proyecto. Tiene acceso integrado al servicio de asistente por voz Alexa y conexión WiFi para conectarse inalámbricamente a nuestra red local.

Aunque existen otros modelos más potentes y actuales que incluso integran su propia pantalla táctil para el control de las aplicaciones instaladas, no haremos uso de esas características en el proyecto propuesto, por lo que no será necesario utilizar estos nuevos dispositivos. Sin embargo, esta solución seguiría funcionando sin inconvenientes en estos dispositivos de posterior generación, pero no en dispositivos de generaciones previas. Esto se debe a que en los dispositivos Echo Dot de 1ª y 2ª generación, existe un conocido problema relacionado con el control de dispositivos del entorno Philips Hue.

Ilustración 15. Dispositivo Echo Show 5ª generación con pantalla integrada

En estas versiones previas de Echo Dot, existe un problema con el firmware de Alexa que estos dispositivos alojan. Durante la comunicación entre Alexa y las bombillas Philips Hue o el Philips Hue Bridge, las cabeceras content-type de los mensajes HTTP de control de estas están marcadas como "application/x-www-form-urlencoded", cuando realmente deberían de ser "application/json" para un correcto “parseo” de los mensajes [13], como indica la API de Philips Hue. Este error está solucionado a partir de la 3ª generación de dispositivos.

5.1.2 Microprocesador ESP-8266 Node MCU v3

La serie de microprocesadores ESP fabricados por Espressif Systems se encuentran en lo alto del mercado de chips con prestaciones similares a él, bajo coste, conexiones inalámbricas por Wifi y Bluetooth, bajo consumo y fácilmente programables. Se caracterizan por ser un competidor directo de otros dispositivos como Arduino, con un coste menor y menos enfocados al aprendizaje hardware de electrónica, con menos kits de desarrollo o elementos electrónicos específicos para él.

Debido a su pequeño tamaño y el resto de las condiciones que ya hemos explicado, se ha decidido utilizar un modelo ESP-8266 NodeMCUv3 para el desarrollo del proyecto. Además, uno de los principales motivos por lo que se ha elegido este modelo ha sido por la compatibilidad con librerías de terceros, ya que detrás de esta pieza de hardware existe una gran comunidad de desarrollo OpenSource, en concreto para soluciones IoT.

Para la programación del chip, se ha utilizado el entorno PlatformIO integrado en Visual Studio Code y tras su programación, el chip es alimentado por la Raspberry Pi debido a su mínimo consumo, que además ejecuta automáticamente su función debido a que almacena el software con el que es programado en su memoria interna.

5.1.3 Raspberry Pi Zero W

Esta pieza de hardware también viene seleccionada por características similares al microprocesador ESP, y son su bajo coste y consumo, su capacidad de establecer conexiones inalámbricas y su sencillo manejo con su sistema operativo propio de Raspberry.

Esta placa es la menos potente de la familia Raspberry, sin embargo, para este proyecto ha sido ideal. Bajo costo y tamaño muy reducido han sido las características idóneas para potenciar la elección de este dispositivo.

La Raspberry Pi Zero W solo tiene una salida de video mini-HDMI y dos entradas USB mini, una entrada de datos y otra para alimentación. También puede alimentarse con una entrada de 12 voltios y tiene conectividad inalámbrica. Para su configuración, primero hay que instalar un sistema operativo compatible con la placa, es este caso, se ha utilizado una instalación limpia de Raspbian, sistema operativo diseñado para toda la gama de Raspberry. Esta instalación se realiza mediante un software del propietario llamado Raspberry Pi Imager, siendo esta sencilla e inmediata, sin necesidad de una configuración extensa. Sin embargo, esta placa nos brinda la oportunidad de una instalación y configuración "headless" del sistema operativo, y posteriormente un uso total de esta. El término "headless" viene relacionado con un control sin necesidad de conectar nuestra Raspberry a un monitor o display para controlarlo, si no que mediante el protocolo SSH podemos realizar la configuración completa.

Para lograr establecer una configuración SSH entre la Raspberry y otro terminar, hay que añadir a la memoria de la Raspberry un archivo de configuración SSH con los datos de conexión necesarios para conectarse a la red Wifi. Desde el terminal, en mi caso un ordenador Windows, podemos establecer una sencilla conexión con nuestro dispositivo "headless" mediante el comando "ssh raspi@<IP>", siendo "raspi" el nombre de usuario por defecto del sistema operativo Raspbian, y la IP aquella que nuestro router asigne a la placa por DHCP. Para obtener esta IP, basta con consultarla en la configuración del router, y realizar una asignación fija a esta para una comunicación correcta entre el resto de las componentes hardware del proyecto.

5.1.4 Smartphone

En este proyecto, el smartphone es un componente necesario para la cómoda y segura ejecución de comandos. Junto con la aplicación Telegram, podemos utilizar el smartphone para confirmar la ejecución de Scripts seguros, y todo el resto de funciones que nos ofrece el BOT para gestión de la aplicación.

5.1.5 Ordenador Windows

Por último utilizaremos un ordenador Windows en el que se ejecutarán los comandos del usuario. Para ello, en el dispositivo se instala un servicio que funciona cuando el ordenador arranca y es capaz de capturar las peticiones pertinentes para la ejecución de comandos.

5.2 Software

El apartado de software es el que más trabajo ha llevado. Este proyecto se divide en 3 aplicaciones o servicios distintos, que son ejecutados en los dispositivos hardware que lo conforman. Los lenguajes utilizados para el desarrollo de estas aplicaciones o servicios son Java y C++. Entraremos en detalle en la explicación de cada uno de los servicios, y todos ellos tienen su propio repositorio de GitHub para controlar el código fuente de estas.

5.2.1 ESP Hub

La primera parte del proyecto es la encargada de la comunicación entre nuestro Echo Dot, Alexa y el microprocesador ESP-8266. Para ello utilizamos la librería ESPAlexa [14], la cual nos proporciona una implementación básica de la API de Philips Hue para lograr la comunicación entre Alexa y nuestro microprocesador. Se encarga de emular un Philips Hue Bridge comunicándose con Alexa y captando las órdenes que Alexa envía cuando le realizamos las peticiones pertinentes.

Ilustración 16. Philips Hue Bridge

Para programar el microprocesador hemos empleado la herramienta PlatformIO de Visual Studio Code. Esta herramienta la podemos obtener en el catálogo de extensiones de VSCode y su instalación es automática y transparente. Tras instalarla, aparecerá una pestaña con el símbolo de la extensión y desde aquí podremos generar un nuevo proyecto para nuestro microprocesador. PlatformIO trabaja con todo tipo de hardware orientado a IoT como puede ser Arduino, ESP-32 o incluso nuestro micro ESP-8266. Al crear un proyecto, le indicamos qué tipo de placa vamos a programar y el framework de desarrollo de nuestro programa.

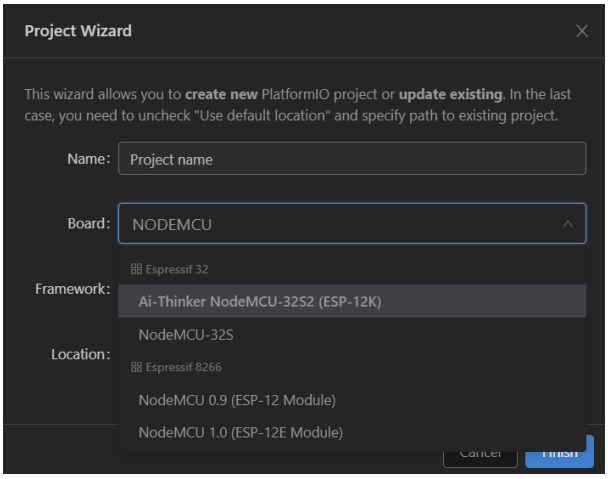


Ilustración 17. Pantalla de creación de proyectos en PlatformIO

En nuestro caso, el proyecto se ha desarrollado sobre una placa NodeMCU v3 (ESP-8266), utilizando el framework de Arduino. La placa acepta la subida de archivos de extensión

"*.ino"*, procedentes de Arduino. Estos archivos utilizan la sintaxis de C++ pero se dividen, al igual que en Arduino, en dos partes. Por un lado tenemos el método setup() y por otra parte el método loop(). El primero se ejecuta una sola vez al inicio del programa, mientras que el segundo método se ejecuta de manera continuada en, como su nombre indica, un bucle que finalizará solo cuando se apague la placa.

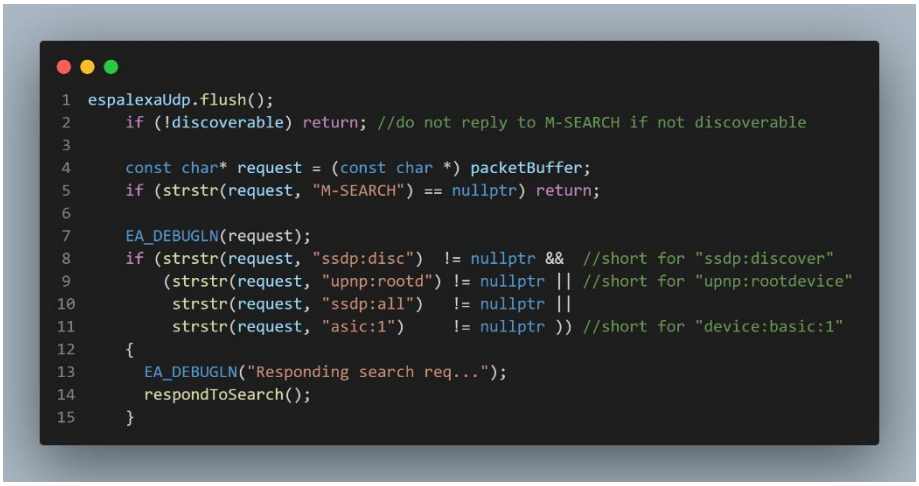
Primero se importan las librerías necesarias, en nuestro caso la ya antes nombrada ESPAlexa [14] y otras librerías que necesita esta última, como "ESP8266WiFi.h" y "ESP8266HTTPClient.h". También se definen algunas constantes que utilizaremos durante el código, como el SSID y contraseña de la red local a la que nuestro microprocesador va a conectarse, y también se crea una instancia de la clase Espalexa a la que llamaremos alexa.

El funcionamiento del programa es sencillo, por un lado, nos conectaremos a nuestra red local para poder comunicarnos correctamente con Alexa. Para ello definimos una función que nos conecta a la red cuyo nombre y contraseña ya hemos indicado y la llamamos ConectarWifi(). Para ello hacemos uso de la librería ESP8266WiFi.h y mostramos en el "Serial Monitor" el resultado de la conexión. En el caso de que sea correcta, se mostrará la IP local asignada a nuestro microprocesador y un mensaje indicando que el servidor HTTP necesario para la comunicación con Alexa está desplegado.

Ilustración 18. Monitor serial del microprocesador ESP8266

Ahora hay que adentrarse en la librería "Espalexa.h". Al instanciarla en nuestra clase main, podemos acceder a todos los métodos que tiene Espalexa. Esta clase tiene dos funcionalidades principalmente, una es la de encargarse de responder a los mensajes SSDP que emite nuestro Echo Dot para descubrirse como dispositivo Philips Hue Bridge, y otra es la de captar las llamadas que hace el usuario para ejecutar un script.

La primera funcionalidad la recoge el método llamado respondToSearch(). Este método implementa una respuesta a los mensajes multicast UDP SSDP M-SEARCH que emite nuestro Echo Dot ante la orden de buscar nuevos nuevos dispositivos en la red. Nuerstro microprocesador capta las request enviadas a la dirección 239, 255, 255, 250 y comprueba sus cabeceras para asegurarse que se trata de un mensaje de búsqueda de dispositivos de Alexa.



```
espalexaUdp.flush();
    if (!discoverable) return; //do not reply to M-SEARCH if not discoverable

    const char* request = (const char *) packetBuffer;
    if (strstr(request, "M-SEARCH") == nullptr) return;

    EA_DEBUGLN(request);
    if (strstr(request, "ssdp:disc")  != nullptr &&  //short for "ssdp:discover"
        (strstr(request, "upnp:rootd") != nullptr || //short for "upnp:rootdevice"
         strstr(request, "ssdp:all")   != nullptr ||
         strstr(request, "asic:1")     != nullptr )) //short for "device:basic:1"
    {
      EA_DEBUGLN("Responding search req...");
      respondToSearch();
    }
```

Ilustración 19. Filtrado de peticiones SSDP Discovery y llamada al método respondToSearch()

Una vez lo comprueba, ejecuta el método respondToSearch(), indicando a Alexa que se trata de un Philips Hue Bridge gracias a las cabeceras “LOCATION”, “SERVER” y “hue-bridgeid”.

```
void respondToSearch()
  {
    IPAddress localIP = WiFi.localIP();
    char s[16];
    sprintf(s, "%d.%d.%d.%d", localIP[0], localIP[1], localIP[2], localIP[3]);

    char buf[1024];

    sprintf_P(buf,PSTR("HTTP/1.1 200 OK\r\n"
      "EXT:\r\n"
      "CACHE-CONTROL: max-age=100\r\n" // SSDP_INTERVAL
      "LOCATION: http://%s:80/description.xml\r\n"
      "SERVER: FreeRTOS/6.0.5, UPnP/1.0, IpBridge/1.17.0\r\n" // _modelName, _modelNumber
      "hue-bridgeid: %s\r\n"
      "ST: urn:schemas-upnp-org:device:basic:1\r\n"  // _deviceType
      "USN: uuid:2f402f80-da50-11e1-9b23-%s::upnp:rootdevice\r\n" // _uuid::_deviceType
      "\r\n"),s,escapedMac.c_str(),escapedMac.c_str());

    espalexaUdp.beginPacket(espalexaUdp.remoteIP(), espalexaUdp.remotePort());
    #ifdef ARDUINO_ARCH_ESP32
    espalexaUdp.write((uint8_t*)buf, strlen(buf));
    #else
    espalexaUdp.write(buf);
    #endif
    espalexaUdp.endPacket();
  }
```

Ilustración 20. Método respondToSearch()

Una vez el dispositivo Echo ha localizado nuestro Philips Hue Bridge emulado, comienza la segunda parte o funcionalidad, que consiste en listar u obtener todos los dispositivos que nuestro bridge controla, y ejecutar la lógica que deseemos cuando el usuario manda a Alexa una orden sobre uno de los dispositivos.

Esta comunicación se lleva a cabo mediante la implementación parcial de la API de Philips Hue. Esta comunicación está definida por Philips y se realiza mediante llamadas HTTP con el protocolo TCP desde el dispositivo Echo a nuestro microprocesador. En primer lugar, Alexa necesita conocer los dispositivos que controla el bridge. Para ello, el Echo envía una petición GET a la URL "https://<ip_bridge>/api/<nombre_usuario>/lights", y el servidor alojado en el microprocesador le responde con los dispositivos asociados a él. Una vez recibe esta información, Alexa ya conoce los dispositivos y nos permite controlarlos mediante comandos de voz. En la librería ESPAlexa están implementadas las funcionalidades de encender, apagar y controlar el brillo de los dispositivos, así que Alexa puede procesar estos 3 tipos de órdenes. Según la orden que ordenemos realizar a Alexa, esta realizará una petición POST a un endpoint del servidor del microprocesador, indicando qué dispositivo quiere manejar, y la acción que quiere que realice.

```
if (body.indexOf("false")>0) //OFF command
      {
        dev->setValue(0);
        dev->setPropertyChanged(EspalexaDeviceProperty::off);
        dev->doCallback();
        return true;
      }

      if (body.indexOf("true") >0) //ON command
      {
        dev->setValue(dev->getLastValue());
        dev->setPropertyChanged(EspalexaDeviceProperty::on);
      }
```

Ilustración 21. Registro de peticiones de encendido y apagado

Como vemos en la *Ilustración 21*, el programa comprueba si existe la palabra "false" o "true" en el cuerpo de la petición del Echo Dot, conociendo así la intención del usuario con el dispositivo o la acción a realizar pasándole una función "Callback".

Una vez explicado el funcionamiento de la librería ESPAlexa, volvemos a nuestro programa. Durante el método setup() únicamente iniciaremos el monitor serial, nos conectaremos a la red local y añadiremos los dispositivos que deseemos, junto con la función de CallBack que se ejecutará cada vez que el usuario quiera encender, apagar o modificar el brillo del dispositivo indicado.

En el método loop() se ejecuta el método de conexión a la red local, pero únicamente para mostrar en el monitor serial unos símbolos que indican la espera hasta que se conecta a la red, y una vez establecida la conexión deja de ejecutarse, y un método loop de "ESPAlexa.h", encargado de escuchar los mensajes UDP SSDP M-SEARCH del Echo Dot.

```
void setup() {
  Serial.begin(9600);
  ConectarWifi();
  alexa.addDevice("Funcion1", FuncionUno);
  alexa.addDevice("Funcion2", FuncionDos);
  alexa.addDevice("Funcion3",FuncionTres);
  alexa.begin();
}

void loop() {
  ConectarWifi();
  alexa.loop();
  delay(1);
}
```

Ilustración 22. Métodos setup y loop del programa

Las funciones Callback definen la lógica de negocio tras las peticiones POST que recibe el servidor por parte del Echo. Estas peticiones son un disparador o trigger para la ejecución de estas funciones y podemos definir el comportamiento que deseemos que tenga el programa. Para nuestro proyecto, cada dispositivo emulado se trata de un script o función que se ejecutará en la máquina local Windows, por lo que el método será una llamada a un API REST alojada en nuestro servidor local indicando el script que el usuario quiere ejecutar

```
void FuncionUno(uint8_t brightness) {

    WiFiClient client;
    HTTPClient http;
    http.begin(client, "http://192.168.1.132:8081/api");  // HTTP
    int httpCode = http.POST("uno");
    String state = "off";
    if(brightness){
      state = "true";
    }
    http.POST(state);
    http.end();

}
```

Ilustración 23. Método callBack de la llamada al dispositivo uno

5.2.2 Raspberry Pi Home Server

Este es el segundo componente software independiente del proyecto y se trata de un servidor Linux alojado en una placa Raspberry Pi Zero W, que actúa como capa intermedia entre el ESP Hub, que recibe las órdenes del usuario mediante comandos de voz de Alexa, y el Alexa Windows service, el servicio ejecutor de los scripts del que hablaremos más adelante.

Este componente consiste en una aplicación web escrita enteramente en Java utilizando el framework Spring. La aplicación está construida sobre una arquitectura basada en el modelo MVC característico de este framework, y utiliza los componentes Spring Boot y Spring Data.

Esta aplicación web se trata únicamente de un diseño backend orientado al control de peticiones HTTP, al estilo de un proxy, aplicando una lógica de negocio con estas peticiones con origen nuestro microcontrolador ESP8266.

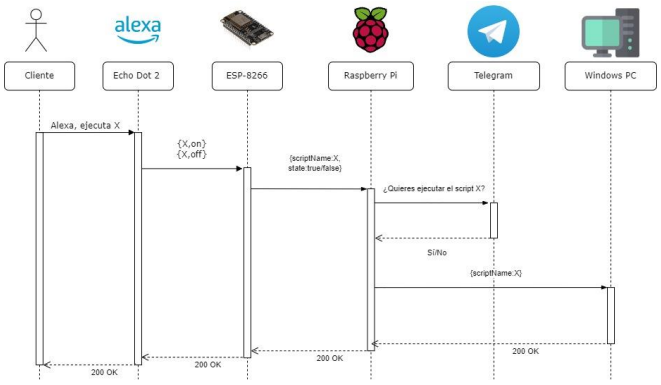


Ilustración 24. Diagrama de comunicación de la aplicación

Como vemos en la *Ilustración 24*, nuestro servidor tiene tres flujos de comunicación. Primero nombraremos los tres flujos de datos que maneja el servidor, explicando de manera superficial su funcionamiento e implementación, y después se indagará más profundamente en la implementación técnica de estos.

En primer lugar, este servidor recibe las peticiones HTTP POST del microprocesador. Estos mensajes contienen dos parámetros como vemos en el código de la *Ilustración 23*, primero informan del nombre del script que el usuario le ha pedido a Alexa ejecutar, y después indica si esta llamada ha sido de activación o de desactivación. Esto se debe al funcionamiento de la librería ESPAlexa utilizada en el ESP Hub. Al emular a una bombilla Philips Hue, Alexa utiliza su API destinada al control de luces, y entre sus posibles comandos tenemos el de apagar y encender, o activar y desactivar.

En segundo lugar, tenemos la comunicación con el usuario mediante Telegram. Esta implementación consiste en el alojamiento de un Bot de la plataforma Telegram que se encarga de comunicarse con el usuario como método adicional de seguridad y autorización. Su función es la de preguntar al usuario, en ciertos casos, si realmente desea continuar con la ejecución de cierta serie de comandos en su ordenador Windows. Pueden darse casos, en los que el usuario de manera inadecuada o equívoca haya indicado a Alexa que desea ejecutar un comando en el ordenador, o incluso que otra persona haya indicado esta ejecución indeseada. Estas posibles situaciones llevan a que el usuario pueda tener una mala experiencia, y por ello nace este servicio de confirmación de ejecución.

Por último, tenemos la tercera y última función del servidor, encargada de comunicarse con el ordenador Windows, dispositivo de ejecución de los scripts deseados por el usuario. Este bloque procesa adecuadamente las confirmaciones que el usuario envía a nuestro Bot de Telegram y envía al ordenador Windows un mensaje HTTP POST con los datos de ejecución

del script, todo de manera segura mediante el encriptado de la comunicación utilizando SSL/TLS y la tecnología que nos proporciona el framework Spring Security para aceptar peticiones únicamente de las direcciones IP deseadas mediante una lista blanca de direcciones.

Una vez presentadas las tres fases de la aplicación y su funcionamiento básico, entramos ahora en el diseño técnico de estas, con todos sus casos de uso y una revisión del código empleado.

Fase 1. Controlador de peticiones

Esta primera fase es la encargada de recoger las peticiones que nos realiza el microprocesador ESP8266, que a su vez recibe las peticiones por parte de Alexa ante las órdenes del usuario.

Este controlador se sitúa en el módulo "controller" de la aplicación Java, siendo consecuentes con la arquitectura recomendada por el framework de Spring. Dentro del modelo MVC, este módulo de la aplicación constituye, como su nombre indica, el módulo "Controller" de esta arquitectura.

La clase se llama ControladorAlexa, y tiene un solo método llamado "executionPermission", debido a la función que tiene, de conceder o no permiso para la ejecución de comandos en un ordenador Windows.

Esta clase utiliza las anotaciones @Controller, para indicarle al framework de Spring que esta clase es un Bean inyectable y manejable por este, y también la anotación @ResponseBody, para asegurarnos que el controlador genere respuestas ante las peticiones pertinentes. Esto es imprescindible si queremos construir un buen controlador, asegurándonos que el consumidor de este endpoint tenga respuesta en todo momento ante cualquier petición que realice a este, y así manejar los problemas que pueden generarse en dicha comunicación.

Dentro de la clase se definen una serie de variables globales que utiliza nuestro método, como el ID del cliente con el que se comunica el BOT, y una instancia inyectada y manejada por Spring del propio BOT, para su control y ejecución de métodos de este.

El método "executionPermission" utiliza la anotación @PostMapping("/api"), la cual cumple la función de escuchar las peticiones HTTP POST que se realizan a la ruta relativa "/api" dentro de nuestra aplicación web.

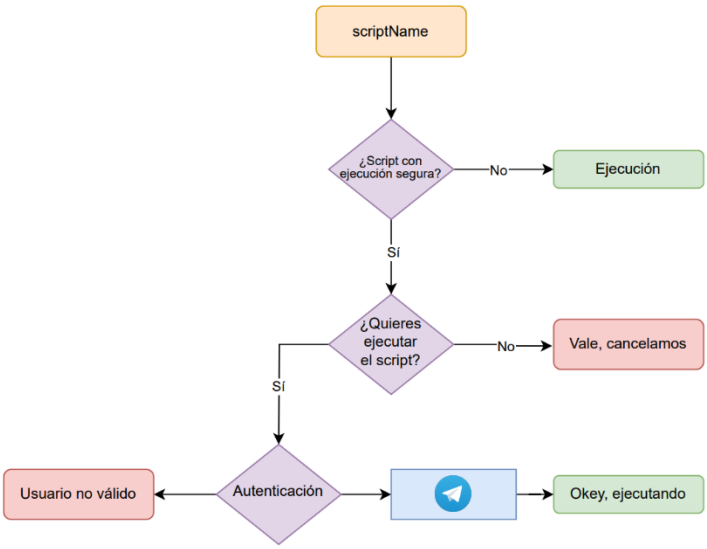


Ilustración 25. Diagrama de flujo de la fase 1

Este controlador requiere de un parámetro llamado “scriptName”, encargado de alojar la información del script que el usuario desea ejecutar. Nuestro ESP-8266 envía en su petición POST el ID del script que el usuario ha ejecutado. Este ID lo recoge nuestro servidor, y mediante una consulta a la base de datos, comprueba si el script que el usuario pretende ejecutar tiene la etiqueta de “ejecución segura”. Esta etiqueta se trata de un dato asociado a cada script en la base de datos llamado “secExec”, y nos indica cuándo un script ha de ejecutarse de manera controlada, solicitando al usuario una confirmación adicional mediante un mensaje de Telegram.

id	nombre	estado	secExec	init
1	ordenador	0	1	1
2	google	1	0	1
3	Script 3	0	1	0
4	Script 3	0	1	0
5	Hola	0	1	1
6	Script 3	0	1	0
7	Script 3	0	1	0
8	Script 3	0	1	0
9	Script 3	0	1	0
10	Script 3	0	1	0

Ilustración 26. Lista de Scripts en la base de datos

En el caso de que el script contenga la etiqueta “secExec” activa, el proceso de ejecución implementa al BOT de Telegram para la confirmación de la ejecución. Al usuario se le pide permiso mediante un sencillo mensaje. El mismo bot envía un InlineButton, tal y como se muestra en la *Ilustración 13*, y el usuario tendrá que pulsar la opción “Sí” o “No” para continuar con el flujo de ejecución del programa. Si el usuario deniega el permiso de ejecución, simplemente se cancela la ejecución y se informa al usuario de esta. Sin embargo,

en caso afirmativo, el mensaje de confirmación pasa a la capa de autenticación de la aplicación.

La capa de autenticación se sitúa en el paquete "authentication", y consta de una sola clase llamada "Authenticator". Esta clase se define como servicio para Spring mediante la anotación @Service, y puede ser inyectada en el resto de los componentes de la aplicación. Consta de dos métodos principales, llamados "checkSecureExecution" y "auth".

El método "checkSecureExecution" es el encargado de recuperar la información de la etiqueta "secExec" del script a ejecutar. Se utiliza en el controlador para decidir si se ejecuta directamente la orden, o se pregunta al usuario.

El segundo método es el de "auth", y es el encargado de verificar que el usuario que responde al BOT de Telegram para confirmar o no la ejecución del script, es un usuario perteneciente a una lista de usuarios con permisos para la confirmación de ejecución. Esta lista se encuentra en la base de datos y contiene una serie de identificadores, generados por Telegram a cada cliente que utiliza la aplicación y llamados "user_ID". Estos identificadores son los identificadores personales del usuario propietario del ordenador Windows donde se ejecutan los comandos y los que este desee que tengan acceso a la confirmación de permisos para las ejecuciones.

Una vez explicada la capa de autenticación y observando de nuevo el diagrama de flujo de esta primera fase de la aplicación en la *Ilustración 25*, observamos como el recuadro en el que aparece la pregunta "¿Script con ejecución segura?" equivale en nuestra aplicación al método "checkSecureExecution" de nuestro autenticador, así como el recuadro de "Autenticación", se refiere al método "auth". Como observamos en el diagrama, si la autenticación del usuario que concede permiso a la ejecución es correcta, nuestro BOT enviará un mensaje al usuario confirmando la ejecución del comando indicado. Ahora profundizaremos en la comunicación entre el BOT y la aplicación, información correspondiente a la segunda fase de comunicación de esta.

Fase 2. Comunicación Telegram-RaspberryPi

Esta segunda fase corresponde a la comunicación entre el servidor alojado en la Raspberry y el BOT de Telegram. Esta comunicación comienza inmediatamente después de que el controlador reciba la orden de ejecutar un script con ejecución segura.

El controlador utiliza la instancia del BOT, inyectada en la clase gracias a Spring y su anotación @Autowired. Este utiliza el método execute(), heredado de la clase de la clase TelegramBots, y envía al usuario un mensaje preguntando si es él el que está intentando ejecutar un comando. Seguido de este mensaje plano de texto, aparece un mensaje que hace uso del componente InlineButton, explicado anteriormente. Este mensaje consta de dos componentes InlineButton, uno para confirmar que queremos continuar con la ejecución, y otro para cancelar la ejecución.

El usuario, al seleccionar una de las dos opciones, enviará un mensaje automáticamente al BOT con el texto del botón pulsado. En el lado del servidor, utilizamos el método onUpdateReceived() para capturar esta interacción cliente-BOT. Como se ha utilizado un componente InlineButton, este tiene una propiedad llamada callBackData, que nos indica cuál de los dos botones ha pulsado el usuario. Cuando este cancela la ejecución, simplemente se

envía una respuesta con el mensaje "Okey, cancelamos", mientras que si la respuesta ante la confirmación de ejecución es positiva, debemos manejar la respuesta del usuario y el tipo script que se desea a ejecutar mediante la lógica de negocio establecida.

Primero, cuando el servidor recibe cualquier tipo de mensaje por parte del cliente, comprueba el ID del usuario que ha establecido esa comunicación. Ante la correcta comprobación de este por parte de la capa de autenticación, el objeto mensaje que recibimos del cliente se envía al método responseHandler(), encargado de aplicar la lógica necesaria. La primera comprobación que realiza es asegurarse si el mensaje del cliente es de un mensaje de texto plano escrito por el propio cliente, si se trata de una respuesta generada al haber pulsado un componente InlineButton y por último, si se trata a una respuesta de un mensaje anterior.

En el caso que estamos tratando, el mensaje se trata de una respuesta InlineButton por parte del cliente, por lo que comprobará el tipo de respuesta que el usuario ha enviado. Si la respuesta es negativa, como hemos indicado antes, simplemente enviará un mensaje al usuario y cancelará la ejecución. Sin embargo, si es positiva, pasamos a la tercera fase de comunicación de la aplicación, en la que esta se comunica con el ordenador Windows para indicarle el script que tiene que ejecutar.

Hemos hablado de la comunicación o funcionalidad básica entre el BOT de Telegram y la aplicación, sin embargo, existen más opciones y funcionalidades implementadas y que este BOT puede ofrecer al usuario para tener una experiencia más completa en el manejo de la aplicación.

Como vemos en la ilustración, el propio chat con el BOT nos proporciona un menú con una serie de comandos disponibles. En primer lugar, tenemos el comando "/help", que nos indica como proceder para la comunicación con el BOT, nos explica cada comando y como utilizarlo. Después tenemos el comando "/list", que proporciona al usuario una lista con todos los scripts disponibles para el usuario, su nombre, si se encuentran activos o no, y si están configurados para una ejecución segura. Esto permite al usuario tener un control sobre los scripts disponibles para ejecutar, y llevar un orden en estos.

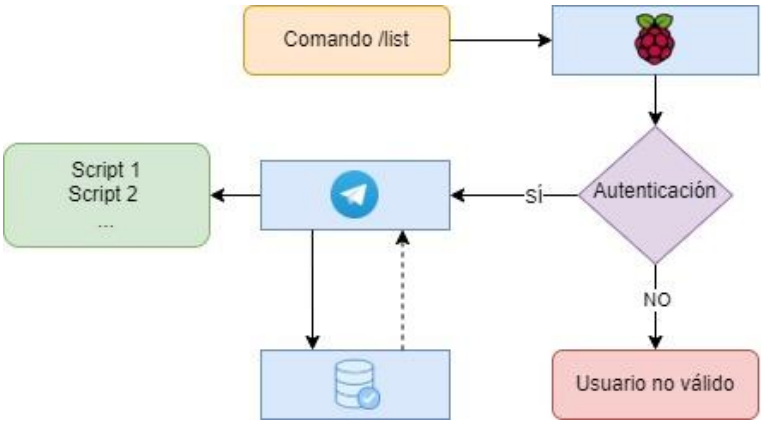


Ilustración 27. Diagrama de flujo del comando "/list"

El comando más complejo en cuanto a lógica y funcionamiento es el comando "/rename", que como su nombre indica, nos permite renombrar los scripts que el usuario tiene disponibles. En este comando el BOT solicita al usuario qué script de la lista quiere renombrar, y le pide que indique el nuevo nombre de este. Una vez recibe la información del

nuevo nombre, lo registra en la base de datos y está disponible para visualizar en el comando "/list".

Este proceso de renombrar es algo complejo y se divide en dos fases, ya que realmente el usuario intercambia información 2 veces con el BOT, una para indicarle qué script hay que renombrar, y después el nuevo nombre. En la primera fase, el usuario utiliza el comando "/rename" para indicar al BOT que quiere cambiar el nombre de un script. La aplicación capta este mensaje y lo interpreta como un mensaje de texto plano que el usuario envía al BOT, y comienza a filtrarlo. Existen 2 posibilidades en este tipo de mensajes, la primera es que se trate de un comando como es este caso, y la segunda que se trate de texto que ha escrito el usuario sin intención de ejecutar una función del BOT. El manejo de ambos es distinto, para determinar si se trata de un comando se compara el texto introducido por el usuario con la lista de comandos disponibles por el BOT y se ejecuta una función de callBack correspondiente a cada comando, mientras que si no se trata de ningún comando, el BOT envía un mensaje explicando al usuario que no comprende lo que le está diciendo, y recomendando que utilice el comando "/help".

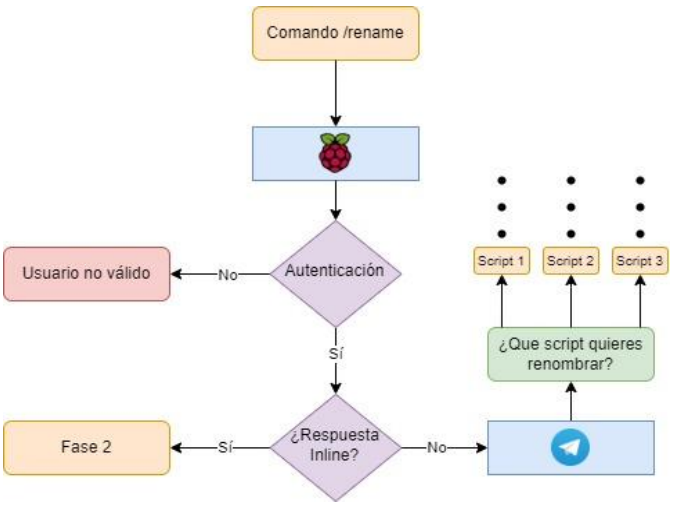


Ilustración 28. Diagrama de flujo de la primera fase del comando "/rename"

Si el BOT verifica que el usuario tiene permisos de autenticación y ha enviado el comando "/rename", envía una serie de diez InlineButtons, cada uno correspondiente a un Script junto con un mensaje pidiendo al usuario que elija cual es el Script que desea renombrar. Cuando el usuario elige un script, se continúa de nuevo con la autenticación de este y se ejecuta la función renameHandler, que corresponde con la fase 2 del proceso de renombrado.

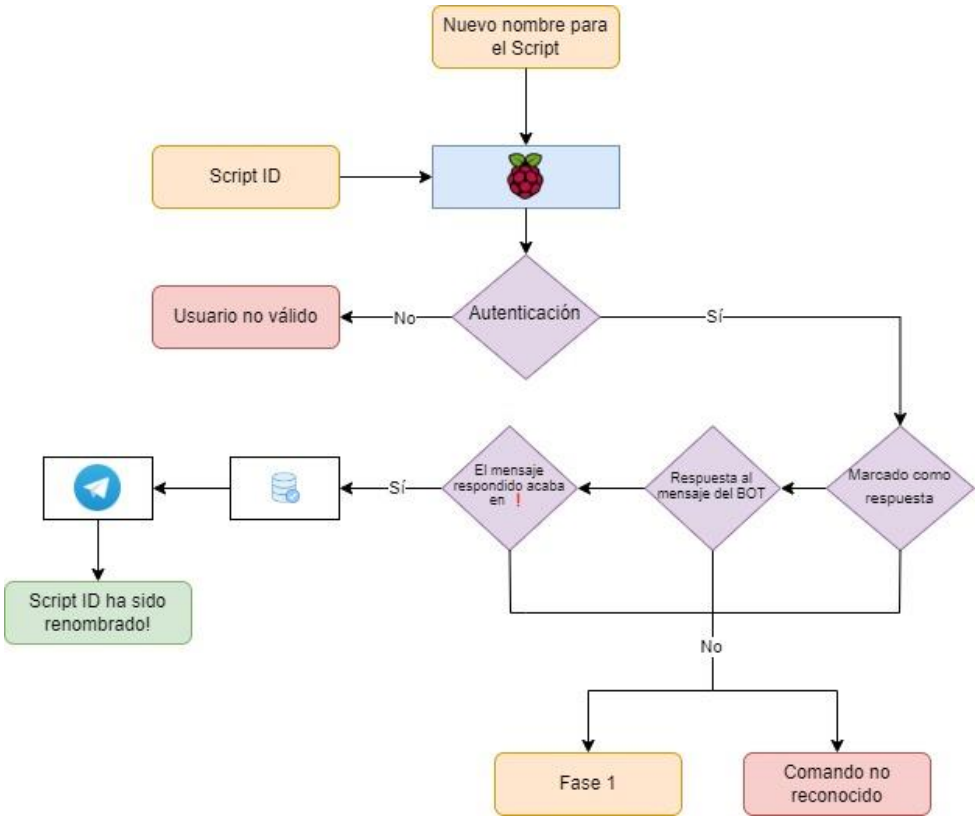


Ilustración 29. Diagrama de flujo de la segunda fase del comando "/rename"

El usuario pulsa el botón correspondiente al Script que desea renombrar, y la aplicación emite un mensaje en el que explica al usuario que, usando la opción "Responder al mensaje", responda al propio mensaje indicado el nombre que se desea asignar al script. Esto es necesario para el filtrado del mensaje en la lógica del BOT, que rápidamente puede filtrar los mensajes del usuario, encontrando la respuesta a su anterior mensaje, y ejecutando una query en la base datos para actualizar el campo nombre del script indicado. Si no se siguen los pasos indicados, se devolverá un mensaje de error y se volverá a comenzar.

Por último, tenemos dos comandos más con una funcionalidad y lógica baste parecidas al anterior explicado, pero algo más sencillos de implementar. Estamos hablando de los comandos "/security" y "/delete". El primero de ellos nos permite intercambiar el modo de ejecución que tiene un script con respecto a su seguridad, invirtiendo el valor de la etiqueta "secExec" del script seleccionado en la base de datos. Por otro lado, tenemos el comando delete, cuya función es únicamente reestablecer una asignación que hemos hecho a uno de los diez comandos disponibles en el proyecto, estableciendo el valor de la etiqueta "init" a false, activando la ejecución segura del script, que por defecto siempre debe estar activa, y por último reestableciendo el nombre que le hemos asignado al Script al nombre inicial de este cuando no estaba inicializado.

Estos dos comandos mantienen una estrecha relación con el comando para renombrar, ya que también son acciones directas sobre la lista de comandos, por lo que están directamente relacionados con la actualización de la base de datos donde estos scripts y su información se encuentran. El flujo de funcionamiento es muy parecido al anterior comando, en primer lugar, el usuario debe escribir el comando que desee en el chat, tanto "/security" como "/delete".

Enseguida, el BOT le preguntará, al igual que hacía con el comando de rename cuál es el Script sobre el que se quiere ejecutar el comando indicado, todo esto mediante una serie de diez Inline Buttons, tal y como funciona también en rename, y cuando el usuario seleccione uno de ellos, la aplicación mediante un filtrado de estos mensajes ejecuta la lógica pertinente sobre los datos del Script en la base de datos.

Los métodos encargados de ejecutar la lógica sobre los scripts son "securityHandler" y "deleteHandler" respectivamente. Estos son métodos sencillos encargados de realizar consultas de UPDATE a la base de datos y realizar las pequeñas modificaciones requeridas por el comando seleccionado. En el primer comando, simplemente se ejecutará una sentencia de UPDATE en la que el valor de "secList "pasa a ser "!secList", invirtiendo así su valor independientemente del modo de ejecución que tenía el script en el momento de la ejecución del comando, y en el segundo comando simplemente la sentencia se encarga de establecer los valores de "init", "name" y "secExec" a los valores por defecto.

Una vez está clara toda la comunicación que establece nuestro servidor con el usuario mediante Telegram, pasamos a la última fase, en la que el servidor se comunicará con el dispositivo Windows para la ejecución de los comandos.

Fase 3. Capa de servicio, comunicación con ordenador Windows

Esta es la última fase de la comunicación, y corresponde con la petición por parte del servidor para la ejecución del script final en el dispositivo Windows del usuario.

Esta comunicación es la más sencilla, y viene precedida por todas las verificaciones de seguridad y autenticación entre el cliente y el servidor. Aquí entra en juego la capa de servicio, donde definimos la función encargada de enviar al dispositivo Windows la autorización final para la ejecución. Para ello utiliza el método scriptExecution, que recibe como parámetro el script a ejecutar y lanza una petición HTTP POST sobre el dispositivo final, con la información del Script a ejecutar. Debemos tener en cuenta aquí también un componente de seguridad, para asegurarnos que el servidor es el único que puede enviar peticiones de ejecución válidas al ordenador.

Cumple un papel muy importante en esta tercera fase el framework Spring Security con su consecuente implementación en la capa de seguridad de la aplicación, en la que definimos una clase de configuración donde indicamos a la aplicación que solo puede comunicarse con la dirección IP correspondiente con el terminal Windows del usuario. La comunicación con el ordenador se realiza mediante cifrado TLS, accediendo a la dirección de este utilizando el protocolo HTTPS.

5.2.3 Alexa Windows Service

Por último, tenemos la aplicación Alexa Windows Service, una sencilla aplicación backend diseñada en Java y reconvertida a servicio de Windows una vez compilada.

Esta aplicación tiene como funcionalidad el comunicarse únicamente con nuestro servidor Raspberry Pi Home Server para coordinar la ejecución de scripts en el dispositivo final. Para el diseño de este componente, hemos primado la transparencia de cara al usuario durante la ejecución de sus procesos o funcionalidades, y establecer una seguridad robusta para evitar

que terceros puedan hacer uso de este servicio para la ejecución indebida de comandos en el terminal.

La aplicación se desarrolla enteramente en Java, utilizando el framework de Spring para asegurarnos de un modelado robusto de esta. Solo existe una capa de desarrollo en la aplicación, la llamada capa controladora, ocupada de la recepción de mensajes HTTP de nuestro servidor. Para ello hemos utilizado las anotaciones que nos proporciona Spring, tal y como hemos hecho en nuestra anterior aplicación, en este caso "@PostMapping" indicando la dirección "/api" en la URL donde se recibirán las peticiones pertinentes, y requiriendo un parámetro de entrada llamado "scriptName", correspondiente con el nombre del script que se desee ejecutar.

El método controlador de las peticiones capta las peticiones POST que realiza el servidor, y extrae el parámetro con el nombre del script a ejecutar. Con esta información, la aplicación crea un objeto Runtime, mediante el cual podemos crear un proceso Windows que logre ejecutar un comando de consola. El método exec() del objeto Runtime devuelve un objeto process, y acepta un parámetro String que será el comando que se ejecutará en el ordenador.

El comando que se lanza es la ejecución de un archivo batch localizado en una carpeta de scripts del sistema, en la que hay diez de estos archivos con nombres "scriptName".bat, donde "scriptName" corresponde con los números del uno al diez. La aplicación simplemente tiene que acceder a la carpeta donde se encuentran los scripts, y ejecutar aquel cuyo nombre coincida con el parámetro scriptName que recibe desde el servidor.

Lo más interesante de esta aplicación web es su instalación como servicio para el sistema operativo, el cual se encuentra corriendo continuamente en segundo plano como cualquier otro servicio que ofrece Windows. Este punto es esencial, teniendo en cuenta que esta transparencia para el usuario final es uno de los objetivos del proyecto, es lógica la idea de implementar esta aplicación como servicio, para que cada vez que el usuario encienda el terminal, automáticamente pueda realizar ejecuciones mediante voz sin necesidad de iniciar y configurar una aplicación Java, que para algunos usuarios con menos conocimientos en el lenguaje o incluso en el manejo de estos equipos, puede convertirse en un gran impedimento para la utilización del proyecto.

Por esta razón, se ha implementado esta aplicación web Java como un servicio Windows utilizando una herramienta software llamada WinSW.NET2. Esta herramienta es un software de código abierto [16] encargado de encapsular ejecutables de Windows como servicios. Funciona tanto con archivos EXE como con archivos JAR, lo cual nos beneficia enormemente para nuestro proyecto Java.

Para generar un archivo JAR desde el proyecto Java, este ha de compilarse mediante Maven. Utilizando los comandos Maven Clean y Maven Install, el mismo Maven compila el código y nos devuelve el compilado en formato JAR.

El software WinSW funciona nivel de consola de Windows, con una serie de sencillos comandos. Tenemos el comando "install" y "uninstall" para, como su nombre indica, instalar o desinstalar un ejecutable como servicio. También entre otros tenemos los comandos de lanzar, detener o reiniciar el servicio, y otro para ver el estado de este.

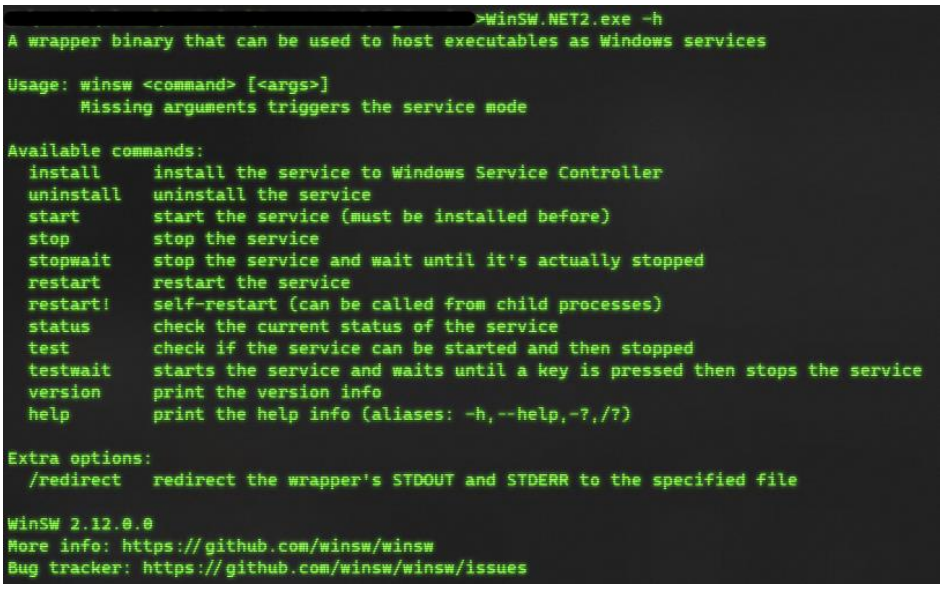


Ilustración 30. Funcionamiento de la herramienta WinSW.NET2

Para la configuración del encapsulado del servicio, se proporciona al programa un documento XML indicando el nombre que se va a asignar al servicio, una pequeña descripción de este y la ruta del ejecutable a encapsular.

Una vez creado correctamente el archivo de configuración, debe ser guardado bajo el nombre "WinSW.NET2.xml", y ya podremos utilizar la herramienta principal. Mediante la ejecución del comando "WinSW.NET2.exe install", nuestro ejecutable se convierte en servicio instantáneamente. La propia herramienta se encarga de gestionar la generación de LOGS de nuestra aplicación, y los almacena en un archivo predeterminado.

Desde este momento, tras la instalación del servicio, el terminal Windows iniciará automáticamente esta aplicación necesaria para la ejecución de comandos mediante los comandos de voz del Echo Dot.

5.3 Despliegue e implementación del proyecto

Una vez explicados en profundidad los dos componentes principales para llevar a cabo el proyecto, vamos a hablar un poco de la implementación final y las relaciones que se definen entre el hardware y el software, el despliegue de los tres proyectos y un poco la experiencia del usuario que se desea lograr, buscando siempre la seguridad y la transparencia para toda persona que quiera utilizar al completo esta herramienta en su red de área local.

Para el despliegue de la herramienta completa, deberemos comenzar desplegando las tres aplicaciones en sus respectivos dispositivos hardware contenedores. Para comenzar deberemos tener lógicamente el dispositivo Echo Dot instalado y configurado para conectarse a la red local. Para la configuración del dispositivo de Amazon, debemos instalar en nuestro Smartphone la aplicación "Amazon Alexa" y buscar dispositivos cercanos mediante Bluetooth, con nuestro Echo Dot en modo configuración, y una vez la aplicación detecte el Echo, nos dejará configurarlo a nuestra elección. En el menú de configuración, seleccionaremos la red local en la que desplegaremos el proyecto, y continuamos con el resto de los componentes.

Ilustración 31. Control de dispositivos desde la app de Alexa

Para continuar, realizaremos el despliegue del proyecto ESP Hub del microprocesador ESP-8266. Para ello, podemos utilizar cualquier tipo de ordenador o terminal, sin embargo, yo utilizaré un ordenador Windows debido a que la Raspberry Pi Zero W estará funcionando en modo "headless", controlada por SSH desde este mismo ordenador, y no tendrá interfaz visual, lo cual es interesante para facilitar el proceso. La aplicación ESP Hub viene preparada para que nuestro Echo Dot identifique diez dispositivos al lanzar la aplicación, los cuales más adelante actuaran como Scripts ejecutables para el usuario, por lo que tan solo habrá que configurar el nombre de la red a la que debe conectarse el dispositivo, indicando el SSID y la contraseña. Para la compilación se utilizará la extensión PlatformIO de Visual Studio Code que se encarga de cargar el binario ya compilado en el dispositivo automáticamente.

Una vez el binario se encuentra en el microcontrolador, este puede conectarse a cualquier fuente de energía mediante un micro-USB tipo B, que por comodidad se recomienda que sea la placa Raspberry Pi Zero W para ahorrar espacio y tener ambas placas juntas. En cuanto el micro es alimentado, este ejecuta el binario compilado que tiene en memoria, conectándose a la red que le hemos especificado, y en instantes está listo para comunicarse con nuestro dispositivo Echo. Gracias a esto, en apenas unos segundos tras el encendido del ESP-8266, si le decimos a Alexa que, si puede descubrir nuevos dispositivos, esta automáticamente comenzará el proceso de descubrimiento de dispositivos con el protocolo SSDP, al que nuestro micro responderá identificándose como un Philips Hue Bridge, y prestándose a controlar los dispositivos a su alcance.

Ya desplegada esta primera aplicación software, comenzamos con la configuración y despliegue del segundo software, la aplicación Raspberry Pi Home Server. En cuanto a hardware, primero debemos configurar la Raspberry Pi para utilizarla por SSH, añadiendo un archivo de

configuración donde indicamos el SSID y contraseña de la red para que la placa se conecte a esta y podamos configurarla. Una vez podamos acceder mediante SSH, utilizaremos el protocolo FTP para enviar desde nuestro terminal Windows la aplicación ya compilada como archivo JAR a nuestra Raspberry. Para el despliegue del binario compilado de la aplicación utilizaremos Tomcat ya que es el servidor con el que más familiarizado estoy, pero podría utilizarse cualquier otro, como NGINX, Apache, etc. Esta aplicación, una vez desplegada, activa el BOT de Telegram para el control de la aplicación. Sin embargo, el BOT necesita una conexión activa a una base de datos para la autenticación del usuario cada vez que este se comunica con él, ya que por propia arquitectura de aplicación todo tipo de comunicación que recibe el BOT es autenticada medicante la verificación del ID del usuario que ha enviado el mensaje.

El usuario deberá desplegar la base de datos con las dos tablas necesarias para la aplicación, una que incluye los usuarios con permisos para comunicarse con el BOT, y por otro lado la que contenga la información necesaria de los Scripts disponibles.

Otro paso importante para que el despliegue funcione correctamente es la configuración IP de los dispositivos implicados dentro de la red local, ya que toda la comunicación del proyecto ocurre en la LAN y es necesario indicar a la aplicación las IP de todo el hardware necesario. Para establecer estas IP y otros datos necesarios como pueden ser las credenciales de acceso a la base de datos se guardan en un archivo de propiedades del proyecto, generado por el propio framework de Spring Boot y llamado “application.properties”.

Ilustración 32. Archivo de configuración application.properties

En la *Ilustración 32* se observa el archivo de properties de la aplicación de Raspberry Pi Home Server que es similar a la del servicio de Windows al utilizar ambas Spring Boot. Es importante en la configuración del router añadir entradas estáticas para asignar las IP manualmente a estos tres dispositivos.

Una vez realizada la configuración de las variables necesarias para el despliegue del proyecto para que este realice sus conexiones pertinentes, el BOT ya funciona correctamente. Para comenzar a utilizarlo, le podemos pedir a Alexa que ejecute algún Script refiriéndonos a él por su nombre por defecto como, por ejemplo, “Alexa, ejecuta el script 1”. Inmediatamente, el BOT de Telegram nos enviará un mensaje de confirmación para la ejecución del Script, aunque la ejecución no será todavía funcional, por la no configuración del servicio Windows final.

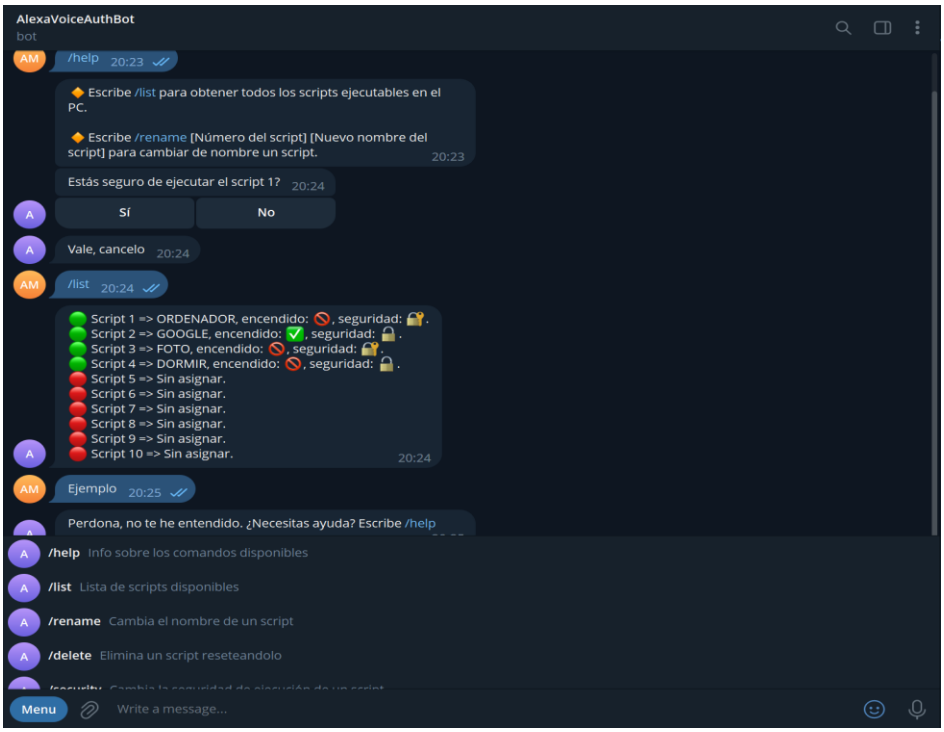


Ilustración 33. Ejemplo de funcionalidades del BOT

En esta imagen podemos ver cómo se ve la aplicación inmediatamente después de iniciar y configurar el servidor controlador de nuestro BOT y la base de datos. Observamos en primer lugar la ejecución del comando "/help", un intento de ejecución del Script número 1 el cual cancela el usuario, el comando "/list" que nos muestra el estado de los Scripts disponibles para el usuario, obteniendo los datos de la base de datos, y por último un ejemplo del manejo de errores de la aplicación cuando el usuario envía un mensaje inesperado para la lógica del funcionamiento del programa. En este caso, el usuario envía un mensaje inesperado para el servidor con el texto "Ejemplo", a lo que el servidor responde con una cadena de texto por defecto, definida por la aplicación a la hora de manejar errores, que se muestra al usuario cuando este envía mensajes inesperados. El texto de respuesta del BOT indica al usuario que ha enviado un mensaje inesperado, y que no es capaz de comprender la intención del mensaje, aconsejándole que ejecute el comando de ayuda para ver qué opciones proporciona el BOT. Existe gran cantidad de errores contemplados para su manejo, y cada uno se trata de una manera distinta, informándose al usuario siempre que algo vaya mal.

Por último, tenemos la configuración del ordenador Windows donde se ejecutarán los comandos que el usuario desee. Para ello habrá que modificar el fichero de configuración "application.properties" de la aplicación Java. Una vez establecidos los valores necesarios, como las IP del hardware requerido para las comunicaciones y el puerto donde escucha la aplicación peticiones por parte del servidor, esta aplicación tiene un parámetro configurable adicional a la otra aplicación de nuestro servidor, y es la ruta de la carpeta con todos los Scripts disponibles para ejecutar. Una vez compilado el programa, instalamos el ejecutable como servicio en Windows y este ya estará listo para ejecutarse automáticamente siempre que el dispositivo esté encendido.

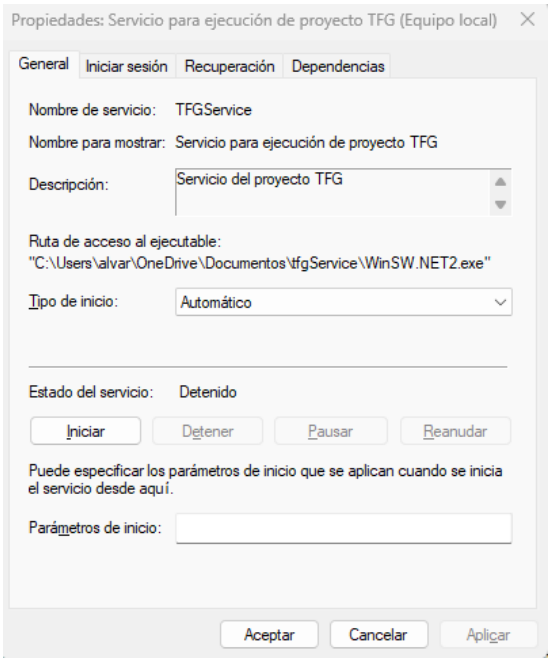


Ilustración 34. Propiedades del servicio Windows TFGService

Para la ejecución de los Scripts, simplemente hay que añadir a la carpeta de Scripts, cuya ruta hemos definido antes de compilar la aplicación, los archivos de comandos de Windows que se deseen, lo que le da al usuario libertad total para hacer cualquier cosa con su ordenador mediante comandos de voz.

Capítulo 6. Conclusiones

El presente proyecto nace a raíz de una idea personal con el objetivo de lograr un sistema de comunicación local y segura entre dispositivos de voz Alexa y ordenadores Windows para la ejecución de comandos por el usuario.

Durante todo el desarrollo se ha abordado la comodidad, seguridad y accesibilidad como los objetivos básicos a lograr.

En primer lugar, se ha adecuado todo el proceso de ejecución de comandos para un público genérico, con una instalación sencilla de toda la herramienta en una red de área local, tanto del hardware como del software.

En cuanto a la seguridad, la comunicación entre los dispositivos es local y cifrada, debido al peligro intrínseco que conlleva la ejecución remota de comandos, a la vez que accesible para su uso por personas con distintos tipos de discapacidad.

Además de los objetivos básicos a lograr, existen una serie de hitos secundarios propuestos en las fases iniciales de la idea. Estos hitos han sido implementados satisfactoriamente en la herramienta desarrollada.

- Comunicación completamente local
- Comunicación cifrada SSL
- Implementación de un BOT de Telegram para control de peticiones de ejecución
- Instalación de aplicación Java como servicio de Windows para su ejecución automática

Tras lo expuesto anteriormente, se puede concluir que los objetivos mostrados en el presente proyecto han sido abordados con éxito por lo que, a pesar de haber supuesto un largo y extenso periodo de desarrollo e implementación de la idea, considero que se han logrado los objetivos propuestos al inicio de este proyecto, diseñando finalmente una solución funcional, accesible y segura.

Adicionalmente, me gustaría añadir que el proyecto tiene un gran margen de mejora para posibles despliegues en entornos de producción. En este estado, el despliegue del proyecto demanda de conocimientos básicos de redes y de hardware, sin embargo, en un desarrollo posterior de esta se intentaría lograr una instalación en la que el usuario o propietario no necesitara estos conocimientos anteriormente nombrados.

Capítulo 7. Listado de figuras

Capítulo 8. Bibliografía

[1] X. Lei, G.-H. Tu, A. X. Liu, K. Ali, C.-Y. Li, y T. Xie, «The Insecurity of Home Digital Voice Assistants -- Amazon Alexa as a Case Study». arXiv, 12 de noviembre de 2019. Disponible en: http://arxiv.org/abs/1712.03327

[2] Amazon, «Create the Interaction Model for Your Skill» [En línea]. Available: https://developer.amazon.com/es-ES/docs/alexa/custom-skills/create-the-interaction-model-for-your-skill.html.

[3] Amazon Developer, «Skill development workflow» [En línea]. Available: https://developer.amazon.com/en-US/docs/alexa/ask-overviews/what-is-the-alexa-skills-kit.html.

[4] Amazon AWS, «AWS | Lambda - Gestión de recursos informáticos» [En línea]. Available: https://aws.amazon.com/es/lambda/.

[5] Alexa Blogs, «Rapidly Create Your Alexa Skill Backend with AWS CloudFormation» [En línea]. Available: https://developer.amazon.com/es/blogs/alexa/post/Tx27NAUCY0KQ34D/rapidly-create-your-alexa-skill-backend-with-aws-cloudformation.

[6] C. Lentzsch, S. J. Shah, B. Andow, M. Degeling, A. Das, y W. Enck, «Hey Alexa, is this Skill Safe?: Taking a Closer Look at the Alexa Skill Ecosystem», en *Proceedings 2021 Network and Distributed System Security Symposium*, Virtual: Internet Society, 2021. doi: 10.14722/ndss.2021.23111.

[7] «Sustaining Complement Quality for Digital Product Platforms: A Case Study of the Philips Hue Ecosystem», doi: 10.1111/jpim.12555.

[8] «Amazon.com: Certified for Humans: Dispositivos Amazon y Accesorios». https://www.amazon.com/b?node=19982322011&ref_=ods_surl_cfh .

[9] «Spring Initializr», *Spring Initializr*. https://start.spring.io .

[10] «17. Web MVC framework». https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html .

[11] T. Dey, «A Comparative Analysis on Modeling and Implementing with MVC Architecture», 2011.

[12] «5. The IoC container». https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/beans.html .

[13] «Workaround for Alexa Bug · Issue #4235 · esp8266/Arduino · GitHub». https://github.com/esp8266/Arduino/issues/4235 .

[14] C. Schwinne, «Aircoookie/Espalexa». 16 de abril de 2023. [En línea]. Disponible en: https://github.com/Aircoookie/Espalexa

[15] R. Bermudez, «Telegram Bot Java Library». 26 de abril de 2023. [En línea]. Disponible en: https://github.com/rubenlagus/TelegramBots

[16] «GitHub - winsw/winsw: A wrapper executable that can run any executable as a Windows service, in a permissive license.» https://github.com/winsw/winsw .