Escuela
Politécnica
Superior

Manual de compilación e instalación de las interfaces desarrolladas

Grado en Ingeniería Robótica

Trabajo Fin de Grado

Autor:
Pilar Navarro García
Tutor/es:
Sergio Luján Mora
Santiago Meliá Beigbeder

Enero 2022

Universitat d'Alacant
Universidad de Alicante

Manual de compilación e instalación de las interfaces desarrolladas

Autor
Pilar Navarro García

Tutor/es
Sergio Luján Mora
LENGUAJES Y SISTEMAS INFORMÁTICOS
Santiago Meliá Beigbeder
LENGUAJES Y SISTEMAS INFORMÁTICOS

Grado en Ingeniería Robótica

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Enero 2022

Índice general

Índice de figuras

1 Introducción

Este es un manual de compilación e instalación de las dos interfaces desarrolladas para el TFG "Desarrollo de interfaces accesibles para una smart home utilizando AWS y Amazon Alexa". Comprende el hardware necesario, la configuración requerida en AWS y todo lo relativo al código desarrollado para cada una de las interfaces.

El documento comienza exponiendo los materiales hardware necesarios, así como el montaje necesario para su correcto funcionamiento.

A continuación se encuentra la configuración de AWS requerida para poder hacer uso de ambas interfaces, explicada paso a paso.

Seguidamente se detalla todo lo relativo a la interfaz visual con BlueDisplay: Una breve explicación de las librerías utilizadas y cómo instalarlas, la distribución del código fuente implementado para la Interfaz Gráfica de Usuario (GUI) con una breve descripción de cada uno de los archivos, una pequeña guía paso a paso de cómo compilar y cargar estos archivos a los microcontroladores, y un sencillo manual de usuario con una breve descripción de cada menú implementado en la interfaz.

Finalmente, se expone todo lo relativo a la interfaz de voz con Amazon Alexa: Una guía paso a paso de cómo crear y configurar una skill de Alexa, así como los intents, slots y utterances empleados.

2 Hardware

2.1 ESP32 con el NodeMCU conectado

Al conectar el ESP32 y el NodeMCU siguiendo el esquema de la Figura 2.1[1] es posible la comunicación serie bidireccional entre ambos dispositivos. Se conectan los pines RX2 y TX2 del ESP32 con los pines D7 y D8 del NodeMCU, es decir, el receptor del primero con el transmisor del segundo y viceversa. También debe interconectarse un pin de tierra o GND de cada dispositivo para reducir los fallos y el ruido en la comunicación.

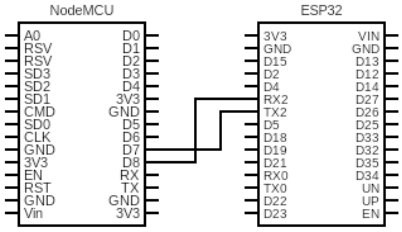


Figura 2.1: Diagrama de la conexión serie del ESP32 con un NodeMCU. Fuente: Elaboración propia.

La Figura 2.2 corresponde a una imagen real de las conexiones citadas anteriormente.

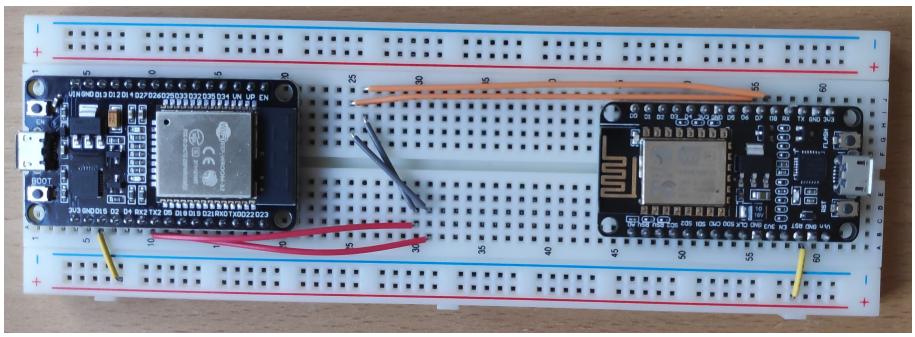


Figura 2.2: Imagen real de la conexión serie del ESP32 con un NodeMCU. Fuente: Elaboración propia.

[1] Este esquema se ha realizado a través del *Circuit Diagram Web Editor*, en `https://www.circuit-diagram.org/editor/`.

2.2 NodeMCU con los dispositivos conectados

Los dispositivos que pueden controlarse están conectados a otro NodeMCU como en la Figura 2.4, siguiendo el esquema eléctrico de la Figura 2.3. Estos dispositivos son:

- Tres leds, que simulan tres bombillas: pasillo, cocina y habitación, conectados a los pines D5, D6 y D7, respectivamente.
- Un sensor DHT11, que mide temperatura y humedad, conectado al pin D1.
- Una fotorresistencia o LDR, que mide la intensidad de luz que incide sobre ella, conectada al pin A0, ya que su lectura es un valor analógico a diferencia de los anteriores que requieren de pines digitales.

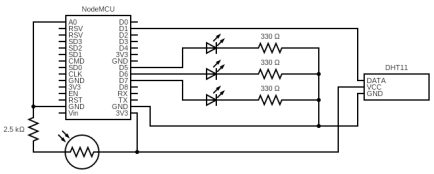


Figura 2.3: Diagrama de los dispositivos físicos empleados con un NodeMCU. Fuente: Elaboración propia.

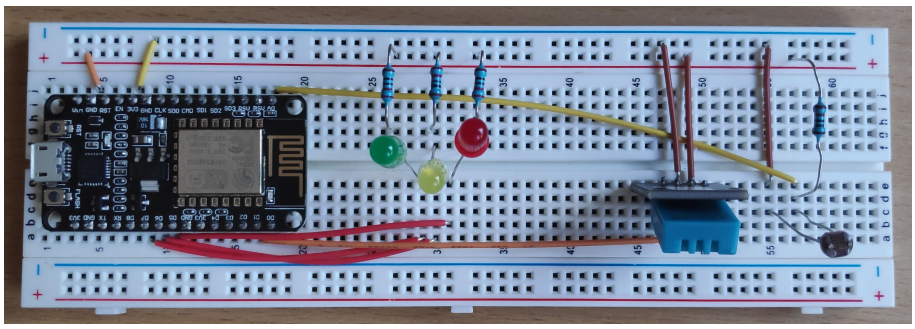


Figura 2.4: Imagen real de los dispositivos físicos empleados con un NodeMCU. Fuente: Elaboración propia.

3 Configuración de Amazon Web Services

3.1 Amazon Web Services IoT Core

Con el fin de permitir la conexión de los dispositivos físicos comentados en el Capítulo 2, es necesario configurar los certificados y las políticas para conceder al objeto de AWS IoT Core y a los NodeMCUs conectados a la nube de AWS los permisos necesarios para que puedan realizar las acciones necesarias. Los pasos a seguir son los siguientes:

1. Crear política en AWS IoT Core > Seguridad > Políticas con las acciones permitidas y denegadas necesarias.
2. Ir a AWS IoT > Administración > Objetos y seleccionar "Crear objetos".
3. Elegir "Crear un único objeto".
4. Establecer el nombre del objeto, en este caso *NodeMCU-with-devices* y asignarle una sombra sin nombre (clásica).
5. Seleccionar la opción de "Generar automáticamente un certificado nuevo (recomendado)".
6. Asociar la política definida.
7. Descargar los documentos generados para instalarlos en el dispositivo. Estos documentos son cuatro: El certificado del dispositivo, la clave pública, la clave privada y el punto de enlace de servicios de confianza de Amazon. Una vez hecho esto, ya estará creado el objeto o *thing*.
8. Activar el certificado en Seguridad > Certificados.

Los certificados descargados se utilizan por los NodeMCU para conectarse a la nube de AWS, ya que necesitan los permisos definidos en estos.

3.2 Amazon Web Services Identity and Access Management

Con la finalidad de dotar de los permisos necesarios a la función Lambda a desarrollar, debe crearse una política a través del servicio AWS Identity and Access Management (IAM) que defina estos permisos para asociarla un rol de IAM y, posteriormente, vincular este rol a la función de Lambda. Este proceso sigue los siguientes pasos:

1. Crear una política en AWS IAM en la que se establecen los permisos de publicación y suscripción del servicio de AWS IoT para los topics utilizados.

2. Crear un rol en AWS IAM y vincular la política creada en el paso anterior.

3. Asociar el rol creado a la función Lambda. Este paso se puede realizar en el proceso de creación de la función Lambda como se explica en el siguiente apartado o, si la función Lambda ha sido creada previamente, editando el rol de ejecución de esta en la consola de la función Lambda > Configuración > Permisos.

3.3 Amazon Web Services Lambda

La función Lambda implementada se encarga de ejecutar las peticiones realizadas a través de Alexa y construir el mensaje que Alexa debe devolver al usuario para terminar la conversación. Para crear una función Lambda se selecciona "Crear una función" desde el panel de funciones del servicio de AWS Lambda (Figura 3.1) y se siguen los siguientes pasos:

1. Se establece el nombre de la función, que aparecerá en el ARN de la misma.

2. Se selecciona el lenguaje de programación a utilizar para la programación de la función. Se ha elegido Python 3.9.

3. Se elige la arquitectura del conjunto de instrucciones para el código de la función. Se ha seleccionado x86_64.

4. Se elige el rol de ejecución entre: Creación de un nuevo rol con permisos básicos de Lambda, uso de un rol existente o creación de un nuevo rol desde la política de AWS templates. En este caso se utiliza un rol existente, previamente creado en AWS IAM, para dar acceso a todos los servicios que se van a utilizar para tener los permisos necesarios para realizar las acciones demandadas.

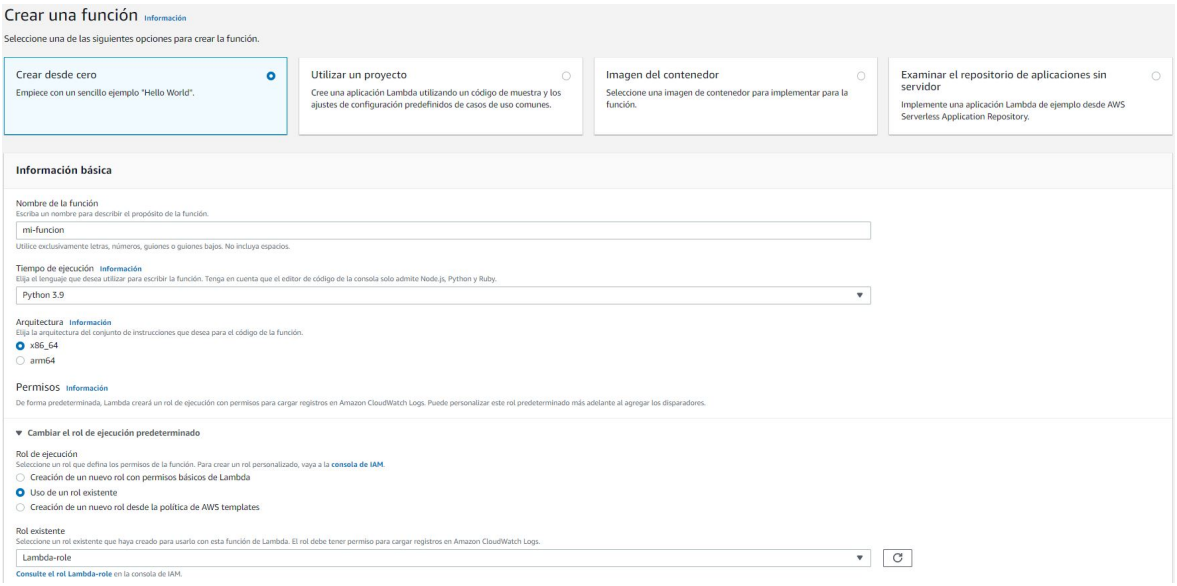


Figura 3.1: Proceso de creación de una función en AWS Lambda. Fuente: Elaboración propia.

Para que la función Lambda creada se ejecute cuando se inicie la skill de Alexa desarrollada, es necesario agregar "Alexa Skills Kit" como un desencadenador de la función Lambda. En la Figura 3.2 se puede observar la información general de la función Lambda, con el desencadenador de Alexa insertado en la parte izquierda (en el capítulo 5 se indicará qué es este desencadenador) y el ARN de la función en la derecha. Este ARN se copiará posteriormente en la configuración de la skill de Alexa.

Figura 3.2: Información general de la función Lambda desarrollada. Fuente: Elaboración propia.

Se han implementado cuatro scripts para la función Lambda creada, cuyo funcionamiento se detalla a continuación:

lambda_function.py Script ejecutado inicialmente al ejecutarse la skill de Alexa. Ejecuta un método de *actions.py* dependiendo del tipo de petición obtenido a través de Alexa.

actions.py Comprueba el intent ejecutado y realiza las acciones solicitadas según este intent y los slots obtenidos. Si es necesario acceder a la sombra del dispositivo físico para leer sus valores actuales o actualizarlos, llama a la función correspondiente implementada en *shadow_connection.py*. Finalmente, llama al método de *response_builders.py* necesario para construir el mensaje que debe contestar Alexa al usuario.

shadow_connection.py Tiene las funciones utilizadas para conectarse a AWS IoT Core y acceder a la sombra del dispositivo físico, ya sea para leer sus valores actuales o para actualizarlos.

response_builders.py Posee los métodos que construyen el mensaje de respuesta al usuario, uno por cada intent de la skill de Alexa.

Para poder realizar una petición o actualización de los atributos de la sombra del dispositivo empleado se siguen los siguientes pasos:

1. Se establece el cliente necesario haciendo uso del Software Development Kit (SDK) de Python para AWS de la siguiente manera:

```
client = boto3.client('iot-data', region_name = 'eu-central-1')
```

2. Para actualizar la sombra del dispositivo se publica el mensaje, en formato JSON, con los nuevos valores de los atributos de esta en el topic correspondiente, donde "updated" es dicho mensaje.

```
client.publish(topic = '$aws/things/NodeMCU-with-devices/shadow/update', qos = 1, payload = ↩
    ↪ updated)
```

3. Para leer los valores actuales de la sombra del dispositivo se obtiene el documento de esta con:

```
shadow = client.get_thing_shadow(thingName = thing_name)
```

   Donde "shadow" es el documento de la sombra del dispositivo en formato JSON y "thing_name" es el nombre del thing o dispositivo, en este caso *NodeMCU-with-devices*.

La respuesta que Alexa devuelve al usuario se define en formato JSON, construida a través de los métodos desarrollados en *response_builders.py* de la función Lambda. Este documento JSON tiene la siguiente estructura:

```
'version': '1.0',
'sessionAttributes': attributes
'response': {
  'outputSpeech': {
    'type': 'PlainText',
    'text': output
  },
  'card': {
    'type': 'Simple',
    'text': "SessionSpeechlet - " + title,
    'content': "SessionSpeechlet - " + output
  },
  'reprompt': {
    'outputSpeech':{
      'type': 'PlainText',
      'text': reprompt
    }
  },
  'should_end_session': should_end_session
}
```

Donde las variables empleadas para rellenar los campos necesarios para la construcción de la respuesta son los siguientes:

- *attributes*: Variables o valores que almacena Alexa mientras la sesión esté activa. Para la finalidad de este trabajo no es necesario emplear estas variables, por lo que se utilizan corchetes vacíos ().
- *output*: Respuesta que dice Alexa. En dispositivos Alexa con pantalla, texto escrito en ella.
- *title*: En dispositivos Alexa con pantalla, título que aparece en grande antes del *output*.
- *reprompt*: Texto que dice Alexa cuando esperaba una respuesta del usuario pero no la ha obtenido en un tiempo determinado o no la ha entendido.

- *should_end_session*: Indica si debe terminar la sesión tras enviar la respuesta de Alexa. Si su valor es *true*, Alexa no esperará respuesta del usuario. En cambio, si su valor es *false*, Alexa esperará unos segundos a que el usuario dé una respuesta.

4 BlueDisplay

Todo el código implementado para esta interfaz se encuentra en un repositorio de GitHub, en el siguiente enlace `https://github.com/png10/TFG-de-Pilar-Navarro-Garcia/tree/main/Interfaz%20visual`.

4.1 Librerías necesarias

Con la finalidad de cargar los archivos implementados en los microcontroladores, es necesario instalar las librerías que han sido utilizadas en el IDE de Arduino. Esta instalación se realiza desde el propio IDE en la opción Programa > Incluir Librería > Administrar Bibliotecas... (véase Figura 4.1).

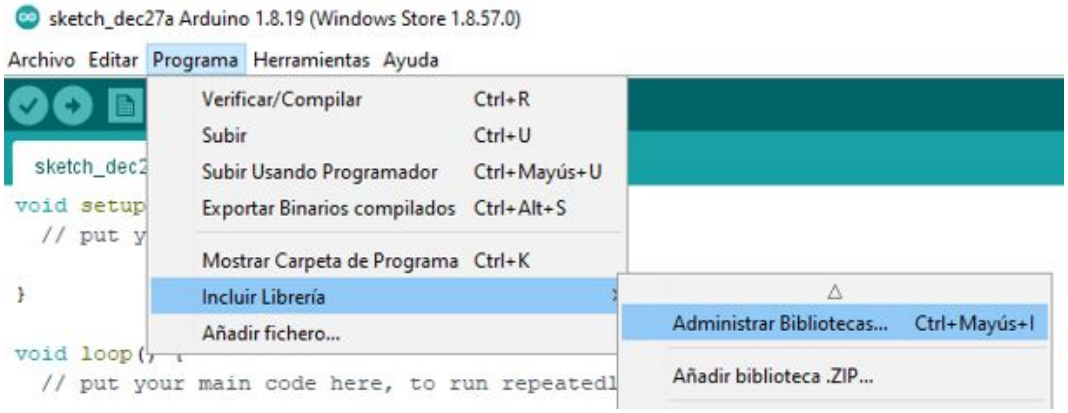


Figura 4.1: Programa > Incluir Librería > Administrar Bibliotecas... en el IDE de Arduino. Fuente: Elaboración propia.

A continuación se exponen las librerías utilizadas a lo largo de todos los archivos implementados que han requerido de la instalación previa de las librerías que las contienen, así como su uso y el nombre de la librería a instalar.

- *ArduinoJson.h*: Permite convertir una cadena de caracteres en un documento en formato JSON (serializar) y el proceso contrario (deserializar) (Blanchon, 2021). En el gestor de librerías del IDE de Arduino se encuentra como *ArduinoJson*.
- *SoftwareSerial.h*: Permite la conexión serie entre dos dispositivos compatibles con Arduino (Stoffregen, 2020). En el gestor de librerías del IDE de Arduino se encuentra como *EspSoftwareSerial*.
- *BlueDisplay.h*: Permite a un dispositivo Android actuar como GUI de un dispositivo

Arduino (Armin, 2020). En el gestor de librerías del IDE de Arduino se encuentra como *BlueDisplay*.

- *PubSubClient.h*: Permite realizar publicaciones y suscripciones simples a través de un servidor que utilice el protocolor de comunicación MQTT (O'Leary, 2020). En el gestor de librerías del IDE de Arduino se encuentra como *PubSubClient*.
- *Ticker.h*: Permite la llamada de funciones de forma periódica (Philhower, 2020), generando interrupciones de manera sencilla. En el gestor de librerías del IDE de Arduino se encuentra como *Ticker*.
- *DHT.h*: Permite la lectura de los sensores de temperatura/humedad de bajo costo de la serie DHT (Herrada, 2021). En el gestor de librerías del IDE de Arduino se encuentra como *DHT sensor library*.

4.2 Distribución del código fuente

Los archivos del código fuente desarrollado está distribuido en tres carpetas, una para cada microcontrolador. La distribución de estos archivos es la siguiente:

***esp32-bluedisplay*:** Corresponde al código implementado para el ESP32, contenedor de la interfaz visual a través de BlueDisplay. Los archivos que contiene son:

- *mybuttons.cpp*: Contiene la implementación de las funciones auxiliares implementadas para los botones. Estas funciones permiten cambiar el estado de un botón (valor, color, encabezado y activación) llamándolas en una única línea de código.
- *mybuttons.h*: Contiene las cabeceras de las funciones implementadas en *mybuttons.cpp*.
- *mycolors.h*: Incluye las variables globales correspondientes a los colores utilizados para los distintos modos de color: Estándar, modo daltónico y modo alto contraste.
- *myvariables.h*: Incluye las declaraciones de las variables globales que se utilizan a lo largo del programa principal.
- *esp32-bluedisplay.ino*: Corresponde al código que se le enviará al ESP32, programado a través del IDE de Arduino.

***nodemcu-serial*:** Esta carpeta incluye los archivos utilizados por el NodeMCU que se encuentra conectado en serie con el ESP32. Su contenido es el siguiente:

- *wifi-credentials.h*: Contiene el Service Set Identifier (SSID) y la contraseña de la red WiFi a la que se va a conectar el NodeMCU, así como las claves necesarias para otorgarle los permisos para conectarse a AWS IoT Core.
- *nodemcu-serial.ino*: Corresponde a la implementación desarrollada para este NodeMCU, teniendo dos únicas funciones: Enviar a través de WiFi el mensaje recibido por vía serie o viceversa. Se ha programado a travñes del IDE de Arduino.

***nodemcu-devices*:** Aquí se encuentran los ficheros desarrollados para el NodeMCU que tiene los dispositivos conectados. Estos ficheros son:

- *myvariables.h*: Incluye las declaraciones de las variables globales que se utilizan a lo largo del programa principal de este NodeMCU.
- *wifi-credentials.h*: Contiene el SSID y la contraseña de la red WiFi a la que se va a conectar el NodeMCU, así como las claves necesarias para otorgarle los permisos para conectarse a AWS IoT Core.
- *nodemcu-devices.ino*: Corresponde al código implementado para este NodeMCU. Se ha programado a través del IDE de Arduino.

4.3 Compilación y carga a los microcontroladores

Inicialmente, es necesario configurar el IDE de Arduino antes de proceder a la carga de los archivos a los microcontroladores. Para ello se siguen los pasos siguientes (del Valle Hernández, s.f.; Santos, s.f.):

1. Añadir el URL para tarjetas adicionales en Archivo > Preferencias (véase Figura 4.2).

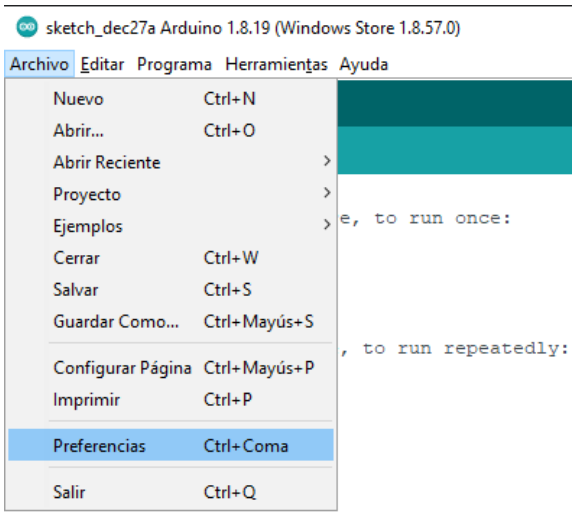


Figura 4.2: Archivo > Preferencias en el IDE de Arduino. Fuente: Elaboración propia.

Los URLs se pegan en el recuadro de "Gestor de URLs Adicionales de Tarjeta" (Figura 4.3), separados por comas. Los URLs a añadir son:

- https://dl.espressif.com/dl/package_esp32_index.json para el ESP32.
- http://arduino.esp8266.com/stable/package_esp8266com_index.json para los NodeMCU.

2. Instalar las tarjetas en Herramientas > Placa > Gestor de Tarjetas... (véase Figura 4.4).

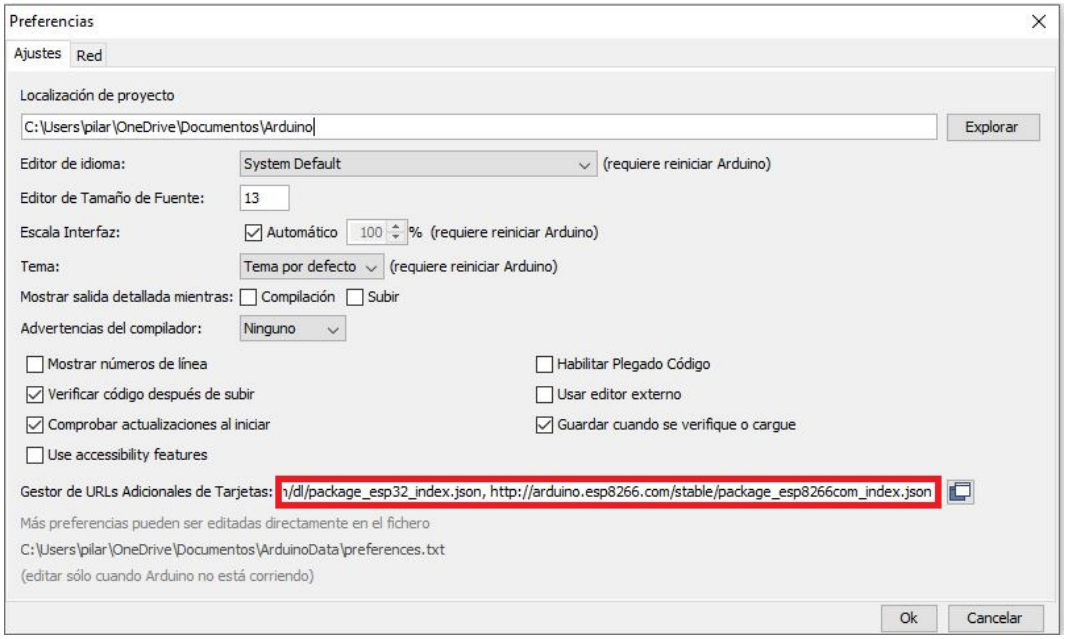


Figura 4.3: Adición de URLs adicionales en el IDE de Arduino. Fuente: Elaboración propia.

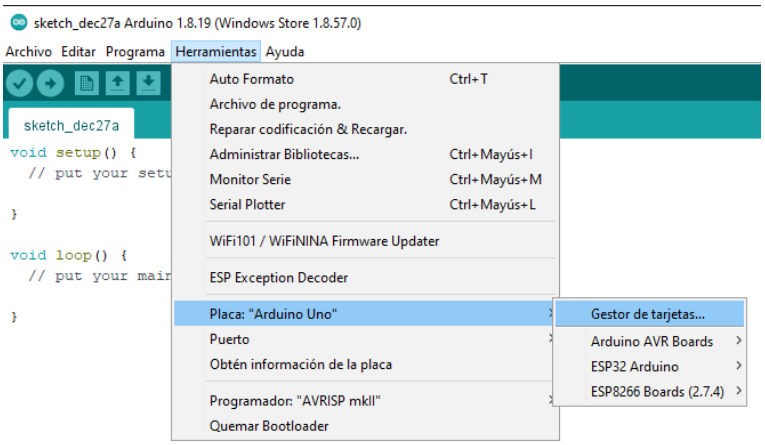


Figura 4.4: Herramientas > Placa > Gestor de Tarjetas... en el IDE de Arduino. Fuente: Elaboración propia.

Para añadir el ESP32 se busca en el gestor “esp32” y para el NodeMCU se busca “esp8266”.

3. Seleccionar la tarjeta a la que se le va a subir el programa en Herramientas > Placa. Para el ESP32 se selecciona ESP32 Arduino > DOIT ESP32 DEVKIT V1, mientras que para el NodeMCU se selecciona ESP8266 Boards (2.7.4) > NodeMCU 1.0 (ESP-12E Module).

4. Seleccionar el puerto serie en el que esté conectado el microcontrolador en Herramientas

> Puerto.

Finalmente, configurado el IDE de Arduino y seleccionado el microcontrolador al que se le va a subir el archivo, se presiona el botón de "Subir" (segundo botón por la izquierda de la Figura 4.5).

Figura 4.5: Botonera del IDE de Arduino. Fuente: Elaboración propia.

4.4 Manual de usuario

Es necesaria la descarga de la aplicación móvil BlueDisplay a través de la tienda de aplicaciones de Android (solamente está disponible para dispositivos con este sistema operativo) para hacer uso de la interfaz. Una vez descargada y con el Bluetooth activado, se selecciona el ESP32 desde la aplicación y aparecerá el menú principal de la GUI.

A través de esta interfaz es posible realizar varias acciones que permiten convertir la casa del usuario en una *smart home*. En el menú principal de esta GUI, mostrado en la Figura 4.6, se puede observar que se ofrecen cuatro opciones principales:

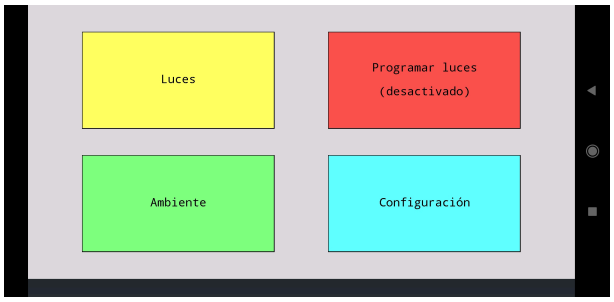
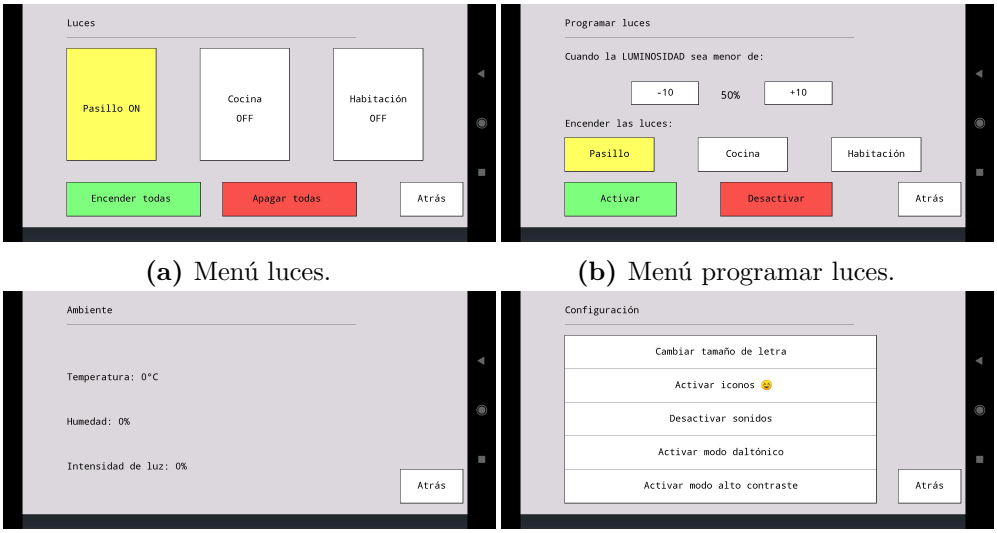


Figura 4.6: Menú principal de la interfaz visual con BlueDisplay. Fuente: Elaboración propia a través de la herramienta de diseño Adobe XD.

- **Luces:** Permite al usuario encender y apagar las luces de su casa, individualmente o todas a la vez (Figura 4.7a).
- **Programar luces:** Ofrece al usuario la posibilidad de programar el encendido de las luces de su casa que desee cuando el porcentaje de luminosidad o nivel de luz sea menor del indicado por él en esta opción (Figura 4.7b).
- **Ambiente:** Indica los valores obtenidos de los sensores utilizados, correspondientes a los datos de temperatura, humedad y luminosidad (Figura 4.7c).

- **Configuración:** Permite al usuario cambiar la configuración de la interfaz a través de varias acciones disponibles: Cambiar el tamaño de letra y activar o desactivar iconos, sonidos, el modo daltónico y el modo de alto contraste (Figura 4.7d).



(a) Menú luces. **(b)** Menú programar luces.

(c) Menú ambiente. **(d)** Menú configuración.

Figura 4.7: Menús diseñados para la interfaz con BlueDisplay. Fuente: Elaboración propia.

La librería *BlueDisplay.h* no contempla la obtención del ancho y alto de la pantalla, por lo que deberían personalizarse estos valores para cada teléfono, suponiendo este hecho una limitación para la creación de una interfaz móvil haciendo uso de esta librería. Los valores establecidos son de 2027x2013[1] (ancho x alto). Esta definición se realiza en las líneas 43-44 del archivo *esp32-bluedisplay.ino*.

[1]Valores utilizados para el teléfono empleado para las pruebas, un Pocophone F1

5 Amazon Alexa

El código fuente desarrollado para la función Lambda creada se encuentra en un repositorio de GitHub, en el enlace `https://github.com/png10/TFG-de-Pilar-Navarro-Garcia/tree/main/Interfaz%20de%20voz/Lambda`.

5.1 Creación de una skill de Alexa

La Voice User Interface (VUI) a través de Amazon Alexa requiere del desarrollo de una skill de Alexa. Lo primero es la creación de una skill de Alexa a través de la consola de desarrollo de Alexa, para lo que se siguen los siguientes pasos (Figura 5.1):

1. Establecer el nombre de la skill, que no interferirá en nada a la hora de programar su funcionamiento (Figura 5.1a).

2. Indicar el *primary locale*, que se refiere al idioma y país donde se va a desarrollar la skill (Figura 5.1a).

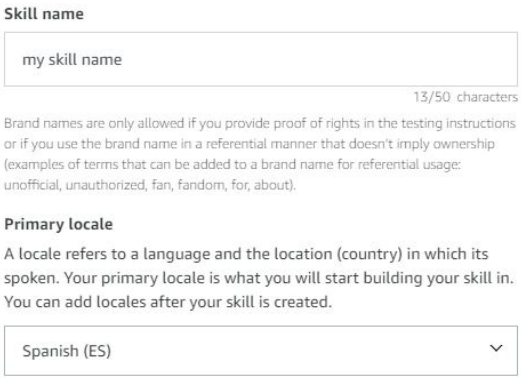
3. Elegir el tipo de skill, que para este proyecto será de tipo custom (Figura 5.1b).

4. Escoger el método para alojar los recursos de backend de la skill, en este caso se emplea el método *provision your own* (Figura 5.1b).

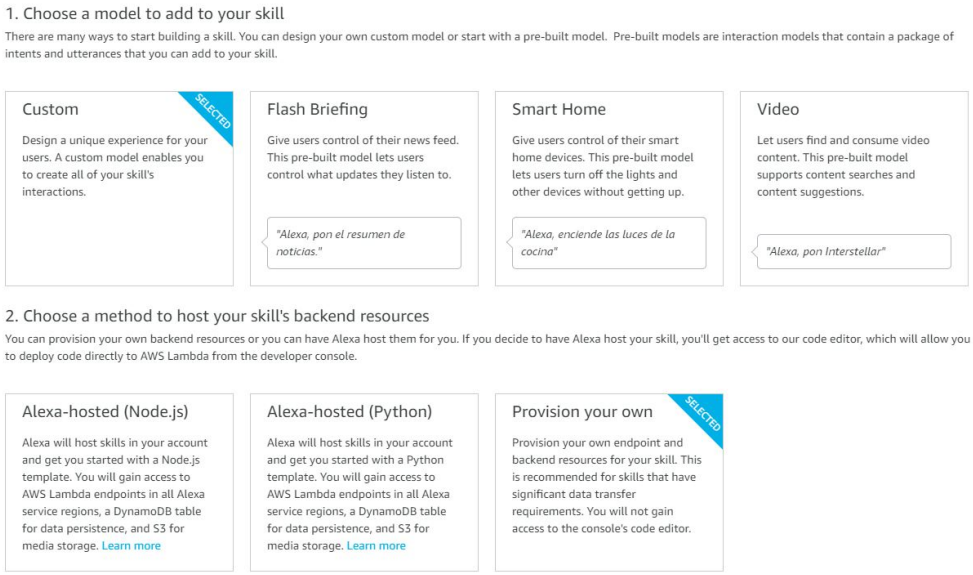
5. Seleccionar la plantilla a través de la cual se empezará a desarrollar la skill o importar una skill. Aquí se ha elegido *start from scratch*, es decir, con una plantilla con la funcionalidad mínima, siendo esta el típico "Hola Mundo" en programación (Figura 5.1c).

A continuación se establece el skill invocation name de manera que sea intuitivo y fácilmente pronunciable, en este caso "mi casa". Este se utiliza para invocar la skill de manera que, previamente a realizar cualquier petición dentro de la skill, el usuario debe invocarla diciendo alguna de las frases siguientes, seguida de la petición deseada:

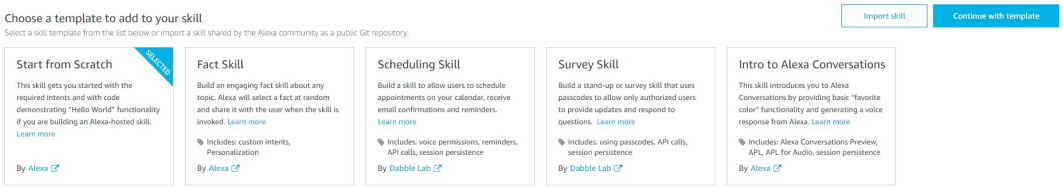
- Abre mi casa y dile que...
- Abre mi casa y...
- Dile a mi casa que...
- Pide a mi casa que...



(a) Elección del nombre y el primary locale.



(b) Elección del tipo de skill y el método para alojar los recursos de backend.



(c) Elección de plantilla.

Figura 5.1: Proceso de creación de una skill de Alexa. Fuente: Elaboración propia.

5.2 Intents, slots y utterances

Lo siguiente es establecer los intents necesarios para dotar de todas las funcionalidades necesarias a la skill, así como los utterances para cada intent y los slots empleados en los utterances.

Los intents desarrollados hacen que la interfaz pueda ofrecer las mismas posibilidades que la interfaz visual con BlueDisplay (a excepción de la configuración de la interfaz). Los intents implementados son los siguientes:

- **DescribeScreenIntent:** Permite al usuario conocer en qué pantalla o menú se encuentra en la interfaz de BlueDisplay, así como las acciones que puede realizar dentro de esta. El usuario también puede preguntar sobre un menú en específico. Es la única funcionalidad exclusiva en esta interfaz.
- **GetLightsIntent:** Indica el estado de todas las luces.
- **GetSpecificLightIntent:** Indica el estado de una luz en específico.
- **UpdateDeviceIntent:** Permite cambiar el estado de uno o varios de los leds conectados.
- **ProgramLightsIntent:** Permite programar el encendido de las luces cuando el nivel de luz sea menor de un umbral. Tanto las luces a encender como el umbral deben ser indicados por el usuario.
- **DeprogramLightsIntent:** Permite eliminar o cancelar la programación de las luces en vigor.
- **GetAllSensorsIntent:** Indica las lecturas de todos los sensores conectados.
- **GetSpecificSensorIntent:** Indica una lectura en específico de los sensores conectados.

Si no se empleara ningún slot, sería necesario crear un intent distinto para cada dispositivo distinto a controlar o consultar su estado. Por lo tanto, se emplean los siguientes slots para el correcto funcionamiento de los intents anteriores y, sobre todo, permitir la escalabilidad del sistema:

- **Screen:** Pantalla o menú a describir en el intent "GetScreenIntent". Los valores que puede tomar son: "menú principal", "luces", "programar luces", "ambiente" y "configuración".
- **LightDevice:** Luz o luces a las que se hace referencia en los intents "GetSpecificLightIntent" y "UpdateDeviceIntent". Puede tomar los valores: "luz del pasillo", "luz de la cocina", "luz de la habitación" y "todas las luces", así como variantes de estas como "cocina" en lugar de "luz de la cocina" y "bombillas" en lugar de "luces".
- **LightAction:** Acción a realizar en el intent "UpdateDeviceIntent". Estas acciones corresponden a los valores: "enciende" y "apaga".
- **SensorDevice:** Sensor específico empleado en el intent "GetSpecificSensorIntent". Los valores aceptados son: "temperatura", "humedad", "luminosidad" y "luz".
- **AMAZON.NUMBER:** Número para el porcentaje empleado en el umbral utilizado en "ProgramLightsIntent". Se trata de un slot prediseñado e incorporado en la consola de Alexa developer.

Volviendo a los intents desarrollados, es necesario definir los utterances para cada uno de ellos. Estos utterances son los que indican a la VUI el intent al que se hace referencia al realizar una petición, por lo que deben diferenciarse correctamente para cada intent con el fin de evitar ambigüedades. Los slots utilizados se encuentran entre corchetes ({}).

- Utterances de "DescribeScreenIntent":
  - Qué puedo hacer aquí.
  - Qué hago en este menú.
  - Dime qué hace este menú.
  - Dime qué hace esta pantalla.
  - Qué puedo hacer en esta opción.
  - Qué puedo hacer en este menú
  - Qué puedo hacer en la opción {Screen}.
  - Qué hago en la opción {Screen}.
  - Qué hago en la pantalla {Screen}.
  - Qué puedo hacer en el menú {Screen}.
- Utterances de "GetLightsIntent":
  - Cuál es el estado de mis luces.
  - Cómo están las luces de mi casa.
  - Cómo están las luces.
  - Cómo están mis luces.
  - Cuál es el estado de las luces de mi casa.
- Utterances de "GetSpecificLightIntent":
  - Cómo he dejado la {LightDevice}.
  - Cómo está la {LightDevice}.
  - Cuál es el estado de la {LightDevice}.
- Utterances de "UpdateDeviceIntent":
  - {LightAction} la {LightDevice}.
  - {LightAction} {LightDevice}.
- Utterances de "ProgramLightsIntent":
  - Programa el encendido de las luces.
  - Programa las luces.
  - Programa que la {LightDevice} se encienda cuando haya un {AMAZON.NUMBER} por ciento de luz.
  - Programa el encendido de la {LightDevice} y la {LightDevice} cuando la luz sea menor del {AMAZON.NUMBER} por ciento.

  - Programa el encendido de la {LightDevice} cuando la luz sea menor del {AMAZON.NUMBER} por ciento.

- Utterances de "DeprogramLightsIntent":
  - Desactiva el programa de las luces.
  - Quita el programa de las luces.
- Utterances de "GetAllSensorsIntent":
  - Cómo están los sensores de mi casa.
  - Cómo están los sensores
  - Cuál es el estado de los sensores de mi casa.
  - Cuál es el estado de mis sensores.
- Utterances de "GetSpecificSensorIntent":
  - Qué {SensorDevice} mide el sensor de {SensorDevice}.
  - Qué {SensorDevice} mide el sensor.
  - Qué {SensorDevice} hay.
  - Qué mide el sensor de {SensorDevice}.
  - Cuál es la lectura del sensor de {SensorDevice}.
  - Cuál es el estado del sensor de {SensorDevice}.

Bibliografía

Armin. (2020). *BlueDisplay Library for Arduino.* Descargado de https://github.com/ArminJo/Arduino-BlueDisplay

Blanchon, B. (2021). *ArduinoJson.* Descargado de https://github.com/bblanchon/ArduinoJson

del Valle Hernández, L. (s.f.). *Guía para configurar un ESP-01, el módulo WiFi basado en ESP8266.* Descargado de https://programarfacil.com/podcast/como-configurar-esp01-wifi-esp8266/

Herrada, E. (2021). *DHT sensor library.* Descargado de https://github.com/adafruit/DHT-sensor-library

O'Leary, N. (2020). *Arduino Client for MQTT.* Descargado de https://github.com/knolleary/pubsubclient

Philhower, E. F. (2020). *Ticker.h.* Descargado de https://github.com/esp8266/Arduino/blob/master/libraries/Ticker/src/Ticker.h

Santos, R. (s.f.). *Installing ESP32 in Arduino IDE (Windows, Mac OS X, Linux).* Descargado de https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/

Stoffregen, P. (2020). *SoftwareSerial.* Descargado de https://github.com/PaulStoffregen/SoftwareSerial

Lista de acrónimos y abreviaturas

AWS	Amazon Web Services.
GUI	Interfaz Gráfica de Usuario.
IAM	Identity and Access Management.
IDE	Entorno de Desarrollo Integrado.
SDK	Software Development Kit.
SSID	Service Set Identifier.
URL	Localizador de Recursos Uniforme.
VUI	Voice User Interface.