UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

DISEÑO Y DESARROLLO DE UN PROTOTIPO DE ESPEJO INTELIGENTE (SMIRROR)

TRABAJO FINAL DEL

Grado en Ingeniería Electrónica Industrial y Automática

REALIZADO POR

Gao Yao, Jia Cheng

TUTORIZADO POR

Berjano Zanón, Enrique

CURSO ACADÉMICO: 2019/2020

RESUMEN

En los últimos años se ha observado un auge en el desarrollo de productos de domótica que facilitan y automatizan muchos procesos del día a día. Estos productos funcionan complementando a los asistentes de voz, los cuales consisten en unos dispositivos que utilizando inteligencia artificial, permiten realizar mediante la voz diferentes funciones como búsquedas en internet, informar sobre el tiempo o controlar otros dispositivos inteligentes. Los más populares actualmente son *Google Home* y *Alexa*. Combinando la domótica con otras tecnologías como los espejos dieléctricos se han creado los espejos inteligentes. Éstos consisten en una pantalla controlada por un microprocesador situada detrás de un espejo dieléctrico. Éste permite el paso de la luz emitida por la pantalla, a la vez que las zonas no iluminadas por ésta se comportan como un espejo convencional.

En este Trabajo de Final de Grado (TFG) se ha diseñado y desarrollado un prototipo de espejo inteligente basándose en la programación de una *Raspberry PI 4*, con objeto de crear una interfaz donde el usuario pueda visualizar ciertos parámetros como la fecha y hora, información atmosférica, y las noticias importantes del día. Por otro lado, se ha programado una placa de desarrollo *NodeMCU* con objeto de controlar mediante la voz una tira de LEDs con tres canales de colores (RGB) colocada en el marco del espejo.

RESUM

Als últims anys s'ha observat un auge en el desenvolupament de productes de domòtica que faciliten i automatitzen molts processos del dia a dia. Aquests productes funcionen complementant als assistents de veu, els quals consisteixen en uns dispositius que utilitzant intel·ligència artificial, permeten realitzar mitjançant la veu diferents funcions com cerques en internet, informar sobre el temps o controlar altres dispositius intel·ligents. Els més populars actualment són Google Home i Alexa. Combinant la domòtica amb altres tecnologies com els miralls dielèctrics s'han creat els miralls intel·ligents. Aquests consisteixen en una pantalla controlada per un microprocessador situada darrere d'un mirall dielèctric. Aquest permet el pas de la llum emesa per la pantalla, alhora que les zones no il·luminades per aquesta es comporten com un mirall convencional.

En aquest Treball de Final de Grau (TFG) s'ha dissenyat i desenvolupat un prototip de mirall intel·ligent basant-se en la programació d'una Raspberry PI 4, a fi de crear una interfície on l'usuari puga visualitzar certs paràmetres com la data i hora, informació atmosfèrica, i les notícies importants del dia. D'altra banda, s'ha programat una placa de desenvolupament NodeMCU a fi de controlar mitjançant la veu una tira de LEDs amb tres canals de colors (RGB) col·locada en el marc del mirall.

ABSTRACT

In recent years there has been a boom in the development of home automation products that facilitate and automate many day-to-day processes. These products work by complementing voice assistants, which consist of devices that use artificial intelligence, allow to perform through voice different functions such as internet searches, report time or control other smart devices. The most popular currently are Google Home and Alexa. Combining home automation with other technologies such as dielectric mirrors, smart mirrors have been created. These consist of a screen controlled by a microprocessor located behind a dielectric mirror. This allows the passage of the light emitted by the display, while the areas not illuminated by the screen behave like a conventional mirror.

In this Final Degree Project (TFG), a prototype smart mirror has been designed and developed based on the programming of a Raspberry PI 4, in order to create an interface where the user can visualize certain parameters such as date and time, atmospheric information, and important news of the day. On the other hand, a NodeMCU development board has been programmed to control by means of the voice a strip of LEDs with three color channels (RGB) placed in the mirror frame.

MEMORIA

Índice de contenido

Índice de ilustraciones

INDICE DE TABLAS

1. OBJETO

El objetivo del TFG ha sido el diseño y desarrollo de un espejo inteligente controlable mediante voz. Este equipo tiene el aspecto de un espejo convencional con la única diferencia de que se pueden ver elementos a través de él como si fuera una pantalla de ordenador o de teléfono inteligente. El TFG ha implicado la programación de un microprocesador y una placa de desarrollo con el fin de que realicen diferentes funciones, tales como obtener la dirección IP del microprocesador (para situar la latitud y longitud de la ubicación). A partir de la ubicación, se puede obtener la información meteorológica actual. Por otro lado, también representa la fecha y hora, además de indicar si esta fecha en concreto es un día festivo o no, y cuándo se trata de días festivos. También, muestra las 5 noticias más relevantes de un periódico digital elegido. Además, expone la información más importante de la canción que esté sonando en una cuenta de *Spotify* en cada momento. Por último, puede controlar con el asistente de voz diferentes parámetros de la iluminación del marco del espejo (como encender, apagar o cambiar el color).

2. ANTECEDENTES

El gran avance de la tecnología en los últimos años ha permitido el desarrollo de nuevas invenciones como los espejos dieléctricos. Este tipo de espejos tienen la propiedad de permitir el paso a una parte de la luz a la vez que reflejan la restante, un uso habitual de éstos es como falso espejo en salas de interrogatorios. En esta situación, el espejo separa dos zonas, una más iluminada que la otra. Por lo que podemos diferenciar dos situaciones:

Si está situado en la zona más iluminada, la luz reflejada en el espejo desde el propio habitáculo será mucho mayor que la que atraviesa desde la zona oscura. Por lo que solo verá un espejo cotidiano (ver Fig. 1).

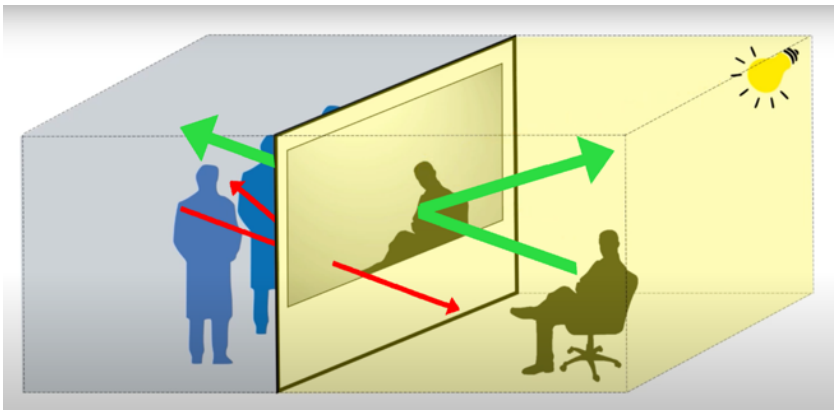


Figura 1. Comportamiento de un espejo dieléctrico desde la zona más iluminada. El espectador se sitúa en la zona más iluminada. La luz del propio habitáculo que es reflejada en el espejo (verde) es mucho mayor que la luz que atraviesa desde la zona menos iluminada (rojo). Por lo que el espectador solo verá su reflejo (tomado de [1]).

Si por el contrario está situado en la zona menos iluminada, la luz que atraviesa desde la zona opuesta será mayor que la reflejada desde la propia zona. Por lo que el espectador podrá ver a través de este espejo (ver Fig. 2).

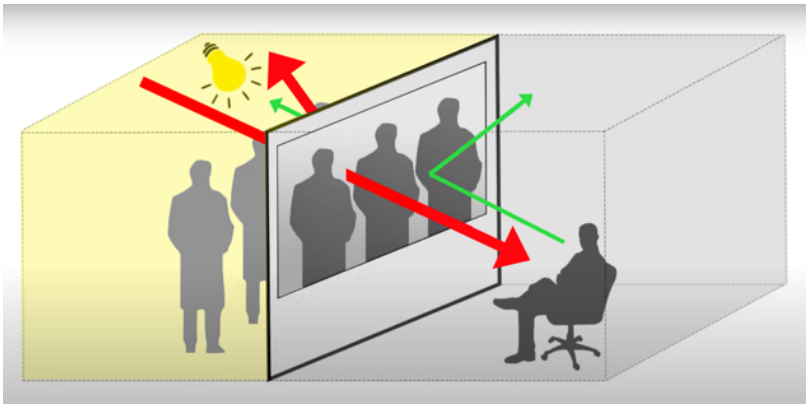


Figura 2. Comportamiento de un espejo dieléctrico desde la zona menos iluminada. El espectador se sitúa en la zona menos iluminada. La luz que atraviesa el espejo desde la zona más iluminada (rojo), es mucho mayor que la luz del propio habitáculo reflejada en el espejo (verde). Por lo que el espectador verá a través del espejo (tomado de [1]).

La combinación de este tipo de espejos con una pantalla luminosa ha dado lugar a los espejos inteligentes. Estos tienen el aspecto de un espejo convencional con la particularidad de poder ver la pantalla a través del espejo (ver Fig. 3).

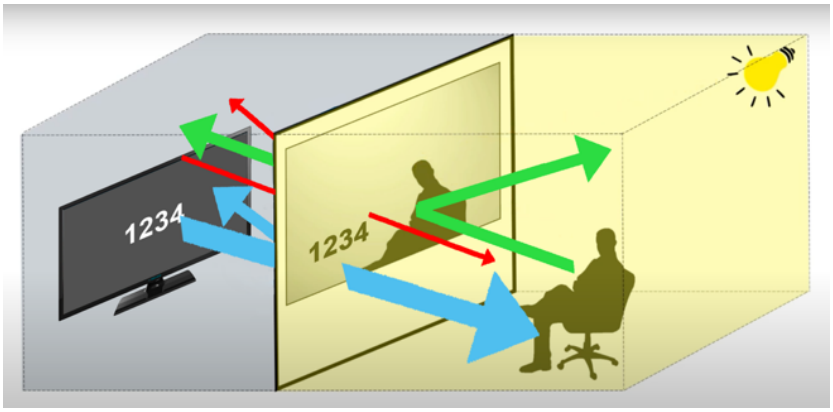


Figura 3. Comportamiento de un espejo dieléctrico ante una pantalla. El espectador se sitúa en la zona más iluminada. Por lo que, la luz reflejada en el espejo por el propio habitáculo (verde) es mucho mayor que la luz que atraviesa desde el habitáculo menos iluminado (rojo). Sin embargo, la luz de la pantalla que atraviesa el espejo (azul) es mucho mayor que las dos anteriores. Por esto, el espectador verá que todo el espejo refleja, menos la zona iluminada por la pantalla, donde se verá lo iluminado por esta (tomado de [1]).

De este modo se pueden lograr infinidad de posibilidades como mostrar la fecha y hora, información meteorológica, noticias, imágenes y videos o su agenda. Actualmente podemos encontrar en el mercado diferentes productos entre los cuales podemos destacar varios.

Por ejemplo, el producto *Mirror* de la empresa *Mirror* (Nueva York, USA) es un espejo inteligente que realiza diversas funciones, entre las que resalta el entrenamiento personal (ver Fig 4). Gracias a su cámara incorporada el dispositivo puede detectar qué tipo de ejercicio se está haciendo e indicarte si estás haciendo algo incorrectamente. Al contar también con micrófono y altavoz permite también hacer llamadas y video llamadas. Sin embargo, no cuenta con pantalla táctil, por lo que todos los controles se deben de manejar desde la aplicación móvil. Este producto tiene un coste de 1280 €.

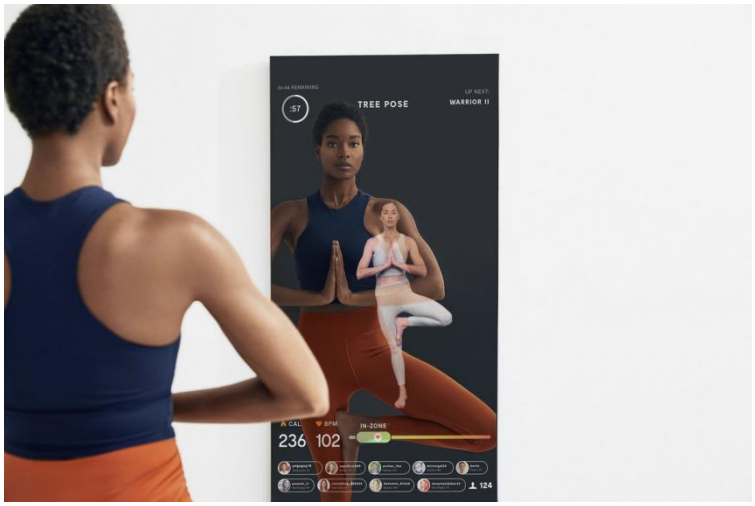


Figura 4. *Mirror*. Espejo inteligente con entrenamiento personal.

Por otro lado encontramos el producto *Bathroom Mirror* de la empresa *Maebow* (Shanghai, China), el cual cuenta con micrófono, altavoz y tecnología Bluetooth, permitiendo entre otras cosas reproducir música y realizar llamadas (ver Fig. 5). También posee una pantalla táctil que permite controlar las diferentes funciones del espejo como mostrar la fecha y hora, humedad, temperatura, o controlar el Bluetooth y el sistema de calefacción para el desempañado. Este espejo tiene un coste de 285 €.

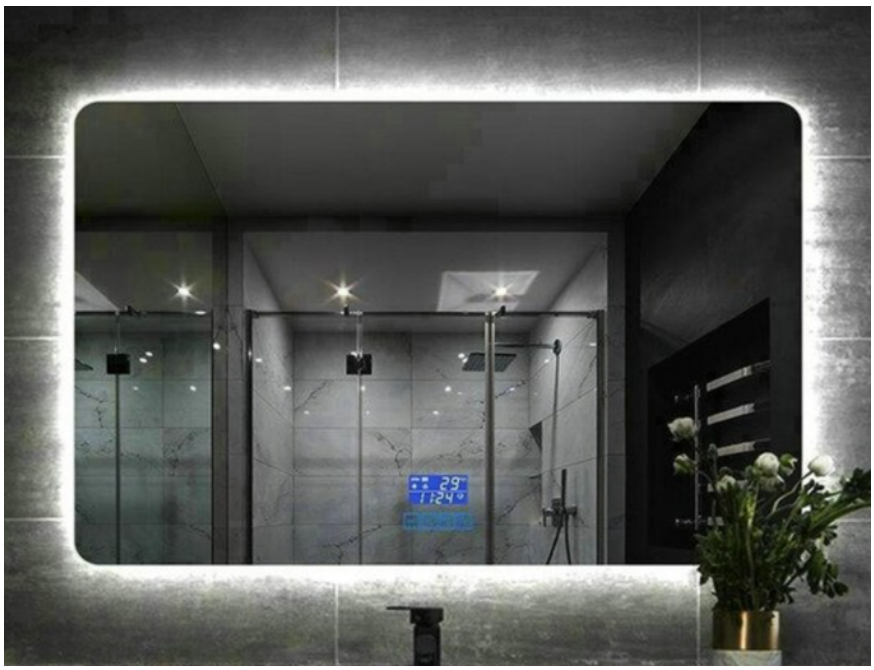


Figura 5. *Bathroom Mirror*. Espejo inteligente con tecnología Bluetooth y pantalla táctil.

3. MOTIVACIÓN

Los espejos inteligentes son productos que no están actualmente muy explotados, puesto que la mayoría de la población no los conoce. Además son dispositivos a los que se les pueden añadir infinidad de funciones como se ha demostrado en el caso de *Mirror*. Se puede considerar que este tipo de producto tiene un gran potencial gracias a su capacidad de adaptarse a las nuevas tecnologías. Por ejemplo, combinándolo con la realidad aumentada, podríamos llegar incluso a probarnos ropa sin salir de casa o controlar nuestra salud con una acción tan cotidiana como mirarnos al espejo.

Por otro lado, los productos que están actualmente en venta o tienen un precio muy elevado, o tienen malas prestaciones como una pantalla y controles muy básicos. Es por esto por lo que en este TFG se ha decidido desarrollar un producto de bajo coste pero con buenas prestaciones tales como una pantalla de alta definición y controles mediante el asistente de voz.

También se ha tomado como ejemplo de un proyecto de Youtube en el que se realizaba un espejo inteligente mediante un microprocesador y código Java Script (tomado de [2]).

4. SOLUCIONES ALTERNATIVAS

La elección entre las diferentes alternativas se va a realizar en base a los componentes constitutivos, tales como una tira de LEDs, la pantalla, el material de reflexión, el microprocesador y la placa de desarrollo (ver Fig. 6):

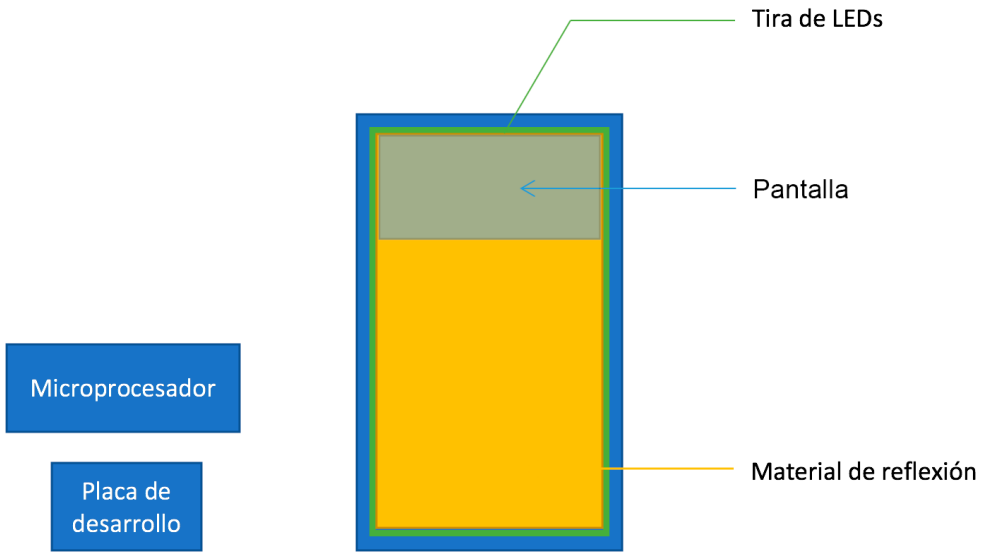


Figura 6. Principales componentes constitutivos del prototipo objeto del proyecto.

Estos componentes constitutivos se interconectarán de manera que una vez programado el microprocesador se muestre en la pantalla una interfaz gráfica. La placa de desarrollo nos permitirá controlar la tira de LEDs mediante la voz. Por último el material reflectante será el encargado de dar a todo el diseño el aspecto de un espejo (ver Fig. 7).

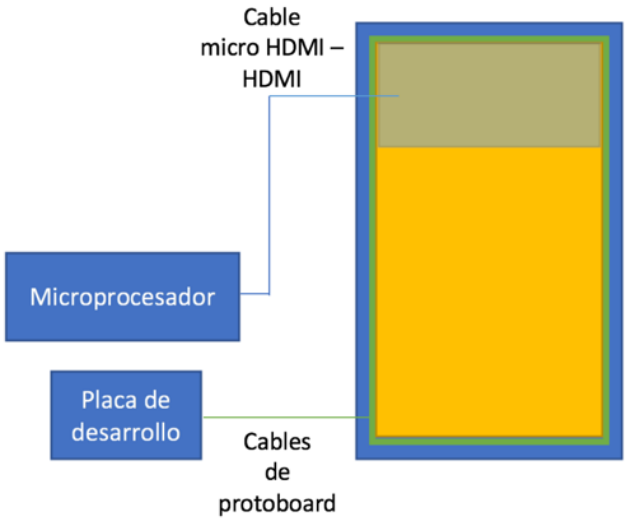


Figura 7. Conexiones entre los diferentes componentes.

4.1 Placa de desarrollo

Para poder controlar el espejo inteligente mediante nuestro asistente de voz necesitamos una placa de desarrollo con conexión Wifi. Existen varias opciones para ello.

En primer lugar tenemos la placa NodeMCU V2 de Az-Delivery (Deggendorf, Alemania), la cual posee un microcontrolador ESP8266 con Wifi y antena PCB integrada (ver Fig. 8). Cuenta también con un conversor CP2102. Para su programación puede usarse el *software* Arduino o Espressif-SDK. Dispone de salidas PWM y un tamaño de 48 x 26 x 13 mm. Tiene un coste de 7.49 €.

Figura 8. NodeMCU V2 de Az-Delivery (Deggendorf, Alemania).

Por otro lado encontramos la placa NodeMCU V3 también de Az-Delivery (Deggendorf, Alemania), la cual posee un también microcontrolador ESP8266 con Wifi y antena PCB integrada (ver Fig. 9). Cuenta con un conversor CH340G. También se programa con Arduino o Espressif-SDK. Dispone de salidas PWM y un tamaño de 58 mm x 31 mm x 13 mm. Tiene un coste de 7.79 €.

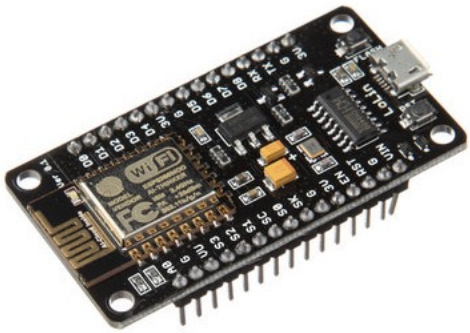


Figura 9. NodeMCU V3 de Az-Delivery (Deggendorf, Alemania).

Entre estas dos opciones se ha decidido utilizar la *NodeMCU V2*, ya que según indica el fabricante la principal diferencia con el *NodeMCU V3* es que ésta última tiene una conexión USB más estable. Sin embargo la comunidad de usuarios no ha notado esta diferencia, recomendando así ambos productos. Así se va a escoger la versión V2 debido a su menor precio.

4.2 Tira de LEDs

Para otorgarle iluminación al espejo se va a incorporar una tira de LEDs en el marco que será controlada mediante la placa de desarrollo. Se han considerado dos alternativas.

En primer lugar una tira *LED 6M Impermeable* de *Omeril* (Nueva York, USA), la cual esta está formada por 180 LEDs RGB de tipo 5050 SMD (Surface Mounting Device). Requiere de una fuente de alimentación de 5 V y 1 A, y tiene un coste de 15.29 €.

La otra opción es la *5m WS2812B Tira de luz LED* Impermeable de *Chinly* (Guangzhou, Guangdong, China). Está formada por 150 LEDs RGB de tipo WS2812b SMD. Necesita una fuente de alimentación de 5 V y 9 A, y tiene un coste de 24.96 €.

Se ha escogido la *Tira LED 6M Impermeable* del fabricante *Omeril* debido a que tiene un menor consumo y también un precio menor.

4.3 Material de reflexión

Para crear el efecto de espejo dieléctrico podemos emplear dos tipos de productos: un vinilo unidireccional o un acrílico bidireccional. El vinilo unidireccional de TTMOW (Shen Zhen, Guang Dong, China)es una lámina de vinilo con un grosor de 0.15 mm la cual se debe adherir a un metacrilato o vidrio (ver Fig. 10). Ésta permitirá el paso de luz, pero en las zonas que tenga el fondo oscuro hará el efecto reflejo. Con unas dimensiones de 40 x 200 cm tiene un precio de 11.88 €. Sin embargo, tiene el inconveniente de que debido a su espesor es un material que se marca muy fácil y es complicado conseguir un acabado perfecto sin suciedad entre ambos materiales.

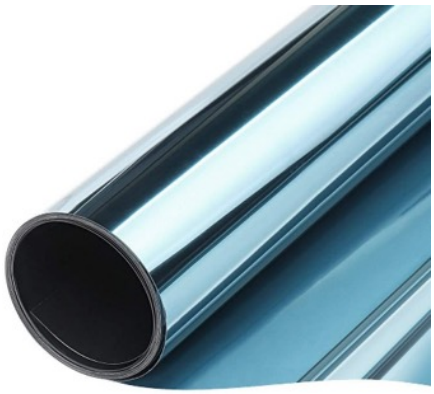


Figura 10. Vinilo unidireccional.

El acrílico bidireccional de Supreme Tech (Virginia, USA) es un material con mayor espesor (1-3 mm) que no hará falta adherirlo a ningún metacrilato o vidrio, por lo que no será marcado en la instalación quedando así mejor acabado (ver Fig. 11). Sin embargo, tiene un precio superior con un coste de 38.46 € para un panel de 456 x 608 x 1 mm.

Figura 11. Acrílico bidireccional.

Se ha decidido utilizar el vinilo unidireccional adherido sobre un metacrilato de 3 mm de grosor ya que es mucho más económico que el acrílico bidireccional.

4.4 Pantalla

Una parte fundamental de este proyecto es la pantalla. Debido al tamaño deseado del espejo se va a escoger una pantalla de 22 pulgadas. Existen varias opciones. Primero encontramos el modelo *22f* de *HP* (Palo Alto, California, USA). Cuenta con una resolución de 1920 x 1080 píxeles con un tiempo de respuesta de 5 ms. Además, dispone de una entrada HDMI (High-Definition Multimedia Interface) y una VGA (Video Graphics Array). Como inconveniente este modelo no incluye altavoces incorporados. Tiene un coste aproximado de 101 €.

También encontramos el modelo *22TK410V-PZ* de *LG* (Seúl, Corea del Sur). Éste cuenta con una resolución de 1920 x 1080 píxeles con un tiempo de respuesta de 5 ms. Además, dispone de una entrada HDMI. Por otro lado este modelo tiene la ventaja de contar con dos altavoces de 5 W incorporados. Este producto tiene un coste aproximado de 145 €.

Se ha decidido emplear la pantalla *Hp 22f* ya que para el uso que va a tener en nuestro proyecto tiene prestaciones más que suficientes y tiene un coste menor.

4.5 Microprocesador

La gran mayoría de información que se va a manejar en este proyecto se va a representar a través de una interfaz gráfica, por lo que se va a utilizar un microprocesador como la *Raspberry Pi* en lugar de un microcontrolador como *Arduino*. La razón es que para crear una interfaz gráfica en *Arduino*, por un lado se necesita una pantalla específica y por lo general suelen ser de reducido tamaño (hasta 10 pulgadas), y por otro lado programar una interfaz gráfica en *Arduino* es más complicado que programarla en *Python* en un microprocesador como la *Raspberry Pi*. Además, los microprocesadores tienen conectores HDMI que permiten la conexión directa a la pantalla.

La *Raspberry Pi* de *Raspberry Pi Foundation* (Cambridge, Reino Unido) es un ordenador reducido u ordenador de placa única SBC (single board computer) de bajo coste. Éste fue desarrollado en Reino Unido con fines académicos, fomentando la enseñanza informática en las escuelas, por la fundación *Raspberry pi* de la *Universidad de Cambridge*. Desde su lanzamiento en 2012 han ido sacando diferentes modelos, de los cuales vamos a explicar los últimos ya que se adaptan mejor a las últimas tecnologías además de garantizar mejores prestaciones.

La Tabla 1 muestra las principales diferencias técnicas entre los dos últimos modelos de *Raspberry pi*.

Tabla 1. Comparación entre los últimos modelos de Raspberry Pi (tomado de [3]).

	Raspberry Pi 4 B	*Raspberry Pi 3 B+*
Procesador	Broadcom BCM2711B0, quadcore Cortex-A72	Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC
GPU	VideoCore VI 500MHz	VideoCore IV 400MHz
Memoria	1/2/4 GB LPDDR4-3200	1GB LPDDR2 SDRAM
Conectividad inalámbrica	Wi-Fi 2,4GHz / 5GHz IEEE 802.11.b/g/n/ac Bluetooth 5.0, BLE	Wi-Fi 2,4GHz / 5GHz IEEE 802.11.b/g/n/ac Bluetooth 4.2, BLE
Conectividad Ethernet	Gigabit Ethernet	Gigabit Ethernet over USB 2.0
Puertos	GPIO 40 pines 2 x Micro HDMI 2 x USB 2.0 2 x USB 3.0 CSI (cámara Raspberry Pi) DSI (pantalla tácil) Toma auriculares / vídeo compuesto Micro SD USB-C (alimentación) Power-over-Ethernet (PoE)	GPIO 40 pines HDMI 4 x USB 2.0 CSI (cámara Raspberry Pi) DSI (pantalla tácil) Toma auriculares / vídeo compuesto Micro SD Micro USB (alimentación) Power-over-Ethernet (PoE)
Precio	60 €	40 €

De la Tabla 1 podemos deducir que el modelo *4 B* cuenta con varias ventajas respecto al anterior modelo como un procesador más potente o un numero mayor de configuraciones de la memoria RAM (Random Access Memory) llegando hasta 4 GB, además de haber aumentado la velocidad de esta al pasar del modelo LPDDR2 al LPDDR4 (tomado de [4]). Por otro lado cuenta con dos puestos USB 3.0, estos más rápidos que los USB 2.0. Además dispone de dos salidas micro HDMI que pueden ofrecer hasta una resolución 4K (3840 × 2160 pixeles). Por último se ha mejorado la conectividad Bluetooth pasando de la versión 4.2 a la 5.0.

En este caso se ha decidido elegir la *Raspberry Pi 4 B*, concretamente el modelo de 4GB de RAM debido a que es un proyecto que puede continuar desarrollándose con el tiempo añadiendo así más funciones. Por esto se ha elegido este modelo ya que es más rápido y potente que la versión anterior (*Raspberry Pi 3 B*+), debido principalmente a que el procesador de este ofrece hasta tres veces el rendimiento de la versión anterior.

5. SOLUCIÓN ADOPTADA

5.1 Materiales y componentes

Se va a realizar una tabla con los diferentes materiales y componentes utilizados para desarrollar el montaje físico del prototipo (ver Tabla 2).

Tabla 2. Listado de materiales y componentes del TFG.

Raspberry pi 4 B 4GB RAM
Adaptador microSD-USB
Tarjeta microSD de 32 GB
Cable micro HDMI – HDMI
Pantalla Hp 22f de 22 pulgadas
Fuente alimentación de la pantalla
Fuente de alimentación 5.1V 3A
NodeMCU V2
Cables para protoboard
Soldador de estaño
Hilo de estaño para soldar
Fuente de alimentación 5V 20 A
Transistores BJT BD139
Resistencias 330 Ω
Protoboard de 4.6 x 3.6 cm
Tira de LEDs RGB
Vinilo unidireccional
Plancha de metacrilato
Marco de DM
4 tornillos
4 embellecedores de tornillos
Regleta de 4 conectores
2 láminas de Goma EVA negras de 60 x 40 cm
Cinta aislante negra de 19 mm de ancho
Cinta de doble cara de espuma

5.2 Montaje físico de los componentes del prototipo

5.2.1 Marco y metacrilato

Se ha encargado a una fábrica de muebles local la creación de un marco de DM (madera de fibra de densidad media) a partir de los planos hechos en *Autocad* (ver Anexo X). La elección de este material es por su fácil manejo y su coste moderado (21 €/m^2) para un tablero de 3 cm de grosor. También se ha encargado a la misma fábrica el corte del metacrilato con las medidas del marco entero (ver Fig. 12).

Figura 12. Marco y metacrilato hechos a medida por una fábrica de muebles local.

5.2.2 Colocación del vinilo en el metacrilato

Una vez tenemos las piezas fabricadas pasamos a adherir el vinilo unidireccional sobre el metacrilato. Para ello debemos separar el vinilo unidireccional de la lámina protectora, dejando así al descubierto la parte adhesiva del vinilo (ver Fig. 13). Posteriormente aplicaremos una solución jabonosa de aproximadamente el 50% de agua y el 50% de jabón sobre el metacrilato y la parte adhesiva del vinilo para a continuación juntarlas, esto facilitará el manejo de la posición del vinilo una vez lo coloquemos sobre el metacrilato. Posteriormente quitaremos las burbujas de aire que han quedado retenidas entre los dos materiales con una tarjeta de plástico para evitar rayar el vinilo, para ello apoyaremos la tarjeta en la parte interna de la burbuja y empujaremos hacia la parte externa, desplazando las burbujas fuera del vinilo. Una vez saquemos todas las burbujas procedemos a cortar las partes sobrantes, para ello ponemos la parte del metacrilato que tiene el vinilo adherido en la parte inferior y lo apoyamos sobre una madera para evitar rayar la superficie donde estemos cortando y procedemos a cortar con la ayuda de un cúter (ver Fig. 14).

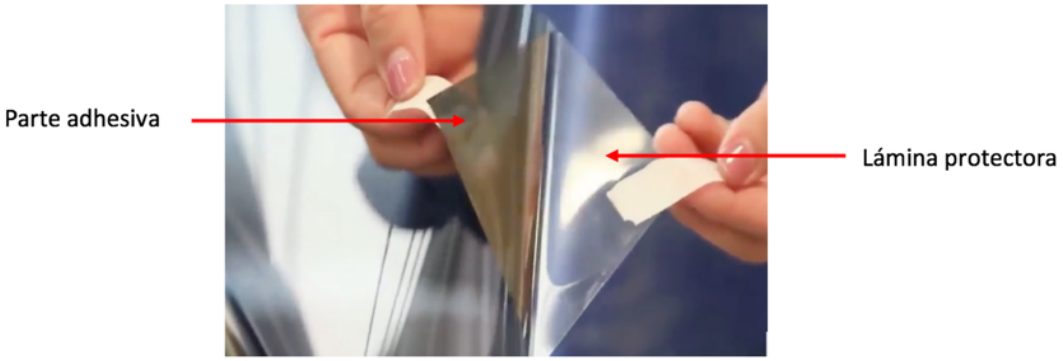


Figura 13. Separación de la lámina protectora del vinilo (tomado de [5]).

Figura 14. Recortando el exceso de vinilo

5.2.3 Colocación de la tira de LEDs.

Antes de unir el espejo al marco debemos colocar la tira de LEDs en su sitio. Primero se deberá recortar la tira con la longitud deseada, para ello lo colocamos en el sitio sin pegar y miramos aproximadamente cuánto debe medir, para posteriormente recortar por la unión más cercana y que aporte menor longitud a la tira (ver Fig. 15). Para realizar una conexión más segura y conseguir la longitud de cable que queramos, se proceden a soldar unos cables a las conexiones de la tira de LEDs (ver Fig. 16). Posteriormente se coloca en el marco con la cinta de doble cara que incluye en su parte trasera (ver Fig. 17).

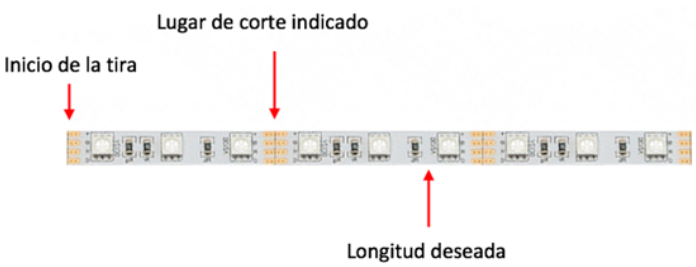


Figura 15. Lugar de corte indicado en la tira de LEDs

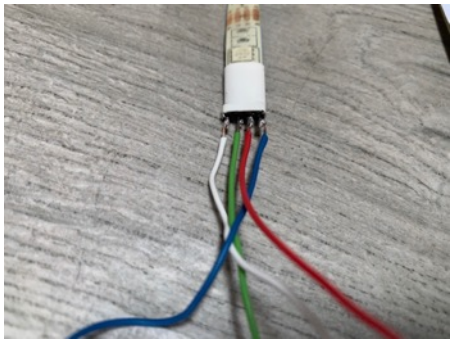


Figura 16. Soldadura de cables a la tira de LEDs

Figura 17. Tira de LEDs colocada en el marco

5.2.4 Fijación del metacrilato al marco

A continuación colocamos el espejo dieléctrico sobre el marco para situar los puntos donde colocaremos los tornillos con los embellecedores (ver Fig. 18). Una vez marcado, procederemos a realizar las perforaciones en el espejo con la ayuda de un taladro y una broca. Una vez hecho los agujeros colocamos el espejo sobre el marco y los fijamos con la ayuda de los tornillos, los cuales quedarán tapados con los embellecedores (ver Fig. 19).

Figura 18. Ambas partes del embellecedor y el tornillo utilizados.

Figura 19. Espejo atornillado al marco con embellecedores (rodeados de rojo).

Una vez tenemos todo montado se coloca la pantalla en su sitio, quedando encajada perfectamente sin la necesidad de ninguna fijación. En las partes que no han sido ocupadas por la pantalla colocaremos goma EVA de color negro para que no atraviese la luz por estas zonas (ver Fig. 20). Por otro lado sellaremos las uniones con cinta aislante negra para evitar cualquier paso mínimo de luz.

Figura 20. Prototipo montado con monitor y goma EVA.

5.2.5 Colocación del resto de componentes

Finalmente añadimos la regleta de 4 conectores para poder conectar todos los dispositivos. También colocaremos el microprocesador con su fuente de alimentación, y conectado a la pantalla. Conectaremos la fuente de alimentación de la pantalla. Por otro lado situaremos el microprocesador con el circuito electrónico creado en la protoboard y lo conectaremos a la tira de LEDs (ver Fig. 21). Para fijar todos los elementos en los sitios correspondientes utilizaremos cinta de doble cara de espuma.

Figura 21. Montaje final por la parte trasera.

Figura 22. Montaje final por la parte delantera.

5.3 Diseño del sistema electrónico para el control de la tira de LEDs

Para controlar los diferentes parámetros de la tira de LEDs utilizaremos la placa de desarrollo. Esta tira de LEDs es de ánodo común por lo que controlando la tensión de cada canal podremos controlar el brillo de éstos independientemente. Sin embargo, si conectamos directamente las salidas de la placa de desarrollo a la tira de LEDs no conseguiremos el funcionamiento adecuado ya que la placa de desarrollo permite la circulación de una corriente máxima por cada entrada/salida de 15 mA la cual no es suficiente ni para iluminar un solo LED. Por este motivo se utilizarán transistores para controlar el flujo de corriente en los diferentes canales. Se han realizado los siguientes cálculos para concretar las especificaciones electrónicas necesarias para el correcto funcionamiento de nuestra tira de LEDs.

Se puede observar en el controlador original que posee 3 salidas de 1.5 A cada una (ver Fig. 23). Por otro lado actualmente podemos destacar dos tipos de transistores los BJT y los MOSFET. Los primeros están formados por tres elementos: emisor, colector y base. Estos transistores son conmutados por corriente, es decir necesitan de una corriente de base para ser activados. Encontramos dos tipos, los NPN y los PNP. Los primeros cuando están activos permiten el paso de la corriente del colector al emisor. Sin embargo en los PNP una vez activos permiten el paso de la corriente del emisor al colector. Una vez activo existe una ganancia entre la corriente de colector y la corriente de base.

Figura 23. Controlador original de la tira de LEDs.

Otra opción es trabajar con transistores MOSFET los cuales también están formados por tres elementos: drenador, puerta y surtidor. Estos transistores son conmutados por tensión, es decir necesitan de una tensión en la puerta mayor que la tensión de mantenimiento para permitir el paso de corriente entre el drenador y el surtidor. Por lo que nos permitiría controlar la intensidad que circula por nada canal de los tres colores sin necesidad de circular corriente por los pines de la placa de desarrollo. Existen dos tipos de MOSFET, los de tipo *n* y los de tipo *p*. Los primeros conmutan ante una tensión positiva en la puerta y mayor que la tensión de mantenimiento. Los de tipo *p* al contrario, conmutan ante una tensión negativa y mayor que la tensión de mantenimiento (en valor absoluto). Sin embargo estos no pueden ser utilizados con nuestra placa de desarrollo ya que ésta ofrece una tensión máxima en sus salidas de 3.3 V lo cual no es suficiente para conmutar la gran mayoría de MOSFETs. Así se van a utilizar los transistores BJT BD139 de *STMicroelectronics* (Ginebra, Suiza). Estos transistores son de tipo NPN, tienen una ganancia entre 25 y 250, permiten una corriente máxima de 1.5 A y una tensión máxima de 80 V entre colector y emisor. Una vez escogido el transistor que se va a utilizar se calculará el valor adecuado de la resistencia de base, la cual convertirá la tensión de salida de la placa de desarrollo en una corriente de base.

Podemos diferenciar tres zonas de trabajo: corte, trabajo y saturación. En la zona de corte, la corriente de base no es suficiente para conmutar el transistor, por lo que la corriente de colector será nula. En la zona de trabajo la corriente de colector será proporcional a la corriente de base. Por último en la zona de saturación, la corriente de colector será máxima. Nuestra intensión es que el transistor trabaje en la zona de saturación cuando apliquemos la tensión máxima de salida en nuestra placa de desarrollo. Así, debemos conseguir una corriente de colector de 1.5 A con una tensión de base de 3.3 V. Para ello debemos primero obtener la ganancia exacta de nuestros transistores con la ayuda de un multímetro digital (ver Fig. 24). Obteniendo como ganancias 145, 151 y 153.Una vez tenemos estas ganancias, sabemos que la $Ic = \beta \cdot Ib$, donde β es la ganancia. Así deducimos que la corriente de base debe ser entre 10.3 mA y 9.8 mA. Esa corriente de base debe ser originada por una tensión de 3.3 V. Así, empleando la Ley de Ohm ($V = I \cdot R \rightarrow V/I = R$) deducimos que el valor de la resistencia debe ser entre 320.4 Ω y 336.7 Ω. Por esto se utilizarán 3 resistencias de 330 Ω, ya que es el valor de resistencia comercial más próximo al valor deseado. La única diferencia que notaremos de usar la resistencia calculada a la resistencia elegida es que llegaremos un poco antes a la corriente de saturación, o no llegaremos a esta por muy poco, algo inapreciable a simple vista.

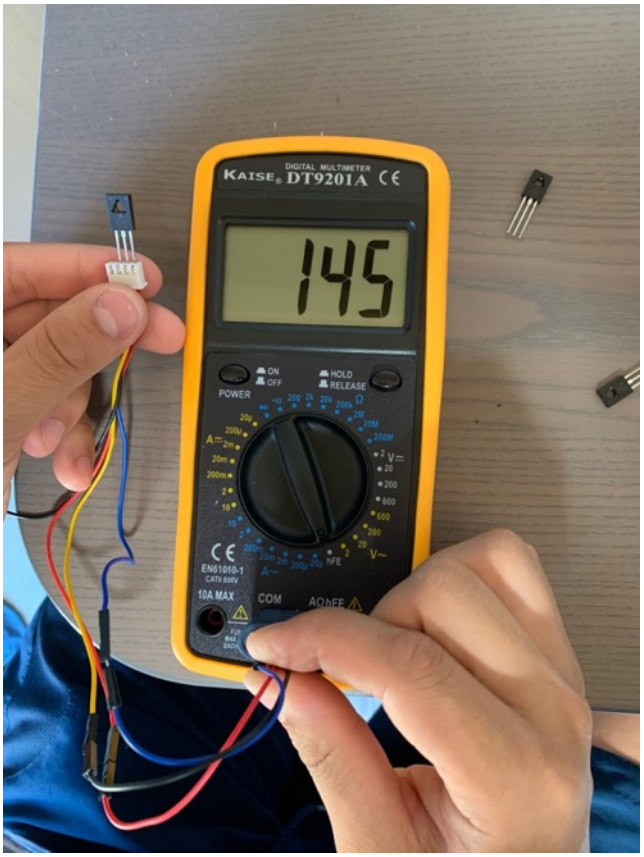


Figura 24. Medición de la ganancia exacta de un transistor.

Una vez escogidos los valores de las resistencias realizamos el diseño del montaje en Proteus (ver Fig. 25). Para posteriormente realizar la construcción en físico (ver Fig. 26).

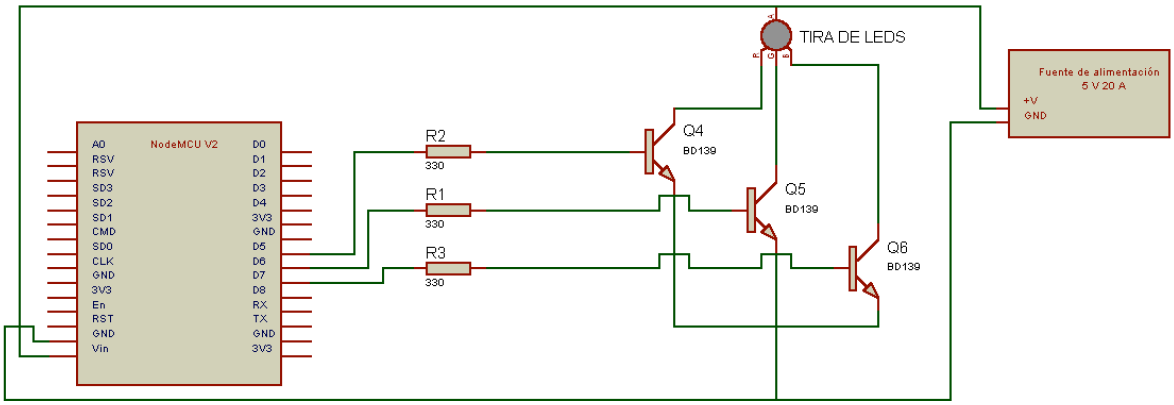


Figura 25. Circuito electrónico diseñado en Proteus.

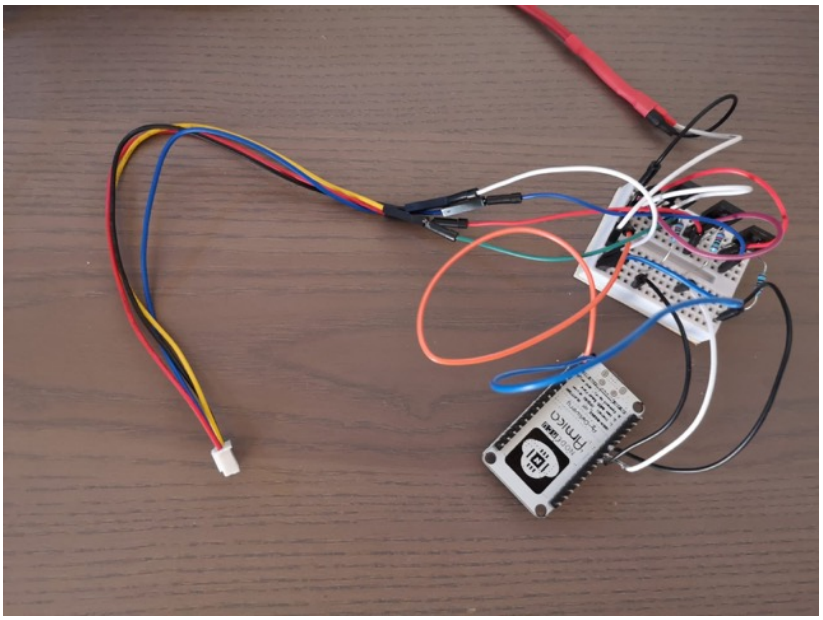


Figura 26. Circuito electrónico para el control de una tira de LEDs mediante la voz.

5.4 Programación de la placa de desarrollo

La programación de la placa de desarrollo se realizará mediante el software *Arduino.* Para ello se seguirá el siguiente flujograma (ver Fig. 27).

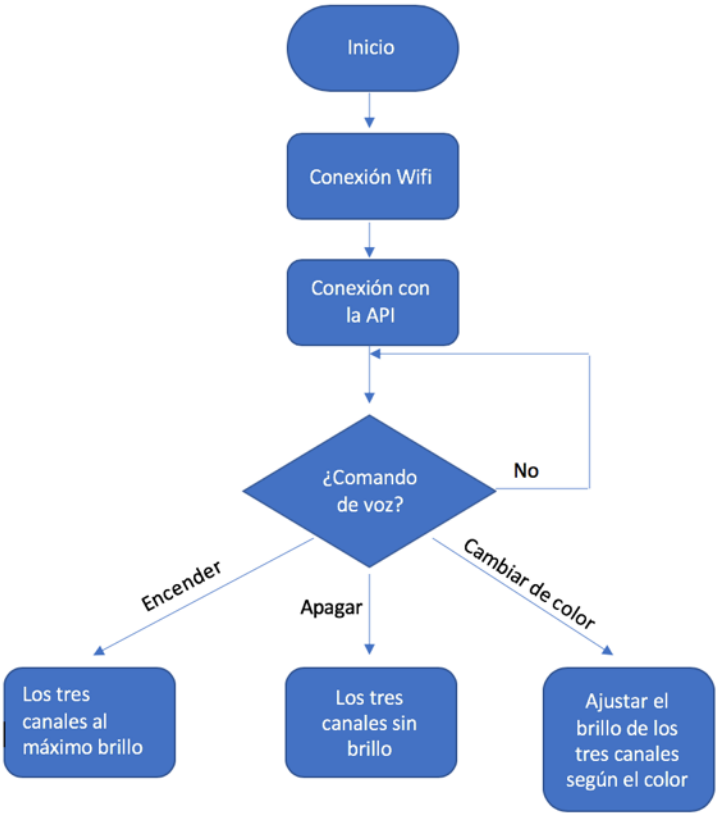


Figura 27. Flujograma de tareas de la placa de desarrollo

Para la programación utilizaremos la API (interfaz de programación de aplicaciones) Sinric, esta nos permitirá registrar un dispositivo en su aplicación para Alexa para así luego poder controlarlo. En la propia página encontramos un enlace a un artículo de Github con varios códigos de ejemplo entre los cuales encontramos uno para el control de una luz (*light_example.ino*). En este código tenemos que realizar varios cambios. Primero debemos cambiar donde pone *#define MyApiKey "xxxxx"*. Aquí debemos sustituir xxxxx por nuestra clave de Sinric que obtendremos de la propia página web una vez nos hayamos registrado. Posteriormente encontraremos *#define MySSID "xxxxx"* donde debemos sustituir xxxxx por el nombre de nuestra red Wifi. Seguidamente encontraremos *#define MyWifiPassword "xxxxx"* donde debemos sustituir xxxxx por la contraseña de nuestra red Wifi. Una vez hecho estos cambios, procedemos a cambiar las funciones. En el código de ejemplo encontramos las funciones de encender y apagar, estas tendrán una función de declaración así *deviceId == "5axxxxxxxxxxxxxxxxxxx"*, aquí debemos de sustituir 5axxxxxxxxxxxxxxxxxxx por el identificador que nos dará Sinric cuando registremos un dispositivo.

Posteriormente encontraremos un apartado donde declara los diferentes comandos que puede recibir desde Alexa. En uno de ellos encontramos *else if(action == "SetColor")* y es aquí donde debemos declarar qué queremos que haga cada canal de led según el color del comando. Para ello subimos el código a la placa de desarrollo y hacemos una

prueba. Al abrir el monitor serial, vemos que cuando mandamos un comando a Alexa de cambiar el color, este lo manda a la aplicación mediante un tono de color en lugar del nombre del color. Así por ejemplo para el color rojo nos devuelve un *Hue=0*. Así en nuestra programación pondremos la siguiente función.

```
if (0 == hue.toInt()){

analogWrite(12, 1023); //R

analogWrite(14, 0);    //G

analogWrite(13, 0);    //B

}
```

Se utilizan las salidas 12, 13 y 14 ya que en las especificaciones del producto vemos que para los pines D5, D6 y D7 son dichas salidas (ver Fig. 28). Con este código se otorgará al canal rojo el brillo al máximo brillo. Así, entrando a la aplicación móvil de Alexa y cambiando el color manualmente podemos ver qué nombres de colores reconoce Alexa y declarar qué queremos que haga la tira de LEDs en cada uno de ellos (ver Fig. 29).

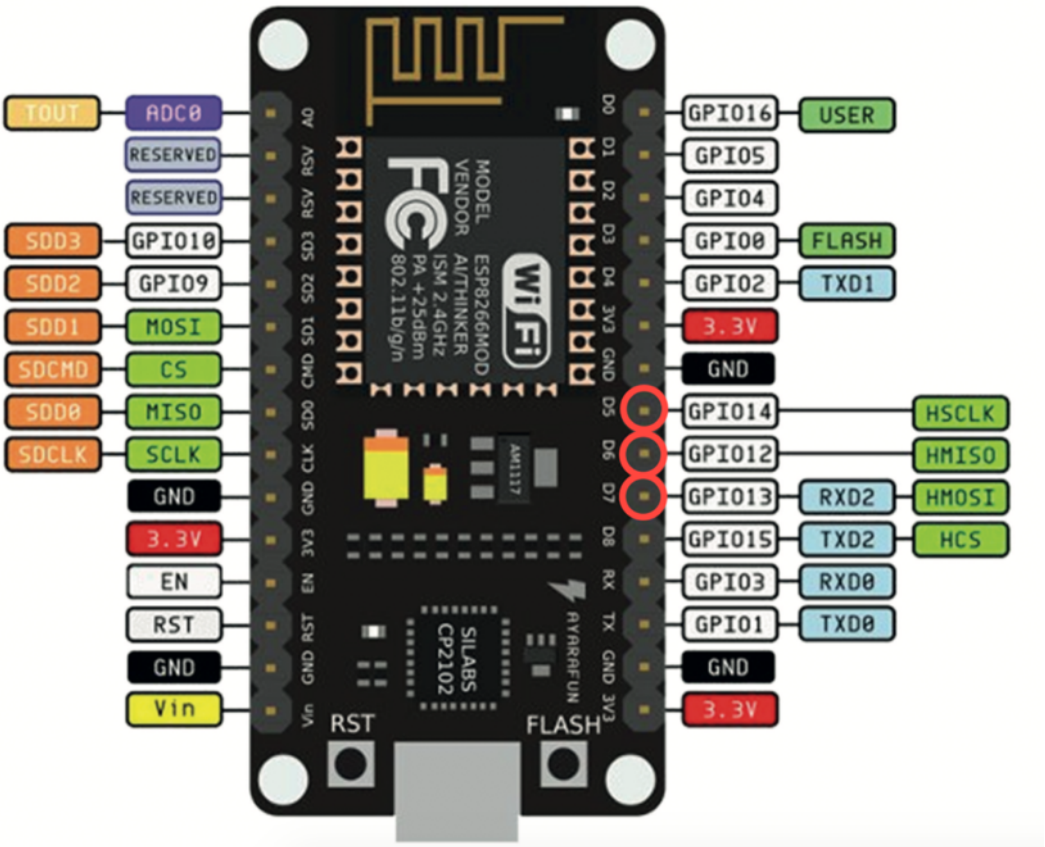


Figura 28. Pines de la placa de desarrollo NodeMCU V2. Se han rodeado con una circunferencia roja las salidas utilizadas (tomado de [6])

Figura 29. Colores reconocidos por Alexa desde la aplicación móvil.

Sin embargo observamos que cuando solicitamos a Alexa poner la tira de LEDs en color blanco, no envía un tono de color, sino una temperatura de color concretamente 4000. Así para definir el color blanco deberemos hacerlo en un apartado diferente. Debemos buscar el apartado donde pone *else if(action == "SetColorTemperature")* y aquí establecer la función para la temperatura de color deseada.

```
if (temp.toInt()==4000){

analogWrite(12, 1023);

analogWrite(14, 1023);

analogWrite(13, 1023);

}
```

Con este código ponemos el brillo de los tres canales al máximo ante el comando de cambiar el color a blanco.

5.5 Programación del microprocesador

La programación de la Raspberry Pi seguirá el flujograma de tareas mostrado en la Figura 30 y la Figura 31, las cuales han sido representadas en dos flujogramas separados por problemas de tamaño si lo representáramos en un único flujograma.

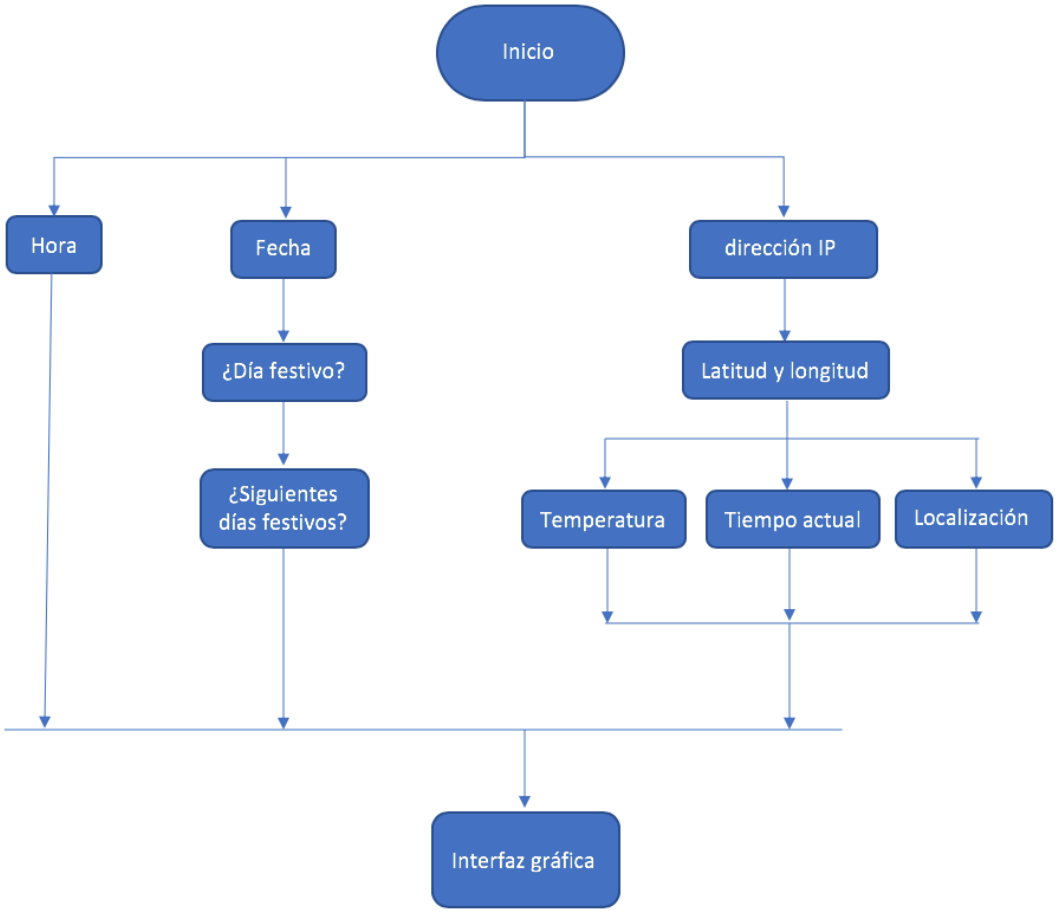


Figura 30. Primera parte del flujograma de tareas del microprocesador.

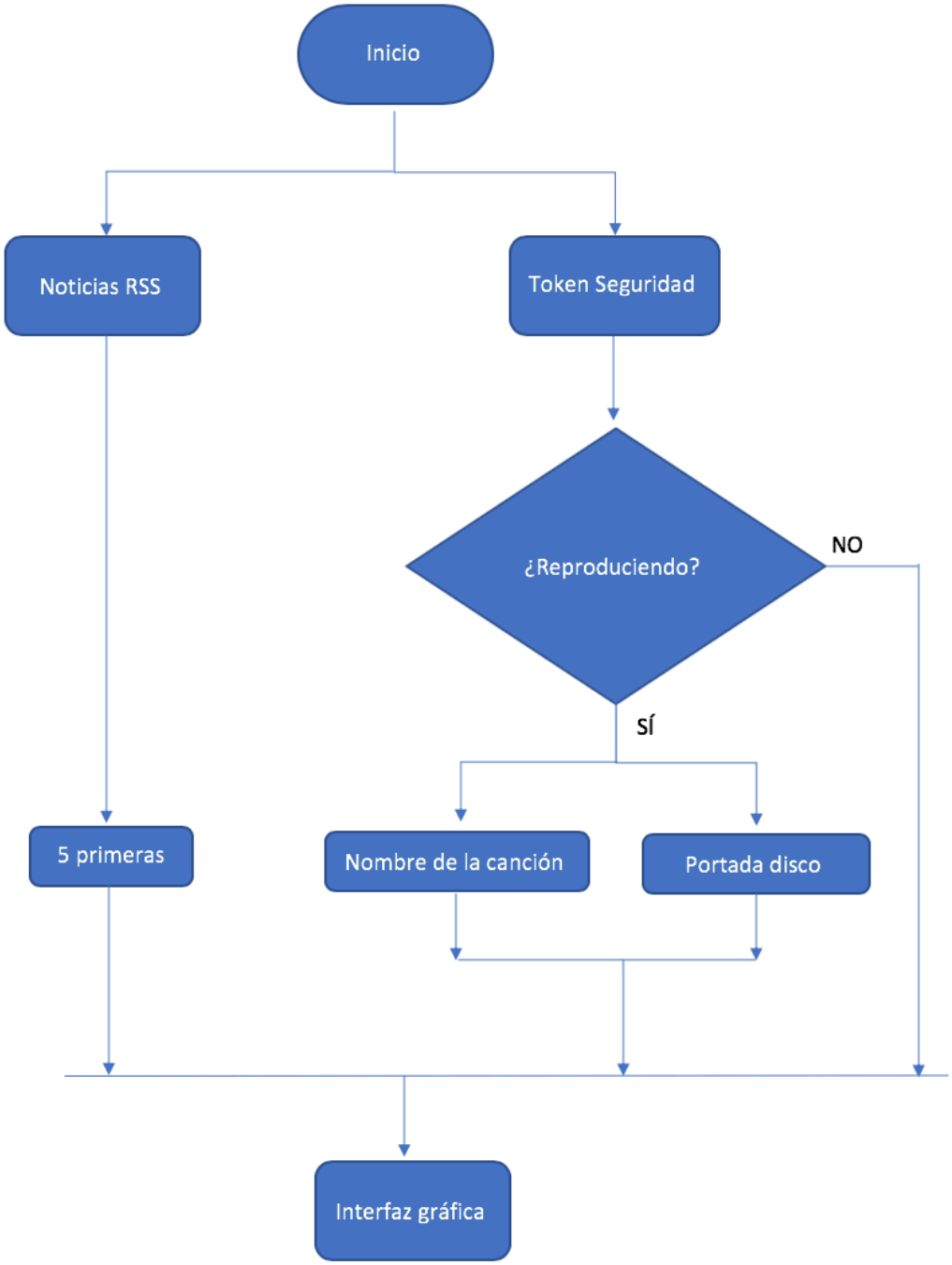


Figura 31. Segunda parte del flujograma de tareas del microprocesador.

Siguiendo los anteriores flujogramas el código del microprocesador hace las siguientes funciones. Crearemos diferentes clases donde agruparemos la informaciñon de cada módulo. Establecemos el local o en caso contrario podemos dejarlo en blanco si no declaramos ninguno tomará el de la Raspberry Pi. Con la información de este local adquirirá la hora y según si definimos la variable *formato_reloj* como 12 o como 24, expresaremos la hora de una forma. Por otro lado del mismo local obtendremos el día de la semana actual y la fecha del calendario. Una vez obtenida la fecha del calendario comprobaremos si esta fecha es un día festivo o no. Si lo es expresaremos el nombre del día festivo, si no lo es expresaremos la fecha y los nombres de los dos siguientes días festivos.

Por otro lado para obtener la información de *Spotify* primero deberemos solicitar el toquen de seguridad, para ello debemos seleccionar nuestro nombre de usuario, la información que nos interesa, el id de cliente, la clave secreta y el enlace de redirección, estos los obtendremos como se explica a continuación en el apartado *SPOTIPY*. Una vez obtenido el toquen de seguridad podremos adquirir la información de *Spotify* y de esa información seleccionar la que sólo nos interesa, en este caso *user-read-currently-playing* que nos devolvará la información de la canción que esté sonando en la cuenta de *Spotify* seleccionada. Una vez obtenida la información de interés, comprobaremos si estamos obteniendo información de *Spotify* o no, si no la obtenemos expresaremos en la interfaz gráfica el título *Spotify*, el logo y la frase *Spotify no iniciado*. En caso de obtener información, comprobaremos si se está reproduciendo actualmente alguna canción o no. Si no se está reproduciendo expresaremos en la interfaz gráfica el mismo título que el caso anterior, el logo y la frase *Pausado*. Pero si sí que se está reproduciendo expresaremos en la interfaz gráfica, el mismo título que los casos anteriores, la portada de la canción y el nombre de ésta.

Para obtener la información climatológica, comprobaremos si hemos declarado alguna latitud y longitud concretas, de no ser así obtendremos la dirección IP de la Raspberry gracias a la API JsonIP. Al ingresar en su página web, ésta nos devolverá nuestra IP en formato JSON la cual parsearemos gracias al módulo con este mismo nombre. Una vez tenemos la IP obtendremos la latitud y longitud de nuestra ubicación gracias a la API IPStack. Al dirigirnos a la dirección web de la API con nuestra IP y la clave que obtenemos al registrarnos, nos devuelve la latitud y longitud de nuestra localización en formato JSON, la cual volveremos a parsear. Una vez tengamos la latitud y la longitud o porque lo hemos declarado manualmente o porque lo acabamos de obtener de la forma que se ha explicado, las emplearemos en la API de *OpenWeatherMap* para obtener la información climatológica. Ésta nos la devolverá también en formato JSON que deberemos de parsear.

Una vez tenemos la información parseada seleccionaremos la que nos interese, en este caso el icono del tiempo actual, el nombre del tiempo actual, el nombre de la localización a partir de la cual se ha obtenido la información y la temperatura. Los iconos los declararemos al principio del código junto a la dirección donde pueden ser encontrados por la Raspberry. Las variables que pueden tomar los iconos los encontraremos en la web de *OpenWeatherMap* y según la nomenclatura que tenga, la vincularemos con la imagen deseada.

Por otro lado para obtener las noticias más relevantes utilizaremos las noticias RSS (Really Simple Syndication) de un periódico digital elegido, en este caso ABC. Primero de todo eliminaremos si hay alguna noticia en la interfaz gráfica. A continuación una vez tengamos la información en formato RSS utilizaremos el módulo *Feedparser* para parsearla. Una vez parseada seleccionaremos las 5 primeras y las expresaremos por la interfaz gráfica incorporando a cada noticia un icono de un periódico, consiguiendo así una diferenciación a simple vista.

Por último creamos una clase genérica que agrupe todas las clases anteriores, ésta nos permite organizar mejor la colocación de las demás clases además de crear configurar una tecla para que active o desactive el modo pantalla completa. Por último cabe destacar que todas las clases se han programado de forma que si ocurre algún error, expresaremos por el terminal en qué clase se ha originado el error.

Respecto al código de programación se ha utilizado el lenguaje *Python* mediante los siguientes módulos.

5.5.1 FEEDPARSER

RSS es un formato empleado para compartir contenidos actualizados frecuentemente, como las novedades o las noticias de una página web. Estos pueden ser leídos sin necesidad de emplear un navegador web, en este caso se va a emplear un módulo de Python encargado de parsear la información obtenida de las RSS. La función principal de este módulo es:

feedparser.parse(RSS_url)

A partir de la cual, obtendremos una lista con las diferentes entradas de la RSS.

5.5.2 REQUESTS

HTTP, siglas de Hypertext Transfer Protocol (protocolo de transferencia de hipertexto) es un protocolo que permite la comunicación en internet. En python tenemos el módulo requests el cual nos facilita el trabajo con HTTP. La función principal de este módulo es:

respuesta = requests.get(“URL”)

Que solicitará una respuesta a la URL deseada y según el contenido de esta, podemos parsearla para obtener la información deseada.

5.5.3 SPOTIPY

Spotify es una aplicación empleada para la música en streaming. Para trabajar esta aplicación en Python se deberá utilizar la librería *spotipy*, que emplearemos de la siguiente manera:

Utilizaremos la función

```
token = util.prompt_for_user_token(username,
                                   ‘user-red-currently-playing’,
                                   client_id='dc97f55ca70e44adb390674543c7c777',
                                   client_secret='a118f7b305d8491e96c3cca8c6863132',
                                   redirect_uri='http://localhost/8888/callback/')
```

Para obtener el token de seguridad a partir de nuestras credenciales e información deseada: nombre de usuario, dato de interés, id del cliente, código secreto y url de redirección.

Para obtener estas credenciales deberemos conectarnos a la plataforma *Spotify for Developers*, en la cual deberemos registrar nuestra app. Una vez registrada, podremos localizar el nombre de usuario, el id del cliente y el código secreto. La url de redirección deberemos establecerla nosotros en la parte superior derecha, clicando en "Edit settings", el cual nos desplegará un menú (ver Fig. 31).

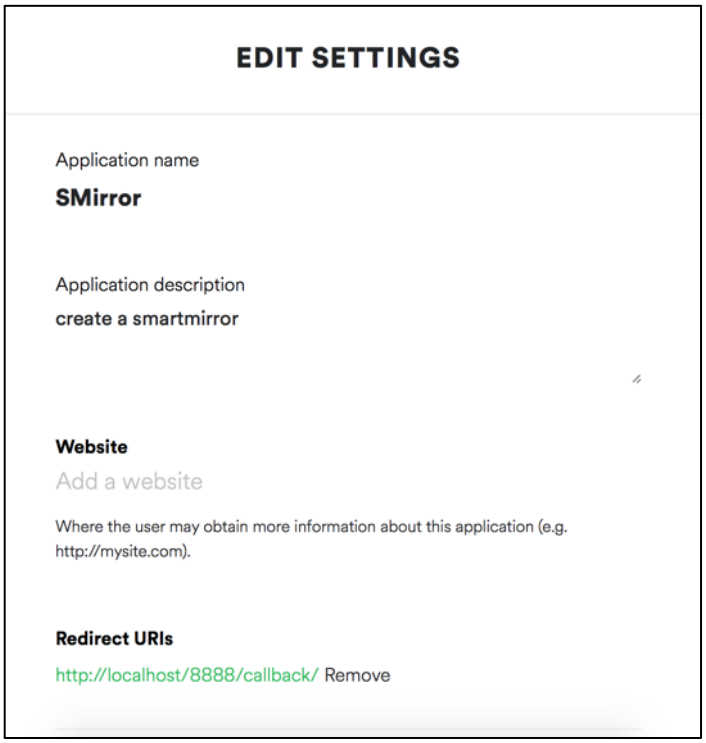


Figura 32. Menú desplegable al registrar un dispositivo en *Spotify for developers*.

Este enlace se encuentra en las instrucciones que podemos encontrar en la plataforma *Spotify for Developers*. Por otro lado, para saber qué debemos como dato de interés en la función, debemos ir al siguiente enlace

https://developer.spotify.com/documentation/web-api/reference/

Y según la información que queramos obtener, seleccionaremos un apartado en el menú izquierdo. En este caso concreto seleccionaremos *Player -> Get the user's currently playing track*. Una vez hemos selecciondo un apartado del menú izquierdo, se nos redirigirá a una página donde podemos encontrar en el apartado *Endpoint,* la variable debemos poner en la función.

Por otro lado debemos declarar una variable que realice la solicitud de la información deseada. Por ejemplo, en este caso:

InformacionSpotify = spotifyObject.currently_playing()

Esto nos devolverá una estructura de objetos a partir de la cual podemos seleccionar los que nos interesen. Además, la información de cada objeto (nombre, tipo de variable y significado) lo podemos encontrar en la misma página donde hemos localizado el *Endpoint.*

5.5.4 JSON

JSON son las siglas de JavaScript Object Notation (Notación de Objetos de Javascript). Este es un formato simple de intercambio de datos, ya que para el humano es intuitivo de entender, y para la máquina es sencillo de generar e interpretar. Además se trata de un formato de texto independiente de cualquier lenguaje de programación pero utiliza estructuras perfectamente entendibles por los programadores de cualquier lenguaje de programación. Dichas características hacen de este formato el ideal para intercambiar datos.

Python cuenta con un módulo a partir del cual utilizando la función:

json.loads (texto)

Podemos parsear cualquier texto en este formato

5.5.5 HOLIDAYS

Este módulo nos permite determinar si una fecha concreta es un día festivo en un determinado país, provincia o estado. Las funciones principales de este módulo son

vacaciones = holidays.Spain()

Donde declaramos una variable que agrupará todos los días festivos del lugar que determinemos, en este caso España. Una vez hecho esto, hay diferentes formas para comprobar si un determinada fecha es festivo o no, en este caso se va a utilizar:

'día/mes/año' in holidays

El cual nos devolverá un True si el día especificado es festivo o un False si no lo es. Una vez sepamos que el día sí que es festivo, podemos comprobar qué día festivo se comemora de la siguiente forma:

holidays.get('día/mes/año)

Devolviendonos en este caso "Año Nuevo"

5.5.6 TKINTER

Tk es una biblioteca gráfica disponible para varios lenguajes de programación, entre los cuales, Tkinter es la capa de este biblioteca utilizada para el lenguaje de programación Python. Es considerado uno de los estándares para la interfaz gráfica en Python y uno de los referentes como herramienta didáctica debido a su sencillez. Gracias a este módulo se puede interactuar con el usuario, como por ejemplo, solicitándole inputs como el ingreso de datos y devolviéndole outputs en la interfaz gráfica. Las funciones principales de este módulo son:

class Nombre(Frame):

Esta función creará un elemento de la clase Frame, definida en el módulo TKinter. Por otro lado nos permitirá crear diferentes subgrupos donde podremos situar los elementos gráficos como imágenes o textos. Una vez creada la clase deberemos utilizar las siguientes funciones:

Frame.__init__(self, parent, bg='color')

Esta función nos permitirá inicializar el frame con las características deseadas. Una vez inicializado se puede añadir diferentes elementos. Por ejemplo, para añadir un texto lo haremos de la siguiente manera

self.Nombre = Label(self, font=('Fuente', tamaño), fg="Color del texto", bg="Color del fondo", text= “texto”)

Para añadir una imagen, primero debemos abrir la imagen con la función:

Nombredelaimagen = Image.open (“Direcciónde larchivo/Nombre.extensión”)

Y posteriormente crear un label que contendrá a la imagen

self.Nombre = Label (self, bg= “Colordefondo”, image=Nombredelaimagen)

Por otro lado, para indicar la posición de estos elementos dentro de los frames utilizaremos la función *pack* de la siguiente forma:

self.Nombre.pack(side=Posición, anchor = Parámetro)

Donde *Posición* puede tomar los valores de TOP, BOTTOM, LEFT y RIGHT. Y donde *anchor* indica la posición del punto de referencia, pudiendo tomar los siguientes valores indicados en la imagen.

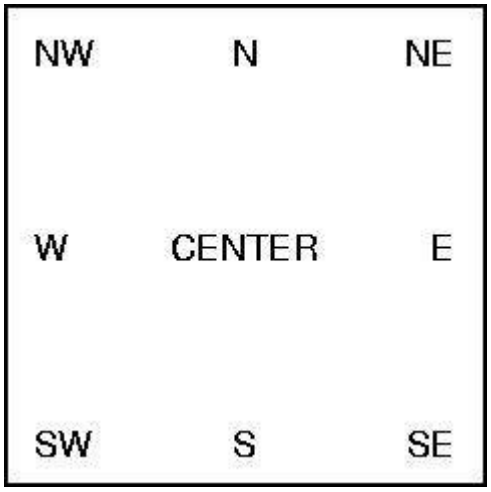


Figura 33. Valores que puede tomar la variable *anchor.*

Por otro lado, se deberá emplear la siguiente función en el programa principal.

Nombre = Tk()

Esta nos permitirá crear la raíz o ventana principal que agrupará los diferentes widgets creados. Y por último se crea el bucle de la aplicación con la siguiente función:

Nombre.mainloop()

Esta hará una función similar a *while True*

5.5.7 TIME

Este módulo proporciona las principales funciones relacionadas con la fecha y hora. Principalmente se empleará la función:

time.strftime(formato[,t])

Esta devuelve los datos de tiempo como una cadena con el formato declarado. Si *t* no está declarado, los datos del tiempo son recogidos desde el local. ValueError se alcanza si en algún momento, *t* se encuentra fuera del rango permitido. Por otro lado *formato* debe ser un string formado a partir de determinados parámetros (ver Tabla 3).

Tabla 3. Parámetros empleados en el módulo Time.

Parámetro	Significado
%d	Día del mes como número decimal [01, 31]
%m	Mes como número decimal [01, 12]
%Y	Año
%H	Hora (reloj 24H) como número decimal [00, 23]
%I	Hora (reloj 12H) como número decimal [01, 12]
%M	Minuto como número decimal [00, 59]
%p	AM o PM

6. CONCLUSIONES

Se ha diseñado y construido un prototipo de espejo inteligente basado en una pantalla de alta resolución y capacidad de ser controlado por la voz, cumpliendo así los objetivos establecidos.

No obstante quedan muchos aspectos por mejorar. Por ejemplo, durante el montaje físico el acabado del vinilo no ha sido perfecto, quedando pequeñas marcas apreciables a corta distancia. Por otro lado se adquirió un vinilo unidireccional con efecto azulado pensando que este efecto era únicamente apreciable en pequeños matices reflejados. Sin embargo la luz que atraviesa el espejo proveniente de la tira de LEDs es afectada por esta tonalidad, quedando un ligero desvío del tono de la iluminación hacia el color azul. También como mejora futura se podría incorporar una tira de LEDs con mayor potencia ya que la actual no ofrece suficientemente brillo como para ser de utilidad iluminando, dejando así como única función la decoración.

Por otro lado en aspectos de programación hemos encontrado el problema de que la ubicación a partir de la IP a veces varía, llegando incluso a marcar tu ubicación en otra provincia. Como mejora futura esto se podría solucionar creando un interfaz que solicitara al usuario el nombre de su localidad, provincia y país. Por lo que buscaríamos la información meteorológica a partir de esta.

Por último podemos decir que se trata realmente de un proyecto al que se le pueden implementar infinidad de mejoras y funciones, siendo así configurable para todos los gustos y necesidades. Como por ejemplo reproducir videos de tu móvil, crear una aplicación móvil para cambiar la posición de los elementos dentro de la interfaz gráfica o que el dispositivo en lugar de ser un complemento para un asistente de voz, fuese directamente el asistente de voz.

7. REFERENCIAS

1. https://www.youtube.com/watch?v=t-gSjdJX5HY (accedido el 24/02/2020)
2. https://www.youtube.com/watch?v=NlMEm5hDnb4 (accedido el 07/03/2020)
3. https://www.xataka.com/ordenadores/raspberry-pi-4-analisis-caracteristicas-precio-especificaciones (accedido el 10/05/2020)
4. https://computerhoy.com/noticias/tecnologia/raspberry-pi-4-vs-raspberry-pi-3-mejora-nuevo-modelo-443801 (accedido el 10/05/2020)
5. https://www.amazon.es/TTMOW-Pel%C3%ADcula-Unidireccional-Protector-Privacidad/dp/B085RKX3P5/ref=sr_1_2_sspa?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&dchild=1&keywords=vinilo%2Bespejo&qid=1595780940&sr=8-2-spons&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUExUFREQUZBQVI2Q0ZQJmVuY3J5cHRlZElkPUExMDE3NzQ5VFowSEpHVjlZMkdRJmVuY3J5cHRlZEFkSWQ9QTA5NDMwOTUxVUZJN0hKNjIwRElKJndpZGdldE5hbWU9c3BfYXRmJmFjdGlvbj1jbGlja1JlZGlyZWN0JmRvTm90TG9nQ2xpY2s9dHJ1ZQ&th=1 (accedido el 23/04/2020)
6. https://cdn.shopify.com/s/files/1/1509/1638/files/NodeMCU_LUA_Amica_V2_Pinout_Diagram.pdf?14596320546790113351 (accedido el 02/06/2020)
7. https://alfredodacosta.com.ar/wp/problemas-con-el-hdmi-en-raspberry-pi/ (accedido el 10/04/2020)
8. http://www.ipv6.mx/index.php/component/content/article/189-ipv4-vs-ipv6-icual-es-la-diferencia (accedido el 09/06/2020)
9. https://python-para-impacientes.blogspot.com/2017/03/el-modulo-time.html (accedido el 30/05/2020)
10. https://guia-tkinter.readthedocs.io/es/develop/chapters/1-intro/1.1-About_Tk.html (accedido el 04/06/2020)
11. https://es.wikipedia.org/wiki/RSS (accedido el 18/06/2020)
12. https://www.pythonforbeginners.com/feedparser/using-feedparser-in-python (accedido el 18/06/2020)
13. https://pypi.org/project/holidays/ (accedido el 25/06/2020)
14. https://www.json.org/json-es.html (accedido el 19/06/2020)
15. https://www.tutorialspoint.com/python/tk_anchors.htm (accedido el 04/06/2020)
16. https://randomnerdtutorials.com/esp8266-pwm-arduino-ide/ (accedido el 15/07/2020)
17. http://panamahitek.com/que-es-y-como-funciona-un-mosfet/ (accedido el 20/07/2020)

8. ANEXO 1:

Configuración del microprocesador Raspberry Pi 4.

Lo primero que debemos hacer es instalar el SO (sistema operativo) en la tarjeta microSD. Ésta la insertamos en el ordenador mediante el adaptador microSD-USB adquirido. Para descargar el SO debemos ir a la página oficial de Raspberry y dentro de ella al apartado “Descargas”. A partir de aquí tenemos dos opciones: descargar directamente el SO o descargar el instalador *Raspberry Pi imager*.

Si decides descargar una imagen del SO. Dentro de dicha página encontramos dos imágenes de sistemas operativos: Raspbian y Noobs. Raspbian es el sistema operativo oficial de Raspberry, éste lo instalas y ejecutas directamente. Mientras que NOOBS es un instalador alternativo a partir del cual puedes elegir entre diversos SOs, una vez instalado, al iniciar la Raspberry Pi podrás escoger qué SO deseas utilizar. Por otro lado puedes descargar *Raspberry Pi imager*. Éste nos permitirá escoger desde nuestro ordenador el sistema operativo que deseemos instalar.

En este caso utilizaremos el primer método y lo utilizaremos para instalar Raspbian, ya que es el sistema operativo más completo e intuitivo. Concretamente se descargará *Raspbian con escritorio* para hacer aún más sencilla su utilización. Por otro lado deberemos descargar *Balena Etcher* para instalar Raspbian en la tarjeta MicroSD. Una vez instalado el SO, insertamos la MicroSD en la Raspberry y conectamos el teclado, el ratón y el monitor.

Una vez hecho esto se enciende la *Raspberry Pi*. Si vemos que la pantalla no detecta señal se debe volver a conectar la tarjeta microSD al ordenador. Dentro de esta tenemos un archivo llamado config.txt. y en este archivo debemos descomentar *hdmi_force_hotplug=1* o *hdmi_force_hotplug=2* dependiendo de a qué salida HDMI vayamos a conectar la pantalla.

Una vez hecho esto ya podemos comenzar con la configuración de la *Raspberry Pi* (idioma, zona horaria, etc). Ahora para trabajar con la *Raspberry* tenemos dos opciones. Desde nuestro ordenador o desde la *Raspberry* directamente.

El método más sencillo es el de trabajar directamente sobre la *Raspberry*, para ello debemos conectar a ésta un monitor, un teclado y un ratón. Sin embargo, aun necesitando una mayor configuración, es más cómodo trabajar desde el ordenador personal con el fin de ahorrar espacio. Para ello debemos seguir los siguientes pasos.

Primero debemos conectar un teclado, ratón y monitor como en el primer método. Seguidamente activaremos la conectividad VNC (Virtual Network Computing) en nuestra *Raspberry*. Para ello clicamos al icono de *Raspbian*, situado en la parte superior izquierda y posteriormente vamos al apartado de *Preferencias* y dentro de ésta nos dirigiremos a *Configuración de la Raspberry*. En el menú superior debemos seleccionar Interfaces y activar VNC (ver Fig. 34). Una vez estamos aquí, también podemos activar SSH (Secure Shell), esto nos permitirá intercambiar archivos con nuestro pc. Así podemos ir guardando copias de seguridad de nuestro código escrito en nuestro ordenador personal.

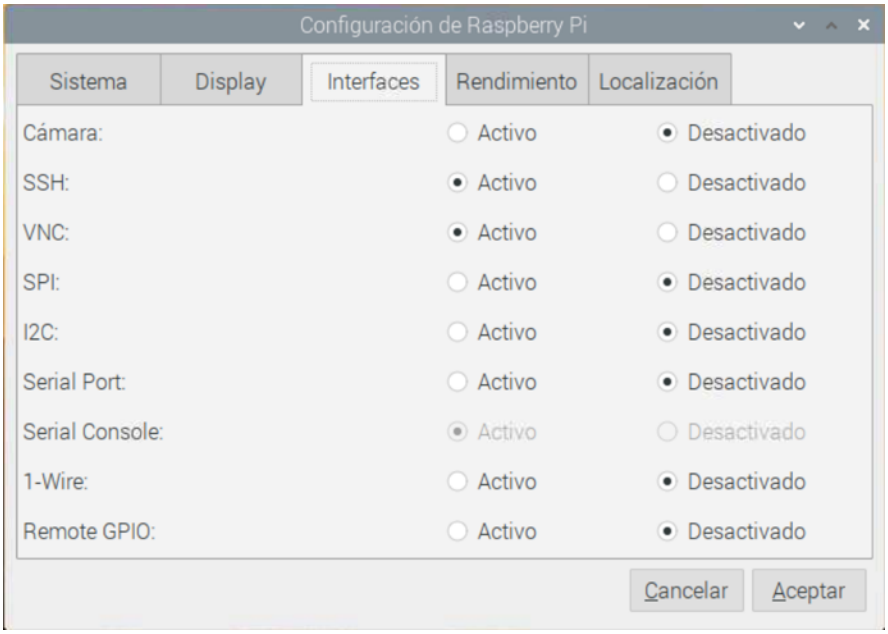


Figura 34. Menú de interfaces. Accedido desde *Configuración de la Raspberry*.

Una vez activado, en la barra superior nos saldrá el icono VNC, si lo clicamos nos saldrá la dirección IP de nuestra *Raspberry* (ver Fig. 35), la cual deberemos apuntar para el paso siguiente.

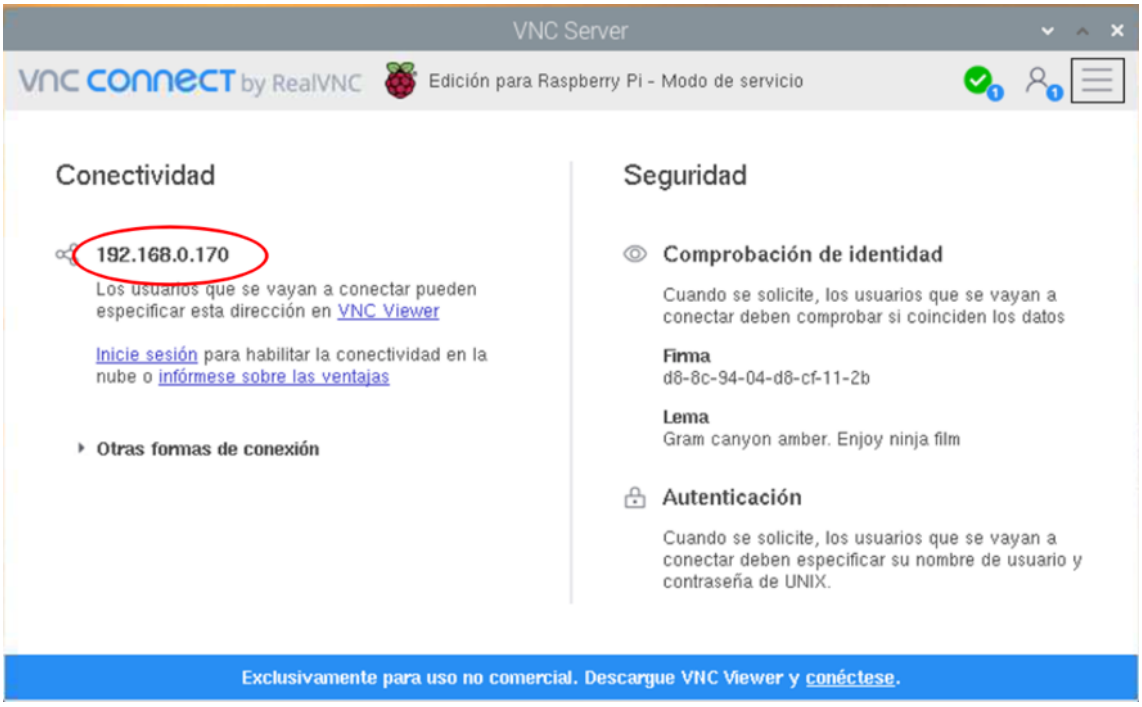


Figura 35. Menú VNC de la *Raspberry Pi*.

Por otro lado, debemos descargar el programa *VNC Viewer* que encontraremos en el siguiente enlace.

https://www.realvnc.com/es/connect/download/viewer/

Una vez descargado e instalado, lo iniciamos. En la barra superior debemos escribir la dirección IP de nuestra Raspberry y le damos a la tecla *enter*. Nos pedirá el usuario y contraseña que establecimos al configurar la *Raspberry*. Una vez los ponemos ya podremos controlar la *Raspberry* desde nuestro ordenador.

A continuación, deberemos establecer una IP estática, esto se debe a que la IP preestablecida al ser dinámica va cambiando con cada reinicio, por lo que perderíamos la IP de nuestro microprocesador al apagarlo o reiniciarlo. Podemos establecer IPs estáticas para las diferentes conectividades que tenemos, sin embargo como para este proyecto solo vamos a utilizar wifi, por lo que estableceremos ésta como estática. Para esto lo primero que vamos a hacer va a ser averiguar la IP que tiene asignada la *Raspberry* actualmente, con este fin abriremos un nuevo terminal y escribiremos "ifconfig" (ver Fig. 36).

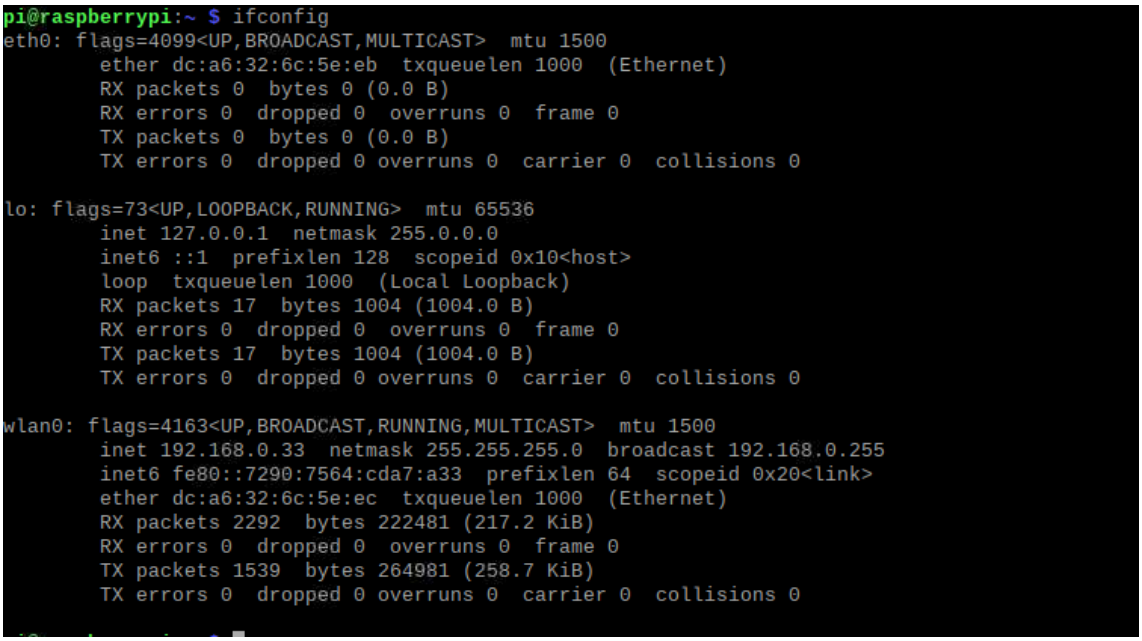


Figura 36. Respuesta del terminal al escribir *ifconfig.*

Aquí observaremos el último apartado, donde pone wlan0 ya que hace referencia al Wifi y anotaremos la IP que aparece en inet, en este caso 192.168.0.33. A continuación vamos al icono de Wifi de la barra superior, hacemos clic derecho sobre él y vamos a *Wireless & Wired Network Settings* (ver Fig. 37).

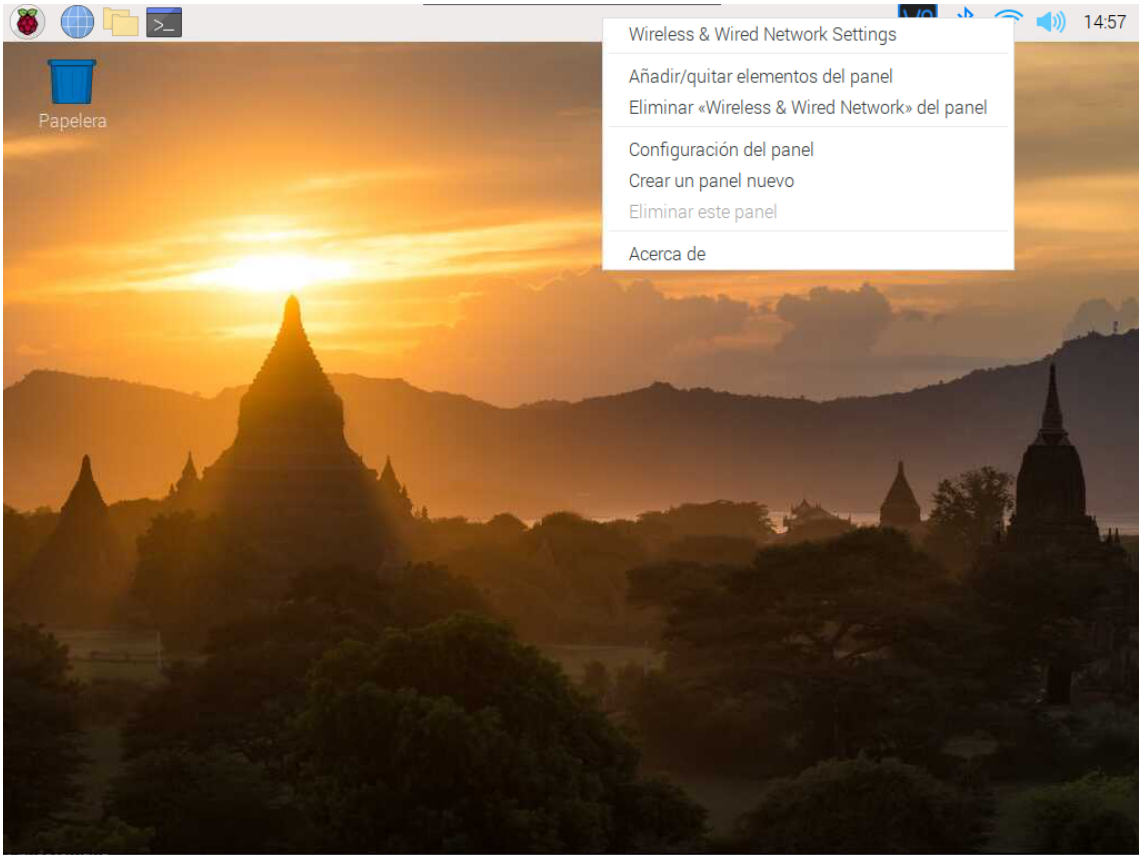


Figura 37. Escritorio del SO Raspbian. En este caso con el menú de configuración de red desplegado.

Aquí debemos seleccionar las opciones *Interface* y *wlan0* (ver Fig. 38). Es aquí donde copiaremos en la casilla donde pone IPv4 la IP anotada anteriormente.

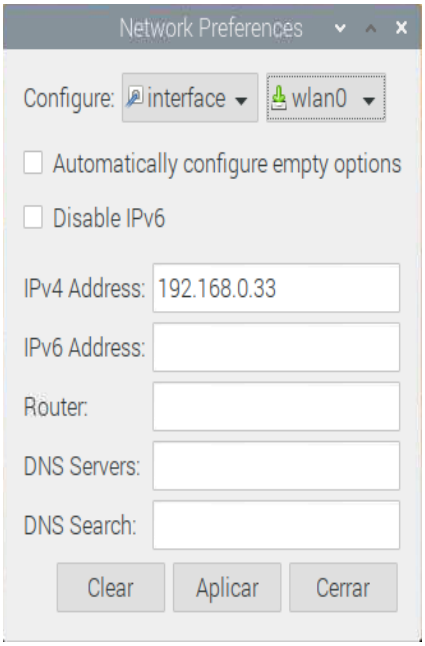


Figura 38. Menú de preferencias de red. Aquí escribiremos la dirección IP estática.

También podríamos introducir la IPv6, pero considerando que es una IP privada y no vamos a tener tantos dispositivos conectados como para necesitar Ipv6. Prescindiremos de ella ya que la única diferencia con la Ipv4 es que nos permite registrar más dispositivos. Por otro lado, si queremos utilizar otra dirección IP, deberemos asegurarnos de que no esté utilizada. Para ver cuales están utilizadas tenemos muchos métodos, como usar programas específicos o entrar en la configuración del *router Wifi*. Por último, deberemos desactivar el salvapantallas, esto evitará que la *Raspberry* apague automáticamente la pantalla cuando iniciemos el espejo inteligente. Para ello volvemos al icono de *Raspbian* → Preferencias → Configuración de Raspberry y desactivamos *Splash Screen* (ver Fig. 39).

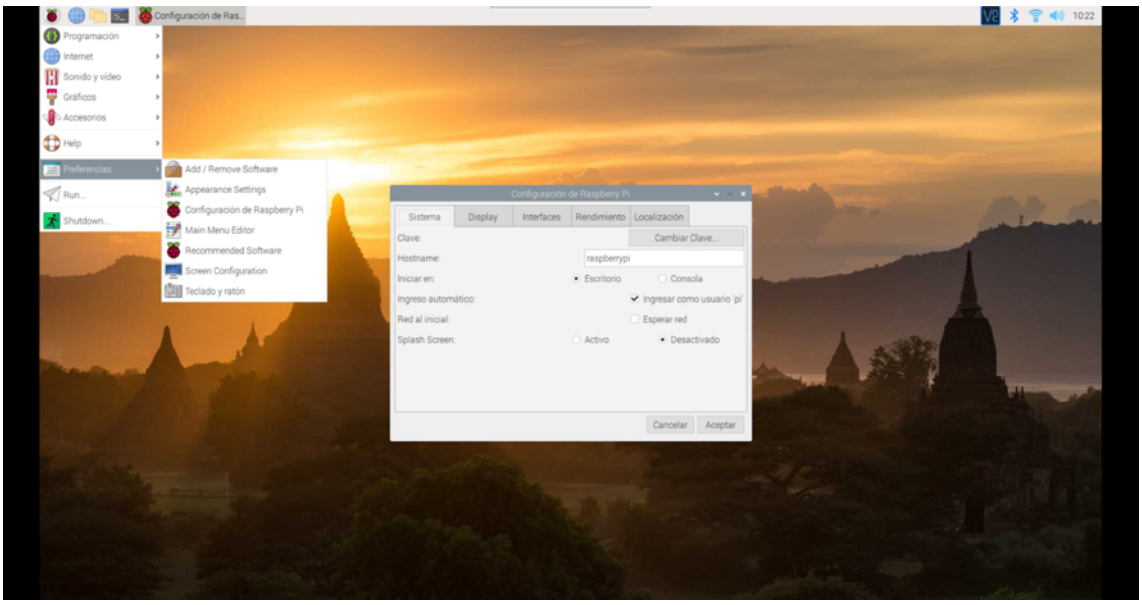


Figura 39. Desactivación del salvapantallas. En este caso encontramos el menú de preferencias desplegado, la ventana de configuración de la Raspberry abierta y *Splash Screen* desactivado.

Una vez hecha la configuración inicial vamos a crear una carpeta llamada SMirror, es en esta donde guardaremos todos los archivos del proyecto. Posteriormente se crea un archivo llamado “SMirror.py dentro de la carpeta SMirror, daremos click derecho con el ratón y posteriormente a *Nuevo archivo*. Para que se nos abra con el editor de *Python*, le daremos click derecho y *Abrir con* (ver Fig. 40).

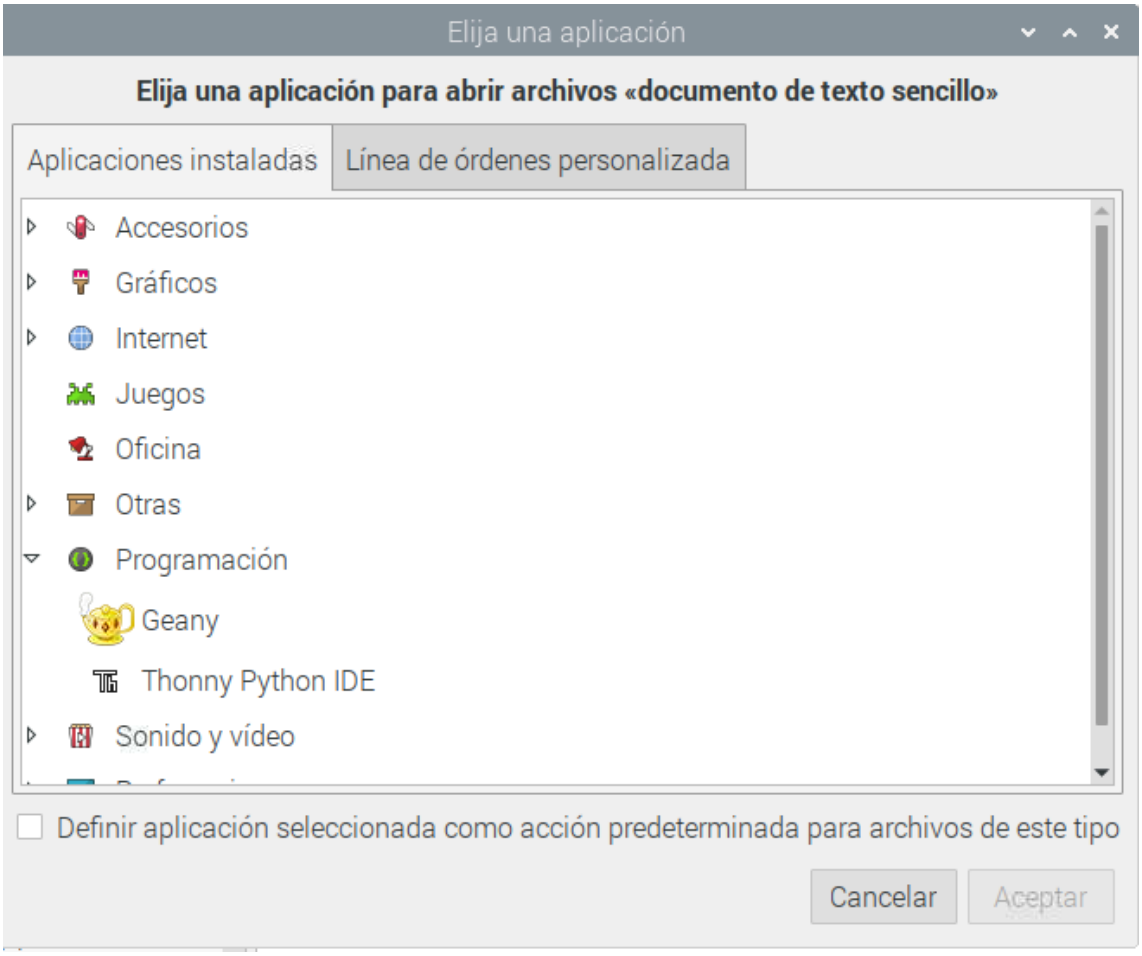


Figura 40. Ventana desplegada al utilizar *Abrir con*.

De las diferentes opciones, desplegamos el apartado de *Programación* y usaremos *Thonny Python IDE*. Ahora ya podemos empezar a escribir el código del programa. Por último vamos a añadir una fuente de letra en este caso *Roboto*. Para añadir una fuente de letra, debemos crear una carpeta ".fonts" en el directorio */home/pi*. Cuando la creemos desaparecerá, pero no es así, solo estará oculta. Tenemos que dar click derecho y posteriormente *Mostrar Ocultos*. A continuación, trasladaremos los archivos descargados de nuestra fuente con la extensión *ttf* a esa carpeta. En nuestro caso queremos usar la fuente Roboto-Thin, sin embargo en Python debemos poner Robotothin para que la reconozca.

Por último debemos configurar la Raspberry para que al iniciarse ejecute automáticamente nuestro script de *Python*. Para ello vamos al terminal y escribimos *contrab -e* editaremos el archivo y añadiremos esta línea.

@reboot /usr/bin/python /home/pi/SMirror/SMirror.py

Ya tendremos configurada la Raspberry para que ejecute automáticamente nuestro código.

9. ANEXO 2:

Código de la placa de desarrollo NodeMCU.

```
/*
 Version 0.1 - Feb 10 2018
*/

#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <ESP8266WiFiMulti.h>
#include <WebSocketsClient.h>
//  https://github.com/kakopappa/sinric/wiki/How-to-add-dependency-
//libraries
#include <ArduinoJson.h> //
//https://github.com/kakopappa/sinric/wiki/How-to-add-dependency-
//libraries

ESP8266WiFiMulti WiFiMulti;
WebSocketsClient webSocket;
WiFiClient client;

#define MyApiKey "cdda5912-96b7-4eb2-afa3-c97b13fda2a3" // TODO:
//Change to your sinric API Key. Your API Key is displayed on
//sinric.com dashboard
#define MySSID "TP-LINK_8878" // TODO: Change to your Wifi network
//SSID
#define MyWifiPassword "doctorferran70" // TODO: Change to your Wifi
//network password

#define API_ENDPOINT "http://sinric.com"
#define HEARTBEAT_INTERVAL 300000 // 5 Minutes

uint64_t heartbeatTimestamp = 0;
bool isConnected = false;

void turnOn(String deviceId) {
  if (deviceId == "5f103ec8ad7a48327f36849a") // Device ID of first
//device
  {
    Serial.print("Turn on device id: ");
    Serial.println(deviceId);
    analogWrite(12, 1023); //rojo
    analogWrite(14, 1023); //verde
    analogWrite(13, 1023); //azul

  }
  else if (deviceId == "5axxxxxxxxxxxxxxxxxx") // Device ID of second
//device
  {
    Serial.print("Turn on device id: ");
    Serial.println(deviceId);
  }
  else {
    Serial.print("Turn on for unknown device id: ");
    Serial.println(deviceId);
  }
}

void turnOff(String deviceId) {
   if (deviceId == "5f103ec8ad7a48327f36849a") // Device ID of first
//device
   {
     Serial.print("Turn off Device ID: ");
     Serial.println(deviceId);
```

```
      analogWrite(14, 0);
      analogWrite(13, 0);
      analogWrite(12, 0);
    }
    else if (deviceId == "5axxxxxxxxxxxxxxxxxx") // Device ID of
//second device
    {
      Serial.print("Turn off Device ID: ");
      Serial.println(deviceId);
  }
  else {
      Serial.print("Turn off for unknown device id: ");
      Serial.println(deviceId);
  }
}

void webSocketEvent(WStype_t type, uint8_t * payload, size_t length) {
  switch(type) {
    case WStype_DISCONNECTED:
      isConnected = false;
      Serial.printf("[WSc] Webservice disconnected from
sinric.com!\n");
      break;
    case WStype_CONNECTED: {
      isConnected = true;
      Serial.printf("[WSc] Service connected to sinric.com at url:
%s\n", payload);
      Serial.printf("Waiting for commands from sinric.com
...\n");
      }
      break;
    case WStype_TEXT: {
        Serial.printf("[WSc] get text: %s\n", payload);
        // Example payloads

        // For Light device type
        //// {"deviceId": xxxx, "action": "setPowerState", value:
//"ON"} // https://developer.amazon.com/docs/device-apis/alexa-
//powercontroller.html
        //{"deviceId": xxxx, "action": "AdjustBrightness", value: 3}
// https://developer.amazon.com/docs/device-apis/alexa-
//brightnesscontroller.html
        //{"deviceId": xxxx, "action": "setBrightness", value: 42} //
//https://developer.amazon.com/docs/device-apis/alexa-
//brightnesscontroller.html
        //{"deviceId": xxxx, "action": "SetColor", value: {"hue":
//350.5,  "saturation": 0.7138, "brightness": 0.6501}} //
//https://developer.amazon.com/docs/device-apis/alexa-
//colorcontroller.html
        //{"deviceId": xxxx, "action": "DecreaseColorTemperature"} //
//https://developer.amazon.com/docs/device-apis/alexa-
//colortemperaturecontroller.html
        //{"deviceId": xxxx, "action": "IncreaseColorTemperature"} //
//https://developer.amazon.com/docs/device-apis/alexa-
//colortemperaturecontroller.html
        // {"deviceId": xxxx, "action": "SetColorTemperature", value:
//2200}
// https://developer.amazon.com/docs/device-apis/alexa-
//colortemperaturecontroller.html

#if ARDUINOJSON_VERSION_MAJOR == 5
```

```
        DynamicJsonBuffer jsonBuffer;
        JsonObject& json = jsonBuffer.parseObject((char*)payload);
#endif
#if ARDUINOJSON_VERSION_MAJOR == 6
        DynamicJsonDocument json(1024);
        deserializeJson(json, (char*) payload);
#endif
        String deviceId = json ["deviceId"];
        String action = json ["action"];

        if(action == "setPowerState") { // Switch or Light
            String value = json ["value"];
            if(value == "ON") {
                turnOn(deviceId);
            } else {
                turnOff(deviceId);
            }
        }
        else if(action == "SetColor") {
            // Alexa, set the device name to red
            //get text:
{"deviceId":"5f103ec8ad7a48327f36849a","action":"SetColor","value":{"h
ue":0,"saturation":1,"brightness":1}};
            String hue = json ["value"]["hue"];
            String saturation = json ["value"]["saturation"];
            String brightness = json ["value"]["brightness"];
             if (0 == hue.toInt()){
            analogWrite(12, 1023);
            analogWrite(14, 0);
            analogWrite(13, 0);

            }
            if (39 == hue.toInt()){
            analogWrite(12, 1023);
            analogWrite(14, 100);
            analogWrite(13, 0);

            }
            else if (60 == hue.toInt()){
            analogWrite(12, 1023);
            analogWrite(14, 900);
            analogWrite(13, 0);
            }else if (120 == hue.toInt() ){
            analogWrite(12, 0);
            analogWrite(14, 1023);
            analogWrite(13, 0);

            }else if (180 == hue.toInt()){
            analogWrite(12, 0);
            analogWrite(14, 1023);
            analogWrite(13, 1023);

            }else if (240 == hue.toInt() ){
            analogWrite(12, 0);
            analogWrite(14, 0);
            analogWrite(13, 1023);

            }
            if (255 == hue.toInt()){
            analogWrite(12, 1023);
            analogWrite(14, 100);
```

```
            analogWrite(13, 800);

            }else if (300 == hue.toInt() ){
            analogWrite(12, 1023);
            analogWrite(14, 900);
            analogWrite(13, 900);

            }
            else if (348 == hue.toInt() ){
            analogWrite(12, 1023);
            analogWrite(14, 100);
            analogWrite(13, 500);

            }



            Serial.println("[WSc] hue: " + hue);
            Serial.println("[WSc] saturation: " + saturation);
            Serial.println("[WSc] brightness: " + brightness);
        }
        else if(action == "SetBrightness") {

        }
        else if(action == "SetColorTemperature") {
          String temp = json ["value"];
          if (temp.toInt()==4000){
           analogWrite(12, 1023);
           analogWrite(14, 1023);
           analogWrite(13, 1023);
          }


        }
        else if(action == "AdjustBrightness") {

        }
        else if (action == "test") {
            Serial.println("[WSc] received test command from
sinric.com");
        }
      }
      break;
    case WStype_BIN:
      Serial.printf("[WSc] get binary length: %u\n", length);
      break;
      default: break;
  }
}

void setup() {
  Serial.begin(115200);

  WiFiMulti.addAP(MySSID, MyWifiPassword);
  Serial.println();
  Serial.print("Connecting to Wifi: ");
  Serial.println(MySSID);

  // Waiting for Wifi connect
  while(WiFiMulti.run() != WL_CONNECTED) {
    delay(500);
```

```
    Serial.print(".");
  }
  if(WiFiMulti.run() == WL_CONNECTED) {
    Serial.println("");
    Serial.print("WiFi connected. ");
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());
  }

  // server address, port and URL
  webSocket.begin("iot.sinric.com", 80, "/");

  // event handler
  webSocket.onEvent(webSocketEvent);
  webSocket.setAuthorization("apikey", MyApiKey);

  // try again every 5000ms if connection has failed
  webSocket.setReconnectInterval(5000);   // If you see 'class
//WebSocketsClient' has no member named 'setReconnectInterval' error
//update arduinoWebSockets
}

void loop() {
  webSocket.loop();

  if(isConnected) {
      uint64_t now = millis();

      // Send heartbeat in order to avoid disconnections during ISP
//resetting IPs over night. Thanks @MacSass
      if((now - heartbeatTimestamp) > HEARTBEAT_INTERVAL) {
          heartbeatTimestamp = now;
          webSocket.sendTXT("H");
      }
  }
}
```

10. ANEXO 3:

Código del microprocesador Raspberry Pi 4.

```
from Tkinter import *
import holidays
import locale
import time
import requests
import json
import traceback
import feedparser
import pyowm
import spotipy
import spotipy.util as util
import spotipy.oauth2 as oauth
from spotipy.oauth2 import SpotifyClientCredentials
from urllib2 import urlopen
import urllib2
from PIL import Image, ImageTk
from contextlib import contextmanager


ui_locale = '' #dejamos el local sin definir
holidays = holidays.Spain()
formato_reloj = 24 # 12 or 24
formato_fecha = "%b %d, %Y" # parametros disponibles en
holidays_format = "%d/%m/%Y"
lastday_format = "31/12/%Y"  #Establecemos el ultimo dia del año actual
weather_api = '537532f633284de15876f670de68b6ed' #Se obtiene al registrarse en
oopenweathermap
loc_api='b359ac4279d54da769b1a1ade9a45ca8' #Se obtiene al registrarse en
http://api.ipstack.com/
latitud = None # Establecer la latitud si la ip nos da una localizacion incorrecta
longitud = None # Establecer la longitud si la ip nos da una localizacion incorrecta
texto_grande = 48
texto_mediano = 28
texto_pequeño = 18
usuario ='1112719242'

#Creamos la funcion que administrara la informacion del local
@contextmanager
def setlocale(name):
        saved = locale.setlocale(locale.LC_ALL)
        try:
            yield locale.setlocale(locale.LC_ALL, name)
        finally:
            locale.setlocale(locale.LC_ALL, saved)

#declaramos las ubicaciones de los diferentes iconos
iconos = {
    '01d': "/home/pi/SMirror/imagenes/01d.png",  #  dia despejado
    '01n': "/home/pi/SMirror/imagenes/01n.png",  #  noche despejada
    '02d': "/home/pi/SMirror/imagenes/02d.png",  #  dia con pocas nubes
    '02n': "/home/pi/SMirror/imagenes/02n.png",  #  noche con pocas nubes
    '03d': "/home/pi/SMirror/imagenes/03d.png",  #  dia con nubes dispersas
    '03n': "/home/pi/SMirror/imagenes/03n.png",  #  noche con nubes dispersas
    '04d': "/home/pi/SMirror/imagenes/04d.png",  #  dia nublado
    '04n': "/home/pi/SMirror/imagenes/04n.png",  #  noche nublada
    '09d': "/home/pi/SMirror/imagenes/09d.png",  #  dia de lluvia ligera
    '09n': "/home/pi/SMirror/imagenes/09n.png",  #  noche de lluvia ligera
    '10d': "/home/pi/SMirror/imagenes/10d.png",  #  dia lluvioso
    '10n': "/home/pi/SMirror/imagenes/10n.png",  #  noche lluviosa
    '11d': "/home/pi/SMirror/imagenes/11d.png",  #  dia de tormenta
    '11n': "/home/pi/SMirror/imagenes/11n.png",  #  noche de tormenta
    '13d': "/home/pi/SMirror/imagenes/13d.png",  #  dia de nieve
    '13n': "/home/pi/SMirror/imagenes/13n.png",  #  noche de nieve
    '50d': "/home/pi/SMirror/imagenes/50d.png",  #  dia con niebla
    '50n': "/home/pi/SMirror/imagenes/50n.png",  #  noche con niebla
}

#creamos la clase del reloj
class Clock(Frame):
    def __init__(self, parent, *args, **kwargs):
        Frame.__init__(self, parent, bg='black') #Creamos el frame que agrupara los
        diferentes elemntos de esta clase
```

```
        #Inicializamos los parametros utilizados
        self.time1 = ''
        self.day_of_week1 = ''
        self.date1 = ''
        self.holidays1 = ''
        self.nextholidays1 = ''
        self.nextholidays2 = ''
        #Creamos los diferentes elementos
        self.timeLbl = Label(self, font=('Robotothin', texto_grande), fg="white",
        bg="black")
        self.timeLbl.pack(side=TOP, anchor=W)
        self.dayOWLbl = Label(self, text=self.day_of_week1, font=('Robotothin',
        texto_pequeño), fg="white", bg="black")
        self.dayOWLbl.pack(side=TOP, anchor=W)
        self.dateLbl = Label(self, text=self.date1, font=('Robotothin',
        texto_pequeño), fg="white", bg="black")
        self.dateLbl.pack(side=TOP, anchor=W)
        self.holidaysLbl = Label(self, text=self.holidays1, font=('Robotothin',
        texto_pequeño), fg="white", bg="black")
        self.holidaysLbl.pack(side=TOP, anchor=W, pady=20)
        self.nextholidays1Lbl = Label(self, text=self.nextholidays1,
        font=('Robotothin', texto_pequeño), fg="white", bg="black")
        self.nextholidays1Lbl.pack(side=TOP, anchor=W)
        self.nextholidays2Lbl = Label(self, text=self.nextholidays2,
        font=('Robotothin', texto_pequeño), fg="white", bg="black")
        self.nextholidays2Lbl.pack(side=TOP, anchor=W)
        self.tick() #Llamamos a una funcion que nos devolvera los datos de interes

    def tick(self):
        try:
            with setlocale(ui_locale):
                if formato_reloj == 12:
                    time2 = time.strftime('%I:%M %p') # Declaramos la estructura del
                    reloj para el formato 12H
                else:
                    time2 = time.strftime('%H:%M') # Declaramos la estructura del
                    reloj para el formato 24H
                # Indicamos la estructura de los otros elementos
                day_of_week2 = time.strftime('%A')
                date2 = time.strftime(formato_fecha)
                holidays2 = time.strftime(holidays_format)
                lastday = time.strftime(lastday_format)
                # Comprobamos si han cambiado los parametros y los actualizamos
                if time2 != self.time1:
                    self.time1 = time2
                    self.timeLbl.config(text=time2)
                if day_of_week2 != self.day_of_week1:
                    self.day_of_week1 = day_of_week2
                    self.dayOWLbl.config(text=day_of_week2)
                if date2 != self.date1:
                    self.date1 = date2
                    self.dateLbl.config(text=date2)
                if holidays2 != self.holidays1:
                    self.holidays1 = holidays2
                    holidays2 in holidays  #Comprobamos si la fecha actual es un dia
                    festivo
                    if holidays2 is True:
                        self.holidaysLbl.config(text=holidays.get(holidays2))   # Si
                        es un dia festivo o expresamos en el label
                        self.holidaysLbl.config(text="Siguientes dias festivos")#
                        Declaramos los siguientes dias festivos
                        festivos=holidays[holidays2: lastday]
                        self.nextholidays1Lbl.config(text=str(festivos[0])+"
                        "+str(holidays.get(festivos[0])))
                        self.nextholidays2Lbl.config(text=str(festivos[1])+"
                        "+str(holidays.get(festivos[1])))
                    else:               #Si la fecha actual no es un dia festivo,
                    solo declaramos los siguientes dias festivos
                        self.holidaysLbl.config(text="Siguientes dias festivos")
                        festivos=holidays[holidays2: lastday]
                        self.nextholidays1Lbl.config(text=str(festivos[0])+"
                        "+str(holidays.get(festivos[0])))
```

```
                        self.nextholidays2Lbl.config(text=str(festivos[1])+"
                        "+str(holidays.get(festivos[1])))

         except Exception as e: #Si ocurre algun error lo expresamos por el terminal
            traceback.print_exc()
            return "Error: %s. No se pudo obtener informacion del reloj." % e
        self.timeLbl.after(200, self.tick) #Llamamos a la función trans 200 ms

#Declaramos la clase SPOTIFY
class Spotify(Frame):
    def __init__(self, parent, *args, **kwargs): #Creamos el frame y los diferentes
    elementos
        Frame.__init__(self, parent, *args, **kwargs)
        #Inicializamos los parámetros utilizados
        self.name=''
        self.image_url=''
        self.config(bg='black')
        self.spotitle = 'SPOTIFY'
        #Creamos los diferentes elementos
        self.spotifyLbl = Label(self, text=self.spotitle, font=('Roboto',
        texto_mediano), fg="white", bg="black")
        self.spotifyLbl.pack(side=TOP, anchor=W)
        #usaremos wraplength para recortar el tamaño máximo del texto
        self.userLbl = Label(self, anchor='w', justify=LEFT, font=('Robotothin',
        12), fg="white", bg="black", wraplength=70)
        self.userLbl.pack(side=LEFT, anchor=N)
        self.spotifyiconLbl = Label(self, bg="black")
        self.spotifyiconLbl.pack(side=LEFT, anchor=N)
        self.get_spoti()#Llamamos a una funcion que nos devolvera los datos de interes

    def get_spoti(self):
        try: #Vamos a obtener el token de seguridad para la funcion de interes
            token =util.prompt_for_user_token(usuario,
                        'user-read-currently-playing',
                        client_id='dc97f55ca70e44adb390674543c7c777',
                        client_secret='a118f7b305d8491e96c3cca8c6863132',
                        redirect_uri='http://localhost/8888/callback/')

            spotifyObject = spotipy.Spotify(auth=token)     #Obtenemos la
            información de Spotify
            spotifydata = spotifyObject.currently_playing() #Seleccionamos los datos
            de interes


            if spotifydata is not None: #Comprobamos si se está reproduciendo alguna
            canción en el momento
                reproduciendo=spotifydata['is_playing']
                if reproduciendo is False: #Si no se esta reproduciendo
                    name2="Pausado"  #Indicaremos que esta pausado
                    #Representaremos el icono de Spotify

                    image_url2="https://i.pinimg.com/236x/f0/5c/bc/f05cbc8c0f8b075d2b4
                    f1f68fee49649.jpg"
                    if self.name != name2:
                        self.name = name2
                        self.userLbl.config(text=name2)
                        spotifyimage=Image.open(urllib2.urlopen(image_url2))
                        spotifyimage = spotifyimage.resize((100, 70), Image.ANTIALIAS)
                        spotifyimage = spotifyimage.convert('RGB')
                        im = ImageTk.PhotoImage(spotifyimage)
                        self.spotifyiconLbl.config(image=im)
                        self.spotifyiconLbl.image=im
                elif reproduciendo is True: #Si se esta reproduciendo
                    name2=spotifydata['item']['name'] #Obtendremos el nombre de la
                    canción
                    # Y la portada del mismo
                    image_url2 = spotifydata['item']['album']['images'][0]['url']
                    if self.name != name2:
                        self.name = name2
                        self.userLbl.config(text=name2)
                        self.image_url = image_url2
                        spotifyimage=Image.open(urllib2.urlopen(image_url2))
```

```
189                         spotifyimage = spotifyimage.resize((100, 100),
                            Image.ANTIALIAS)
190                         spotifyimage = spotifyimage.convert('RGB')
191                         im = ImageTk.PhotoImage(spotifyimage)
192                         self.spotifyiconLbl.config(image=im)
193                         self.spotifyiconLbl.image=im
194                 else: #Si no tenemos informacion de Spotify
195                     name2="Spotify no iniciado" #Indicaremos que no esta iniciado
196                     # Y expresaremos el logo de Spotify
197
                        image_url2="https://i.pinimg.com/236x/f0/5c/bc/f05cbc8c0f8b075d2b4f1f6
                        8fee49649.jpg"
198                     if self.name != name2:
199                             self.name = name2
200                             self.userLbl.config(text=name2)
201                             spotifyimage=Image.open(urllib2.urlopen(image_url2))
202                             spotifyimage = spotifyimage.resize((100, 70), Image.ANTIALIAS)
203                             spotifyimage = spotifyimage.convert('RGB')
204                             im = ImageTk.PhotoImage(spotifyimage)
205                             self.spotifyiconLbl.config(image=im)
206                             self.spotifyiconLbl.image=im
207                     #self.after(1000, self.get_spoti)
208             except Exception as e: #Si ocurre algun error lo expresamos por el terminal
209                 traceback.print_exc()
210                 return "Error: %s. No se pudo obtener informacion de Spotify." % e
211             self.after(1000, self.get_spoti) #Repetir la funcion tras 1 segundo
212
213
214
215 class Weather(Frame):
216     def __init__(self, parent, *args, **kwargs): #Creamos el frame
217         Frame.__init__(self, parent, bg='black')
218         #Inicializamos las variables utilizadas
219         self.temperature = ''
220         self.forecast = ''
221         self.location = ''
222         self.currently = ''
223         self.icon = ''
224         #Creamos los diferentes elementos
225         self.degreeFrm = Frame(self, bg="black")
226         self.degreeFrm.pack(side=TOP, anchor=E)
227         self.temperatureLbl = Label(self.degreeFrm, font=('Robotothin',
            texto_grande), fg="white", bg="black")
228         self.temperatureLbl.pack(side=LEFT, anchor=N)
229         self.iconLbl = Label(self.degreeFrm, bg="black")
230         self.iconLbl.pack(side=LEFT, anchor=W, padx=20)
231         self.currentlyLbl = Label(self, font=('Robotothin', texto_mediano),
            fg="white", bg="black")
232         self.currentlyLbl.pack(side=TOP, anchor=W)
233         self.forecastLbl = Label(self, font=('Robotothin', texto_pequeño),
            fg="white", bg="black")
234         self.forecastLbl.pack(side=TOP, anchor=W)
235         self.locationLbl = Label(self, font=('Robotothin', texto_pequeño),
            fg="white", bg="black")
236         self.locationLbl.pack(side=TOP, anchor=W)
237         self.get_weather() #Llamamos a una funcion que nos devolvera los datos de
            interes
238
239     def get_ip(self):#Obtenemos la ip del dispositivo
240         try:
241             ip_url = "http://jsonip.com/"
242             req = requests.get(ip_url)
243             ip_json = json.loads(req.text)
244             return ip_json['ip']
245         except Exception as e:
246             traceback.print_exc()
247             return "Error: %s. No se pudo obtener la IP." % e
248
249     def get_weather(self): #Con la IP obtenemos la latitud y la longitud de nuestra
        ubicacion
250         try:
251
```

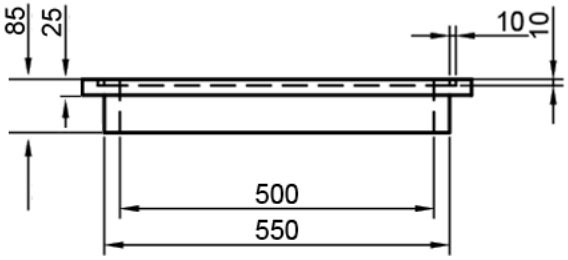
```
252             if latitud is None and longitud is None: #Si no hemos declarado una
                latitud y una longitud
253                 location_req_url =
                    "http://api.ipstack.com/%s?access_key=%s&format=1" %
                    (self.get_ip(),loc_api)
254                 r = requests.get(location_req_url)
255                 location_obj = json.loads(r.text)
256
257                 lat = location_obj['latitude']
258                 lon = location_obj['longitude']
259
260                 location2 = "%s, %s" % (location_obj['city'],
                    location_obj['region_code'])
261
262                 # obtenemos el tiempo con la latitud y longitud
263                 weather_req_url =
                    "http://api.openweathermap.org/data/2.5/forecast?lat=%s&lon=%s&appid=%
                    s&lang=es" % (lat,lon,weather_api)
264
265             else: #Obtenemos el tiempo para la latitud y longitud si han sido
                declaradas
266                 location2 = ""
267                 # get weather
268                 weather_req_url =
                    "http://api.openweathermap.org/data/2.5/forecast?lat=%s&lon=%s&appid=%
                    s&lang=es" % (latitud,longitud,weather_api)
269             #Escogemos los datos de interes con la ayuda de JSON
270             r = requests.get(weather_req_url)
271             weather_obj = json.loads(r.text)
272
273             degree_sign= u'\N{DEGREE SIGN}'
274             temperature2 = "%s%s" %
                (str(int(weather_obj['list'][0]['main']['temp'])-273), degree_sign)
275
276             forecast2 = weather_obj['list'][1]['weather'][0]['description']
277             #Actualizamos los datos de interes
278             icono = weather_obj['list'][0]['weather'][0]['icon']
279             icono2 = None
280
281             if icono in iconos:
282                 icono2 = iconos[icono]
283
284             if icono2 is not None:
285                 if self.icon != icono2:
286                     self.icon = icono2
287                     image = Image.open(icono2)
288                     image = image.resize((50, 50),Image.ANTIALIAS)
289                     photo = ImageTk.PhotoImage(image)
290
291                     self.iconLbl.config(image=photo)
292                     self.iconLbl.image = photo
293             else:
294                 self.iconLbl.config(image='')
295             if self.forecast != forecast2:
296                 self.forecast = forecast2
297                 self.forecastLbl.config(text=forecast2)
298             if self.temperature != temperature2:
299                 self.temperature = temperature2
300                 self.temperatureLbl.config(text=temperature2)
301             if self.location != location2:
302                 if location2 == ", ":
303                     self.location = "Cannot Pinpoint Location"
304                     self.locationLbl.config(text="Cannot Pinpoint Location")
305                 else:
306                     self.location = location2
307                     self.locationLbl.config(text=location2)
308         except Exception as e:
309             traceback.print_exc()
310             print ("Error: %s. No se pudo obtener el tiempo." % e)
311
312         self.after(600000, self.get_weather)
313
```

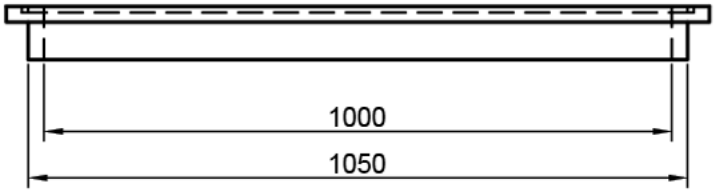
```
314
315 #Declaramos la clse de las noticias
316 class News(Frame):
317     def __init__(self, parent, *args, **kwargs): #Creamos el frame
318         Frame.__init__(self, parent, *args, **kwargs)
319         #Inicializamos los parametros utilizados
320         self.config(bg='black')
321         self.title = 'Noticias'
322         #Creamos los diferentes modulos
323         self.newsLbl = Label(self, text=self.title, font=('Roboto', texto_mediano),
            fg="white", bg="black")
324         self.newsLbl.pack(side=TOP, anchor=W)
325         self.headlinesContainer = Frame(self, bg="black")
326         self.headlinesContainer.pack(side=TOP)
327         self.get_headlines() #Llamamos a una funcion que nos devolvera los datos de
            interes
328
329     def get_headlines(self):
330         try:
331             # borramos las entradas anteriores
332             for widget in self.headlinesContainer.winfo_children():
333                 widget.destroy()
334             headlines_url = "https://www.abc.es/rss/feeds/abc_ultima.xml" #Obtenemos
                las nuevas entradas de ABC
335             feed = feedparser.parse(headlines_url)  #Parseamos la informacion
336             for post in feed.entries[0:5]:          #Expresamos los 5 primeros
                titulares en otro frame
337                 headline = NewsHeadline(self.headlinesContainer, post.title)
338                 headline.pack(side=TOP, anchor=W)
339         except Exception as e:
340             traceback.print_exc()
341             print ("Error: %s. No se pudo obtener las noticias." % e)
342
343         self.after(600000, self.get_headlines) #Repetimos tras 10 minutos
344
345
346 class NewsHeadline(Frame):  #Creamos la clase que expresará las 5 noticias
347     def __init__(self, parent, event_name=""):
348         Frame.__init__(self, parent, bg='black')
349         image = Image.open("/home/pi/SMirror/imagenes/Noticias.png")
350         image = image.resize((25, 25), Image.ANTIALIAS)
351         image = image.convert('RGB')
352         photo = ImageTk.PhotoImage(image)
353         self.iconLbl = Label(self, bg='black', image=photo)
354         self.iconLbl.image = photo
355         self.iconLbl.pack(side=LEFT, anchor=N)
356         self.eventName = event_name
357         self.eventNameLbl = Label(self, anchor='w', justify=LEFT, text=self.eventName,
358                                   font=('Robotothin', 12), fg="white", bg="black",
                                      wraplength=375)
359         self.eventNameLbl.pack(side=LEFT, anchor=N)
360
361
362 #Creamos una clase principal que agrupara a todas las demas
363 class FullscreenWindow:
364
365     def __init__(self):
366         self.tk = Tk() #Iniciamos el modulo Tkinter
367         self.tk.configure(background='black')
368         #Creamos los diferentes frames donde situaremos las clases creadas
369         self.topFrame = Frame(self.tk, background = 'black')
370         self.bottomFrame = Frame(self.tk, background = 'black')
371         self.topFrame.pack(side = TOP, fill=BOTH, expand = YES)
372         self.bottomFrame.pack(side = BOTTOM, fill=BOTH, expand = YES)
373         #Iniciamos la interfaz grafica con la pantalla completa
374         self.state = True
375         self.tk.attributes("-fullscreen", self.state)
376         #Declaramos la tecla enter para activar o desactivar la pantalla completa
377         self.tk.bind("<Return>", self.alternar_fullscreen)
378
379         # Incluimos el modulo del reloj
380         self.clock = Clock(self.topFrame)
```

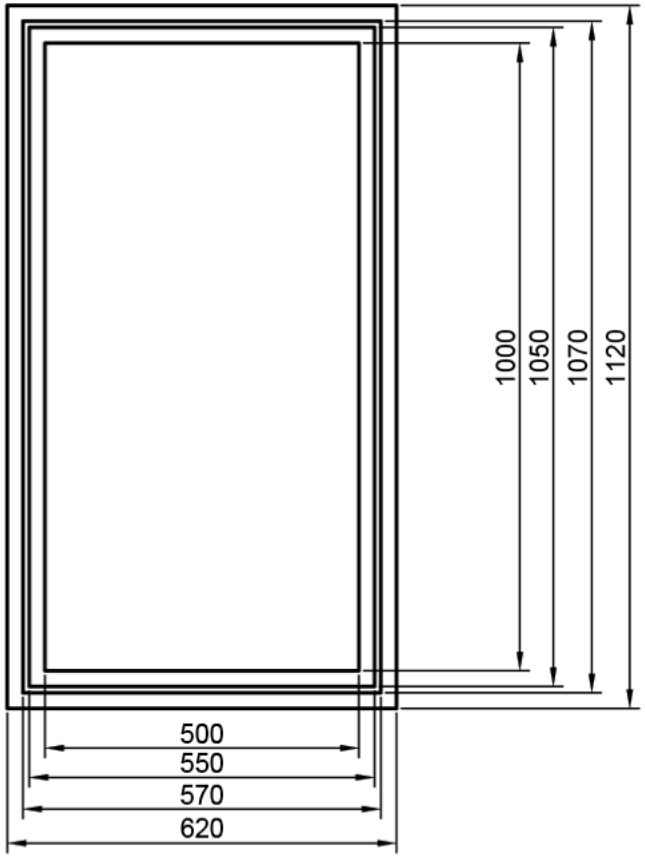
```
381             self.clock.pack(side=LEFT, anchor=N, padx=100, pady=60)
382             # Incluimos el modulo del tiempo
383             self.weather = Weather(self.topFrame)
384             self.weather.pack(side=RIGHT, anchor=N, padx=50, pady=60)
385             # Incluimos el modulo de Spotify
386             self.spotify = Spotify(self.bottomFrame)
387             self.spotify.pack(side=RIGHT, anchor=S, padx=100, pady=60)
388             # Incluimos el modulo de las noticias
389             self.news = News(self.bottomFrame)
390             self.news.pack(side=LEFT, anchor=S, padx=100, pady=60)
391
392         def alternar_fullscreen(self, event=None):
393             self.state = not self.state  # Alternamos el elemento booleano
394             self.tk.attributes("-fullscreen", self.state)
395             return "break"
396
397
398
399     if __name__ == '__main__':
400         w = FullscreenWindow()#Iniciamos la clase principal
401         w.tk.mainloop()
402
403
```

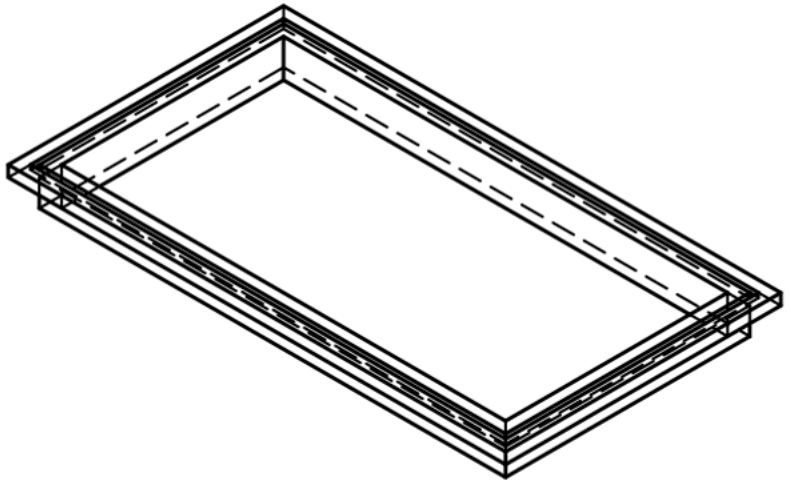
PLANOS

Índice de contenido

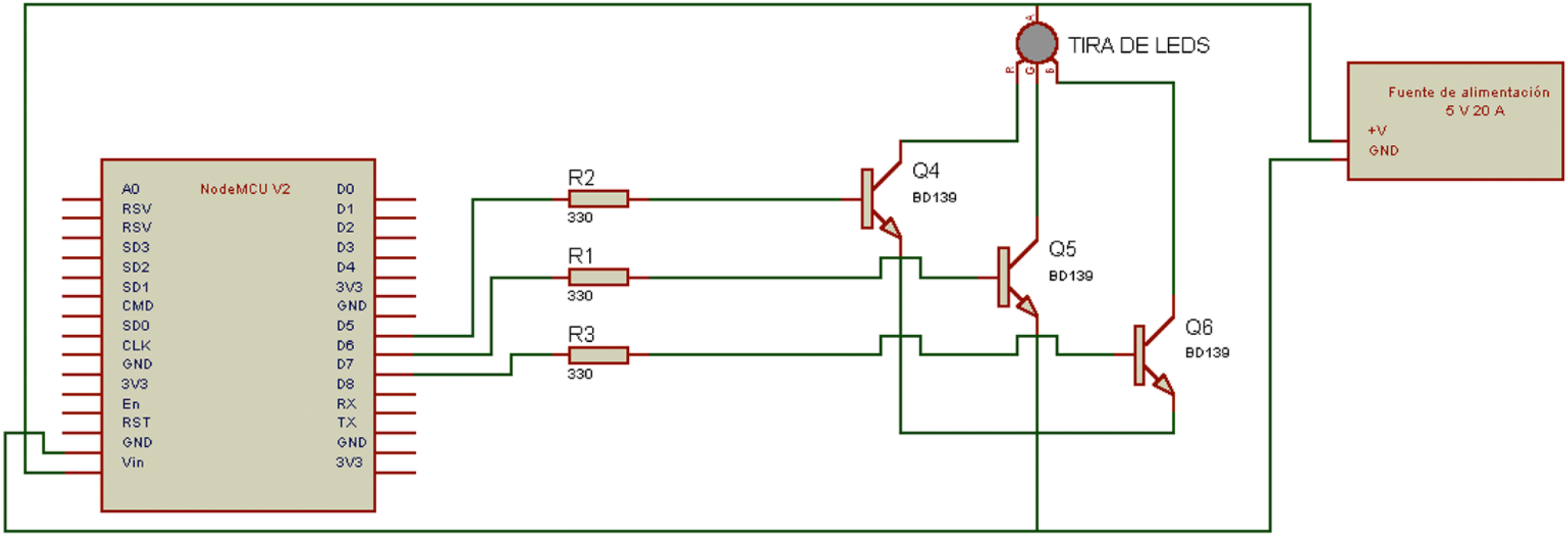








REFERENCIA	NOMBRE PLANO	FECHA
P-001	Marco SMirror	13-06-20
PROMOTOR:	Trabajo Final de Grado	ESCALA
AUTOR:	Gao Yao, Jia Cheng	1:12
TÍTULO DEL PROYECTO:	DISEÑO Y DESARROLLO DE UN PROTOTIPO DE ESPEJO INTELIGENTE (SMIRROR)	Nº PLANO
		001



REFERENCIA	NOMBRE PLANO	FECHA
P-002	Control tira LEDs	15-06-20
PROMOTOR: Trabajo Final de Grado		ESCALA
AUTOR: Gao Yao, Jia Cheng		
TÍTULO DEL PROYECTO: DISEÑO Y DESARROLLO DE UN PROTOTIPO DE ESPEJO INTELIGENTE (SMIRROR)		Nº PLANO 002

PRESUPUESTO

Índice de contenido

1. Cuadros de precios

En las tablas 1, 2 y 3 se presentan los costes de diseño y desarrollo del prototipo de espejo inteligente SMirror, divididos en tres partidas: materiales, mano de obra y equipos. Por último se calculará la suma total de estos cuadros y se les añadirán los gastos generales, el beneficio industrial y el IVA (Impuesto sobre el Valor Añadido).

Tabla 4. Costes de los materiales

Concepto	Importe		
	Precio (euros)	Cantidad (ud)	Total (euros)
Raspberry pi 4 B 4GB RAM	59.96	1	59.96
Adaptador microSD-USB	5.99	1	5.99
Tarjeta microSD de 32 GB	6.98	1	6.98
Cable micro HDMI – HDMI	7.59	1	7.59
Pantalla Hp 22f de 22 pulgadas con fuente de alimentación	101.15	1	101.15
Fuente de alimentación 5.1V 3A	8.99	1	8.99
NodeMCU V2	7.49	1	7.49
Cables para protoboard	6.99	1	6.99
Hilo de estaño para soldar	9.99	1	9.99
Fuente de alimentación 5V 20 A	19.99	1	19.99
Transistores BJT BD139	0.65	3	1.95
Resistencias 330 Ω	0.04	3	0.12
Protoboard de 4.6 x 3.6 cm	1.45	1	1.45
Tira de LEDs RGB	15.29	1	15.29
Vinilo unidireccional 40 x 200 cm	11.88	1	11.88
Plancha de metacrilato	20.00	1	20
Marco de DM	60.00	1	60
tornillos	0.02	4	0.08
embellecedores de tornillos	0.86	4	3.44
Regleta de 4 conectores	7.96	1	7.96
2 láminas de Goma EVA negras de 60 x 40 cm	0.60	2	1.20
Cinta aislante negra 20 m x 19 mm	1.30	1	1.30
Cinta de doble cara de espuma 5 m x 2 cm	2.99	1	2.99
Total			362.78

Tabla 5. Coste de la mano de obra

Concepto	Importe		
	Precio (euros/hora)	Cantidad (horas)	Total (euros)
Ingeniero Técnico	15	220	3300
Total			3300

Tabla 6. Coste del equipo

Concepto	Importe
	Total (euros)
Soldador de estaño	11.99
Multímetro digital	15.40
Total	953.27

Tabla 7. Amortización del equipo

Concepto	Importe		
	Tiempo (meses)	Precio (euros/mes)	Total (euros)
Ordenador	6	30	180

2. Resumen del presupuesto

La Tabla 4 resume las diferentes partidas. Por otro lado se le añade el 13% de gastos generales, el 6% de beneficio industrial y el 21% de IVA (Impuesto sobre el Valor Añadido), proporcionando un total de 6462.47 €.

Tabla 8. Resumen del presupuesto

Concepto	Total (euros)
Coste de los materiales	362.78
Coste de la mano de obra	3300
Coste del equipo	27.39
Amortización del equipo	180
Total de cuadros	3870.17
Gastos generales (13%)	503.12
Beneficio industrial (6%)	232.21
IVA (21%)	812.74
Total	5418.24

El presupuesto total es de cinco mil cuatrocientos diez y ocho euros (5418 €).

PLIEGO DE CONDICIONES

Índice de contenido

1. Objetivo

El presente documento muestra las propiedades mínimas que debe cumplir el diseño y desarrollo de un prototipo de espejo inteligente (SMirror).

2. Normativa

Lo primero de todo se debe cumplir las siguientes normativas.

- **UNE-EN IEC 62031:2020**

  Módulos LED para alumbrado general. Requisitos de seguridad.

- **UNE-EN 60063:2015**

  Series de valores preferentes para resistencias y condensadores. (Ratificada por AENOR en julio de 2015.)

- **UNE-EN IEC 63000:2018**

  Documentación técnica para la evaluación de los productos eléctricos y electrónicos con respecto a la restricción de sustancias peligrosas (Ratificada por la Asociación Española de Normalización en abril de 2019.)

- **UNE-EN 150003:1991 (Ratificada)**

  Especificación marco de detalle: Características nominales del encapsulado de los transistores bipolares para amplificación de baja frecuencia. (Ratificada por AENOR en noviembre de 1996.)

3. Condiciones de los materiales

Se describen los elementos principales que componen el espejo inteligente SMirror.

3.1 Raspberry Pi

Raspberry Pi modelo 4 B de 4 GB de RAM. Con una tarjeta microSD de 32 GB. Capaz de reproducir video con una resolución 4K a 60 FPS (frames por segundo). Conectividad Wifi, Bluetooth y Ethernet. Con dos conectores USB 3.0, dos conectores USB 2.0 y una salida auxiliar de audio.

3.2 Pantalla

Pantalla de 22 pulgadas Hp con resolución FHD 1920 x 1080 píxeles. Cuenta con un tiempo de respuesta de 5 ms, una entrada HDMI y una entrada VGA.

3.3 Placa de desarrollo

Placa de desarrollo AZDelivery NodeMCU V2 con microcontrolador ESP8266-12F y chip CP2102. Cuenta con una antena PCB integrada con Wifi 802.11 b/g/n. Dimensiones de 48 x 26 x 13 mm.

3.4 Tira de LEDs

Tira de LED Omeril 6m impermeable. Con 180 LEDs 5050 RGB y clase de eficiencia energética A+. Voltaje de funcionamiento 5 V e impermeabilidad IP67. Con una anchura de 1 cm.

3.5 Vinilo unidireccional

Vinilo unidireccional de TTMOW con un grosor de 0.15 mm. Refleja el 85% de la radiación térmica y el 99% de la radiación UV. Formada con acrílico SL320.

4 Requisitos mecánicos

Se muestran los diferentes requisitos mecánicos que debe cumplir el espejo inteligente SMirror.

4.2 Dimensiones del marco

Largo: 1120 mm.

Ancho: 620 mm.

Alto: 85 mm.

5 Condiciones de ejecución

5.1 Responsabilidad del ejecutor

La responsabilidad adquirida por el ejecutor del proyecto es:

- Programación del microcontrolador.
- Conexión del microcontrolador con la pantalla.
- Programación de la placa de desarrollo.
- Conexión de la placa de desarrollo con la tira de LEDs.
- Puesta en marcha del producto

5.2 Programación del microcontrolador

Se llevará a cabo la programación del microcontrolador mediante código Python.

5.3 Conexión del microcontrolador con la pantalla

Se llevará a cabo la conexión entre ambos y la verificación de una correcta comunicación.

5.4 Programación de la placa de desarrollo

Se llevará a cabo la programación de la placa de desarrollo mediante código Arduino.

5.5 Conexión de la placa de desarrollo con la tira de LEDs

Se llevará a cabo la conexión entre ambos y la verificación de una correcta comunicación.

5.6 Puesta en marcha

Se deberá llevar a cabo la comprobación del correcto funcionamiento de todas las funciones que incorpora el producto.

5.7 Garantía

Durante los dos años posteriores a su venta se ofrecerá la posibilidad solucionar pequeños problemas o sustituir pequeñas elementos defectuosos no detectados durante el proceso de fabricación o puesta en marca.

6 Pruebas y ajustes finales

6.1 Prueba de la pantalla

Se revisará la correcta conexión del microcontrolador con la pantalla además de comprobar el correcto funcionamiento de la pantalla y la comunicación entre estos.

6.2 Prueba de la tira de LEDs

Se revisará la correcta conexión de la placa de desarrollo con la tira de LEDs además de comprobar el correcto funcionamiento de la tira de LEDs y la comunicación entre estos.

6.3 Prueba de funcionamiento

Durante los diez días posteriores a la fabricación se llevará a cabo la puesta en marcha, donde se comprobará el correcto funcionamiento de todas las funciones así como el inicio automático del código creado. También se harán los cambios oportunos para mejorar el producto/código.