Universidad Zaragoza
1542

Trabajo Fin de Grado

Diseño e implementación de un sistema para el control remoto de dispositivos domóticos

Design and implementation of a remote control system for home automation devices

Autor

Javier Guajardo

Director

José Ramón Gállego Martínez

Ingeniería Electrónica y Comunicaciones / Escuela de ingeniería y arquitectura
EINA

2022

Diseño e implementación de un sistema para el control remoto de dispositivos domóticos

Resumen

En este proyecto se ha desarrollado una arquitectura basada en microcontroladores para dar servicio a las necesidades domóticas de una vivienda.

Se han definido los siguientes casos de uso:

- Luces de un acuario: Se trata de encender y apagar dos juegos de luces instalados en un acuario. Una tira LED de 12V y una lámpara led de 5V.
- Riego de jardín: Queremos poder encender y apagar el riego de un jardín compuesto por dos zonas, pudiéndose activar o desactivar cada zona de modo independiente. También se quiere poder iniciar una secuencia de riego programada.
- Estación meteorológica: Queremos poder obtener la lectura de los valores de temperatura y humedad capturados por un sensor y guardarlos en una tarjeta SD, para poder consultarlos desde una aplicación cliente.

Luces de Acuario

El primer caso de uso está implementado en un dispositivo llamado "Servidor Acuario". Su núcleo se basa en un microcontrolador ESP8266-01S que, gracias a su capacidad WIFI, es capaz de recibir peticiones y procesarlas. Una pequeña lógica de circuito hace que pueda actuar sobre los elementos electrónicos que controlan las luces de nuestro acuario, que son dos relés situados en la misma placa. Se ha elegido el ESP8266 de "Espressif Systems" en particular su versión simplificada ESP8266-01S, ya que tiene un coste muy bajo, tiene capacidad para conectarse al WIFI, consume muy poca corriente, está especializado en el desarrollo de aplicaciones IoT y su tamaño es muy reducido, facilitando así el montaje del producto final.

Riego de jardín

Para conseguir el segundo caso de uso, se ha diseñado una arquitectura hardware para controlar las electroválvulas del jardín y se ha implementado con un circuito de relés controlables por un microcontrolador. El microcontrolador utilizado será el ESP32 también de "Espressif Systems" denominado "ServidorRiego", en particular el modelo "ESP32 NodeMCU" que monta un microcontrolador ESP32-WROOM-32D fabricado por "Az-Delivery". Se ha elegido el ESP32 ya que tiene un coste muy reducido, capacidad de comunicación vía WIFI, esencial para

nuestro proyecto y muchos más pines que su anterior versión, el ESP8266. Esto último es imprescindible ya que se ha resuelto la actuación sobre las electroválvulas con 4 relés controlables por el microcontrolador (2 para cada electroválvula) que el ESP8266 no sería capaz de controlar debido a su escasez de pines. Otra de las ventajas de este microcontrolador es la capacidad *Dual Core*, esto quiere decir que dispone de un segundo *core* o núcleo de proceso capaz de realizar tareas en paralelo con el primero. Puede escuchar y gestionar llamadas entrantes en primer núcleo mientras realiza en el segundo núcleo tareas en segundo plano.

Estación meteorológica

El tercer caso de uso es una estación meteorológica capaz de proporcionar información puntual de temperatura y humedad y sus medias temporales. Para ello, se guardan los valores de humedad y temperatura de cada minuto en una tarjeta SD. Estos valores son proporcionados por un sensor DHT22 conectado al microcontrolador. A este dispositivo se le llama "ServidorClima". El microcontrolador elegido es el ESP32 como en el caso anterior debido, principalmente, a su capacidad WIFI y a la habilidad de levantar una tarea en segundo plano. Esta habilidad se ha aprovechado para tener en una tarea escuchando a posibles clientes que requieran de datos actualizados mientras que otra se encarga de validar si cambia el minuto actual, hora, mes o año para actualizar la estructura de carpetas y ficheros con los valores leídos en la SD e ir actualizando las medias.

Para el proyecto se ha elegido una arquitectura definida por tres capas:

- Capa Cliente: Hay dos canales por los que poder acceder:
  - Canal Móvil Aplicación desarrollada para móviles (Android).
  - Interfaz Web (http).
- Capa Frontal o Núcleo: Hace posible la comunicación entre la capa Cliente y la capa de *Backend* y está encargada de clasificar y validar las peticiones para después derivarlas a los distintos servidores de *backend*.
- Capa *Backend*: Interactúa con los dispositivos domóticos a petición de la capa frontal o núcleo.

Existe una copia del código desarrollado para el proyecto así como de los fritzing, draw.io y archimate que representan los esquemas de componentes y flujos en el repositorio público github:
https://github.com/jguajardobravo/domosapiens.git

Tabla de Contenidos

1 Introducción

1.1 Objetivo

El objetivo es proporcionar un medio centralizado de control de los elementos domóticos de un domicilio proporcionando a los usuarios un entorno gráfico de gobierno de los mismos basado en IoT.

La centralización de la arquitectura nos proporciona las siguientes ventajas:

- Usabilidad del producto: Simplifica las funciones del usuario en una única aplicación, permitiendo la realización de acciones complejas que pueden incluir uno o más dispositivos con instrucciones simples.
- Escalabilidad del producto: Permite añadir nuevos elementos de distintas tecnologías con relativa facilidad. En el futuro, se puede contemplar la inclusión del concepto *Plug&Play*.
- Mantenibilidad: Se consigue un entorno más homogéneo ya que todos los dispositivos responden ante un mismo protocolo centralizado, permitiendo que el mantenimiento sea significativamente más sencillo.

1.2 Contexto

Es habitual tener que contar con varias aplicaciones para conseguir gobernar la totalidad de elementos IoT para dar soporte a las necesidades domóticas de un edificio o vivienda. Dependemos de aplicaciones como *SmartLife*, *eWeLink*, etc.… para controlar diferentes dispositivos ya que cada fabricante de dispositivos elige un protocolo y se suscribe a una aplicación. Esto afecta mucho a la experiencia de usuario, que se ve obligado a utilizar distintas piezas de software para cada dispositivo cuando las funcionalidades son similares. En determinadas ocasiones incluso nos impide realizar rutinas compuestas por varios dispositivos si estos no atienden a la misma aplicación domótica, teniendo que acudir a intérpretes de software como los asistentes *Google Assistant* o *Alexa*, que son capaces de conectase a estas aplicaciones antes mencionadas y gobernarlas.
Para poder utilizar las aplicaciones ya mencionadas es necesario tener acceso a internet, ya que el servidor frontal con el que se interactúa es externo a tu domicilio. Este modelo hace al usuario dependiente de un servicio externo y de conexión a internet, lo cual no siempre es posible en determinados ambientes rurales.
Depender de un servicio externo, además de la incomodidad que supondría que dicho servicio no estuviera disponible, hace que el estado de dispositivos de nuestro domicilio sea conocido por un tercero, lo cual es información privada que podría incluso ser usada en nuestro perjuicio. Conocer que llevo 10 días sin encender la luz del salón puede interpretarse como que posiblemente estoy de vacaciones.

En este proyecto se propone un sistema centralizado que permita actuar sobre los dispositivos conectados y que sea independiente; evitando conflictos de privacidad y confidencialidad.
Los clientes, el servidor frontal y los *backend* están en la misma subred por lo que no se hace necesaria la conexión a internet para el funcionamiento del sistema. La única excepción se encuentra en la estación meteorológica, en la que se accede a la hora en tiempo real usando un servicio de internet. Este servicio se podría sin embargo sustituir por un reloj independiente. En el proyecto se ha elegido la primera opción por simplicidad. Sin embargo, la conexión a internet nos puede proporcionar, además de acceso a servicios como el de la hora, la capacidad de actuar sobre nuestra arquitectura domótica desde el exterior de nuestro domicilio.
Esto último no supone una dependencia de un tercero ni suministrar información privada a nadie, por lo que es una ventaja a tener en cuenta.

1.3 Solución propuesta

Para cumplir el objetivo planteado, se ha diseñado una arquitectura de microcontroladores basada en capas:

- Capa Cliente: Son los dispositivos físicos que acceden a nuestro sistema y que cuentan con interfaz de usuario para visualizar la respuesta.
  Se han diseñado dos tipos de clientes distintos:
  - APP Móvil: Aplicación Android con las opciones de usuario para la gestión de los dispositivos domóticos.
  - WEB: Cualquier navegador que acceda a nuestro servidor web vía HTTP.

  Esto hace que desde un *smartphone* podamos acceder mediante las dos interfaces y desde un ordenador convencional podremos acceder vía HTTP desde su navegador.
  Podemos ver definidos los componentes de la capa cliente en la Figura 1. Estos son, "PC" y "Móvil".

- Capa Frontal: La capa frontal consta de dos puntos de acceso implementados en un controlador ESP32 con capacidad WIFI (HTTP y TCP). Se ha optado por el punto de acceso TCP para conseguir controlar el protocolo de tramas enviado por completo al menos en uno de los canales. Esto nos proporciona más libertad a la hora de construir los diálogos y controlar la comunicación.
  Esta capa también hace de núcleo de la arquitectura, resolviendo la lógica de comunicación hacia los distintos servidores de *backend* (uno por cada caso de uso propuesto). Podemos ver el elemento que compone la capa Frontal en la Figura 1 bajo el nombre de "ServidorFrontal".

- Capa *Backend*: Es un conjunto de dispositivos basados en los microcontroladores ESP32 y ESP8266-01S para la actuación de los distintos elementos domóticos finales vía WIFI. Denominados "Servidor Acuario, Servidor Clima y Servidor Riego" en la Figura 1.

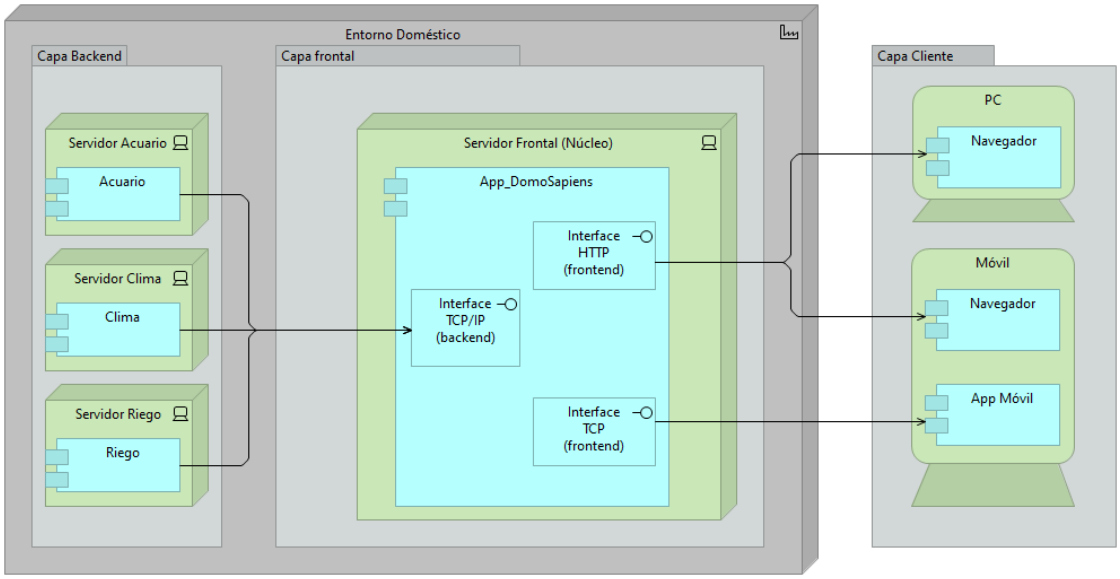

Figura 1. Esquema de la arquitectura principal de la solución desarrollada

Gracias a la programación de microcontroladores con capacidad WIFI se consigue configurar a medida las funciones de cada dispositivo y combinarlos como se estime oportuno para poder programar "rutinas" y ampliar la funcionalidad y usabilidad de los elementos eléctricos finales.

Para conseguir este objetivo, se dedica un microcontrolador con capacidad WIFI a cada dispositivo domótico.

Estos microcontroladores, se han programado para conectarse a la red local y actuar como servidores. Estos servidores son tres, uno por cada caso de uso propuesto. Para cada uno de ellos se ha diseñado la lógica circuital personalizada que hace que sean capaces de actuar sobre los dispositivos domóticos finales. Gracias a esto, se consiguen tres módulos independientes con capacidad de comunicarse vía WIFI y capaces de servir al servidor frontal bajo demanda.

El servidor frontal es el elemento de gobierno de nuestro proyecto, es un microcontrolador con la lógica para gobernar todos los casos de uso propuestos que interactúa con los clientes gracias a su capacidad WIFI. Este elemento principal atiende a los clientes por las Interfaces HTTP y TCP y deriva las instrucciones a los distintos servidores de *backend* (uno por cada caso de uso).

La interfaz HTTP se ha implementado para dar servicio a la navegación WEB. Es el estándar de comunicación para servir páginas HTML en los navegadores.

La interfaz TCP nos permite más libertad, ya que se puede diseñar la trama a enviar. También nos ofrece más libertad a la hora de programar, ya que se está programando a más bajo nivel. Se ha utilizado tanto para la aplicación móvil como para la comunicación interna de nuestro proyecto aunque se podría haber realizado también vía HTTP.

1.4 Tecnologías utilizadas

- Android Studio para el desarrollo de la app móvil (Java).
- IDE Arduino para desarrollar la funcionalidad de cada uno de los microcontroladores (C++).
- Elementos electrónicos primarios para el funcionamiento y gobierno de la energía de cada componente de la arquitectura (Resistencias Pull-up, condensadores, alimentación, relés, etc...).
- Archimate para el modelado de la arquitectura.
- Fritzing para elaborar los esquemas electrónicos de cada dispositivo.
- Draw.io para elaborar los flujos de programa para las aplicaciones.

2 Entorno de Desarrollo

2.1 AndroidStudio

Android Studio es el entorno de desarrollo integrado oficial para la plataforma Android. Fue anunciado el 16 de mayo de 2013 en la conferencia Google I/O, y reemplazó a Eclipse como el IDE oficial para el desarrollo de aplicaciones para Android.
He seleccionado esta herramienta IDE porque me proporciona las siguientes ventajas:

- Compilar directamente sobre mi dispositivo móvil.
- Ofrece una instancia virtual de móvil para hacer pruebas.
- Permite depurar el código línea a línea tanto desde la instancia virtual como desde mi dispositivo (*debug*).
- Permite gestionar librerías externas de una manera sencilla.
- Entorno de desarrollo muy amigable con resaltado de errores y autocompletado.

Se ha utilizado la arquitectura propia de la aplicación para servir 3 *layouts* para los 3 casos de uso propuestos y una arquitectura basada en hilos para interactuar con nuestro servidor frontal de una forma dinámica y sin bloqueos en la aplicación. Todo ello implementado en el lenguaje Java. El último caso de uso ha podido ser implementado gracias a la descarga e implementación de las librerías *GraphView* y sus dependencias en AndroidStudio.

2.2 IDE de Arduino

Para la programación de los microcontroladores utilizo el IDE de Arduino. Se trata de un entorno de desarrollo en el que se realiza la programación de cada una de las placas de Arduino o módulos compatibles (como es el caso). El código se desarrolla utilizando el lenguaje C++. Utilizando las librerías públicas específicas para cada microcontrolador y periférico utilizado conseguimos la conexión al wifi y el resto de las funcionalidades que se necesiten implementar.

Cabe destacar que la mayoría de las librerías instaladas para los microcontroladores utilizados venían con una gran variedad de ejemplos para conectarlos. Esto es una ventaja importante en el trabajo con estos dispositivos, porque no existe una documentación publicada con el nivel de detalle suficiente.

Los sketches del IDE de Arduino constan de dos métodos principales: El método void setup() y el método void loop(). El método setup() se ejecuta una vez al arrancar el microcontrolador o al reiniciarlo. El método loop() es un bucle infinito donde se implementa la lógica de trabajo que se quiera.

En este proyecto, utilizaremos el método setup() en todos los servidores para conectarse a la red WIFI e inicializar los objetos necesarios para la funcionalidad correcta del método loop(). El método loop() depende de la lógica interna del servidor así que diferirá mucho entre nuestros servidores.

2.3 ArchiMate

ArchiMate [1] (originalmente de Architecture-Animate) ofrece un lenguaje común para describir la construcción y operación de procesos comerciales, estructuras organizacionales, flujos de información, sistemas de TI e infraestructura técnica. Es un lenguaje de modelado de arquitectura empresarial abierto e independiente para respaldar la descripción, el análisis y la visualización de la arquitectura dentro de los dominios de negocios de una manera no ambigua.

En este proyecto se ha utilizado tanto para elaborar un modelado inicial de la arquitectura como para explicar, de una manera muy visual, todos los componentes tanto de la solución explicada en este TFG como de la solución alternativa propuesta.

2.4 Fritzing

Fritzing [2] es una iniciativa de *hardware* de código abierto que hace que la electrónica sea accesible como material creativo para cualquier persona. Ofrece una herramienta de *software* que permite a los usuarios documentar sus prototipos, compartirlos con otros, enseñar electrónica en un aula y diseñar y fabricar PCB profesionales.

En este TFG se ha utilizado para elaborar los esquemas de los circuitos electrónicos de cada servidor backend o caso de uso propuesto.

2.5 Draw.io

Draw.io [3]es un *software* libre *cloud* utilizado para hacer diagramas y gráficos. Tienen una gran selección de formas y cientos de elementos visuales para hacer un diagrama o gráfico de manera fácil.

En este proyecto, se ha utilizado para elaborar los flujos de aplicación implementados en los distintos microcontroladores, así como para explicar tanto la lógica de nuestra página WEB como la lógica de nuestra App Android.

3 Conectividad

Dentro de la red local doméstica, cada dispositivo con capacidad WIFI tendrá su propia dirección IP para poder ser utilizado desde el resto de nuestra arquitectura. Cada uno de los microcontroladores con capacidad WIFI cuenta con una MAC *address* única que nos va a permitir configurar el mapa de direcciones estáticas de nuestro *router* como se ve en la Figura 2.

Figura 2. Mapa de Direcciones estáticas de los dispositivos del sistema

El motivo por el que hay que reservar una dirección estática para cada MAC *address* es que el modo estándar de conexión de los dispositivos a un *router* es mediante DHCP y esto hace que el *router* elija la dirección IP que te va a asignar imposibilitando la localización de dispositivos de un modo sencillo. Creando este mapa de direcciones reservamos estas direcciones IP en concreto para nuestra arquitectura domótica, indicando al *router* que no debe asignar estas direcciones IP a otros dispositivos.

Para conseguir que los clientes se conecten desde cualquier sitio con conexión a internet tenemos varias opciones:

- La primera sería tener guardada la dirección IP en nuestras aplicaciones. Esta solución, nos proporcionaría la conectividad desde el exterior durante un tiempo, hasta que la dirección IP pública de nuestro *router* local cambie. El cambio de dirección ocurre esporádicamente por lo que podría llegar a ser una opción válida.

- Otra posible solución que se ha barajado es la de contratar una dirección IP fija a tu operadora. Fue rechazada debido a su alto coste.

- La última de las soluciones, la elegida, ha sido la de registrarme en un servicio de DNS dinámico gratuito. El servicio elegido se llama *NOIP* [4] y te proporciona la resolución de un nombre hacia la dirección IP pública de tu *router*. Para configurar este servicio, es necesario darse de alta en su página y renovar tu membresía cada 30 días. El *router* ya dispone de opciones para introducir un servidor de DNS dinámico como podemos observar en la Figura 3.

Figura 3. Configuración Router DynamicDNS

El *router* publica en el servicio mencionado su dirección IP pública de manera que podamos conectarnos a ella si cambia.

En las tres opciones mencionadas, es necesario además configurar el NAT en nuestro *router* ya que, cuando el *router* recibe una petición hacia un puerto, no sabría a cuál de los dispositivos conectados a nuestra subred redirigirla. Por ello se ha realizado NAT desde los puertos de entrada 80(HTTP) y 1996(TCP) a los mismos puertos de la dirección IP de nuestro servidor Frontal. Podemos ver la configuración NAT establecida en nuestro *router* en la Figura 4.

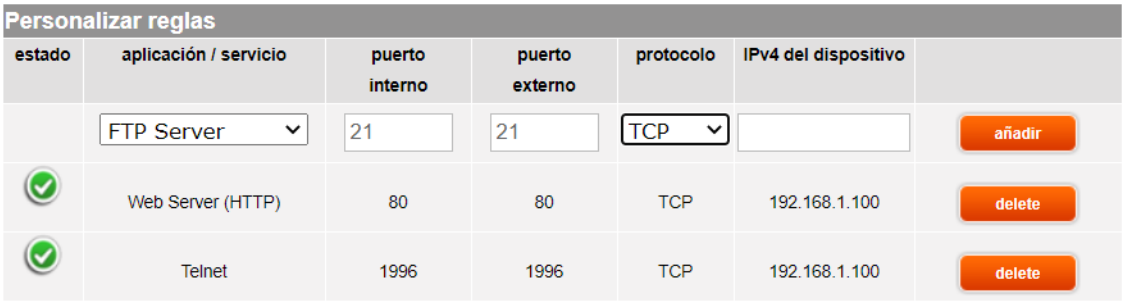

Figura 4. Configuración Router NAT

4 Hardware utilizado

4.1 Microcontrolador ESP32

El módulo ESP32 es una solución de Espressif [5]. Se ha elegido para este proyecto el modelo ESP32-WROOM-32D fabricado por Az-delivery [6]. Cuenta con Wi-Fi de 2.4 GHz de modo dual y chip Bluetooth y tecnología de baja potencia de 40nm todo en uno, integrada y certificada, que proporciona no solo la radio inalámbrica, sino también un procesador integrado con interfaces para conectarse con varios periféricos. El procesador es del modelo Tensilica Xtensa LX6 dual-core, con una frecuencia de reloj de 240MHz. Las frecuencias operativas pueden controlarse independientemente entre 80 MHz y 240 MHz. Los periféricos del procesador facilitan la conexión a una gran variedad de interfaces externas como:

Figura 5. ESP32-WROOM-32D

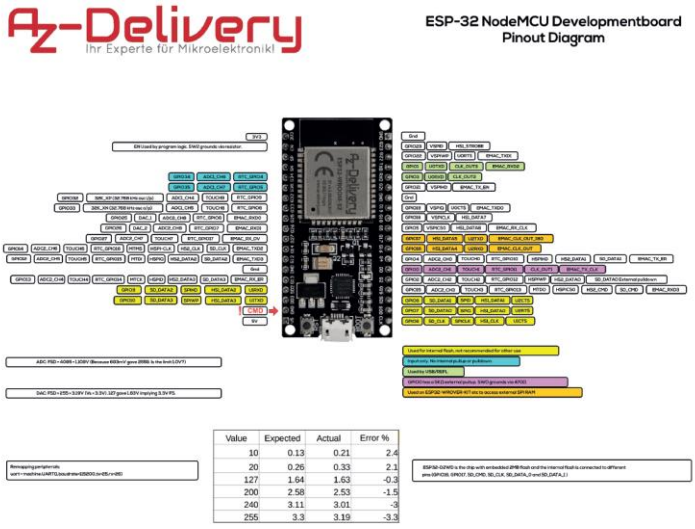

Figura 6. ESP32-WROOM-32D *PINOUT*

- Interfaz periférica serial (SPI)
- I2C
- Transmisor receptor asíncrono universal (UART)
- I2S
- Ethernet
- Tarjetas SD
- Interfaces táctiles y capacitivas

El sistema operativo en tiempo real que utiliza se denomina (FreeRTOS) [7] y permite que aproximadamente el 80% de la potencia de procesamiento esté disponible para la programación y el desarrollo de aplicaciones de usuario que consigue un consumo energético muy bajo.

Hay varios módulos ESP32 diferentes que un desarrollador puede seleccionar según sus necesidades de aplicación. El primer módulo ESP32, el más popular y el que se ha utilizado en este proyecto, es el **ESP32-WROOM-32D**, que funciona hasta 240 MHz. El ESP32-WROOM-32D utilizado lo podemos ver en la Figura 5 y sus pines en la Figura 6.

Este procesador cuenta con la característica de disponer de dos núcleos, lo que le permite trabajar en dos procesos o hilos simultáneamente. La forma de trabajar con estos dos núcleos consiste en poder lanzar tareas para que el microcontrolador gestione su ejecución.

Hay dos formas principales de lanzar tareas:

- Vincularlas a un núcleo en concreto del microprocesador (núcleo 0 o núcleo 1), utilizando el método xTaskCreatePinnedToCore();
- Hacerlas independientes (no vinculadas) y que el microcontrolador decida cuál de los dos núcleos servirá nuestra tarea (incluso usándose ambos, ya que se encarga de paginar las tareas en vuelo) utilizando el método xTaskCreate().

Existe una extensa documentación sobre los métodos a utilizar en la gestión de tareas. En nuestro caso, utilizamos ambas modalidades según el trabajo que queremos realizar. Y también se utiliza el método vTaskDelete(), que nos permite finalizar una tarea en ejecución.

4.2 Microcontrolador ESP8266-01S

ESP8266-01S [8] tiene características en chip altamente integradas. El chip ofrece confiabilidad, compacidad y robustez. Está integrado con un procesador Tensilica de 32 bits que alcanza una velocidad máxima de reloj de 160 MHz. También incorpora interfaces periféricas digitales estándar, interruptores de antena, amplificador de potencia, amplificador de recepción de bajo ruido, filtros y módulos de administración de energía. Diseñado para dispositivos móviles, dispositivos electrónicos portátiles y aplicaciones de IoT.

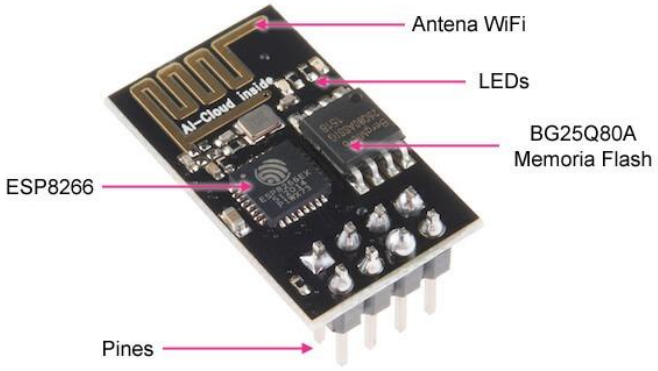

Figura 7. ESP8266-01S

El sistema operativo en tiempo real (RTOS) y la pila WiFi permiten que aproximadamente el 80% de la potencia de procesamiento esté disponible para la programación y el desarrollo de aplicaciones de usuario que consigue un consumo energético muy bajo.

En la figura 7 se pueden ver las partes más importantes. En dicha figura, ESP8266 es el microcontrolador del módulo ESP-01.Pines, donde conectaremos la alimentación, sensores y transmisión de programa. BG25Q80A es la memoria flash donde residen los programas o sketchs. El ESP8266 no dispone de este tipo

de memoria y por eso es un chip aparte. LEDs que nos informan de si está encendido o no y de la transmisión de datos (Tx y Rx). Por último, la antena WIFI para conectarse a una red/Internet.

El ESP-01 tiene 8 pines, cada uno de ellos está pensado para una tarea concreta. Podemos ver su distribución en la Figura 8.

1. GND es la toma de tierra.
2. GPIO2 es una entrada salida de propósito general. Es el pin digital número 2.
3. GPIO0 es una entrada salida de propósito general. Es el pin digital número 0.
4. RXD es el pin por donde se van a recibir los datos del puerto serie. Trabaja a 3,3 V. También se puede utilizar como pin digital GPIO: sería el número 3.

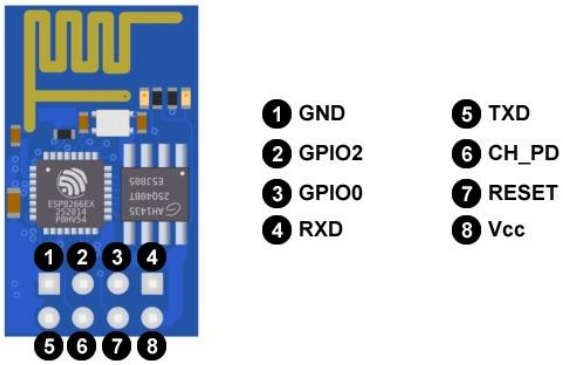

Figura 8. ESP8266-01S

5. TXD es el pin por donde se van a transmitir los datos del puerto serie. Trabaja a 3,3 V. También se puede utilizar como pin digital GPIO: sería el número 1.
6. CH_PD pin para apagar y encender el ESP-01: si lo ponemos a 0 V (LOW) se apaga, y a 3,3 V (HIGH) se enciende.
7. RESET pin para resetear el ESP-01: si lo ponemos a 0 V (LOW) se resetea.
8. Vcc es por donde alimentamos el ESP-01. Funciona a 3,3 V y admite un máximo de 3,6 V. La corriente suministrada debe ser mayor que 200 mA.

El módulo ESP32-WROOM-32 ya descrito en el punto 4.1 es una versión mejorada del ESP8266-01S e incorpora un botón en la placa que le hace entrar en modo programación. Sin embargo, el ESP8266-01S no incorpora dicha funcionalidad y hay que implementarla en hardware. La documentación dicta que hay que cortocircuitar unos pines específicos del microcontrolador para entrar en modo programación. Es por esto por lo que se ha diseñado y construido una placa con dicha funcionalidad, que se muestra en la Figura 9.

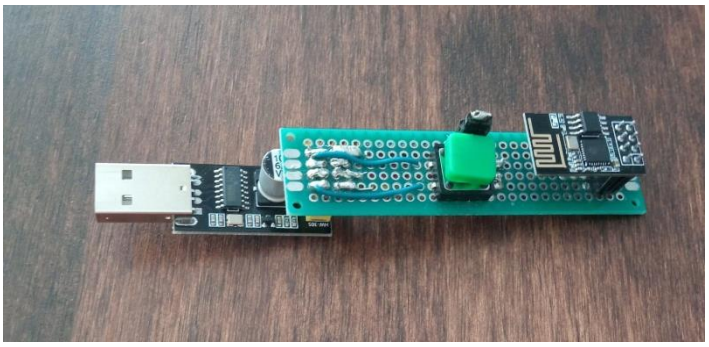
Figura 9. Montaje de la placa para la programación

El modo de funcionamiento es el siguiente: Para activar el modo de programación del ESP8266-01S se introduce en el puerto USB con el botón verde pulsado, liberándolo a continuación. Para ejecutar el programa basta con conectarlo sin pulsar el botón verde o sin la placa de programación. En la Figura 11 se observa el USB solo, mientras que el microcontrolador y el alimentador pueden verse en la Figura 10.

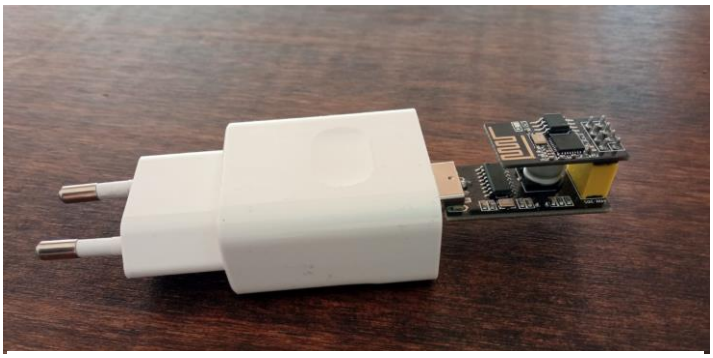

Figura 10. Montaje completo de un servidor simple

Figura 11. Adaptador USB para ESP8266

4.3 Componentes utilizados

4.3.1 Módulo lector de memoria SD

Figura 12. Modulo lector memoria SD

Este módulo mostrado en la figura 12 es completamente compatible con Arduino y con los dos microcontroladores ya mencionados (ESP8266 y ESP32).
Esta tarjeta de interfaz está diseñada para acceder a la memoria SD en modo SPI, por lo que los pines para controlar la tarjeta SD se etiquetan claramente con los nombres MOSI, MISO, SCK, CS, 5V, 3.3V y GND. Posee un regulador de tensión incluido para alimentar con 5 y suministrar al lector SD 3.3 V (Su voltaje nativo). Su interfaz de comunicación SPI permite poder utilizarlo con Arduino o PIC. En nuestro caso de uso, lo alimentamos con 3.3V ya que el microcontrolador también funciona con 3.3V. [9]

4.3.2 Sensor AM2302

Figura 13. Sensor de temperatura y humedad AM2302

El sensor AM302 mostrado en la figura 13 incorpora el sensor DHT22 en su interior [10]. Se ha elegido este sensor ya que su coste en muy reducido y es mucho más preciso y más tolerancia que su hermano menor DHT11. El protector que presenta este modelo facilita su instalación *outdoor* protegiendo el sensor.

4.3.3 Relé 5V

Figura 14. Modulo lector memoria SD

Un relé es un interruptor mecánico que es accionado eléctricamente, dejando pasar la corriente o no, y se puede controlar con voltajes bajos, como los 5v que proporciona Arduino desde sus pines de alimentación o los digitales. Podemos verlo en la figura 14.
Un relé desactivado tiene unidos el COMÚN y el NC (*Normally Closed* – Normalmente Cerrado), y cuando es activado quedan unidos el COMÚN y el NO (*Normally Open* – Normalmente abierto).
El módulo relé tiene además de los pines ya mencionados 3 patillas: Vcc, IN y GND, donde Vcc es la alimentación de 5v, IN la entrada de voltaje y GND la tierra. Normalmente en las entradas de los relés van conectadas la línea de conexión positiva del dispositivo que se vaya accionar.

4.3.4 Estabilizador LM2596

Figura 15. Estabilizador LM2596

El convertidor de voltaje DC-DC Step-Down 2A LM2596 [11] tiene como función entregar un voltaje de salida constante inferior al voltaje de entrada frente a variaciones del voltaje de entrada o de carga. Soporta corrientes de salida de hasta 3A, voltajes de entrada entre 4.5V a 40V y voltajes de salida entre 1.23V a 37V. El voltaje de salida se selecciona mediante un potenciómetro multivuelta. Podemos verlo en la figura 15.

4.3.5 Display LCD 16x2

La pantalla LCD 16x2 de AZ-Delivery consiste en una práctica pantalla LCD de 2x16 caracteres y una interfaz I2C con 6 direcciones I2C seleccionables, con la que se pueden operar hasta 6 pantallas simultáneamente [12].
Gracias a la interfaz I2C, la pantalla tiene una visualización rápida de transmisión y muestra 2 líneas de 16 caracteres de color negro sobre fondo verde. El contraste se puede ajustar mediante un potenciómetro, el cual hemos utilizado en nuestro proyecto.

Figura 16. *Display* LCD 16x2

En la figura 16 podemos observar la parte delantera y trasera de un LCD 16x2.

5 Capa *Backend*

5.1 Servidor Luces de acuario

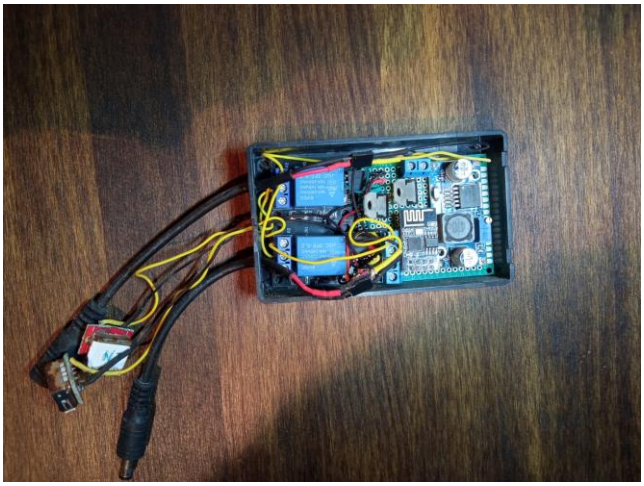

Figura 17. Montaje Servidor Acuario

El servidor de Acuario controla dos luces LED situadas en un acuario. Una de ellas es una tira RGB de 12V y la otra una lampara led 5V.
El servidor de acuario solo precisa de dos pines de señal para la actuación de los dispositivos. Por lo que es suficiente utilizar el módulo ESP8266-01S que le proporciona capacidad WIFI y los dos pines necesarios.
Podemos ver el montaje final en la figura 17.
Desde el cliente es posible realizar las siguientes llamadas:

- "LEDS": Enciende/Apaga, según el estado en que se encuentre, la lámpara led de 5V. Devuelve el estado actual de ambas lámparas.
- "TIRA": Enciende/Apaga, según el estado en que se encuentre, la tira led de 12V. Devuelve el estado actual de ambas lámparas.
- "EstadoAcuario": Devuelve el estado actual de ambas lámparas, sin actuar sobre ellas.

5.1.1 Descripción del Circuito

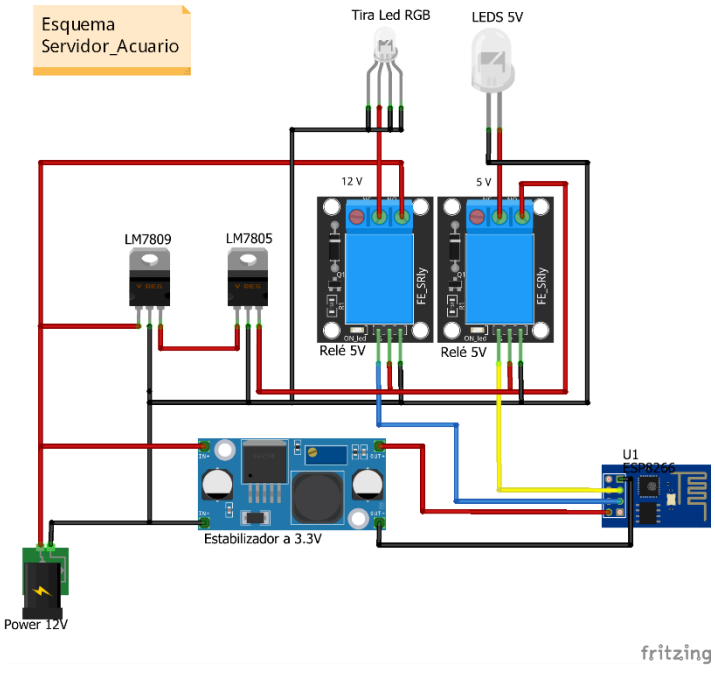

Figura 18. Esquema Servidor Acuario

El circuito recibe la tensión de una fuente de alimentación de 12V DC. Esta fuente de 12V está conectada al estabilizador LM2596 que baja la tensión a los 3.3V necesarios para alimentar el ESP8266 proporcionándole una tensión estabilizada (sin picos) para proteger el microcontrolador. Esta fuente de alimentación también alimenta a la tira led de 12V realizándose el apagado y encendido mediante un relé de 5V.

La tensión de 5V se consigue mediante 2 transistores reguladores de tensión dispuestos en serie LM7809 y LM7805 que bajan la tensión de 12V a 9V y de 9V a 5V respectivamente. Con el segundo regulador LM7805 hubiera sido suficiente

ya que aguanta voltajes de entrada de hasta 18V, pero se recomienda bajar la tensión en dos fases para evitar el calentamiento de los dispositivos si van a usarse.
Podemos ver en la figura 18 el fritzing elaborado con el esquema de componentes.

5.1.2 Lógica del Servidor

La funcionalidad del servidor de *backend* es simple, al recibir uno de los tres posibles comandos ("EstadoAcuario", "LEDS" o "TIRA") desde el servidor frontal, devuelve el estado o enciende/apaga cada uno de los dispositivos y contesta con el estado. La Figura 19 muestra el flujo del Servidor de Acuario, que se describe a continuación.

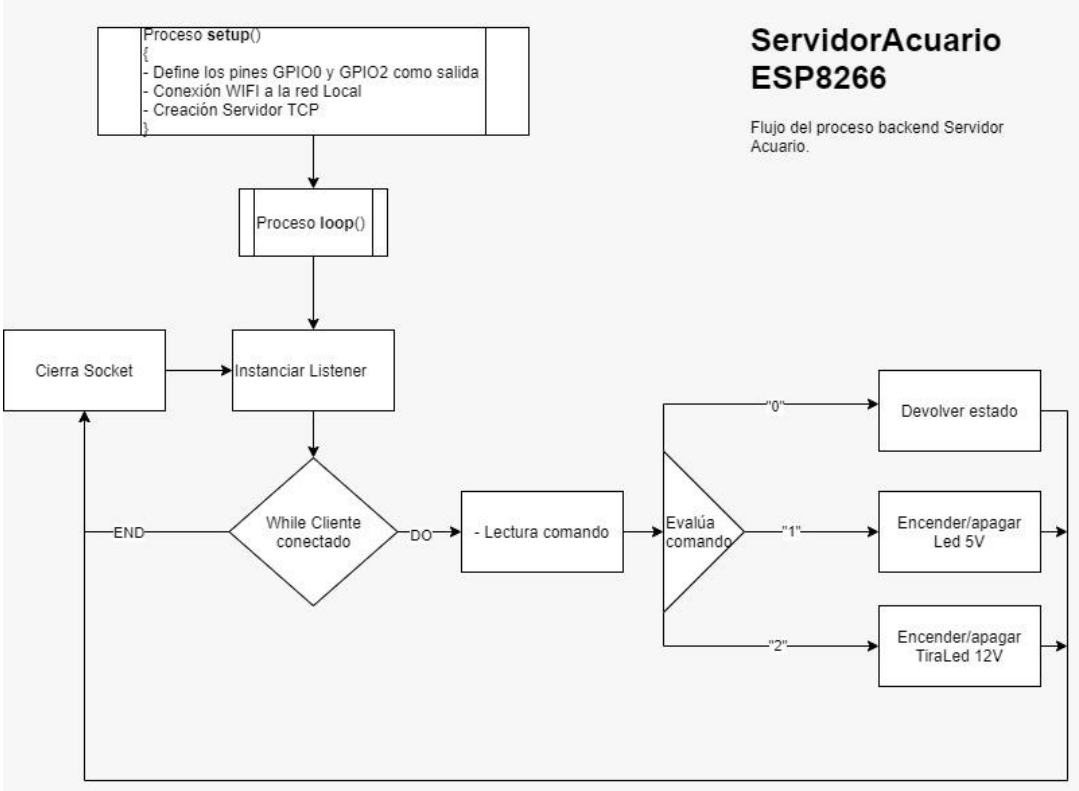

Figura 19. Flujo Servidor Acuario draw.io

EL método setup() realiza la conexión al *router*, reserva los pines de salida del microcontrolador e instancia el *listener* propio de un servidor TCP.
El bucle infinito del proceso o "loop()" está constantemente preguntando si se ha conectado un cliente. Cuando recibe una conexión del servidor frontal lee el comando entregado y realiza una de las tres acciones posibles:

- Devolver los estados de tensión de los dos relés. para que el cliente pueda conocer si las luces están apagadas o encendidas. Cada relé tiene un estado de tensión de señal que será alto (3.3V) o bajo (0V), que cambiará el estado físico del mismo.

Para saber el estado de tensión del relé nos basta con utilizar el método *digitalRead(pin_number)* para saber si el microcontrolador le está suministrando tensión cerrando así el circuito del relé.

- Cambiar el estado del relé de 5V ("LEDS"). Si está apagado, lo enciende. Si está encendido, lo apaga. Para activarla envía una señal alta por el pin GPIO0, o sea 3.3V. Para desactivarlo envía al mismo pin una señal baja 0V. Esto se consigue utilizando el método *digitalWrite(pin_number, HIGH|LOW)*. Para elegir el valor HIGH o LOW previamente se utiliza el método *digitalRead(pin_number)* como en el caso anterior. Termina el proceso contestando con el estado de los relés al servidor y cerrando la conexión.
- Cambiar el estado del relé de 12V ("TIRA"). Si está apagado, lo enciende. Si está encendido, lo apaga. Como en el caso anterior, lo realiza poniendo una tensión de 3.3V o 0V, pero esta vez sobre el GPIO1. Termina el proceso contestando con el estado de los relés al servidor y cerrando la conexión.

Una vez realizado el comando, queda de nuevo en espera de una nueva solicitud.

5.2 Riego Jardín

El Servidor Riego controla los solenoides de dos electroválvulas situadas en un jardín. Cada una de ellas riega una zona distinta del jardín. Las electroválvulas funcionan con 9V DC. La particularidad de estas electroválvulas es que tanto en estado abierto como en estado cerrado funcionan de un modo pasivo. Es decir, necesitan tensión para realizar el cambio de abierto a cerrado. La tensión a aplicar en cada estado es **inversa** (0V-9V para abrir y 9V-0V para cerrar).
Cuando aplicas la diferencia de potencial adecuada sobre la electroválvula, deja pasar el agua y si dejas de aplicar esta diferencia de potencial, el agua sigue pasando. Para cerrar la electroválvula será necesario aplicarle la tensión opuesta.

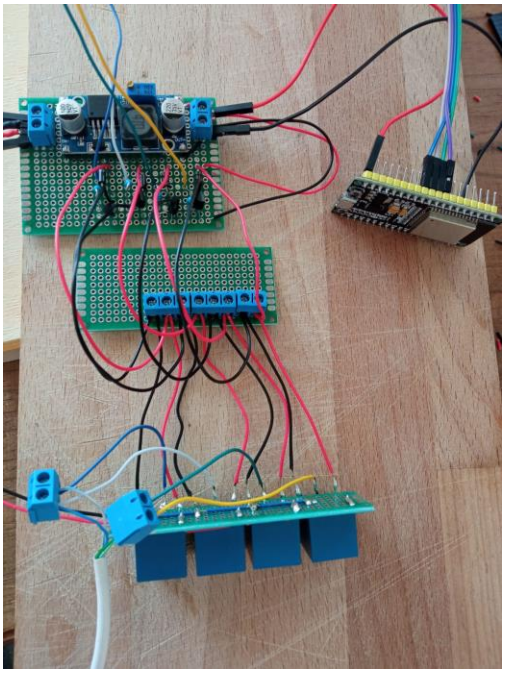

Figura 20 Montaje servidor Riego

Esto obliga a utilizar 2 relés para controlar cada electroválvula. Es por esto por lo que se ha elegido un ESP32-WROOM-32D que dispone de múltiples pines para controlarlos.
Podemos observar el montaje final soldado de nuestro servidor de riego en la figura 20.

5.2.1 Descripción del Circuito

Cada relé recibe 0V en el pin NC (Normally Close - Normalmente Cerrado) y 9V en el NO (Normally Open - Normalmente abierto). Sacando por la pata común 0V cuando no recibe señal del microcontrolador y 9V cuando la recibe.

El circuito actuador de los relés (9V) es independiente del circuito de control de 5V.

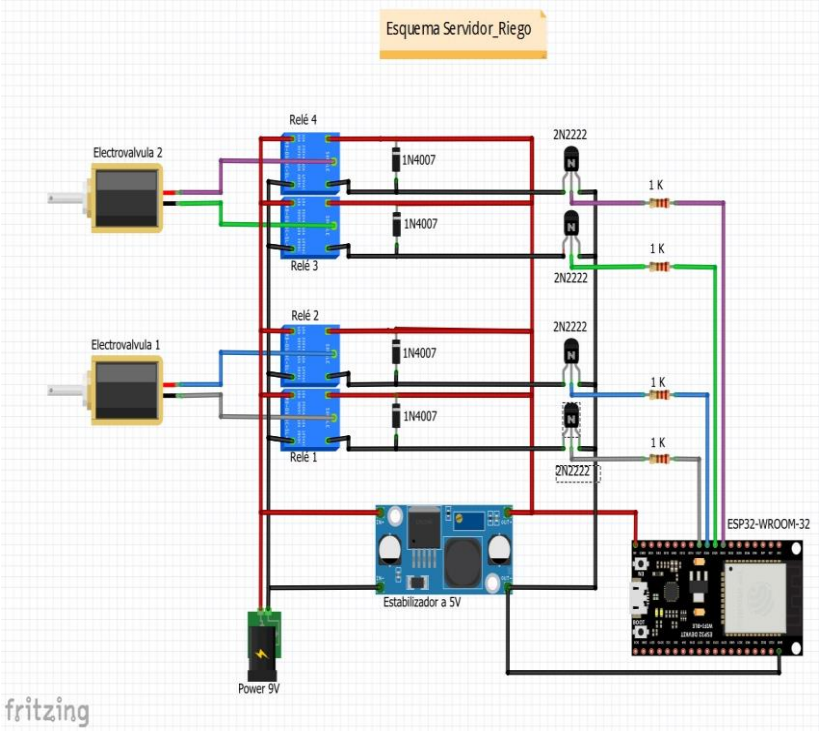

Figura 21. Esquema Servidor Riego

El circuito de control de la bobina de cada electroválvula se compone de 1 transistor NPN 2N2222, que conecta su base al GPIO del microcontrolador mediante una resistencia de 1KΩ. También se ha añadido un diodo 1N4007 para permitir la descarga de la bobina.

Todas las conexiones se representan con el fritzing que observamos en la Figura 21.

5.2.2 Lógica del Servidor

La Figura 22 muestra el flujo del Servidor de Riego, cuyo funcionamiento se describe a continuación. La lógica de este Servidor es la siguiente: Hay 5

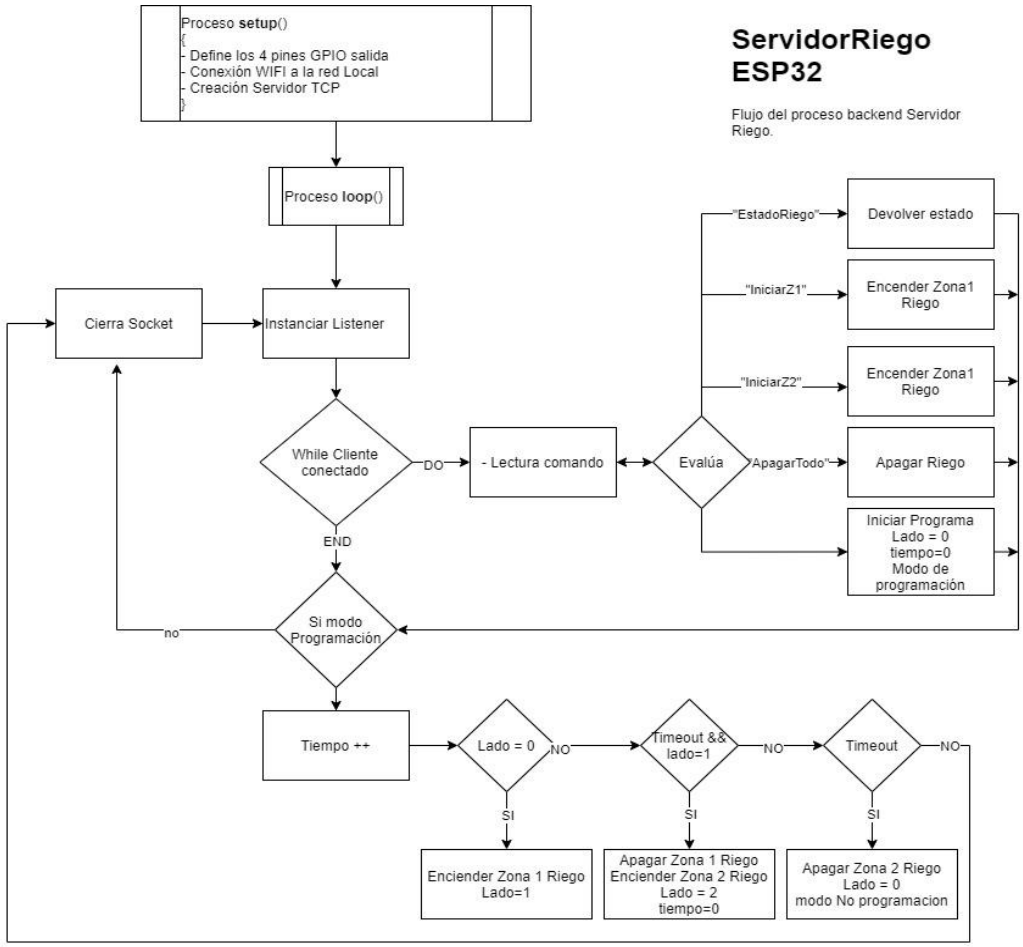

Figura 22 Flujo Servidor Riego Draw.io

Comandos posibles a recibir (“IniciarZ1”,”IniciarZ2”,”ApagarTodo”,”EstadoRiego” e “IniciarPrograma”)

- Al recibir los comandos IniciarZ1 o Iniciar Z2 se enciende uno de los dos lados de riego y devuelve el estado “Lado Derecho/Izquierdo Encendido”.
- Al recibir el comando “ApagarTodo” se apaga la Z1 y la Z2 y se devuelve el estado “RiegoApagado”.
- Al recibir el comando “EstadoRiego” se devuelve el estado de riego.
- Al recibir el comando “Iniciar Programa” se activa el modo programación encendiendo primero el lado izquierdo. Al finalizar el timeout apaga el lado izquierdo, enciende el lado derecho y reinicializa el timeout. Por último, al finalizar este timeout apaga el lado derecho.
- También es posible recibir el comando con dos campos: “Iniciar Programa” y el número de minutos “minutos” para programar la secuencia de riego.

5.3 Estación meteorológica

El servidor Clima está situado en la capa de *backend* y será el encargado de recopilar, validar, clasificar, y transmitir los datos para mostrarlos en las aplicaciones cliente en forma de dos gráficas. Estas gráficas podrán ser actualizadas bajo petición y los clientes podrán variar la escala de visualización de estas (años, meses, días y horas).

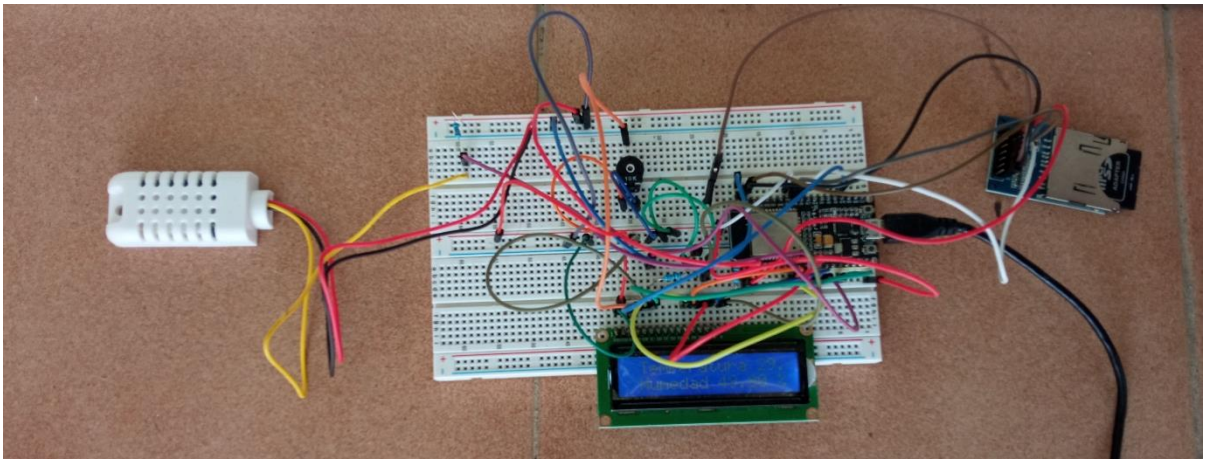

Figura 23 Montaje Servidor Clima Draw.io

Para conseguir nuestro objetivo, se ha diseñado el servidor de *backend* ya mencionado, llamado Servidor Clima.
El servidor clima se ha programado sobre un microcontrolador ESP32. Hemos elegido este microcontrolador en vez de la versión más antigua y reducida, la cual tiene un menor consumo, por dos motivos principales:

- Numero de pines: Como ya sabemos el número de pines del ESP8266-01S es muy limitado tal y como se aprecia en la Figura 8. Dado que en este caso de uso vamos a necesitar un sensor para recopilar los datos de humedad y temperatura, una tarjeta SD para almacenar los mismos y además un *display* LCD que nos proporciona la lectura cada minuto, se precisa un microcontrolador con más pines.
- Capacidad Dual-Core: Necesitaremos hacer uso de la capacidad Dual-core de nuestro microcontrolador ESP32 ya que en este caso de uso vamos a necesitar incluir la lógica de comunicación propia de un servidor de *backend* y la lógica de toma de valores y escritura en la base de datos, así como de visualización en el *display* LCD. No es una base de datos como tal, sino que está implementada como FileSystem (arquitectura de carpetas en árbol) implementada sobre una tarjeta SD de 32Gb.

Podemos observar el montaje final del servidor clima en la figura 23.

5.3.1 Descripción del Circuito

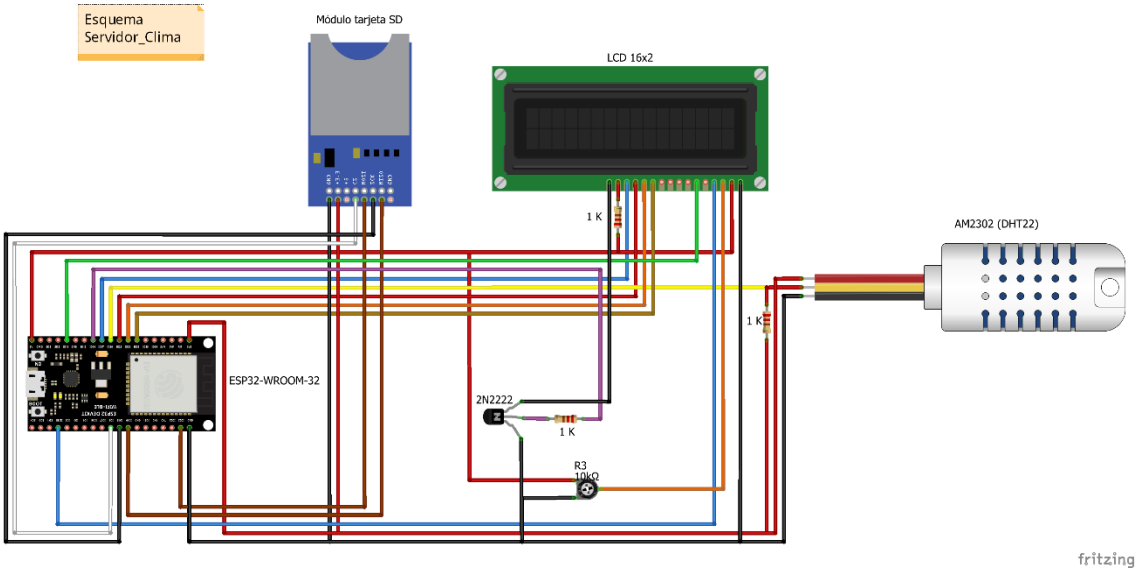

Figura 24 Flujo Servidor Clima

Este circuito mostrado en la figura 24 se alimenta con un cable USB a 5V en la entrada del microcontrolador. Será el microcontrolador quién se encargue de alimentar los otros tres componentes existentes en nuestro circuito. El sensor AM2302 y el lector de tarjetas SD funcionan a un voltaje de 3.3V mientras que la pantalla LCD de 16x2 funciona a 5V. El microcontrolador es capaz de suministrar ambos voltajes en los pines 3.3V y 5V respectivamente siempre que se alimente con 5V. En nuestro caso, alimentamos el microcontrolador mediante la entrada micro-USB que dispone en su placa y que suministra 5V.

El lector de tarjetas SD tiene 6 pines que son MOSI, MISO, SCK, CS, 5V, 3.3V y GND. Usaremos todos los pines exceptuando la entrada de 5V. La librería del dispositivo prefija los pines a utilizar desde el microcontrolador como podemos ver en la Figura 25. La librería SD.h del IDE de Arduino incluye varios ejemplos explicativos del funcionamiento de los métodos para la gestión de la tarjeta SD.

MicroSD card module	ESP32
3V3	3.3V
CS	GPIO 5
MOSI	GPIO 23
CLK	GPIO 18
MISO	GPIO 19
GND	GND

Figura 25 Tarjeta SD mapa de pines

Para la actuación sobre el *display* LCD se utiliza la librería LiquidCrystal.h, que dispone de métodos de alto nivel para enviar mensajes de texto a la pantalla y hacer el *scroll* horizontal de los mismos. Los pines utilizados desde el microcontrolador se especifican en la inicialización del microcontrolador en la creación del Objeto lcd "LiquidCrystal lcd (15, 13, 32, 33, 25, 27); "

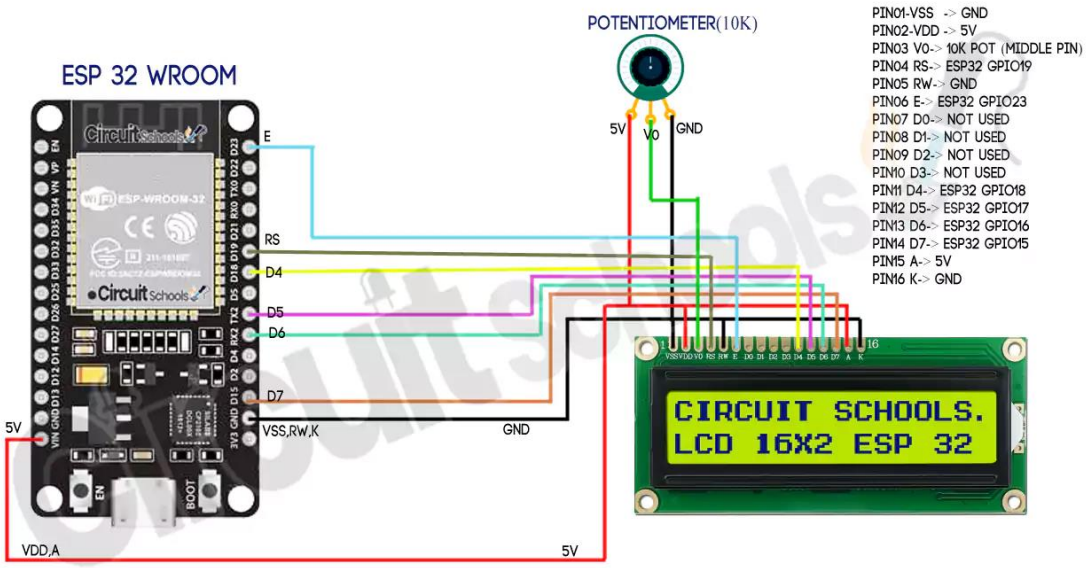

Figura 26 Mapa de conexiones ESP32-displayLCD

En la Figura 26 se explica el mapa de pines para conectar el ESP32 y el *display* LCD. Cabe destacar, la utilización del potenciómetro de la figura para aumentar el contraste y conseguir obtener un fondo distinto de verde.
Además de esta funcionalidad, hemos implementado en hardware la capacidad de apagar la retroalimentación del *display* desde el microcontrolador para que no esté constantemente iluminado gracias a un transistor NPN (Figura 24). Este display LCD se utiliza para mostrar con un scroll horizontal la lectura obtenida desde el sensor cada minuto.

El sensor AM2302 se conecta con sus pines de tensión a los pines 3.3V y GND del microcontrolador y el tercero de sus pines sirve para recibir los datos del microcontrolador conectándolo a una de sus entradas digitales mediante una resistencia de 1K que especifica el fabricante.

5.3.2 Lógica del servidor

La Figura 27 muestra el flujo del Servidor de Clima, cuyo funcionamiento se describe a continuación: El servidor Clima, en su proceso setup() realiza las siguientes instrucciones:

- Conexión WIFI a la red local como el resto de servidores.
- Inicialización del sensor de temperatura
- Inicialización del displayLCD
- Inicialización del lector de tarjetas SD
- Inicialización listener propio de un servidor
- Se arranca en segundo plano una tarea ligada al core 0 mediante el método xTaskCreatePinnedToCore(0);

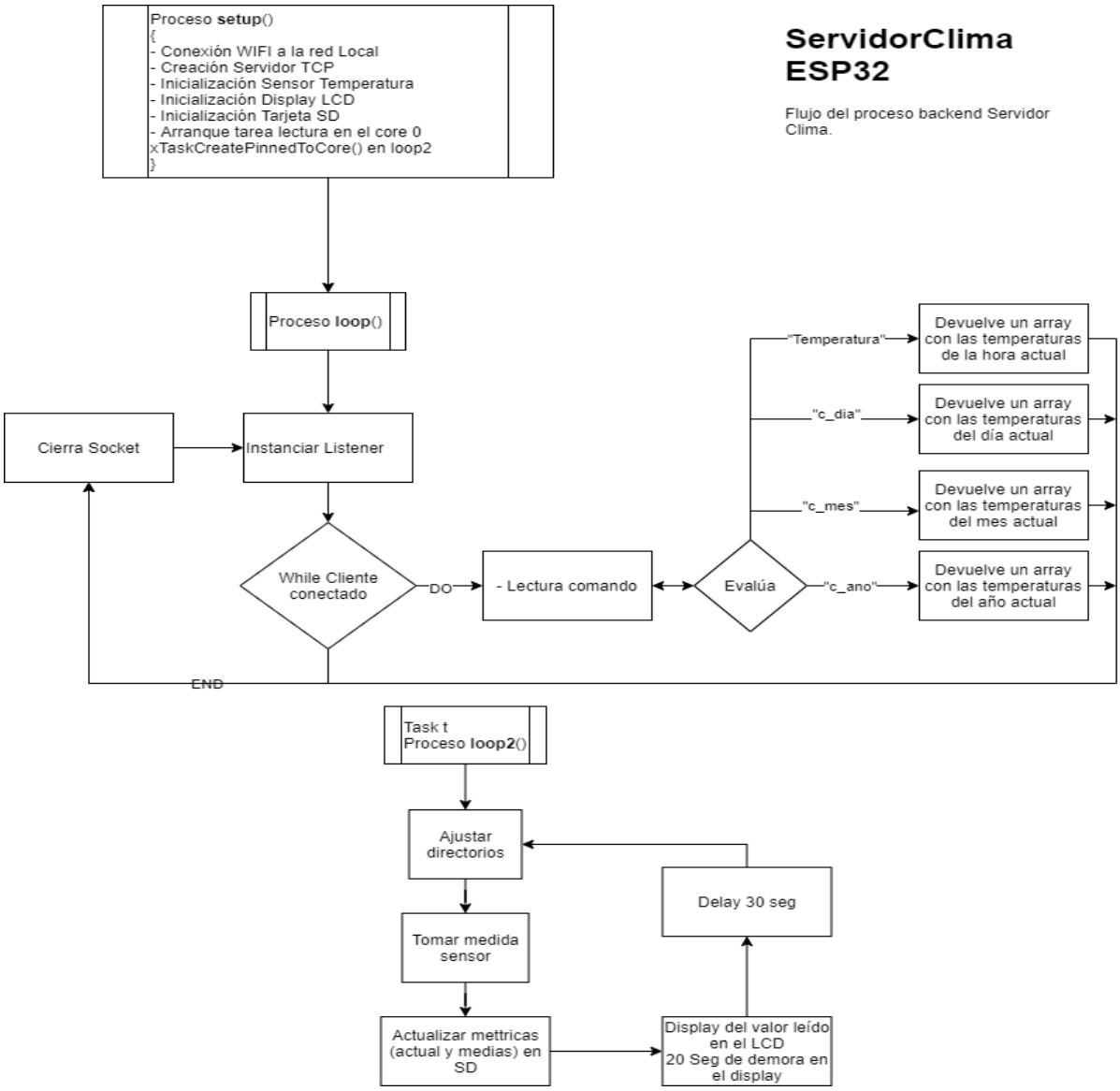

Figura 27 Flujo Servidor Clima Draw.io

Esta tarea secundaria se arranca dentro de un bucle infinito y se encarga de preguntar a un servicio externo la hora actual a través de internet. Este servicio se llama "pool.ntp.org". Si al recibir la información del tiempo actual, el minuto ha cambiado, la tarea se encarga de crear la estructura de directorios si fuese necesaria, así como escribir en el fichero correspondiente a la hora en curso los valores leídos por el sensor. A continuación, con todos los valores de la hora grabados en el fichero, se actualiza la media de dicha hora en el fichero de medias del día, que tendrá un total de 24 registros al finalizar el día. Se calcula de nuevo la media de este día para actualizar la media del mes en curso. Y lo mismo se hace con la media del año. De esta manera mantenemos en cada lectura actualizadas todas las medias anteriores. A continuación, se visualiza en el displayLCD el valor leído y se espera 50 segundos para volver a iniciar el bucle y, con él, la consulta de la hora.

En la tarea principal se escuchan posibles peticiones del servidor frontal. En función del comando leído, se ofrecen los valores correspondientes a las lecturas de la hora, día, mes y años actuales.

6 Capa Frontal

Para resolver la capa frontal, se han diseñado e implementado dos arquitecturas distintas:

- Solución A: Utilización de un solo microcontrolador para resolver la lógica cliente o lógica de canal de nuestro proyecto.
- Solución B: Utilización de 3 microcontroladores que mantienen desacopladas las capas frontal y middleware.

El motivo por el que se han desarrollado dos soluciones es que ambos planteamientos son igualmente válidos y cada uno de ellos ofrece sus ventajas y desventajas. Desde un principio, se pensó implementar la solución A.

Esta primera solución, aunque es más simple electrónicamente, ocasionó muchos problemas en la lógica software en el microcontrolador como la gestión de la memoria en las tareas en segundo plano, por lo que se diseñó e implementó la solución B. Estos problemas se solventaron más tarde volviendo a implementar la solución A.

6.1 Solución alternativa (Solución B)

Como podemos observar en la Figura 28, la arquitectura de nuestro proyecto cambia totalmente respecto a la planteada en la introducción del TFG, en

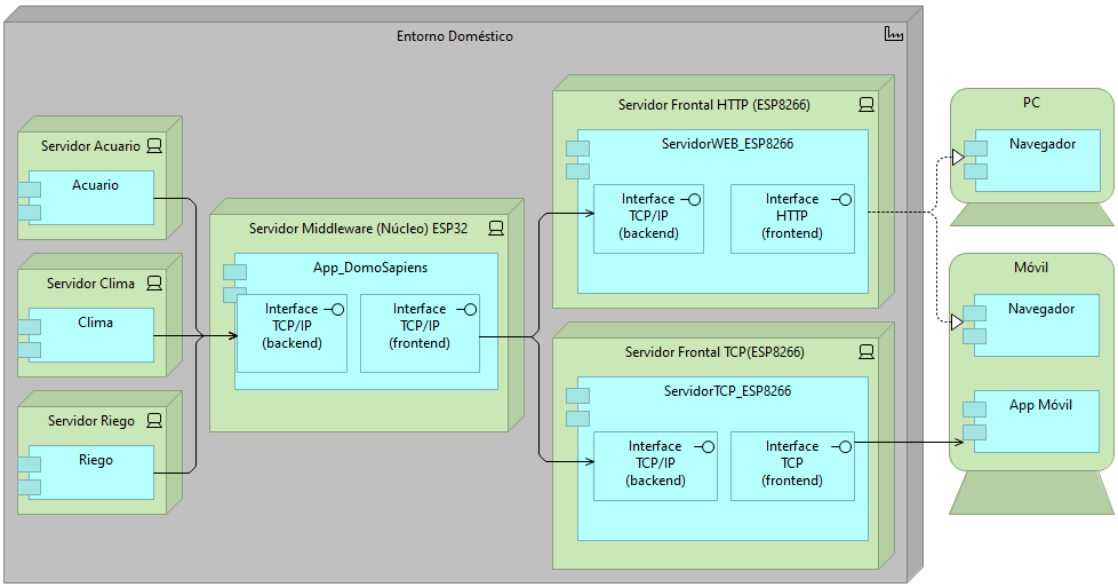

Figura 28. Esquema de la arquitectura de la solución alternativa

concreto, la capa frontal. La solución propuesta ha sido la de desacoplar la capa frontal de la capa de *middleware*, otorgándole menos carga de trabajo a un microcontrolador y solucionando el problema surgido inicialmente relacionado con la gestión de tareas, ya que aparece el concepto de *middleware*. El servidor *middleware* descrito en la figura, se comunica con ambas interfaces al mismo tiempo, la interfaz HTTP y la interfaz TCP evitando así bloqueos.
La arquitectura de nuestro proyecto en caso de implementar esta solución alternativa sería el siguiente:

- Capa Cliente: Aplicación Android con las opciones de usuario para la gestión de los dispositivos domóticos. A esta capa también pertenece cualquier navegador que acceda a nuestro servidor web vía HTTP.
- Capa Frontal: La capa frontal consta de dos puntos de acceso sobre dos servidores implementados en controladores ESP8266-01S. Cada uno de ellos está dedicado a una interfaz específica (HTTP o TCP). Estos servidores se conectan al *middleware* que se encargará de enviarlo a la capa *backend*.
- Capa Middleware: La capa *middleware* hace de núcleo de la arquitectura resolviendo la lógica de comunicación hacia los distintos servidores de *backend* denominado Servidor Middleware en la Figura 28.
- Capa Backend: "Servidor Acuario, Servidor Clima, Servidor Riego". Es un conjunto de dispositivos basados en los microcontroladores ESP32 y ESP8266-01S para la actuación de los distintos elementos domóticos.

6.1.1 Capa Frontal

La capa frontal consta de dos dispositivos que hacen de interfaz entre la capa cliente y la capa middleware. Estos dispositivos se denominaron ServidorTCP_ESP8266 y ServidorWEB_ESP8266 en la Figura 28.

Ambos dispositivos reciben comandos de la capa cliente y unifican los comandos enviándolos al servidor *middleware*, desacoplándolo de la capa cliente ya que ambos dispositivos implementan la misma funcionalidad.

Las dos implementaciones de la capa frontal se han resuelto con la misma arquitectura hardware (Figura 10). Que consta de 3 componentes:

1. Alimentador USB a 5V.
2. Adaptador USB para ESP8266. La importancia de este adaptador es que es capaz de suministrar 3.3V desde una fuente estándar USB a 5V, ya que el módulo ESP8266-01S trabaja a 3.3V.
3. Microcontrolador ESP8266-01S que funciona de modo independiente ya que solo consta de lógica de comunicación WIFI.

6.1.1.1 ServidorTCP_ESP8266

El servidor TCP está cargado con un código C++ desarrollado en el entorno IDE de Arduino. El esquema de procesos se ha elaborado con draw.io y lo podemos ver en la figura 29.
Tal y como se ha explicado anteriormente, los sketches del IDE de Arduino constan de dos métodos principales: el método void setup() y el método void loop(). El método setup se ejecuta una vez al arrancar el microcontrolador o al reiniciarlo. El método loop es un bucle infinito donde se implementa la lógica de trabajo que se quiera.
En nuestro caso, el método Setup() se conecta a la red local e inicia el Servidor TCP (puerto 1996).
El método loop() contiene la lógica de comunicación con la aplicación móvil y la llamada al servidor middleware para trasladarle el comando seleccionado por el cliente así como recibir la respuesta del *middleware* y retransmitirla al cliente.

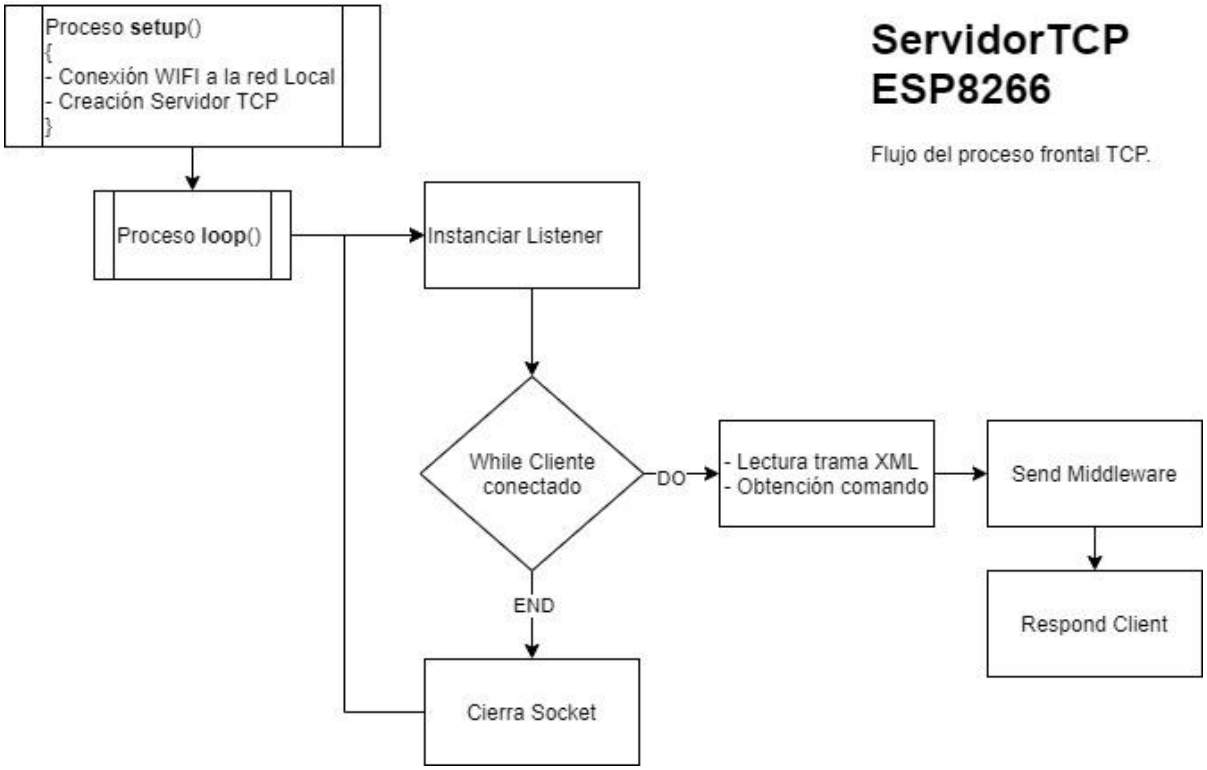

Figura 29. Flujo Servidor TCP Draw.io

6.1.1.2 ServidorWEB_ESP8266

El servidor WEB está cargado con un código C++ representado en la figura 30 está desarrollado en el entorno IDE de Arduino estructurado de forma análoga al servidorTCP.
Este servidor web ofrece la misma funcionalidad que el anterior (en el puerto 80) y la lógica de presentación de nuestra página web (HTML) al cliente.
La lógica de presentación ofrecida no solo se encarga de enviar la página, sino también de evaluar los comandos recibidos por el middleware e ir actualizando dicha lógica de presentación coloreando los botones en función del estado de los dispositivos.

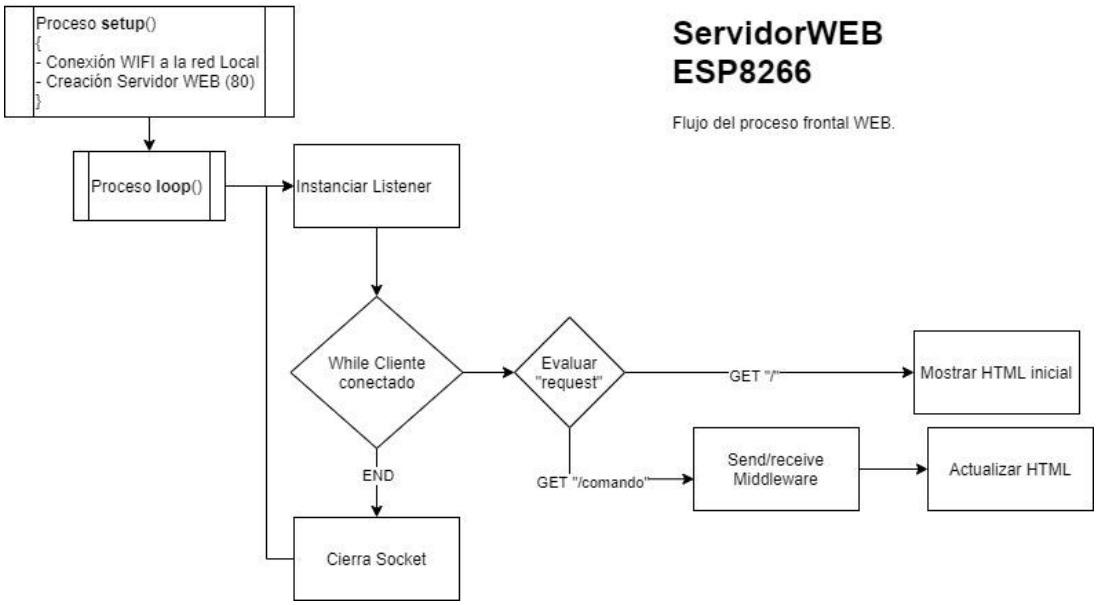

Figura 30. Flujo del servidor WEB draw.io

6.1.2 Capa Middleware

La capa middleware o núcleo, tiene como elemento principal el microcontrolador ESP32. El servidor Middleware está desarrollado con el IDE de Arduino en C++ representado en la figura 31. Su lógica está desacoplada de la lógica frontal respondiendo de igual manera independientemente del frontal con el que interactúa. Tiene almacenadas en memoria las direcciones IP de todos los *backends*. Su funcionamiento es el siguiente:

Primero, en el método setup, nuestro microcontrolador se conecta a la red local e inicia un servidor TCP que queda a la escucha de clientes Web y TCP. La comunicación con los frontales se hace mediante una trama xml predefinida.
←<request><comand></comand></request>
→<response><ack></ack><estado></estado></response>

En función del comando recibido, elige el *backend* al que tiene que redirigir la petición realizada por el cliente. Cuando recibe la "<response></response>", la devuelve al socket Frontal con el que dialoga.

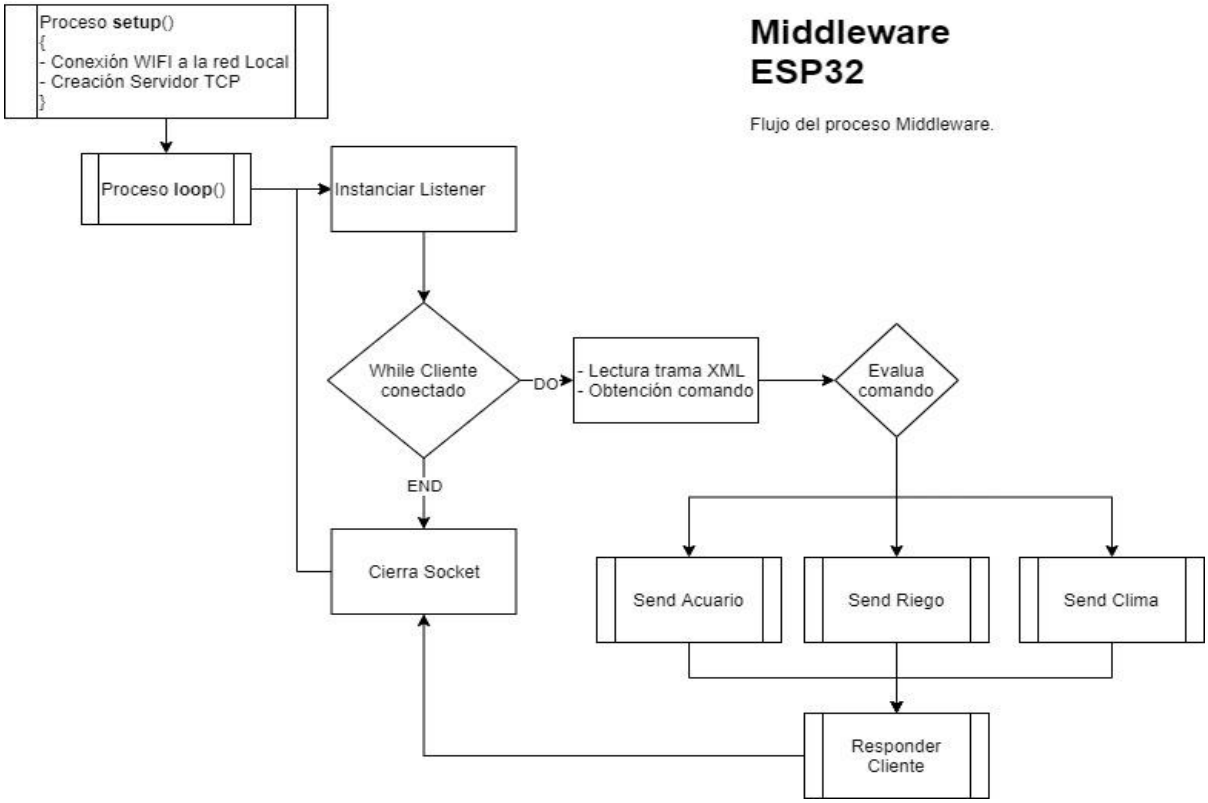

Figura 31. Flujo Servidor *Middleware* Draw.io

6.1.3 Ventajas

Este nuevo diseño de la arquitectura tiene la ventaja fundamental de tener las capas Frontal y Middleware desacopladas. Esto permite implementar a futuro otras interfaces como la de infrarrojos, sin tener que tocar el resto de los elementos de nuestra arquitectura.
También cabe destacar la ventaja en la mantenibilidad, si las interfaces están separadas. Ante un posible error, solo caería una de las interfaces y no todo nuestro servidor frontal y núcleo de la aplicación. Esta solución, aunque es más costosa de implementar inicialmente, me parece una solución mejor en un entorno con más casos de uso de los propuestos en la solución inicialmente planteada.

6.1.4 Desventajas

La principal desventaja es el retardo en las peticiones, al haber un nodo más, aumenta el retardo extremo a extremo. También aumenta la probabilidad de error, aunque no se llega a apreciar.
La desventaja en recursos no es considerable ya que el incremento de microcontroladores utilizados no consume prácticamente nada y su coste es muy bajo.
Esta solución requiere de más electrónica para alimentar todos los dispositivos.

6.2 Solución propuesta (Solución A)

Finalmente, tras superar los problemas en la gestión de las tareas, el proyecto se ha resuelto implementando una sola pieza que hace de frontal y de núcleo de la aplicación tal y como se explica en la sección 1.3 Solución Propuesta.
Las dos interfaces de comunicación con el cliente (HTTP y TCP) son implementadas por dos *listener* que corren uno en cada core permitiendo que se puedan estar utilizando ambas simultáneamente y sin bloqueos del mismo modo que en la solución B.
El core dedicado a la interfaz HTTP tiene que servir las páginas al cliente con los estados ya actualizados del mismo modo que lo hacía el servidor WEB_ESP8266 y como ya se ha explicado en el punto 6.1.1.2.
El core dedicado a la interfaz TCP se encarga de servir a los posibles clientes Android tal y como lo hacía el servidor TCP_ESP8266 descrito en el punto 6.1.1.1 de la solución B.

Ambos procesos de ambos núcleos, representados con líneas roja y verde respectivamente en la figura 32, utilizan los mismos métodos para comunicarse con los *backends*, tal y como se explica en la capa *middleware* descrita en el punto 6.1.2.

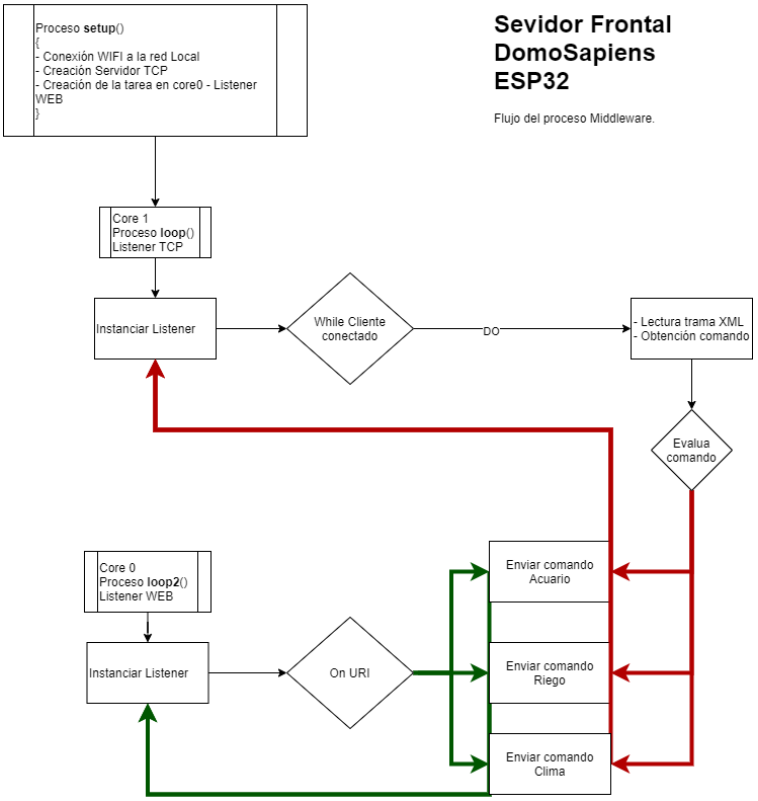

Figura 32. Flujo Servidor Frontal solución A Draw.io

6.2.1 Ventajas

La ventaja principal de esta arquitectura es la reducción del número de componentes, ya que concentramos el servicio que antes dábamos con 3 microcontroladores en un solo micro.
Otra ventaja es la reducción por tanto del consumo eléctrico y el número de alimentaciones necesarias.

6.2.2 Desventajas

Pese a ser la solución elegida, la desventaja de esta implementación es importante, ya que en el caso de querer incrementar el número de tecnologías cliente (p.e. infrarrojos) no podríamos dedicarle un *core* a dicha solución porque este microcontrolador solo tiene 2 *cores* y tendríamos que ir paginando las tareas de escucha haciendo mucho más complejo en desarrollo.

7 Capa cliente

La capa cliente es la capa encargada de proporcionar la lógica de presentación de nuestro proyecto.
Se han diseñado y programado dos aplicaciones. Una es la app móvil desarrollada con Android y la otra es el conjunto de páginas servidas desde el servidor Frontal WEB ejecutado sobre el navegador del cliente.

Esta aplicación, aprovecha la capacidad de comunicación del terminal móvil para poder acceder al servidor desde dentro o fuera de la red local.

7.1 Aplicación Android

La aplicación Android utilizada está desarrollada con AndroidStudio y, proporciona los flujos de pantalla necesarios para abordar los 3 casos de uso propuestos.

La estructura de tramas utilizada está basada en etiquetas xml, dispuestas de la siguiente forma:
→ "<request><comando>**comando**</comando></request>"
←"<response><ack>**ack**</ack><estado>**estado**</estado></response>".

Las tramas pueden tener sub-etiquetas en función del servicio, pero siempre responderán a este patrón. Estas variaciones serán explicadas en cada servicio.

La aplicación principal denominada MainActivity consta de 4 botones. 3 de ellos que sirven las distintas *activities* para los tres casos de uso propuestos y el cuarto es para levantar un navegador en el cliente, pasando de ser "clienteAndroid" a "clienteWeb", interactuando con el servidor frontal vía HTTP.

7.1.1 Servicio Luces Acuario

El servicio de luces de acuario tiene la estructura de tramas ya descrita:
→"<request>
<comando> **comando** </comando>
</request>".
← "<response>
<ack> **ack** </ack>
<estado> **estado** </estado>
</response>".

Las posibilidades dentro de esta trama son las siguientes:
Comandos:

- "LEDS" -> Enciende o apaga los leds
- "TIRA" -> Enciende o apaga la tira
- "EstadoAcuario" -> Pide el estado del acuario

Estados desde el servidor Frontal:

- "Leds Encendidos"
- "Leds Apagados"
- "Los dos Encendidos"
- "Los dos Apagados"
- "Error en evaluación luces"

ACK: "0" - "1".

Con la carga del *layout,* figura 33, lo primero que hace nuestra aplicación es preguntar el estado de las luces al servidor para colorear los botones rojo/verde en función del estado de las luces en ese momento. Este proceso se realiza en un Hilo independiente para no bloquear la aplicación.

Ambos botones actúan de la misma manera. Levantan un Hilo en el dispositivo Android que, primero envía el comando "LEDS" o "TIRA" a la capa frontal para que actúe sobre el dispositivo domótico, después valida la trama obtenida en función del ack y la estructura. A continuación, recoge el estado para colorear los botones o imprimir el error devuelto por el backend.

También se puede mostrar el mensaje de error "Respuesta errónea con el servidor frontal" si no encuentra alguna de las etiquetas principales durante el *parseo* de la trama *(análisis de la trama)*, "Imposible conectar con servidor Frontal" si salta el TimeOut de 30 segundos de IP al abrir el socket o "Error conexión con servidor riego" si el servidor frontal ha tenido problemas para conectar con el servidor *backend*. Si se recibe ack="0", mostraría el mensaje de error gracias al método *Toast*.

Figura 33. Layout Servicio Acuario

7.1.2 Servicio Riego

El servicio de riego tiene la estructura de tramas ya descrita pero con alguna modificación:

→"<request>
<comando> **comando** </comando>
</request>".
← "<response>
<ack> **ack** </ack>
<estado> **estado** </estado>

</response>".

Las posibilidades dentro de esta trama son las siguientes:
Comandos:

- "IniciarZ1"
- "IniciarZ2"
- "Apagar Todo"
- "IniciarProgramacion"
- "EstadoRiego"
- "IniciarProgramacion<min>minutos</min>"

Estados:

- "Riego Apagado"
- "Lado Izquierdo riego encendido"
- "Lado derecho riego encendido"
- "Programación Lado derecho"
- "Programación Lado izquierdo"

ACK: "0" - "1".

Con la carga del Layout "Riego", representado en la figura 34, se arranca un Hilo que pide el estado de riego a la capa frontal. Este Hilo vive mientras estemos en el layout y, en el caso de detectar que se lanza un comando con una programación de Riego, se entra en un bucle consultando el estado para detectar el cambio de zona y el apagado de la programación. Con esto conseguimos proporcionar una lógica de presentación adecuada al estado del riego en cada momento coloreando los botones de Verde/Rojo de cada zona.

También se puede mostrar el mensaje de error "Respuesta errónea con el servidor frontal" si no encuentra alguna de las etiquetas principales durante el *parseo* de la trama, "Imposible conectar con servidor Frontal" si salta el TimeOut de 30 segundos de IP al abrir el socket o "Error conexión con servidor riego" si el servidor frontal ha tenido problemas para conectar con el servidor *backend*. Si se recibe ack="0", mostraría el mensaje de error gracias al método Toast.

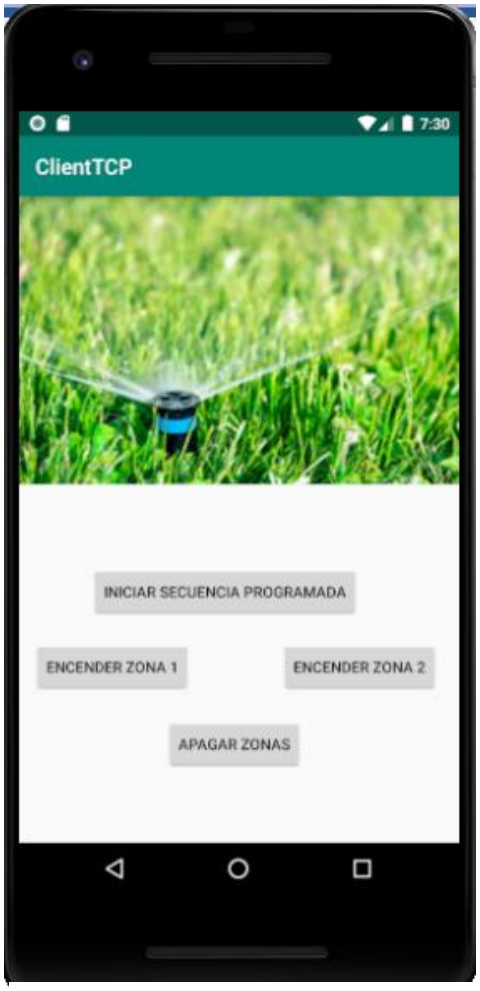

Figura 34. *Layout Servicio Acuario*

El botón EncenderZ1 lanza un Hilo independiente que manda el comando "IniciarZ1" al servidor Frontal TCP. En la respuesta se recibe de nuevo el estado para colorear los botones, o imprimir el error devuelto por el *backend*.

El botón EncenderZ2 funciona de forma análoga al botón anterior a excepción del comando utilizado "IniciarZ2".

El botón Iniciar Secuencia Programada lanza un Hilo independiente enviando el comando "IniciarProgramación" con el número de minutos introducido por el usuario o con 5 minutos cada lado por defecto y activa el hilo de consulta periódica de estado (si está desactivado), quedando en espera de la evolución de la programación de riego utilizando la misma lógica que en la carga del Layout.

7.1.3 Servicio Clima

La estructura de tramas del servicio Clima es la más personalizada:
→<request><comando> **comando**</comando></request>
←<response> <ack> **ack** </ack> <estado>
<leyenda> **leyenda** </leyenda>
<temperatura> **temperatura** </temperatura>
<humedad> **humedad** </humedad>
</estado> </response>

Las posibilidades dentro de esta trama son las siguientes:
Comandos:

- "Temp_hora"
- "Temp_dia"
- "Temp_mes"
- "Temp_ano"

Estados:

- "leyenda"-> Los índices de los minutos, horas, meses, años, solicitados (podría haber saltos)
- "temperatura"-> Los valores de temperatura solicitados divididos por comas en grados centígrados.
- "humedad"-> Los porcentajes de humedad solicitados divididos por comas

ACK: "0" - "1".

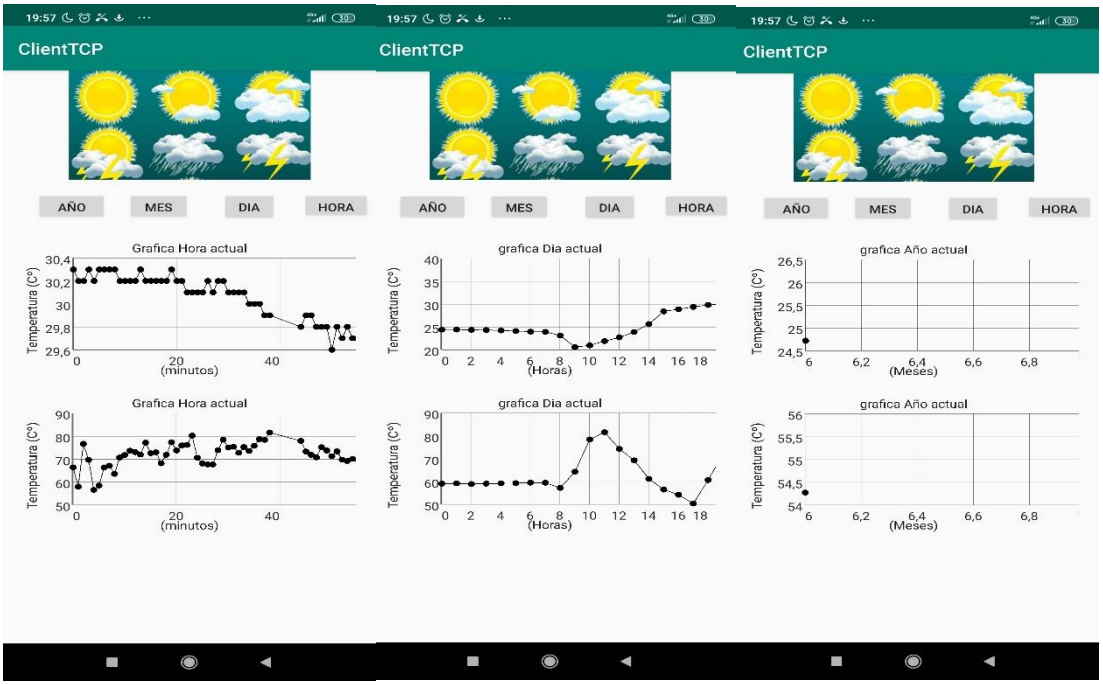

Figura 35. *Layout Servicio Clima*

Este servicio ofrece gráficas en tiempo real para el cliente tanto de humedad como de temperatura. Los datos disponibles dependen del tiempo que se tenga el servidor encendido. Como se puede ver en la Figura 35, en la gráfica anual solo tenemos registrado el mes de junio, ya que fue entonces cuando se programó el proceso de cálculo de medias para la visualización de las gráficas de la Figura 35.

El *layout* consiste en: 2 *GraphViews* (una por cada gráfica) y 4 botones (“años”, “meses”, ”días”, “horas”).

Para conseguir mostrar unas gráficas adecuadas nos hemos aprovechado de las librerías *LineGraph*, *GraphView*, *LabelRenderer* y *GraphViewRenderer* para conseguir adaptar cada gráfica.

7.1.4 Flujo App Android

En la figura 36 se muestra el flujo de la aplicación y todas las funcionalidades para abordar los 3 casos de uso propuestos.

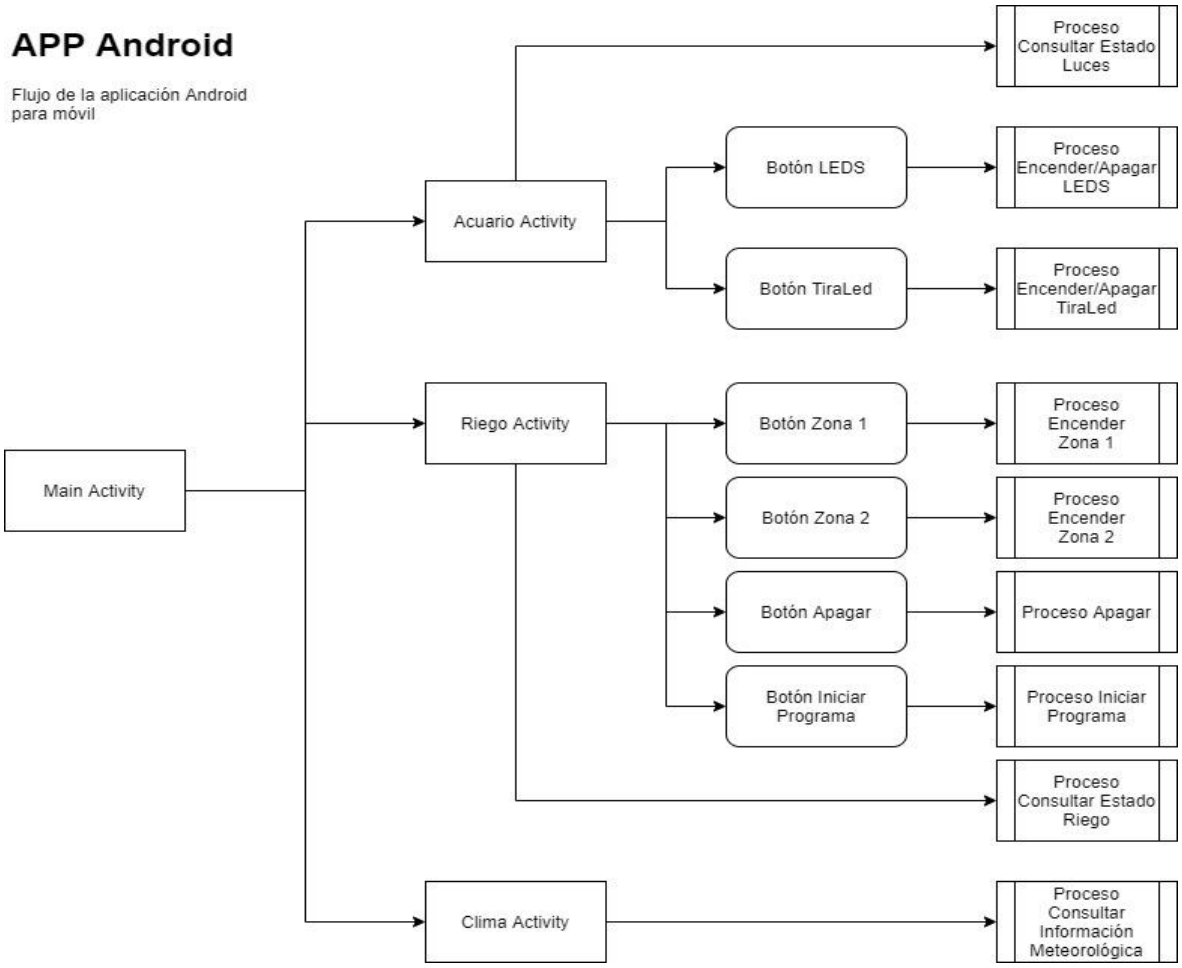

Figura 36. Flujo Aplicación Móvil Completa Draw.io

7.2 Cliente Web

Se ha habilitado también el acceso web mediante navegador utilizando el servidor frontal para servir las páginas HTML como podemos ver en la figura 37.

Al entrar en la página, el servidor frontal hará todas las peticiones a los distintos servidores de backend y, le mandará al cliente web que se está conectando un *String* que contiene la página WEB entera, con los estados de los dispositivos domóticos ya dibujados. Incluso se manda un Javascript que será renderizado por nuestro explorador para mostrar las gráficas de temperatura y humedad.

Al pulsar los distintos botones mostrados en la figura conseguiremos la misma funcionalidad explicada en la Aplicación Android en la interfaz TCP, ya que han sido programadas de la misma manera. La peculiaridad de este cliente es que es servido por el Servidor Frontal y lo único que hace es mandar comandos post a la dirección del frontal para que le actualice la página coloreando los botones en función de la respuesta.

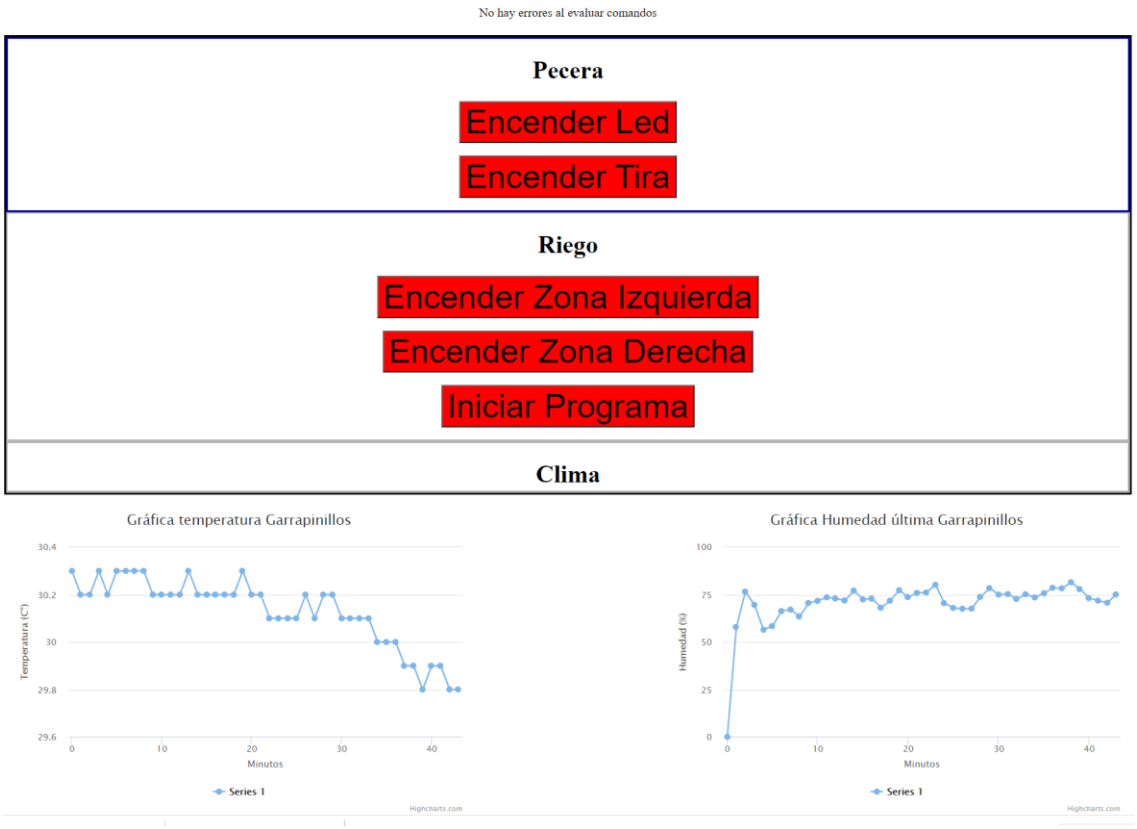

Figura 37. Imagen del servicio WEB Draw.io

8 Conclusiones

Se ha conseguido implementar con éxito una arquitectura con microcontroladores que nos permite manejar dispositivos en nuestro entorno doméstico y gobernarlos a través de una aplicación WEB y una aplicación móvil.

El trabajo realizado demuestra que, con un coste muy bajo y utilizando las actuales tecnologías de microcomputación, se pueden realizar implementaciones de servicios de alto nivel como es el caso de un servidor WEB y la utilización de dispositivos IoT.

Como muestra, el coste de los componentes con los que se ha llevado a cabo mi proyecto ha sido:

- 3 ESP32 dual-core 2,76 € *3 = 9,78 €
- 1 ESP8266-01S 0,99 €
- USB para ESP8266-01 0,71 €
- 6 relés de 5V 1,90 € las 6 unidades
- Un lector de tarjetas Sd 2 €
- Una tarjeta SD de 16Gb 3,81 €
- Display LCD 2,85 €
- Sensor de temperatura 2,79 €
- Estabilizador LM2596 0,52 €
- Componentes menores como resistencias, potenciómetros, transistores y estabilizador 5 € (estimación)

Total 30,35 €

Es un coste verdaderamente bajo para el servicio que se puede llegar a obtener con este tipo de tecnologías. Incrementar la funcionalidad añadiendo casos de uso es algo sencillo y con un coste verdaderamente despreciable.

Una aplicación muy importante para trabajar con IoT mediante microcontroladores estaría dirigida al mundo rural. Los cultivos, el cuidado del ganado, las explotaciones fotovoltaicas, etc, podrían ser observadas, monitorizadas y controladas con dispositivos de muy bajo coste que harían mucho más sencilla y productiva su explotación.
Existen muchos sensores que podrían aplicarse en este campo como los sensores de luminosidad, sensores de humedad en la tierra, el ya utilizado sensor de temperatura y humedad ambientales, sensores de temperatura del agua, así como las nuevas implementaciones sobre ESP32 con cámaras de video que

permitirán observar el estado de los cultivos, el tamaño de las frutas u hortalizas en tiempo real con un coste muy por debajo del que compañías multinacionales comerciales están reclamando para proporcionar un servicio mucho menos personalizado.

A nivel personal, hacer este trabajo de fin de grado me ha puesto en contacto con una tecnología que, aunque ya conocía, no había tenido la oportunidad de profundizar y verdaderamente me parece impresionante la capacidad informática de unos dispositivos de menos de 3 € que pueden competir con el servicio al que se dedican habitualmente computadoras sofisticadas que además se encuentran expuestas a los pagos de unas licencias a las multinacionales propietarias del software que impiden que se puedan utilizar masivamente.

Tengo la intención de continuar evolucionando mi proyecto para hacerlo más ergonómico y sofisticado. Esto me va a permitir seguir en contacto con la evolución de estos microcontroladores que día a día van sacando nuevos productos y nuevas mejores y mantenerme activo en los foros de aficionados y profesionales de este campo con los que ya he participado para elaborar este proyecto.

9 Bibliografía

[1] © 1995-2021 The Open Group [Official Website]
https://www.opengroup.org/archimate-forum/archimate-overview

[2] © Fritzing.org [Official Website]
https://fritzing.org/

[3] © 2021 Computer Hope [Website]
https://www.computerhope.com/jargon/d/drawio.html

[4] ©1999-2021 • No-IP.com [Service Web]
https://www.noip.com

[5] ESP32 Genuino. Documentación oficial fabricante. @2021 ESPRESSIF SYSTEMS (SHANGHAI) CO., LTD. All rights reserved.:
https://www.espressif.com/en/products/modules/esp32

[6] ESP32-WROOM-32D (la versión utilizada). Albert y Andreas, AZ-Delivery Vertriebs , © AZ-Delivery[Official Site]
https://www.az-delivery.de/es/products/esp32-developmentboard

[7] © Copyright 2016 - 2021, Espressif Systems (Shanghai) Co., Ltd [Official OS Documentation]
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html

[8] ESP826601-S. Documentación oficial fabricante. @2021 ESPRESSIF SYSTEMS (SHANGHAI) CO., LTD. All rights reserved. [Datasheet/ Official documentation]
https://www.espressif.com/en/products/modules/esp8266

[9] © AranaCorp 2020-2021 [Website]
https://naylampmechatronics.com/35-modulo-lector-de-memoria-sd-card.html

[10] Jose Luis, (5 mar 2018) Cambatronics Online Re: Tutorial DHT22 [video tutorial youtube]
https://www.youtube.com/watch?v=55C9Jwd1LDQ

[11] LM2596 SIMPLE SWITCHER® Power Converter 150-kHz 3-A Step-Down Voltage Regulator datasheet [Website]
https://www.ti.com/product/LM2596

[12] CircuitSchools, Staff. (September 2, 2020) Re: Interfacing 16X2 LCD Module with ESP32 with and without I2C[Web tutorial]
https://www.circuitschools.com/interfacing-16x2-lcd-module-with-esp32-with-and-without-i2c/