INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE COMPUTO

INTERNET DE LAS COSAS | EMBEDDED SYSTEMS

Reporte

“Monitoreo de estacionamiento”

Grupo: 6CM3

Integrantes

- García Escamilla Bryan Alexis
- Hernández García José Alberto
- Meléndez Macedonio Rodrigo
- Pérez Rosales Luis Eladio

Profesor

Alemán Arce Miguel Ángel

Fecha de entrega: 18 de diciembre de 2025

ÍNDICE

MONITOREO DE ESTACIONAMIENTO

Introducción

En la actualidad, las ciudades enfrentan un desafío constante relacionado con la movilidad y el acceso a espacios de estacionamiento. La alta concentración de vehículos, combinada con la falta de información en tiempo real sobre la disponibilidad de lugares, genera situaciones de estrés, pérdidas de tiempo y una mayor congestión vehicular. A esto se suman problemas secundarios como el incremento en el consumo de combustible y la contaminación ambiental, que afectan directamente la calidad de vida de las personas.

Ante este panorama, es necesario plantear soluciones innovadoras que permitan optimizar el uso de los espacios y mejorar la experiencia de los usuarios. La integración de tecnologías modernas en la vida cotidiana ha abierto paso al Internet de las Cosas (IoT), el cual permite conectar dispositivos físicos a plataformas digitales para monitorear, automatizar y controlar procesos de manera remota y eficiente.

Con base en esta visión, nuestro proyecto se centra en el desarrollo de un **sistema de identificación de espacios libres para estacionamiento**, con el cual se busca proporcionar información precisa y en tiempo real sobre la disponibilidad de lugares, así como gestionar de forma automática el acceso al estacionamiento. Este tipo de soluciones no solo benefician a los usuarios finales al reducir el tiempo de búsqueda de un lugar, sino que también contribuyen a una movilidad urbana más organizada y sostenible.

Descripción del proyecto

Monitoreo de estacionamiento con aplicación IoT.

Este proyecto consta de un sistema con tecnología IoT basado en ESP32. El sistema obtiene datos de la disponibilidad de espacios en un estacionamiento empleando sensores magnéticos los cuales son enviados a la plataforma Blynk, donde el usuario puede visualizarlos en tiempo real desde su celular mediante Internet. Además, el sistema controla automáticamente el acceso al estacionamiento usando seguidores de Línea TCRT5000 Óptico Infrarrojo para la entrada y salida de autos. Tiene como finalidad ofrecer una solución práctica y accesible a la búsqueda de estacionamientos disponibles.

Planteamiento del problema

Uno de los principales problemas a los que se enfrentan los conductores en zonas urbanas es la dificultad para encontrar estacionamiento disponible. Este inconveniente se traduce en recorridos prolongados, pérdida de tiempo y aumento en los niveles de tráfico en las calles cercanas. Además, la incertidumbre de no saber si un estacionamiento tiene lugares libres provoca que muchas personas eviten ciertas zonas, lo que afecta también a la dinámica comercial y social del área.

El hecho de no contar con un sistema que informe de manera clara y oportuna sobre los espacios disponibles provoca consecuencias directas como:

- Incremento en la congestión vial.
- Mayor consumo de combustible y emisiones contaminantes.
- Estrés en los conductores por la falta de organización.
- Menor eficiencia en el uso de los espacios de estacionamiento.

Si bien existen algunas soluciones en grandes ciudades, estas suelen ser costosas, poco accesibles o difíciles de implementar en espacios pequeños. Por ello, se requiere una alternativa que sea práctica, funcional y adaptable a distintos contextos.

El proyecto busca resolver este problema desarrollando una propuesta accesible, capaz de detectar en tiempo real los espacios disponibles y comunicar esta información a los usuarios de forma sencilla y confiable.

Objetivo general

Desarrollar un sistema inteligente de identificación de espacios libres para estacionamiento basado en tecnología IoT, que permita a los usuarios conocer en tiempo real la disponibilidad de lugares y gestione de manera automática el acceso, con el fin de optimizar la movilidad y mejorar la experiencia del conductor.

Objetivos específicos

- Analizar las principales problemáticas relacionadas con la búsqueda de estacionamiento en zonas urbanas.
- Implementar un sistema de detección que registre la ocupación o disponibilidad de cada espacio de estacionamiento.
- Desarrollar un mecanismo automatizado que controle el acceso al estacionamiento de acuerdo con la disponibilidad de lugares.
- Diseñar una aplicación móvil que permita a los usuarios visualizar en tiempo real la información del estacionamiento.
- Evaluar el funcionamiento del sistema en escenarios de prueba, asegurando su confiabilidad y facilidad de uso.
- Promover el uso de tecnologías IoT como herramienta para generar soluciones prácticas a problemas cotidianos.

Justificación

Actualmente aún persisten problemas al encontrar lugares disponibles en estacionamientos, afectando la dinámica comercial y social de zonas donde estos predominan, más que nada en lugares pequeños donde este problema se encuentra en mayor intensidad, derivando al aumento en los niveles de tráfico, pérdida de tiempo y recorridos prolongados. Por ello resulta necesario la creación de un sistema accesible y económico para optimizar la búsqueda de espacios disponibles. Nuestra propuesta ofrece una alternativa fácil de usar para

monitorear y controlar un estacionamiento mediante un sistema con tecnología IoT usando la plataforma Blynk desde un teléfono celular con acceso a Internet. El uso de la ESP32 permite tener un sistema económico, eficiente y con conexión a Internet integrada, con esto el usuario obtiene información en tiempo real sobre la disponibilidad de espacios en un estacionamiento, un control automático del acceso al estacionamiento evitando el tráfico, pero mejorando la movilidad y experiencia del conductor.

Materiales

Para la implementación del sistema propuesto se seleccionaron distintos materiales y componentes electrónicos que, en conjunto, permiten integrar la parte física de detección y control con la conectividad necesaria para el funcionamiento de nuestro sistema IoT. Cada elemento cumple un rol específico dentro de la arquitectura general, contribuyendo al objetivo de identificar y gestionar los espacios libres en un estacionamiento.

A continuación, se muestra una tabla con los dispositivos electrónicos utilizados para nuestro sistema IoT. Para cada uno se muestran sus características principales, cantidad y la liga a la página en donde se puede adquirir[1].

Dispositivo	Cantidad
ESP-WROOM-32	1
Sensor magnético KY-024	6
Servomotor SG90 RC 9g	1
Seguidor de Línea TCRT5000 Óptico Infrarrojo	2
Fuente de alimentación para protoboard MB-102	1

Tabla 1. Lista de dispositivos electrónicos.

A continuación, se especifica la funcionalidad en el sistema de cada uno de los dispositivos listados anteriormente.

1. Microcontrolador ESP-WROOM-32.

La ESP32 representa el cerebro de todo el sistema. Su función principal es leer la información de los sensores en las plazas de aparcamiento, controlar internamente la entrada y salida de vehículos en el estacionamiento, establecer la comunicación mediante conexión WiFi con la plataforma **Blynk**, así como recibir datos desde esta. Gracias a su capacidad de manejar múltiples entradas y salidas digitales, así como su compatibilidad con servicios en la nube, la ESP32 es el puente entre el hardware y la plataforma que consultará el usuario.

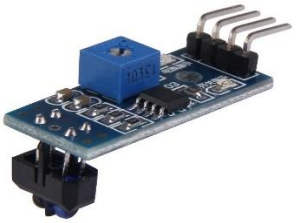
[1] Cada nombre del dispositivo está ligado a la página donde puede adquirirlo y en donde podrá tener acceso a la ficha técnica correspondiente.

2. **Sensores magnéticos KY-024.**

Para determinar los lugares disponibles y libres del estacionamiento se emplean seis sensores magnéticos KY-024 (6 espacios disponibles) instalados debajo de cada una de las plazas de aparcamiento. Para esto el sensor debe de detectar un campo magnético para que en su salida digital y analógica se dé un cambio en su valor, para ello cada uno de los vehículos requiere tener un imán por debajo para que se genere dicho campo. Entonces de esta forma el ESP32 podrá saber si una plaza está disponible o no.

3. **Seguidor de línea TCRT5000**

Estos sensores colocados en la salida y entrada del estacionamiento se encargarán de detectar cuando un vehículo quiera entrar o salir del estacionamiento, dando en su salida digital un estado en alto, el cual la ESP32 leerá e interpretara y ejecutará la acción correspondiente. Mientas haya lugar en el estacionamiento se seguirá leyendo la salida del sensor en de la entrada, de lo contrario este se ignorará, mientras que el sensor de la salida, su valor siempre se leerá. Para cualquiera de los dos sensores, estos necesitan que el vehículo permanezca tres segundos para que sea detectado.



4. **Micro servomotor.**

El micro servomotor se encargará de tomar el papel de la barra automática del estacionamiento que permitirá el acceso y salida de los vehículos dependiendo de lo que la

ESP32 lea de los sensores TCRT500 y las acciones que tome de acuerdo con ello. Este se elevará para permitir la entrada y salida de vehículos, y después de 3 segundo volverá a su posición normal.

5. Fuente de alimentación MB-102

Para alimentar todo el circuito del estacionamiento se usará al módulo MB-102, ya que este nos proporciona 5V y 3.3V de forma separada. Para alimentar los sensores KY-024 y TCRT5000 se usarán los 3.3V mientras que para el micro servo y la ESP32 los 5V.

6. Aplicación y software.

Finalmente, el ecosistema del proyecto se complementa con la plataforma **Blynk**, la cual cumple la función de interfaz de usuario. Desde esta aplicación se podrán consultar en tiempo real las plazas libres y ocupadas del estacionamiento, así como los vehículos dentro en ese momento, también se podrá visualizar el estado general del sistema, mensajes cuando un carro entra y sale, y como extra un switch para poder bloquear el acceso al estacionamiento permitiendo únicamente la salida de vehículos.

En conjunto, la selección de estos materiales responde a la necesidad de contar con un sistema IoT accesible, adaptable y funcional, capaz de integrar la detección física, el procesamiento digital y la interacción remota en una sola solución tecnológica.

Información sobre los pines de la ESP32 (versión con 30 GPIOS)

- **Distribución de los pines**

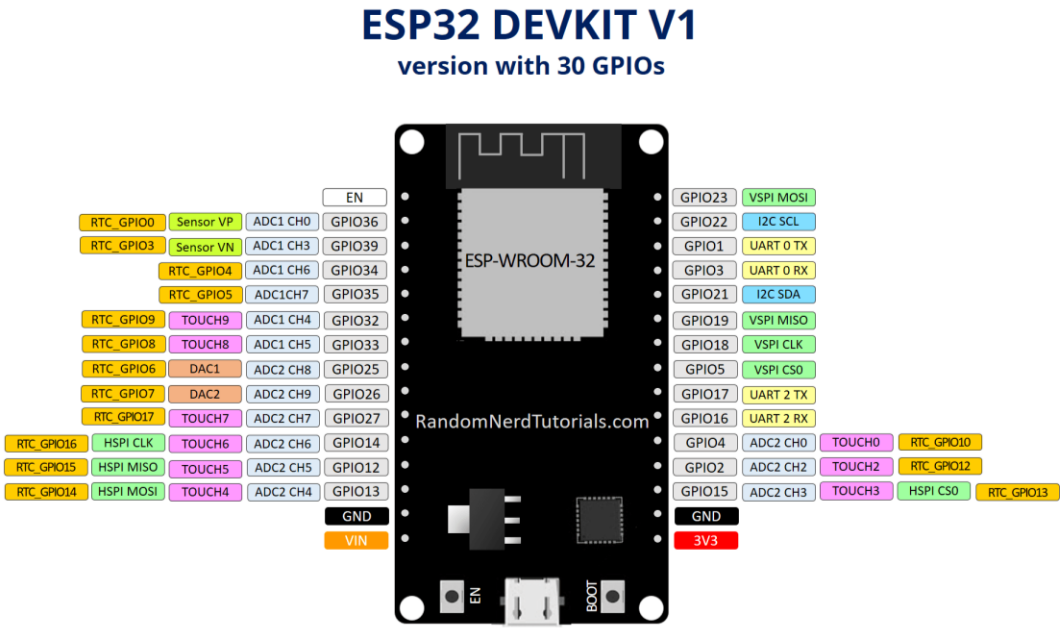


Imagen 1. Distribución de los pines para la versión con 30 GPIOs.

- **Consideraciones sobre los pines**

A pesar de que esta versión tenga 30 pines, eso no significa que todos puedan usarse, y esto tiene un porque:

1. **Pines solo de entrada**

Los siguientes pines solo sirven para entradas analógicas o digitales:

- GPIO 34-36
- GPIO 39

2. **Pines reservados para el arranque (strapping pins)**

Durante el arranque, la ESP32 lee estos pines para decidir como iniciar. Si se le conecta algo que cambie su voltaje, puede que la ESP32 no arranque, entre en modo bootloader o bloquearse, tales como:

- GPIO 0
- GPIO 2
- GPIO 4
- GPIO 5
- GPIO 12
- GPIO 15

3. Pines dedicados al Flash y PSRAM

La memoria Flash está conectada por SPI interno a estos pines:

- GPIO 6-11

4. Pines que la placa ya utiliza correctamente

Esto depende de la placa, pero normalmente se usan para lo siguiente:

- GPIO 1 (TX0): Usado para manejar por Serial
- GPIO 3 (RX0): Usado para programación y consola
- GPIO 6-11: Memoria flash (prohibidos)
- GPIO 5: A veces conectado al circuito de boot
- GPIO 15: Usado en arranque

Estos se pueden usar, pero hay que tener precauciones.

5. Pines sensibles o no recomendados para cargas

Algunos pines son muy sensibles y causan problemas si les conectas sensores o cargas:

- GPIO 12: Cambiar su nivel rompe el voltaje del flash
- GPIO 2: Debe permanecer en HIGH para arrancar bien
- GPIO 15: Debe estar en LOW en arranque

6. Pines ADC (lecturas analógicas) que se ven interrumpidas por el WiFi

Cuando el WiFi está activo, los ADC2 de la ESP32 pierden precisión o dejan de funcionar bien:

- GPIO 0
- GPIO 2
- GPIO 4
- GPIO 12-15
- GPIO 25-27

En cambio, los pines ADC1 no se ven afectados por el WiFi:

- GPIO 32-36
- GPIO 39

7. PWM puede fallar en algunos pines durante WiFi

Porque el WiFi usa interrupciones de alta prioridad, el PWM puede generar jitter, perder precisión y vibrar en servos, por lo que los pines más seguros para PWM con WiFi activo son:

- GPIO 4
- GPIO 16-19
- GPIO 21-23
- GPIO 25-27

Circuito del proyecto

Una vez tomado en cuenta la información sobre los pines de la ESP32 y considerando que se usará PWM para mover el micro servo y WiFi para la conexión con Blynk. Los pines elegidos para cada uno de los dispositivos del proyecto son los siguientes:

GPIO	Función	Dispositivo
21	Entrada digital	TCRT5000
22	Entrada digital	TCRT5000
32	Entrada digital	KY-024
33	Entrada digital	KY-024
23	Entrada digital	KY-024
19	Entrada digital	KY-024
18	Entrada digital	KY-024
17	PWM	Micro servo

Tabla 2. Pines usados para el circuito del proyecto.

Por tanto, el circuito resultante es el siguiente:

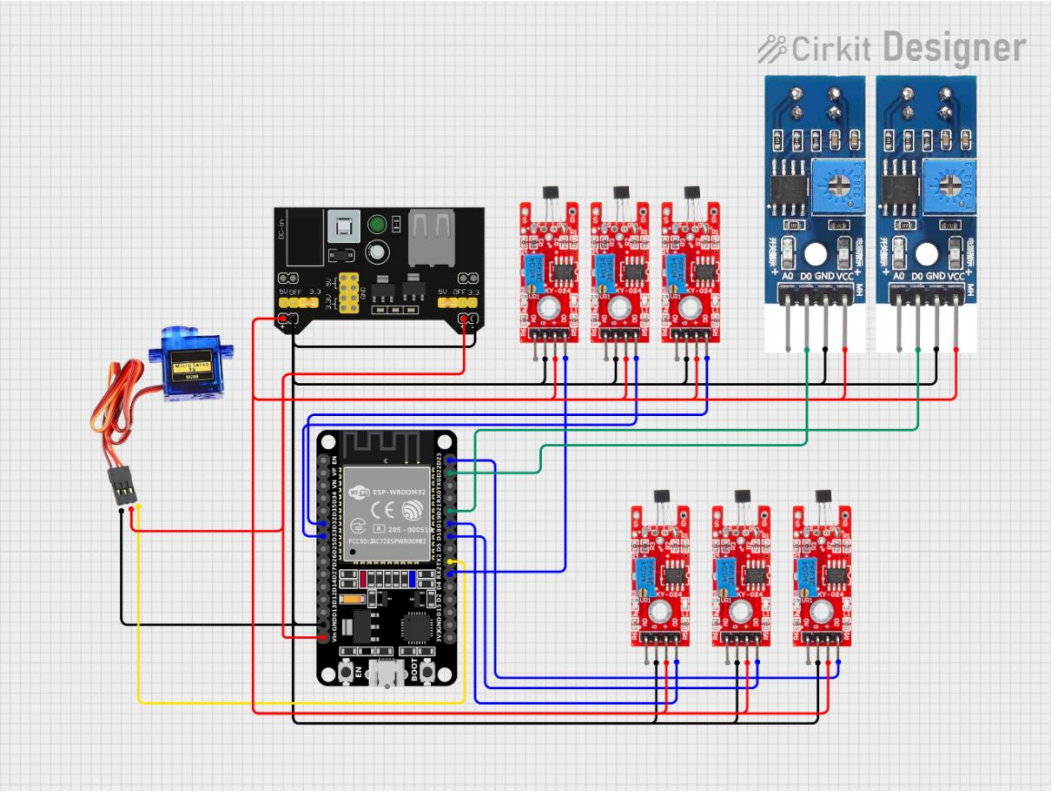


Imagen 2. Circuito del proyecto.

Se vuelve a recalcar que, para alimentar el circuito se usan dos fuentes, una de 3.3V y otra de 5V y cabe destacar que la ESP32 funciona con 3.3V, por lo que no se pueden ingresar 5V en ninguno de sus pines, de lo contrario esta podría fallar y quedar inutilizable, por lo mismo los sensores se alimentan con 3.3V a pesar de permitir también 5V, esto se debe a que se usarán las salidas digitales de ambos sensores (TCRT5000 y KY-024) y si estos son alimentados con 5V entonces la salida digital en estado alto se acerca a este voltaje.

La ESP32 cuenta con el pin 3.3V que proporciona dicho voltaje pero también se puede usar para alimentarla, sin embargo, se requiere que los 3.3V sean estables y provengan de un regulador de excelente calidad, de lo contrario se pueden presentar problemas en el funcionamiento de la ESP32, es por esto por lo que se usa el pin VIN, pues soporta voltajes desde 5V-12V, pero lo recomendable es usar 5V para que el regulador integrado baje el voltaje a 3.3V y no disipe mucho calor con voltajes mayores.

Instalación del controlador CP210x

Para comenzar a utilizar la ESP32 y programarla desde la PC o Laptop se debe de instalar su controlador, para este caso la placa incorpora el chip CP2102 el cual se encarga de programar y establecer comunicación USB-Serial.

- **CP210x Controlador universal**[2]

Ya instalado el controlador ya se puede conectar la placa a la PC. Para comprobar que se instaló correctamente el controlador y le asigno un puerto COM a la placa, sigue los siguientes pasos:

1. Abrir **“Administrador de dispositivos”**
2. Conectar la placa a la PC
3. Dar clic en **“Puertos (COM, LPT)”**
4. El administrador se actualiza y debe reconocer la placa, usualmente aparece como **“Silicon Labs CP210x USB to UART Bridge (COMx)”**

[2] Instalar el controlador de acuerdo con el sistema operativo de su PC o Laptop. En caso de haber instalado anteriormente no instalar el controlador.

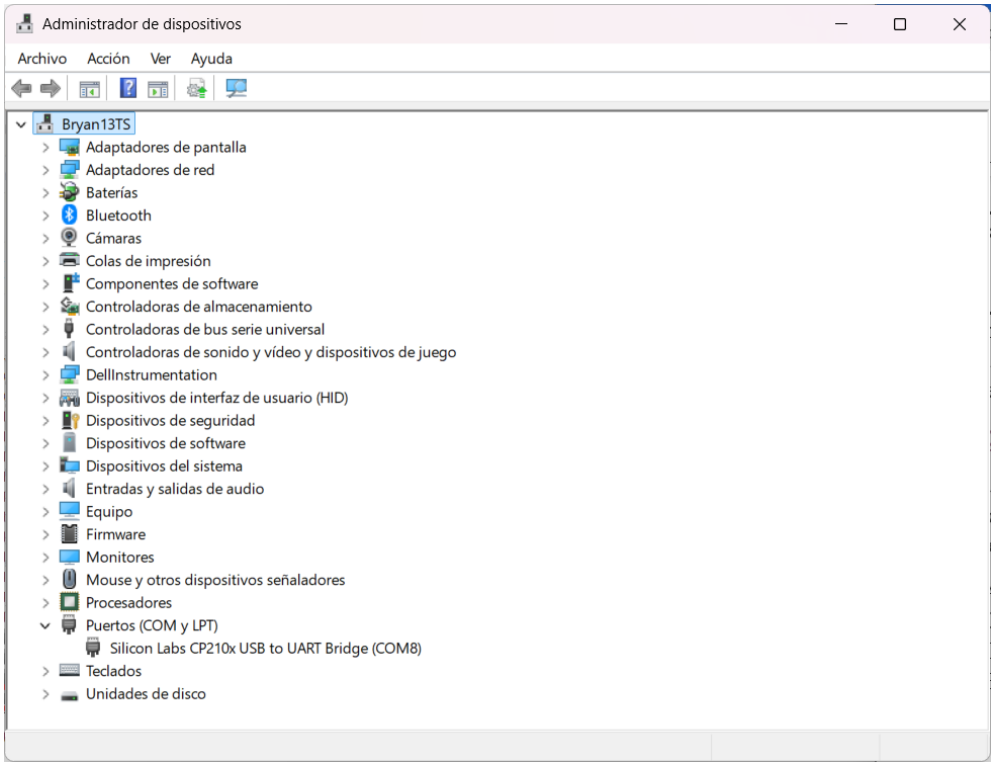


Imagen 3. Reconocimiento de la placa y asignación de puerto COM.

Instalación del soporte Arduino-ESP32 desde Arduino IDE

Para instalar el soporte Arduino-ESP32 desde Arduino IDE primero se requiere haber instalado este último (en coso de ya tenerlo instalado, saltarse esta parte). Para ello nos dirigimos a la siguiente página **Arduino IDE**. Una vez ahí se mostrará algo como lo siguiente:

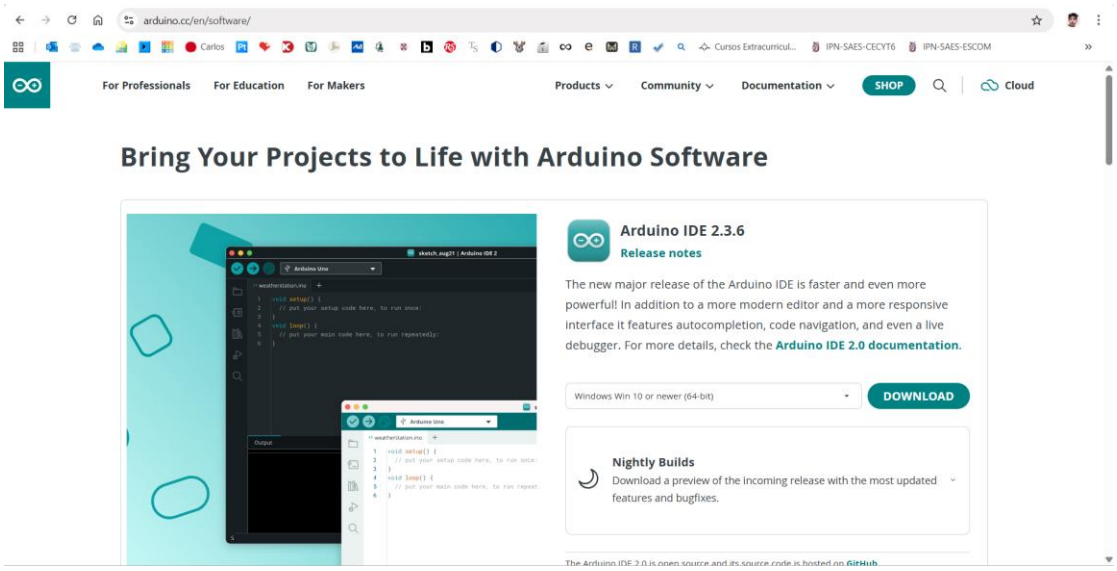


Imagen 4. Página inicial para la descarga de Arduino IDE.

En la página aparecerá inmediatamente el botón para descargar el IDE y a un lado el sistema operativo para el que se descargará:

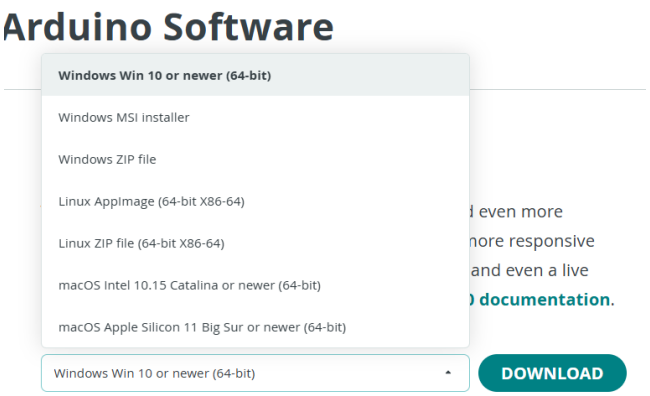


Imagen 5. Opciones de sistemas operativos.

En nuestro caso estaremos usando **Windows 11**, por lo que elegimos la opción **Windows Win 10 or newer (64-bit)**.

Una vez instalado Arduino IDE, vera la siguiente interfaz, en donde se puede empezar a escribir el código fuente del proyecto:

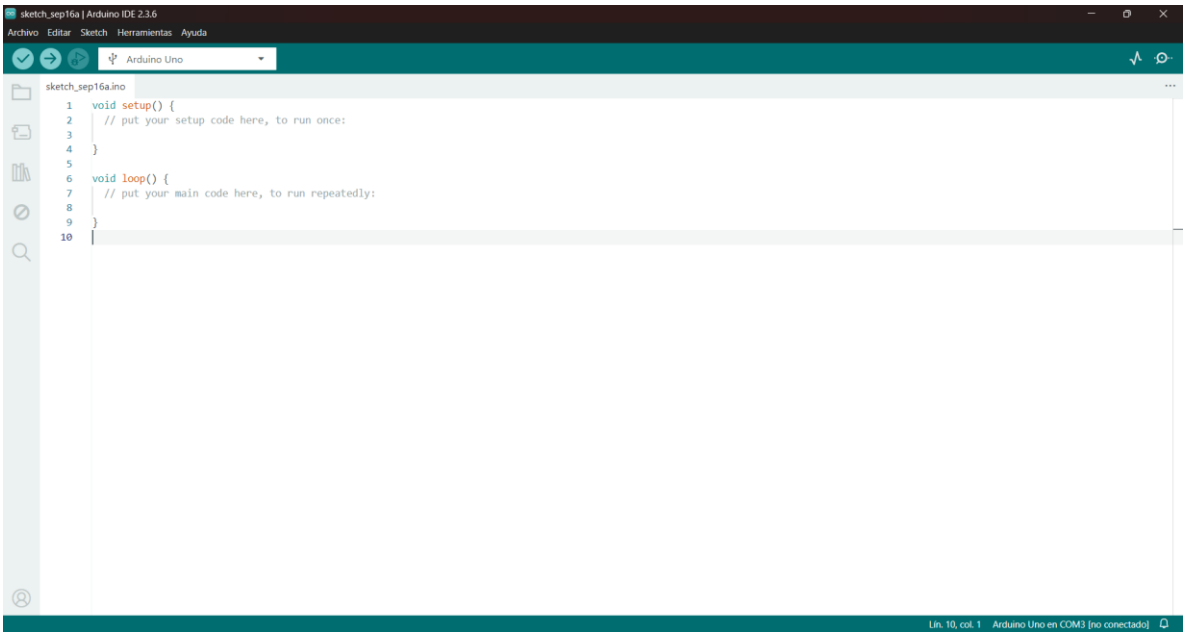


Imagen 6. Interfaz principal de Arduino IDE.

Una vez instalado Arduino IDE, nos dirigimos a la siguiente página de **Espressif Systems** para instalar el soporte **Arduino-ESP32**, en la que se nos mostrara lo siguiente:

Imagen 7. Guía para la instalación del soporte Arduino-ESP32.

En esta página se mostrará una guía de como instalar el soporte Arduino-ESP32. Se nos muestran varias opciones de instalación, en nuestro caso haremos la **instalación mediante Arduino IDE**. Para ello nos dirigimos en el Arduino IDE nos dirigimos a la pestaña **Achivo/Preferencias** en la parte superior izquierda del IDE.

Se mostrará una ventana como la siguiente:

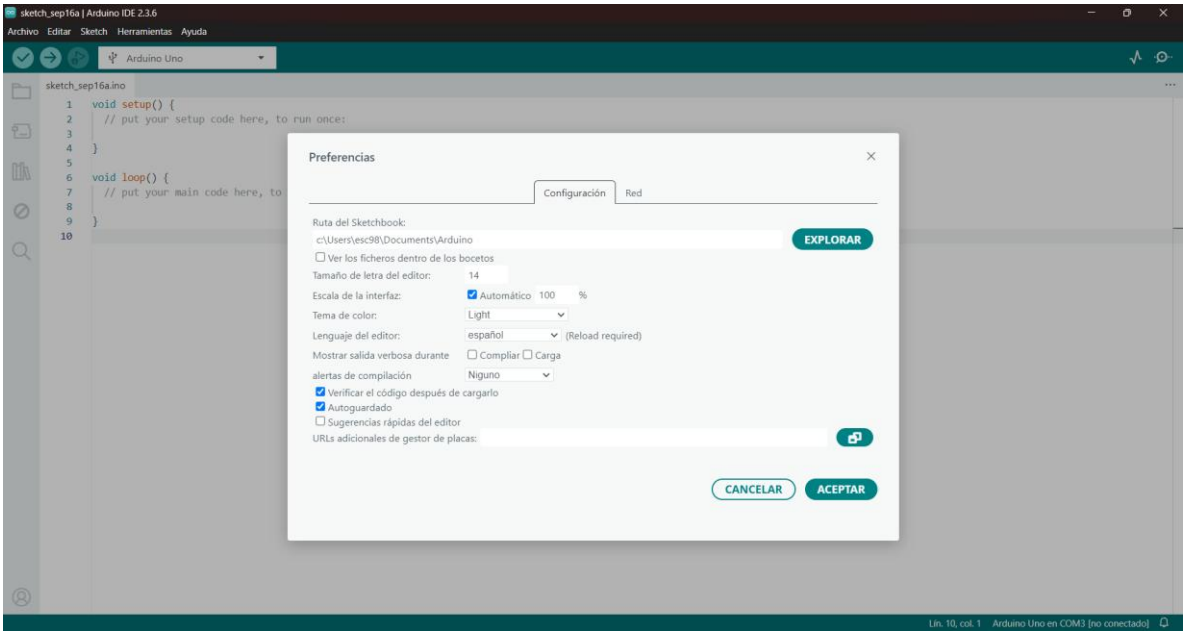


Imagen 8. Ventana Preferencias.

En esta ventana seleccionamos **URLs adicionales de gestor de placas**, esto abrirá una segunda ventana como la siguiente:

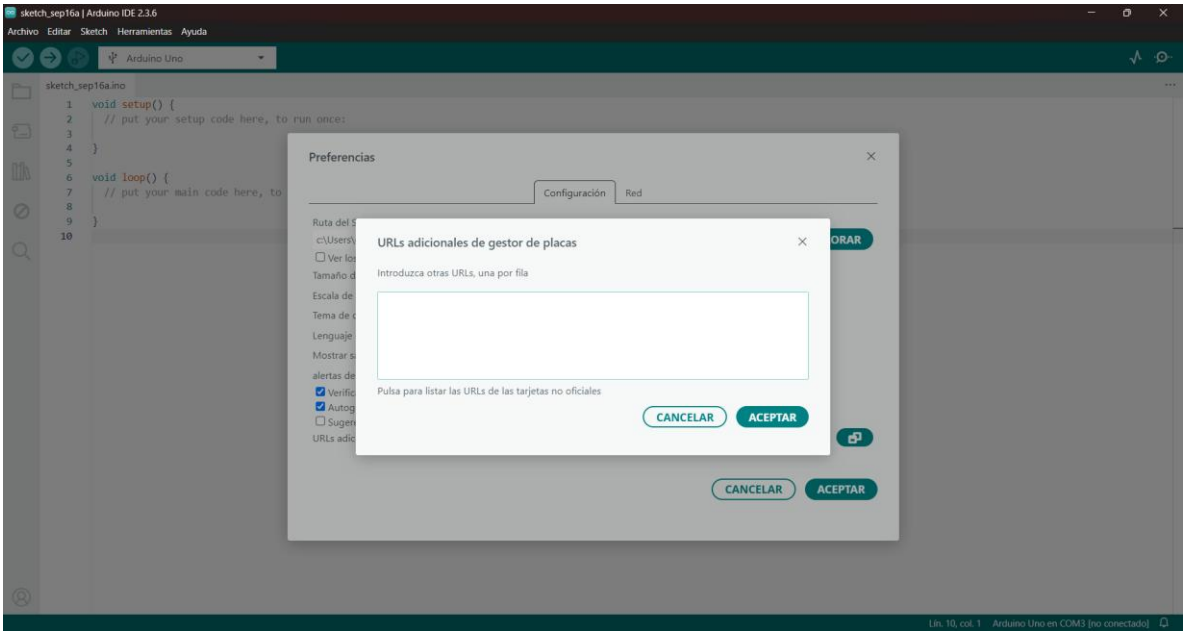


Imagen 9. Ventana URLs adicionales de gestor de placas.

Volvemos a la página de Espressif Systems y buscamos el siguiente enlace:

- Enlace de versión estable:

```
https://espressif.github.io/arduino-esp32/package_esp32_index.json
```

Imagen 10. Enlace para la versión estable de Arduino-ESP32.

Copiamos el enlace y lo pegamos en la ventana actual del Arduino IDE y damos aceptar:

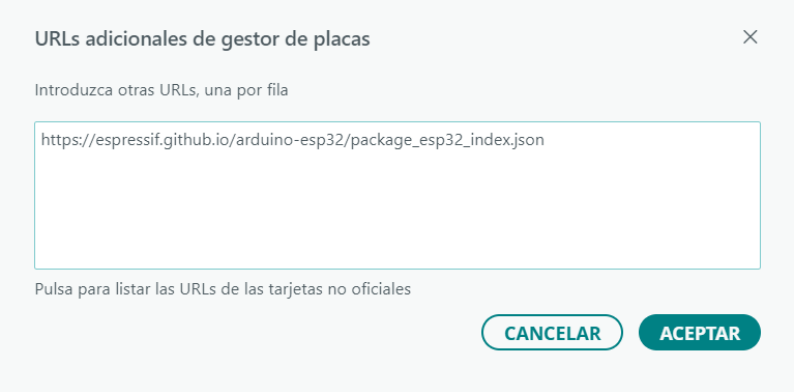


Imagen 11. Enlace copiado en la ventana URLs adicionales de gestor de placas.

Ahora tenemos que abrir el **Gestor de Placas**, para ello nos dirigimos a la pestaña **Herramientas/Placa/Gestor de placas** en la parte superior izquierda del IDE o usando el atajo Ctrl + Mayús + B y del lado izquierdo del IDE se mostrara un buscador en donde escribiremos **esp32**.

Se nos mostraran dos opciones de soporte, una de Arduino y una de Espressif Systems:

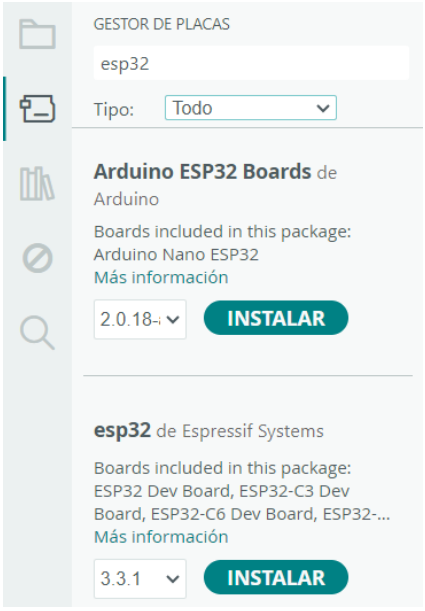


Imagen 12. Búsqueda del soporte para Arduino-ESP32.

Instalamos la de Espressif Systems y reiniciamos el IDE. Una vez instalados nos dirigimos a la pestaña **Herramientas/Placa**, y vemos que se agregó la placa **esp32**. Si colocamos el cursor sobre la placa se desplegarán todas las tarjetas basadas en el ESP32.

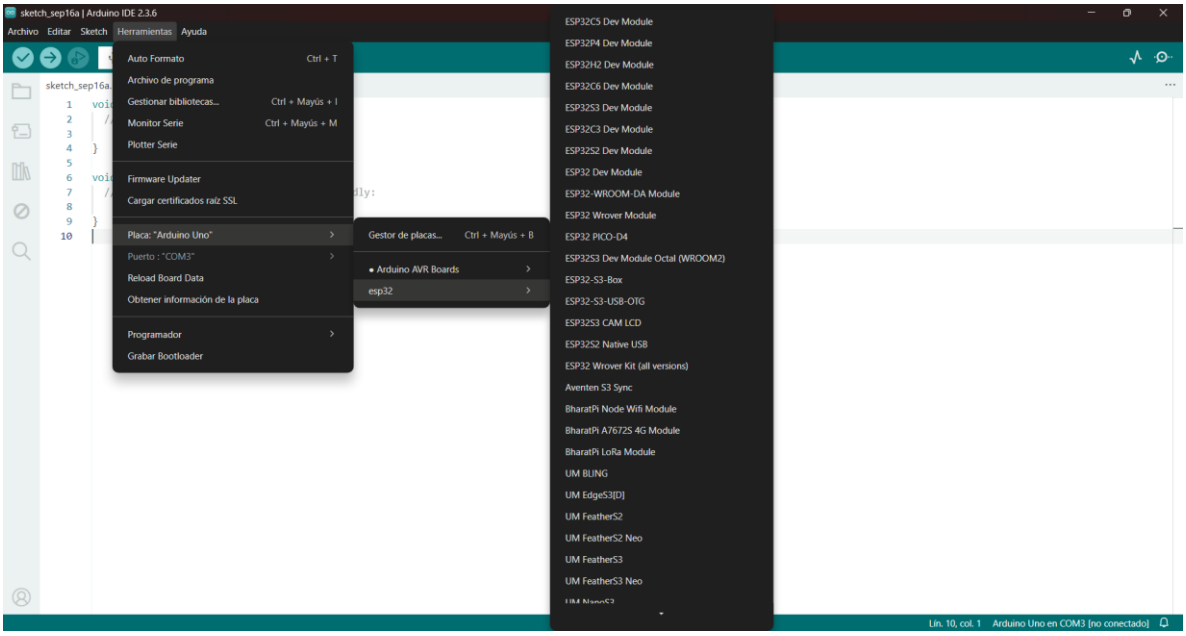


Imagen 13. Verificación de la instalación del soporte Arduino-ESP32.

Y con esto verificamos la correcta instalación del soporte Arduino-ESP32.

Instalación de las bibliotecas necesarias para el proyecto

- **ESP32Servo**

Anteriormente se mencionó que se va a ocupar un micro servomotor, sin embargo, para ello se requiere de una biblioteca para poder controlar el servo, pero hay un detalle, no se puede usar la biblioteca estándar de Arduino **Servo.h**, puesto que esta fue diseñada para Arduino UNO, Arduino Mega, ATmega328/ATmega2560 y porque estos tienen un sistema de temporizadores (Timers) muy diferentes al de la ESP32, por lo que la biblioteca **ESP32Servo.h** es la opción correcta, pues fue modificada para usar los PWM internos (LEDC) de la ESP32 y también porque es 100% compatible con la arquitectura ESP32.

Para descargar está biblioteca hay dos opciones, hacerlo desde el mismo IDE o desde la página oficial de GitHub, en esta ocasión se explicarán ambas formas.

1. **Descargar desde Arduino IDE**

Es la forma más sencilla para descargar la librería, para ello en el gestor de bibliotecas del IDE buscar **ESP32Servo**. Se mostrará el siguiente resultado:

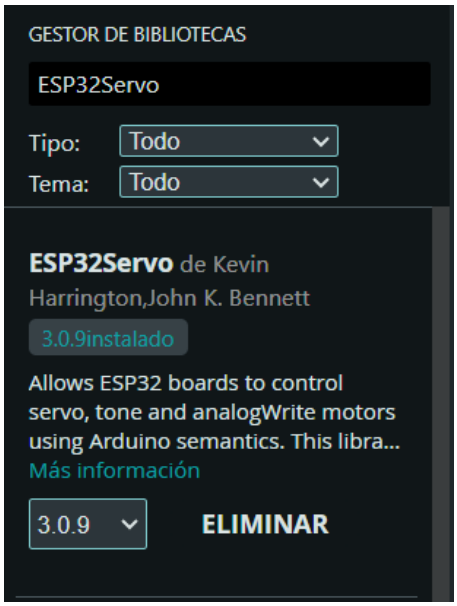


Imagen 14. Biblioteca ESP32Servo desde el gestor de bibliotecas.

Simplemente se le da clic en Instalar y listo ahora se puede usar en cualquier programa.
Para esto se importa como

```
#include <ESP32Servo.h>
```

2. **Descargar desde GitHub**

Aunque para esta forma hay que hacer unos pasos extra, también resulta necesario saber cómo instalar bibliotecas así pues existen algunas que no se encuentran desde el gestor de bibliotecas de Arduino IDE.

Para descargar la biblioteca ESP32Servo nos dirigimos al repositorio de GitHub ESP32Servo, se mostrará lo siguiente:

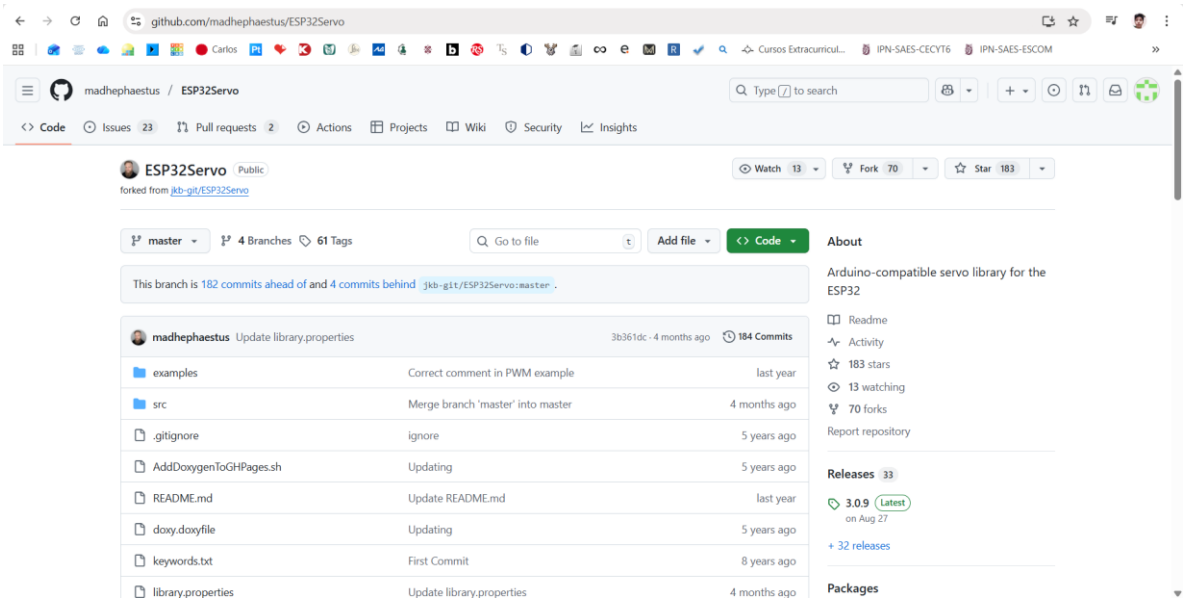


Imagen 15. Repositorio de GitHub de **ESP32Servo**.

Lo siguiente es hacer clic al botón verde el cual abrirá la siguiente pestaña. Aquí se descargan la biblioteca como ZIP.

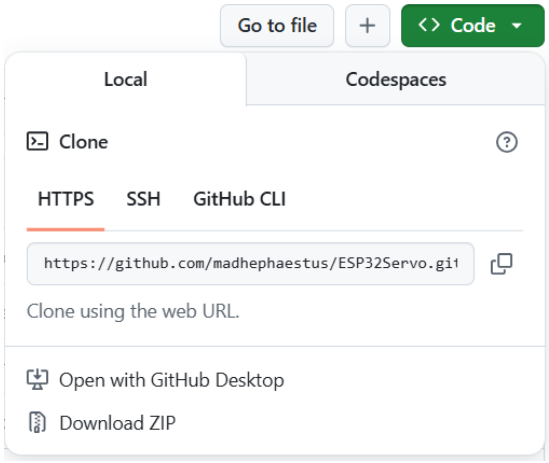


Imagen 16. Descargar biblioteca como ZIP.

Después abrir Arduino IDE, y en la parte superior izquierda seleccionar Sketch/Incluir biblioteca/Añadir biblioteca ZIP, esto abrirá una ventana en donde se deberá seleccionar el ZIP descargado e inmediatamente comenzará la instalación, una vez terminada, el propio IDE lo indicará.

- **Blynk**

De igual manera se mencionó que para la interfaz de usuario se va a usar la plataforma Blynk, que va a recibir información desde la ESP32 y enviar información a esta. Para establecer la comunicación con dicha plataforma se requiere la biblioteca **Blynk**, que será la encargada de conectar la ESP32 a Internet (usando WiFi), conectar la ESP32 con los servidores de Blynk, recibir datos desde la plataforma, enviar datos a la plataforma, mantener viva la conexión usando el protocolo propio de Blynk (basado en TCP) con keep-alive, reconexión automática, entre otros.

1. **Descargar desde Arduino IDE**

Al igual que con la biblioteca ESP32Servo, Blynk también se puede instalar desde el IDE, por lo que en el gestor de bibliotecas se busca **Blynk**. Se mostrará lo siguiente:

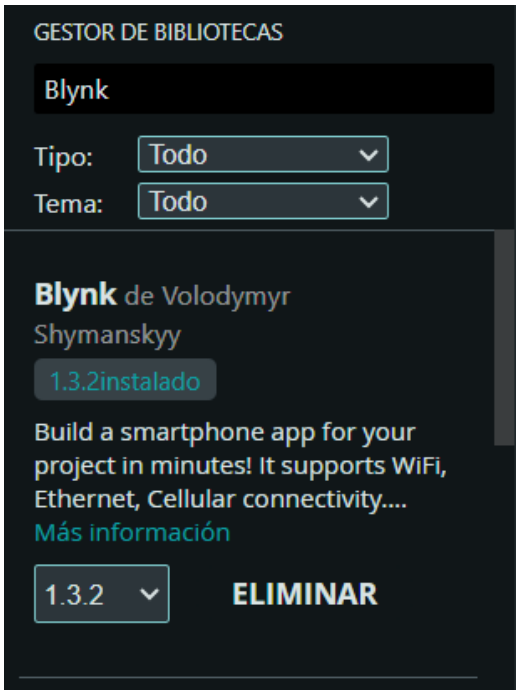


Imagen 17. Biblioteca Blynk desde el gestor de bibliotecas.

Simplemente se le da clic en Instalar y listo ahora se puede usar la biblioteca para la ESP32. Para esto se importa como:

```
#include <BlynkSimpleEsp32.h>
```

2. **Descargar desde GitHub**

Nos dirigimos al repositorio de GitHub **blynk-library** donde se mostrará lo siguiente:

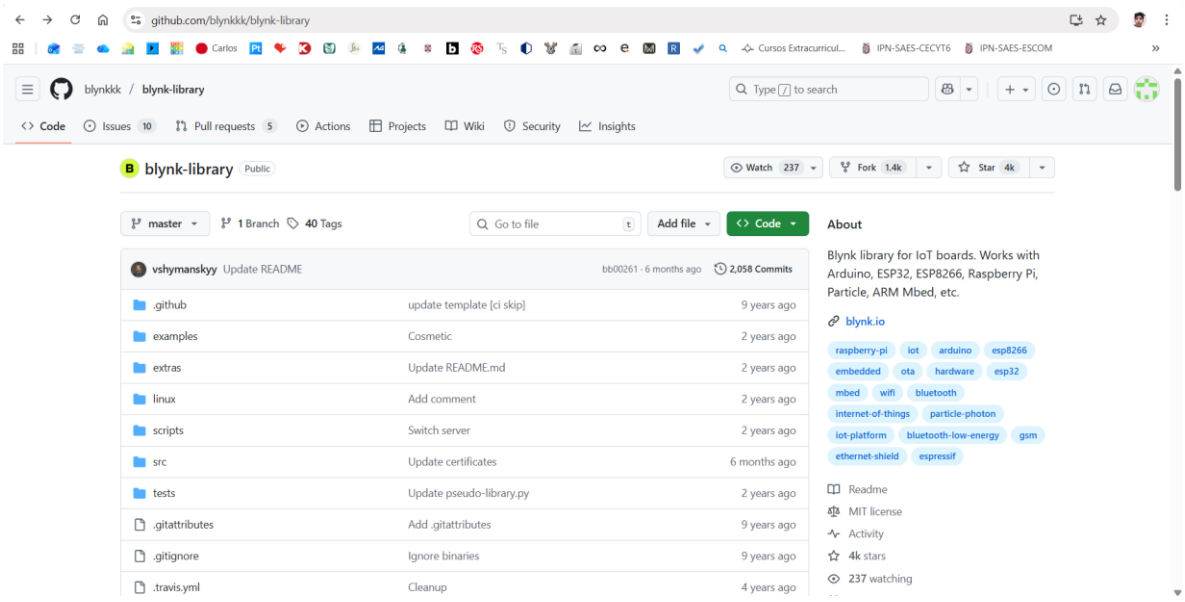


Imagen 18. Repositorio de GitHub de **blynk-library**.

Y al igual que con la anterior, lo siguiente es hacer clic al botón verde, se abrirá la siguiente pestaña. Aquí se descargan la biblioteca como ZIP.

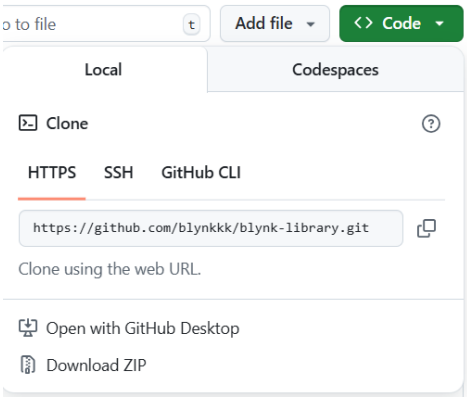


Imagen 19. Descargar biblioteca como ZIP.

Después abrir Arduino IDE, y en la parte superior izquierda seleccionar Sketch/Incluir biblioteca/Añadir biblioteca ZIP, esto abrirá una ventana en donde se deberá seleccionar el ZIP descargado e inmediatamente comenzará la instalación, una vez terminada, el propio IDE lo indicará.

Creación de la interfaz en Blynk (versión web)

Para la creación de la interfaz donde el usuario podrá ver que plazas del estacionamiento están disponibles y cuales no, mensajes de la salida/entrada de vehículos, el estado del sistema actual y un switch para bloquear/desbloquear la entrada de vehículos, es necesario contar con una cuenta en la plataforma Blynk.

Una vez creada la cuenta, podrá ver la página principal de Blynk.

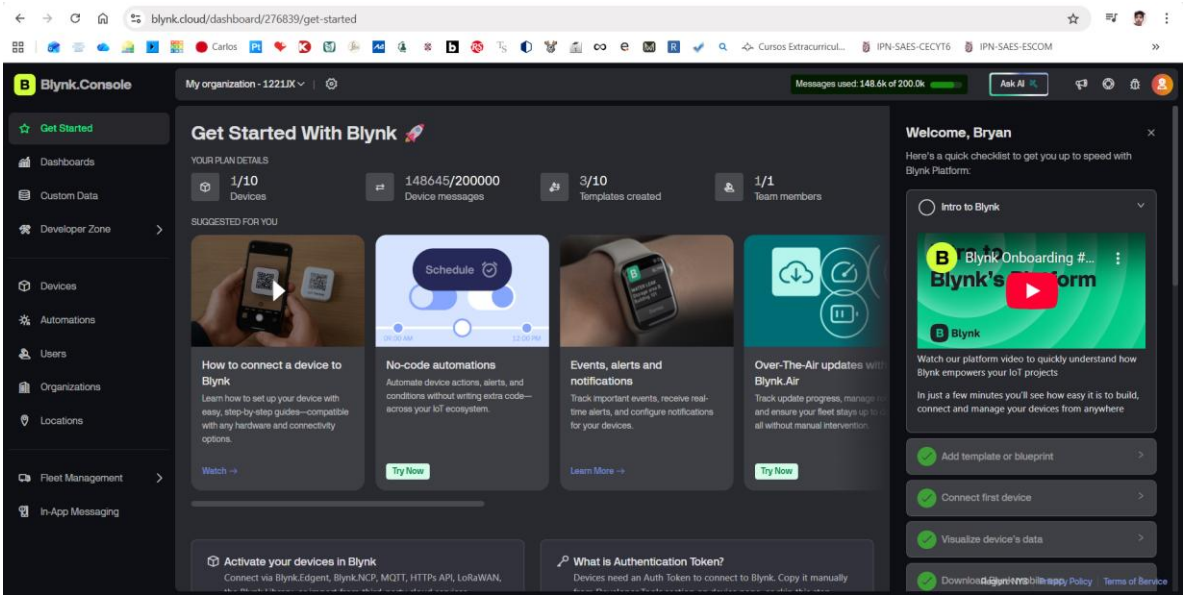


Imagen 20. Página principal de Blynk.

Lo que sigue a continuación son los pasos para crear dicha interfaz para el usuario.

1. Configurar una Plantilla

Del lado izquierdo de la página se observa un menú con diferentes secciones de la plataforma, la que nos importa por el momento es la **Zona de desarrollador (Developer Zone)**, ya que en esta podremos configurar la plantilla para la interfaz.

Por defecto en esta sección se nos mostrará la sección **Plantillas (Templates)** para empezar a configurar la plantilla.

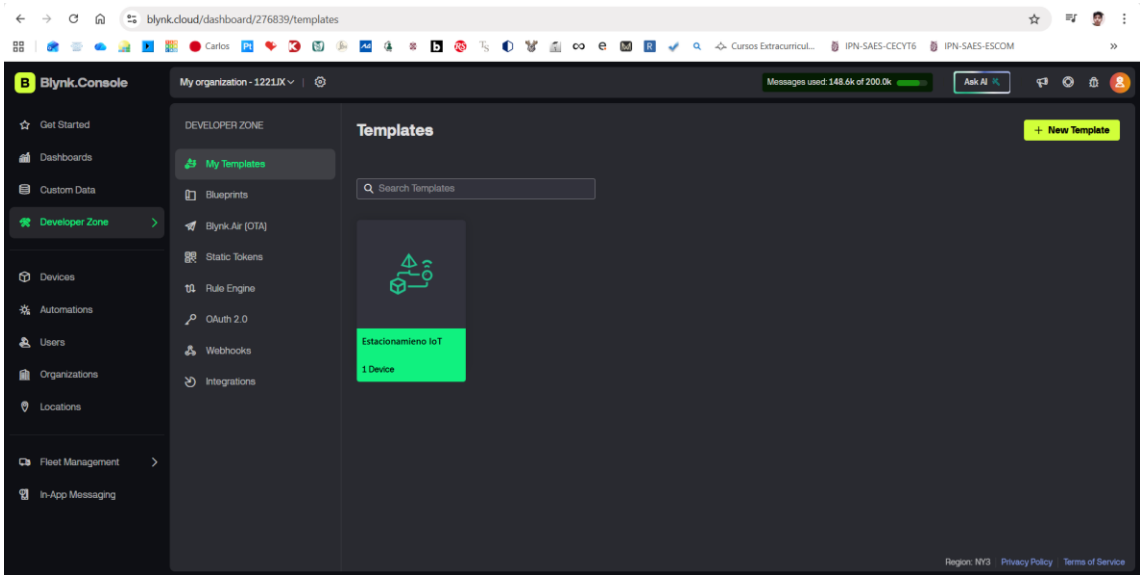


Imagen 21. Sección para configurar plantillas.

De clic al botón en la parte superior derecha **Nueva Plantilla** (**New Template)** para empezar con la configuración. Se abrirá una ventana en donde escribiremos el nombre de la plantilla, en el hardware seleccionaremos la **ESP32**, en el tipo de conexión **WiFi** y escribiremos una descripción breve de la plantilla.

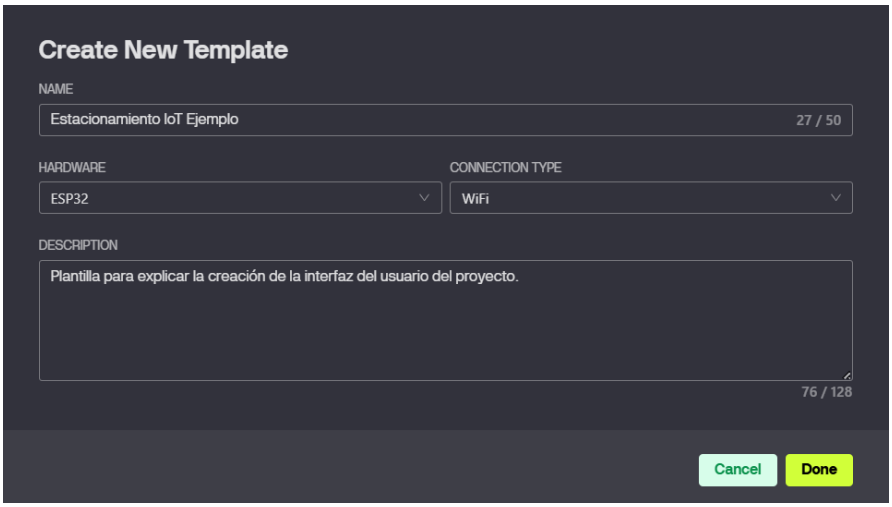


Imagen 22. Configuración de la plantilla.

2. Configurar los Flujos de Datos

Una vez configurada la plantilla, lo siguiente será configurar los **Flujos de Datos (Datastreams)**, canales que se utilizaremos para enviar datos entre la ESP32 y BlynK.Cloud.

Se usarán también para recibir información desde la ESP32 sobre que plazas de aparcamiento están disponibles y ocupadas, mensajes de la salida/entrada de vehículos, información sobre el estado del estacionamiento actual y para enviar datos desde Blynk mediante un widget de tipo switch para bloquear/desbloquear la entrada de vehículos al estacionamiento.

Para ello damos clic en el botón **Nuevo Flujo de Datos** (**New Datastreams)** en la esquina superior derecha y seleccionamos **Pin virtual (Virtual Pin)**.

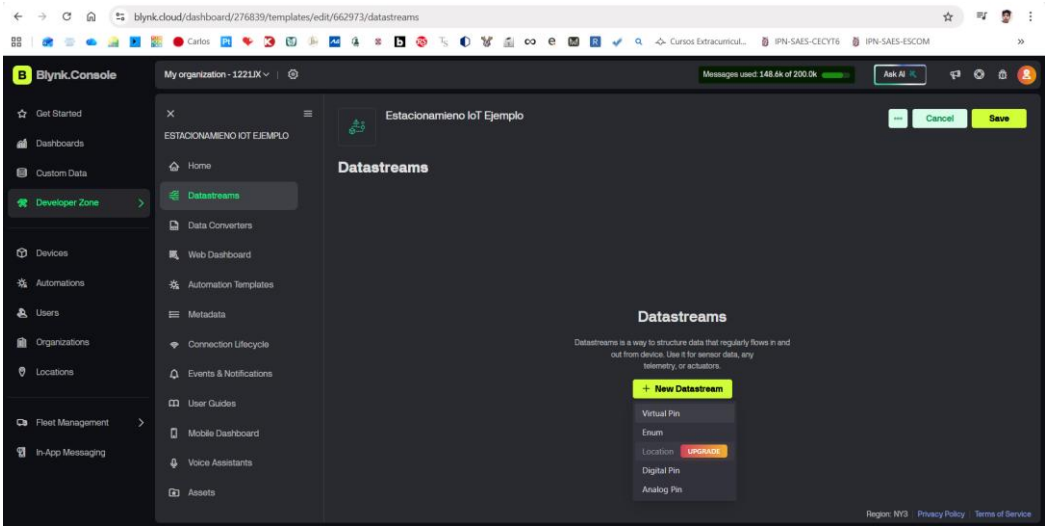


Imagen 23. Añadir Datastreams.

Se abrirá una ventana donde podremos configurar los Datastremas. En nuestro caso configuraremos diez.

- **Datastreams (V0-V5)**

Estos primeros seis Datastreams corresponden a las seis plazas de aparcamiento que tendrá el estacionamiento. En el nombre de cada Datastream pondremos el número de la plaza correspondiente mientras que el pin se deja por defecto y el tipo de dato será **Entero** (**Integer)** ya que estamos tomando la salida digital de los sensores KY-024 por lo que el valor mínimo será 0 pues corresponde al estado en bajo del sensor (plaza disponible), mientras que el máximo será 1 que corresponde al estado en alto del sensor (plaza ocupada) y el valor por defecto será 0 pues en un inicio no habrá ningún vehículo en ninguna de las plazas.

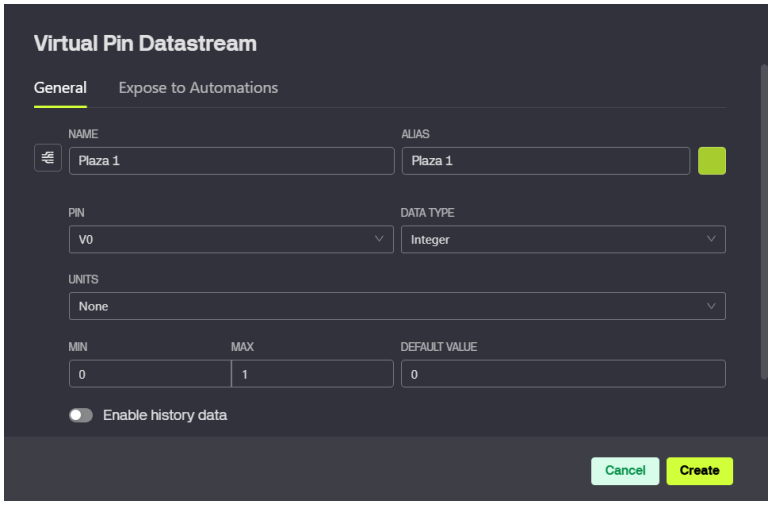


Imagen 24. Datastream para las plazas de aparcamiento.

- **Datastream (V6)**

Este Datastream tiene las mismas configuraciones que los seis anteriores solo que su uso será diferente, será el que bloqueará o desbloqueará la entrada del estacionamiento. Para bloquear la entrada se enviará un uno y para desbloquearla un cero. El valor por defecto será cero para que la entrada este desbloqueada en un inicio, lo único diferente es que el nombre será **Bloqueo manual**.

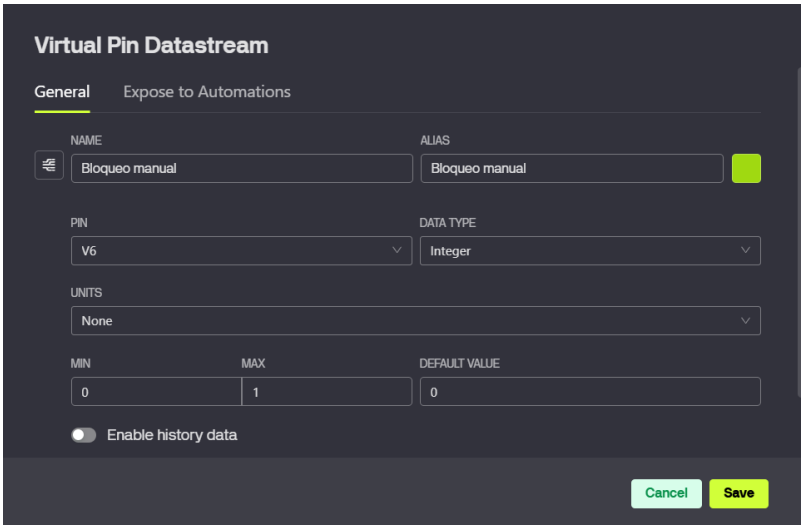


Imagen 25. Datastream para el bloqueo manual.

- **Datastream (V7)**

Para este Datastream todo cambia, ahora el tipo de dato será una **Cadena de Caracteres (String)** pues se recibirán mensajes de este tipo que indican cuando un vehículo entra/sale y cuando no hay ningún movimiento. Su valor por defecto será el mensaje **Sin cambios** porque al inicio ningún vehículo entrará o saldrá del estacionamiento.

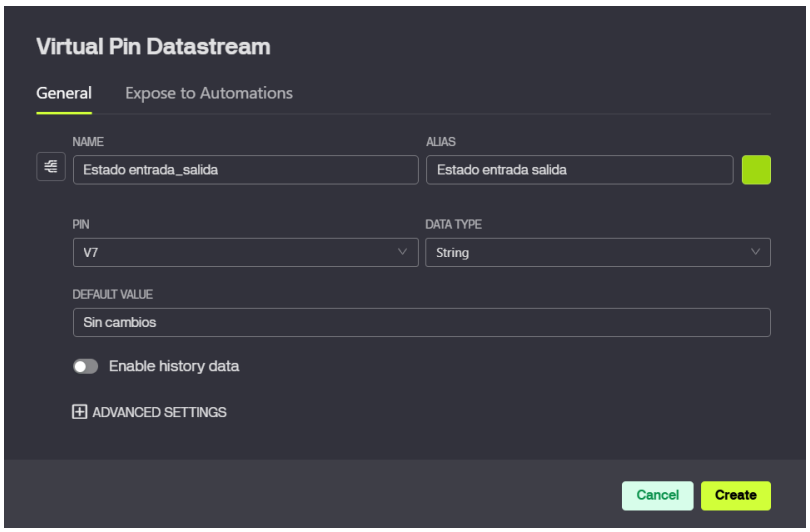


Imagen 26. Datastream para el estado de la entrada/salida.

- **Datastream (V8)**

Este Datastream tendrá el mismo tipo de dato que el anterior, solo que su propósito será recibir mensajes que indican el estado del estacionamiento, si el estacionamiento está lleno o si hay espacio disponible o si la salía está bloqueada/desbloqueada. Por ende, el valor por defecto será **Espacio disponible** porque el estacionamiento no estará lleno en un inicio.

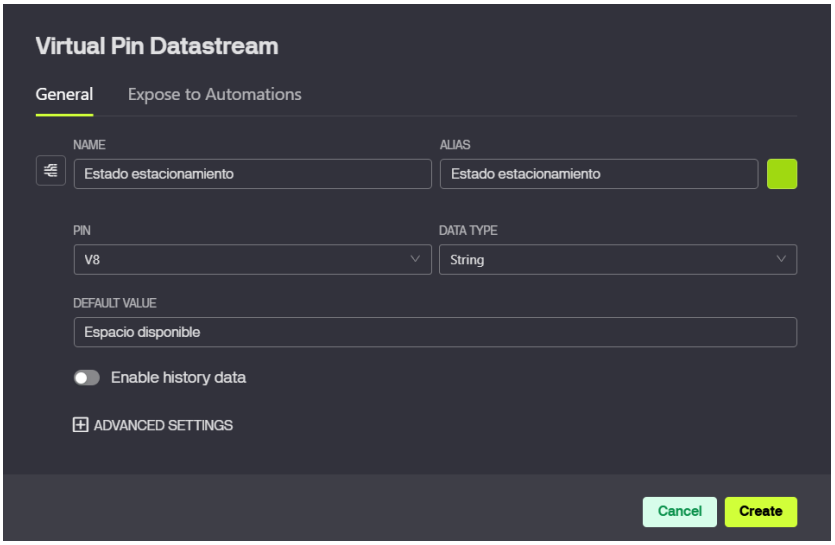


Imagen 27. Datastream para el estado del sistema.

- **Datastream (V9)**

Este Datastream va a ser de tipo **Entero (Integrer)**, pues es el que se encargará de mostrar cuantos vehículos hay actualmente en el estacionamiento y como son seis plazas el mínimo de vehículos será cero mientras que el máximo será seis, y el valor por defecto será cero porque se deduce que no hay ningún vehículo dentro en un inicio.

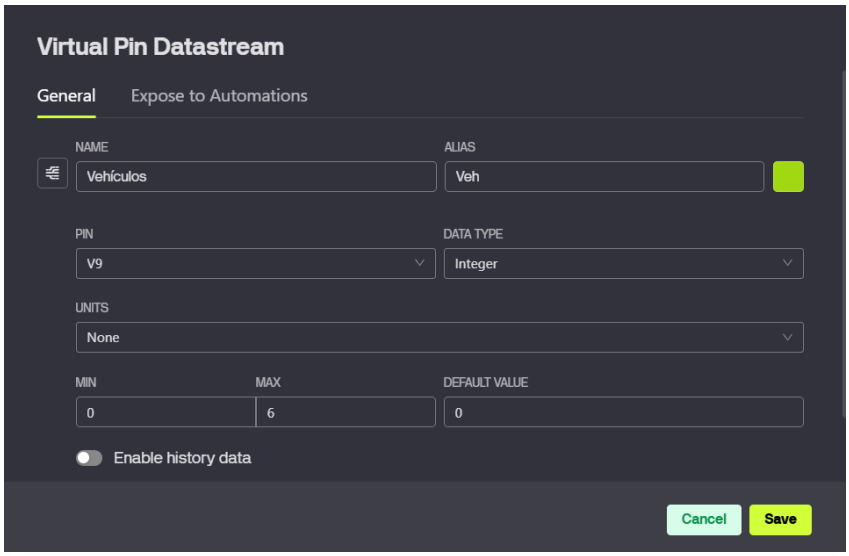


Imagen 28. Datastream para la cantidad de vehículos en el estacionamiento.

3. Configurar el Panel Web

Después de haber configurado los Datastreams, lo siguiente será configurar el **Panel Web (Web Dashboard)** para interactuar con la ESP32 y visualizar los datos que se envían desde este. Para ello nos dirigimos a la pestaña **Web Dashboard** y hacemos clic en el botón **Editar (Edit)** en la esquina superior derecha y arrastramos los siguientes widgets en el lienzo.

- **Widgets para las plazas del estacionamiento**

Para las plazas del estacionamiento se usaremos displays de tipo **LED** esto para mostrar cuando una plaza este ocupada (prendiendo el led) y disponible (apagando el LED). Arrastramos los seis displays y colocaremos de la siguiente forma:

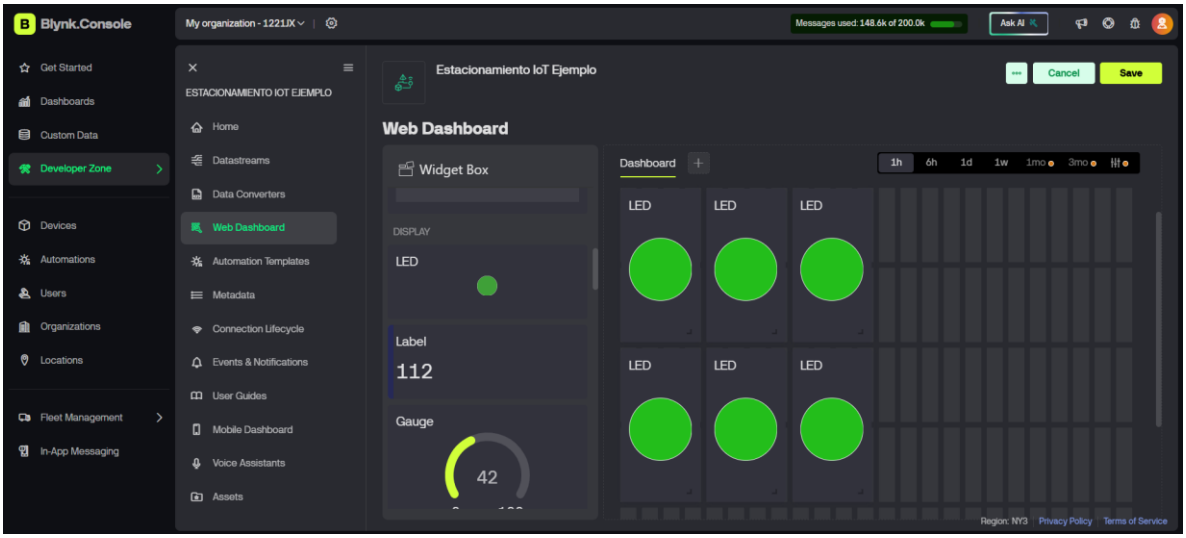


Imagen 29. Widgets para las plazas del estacionamiento.

- **Widgets para el bloqueo/desbloqueo manual de la entrada del estacionamiento**

Para el control del bloqueo manual de la entrada se usarán controles de tipo **Switch**, este se activará para bloquear la entrada y desactivará para desbloquearla. Para este caso volveremos a usar un display de tipo **LED** solo para indicar el bloqueo en el **Panel Web**. Arrastramos estos widgets y los colocamos de la siguiente forma:

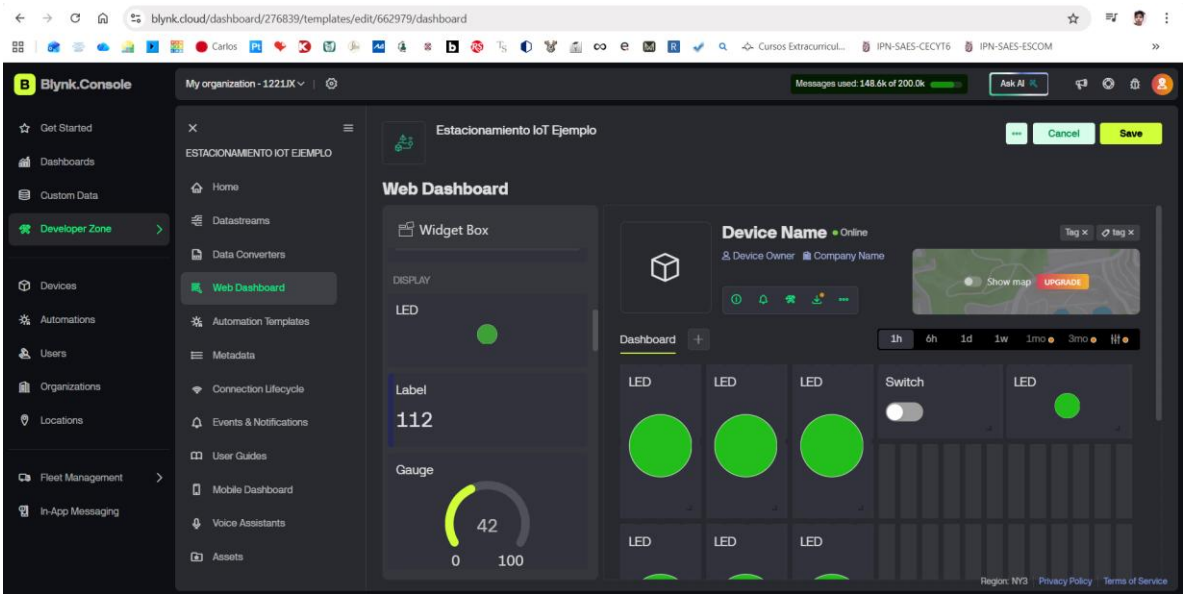


Imagen 30. Widgets para el bloqueo/desbloqueo de la entrada del estacionamiento.

- **Widget para el estado de la entrada/salida del estacionamiento**

Para mostrar los mensajes sobre el estado de la entrada/salida del estacionamiento se usará un display de tipo **Label**. Lo arrastramos y lo colocamos de la siguiente forma:

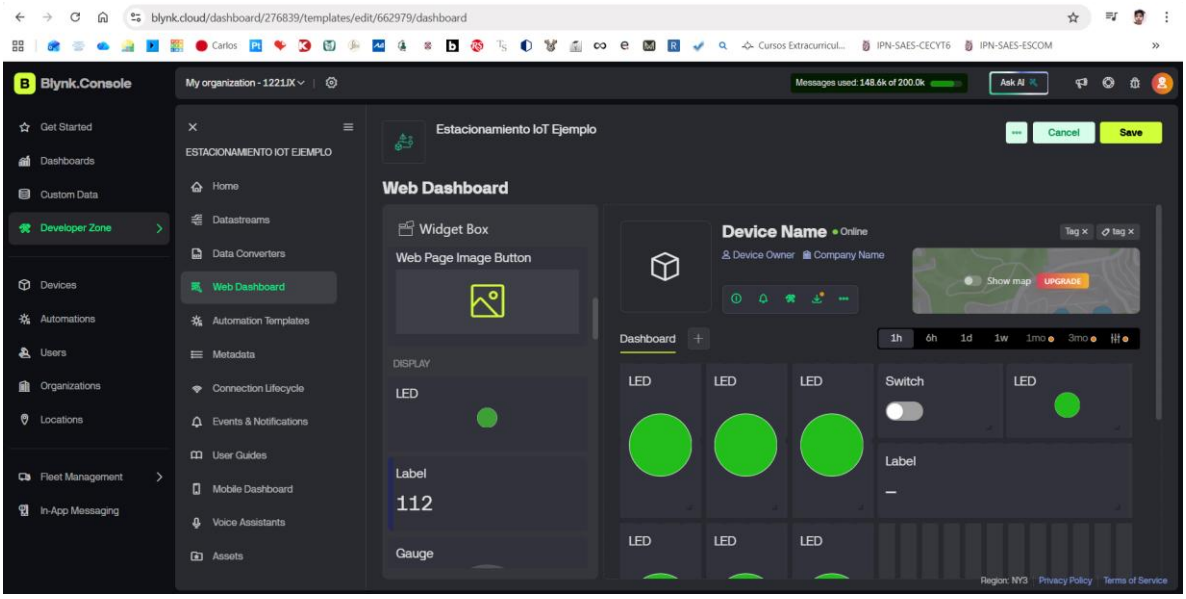


Imagen 31. Widget para el estado de la entrada/salida del estacionamiento.

- **Widget para el estado del estacionamiento**

Para esto se usará el mismo widget que el anterior. Arrastramos y colocamos el widget de la siguiente forma:

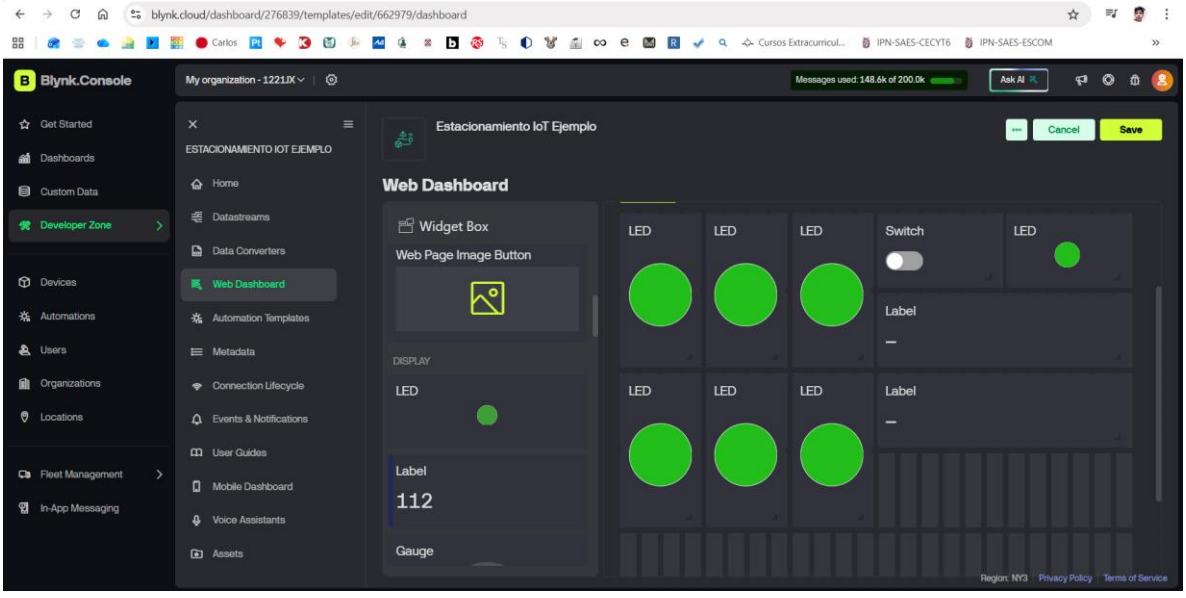


Imagen 32. Widget para el estado del estacionamiento.

- **Widgets para la cantidad de vehículos en el estacionamiento**

Para mostrar la cantidad de vehículos en el estacionamiento se usará nuevamente un display de tipo **Label**, y como un agregado a esto por cada vez que se actualice este número se va a reproducir una alarma, para ello se usará un display de tipo **Alarm and Sound**. Arrastramos y colocamos ambos widgets de la siguiente forma:

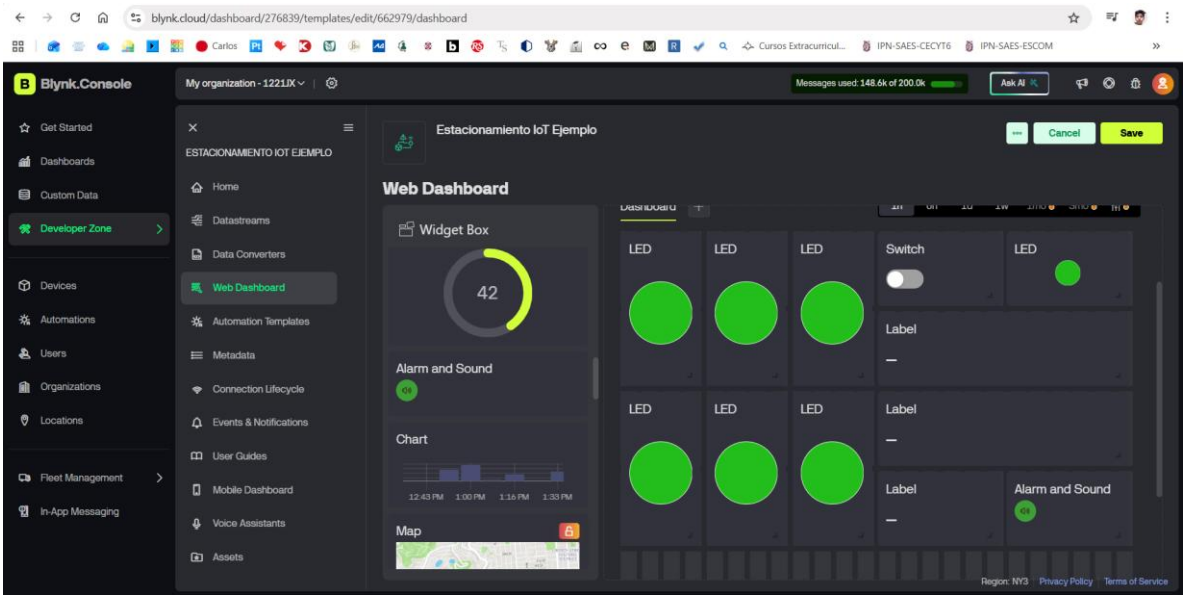


Imagen 33. Widgets para la cantidad de vehículos en el estacionamiento.

Cabe destacar que es necesario guardar el progreso, de lo contrario después de un tiempo todo el lienzo se limpiará. Para este punto el Panel Web, ya se encuentra terminado y se verá de la siguiente forma:

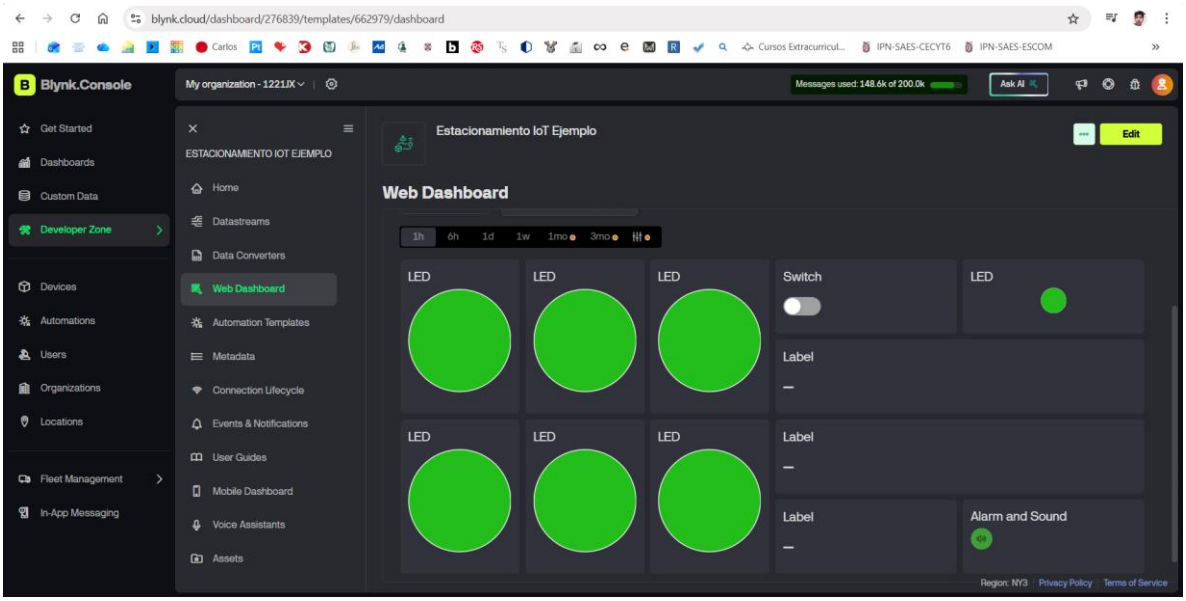


Imagen 34. Panel Web final.

4. Configurar los widgets

Hasta el momento solo hemos configurado la Plantilla del Panel Web, falta configurar cada uno de los widgets, por lo que colocamos el cursor sobre cada uno de ellos y damos clic en el botón con la imagen de engranaje.

Configuraremos los widgets de la siguiente forma:

- **Widgets para las plazas del estacionamiento**

Para cada uno de estos widgets le asignamos su Datastream correspondiente al número de plaza, Datastreams del V0-V5.

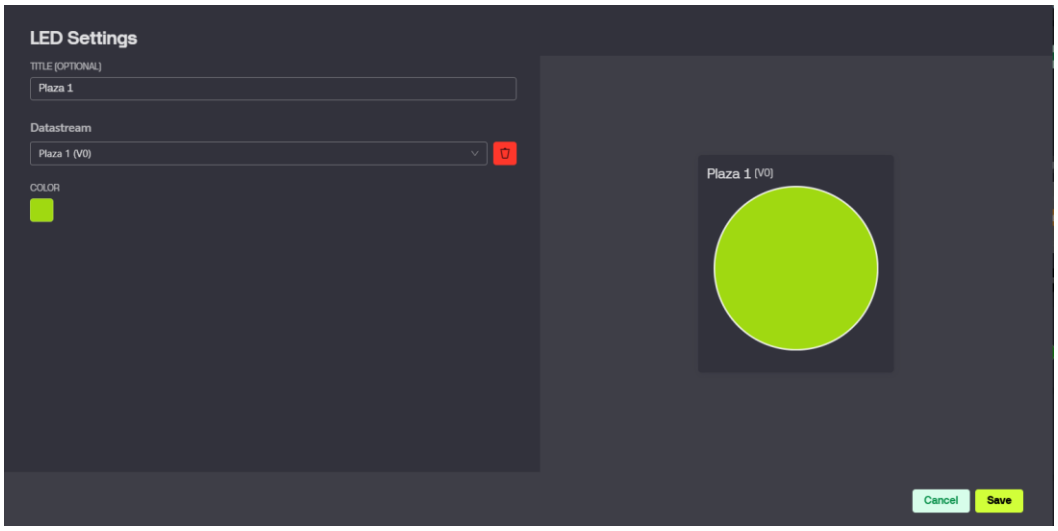


Imagen 35. Configuración del widget para las plazas de aparcamiento.

- **Widgets para el bloqueo/desbloqueo manual de la entrada del estacionamiento**

Para el widget de tipo **Switch** le asignamos el Datastream V6 y lo configuraremos de la siguiente forma, mientras que para el tipo **LED** le asignamos el mismo.

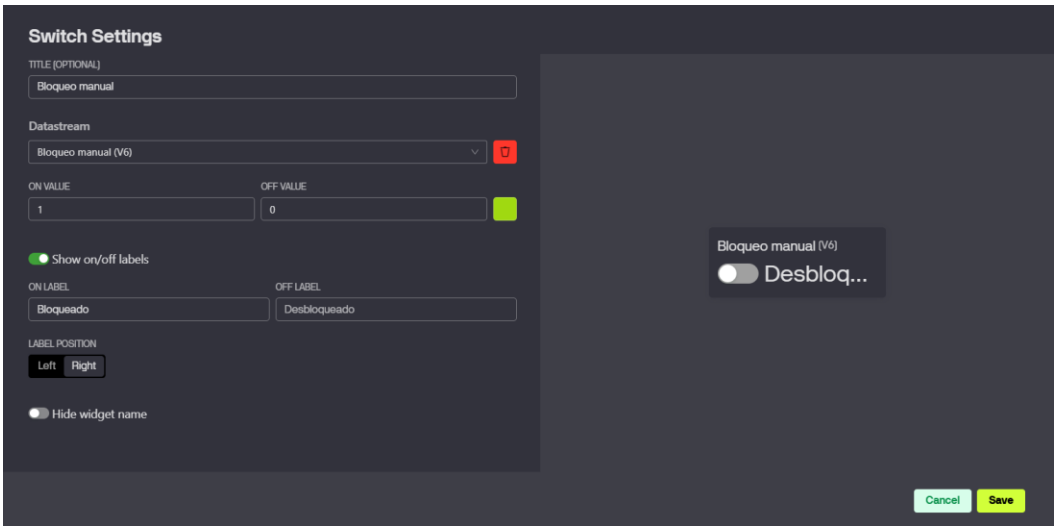


Imagen 36. Configuración del widget para el bloqueo/desbloqueo de la entrada del estacionamiento.

- **Widget para el estado de la entrada/salida del estacionamiento**

Para este widget le asignamos el Datastream V7 y lo configuramos de la siguiente forma:

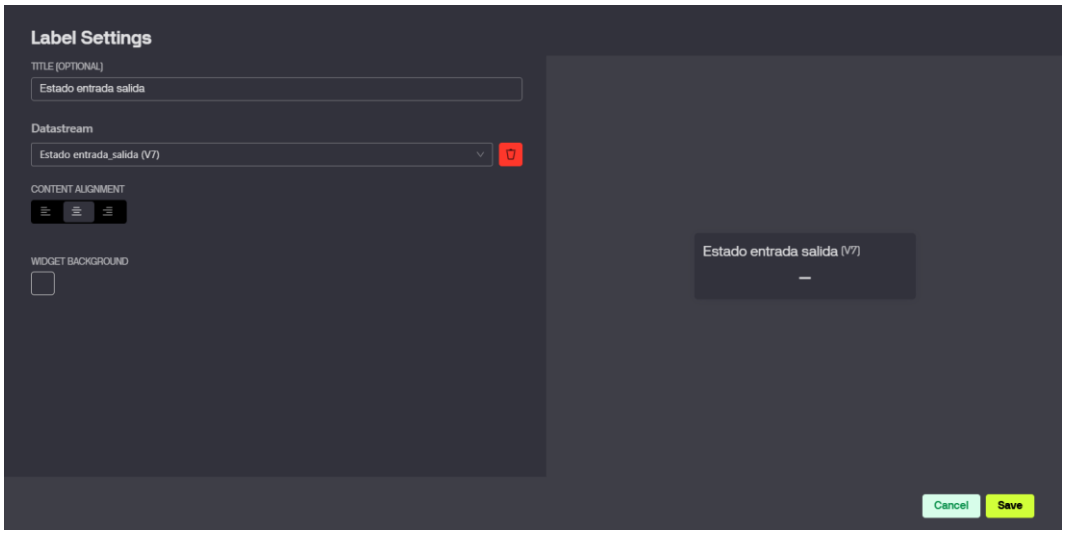


Imagen 37. Configuración del widget para el estado de la entrada/salida del estacionamiento.

- **Widget para el estado del estacionamiento**

Para este widget le asignamos el Datastream V8 y lo configuramos de la siguiente forma:

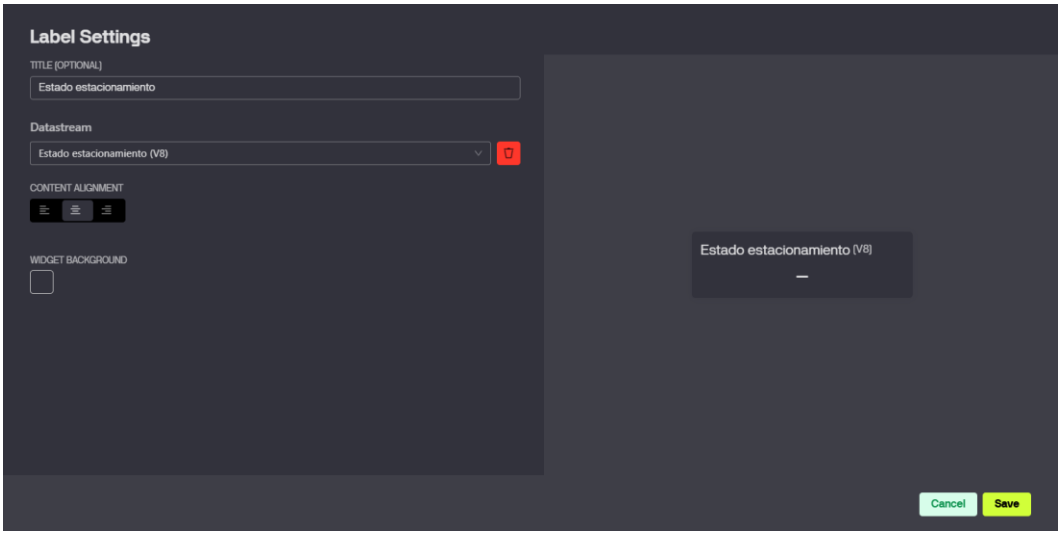


Imagen 38. Configuración del widget para el estado del estacionamiento.

- **Widgets para la cantidad de vehículos en el estacionamiento**

Para este widget asígnele le asignamos el Datastream V9. Este widget cambiará de color verde a rojo cuando los vehículos superan los cuatro dentro de estacionamiento. También

contará con una barra de nivel que se va a ir llenando, dependiendo la cantidad de vehículos dentro.

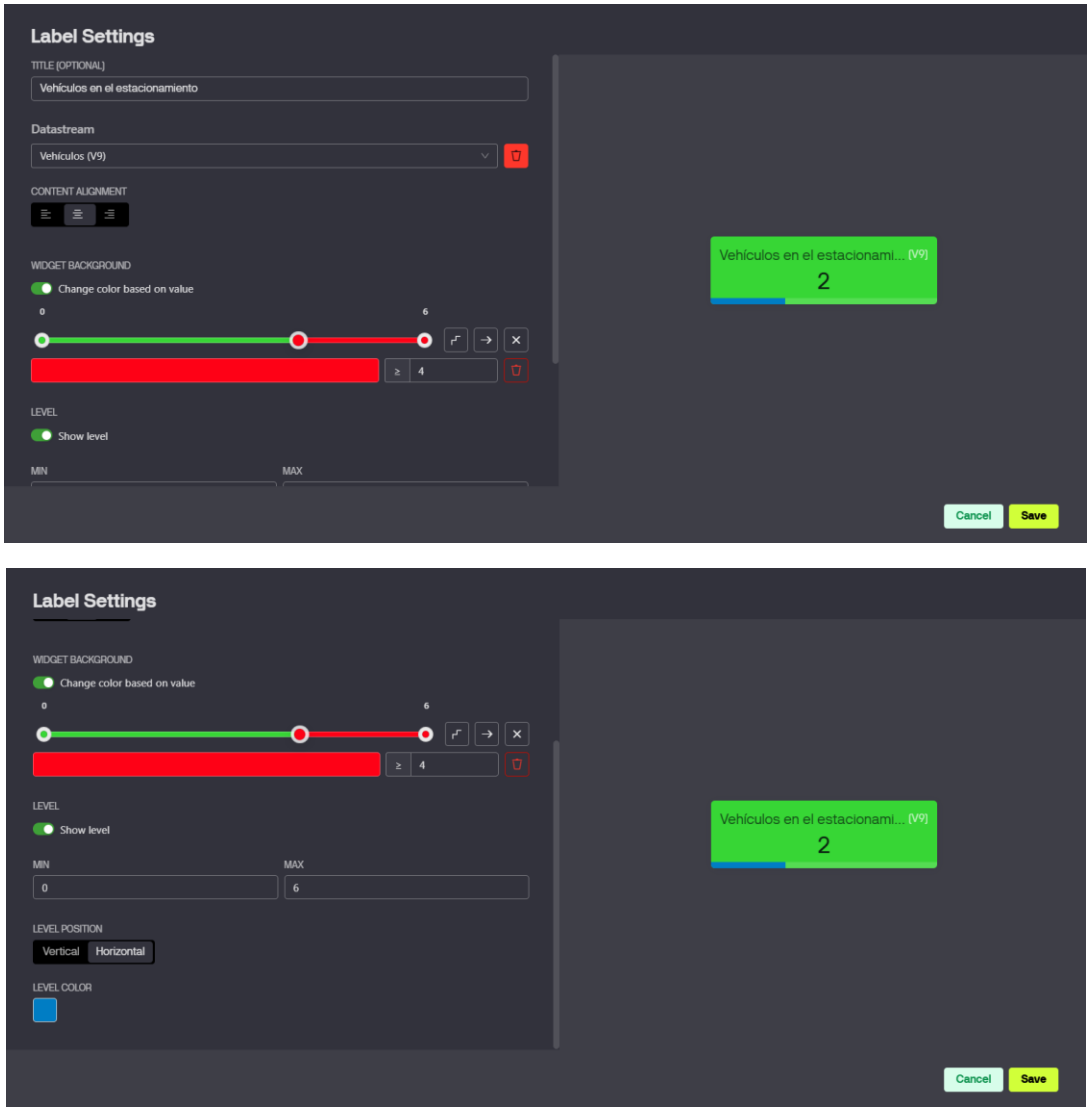


Imagen 39. Configuración del widget para la cantidad de vehículos en el estacionamiento.

Para el widget de la alarma se le asigna el mismo Datastream y también se indicará que la alarma deberá de sonar una vez y que cambiará el título por **Vehículo detectado**.

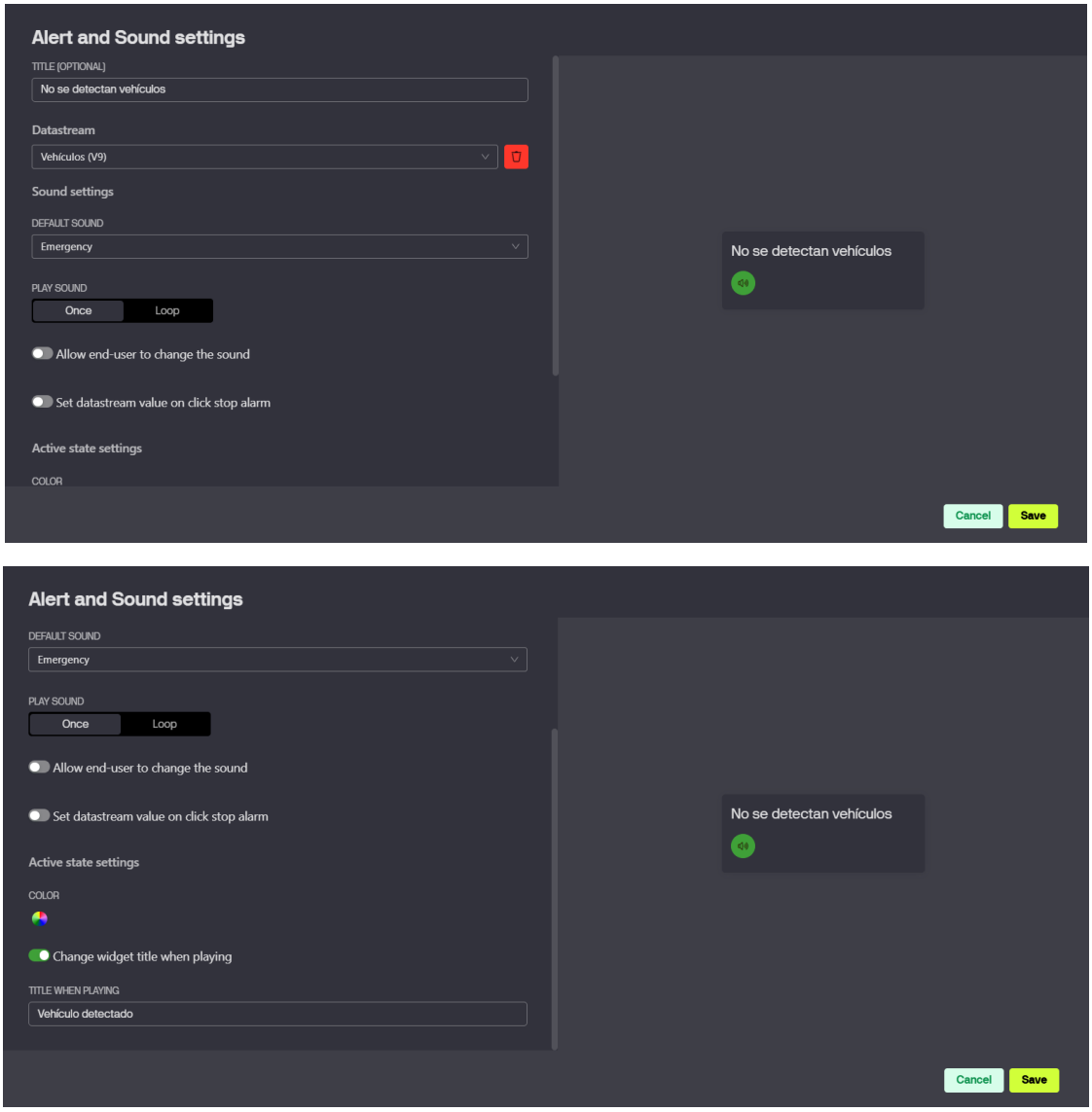


Imagen 40. Configuración del widget para la alarma al detectarse un vehículo.

5. Agregar un dispositivo a la Plantilla

Hasta este punto ya hemos configurado toda la interfaz de usuario, sin embargo, aún no se puede conectar con la ESP32, lo que sigue será aplicar la plantilla a un dispositivo, en nuestro caso agregar un dispositivo a la Plantilla.

Para ello nos dirigimos a la página principal de la Plantilla y le damos clic a la opción **Agregar Primer Dispositivo (Add first Device)**, y escribiremos el nombre del dispositivo, que en nuestro caso será **Estacionamiento-IoT**.

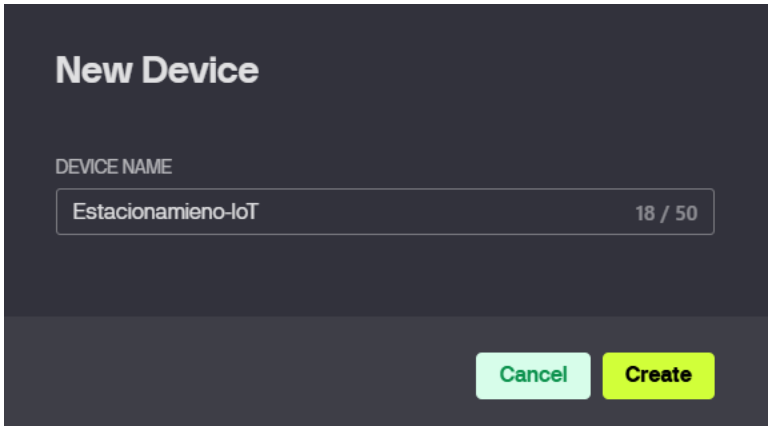


Imagen 41. Agregar un dispositivo a la Plantilla.

Una vez creado el dispositivo, en la parte superior derecha de la página aparecerá una notificación con la información[3] necesaria en forma de macros para conectar la ESP32 con la Plantilla.

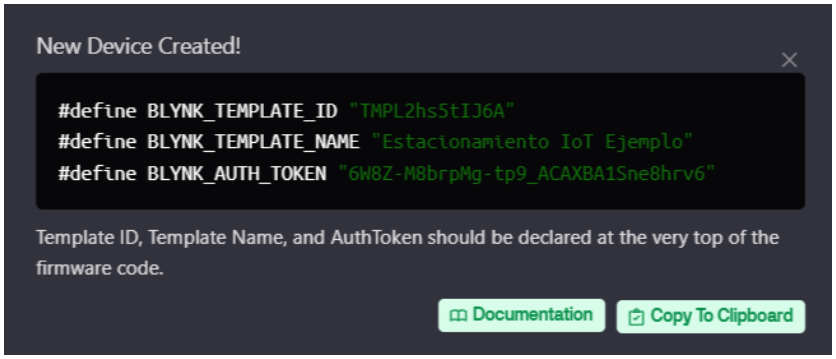


Imagen 42. Información para la conexión con la Plantilla.

Esta información se deberá de colocar hasta arriba del código.

Para más información consulte **Set Up Web Dashboard** de la documentación oficial de Blynk.

Creación de la interfaz en Blynk (Móvil)

La plataforma Blynk también cuenta con su aplicación móvil en donde también podremos monitorear y controlar el estacionamiento, sin embargo, hay que hacer la Plantilla exclusiva para la aplicación móvil[4]. A continuación, se muestran los pasos para crear la Plantilla.

[3] Recuerde no compartir públicamente esta información, de lo contrario cualquiera podrá acceder a su plantilla de Blynk.
[4] Nos limitaremos a solo explicar de manera general la creación de la Plantilla en la versión móvil.

1. Descargar la aplicación móvil

En este caso para descargar la aplicación usaremos Play Store y buscaremos **Blynk IoT** y el resultado es el siguiente:

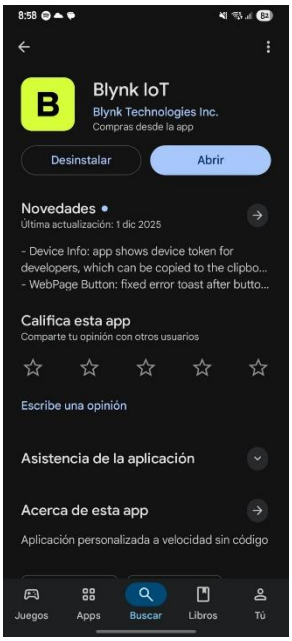


Imagen 43. Aplicación móvil de Blynk.

2. Seleccionar la Plantilla creada desde la versión web

Al abrir la aplicación debemos de iniciar sesión con la misma cuenta que creamos anteriormente. Deberemos de ver inmediatamente la Plantilla creada desde la versión web Blynk .

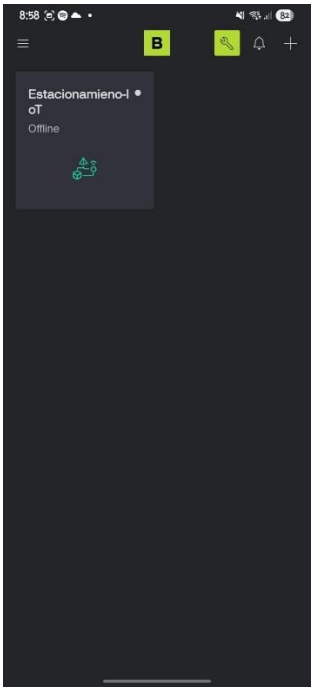


Imagen 44. Plantilla creada anteriormente, pero desde la aplicación móvil.

3. Crear la Plantilla

Seleccionamos la Plantilla creada e inmediatamente se mostrará la Plantilla móvil vacía.

Para empezar a agregar widgets, seleccionaremos el botón verde con la imagen de herramienta en la parte superior, de esta forma entramos al modo editor en la cual en la parte inferior aparecerá un menú flotante con varias opciones. Seleccionaremos el botón agregar (botón verde), esto nos mostrará una lista con los diferentes widgets que se pueden agregar a la Plantilla.

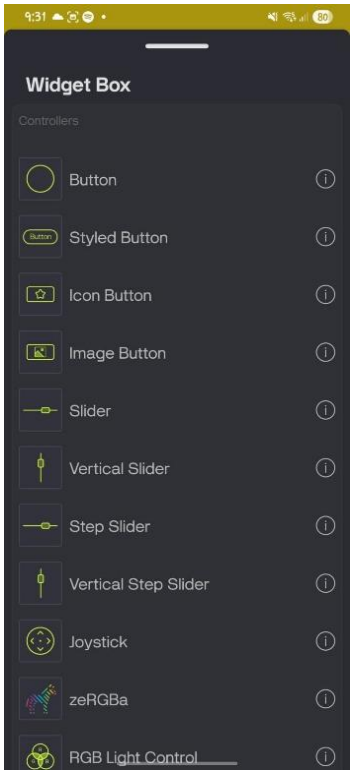


Imagen 45. Menú de widgets.

Buscamos y seleccionamos los widgets que usamos en la versión web e interfaces de tipo texto el nombre de cada plaza de aparcamiento y los títulos de las partes de la interfaz, estos aparecerán en el lienzo. Para poder configurar un widget lo seleccionamos y le asignamos el Datastream correspondiente, luego arrastramos el widget en cualquier posición del panel. Al final obtendremos una Plantilla como la siguiente:

Imagen 46. Plantilla resultante.

Cuando conectemos la ESP32 con Blynk, entonces podremos monitorear el estacionamiento de la misma forma que con la versión web, lo que aparece en esa versión también se reflejará en la versión móvil.

Para más información consulte **Set Up Mobile App Dashboard** de la documentación oficial de Blynk.

Conexión con Blynk

- ## Activación del dispositivo

Para la autenticación en la plataforma Blynk, cada dispositivo debe tener un AuthToken único. Este AuthToken es el identificador principal de cada dispositivo en la nube de Blynk. Dependiendo del hardware, la conectividad y el caso de uso de IoT en el que se trabaje, la forma de obtener AuthTokens para el dispositivo puede variar.

Hay tres formas de activar dispositivos en la plataforma Blynk:

1. Crear manualmente un dispositivo usando Blynk.Console para la creación de prototipos iniciales (funciona para cualquier hardware)
2. Aprovisionar dinámicamente AuthTokens y credenciales WiFi para **dispositivos compatibles**.
3. Utilizar tokens estáticos para redes celulares, Ethernet y otros métodos de conexión que no sean WiFi.

En nuestro caso nos inclinaremos por la primera opción. Para esta parte del reporte usaremos uno de los ejemplos que proporciona la biblioteca de Blynk en Arduino IDE y también una plantilla de ejemplo.

- ## Activación manual del dispositivo

La creación manual de dispositivos se puede utilizar con cualquier dispositivo compatible con Blynk sin importar el tipo de conexión que se utilice.

Para ello seguiremos los siguientes pasos:

Paso 1: Preparación del boceto

1. Instalar la última versión de Blynk Library en el IDE que se use (en este caso Arduino IDE).
2. Después de esto nos dirigimos a Archivo > Ejemplos > Blynk > Boards_WiFi > ESP32_WiFi para ver el ejemplo con la ESP32.

```
/* Comment this out to disable prints and save space */
#define BLYNK_PRINT Serial

/* Fill in information from Blynk Device Info here */
//#define BLYNK_TEMPLATE_ID           "TMPxxxxxx"
//#define BLYNK_TEMPLATE_NAME         "Device"
//#define BLYNK_AUTH_TOKEN            "YourAuthToken"


#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>

// Your WiFi credentials.
// Set password to "" for open networks.
char ssid[] = "YourNetworkName";
char pass[] = "YourPassword";

void setup()
{
  // Debug console
  Serial.begin(9600);

  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
}

void loop()
{
  Blynk.run();
}
```

Imagen 47. Ejemplo ESP32_WiFi.

Hay que prestar atención a las siguientes cuatro líneas, pues se van a tener que rellenar con la información que proporciona Blynk, que si prestamos atención es la misma información que obtuvimos al Agregar un dispositivo a la Plantilla.

```
//#define BLYNK_TEMPLATE_ID           "TMPxxxxxx"
//#define BLYNK_TEMPLATE_NAME         "Device"
//#define BLYNK_AUTH_TOKEN            "YourAuthToken"
```

```
char ssid[] = "YourNetworkName";
char pass[] = "YourPassword";
```

Imagen 48. Líneas para rellenar con información de Blynk.

Paso 2: Obtener el ID de la plantilla

1. Iniciamos sesión con nuestra cuenta de desarrollador de Blynk.Console, y nos dirigimos a la **Zona de desarrollador**. Para esta sección del reporte crearemos una Plantilla de ejemplo con el nombre **Ejemplo Activacion**.
2. En la página de inicio de la Plantilla, se podemos observar en la esquina inferior derecha la siguiente información:

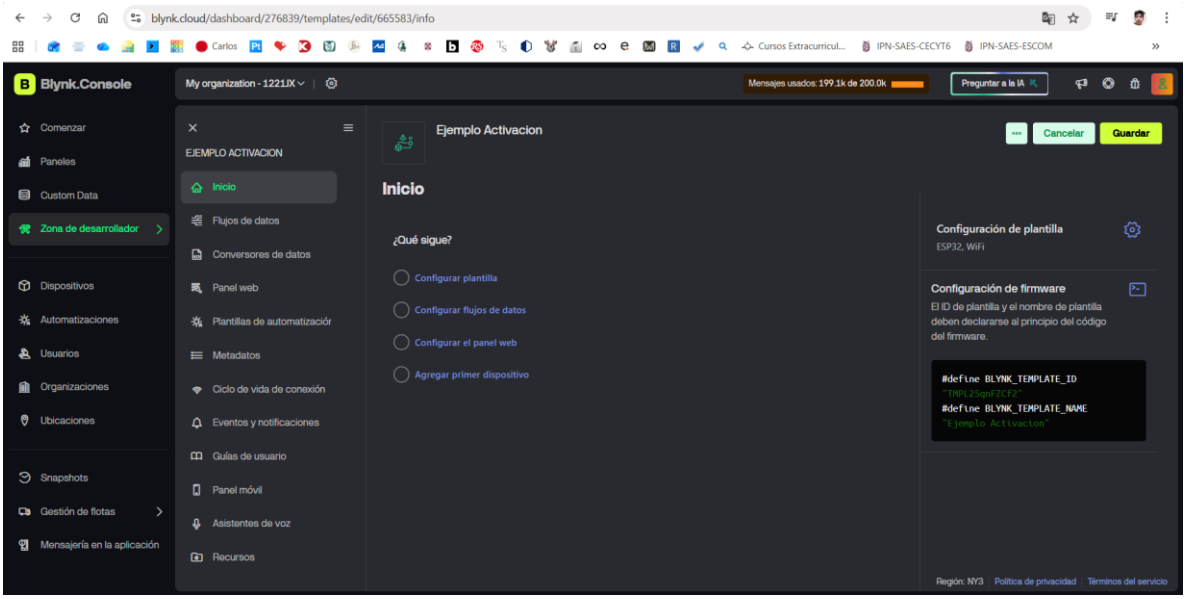


Imagen 49. Página de inicio de la plantilla.

```
#define BLYNK_TEMPLATE_ID
"TMPL25qnFZCf2"
#define BLYNK_TEMPLATE_NAME
"Ejemplo Activacion"
```

Imagen 50. Nombre e ID de la plantilla.

Esta información la deberemos de copiar y pegar nuestro boceto.

Paso 3: Obtener el token de autorización

Cuando la Plantilla esta lista, nos dirigimos a **Dispositivos** en la parte lateral izquierda, creamos un nuevo dispositivo a partir de la Plantilla creada con el nombre **Dispositivo de**

ejemplo.

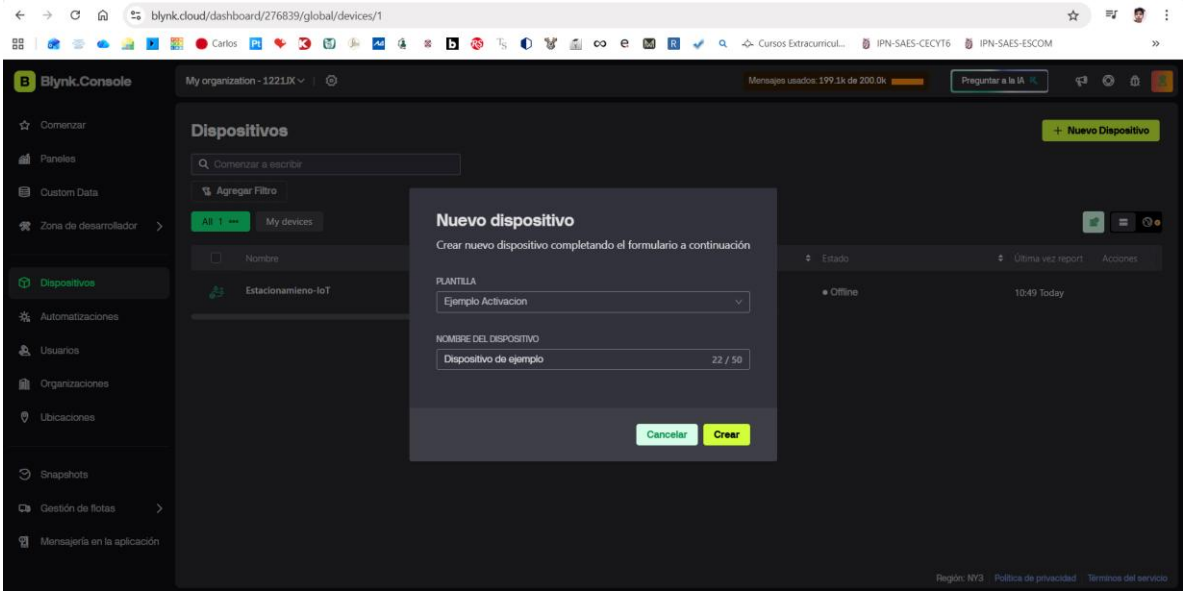


Imagen 51. Crear un nuevo dispositivo.

Una vez creado el dispositivo, se mostrará una notificación en la esquina superior derecha con el ID de la plantilla y el token de autenticación.

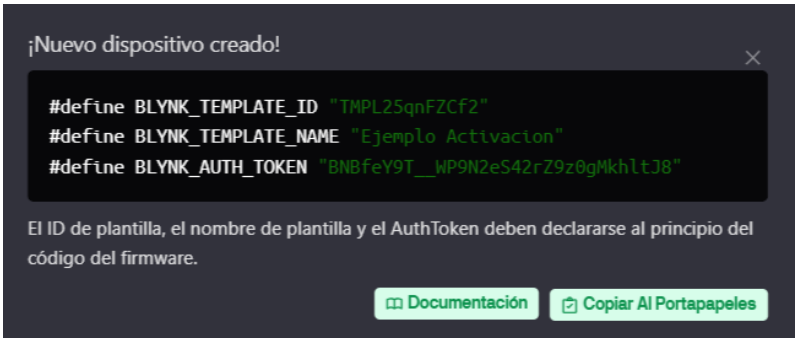


Imagen 52. Notificación al crear el dispositivo.

Hasta aquí ya se cuenta con la información necesaria para nuestro boceto.

Paso 4: Actualización del boceto con el ID de plantilla y el AuthToken

Añadimos `TEMPLATE_ID`, `TEMPLATE_NAME` y `AUTH_TOKEN` al boceto.

```
/* Comment this out to disable prints and save space */
#define BLYNK_PRINT Serial

/* Fill in information from Blynk Device Info here */
#define BLYNK_TEMPLATE_ID "TMPL25qnFZCf2"
#define BLYNK_TEMPLATE_NAME "Ejemplo Activacion"
#define BLYNK_AUTH_TOKEN "BNBfeY9T__WP9N2eS42rZ9z0gMkhltJ8"


#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>

// Your WiFi credentials.
// Set password to "" for open networks.
char ssid[] = "YourNetworkName";
char pass[] = "YourPassword";

void setup()
{
  // Debug console
  Serial.begin(9600);

  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
}

void loop()
{
  Blynk.run();
}
```

Imagen 53. Ejemplo ESP32_WiFi actualizado.

Luego colocamos el nombre de nuestra red WiFi y su contraseña en los arreglos `ssid[]` y `pass[]` respectivamente.

Cargamos al boceto al dispositivo y al abrir el Monitor Serial. Esperamos hasta que veamos algo como esto:

```
Blynk v.X.X.X
Your IP is 192.168.0.11
Connecting to...
Ready (ping: 40ms)
```

Imagen 54. Conexión establecida.

Listo, con esto el dispositivo está en línea y en Blynk inmediatamente aparecerá el estado como en línea.

Para más información consulte **Manual Device Activation** de la documentación oficial de Blynk.

Enviar datos desde el hardware a Blynk y Pines Virtuales

Cómo fluyen los datos desde del dispositivo a Blynk

Con Blynk se pueden enviar datos sin procesar o procesados desde cualquier sensor o actuador conectado a la MCU.

Al enviar datos a Blynk, estos fluyen a través de un flujo de datos mediante el protocolo Blynk. Cada valor se marca automáticamente con la fecha y hora, y se almacena en la base de datos de Blynk.Cloud.

Recordemos que un Datastream es un canal que le dice a Blynk qué tipo de datos fluyen a través de él.

Pines Virtuales

En la sección **Configurar Datastreams**, se indicó que se eligiera la opción **Virtual Pin** para todos los Datastreams, sin embargo, no se explicó el por qué se elegio ni cómo funciona, en esta sección se explicará todo esto.

Lo pines virtuales permiten intercambiar datos entre el hardware (ESP32) y la aplicación Blynk. Los pines virtuales se pueden ver como canales para enviar datos. Cabe aclarar que los pines virtuales son diferentes de los pines GPIO físicos del hardware. Los pines virtuales no tienen representación física.

Los pines virtuales se utilizan comúnmente para interactuar con otras bibliotecas e implementar lógica personalizada. El dispositivo puede enviar datos mediante `Blynk.virtualWrite(pin, value)` y recibir datos de la aplicación mediante `BLYNK.WRITE(vPIN)`.

Flujo de datos de pines Virtuales

Los pines virtuales son una abstracción de Blynk diseñada para intercambiar datos entre el hardware y Blynk. Cualquier dispositivo conectado al hardware podrá comunicarse con Blynk. Con los pines virtuales se puede enviar información desde la aplicación, procesarla en el microcontrolador y luego enviarla de vuelta el smartphone. Se pueden activar funciones, leer dispositivo I2C, convertir valores, controlar servos y motores de CC, etc.

Los pines virtuales se pueden usar para interactuar con bibliotecas externas e implementar funcionalidades personalizadas.

¿Por qué utilizar PIN virtuales para enviar datos al dispositivo?

- Los pines virtuales son independientes del hardware. Esto significa que será mucho más fácil portar el código de una plataforma a otra en el futuro.
- Hay más control sobre las funciones de los widgets al usar pines virtuales.
- Los pines virtuales son más predecibles (estables) que manipular los pines digitales.

¿Cómo se relacionan los pines virtuales con los pines GPIO del hardware?

Los pines virtuales son simplemente una forma de enviar un mensaje desde la aplicación al código que se ejecuta en la placa (a través del servidor Blynk). No existe relación entre los pines virtuales y ninguno de los pines GPIO físicos del hardware.

Enviar datos con la API de firmware de la biblioteca Blynk

Ese método utiliza el protocolo Blynk y es el método más común y fácil de usar cuando se requiere enviar datos en tiempo real.

Primero, se debe de configurar una **plantilla** con un **flujo de datos** para configurar qué tipo de datos se envían al hardware. Para ello se usa `Blynk.virtualWrite(vPin, valor)`.

`Blynk.virtualWrite(vPin, valor)`[5]

Envía datos en varios formatos a pines virtuales.

```
// Envía string
Blynk.virtualWrite(pin, "abc");

// Envía integer
Blynk.virtualWrite(pin, 123);

// Envía float
Blynk.virtualWrite(pin, 12.34);

// Envía multiples valores como un array
Blynk.virtualWrite(pin, "hello", 123, 12.34);

// Envía datos RAW
Blynk.virtualWriteBinary(pin, buffer, length);
```

Imagen 55. Datos en varios formatos para `Blynk.virtualWrite()`.

Para los pines virtuales con números mayores a 127, la `V128` sintaxis no está disponible. Se debe de usar un PIN virtual simple, por ejemplo:

```
Blynk.virtualWrite(128, "abc");
```

Imagen 56. PIN virtual simple.

Usaremos `Blynk.virtualWrite()` para enviar información a los Pines virtuales `V0`–`V5`, `V7`, `V8`, `V9`, que ya se configuraron anteriormente tipo:

```
const int sensores_pines[6] = {16, 33, 32, 19, 18, 23};
const int virtual_pines[6] = {V0, V1, V2, V3, V4, V5};
```

[5] Se debe de utilizar `BlynkTimer` al usar este comando para enviar datos. De lo contrario, el hardware podría desconectarse del servidor.

```
for (int i = 0; i < 6; i++) {
  int lectura = digitalRead(sensores_pines[i]);
  Blynk.virtualWrite(virtual_pines[i], lectura);
}
```

Imagen 57. Estado de las plazas de aparcamiento a los pines virtuales `V0-V5`.

```
Blynk.virtualWrite(V7, "Sin cambios");
Blynk.virtualWrite(V7, "Vehículo ENTRANDO...");
Blynk.virtualWrite(V7, "Vehículo SALIENDO...");
```

Imagen 58. Mensajes que indican cuando un vehículo entra/sale y cuando no hay ningún movimiento al pin virtual `V7`.

```
Blynk.virtualWrite(V8, "Estacionamiento LLENO");
Blynk.virtualWrite(V8, "Espacio disponible");
Blynk.virtualWrite(V8, "Entrada BLOQUEADA (Manual)");
Blynk.virtualWrite(V8, "Entrada DESBLOQUEADA (Manual)");
```

Imagen 59. Mensajes que indican el estado del estacionamiento al pin virtual `V8`.

```
Blynk.virtualWrite(V9, carros);
```

Imagen 60. Cuantos vehículos hay en el estacionamiento al pin `V9`.

Por otro lado, para recibir datos desde Blynk al hardware se usa `BLYNK_WRITE(vPin)`.

BLYNK_WRITE(vPIN)

`BLYN_WRITE` es una función que se llama cada vez que el dispositivo recibe una actualización del valor del PIN virtual del servidor (o aplicación).

Para leer los datos recibidos se debe usar:

```
BLYNK_WRITE(V0) {
  int value = param.asInt(); // Obtiene el valor como entero

  // El parámetr puede contener multiples valores, en tal caso:
  int x = param[0].asInt();
  int y = param[1].asInt();
}
```

Imagen 61. Leer datos con `BLYNK_WRITE`.

En nuestro caso solo usaremos el Pin virtual `V6` de tipo entero, para interpretar el dato que entra desde del widget de tipo switch que bloquea o desbloquea la entrada del estacionamiento.

```
BLYNK_WRITE(V6) {
  int estado = param.asInt(); // 1: bloquear, 0: desbloquear

  if (estado == 1) {
    bloqueo_manual = true;
    Blynk.virtualWrite(V8, "Entrada BLOQUEADA (Manual)");
  } else {
    bloqueo_manual = false;
    if (bloqueo_auto) Blynk.virtualWrite(V8, "Estacionamiento LLENO");
    else Blynk.virtualWrite(V8, "Entrada DESBLOQUEADA (Manual)");
  }

  actualizar_bloqueo_general();
}
```

Imagen 62. Recibir datos sobre del bloqueo manual desde el pin virtual `V6`.

Tipos de datos de Pin virtual

Todos los valores de los pines se envían siempre como cadenas y no existen límites prácticos para la cantidad de datos que se pueden enviar. Sin embargo, existen ciertas limitaciones de hardware al trabajar con números. Por ejemplo, el entero en Arduino es de 16 bits, lo que permite un rango de -32768 a 32767.

Para interpretar datos entrantes como números enteros, flotantes, dobles y cadenas, se utiliza:

```
param.asInt();
param.asFloat();
param.asDouble();
param.asStr();
```

Imagen 63. Tipos de datos de Pin virtual.

En nuestro caso usamos el tipo `param.asInt()`, porque el datastream del switch está configurado como entero, por lo que esperamos leer valores enteros (1 o 0).

Para más información consulte Virtual Pins y Send Data From Hardware To Blynk de la documentación oficial de Blynk.

Estado en línea/fuera de línea del dispositivo

- **Estados de los dispositivos en línea**

Blynk siempre actualiza el estado en línea del dispositivo en tiempo real. En cuanto al dispositivo envía el `login` comando, el estado se envía a la web y a las aplicaciones móviles. Así, siempre se podrá ver si el dispositivo está conectado o desconectado.

Esta es la razón por la que al conectar la ESP32 a Blynk, inmediatamente en la plataforma se actualiza su estado y se muestra en línea.

- **Estado de los dispositivos fuera de línea**

Idealmente, cuando un dispositivo cierra una conexión TCP con un código como `connection.close()` este, el servidor recibirá una notificación sobre el cierre de la conexión. Esto permite obtener una actualización de estado instantánea en la interfaz de usuario. Sin embargo, en la práctica, este caso es bastante excepcional. En la mayoría de los casos, no existe una forma sencilla e instantánea de saber que la conexión ya no está activa.

Por eso Blynk utiliza `HEARTBEAT` un mecanismo. Con este enfoque, el hardware envía `ping` comandos periódicamente con una frecuencia predefinida (cada 45 segundos por defecto, `BLYNK_HEARTBEAT`). Si el hardware no envía nada en 45 segundos, el servidor esperará 103 segundos más. Después, se asume que la conexión se ha interrumpido y el servidor la ha cerrado. Por lo tanto, la interfaz de usuario mostrará la actualización del estado de la conexión con un ligero retraso, es por esta razón que al desconectar/apagar el dispositivo, aunque sepamos que ya no está en línea en Blynk todavía aparezca y después de un rato aparezca como fuera de línea.

Para más información consulte **Device Online/Offline Status** de la documentación oficial de Blynk.

Protocolo Blynk

La biblioteca Blynk utiliza un protocolo binario propietario, optimizado para aplicaciones IoT, el cual opera sobre una conexión TCP/IP establecida entre la ESP32 y el servidor Blynk.Cloud.

Cada mensaje intercambiado entre el dispositivo y el servidor consta de 2 partes principales:

1. Encabezado (Header)

El encabezado tiene un tamaño fijo de 5 bytes y contiene información necesaria para la gestión del mensaje:

- Comando de protocolo (1 byte);

Indica el tipo de mensaje o acción a realizar (escritura, lectura, respuesta, heartbeat).

- ID del mensaje (2 bytes);

Identificador único que permite asociar solicitudes con sus respuestas.

- Longitud del cuerpo o estado (2 bytes);

Indica el tamaño del cuerpo del mensajea en bytes, o, en algunos casos, el estado de la respuesta.

2. Cuerpo del mensaje (Body).

El cuerpo del mensaje es de longitud variable y contiene los datos transmitidos, tales como:

- Valores de sensores
- Comando de control

- Estados de pines virtuales
- Cadenas de texto o datos numéricos

El tamaño máximo del cuerpo está determinado por el campo de longitud de 2 bytes, permitiendo hasta 65,535 bytes, aunque en la práctica se utilizan mensajes muchos más pequeños.

Estructura del mensaje Blynk:

Comando	ID. del mensaje	Longitud/Estado	Cuerpo
1 byte	2 bytes	2 bytes	Variable

El tamaño del mensaje es 1 byte + 2 bytes + 2 bytes + messageBody.length.

Explicación del código del programa del estacionamiento

```
#define BLYNK_TEMPLATE_ID "TMPL27qOzqAiv"
#define BLYNK_TEMPLATE_NAME "Estacionamiento IoT"
#define BLYNK_AUTH_TOKEN "uIDV6IpETIRlHTezdR6IIqrZpoJSGW1X"

#include <WiFi.h>
#include <BlynkSimpleEsp32.h>
#include <ESP32Servo.h>
```

Imagen 64. Información para la conexión a Blynk.

Se definen los datos requeridos para la conexión del sistema con la plataforma Blynk, una herramienta IoT que permite visualizar en tiempo real el estado del estacionamiento. Después se incluyen las librerías necesarias: la librería WiFi para conectar la ESP32 a la red, la librería de Blynk para comunicar los datos con la aplicación móvil y la librería ESP32Servo que facilita el control del servo encargado de abrir y cerrar la barrera del estacionamiento.

```
// ======> CONSTANTES <======
#define LED_INDICADOR 2 // led de la esp32 que indica bloqueo del sistema
#define SENSOR_ENTRADA 21
#define SENSOR_SALIDA 22
#define MAX_CARROS 6

#define POS_CERRADO 20 // posición del servo cuando está cerrado (°)
#define POS_ABIERTO 110 // posición del servo cuando está abierto (°)

#define TIEMPO_ABIERTA 5000 // 5 segundos apertura
#define LECTURAS_FILTRO 20 // 20 lecturas * 100ms = 2 segundos
#define TIEMPO_LECTURA_PRINC 100L // 100 ms para la lógica principal

// ======> SENSORES DE PLAZAS <======
const int sensores_pines[6] = {16, 33, 32, 19, 18, 23};
const int virtual_pines[6] = {V0, V1, V2, V3, V4, V5};
#define TIEMPO_LECTURA_PLAZAS 1000L // 1 segundo para actualizar LEDs en Blynk
```

Imagen 65. Definición de constantes para placa ESP32.

En el apartado se establecen qué pines se utilizan, los límites del sistema, las posiciones del mecanismo de control y los parámetros de tiempo para el funcionamiento de la lógica.

```
// ======> SENSORES DE PLAZAS <======
const int sensores_pines[6] = {16, 33, 32, 19, 18, 23};
const int virtual_pines[6] = {V0, V1, V2, V3, V4, V5};
#define TIEMPO_LECTURA_PLAZAS 1000L // 1 segundo para actualizar LEDs en Blynk
```

Imagen 66. Sensores a la ESP32.

Se conectan 6 sensores físicos en pines específicos de la ESP32 con 6 pines virtuales de la plataforma Blynk, y define la frecuencia con la que se actualiza esa información.

```
// ======> BLOQUEOS <======
bool bloqueo_manual = false; // activado desde Blynk
bool bloqueo_auto = false; // activado al llenarse el estacionamiento
bool bloqueado = false; // bloqueo total (manual || auto)

// ======> filtros <======
int ultimo_estado_entrada = HIGH;
int contador_entrada = 0;

int ultimo_estado_salida = HIGH;
int contador_salida = 0;

// ======> servo <======
int posicion_actual = POS_CERRADO;
int posicion_objetivo = POS_CERRADO;

// ======> barrera <======
bool barrera_abierta = false;
unsigned long tiempo_apertura = 0;

// ======> carros <======
int carros = 0;
```

Imagen 67. Variables de estado globales.

- **Bloqueos**

`1. bool bloqueo manual`

Indica un bloqueo que puede ser activado manualmente, a través de la interfaz de la aplicación Blynk. Inicialmente está en `false` (desactivado).

`2. bloqueo auto`

Indica un bloqueo automático, que se activa cuando el estacionamiento se llena. Inicialmente está en `false`.

`3. bloqueado`

Indica el estado de bloqueo total del sistema. Se activa si se cumple la condición de bloqueo manual o bloqueo automático. Inicialmente está en `false`.

- **Filtros**

Estas variables se utilizan para implementar un mecanismo de filtro, para asegurar lecturas estables y confiables de los sensores de entrada y salida, evitando falsos positivos debido al ruido o rebotes eléctricos.

`1. int ultimo_estado_entrada`

Almacena el último estado leído del sensor de entrada.

2. `int contador_entrada`

Un contador para el sensor de entrada. Se incrementa o decrementa para confirmar que la lectura ha sido estable durante el tiempo definido.

3. `int ultimo_estado_salida`

Almacena el último estado leído del sensor de salida.

4. `int contador_salida`

Un contador para el sensor de salida, usado para el filtrado.

- **Servo**

1. `int posicion_actual`

Almacena la posición actual del servomotor. Se inicializa a la posición de la barrera cerrada (cuyo valor numérico está definido por la constante `POS_CERRADO`).

2. `int posicion_objetivo`

Almacena la posición deseada a la que el servomotor debe moverse. Se inicializa a la posición cerrada.

```
void actualizar_bloqueo_general() {
  bloqueado = (bloqueo_manual || bloqueo_auto);
  digitalWrite(LED_INDICADOR, bloqueado ? HIGH : LOW);
}
```

Imagen 68. Actualiza el bloqueo general.

Esta función evalúa si el estacionamiento debe considerarse bloqueado combinando los estados del bloqueo manual y automático. Si cualquiera de ellos está activo, se enciende el LED indicador para alertar visualmente al personal o al usuario. Esto garantiza coherencia entre los diferentes modos de bloqueo.

```
void mover_servo() {
  if (posicion_actual < posicion_objetivo) {
    posicion_actual += 4;

    if (posicion_actual > posicion_objetivo) posicion_actual = posicion_objetivo;
  }

  if (posicion_actual > posicion_objetivo) {
    posicion_actual -= 4;

    if (posicion_actual < posicion_objetivo) posicion_actual = posicion_objetivo;
  }

  SERVOM.write(posicion_actual);
}
```

Imagen 69. Movimiento del servo.

El servo se mueve gradualmente hacia su posición objetivo, avanzando en pasos de cuatro grados por ciclo. Este desplazamiento suave evita daños mecánicos y hace que la barrera

opere con menos vibración y ruido, la función revisa si debe moverse hacia adelante o hacia atrás y ajusta la posición hasta alcanzar el objetivo.

```
void abrir_barrera() {
  posicion_objetivo = POS_ABIERTO;
  barrera_abierta = true;
  tiempo_apertura = millis();
}
```

```
void cierre_automatico() {
  if (barrera_abierta && (millis() - tiempo_apertura >= TIEMPO_ABIERTA)) {
    posicion_objetivo = POS_CERRADO;
    barrera_abierta = false;
    Blynk.virtualWrite(V7, "Sin cambios");
  }
}
```

Imagen 70. Apertura y cierre automático de la barra del estacionamiento.

La función `abrir_barrera()` establece la posición objetivo del servo en el ángulo correspondiente a la barrera abierta y registra el momento de apertura. Posteriormente, la función `cierre_automatico()` se encarga de cerrar la barrera después de cinco segundos si un vehículo ya ha pasado. Este mecanismo evita que la barrera quede abierta accidentalmente.

```
void leer_sensores_plazas() {
  for (int i = 0; i < 6; i++) {
    int lectura = digitalRead(sensores_pines[i]);
    Blynk.virtualWrite(virtual_pines[i], lectura);
  }
}
```

Imagen 71. Actualización del estado de las plazas de aparcamiento.

Aquí se actualiza constantemente el estado de cada lugar del estacionamiento. Mediante un ciclo, el sistema revisa los seis sensores HY-024 y envía su lectura como ocupada o libre hacia la plataforma Blynk. Esto permite que el usuario vea en tiempo real qué espacios están disponibles.

```
void controlar_estacionamiento() {
  int lectura_entrada = digitalRead(SENSOR_ENTRADA);
  int lectura_salida = digitalRead(SENSOR_SALIDA);
```

Imagen 72. Lectura de la entrada y salida del estacionamiento.

Esta es la función más importante del programa, pues coordina todo el proceso de entrada y salida de autos.

```
// -------------------- ENTRADA --------------------
if (lectura_entrada == LOW) {
  contador_entrada ++;
} else {
  contador_entrada = 0;
}

if (!bloqueado && contador_entrada >= LECTURAS_FILTRO) {
  if (!barrera_abierta) {
    abrir_barrera();
    Blynk.virtualWrite(V7, "Vehículo ENTRANDO...");
  }

  if (carros < MAX_CARROS) carros++;

  if (carros >= MAX_CARROS) {
    bloqueo_auto = true;
    Blynk.virtualWrite(V8, "Estacionamiento LLENO");
  }

  actualizar_bloqueo_general();

  contador_entrada = 0;
}

ultimo_estado_entrada = lectura_entrada;
```

Imagen 73. Control de la entrada del estacionamiento.

El sistema lee los sensores para identificar la presencia de vehículos. Si uno intenta entrar, se incrementa un contador que actúa como filtro anti-rebote. Cuando este contador alcanza el número requerido, se confirma que no fue un falso disparo del sensor. Si el estacionamiento no está bloqueado, la barrera se abre y el contador de vehículos aumenta. Si llega al límite permitido, se activa el bloqueo automático y se envía un mensaje a Blynk notificando que el estacionamiento está lleno.

```
// -------------------- SALIDA --------------------
if (lectura_salida == LOW) {
  contador_salida++;
} else {
  contador_salida = 0;
}

if (contador_salida >= LECTURAS_FILTRO) {
  if (!barrera_abierta) {
    abrir_barrera();
    Blynk.virtualWrite(V7, "Vehículo SALIENDO...");
  }

  if (carros > 0) carros--;

  if (bloqueo_auto && carros < MAX_CARROS) {
    bloqueo_auto = false;
    Blynk.virtualWrite(V8, "Espacio disponible");
  }

  actualizar_bloqueo_general();

  contador_salida = 0;
}

ultimo_estado_salida = lectura_salida;
```

Imagen 74. Control de la salida del estacionamiento.

El proceso es similar para las salidas, pero en este caso el contador de vehículos disminuye y, si estaba activo el bloqueo automático, este se desactiva. Finalmente, en cada ciclo la función actualiza el servo y la interfaz de Blynk.

```
BLYNK_WRITE(V6) {
  int estado = param.asInt(); // 1: bloquear, 0: desbloquear

  if (estado == 1) {
    bloqueo_manual = true;
    Blynk.virtualWrite(V8, "Entrada BLOQUEADA (Manual)");
  } else {
    bloqueo_manual = false;
    if (bloqueo_auto) Blynk.virtualWrite(V8, "Estacionamiento LLENO");
    else Blynk.virtualWrite(V8, "Entrada DESBLOQUEADA (Manual)");
  }

  actualizar_bloqueo_general();
}
```

Imagen 75. Bloqueo manual desde Blynk.

Se recibe el comando manual de bloqueo desde la aplicación móvil. Si el usuario activa el bloqueo manual, el sistema se bloquea sin importar si hay espacio disponible y se notifica el cambio desde Blynk. Si el usuario lo desactiva, el sistema se desbloquea a menos que el estacionamiento esté completamente lleno.

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

  SERVOM.attach(17);
  pinMode(LED_INDICADOR, OUTPUT);
  pinMode(SENSOR_ENTRADA, INPUT);
  pinMode(SENSOR_SALIDA, INPUT);

  for (int i = 0; i < 6; i++) {
    pinMode(sensores_pines[i], INPUT);
  }

  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, password);

  timer.setInterval(TIEMPO_LECTURA_PRINC, controlar_estacionamiento);
  timer.setInterval(TIEMPO_LECTURA_PLAZAS, leer_sensores_plazas);

  actualizar_bloqueo_general();
}

void loop() {
  Blynk.run();
  timer.run();
}
```

Imagen 76. Funciones `setup()` y `loop()`.

En la función `setup()` se establece la velocidad de transmisión de datos en el monitor serial, así como el modo de los pines de la ESP32 que usamos en el proyecto. Por otro lodo usamos `Blynk.begin(BLYNK_AUTH_TOKEN, ssid, password)` para conectarnos con Blynk.Cloud por medio de una red WiFi. Usamos `timer.setInterval(TIEMPO_LECTURA_PRINC, controlar_estacionamiento)` y `timer.setInterval(TIEMPO_LECTURA_PLAZAS, leer_sensores_plazas)` para mandar a llamar a las funciones que controla el estacionamiento y para monitorear las plazas de aparcamiento cada 100ms y 1s respectivamente.

Mientras que en la función `loop()`, solo iniciamos la función `Blynk.run()`, esta función se llama con frecuencia para procesar los comandos entrantes y realizar el mantenimiento de la conexión Blynk y `Blynk.timer()` para programar las funciones anteriores y evitar que envíen datos constantemente a Blynk para que Blynk.Cloud no corte la conexión.

Alcance

Este proyecto funciona correctamente como prueba de concepto, pues todos los materiales, sensores y dispositivos utilizados para su desarrollo funcionan a menor escala para demostrar su funcionamiento, sin embargo, en caso de que el proyecto se tenga que implementar a gran escala como un prototipo, entonces, gran parte de los elementos utilizados tendrían que ser reemplazados por unos que cumplan con las necesidades del prototipo. A continuación, se plantean algunas ideas sobre algunos de los elementos de la prueba de concepto que se podrían cambiar en caso de querer implementar el proyecto a mayor escala:

Sensores magnéticos.

En un entorno real los sensores magnéticos no son útiles para detectar si un carro esta estacionado, pues no cuentan con algún elemento magnético en la parte inferior de estos como para ser detectados por el sensor. Por esa razón, para remplazar los sensores magnéticos lo que podríamos hacer es implementar placas de presión exclusivas para carros en cada lugar del estacionamiento, de esta forma el dispositivo solo detectará los vehículos. Al quitar los sensores magnéticos, también se eliminan los imanes utilizados en los vehículos de juguete de la prueba de concepto.

Como segunda propuesta, podríamos implementar sensores de campo magnético (Magnetómetros) para detectar la alteración del campo magnético de la tierra cando un metal o en este caso el carro se pone encima del sensor.

Sensores infrarrojos.

Del mismo modo que con los sensores magnéticos, los sensores infrarrojos presentarían algunas deficiencias en caso de querer implementar el proyecto en un espacio físico a escala, pues la principal falla del sensor infrarrojo es que detecta cualquier cosa u objeto que se ponga delante de este y también presenta dificultades para detectar objetos que son muy oscuros. Para resolver estas problemáticas, la idea que se propone es implementar cámaras especiales que detecten únicamente a los vehículos antes de que entren al estacionamiento y antes de que se levante la plumilla, de este modo nos aseguraríamos de que solo se lleve a cabo el conteo de los vehículos que ingresan y salen del estacionamiento.

Mapa.

De momento esta funcionalidad no fue implementada durante la prueba de concepto, pero podría agregarse en caso de realizar el proyecto en un entorno real. La idea es usar el widget Mapa en la plantilla del proyecto para mostrar la ubicación del estacionamiento y de esta forma entender cómo se encuentran distribuidos las plazas de aparcamiento, así como la salida y entrada de este. Esto puede facilitar la movilidad en la zona y evitar perder tiempo tratando de ubicarse en el estacionamiento.

Para el mapa es necesario que el dispositivo que esté conectado a la plantilla de Blynk envíe su ubicación usando coordenadas GPS (latitud y longitud) que se envía desde el firmware, usando un módulo o una API para obtener las coordenadas. Las coordenadas se envían a Blynk usando pines virtuales para que el widget los lea y ponga una marca en esas coordenadas y actualizándose cada que se reciban coordenadas nuevas.

Red de estacionamientos.

Al igual que hicimos una interfaz para representar el estacionamiento de la prueba de concepto, podríamos crear diferentes interfaces personalizadas para cada caso o tipo de estacionamiento en el que se quiera implementar el proyecto.

Sistema de usuarios basado en roles.

Por el momento, solo el administrador puede ver las plantillas en Blynk y la información enviada desde el estacionamiento, sin embargo, Blynk maneja un sistema de usuarios basado en roles. El administrador tiene control total sobre los dispositivos, plantilla y widgets, mientras que los usuarios invitados a pesar de compartir la misma plantilla y ver los dispositivos de este, nosotros como administrador podemos limitar las acciones de los invitados. Para el caso del bloqueo manual de la entrada del estacionamiento, podríamos crear una plantilla exclusiva para los invitados donde no se

incluya el switch y que el permiso sea de solo lectura para que no pueda modificar nada y una para el administrador igual a la que se explicó anteriormente.

De esta forma nosotros podemos tomar el rol de Publisher mientras que los invitados el rol de clientes, sin embargo, para poder acceder a este sistema se requiere contar con alguno de los planes de pago que ofrece Blynk.

Aprovisionamiento Wi-Fi.

Originalmente, se tenía planeado usar la biblioteca `Blynk.Edgent.h` para que desde la aplicación móvil de Blynk se puedan escanear los dispositivos cercanos y las redes WiFi cercanas, de esta forma nosotros podríamos elegir que red WiFi usar para conectarnos con el estacionamiento, sin embargo, resulto ser un poco confuso, ya que el código de Arduino consiste en 10 archivos. Al final esta funcionalidad se terminó descartando.

Además, otra de las funcionalidades que podríamos implementar a futuro seria la de un sistema de bloqueo en caso de que la conexión a la red WiFi sea inestable o se pierda la conexión completamente. Si esto llegará a ocurrir en determinado momento, el sistema sería capaz de bloquear el estacionamiento hasta que se vuelva a establecer la conexión automáticamente, esto significa que ningún vehículo podrá acceder al estacionamiento, pero si se permitirá la salida de estos siempre y cuando estén dentro de la zona.

ESP32.

Como ya se mencionó antes, las capacidades de un solo dispositivo ESP32 son muy limitadas y en caso de que queramos pasar el proyecto a un entorno real, tendríamos que utilizar múltiples dispositivos ESP32 para controlar una mayor cantidad de sensores al mismo tiempo. A su vez, cada dispositivo ESP32 tendría que estar conectado vía Wi-Fi a un Raspberry PI que se va a encargar de recopilar la información de cada dispositivo ESP32. De esta forma, dependiendo de las dimensiones del estacionamiento, podríamos utilizar una cantidad n de dispositivos y sensores para escalar el proyecto.

Aplicación Blynk.

La Aplicación Blynk presento muchas limitaciones al momento de querer utilizar ciertas funcionalidades de esta, pues su versión gratuita ofrece lo minimo para presentar una prueba de concepto como en nuestro caso, sin embargo, debido a su alto costo tendríamos que remplazar la aplicación para recopilar los datos y mostrarlos respectivamente dentro de una interfaz. El remplazo ideal sería utilizar una Arquitectura más compleja para controlar todo lo que hacía Blynk, en este caso, usaríamos el siguiente modelo:

- **MQTT:** Blynk usaba el protocolo TCP/IP para la transferencia de los datos desde el ESP32 hasta la aplicación, por lo que usar el protocolo MQTT sería la mejor opción para remplazar esta funcionalidad. Es importante mencionar, que para usar el protocolo MQTT se necesitaría un Broker como Mosquitto para recibir y enviar los mensajes.
- **Node-RED:** Para interpretar los mensajes enviados desde el ESP32 mediante MQTT, es necesario un intermediario como Node-RED, este decidiría que hacer dependiendo del mensaje que se envía desde el ESP32 y realizaría los cálculos necesarios para mostrarlos en una interfaz gráfica.

- **ThingsBoard y Grafana:** Tanto ThingsBoard como Grafana son softwares ideales para visualizar los datos dependiendo de lo que queramos mostrar. Con Grafana podríamos analizar los datos y mostrarlos en diferentes gráficos para mejorar la gestión del estacionamiento. En el caso de ThingsBoard, podríamos utilizarlo como panel de control en tiempo real para operar el lugar, crear mapas interactivos y gestionar diferentes dispositivos como los ESP32 de todo el estacionamiento.

Conclusión

La plataforma Blynk resulta ser una muy buena opción para aplicaciones IoT a pesar de contar con el plan gratuito, pues proporciona herramientas y mecanismos suficientes que facilitan la creación de la interfaz de usuario, la conexión del hardware con la nube, la visualización de datos, control y monitoreo remoto.

Una de las herramientas que proporciona Blynk son los pines virtuales, canales que permiten intercambiar datos entre el estacionamiento y Blynk. De las ventajas de los pines virtuales, se encuentra el hecho de que no tienen relación con los GPIOS físicos de la ESP32 o de cualquier otro dispositivo lo que hace más sencillo portar el código entre plataformas. También proporcionan más control sobre las funciones de los widgets de la plantilla en Blynk y por supuesto sobre los GPIOS físicos y por último facilitan la visualización de los datos leídos y enviados a través de estos canales en la interfaz creada en Blynk.

Uno de los mecanismos es el HERTBEAT, que se encarga de actualizar el estado en línea del dispositivo en tiempo real, en donde la ESP32 envía ping comandos periódicamente con una frecuencia predefinida (cada 45 segundos). Si la ESP32 agota estos 45 segundos, el servidor de Blynk (Blynk.Cloud) espera 103 segundos más, si también se agotan entonces se asume que la conexión se ha interrumpido y por tanto en la interfaz se actualiza el estado de la conexión. Este mecanismo resulta muy útil, pues no ayuda a saber si el estacionamiento está funcionando correctamente o si está fallando en este caso presentándose un retraso en el envío de ping comandos.

Y por último lo más importante, la API de Blynk, aquella que se encarga de todo los relacionado con la conexión entre la ESP32 y Blynk. Cuenta con la función Blynk.virtualWrite(vPin, valor), para enviar datos a través de los pines virtuales a Blynk y BLYNK_WRITE(vPIN) para recibir actualizaciones del valor de un pin virtual, también cuenta con una biblioteca para el aprovisionamiento WiFi, y no menos importante también es compatible con varias placas, plataformas y protocolos.

Debido a estas herramientas y sistemas, es posible llevar el proyecto a gran escala. Esto lo podríamos lograr haciendo uso del sistema de usuarios basado en roles con el que cuenta Blynk, que consiste en invitar a varios usuarios a la organización, para que puedan interactuar con las plantillas y dispositivos. Con esto podríamos tomar el rol de Publisher, aquellos que distribuyen la información sobre el estacionamiento, y los invitados el rol del cliente, pues podrían interactuar con dicha información, además esto toma fuerza si tomamos en cuenta la creación de una red de estacionamientos, en donde nosotros replicamos el proyecto a varios

estacionamientos, proporcionándole a los clientes información sobre el estacionamiento correspondiente.

Bibliografía

Blynk. (s.f.). *Blynk Protocol*. Obtenido de BLYNK.DOCS: https://docs.blynk.io/en/blynk-library-firmware-api/blynk-protocol

Blynk. (s.f.). *Blynk.Edgent overview*. Obtenido de BLYNK.DOCS: https://docs.blynk.io/en/blynk.edgent/overview

Blynk. (s.f.). *Device Online/Offline Status*. Obtenido de BLYNK.DOCS: https://docs.blynk.io/en/blynk-library-firmware-api/devices-online-status

Blynk. (s.f.). *Manual Device Activation*. Obtenido de BLYNK.DOCS: https://docs.blynk.io/en/getting-started/activating-devices/manual-device-activation

Blynk. (s.f.). *Send Data From Hardware To Blynk*. Obtenido de BLYNK.DOCS: https://docs.blynk.io/en/getting-started/how-to-display-any-sensor-data-in-blynk-app

Blynk. (s.f.). *Set Up Datastreams*. Obtenido de BLYNK.DOCS: https://docs.blynk.io/en/getting-started/template-quick-setup/set-up-datastreams

Blynk. (s.f.). *Set Up Mobile App Dashboard*. Obtenido de BLYNK.DOCS: https://docs.blynk.io/en/getting-started/template-quick-setup/set-up-mobile-app-dashboard

Blynk. (s.f.). *Set Up Web Dashboard*. Obtenido de BLYNK.DOCS: https://docs.blynk.io/en/getting-started/template-quick-setup/set-up-web-dashboard

Blynk. (s.f.). *Users*. Obtenido de BLYNK.DOCS: https://docs.blynk.io/en/blynk.console/users

Blynk. (s.f.). *Virtual Pins*. Obtenido de BLYNK.DOCS: https://docs.blynk.io/en/blynk-library-firmware-api/virtual-pins

ELECTRONICS, U. (s.f.). *ESP32 DEVKIT V1 30 PINES USB-C/MicroUSB*. Obtenido de UNIT ELECTRONICS TIENDA DE COMPONENTES ELECTRÓNICOS: https://uelectronics.com/producto/esp32-devkit-v1-30-pines-usb-c-microusb/

ELECTRONICS, U. (s.f.). *Seguidor de Linea TCRT5000 Optico Infrarrojo*. Obtenido de UNIT ELECTRONICS TIENDA DE COMPONENTES ELECTRÓNICOS: https://uelectronics.com/producto/sensor-seguidor-de-linea-tcrt5000-optico-infrarrojo/

ELECTRONICS, U. (s.f.). *Sensor de Campo Magnético KY-024*. Obtenido de UNIT ELECTRONICS TIENDA DE COMPONENTES ELECTRÓNICOS: https://uelectronics.com/producto/sensor-de-campo-magnetico-ky-024/