# Computación Ubicua

Grado en Ingeniería Informática

Universidad de Alcalá

Marcos Tajuelo García (09086043P)

Adrián Ajo Matarredona (53757353M)

Marcos Navarro Juan (05324619G)

Junio 2022-2023

# Contenido

Resumen....3
1. Introducción....4
2. Análisis del problema....4
3. Objetivos y alcance del proyecto....4
A corto plazo....4
A medio plazo....5
A largo plazo....5
Alcance....5
4. Ideas descartadas....5
5. Desarrollo....7
5.1. Tecnologías a utilizar....7
5.2. Arquitectura del Proyecto....7
5.2.1. Introducción a la Arquitectura....7
5.2.2. Capa de Percepción....8
5.2.2.1. Sensores y Componentes Electrónicos....8
5.2.2.2. Programas en Arduino....12
5.2.3 Capa de Red....14
5.2.4 Capa de Procesado....15
5.2.4.1 Algoritmo Semáforos Python....15
5.2.4.2 Conexión MQTT....16
5.2.4.3 Node.js....18
5.2.4.4 Bases de Datos....26
5.2.4.5 Ciencia de Datos Obtenidos....31
5.2.5 Capa de aplicación....36
5.2.5.1 Aplicación Web....36
5.3. Plan de desarrollo....42
6. Conclusiones....42
7. Bibliografía....43
Anexo I – Manual de instalación....44
Archivos Arduino....44
Archivo Python....46
Archivo Red Neuronal....46
Base de Batos Firebase....46
Servidor e Interfaz web....47
Anexo II – Manual de Usuario de la aplicación Web....48
Anexo V – Hojas de características de los componentes....50

# Resumen

En este documento vamos a mostrar todo lo relacionado con nuestro proyecto, desde la razón de la realización del proyecto hasta la forma, arquitectura y tecnologías detalladas que vamos a utilizar para la realización del mismo, así como la organización que seguiremos para realizarlo.

Primero empezamos con una breve introducción en la cual detallamos en qué consiste el proyecto. Seguido de esto explicaremos los problemas que se dan en las vías públicas para los cuales propondremos soluciones, así como los objetivos que nos marcamos con este proyecto a corto, medio y largo plazo.

También detallaremos las ideas descartadas que hemos tenido y las razones por las que las descartamos, y por último todo lo relacionado con la arquitectura del proyecto y la planificación interna del mismo.

Además, añadimos la bibliografía al final del documento.

# 1. Introducción.

En la actualidad la gestión de las grandes ciudades supone un gran reto, existen una serie de problemas que están intrínsecos a todas las grandes ciudades, como puede ser la masiva población, la contaminación acústica, la gestión de residuos, … La manera en la que está gestionada la ciudad puede suponer grandes mejoras en la salud de la población, tanto mental como física.

Existen diferentes perspectivas a la hora de gestionar una ciudad. En toda empresa, vivienda, comunidad la hora de gestionar los recursos es fundamental para desempeñar un buen desarrollo. Una buena gestión de los recursos de la ciudad puede suponer una gran diferencia en el medio ambiente, en la calidad de vida de las personas y en la economía. Por todos estos factores, se busca crear un sistema tecnológico que permita a cualquier ciudad incrementar la eficiencia de sus recursos incidiendo en aspectos públicos, como puede ser la vía pública.

# 2. Análisis del problema

El objetivo del proyecto es gestionar los recursos de una manera más inteligente y otorgar mayor calidad de vida a la población.

Para ello, se ha determinado una serie de factores que se pueden gestionar y controlar de mejor forma. Las ciudades se componen de vías públicas, por lo tanto, se ha enfocado el proyecto hacia esa dirección

Se ha analizado la vía pública de las ciudades y se presentan los siguientes problemas:

1. Funcionamiento, en situaciones ilógicas de los semáforos: Se ha analizado que los semáforos no se actualizan a las necesidades cambiantes de la vía pública. Los semáforos convencionales funcionan a partir de una programación básica, donde van cambiando de estado a partir de una cantidad de tiempo determinada. Esto supone que se puedan generar atascos.

2. Colisiones o atropellos debido a la poca visibilidad o a la falta de señalización para avisar a los ciudadanos de posibles situaciones de riesgo.

# 3. Objetivos y alcance del proyecto

Se establecen los siguientes objetivos y envergadura del proyecto:

## A corto plazo.

- Maquetar un prototipo funcional del sistema.
- Desarrollo de servidor propio, en el que se almacena toda la información del sistema recogida por los distintos sensores
- Desarrollo de nuestra propia Web, donde se muestra la información almacenada en el servidor.
- Búsqueda de nuevas fuentes de financiación para el desarrollo del proyecto.

- Establecer un área de trabajo, una oficina, para poder desarrollar toda la arquitectura del sistema, desde los sensores hasta la creación de la página web.

### A medio plazo

- Implementación del sistema en diferentes barrios de la ciudad.
- Realizar una ampliación del personal de trabajo.
- Análisis de la eficiencia del sistema e implementar mejoras en función de su necesidad y viabilidad.
- Exponer el producto realizado a otras ciudades y así ampliar nuestra área de negocio.
- Búsqueda de un proveedor que nos suministre periódicamente los componentes necesarios de nuestro sistema.
- Desarrollo de un sector de marketing en la propia empresa, que mejore las relaciones con los clientes y permita desarrollar nuestra arquitectura en nuevos lugares.

### A largo plazo

- Invertir en I+D para analizar más sectores que se puedan mejorar utilizando dispositivos ubicuos.
- Expandir el mercado de trabajo actual gracias a la inversión en I+D.
- Expandirse tanto como empresa para poder salir del ámbito nacional y llegar a diferentes países.

### Alcance.

Realizando un estudio de mercado, se ha comprobado que la implementación de arquitectura como la que planteamos o las llamadas “Smart Cities”, tienen una presencia escasa o no están planteadas de una manera eficiente.

Por lo cual, con nuestro producto podemos introducirnos en un mercado con muy pocos años de vida y que va a suponer un aspecto fundamental en cualquier ciudad en un futuro muy próximo.

## 4. Ideas descartadas

Durante el diseño de la arquitectura del sistema se fueron planteando una inmensa cantidad de ideas, a partir del método Brainstorming, que no fueron aceptadas por distintos motivos, obteniendo el prototipo final de la arquitectura.

Las ideas descartadas y sus correspondientes motivos son las siguientes:

- Sensor de temperatura en la vía pública que detecte si hay hielo. En primera instancia se consideró la implementación del sensor inteligente “Lufft IRS31Pro-UMB”, el cual es capaz de detectar la temperatura de la superficie de la carretera y su estado, por ejemplo, de la existencia de hielo en la carretera. Incluso este sensor es capaz de detectar a qué temperatura se va a producir el grado de congelación. La idea fue descartada en la etapa de investigación, debido al alto precio del sensor y su falta de disponibilidad, ya que es relativamente nuevo y esto dificulta su disponibilidad. También se prefirió dar prioridad en cuestiones económicas y de tiempo a otras funcionalidades.

- Sensor de visibilidad para detectar diferentes elementos que pueden dificultar la visión. Para desempeñar este trabajo se pensó en el “CS125”, el cual es un sensor de visibilidad y tiempo presente. Este sensor se caracteriza por poder detectar qué tipo de partículas se encuentran en ese instante en el área de trabajo y la cantidad. Por lo cual también se podría utilizar este mismo sensor para detectar la cantidad de CO2 en el aire, como indican las especificaciones. Esta idea se descartó, a pesar de aportar gran funcionalidad, debido a que el sensor en el mercado supera el coste presupuestado.

- Sensor de ultrasonido para detectar la proximidad implementada en los propios vehículos. El sensor que se tuvo en cuenta y que era idóneo para desempeñar la función fue el “SENO0280”. Se quería implementar este tipo de sensores en todos los coches en circulación para que cuando el sensor se activará, porque detecta un obstáculo, mandará la información al conductor en modo aviso o que directamente detuviera el coche. Esta idea se descartó a pesar de que el sensor era idóneo para implementación teniendo en cuenta la disponibilidad y el precio, debido a que es inviable obligar a cada propietario del vehículo a implementar este tipo de sensor. También se saldría de la idea del proyecto, de realizar una vía pública inteligente y trabajar con los datos.

- La implementación de un GPS para la monitorización de los vehículos en tiempo real. Se enfoca esta idea para que, con la información recogida, implementar algoritmos de aprendizaje automático, que pudieran detectar cuándo se produciría un nuevo atasco, colisiones o incidencias en la vía. Se descartó la idea, por el mismo problema que la idea anterior, que es inviable obligar a cada propietario del vehículo a implementar el GPS, y por protección de datos.

- Tira de presión para encender las farolas cuando se detectan los vehículos. Para mejorar la eficiencia energética y no tener las farolas encendidas todo el tiempo sin necesidad. Se pensó en utilizar tiras de presión para detectar cuando un vehículo

pasaba por cierta zona, y en el instante de después iluminar. Las tiras de presión se adecuaban a nuestro proyecto por el precio y por el nivel de implementación. Se rechazó esta idea, por la implementación de sensores LDR, que consisten en detectar si hay algo de luz. Cuando el sensor no detecte la luz suficiente, empezará a ser de noche por lo que indicará a la farola de encenderse, el funcionamiento contrario cuando amanezca. Los sensores LDR son aún más asequibles teniendo en cuenta dinero y disponibilidad.

- Aplicación móvil con Android. En una primera instancia se pensó en la realización de una aplicación móvil desarrollada con Android Studio para representar la información recogida y que el usuario pudiera accionar con ella. Se descartó porque se pensó que con representar la información a través de una página Web era suficiente, no es necesario implementar una aplicación Web, ya que apenas daba funcionalidad al sistema y podría sumar un tiempo considerable al desarrollo.

# 5. Desarrollo

## 5.1. Tecnologías a utilizar

- Sensor de presión.
- Leds.
- Resistencias.
- ESP8266.
- MQTT.
- EMQX.
- Firebase.
- Firestore.
- Node.js.
- Brain.js.
- TypeScript.
- React.

## 5.2. Arquitectura del Proyecto

### 5.2.1. Introducción a la Arquitectura.

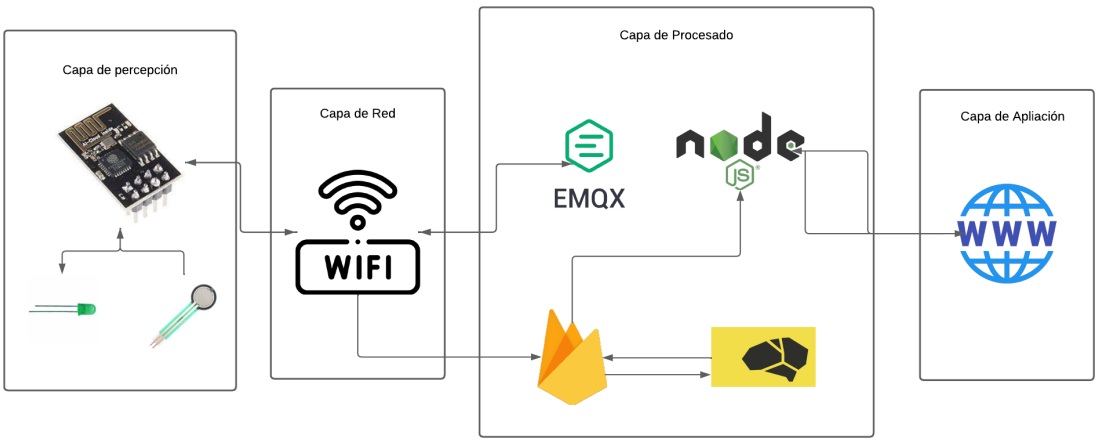



*Figura 1. Esquema de la Arquitectura del Proyecto.*

En la capa de percepción utilizamos los sensores de presión para obtener información. Esta información es enviada a Firebase.

Además, mediante MQTT conectamos la ESP8266 con un archivo .py que recibe datos de Arduino, y tras analizarlos devuelve una respuesta, la cual activa los leds correspondientes.

Los datos almacenados en firebase se utilizan para entrenar la red neuronal de brain.js. Tras ser entrenada, esta red neuronal genera predicciones, las cuales son almacenadas en Firestore, ya que no se necesita una base de datos en tiempo real.

El servidor, creado con node.js obtiene los datos de las bases de datos y se los proporciona a la aplicación web cuando estos son requeridos desde la interfaz.

Por último, los datos se muestran en la interfaz.

### 5.2.2. Capa de Percepción.

En nuestro proyecto se crea una vía pública inteligente. La capa de percepción tiene el objetivo de obtener toda la información, que nosotros consideramos importante, perteneciente a la vía pública, donde más tarde la información obtenida es de vital importancia para el resto del proyecto. Esta capa de percepción detecta los cambios y obtiene medidas a través de los sensores y actúa en consecuencia.

En nuestro proyecto se consideran dos escenarios, con sus correspondientes sensores y demás componentes electrónicos.

Los escenarios que se consideran en nuestro proyecto son:

- Un paso de peatones con poca visibilidad.
- Un cruce de carretera para el cual queremos agilizar la circulación dependiendo de la cantidad de vehículos que haya en un lado u otro del cruce.

Con toda la información recabada se almacena en fuentes de datos para el posterior tratamiento de datos y también actúa en tiempo real sobre la vía produciendo cambios de vital importancia.

#### 5.2.2.1. Sensores y Componentes Electrónicos.

Como anteriormente se ha expuesto, en nuestro proyecto se tienen en cuenta dos posibles escenarios, los cuales utilizan los siguientes componentes electrónicos.

#### *5.2.2.1.1. Primer Escenario*

Para el primer escenario, el paso de peatones se ha utilizado un sensor de presión, exactamente el sensor "*MF01-N-221-A01*" y un led que se activa cuando el sensor se ha activado. El sensor tiene el objetivo de detectar cuando una persona pasa por el paso de peatones, encendiendo el led y contrarrestando la vía con poca visibilidad. Haciendo la presencia del peatón notable tanto para los vehículos como para los demás viandantes.

Se ha escogido este el sensor de presión "*MF01-N-221-A01*", debido a que tiene un bajo coste, es sencillo de utilizar, requiere de menos de 1mA para funcionar y está perfectamente adaptado para el funcionamiento en proyectos de Arduino. Aunque al ser un sensor de bajo coste su precisión en algunos momentos puede ser baja.

*Figura 2. Sensor de Presión MF01-N-221-A01.*

Tanto el sensor como el led, como la protoboard están conectadas a una ESP8266, para poder realizar el programa descrito en Arduino.

También se ha utilizado dos tipos de resistencias:

- La resistencia de 330 Ohmios para el funcionamiento del led.

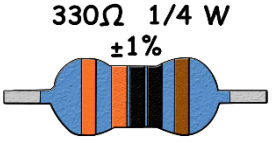



*Figura 3. Resistencia 10k Ohmios.*

- La resistencia de 10k Ohmios para el funcionamiento del sensor de presión.


*Figura 4. Resistencia 10k Ohmios.*

#### *5.2.2.1.2 Segundo Escenario*

Para el segundo escenario, la intersección, se ha utilizado cuatro sensores de presión, seis leds diferentes que se caracterizan por se de color verde, amarillo y rojo simulando los colores de

un semáforo y una protoboard que conecta todos los componentes y otra ESP8266 realizar el programa correspondiente. Esta vez los sensores de presión utilizados son los sensores de presión FRS 402. Este tipo de sensor comparte las características descritas anteriormente del otro sensor. Se ha escogido esta vez estos sensores debido a su disponibilidad.

*Figura 5. Sensor de Presión FRS 402.*

Los cuatro sensores de presión se utilizan por parejas, poniendo dos sensores en cada tramo. Uno al principio del tramo y otro al final de este. Se dispone de esta forma para saber el número concreto de coches que hay en cada tramo, que es de vital importancia para determinar el funcionamiento de los semáforos.

Entonces dos de estos sensores, más tres leds de colores verde, amarillo y rojo con sus respectivas resistencias componen el tramo 1. Y la misma composición para el tramo 2.

#### *5.2.2.1.3 ESP8266.*

El ESP8266 es un módulo de conectividad Wi-Fi muy popular que se utiliza en aplicaciones de Internet de las cosas (IoT). Fue desarrollado por la compañía china Espressif Systems.

El módulo ESP8266 contiene un microcontrolador con potencia de procesamiento y memoria y un chip Wi-Fi que permite la comunicación inalámbrica. El microcontrolador es compatible con el lenguaje de programación Arduino. También es compatible con implementaciones de servidores web y puede comunicarse con otras placas o dispositivos a través de protocolos de comunicación como HTTP, MQTT y TCP/IP.

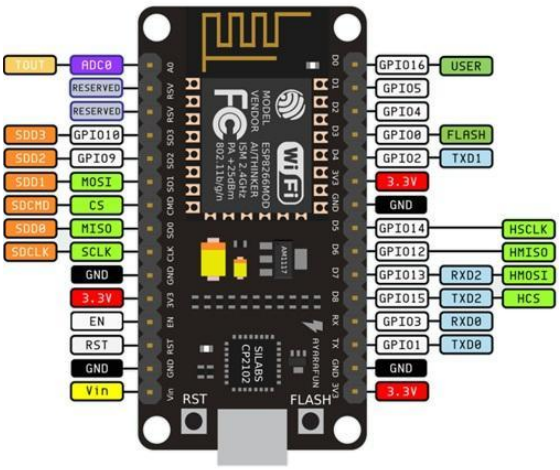



*Figura 6. ESP2866 con los pines detallados.*

Para nuestro prototipo se utilizan dos placas ESP8266. Pese a que estamos utilizando sensores de presión que transmiten entradas analógicas, los estamos utilizando como digitales debido a que las ESP8266 solo tienen un pin de entrada analógica. Podemos hacer esto ya que solo

necesitamos registrar cambios de presión en los sensores y no medir una presión concreta, ni medir la fuerza de la presión con precisión.

Sin embargo, en la ESP8266 en la que corre el programa de los semáforos sí que utilizamos uno de los sensores de presión como analógico debido a la cantidad de pines disponible.

A continuación, se muestran dos esquemas de los dos circuitos implementados en nuestro proyecto, para la realización del esquema se ha utilizado el simulador web Tinkercad.

En este caso se utiliza una placa Arduino uno como sustitución debido a que no existe la ESP8266 en el simulador, sin embargo, los sensores y leds están conectados de la misma forma, con distribución entre analógicos y digitales.

#### *5.2.2.1.4 Esquemas.*

Esquema funcionamiento del Paso de Peatones:

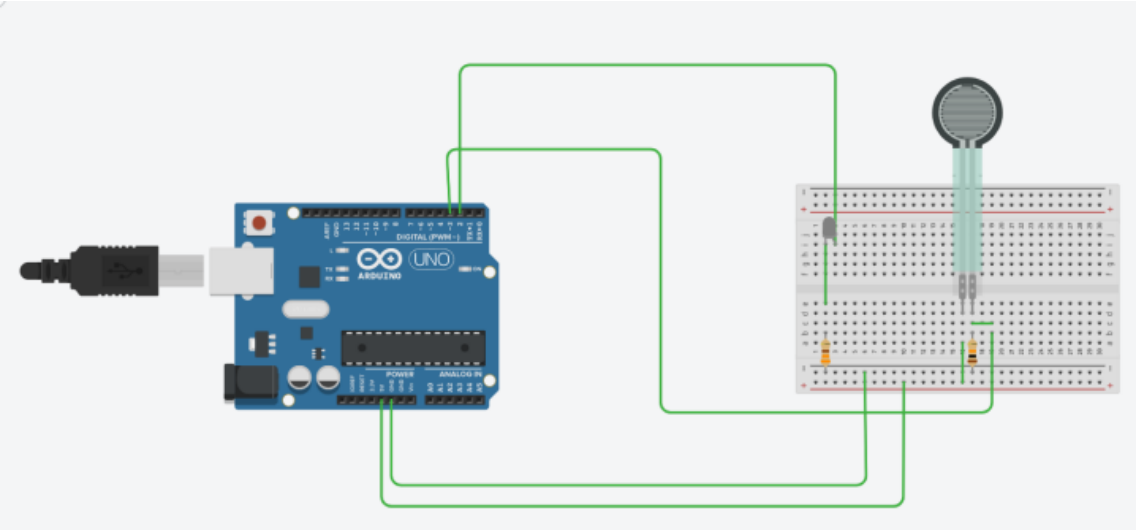


*Figura 7: Circuito del Paso de Peatones*

El sensor de presión se conecta al pin D1 de la ESP8266 que como muestra en la figura corresponde al 5 en Arduino. Después el led se conecta al pin D2, que se corresponde con el 4.

Esquema del funcionamiento de los Semáforos:

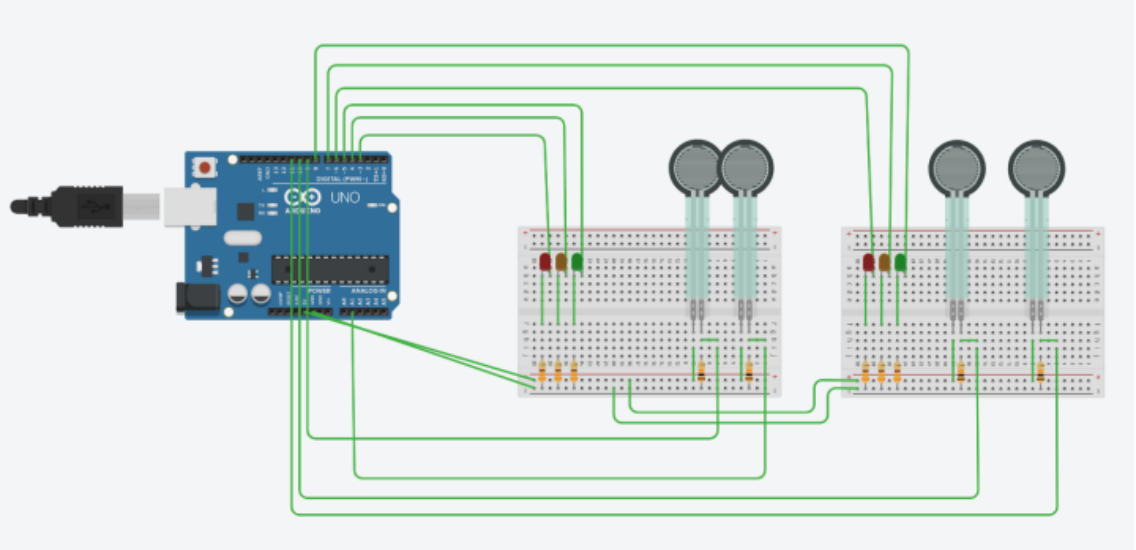


*Figura 8: Circuito de Semáforos*

Para los sensores de presión del tramo 1 los conectamos a los pines D1 y D2 de la ESP8266, que corresponden en el Arduino con el 5 y 4 respectivamente, como se puede observar en la figura 5. Después los leds del semáforo se conectan a los pines D3, D4 y D5 que se corresponden al 0, 2 y al 14 respectivamente.

Para el segundo tramo, los dos sensores de presión se conectan a los pines D7 y A0, siendo a A0 un pin analógico. Que se corresponden a como se puede observar en la figura 5 al 13 y A0. Para los leds se conectan a los pines D6, D0 y D8, que se corresponde a 12, 16 y 15.

### 5.2.2.2. Programas en Arduino.

Al tener dos escenarios diferentes con dos ESP8266, se ha de tener dos programas diferentes en Arduino para cargar el correcto en cada ESP8266.

Para la realización de los dos programas es necesario importar las siguientes librerías:

- EspMQTTClient by Patrick Lapointe: Esta librería permite tanto enviar mensajes a través de MQTT como recibirlos.
- PubSubClient by Nick O`Leary: Es otra librería que se apoya en EspMQTTClient que permite la comunicación mediante el protocolo MQTT.
- Firebase Arduino Client: Esta librería permite agregar información a las bases de datos de Google (Firebase).
- Arduino.Json by Benoit Blanchon: Esta librería permite almacenar la información siguiendo un formato json.
- NTPClient by Fabrice Weinberg: Esta librería permite obtener y sincronizar la hora de un servidor NTP. Evitando trabajar con delays y controlando la información con tiempo real.

Se evita la utilización de la función del Arduino *delay()*, para que los sensores recaben información en todo el proceso.

#### *5.2.2.2.1 Paso de Peatones.*

Para el paso de peatones utilizamos un sensor de presión que determina si hay un peatón esperando para cruzar la calle. En caso de que lo haya se enciende un led, simbolizando una señal luminosa para avisar a los conductores. Cada vez que se presiona el sensor, se envía a la base de datos Firebase la información.

#### *5.2.2.2.2 Semáforos.*

Lo primero que se instancia en las diferentes propiedades para poder mandar información a la base de datos Firebase como la información para poder utilizar el protocolo MQTT.

Las primeras funciones que se definen son las funciones *callback()* y *reconnect()*, que se utilizan para poder recibir los mensajes a través del MQTT y por si se pierde la conexión con el MQTT volver a conectarla.

Después en la función *setup()*, se declara todos los sensores y leds a utilizar y se produce la conexión tanto con la base de datos Firebase como con el protocolo MQTT

En la principal *loop()*, es donde se lleva a cabo todo el programa. Lo primero que se realiza es contabilizar el número de coches que hay en cada tramo, para ello se utilizan dos funciones auxiliares *enviarCoches1()* y *enviarCoches2()*.

Estas funciones lo que hacen es comprobar si los sensores del tramo en cuestión están activados, si el primer sensor de tramo se activa entonces se suma 1 a los coches que pasan por ese tramo. Si es el sensor de presión 2 el que se activa se resta 1 a los coches. Obteniendo así el número de coches que hay en cada tramo. Una vez calculado el número de coches de cada tramo esta información es enviada a un archivo Python a través del protocolo MQTT. Las dos funciones realizan esta misma funcionalidad, pero cada uno para su tramo, obtenido en el archivo Python cuántos coches hay en cada tramo en ese instante de tiempo.

```
//Se envian al MQTT los coches que hay en el primer Tramo
void enviarCoches1(int dia_semana){

  int comprobar1 = digitalRead(sensor1);
  int comprobar2 = digitalRead(sensor2);
  //Si el primer sensor se activa
  if(comprobar1 == HIGH){
    Serial.println("Se ha Pulsado el SENSOR 1 del TRAMO 1");
    coches1++ ;//Se suma 1 a los coches que pasan
  if (comprobar2 == HIGH){
    Serial.println("Se ha Pulsado el SENSOR 2 del TRAMO 1");
    coches1--;
  }
  if (coches1 < 0) {
    coches1 = 0;
  }
  itoa(coches1, buffer1, 10);  // Convertir el número entero a cadena
  const char* cadena = buffer1;  // Asignar la cadena a un puntero const char*
  // Envía un mensaje MQTT al tópico especificado con el contenido deseado
  client.publish(publishTopic, cadena);
  Serial.println("[MQTT] Se envia la informacion al MQTT");
  Serial.println("En el tamo 1 hay ");
  Serial.println(coches1);
  Serial.println("-----------------------");
}
```

*Figura 9: Función enviarCoches1()*

No se podrá enviar nuevamente información a Python a través de MQTT hasta que el mensaje anterior haya sido procesado y se haya recibido la respuesta en el Arduino.

Al archivo Python, le llega cuántos coches hay en ese momento en cada tramo, entonces el archivo Python envía como ha de ser los colores de los dos semáforos al Arduino a través del protocolo MQTT.

Una vez que el coche haya pasado por los dos sensores se mandará la base de datos Firebase que ha pasado un coche por ese tramo, indicando el tramo en cuestión, el día del mes y la fecha del suceso.

Entonces en la función *loop()*, se comprueba si le ha llegado este mensaje al Arduino, si es así se comprueba el valor del mensaje.

Si el mensaje es "igual", significa que el número de coches en el tramo 1 y 2 es el mismo o que no hay no hay coches pasando en ese momento. Entonces los colores de los semáforos actuarán en función de un valor predeterminado. Se llama a la función *algoritmoPredeterminado(),* el cual a través de un contador va enciendo y apagando los leds pertinentes para sin provocar ninguna situación anómala.

```
void algoritmoPredeterminado(){
  digitalWrite(VERDE1, LOW);
  digitalWrite(NARANJA1, LOW);
  digitalWrite(ROJO1, LOW);
  digitalWrite(VERDE2, LOW);
  digitalWrite(NARANJA2, LOW);
  digitalWrite(ROJO2, LOW);

  // Encender el LED correspondiente al estado actual
  if (estado == 0) {
    digitalWrite(VERDE1, HIGH);
    digitalWrite(ROJO2, HIGH);
  } else if (estado == 1) {
    digitalWrite(NARANJA1, HIGH);
    digitalWrite(ROJO2, HIGH);
  } else if (estado == 2) {
    digitalWrite(ROJO1, HIGH);
    digitalWrite(VERDE2, HIGH);
  }
  else if (estado == 3) {
    digitalWrite(ROJO1, HIGH);
    digitalWrite(NARANJA2, HIGH);
  }

  // Incrementar el estado
  estado++;

  // Si el estado alcanza 3, reiniciar a 0
  if (estado == 4) {
    estado = 0;
  }
```

*Figura 10: Algoritmo Predeterminado*

Si el mensaje no es "igual", entonces es que el archivo Python indica que color de los dos semáforos tiene que están On. Por ejemplo, si le llega el mensaje "r1v2", se apagarían todos los leds y a continuación se encenderá el led rojo del semáforo 1 y el led verde del semáforo 2.

A través del loop se va comprobando si se pulsan los sensores, mandando y recibiendo información para encender los semáforos.

### 5.2.3 Capa de Red.

Para el proyecto es necesario establecer comunicaciones entre los diferentes dispositivos en una red. Estas comunicaciones se establecen en esta capa a través de la tecnología inalámbrica Wi-Fi.

En el proyecto utilizamos esta tecnología para establecer comunicaciones entre la capa de percepción, más concretamente con ESP8266, y las diferentes tecnologías de la capa de procesado. Las comunicaciones ha destacar son las siguientes:

- Conexión de doble sentido entre la ESP8266, utilizando el programa de Arduino y el apoyándonos en el protocolo de mensajería MQTT.
- Conexión de un sentido entre la ESP8266, también utilizando Arduino, hacia la base de datos de Firebase Database Realtime.

### 5.2.4 Capa de Procesado.

#### 5.2.4.1 Algoritmo Semáforos Python.

En el lenguaje de programación Python, se escribe el código pertinente que está conectado con el Arduino a través del protocolo MQTT.

El objetivo de este código es indicar al Arduino que colores encender de que semáforos sabiendo el color de los semáforos anteriormente y teniendo en cuenta en qué tramo hay más coches. El tramo que tenga más coches tendrá prioridad sobre el otro tramo.

Para poder utilizar en Python es necesario importar la biblioteca “paho-mqtt”. Lo primero que se realiza es conectarse con MQTT y definir los tópicos del que va a escuchar y de donde va a enviar. Desde el tópico “arduinoUAH2023/prueba” va a escuchar todos los mensajes provenientes desde el Arduino y desde el tópico “arduinoUAH2023/prueba2” envía todos los mensajes al Arduino.

```
topicEscucha = "arduinoUAH2023/prueba"
topicEnviar = "arduinoUAH2023/prueba2"
```

*Figura 11: Tópicos de Escucha y Envío.*

Una vez que se conecta al bróker comprueba si le han llegado mensajes provenientes desde el tópico “arduinoUAH2023/prueba”. Los mensajes se corresponden a la cantidad de coches en cada tramo. Los mensajes que llegan se guardan en un vector, una vez recibidos se ejecuta el código. Se verifica que los dos mensajes han llegado correctamente si el tamaño del vector módulo 2 es igual a cero.

Una vez que recibidos los mensajes, se pueden dar los siguientes casos:

- Que haya el mismo número de coches en el tramo 1 y en el 2:Se contempla también que no haya ningún coche en los dos tramos, por lo que los dos tramos serán igual a 0. Se envía al Arduino el mensaje “igual”.

```
#Si los dos tienen el mismo numero de vehiculos
if( mensajes[0] == mensajes[1]):
    client.publish(topicEnviar, "igual")
    print("Envio1 --> igual")
```

*Figura 12: Tópicos de Escucha y Envío.*

Si se produce esto, teniendo en cuenta el color de los semáforos en el estado anterior se determina el nuevo estado del semáforo y teniendo en cuenta en qué ciclo se está. Una vez sabiendo que colores debe tener cada semáforo se unifica la respuesta en un único mensaje y se envía al Arduino a través del tópico “arduino UAH 2023/prueba2”.

El funcionamiento es el siguiente, por ejemplo, si el semáforo 1 está en rojo y el semáforo 2 está en verde en el primer ciclo, el siguiente paso será seguir en rojo el semáforo 1 y poner el según semáforo en naranja para que les dé tiempo a los coches del segundo tramo a pasar antes de que se ponga en rojo. En el siguiente ciclo el primer semáforo pasa de estar rojo a verde y el segundo semáforo pasa de estar en naranja a rojo.

Como el primer tramo tiene más coches que el segundo, se busca poner en verde el primer semáforo y poner en rojo el segundo semáforo.

Una vez que enviado los mensajes, el vector donde se guardaban los mensajes recibidos se reinicia y se vacía y pasa lo mismo con el contador de ciclos.

```
#Si el sensor 1 esta tiene MAS coches que el sensor 2
if (mensajes[0] > mensajes[1]):
    contador1 +=1
    if (contador1==1):
        switcher1 = {
            "v1": "v1",
            "r1": "r1",
            "a1": "v1"
        }
        switcher2 = {
            "v2": "a2",
            "r2": "r2",
            "a2": "r2"
        }
        color1 = switcher1.get(colorActual1, "Opción inválida")
        color2 = switcher2.get(colorActual2, "Opción inválida")
    else:
        switcher1 = {
            "v1": "v1",
            "r1": "v1",
            "a1": "v1"
        }
        switcher2 = {
            "v2": "a2",
            "r2": "r2",
            "a2": "r2"
        }
        color1 = switcher1.get(colorActual1, "Opción inválida")
        color2 = switcher2.get(colorActual2, "Opción inválida")

    colorActual1 = color1
    colorActual2 = color2

    coloresFinal = color1 + color2
    client.publish(topicEnviar, coloresFinal)
    print("Envio1 --> " + coloresFinal)
```

*Figura 13: Tópicos de Escucha y Envío.*

- Que en el tramo 2 haya más coches que en el primero: en este caso, el segundo semáforo se buscará ponerlo en verde mientras que el primer semáforo tendrá que estar en rojo, teniendo en cuenta a través de los ciclos la transición de colores. No se puede poner en rojo directamente si anteriormente está en verde, deberá pasar por el color naranja.

Una vez determinado qué colores de semáforos se deben de encender se envía al Arduino a través del protocolo MQTT siguiendo el formato “r1v2”, rojo el primer semáforo y verde el segundo.

#### 5.2.4.2 Conexión MQTT.

Para la conexión MQTT hacemos uso de la página https://mqttx.app/. Es una herramienta cliente MQTT multiplataforma de código abierto que proporciona una interfaz gráfica de usuario (GUI) para facilitar la prueba, el monitoreo y la depuración de aplicaciones basadas en MQTT.

Al utilizar esta conexión MQTT nos ofrece las siguientes características:

- Permite establecer conexiones con servidores MQTT y gestionar las múltiples conexiones.
- Al proporcionar una interfaz interactiva se puede observar los mensajes que se publican y los mensajes que se reciben al suscribirse.
- Permite monitorizar las diferentes conexiones a través de una interfaz que facilita el entendimiento de los mensajes y visualización.

Debido a estas características se ha decido usar esta aplicación en la nube para establecer las conexiones MQTT.

Como hemos indicado anteriormente, tenemos una conexión establecida, donde tenemos dos tópicos.

- En el tópico “arduinoUAH2023/prueba”, se envían todos los mensajes provenientes del Arduino indicando la cantidad de coches en cada tramo al archivo en Python. El primer mensaje al establecer la conexión es del tramo 1 y el segundo mensaje corresponde al tramo 2.

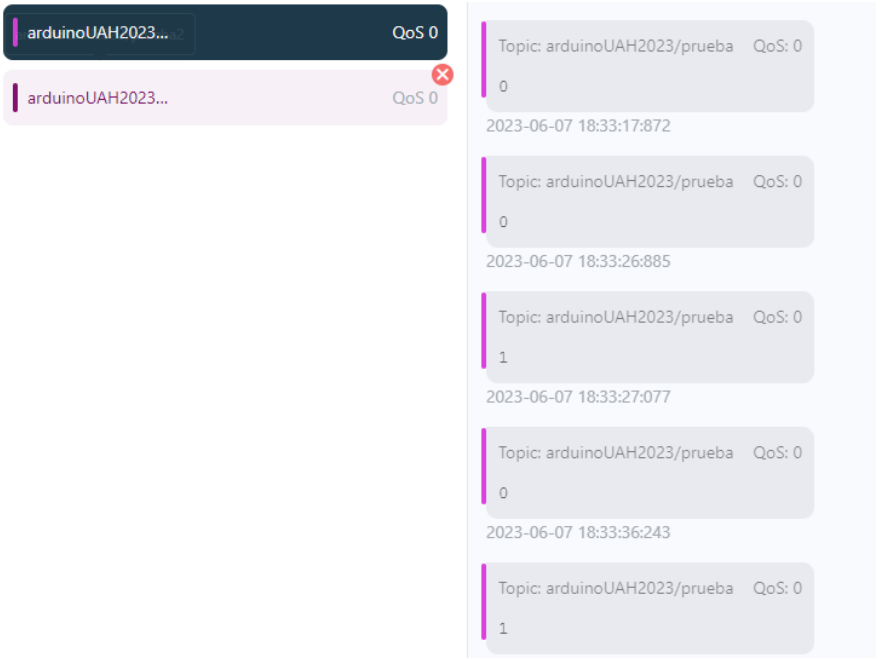



*Figura 14: Mensajes del Tópico "arduinoUAH2023/prueba"*

En la figura 14 se puede ver los mensajes enviados desde el Arduino al Python, siendo el número que aparece el número de coches en ese momento.

- En el tópico "arduinoUAH2023/prueba2", se envían los mensajes provenientes del archivo Python, donde en un mismo mensaje se indica el color que se tiene que encender en cada semáforo. Este mensaje se envía a Arduino.

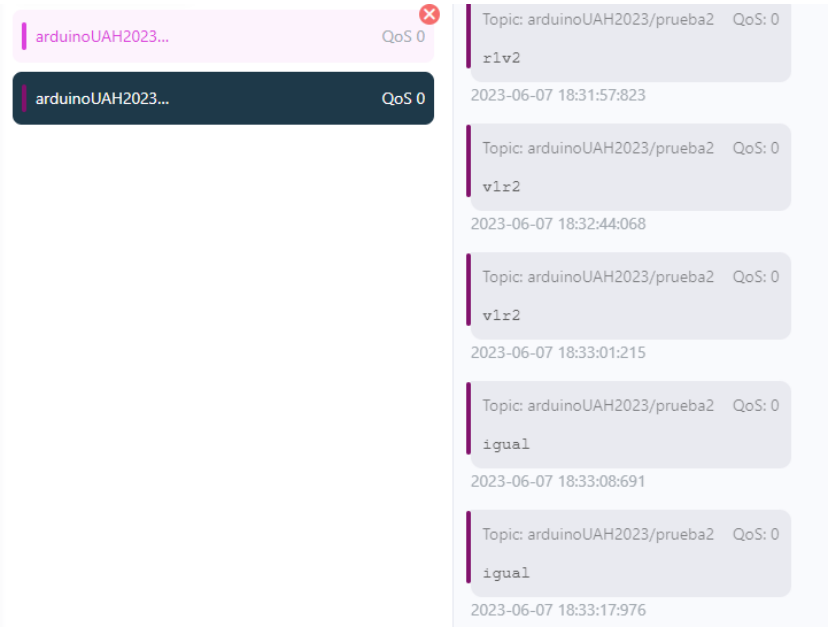



*Figura 15: Mensajes del Tópico "arduinoUAH2023/prueba2"*

En la figura 15 se puede ver los mensajes enviados desde el Python al Arduino, indicando que colores se tienen que encender en cada semáforo.

#### 5.2.4.3 Node.js.

Node.js es un entorno de servidor capaz de ejecutar código JavaScript sin necesidad de un navegador. En nuestro proyecto lo utilizaremos para llevar a cabo las consultas necesarias para obtener la información procedente de la base de datos y, posteriormente, enviarla hacia la interfaz.

Junto a Node.js utilizaremos Express.js, un framework de Node.js que nos permitirá crear un servidor http y definir sus "endpoints" a los que se podrán hacer consultas desde nuestra interfaz.

#### 5.2.6.1 Instalación

Para instalar Node.js deberemos descargar y ejecutar el ejecutable (.exe) desde la página oficial https://nodejs.org/es/download/.

Ahora, para gestionar las dependencias del proyecto, deberemos hacer uso de un gestor de paquetes. Por defecto, Node.js se instala junto a npm, un conocido gestor de paquetes. En nuestro proyecto, utilizaremos yarn, otro gestor de paquetes notablemente más rápido y que utiliza un "bloqueo de versiones" que disminuye las posibilidades de incompatibilidades entre dependencias.

#### 5.2.6.2 Estructura y configuración del servidor

El servidor está compuesto de varias carpetas y archivos: server.js, el directorio "auth" y el directorio "router".

##### 5.2.6.2.1 Server.js

El programa principal está definido en el archivo server.js. En la parte superior del archivo se definen todos los distintos "imports" necesarios para el correcto funcionamiento del servidor.
El módulo "body-parser" es un middleware que permite analizar el cuerpo de las solicitudes http entrantes. Cors es otro middleware que permite las solicitudes al servidor desde otros orígenes, pudiendo restringir el acceso según se considere oportuno (en este caso se permite el acceso desde cualquier origen). Finalmente, el módulo "dotenv" facilita la definición de parte de la configuración del servidor, como el puerto (3000) en el que va a ejecutarse, en un fichero externo y acceder a su valor desde aquellos que requieran dicha información.

```
const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const app = express();
const cors = require('cors');
const admin = require('firebase-admin');
const serviceAccountMQTT = require('./server/auth/firebase_rtdb.json');
const serviceAccountPredictions = require('./server/auth/firebase_predictions.json');
require('dotenv').config();
```

*Figura 16: Módulos importados.*

Además de importar módulos externos, también se hace uso de las credenciales de las bases de datos del proyecto, de las cuales se hablará en mayor detalle en los puntos posteriores de esta memoria, configurándose la conexión a ambas para su uso posterior.

```
const adminRTDB = admin.initializeApp({
    credential: admin.credential.cert(serviceAccountMQTT),
    databaseURL: process.env.FIREBASE_RTDB_URL
},"adminRTDB");

const adminPredictions = admin.initializeApp({
    credential: admin.credential.cert(serviceAccountPredictions),
},"adminPredictions");
```

*Figura 17: Configuración de la conexión a las bases de datos.*

Una vez que todas las herramientas y configuraciones están en su lugar,se procede a definir las rutas que manejará el servidor. Se han configurado dos rutas principales que están vinculadas a sus respectivos enrutadores (/peatones y /coches). El archivo también especifica un directorio de archivos estáticos, permitiendo que la aplicación sirva archivos directamente a los clientes, y define un manejador para la ruta raíz que sirve el archivo principal de la aplicación web, index.html.

Finalmente, el archivo culmina con el servidor poniéndose en marcha, escuchando en el puerto especificado y esperando a que lleguen las solicitudes HTTP.

```
app.use('/API/peatones/', peatonRouter);
app.use('/API/coches/', cocheRouter);

app.get('/', (req, res) => {
    res.sendFile(path.join(__dirname, 'dist', 'index.html'));
});

app.listen(port, () => console.log(`Server listening on port ${port}!`));
```

*Figura 18: Rutas principales y ejecución del servidor.*

#### 5.2.6.2.2 Auth

Este directorio contiene dos archivos .json: firebase_rtdb.json y firebase_predictions.json.

Cada uno de estos archivos contiene las credenciales necesarias para poder conectarse a la base de datos a la que pertenecen y permiten al servidor realizar consultas y recuperar los datos especificados.


*Figura 19: Ejemplo de las credenciales de acceso a las bases de datos.*

### 5.2.6.2.3 Router

Este directorio contiene dos archivos .json: routesPeaton.js y routesCoche.js.

El archivo routesPeaton contiene los endpoints finales de las urls que comienzan por "/API/peatones" y, a su vez, cada uno de estos endpoints acceden a la base de datos correspondiente, recuperan la información requerida y la devuelven en formato JSON. El funcionamiento del archivo routesCoche es análogo, con la única diferencia de que comprende las rutas que comienzan por "/API/coches".

Ambos archivos contienen los siguientes endpoints:

1. `'/por_dia'`: endpoint que acepta solicitudes HTTP de tipo GET. Accede a la base de datos en tiempo real (rtdb) y recupera la cantidad de peatones/coches total que han accedido a la calle cada uno de los días de la semana. Si no lo logra, en su lugar envía un mensaje de error con los detalles.

2. `'/por_semana'`: endpoint que acepta solicitudes HTTP de tipo GET. Accede a la base de datos en tiempo real (rtdb) y recupera la cantidad de peatones/coches total que han accedido a la calle cada una de las semanas del mes actual. Si no lo logra, en su lugar envía un mensaje de error con los detalles.

3. `'/por_mes'`: endpoint que acepta solicitudes HTTP de tipo GET. Accede a la base de datos en tiempo real (rtdb) y recupera la cantidad de peatones/coches total que han accedido a la calle cada uno de los meses del año. Si no lo logra, en su lugar envía un mensaje de error con los detalles.

4. `'/predicciones/hoy'`: endpoint que acepta solicitudes HTTP de tipo GET. Accede a la base de datos con las predicciones generadas por la red neuronal del proyecto (se explicará más adelante) y devuelve la probabilidad, expresada en porcentaje, de que haya tráfico (coches o peatones) elevado para el día en el que se realiza la petición. Si no lo logra, en su lugar envía un mensaje de error con los detalles.

5. `'/predicciones/semana'`: endpoint que acepta solicitudes HTTP de tipo GET. Accede a la base de datos con las predicciones generadas por la red neuronal del proyecto y devuelve la probabilidad, expresada en porcentaje, de que haya tráfico elevado (coches o peatones) para la semana en la que se realiza la petición. Si no lo logra, en su lugar envía un mensaje de error con los detalles.

```
router.get('/por_dia',(req, res) => {
    try{
        let dias = { 'L':0, 'M':0, 'X':0, 'J':0, 'V':0, 'S':0, 'D':0 };
        const ref = req.adminRTDB.database().ref('Semaforos');
        ref.once('value', (snapshot) => {
            const data = snapshot.val();
            for(let key in data) {
                let fecha = moment(data[key].Fecha);
                let dia = fecha.format('d');
                switch (dia) {
                    case '1': dias.L++; break;
                    case '2': dias.M++; break;
                    case '3': dias.X++; break;
                    case '4': dias.J++; break;
                    case '5': dias.V++; break;
                    case '6': dias.S++; break;
                    case '0': dias.D++; break;
                }
            }
            let diasArray = [];
            for (let dia in dias) {
                diasArray.push({ name: dia, value: dias[dia] });
            }
            diasArray.push("días", "vehículos")
            res.json(diasArray);
        }).catch((error) => {
            res.json({ error: error.message });
        });
    }
    catch(error){
        res.json({ error: error.message });
    }
});
```

*Figura 20: Endpoint de obtención del tráfico peatonal diario.*

```
router.get('/por_semana', (req, res) => {
    try {
        let semanas = { '1':0, '2':0, '3':0, '4':0, '5':0 };
        const ref = req.adminRTDB.database().ref('Semaforos');
        ref.once('value', (snapshot) => {
            const data = snapshot.val();
            for(let key in data) {
                let fecha = moment(data[key].Fecha);
                let semana = fecha.week();
                if (semana >= 1 && semana <= 7) semanas['1']++;
                else if (semana >= 8 && semana <= 14) semanas['2']++;
                else if (semana >= 15 && semana <= 21) semanas['3']++;
                else if (semana >= 22 && semana <= 28) semanas['4']++;
                else semanas['5']++;
            }
            let semanasArray = [];
            for (let semana in semanas) {
                semanasArray.push({ name: semana, value: semanas[semana] });
            }
            semanasArray.push("semanas", "vehículos")
            res.json(semanasArray);
        }).catch((error) => {
            res.json({ error: error.message });
        });
    } catch(error) {
        res.json({ error: error.message });
    }
});
```

*Figura 21: Endpoint de obtención del tráfico peatonal semanal.*

```
router.get('/por_mes', (req, res) => {
    try {
        let meses = {
            'Ene':0, 'Feb':0, 'Mar':0, 'Abr':0,
            'May':0, 'Jun':0, 'Jul':0, 'Ago':0,
            'Sep':0, 'Oct':0, 'Nov':0, 'Dic':0
        };
        const ref = req.adminRTDB.database().ref('Semaforos');
        ref.once('value', (snapshot) => {
            const data = snapshot.val();
            for(let key in data) {
                let fecha = moment(data[key].Fecha);
                let mes = fecha.format('M');
                switch (mes) {
                    case '1': meses.Ene++; break;
                    case '2': meses.Feb++; break;
                    case '3': meses.Mar++; break;
                    case '4': meses.Abr++; break;
                    case '5': meses.May++; break;
                    case '6': meses.Jun++; break;
                    case '7': meses.Jul++; break;
                    case '8': meses.Ago++; break;
                    case '9': meses.Sep++; break;
                    case '10': meses.Oct++; break;
                    case '11': meses.Nov++; break;
                    case '12': meses.Dic++; break;
                }
            }
            let mesesArray = [];
            for (let mes in meses) {
                mesesArray.push({ name: mes, value: meses[mes] });
            }
            mesesArray.push("meses", "vehículos")
            res.json(mesesArray);
        }).catch((error) => {
            res.json({ error: error.message });
        });
    } catch(error) {
        res.json({ error: error.message });
    }
);
```

*Figura 22: Endpoint de obtención del tráfico peatonal mensual.*

```
router.get('/predicciones/hoy', (req, res) => {
    try {
        let hoy= moment().startOf('day');
        let manana = moment(hoy).add(1, 'day');
        let transitadoValues = [];
        const prediccionSemaforosTramo1Ref = req.adminPredictions.firestore().collection('PrediccionSemaforosTramo1');
        const prediccionSemaforosTramo2Ref = req.adminPredictions.firestore().collection('PrediccionSemaforosTramo2');

        let getPrediccionSemaforosTramo1 = prediccionSemaforosTramo1Ref
        .where('fecha', '>=', hoy.toDate())
        .where('fecha', '<', manana.toDate())
        .get();

        let getPrediccionSemaforosTramo2 = prediccionSemaforosTramo2Ref
        .where('fecha', '>=', hoy.toDate())
        .where('fecha', '<', manana.toDate())
        .get();

        Promise.all([getPrediccionSemaforosTramo1, getPrediccionSemaforosTramo2])
        .then((snapshots) => {
            snapshots.forEach(snapshot => {
                if (snapshot.empty) {
                    res.json({ error: 'No matching documents.' });
                    return;
                }

                snapshot.forEach(doc => {
                    transitadoValues.push(doc.data().prediccionDia.Coches);
                });
            });

            if (transitadoValues.length > 0) {
                const transitado = Math.round(transitadoValues.reduce((a, b) => a + b, 0)/transitadoValues.length*100).toString()+"%";
                res.json(transitado);
            } else {
                res.json({ error: 'No data available.' });
            }
        })
        .catch(error => {
            res.json({ error: error.message });
        });
    } catch (error) {
        res.json({ error: error.message });
    }
});
```

*Figura 23: Endpoint de obtención de la predicción del tráfico vehicular del día.*

```
router.get('/predicciones/semana', (req, res) => {
    try {
        let hoy= moment().startOf('day');
        let ultimo_dia = moment(hoy).add(7, 'day');
        let transitadoValues = [];
        const prediccionSemaforosTramo1Ref = req.adminPredictions.firestore().collection('PrediccionSemaforosTramo1');
        const prediccionSemaforosTramo2Ref = req.adminPredictions.firestore().collection('PrediccionSemaforosTramo2');

        let getPrediccionSemaforosTramo1 = prediccionSemaforosTramo1Ref
        .where('fecha', '>=', hoy.toDate())
        .where('fecha', '<', ultimo_dia.toDate())
        .get();

        let getPrediccionSemaforosTramo2 = prediccionSemaforosTramo2Ref
        .where('fecha', '>=', hoy.toDate())
        .where('fecha', '<', ultimo_dia.toDate())
        .get();

        Promise.all([getPrediccionSemaforosTramo1, getPrediccionSemaforosTramo2])
        .then((snapshots) => {
            snapshots.forEach(snapshot => {
                if (snapshot.empty) {
                    res.json({ error: 'No matching documents.' });
                    return;
                }

                snapshot.forEach(doc => {
                    doc.data().Semana.forEach(doc => {
                        transitadoValues.push(doc.Coches);
                    });
                });
            });

            if (transitadoValues.length > 0) {
                const transitado = Math.round(transitadoValues.reduce((a, b) => a + b, 0)/transitadoValues.length*100).toString()+"%";
                res.json(transitado);
            } else {
                res.json({ error: 'No data available.' });
            }
        })
        .catch(error => {
            res.json({ error: error.message });
        });
    } catch (error) {
        res.json({ error: error.message });
    }
});
```

*Figura 24:Endpoint de obtención de la predicción del tráfico vehicular de la semana.*

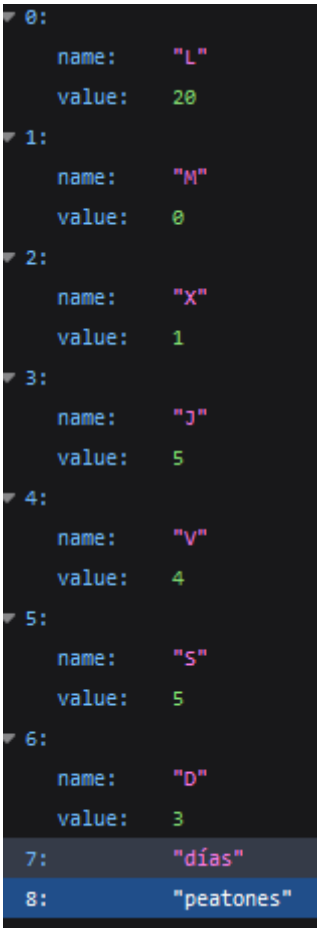



*Figura 25: Ejemplo de respuesta exitosa.*

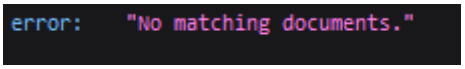



*Figura 26: Ejemplo de respuesta fallida.*

#### 5.2.4.4 Bases de Datos.

Para el proyecto era fundamental tener una base de datos en tiempo real y no relacional. Y tener otra base de datos con una estructura de datos flexible basada en documentos y colecciones.

Es necesario trabajar con una base de datos en tiempo real debido a que las operaciones que se realizan se tienen que almacenar de forma instantánea, que en pocos milisegundos la información quede almacenada correctamente. Es necesario que al momento de que ocurra un nuevo suceso en la vía pública quede almacenado para poder trabajar con él.

También es necesario trabajar con una base de datos que no fuera relacional para evitar trabajar con estructuras definidas, como pueden ser tablas en bases de datos de lenguaje SQL. Este punto es necesario ya que con la información que se trabaja es preferible que se almacene en forma de árboles, con su raíz y sus respectivos nodos.

Esta base de datos en tiempo real se va a utilizar para almacenar toda la información proveniente de la vía pública, de los diferentes sensores.

La otra base de datos es necesaria que tenga una estructura flexible basada en documentos y colecciones, debido a las siguientes cuestiones:

- Proporciona una flexibilidad en el esquema de datos, a diferencia de las bases de datos con esquemas rígidos como pueden ser las bases de datos SQL
- Nos permite tener una mayor agilidad en el desarrollo realizando cambios rápidos en el modelo de datos sin preocuparte.
- Al tener la base de datos basada en documentos las consultas a la base de datos serán de menos complejidad y de una forma más flexible.

Esta segunda base de datos almacenará todas las predicciones que se realicen a través de la red neuronal.

#### 5.2.4.4.1 Firebase-Google

Firebase es una plataforma digital desarrollada por Google, la cual proporciona multitud de servicios, también soporta múltiples plataformas por lo que es de vital importancia para el proyecto. Proporciona un acceso gratuito a través de un mes de prueba.

Para el desarrollo de las dos bases de datos de nuestro proyecto se realizan en la plataforma de Google Firebase.

Se utiliza esta plataforma porque nos brinda la posibilidad de tener una base de datos en tiempo real (Realtime Database) para almacenar la información recabada por los sensores y al mismo tiempo tener otra base datos (Firestore) para almacenar las predicciones tanto del paso de peatones como del semáforo en los dos tramos.

#### 5.2.4.4.2 Base de Datos en Tiempo Real

Teniendo estos factores en cuenta la base de datos que más factores cumple para el proyecto es “Firebase Realtime Database”, ya que sus principales características son que es una base de datos en tiempo real, lo cual significa que las diferentes operaciones no tardan más que unos pequeños milisegundos en ejecutarse y no es relacional. Nos permite almacenar la información en árboles de n nodos.

La estructura de la base de datos en las siguiente:

- Se tiene dos nodos raíz, lo cuales son *“Paso de Peatones”* y *“Semáforos”*. Son independientes entre ellos.
- En cada nodo raíz se tiene una serie de nodos hijos, donde estos nodos hijos son el padre de los atributos que se almacenarán. Por lo cual el número de nodos que son hijos de la raíz corresponde al número de incidencias capturadas.
- En cada nodo hijo se guarda la información correspondiente con el suceso.
    - En el nodo Paso de Peatones, los atributos que se almacenan y tipo de dato son los siguientes:
        - Día del mes (Integer).
        - Fecha (Data-TimeLocal).
    - En el nodo Semáforos, los atributos que se almacenan son:
        - Coches (Int).
        - Día del mes (Integer).

- Fecha (Data-TimeLocal).
- Tramo, puede ser tramo 1 o 2 (Int).

Visualización real de la base de datos:


*Figura 27: Base de Datos en Tiempo Real (Paso de Peatones).*


*Figura 28: Base de Datos en Tiempo Real (Semáforo).*

### 5.2.4.4.3 Base de Datos Firestore

Al almacenar las predicciones no se necesita una base de datos en tiempo real y una base de datos basada en documentos y colecciones es perfecta para almacenar este tipo de datos.

La estructura de la base de datos en las siguiente:

- Se tienen tres tipos de colecciones independientes entre ellas. Las colecciones son *"PrediccionPeatones"*, "*PrediccionSemaforosTramo1"* y *"PrediccionSemaforosTramo2".*
  - La*"PrediccionPeatones"* se corresponde con la predicción del paso de peatones.
  - La "*PrediccionSemaforosTramo1"* se corresponde con la predicción del primer tramo de carretera.
  - La *"PrediccionSemaforosTramo2"* se corresponde con la predicción del segundo tramo de carretera.

- En el documento se tiene los identificadores correspondientes a cada predicción realizada.
- En el campo se tienen los diferentes atributos de cada documento. Para los tres tipos de colecciones se tiene el mismo formato, el cual es el siguiente:
    - Semana, donde se obtienen las predicciones del mismo día que se ha realizado la predicción y de los seis días siguientes. Cada día está formado por el campo *"Transitado"* en el cual hay un porcentaje entre 0 y el 1 indicando la probabilidad de que este transitado.
    - Fecha, donde se tiene la fecha en la que se realiza la predicción de tipo Data-TimeLocal.
    - PrediccionDia, donde se tiene la predicción de ese mismo día.

Visualización real de la base de datos:

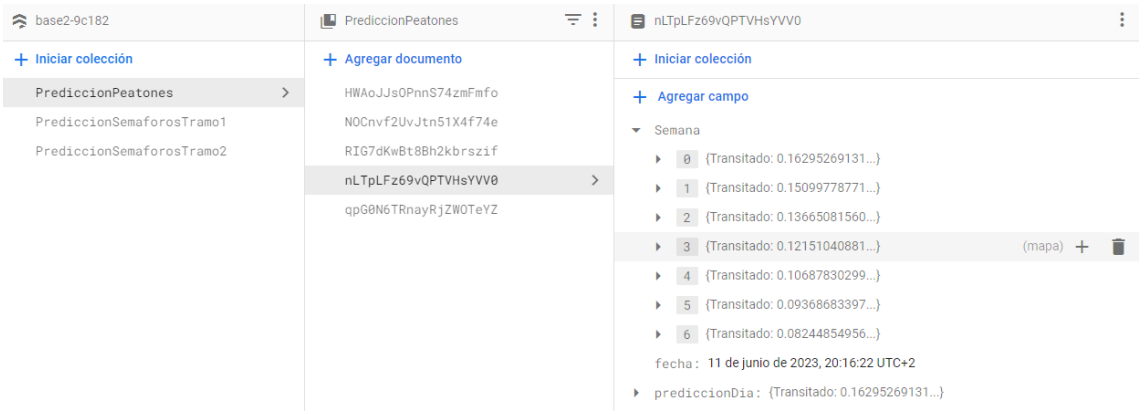



*Figura 29: Base de Datos Firestore (Peatones).*

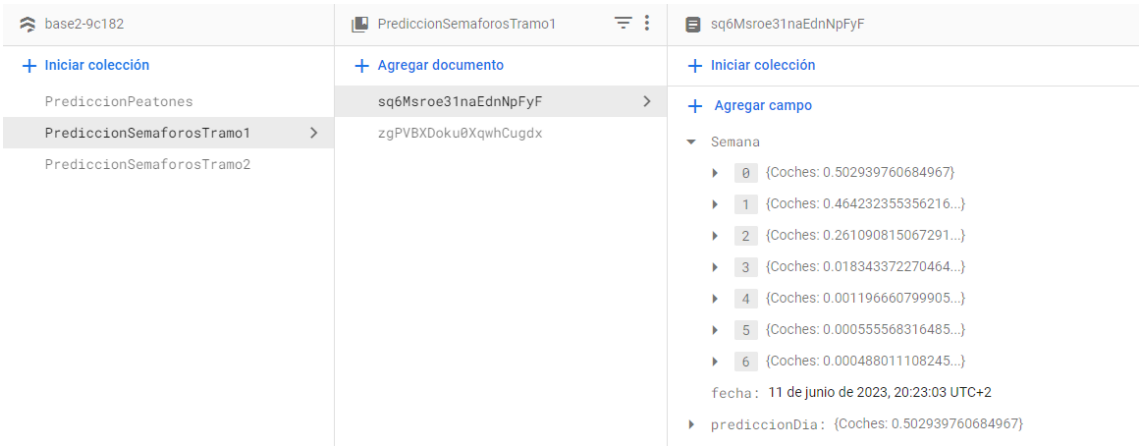



*Figura 30: Base de Datos Firestore (Semáforos Tramo 1).*

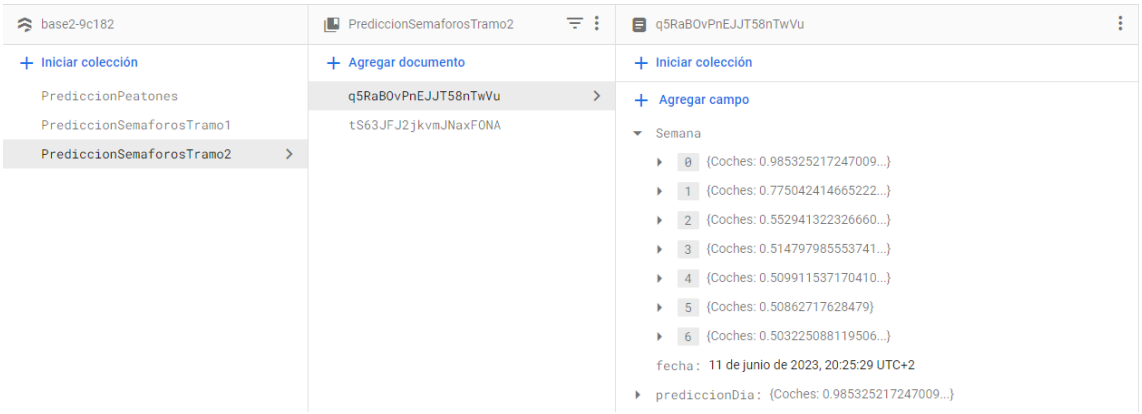



*Figura 31: Base de Datos Firestore (Semáforos Tramo 2).*

### 5.2.4.4.4. Poblar Bases de Datos

Para simular diferentes aspectos del proyecto ha sido necesario tener una base de datos, con un gran conjunto de información en su interior, ha sido necesario por ejemplo para poder realizar la red neuronal y poder entrenarla, para la representación de diferentes aspectos en la interfaz, para realizar las diferentes consultas.

La creación e inserción de valores simulados se ha realizado a través de un Script (poblarBdD.html). Lo primero que se realiza es la conexión con la base de datos, después la creación de los diferentes atributos para cada nodo, de forma totalmente aleatoria, menos la fecha que es a partir de este último año 2023.

Por último, se insertan los diferentes valores, teniendo en cuenta el nodo raíz (“Paso de Peatones” o “Semáforo”), verificando que no se encuentran ya registrados en la base de datos.

```
function insertDataP(){
    var diaP = NumerosAleatorios(1, 12)
    var fechaP = new Date(2023,5,diaP)
    var fechaFinal = fechaP.toISOString()
    //console.log("dia ", diaP, "fecha= ", fechaFinal)
    var IDSP =  NumerosAleatorios(0, 1500); // se pueden de 0 hasta mil

    //if (vacio(IDSP)){
        get(child(ref(db), "Paso de Peatones/" + IDSP)).then ((snapshot)=>{
            if(!snapshot.exists()){
                console.log("Se inserta")
                set(ref(db, "Paso de Peatones/"+ IDSP), {
                    Dia: diaP-1,
                    Fecha: fechaFinal
                })
                .then(()=>{
                })
                .catch((error)=>{
                    alert("Error"+error);
                });
            }else{
                    console.log("El suceso ya existe peatones")
                }
        })

    }
```

*Figura 32: Función Insertar Datos aleatorios (Paso de Peatones).*

```
function insertDataS(){
    var diaS = NumerosAleatorios(1, 30)
    var fechaS = new Date(2022,5,diaS);
    var fechaFinal = fechaS.toISOString();
    var IDSP =  NumerosAleatorios(0, 1500); // se pueden de 0 hasta 1
    var Tramo = NumerosAleatorios(1, 2);

    //if (vacio(IDSP)){
        get(child(ref(db), "Semaforos/" + IDSP)).then ((snapshot)=>{
            if(!snapshot.exists()){
                console.log("Se inserta")
                set(ref(db, "Semaforos/"+ IDSP), {
                    Coches: 1,
                    Fecha: fechaFinal,
                    Dia: diaS-1,
                    Tramo: Tramo
                })
                .then(()=>{
                })
                .catch((error)=>{
                    alert("Error"+error);
                });
            }else{
                console.log("El suceso ya existe peatones")
            }
        })
    }
```

*Figura 33: Función Insertar Datos aleatorios (Semáforo).*

### 5.2.4.5 Ciencia de Datos Obtenidos.

#### 5.2.4.5.1.Introducción

A partir de los datos recabados por los sensores, tanto el sensor situado en el paso de peatones, como los cuatro sensores que se encuentran en la carretera, se realiza un análisis de datos.

La ciencia de datos se emplea en el proyecto con el objetivo de extraer información significativa. A través de la ciencia de datos se puede entender lo que está sucediendo en la vía pública, detectar posibles problemas, mejorar la eficiencia de la tecnología utilizada y aportar información de valor a los ciudadanos. Todas estas características poseen un objetivo global, el cual es incrementar en ciertos ámbitos la calidad de vida de las personas.

En concreto se va a realizar en el proyecto un análisis predictivo. Consiste en utilizar un histórico de datos para realizar previsiones precisas sobre patrones que pueden producirse en un futuro. Se caracteriza en técnicas de Machine Learning, la coincidencia de patrones y el modelado predictivo. En todas las técnicas del análisis predictivo, se le introduce una serie de datos con conclusiones positivas y negativas, se entrena al ordenador para realizar ingeniería inversa a las conexiones de causalidad en los datos. A partir de un registro de datos del pasado se puede obtener, con probabilidades, que puede suceder en un futuro. Debido a esta funcionalidad del análisis predictivo, de predecir lo que puede ocurrir en un futuro, se ha escogido para el proyecto.

Es una gran ventaja, que, a partir de un histórico de datos en referencia a las vías públicas, como es en nuestro caso, conocer cuando y donde van a cruzar peatones o cuántos vehículos van a circular por la vía en cada momento y donde lo harán. Esta funcionalidad, nos permite modelar un sistema predictivo para conocer múltiples factores que afectan a diario a las personas que circulan por esa ubicación.

Se realiza el análisis predictivo a partir de la red neuronal “Brain.js”.

#### 5.2.4.5.2.Obtención de los datos

Los datos utilizados para el análisis predictivo son recabados a partir de los sensores implementados descritos anteriormente. Los datos procedentes del paso de peatones se obtienen a través de un sensor situado en un paso de peatones. Los datos procedentes de la calzada de circulación se han obtenido a través de 4 sensores situados estratégicamente para obtener el número exacto de vehículos que circulan por la vía.

A partir del sensor situado en el paso de peatones, se obtiene la fecha exacta en él ha ocurrido el suceso, mientras que en los sensores situados en la calzada recogen cuando ha pasado un coche y la fecha exacta en que ha ocurrido. Por lo tanto, para el paso de peatones y el de la vía se puede obtener cuantos coches o peatones transcurren a lo largo del día.

#### 5.2.4.5.3. Brain.js

Una red neuronal es un método para enseñar a las computadoras a procesar información, que está inspirada en el cerebro humano. La red neuronal es un tipo de machine learning, utiliza nodos interconectados con una estructura por capas. Estos nodos forman una red compleja que tiene un alto nivel de interconexión, esta red estaría formada por tres capas:

- Capa de Entrada: Consiste en la información que se introduce en la red neuronal, los diferentes nodos procesan los datos, los analizan, los clasifican y los envían a la siguiente capa.
- Capa Oculta: Toman la información de la capa de entrada o a partir de otras capas ocultas. Se tiene una gran cantidad de capas ocultas.
- Capa de Salida: Proporciona el resultado final, puede estar compuesto de un nodo o de varios nodos.

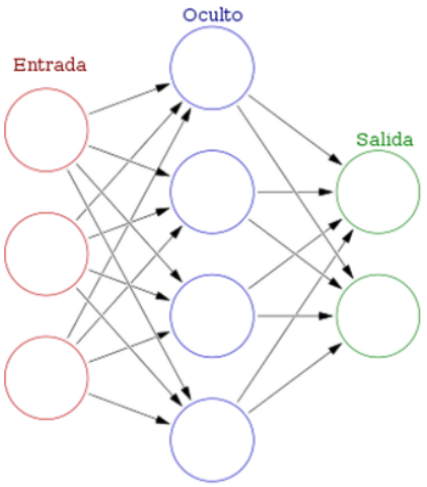



*Figura 34: Esquema básico de una Red Neuronal.*

Como red neuronal para nuestro proyecto se utiliza Brain,js, que es una biblioteca desarrollada por JavaScript que permite implementar y entrenar redes neuronales en el navegador o en un entorno Node.js. Proporciona una interfaz fácil de usar para construir y entrenar modelos de redes neuronales utilizando algoritmos de aprendizaje automático.

Para que la red neuronal Brain.js realice predicciones válidas, se la tiene que entrenar. Para entrenarla se le tiene que pasar una gran cantidad de datos donde están formados por un input y un output los cuales tienen que estar asociados. La red neuronal recoge toda la información y la procesa a partir de una serie de algoritmos que funcionan con patrones de coincidencias.

Una vez ya entrenada la red neuronal va a ser capaz de esclarecer un output a partir de un input.

#### 5.2.4.5.4. redNeuronal.js

Lo primero que se realiza en nuestro código es llamar al script "Brain.js", donde se encuentra la red neuronal al completo. Después se realizan las correspondientes consultas para la obtención de la información. Se va a realizar una consulta en el nodo "Paso de Peatones", donde obtendremos toda la información correspondiente del nodo, se realiza el mismo procedimiento para el nodo "Semáforos".

Con toda la información extraída de la base de datos, se calcula el número de veces que pasa un peatón por el paso de peatones en un día respecto del mes y se realiza lo mismo con los coches que pasan por la vía diferenciando en que tramo lo hacen. Esto se realiza a través de las funciones *"peatonesDia()"* y *"cochesDia()"*.

```
function peatonesDia(){
    for (var i = 0; i < nodosP.length; i++) {
        var dia = nodosP[i].Dia;
        if (contadorPeatonesDia[dia]) {
            contadorPeatonesDia[dia]++;
        } else {
            contadorPeatonesDia[dia] = 1;
        }
    }
    //console.log("Contador por dia",contadorPeatonesDia);
}
```

```
function cochesDia(){
    for (var i = 0; i < nodosS.length; i++) {
        var dia = nodosS[i].Dia;
        if (nodosS[i].Tramo == 1){
            if (contadorCochesDiaT1[dia] ) {
                contadorCochesDiaT1[dia]++;
            } else {
                contadorCochesDiaT1[dia] = 1;
            }
        }else if (nodosS[i].Tramo == 2){
            if (contadorCochesDiaT2[dia] ) {
                contadorCochesDiaT2[dia]++;
            } else {
                contadorCochesDiaT2[dia] = 1;
            }
        }
    }

    //console.log("Contador por dia tramo 1",contadorCochesDiaT1);
    //console.log("Contador por dia tramo 2",contadorCochesDiaT2);
}
```

*Figura 35 y Figura 36: Funciones peatonesDia() y cochesDia().*

Una vez sabiendo el número de peatones o de coches que han pasado por ese tramo se entrena a la red neuronal. Tendremos dos redes neuronales entrenadas de forma diferente.

La primera estará entrenada para el paso de peatones. Se entrena utilizando la función *"trainingP"*. Esta función va introduciendo inputs cada día del mes y considerando como esta de transitado el paso de peatones, se considera transitado si hay más de seis peatones ese mismo día. Consideramos ese número por poner un número para indicar cuando hay tráfico y cuando no. Entonces en los días que se supere ese número se indica a través de un output que

está transitado y los días que no se supere no estará transitado. Una vez entrenada la red neuronal la función la devuelve.

```
let net = new brain.NeuralNetwork();
for (var dia in contadorPeatonesDia) {
    if (dia != "undefined"){
        if (contadorPeatonesDia[dia]>= 6){ //Si se tienen mas de 6 peatones en un dia estara transitado
            datos.push ({
            "input":{"Dia":dia},
            "output":{"Transitado":1}
        })
        }else{ //Si no, no lo estara
            datos.push ({
            "input":{"Dia":dia},
            "output":{"Transitado":0}
        })
        }
    }
}
```

*Figura 37: Parte de la función "trainingP()".*

El mismo procedimiento explicado anteriormente se realizará para entrenar la red neuronal para vía pública transitada por coches. El número que se considera si hay trafico o no es 4 coches o más, por lo tanto, si ese mismo día han pasado más de 4 coches se considerará tráfico, sino, no. A la hora de entrenar a la red hay que diferenciar si hay tráfico en el tramo 1 o 2, por lo cual, será otro atributo en los inputs de la red neuronal. Por último, la función retorna la red neuronal entrenada.

```
let net = new brain.NeuralNetwork();
for (var dia in contadorCochesDiaT1) {
    if (dia != "undefined"){
        if (contadorCochesDiaT1[dia]>= 4){ //Si se tienen mas de 4 Coches en un dia habra trafico
            datos.push ({
                "input":{"Dia":dia, "Tramo":1},
                "output":{"Coches":1}
            })
        }else{ //Si no, no lo habra
            datos.push ({
                "input":{"Dia":dia, "Tramo":1},
                "output":{"Coches":0}
            })
        }
    }
```

*Figura 38: Parte de la función "trainingS()".*

Una vez entrenadas ambas redes neuronales se ejecutan introduciéndose en el caso de la red neuronal de paso de peatones el día y en el caso de la red neuronal el día y el tramo. Esta funcionalidad se realiza utilizando las funciones "*prediccionDiaP()*", "*prediccionDiaSTramo1()*" y "*prediccionDiaSTramo2()*". Como el siguiente ejemplo:

```
function prediccionDiaSTramo1(net){
    let resultadoSTramo1 = net.run({"Dia":hoy, "Tramo":1})
    console.log("probabilidades de que este con trafico", resultadoSTramo1);
    return resultadoSTramo1
}
```

*Figura 39: Predicción de la vía del tramo 1.*

De la misma forma se realizan las predicciones de los seis días siguientes.

Una vez obtenidos cada resultado se almacena en la base de datos Firestore. Para ello se inicializa y conecta a la base de datos a través de las credenciales. Una vez que se produce la conexión se insertan los resultados teniendo en cuenta en qué colección van.

### 5.2.4.5.5.Discusión de los Resultados de las Predicciones

Como la mayoría de los datos de la base de datos son datos aleatorios y los datos aleatorios tienden a igualarse entre las posibilidades, las predicciones dependiendo del día salían muy similares. Por lo cual es un hecho a destacar de que las predicciones pueden que estén bien hechas.

Entonces se ha decidido realizar lo contrario. Se ha introducido multitud de datos correspondiente a un solo día del mes, en concreto el día 11 de mayo de 2023, para que, a la hora de realizar las predicciones, para el día 11 de junio de 2023 se obtuviera que es muy probable que la calle estuviera transitada y que los seis días siguientes la probabilidad decrementará notoriamente.

Se ha realizado este experimento sobre el tramo 1. A continuación se muestra la cantidad de coches que han pasado por el tramo 1 en el mes de mayo.

```
1: 1
2: 2
4: 4
5: 1
7: 2
8: 1
9: 1
11: 21
14: 1
15: 1
16: 1
17: 3
19: 2
22: 2
24: 1
27: 1
```

*Figura 40: Número de coches en los días de mayo.*

Como se puede observar el día donde más coches han pasado por el tramo 1 ha sido el 11 de mayo, por lo cual, la predicción que debería hacer la red neuronal para el 11 de junio debería tener muchas posibilidades de estar transitado y el resto de los seis días deberá tener una probabilidad bastante baja.

Al realizar la predicción para el 11 de junio de 2023 y los siguientes seis días se obtiene el siguiente resultado:


*Figura 41: Probabilidad de Tráfico el día 11 de junio de 2023 y sus seis siguientes.*

Como se puede observar en los resultados, el día 11 de junio de 2023 obtiene una probabilidad de tener tráfico de un 98% y sus seis días siguientes tienen una probabilidad que casi no llegan al 1%. Se han obtenido los resultados esperados, los resultados tienen sentido y por lo tanto pueden indicar que el proyecto, aplicado a la vida real, con datos reales puede predecir el tráfico de una forma notable.

También hay que indicar que el error en el experimento realizado ha sido del 0.0704141352 y la red neuronal ha tenido 20.000 iteraciones.

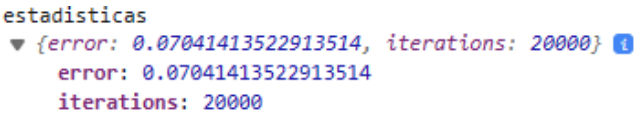



*Figura 42: Estadísticas del Experimento.*

### 5.2.5 Capa de aplicación

La capa de aplicación busca poder mostrar la información obtenida por los sensores y analizada en el servidor de forma clara y concisa al usuario, facilitando todo lo posible su comprensión y centrándose en la inmediatez.

Está compuesta por una página web, diseñada para móviles.

#### 5.2.5.1 Aplicación Web

La aplicación se ha desarrollado en React, un framework de JavaScript y TypeScript que facilita la creación de interfaces permitiendo su división en módulos bien diferenciados, reutilizables, actualizables dinámicamente y capaces de comunicarse entre sí, y TypeScript, lenguaje de programación similar a JavaScript pero con algunos añadidos tales como el tipado de datos, que ofrece robustez y seguridad al proyecto, evitando en la medida de lo posible la utilización de tipos de datos erróneos.

El proyecto está dividido en tres carpetas: pages, components y utils. Además, en la raíz del directorio, contamos con el archivo App.tsx.

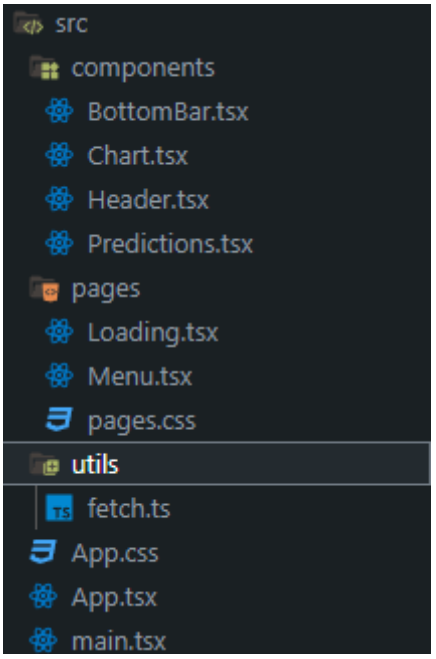



*Figura 43: Estructura del directorio con el código fuente de la página web.*

En la estructura de la aplicación, hay un directorio llamado `pages` que alberga el componente `Menu.tsx`. Este componente es esencial ya que tiene la responsabilidad de cargar los gráficos con los datos que provienen del backend. De esta manera, visualiza información crucial en una forma que los usuarios pueden entender y utilizar fácilmente.

Por otro lado, la carpeta `components` desempeña un papel vital en la construcción de la interfaz de usuario de la aplicación. Incluye varios componentes que trabajan juntos para ofrecer una experiencia de usuario fluida.

El directorio utils cuenta con un archivo “fetch.ts”, que contiene las funciones encargadas de recuperar los datos que se necesiten desde los componentes, realizando solicitudes http al servidor.

```
const backend = import.meta.env.VITE_APP_BACKEND_URL as string || 'http://localhost:3000';


export const getGraphData = (type:string, time:string) =>{
    const myHeaders = new Headers();
    myHeaders.append("Content-Type", "application/json");


    const requestOptions: RequestInit = {
        method: 'GET',
        headers: myHeaders,
        redirect: 'follow' as RequestRedirect
    };

    return fetch(`${backend}/API/${type}/${time}`,requestOptions)
}


export const getPredictionData = (type:string,time:string) =>{
    const myHeaders = new Headers();
    myHeaders.append("Content-Type", "application/json");


    const requestOptions: RequestInit = {
        method: 'GET',
        headers: myHeaders,
        redirect: 'follow' as RequestRedirect
    };

    return fetch(`${backend}/API/${type}/predicciones/${time}`,requestOptions)
}
```

*Figura 44: Métodos para la recuperación de datos procedentes del backend..*

El componente App.tsx será la página principal del proyecto, compuesta por los componentes Header.tsx, Menu.tsx y BottomBar. En él se define un método(“toggleType”) ,que se transfiere al resto de componentes hijos, y que permite modificar el valor del atributo “type”, encargado de identificar el tipo de la información que se va a mostrar desde dichos componentes.

```tsx
const App = () => {

  const [loading, setLoading] = React.useState(true)
  const [type, setType] = React.useState<string>("peatones")

  const toggleType = (new_type: string) => {
    if( type != new_type){
      setType(new_type)
      setLoading(true)
    }
  }

  useEffect(() => {
    setTimeout(() => {
      setLoading(false)
    }, 1000)
  }, [type])

  if (loading) {
    return (
      <div id="App">
        <Loading/>
      </div>
    )
  }

  else{
    return (
      <div id="App" className='loaded'>
        <Header type={type}/>
        <Menu type = {type}/>
        <BottomBar toggleType={toggleType} />
      </div>
    )
  }
}
```

*Figura 45: Estructura del componente App.tsx.*

El componente Menu.tsx está, a su vez, compuesto por los componentes Chart.tsx y Predictions.tsx. El primero muestra un gráfico con la cantidad de vehículos o peatones que atraviesan nuestra calle en un periodo de tiempo (cantidad en el eje Y, periodo de tiempo en el eje X), mientras que el segundo carga las probabilidades, predichas por la red neuronal y almacenadas en una de nuestras bases de datos, como un porcentaje. Además, el menú también cuenta con un selector que permite modificar el valor de la variable time (compartida entre los componentes hijos de Menu.tsx) que permite modificar el periodo de tiempo por el que se desea buscar (dia/semana/mes).

```tsx
const Menu:React.FC<MenuProps> = ({type}) => {

  const [time, setTime] = useState<string>("por_dia")

  const handleTime = (e: React.ChangeEvent<HTMLSelectElement>) => {
    setTime(e.target.value)
  }

  return (
    <div className='Menu'>
      <div className="select-container">
        <select className="select-style" onChange={handleTime}>
            <option value="por_dia">Diario</option>
            <option value="por_semana">Semanal</option>
            <option value="por_mes">Mensual</option>
        </select>
      </div>
      <Chart type={type} time={time}/>
      <Predictions  type={type} />
    </div>
  )
}
```

*Figura 46: Estructura del componente Menu.tsx.*

Tanto el componente Chart como el componente Predictions, antes de renderizarse, realizan las solicitudes http correspondientes al servidor y, una vez cargados los datos, se actualizan con la información recibida. Mientras que la información se está obteniendo del servidor se muestra el componente Loading.tsx, que no es más que una pantalla de carga. Cada vez que se actualice el valor de la variable time o de la variable type, se solicitarán los nuevos datos al servidor de la forma que se acaba de explicar.

El componente BottomBar está compuesto por dos imágenes. Cada una de ellas, al pulsarse, modifican el tipo de los datos que se deben mostrar (type), forzando la actualización de los componentes que también accedan al valor de dicha variable.

```tsx
const BottomBar:React.FC<BottomBarProps> = ({toggleType}) => {

  return (
    <div id="BottomBar" className='BottomBar'>
      <div className='BottomBar-Left'>
        <img
          src='/assets/BottomBar_icon1.png'
          className='BottomBarImage'
          onClick={()=>{toggleType("peatones")}}/>
      </div>
      <div className='BottomBar-Right'>
        <img
          src='/assets/BottomBar_icon2.png'
          className='BottomBarImage'
          onClick={()=>{toggleType("coches")}}/>
      </div>
    </div>
  )
}
```

*Figura 47: Estructura del componente BottomBar.tsx.*

Finalmente, el componente Header muestra el nombre de la información que se está visualizando en pantalla (tráfico de peatones/coches).

En cuanto al estilo de la página, hemos utilizado una única hoja de estilos .css y definido la apariencia de cada componente en una clase diferente. También se han añadido algunas animaciones.

## 5.3. Plan de desarrollo

Para la recuperación de este trabajo empezamos a realizar las modificaciones después de la evaluación ordinaria del segundo cuatrimestre, por lo que dispusimos de dos semanas aproximadamente. Durante estas dos semanas seguimos el siguiente plan de desarrollo:

| | Día 1 | Día 2 | Día 3 | Día 4 | Día 5 | Día 6 | Día 7 | Día 8 | Día 9 |
|---|---|---|---|---|---|---|---|---|---|
| Mejora Interfaz | | | | | | | Marcos T. | Marcos T. | Marcos T. |
| Mejora Servidor | Marcos T. | Marcos T. | Marcos T. | Marcos T. | Marcos T. | Marcos T. | | | |
| Bases de datos | Marcos N. y Adrián | Marcos N | Marcos N. | | | | | | |
| Modificación red neuronal | | | | | | | | Marcos N. | Marcos N. |
| MQTT | | Marcos N. | Marcos N. | Marcos N. | Marcos N. y Adrián | | | | |
| Modificación Arduino | | Marcos N. y Adrián | Marcos N. y Adrián | Marcos N. y Adrián | Marcos N. | Marcos N. | Adrián | Adrián | |
| Documentación | | | | | | | | | |

| Día 10 | Día 11 | Día 12 | Día 13 | Día 14 |
|---|---|---|---|---|
| Marcos T. | Marcos T. | Marcos T. | | |
| | | | | |
| Marcos N. | | | | |
| | | | | |
| | | | | |
| | | | Adrián | Adrián |

Las celdas sin nombres representan una tarea realizada entre todos.

# 6. Conclusiones.

En conclusión el proyecto realizado, se ha conseguido implementar todos los objetivos predispuestos. Cabe destacar que el proyecto expuesto si se aplicase en un entorno real y tuviera la suficiente financiación para implementar todos los organismos descritos supondría una mejora diaria para los transeúntes de la vía, incluso pudiendo escalar a tantas vías como fuera necesario. Los aspectos de la vida cotidiana que mejoran al utilizar el proyecto, con el simple hecho de usar nuestra aplicación son los siguientes:

- Eficiencia en el transporte tanto de los tramos como en paso de peatones gracias a las predicciones: Al realizar una predicción de cómo de transitada está la vía en ese día, al usuario medio le aporta una gran ventaja a la hora de planificar su trayecto y evitar diferentes atascos.
- Eficiencia en el transporte por el uso de los semáforos: Al implementar un algoritmo en tiempo real sobre los tiempos de los semáforos y cómo se distribuyen los tiempos, provocará que la circulación sea más eficiente y perdiendo menos tiempo en los tramos.
- Seguridad en la vía: Al utilizar tanto las predicciones como el algoritmo de los semáforos va provocar una eficiencia en la circulación, lo que a su vez, provocará una mayor seguridad y menos ocurrencia de problemas.
- Seguridad en el paso de peatones: Al iluminar el paso de peatones, exactamente cuando un viandante pase por la vía, provocará que los diversos vehículos tengan una mayor visibilidad de lo que ocurre. También al iluminar la vía solo cuando es necesario, el gasto de la energía usada en ese tramo se usará de una forma muy eficiente.

# 7. Bibliografía.


1. https://www.e-ika.com/sensor-de-presion-fsr402-de-125mm
2. https://programarfacil.com/podcast/esp8266-wifi-coste-arduino/
3. https://www.tinkercad.com/
4. https://mqttx.app/
5. https://pypi.org/project/paho-mqtt/
6. https://www.juanjobeunza.com/esp32-firebase/
7. https://rntlab.com/question/ntpclient/
8. https://github.com/mobizt/Firebase-ESP-Client
9. https://randomnerdtutorials.com/esp8266-pinout-reference-gpios/
10. https://github.com/taranais/NTPClient
11. https://aws.amazon.com/es/what-is/data-science/
12. https://aws.amazon.com/es/what-is/neural-network/
13. https://www.juanjobeunza.com/esp32-firebase/
14. https://rntlab.com/question/ntpclient/
15. https://github.com/mobizt/Firebase-ESP-Client
16. https://randomnerdtutorials.com/esp8266-pinout-reference-gpios/
17. https://github.com/taranais/NTPClient
18. https://expressjs.com/
19. https://www.w3schools.com/
20. https://www.typescriptlang.org/
21. https://nodejs.org/es/
22. https://firebase.google.com/docs/firestore/quickstart?hl=es-419
23. https://firebase.google.com/docs/database?hl=es-419
24. https://brain.js.org/#/
25. https://keepcoding.io/blog/ejemplo-red-neuronal-profunda-tensorflow/

# Anexo I – Manual de instalación.

## Archivos Arduino.

En este anexo vamos a explicar como instalar tanto el programa arduino y todas las librerías necesarias para el funcionamiento de nuestro proyecto.

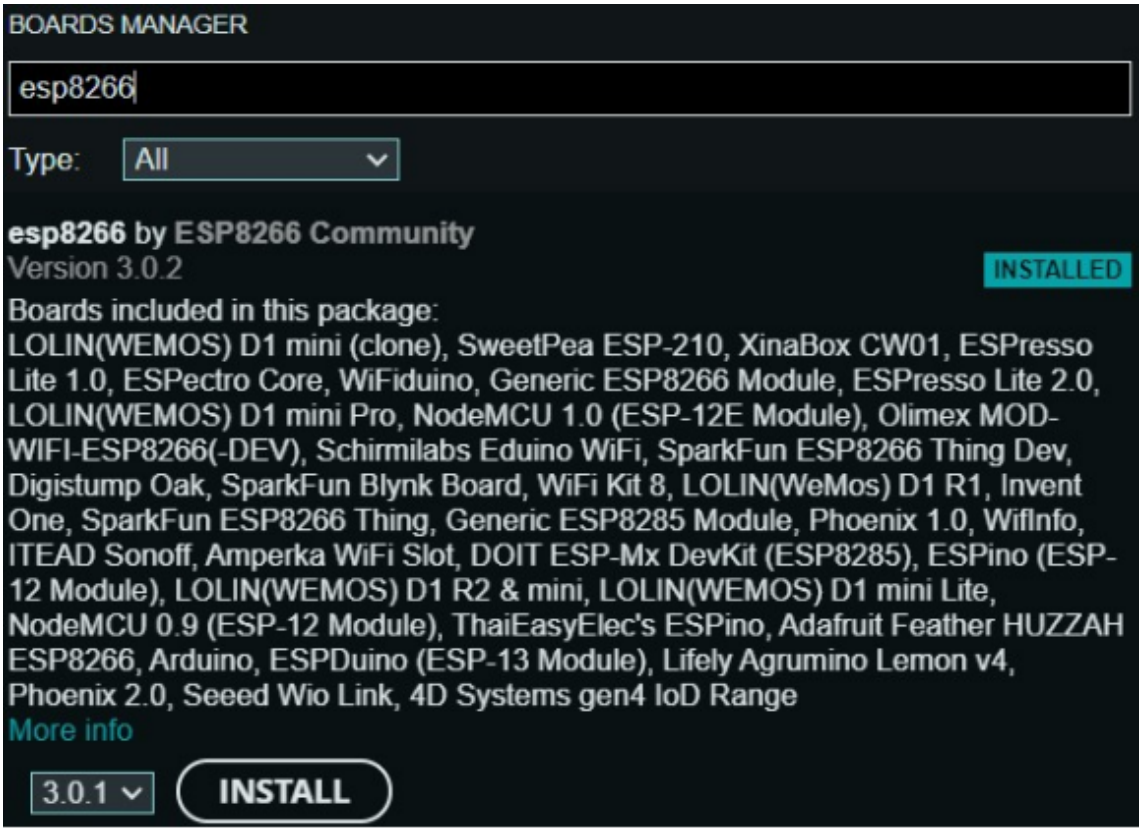
Para descargar el ide de Arduino hay que seguir unos pasos muy sencillos. Simplemente visitar su página web o clicar en el siguiente enlace: https://www.arduino.cc/en/software.

Como vamos a utilizar la ESP8266 y utilizamos el ide de Arduino es necesario hacer que este último reconozca a la primera. Por tanto es necesario entrar en el siguiente repositorio de gitHub: https://github.com/esp8266/Arduino.

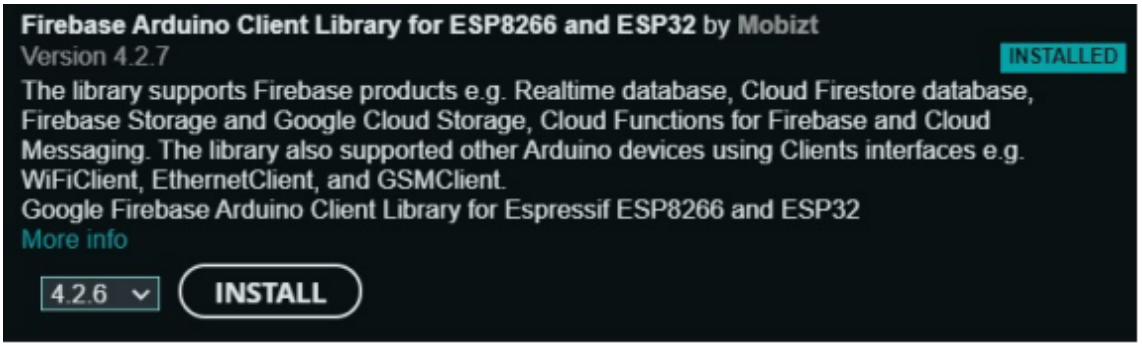
En el, en el apartado de contents encontramos una url que debemos copiar. En este momento la URL es: https://arduino.esp8266.com/stable/package_esp8266com_index.json.

Después de copiar la URL debemos ir al ide de Arduino y en el apartado de Archivo → Preferencias y pegamos la URL en el apartado Gestor de URLs Adicionales de Tarjetas.

Por último, en el gestor de tarjetas debemos buscar esp8266 y descargar el paquete que aparece:

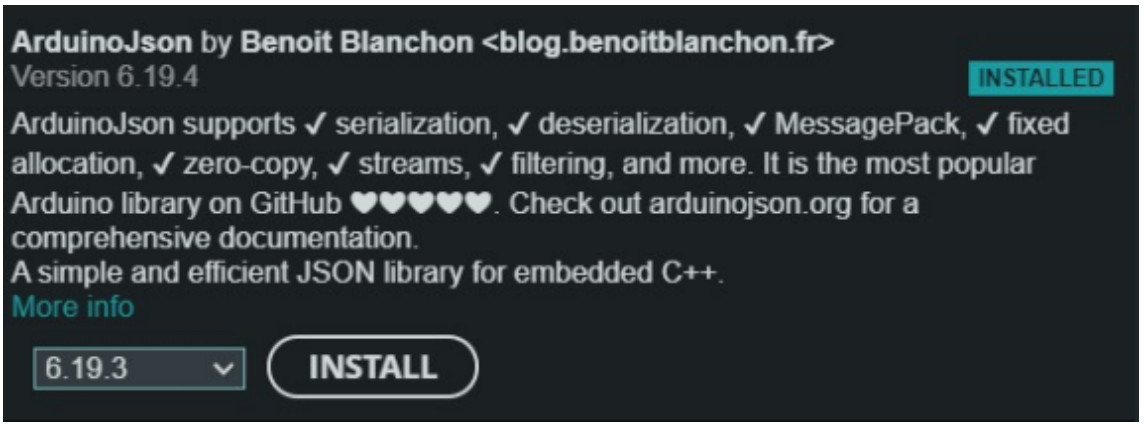



También se deberán instalar varias varias librerías para el correcto funcionamiento de los programas, las cuales se encontrarán en el gestor de librerías. La primera de ellas es la siguiente:




Esta librería es necesaria ya que es necesaria para que la ESP8266 se pueda conectar con una base de datos de Firebase.

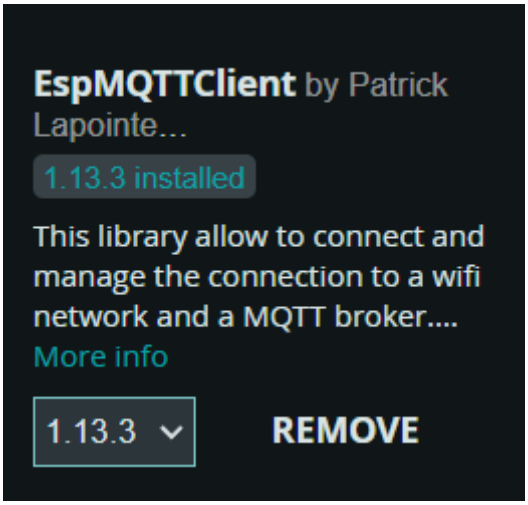
La siguiente sería:

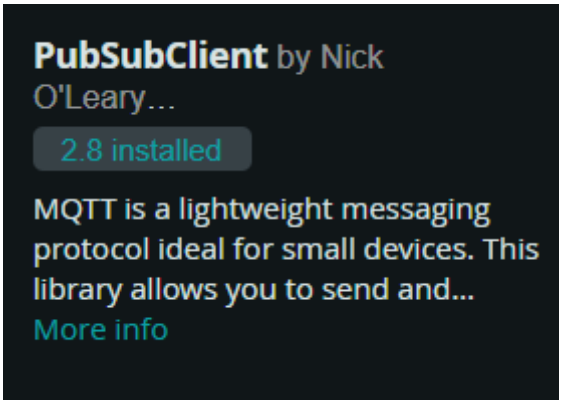



Esta librería nos permite crear JSONs, los cuales son necesarios ya que en nuestro proyecto enviamos la información a la base de datos en formato JSON.

Debemos instalar otra librería, pero esta no se encontrará en el gestor de librerías. Se trata de la librería NTPClient de taranais. Para ello debemos entrar en el siguiente repositorio de gitHub y descargar el código en una carpeta en formato ZIP: https://github.com/taranais/NTPClient

También debemos instalar otras dos librerías para que funcione el MQTT:






## Archivo Python.

Para poder usar el protocolo de comunicaciones MQTT en Python es necesario importar en el archivo .py e instalar la biblioteca: paho-mqtt 1.6.1, lo podemos encontrar en https://pypi.org/project/paho-mqtt/

Desde la terminal habría que introducir el siguiente comando:

- pip install paho-mqtt

## Archivo Red Neuronal.

Para poder usar la red neuronal que se aplica al proyecto es de vital importancia descargarla e importarla en el archivo JavaScript.

Se descarga desde la siguiente pagina web: https://sourceforge.net/projects/brain-js.mirror/

## Base de Batos Firebase.

Para poder acceder a cualquiera de las dos bases de datos implementadas, es necesario importar los siguientes comandos en los archivos, se utiliza JavaScript, por ello tiene la siguiente sintaxis:

import { getAuth } from 'https://www.gstatic.com/firebasejs/9.14.0/firebase-auth.js'

import { getFirestore } from 'https://www.gstatic.com/firebasejs/9.14.0/firebase-firestore.js'

También será necesario configurar la base de datos en el script, para la base de datos en tiempo real tendrá la siguiente información:

apiKey: "AIzaSyAaua17lgfeQY6Ji7svpyCofhWc-oecwqI",

authDomain: "basededatos-f1968.firebaseapp.com",

databaseURL: "https://basededatos-f1968-default-rtdb.europe-west1.firebasedatabase.app",

projectId: "basededatos-f1968",

storageBucket: "basededatos-f1968.appspot.com",

messagingSenderId: "558794073723",

appId: "1:558794073723:web:0490583584bf5ce831e4de",

measurementId: "G-80G882XZZB"

y para la base de datos Firestore:

apiKey: "AIzaSyCi_RJboqrKT4GL2PGHxtsW6pCxTGLl1BU",

authDomain: "base2-9c182.firebaseapp.com",

projectId: "base2-9c182",

storageBucket: "base2-9c182.appspot.com",

messagingSenderId: "10360052669",

appId: "1:10360052669:web:3f881cf35279fdf719d0d4"

## Servidor e Interfaz web.

Para instalar Node.js deberemos acceder a la página web oficial ,donde descargaremos la versión compatible con nuestro sistema operativo.

Junto a Node.js se nos instalará el gestor de paquetes npm. Para comprobar que se han instalado correctamente, desde una terminal, utilizaremos los siguientes comandos:

- node --version

- npm --version

Haciendo uso del gestor de paquetes npm instalaremos otro gestor de paquetes, yarn utilizando el siguiente comando:

- npm install --global yarn

Para comprobar que yarn se ha instalado correctamente utilizaremos este comando:

- yarn --version

Finalmente, para ejecutar el proyecto deberemos utilizar los siguientes comandos:

1. yarn install.
2. yarn build.
3. yarn start.

El primer comando, yarn install, se encargará de actualizar las dependencias del proyecto, instalando además aquellas que se encuentren en el archivo package.json si no lo estaban previamente.

El comando yarn build se encargará de crear la build de producción del proyecto, preparada para el despliegue de la aplicación.

Finalmente, el comando yarn start pondrá en ejecución el servidor http y se podrá acceder a él a través de cualquier navegador utilizando el localhost ( o la dirección ip 127.0.0.1) como dirección ip y el puerto 3000 (modificable desde el archivo de las variables de entorno, .env).

## Anexo II – Manual de Usuario de la aplicación Web.


*Figura 48: Pantalla de carga.*

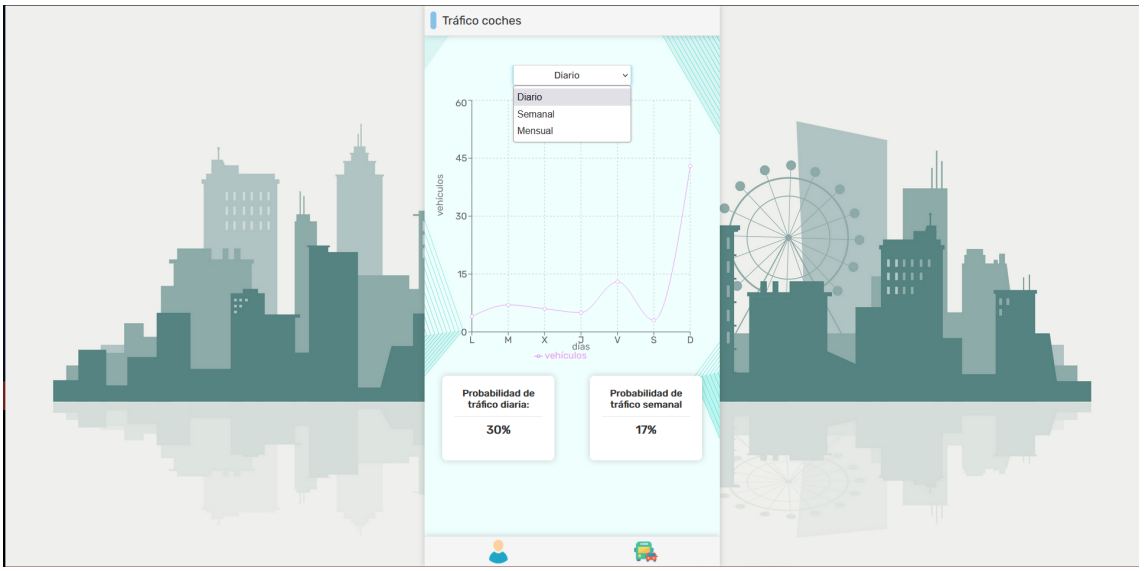



*Figura 49: Pantalla principal.*

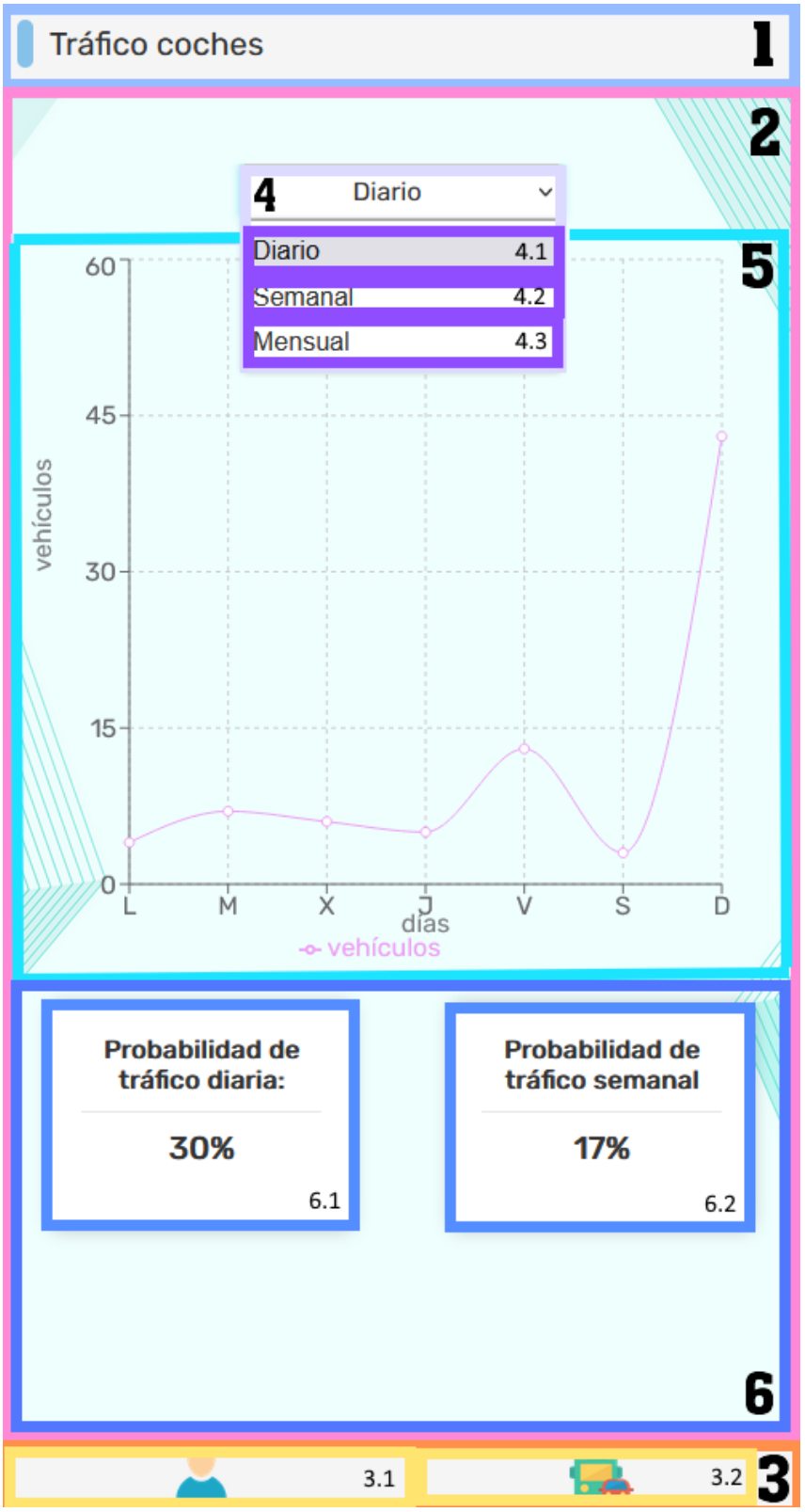



*Figura 50: Estructura de la pantalla principal.*

1. **Header.** Muestra el tipo de la información que se va a mostrar (Tráfico de peatones/Tráfico de coches).
2. **Menú**. Contiene la información a mostrar, en diferentes formatos.

3. **BottomBar**. Permite alternar entre los dos tipos de información disponibles.

   3.1 **Peatones**: icono que, al pulsarlo, permite mostrar la información del tráfico relativa a los peatones.

   3.2 **Vehículos**: icono que, al pulsarlo, permite mostrar la información del tráfico relativa a los vehículos.

4. **Selector temporal**. Permite seleccionar la información a mostrar en un periodo de tiempo determinado.

   4.1 **Diario**. Muestra la información en función de los días de la semana.

   4.2 **Semanal**. Muestra la información en función de las semanas del mes.

   4.3 **Mensual**. Muestra la información en función de los meses del año.

5. **Chart**. Gráfico lineal que muestra la cantidad total de vehículos/personas que han atravesado la calle ,en el eje Y, y el periodo de tiempo en el que lo han hecho, en el eje X.
6. **Predictions**: muestra las predicciones para el tráfico (peatones/vehículos) en diferentes periodos de tiempo.

   6.1 **Probabilidad de tráfico diaria**: muestra la probabilidad ,en porcentaje, de que haya un elevado tráfico ese mismo día.

   6.2 **Probabilidad de tráfico semanal**: muestra la probabilidad media, en porcentaje, de que haya un elevado tráfico a lo largo de la semana.

## Anexo V – Hojas de características de los componentes

Led → https://www.farnell.com/datasheets/1498852.pdf

ESP8266 → https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf

Sensores de presión → https://www.electronicaembajadores.com/datos/pdf1/ss/ssfr/fsrguide.pdf