



UNIVERSITAT POLITÈCNICA  
DE CATALUNYA  
BARCELONATECH

PTIN  
Proyecto GESYS

Documentación

# Índice

|  |           |
|--|-----------|
| <b>Desplegament de hardware, seguretat i validació del projecte</b>                              | <b>6</b>  |
| <b>Planificación de escenarios</b>   | <b>6</b>  |
| <b>Planificación materiales coche</b>  | <b>8</b>  |
| <b>Maqueta coche</b>   | <b>10</b> |
| Hacer planos del coche   | 10        |
| Búsqueda de materiales estructura coche  | 10        |
| Plantear materiales coche  | 11        |
| Implementar que el coche camine correctamente  | 11        |
| Maqueta coche  | 11        |
| Habilitar sitio para la raspberry  | 11        |
| Montar coche   | 11        |
| Establecer sistema de funcionamiento del coche   | 13        |
| <b>Estadística consumo central</b>   | <b>13</b> |
| <b>Catálogo y análisis</b>   | <b>15</b> |
| <b>Estimación de tiempos</b>   | <b>16</b> |
| Tiempo hasta X porcentaje.   | 16        |
| X porcentaje en tiempo Y.  | 17        |
| <b>Simulación carga batería coche + muestra del tiempo restante para completar la carga:</b>     | <b>18</b> |
| <b>Simulación descarga batería coche + muestra del tiempo restante de batería:</b>               | <b>19</b> |
| <b>Mejora simulación batería (km restantes)</b>  | <b>19</b> |
| <b>Comunicación de la entrada con el edge</b>  | <b>20</b> |
| Cómo funciona?   | 20        |
| Ejecución del programa   | 21        |
| <b>Número de plazas ocupación</b>  | <b>23</b> |
| <b>Estadísticas de ocupación diaria</b>  | <b>27</b> |
| <b>Horas en que más se usa el coche</b>  | <b>27</b> |
| <b>Estudio sobre crecimiento de vehículos eléctricos en Vilanova i la Geltrú</b>                 | <b>28</b> |
| España   | 28        |
| Cataluña   | 29        |
| <b>Estimación de número de zonas de recarga eléctrica para vehículos en Vilanova i la Geltrú</b> | <b>30</b> |
| <b>Estimación número de plazas por zona de recarga en vilanova i la Geltrú</b>                   | <b>31</b> |
| <b>Mapa de Vilanova i la Geltrú con las centrales situadas</b>                                   | <b>32</b> |
| <b>Caso de uso electrolinera</b>   | <b>34</b> |
| <b>Estudio tipos de coches</b>   | <b>34</b> |
| <b>Planificación materiales zona de recarga</b>  | <b>35</b> |
| <b>Maqueta zona de recarga</b>   | <b>35</b> |
| Planos de zona de recarga  | 35        |
| Evidencia de maqueta   | 36        |
| Funcionamiento entrada y salida  | 36        |
| <b>Construcción maqueta definitiva</b>   | <b>38</b> |

|   |            |
|---|------------|
| <b>Gestor punto de carga</b>  | <b>39</b>  |
| <b>Tiempo descarga</b>  | <b>40</b>  |
| <b>Tiempo apertura y cierre de la barrera y cálculo del tiempo de paso del coche</b>  | <b>40</b>  |
| <b>Configuración base punto de carga</b>  | <b>41</b>  |
| <b>Simulación averías punto de carga</b>  | <b>42</b>  |
| <b>Cantidad clientes atendidos por un punto de carga</b>                              | <b>43</b>  |
| <b>Comunicación hacia el edge estado punto de y número clientes atendidos</b>         | <b>43</b>  |
| <b>Comunicación hacia el edge el número de plazas libres y ocupadas (definición).</b> | <b>44</b>  |
| <b>Script reconocimiento texto matrícula.</b>   | <b>50</b>  |
| <b>Entrada (barrera+cam)</b>  | <b>67</b>  |
| <b>Mejora en el sistema de la cámara</b>  | <b>68</b>  |
| <b>SIMULACIÓN 32 CARGADORES</b>   | <b>70</b>  |
| <b>Nombre estaciones:</b>   | <b>72</b>  |
| <b>NFC</b>  | <b>73</b>  |
| Datos que transmite el NFC del coche  | 73         |
| Programa de lectura (puntocargaNFC.py):   | 73         |
| Programa de escritura (escribirNFC.py)  | 74         |
| Comunicación con el edge  | 74         |
| Post Comunicación con el edge   | 74         |
| <b>Desplegament d'Infraestructura i Arquitectura</b>                                  | <b>76</b>  |
| <b>INFRAESTRUCTURA</b>  | <b>76</b>  |
| Paradigma edge-cloud  | 76         |
| Actores   | 76         |
| Datos a recibir-enviar  | 77         |
| Comunicación en la infraestructura  | 79         |
| MQTT (message queue telemetry transport)  | 79         |
| <b>ENVÍO DE MENSAJES:</b>   | <b>80</b>  |
| Tipos de mensajes   | 81         |
| QoS   | 81         |
| Ejemplo de petición   | 82         |
| <b>ESQUEMA SOLICITUD RESERVA</b>  | <b>82</b>  |
| <b>ESQUEMA DE ESTADO DE SOPORTE DE CLIENTE-TRABAJADOR</b>                             | <b>83</b>  |
| <b>CREACIÓN DE UN USUARIO</b>   | <b>84</b>  |
| <b>PUBLICACIÓN DE UN NUEVO ELEMENTO EN EL CLOUD</b>                                   | <b>85</b>  |
| Puertos   | 85         |
| MICROSERVICIOS  | 86         |
| <b>Boceto general de la infraestructura</b>   | <b>87</b>  |
| <b>Comunicación con MQTT</b>  | <b>88</b>  |
| <b>API</b>  | <b>101</b> |
| Tecnologías:  | 101        |
| Documentación:  | 102        |
| Fichero .yml  | 102        |
| HEADER  | 103        |
| PATHS   | 103        |

|  |            |
|--|------------|
| COMPONENTES  | 103        |
| GITHUB PAGES   | 107        |
| IMPLEMENTACIÓN   | 108        |
| MAIN   | 108        |
| Servidores de produccion:  | 108        |
| <b>Flask</b>   | <b>112</b> |
| ENTORNO  | 114        |
| Llamada a la API   | 115        |
| ROUTES   | 115        |
| CONTROLLER   | 117        |
| TOKEN  | 117        |
| MARSHMALLOW  | 118        |
| Comentarios API  | 119        |
| Bibliografia API   | 120        |
| Códigos de error y su definición   | 120        |
| <b>DOCKERS</b>   | <b>120</b> |
| Docker para el despliegue del frontend (página de administración)          | 120        |
| Docker NGINX cloud   | 122        |
| Docker broker  | 122        |
| Docker MYSQL   | 122        |
| Docker API:  | 122        |
| <b>Desplegament de servei Web i definició de Protocols</b>                 | <b>124</b> |
| <b>PROTOCOLS</b>   | <b>124</b> |
| <b>1.DNS</b>   | <b>124</b> |
| 1.1 Documentación  | 124        |
| 1.2 Modo de aplicación   | 124        |
| <b>2.API REST</b>  | <b>124</b> |
| 2.1 Documentación  | 124        |
| 2.2 Modo de aplicación   | 125        |
| <b>3.SSH</b>   | <b>125</b> |
| 3.1 Documentación  | 125        |
| 3.2 Modo de aplicación   | 126        |
| <b>4.MQTT</b>  | <b>127</b> |
| 4.1 Documentación  | 127        |
| 4.2 Modo de aplicación   | 128        |
| <b>5.HTTPS</b>   | <b>128</b> |
| 5.1 Documentación  | 128        |
| 5.2 Modo de aplicación   | 128        |
| <b>Protocolo de Reserva / Entrada al Recinto</b>                           | <b>129</b> |
| 6.1. ¿Cómo puedo acceder al recinto?                                       | 129        |
| 6.2. ¿Dónde podemos obtener una reserva y que requisitos hemos de cumplir? | 129        |

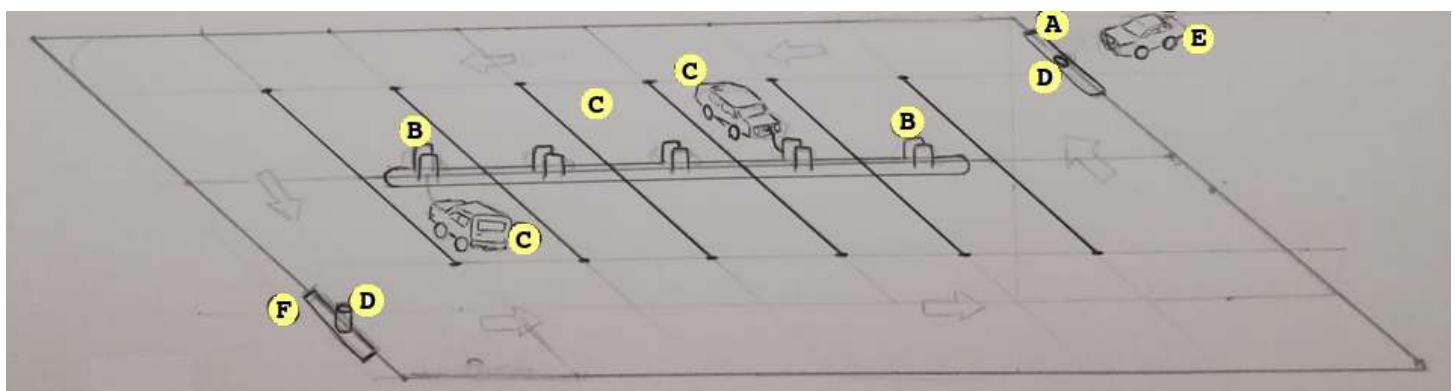
|   |            |
|---|------------|
| 6.3. ¿Qué datos debe proporcionar el cliente para la reserva? | 129        |
| 6.4. ¿Qué debe hacer la aplicación/trabajador?                | 130        |
| 6.5. ¿Cuándo se paga la reserva?                              | 130        |
| 6.6. ¿Una vez tengo la reserva como acceder al recinto?       | 131        |
| 6.7. Gestión de reservas internamente                         | 131        |
| <b>7. Protocolo de multas en una reserva</b>                  | <b>132</b> |
| <b>8. Protocolo de Login</b>                                  | <b>133</b> |
| 8.1. Login en la web  | 133        |
| 8.2. Login en la app  | 133        |
| 8.3. Seguridad en la contraseña                               | 134        |
| <b>WEB</b>  | <b>134</b> |
| LANDING   | 135        |
| LOGIN   | 135        |
| NAVBAR  | 135        |
| PANTALLA DE INICIO  | 137        |
| <b>Barra superior</b>   | <b>138</b> |
| PERFIL  | 138        |
| ESTACIONES  | 138        |
| ESTACIÓN ESPECÍFICA   | 140        |
| TRABAJADORES  | 140        |
| RESERVAS  | 142        |
| RESERVA [EDITAR]  | 143        |
| ESTADÍSTICAS  | 144        |
| PROMOCIONES   | 145        |
| AVERÍAS   | 146        |
| CLIENTES  | 149        |
| SOPORTE TÉCNICO   | 150        |
| <b>Aplicación móvil y base de datos</b>                       | <b>152</b> |
| <b>Aplicación móvil</b>                                       | <b>152</b> |
| Tecnologías utilizadas para el desarrollo                     | 152        |
| Diseño de la aplicación                                       | 153        |
| Implementación de la aplicación                               | 154        |
| <b>Base de datos</b>  | <b>163</b> |

# Desplegament de hardware, seguretat i validació del projecte

## Planificación de escenarios

Por planificación de escenarios entendemos que en este se comprenden:

- espacio destinado a zona de recarga eléctrica para los vehículos
- vehículo



\*\*Nota: esta infraestructura estará provista de un techo cubriendo la zona de estacionamiento para la recarga.

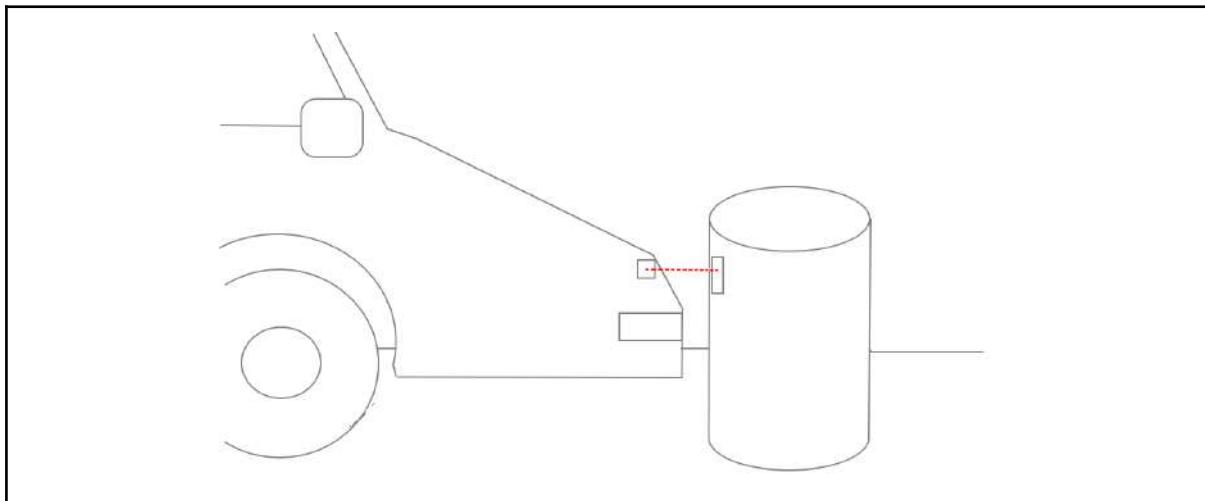
### Explicación de los puntos de la imagen

- |  |
|--|
| <ul style="list-style-type: none"><li>• A: Entrada</li><li>• B: Punto de corriente para efectuar la recarga del vehículo</li><li>• C: Plazas para coches y coche recargando</li><li>• D: Pivotе móvil</li><li>• E: Coche entrando a la zona de recarga</li><li>• F: Salida</li></ul> |
|--|

Es necesario que la instalación pueda identificar los coches a medida que entren o salgan del espacio establecido como zona de recarga.

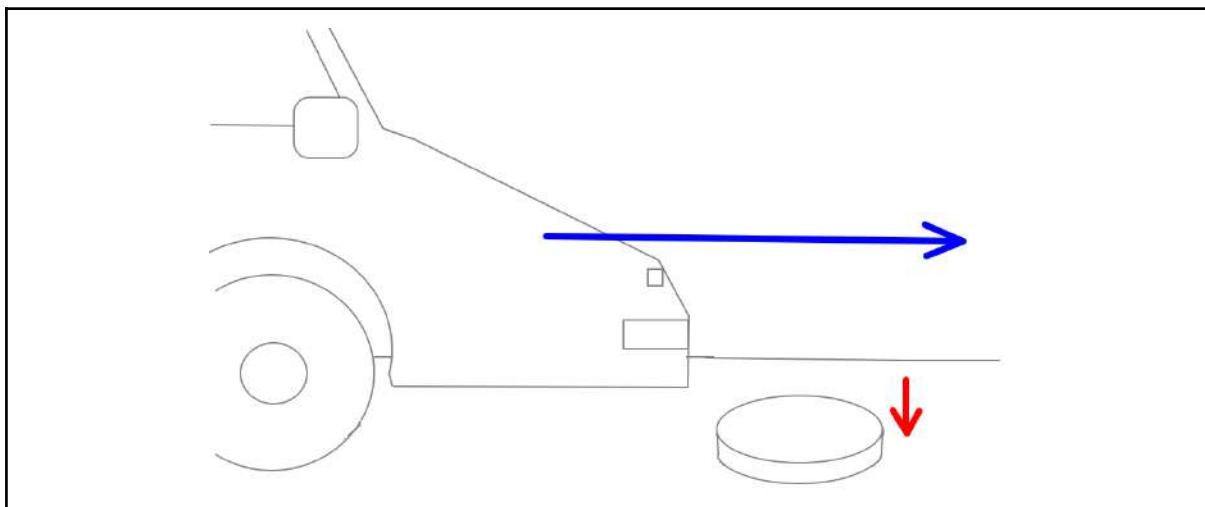
Para esto hemos pensado en un mecanismo de identificación mediante NFC.

Inicialmente hemos pensado que el coche podría tener un dispositivo NFC en el capó, en la parte más delantera posible y los pivotes constan de otro NFC, tal que así:



De esta manera, cuando un vehículo vaya a entrar a la zona de recarga los NFC's mencionados interactuarán y mediante esto se identificará al vehículo.

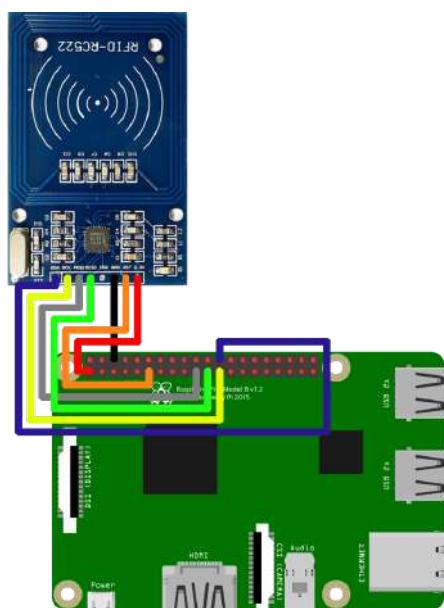
Una vez haya sido realizada la identificación del vehículo el pivote, al ser éste móvil, bajará hasta quedar a la altura del suelo, permitiendo así que el coche pueda pasar dentro a realizar su recarga.



## Planificación materiales coche

Cantidades por coche:

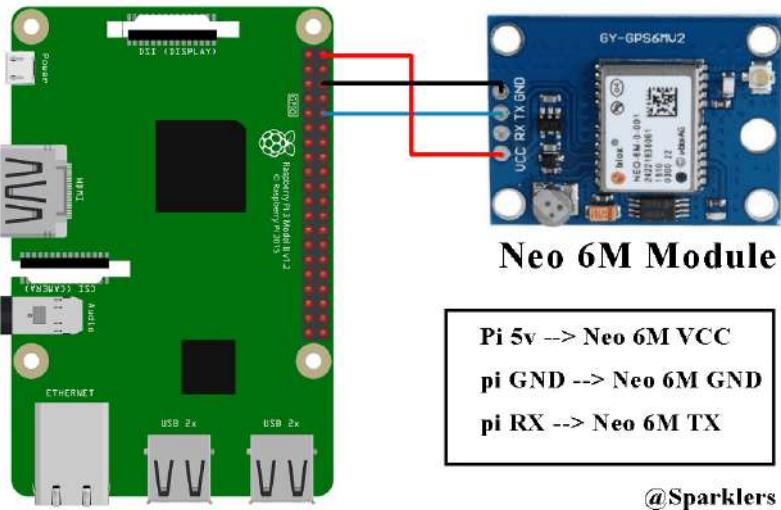
- Sensor NFC x1



- Raspberry Pi 3 Model B x1



- SensorGPS x1



- Rueda (LEGO) x4



Total:

Si queremos hacer 2 coches por lo tanto

- Sensor NFC x2
- Raspberry Pi 3 Model B x2
- SensorGPS x2
- Rueda (LEGO) x8

*Opcionalmente para hacer el coche más bonito podemos usar materiales como plástico, cartón, papel para hacer el chasis del coche.*

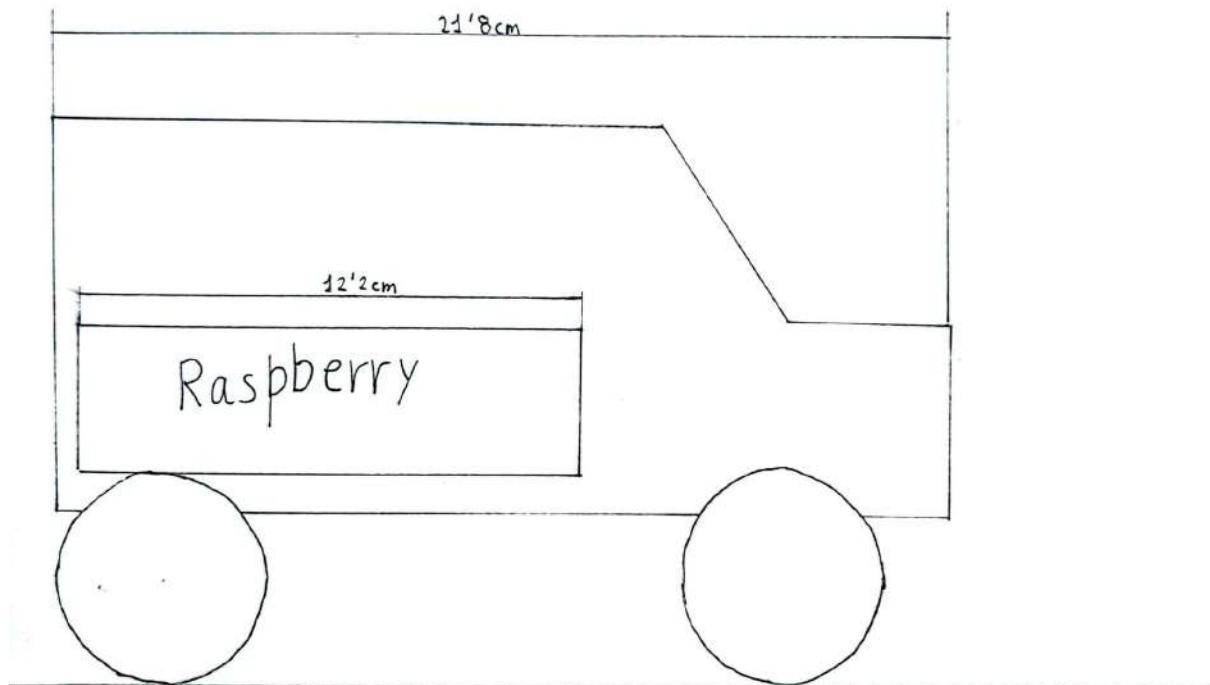
## Maqueta coche

- Hacer planos del coche

Para los planos del coche hemos tenido en cuenta las medidas de la raspberry y de la pila para hacer funcionar el coche.

Una vez se han mirado las medidas de esto, se puede decidir de qué medidas queremos que sea el coche y proceder a hacer los planos, plantillas para luego su construcción.

Ya que la raspberry tiene unas medidas de 12,2 x 7,6 x 3,4cm se ha decidido que el coche tenga unas medidas de 21,8x9x9,3cm. La altura respecto al suelo será de 4cm debido a que las ruedas de las que disponemos son de 5cm de diámetro.



- Búsqueda de materiales estructura coche

El coche tiene que cumplir con un mínimo de resistencia debido a sus posibles golpes y al peso de los componentes que lleva el coche dentro, la temperatura de los componentes que pueda llevar el coche dentro también influyen mucho a la hora de escoger el material del que debe estar hecha la maqueta. Por todos estos motivos se cree oportuno que el coche esté hecho de madera. La madera es resistente al calor, difícil de romper o rayar, y aguantará bien el peso de los componentes.

- Plantear materiales coche

Para los materiales del coche necesitaríamos sobre todo una raspberry y una matrícula lo más nítida posible para que pueda ser leída por el lector de matrículas. Para que el coche pueda andar solo necesitamos una pila, un botón, un mini motor, cuatro ruedas, una goma para unir las ruedas de delante con el motor y dos barras pequeñas y finas de hacerlo para hacer de ejes rotatorios de las ruedas.

- Implementar que el coche camine correctamente

Gracias a un motor y a 4 ruedas junto con una goma que une el motor con una de las ruedas el coche es capaz de andar. El coche lleva incorporado un pequeño botón para encender y apagar el motor del coche. El coche funciona gracias a la potencia de una pila la cual está conectada al motor.

- Maqueta coche

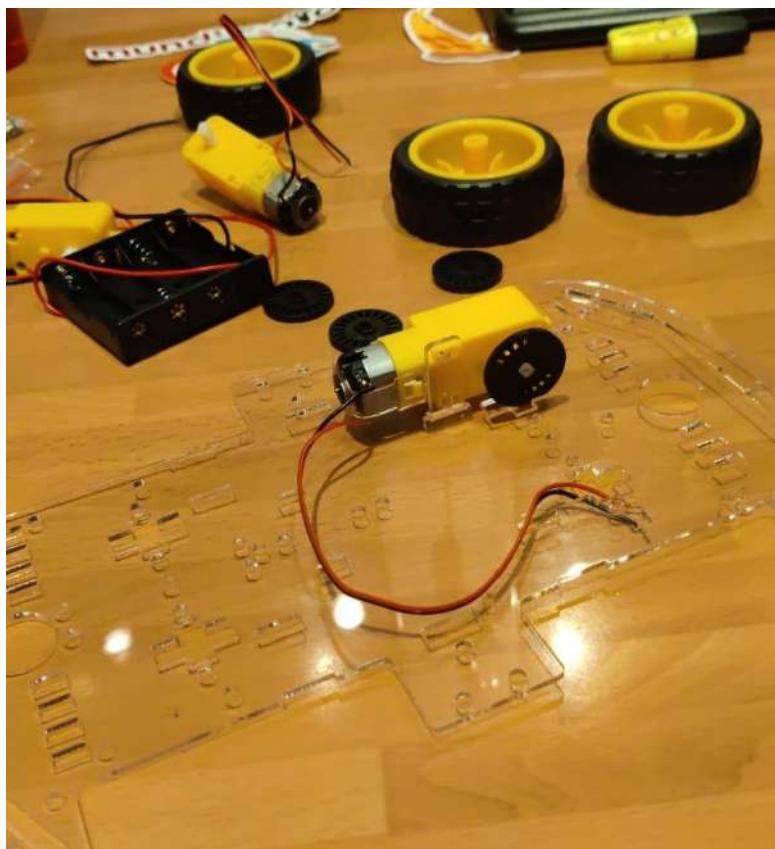
La maqueta del coche ha sido fácil de realizar gracias a que ya teníamos los materiales, estos han sido reciclados de otros proyectos personales que teníamos como el motor y las ruedas.

- Habilitar sitio para la raspberry

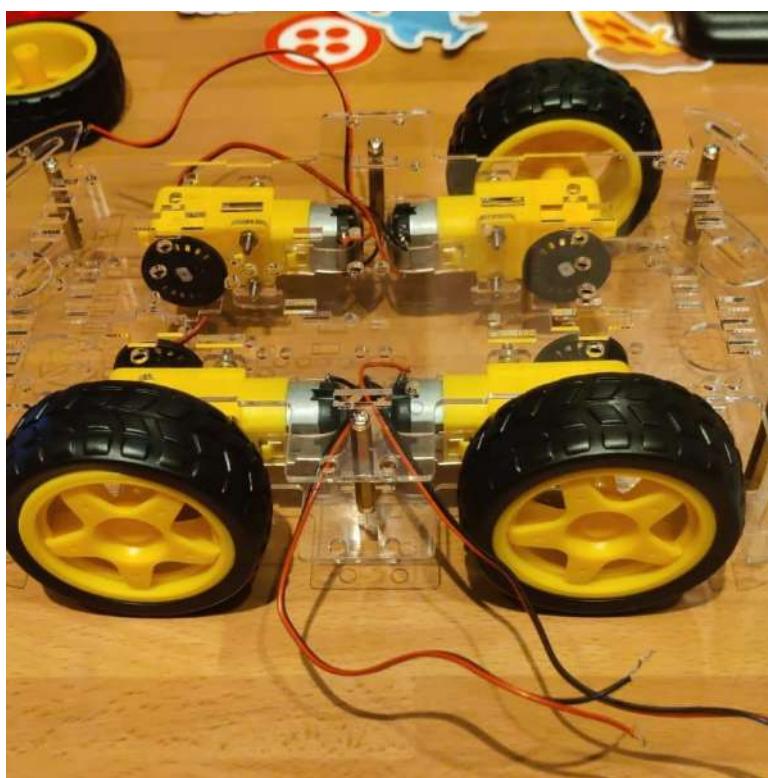
Se ha dejado un sitio lo suficientemente grande para las dimensiones que tiene la raspberry. Debido a que ya no usaremos un nfc no es necesario dejar más sitio del que ya tenemos. El coche se ha diseñado a esta escala para que sea posible la inserción de la raspberry dentro suyo sin ningún problema.

- Montar coche

Para montar el coche hemos seguido las instrucciones que venían dentro de su caja. Seguidamente podemos ver una foto del proceso y otra del resultado final:



Finalmente el resultado fue el siguiente aunque una rueda no queda muy sujetada y a veces se cae:



- Establecer sistema de funcionamiento del coche

Con el uso de un arduino one, un bluetooth, un L924 y unos cables para conectarlo con el motor hemos podido hacer posible que el coche sea controlado a través del móvil.

## Estadística consumo central

Dependiendo de:

La carga rápida suele ser en corriente continua, para la que hay normalizadas una serie de potencias (50 kW y 100 kW son carga rápida, 150 kW carga super-rápida, 350 kW carga ultra-rápida). Por otro lado, las cargas semi-rápidas suelen ser en corriente alterna: 11 kW o 22 kW. En algunos países del norte de Europa, estas dos últimas potencias están muy estandarizadas en los hogares unifamiliares, si bien en España no es tan común.

*(Suponemos que los parques de carga tienen más potencia de carga que los hogares estándares)*

Dentro de unos años, lo más probable es que haya muchos coches eléctricos con baterías de unos 65 kWh de capacidad (es decir, unos 400 km de autonomía reales en ciclo mixto). A 11 kW, el tiempo de carga de esta batería sería de unas 6 horas, mientras que a 22 kW de 3 horas.

*En el caso de la carga rápida, se llegaría al 80% de carga en aproximadamente 1 hora y 20 minutos a 50 kW; 40 minutos a 100 kW; 25 minutos a 150 kW; y 10 minutos a 350 kW.*

| KW  | 80% (recomendado para cargar) | 100% (carga completa) |
|-----|-------------------------------|-----------------------|
| 50  | 1h 20 min                     | 1h 40 min             |
| 100 | 40 min                        | 50 min                |
| 150 | 25 min                        | 31 min                |
| 350 | 10 min                        | 13 min                |

La mayor parte de los coches eléctricos del mercado pueden cargar como máximo a 7,4 kW de potencia en monofásica, a veces incluso menos. Dependiendo de nuestras necesidades, podremos contratar diferentes potencias: 2,3 kW; 3,45 kW; 4,6 kW; 5,75 kW; 6,9 kW... (estas son las antiguas potencias estandarizadas, ahora cada usuario puede contratar la potencia exacta que desee conveniente). Así, si conectamos el coche a diario o cada pocos días, una potencia de 3,45 kW debería servir de sobra, si bien sería recomendable tener algo más de margen y apostar por 4,6 kW o 5,75 kW.

Las electrolineras son puntos de suministro de energía donde poder cargar el coche eléctrico y que son muy útiles fundamentalmente en viajes largos.

Los planes europeos para la electrificación del coche prevén la instalación de electrolineras dentro de las gasolineras convencionales, así como la creación de parques de recarga.

Normalmente se trata de puntos de carga rápida que funcionan a una potencia superior a los puntos de recarga domésticos. Por eso es importante que el coche eléctrico sea compatible con potencias de carga por encima de los 150 kW en corriente continua.

Suponiendo que la media de coches usarán los 150 kW en corriente continua, tardarán en cargarse completamente de entre 35 a 40 minutos:

#### Cálculos:

Cálculo carga completa a partir de datos de carga del 80%

$$\begin{aligned} \text{cargacompleta}(n) &= \frac{\text{listatiempos}(n) \cdot 1}{0,8} \\ \text{cargacompleta}(150\text{kW}) &= \frac{30 \cdot 1}{0,8} = 37,5 \simeq 38 \text{ minutos} \end{aligned}$$

$$kw = kwmax * n \text{ estaciones de carga}$$

$$kwh = kw * h$$

1 vatio equivale a consumir 1 Julio durante 1 Segundo. Por tanto, 1 kW, representa el consumo de 1.000 Julios durante 1 Segundo. Por ejemplo, un equipo que consume 1 kW y esté encendido 1 hora, gastará un total de 3.600.000 Julios. Esto equivale a decir que ha gastado **1 kWh (kilovatiohora) de energía**.

Para calcular el coste energético habrá que usar el valor kW para multiplicarlo por el número de horas que la estación de carga estará operativa.

#### Ejemplo:

Supongamos que pasarán una media de 10 coches en un solo puesto de carga, es decir 40 minutos x 10 = 400 minutos = 6,66667 horas haciendo así el consumo de 150 x 6,666 en

Teniendo en consideración los coches que tendrán carga más rápida deberíamos tener en cuenta el porcentaje de coches que usarán una potencia o otra, dependiendo del estudio.

#### Precio:

Cargar la batería de 40 KWh de un coche eléctrico al 100% costaría alrededor de 6 €

La recarga para 100 km podría oscilar entre menos de 1 euro y 2,00 euros

### Tipos de carga de coche

<https://www.lugenergy.com/modos-de-recarga-vehiculos-electricos/>

## Catálogo y análisis

El objetivo de este apartado es básicamente analizar cómo se encuentra ahora mismo el mercado y las situaciones de los coches más vendidos con la finalidad de poder especular de momento las diferentes funciones de estimación de tiempo, como por ejemplo cuánto tardará en cargar hasta que se llene el margen restante de batería y/o lo que se cargará en un determinado tiempo.

Lo investigado entonces son los diferentes tipos de conectores, los diferentes valores de potencia máxima aceptada para determinados coches y todo esto lo resumimos en unas tablas donde tenemos sintetizada dicha información.

Tenemos tablas con diferentes modelos de coches, la capacidad útil de su batería y diferentes valores estimados de carga que muchas veces los mismos fabricantes nos proporcionan dependiendo de las capacidades de la torre de carga.

| Tipos de cargadores(potencia) | Potencia  | Tiempo Aprox | Ej. de Bateria. |
|-------------------------------|-----------|--------------|-----------------|
| 1-Super-Lento                 | 1.4-2.3kW | 12H          | 24kW            |
| 2-Convencional                | 2,8-3.7kW | 6-8H         | 24kW            |
| 3-Semirrapida                 | 7,4 kW    | 4H           | 24kW            |
| 4-Rápida                      | 11-300 kW | <2H          | 24kW            |

En la tabla anterior tendríamos un resumen de la más típica clasificación de los diferentes tipos de cargas.

Por una parte los de la categoría 1 no suelen ser muy utilizados a la hora de recargar las baterías. Mientras que las de la categoría 2 suelen ser las más usadas por ejemplo a la hora de recargar las baterías en las casas.

Si vamos escalando hacia arriba, tendríamos las de 7.4 que a no ser que tengas una tarifa contratada bastante elevada, no suele ser muy común encontrarse en sistemas domésticos. Por ende, en puntos de carga, lo más común en ofrecerse suelen ser desde 7.4-11 kW en adelante.

Una vez estudiamos todos estos datos, podemos empezar a pensar y estructurar nuestras funciones de estimaciones de tiempo.

## Estimación de tiempos

Desde el POV de la central de recarga una de las cosas a gestionar sería el tiempo que ocupará cada coche en cada plaza con la finalidad por ejemplo de ofrecer información como en cuanto tiempo tardara en cargarse el coche hasta cierto porcentaje(o en su totalidad) y de esta manera también conoceremos cuando dicha plaza estará desocupada u ocupada.

Para estas estimaciones consideramos dos casos:

- Conocer cuento tiempo tardara en cargarse hasta un porcentaje determinado.
- Cuánto cargará en un tiempo determinado.

### Tiempo hasta X porcentaje.

Este caso es en el caso que el cliente deseé conocer por ejemplo cuento tiempo tardara en cargar su coche hasta un porcentaje deseado.

Para realizar este cálculo de estimación necesitamos conocer los siguientes parámetros:

- Capacidad total(en kWh)
- Porcentaje actual(en %)
- Porcentaje al que quiere llegar(en %)
- Potencia de carga que piensa contratar(en kW de entre las opciones ofrecidas)

Supongamos el siguientes datos:

- Capacidad total = 54,5 kWh
- Porcentaje actual = 25%
- Queremos cargar hasta el 90%
- Contratamos 7,4 kW.

Pasos para realizar la estimación de tiempo.

1. Restamos el porcentaje de carga actual al total
2. Restamos al valor anterior, el porcentaje de diferencia entre el objetivo de carga y el total.
3. Tendríamos la cantidad de energía a cargar y podemos aplicar la fórmula de cálculo de estimación.

Formula a aplicar:

$$T = \frac{\text{Energía a cargar(kWh)}}{\text{Potencia Contratada(kw)}}$$

El pseudocódigo seria el siguiente:

```
//Recibimos parámetros(capacity, actpercenbat, objpercentbat, power).  
//Calculo bateria restante.  
    //bateriaRestante = capacity* ( (100-actpercentbat) /100 ).  
//Calculo energia total a cargar.  
    //energiaCargar = bateriaRestante-capacity* ( (100-objpercentbat)  
/100 ).  
//Aplicar formula de calculo.
```

```
//Retornar valor.
```

## X porcentaje en tiempo Y.

Este sería en el caso de que el cliente disponga sólo de un tiempo Y y quiera conocer cuánta batería se cargaría en dicho margen.

Para realizar esta estimación necesitamos los siguientes parámetros:

- Capacidad total (kWh)
- Porcentaje actual(en %)
- Tiempo disponible (En minutos u hora (especificar))
- Potencia que se desea contratar.

Supongamos los siguientes datos:

- Capacidad total = 54,5 kWh
- Porcentaje actual = 25%
- Tiempo disponible = 40 minutos.
- Contratamos 7,4kW

Pasos para realizar la estimación de carga total:

1. Si el tiempo nos lo dan en minutos los pasamos a horas
  - a.  $\text{tiempoH} = \text{tiempoM} / 60$
2. Calculamos cuánta carga se rellenará en el tiempo determinado.
  - a.  $\text{CargaElectrica} = \text{PotenciaContratada} * \text{tiempoH}$
3. Calculamos que porcentaje de carga es respecto al total dicha quantia:
  - a.  $\text{Porcentaje} = \frac{\text{CargaElectrica} * 100}{\text{CapacidadTotal}}$
4. Sumamos el porcentaje extra respecto al porcentaje que nos da que tenia.
5. Retornamos valor nuevo.

```
//Recibimos parámetros(capacity, actpercenbat, tiempo, formatoTiempo,  
power).  
  
//Tiempo en HORAS ?  
    // 0 : Tiempo = tiempo / 60  
    // 1 : Continue  
  
// Energia recargada  
    // Energia = PotenciaContratada * Tiempo  
  
//Porcentaje respecto al total  
    // Porcentaje = (Energia * 100) / Capacity  
  
//Suma total porcentaje  
    // TotalPorce = Porcentaje + actpercenbat  
  
// Return.
```

Webgrafía:

Cataleg:

<https://www.motor.es/noticias/ranking-ventas-coches-electricos-2021-202175019.html>

<https://www.caranddriver.com/shopping-advice/g32463239/new-ev-models-us/>

<https://cargacar.com/>

(Páginas oficiales de cada marca).

## Simulación carga batería coche + muestra del tiempo restante para completar la carga:

Tendremos 2 formas de indicar la carga de la batería del coche:

La primera trata de indicar hasta qué porcentaje nos gustaría cargar la batería de nuestro coche.

```
def carga_limite(tiempoc):
    global bateria
    if bateria < 100: # Si la bateria es menor a 100 cargar bateria
        limite = 0 # Límite hasta el cual cargar
        while limite < bateria: # Mientras el límite sea menor que el porcentaje actual de bateria
            limite = int(input('\tPorcentaje hasta donde cargar: '))
            tempo = (limite-bateria)*tiempoc #tiempo restante de la carga
            if bateria > limite: # Si la bateria es mayor al límite que queremos llegar = Mal límite
                print('\t Indique un porcentaje válido')
            else:
                while bateria < limite and tempo > 0: # Cargar siempre que la bateria sea menor que el límite
                    try:
                        printad(bateria, tempo)
                        time.sleep(tiempoc) # Esperar tiempo designado
                        clear()
                        bateria += 1
                        tempo -=1
                    except Exception as e:
                        pass
        else:
            print('\t Bateria ya cargada')
```

La segunda forma de cargar será indicando cuánto porcentaje queremos aumentar nuestra batería.

```
def carga_x(tiempoc):
    global bateria
    if bateria < 100: # Si la bateria es menor a 100 cargar bateria
        limite = int(input('\tPorcentaje que quiere cargar: ')) #Indicamos cuánto porcentaje queremos cargar
        tempo = limite*tiempoc #tiempo restante de la carga
        limite += bateria
        #Cargar siempre que la bateria sea menor a la suma de la
        #cantidad a cargar y la bateria actual y menor que 100
        while bateria < limite and bateria < 100 and tempo > 0:
            try:
                printad(bateria, tempo)
                time.sleep(tiempoc) # Esperar tiempo designado
                clear()
                bateria += 1
                tempo -=1
            except Exception as e:
                pass
    else:
        print('\t Bateria ya cargada')
```

Para calcular el tiempo restante lo que hemos hecho ha sido basarnos en el estudio sobre los tiempos de carga del sprint anterior y hemos llegado a la conclusión de que el tiempo

que tardará un coche en cargar de 0% a 100% su batería será de 1h y 40 min = 100 min, por lo tanto, llegamos a la conclusión que cada 1% es 1 min = 60s. Eso sí, para la simulación hemos puesto el tiempo real en módulo 60, por lo tanto, ese minuto se verá como 1 segundo en la simulación.

## Simulación descarga batería coche + muestra del tiempo restante de batería:

Para tener un tiempo de descarga más o menos coherente nos hemos basado en los estudios hechos en el sprint pasado y hemos buscado un poco sobre la autonomía de los vehículos eléctricos. Teniendo estas dos cosas en cuenta, hemos llegado a la conclusión de que, más o menos, suelen tener una autonomía de aproximadamente 400 km. Sabiendo este dato podemos suponer aproximadamente el tiempo de vida de una batería al completo, que sería de unas 5 horas = 300 min. Este tiempo se debe a que no siempre se conduce a 100 km/h, sinó que a veces se conduce a más y, la mayoría del tiempo, a menos, por lo tanto, pensamos que es una estimación de la durabilidad de la batería bastante acertada. Hemos calculado que cada 1% de batería tarda en descargarse 3 minutos = 180 segundos, pero como es una simulación haremos que se descargue 1% cada 3 segundos.

Esta sería la parte del código que simula la descarga de la batería:

```
def descarga(tiempod):
    global bateria
    tempo = bateria*tiempod #tiempo restante descarga
    if bateria > 0: # Si la bateria es mayor a 0 descargar bateria
        while bateria > 0 and tempo > 0: # Descargar siempre que la bateria y el tempo sea mayor a 0
            try:
                # Cortar descarga de bateria cuando se presiona la tecla q
                if keyboard.is_pressed('q'):
                    break
                else:
                    printa(bateria, tempo)
                    time.sleep(tiempod) # Esperar tiempo designado
                    clear()
                    bateria -= 1
                    tempo -=tiempod
            except Exception as e:
                pass
    printa(bateria, tempo)
```

## Mejora simulación batería (km restantes)

Después de hacer un estudio del mercado actual de coches eléctricos, hemos podido hacer una media aproximada de la autonomía de la batería en km y serían unos 400 km aprox. Por lo que en nuestra simulación, 1% de batería equivale a 4 km y, por lo tanto, cada vez que se descargue un 1% de la batería la autonomía restante será de 4 km menos y viceversa cuando estemos cargando el vehículo.

[89%] [███████████]  
Tiempo restante de batería: 267 segundos  
Kilómetros de autonomía restantes: 356 km ]

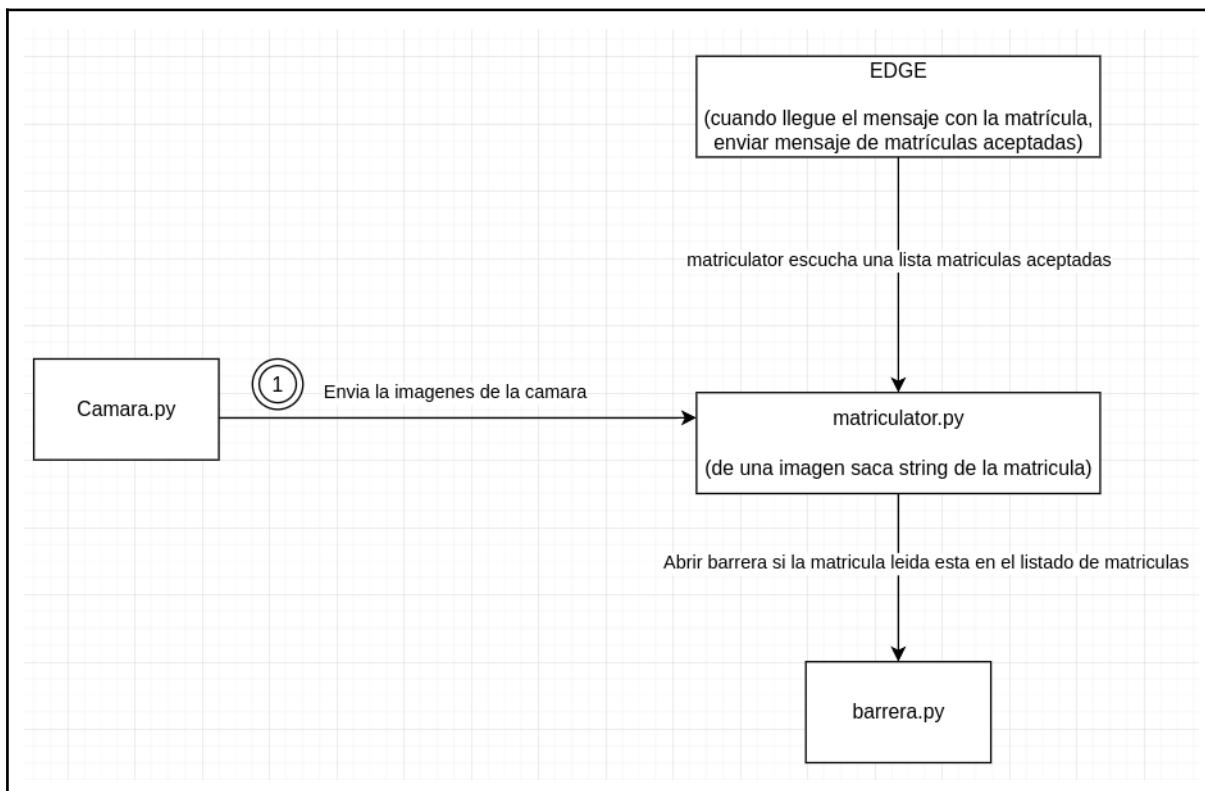
## Comunicación de la entrada con el edge

Hemos pensado cómo queremos el sistema de la entrada.

Tenemos 4 componentes.

- programa encargado del reconocimiento de la matrícula (matriculator.py)
- cámara ( camara.py )
- barrera ( barrera.py )
- edge ( EDGE )

Para acompañar a la explicación de funcionamiento contemplamos el siguiente diagrama:



## Cómo funciona?

1. El edge envía cada segundo una lista de matrículas aceptadas
2. El programa encargado de reconocer la matrícula está escuchando lo que le llega del edge constantemente.
3. Llega un vehículo y camara.py capta la imagen de este y la pasa al programa reconocedor de matrículas.

- El matriculator.py gestiona la imagen y obtiene la matrícula en modo texto y sin espacios (por comodidad) y comprueba en la lista de matrículas aceptadas si la matrícula de la imagen puede pasar o no.
  - Si la matrícula está en la lista de aceptadas el matriculator.py envía la orden a barrera.py de que se abra para dejar pasar al vehículo, se espera el tiempo de paso y luego se baja.
  - Contrariamente, si no se encuentra en la lista, será un vehículo no reconocido y la barrera no se abrirá.

## Ejecución del programa

### 1. Abrir 2 terminales

Primero ejecutaremos el programa en una de las terminales  
**( python3.6 detectamatr.py )**

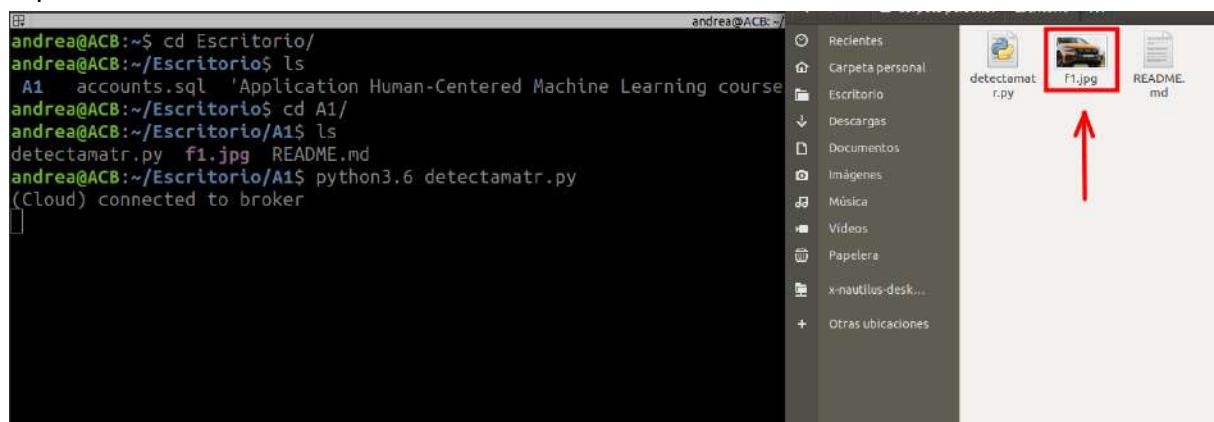
En una le enviaremos la lista de matrículas por el broker

```
( mosquitto_pub -h test.mosquitto.org -p 1883 -t
"gesys/estacion/12/lista_admitidos" --qos 2 -m '{"matr_list": ["4950KZK"]}' )
```



```
andrea@ACB:~/Escritorio/A1$ python3.6 detectamatr.py
(Cloud) connected to broker
```

- La imagen debe llamarse meh.jpg, una vez haya leído la imagen la renombra a f1.jpg (porque ya no la quiere leer más).
- Abrimos la carpeta donde tenemos la imagen que va a leer. Veremos que tiene como nombre el f1.jpg, si tenemos la terminal de la ejecución del programa al lado vemos como no hace nada y solamente nos avisa de que está conectado al broker esperando.

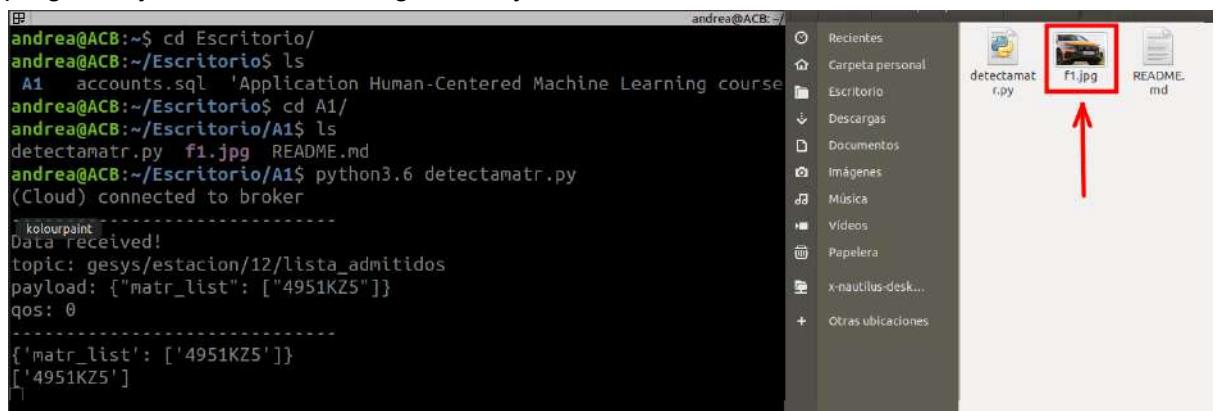


- Desde la otra terminal le pasamos la lista de matrículas aceptadas con la matrícula del vehículo de la imagen y vemos como el programa avisa de que la recibe pero sigue sin gestionar nada.

```
andrea@ACB:~/Escritorio/A1$ python3.6 detectamatr.py
(Cloud) connected to broker
-----
Data received!
topic: gesys/estacion/12/lista_admitidos
payload: {"matr_list": ["4951KZ5"]}
qos: 0
-----
{'matr_list': ['4951KZ5']}
['4951KZ5']

andrea@ACB:~$ cd Escritorio/A1/
andrea@ACB:~/Escritorio/A1$ ls
detectamatr.py f1.jpg README.md
andrea@ACB:~/Escritorio/A1$ mosquitto_pub -h test.mosquitto.org -p 1883 -t "gesys/estacion/12/lista_admitidos" --qos 2 -m '{"matr_list": ["4951KZ5"]}'
andrea@ACB:~/Escritorio/A1$
```

- A un lado ponemos la terminal donde se está ejecutando el programa y al otro la carpeta donde se encuentra la imagen a gestionar.
- Cambiamos el nombre a la imagen f1.jpg por el de meh.jpg y veremos que el programa ya la ha tomado, la gestiona y nos devuelve la matrícula del vehículo.



Cambiamos el nombre:



Vemos que gestiona la imagen meh.jpg y conseguimos la matrícula en modo texto.

```
andrea@ACB:~/Escritorio/A1$ python3.6 detectamatr.py
(Cloud) connected to broker
-----
Data received!
topic: gesys/estacion/12/lista_admitidos
payload: {"matr_list": ["4951KZ5"]}
qos: 0
-----
{'matr_list': ['4951KZ5']}
['4951KZ5']
CUDA not available - defaulting to CPU. Note: This module is much faster with a GPU.
4950KZK
['4951KZ5']
```

7. Comprueba si la matrícula se encuentra en la lista que le ha llegado del broker y como está sacará un mensaje de “abriendo barrera...”, se esperará 20 segundos (tiempo para que pase el vehículo, para la demo lo hemos bajado a 5 segundos pero en un futuro se mirará si el coche ya ha pasado con la cámara ) y procederá a cerrarla avisando con el mensaje de “cerrando barrera...”

```
andrea@ACB:~/Escritorio/A1$ python3.6 detectamatr.py
(Cloud) connected to broker
-----
Data received!
topic: gesys/estacion/12/lista_admitidos
payload: {"matr_list": ["4950KZK"]}
qos: 0
-----
{'matr_list': ['4950KZK']}
['4950KZK']
CUDA not available - defaulting to CPU. Note: This module is much faster with a GPU.
4950KZK
['4950KZK']
Dentro
Abriendo barrera...
Vehículo con matrícula 4950KZK pasando
Ya ha pasado.
Cerrando barrera...
```

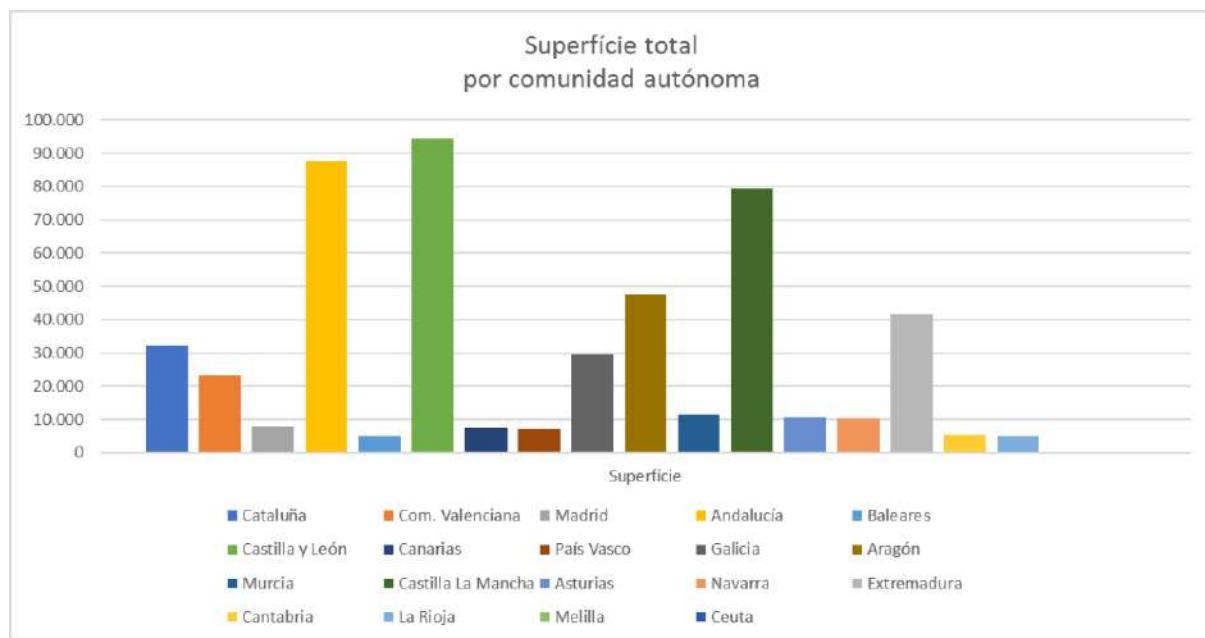
## Número de plazas ocupación

Tomaremos como referencia datos de España:



La media es de 789 puntos de recarga/comunidad

| Comunidad Autónoma | Puntos de carga |
|--------------------|-----------------|
| Cataluña           | 3.549           |
| Com. Valenciana    | 1.661           |
| Madrid             | 1.601           |
| Andalucía          | 1.327           |
| Baleares           | 888             |
| Castilla y León    | 737             |
| Canarias           | 728             |
| País Vasco         | 547             |
| Galicia            | 454             |
| Aragón             | 389             |
| Murcia             | 338             |
| Castilla La Mancha | 295             |
| Asturias           | 239             |
| Navarra            | 218             |
| Extremadura        | 206             |
| Cantabria          | 150             |
| La Rioja           | 69              |
| Melilla            | 9               |
| Ceuta              | 6               |



La media es de 29.764 km<sup>2</sup>/comunidad.

Suponemos que de todos los coches eléctricos que harán uso de puntos de carga, la menor autonomía será de unos 200 km.

Haciendo uso de la superficie media de una comunidad(29.764km) -->

$29.764\text{km}/200\text{km}=148,82=150$  áreas Recarga -> de esta manera todo vehículo puede llegar a cualquier lugar.

Hay un total de 789 puntos de recarga a repartir en 150 áreas, por lo que tendremos 5 puntos de recarga en cada una de las 150 áreas.

En conclusión podemos decir que este valor obtenido, nos sirve como un punto de referencia para futuras reuniones con el cliente (ya que realmente no sabemos en qué lugar nos encontramos (ciudad?, país?)) que podremos escalar una vez conozcamos donde se sitúan las zonas de recarga.

| Comunidad Autónoma | Superficie |
|--------------------|------------|
| Cataluña           | 32.113     |
| Com. Valenciana    | 23.255     |
| Madrid             | 8.028      |
| Andalucía          | 87.599     |
| Baleares           | 4.992      |
| Castilla y León    | 94.224     |
| Canarias           | 7.447      |
| País Vasco         | 7.234      |
| Galicia            | 29.575     |
| Aragón             | 47.720     |

|                    |        |
|--------------------|--------|
| Murcia             | 11.314 |
| Castilla La Mancha | 79.461 |
| Asturias           | 10.604 |
| Navarra            | 10.391 |
| Extremadura        | 41.634 |
| Cantabria          | 5.321  |
| La Rioja           | 5.045  |
| Melilla            | 12     |
| Ceuta              | 20     |

#### Webgrafía

- 1-<https://www.hibridosyelectricos.com/articulo/sector/desigualdades-cargadores-coches-electricos-comunidades-autonomas-espa%C3%B1a/20220222100516054799.html>
- 2-[https://es.wikipedia.org/wiki/Anexo:Comunidades\\_y\\_ciudades\\_aut%C3%B3nomas\\_de\\_Espa%C3%B1a](https://es.wikipedia.org/wiki/Anexo:Comunidades_y_ciudades_aut%C3%B3nomas_de_Espa%C3%B1a)

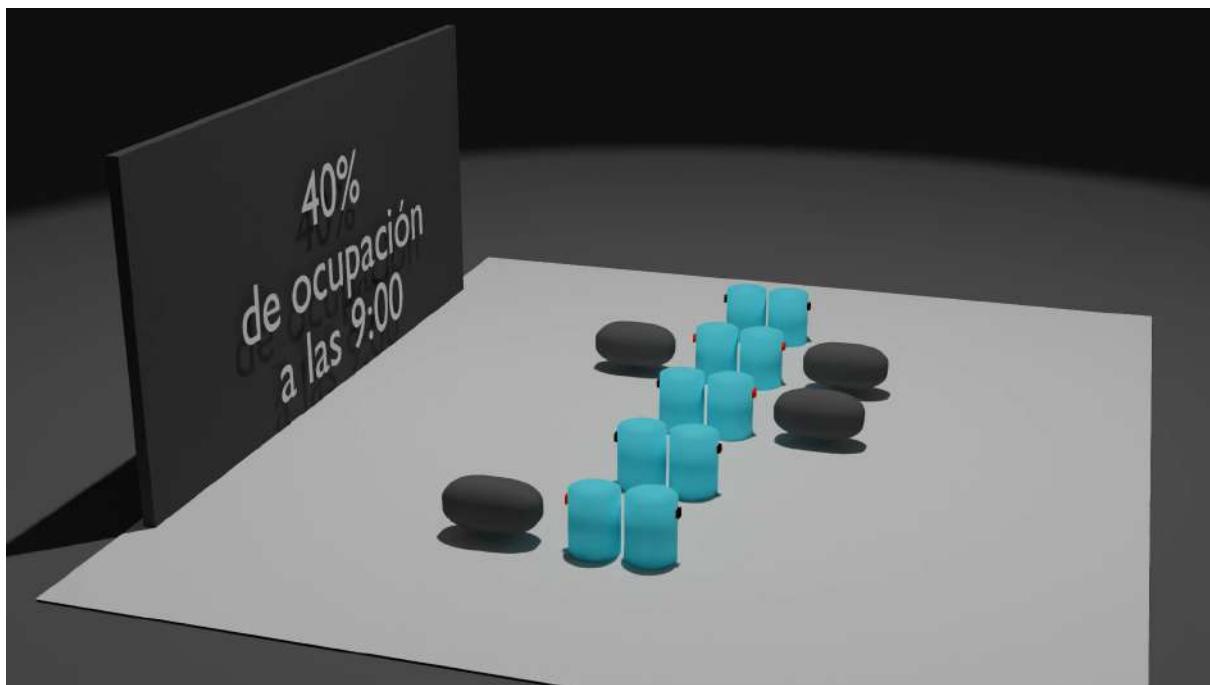
## Estadísticas de ocupación diaria

En función del tiempo medio de recarga (es decir, cuánto tiempo transcurre desde que un vehículo comienza su recarga hasta que sale de la zona de recarga) podemos hacer un recuento de cuántos vehículos hay recargando dentro de una zona de recarga en un determinado instante de tiempo, por ejemplo:

- Imaginemos que de media, los vehículos tardan unos 15 minutos en recargar sus vehículos (desde que entran hasta que salen han transcurrido esos 15 minutos)
- Podemos obtener fácilmente la cantidad de vehículos que hay recargando, si se hace un recuento periódicamente cada 15 minutos.

De cara al propietario, le podemos ofrecer de forma gráfica una función que exprese estos datos. Por ahora lo realizaremos de forma diaria, pero estas mismas estadísticas se pueden realizar en una mayor escala de tiempo (semanas, meses,...) si se desea.

Si nos centramos en el apartado técnico, una forma de hacer ese recuento es haciendo uso de sensores de proximidad(ultrasonido, infrarrojo...) situados en las áreas de recarga de manera que detecten si un vehículo se encuentra recargando en una plaza o no. En la siguiente imagen podemos observar cual es la idea en el funcionamiento de estas estadísticas si por ejemplo se hiciese a las 9:00h:



Video completo:

<https://drive.google.com/file/d/150xxxxy6JvuQJ87c70TbpZZ2-ts8y-uib/view?usp=sharing>

Como podemos ver cada punto de recarga dispondría de un sensor (NFC).

## Horas en que más se usa el coche

Entre semana encontramos que las peores horas para coger el coche son entre las 7:30 y las 9.30 ya que suelen ser las horas en las que la gente suele ir a trabajar, por la tarde sería entre las 17:30 y las 19:30. Los fines de semana la cosa cambia debido a las grandes

salidas que se realizan desde las grandes ciudades hacia las segundas residencias, por eso los viernes de 16 a 22 se crean grandes atascos al salir de las ciudades y los domingos entre las 19 y las 22 en sentido contrario, para volver a la ciudad. En cambio los sábados la mayor concentración de coches estaría entre las 9 y las 13 debido a que la gente suele dirigirse a lugares cercanos a pasar la mañana.

Después de obtener todos estos datos podemos llegar a la conclusión que la mayoría de coches se cargarán al iniciar o finalizar las jornadas de mayor tráfico.

[https://www.niusdiario.es/economia/motor/cuales-son-horas-mas-conflictivas-para-viajar-coche-evn3m\\_18\\_3000120153.html](https://www.niusdiario.es/economia/motor/cuales-son-horas-mas-conflictivas-para-viajar-coche-evn3m_18_3000120153.html)

## Estudio sobre crecimiento de vehículos eléctricos en Vilanova i la Geltrú

### España

La Comisión Europea ha propuesto el fin de la venta de vehículos de gasolina, diésel e híbridos para 2035, la ley de cambio climático española prohíbe su venta en 2040 y el Pacto Verde europeo establece 2050 como la fecha a partir de la cual la Unión debería ser "neutra en emisiones".

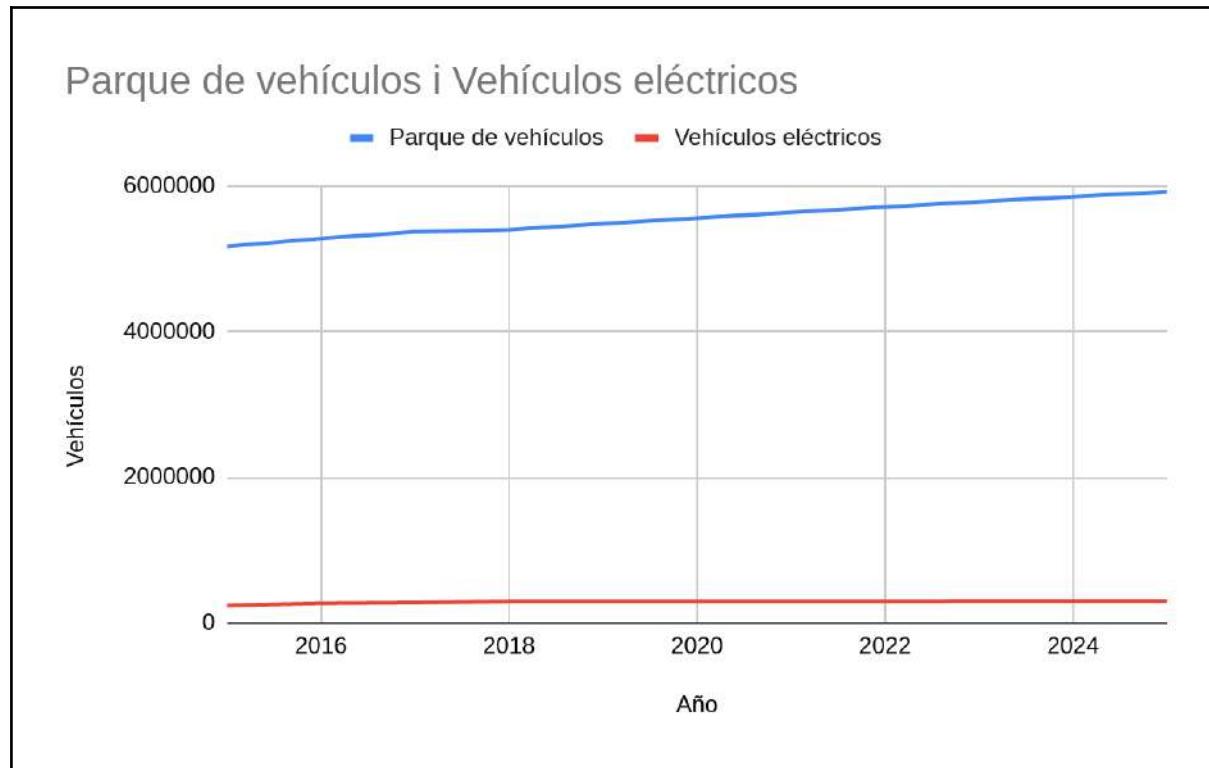
Durante el mes de septiembre de 2021 se matricularon en España 4.190 vehículos eléctricos puros e híbridos enchufables, incluyendo turismos (2.932) pero también motocicletas (540), ciclomotores (442), furgonetas (163), cuadriciclos (80) y otro tipo de vehículos.

En total, es un 23,6% más que en el mismo mes de 2020, pero a este ritmo el crecimiento anual del vehículo eléctrico sería de unas 35.000 unidades al año. En los próximos 9 años, esto supondría 316.656 unidades, lo que unido a los que ya están en las calles apenas superaría el medio millón a finales de 2030.

## Cataluña

Las predicciones de escenarios de vehículos eléctricos en Cataluña entre 2015 y 2025

|                       | 2015      | 2016      | 2017      | 2018      | 2019      | 2020      | 2021      | 2022      | 2023      | 2024      | 2025      |
|-----------------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| Parc total            | 5.167.175 | 5.276.601 | 5.366.036 | 5.396.560 | 5.479.240 | 5.557.214 | 5.633.327 | 5.707.805 | 5.781.585 | 5.849.793 | 5.918.506 |
| Matriculacions totals | 243.990   | 270.753   | 281.318   | 299.668   | 300.662   | 301.311   | 301.834   | 302.373   | 302.989   | 303.430   | 304.046   |

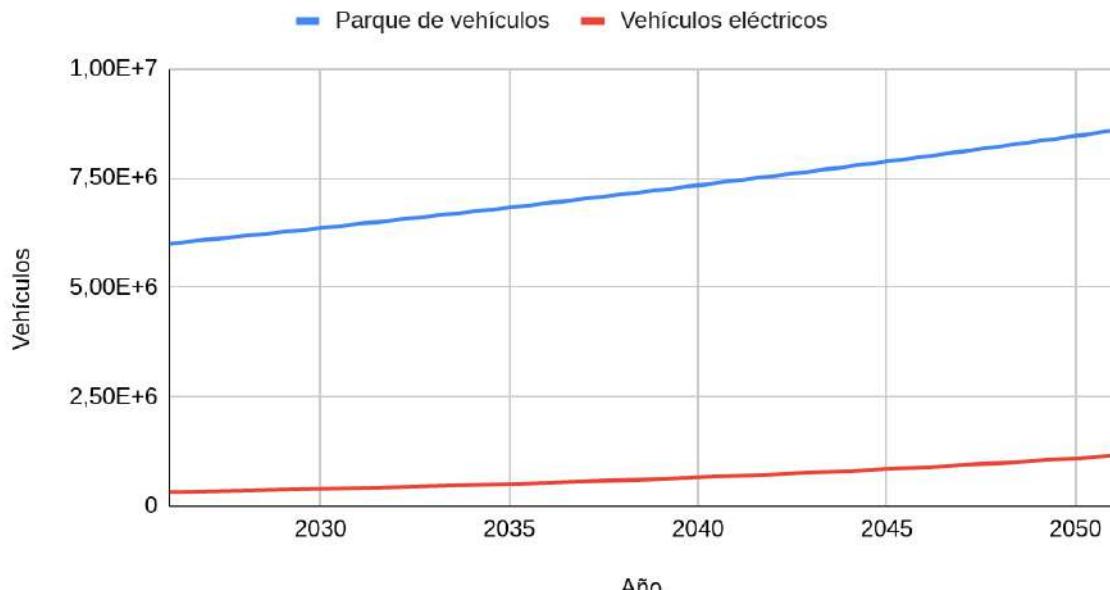


De los datos de la tabla podemos extraer 2 medias, la media de crecimiento de vehículos eléctricos respecto al parque total y la media de crecimiento de parque total de un año al otro con unos valores de 5,24% y un 1,44%, respectivamente.

Por lo que podemos suponer entonces que, si Vilanova i la Geltrú contaba con un total de 45.385 vehículos en 2021, éstos se incrementarán en años futuros un 1,44% cada año, lo que realmente nos supone un menor crecimiento respecto a la velocidad de crecida de los vehículos eléctricos, cuya media de crecimiento anual con respecto al parque total de vehículos es de un 5,24%.

Ergo tendríamos un crecimiento con este perfil:

## Parque de vehículos i Vehículos eléctricos



## Estimación de número de zonas de recarga eléctrica para vehículos en Vilanova i la Geltrú

Entendemos que no se puede estimar el número de zonas de recarga en un ciudad solamente teniendo en cuenta que un coche tiene como autonomía mínima 200km, puesto que no son gasolineras, el coche no se recarga en 5 minutos, tarda horas, incluso.

Dado que no se puede decir un número en base a la autonomía de los coches deberemos partir de un caso familiar y parecido, gasolineras.

Vilanova i la Geltrú tiene unos 34 km<sup>2</sup> y cuenta con 15 gasolineras dispersadas por la ciudad, lo que nos dice que a cada gasolinera le tocan 2,27 km<sup>2</sup>.

Contando que Vilanova tiene aproximadamente 67.458 habitantes censados ( datos del 2021 ) y que el número de vehículos totales es 45.385 podemos suponer que cada gasolinera debería abastecer de combustible a un promedio de unos 3025 vehículos aproximadamente.

| Parque de vehículos    |  |        |
|------------------------|--|--------|
| 2020                   |  |        |
| Turismos               |  | 30.241 |
| Motocicletas           |  | 8.961  |
| Vehículos Industriales |  | 5.344  |
| Otros                  |  | 839    |
| Total                  |  | 45.385 |

Una gasolinera tiene como mucho unas 8 plazas para que los vehículos efectúen el estacionamiento para cargar combustible, pero dicha recarga tarda 5 minutos de media. Esto en los vehículos eléctricos no se puede aplicar, puesto que la recarga de éstos es variable en tiempo, nuestro servicio tardará más.

Para poder estimar el número de zonas de recarga eléctrica en la ciudad deberemos tener en cuenta el número de vehículos eléctricos actuales y también su crecimiento en los próximos años.

Según las suposiciones del estudio de crecimiento para los próximos años el número de vehículos eléctricos incrementará ( crece más rápidamente que el parque total de vehículos de la ciudad ).

Actualmente es de un 5,29% respecto al parque total de vehículos de Vilanova i la Geltrú. Según el estudio se espera que para 2051 el porcentaje de vehículos eléctricos ocupe un 13% de los vehículos totales de la ciudad.

Si en el 2022 tenemos 15 gasolineras para unos 45.385 vehículos no eléctricos, con tiempos de recarga de unos 5 minutos de media, podemos contar con que para los 25.321 vehículos eléctricos del 2051 y teniendo en cuenta que a causa de la necesidad de reducir el uso de combustibles fósiles, el crecimiento del parque de vehículos eléctricos será mayor de lo esperado, necesitaremos al menos 4 zonas de recarga en la ciudad.

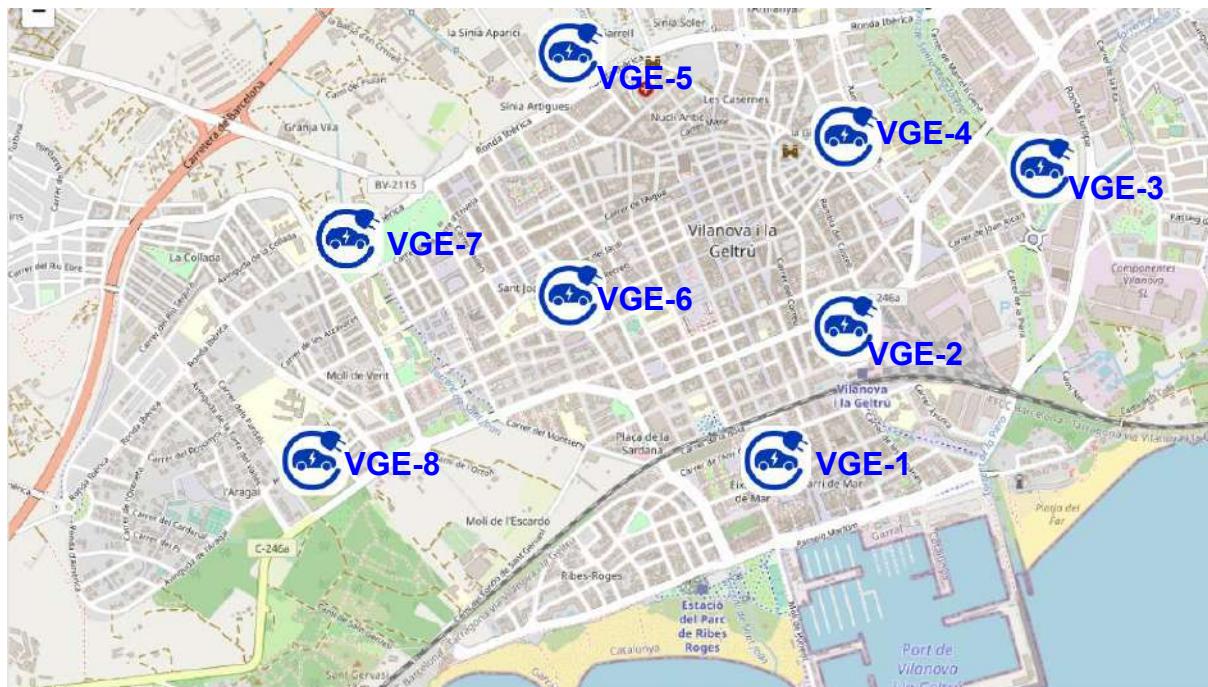
## Estimación número de plazas por zona de recarga en vilanova i la Geltrú

Si para 5 minutos de media en una gasolinera hacen falta 8 plazas, para 40 minutos de media en una zona de recarga eléctrica harán falta 64 plazas.

64 plazas \* 4 zonas = 256 plazas totales, a dividir si conviene, entre más zonas de menor tamaño.

Por ejemplo 256 plazas / 8 zonas = 32 plazas por zona

## Mapa de Vilanova i la Geltrú con las centrales situadas



|             | E1                     | E2                     | E3                     | E4                     |
|-------------|------------------------|------------------------|------------------------|------------------------|
| Coordenadas | 41.217606,<br>1.727072 | 41.221002,<br>1.730369 | 41.225431,<br>1.737627 | 41.227420,<br>1.728166 |

|             | E5                     | E6                     | E7                     | E8                     |
|-------------|------------------------|------------------------|------------------------|------------------------|
| Coordenadas | 41.229674,<br>1.721478 | 41.222119,<br>1.718915 | 41.223434,<br>1.710113 | 41.217122,<br>1.709477 |

Hemos elegido estas 8 localizaciones para asentar las centrales, ya que se reparten bastante bien la superficie de Vilanova. Además hay 4 zonas muy clave:

- La zona E1 estaría situada bastante cerca de la playa, así que, a parte de cubrir la zona de debajo de las vías del tren, también si un día tienes poca batería en el coche y tienes pensado ir a la playa pues puedes aprovechar y reservar para ese rato que vas a estar disfrutando de las maravillosas playas de Vila.
- La zona E2 se situaría, también, en un punto estratégico clave: al lado del edificio A de la EPSEVG. Nos hemos fijado que a día de hoy ya hay un punto de carga y si es así es porque la localización es GOD, por lo tanto; aprovecharemos esta información que tenemos y situaremos una estación de carga allí.

- La zona E5 creemos que sería una de las estaciones más clave, ya que la construiremos delante de un hospital y sabemos que allí siempre hay mucho tráfico de gente y horas en las que el coche no se usa, que se pueden aprovechar para cargar el vehículo.
- La zona E6, sería una situación parecida a la de la E2, pero en este caso es un colegio. Los profesores pueden aprovechar y cargar sus vehículos durante X tiempo mientras están dando clase.

## Caso de uso electrolinera

- El cliente sitúa su vehículo en la entrada(barrera cerrada) de manera que la cámara pueda leer su matrícula.
- Se comprueba que la matrícula sea la correcta(se abre la barrera).
- El cliente entra con su vehículo en la electrolinera.
- Si el cliente no pagó con tarjeta, paga en efectivo antes de realizar la recarga.
- El cliente ocupa su plaza.
- El cliente recarga su vehículo.
- El cliente sale de la electrolinera.

## Estudio tipos de coches

| Modelo                | Marca      | Tipo carga | Capacidad Batería |
|-----------------------|------------|------------|-------------------|
| 500e Cabrio eléctrico | Fiat       | Normal     | 42 kW             |
| Taycan eléctrico      | Porsche    | Rápida     | 93,4 kW           |
| e-tron GT eléctrico   | Audi       | Rápida     | 93,4 kW           |
| Leaf eléctrico        | Nissan     | Normal     | 42,6 kW           |
| Ioniq eléctrico       | Hyundai    | Normal     | 72,6 kW           |
| i3 eléctrico          | BMW        | Normal     | 42,2 kW           |
| ID.3 eléctrico        | Volkswagen | Rápida     | 77 kW             |
| 2 eléctrico           | Polestar   | Normal     | 69 kW             |
| UX300e eléctrico      | Lexus      | Normal     | 54,3 kW           |

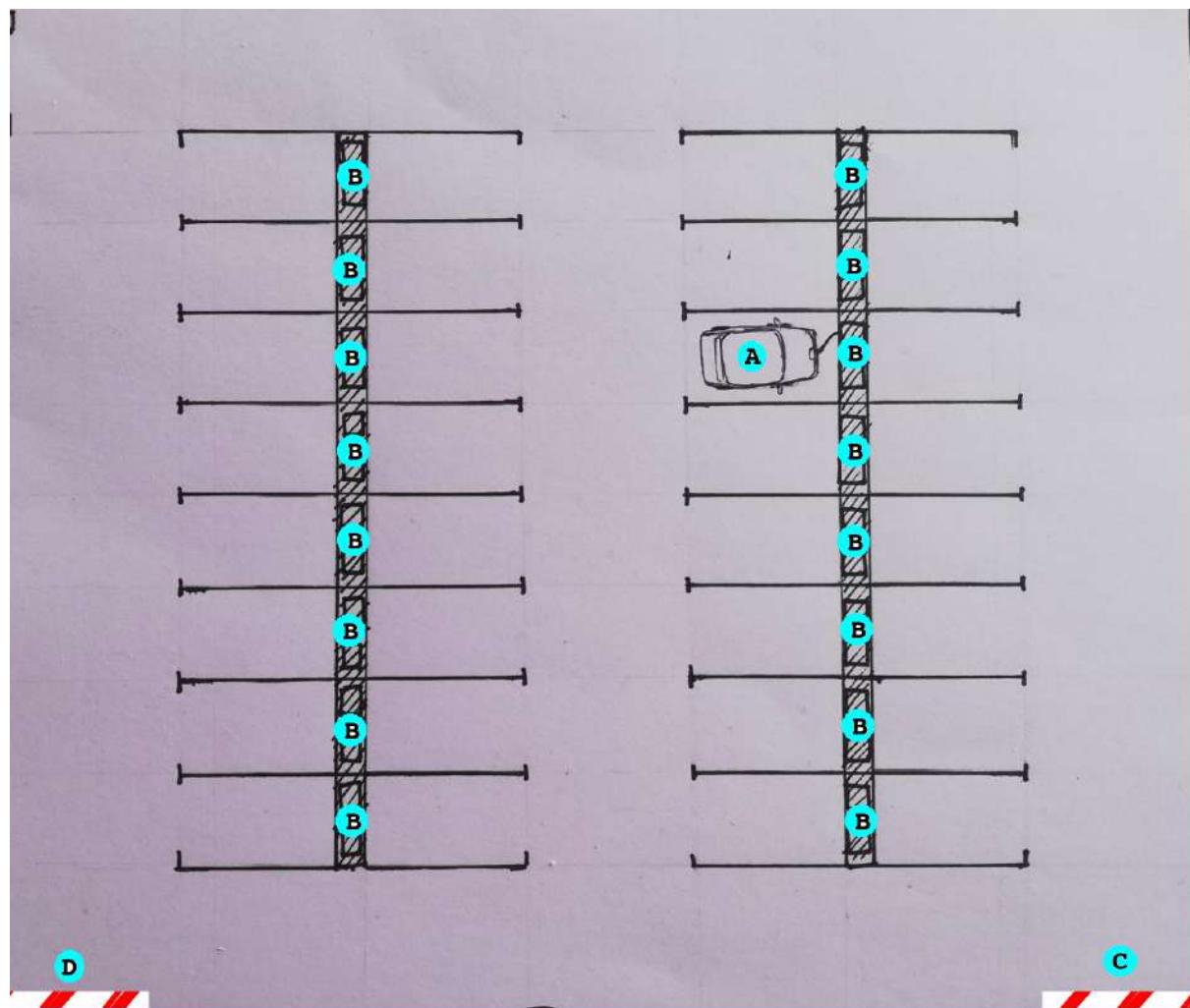
|               |     |        |       |
|---------------|-----|--------|-------|
| EV6 eléctrico | Kia | Rápida | 77 kW |
|---------------|-----|--------|-------|

## Planificación materiales zona de recarga

Los materiales que necesitamos de momento sería una raspberry, de momento un nfc, unos pilones que suban y bajen, los puntos de carga necesarios calculados en otra tarea y columnas junto con un techo para resguardar los coches mientras se cargan. Todo esto es lo principal que consideramos indispensable ahora mismo en un principio. Somos conscientes de que en los siguientes sprints seguramente nos surgirán más necesidades de hardware que tendremos que ir hablando con el cliente técnico.

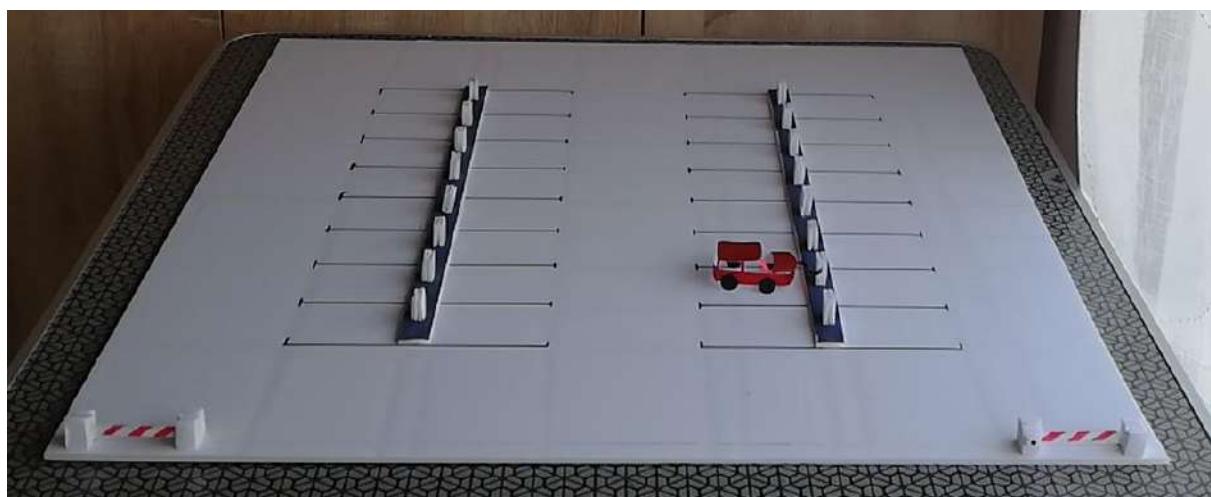
## Maqueta zona de recarga

### Planos de zona de recarga



- A → Coche efectuando una recarga
- B → punto de recarga eléctrica ( uno para las 2 plazas que se encuentran frente a frente)
- C → entrada
- D → salida

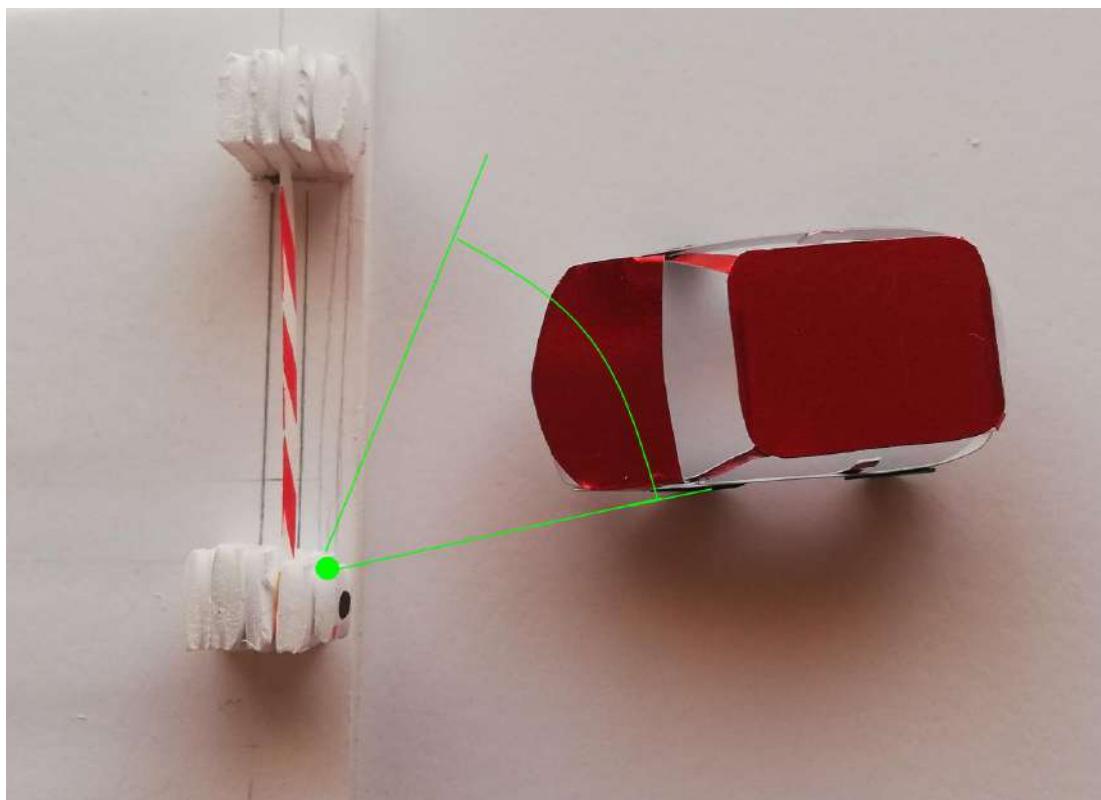
Evidencia de maqueta



### Funcionamiento entrada y salida

Entrada y salida constan de una barrera para dejar o no dejar pasar a los vehículos.  
El funcionamiento es el siguiente:

Las cabinas de los lados de la barrera portan una cámara en la esquina interior izquierda, del lado del que se le aproximan los vehículos.



#### *Campo de visión de la cámara*

La cámara enfoca al coche, capta su matrícula y la pasa a texto para posteriormente poder realizar la identificación del vehículo.

Si lo identifica la barrera se levantará y lo dejará pasar, de lo contrario no se moverá.



*Si identifica la matrícula del vehículo la barrera se levanta y le deja paso a éste*

## Construcción maqueta definitiva

Cuando empezamos el sprint estuvimos barajando varias formas de aguantar la maqueta:

- Triángulos de cartón --> Descartado porque se doblaba, ya que tenía que enganchar los costados para hacer la forma con precinto y no aguantaba bien.
- Corchos --> Descartado porque no tenía suficientes y como la base de estos no era del todo recta pues bailaba).
- Cartón de leche --> Aprobado porque aguantaba bien el peso y levantaba lo suficiente la base para que quepan las cosas abajo.

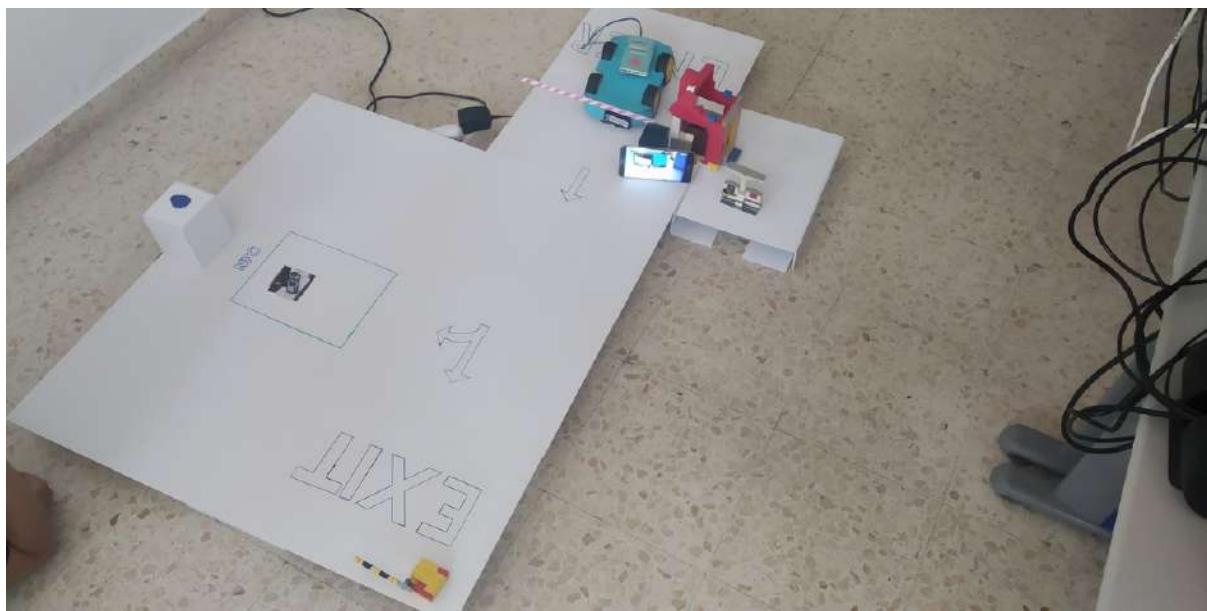
Hemos utilizado para hacer la base 8 placas de medida A3 de cartón pluma juntadas con precinto.

Para mantener elevada la base y que aguante el peso del coche hemos utilizado 9 cartones de leche (estirados) como pilares para aguantar en alto la maqueta ya que como hemos comentado antes es la mejor opción. Hemos hecho pruebas y aguanta bien el peso del coche.

Al final, la estructura que habíamos pensado para la entrada a la central era una rampita que subiera a la estructura general, pero vimos que no era factible, así que lo que hemos hecho ha sido poner la estructura de la entrada, con sus muescas/huecos para cables y sensores, al mismo nivel que el resto de la base.

Para la parte de la entrada, hemos hecho unas muescas en la base de la maqueta para meter los cables de la barrera y que no vayan por fuera. También hemos colocado el sensor de ultrasonidos en un hueco que hemos hecho en la base para que justo cuando el coche se pare encima lo detecte.

La barrera está colocada justo al lado de la cabina. Como al final no hemos podido tener una cámara decente y hemos tenido que utilizar el móvil, así que no hemos encontrado una forma práctica de colocarlo en la maqueta, pues lo hemos apoyado en la estructura de la barrera (sujeto a mejora) para que pueda leer bien la matrícula.



## Gestor punto de carga

### Explicación funcionamiento:

En cada punto de carga se ejecuta un programa *scriptEthernet.sh* encargado de gestionar el proceso de carga para cada nuevo cliente, su principal objetivo es:

1. Atender a cada nuevo cliente
2. Gestionar la mala ocupación de una plaza

Con tal de realizar el primer objetivo, la simulación del proceso de carga se llevará a cabo mediante un cable Ethernet(simula el enchufe), dicho cable siempre estará conectado al punto de carga por un extremo y el otro extremo será el que se conecte al vehículo a cargar. El programa *scriptEthernet.sh* está implementado para ejecutarse permanentemente en un punto de carga a la espera de que un vehículo se conecte al otro extremo del cable Ethernet, una vez se haya conectado, el cliente podrá iniciar su recarga, es decir, *scriptEthernet.sh* ejecutará el programa encargado de la recarga de la batería *simulacionBateria.py*.

Una vez ha finalizado el proceso de recarga (termina el programa *simulacionBateria.py*), el gestor *scriptEthernet.sh* comienza con su segundo objetivo(gestionar mala ocupación de la plaza) haciendo uso de alertas que se mostrarán en la pantalla del punto de carga, estas alertas tienen la finalidad de informar al cliente sobre el fin de la recarga del vehículo y del tiempo que este dispone para desocupar la plaza, en caso de sobrepasar el tiempo límite, comenzará el proceso de llamar al servicio de grúa.

### Detalles complementarios:

Con tal de simular la falla de un punto de carga, ya sea por que el enchufe esté en un mal estado o que el voltaje liberado pueda ser perjudicial para el ser humano, se ha decidido que este hecho suceda de forma totalmente aleatoria, es decir, dado que no tenemos un método real para gestionar un cable en mal estado hemos optado por hacer que todos los puntos de carga tengan una probabilidad de fallo en X tiempo.

## Tiempo descarga

Después de haber visto algunas páginas podemos darnos cuenta de que la autonomía de los vehículos depende muchísimo de las condiciones en las que se esté conduciendo, por esa razón creo que dado que este proyecto es una simulación y no podemos medir estas situaciones podemos intentar hacer un estándar siempre pensando en lo peor posible.

Dado que hemos encontrado diferentes autonomías de coches lo más correcto sería coger una media de 200-250 kilómetros de autonomía

Webgrafía:

<https://www.autobild.es/noticias/asi-quedan-autonomia-coches-electricos-mas-vendidos-nuevo-ciclo-wltp-250680>

<https://www.xataka.com/automovil/cuanta-autonomia-real-tiene-un-coche-electrico>

## Tiempo apertura y cierre de la barrera y cálculo del tiempo de paso del coche

5 sec, 12 sec, 9 sec, 0'7 sec

Hemos estado investigando diferentes gamas y modelos de barreras para controlar la entrada y salida de vehículos de las electrolineras y hemos llegado a la conclusión que hay una gran variedad de modelos, ya que hay algunos modelos que solo son aptos para residencias, parkings, autopistas o usos industriales. Se diferencian por categorías según su longitud y su máximo de ciclos por día. Mirando todas las opciones posibles creemos que el modelo que más se adaptaría a nuestro proyecto sería el de parking ya que al igual que el modelo residencial tiene una longitud de brazo de entre 3 y 4 metros, pero a diferencia del residencial, el tiempo de subida y bajada de la barrera es un poco mayor. Podríamos plantear que fuera más rápido pero creemos que se saldría de presupuesto, ya que a mayor velocidad mayor es el coste de la barrera, por lo tanto, es mejor optar por alguna que tarde un poco más y que sea más económica. El tiempo de apertura y cierre de la barrera promedio en este caso es de 4 segundos respectivamente (8 segundos en total).

Ahora sabiendo el modelo de barrera que podríamos utilizar falta decidir cuánto tiempo se mantendrá la barrera abierta para que le dé tiempo a pasar al coche. Tampoco queremos que el tiempo de entrada total (abrir+cerrar+tiempo abierta) de entrada/salida de un vehículo sea demasiado largo para evitar colas, por lo tanto, en base a un estudio que hemos hecho visitando los diferentes parkings del Mercadona y calculando el tiempo que se mantiene abierta su barrera hemos decidido que el tiempo será de 7 segundos, así el tiempo total de que pase un coche será de 15 segundos.

**Tiempo barrera abierta = 7 segundos.**

**Tiempo en abrir/cerrar barrera = 4 segundos por acción.**

**Tiempo total = 15 segundos.**

## Configuración base punto de carga

Al encender un punto de carga, automáticamente se ejecuta el programa global que inicia todos los procesos necesarios para que funcione correctamente.

Funcionar correctamente quiere decir que el punto de carga puede:

- Detectar un vehículo conectado y suministrarle energía
- Comunicarse con el edge
- Averiarse (simulado)

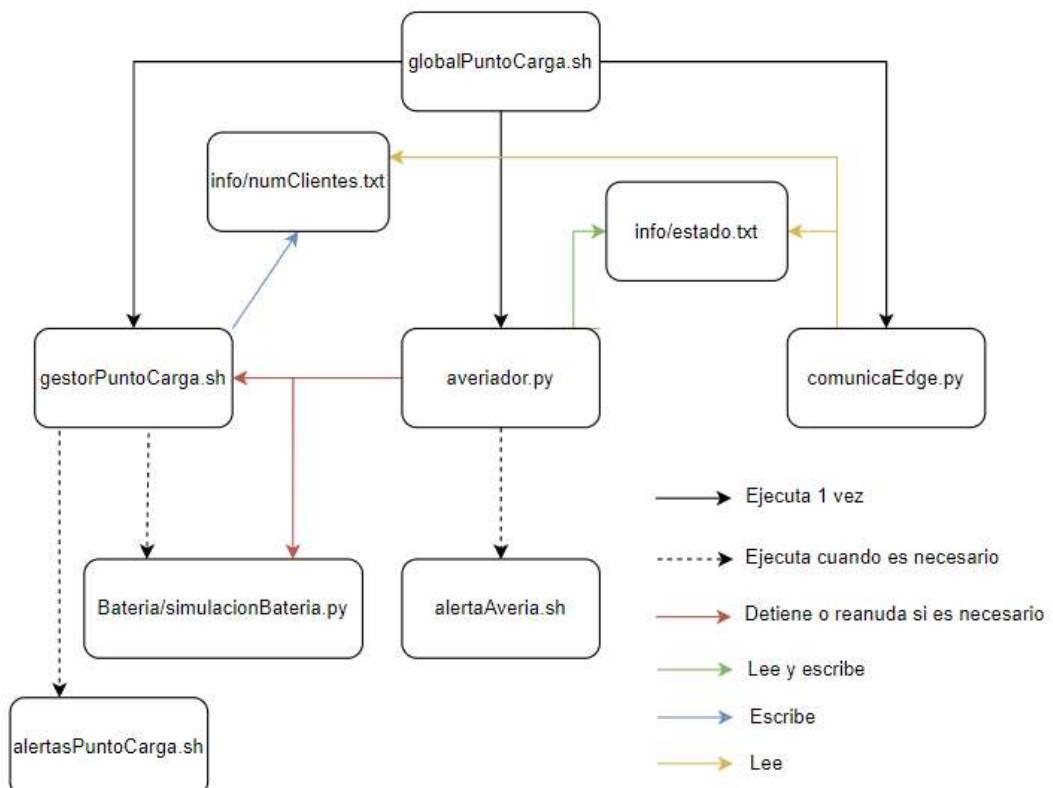
Todo lo necesario para que un punto de carga funcione se encuentra en la carpeta PuntoCarga, dentro de la carpeta PuntoCarga hay lo siguiente:

1. globalPuntoCarga.sh: inicia averiador.py, gestorPuntoCarga.sh y comunicaEdge.py
  - a. averiador.py: simula averías del punto de carga modificando estadoPuntoCarga.txt
    - i. alertaAveria.sh: muestra por pantalla un mensaje de avería
  - b. gestorPuntoCarga.sh: detecta coche conectado con el enchufe y gestiona:
    - i. Bateria/simulacionBateria.sh: muestra el progreso de la recarga
    - ii. alertasPuntoCarga.sh: muestra alertas en caso de exceder tiempo ocupación plaza
  - c. comunicaEdge.py: envía al edge toda la info necesaria sobre el punto
2. info/estado.txt: almacena el estado actual del punto de carga
3. info/numClientes.txt: almacena la cantidad de clientes que han utilizado el punto de recarga

Para automatizar esto y que todo funcione al encender la raspberry, colocamos la carpeta PuntoCarga en /home/pi y editamos el archivo crontab con crontab -e y añadimos:

```
@reboot sleep 60 && /home/pi/PuntoCarga/.globalPuntoCarga.sh
```

Resumen:



## Simulación averías punto de carga

Cada punto de carga debe poder averiarse, para ello creamos un script *averiador.py* que nos permita simular este suceso, para ello simplemente generamos un valor aleatorio y en caso de que este coincida con el valor de alguna de las averías, el estado del punto de carga pasará a ser el de la avería correspondiente.

Con tal de simular las posibles averías de un punto de carga hemos elegido las siguientes:

```
averias = {
    1:"enchufe",          #  Cable roto, no transmite
    2:"voltage",          #  El voltage suministrado no es el correcto (elevado o deficiente)
    3:"pantalla",         #  El display del punto de carga falla
    4:"circuito interno"  #  Circuito que gestiona la electricidad de todo el punto de carga
}
```

Cada avería está representada por un número, este número permite que gestionemos mejor el estado actual del punto de carga, ya que lo haremos haciendo uso de un archivo auxiliar *estadoPuntoCarga.txt* que únicamente debe almacenar el estado del punto de carga, de manera que podremos escribir y leer de este, de manera que una vez se repare el punto de carga, desde la web de gestión simplemente deberán modificar el contenido del archivo *estadoPuntoCarga.txt* a 0 (0 indica estado funcional).

En caso de que se genere una avería, se mostraría un mensaje por la pantalla del punto de carga a la espera de que este sea reparado, la alerta se muestra a partir de el script *alertaAveria.sh*:

```
$1= mensaje que indica el tipo de avería
$2=/home/pi/PuntoCarga

echo $1
rutaActual=$2

state=`cat $rutaActual/info/estado.txt`
while [ $state != "0" ]; do
    state=`cat $rutaActual/info/estado.txt`
done

exit 0
```

Por último, es necesario detener el script global del punto de carga *gestorPuntoCarga.sh* en caso de que exista una avería en este, dado que todo son procesos, podemos detener y reanudar el script en cualquier momento:

```
#  Se detienen todos los procesos de atención al cliente del punto de carga
PIDsimulacionBateria = subprocess.getoutput('ps aux | grep simulacionBateria | grep python3 | awk '+'{print $2}')
PIDgestorPuntoCarga = subprocess.getoutput('ps aux | grep PuntoCarga | grep /bin/bash | grep -v "sh -c" | awk '+'{print $2}')
if PIDsimulacionBateria:
    subprocess.getoutput('kill -STOP '+str(PIDsimulacionBateria))
if PIDgestorPuntoCarga:
    subprocess.getoutput('kill -STOP '+str(PIDgestorPuntoCarga))
```

PUNTO CARGA FUERA DE SERVICIO

```
#  Se reanudan todos los procesos del punto de carga
if PIDsimulacionBateria:
    subprocess.getoutput('kill -CONT '+str(PIDsimulacionBateria))
if PIDgestorPuntoCarga:
    subprocess.getoutput('kill -CONT '+str(PIDgestorPuntoCarga))
```

PUNTO CARGA FUNCIONA DE NUEVO

Con esto, en caso de que se genere una avería y el punto de carga estuviera recargando un vehículo, este proceso de recarga se detendrá automáticamente y el cliente debe recibir el servicio de recarga en otro punto o bien recibir reembolso.

## Cantidad clientes atendidos por un punto de carga

Para conocer la cantidad total de clientes atendidos por un punto de carga, dentro de la carpeta PuntoCarga, existe el archivo info/numClientes.txt en el cual se almacena esta información, este valor se va actualizando dentro del archivo cada vez que un cliente se conecta al punto de carga, tal y como podemos ver en *gestorPuntoCarga.sh*:

```
24 clienteNum=0
25 while true; do
26     # Obtenemos el estado de la interficie -> [up, down, unknown]
27     state=$(cat /sys/class/net/$interfaceName/operstate)
28     while [[ "$state" != "up" ]]; do
29         state=$(cat /sys/class/net/$interfaceName/operstate)
30     done
31
32     clienteNum=$((clienteNum + 1))
33     echo "Atendiendo a cliente nº $clienteNum"
34     echo $clienteNum > $rutaActual/info/numClientes.txt
```

## Comunicación hacia el edge estado punto de y número clientes atendidos

Dentro del sistema de cada punto de carga, tenemos el archivo *estadoPuntoCarga.txt* que almacena el estado actual del punto de carga, los valores posibles de este archivo son 0,1,2,3 o 4, 0 indica que no hay avería y el resto de números representan diferentes averías que puede tener el punto. Para comunicar el estado actual de un punto de carga al edge(y que se pueda detectar en la web de gestión), basta con leer el valor actual del archivo *estadoPuntoCarga.txt* y enviarlo de forma periódica. Para ello hemos creado un nuevo script *comunicacionEdge.py* que se ejecuta en el punto de carga y realiza esta tarea.

En cuanto a la cantidad de clientes atendidos, simplemente se lee el valor del archivo *info/numClientesAtendidos.txt* y se envía directamente, dado que aún no se ha especificado cómo tratar este dato, no lo enviaremos aunque ya tenemos la base de como hacerlo.

Todo esto viene gestionado por *comunicaEdge.py*, como podemos ver a continuación el formato en que se envían los datos, se indicarán la estación a la que pertenece el punto, el identificador del punto dentro de la estación y finalmente los datos, dado que solo enviamos el estado del punto y la comunicación se realiza mediante MQTT, especificamos que el topic sea “estación/numEstación/cargador/numCargador/estado”, quedando de la siguiente manera:

```
32 # Loop forever
33 while True:
34     estado = state_File.read()
35     #numClientes = numClientes_File.read()
36     state_File.seek(0, 0)
37     #numClientes_File.seek(0, 0)
38     MQTT_MSG = json.dumps({"estacion": "1", "cargador": "1", "estado": estado});
39     #MQTT_MSG = json.dumps({"estacion": "1", "cargador": "1", "estado": estado, "numClientes": numClientes});
40     client.publish(MQTT_TOPIC, MQTT_MSG)
41
```

## Comunicación hacia el edge el número de plazas libres y ocupadas (definición).

Antes de hacer la comunicación con el edge, primero debemos definir qué datos vamos a transmitir para evitar problemas. Lo más básico de todo sería indicar cuántas plazas hay ocupadas y cuantas desocupadas (como hay 32 plazas, sabiendo las ocupadas con una simple resta sabemos las libres) pero, para que quede más claro, la idea sería pasar el número de la plaza (1-32) con su estado (libre / ocupado). De esta manera no pasaremos solo un integer indicando la disponibilidad de la electrolinera, sino concretamos cuales son las plazas libres y ocupadas.

Una idea que se nos ha ocurrido para transmitir estos datos sería con un array “plaza” de 32 posiciones (plaza[0] → 1 / plaza[31] → 32) donde cada posición tenga un 1 o un 0 dependiendo si está ocupada o no respectivamente (creemos que hacerlo así puede ser más sencillo que poniendo algún string o char). Por lo tanto si en la posición plaza[18] = 0 diremos que la plaza 19 está libre y si plaza[23] = 1 diremos que la plaza 24 está ocupada. Si nos da demasiado TOC que las posiciones del array y de las plazas no sean las mismas pues siempre se puede cambiar la numeración de las plazas de 1-32 a 0-31 y ya estaría.

## TENER RASP CON TENSORFLOW + CAMERA

1- Crear un directorio donde almacenar lo que descargamos el Tensorflow lite desde github.

-Nos situamos en el directorio donde queremos que se encuentre el directorio contenedor de todos los ficheros.

-Ejecutar la siguiente comanda:

**git clone**

<https://github.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi.git>

-Para que todo sea mas facil y no escribir ese nombre tan largo del directorio, ejecutamos por ejemplo:

**mv TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi tflite**

-Ya tenemos la base de tensorflow que nos permitirá detectar objetos!.

2-Hacemos un **cd a tflite**(o el nombre que se haya escogido)

3-Creamos un entorno virtual de tal manera que evitemos cualquier error/complicaciones y sepamos que cosas instalamos

-Ejecutamos la siguiente comanda:

**pip3 install virtualenv**

-Ejecutamos una comanda como la siguiente:

**python3 -m venv entorn-tflite**

-Esto nos creara un directorio dentro de tflite que sera nuestro entorno virtual

4-Activamos el entorno virtual

-Con la comanda siguiente:

**source entorn-tflite/bin/activate**

¡OJO! ¡HABRÁ QUE HACERLO SIEMPRE QUE INICIEMOS UNA TERMINAL (PASO 2+PASO 4)!

5-Ejecutamos el script get\_pi\_requirements.sh que viene en el git clone (puede tardar unos minutos, CHILL)

Instalamos opencv

**pip install opencv-python**

6-Podemos crear un directorio donde guardaremos nuestro modelo para detectar matriculas.

**-mkdir model1**

-Guardaremos en dicho directorio:

**-detect.tflite**

**-labelmap.txt**

7-Ejecutamos el programa que nos permite detectar objetos usando dicho modelo.

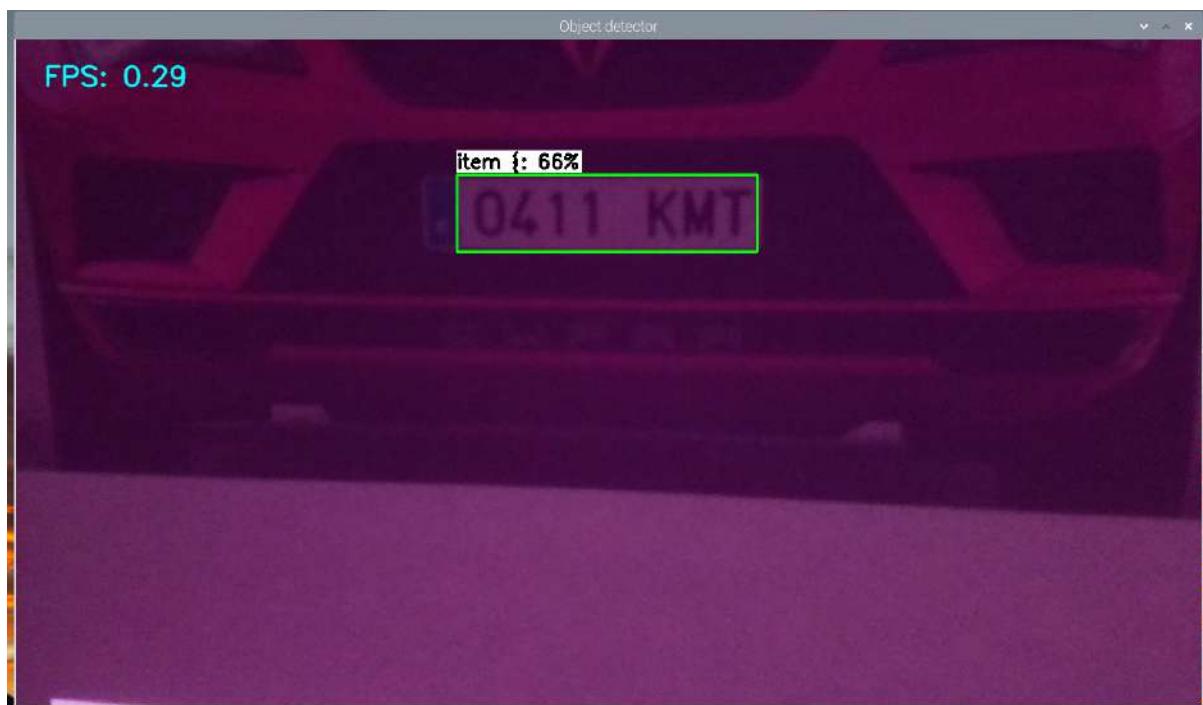
-Se hace mediante la comanda siguiente:

**python3 TFLite\_detection\_webcam.py --modeldir=model1**

-Automáticamente se nos abrirá una ventana con la webcam y veremos que va captando, posteriormente podemos pasar una matrícula de tal manera que la cámara tenga visión de ello y veremos con qué nivel de certeza nos dice si es una matrícula o no.



XD



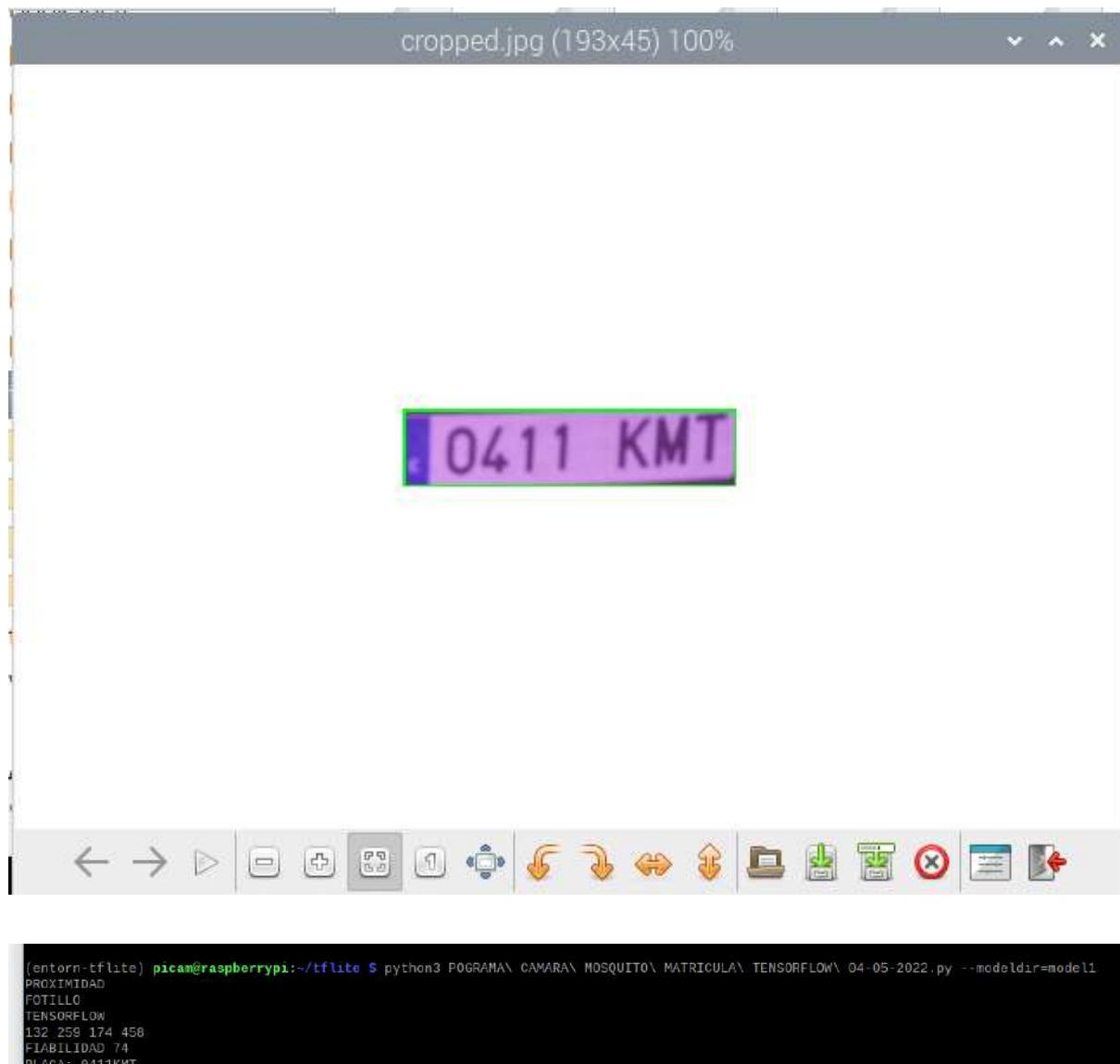
## EJECUTAR PROGRAMA SENSOR + CAMARA + DETECTOR MATRÍCULA + MQTT

Instalar todas las librerías (DEMASIADAS)

Preparar entorno con pasos anteriores y ejecutar programa

```
python3 POGRAMA\ CAMARA\ MOSQUITO\ MATRICULA\ TENSORFLOW\ 04-05-2022.py --modeldir=model1
```

Aqui vemos como el programa ha sacado la matrícula de la imagen

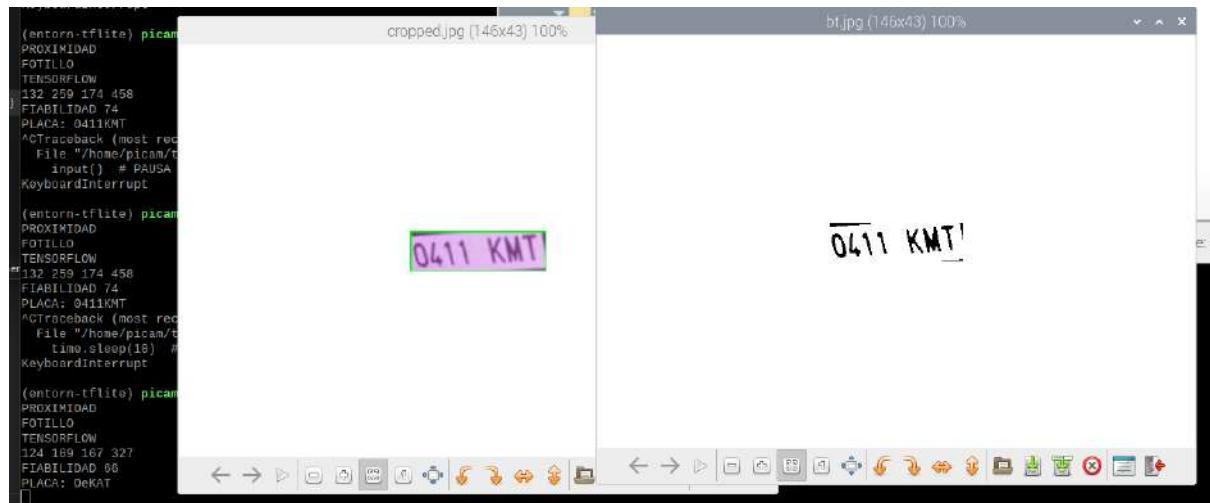


A veces no lee bien si está de lado o algo.

He decidido implementar que la cámara haga una foto en vez de que este continuamente grabando para ahorrar recursos



Aunque debería



No se porque, he probado de 1000 maneras :(

*En resumen, he hecho un script “global” el cual al detectar un true o false del sensor de proximidad (falta implementar) active la cámara, haga una foto y consiga la matrícula de la foto.*

## Script reconocimiento texto matricula.

El sistema que se implementará para el reconocimiento de clientes y darles paso, tras varias reuniones y discusiones, se ha decantado a realizarlo mediante la comprobación por matrículas.

Para ello, lo que haremos será intentar dividir el problema con la finalidad de llegar progresivamente a tener un código que sea capaz de reconocer las matrículas captadas por la cámara mismo, es decir, consiste en dos fases:

- 1) Realizar código que reconozca y extraiga matrícula de una foto.(script actual)
- 2) Realizar código que reconozca y extraiga foto de una cámara.(próximamente)

En un principio tras una ardua investigación implementaremos el reconocimiento de matrículas con python y diferentes herramientas.

Las librerías que usaremos serán las siguientes.

### **OpenCV.**

Librería amplia y de código abierto, incluye cientos de algoritmos, de los cuales lo que nos interesarán son algunos usados en casos como:

- Cambiar de RGB a escala de grises.
- Aplicar filtros con la finalidad de reducir distorsiones
- Quedarnos con los bordes.
- Encontrar contornos(figuras geométricas).

### **imutils**

- Herramienta que nos simplificará datos de cara a tratarlos.

### **numpy**

- ofrece herramientas que facilitan el tratado sobre los diferentes valores que se almacenan.

### **easyocr**

- herramienta para extraer textos, una alternativa sería tesseract, de momento mientras que no encontramos fallos, usamos este ya que es bastante sencillo.

### **matplotlib-pyplot**

- para mostrar imágenes.

Pasos para reconocer textos:

Leer foto.

De momento lo que hacemos en este sprint, es leer imagen a imagen y analizarlo, para ello usamos la herramienta cv2 y su función imread() al cual le pasamos el path y el fichero en cuestión.

```
img = cv2.imread('/home/ptinscr/fotosMatricula/c1.jpg')
```

Filtros:

1. Filtro gray.

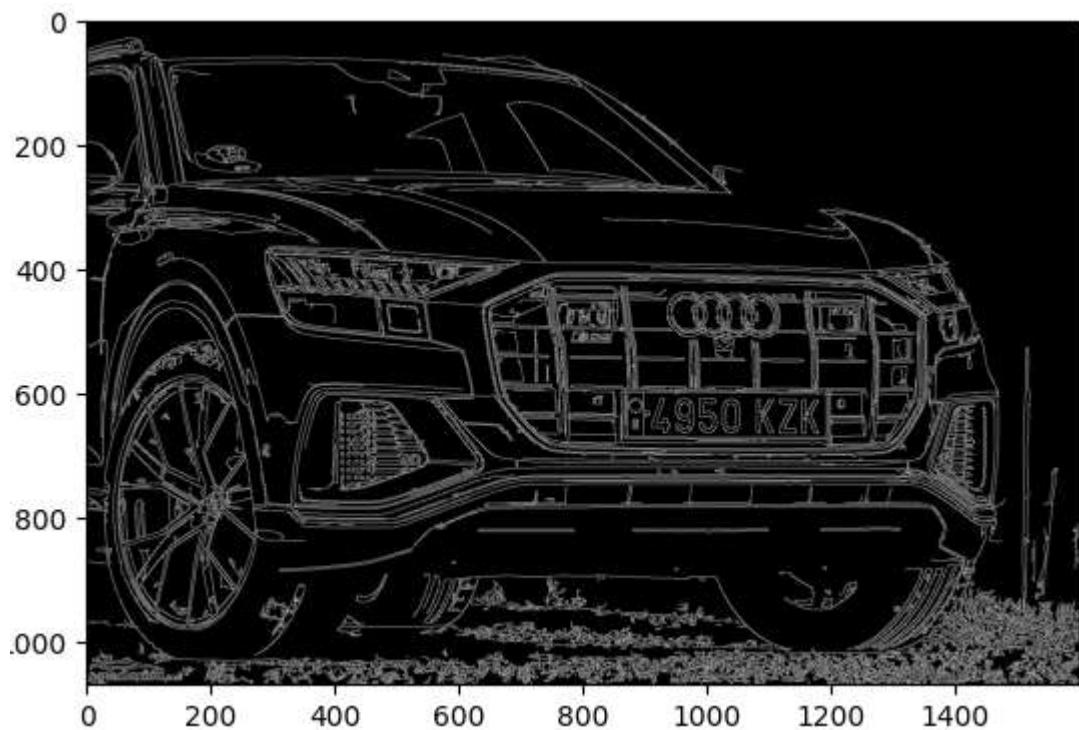
```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```



2. Filtro bordes + reducción de distorsión.

Sin reducción

```
edges = cv2.Canny(gray,30,200) #Se queda con bordes
```

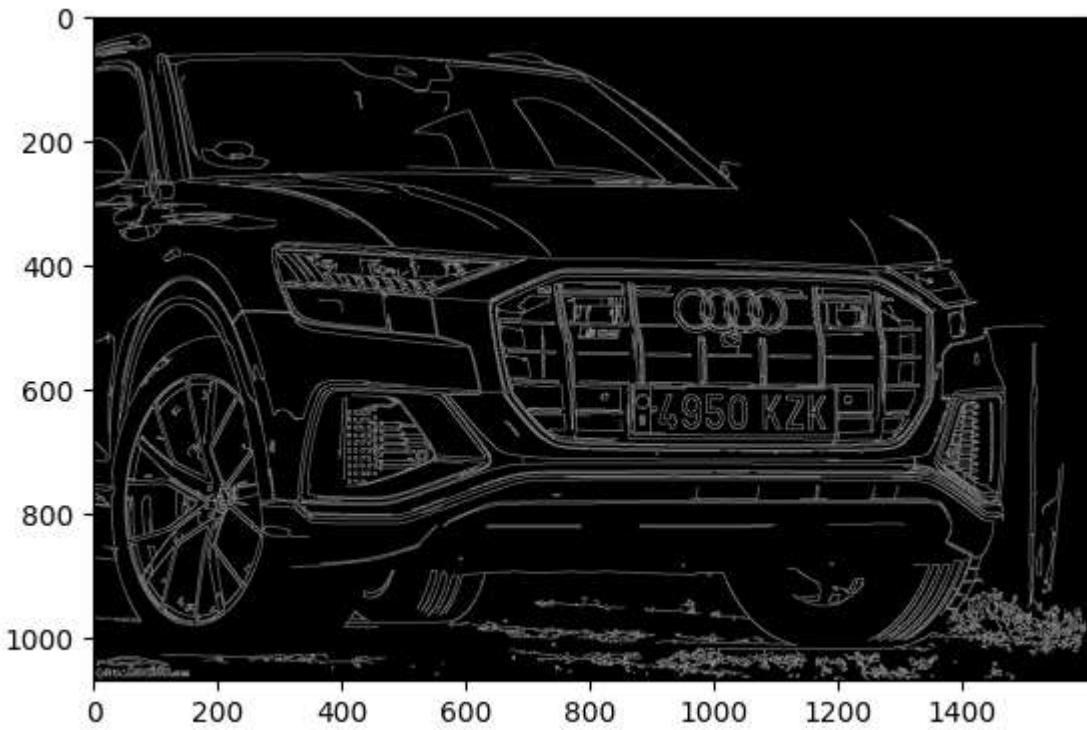


Con reducción

```

bilFilter = cv2.bilateralFilter(gray,11,17,17) #Reducción
de distorsiones(Noise)
edges = cv2.Canny(bilFilter,30,200) #Se queda con bordes

```



Detectar figuras rectangulares.

1. Dibujar contornos:

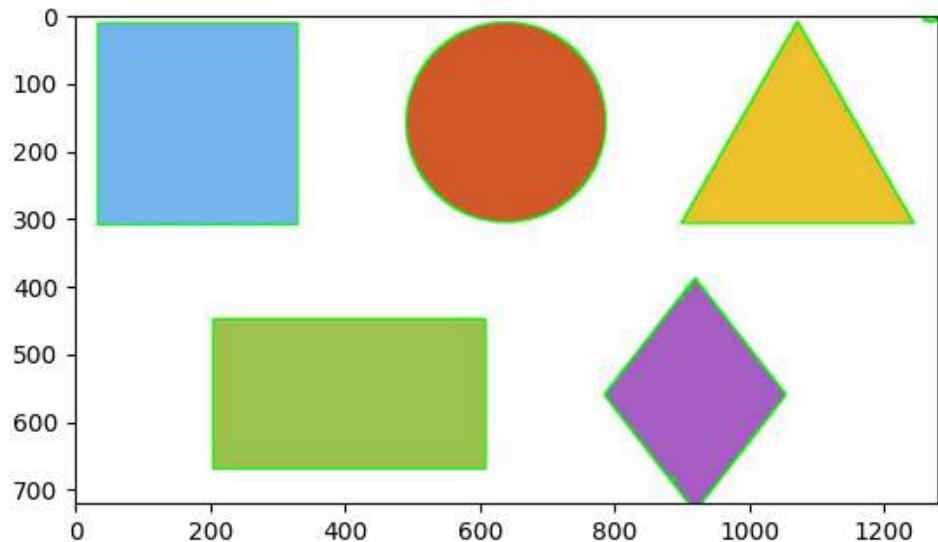
```

img = cv2.imread('/home/ptinscr/fotosMatricula/aaa.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray,30,200) #Se queda con bordes

contAux =
cv2.findContours(edges.copy(),cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
contours = imutils.grab_contours(contAux)
contours =
sorted(contours, key=cv2.contourArea, reverse=True)[:10]

for c in contours:
    cv2.drawContours(img,[c],0,(0,255,0),2)

```



2. Detectar rectangulo.

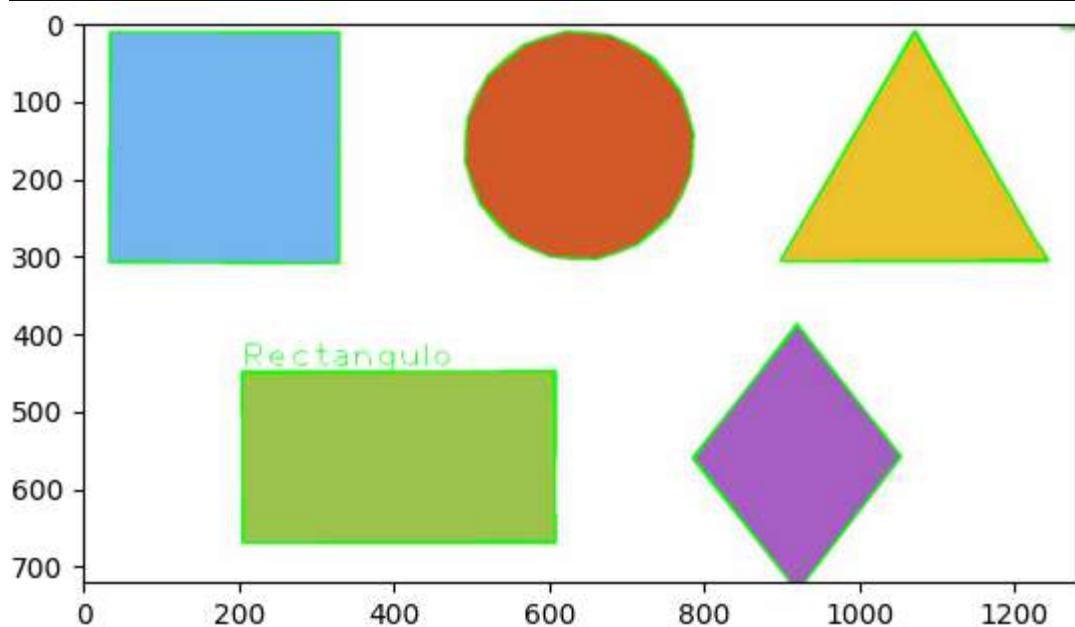
```

for c in contours:
    #epsilon = 0.1*cv2.arcLength(c,True)
    approx = cv2.approxPolyDP(c,10,True)
    print(len(approx))
    x,y,w,h = cv2.boundingRect(approx)

    if len(approx) == 4:
        aspect_ratio = float(w)/h
        if aspect_ratio > 1.2:

cv2.putText(img,'Rectangulo',(x,y-5),1,3,(0,255,0),1)

```



Aplicado a coche:



Por ende, definimos una parte del código que nos encuentra rectángulos, marca la ubicación y finalmente lo señalamos:

La idea que implementamos para poder reconocer las matrículas es:

1. Tenemos diferentes contornos, quizás varios rectángulos o alguna figura parecida.
2. Para detectar el rectángulo, de momento aplicamos una idea sencilla que posteriormente podría ser mejorada.
  - a. El número de lados.

En este sentido, queremos figuras de 4 lados.

Este es un valor bastante complicado de conseguir acotar en imágenes, difícil en el sentido de que dependiendo del contexto del problema y la foto algunos valores para la obtención de los contornos pueden ser arbitrarios y no coincidir en ningún aspecto incluso con otros.

- b. Relación de aspecto:



La relación de aspecto la calculamos básicamente dividiendo la altura por el ancho.

El cuadrado en este caso, como todos sus lados son iguales, tendría una relación de aspecto de 1, mientras que en el rectángulo tendrá una relación de aspecto mayor a ello.

Por ende, tomando el ejemplo, nos daría una relación de aspecto de  $21/9 = 2.3$  y las matrículas de los coches cabe esperar que sean rectángulos(de momento que ignoramos motos). Para suavizar un poco tema de ángulos y otros, este lo reducimos un poco y lo ponemos a 1.5 para así intentar evitar quedarnos con cuadrados que tengan desperfectos por los ángulos y quedarnos con rectángulo

3. Hay otro problema, puede ser que como dijimos anteriormente alguna figura se nos cuele, o que algún rectángulo no tenga nada que ver, por ende, lo que haremos será, quedarnos con una lista de los contornos que se encuentren, e ir recorriendo. Si el primer contorno que encontramos es el bueno, nos quedamos con ese, si no lo es, buscamos otro con la finalidad de encontrar el correcto, cabe obviamente la posibilidad de que no acabemos de encontrarlo.

En resumen, las ideas explicadas anteriormente se vería reflejado en código de la siguiente manera:

```
for c in contours:
    approx = cv2.approxPolyDP(c,10,True) #Aproximem polinomis
    x,y,w,h = cv2.boundingRect(approx) #Dades del polinomi
    if len(approx) ==4 :
        aspect_ratio = float(w)/h
        if aspect_ratio > 1.5:
            location = approx
    if location is None:
        continue
    else:
        #El tractem.
```

Para probar lo anterior, lo que hacemos es pues añadir dos líneas de código que nos imprime el contorno y le imprimimos “Rectángulo”

```
cv2.putText(img,'Rectangulo',(x,y-5),1,3,(0,255,0),2)
cv2.drawContours(img,[approx],0,(0,255,0),2)
```



Obviamente dependiendo de la calidad de imagen y el ángulo de la foto, la detección de la matrícula será más o menos sencilla. Como hemos visto en las anteriores capturas, en ambos casos funciona correctamente.

Por ende, pasamos al siguiente paso, quedarnos con esa parte, que es lo que nos interesa.

#### **Recorte Zona.**

Lo que haremos es recortar por zonas tenemos por ejemplo la siguiente imagen:



Esa imagen nos ayudará a probar algoritmos para quedarnos con contornos rectangulares ya que no presenta mucha complicación y el contorno más resaltante es efectivamente el de la matrícula misma.

Para recortar dicho contorno lo que haremos será definir básicamente lo que es una máscara.

Para ello básicamente lo que hacemos es lo siguiente:

1. Declaramos una máscara vacía con ceros y numpy que básicamente será como si fuese una imagen totalmente gris.
2. Señalamos el contorno que queremos en la máscara, en este caso, le pasaremos pues la 'location' la cual queremos trabajar.
3. Finalmente, con una función AND aplicamos la máscara sobre la imagen original.

El código entonces es el siguiente:

```
mask = np.zeros(gray.shape, np.uint8)
auxmasked = cv2.drawContours(mask, [location], 0, 255,
-1)
auxmasked = cv2.bitwise_and(img, img, mask=mask)
plt.imshow(cv2.cvtColor(auxmasked, cv2.COLOR_BGR2RGB))
```

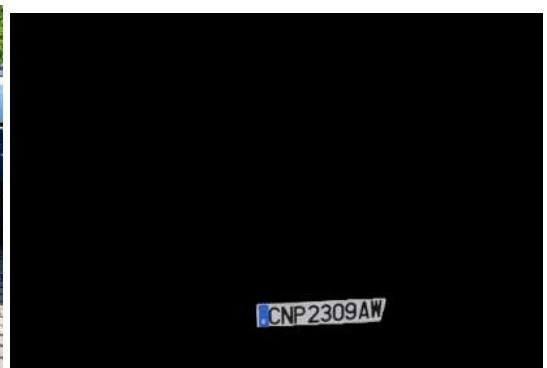
Resultado 1:



Resultado 2:



Resultado 3:



Resultado 4:



Resultado 5:





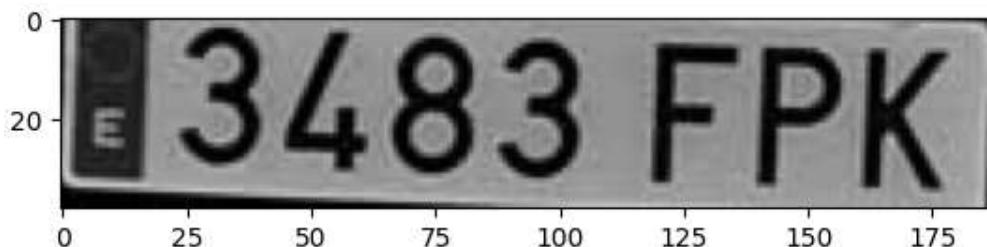
### Reconocimiento texto.

En el paso anterior podemos ver que hemos recortado ya lo que viene siendo el contorno de la matrícula pero todo el resto que antes pertenecía a una imagen ahora esta sin nada, lo que procedemos a hacer entonces es quedarnos únicamente con la zona que hemos recortado, para hacerlo básicamente haremos lo siguiente:

1. Quedarnos con las coordenadas x,y de la imagen que NO tiene color negro.
2. Nos quedamos con el mínimo y el máximo de x,y.
3. Recortamos finalmente dicha zona gracias a la combinación de dichas limitaciones

```
(x, y) = np.where(mask==255)
(x1, y1) = (np.min(x), np.min(y))
(x2, y2) = (np.max(x), np.max(y))
img_recortada = gray[x1:x2+1, y1:y2+1]
```

De esta manera, no tendremos un mini cuadrito y mucha imagen negra, tendremos entonces gracias a lo anterior algo como lo siguiente:

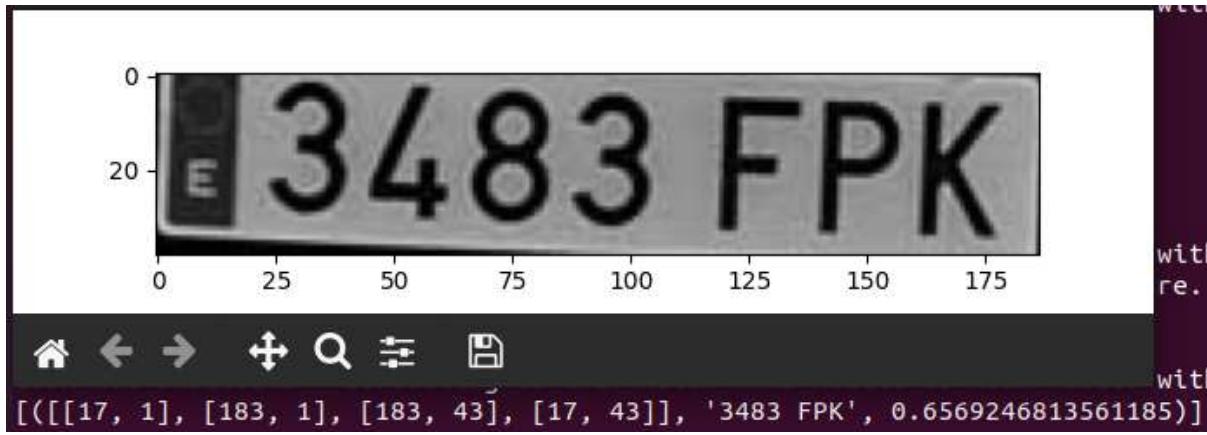


Para sacar el texto, utilizaremos la herramienta easyocr que nos permite leer texto de imágenes.

Lo usaremos de la siguiente manera:

```
reader = easyocr.Reader(['es'])
result = reader.readtext(recortado)
print(result)
```

El resultado que nos da es el siguiente:



```
>>[([[[17, 0], [185, 0], [185, 38], [17, 38]], '3483 FPK', 0.7062612349813593)]<<
```

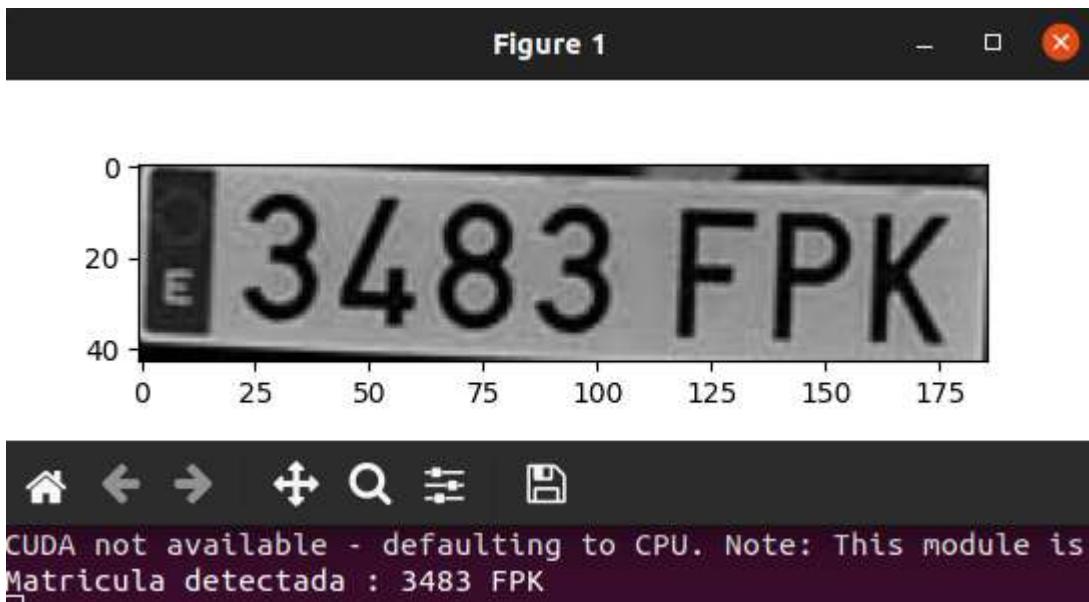
De este modo, aparte del contenido del texto, nos extrae también las coordenadas de donde lo ha sacado.

Lo que haremos entonces será de toda esta lista, quedarnos únicamente con el contenido de la matrícula, el resultado que nos ofrece se puede tratar como si fuese un array.

Para ello, el dato que nos interesa se encuentra en la posición 1 del array (comenzando desde 0 claro), en la posición anterior se encuentra todo junto las coordenadas como una única variable.

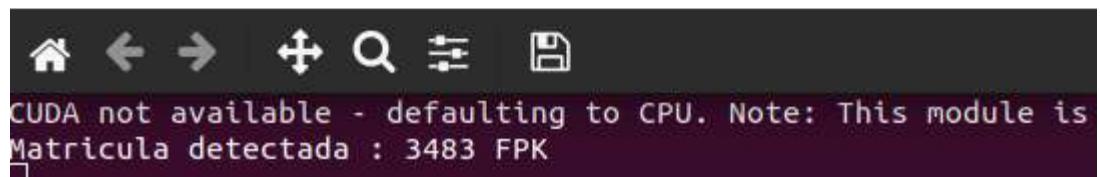
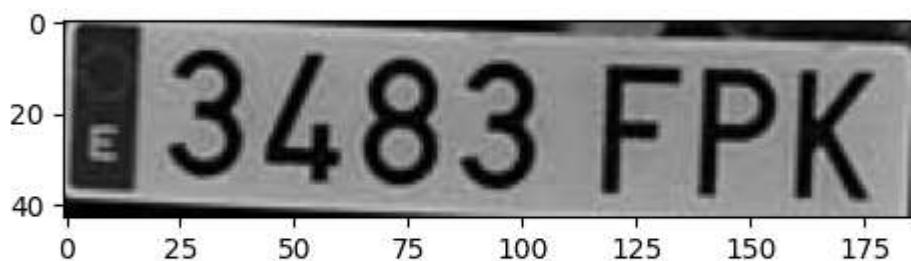
```
if len(result) > 0:  
    print("Matricula detectada : "+ result[-1][1])  
    break
```

De esta manera por la terminal veríamos algo tal que así:



Probemos con diferentes fotografías.

Resultado 1:



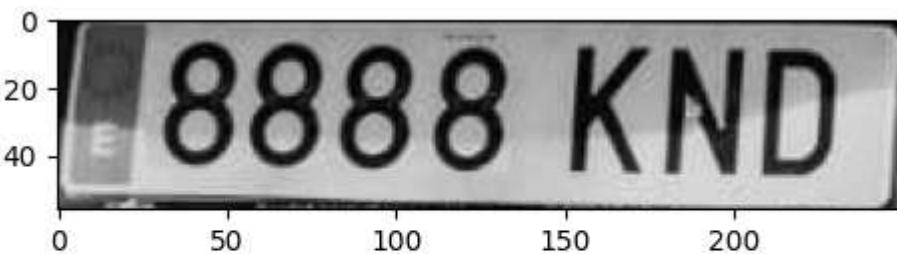
Resultado 2:



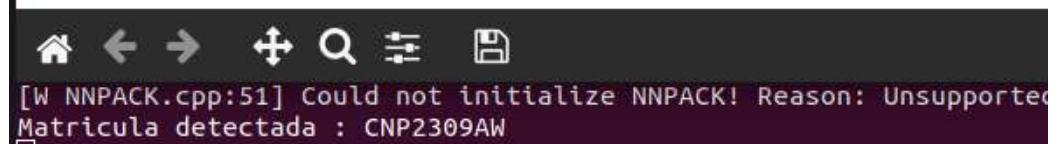
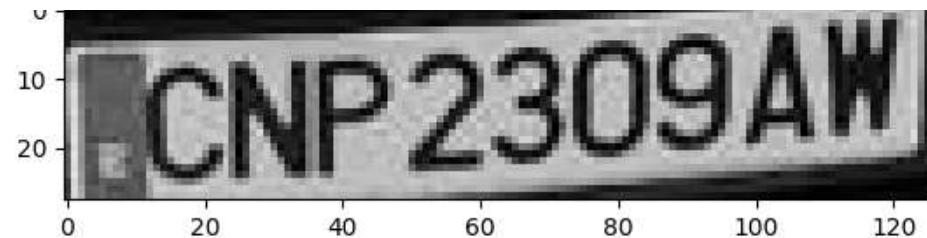
Resultado 3:



Resultado 4:



Resultado 5:



Comprobación de matrícula.

En este apartado básicamente lo que hacemos es simular la admisión o inadmisión de la matrícula reconocida..

Será bastante simple, crearemos una lista con algunas matrículas a aceptar, y los valores que obtengamos los compararemos con los de la lista.

La lista es la siguiente:

```
llistaAdmesos = ["2306XPE", "4950KZK", "CNP2309AW", "3245LCX"]
```

Primero de todo, para comparar lo que haremos sera quitar los espacios en blanco para evitar complicaciones, se hace de la manera siguiente:

```
matr_detectada = result[-1][1]
matr_detectada = matr_detectada.replace(" ", "")
```

De esta manera si tenemos “1234 ABC” nos quedaria algo como “1234ABC”.

Lo comparamos con la lista de admitidos:

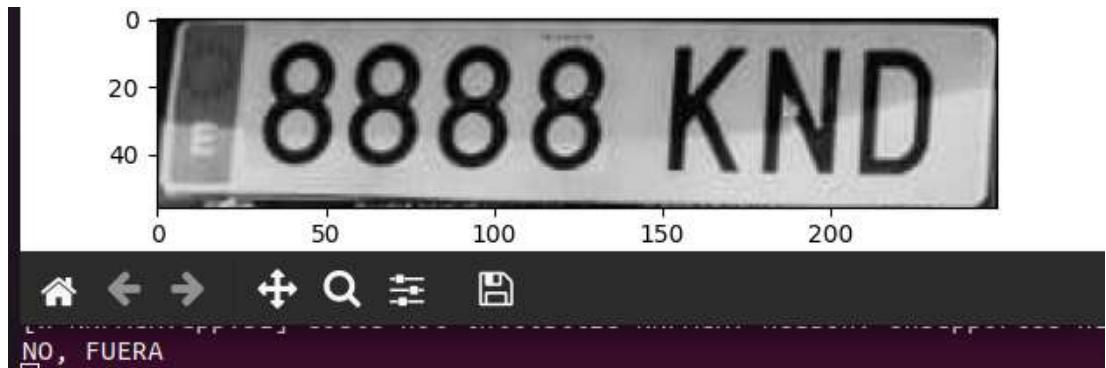
```
if matr_detectada in llistaAdmesos:
```

```

        print("Matricula aceptadaaaa : "+matr_detectada)
    else:
        print("NO, FUERA")

```

Example 1:



Para que sea todo más visual lo que haremos sera lo siguiente:

1. Si esta aceptada, lo pintamos de verde y ponemos el texto de la matricula cerca y de color verde tambien.
2. Si no lo esta, lo ponemos de color rojo.

El código entonces queda de la siguiente manera:

```

if matr_detectada in llistaAdmesos:
    aceptat = cv2.putText(img,
text=matr_detectada, org=(location[0][0][0],
location[1][0][1]+50), fontFace=font, fontScale=1,
color=(0,255,0), thickness=2, lineType=cv2.LINE_AA)
    plt.imshow(cv2.cvtColor(aceptat,
cv2.COLOR_BGR2RGB))
else:
    noaceptat = cv2.putText(img,
text=matr_detectada, org=(location[0][0][0],
location[1][0][1]+50), fontFace=font, fontScale=1,
color=(0,0,255), thickness=2, lineType=cv2.LINE_AA)
    cv2.drawContours(img, [location], -1,
(0,0,255), 5)
    plt.imshow(cv2.cvtColor(noaceptat,
cv2.COLOR_BGR2RGB))
break

```

Pruebas:

```
llistaAdmesos = ["2306XPE", "4950KZK", "CNP2309AW", "3245LCX"]
```







## Entrada (barrera+cam)

En el proceso para poder entrar a una electrolinera, se efectúa de la siguiente manera:

```
if __name__ == "__main__":
    # Siempre a la espera de algún vehículo
    while True:
        print('PROXIMIDAD')
        if proximidad():
            print('FOTILLO')
            fotillo()           # Detecta vehículo en frente de la cámara
            print('TENSORFLOW')
            tensorflow()       # A partir de la foto se busca la matrícula
            print('MOSQUITO')
            mosquito()          # Se envía la matrícula al broker para que se valide en el edge
            if validezEdge == "0":
                barrier()        # Si la matrícula es válida, se abre la barrera
            else:
                print("Matrícula no válida")
            time.sleep(5) # Sleep entre iteración y iteración de
```

1. Podemos ver, que el sistema de entrada dispone de un detector de proximidad que se activará cuando detecte a un vehículo a una cierta distancia.
2. Una vez detectado el vehículo, la cámara se activará y realizará una foto en la que se pueda ver la matrícula de este.
3. Tras obtener la foto, ésta será analizada mediante una IA con tensorflow con tal de obtener correctamente la matrícula de la foto con el vehículo que pretende entrar a la electrolinera.
4. La matrícula obtenida se envía al broker para que en el edge la validen, la respuesta del broker indicará si la matrícula detectada es válida o no, en caso de no ser válida no se abre la barrera y en caso de que se reciba que la matrícula es válida, se abrirá la barrera.
5. Tras finalizar con un vehículo, el sistema vuelve al primer paso.

Respecto a la comunicación con MQTT

```
def mosquito():
    # creamos suscriptor
    client = mqtt.Client(client_id='PiCam', clean_session=True)

    # asignamos callback para cuando el cliente establezca conexión con el broker
    client.on_connect = on_connect
    # asignamos callback para cuando el cliente reciba un mensaje del broker
    client.on_message = on_message

    # establecemos conexión
    client.connect(host="test.mosquitto.org", port=1883)
    # publicamos mensaje del cliente al broker
    publish(client)
    print("Waiting to receive message from broker...")
    client.loop_forever()
    print("Data received, connection with broker closed!")
    time.sleep(1)
```

Primeramente creamos el cliente y le asignamos un ID=Picam y dado que no queremos mantener ninguna información entre sesiones, declaramos clean\_session=True.

Una vez creado el cliente, asignaremos como será su comportamiento tanto al conectarse al broker como al recibir un mensaje de este.

Ahora ya podemos conectar al cliente con el broker, para ello indicaremos el host(@broker) y el puerto por el que atiende este.

Finalmente iniciaremos el proceso para publicar la información(matrícula) al broker y esperaremos una respuesta de este.

El `loop_forever()` lo detendremos en el momento en el que el cliente(Picam) reciba el mensaje del broker.

## Mejora en el sistema de la cámara

Resumiendo, la detección de matrícula se divide en dos pasos:

1. Recortar la matrícula mediante modelos
2. Extraer el texto de la zona recortada

El paso 1 ya se realiza de manera correcta y más del 90% de las veces recorta de manera correcta, por ende, el problema recae principalmente en la extracción de texto.

Las alternativas que se contemplan son básicamente mezclar diferentes tipos de filtros y diferentes métodos de lectura.

Alternativa 1:

- Aplicar directamente un filtro gris y leer el texto:

```
gray = cv2.cvtColor(cropped_image, cv2.COLOR_BGR2GRAY) # Gris
matricula = pytesseract.image_to_string(gray, config='--psm 7')
```

Alternativa 2:

- Sobre la alternativa1 aplicamos otro filtro, el de threshold.

```
gray = cv2.cvtColor(cropped_image, cv2.COLOR_BGR2GRAY) # Gris
_, bt = cv2.threshold(gray, 150, 255, cv2.THRESH_BINARY)
matricula = pytesseract.image_to_string(bt, config='--psm 7')
```

Las dos alternativas mencionadas es capaz de extraer texto de una imagen tranquilamente pero para que sea correcta la extracción, es necesario que la calidad de la imagen sea bastante buena, y por ende, en muchas ocasiones como la calidad de la imagen no es buena, acaba sacando muchos resultados erróneos, por ende, es necesario buscar algún otro método de extracción de texto.

Alternativa 3:

En lugar de simplemente intentar extraer todo el texto de una imagen, lo que se puede intentar hacer es básicamente que cada carácter de la imagen tenga su recorte, es decir, definir los contornos para cada carácter.

A parte de aplicar los filtros anteriores que facilitaban la lectura de una imagen, lo que haremos será aplicar algunos otros que nos facilitaran a la hora de detectar los contornos de los diferentes caracteres, como por ejemplo el `dilate()` que hace que la imagen tenga menos definición y asi sea mas facil diferenciar contornos importantes: lo vemos claro en la siguiente imagen que no es exactamente un coche:



De esta manera, podremos leer caracter a caracter y esto nos aporta ciertas ventajas.

1. Cada carácter será leído de forma independiente.
2. Cada carácter tendrá un contorno definido y esto nos permitirá diferenciar entre tamaños de contornos, sabiendo que en una imagen recortada de la matrícula, los caracteres que nos interesa ocupan bastante en la imagen, por ende, si hay letras pequeñas o alguna otro detalle pequeño que no nos interesa, pues podemos filtrarlos y eliminarlos. De esta manera, no tendremos letras o cosas que no nos interesan.

La implementación de lo anterior es la siguiente.

```
gray = cv2.cvtColor(cropped_image, cv2.COLOR_BGR2GRAY) # Gris
#cv2.imwrite(f'gray.jpg', gray)
ret, tres = cv2.threshold(gray, 0, 255, cv2.THRESH_OTSU |
cv2.THRESH_BINARY_INV)

rect_kern = cv2.getStructuringElement(cv2.MORPH_RECT, (5,5))
dilation = cv2.dilate(tres, rect_kern, iterations = 1)

try:
    contours, hierarchy = cv2.findContours(dilation, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
except:
    ret_img, contours, hierarchy = cv2.findContours(dilation,
cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
sorted_letter_countours = sorted(contours, key=lambda ctr:
cv2.boundingRect(ctr) [0])
matricula=""

for cntour in sorted_letter_countours:
    x,y,w,h = cv2.boundingRect(cntour)
    height, width=cropped_image.shape[:2]

    if height/float(h) > 7: continue

    ratio = h/float(w)

    if ratio<1.5: continue
```

```

if width/float(w) > 20: continue

area = h*w

if area<100: continue
rect = cv2.rectangle(cropped_image, (x,y), (x+w, y+h), (0,255,0),2)
roi = tres[y-5:y+h+5, x-5:x+w+5]
roi = cv2.bitwise_not(roi)

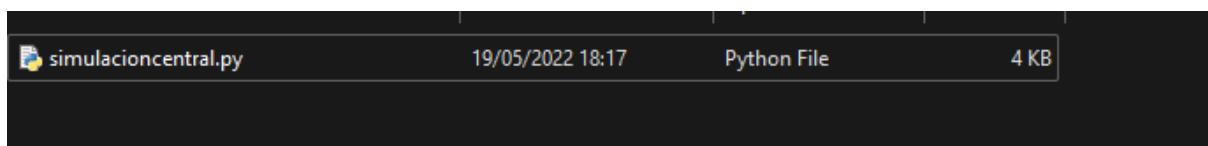
try:
    roi = cv2.medianBlur(roi, 5)
except:
    continue
text = pytesseract.image_to_string(roi, config='--tessedit_char_whitelist=0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ --psm 8
--oem 3')
matricula = matricula + text

```

Tras diferentes pruebas, la diferencia de éxito entre alternativas no es que sea exageradamente superior unas entre otras, pero en la mayoría de casos que tanto la alternativa 1 como la alternativa 2 no leía, en la alternativa 3 que es la compleja sí que obtenemos respuesta.

El método complejo en ocasiones dependiendo del ángulo y debido a las limitaciones de la raspberry, no acaba de dar buenos resultados, pero, debido a los resultados positivos, será con este el que nos quedamos.

## SIMULACIÓN 32 CARGADORES



Aquí se ve como se crean los threads para posteriormente con un random, asignar una tarea para hacer

5% carga  
5% averia

```
C:\Windows\py.exe
> Thread 17 iniciado!
> 16:06:14: Main: create and start thread 18.
> Thread 18 iniciado!
> 16:06:14: Main: create and start thread 19.
> Thread 19 iniciado!
> 16:06:14: Main: create and start thread 20.
> Thread 20 iniciado!
> 16:06:14: Main: create and start thread 21.
> Thread 21 iniciado!
> 16:06:14: Main: create and start thread 22.
> Thread 22 iniciado!
> 16:06:14: Main: create and start thread 23.
> Thread 23 iniciado!
> 16:06:14: Main: create and start thread 24.
> Thread 24 iniciado!
> 16:06:14: Main: create and start thread 25.
> Thread 25 iniciado!
> 16:06:14: Main: create and start thread 26.
> Thread 26 iniciado!
> 16:06:14: Main: create and start thread 27.
> Thread 27 iniciado!
> 16:06:14: Main: create and start thread 28.
> Thread 28 iniciado!
> 16:06:14: Main: create and start thread 29.
> Thread 29 iniciado!
> 16:06:14: Main: create and start thread 30.
> Thread 30 iniciado!
> 16:06:14: Main: create and start thread 31.
> Thread 31 iniciado!
```

Mediante comunicación del edge se mirara si un coche que llega en la plaza es la suya o no.

```
Thread 31 iniciado!
[+] Coche con matricula 5710UBM en cargador 23!
Coche no admitido en la plaza 23!
```

```

18/14/2021 16:15:31.000 main: create and start thread 31.
Thread 31 iniciado!
[+] Coche con matricula 5710UBM en cargador 23!
Coche no admitido en la plaza 23!
[+] Coche con matricula 7335ZWO en cargador 24!
Averia en el cargador 28 con averia enchufe
[+] Coche cargando en cargador 24!
{"cargador": 28, "averia": 1}
{"cargador": 24, "matricula": "7335ZWO"}
[+] Coche con matricula 6311PPA en cargador 11!
Coche no admitido en la plaza 11!
[!] Cargador 28 con averia enchufe
Averia en el cargador 20 con averia enchufe
{"cargador": 20, "averia": 1}
{"cargador": 24, "kwh": 7.4, "fecha": "22-05-2022 16:15:31", "matricula": "7335ZWO"}
{"cargador": 24 }
[+] Coche con matricula 3250TBG en cargador 23!
Coche no admitido en la plaza 23!
Averia en el cargador 8 con averia c.interno
 {"cargador": 8, "averia": 4}
|

```

Se ven los diferentes mensajes que se le enviará al edge.

```

[+] Coche cargando en cargador 24!
{"cargador": 24, "matricula": "7335ZWO"}
Averia en el cargador 29 con averia enchufe
{"cargador": 29, "averia": 11}

```

Se puede ver como después de x tiempo, se arregla la avería del cargador y vuelve a estar operativo.

Faltaria reservar un thread para poner el sistema nfc.

Nombre estaciones:



VGE-1 --> Vilanova i la Geltrú Electrolinera 1  
VGE-2 --> Vilanova i la Geltrú Electrolinera 2  
VGE-3 --> Vilanova i la Geltrú Electrolinera 3  
VGE-4 --> Vilanova i la Geltrú Electrolinera 4  
VGE-5 --> Vilanova i la Geltrú Electrolinera 5  
VGE-6 --> Vilanova i la Geltrú Electrolinera 6  
VGE-7 --> Vilanova i la Geltrú Electrolinera 7  
VGE-8 --> Vilanova i la Geltrú Electrolinera 8

## NFC

Datos que transmite el NFC del coche

Lo que va a transmitir el NFC del coche va a ser matricula y su potencia de carga en formato json (potencia en formato kW)

Ejemplo:

```
{"matricula": "848900W", "potencia": "450"}
```

Programa de lectura (puntocargaNFC.py):

Se mantendrá a la espera de recibir una tarjeta nfc para luego procesar sus datos y verificar si corresponde a esa plaza o no.

```
import RPi.GPIO as GPIO
from mfrc522 import SimpleMFRC522
import json
reader = SimpleMFRC522()

try:
    print('Listo para leer!')
    id, text = reader.read()
    print(id)
    print(text)
    textojson = json.loads(text)
    matricula = textojson["matricula"]
    potencia = textojson["potencia"]
    print(f'Matricula: {matricula}, Potencia: {potencia}')
except Exception as e:
```

```

        print(e)
finally:
    GPIO.cleanup()

```

## Programa de escritura (escribirNFC.py)

Lo utilizaran terceras personas en el momento de identificar al coche, el cual se le pondra la tarjeta nfc con sus datos para luego poderlos procesar en el punto de carga.

```

import RPi.GPIO as GPIO
from mfrc522 import SimpleMFRC522

reader = SimpleMFRC522()

try:
    matricula = input('Matricula: ')
    potencia = input('Potencia: ')
    print("Listo para escribir, acerca el nfc al sensor!")

    reader.write(f'{{"matricula":"{matricula}"}}')
    print(f"Escrito en nfc valores: [Matricula : {matricula} , Potencia : {potencia}]")
finally:
    GPIO.cleanup()

```

## Comunicación con el edge

En el módulo que contiene el nfc, es decir, el punto de carga deberá comunicar con el edge sobre si el coche que se ha instalado, pertenece a este espacio o no.

```

idPuntoCarga = '2'
topicSub = f"gesys/edge/puntoCarga/{idPuntoCarga}"
topicPub = f"gesys/edge/puntoCarga/vehiculo"

```

```
client.connect(host="test.mosquitto.org", port=1883)
```

## Post Comunicación con el edge

Una vez enviada la matrícula al edge, nos llegará un mensaje confirmando si pertenece o no a ese punto de carga.

En función del mensaje recibido, permitiremos que el coche pueda cargarse o enviaremos una alerta de que el coche está mal situado.

```
if int(validezEdge) != int(idPuntoCarga):
    print(f'Coche en cargador incorrecto [{idPuntoCarga}], por favor cargar
    en cargador número: [{validezEdge}]')
else:
    print('Coche en cargador correcto, cargando...')
```

Variable validezEdge, la devuelve el broker en formato json

```
validezEdge = msgPayloadJSON["idPuntoCarga"]

{"idPuntoCarga": 2}
```

# Desplegament d'Infraestructura i Arquitectura

## INFRAESTRUCTURA

### Paradigma edge-cloud

La infraestructura del proyecto se basa en el uso del paradigma edge-cloud. Se trata de tener un servidor cloud con gran potencia que realice los cálculos más costosos y contenga la gran parte de la información. De mientras, hay servidores edge desplegados en una capa inferior al del cloud, siendo cercanos a los usuarios, que permiten resolver de forma eficaz y rápida (sobre todo, baja latencia) las solicitudes de los clientes. Estas solicitudes no representan un gasto computacional exagerado y permiten ser realizadas con los datos locales. Aparte, estos servidores edge son los responsables de tratar los IoT, que son los sensores comentados anteriormente en este documento.

En dicho paradigma, el mayor problema es al escalar la infraestructura, realizar la comunicación entre los diferentes servidores edge y mantener la redundancia de todos los servidores.

En nuestro caso, que veremos más adelante a fondo, tendremos un servidor cloud que hará cálculos más costosos como ahora: estadísticas, creación de elementos que bajen al edge (ej. promociones), gestión de trabajadores y clientes, modificación de reservas, etc.

En cambio, en el servidor edge hará cosas más sencillas, como simplemente hacer peticiones de obtención de datos con alguna creación nueva de datos (ej. una reserva nueva) que subirán al cloud.

### Actores

En un primer momento, identificamos los actores plausibles a ser enviados o recibidos por los diferentes actores del escenario. Estos actores son:

Usuarios:

- **Consumidor:** Consumidor del vehículo. Será uno de los consumidores del proyecto. Este consumidor será el que haga solicitudes a la aplicación app. También recibirá datos sobre el estado de su vehículo, el estado de la electrolinera que está mirando (en caso de que esté mirando alguna) y de la zona en general. Estos datos serán puramente informativos para él. También generará datos al hacer solicitudes.
- **Trabajadores:** Serán trabajadores de las electrolineras. Otro cliente potencial de la aplicación. A este rol, le interesa sobre todo recibir datos para saber cómo tratarlos, por lo tanto, no serán datos informativos si no también con capacidad de ser tratados. Generará datos, al crear nuevas solicitudes (principalmente al cloud, al hacer uso de la aplicación web) pero también tendrá bastantes solicitudes de obtención de datos.

Elementos de las áreas:

Todos los datos serán generados por IoT o por el dispositivo que se está usando (Teléfono).

- **Vehículo:** Elemento de transporte del consumidor, puede ser cualquier tipo de vehículo eléctrico, obligatoriamente electrónico. Este vehículo generará datos que serán de gran utilidad para el proyecto.

- **Electrolineras:** Infraestructura física con cargadores electrónicos para vehículos que permiten cargar la batería a los vehículos eléctricos. Estas electrolineras recibirán datos y generan datos. Generarán datos que irán hacia los clientes. Los datos que reciba serán de los vehículos de la área y de los clientes de la área, pero también cabe la posibilidad de que provengan de otras áreas (recordar que los IoT son móviles).

Una vez identificados los usuarios, definimos que datos puede llegar a enviar cada uno de ellos.

## Datos a recibir-enviar

### **Consumidor**

#### **Envía:**

- Matrícula de sus vehículos: Matrícula de los coches de los cuales es propietario.
- Reservas de plaza: Reservar plaza en una electrolinera para su vehículo
- Pago: Pago realizado
- Peticiones: Como ahora averías en la estación o peticiones a los trabajadores

#### **Recibe:**

- Electrolineras: Electrolineras que están cerca y disponibles para cargar su vehículo con su pertinente información.
  - Estado
  - Ofertas
  - Tipo de red
  - Tipo de energía
- Batería de sus vehículos: Estado de la batería de sus coches
- Estado de sus vehículos: En caso de que algún vehículo necesite revisión, avisar
- Respuesta de reserva: Si se ha reservado, recibir la respuesta de la electrolinera a su solicitud (Aceptada/Rechazada).
- Plaza de reserva: La plaza reservada para su vehículo
- Precio por Kwh: Precio a kwh
- Precio a pagar: Precio a pagar por el servicio recibido y el tiempo de carga usado.
- Hora de reserva: El cliente elige cuánto tiempo estará cargando su vehículo. Como máximo, una hora.
- Fin de carga: Cuando esté cargando un vehículo, que le llegue un aviso de que está plenamente cargado

### **Trabajadores**

#### **Envía:**

- Ofertas: Si tiene alguna oferta de precio
- Peticiones: Resuelve averías o peticiones que vayan surgiendo
- Usuarios: Responsable de crear nuevos usuarios en el sistema

#### **Recibe:**

- Cantidad vehículos: Cuántos vehículos han sido cargados en sus electrolineras
- Saldo: Cantidad de dinero recibido por los servicios por dia

- Estado de las electrolineras: Debe ser algo diario, cada casi 15 minutos debería actualizarse más o menos. Interesa saber si la electrolinera está saturada.
  - Estado de electrolinera
- Cantidad de vehículos eléctricos en una área: Para así se sepa donde hay más clientes y por ende más necesidad de electrolineras.
- Potencia usada: Potencia total de sus electrolineras en un dia.

## Vehículo

### Envía:

- Capacidad de carga: Cuál es la capacidad que puede cargarse la batería del vehículo.
- Estado de la carga: Cuánta batería le queda al vehículo. Si hay una reserva, deberá indicar cuando llega a una batería óptima.
- Matrícula: Número de identificación del vehículo
  - Con la matrícula se podría saber que tipo de marca y modelo es.
- Propietario: Quien es el propietario del vehículo (Por ejemplo DNI o nombre para identificarlo (preferentemente lo primero)).

## Electrolineras

### Envía:

- Potencia de red: Potencia de los cargadores
- Estado: Si tiene sitio para reservar
- Confirmación de la reserva: Plaza reservada para el vehículo
- Información de los cargadores:
  - Tipo
  - Potencia media
  - Precio por minuto: El precio de su kwh
- Tiempo de reserva agotado: El tiempo de la reserva del vehículo se ha visto superado.
- Potencia usada: Cantidad de vatios que se ha usado en un día.
- Ubicación: La ubicación de la electrolinera.

### Recibe:

- Reservas: Cuando un consumidor haga una reserva, debe recibir la siguiente información:
  - Matricula: Matricula del coche que va a cargar
  - % de carga: El % de carga que solicita el cliente
  - Pagado: Debe ser un sí para tramitar la reserva
  - CantidadPago: Cantidad de dinero pagado
  - Cargador: Que cargador está ocupado.

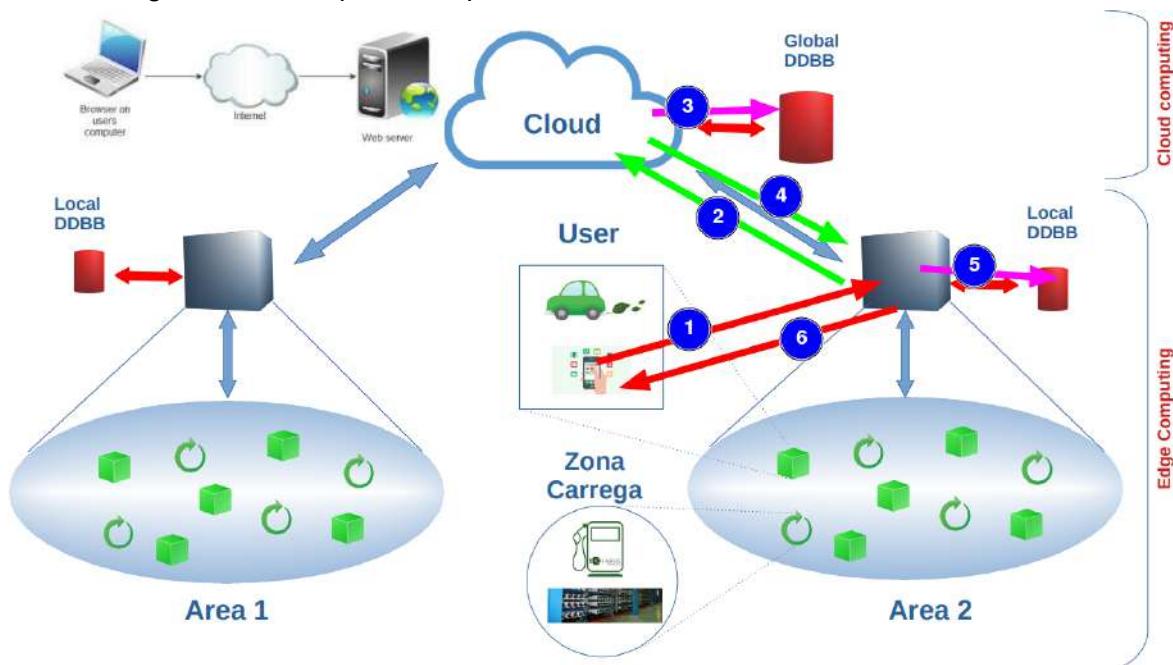
Como podemos observar, los datos a tratar son bastantes. Es por ello que necesitaremos un servicio que permita tratar y resolver las peticiones de los usuarios. Este servicio es la API. En la api haremos peticiones CRUD siendo estás un HTTP request desde la aplicación o app. Lo explicaremos más en detalle en su apartado correspondiente.

## Comunicación en la infraestructura

Como hemos visto antes, basaremos la infraestructura en el paradigma edge-cloud. Es por ello, que debemos definir previamente cómo se realizará la comunicación en la infraestructura.

Para ello:

1. Los usuarios consumidores (clientes de la aplicación app) realizan todas las peticiones directamente al edge. Si esta respuesta es adecuada, se saltaría al paso número 6.
2. En caso que la petición al edge no se pueda obtener adecuadamente, ya sea porque no hay dichos datos en la base de datos local o porque no se pueda resolver en el edge, se hará una solicitud al cloud.
3. El cloud gestionará la petición, mirará en su base de datos local, que es la general de toda la infraestructura y resolverá de forma interna la petición.
4. Una vez resuelta la petición, se la devuelve al edge.
5. Edge guardará los cambios en la base de datos local.
6. El edge envía la respuesta esperada al cliente.



En cambio, para los usuarios trabajadores, todas las peticiones que hagan mediante la aplicación web se harán directamente en el cloud, tal como está reflejado en el esquema anterior (Parte izquierda arriba). Por ende, cloud simplemente deberá actualizar su base de datos general y enviar la actualización a sus servidores edge en caso que corresponda hacerlo.

Para realizar la comunicación IoT-Edge-Cloud, haremos uso del protocolo MQTT.

## MQTT (message queue telemetry transport)

Protocolo para enviar información de tipo telemétrico

Protocolo orientado al dato, si necesitamos pasar un dato concreto de una temperatura, lo que mandamos es el valor de la temperatura.

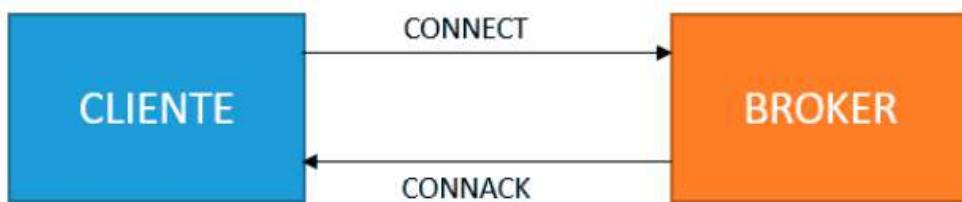
Usa la política publicación suscripción, un elemento publica los datos y otros elementos se suscriben a estas publicaciones. Hay que tener cuidado porque se podrían perder los datos, si alguien los publica pero nadie los suscribe el sistema los podría descartar o tener problemas de conectividad.

Aparte:

- A causa de que a veces los dispositivos IoT tienen limitaciones de potencia, consumo y ancho de banda se usa el protocolo MQTT(sencillo y ligero).
- Basado en la pila TCP/IP como base para la comunicación.
- Puertos empleados 1883 y 8883.

#### ENVÍO DE MENSAJES:

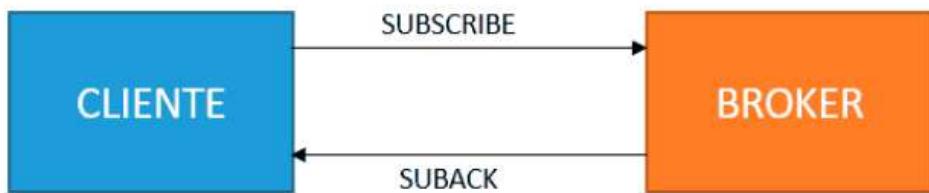
- Cliente envía mensaje CONNECT (información como nombre de usuario, contraseñas, id's...).
- La respuesta es con un mensaje CONNACK (contiene el resultado como conexión aceptada, rechazada...).



- Cuando el cliente quiere enviar un mensaje envía mensajes PUBLISH (topic + payload).



- Cuando se quiere suscribir y desuscribirse se usan mensajes SUBSCRIBE Y UNSUBSCRIBE, el servidor responde con los mensajes SUBCACK O UNSUBACK respectivamente.



- Para asegurar que la conexión está activa los clientes mandan periódicamente un mensaje PRINGEQ y el servidor le responde con el mensaje PINGRESP.
- El cliente para desconectarse envía mensaje DISCONNECT.

### Tipos de mensajes

| Control packet | Direction of flow                    | Description                                |
|----------------|--------------------------------------|--|
| CONNECT        | Client to Server                     | Client request to connect to Server        |
| CONNACK        | Server to Client                     | Connect acknowledgment                     |
| PUBLISH        | Client to Server or Server to Client | Publish message                            |
| PUBACK         | Client to Server or Server to Client | Publish acknowledgment                     |
| PUBREC         | Client to Server or Server to Client | Publish received (assured delivery part 1) |
| PUBREL         | Client to Server or Server to Client | Publish release (assured delivery part 2)  |
| PUBCOMP        | Client to Server or Server to Client | Publish complete (assured delivery part 3) |
| SUBSCRIBE      | Client to Server                     | Client subscribe request                   |
| SUBACK         | Server to Client                     | Subscribe acknowledgment                   |
| UNSUBSCRIBE    | Client to Server                     | Unsubscribe request                        |
| UNSUBACK       | Server to Client                     | Unsubscribe acknowledgment                 |
| PINGREQ        | Client to Server                     | PING request                               |
| PINGRESP       | Server to Client                     | PING response                              |
| DISCONNECT     | Client to Server                     | Client is disconnecting                    |

### QoS

El nivel de 'Quality of service' es un acuerdo entre el emisor del mensaje y el receptor que define la garantía de entrega de un mensaje.

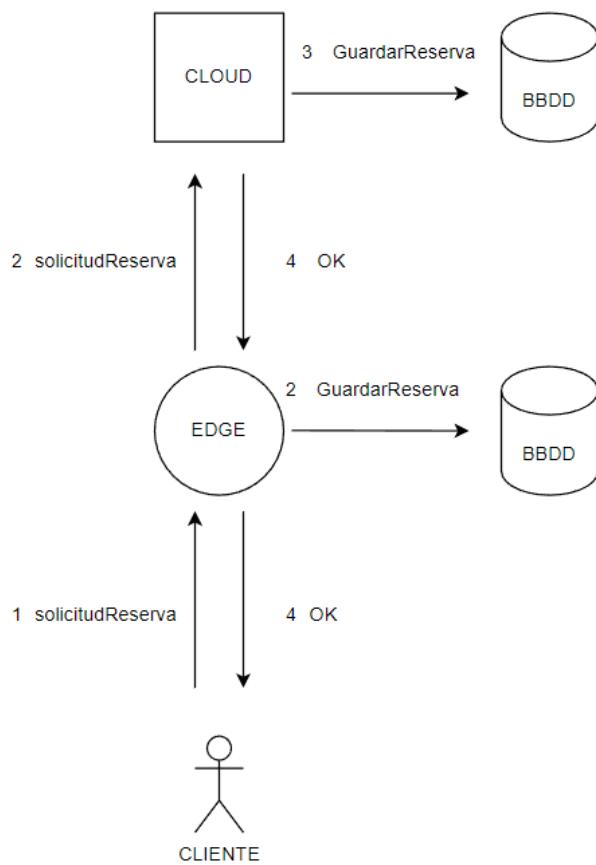
Existen tres tipos de QoS:

- Como mucho una vez (0)
- Al menos una vez (1)
- Exactamente una vez (2)

## Ejemplo de petición

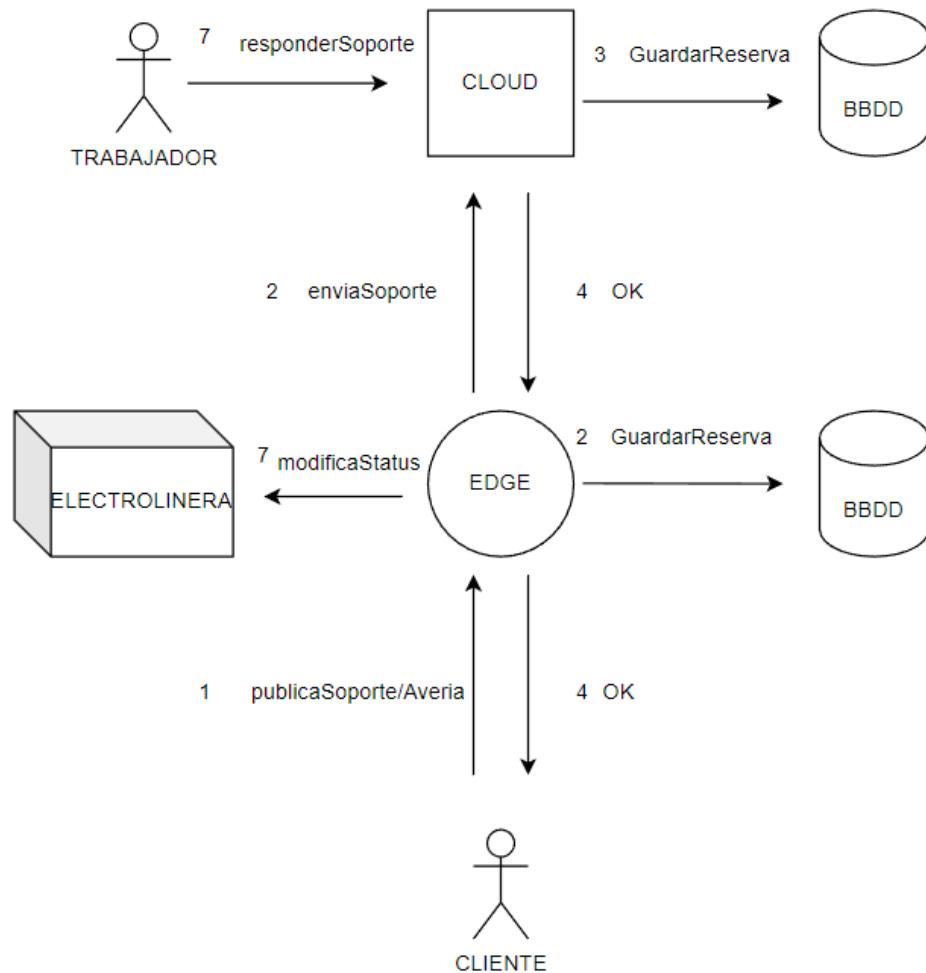
A continuación, veremos ejemplos de peticiones entre clientes-edge-cloud:

### ESQUEMA SOLICITUD RESERVA



Un ejemplo sería el de una solicitud de reserva dada por un cliente. Esta solicitud irá directa al edge, que el edge reenviará mediante MQTT al cloud. Ambos guardaron la reserva a su base de datos y respondería el cloud al edge un ok de haber recibido el mensaje y el edge al cliente de haber recibido la solicitud.

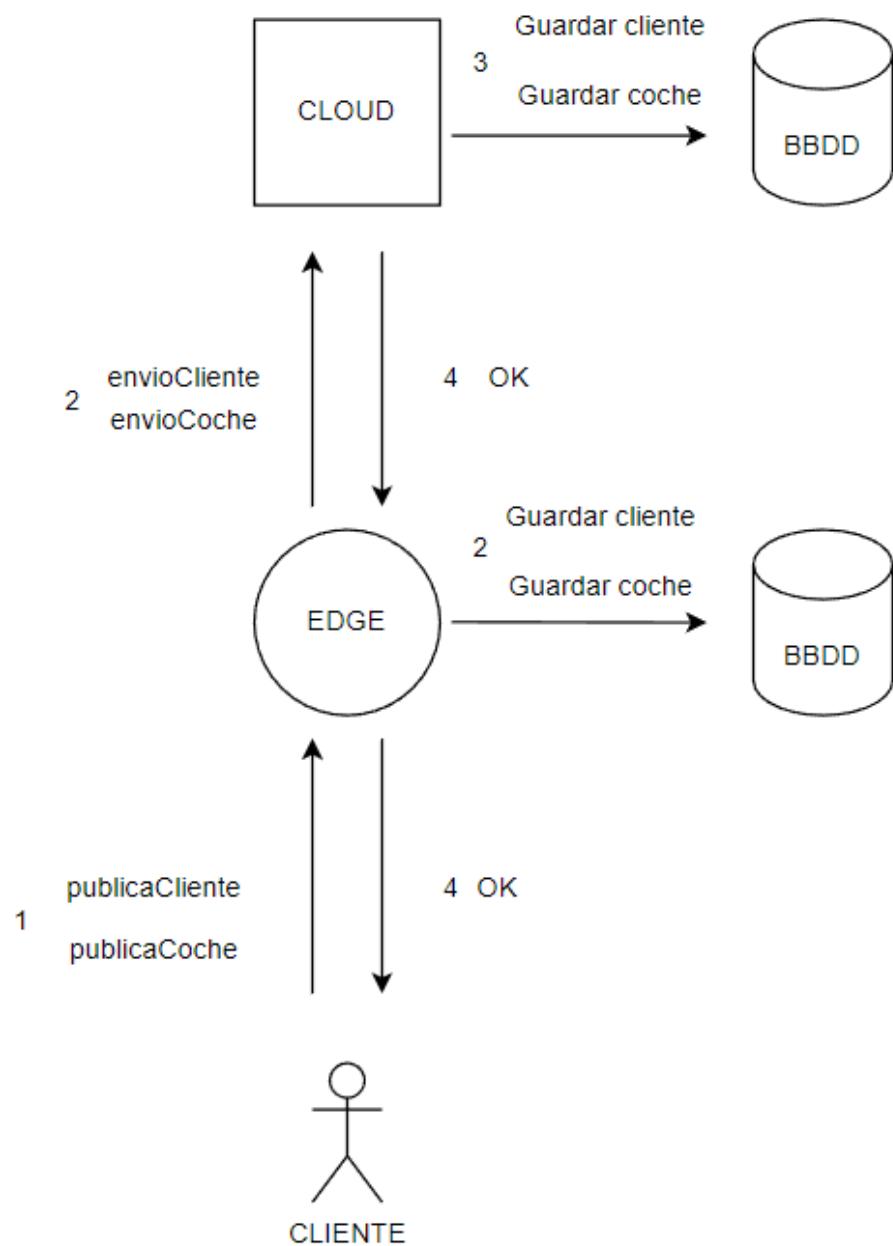
## ESQUEMA DE ESTADO DE SOPORTE DE CLIENTE-TRABAJADOR



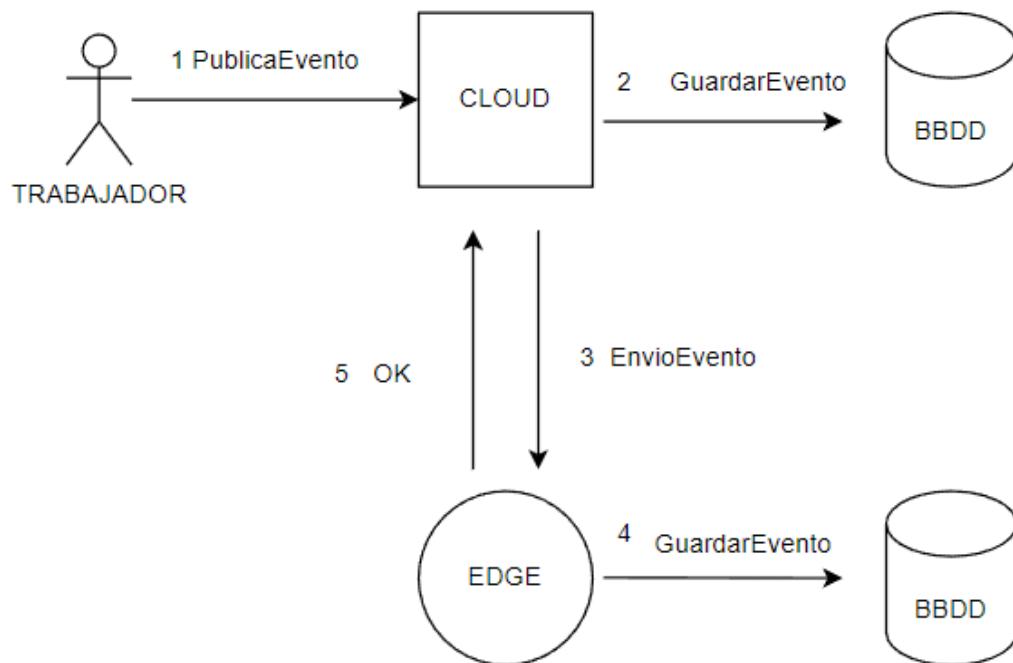
En este otro, vemos como hay más actores a modificar. Sigue haciendo lo mismo que en el anterior ejemplo, a diferencia de que en este caso, el trabajador puede responder dicha solicitud mediante un mensaje cuando quiera y además de eso, el estado de la electrolinera se verá modificado al tener una posible avería.

Esto lo veremos reflejado en código.

## CREACIÓN DE UN USUARIO



## PUBLICACIÓN DE UN NUEVO ELEMENTO EN EL CLOUD



En este caso, especificamos el caso de PublicaEvento como la posibilidad de publicar, modificar y eliminar:

- Reserva
- Promociones
- Averia
- Clientes
- Soporte
- Estaciones

Estos eventos, como afectan a los clientes serán los primeros en ser actualizados en la base de datos de los edge, al ser la que interactua con ellos.

Hay otros publicaciones, como:

- Trabajador

Que al ser algo más interno de la estructura de la empresa, no tiene porque verse enviada al edge con alta prioridad.

## Puertos

Una vez tenemos claro lo que se va a necesitar, tenemos que definir que puertos vamos a necesitar ya que los señores del Craax deben saberlo con antelación:

- Puerto 22 -> SSH: Para poder conectarnos remotamente a las máquinas.
- Puerto 80 -> HTTP: Para poder hacer solicitudes HTTP a la API.
- Puerto 443 -> HTTPS: Para poder hacer solicitudes HTTP seguras a la API.
- Puerto 42069 -> MQTT: Puerto por el que se enviarán los mensajes MQTT.

## MICROSERVICIOS

Los microservicios los definimos de la siguiente forma:

- Microservicios o servicios son programas que se ejecutan y se comunican entre sí. Estos servicios sirven para resolver solicitudes que lleguen tanto al cloud como al edge. Los microservicios o servicios son parecidos, simplemente se diferencian por la potencia que consumen. No consumirá lo mismo la web que un microservicio que monitoree la infraestructura (por poner un ejemplo). Por eso los englobamos en microservicios.

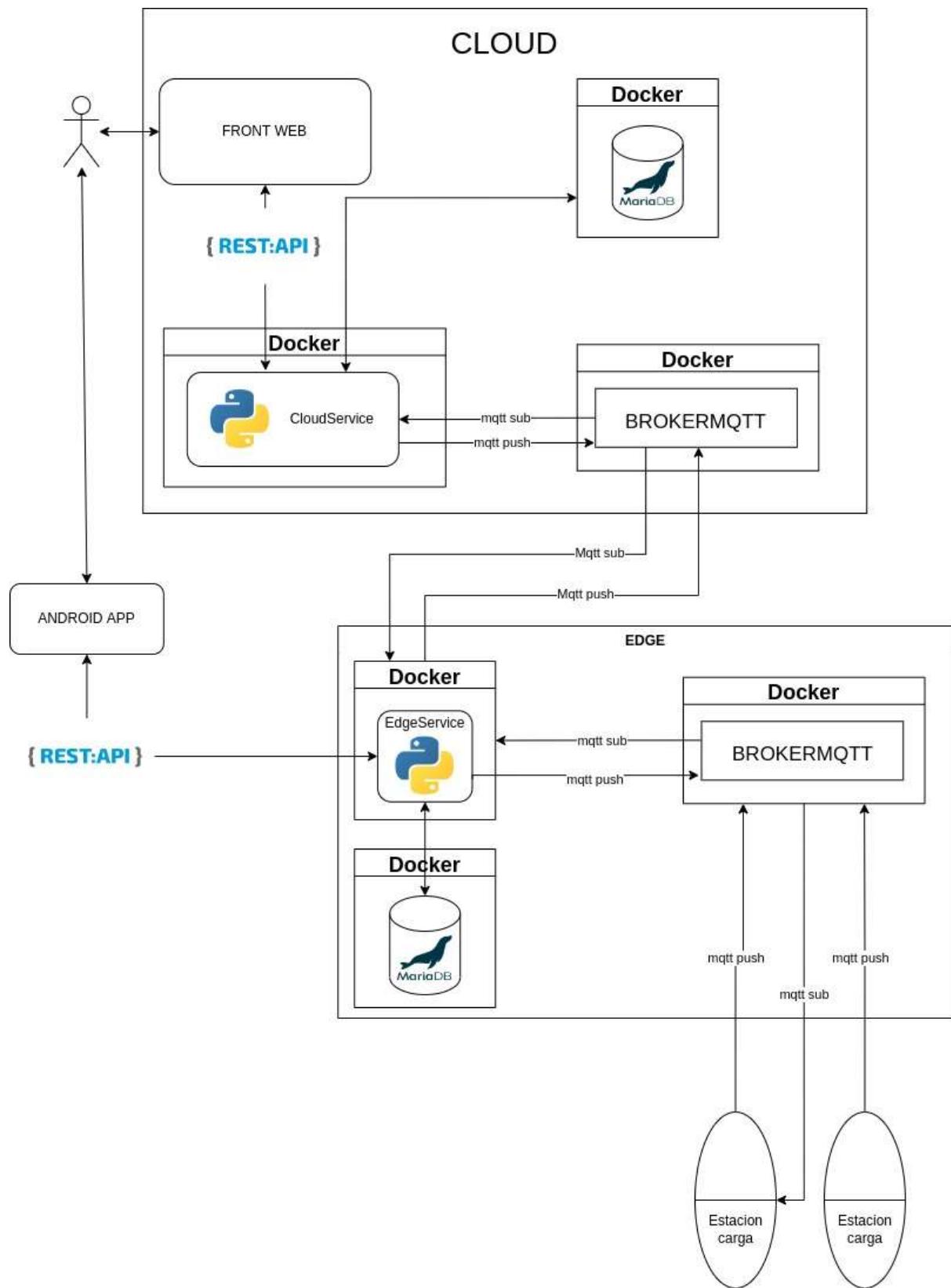
Un ejemplo de ello, es por ejemplo:

- Cloud Service: Es la API que estará implementada en cloud que sirve como el backend de la página web
- Edge Service: API en la capa de edge que se comunicará con la aplicación
- Broker MQTT: Intermediario del protocolo MQTT que sirve para gestionar la conexión publish-suscriber
- Las bases de datos de MySQL que estén desplegadas en los nodos.
- La aplicación web también es un servicio que se desplegará en la infraestructura.

Para desplegar dichos servicios, usaremos la tecnología docker. La tecnología docker nos permite lanzar procesos que actúan como "máquinas virtuales", permitiendo un uso eficiente de la capacidad computacional del ordenador.

Los dockers funcionan mediante códigos que generan imágenes y estas imágenes son luego utilizadas para ejecutar contenedores, que es como se llaman a los procesos dockers abiertos en el servidor. Para el despliegue de toda la infraestructura, usaremos dockers.

## Boceto general de la infraestructura



# Comunicación con MQTT

El tratamiento de mensajes se ha realizado de la siguiente manera:

- Hemos definido previamente los mensajes requeridos en cada sección.
- Utilizando el broker de mosquitto (broker OpenSource MQTT) → test.mosquitto.org para ir testeando
- En el fichero mqtt.py hemos ido haciendo diversas funciones cada una en función del tipo de datos que hay que tratar según lo que se recibe y se tiene que enviar.

A continuación adjuntamos los datos que tratamos en el edge respecto al IoT y el código relevante en python para acabar de visualizarlo:

Como podemos observar hay un apartado “topic” esencial que explicamos brevemente en qué consiste:

- Topic : Los broker en MQTT aplican un filtrado a los mensajes que son recibidos desde los publishers, para distinguir a que subscribers se entregan. Básicamente consisten en una cadena de texto los cuales pueden estar formados por uno o varios niveles separados por “/”.

## Vehículo

### Publica

|   |  |
|---|--|
| Notificar batería coche (battery 0-100) |  |
| topic                                   | gesys/edge/vehiculo                    |
| payload                                 | {"battery": 0, "matricula":"34543FGC"} |

### Se subscribe

|                                     |  |
|-------------------------------------|--|
| Enviar % carga que cargara el coche |  |
| topic                               | gesys/vehiculo/{matricula}/cargaMaxima |
| payload                             | {"battery": 40}                        |

```
def process_battery(bateria, id_matricula):
    print("-----")
    print("Tratando batería vehículo")
    m = Vehiculo.query.filter(Vehiculo.matricula ==
id_matricula).one_or_none()
    if m:
        ahora = datetime.today()
        for reserva in m.reservas:
            if reserva.id_vehiculo == id_matricula:
```

```

        if (reserva.fecha_entrada - timedelta(minutes=5)) < ahora <
reserva.fecha_salida: *
            m.procentaje_bat = bateria
            payload = {"battery": m.procentaje_bat}
            publish.single("gesys/vehiculo/{}".format(id_matricula),
payload=json.dumps(payload), qos=QOS, hostname=EDGE_BROKER, port=EDGE_PORT)
            print("Porcentaje bateria: {}={}%".format(m.matricula,
m.procentaje_bat))
            db.session.commit()
            return
        print("Reserva para este vehiculo no encontrada")
    else:
        print("Vehículo no encontrado")

```

\*Cabe remarcar que se hace una comprobación para asegurarnos de que la reserva es para ese momento con el vehículo pertinente.

### Punto carga

#### Publica

| Notificar averías del cargador |   |
|--------------------------------|---|
| topic                          | gesys/edge/puntoCarga/averia  |
| payload                        | {"idPuntoCarga": 2, "averia": 0}  |
| tipos averia                   | 0: ok<br>1: enchufe<br>2: voltaje<br>3: pantalla<br>4: circuito interno |

```

def process_averias(id_carga, num_averia):
    print("-----")
    print("Tratando averias")
    c = Cargador.query.filter(Cargador.id_cargador == id_carga).one_or_none()
    if c:
        c.estado = AVERIAS[num_averia]
        db.session.commit()
        print(c.estado)

    else:
        print("Cargador no encontrado")

```

#### Publica

|   |  |
|---|--|
| Notificar el consumo al final de la carga (MARCA LIBRE) |  |
| topic   | gesys/edge/puntoCarga/consumo                            |
| payload   | {"idPuntoCarga": 2, "kwh": 432, "matricula": "34543FGC"} |

```

def process_carga_final(id_carga, kwh, id_matricula):
    print("-----")
    print("Tratando consumo final del vehículo")
    date = datetime.now().replace(microsecond=0, second=0, minute=0)
    cargador = Cargador.query.filter(Cargador.id_cargador ==
id_carga).one_or_none()
    if not cargador:
        print("Cargador not found")
        return

    v = Vehiculo.query.filter(Vehiculo.matricula ==
id_matricula).one_or_none()
    if not v:
        print("Vehiculo no encontrado")
        return

    c = Consumo.query.filter(Consumo.id_cargador == id_carga, Consumo.id_horas ==
date).one_or_none()
    if not c:
        e = Estacion.query.filter(Estacion.id_estacion ==
cargador.estacion_id).one_or_none()
        c = Consumo(id_carga, date, 0, e.potencia_contratada) *
db.session.add(c)
        print("Consumo no encontrado.... Creandolo...")
        db.session.commit()

        potencia_anterior = c.potencia_consumida
        c.potencia_consumida = c.potencia_consumida+kwh
        c.estado = "libre"
        db.session.commit()
        print("Registrando consumo del cargador: {} -- Potencia consumida
anterior: {} -- Potencia consumida={}".format(c.id_cargador,
potencia_anterior, c.potencia_consumida))
    
```

\*En el caso de que el consumo que buscamos no exista se generar un consumo nuevo para la estación de ese cargador y lo ponemos a 0.

## Publica

|  |   |
|--|---|
| Comprobar que el vehículo está en el cargador adecuado (MARCA OCUPADO) |   |
| topic  | gesys/edge/puntoCarga/vehiculo              |
| payload  | {"idPuntoCarga": 2, "matricula":"34543FGC"} |

## Se suscribe

|  |  |
|--|--|
| Confirmar al punto de carga que se tiene que cargar el coche |  |
| topic  | gesys/edge/puntoCarga/{idPuntoCarga}         |
| payload  | {"idPuntoCarga": 2, "cargaLimiteCoche": 60)} |

```
def process_punto_carga(id_carga, id_matricula):
    print("-----")
    print("Comprobando que el vehículo está en el cargador adecuado")
    cargador = Cargador.query.filter(Cargador.id_cargador ==
id_carga).one_or_none()
    if not cargador:
        print("Cargador not found")
        return

    v = Vehiculo.query.filter(Vehiculo.matricula ==
id_matricula).one_or_none()
    if not v:
        print("Vehiculo no encontrado, en el cargador {}, llamando a la
grua...".format(id_carga))
        return

    # Tenemos el vehículo con la matrícula
    ahora = datetime.today()
    for reserva in cargador.reservas:
        if reserva.id_vehiculo == id_matricula:
            if (reserva.fecha_entrada - timedelta(minutes=5)) < ahora <
reserva.fecha_salida:
                cargador.estado = "ocupado"
                payload = {"idPuntoCarga": cargador.id_cargador,
"cargaLimiteCoche": reserva.procetnaje_carga}
                publish.single("gesys/edge/puntoCarga/{}".format(id_carga),
payload=json.dumps(payload), qos=QOS, hostname=EDGE_BROKER, port=EDGE_PORT)
                send_to_cloud("gesys/cloud/puntoCarga", {"ocupado": True,
"cargador_id": cargador.id_cargador})
```

```
print("El cargador {}, no tiene ninguna reserva, pero el coche {} esta ocupando la plaza. Llamando a la grua...".format(id_carga, id_matricula))
```

Notificar carga del coche al cargador que esta cargando el coche, cada vez que el coche envíe la carga se envía esto. (PANTALLA MUESTRA CARGA) (battery 0-100)

|         |                            |
|---------|----------------------------|
| topic   | gesys/vehiculo/{matricula} |
| payload | {"battery": 0}             |

### Camara

#### Publica

|         |  |
|---------|--|
| topic   | gesys/edge/camara                              |
| payload | {"matricula": "34543FGC", "id_estacio": "VG1"} |

### Barrera

#### Se suscribe

|         |                             |
|---------|-----------------------------|
| topic   | gesys/estaciones/VG1/camara |
| payload | 1 -> ABRIR<br>0 -> cerrar   |

```
def process_camera_event(matricula, id_estacio):
    print("-----")
    print("Tratando proceso de camara")
    i = Estacion.query.filter(Estacion.nombre_est == id_estacio).one_or_none()
    if i:
        ahora = datetime.today()
        for cargador in i.cargadores:
            for reserva in cargador.reservas:
                if reserva.id_vehiculo == matricula:
                    if (reserva.fecha_entrada - timedelta(minutes=5)) < ahora < reserva.fecha_salida:
                        print("Hay una reserva valida, abriendo barrera...")

    publish.single("gesys/estaciones/{}/camara".format(id_estacio),
                  payload="1", qos=QOS,
                  hostname=EDGE_BROKER, port=EDGE_PORT)
                  print("SEND: ABRIR")
                  return

    else:
        print("Estacion no encontrada...")
```

```

print("Reserva no encontrada mandando no abrir barrera...")
publish.single("gesys/estaciones/{}/camara".format(id_estacio),
               payload="0", qos=QOS, hostname=EDGE_BROKER, port=EDGE_PORT)
print("SEND: CERRAR")

```

Por la parte que respecta EDGE TO CLOUD, los datos a tratar son los siguientes:

### Reservas

| Registrar reservas en el cloud |   |
|--------------------------------|---|
| topic                          | gesys/cloud/reservas  |
| payload                        | {         "fecha_entrada": "2022-06-20T11:00:00",         "id_cargador": 12,         "procetnaje_carga": 30,         "precio_carga_completa": 30.0,         "estado": true,         "precio_carga_actual": 5.0,         "id_cliente": 1,         "id_reserva": 10,         "asistida": true,         "fecha_salida": "2022-06-20T18:00:00",         "id_vehiculo": "4950KZK",         "tarifa": 12.7,         "estado_pago": true     } |

| Remove reserva |                                  |
|----------------|----------------------------------|
| topic          | gesys/cloud/reservas/remove      |
| payload        | {         "id_reserva": 10     } |

| Registrar reservas en el edge |   |
|-------------------------------|---|
| topic                         | gesys/edge/reservas   |
| payload                       | {         "fecha_entrada": "2022-06-20T11:00:00",         "id_cargador": 12,         "procetnaje_carga": 30,         "precio_carga_completa": 30.0,         "estado": true,     } |

|  |  |
|--|--|
|  | <pre> "precio_carga_actual": 5.0, "id_cliente": 1, "id_reserva": 10, "asistida": true, "fecha_salida": "2022-06-20T18:00:00", "id_vehiculo": "4950KZK", "tarifa": 12.7, "estado_pago": true } </pre> |
|--|--|

| Editar reservas en el edge |  |
|----------------------------|--|
| topic                      | gesys/edge/reservas/edit   |
| payload                    | <pre>{   "fecha_entrada": "2022-06-20T11:00:00",   "id_cargador": 12,   "procetnaje_carga": 30,   "precio_carga_completa": 30.0,   "estado": true,   "precio_carga_actual": 5.0,   "id_cliente": 1,   "id_reserva": 10,   "asistida": true,   "fecha_salida": "2022-06-20T18:00:00",   "id_vehiculo": "4950KZK",   "tarifa": 12.7,   "estado_pago": true }</pre> |

| Remove reserva en el edge |                            |
|---------------------------|----------------------------|
| topic                     | gesys/edge/reservas/remove |
| payload                   | {"id_reserva": 10}         |

## Clients

| Nuevo cliente añadido en la app, subirlo al cloud. edge-to-cloud |   |
|--|---|
| topic  | gesys/cloud/clientes  |
| payload  | <pre>{   "nombre": "Pere",   "apellido": "Roca",   "dni": "123281A",   "telefono": 62554433,   "email": "pere.roca@boss.com",</pre> |

|  |   |
|--|---|
|  | <pre>"username": "perealoca", "password": "1", "foto": "a" "saldo": 2 }</pre> |
|--|---|

|                              |                                   |
|------------------------------|-----------------------------------|
| Remove cliente edge-to-cloud |                                   |
| topic                        | gesys/cloud/clientes/remove       |
| payload                      | <pre>{   "dni": "123281A" }</pre> |

|                |  |
|----------------|--|
| Editar cliente |  |
| topic          | gesys/cloud/clientes/edit  |
| payload        | <pre>{   "nombre": "Pere",   "apellido": "Roca",   "dni": "123281A",   "telefono": 62554433,   "email": "pere.roca@boss.com",   "username": "perealoca",   "password": "1",   "foto": "a"   "saldo": 2 }</pre> |

### Cliente vehículo

|                                   |   |
|-----------------------------------|---|
| push de vehiculos de los clientes |   |
| topic                             | gesys/cloud/clientes/vehiculo   |
| payload                           | <pre>{   "matricula": "2322",   "procentaje_bat": 15,   "cliente": "123319N",   "modelo": "500e Cabrio electrico" }</pre> |

|                 |                                      |
|-----------------|--------------------------------------|
| Remove vehiculo |                                      |
| topic           | gesys/cloud/clientes/vehiculo/remove |
| payload         | {}                                   |

|  |                           |
|--|---------------------------|
|  | "matricula": "2322",<br>} |
|--|---------------------------|

|                                  |   |
|----------------------------------|---|
| actualizar batteria desde la app |   |
| topic                            | gesys/cloud/clientes/vehiculo/edit                      |
| payload                          | {<br>"matricula": "2322",<br>"procentaje_bat": 15,<br>} |

## Mensajes soporte

| Crear ticket edge-cloud |  |
|-------------------------|--|
| topic                   | gesys/cloud/soporte/add  |
| payload                 | {           "id_ticket": 2,           "id_cliente": 1,           "estado": "Pendiente",           "asunto": "App no me va",           "fecha": "2022-06-27T19:55:31.465387",           "mensaje": "Problema con la aplicacion movil"         } |

| Responde ticket edge-cloud |   |
|----------------------------|---|
| topic                      | gesys/cloud/soporte/response  |
| payload                    | {           "id_usuario": 1,           "contenido": "no se que me esta contando",           "id_mensaje": 3,           "date": "2022-06-27T19:57:48.507218",           "id_ticket": 2         } |

| Remove ticket edge-cloud |                                      |
|--------------------------|--------------------------------------|
| topic                    | gesys/cloud/soporte/remove           |
| payload                  | {           "ticket_id": 2         } |

| Remove msg edge-cloud |                                    |
|-----------------------|------------------------------------|
| topic                 | gesys/cloud/soporte/message/remove |
| payload               | {"msg_id": 2}                      |

| Responde ticket cloud-edge |   |
|----------------------------|---|
| topic                      | gesys/edge/soporte/response   |
| payload                    | {           "date": "2022-06-29T04:10:57.321908",           "id_ticket": 9,         } |

|  |  |
|--|--|
|  | <pre>"id_mensaje": 13, "contenido": "no se que me esta contando", "id_usuari": 1 }</pre> |
|--|--|

|                          |                                   |
|--------------------------|-----------------------------------|
| Remove ticket cloud-edge |                                   |
| topic                    | gesys/edge/soporte/remove         |
| payload                  | <pre>{     "ticket_id": 2 }</pre> |

|                       |                                   |
|-----------------------|-----------------------------------|
| Remove msg cloud-edge |                                   |
| topic                 | gesys/edge/soporte/message/remove |
| payload               | {"msg_id": 2}                     |

|                   |  |
|-------------------|--|
| Set Status ticket |  |
| topic             | gesys/edge/soporte/status                                    |
| payload           | <pre>{     "estado": "Resuelto",     "id_ticket": 9, }</pre> |

## Trabajadores

|                                    |  |
|------------------------------------|--|
| Enviar trabajadores nuevos al edge |  |
| topic                              | gesys/edge/trabajador  |
| payload                            | <pre>{     "apellido": "234",     "id_trabajador": 207,     "id_usuari": 207,     "foto":     "https://this-person-does-not-exist.com/img/avatar-754c5f551521071 73073b232e864e6b.jpg",     "id_estacion": 1,     "telefono": "234", }</pre> |

|  |  |
|--|--|
|  | <pre> "email": "234", "nombre": "234", "question": "234", "type": "trabajador", "dni": "22234", "ultimo_acceso": "1970-01-01T01:00:00", "cargo": "234", "estado": "true", "username": "23422", "password": "b'\xb3{v\xca\xff&amp;\xd3\"b\xxa8\xe0\xc8\x8f&amp;\xe8d\xc1\xabY\xda\xd5M\xe0\x9aqL\x81P\xef\xb2;}" } </pre> |
|--|--|

### Punto de carga

|                               |  |
|-------------------------------|--|
| Notificar plaza libre/ocupada |  |
| topic                         | gesys/cloud/puntoCarga/estado                            |
| payload                       | <pre>{     "ocupado": True,     "cargador_id": 2 }</pre> |

|                                      |   |
|--------------------------------------|---|
| Notificar plaza consumo de una plaza |   |
| topic                                | gesys/cloud/puntoCarga/consumo                        |
| payload                              | <pre>{     "idPuntoCarga": 2,     "kwh": 432, }</pre> |

## Averias

| Notificar averias del cargador |   |
|--------------------------------|---|
| topic                          | gesys/cloud/puntoCarga/averia   |
| payload                        | {"idPuntoCarga": 2, "averia": 0}  |
| tipos<br>averia                | 0: ok<br>1: enchufe<br>2: voltaje<br>3: pantalla<br>4: circuito interno |

# API

El backend del proyecto consta de diferentes partes y tecnologías. En la 1a parte de este apartado, explicaremos las tecnologías usadas, su link de referencia y como funciona. Luego, una vez claras las tecnologías implementadas, separaremos la documentación y el código base para poder explicar con más claridad.

## Tecnologias:

Las tecnologías que hemos usado son las siguientes:

- **Swagger:** Es una herramienta para crear API REST en el que está basada en el lenguaje de marcas YAML en ficheros con la extensión .yml que permite crear una página interactiva, en la cual, puedes especificar diferentes operaciones GET, POST, PUT y DELETE. Esta página, además de eso, permite documentar, poner de ejemplo muestras de respuestas de las diferentes operaciones y sobre todo, lo más importante, te permite seleccionar el escenario en el que quieras trabajar, te genera un comando para el comando curl necesario para poder hacer solicitud y te permite gestionar el formato de data que quieras enviar.  
<https://swagger.io/>
- **Github pages:** Con github pages, hemos tenido la posibilidad de desplegar un portal web con un archivo .html que nos ha dado la opción de crear una página en línea sin ningún coste. Hemos usado el github pages para desplegar nuestro .yml del swagger a los equipos A3 y A4, que son los principales consumidores de la API  
<https://pages.github.com/>
- **SQLalchemy:** Hemos usado el ORM de SQLAlchemy. Como cualquier otro ORM, permite modular objetos relacionales en python, haciendo que su integración en un sistema python-sql sea mucho más sencillo y más interactivo.  
<https://www.sqlalchemy.org/>
- **Marshmallow:** Es también un ORM, pero más basado en convertir datos complicados de tratar o grandes tipos de datos en datos que sean de fácil lectura en python. Hemos usado marshmallow junto con SQLAlchemy para poder modular los objetos que devuelven las querys.  
<https://marshmallow.readthedocs.io/en/stable/index.html>
- **Flask:** Flask es uno de los frameworks disponibles en Python. Es reconocido por su gran capacidad de adaptarse al entorno y por dar mucha libertad a los programadores. Hemos usado flask para poder realizar la parte interactiva al backend, siendo este, el responsable de tratar las peticiones enviadas a la URI y devolver una respuesta.  
<https://flask.palletsprojects.com/en/2.1.x/>
- **Dockers:** Hemos usado la tecnología de los dockers para poder desplegar los diferentes servicios, ya sea servidor cloud y/o edge, como ahora, la API misma junto

con la base de datos, para su uso se requiere permisos de administrador del sistema o root. <https://www.docker.com/>

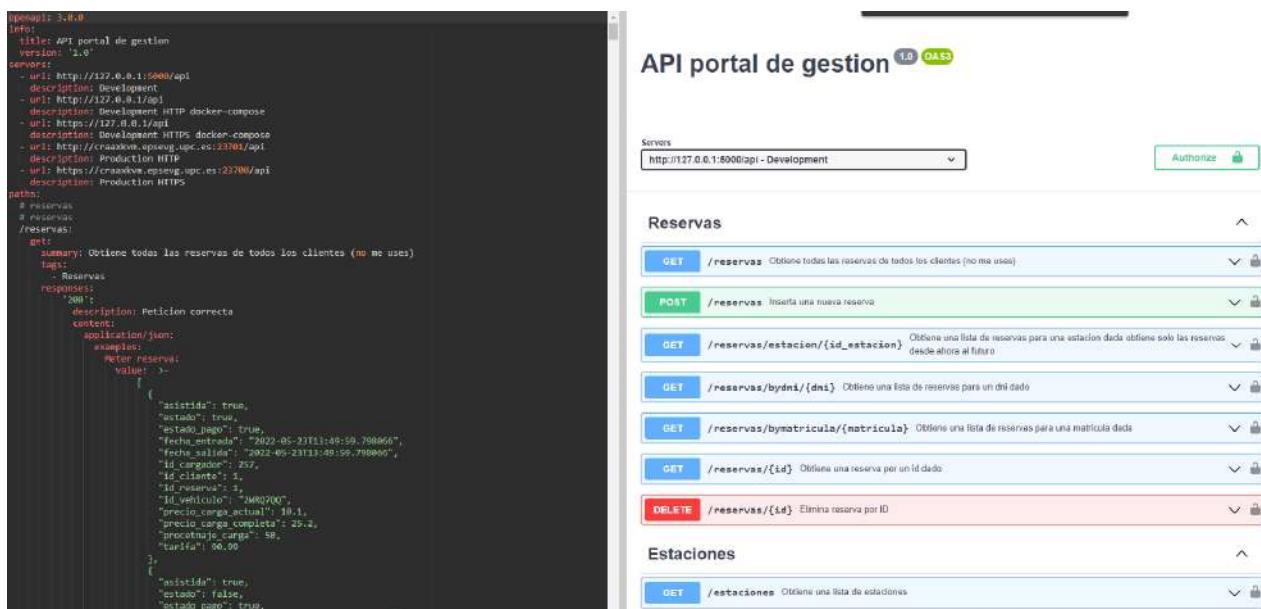
- **Makefile:** Hemos hecho uso del comando de make (comúnmente usado para compilar proyectos de C/C++) para poder usarlo a modo script para desplegar servicios de APIs y base de datos ya sea en cloud y/o edge y a más para invocar a flake8. El fichero está escrito mediante reglas y cada regla estipula los comandos/acciones que va a seguir. El fichero por defecto tiene el nombre Makefile y se suele colocar en la raíz del proyecto.
- **Flake8:** Se ha usado para hacer que el código huele bien, en otras palabras, se hace que el código escrito en python sea más entendible, más fácil de mantener y sigue el estándar o el convenio de escritura para ese lenguaje.  
<https://flake8.pycqa.org/en/latest>

Estás son las tecnologías usadas. Separaremos el escrito en dos partes básicas, la documentación y la implementación del código. Veremos ejemplos de códigos junto con comentarios.

## Documentación:

Como hemos mencionado anteriormente, para documentar cómo usar la API hemos hecho uso de la herramienta swagger, que nos permite crear una documentación de forma fácil y que sea sencilla de usar.

Para empezar a crear nuestra documentación, usamos el swagger editor, un editor online que permite ver en tiempo real los cambios que realizamos en la página, así como sus errores.



The screenshot shows the Swagger Editor interface. On the left, the YAML code for the API documentation is displayed. On the right, the generated API documentation is shown, including endpoints for Reservas and Estaciones, with their respective methods (GET, POST, DELETE) and descriptions.

```

openapi: 3.0.0
info:
  title: API portal de gestion
  version: '1.0'
servers:
  - url: http://127.0.0.1:5000/api
    description: Development
  - url: http://127.0.0.1/api
    description: Development HTTP docker-compose
  - url: https://127.0.0.1/api
    description: Development HTTPS docker-compose
  - url: https://craxxwve.epsevg.upc.es:23790/api
    description: Production HTTPS
paths:
  /reservas:
    get:
      summary: Obtiene todas las reservas de todos los clientes (no me uses)
      tags:
        - Reservas
      responses:
        '200':
          description: Petición correcta
          content:
            application/json:
              examples:
                - valor:
                    value: |
                      [
                        {
                          "asistida": true,
                          "estado": true,
                          "estado_pago": "TRUE",
                          "fecha": "2022-05-23T11:48:59.700Z",
                          "fecha_salida": "2022-05-23T13:49:59.700Z",
                          "id_cargado": 257,
                          "id_cliente": 1,
                          "id_reserva": 3,
                          "id_vehiculo": "M00200",
                          "precio_carga_acabada": 18.1,
                          "precio_carga_comprada": 20.4,
                          "procentaje_carga": 50,
                          "tarifa": 00.00
                        },
                        {
                          "asistida": true,
                          "estado": false,
                          "estado_pago": true
                        }
                      ]

```

**API portal de gestion** 1.0 DRAFT

Servers http://127.0.0.1:5000/api - Development

Reservas

GET /reservas: Obtiene todas las reservas de todos los clientes (no me uses)

POST /reservas: Inserta una nueva reserva

GET /reservas/estacion/{id\_estacion}: Obtiene una lista de reservas para una estación dada obtiene solo las reservas desde ahora al futuro

GET /reservas/bydni/{dni}: Obtiene una lista de reservas para un dni dado

GET /reservas/bymatricula/{matricula}: Obtiene una lista de reservas para una matrícula dada

GET /reservas/{id}: Obtiene una reserva por un id dado

DELETE /reservas/{id}: Elimina reserva por id

Estaciones

GET /estaciones: Obtiene una lista de estaciones

Como podemos ver en la imagen anterior, en el swagger editor tenemos en la parte izquierda el código yaml(.yml) y en la parte derecha el resultado de este código, la

documentación. Podemos ver que hay operaciones GET, POST, DELETE. Además de eso, nos permite cambiar de escenario y usar tokens.

### Fichero .yml

La parte del frontend, Swagger es el responsable de generarlo automáticamente. Solo debemos ofrecer un código .yml que cumpla con las necesidades de la herramienta. El código tiene diferentes partes y a la vez, diferentes posibilidades.

### HEADER

En el header, nos permite definir los servidores a utilizar (escenarios), la versión de la API y el título que le debemos dar.

```
openapi: 3.0.0
info:
  title: API portal de gestión
  version: '1.0'
servers:
  - url: http://127.0.0.1:5000/api
    description: Development
  - url: http://127.0.0.1/api
    description: Development HTTP docker-compose
  - url: https://127.0.0.1/api
    description: Development HTTPS docker-compose
  - url: http://craaxkvm.epsevg.upc.es:23701/api
    description: Production HTTP
  - url: https://craaxkvm.epsevg.upc.es:23700/api
    description: Production HTTPS
```

Una vez definido los headers, definiremos los PATHS.

### PATHS

Los paths serán las URIs que tendrán a disposición nuestros usuarios de la app y web. En estos, podremos definir las comandos que vamos a utilizar, junto con ejemplos, tipo de formato de data y descripciones:

```
paths:
  # reservas
  # reservas
  /reservas:
    get:
      summary: Obtiene todas las reservas de todos los clientes
      (no me uses)
```

```

tags:
  - Reservas
responses:
  '200':
    description: Peticion correcta
    content:
      application/json:
        examples:
          Meter reserva:
            value: >-
              [
                {
                  "asistida": true,
                  "estado": true,
                  "estado_pago": true,
                  "fecha_entrada":
                    "2022-05-23T13:49:59.798066",
                  "fecha_salida":
                    "2022-05-23T13:49:59.798066",
                  "id_cargador": 257,
                  "id_cliente": 1,
                  "id_reserva": 1,
                  "id_vehiculo": "2WRQ7QQ",
                  "precio_carga_actual": 10.1,
                  "precio_carga_completa": 25.2,
                  "procentaje_carga": 50,
                  "tarifa": 90.99
                },
                {
                  "asistida": true,
                  "estado": false,
                  "estado_pago": true,
                  "fecha_entrada":
                    "2022-05-23T13:49:59.808033",
                  "fecha_salida":
                    "2022-05-23T13:49:59.808033",
                  "id_cargador": 258,
                  "id_cliente": 1,
                  "id_reserva": 2,
                  "id_vehiculo": "2WRQ7QQ",
                  "precio_carga_actual": 60,
                  "precio_carga_completa": 50,
                }
              ]

```

```

        "procetnaje_carga": 33,
        "tarifa": 44.44
    }
]
'400':
  description: Peticion incorrecta
  content:
    application/json:
      examples:
        Error al meter reserva:
          value: >-
            {"error": "Malformed request syntax."}

# mandamos/resolvemos 1 reserva
post:
  summary: Inserta una nueva reserva
  tags:
    - Reservas
  requestBody:
    required: true
    content:
      application/json:
        schema:
          type: object
          required:
            - id_estacion
            - fecha_inicio
            - fecha_final
            - id_vehiculo
            - id_cliente
            - tarifa
            - asistida
            - porcentaje_carga
            - precio_carga_completo
            - precio_carga_actual
            - estado_pago
        properties:
          id_estacion:
            type: string
          fecha_inicio:
            type: string
          fecha_final:

```

```

        type: string
    id_vehiculo:
        type: string
    id_cliente:
        type: string
    tarifa:
        type: number
    asistida:
        type: boolean
    porcentaje_carga:
        type: number
    precio_carga_completo:
        type: number
    precio_carga_actual:
        type: number
    estado_pago:
        type: boolean
example:
    id_estacion: "VG1"
    fecha_inicio: "20-06-2022 11:00"
    fecha_final: "20-06-2022 18:00"
    id_vehiculo: "655RR68"
    tarifa: 12.7
    asistida: True
    porcentaje_carga: 30
    precio_carga_completo: 30
    precio_carga_actual: 5
    estado_pago: True
responses:
'200':
    description: Peticion correcta
    content:
        application/json:
            examples:
                Meter reserva:
                    value: >-
                    {
                        "asistida": true,
                        "estado": true,
                        "estado_pago": true,
                        "fecha_entrada":
                            "2022-05-23T13:49:59.798066",

```

```

        "fecha_salida":  

        "2022-05-23T13:49:59.798066",  

            "id_cargador": 257,  

            "id_cliente": 1,  

            "id_reserva": 1,  

            "id_vehiculo": "2WRQ7QQ",  

            "precio_carga_actual": 10.1,  

            "precio_carga_completa": 25.2,  

            "procetnaje_carga": 50,  

            "tarifa": 90.99  

        }  

'400':  

    description: Peticion incorrecta  

    content:  

        application/json:  

        examples:  

            Error al meter reserva:  

            value: >-  

                {"error": "Malformed request syntax."}

```

## COMPONENTES

Al final del fichero podremos añadir componentes que nos permitirán añadir personalizaciones, como ahora, el acceso a las solicitudes mediante un token.

```

components:  

    securitySchemes:  

        token:  

            type: apiKey  

            description: API key to authorize requests.  

            name: x-access-tokens  

            in: header  

        security:  

            - token: []

```

Los archivos definidos en la documentación se encuentran en el siguiente link:  
<https://github.com/PTIN2022/A2/tree/main/API>

## GITHUB PAGES

Como hemos mencionado anteriormente, para permitir la interacción con el documento al resto de equipos, usamos el github pages que nos permite desplegar un dominio en la red internet de forma gratuita.

Para ello, lo configuramos para que en cada reinicio de la rama main (cada merge, commit, etc.) se desplegará de nuevo.

Para ello, se creó un index.html, responsable de desplegar el frontend del swagger, permitir cambiar de ramas en el main y cambiar entre la documentación de la api web y api app.

<https://github.com/PTIN2022/A2/blob/main/index.html>

Página de la API:

<https://ptin2022.github.io/A2/>

The screenshot shows the API portal de gestion's Swagger UI interface. At the top, there's a navigation bar with tabs for 'main', 'WEB' (which is highlighted in blue), and 'APP'. Below the navigation is the Swagger UI header with the title 'API portal de gestion' and a GitHub link. The main content area is divided into sections: 'Incidencias' and 'Estaciones'. Under 'Incidencias', there are several API endpoints listed with their methods (GET, POST, PUT, DELETE) and descriptions. Each endpoint has a lock icon next to it. The 'Estaciones' section is partially visible below the 'Incidencias' section.

## IMPLEMENTACIÓN

### MAIN

Para desplegar la API, usaremos flask, que nos permitirá crear un backend de forma sencilla.

Como tal flask nos ofrece un servidor de desarrollo, pero para pasarlo a producción hemos usado: gunicorn “WSGI HTTP Server for UNIX”.

Servidores de producción:

Tenemos 2 servidores, klaus y edge.

Para ambos servidores tenemos las mismas dependencias:

1. docker
2. docker-compose
3. mosquitto\_sub (pruebas)
4. mosquitto\_pub (pruebas)

Los servidores tienen los siguientes puertos:

- Puerto 22 -> SSH: Para poder conectarnos remotamente a las máquinas.
- Puerto 80 -> HTTP: Para poder hacer solicitudes HTTP a la API.

- Puerto 443 -> HTTPS: Para poder hacer solicitudes HTTP seguras a la API.
- Puerto 42069 -> MQTT: Puerto por el que se enviarán los mensajes MQTT.

La manera que hemos desplegado los servicios de docker es usando docker-compose:  
Ejemplo edge:

```

docker-compose-edge.yml →
1 version: "3.9"
2 services:
3   api:
4     build: ./edge-service-api
5     expose:
6       - 5000
7     depends_on:
8       - brokeredge
9       - mysql
10    environment:
11      SQLALCHEMY_DATABASE_URI: $SQLALCHEMY_DATABASE_URI
12      PYTHONUNBUFFERED: 1
13      MQTT_BROKER_URL: $MQTT_BROKER_URL
14      MQTT_BROKER_PORT: $MQTT_BROKER_PORT
15      MQTT_LOCAL_CLOUD_URL: $MQTT_LOCAL_CLOUD_URL
16      MQTT_LOCAL_CLOUD_PORT: $MQTT_LOCAL_CLOUD_PORT
17    restart: unless-stopped
18
19   nginx:
20     build: ./edge-nginx
21     ports:
22       - "80:80"
23       - "443:443"
24     depends_on:
25       - api
26     volumes:
27       - nginxcerts:/etc/ssl/certs
28     restart: unless-stopped
29
30   brokeredge:
31     build: ./broker
32     ports:
33       - "42069:1883"
34     expose:
35       - 1883
36     restart: unless-stopped
37
38   mysql:
39     image: mysql:8
40     ports:
41       - 3306:3306
42     volumes:
43       - ~./gesys/edge-mysql-final:/var/lib/mysql
44     environment:
45       MYSQL_ROOT_PASSWORD: $MYSQL_ROOT_PASSWORD
46       MYSQL_DATABASE: $MYSQL_DATABASE
47     restart: unless-stopped
48
49   volumes:
50     nginxcerts:

```

Aquí tenemos declarados los 4 dockers, sus puertos expuestos y sus variables de entorno de cada uno.

cloud:

```
docker-compose-cloud.yml
1 version: "3.9"
2 services:
3   api:
4     build: ./cloud-service-api
5     expose:
6       - 5000
7     depends_on:
8       - brokercloud
9       - mysql
10    environment:
11      SQLALCHEMY_DATABASE_URI: $SQLALCHEMY_DATABASE_URI
12      PYTHONUNBUFFERED: 1
13      MQTT_BROKER_URL: $MQTT_BROKER_URL
14      MQTT_BROKER_PORT: $MQTT_BROKER_PORT
15      MQTT_LOCAL_EDGE_URL: $MQTT_LOCAL_EDGE_URL
16      MQTT_LOCAL_EDGE_PORT: $MQTT_LOCAL_EDGE_PORT
17    restart: unless-stopped
18
19   front:
20     build: ../GeSyS-Front
21     expose:
22       - 3000
23     restart: unless-stopped
24
25   nginx:
26     build: ./cloud-nginx
27     ports:
28       - "80:80"
29       - "443:443"
30     depends_on:
31       - api
32       - front
33     volumes:
34       - nginxcerts:/etc/ssl/certs
35     restart: unless-stopped
36
37   brokercloud:
38     build: ./broker
39     ports:
40       - "42069:1883"
41     expose:
42       - 1883
43     restart: unless-stopped
44
45   mysql:
46     image: mysql:8
47     ports:
48       - 3306:3306
49     volumes:
50       - ~/.gesys/cloud-mysql-final:/var/lib/mysql
51     environment:
52       MYSQL_ROOT_PASSWORD: $MYSQL_ROOT_PASSWORD
53       MYSQL_DATABASE: $MYSQL_DATABASE
54       MYSQL_ROOT_HOST: "%"
55     restart: unless-stopped
56
57   volumes:
58     nginxcerts:
```

Como podemos ver el docker-compose del cloud también tiene sus 4 servicios.

Para levantar los dockers en edge simplemente clonamos el repositorio A2 y dentro hacemos un make edge

```
# git clone https://github.com/PTIN2022/A2
# cd A2
# make edge
```

Para cloud debemos clonar el repositorio A2 y el del front, situarlos en la misma carpeta y después entrar en el A2 y hacer un make cloud.

```
# git clone https://github.com/PTIN2022/A2
# git clone https://github.com/PTIN2022/GeSyS-Front
# cd A2
# make cloud
```

Una vez hecho los make habrán arrancado todos los servicios, se habrán autoconfigurado y podremos ver los logs usando make cloud-logs y make edge-logs.

## Flask

Para eso, generaremos un proyecto en flask que contendrá el app.py, donde estará la configuración necesaria para arrancar nuestro backend.

Podremos incluir diferentes configuraciones, como ahora la del MQTT o el hash a utilizar para el token.

```
app.config["SQLALCHEMY_DATABASE_URI"] =
os.getenv('SQLALCHEMY_DATABASE_URI', "sqlite:///test.db")
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False # TODO: review
app.config["TESTING"] = False
app.config['MQTT_BROKER_URL'] = os.getenv('MQTT_BROKER_URL',
'test.mosquitto.org') # use the free broker from HIVEMQ
app.config['MQTT_BROKER_PORT'] = int(os.getenv('MQTT_BROKER_PORT',
"1883")) # default port for non-tls connection
app.config['MQTT_USERNAME'] = os.getenv('MQTT_USERNAME', '') # set the
username here if you need authentication for the broker
app.config['MQTT_PASSWORD'] = os.getenv('MQTT_PASSWORD', '') # set the
password here if the broker demands authentication
app.config['MQTT_KEEPALIVE'] = int(os.getenv('MQTT_KEEPALIVE', "5")) # # set the time interval for sending a ping to the broker to 5 seconds
app.config['MQTT_TLS_ENABLED'] = os.getenv('MQTT_TLS_ENABLED', False) # set TLS to disabled for testing purposes
app.config["ON_TEST"] = bool(os.getenv('ON_TEST', False))

app.config['SECRET_KEY'] =
'bf9d91da2b703c30e770279ee82b17692def66a956b25b7c2d92f4088dfea293'
```

```

app.config['SALT'] =
'\\xd2\\xf\\xca\\x0c\\xc5\\xe6:\\xa9\\xeb<\\x07j\\r\\xb6\\xef\\xda$\\xb8\\xc5X\\ak\\xab\\x
9d\\x0e\\x99\\xaf\\xc7\\x94\\xba'.encode("utf-8")
app.config["EXPIRE_TOKEN_TIME"] = 2*60 # mins

```

Podemos ver las diferentes configuraciones. Las configuraciones más importantes en la API son la 'SECRET\_KEY' y la de 'EXPIRE\_TOKEN\_TIME', que permite tener la clave privada para identificar las claves públicas y permiten darle una vida al token generado para el usuario.

Además de eso, podemos generar nuestra BD, mediante la función init\_db() que la llamaremos en el body main de la aplicación.

```

def init_db():
    time.sleep(5)
    db.init_app(app)
    with app.app_context():
        insert = bool(os.getenv('INSERT_FAKE', False))
        if insert:
            db.drop_all()
            db.create_all()

        if insert:
            fakedata()

```

Como podemos ver, se inicializa la app en la base de datos y en caso que hayamos indicado mediante una comanda externa (Makefile) si queremos generar datos, se generarán. Los datos falsos se pueden generar de dos formas: mediante un archivo estático que hemos definido o un archivo con generación aleatoria de valores.

Estos dos archivos se pueden encontrar en el siguiente link con nombre; fake\_data.py y fake\_data\_statico.py. Los dos son iguales tanto en el edge como en el cloud:

<https://github.com/PTIN2022/A2/tree/main/edge-service-api/utils>

En el main de la aplicación, tenemos lo siguiente:

```

lock.acquire()
try:
    init_db()
finally:
    lock.release()

if __name__ == "__main__": # pragma: no cover
    print("=====")

```

```

print("Test me on: http://ptin2022.github.io/A2/")
print("=====")

insert = bool(os.getenv('INSERT_FAKER', False))
if not insert:
    app.run(host="0.0.0.0")

```

Básicamente, realiza un lock para generar la base de datos y luego enciende el servidor en backend.

Para la parte de la api, hay que generar las URLs, que son los PATHs a los que se llamarán para generar las peticiones CRUD que ofrecemos.

Para ello, las registramos en la aplicación mediante la comanda `app.register_blueprint()`. Respectivamente, esto último cambia entre el edge y el cloud, ya que el edge hace uso de unas apis y cloud de otras.

#### CLOUD:

```

app.register_blueprint(incidencias, url_prefix='/api')
app.register_blueprint(estaciones, url_prefix='/api')
app.register_blueprint(trabajador, url_prefix='/api')
app.register_blueprint(clientes, url_prefix='/api')
app.register_blueprint(reservas, url_prefix='/api')
app.register_blueprint(promociones, url_prefix='/api')
app.register_blueprint(soporte, url_prefix='/api')
app.register_blueprint(estadisticas, url_prefix='/api')
app.register_blueprint(login, url_prefix='/api')
app.register_blueprint(logout, url_prefix='/api')

```

#### EDGE:

```

app.register_blueprint(reservas, url_prefix="/api")
app.register_blueprint(estaciones, url_prefix="/api")
app.register_blueprint(clientes, url_prefix="/api")
app.register_blueprint(login, url_prefix='/api')
app.register_blueprint(logout, url_prefix='/api')
app.register_blueprint(soporte, url_prefix='/api')
app.register_blueprint(avisos, url_prefix='/api')
app.register_blueprint(cupones, url_prefix='/api')
app.register_blueprint(pagos, url_prefix='/api')
app.register_blueprint(modelos, url_prefix="/api")
app.register_blueprint(vehiculos, url_prefix="/api")

```

## ENTORNO

Una vez tenemos definido estos conceptos básicos del main, vamos a ver cómo se desarrolla una petición en la API.

Para hacer dicha petición a la API, primero se debe generar una petición http/https que tenga incluido los campos necesarios y los headers necesarios.

Para generar bien estas solicitudes, haremos uso del swagger y de una aplicación externa llamada "Insomnia" que nos permitirá chequear de forma fácil y sencilla las peticiones.

Haremos un ejemplo con /estaciones de la API APP.

Para ello, solicitaremos todas las estaciones disponibles en la BBDD mediante una petición GET al edge.

Generamos la petición en el swagger, que nos ofrecerá un CURL.

The screenshot shows the Insomnia REST Client interface. At the top, it says "GET /estaciones Obtener una lista de estaciones". Below that, there's a "Parameters" section with "No parameters". Under "Responses", there's a "Curl" section containing the following code:

```
curl -X 'GET' \
  'http://craaxkvm.epsevg.upc.es:23701/api/estaciones' \
  -H 'accept: application/json'
```

Este CURL, lo copiaremos. En insomnia generamos una nueva solicitud Http request y pegaremos el curl. Insomnia está muy bien para trabajar en estos entornos porque además de ser una herramienta flexible para probar las llamadas a los end-points, también permite generar el código para diferentes lenguas de programación, incluyendo los headers y datos pertinentes.

The screenshot shows the Insomnia REST Client interface after sending the curl request. The status bar at the top says "200 OK" with a response time of "21.9 ms" and a size of "2.5 KB". The "Preview" tab shows the JSON response body, which contains two station objects:

```
[{"id": 1, "capacidad": 32, "ciudad": "Vilanova i la geltr\u00fa", "direccion": "Ronda de l'exposici\u00f3n", "estado": "Activa", "id_estacion": 1, "latitud": 41.2176, "longitud": 1.37279, "moder_est": "M01", "ocupacion_actua": 1, "piso": "P01", "potencia_computadora": 230, "potencia_maquina": 100, "telefono": "+34626492240", "zona": "Zona Industrial"}, {"id": 2, "capacidad": 32, "ciudad": "Vilanova i la geltr\u00fa", "direccion": "Ronda de l'exposici\u00f3n", "estado": "Activa", "id_estacion": 2, "latitud": 41.2176, "longitud": 1.37279, "moder_est": "M01", "ocupacion_actua": 1, "piso": "P01", "potencia_computadora": 230, "potencia_maquina": 100, "telefono": "+34626492240", "zona": "Zona Industrial"}]
```

Aquí podemos ver como se nos devuelve el resultado de la petición junto a su status (200 ok).

## Llamada a la API

Ahora que tenemos definido el entorno en el que trabajaremos, vamos a explicar como funciona de manera interna.

Un usuario realiza una llamada a un EP. Este endpoint suele ser una URI, que puede ser estática (sin valores) o con valores dentro de dicha URI. Esta llamada se realizará a los

puertos 23600/23601 y 23700/23701 en los servidores del CRAAX, que son los puertos HTTP/HTTPS respectivos.

## ROUTES

Una vez dentro, la aplicación backend redirigirá las llamadas a los diferentes archivos que hemos definido en routes.

Estos archivos son los que contienen todas las URIs de los diferentes EP disponibles. Cada uno de ellos es responsable de una sección (definida en el MAIN anterior, con el **register\_blueprint**). Aquí se tratan los datos y se envían al controller, responsable de devolver una respuesta.

Para definir un archivo en routes es muy sencillo, ya que solo debemos incluir 4 campos necesarios.

El campo Blueprint, que identifica dicho EP. Este campo solo hace falta declararlo una vez:

```
promociones = Blueprint('promociones', __name__)
```

La URI y que tipo de llamada (GET, POST, PUT, DELETE), en ese EP:

```
@promociones.route('/promociones', methods=['GET'])
```

Si es necesario un token o no:

```
@token_required
```

Y para finalizar la dicha función:

```
def get_promocion(current_trabajador):
    if current_trabajador.cargo == "administrador" or
    current_trabajador.cargo == "encargado":
        respuesta = control.get_all_promociones()
        return jsonify(respuesta)
    else:
        return jsonify({"error": "User not authorized."}), 401
```

Ponemos un ejemplo completo de como quedaría la función:

```
@promociones.route('/promociones/<id_promo>', methods=['GET'])
@token_required
def get_promo_id(current_trabajador, id_promo):
    if current_trabajador.cargo == "administrador" or
    current_trabajador.cargo == "encargado":
        respuesta = control.get_promo_id(id_promo)
        if respuesta:
            return jsonify(respuesta), 200
        else:
            return jsonify({"error": "Promocion not found."}), 404
    else:
```

```
return jsonify({"error": "User not authorized."}), 401
```

El ejemplo anterior podemos observar que se le pasa un parámetro por la uri que es id\_promo, la cual de forma automática se le pasa como un parámetro de la función que definimos para dicha llamada. Aparte, en la parte la función devolvemos los status error HTTP pertinentes, y además de ello, comprobamos los cargos de los trabajadores mediante el token.

Dentro de la función podemos ver cómo llamamos a una función de controller. Es el archivo con las funciones responsables de devolver un valor dependiendo de lo que se solicite.

## CONTROLLER

En este archivo creamos las funciones que mediante querys devolvemos los valores que nos han pedido.

Un ejemplo muy sencillo es el siguiente, seguido de los ejemplos anteriores:

```
def get_all_promociones():
    p = Promociones.query.all()
    return PromocionesSchema(many=True).dump(p)
```

En esta función devolvemos todas las promociones existentes. Para devolver los datos, hacemos uso de marshmallow que permite tratar los datos de los modelos ágilmente. En el ejemplo de arriba, simplemente con un many=True nos permite devolver todos los valores. Otro ejemplo, también seguido del código visto anteriormente, vemos que las querys también son muy fáciles de hacer y devolver los datos junto con el marshmallow se hace forma ágil:

```
def get_promo_id(id_promo):
    p = Promociones.query.filter(Promociones.id_promo ==
id_promo).one_or_none()
    return PromocionesSchema().dump(p)
```

## TOKEN

Para verificar que el usuario que realiza la llamada al EP es un usuario con permisos, usamos el token. El token es una clave en SHA256, que permite identificar el trabajador, si la clave es válida y cuanto tiempo le queda a la clave.

Para hacer esto posible, generamos un EP específico que se llama "Autenticación", tanto en cloud como en edge.

Este EP permite obtener el token y eliminarlo.

```
@login.route('/login', methods=['POST'])
...
@logout.route('/logout', methods=['GET'])
...
```

En las funciones control de dicho login, lo que se hace es llamar a unas funciones que hemos declarado en un archivo externo llamado "util.py" dentro de la carpeta utils.

Hay dos funciones principales:

```
def token_required(f):
```

```

@wraps(f)
def decorator(*args, **kwargs):
    token = None
    if 'x-access-tokens' in request.headers:
        token = request.headers['x-access-tokens']
    if not token:
        return jsonify({'message': 'a valid token is missing'})
    try:
        data = jwt.decode(token, app.config['SECRET_KEY'],
algorithms=["HS256"])
        current_trabajador = Trabajador.query.filter(Trabajador.email
== data['email']).one_or_none()
    except jwt.exceptions.ExpiredSignatureError:
        return jsonify({'message': 'token is expired'})
    except: # noqa: E722
        # TODO: cuidado aqui va cualquier excepcion
        return jsonify({'message': 'token is invalid'})

    return f(current_trabajador, *args, **kwargs)
return decorator

```

Esta función, dado un token, comprueba si es válido. Si no es válido, devuelve un mensaje con el error. Si el token es válido, se devuelve al usuario.

Este función es la que llaman los EP cuando llevan:

```
@token_required
```

También creamos una función para encriptar las contraseñas de los usuarios. Esta función es la siguiente:

```

def encrypt_password(password):
    key = hashlib.pbkdf2_hmac(
        'sha256', # The hash digest algorithm for HMAC
        password.encode('utf-8'), # Convert the password to bytes
        app.config['SALT'], # Provide the salt
        100000 # It is recommended to use at least 100,000 iterations of
SHA-256
    )
    return str(key)

```

Básicamente, dado una contraseña, se encripta usando la clave privada del servidor junto con SHA256 y la devuelve en un formato string.

## MARSHMALLOW

Marshmallow funciona mediante la creación de una clase que representa la esquematización de los datos de cada uno de los modelos. Estas clases se definen como <modelo>Schema y usan las funciones de la librería SQLAlchemyAutoSchema para modelizar los datos. A continuación veremos unos ejemplos de su uso.

Nos ha permitido modificar directamente los valores a devolver, como ahora, los elementos que devuelve un ticket:

```
class TicketSchema(SQLAlchemyAutoSchema):
    class Meta:
        fields = ('id_ticket', 'fecha', 'asunto', 'estado', 'mensaje',
        'id_cliente')
```

En los cuales indicamos que queremos que devuelva en concreto

O si queremos excluir algun campo como ahora, la contraseña.

```
class ClienteSchema(SQLAlchemyAutoSchema):
    vehiculos = Nested(VehiculoSchema, many=True)

    class Meta:
        model = Cliente
        exclude = ('password',)
```

Además, en la imagen se puede observar que llamamos a una función Nested que nos permite, por cada llamada al ClienteSchema, devolver su vehículo.

*Estos Schemas pertenecen a los modelos, los cuales los veremos reflejados más adelante cuando se hable de la base de datos y su modelización con SQLAlchemy.*

## Comentarios API

Este apartado ha intentado ser un reflejo rápido y por encima de lo que hemos realizado en la parte de las APIs. No por ello, no ha sido costoso. Hemos realizado más de 40 EP los cuales ofrecen una respuesta. Todos los EP que hemos creado, siguen el esquema comentado en la implementación y en la documentación de este apartado, con cada uno de ellos teniendo aspectos diferentes que se adaptan a las necesidades de nuestros usuarios. A continuación, ofreceremos links a nuestro repositorio público de los archivos fundamentales a la API, para que puedan ser examinados a gusto. Hay casos especiales como ahora, en ReservasController, el cual invitamos ver, para que se vea a los que nos referimos al decir que cada EP se adapta a las necesidades de los usuarios

<https://github.com/PTIN2022/A2/blob/main/edge-service-api/controller/reservasController.py>

En este caso, en la función post\_reservas comprobamos que dado una estación y una reserva, haya cargadores libres entre las franjas horarias de dicha reserva para esa estación. Si no hay cargadores libres, se devuelve un error.

## Bibliografia API

MODELO (igual para edge y cloud):

<https://github.com/PTIN2022/A2/blob/main/cloud-service-api/models/model.py>

CLOUD

<https://github.com/PTIN2022/A2/tree/main/cloud-service-api/routes>

<https://github.com/PTIN2022/A2/tree/main/cloud-service-api/controller>

<https://github.com/PTIN2022/A2/tree/main/cloud-service-api/utils>

EDGE

<https://github.com/PTIN2022/A2/tree/main/edge-service-api/routes>

<https://github.com/PTIN2022/A2/tree/main/edge-service-api/utils>

<https://github.com/PTIN2022/A2/tree/main/edge-service-api/controller>

Códigos de error y su definición

Durante el desarrollo de la API, hemos usado la siguiente lista de errores para indicar al usuario cuando algo no iba mal.

### CÓDIGOS DE ERROR - CLIENTE

Los códigos de error comienzan por valor 400. Para ello, haremos uso de los siguientes códigos:

#### 400 Bad request

La sintaxis ha sido incorrecta o el servidor no ha podido procesarla.

#### 401 Unauthorized

El usuario que intenta ejecutar la solicitud es un intruso

#### 403 Forbidden

Falta de permisos para ejecutar la solicitud

#### 404 Not found

Solicitud no encontrada.

#### 408 Request Timeout

Tiempo de solicitud agotado

## DOCKERS

Docker para el despliegue del frontend (página de administración)

Usaremos la guia que nos ofrece next.js para crear el docker. Se encuentra en el siguiente link:

<https://github.com/vercel/next.js/tree/canary/examples/with-docker>

Para poder crear el docker adecuadamente, copiaremos el archivo Dockerfile que se encuentra en el github y lo pegaremos en el directorio donde se encuentra el frontend.

Una vez tenemos el fichero Dockerfile, debemos hacer unos cuantos ajustes.

Debemos comentar cualquier comando a yarn ya que nosotros usaremos el npm.

Lo siguiente es añadir una línea extra en el archivo de configuración de next.config.js que se encuentra en la carpeta raíz del frontend. Debemos añadir un modulo experimental:

```

module.exports = {
  experimental: {
    outputStandalone: true,
  },
}

```

Puede compaginarse con otros módulos.

Una vez tenemos estos elementos modificados, solo debemos ejecutar estas dos comandas:

**sudo docker build -t frontend .**

Nos montará la imagen en base del archivo Dockerfile que tenemos en la carpeta raíz del frontend.

Una vez se monte adecuadamente la imagen (Debe salir como successfully built y successfully tagged).

Ejecutamos la comanda siguiente para abrir la página por el puerto 8080 (HTTP).

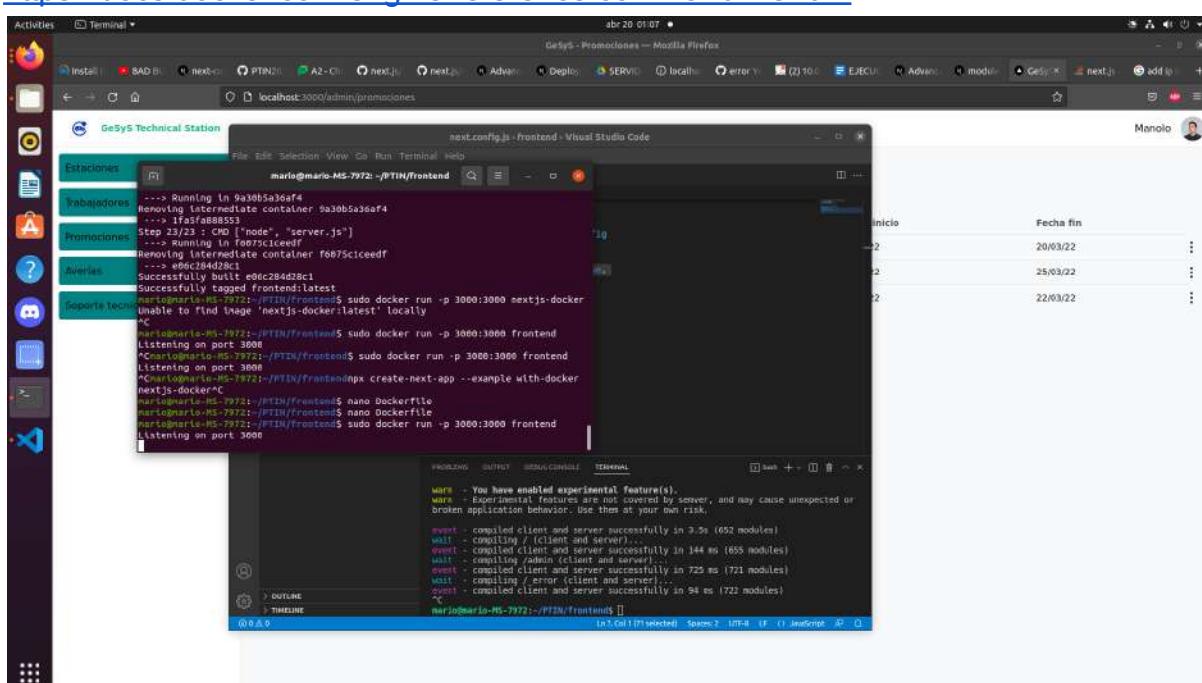
**sudo docker run -p 8080:3000 frontend**

Y ya estaría funcionando.

Página web:

<http://craaxkvm.epsevg.upc.es:23601/login>

<https://docs.docker.com/engine/reference/commandline/run/>



1. Captura del docker funcionando en local

## Docker NGINX cloud

Tenemos un docker nginx haciendo proxypass al front y a la api, siendo que todo lo que vaya a /api redirigirá al docker api y el resto redirigirá al front.

Este nginx también crea el servidor SSL usando unos certificados creados en el propio docker:

```
FROM nginx:1.21-alpine

RUN apk update \
    && apk add openssl

RUN rm /etc/nginx/conf.d/default.conf
COPY ./nginx.conf /etc/nginx/conf.d
COPY ./snippets /etc/nginx/snippets
COPY ./entrypoint.sh /

ENTRYPOINT ["./entrypoint.sh"]
```

Donde el entry point es:

```
# bin/sh
# Gen keys

if ! [ -f /etc/ssl/certs/nginx-selfsigned.key ]; then
    openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/certs/nginx-selfsigned.key -out /etc/ssl/certs/nginx-selfsigned.crt -subj '/CN=craaxkvm.epseyg.es/0=Gesys./C=ES'
fi

if ! [ -f /etc/ssl/certs/dhparam.pem ]; then
    openssl dhparam -out /etc/ssl/certs/dhparam.pem 2048
fi

nginx -g "daemon off;"
```

Como podemos ver generamos los certificados ssl autofirmados y el diffieHellman.  
Para edge tenemos el mismo docker pero sin la parte de front (solo lo usamos para el ssl)

## Docker broker

Usamos una imagen de ubuntu y broker mosquitto

```
# syntax=docker/dockerfile:1
FROM ubuntu:latest
ARG DEBIAN_FRONTEND=noninteractive
WORKDIR /app
RUN apt-get update && apt-get install -y --no-install-recommends apt-utils
RUN apt install -y mosquitto && apt install -y mosquitto-clients
COPY .
CMD ["mosquitto", "-c", "mosquitto.conf"]
```

## Docker MYSQL

Usamos la imagen oficial de mysql versión 8, donde tenemos un volumen para los datos de la misma para sobrevivir a reinicios de los dockers.

## Docker API:

Usamos la imagen oficial de python:3.10

```
FROM python:3.10

COPY requirements/requirements.txt .
RUN pip install -r requirements.txt
COPY . /opt/stack
WORKDIR /opt/stack

ENV TZ=Europe/Madrid
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && dpkg-reconfigure -f noninteractive tzdata

CMD gunicorn -b 0.0.0.0:5000 --error-logfile "-" --capture-output --enable-stdio-inheritance --log-level "debug" app:app
```

Copiamos nuestro stack cloud-service-api o edge-service api e instalamos el requerimientos del proyecto y arrancamos un gunicorn con la aplicación flask, sacando los logs por pantalla

# Desplegament de servei Web i definició de Protocols

## PROTOCOLS

### 1.DNS

#### 1.1 Documentación

El **DNS** (*Domain Name System, Sistema de Nombres de Dominio*) es un conjunto de protocolos y servicios que permite a los usuarios utilizar nombres en vez de tener que recordar direcciones IP numéricas. Ésta es ciertamente la función más conocida de los protocolos DNS: la asignación de nombres a direcciones IP. Por ejemplo, si la dirección IP de [www.upc.es](http://www.upc.es) es 150.214.170.139, la mayoría de la gente llega a este equipo especificando [www.upc.es](http://www.upc.es) y no la dirección IP. Además de ser más fácil de recordar, el nombre es más fiable. La dirección IP podría cambiar por muchas razones, sin que tenga que cambiar el nombre del dominio.

#### 1.2 Modo de aplicación

Es uno de los elementos básicos para tener una buena conexión a Internet y así interrelacionar de manera mucho más cómoda el dominio de la página web con el dominio de la aplicación móvil del usuario.

Utilizar DNS como consecuencia implica una navegación web más rápida, un servicio confiable, y al mismo tiempo, una mejor protección.

## 2.API REST

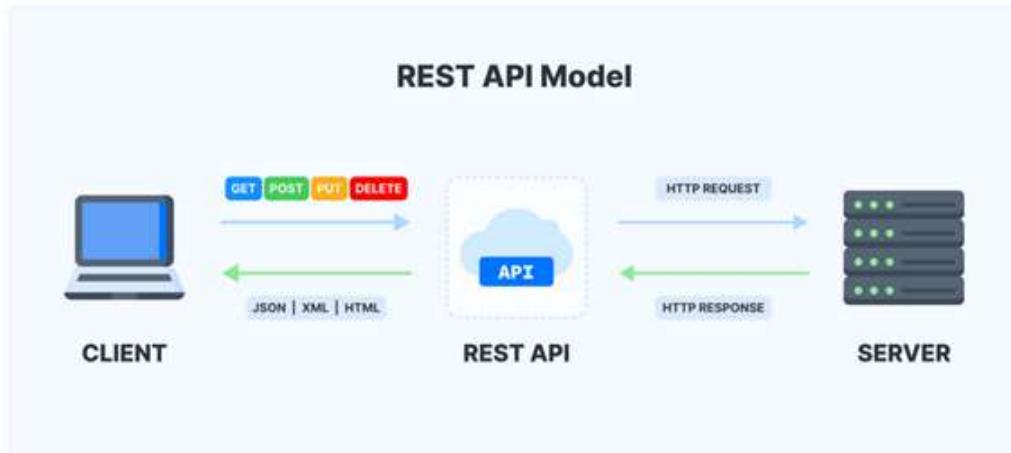
#### 2.1 Documentación

Las API son un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones, **permitiendo la comunicación entre dos aplicaciones** de software a través de un conjunto de reglas, donde se establece el contenido que se necesita por parte del consumidor (la llamada) y el que requiere el productor (la respuesta).

La API utiliza varios protocolos y arquitecturas para enviar solicitudes y respuestas.

En este caso nos centramos en API REST que utiliza implementaciones cliente-servidor de forma independiente. REST»: REpresentational State Transfer, es un tipo de arquitectura de desarrollo web que se apoya totalmente en el

estándar(protocolo) HTTP para diseñar API. Así, cualquier dispositivo / cliente que entienda de HTTP puede hacer uso de su propia API REST.



## 2.2 Modo de aplicación

Se aplica este protocolo por las siguientes funcionalidades:

Independencia: Debido a que hay una separación entre cliente servidor , el protocolo facilita que los desarrollos de las diferentes partes de un proyecto se puedan dar de manera independiente. Se adapta a cualquier tipo de sintaxis o plataforma.

Flexibilidad y portabilidad: Al haber una separación cliente-servidor hace que se puede migrar a otros servidores y practicar cambios en la base de datos en todo momento de manera transparente.De esta forma el front y el back se pueden alojar en servidores diferentes.

Escalabilidad: Este protocolo destaca por su escalabilidad. Los desarrolladores pueden integrar fácilmente componentes y características adicionales con el diseño de la API REST sin crear nuevas aplicaciones.

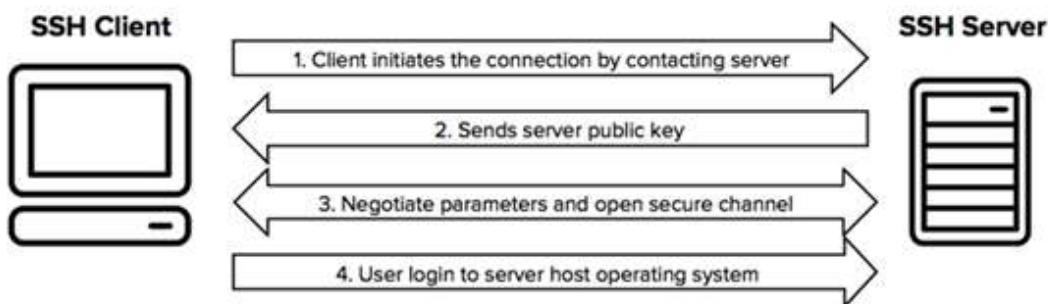
## 3.SSH

### 3.1 Documentación

Secure SHell (SSH) es un protocolo que permite un host conectarse remotamente a otro vía terminal, permitiendo administrar el sistema, controlar procesos y transferir archivos. SSH destaca por su capacidad de encriptación de la sesión de conexión y autenticación de usuario y servidor.

SSH funciona con el protocolo TCP en capa transporte en puerto 22, y los elementos que proporcionan la seguridad a éste protocolo, son principalmente sus 3 tecnologías de cifrado: Cifrado simétrico, cifrado asimétrico y hashing.

1. genera una clave privada clave que es compartida entre el cliente y el servidor en la conexión SSH ya que durante esta conexión, encriptan y a la misma vez desencriptan los mensajes enviados en el canal. Esta clave nunca es transmitida, sino que es generada por cada parte con un algoritmo de intercambio de claves generan la clave desde los datos públicos de cada parte.
2. Consiste en que las dos partes generan un par de claves pública-privada, donde la pública es compartida y la privada nunca revelada. Los mensajes que se encriptan con la clave pública, sólo pueden descifrarse con su respectiva clave privada. Es utilizada durante el cifrado simétrico para transmitir información segura y luego establecer un canal de forma segura.
3. abiertamente y la privada nunca es revelada El hashing es la tecnología que se usa para verificar la autenticidad de cada uno de los mensajes



### 3.2 Modo de aplicación

SSH nos permite acceder a servidores con cualquier sistema operativo instalado o cualquier tipo de gadget que esté conectado a internet y que tenga establecido este protocolo para la comunicación.

SSH proporciona los siguientes tipos de protección:

1. Una vez se ha realizado una primera conexión a un servidor, el cliente puede verificar que se está conectando al mismo servidor durante posteriores sesiones.
2. El cliente transmite su información de autenticación al servidor, como el nombre de usuario y la contraseña, en formato cifrado extremo a extremo.

3. Todos los datos enviados y recibidos durante la conexión se transfieren por medio de encriptación.
4. El cliente tiene la posibilidad de usar X11 desde aplicaciones lanzadas desde el intérprete de comandos Esta técnica proporciona una interfaz gráfica segura y un medio seguro para usar aplicaciones gráficas sobre una red.

Como el protocolo SSH cifra todo lo que envía y recibe, se puede usar para asegurar protocolos inseguros, El servidor SSH puede convertirse en un conducto para convertir en seguros los protocolos inseguros mediante el uso de una técnica llamada reenvío por puerto incrementando la seguridad del sistema y la privacidad de datos.

## 4.MQTT

Dado que el grupo que lo requiere ya ha hecho la búsqueda que requería, la tarea será más bien de validación haciendo igualmente un pequeño resumen.

### 4.1 Documentación

Protocolo que se usa principalmente para la comunicación en el borde de la red en la estructura continuum, es decir se suele ubicar en la zona **IoT- Edge**.

Se caracteriza por ser un protocolo ligero en cuanto al uso de recursos y a cabeceras, aun implementando sobre **TCP**.

Este protocolo emplea el paradigma **Publisher-Subscriber**, con un broker intermedio entre el publicador y el suscriptor.

Las suscripciones se hacen en base a **temas**, que siguen un orden jerárquico.

El **broker** suele tener una base de datos pequeña y por ese motivo no suele guardar los datos, es decir que una vez los ha enviado a los suscriptores a los que les pertoca los suele eliminar.

Para solucionar eso, el Publicador puede marcar los datos con el flag **retain** y de esta forma el broker los conservará, esperando a nuevos suscriptores del tema para enviarselo.

Uno de los broker open source más populares para la implementación de este protocolo es Mosquitto.

MQTT nos permite regular la carga deseada sobre la red y QoS, permitiendo ajustar entre 3 diferentes **QoS** {0:best effort, 1:at least once. 2: exactly once} donde según aumenta el número, aumenta la calidad del servicio y también la carga sobre la red.

Cabe tener en cuenta que MQTT deja bastante en las manos del implementador la **securización** de su comunicación.

Más información sobre la última versión de MQTT:

<https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>

Más sobre mosquitto:

<https://mosquitto.org/>

## 4.2 Modo de aplicación

Un esquema que define de manera simple el comportamiento de MQTT:



# 5. HTTPS

## 5.1 Documentación

**HTTPS** (*Protocolo de transporte de hipertexto seguro*) es lo mismo que HTTP, pero utiliza una capa de conexión segura (SSL) por motivos de seguridad. Algunos ejemplos de sitios que usan HTTPS incluyen sitios web de banca e inversión, sitios web de comercio electrónico y la mayoría de los **sitios web que requieren que inicie sesión**.

Si bien esto es muy poco probable, no es un pensamiento reconfortante que alguien pueda estar capturando su número de tarjeta de crédito u otra información personal que ingresa en un sitio web. Por lo tanto, los sitios web seguros utilizan el protocolo HTTPS para cifrar los datos que se envían de un lado a otro con cifrado SSL. Si alguien capturara los datos que se transfieren a través de HTTPS, serían irreconocibles.

## 5.2 Modo de aplicación

Para la página web de gestión del proyecto, se aplica este protocolo porque los sitios web que utilizan el protocolo HTTP estándar transmiten y reciben datos de forma no segura. Esto significa que es posible que alguien escuche a escondidas los datos que se transfieren entre el usuario y el servidor web. Y por tanto, podría provocar que hubiera opción de robo en los datos de los usuarios que estuvieran utilizando el servicio.

# Protocolo de Reserva / Entrada al Recinto

## 6.1. ¿Cómo puedo acceder al recinto?

Si un cliente quiere acceder al recinto de carga para recargar su vehículo deberá adquirir una reserva. Sin esta reserva no tendrá permitida la entrada.

## 6.2. ¿Dónde podemos obtener una reserva y que requisitos hemos de cumplir?

Primeramente para obtener reservas debemos estar registrados en la aplicación móvil una vez registrados debemos asegurarnos de tener introducida en la aplicación la matrícula del vehículo que queremos cargar y tener suficiente dinero en nuestra cartera para efectuar el pago.

Una vez hemos cumplido los requisitos especificados anteriormente tenemos dos opciones para efectuar nuestra reserva:

- Acercarnos a una electrolinera y pedirle a algún trabajador una reserva y proporcionarle los datos necesarios.
- Acceder a la página de reservas de la aplicación móvil y solicitar una reserva con los datos pertinentes de nuestra reserva.

(SIEMPRE SE RECOMIENDA RESERVAR DESDE LA APP)

## 6.3. ¿Qué datos debe proporcionar el cliente para la reserva?

Si el cliente efectúa la reserva desde su aplicación deberá proporcionar los siguientes datos:

- Estación donde desea recargar
- Hora de inicio
- Hora de fin
- Fecha (dia)



Si el cliente por algún motivo no puede acceder a su aplicación puede obtener una reserva contactando con algún trabajador.

Los datos que deberá proporcionarle al trabajador serán:

- Estacion
- Hora Inicio

- Hora de finalizacion
- Fecha
- DNI



El DNI es una forma de asegurarnos de que se ha hecho una reserva de un vehículo del cliente y no otros vehículos.

#### 6.4. ¿Qué debe hacer la aplicación/trabajador?

Una vez el cliente ha introducido todos los datos para su reserva la aplicación le podrá indicar 2 cosas:

- La reserva se ha efectuado correctamente.  
La aplicación procede a indicar la plaza al que deberá acceder a su hora de reserva.  
Si se ha hecho la reserva mediante un trabajador el trabajador le indicará que plaza se le ha reservado
- La reserva no se ha efectuado correctamente debido a algún error o por que no hay plazas libres a esa hora.  
(En un futuro la aplicación podría indicar en qué horarios podría estar libre alguna plaza).  
Si la reserva se ha efectuado mediante un trabajador, el trabajador puede indicar qué plazas están libres mirando la lista de reservas.

#### 6.5. ¿Cuándo se paga la reserva?

El pago se efectuará automáticamente, retirando el dinero necesario de la cartera que tenga el cliente en su aplicación, una vez la reserva se haya efectuado correctamente.

El precio de la carga estará determinado por el tiempo de reserva y el precio de la electricidad. (Grupo A1).

Si por algún motivo el cliente se retira de la plaza antes de terminar la reserva no se le devolverá el dinero.

Si el cliente anula la reserva con un margen de máximo 30 min antes de la reserva, se le devolverá el dinero de la reserva.

## 6.6. ¿Una vez tengo la reserva como acceder al recinto?

Cuando ya tenemos nuestra reserva hecha deberemos ir a la estación donde tenemos la reserva y acercarnos a la valla la cual nos detectará y analizará la matrícula.

Si la matrícula no está registrada en las reservas no nos dejará pasar.

Si está reservada se nos abrirá la valla y podremos acceder a la plaza que tengamos asignada.

## 6.7. Gestión de reservas internamente

Para las reservas habrá una conexión entre los cuatro grupos.

Para añadir reservas tanto la aplicación como la web deberán enviar a la API la información de la reserva.

### **Los datos que enviaremos a la API:**

Endpoint: /api/app/newReserva POST

```
{  
    "estacion": VGA1,  
    "hora_inicio": "10:20",  
    "hora_fin": "11:00",  
    "matricula": "XXXXXX",  
    "fecha": "20/03/2022"  
    "DNI": "xxxxxxxxX"  
}
```

Lo trato con un python

Mirar en base de datos si está libre.

La API deberá responder:

**OK:**

Endpoint /api/app/newReserva 200 OK

```
{
```

```

        "success": True,
        "plaza/cargador": 3,
    }
Error:
Endpoint /api/app/newReserva 400 ERROR
{
    "success": False,
    "siguienteHoraLibre": "12:00" // Opcional
}

```

Se deberá actualizar la lista de todas las reservas en la página WEB.

#### **GET de todas las reservas**

Se deberá actualizar la lista de matriculas de la cámara con las matriculas que estén en reserva ese día a esa hora

## 7. Protocolo de multas en una reserva

1. Multa por estacionamiento y uso ineficiente del punto de recarga.

Independientemente de si se trata de un coche de recarga eléctrica o no, siempre y cuando no se esté utilizando el punto de recarga, estar estacionado en esa plaza **conlleva multas**, ya que se trata de una **plaza reservada**.

Esta es una falta grave conforme lo impuesto por el Real Decreto Legislativo 6/2015, por el que se aprueba el texto refundido de la Ley sobre Tráfico, Circulación de Vehículos a Motor y Seguridad Vial. Por tanto, ante esta situación, los conductores que no sean poseedores de un vehículo electrificado (híbrido o 100% eléctrico), deben tener claro que la utilización de estas plazas es **exclusiva para la recarga** del vehículo eléctrico.

#### **Artículo 80. Tipos.**

1. Las infracciones leves serán sancionadas con multa de hasta 100 euros; las graves, con multa de 200 euros, y las muy graves, con multa de 500 euros.  
Tal como se especifica en el fragmento, el estacionamiento en un zona de carga de vehículos eléctricos comporta una falta grave, con lo cual llega hasta los 200 euros de multa, que puede ser reducida a la mitad por pronto pago.
2. Multa por cancelar una reserva a tiempo parcial.  
Si el cliente cancela la reserva en mitad de la reserva se le devolverá el pago pero con una multa en proporción al tiempo perdido.
3. Multa por negligencia de uso de la reserva.  
Si el cliente no cancela la reserva durante el tiempo debido y no efectúa su carga no tendrá derecho a ningún tipo de devolución.
4. Superar la reserva o carga execisiva.

Se considerará como estacionamiento indebido, por lo tanto una infracción grave.

5. Multa por incumplimiento de pago o estafa.  
Irse sin pagar de una electrolinera demanda una indemnización del servicio más el perjuicio provocado según la gravedad del suceso, además de comprometerse a un delito penal.  
Es la estación de servicio quien solicita la multa en cada caso.  
Los empleados de las gasolineras están obligados a denunciar en cada caso.

## 8. Protocolo de Login

### 8.1. Login en la web

El inicio de sesión en la página web únicamente está permitido a los trabajadores de la empresa Gesys.

Para hacer el inicio de sesión el usuario debe estar previamente registrado en la página. El registro se lleva a cabo por un Administrador de la web, los datos del registro se guardan en una base de datos a la cual se accederá en el momento del inicio de sesión para verificar que el inicio es correcto.

Una vez el registro ya está hecho, el usuario puede iniciar en cualquier momento introduciendo su usuario y contraseña en la página de login.

En caso de que el usuario introducido no esté registrado o la contraseña sea incorrecta, se enviará un aviso de que el usuario o contraseña son incorrectos.

Si los datos introducidos son correctos se permitirá el acceso a la web y dependiendo del cargo asignado al usuario se permitirá acceso a ciertos apartados de web.

### 8.2. Login en la app

El inicio de sesión en para cualquier usuario que tenga una cuenta registrada en la app de Gesys.

La cuenta puede ser creada en cualquier momento por cualquier usuario, para ello deberá introducir los datos que se piden en el momento de creación del usuario. Estos datos se guardarán en la base de datos de la empresa.

Una vez registrada la cuenta, el usuario podrá iniciar sesión en cualquier momento introduciendo su nombre de usuario y la contraseña previamente escogida por él.

En caso de que el usuario introducido no esté registrado o la contraseña sea incorrecta, se enviará un aviso de que el usuario o la contraseña son incorrectos.

Si los datos introducidos son correctos se permitirá el acceso a la app y hacer uso de lo que la app ofrece.

### 8.3. Seguridad en la contraseña

Al momento de crear la contraseña, se pedirá que esta tenga un mínimo de 8 caracteres y que contenga como mínimo 1 mayúscula, 1 minúscula y 1 número.

Con esto aumentaremos la seguridad para evitar que otros usuarios se conecten en la cuenta de otro usuario.

## WEB

Web implementada principalmente con una base de REACT usando plantillas de [Mantine](#) tanto para botones, tablas, iconos, etc.

Como lenguaje de programación principalmente se ha usado TypeScript, que resulta ser como JavaScript, pero con tipos, lo cual nos hacía más cómodo el trabajo.

Este sitio está pensado como página de administración sobre las diferentes centrales y con la posibilidad de ver la información de todo lo que está pasando en las diferentes estaciones y poder solucionar cualquier problema.

Enlace al repositorio con todo el código  
<https://github.com/PTIN2022/GeSyS-Front>

## LANDING

Página principal que podrá ver cualquier persona, con un enlace a la playstore para que se puedan bajar la app



## LOGIN

Página para iniciar sesión donde se introducirá el mail y contraseña del usuario



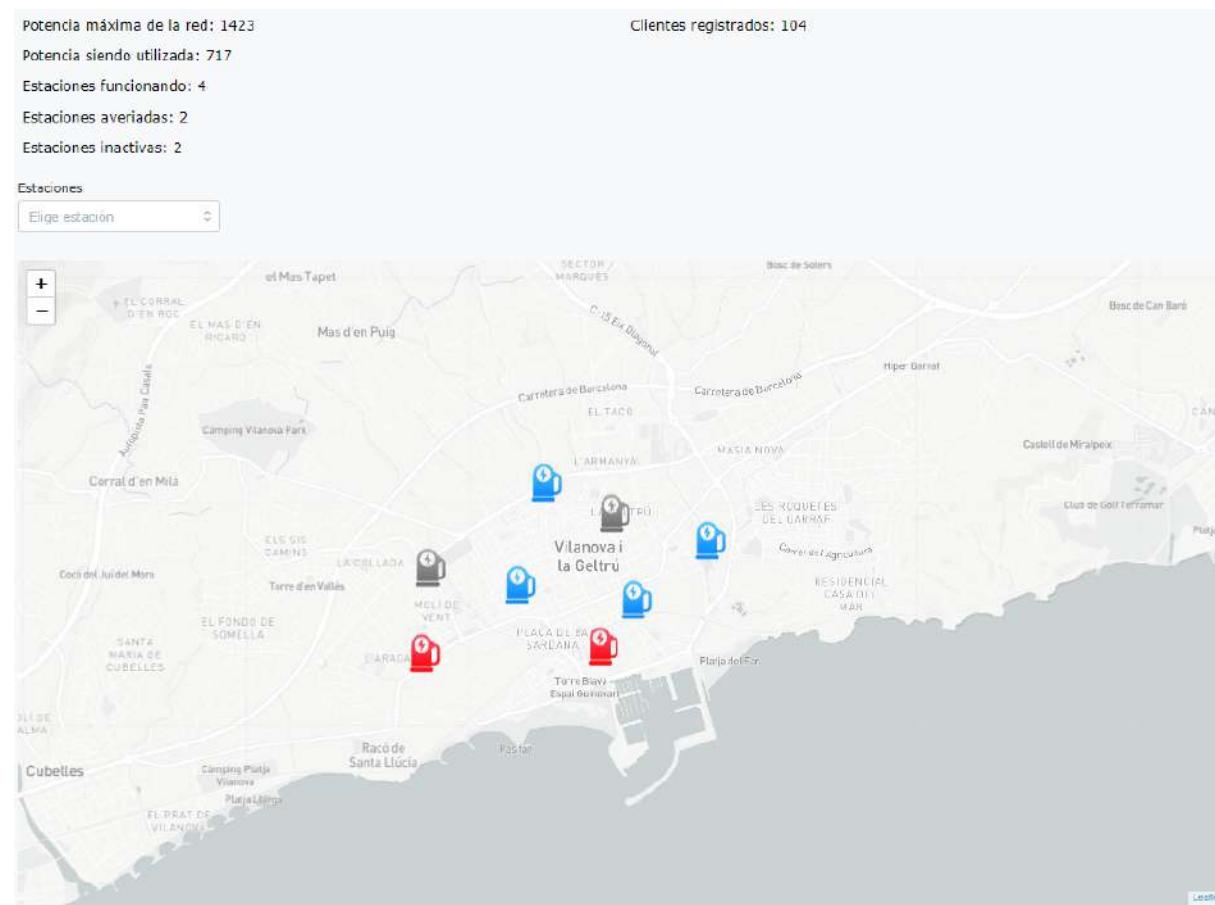
Si los dos campos son correctos se nos devolverá un token con el que haremos todas las peticiones CRUD.

## NAVBAR



Menú lateral para moverse por las diferentes páginas de manera fácil y cómoda, que en caso de ser administrador mostrará todas las siguientes opciones y que en caso de ser un trabajador se restringirá el acceso a algunas páginas como estaciones, estadísticas o promociones, dado que este no las debería de gestionar.

## PANTALLA DE INICIO



Pantalla pensada para entrar en un primer contacto con la web y recibir una información general del estado de las diferentes estaciones.

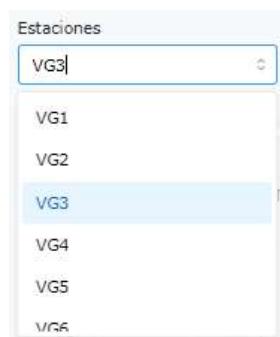
Tal y como se puede ver, en la parte superior se da una información general del conjunto de estaciones y también podemos ver como parte principal, un mapa con las diferentes estaciones situadas en él, con un código de color que nos informa rápidamente del estado de las diferentes estaciones.

También disponemos de una pequeña barra de búsqueda que nos centrará esa estación en el mapa.



Y para finalizar, en esta página tenemos la opción de clicar en cualquiera de las estaciones para desplegar un PopUp y de esta forma ver un poco de información más precisa de la misma.

La opción ver más nos llevará a la página de la estación en concreto, que más adelante mostraremos.

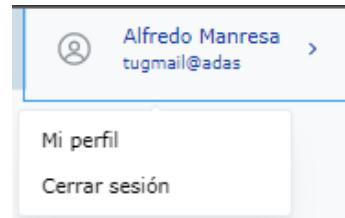


## Barra superior



En esta hay dos opciones principales, en la izquierda el logo de Gesys que nos llevará a la página de inicio.

La segunda opción se encuentra en el extremo derecho de esta y al clicar en ella nos permitirá tanto cerrar sesión como ver el perfil del usuario que está conectado.



## PERFIL

Página con el objetivo de mostrar la información de un usuario específico

Perfil Trabajador  
Mira la información personal a continuación

Alfredo Manresa

|                    |                  |
|--------------------|------------------|
| Nombre             | Apellido         |
| Alfredo            | Manresa          |
| Numero de Telefono | DNI              |
| 78254239           | 1111111111       |
| Correo electronico | Cargo de Empresa |
| tugmail@adas       | administrador    |

**Cambiar contraseña**

El usuario tendrá opción a cambiar la contraseña

Cambiar contraseña

Nueva contraseña \*

La contraseña debe tener un mínimo de 8 caracteres

Nueva contraseña

Confirmar contraseña \*

Confirmar contraseña

Guardar

## ESTACIONES

Primera opción que encontramos en el navbar lateral visto anteriormente.

En esta página se puede ver un listado de las diferentes estaciones, junto a su información principal, como su estado, nombre, dirección, consumo en KwH, Ocupación actual o el número de contacto del encargado.

### Estaciones

| Estado | Estacion | Dirección                  | Kwh     | Ocupación | Encargado    | ⋮ |
|--------|----------|----------------------------|---------|-----------|--------------|---|
| ●      | VG1      | Rambla de L'exposicio      | 150/230 | 1/32      | +34762487248 | ⋮ |
| ●      | VG2      | Rambla de A                | 110/220 | 15/32     | +34762854712 | ⋮ |
| ●      | VG3      | A veces                    | 70/120  | 8/32      | +34785123478 | ⋮ |
| ●      | VG4      | Rambla de Shrek            | 20/300  | 1/32      | +34745821523 | ⋮ |
| ●      | VG5      | Casoplon del coletas       | 15/30   | 14/32     | +34797458744 | ⋮ |
| ●      | VG6      | Casa de ibai               | 32/45   | 20/32     | +34768220011 | ⋮ |
| ●      | VG7      | Rambla de Redes Multimedia | 190/278 | 26/32     | +34798544552 | ⋮ |
| ●      | VG8      | Bar pepin                  | 130/200 | 32/32     | +34768855471 | ⋮ |

Los estados de las estaciones se muestran con un código de color

- Verde: Activa
- Amarilla: Dañada
- Rojo: Inactiva

Se puede apreciar que cada fila contiene también un pequeño menú que nos permitirá cambiar el estado de la estación y ver más información de esta misma.

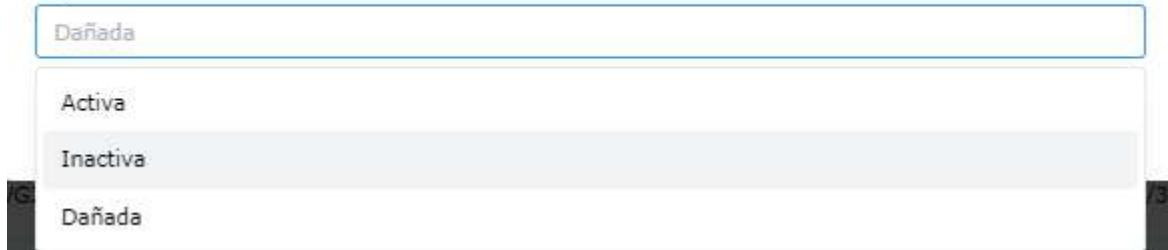


Si le damos a la opción de Editar nos mostrará el siguiente modal.

## Modifique el estado de la estación

X

Elije el nuevo estado



Si se elige la opción de ver más, igual que en el botón del PopUp del inicio, este nos llevará a la página de la estación en concreto.

## ESTACIÓN ESPECÍFICA

Página pensada para ver el estado de forma más precisa de una estación en concreto, en este caso de la estación 2.

Podemos ver en la parte superior un pequeño resumen de estado consumo y ocupación junto a la parte inferior que nos muestra un mapeo con cada una de las plazas y vemos cuáles están libres o no.



También se puede modificar el estado de la estación, si presionamos el botón del estado. En caso de clicar se desplegará el mismo modal que hemos visto en la página de la lista de estaciones que nos permite seleccionar el nuevo estado.



## TRABAJADORES

Se trata del segundo apartado en el navbar, pensado para listar los trabajadores de la empresa, enseñando su información principal. con la opción de aplicar algun filtro por los diferentes campos para una navegación entre los trabajadores mas sencilla.

## Trabajadores

| Añadir Trabajador |            |             |               |                      |
|-------------------|------------|-------------|---------------|----------------------|
| Filtrar por...    |            |             |               |                      |
| Foto              | Dni        | Nombre      | Rol           | Ultimo Acceso        |
| AL                | a111111111 | Alfredo     | administrador | 19/09/2021, 23:56:41 |
| MA                | b222222222 | Marc        | administrador | 17/09/2021, 23:56:41 |
| CI                | c333333333 | Cinta       | administrador | 16/09/2021, 23:56:41 |
| CA                | 53326212H  | Carolyn     | encargado     | 15/07/2021, 11:23:11 |
| CH                | 06788645D  | Christopher | encargado     | 03/05/2022, 13:46:44 |
| JA                | 03098531F  | James       | encargado     | 12/02/2022, 10:32:41 |
| MA                | 08484623D  | Marie       | encargado     | 22/10/2020, 04:28:42 |
| RO                | 94349990X  | Robert      | encargado     | 10/05/2021, 18:27:21 |
| JO                | 99094474P  | John        | encargado     | 05/08/2020, 01:04:29 |
| RE                | 72448156N  | Renee       | encargado     | 20/07/2020, 14:54:12 |
| PE                | 19293608J  | Peter       | encargado     | 02/05/2022, 03:37:08 |
| JO                | 91950538N  | John        | trabajador    | 30/12/2020, 04:45:34 |
| AM                | 92832519Y  | Amy         | trabajador    | 05/08/2021, 19:35:54 |

Con la opción de añadir Trabajador que solo se muestra a administradores y responsables, también podremos añadir nuevos trabajadores gracias al modal que se despliega al clicar el botón.

Introduzca los datos del nuevo trabajador

|   |   |
|---|---|
| Nombre<br><input type="text" value="Pedro"/>                    | Apellido<br><input type="text" value="Benito"/> |
| Numero de Telefono<br><input type="text" value="Telefono"/>     | DNI<br><input type="text" value="483.878.878"/> |
| Correo electronico<br><input type="text" value="@ @gmail.com"/> | Cargo<br><input type="text"/>                   |
| Contraseña<br><input type="text" value="1234Queso"/>            |   |
| Username<br><input type="text" value="qwerty3"/>                |   |
| <b>Guardar</b>  |   |

Igual que la mayoría de tablas, esta también contiene un menú lateral

- Ver más: lleva al perfil del trabajador
- Eliminar: Elimina el trabajador después de un mensaje de confirmación.



El filtro consta de un primer selector, que nos permite elegir el campo por el que queremos filtrar y una vez seleccionado este, aparecerá un segundo selector donde podremos aplicar el filtro

The screenshot shows a user interface for filtering data. On the left, there is a table with columns 'Foto' and 'Dni'. A search bar labeled 'Filtrar por...' contains the placeholder 'Nombre'. Another search bar labeled 'Filtrar por nombre:' also contains the placeholder 'Nombre'. Below these, a dropdown menu lists names: Nathan, Susan, Andrew, Christian, and Brittany. At the bottom of the interface, there is a row with the text 'AN 48840760S Angela trabajador 07/10/2020, 13:05:14'.

## RESERVAS

Sitio pensado para gestionar las reservas.

Tiene una estructura muy parecida a la de trabajadores, con la opción de añadir, filtrar y una tabla con la información principal de cada reserva

### Reservas

The screenshot shows a 'Reservas' (Reservations) page. At the top, there is a button 'Añadir Reserva' (Add Reservation). Below it, a section titled 'Elige que filtrar' (Choose what to filter) has a dropdown menu with the placeholder 'Pick one'. The main area displays a table of reserved slots with the following columns: ID, Reservante, Matricula, nºPlaza, Fecha, Fecha Fin, Tarifa, Coste[€], and an ellipsis column for more options. The data in the table is as follows:

| ID | Reservante | Matricula | nºPlaza | Fecha           | Fecha Fin       | Tarifa | Coste[€] |   |
|----|------------|-----------|---------|-----------------|-----------------|--------|----------|---|
| 1  | 30         | 5134FFJ   | 28      | 8/10/2020 12:32 | 22/7/2021 18:56 | 63.92  | 45.193   | ⋮ |
| 2  | 14         | QR1RQRR   | 55      | 11/6/2022 10:30 | 31/3/2021 8:48  | 28.309 | 50.754   | ⋮ |
| 3  | 24         | 5134FFJ   | 71      | 6/10/2021 14:55 | 23/8/2020 17:22 | 43.202 | 70.901   | ⋮ |
| 4  | 56         | 0Z0Z200   | 24      | 11/2/2021 14:9  | 6/3/2021 18:55  | 74.092 | 98.624   | ⋮ |
| 5  | 81         | 7C7C1B1   | 195     | 27/5/2022 8:6   | 6/12/2021 0:9   | 52.513 | 51.534   | ⋮ |
| 6  | 35         | Y39EXE8   | 140     | 31/7/2021 4:15  | 20/8/2021 11:34 | 90.751 | 39.794   | ⋮ |
| 7  | 1          | HLH00HO   | 170     | 29/10/2021 12:7 | 13/5/2022 4:38  | 29.612 | 68.469   | ⋮ |
| 8  | 19         | Y39EXE8   | 102     | 6/2/2021 23:9   | 5/8/2020 18:0   | 90.101 | 78.362   | ⋮ |

Al clicar en el botón de añadir, se abre un modal con los siguientes campos a rellenar.

El campo de estación muestra un desplegable con todas las estaciones no inactivas, y otros campos, como el precio final, se calcula de forma automática con base en la tarifa seleccionada

Introduzca los datos de la reserva

Estación

Inicio Reserva Fin Reserva

Fecha

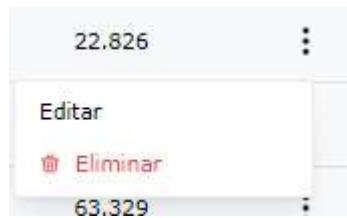
Matrícula DNI

Tarifa

**Guardar**

En este caso, las opciones que muestra el menú vuelven a ser:

- Editar: enlace a la página de reserva específica para poder editar cómodamente algunos campos como ahora la fecha.
- Eliminar: Esa reserva será eliminada de la base de datos.



## RESERVA [EDITAR]

Tiene dos estados principales, uno de solo lectura

Reserva: 10

Datos de la Reserva  
Mira y edita la información de la reserva 10

**Editar**

nºPlaza

235

Inicio Reserva Fin Reserva Matrícula

02:19 12:44 55E88PP

ID reservante Fecha

81 octubre 14, 2020

Y que al presionar el botón editar nos dejará modificar alguno de los campos de la reserva

Reserva: 5

**Datos de la Reserva**  
Mira y edita la información de la reserva 5

**Guardar Cambios**

nºPlaza  
195

Inicio Reserva      Fin Reserva      Matricula  
 08:06        00:09       7C7C1B1

ID reservante      Fecha  
      mayo 27, 2022

Una vez hechos los cambios deseados, si se presiona el botón de Guardar, estos cambios serán guardados en la base de datos.

## ESTADÍSTICAS

Página pensada de cara a responsables de un conjunto de estaciones, para ver los consumos que estas han ido teniendo y que pueda darse cuenta de horas con bajo consumo para generar promociones



La página cuenta con un primer filtro sobre estación, que permite seleccionar la estación sobre la que se quieren ver los datos, junto a un segundo filtro para seleccionar el periodo de tiempo a visualizar.

Como parte principal tenemos el gráfico, que en este caso está mostrando un consumo muy estable.

Como el consumo no se ha acercado a la potencia contratada, salta un aviso recomendando el posible uso de promociones.

## PROMOCIONES

Tabla para gestionar las promociones, visualizarlas, añadirlas o editarlas.

Se puede filtrar por promociones activas o inactivas

| Promociones             |                  |                 |              |            |   |
|-------------------------|------------------|-----------------|--------------|------------|---|
| Mostradas:              | Añadir Promoción | Mostrar Activas |              |            |   |
| Mostradas:<br>Inactivas |                  |                 |              |            |   |
| Id promoción            | Id estación      | Descuento [%]   | Fecha inicio | Fecha fin  |   |
| 1                       | 1                | 26              | 10/10/2020   | 21/07/2021 | ⋮ |
| 1                       | 3                | 26              | 10/10/2020   | 21/07/2021 | ⋮ |
| 1                       | 5                | 26              | 10/10/2020   | 21/07/2021 | ⋮ |
| 16                      | 1                | 10              | 29/06/2022   | 06/07/2022 | ⋮ |
| 9                       | 3                | 96              | 12/08/2020   | 01/05/2022 | ⋮ |
| 9                       | 4                | 96              | 12/08/2020   | 01/05/2022 | ⋮ |
| 3                       | 4                | 100             | 11/03/2021   | 04/08/2021 | ⋮ |

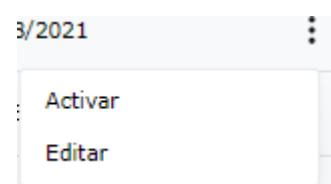
Al clicar en añadir promoción se despliega un modal parecido al de otras páginas

Introduzca los datos de la nueva promoción ×

|  |  |
|--|--|
| Estación   | Descuento [%]                                |
| <input type="text" value="Selecciona una estación"/>                     | <input type="text" value="0"/>               |
| Fecha Inicio   | Fecha Fin                                    |
| <input type="button" value="Escoger Fecha"/>                             | <input type="button" value="Escoger Fecha"/> |
| Descripción  |  |
| Promoción para incentivar el uso de la estación en esta hora de poco uso |  |
| <input type="button" value="Guardar"/>                                   |  |

En el menú de cada una de las promociones tenemos las opciones de:

- Activarla
- Editarla: se abre la página de la promoción específica, por defecto en modo de solo lectura y al darle al botón editar podremos cambiar alguno de sus campos.



## Promocion 3

**Editar**

Descuento [%]

100

Fecha Inicio: marzo 11, 2021

Fecha Fin: agosto 4, 2021

Descripción

HKWJMD8RVH36NCP6KYFPECFTPYZ1M3THWZF004A8Y66ETMDJSI1XWAQVUXTH5WC6IH6IV02FY01S8NQ487VZAIUA0SXL37I67  
8UMV860FGZUNQ0YVALEWCQ6JMUJ83FR1VXDRICLDO1MWR0YMU1NGSMAL45WK9SPNPLAH8NXB1UHCZRL8J92MQC07Z75  
1IHJ2HX7R8A7804HE09IHDNWBY8QJNTHJ3T1R49EPOPGXMY2T0OSERZFSRSP

**Eliminar Promoción**

Notar que desde esta página también se podrá eliminar la promoción.

En caso de darle a editar, podremos modificar la promoción y al guardar cambios esto se escribirá en la base de datos.

**Guardar**

Descuento [%]

100

Fecha Inicio: marzo 11, 2021

Fecha Fin: agosto 4, 2021

Descripción

HKWJMD8RVH36NCP6KYFPECFTPYZ1M3THWZF004A8Y66ETMDJSI1XWAQVUXTH5WC6IH6IV02FY01S8NQ487VZAIUA0SXL37I67  
8UMV860FGZUNQ0YVALEWCQ6JMUJ83FR1VXDRICLDO1MWR0YMU1NGSMAL45WK9SPNPLAH8NXB1UHCZRL8J92MQC07Z75  
1IHJ2HX7R8A7804HE09IHDNWBY8QJNTHJ3T1R49EPOPGXMY2T0OSERZFSRSP

**Eliminar Promoción**

## AVERÍAS

Página con el objetivo de mostrar las diferentes incidencias reportadas en las estaciones. La información de la estación, fecha, estado y descripción se muestra con una tabla.

## Averías

| Añadir Incidencia |                      |             |   |  |
|-------------------|----------------------|-------------|---|--|
| Estación          | Fecha                | Estado      | Descripción                                     | Editar Incidencia                          |
| VG2               | 29/06/2022, 00:00:00 | Pendiente   | Cargador no va                                  | <button>Incidencia</button> <span>⋮</span> |
| VG1               | 15/06/2022, 00:00:00 | Pendiente   | El coche de la DEMO no funciona                 | <button>Incidencia</button> <span>⋮</span> |
| VG1               | 29/06/2022, 00:00:00 | No Resuelto | El cargador no carga :)                         | <button>Incidencia</button> <span>⋮</span> |
| VG4               | 07/06/2022, 00:00:00 | Pendiente   | El cargador de la planta 2m plaza 1 no funciona | <button>Incidencia</button> <span>⋮</span> |

Hay la opción de añadir nuevas incidencias, y se hace desde el siguiente menú/modal

**Añadir Incidencia** ×

Estación

Fecha

Estado

Descripción

**Guardar**

Por cada incidencia podemos editarla pulsando el botón Incidencia, que abre el siguiente modal, que al darle a guardar actualizará la base de datos con los nuevos valores.

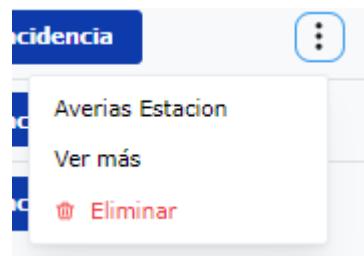
**Modifique los datos de la Incidencia** ×

Estación

Estado

Descripción

**Guardar**



También por cada incidencia, hay un menú con 3 opciones

- Averías estaciones: muestra todas las averías para la misma estación que la avería seleccionada

| Estación VG1 |          |                     |             |                                 |
|--------------|----------|---------------------|-------------|---------------------------------|
| id_averia    | Estacion | Fecha               | Estado      | Descripción                     |
| 2            | VG1      | 2022-06-15T00:00:00 | Pendiente   | El coche de la DEMO no funciona |
| 3            | VG1      | 2022-06-29T00:00:00 | No Resuelto | El cargador no carga :)         |

- Ver más: muestra información, no editable de la propia avería

### Averia 3

#### Informacion de la averia numero: 3

Consulta la informacion de la averia a continuacion...

|                         |             |
|-------------------------|-------------|
| ID Averia               | Estado      |
| 3                       | No Resuelto |
| Estacion                | Trabajador  |
| VG1                     |             |
| Descripción             |             |
| El cargador no carga :) |             |

- Eliminar: eliminar la avería en cuestión

## CLIENTES

Siguiendo con la estructura ya vista en otras páginas, clientes nos muestra la información básica, en forma de tabla, de los clientes registrados al sistema.

### Clientes

| Añadir Cliente    |          |           |                           |
|-------------------|----------|-----------|---------------------------|
| Elige que filtrar |          |           |                           |
| Pick one          |          |           |                           |
| Nombre            | Apellido | DNI       | Email                     |
| Mario             | Kochan   | XD12378N  | prueba@gmail.com          |
| Justin            | Taylor   | 70590495Z | efreeman@hotmail.com      |
| Wendy             | Hampton  | 26667884P | powellcasey@gmail.com     |
| David             | Weiss    | 48144355D | leroyking@gmail.com       |
| Michael           | Moore    | 72146484Q | brianmartinez@gmail.com   |
| Matthew           | Valencia | 14877578V | sarahmcdowell@hotmail.com |
| Brenda            | Marshall | 98378942J | williamnguyen@gmail.com   |
| Steve             | Brooks   | 59409430F | mossjasmin@hotmail.com    |

Seguimos contando con la opción de Añadir y de filtrar

Al clicar en añadir, siguiendo con lo visto anteriormente, se despliega un modal donde habrá que llenar la información del nuevo cliente.

Los campos de username y password, son generados de forma automática y por defecto, en un principio ambos consisten en el formato: nombre.apellido

Introduzca los datos del Cliente

|          |                        |
|----------|------------------------|
| Nombre   | Manolo                 |
| Apellido | Garcia                 |
| Teléfono | 123456789              |
| DNI      | 12345678J              |
| Email    | ManuelGarcia@gmail.com |

**Guardar**

Por lo que respecta al filtro, sigue con el mismo formato que los anteriores

| Elige que filtrar |          | Elemento a filtrar:  |
|-------------------|----------|----------------------|
| Email             |          | efree                |
| Nombre            | Apellido | efreeman@hotmail.com |
| Justin            | Taylor   | 70590495Z            |
|                   |          | efreeman@hotmail.com |

Editar Cliente

**Eliminar**

De igual forma que las otras tablas, sigue constando de un menú con varias opciones

- Editar: mostrará un modal como el de añadir clientes, pero en vez de con datos vacíos, poniendo los datos actuales del cliente
- Eliminar: Después de confirmar, se eliminará el cliente de la base de datos

El modal de editar cliente tiene la siguiente forma

## SOPORTE TÉCNICO

Página principal donde poder previsualizar diferentes tickets con los clientes junto a información básica de estos.

| Soporte Técnico     |                                   |           |                            |           |                         |  |
|---------------------|-----------------------------------|-----------|----------------------------|-----------|-------------------------|--|
| Filtrar por estado: |                                   |           |                            |           |                         |  |
| Ticket ID           | Asunto                            | Estado    | Fecha                      | Client ID |                         |  |
| 1                   | No me carga el coche              | Pendiente | 15 de junio de 2022, 13:00 | 30        | <a href="#">Ver mas</a> |  |
| 2                   | No me detecta el coche            | Pendiente | 16 de junio de 2022, 12:11 | 20        | <a href="#">Ver mas</a> |  |
| 3                   | Donde se da para cargar           | Pendiente | 18 de junio de 2022, 18:12 | 56        | <a href="#">Ver mas</a> |  |
| 4                   | Donde hay baños?                  | Pendiente | 12 de junio de 2022, 11:00 | 80        | <a href="#">Ver mas</a> |  |
| 5                   | Hay un raton muerto en la entrada | Pendiente | 11 de mayo de 2022, 20:11  | 95        | <a href="#">Ver mas</a> |  |
| 6                   | Gjsosok                           | Pendiente | 30 de junio de 2022, 7:58  | 207       | <a href="#">Ver mas</a> |  |
| 7                   | No funciona                       | Pendiente | 30 de junio de 2022, 8:28  | 207       | <a href="#">Ver mas</a> |  |

La página cuenta con un filtro que permite filtrar por estados.

| Filtrar por estado: |                        |
|---------------------|------------------------|
| Ticket ID           | Asunto                 |
| 1                   | No me carga el coche   |
| 2                   | No me detecta el coche |

- Todos
- Resuelto
- No resuelto
- En curso
- Pendiente

Cada ticket tiene la opción de ver más donde podremos ver el chat que le corresponde

## Ticket 3

Que boton es cojones

18/6/2022, 18:12:08 (Hace 12 días)

Estado:

Pendiente

Cliente id. 1

18 de junio de 2022, 18:15

Hay un boton verde

Cliente id. 56

18 de junio de 2022, 18:40

Me he equivocado, pensaba que hablaba con los de roomba

Enviar mensaje...

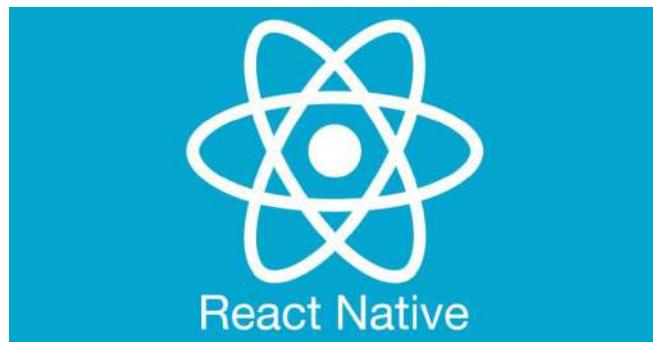
Enviar

# Aplicación móvil y base de datos

## Aplicación móvil

Tecnologías utilizadas para el desarrollo

Nuestra aplicación móvil está desarrollada sobre ReactNative, un framework JavaScript para crear aplicaciones reales nativas para iOS y Android, basado en la librería de JavaScript React para la creación de componentes visuales, cambiando el propósito de los mismos para, en lugar de ser ejecutados en navegador, correr directamente sobre las plataformas móviles nativas, en este caso iOS y Android. Es decir, en lugar de desarrollar una aplicación web híbrida o en HTML5, lo que obtienes al final como resultado es una aplicación real nativa.



En nuestro caso solo hemos implementado la aplicación para dispositivos con SO Android.

Para administrar la sesión hemos utilizado Redux, un administrador de sesiones creado específicamente para React Native. Este nos permite tener todos los valores de interés (así como información del usuario, de las estaciones, etc) centralizadas para poder acceder a ellas fácilmente sin necesidad de hacer llamadas constantes a la API.

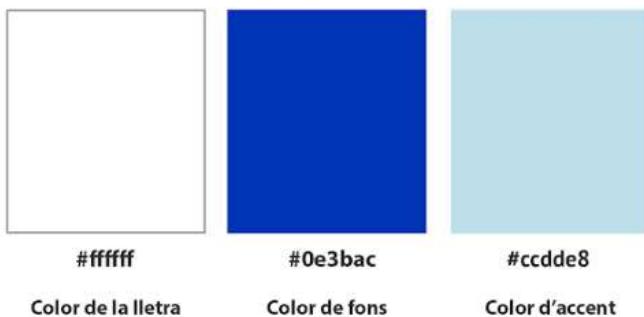


## Diseño de la aplicación

Como se acordó con los clientes, el diseño de la aplicación y la página web se basará en la siguiente paleta de colores:

### Full d'estil

#### Paleta de colors



#### Fonts

**Header 1 [Century Gothic, Bold, 14, #ffffff]**

**Header 2 [Century Gothic, Bold, 11, #ffffff]**

Paragraph [Century Gothic, 9, #ffffff]

  Lorem ipsum dolor sit amet, consectetur  
  adipiscing elit, sed do eiusmod tempor  
  incididunt ut labore et dolore magna aliqua.



37 %

63 %

Y el logo será el siguiente:



## Implementación de la aplicación

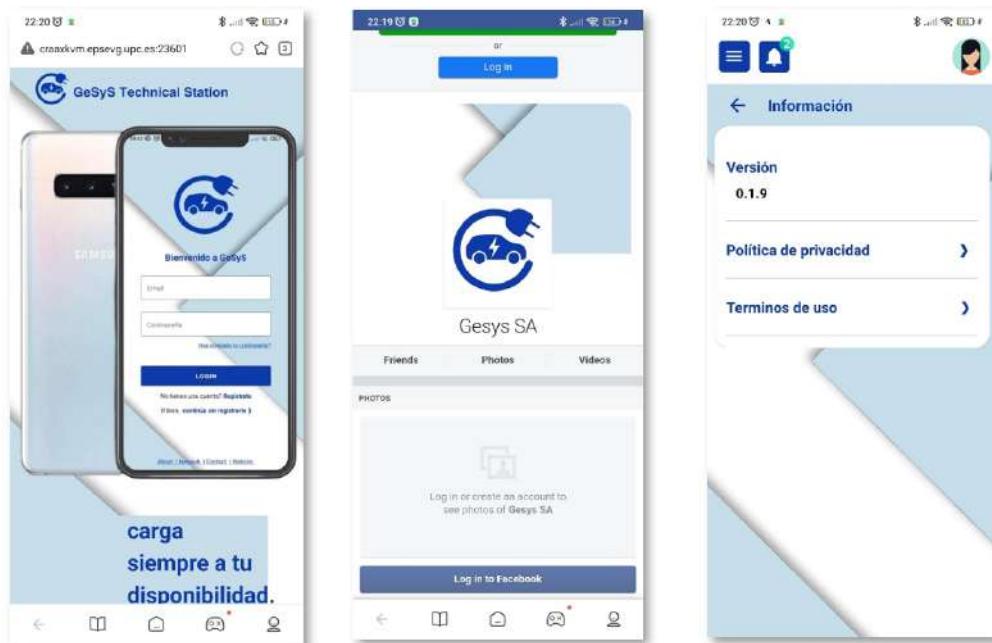
En cuanto a la aplicación perse, nos encontramos con la siguientes pantallas:

### Login

En la pantalla de LogIn nos podemos encontrar con la forma en la que el usuario iniciará sesión.

Como podemos observar, también nos encontramos con un botón para recuperar la contraseña, para crear una cuenta y para proceder sin iniciar sesión para poder visualizar las electrolineras.

A parte, podemos observar tres botones inferiores que redirigen al usuario a la información de la aplicación, la página web de GeSys y el perfil de Facebook de este.



## Contraseña olvidada

La pantalla de contraseña olvidada permite al usuario recuperar su contraseña para poder acceder a la aplicación en caso que la haya perdido. Esta pide al usuario el correo vinculado al usuario.

Esta pantalla no es funcional actualmente.



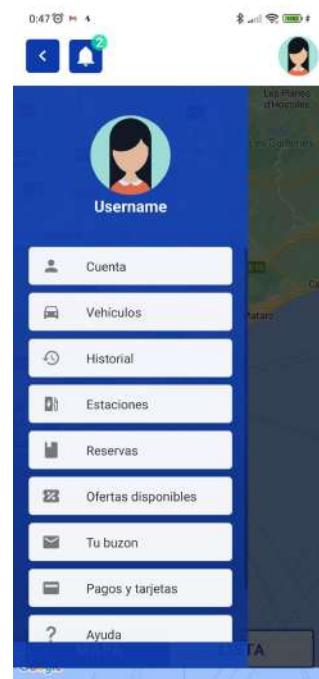
## Registro de usuario

La pantalla de registro de usuario permite a los clientes de GeSyS crear un usuario para poder gestionar sus reservas, vehículos, tickets, etc.



## SideBar

Una vez hayamos iniciado sesión (ya sea con un usuario existente o creando uno de nuevo) entramos en la aplicación como tal. Antes de presentar las diferentes pantallas que esta tiene, presentaremos la forma de navegar por esta. El usuario podrá navegar mediante una barra lateral.



## Mapa de estaciones

La primera pantalla con la que se encuentra el usuario al iniciar la aplicación es el mapa de estaciones.

Las estaciones que aparecen en el mapa se pueden clickear y el cliente puede acceder al detalle de la estación.



## Lista de estaciones

Para visualizar las estaciones el usuario también cuenta con un listado de las estaciones, el cual se puede filtrar tanto por promociones como por ubicación (km de distancia desde el usuario). En esta lista podremos ver los detalles de cada estación o realizar una reserva directamente.

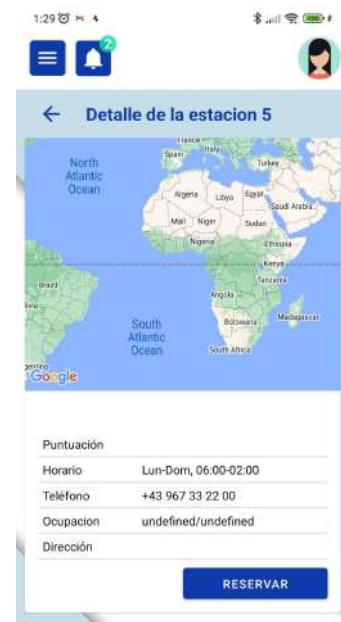


## Detalles de estación

Tanto si llegamos mediante el mapa como la lista, podremos visualizar los detalles de una estación.

Aquí podremos ver información como la ocupación, el horario o el teléfono de contacto.

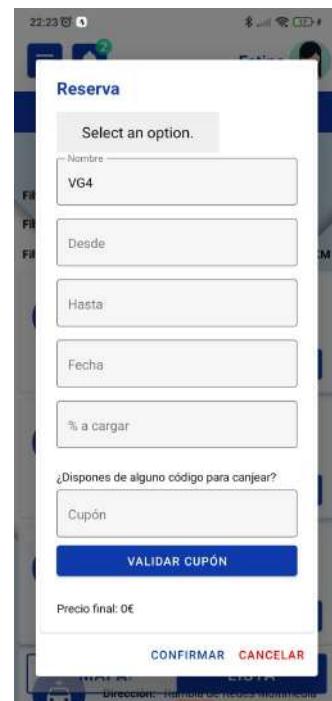
También podremos realizar la reserva igual que podríamos des de la pantalla de lista de estaciones.



## Formulario para realizar reserva

Éste formulario permitirá al cliente realizar una reserva.

Como se puede observar, este pide diferentes parámetros. Entre ellos la fecha/hora de entrada y salida, introducir un cupón si se tiene y el % de carga que quiere efectuar el usuario.



## Lista de cupones

En la lista de cupones el usuario podrá consultar cupones disponibles en las diferentes estaciones.

Así podrá elegir el que más se le convenga y se adapte al uso que quiere realizar.



## Listado de vehículos

En el listado de vehículos el usuario podrá ver listados los vehículos registrados a su nombre en el sistema.

A parte, como se puede observar, hay un botón que permite al usuario añadir un nuevo vehículo.



## Añadir un vehículo

En el formulario para añadir un nuevo vehículo el usuario deberá seleccionar el modelo del vehículo de un desplegable.

Una vez seleccionado el modelo los detalles técnicos de este (potencia de carga, capacidad de la batería, etc) ya se establecerán automáticamente.

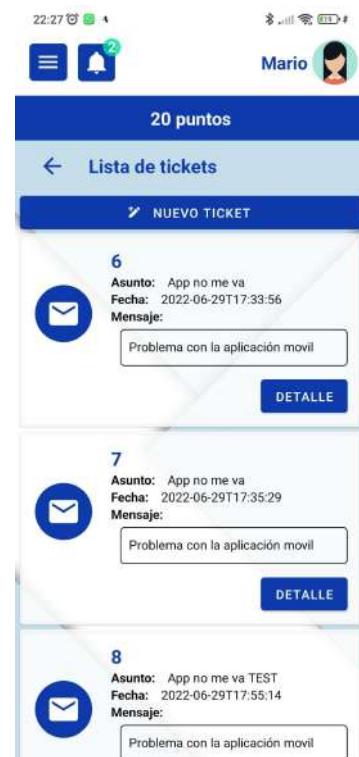
Una vez seleccionado el modelo, el usuario solo deberá indicar el modelo y matrícula del vehículo.



## Listado de tickets

En el listado de tickets el usuario podrá consultar todos los tickets que ha enviado a la empresa.

En esta misma pantalla podrá consultar los detalles de cualquiera de sus tickets (ya que cada ticket tiene un historial de mensajes) y podrá crear nuevos tickets.



## Detalles de ticket

Como hemos dicho, cada ticket tiene un historial de mensajes que el usuario ha podido tener con uno de los trabajadores de GeSys.

Este intercambio de mensajes es el que se muestra en los detalles del ticket.



## Creación de tickets

En esta pantalla el usuario podrá crear tickets para contactar con la empresa.

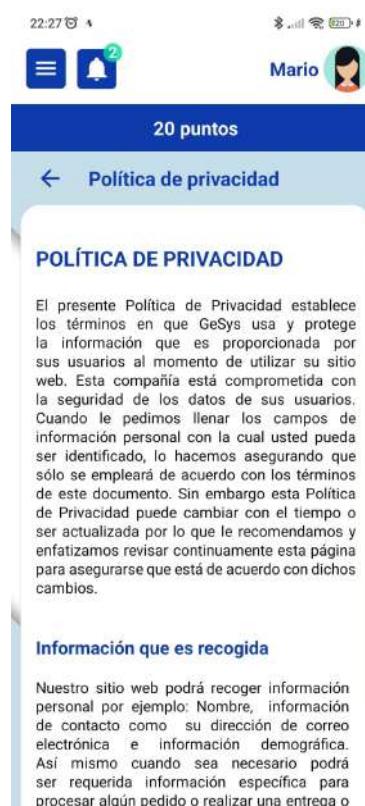
La creación de tickets es relativamente sencilla, simplemente requiere un motivo y una pequeña descripción.



## Política de privacidad

En esta pantalla el usuario puede consultar la política de privacidad de GeSys.

Esta política es la política general europea.



## Términos y condiciones de uso

En esta pantalla el usuario puede consultar los términos y condiciones de uso de GeSyS.

Estos son los términos y condiciones de uso generales europeos.

The screenshot shows a mobile application interface. At the top, there are icons for battery level, signal strength, and a profile picture of 'Mario'. Below this is a blue header bar with the text '20 puntos' and a back arrow labeled 'Términos de uso'. The main content area has a white background with a blue header 'Términos y Condiciones de Uso'. Underneath, a section titled 'INFORMACIÓN RELEVANTE' contains text about the necessity of accepting terms and conditions for product purchases. Another section below discusses password management and third-party purchases. At the bottom of the screen, there is a green button labeled 'AÑADIR SALDO'.

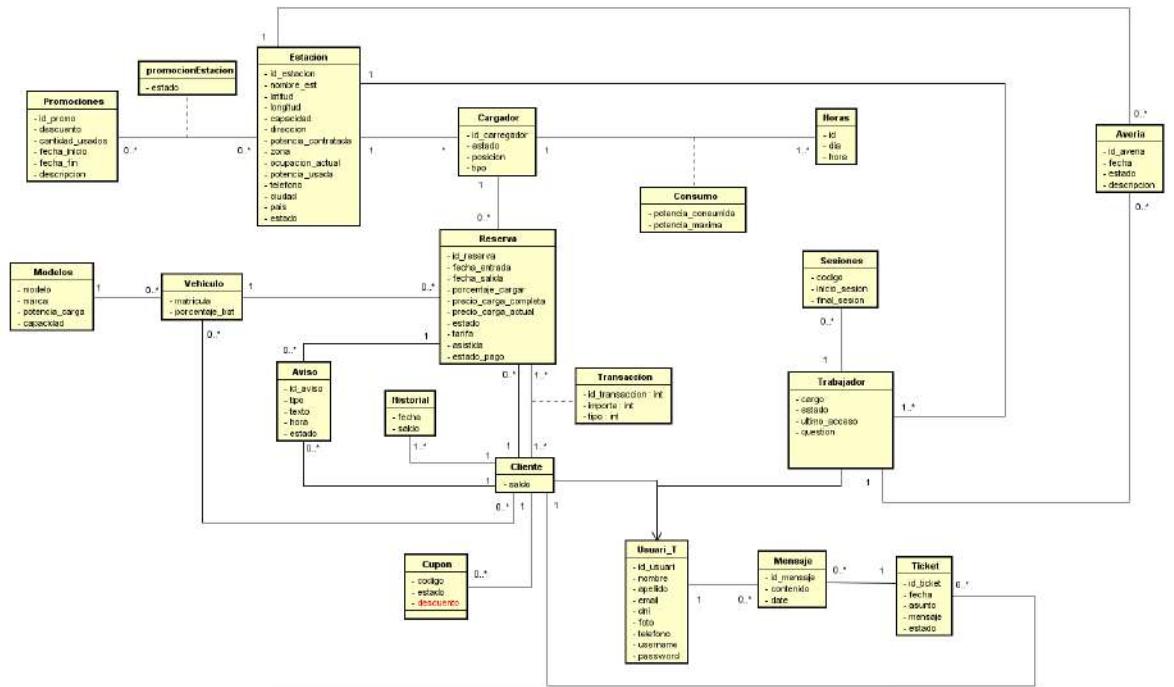
## Historial de usos

En esta pantalla el usuario puede consultar su historial de usos (reservas) con la fecha e importe.

The screenshot shows a mobile application interface. At the top, there are icons for battery level, signal strength, and a profile picture of 'Fatine'. Below this is a blue header bar with the text '16.35 puntos' and a back arrow. The main content area has a white background with a green header 'Historial'. A red bar at the bottom displays the date '30-5-2022, -16.35€'. Above this bar, there is a green button labeled 'AÑADIR SALDO'.

## Base de datos

El desarrollo de la Base de Datos comenzó a partir de un UML, creado con la aplicación Astah, que a lo largo del proyecto tuvo que ser cambiado dependiendo de las necesidades que se presentaban en los distintos grupos para el correcto funcionamiento de las herramientas que fueran desarrollando y también utilizando el feedback de los clientes con los cuales se mantuvieron reuniones a lo largo de todos los sprints. De esta forma se llegó al siguiente resultado como base de la Base de Datos:



1. UML

Tomando como referencia el UML anterior, para la creación de las tablas de la Base de Datos se utilizó SQLAlchemy, un Mapping-Tool que es utilizada por desarrolladores para crear bases de datos y manipular sus datos de manera sencilla. Lo que hace esta librería es permitir crear scripts para crear las tablas usando SQL de una manera más sencilla. De tal manera se crearon tablas para cada uno de los componentes del UML que pasaron a ser utilizados por la API.

Las tablas creadas fueron las siguientes:

```

1  from utils.db import db
2  from marshmallow_sqlalchemy import SQLAlchemyAutoSchema
3
4
5  class Averia(db.Model):
6
7      id_averia = db.Column(db.Integer, nullable=False, primary_key=True, autoincrement=True)
8      fecha = db.Column(db.DateTime, nullable=False)
9      estado = db.Column(db.String(30), nullable=False)
10     descripcion = db.Column(db.String(300))
11
12     id_trabajador = db.Column('id_trabajador', db.ForeignKey(
13         'trabajador.id_usuari'), nullable=False)
14     id_estacion = db.Column(db.Integer, db.ForeignKey(
15         'estacion.id_estacion'), nullable=False)
16
17     def __init__(self, fecha, estado, descripcion, id_trabajador, id_estacion):
18         self.fecha = fecha
19         self.estado = estado
20         self.descripcion = descripcion
21         self.id_trabajador = id_trabajador
22         self.id_estacion = id_estacion
23
24
25
26     class AveriaSchema(SQLAlchemyAutoSchema):
27         class Meta:
28             model = Averia
29

```

## 2. Tabla Averias

```

from utils.db import db
from marshmallow_sqlalchemy import SQLAlchemyAutoSchema


class Aviso(db.Model):
    id_aviso = db.Column(db.Integer, nullable=False, primary_key=True, autoincrement=True)
    tipo = db.Column(db.String(20), nullable=False)
    texto = db.Column(db.String(300), nullable=False)
    hora = db.Column(db.DateTime, nullable=False)

    id_reserva = db.Column(db.Integer, db.ForeignKey("reserva.id_reserva"), nullable=False)
    id_cliente = db.Column(db.Integer, db.ForeignKey("cliente.id_usuari"), nullable=False)

    def __init__(self, tipo, texto, hora, id_reserva, id_cliente): #need
        self.tipo = tipo
        self.texto = texto
        self.hora = hora
        self.id_reserva = id_reserva
        self.id_cliente = id_cliente

    class AvisoSchema(SQLAlchemyAutoSchema):
        # estacion= fields.Nested(EstacionSchema)
        class Meta:
            model = Aviso

```

## 3. Tabla Avisos

```

from utils.db import db
from marshmallow_sqlalchemy import SQLAlchemyAutoSchema
import models.reserva
import models.horas

class Cargador(db.Model):

    id_cargador = db.Column(db.Integer, nullable=False, primary_key=True, autoincrement=True)
    estado = db.Column(db.String(30), nullable=False)
    posicion = db.Column(db.Integer, nullable=False)
    tipo = db.Column(db.String(100), nullable=False)

    estacion_id = db.Column(db.Integer, db.ForeignKey("estacion.id_estacion"), nullable=False)
    reservas = db.relationship("Reserva", backref="cargador")
    #horass = db.relationship("Horas", backref="cargador")

    def __init__(self, estado, posicion, tipo, estacion_id):
        self.estado = estado
        self.posicion = posicion
        self.tipo = tipo
        self.estacion_id = estacion_id

class CargadorSchema(SQLAlchemyAutoSchema):
    # estacion= fields.Nested(EstacionSchema)
    class Meta:
        model = Cargador

```

#### 4. Tabla Cargador

```

from utils.db import db
from models.usuari_t import Usuari_t
from marshmallow_sqlalchemy import SQLAlchemyAutoSchema
import models.aviso
import models.reserva
import models.ticket
import models.cupon

class Cliente(db.Model):
    id_cliente = db.Column('id_usuari', db.ForeignKey('usuari_t.id_usuari'), nullable=False, primary_key=True)
    __table_args__ = (
        db.PrimaryKeyConstraint(id_cliente),
        {},
    )
    saldo = db.Column(db.FLOAT, nullable=False)

    avisos = db.relationship("Aviso", backref="cliente")
    reservas = db.relationship("Reserva", backref="cliente")
    ticket = db.relationship("Ticket", backref="cliente")
    cupones = db.relationship("Cupon", backref="cliente")

    def __init__(self, id_cliente, saldo):
        self.id_cliente = id_cliente
        self.saldo = saldo

class ClienteSchema(SQLAlchemyAutoSchema):
    class Meta:
        model = Cliente

```

#### 5. Tabla Clientes

```

def sessionadd(c98):
    c98 = Cliente(
        "Hannah",
        "Wallace",
        "hollandlinda@gmail.com",
        "57674118Y",
        "https://ui-avatars.com/api/?name=Hannah",
        12269483,
        "Wallace57674118",
        encrypt_password("Hannah57674118"),
    )
    db.session.add(c98)

```

## 5.1 Console Log tabla Clientes

```

from utils.db import db
from models.cargador import Cargador
from models.horas import Horas
from marshmallow_sqlalchemy import SQLAlchemyAutoSchema

class Consumo(db.Model):
    id_cargador = db.Column('id_cargador', db.ForeignKey('cargador.id_cargador'), nullable=False)
    id_horas = db.Column('id_horas', db.ForeignKey('horas.id'), nullable=False)
    __table_args__ = (
        db.PrimaryKeyConstraint(id_cargador, id_horas),
        {},
    )
    potencia_consumida = db.Column(db.Integer, nullable=False)
    potencia_maxima = db.Column(db.Integer, nullable=False)

    def __init__(self, id_cargador, id_horas, potencia_consumida, potencia_maxima):
        self.id_cargador = id_cargador
        self.id_horas = id_horas
        self.potencia_consumida = potencia_consumida
        self.potencia_maxima = potencia_maxima

class ConsumoSchema(SQLAlchemyAutoSchema):
    class Meta:
        model = Consumo

```

## 6. Tabla Consumo

### 6.1

```

from utils.db import db
from marshmallow_sqlalchemy import SQLAlchemyAutoSchema

class Cupon(db.Model):
    cupon = db.Column(db.String(20), nullable=False,
                      primary_key=True)
    id_cliente = db.Column(db.Integer, db.ForeignKey(
        "cliente.id_usuari"), nullable=False)
    estado = db.Column(db.String(30), nullable=False)

    def __init__(self, cupon, id_cliente, estado): # need
        self.cupon = cupon
        self.id_cliente = id_cliente
        self.estado = estado

class CuponSchema(SQLAlchemyAutoSchema):
    # estacion= fields.Nested(EstacionSchema)
    class Meta:
        model = Cupon

```

## 7. Tabla Cupones

```

from utils.db import db
import models.cargador # noqa: F401
import models.promociones # noqa: F401
import models.trabajador
import models.averia
from marshmallow_sqlalchemy import SQLAlchemyAutoSchema

class Estacion(db.Model):
    id_estacion = db.Column(db.Integer, nullable=False, primary_key=True, autoincrement=True)
    nombre_est = db.Column(db.String(20), nullable=False, unique=True) #unico
    latitud = db.Column(db.Integer, nullable=False)
    longitud = db.Column(db.Integer, nullable=False)
    capacidad = db.Column(db.Integer, nullable=False)
    direccion = db.Column(db.String(300), nullable=False)
    potencia_contratada = db.Column(db.Integer, nullable=False)
    zona = db.Column(db.String(300), nullable=False)
    ocupacion_actual = db.Column(db.Integer, nullable=False)
    potencia_usada = db.Column(db.Integer, nullable=False)
    telefono = db.Column(db.String(50), nullable=False)
    ciudad = db.Column(db.String(100), nullable=False)
    pais = db.Column(db.String(100), nullable=False)
    estado = db.Column(db.String(30), nullable=False)

    encargado = db.Column('id_trabajador', db.ForeignKey('trabajador.id_usuari'), nullable=True)

    def __init__(self, nombre_est, latitud, longitud, capacidad, direccion, potencia_contratada, zona, ocupacion_actual, potencia_usada, telefono, ciudad, pais, estado):
        self.nombre_est = nombre_est
        self.latitud = latitud
        self.longitud = longitud
        self.capacidad = capacidad
        self.direccion = direccion
        self.potencia_contratada = potencia_contratada
        self.zona = zona
        self.ocupacion_actual = ocupacion_actual
        self.potencia_usada = potencia_usada
        self.telefono = telefono
        self.ciudad = ciudad
        self.pais = pais
        self.estado = estado
        self.encargado = encargado

class EstacionSchema(SQLAlchemyAutoSchema):
    class Meta:
        model = Estacion

```

## 8. Tabla Estaciones

```

from utils.db import db
from marshmallow_sqlalchemy import SQLAlchemyAutoSchema

class Horas(db.Model):
    id = db.Column(db.Integer, nullable=False, primary_key=True, autoincrement=True)
    dia = db.Column(db.DateTime, nullable=False)
    hora = db.Column(db.DateTime, nullable=False)

    #id_cargador = db.Column(db.Integer, db.ForeignKey("cargador.id_cargador"), nullable=False)

    def __init__(self, dia, hora):  #,id_cargador
        self.dia = dia
        self.hora = hora
        #self.id_cargador = id_cargador

class HorasSchema(SQLAlchemyAutoSchema):
    # estacion= fields.Nested(EstacionSchema)
    class Meta:
        model = Horas

```

## .9. Tabla Horas

```

from utils.db import db
from marshmallow_sqlalchemy import SQLAlchemyAutoSchema

class Mensaje(db.Model):

    id_mensaje = db.Column(db.Integer, nullable=False, primary_key=True, autoincrement=True)
    contenido = db.Column(db.String(300), nullable=False)
    date = db.Column(db.DateTime, nullable=False)
    id_usuari = db.Column(db.Integer, db.ForeignKey('usuari_t.id_usuari'), nullable=False)

    id_ticket = db.Column(db.Integer, db.ForeignKey(
        'ticket.id_ticket'), nullable=False)

    def __init__(self, contenido, date, id_usuari, id_ticket):
        self.contenido = contenido
        self.date = date
        self.id_usuari = id_usuari
        self.id_ticket = id_ticket

class MensajeSchema(SQLAlchemyAutoSchema):
    class Meta:
        model = Mensaje

```

## 10. Tabla Mensajes

```

from utils.db import db
from marshmallow_sqlalchemy import SQLAlchemyAutoSchema
from sqlalchemy.dialects.mysql import FLOAT
import models.vehiculo

class Modelo(db.Model):

    modelo = db.Column(db.String(100), nullable=False, primary_key=True)
    marca = db.Column(db.String(30), nullable=False)

    # potencia_carga --> si es true = Carga Rapida, False = Normal
    potencia_carga = db.Column(db.Boolean, nullable=True)
    capacidad = db.Column(db.FLOAT, nullable=False)

    vehiculo = db.relationship("Vehiculo", backref="modelo")

    def __init__(self, modelo, marca, potencia_carga, capacidad):
        self.modelo = modelo
        self.marca = marca
        self.potencia_carga = potencia_carga
        self.capacidad = capacidad

class ModeloSchema(SQLAlchemyAutoSchema):
    class Meta:
        model = Modelo

```

## 11. Tabla Modelos

```

from utils.db import db
from marshmallow_sqlalchemy import SQLAlchemyAutoSchema

class Promociones(db.Model):

    id_promo = db.Column(db.Integer, nullable=False, primary_key=True, autoincrement=True)
    descuento = db.Column(db.Integer, nullable=False)
    cantidad_usados = db.Column(db.Integer, nullable=False)
    fecha_inicio = db.Column(db.DateTime, nullable=False)
    fecha_fin = db.Column(db.DateTime, nullable=False)
    estado = db.Column(db.String(30), nullable=False)
    descripcion = db.Column(db.String(300), nullable=False)

    def __init__(self, descuento, cantidad_usados, fecha_inicio, fecha_fin, estado, descripcion):
        self.descuento = descuento
        self.cantidad_usados = cantidad_usados
        self.fecha_inicio = fecha_inicio
        self.fecha_fin = fecha_fin
        self.estado = estado
        self.descripcion = descripcion

class PromocionesSchema(SQLAlchemyAutoSchema):
    class Meta:
        model = Promociones

```

## .12. Tabla Promociones

```

from utils.db import db
from marshmallow_sqlalchemy import SQLAlchemyAutoSchema
from sqlalchemy.dialects.mysql import FLOAT
import models.aviso

class Reserva(db.Model):
    id_reserva = db.Column(db.Integer, nullable=False, primary_key=True, autoincrement=True)
    fecha_entrada = db.Column(db.DateTime, nullable=False)
    fecha_salida = db.Column(db.DateTime, nullable=False)
    proctenaje_carga = db.Column(db.Integer, nullable=False)
    precio_carga_completa = db.Column(db.FLOAT, nullable=False)
    precio_carga_actual = db.Column(db.FLOAT, nullable=False)
    estacionamiento = db.Column(db.Boolean, nullable=True)
    tarifa = db.Column(db.FLOAT, nullable=False)
    asistida = db.Column(db.Boolean, nullable=True)
    estado_pago = db.Column(db.Boolean, nullable=True)

    id_cargador = db.Column(db.Integer, db.ForeignKey("cargador.id_cargador"), nullable=False)
    id_vehiculo = db.Column(db.Integer, db.ForeignKey(
        "vehiculo.matricula"), nullable=False)
    id_cliente = db.Column(db.Integer, db.ForeignKey("cliente.id_usuario"), nullable=False)

    avisos = db.relationship("Aviso", backref="reserva")

    def __init__(self, fecha_entrada, fecha_salida, proctenaje_carga, precio_carga_completa, precio_carga_actual, estacionamiento, tarifa, asistida, estado_pago, id_cargador, id_vehiculo, id_cliente):
        self.fecha_entrada = fecha_entrada
        self.fecha_salida = fecha_salida
        self.proctenaje_carga = proctenaje_carga
        self.precio_carga_completa = precio_carga_completa
        self.precio_carga_actual = precio_carga_actual
        self.estacionamiento = estacionamiento
        self.tarifa = tarifa
        self.asistida = asistida
        self.estado_pago = estado_pago
        self.id_cargador = id_cargador
        self.id_vehiculo = id_vehiculo
        self.id_cliente = id_cliente

class ReservaSchema(SQLAlchemyAutoSchema):
    # estacionamiento fields nested (estacionamientoSchema)
    class Meta:
        model = Reserva

```

### .13. Tabla Reservas

```

from utils.db import db
from marshmallow_sqlalchemy import SQLAlchemyAutoSchema

class Sesiones(db.Model):
    codigo = db.Column(db.Integer, nullable=False, primary_key=True, autoincrement=True)
    inicio_sesion = db.Column(db.DateTime, nullable=False)
    final_sesion = db.Column(db.DateTime, nullable=False)

    id_trabajador = db.Column('id_trabajador', db.ForeignKey(
        'trabajador.id_usuario'), nullable=False)

    def __init__(self, inicio_sesion, final_sesion, id_trabajador):
        self.inicio_sesion = inicio_sesion
        self.final_sesion = final_sesion
        self.id_trabajador = id_trabajador

class SesionesSchema(SQLAlchemyAutoSchema):
    class Meta:
        model = Sesiones

```

### 14. Tabla Sesiones

```

from utils.db import db
from marshmallow_sqlalchemy import SQLAlchemyAutoSchema
import models.mensaje

class Ticket(db.Model):

    id_ticket = db.Column(db.Integer, nullable=False, primary_key=True, autoincrement=True)
    fecha = db.Column(db.DateTime, nullable=False)
    asunto = db.Column(db.String(30), nullable=False)
    estado = db.Column(db.String(30), nullable=False)
    mensaje = db.Column(db.String(300), nullable=False)

    id_cliente = db.Column('id_cliente', db.ForeignKey('cliente.id_usuari'), nullable=False)
    mensajes = db.relationship("Mensaje", backref="ticket")

    def __init__(self, fecha, asunto, mensaje,estado, id_cliente):
        self.fecha = fecha
        self.asunto = asunto
        self.mensaje = mensaje
        self.estado = estado
        self.id_cliente = id_cliente

class TicketSchema(SQLAlchemyAutoSchema):
    class Meta:
        model = Ticket

```

## 15. Tabla Tickets

```

from utils.db import db
from marshmallow_sqlalchemy import SQLAlchemyAutoSchema
from models.usuari_t import Usuari_t
import models.averia
import models.sesiones

class Trabajador(db.Model):
    id_trabajador = db.Column('id_usuari', db.ForeignKey('usuari_t.id_usuari'), nullable=False, primary_key=True)
    __table_args__ = (
        db.PrimaryKeyConstraint(id_trabajador),
        {}
    )
    cargo = db.Column(db.String(20), nullable=False)
    estado = db.Column(db.String(30), nullable=False)
    ultimo_acceso = db.Column(db.DateTime, nullable=False)
    question = db.Column(db.String(300), nullable=False)

    averia = db.relationship("Averia", backref="trabajador")
    sesion = db.relationship("Sesiones", backref="trabajador")

    id_estacion = db.Column(db.Integer, db.ForeignKey(
        'estacion.id_estacion'), nullable=False)

    def __init__(self, id_trabajador, cargo, estado,ultimo_acceso, question, id_estacion):
        self.id_trabajador = id_trabajador
        self.cargo = cargo
        self.estado = estado
        self.ultimo_acceso = ultimo_acceso
        self.question = question
        self.id_estacion = id_estacion

class TrabajadorSchema(SQLAlchemyAutoSchema):
    class Meta:
        model = Trabajador

```

## .16. Tabla Trabajadores

```

from utils.db import db
from marshmallow_sqlalchemy import SQLAlchemyAutoSchema
from sqlalchemy.dialects.mysql import FLOAT
import models.mensaje

class Usuari_t(db.Model):

    id_usuari = db.Column(db.Integer, nullable=False, primary_key=True, autoincrement=True)
    nombre = db.Column(db.String(30), nullable=False)
    apellido = db.Column(db.String(30), nullable=False)
    email = db.Column(db.String(30), nullable=False)
    dni = db.Column(db.String(15), nullable=False)
    foto = db.Column(db.String(15), nullable=False)
    telefono = db.Column(db.Integer, nullable=False)
    username = db.Column(db.String(30), nullable=False)
    password = db.Column(db.String(30), nullable=False)

    mensajes = db.relationship("Mensaje", backref="usuari_t")

def __init__(self, nombre, apellido, email, dni, foto, telefono, username, password):
    self.nombre = nombre
    self.apellido = apellido
    self.email = email
    self.dni = dni
    self.foto = foto
    self.telefono = telefono
    self.username = username
    self.password = password

class Usuari_tSchema(SQLAlchemyAutoSchema):
    # estacion= fields.Nested(EstacionSchema)
    class Meta:
        model = Usuari_t

```

17. Tabla Usuarios

```

from utils.db import db
from marshmallow_sqlalchemy import SQLAlchemyAutoSchema
import models.reserva

class Vehiculo(db.Model):
    matricula = db.Column(db.String(25), nullable=False, primary_key=True)
    procentaje_bat = db.Column(db.Integer, nullable=False)

    reservas = db.relationship("Reserva", backref="vehiculo")

    modelos = db.Column(db.String(100), db.ForeignKey(
        "modelo.modelo"), nullable=False)

    def __init__(self, matricula, procentaje_bat, modelos):
        self.matricula = matricula
        self.procentaje_bat = procentaje_bat
        self.modelos = modelos

class VehiculoSchema(SQLAlchemyAutoSchema):
    # estacion= fields.Nested(EstacionSchema)
    class Meta:
        model = Vehiculo

```

## 18. Tabla Vehículos

Para llenar la Base de Datos se hizo la creación de un faker para poder agregar datos (algunos aleatorios) para las tablas. El script se hizo para que se iniciara cuando se generará también las tablas de la siguiente manera:

Se configura el script para que pase a SQL:

```

from faker import Faker
fake = Faker('es_ES')

def init_db():
    db.init_app(app)
    with app.app_context():
        db.create_all()

app = Flask(__name__)

app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///test.db" # TODO: Pass to mysql
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False # TODO: review
app.config["TESTING"] = False

app.register_blueprint(estaciones)
app.register_blueprint(trabajador)

```

## .19. Configuración Script app

Los datos de las estaciones fueron agregadas a mano puesto que son fijos y ya estaban establecidos desde los sprint iniciales:

```

def init_db():
    db.init_app(app)
    with app.app_context():
        db.create_all()

    app = Flask(__name__)

    app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///test.db" # TODO: Pass to mysql
    app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False # TODO: review
    app.config["TESTING"] = False

    app.register_blueprint(estacion)
    app.register_blueprint(trabajador)

    if app.config["TESTING"] is False:
        if os.path.exists("./test.db"):
            os.remove("./test.db")

    init_db()
    with app.app_context():
        #####Estaciones#####
        estados = ['activas', 'averia', 'desactivas']
        e1 = Estacion("VG1", 41.217606, 1.727072, 32, "Rambla de L'exposicio", 20, "Zona industrial", 1,
                      150, '+34762487248', "Vilanova i la geltru", "Espana", random.choice(estados)) # , t.id_trabajador
        db.session.add(e1)
        e2 = Estacion("VG2", 41.221002, 1.730369, 32, "Rambla de A", 20, "Zona mayonesa", 15, 130,
                      "+34762854712", "Vilanova i la geltru", "Espana", random.choice(estados)) # , t.id_trabajador
        db.session.add(e2)
        e3 = Estacion("VG3", 41.225431, 1.7337627, 32, "A veces", 20, "Zona M de motomami", 8, 130,
                      "+34785123478", "Vilanova i la geltru", "Espana", random.choice(estados)) # , t.id_trabajador
        db.session.add(e3)
        e4 = Estacion("VG4", 41.2277420, 1.728166, 32, "Rambla de Shrek", 20, "Zona memes de baki", 1, 130,
                      "+34745821523", "Vilanova i la geltru", "Espana", random.choice(estados)) # , t.id_trabajador
        db.session.add(e4)
        e5 = Estacion("VG5", 41.229674, 1.721478, 32, "Casopion del coletas", 20, "Zona SEAX", 14, 130,
                      "+3479758744", "Vilanova i la geltru", "Espana", random.choice(estados)) # , t.id_trabajador
        db.session.add(e5)
        e6 = Estacion("VG6", 41.222119, 1.718915, 32, "Casa de Ibai", 20, "Zona el bicho", 20, 130,
                      "+34768220011", "Vilanova i la geltru", "Espana", random.choice(estados)) # , t.id_trabajador
        db.session.add(e6)
        e7 = Estacion("VG7", 41.223434, 1.710113, 32, "Rambla de Redes Multimedia", 20, "Zona XAMJ", 26,
                      130, "+34798544552", "Vilanova i la geltru", "Espana", random.choice(estados)) # , t.id_trabajador
        db.session.add(e7)
        e8 = Estacion("VG8", 41.217122, 1.709477, 32, "Bae pepin", 20, "Zona vip", 32, 130,
                      "+34768855471", "Vilanova i la geltru", "Espana", random.choice(estados)) # , t.id_trabajador
        db.session.add(e8)
        db.session.commit()
        estacioness = [e1,e2,e3,e4,e5,e6,e7,e8]
    
```

```

e1 = Estacion( # , t.id_trabajador
    'VG1',
    41.217606,
    1.727072,
    32,
    "Rambla de L'exposicio",
    230,
    'Zona industrial',
    1,
    150,
    '+34762487248',
    'Vilanova i la geltru',
    'Espa\xc3\xb3la',
    'Activa'
)
db.session.add(e1)
    
```

## .20. Introducción de Estaciones

Para la generación de cliente se utilizó la biblioteca de faker que nos permite generar nombres aleatorios, números de teléfono, ocupación, etc. También se generó DNI's:

```

#### CLIENTE ####
usuarios = []
clientes = []
for i in range(100):
    l = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L',
         'M', 'N', 'P', 'Q', 'R', 'S', 'T', 'V', 'W', 'X', 'Y', 'Z']
    num = '{:08}'.format(random.randrange(1, 10**8))
    dni = num + random.choice(l)

    nombre = fake.first_name()
    apellido = fake.last_name()
    email = fake.free_email()

    foto = fake.name()
    telefono = '{:09}'.format(random.randrange(1, 10**8))
    username = nombre + num
    password = apellido + num

    #### USUARIO ####
    usuario_te = Usuario_t(nombre, apellido, email, dni, foto, telefono, username, password)
    db.session.add(usuario_te)
    usuarios.append(usuario_te)

    db.session.commit()

    #### CLIENTES ####
    saldo = round(random.uniform(1.0, 100.0), 3)
    ce = Cliente(usuario_te.id_usuari, saldo)
    db.session.add(ce)
    clientes.append(ce)

    db.session.commit()

```

```

c100 = Cliente(
    "Mario",
    "Kochan",
    "prueba@gmail.com",
    "XD12378N",
    "https://ui-avatars.com/api/?name=pato",
    341341231,
    "mariuski",
    encrypt_password("1"),
)
db.session.add(c100)

```

## .21. Creación de Clientes

Para la generación de trabajadores se utilizó el mismo método que para Clientes pero esta vez agregando algunos parámetros extra propios del Trabajador:

```

#### TRABAJADORES ####
trabajadores = []
for i in range(100):
    l = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L',
         'M', 'N', 'P', 'Q', 'R', 'S', 'T', 'V', 'W', 'X', 'Y', 'Z']
    num = '{:08}'.format(random.randrange(1, 10**8))
    dni = num + random.choice(l)

    nombre = fake.first_name()
    apellido = fake.last_name()
    email = fake.free_email()

    foto = fake.name()
    telefono = '{:09}'.format(random.randrange(1, 10**8))
    username = nombre + num
    password = apellido + num

    #### USUARIO ####
    usuario_te = Usuario_t(nombre, apellido, email, dni, foto, telefono, username, password)
    db.session.add(usuario_te)
    usuarios.append(usuario_te)

    db.session.commit()

    #### TRABAJADOR ####
    cargos = ['administrador', 'encargado', 'trabajador']
    estados = ['Activo', 'Inactivo']
    cargo = random.choice(cargos)
    estado = random.choice(estados)
    estacion = random.choice(estaciones)
    ultimo_acceso = fake.date_time_between(start_date="-2y", end_date="now")

    tr = Trabajador(usuario_te.id_usuari, cargo, estado, ultimo_acceso, "Amigo de la infancia?", estacion.id_estacion)

    db.session.add(tr)
    trabajadores.append(tr)

    db.session.commit()

```

```

# ### Trabajador
t100 = Trabajador(
    'Alfredo',
    'Manresa',
    'tugmail@adas',
    'a111111111',
    'asda',
    78254239,
    'alfredo',
    encrypt_password('1'),
    'administrador',
    'Activo',
    datetime.fromisoformat('2021-09-19 23:56:41'),
    'Amigo de la infancia?',
    1
)

```

## .22. Creacion de Trabajadores

Las promociones se crearon utilizando un número random de descuento y posteriormente agregandola a una de las estaciones disponibles que hemos guardado anteriormente:

```

promociones = []
for i in range(10):
    descuento = fake.random_int(min=0, max=100)
    cantidad_usados = fake.random_int(min=0, max=300)
    fecha_inicio = fake.date_time_between(start_date="-2y", end_date="now")
    fecha_fin = fake.date_time_between(start_date="-2y", end_date="now")
    e = ['activa', 'desactiva']
    estado = random.choice(e)
    descripcion = fake.text()

    p = Promociones(descuento, cantidad_usados, fecha_inicio, fecha_fin, estado, descripcion)
    db.session.add(p)
    promociones.append(p)
db.session.commit()

#omocionEstacion ####
promocionesEstaciones = []
for i in range(5):
    estacion = random.choice(estaciones).id_estacion
    promo = random.choice(promociones).id_promo
    for j in range(len(promocionesEstaciones)):
        if promocionesEstaciones[j].id_estacion == estacion and promocionesEstaciones[j].id_promo == promo:
            i=i-1
            break
    p = PromocionEstacion(estacion, promo)
    db.session.add(p)
    promocionesEstaciones.append(p)

db.session.commit()

p0 = Promociones(
    16,
    0,
    datetime.fromisoformat('2020-10-10 20:19:22'),
    datetime.fromisoformat('2021-07-21 09:34:32'),
    "IKTGB93DLXBSWR63NUJHMG0J4INQWBBGPJLJDNHP7SKJ66I1BLB5C2R2SJJJSX9MCLVWH6H8KK8RW1H52E90M8VQ9VH9PBBNCSC217H06JE2TZJ0N1JU"
)
db.session.add(p0)

r1 = PromocionEstacion(8, 2, "inactiva")
r13 = PromocionEstacion(8, 3, "inactiva")
r22 = PromocionEstacion(8, 5, "activa")
r0 = PromocionEstacion(8, 14, "inactiva")
r7 = PromocionEstacion(8, 10, "inactiva")
r16 = PromocionEstacion(8, 13, "inactiva")

```

## 23. Creación de Promociones

Para los cargadores se crearon y posteriormente se agregaron a una estación aleatoria guardada en la lista de estaciones que tenemos:

```
####Cargadores#####
cargadores = []
for i in estacioness:
    for j in range(32):
        estadoss = ["ocupado","libre"]
        tiposs = ["Carga Normal","Carga Rápida"]
        estado = random.choice(estadoss)
        posicion = j
        tipo = random.choice(tiposs)
        sta = i.id_estacion

        carg = Cargador(estado,posicion,tipo,sta)
        db.session.add(carg)
        cargadores.append(carg)

db.session.commit()

carg10 = Cargador("ocupado", 0, "Carga Normal", 1)
db.session.add(carg10)
carg11 = Cargador("ocupado", 1, "Carga Rápida", 1)
db.session.add(carg11)
carg12 = Cargador("libre", 2, "Carga Rápida", 1)
db.session.add(carg12)
carg13 = Cargador("ocupado", 3, "Carga Rápida", 1)
db.session.add(carg13)
carg14 = Cargador("ocupado", 4, "Carga Normal", 1)
db.session.add(carg14)
carg15 = Cargador("ocupado", 5, "Carga Rápida", 1)
db.session.add(carg15)
carg16 = Cargador("ocupado", 6, "Carga Normal", 1)
db.session.add(carg16)
carg17 = Cargador("ocupado", 7, "Carga Rápida", 1)
db.session.add(carg17)
carg18 = Cargador("ocupado", 8, "Carga Rápida", 1)
db.session.add(carg18)
```

#### 24. Creación de Cargadores

Las horas y el consumo son creadas a la par para que de esta manera se pueda agregar el consumo a las horas establecidas anteriormente:

```

##### HORAS #####
horas = []
for i in range(100):
    dia = datetime.today()
    hora = datetime.today()

    h1 = Horas(dia, hora)
    db.session.add(h1)
    horas.append(h1)

db.session.commit()

##### CONSUMO #####
consumos = []
for i in range(100):

    cargador = random.choice(cargadores).id_cargador

    hora = random.choice(horas).id

    potencia_consumida = fake.random_int(min=0, max=100)
    potencia_maxima = fake.random_int(min=0, max=100)

    co1 = Consumo(cargador, hora, potencia_consumida, potencia_maxima)
    db.session.add(co1)
    consumos.append(co1)

db.session.commit()

horas = []

start = datetime(2022, 1, 1)
end = datetime(2030, 1, 1)

while start < end:
    a = Horas(start)
    db.session.add(a)
    start = start + relativedelta(minutes=60)
    horas.append(a)

db.session.commit()

consumos = []
cons11 = Consumo(1, datetime.fromisoformat("2022-06-10 00:00:00"), 0, 230)
db.session.add(cons11)
consumos.append(cons11)
cons12 = Consumo(1, datetime.fromisoformat("2022-06-10 01:00:00"), 59, 230)
db.session.add(cons12)
consumos.append(cons12)
cons13 = Consumo(1, datetime.fromisoformat("2022-06-10 02:00:00"), 216, 230)
db.session.add(cons13)

```

## .25. Creación de Horas y Consumo

La creación de modelos fue hecha a mano, investigando algunos modelos que será de los más populares a futuro, de esta manera el cliente podrá escoger un vehículo igual o por lo menos similar al que tiene dependiendo de sus especificaciones:

```
#####Modelos#####
model_list = ["500e Cabrio eléctrico", "Taycan eléctrico", "e-tron GT eléctrico", "Leaf eléctrico", "Ioniq eléctrico", "i3 eléctrico", "ID.3 eléctrico",
mod = Modelo(model_list[0], "Fiat", False, 42)
db.session.add(mod)
mod2 = Modelo(model_list[1], "Porsche", True, 93.4)
db.session.add(mod2)
mod3 = Modelo(model_list[2], "Audi", True, 93.4)
db.session.add(mod3)
mod4 = Modelo(model_list[3], "Nissan", False, 42.6)
db.session.add(mod4)
mod5 = Modelo(model_list[4], "Hyundai", False, 72.6)
db.session.add(mod5)
mod6 = Modelo(model_list[5], "BMW", False, 42.2)
db.session.add(mod6)
mod7 = Modelo(model_list[6], "Volkswagen", True, 77)
db.session.add(mod7)
mod8 = Modelo(model_list[7], "Polestar", False, 69)
db.session.add(mod8)
mod9 = Modelo(model_list[8], "Lexus", False, 54.3)
db.session.add(mod9)
mod10 = Modelo(model_list[9], "Kia", True, 77)
db.session.add(mod10)
db.session.commit()
```

```
model_list = [
'500e Cabrio electrico',
'Taycan electrico',
'e-tron GT electrico',
'Leaf electrico',
'Ioniq electrico',
'i3 electrico',
'ID.3 electrico',
'2 electrico',
'UX300e electrico',
'EV6 electrico',
]
mod = Modelo(model_list[0], 'Fiat', False, 42)
db.session.add(mod)
mod2 = Modelo(model_list[1], 'Porsche', True, 93.4)
db.session.add(mod2)
mod3 = Modelo(model_list[2], 'Audi', True, 93.4)
db.session.add(mod3)
mod4 = Modelo(model_list[3], 'Nissan', False, 42.6)
db.session.add(mod4)
mod5 = Modelo(model_list[4], 'Hyundai', False, 72.6)
db.session.add(mod5)
mod6 = Modelo(model_list[5], 'BMW', False, 42.2)
db.session.add(mod6)
mod7 = Modelo(model_list[6], 'Volkswagen', True, 77)
db.session.add(mod7)
mod8 = Modelo(model_list[7], 'Polestar', False, 69)
db.session.add(mod8)
mod9 = Modelo(model_list[8], 'Lexus', False, 54.3)
db.session.add(mod9)
mod10 = Modelo(model_list[9], 'Kia', True, 77)
db.session.add(mod10)
db.session.commit()
```

## .26. Creación de modelos

La creación de vehículos está ligada tanto a los modelos como a los clientes:

```

#####Vehiculos#####
vehiculos = []
for i in range(50):

    letras = ''.join(random.choices(string.ascii_uppercase, k=3))
    numeros = ''.join(random.choices(string.digits, k=4))
    matricula = ''.join(random.choices(letras+numeros, k=7))
    procentaje_bat = random.randint(0, 100)
    modelo = random.choice(model_list)

    v = Vehiculo(matricula, procentaje_bat, modelo)
    db.session.add(v)
    vehiculos.append(v)

db.session.commit()

##### VehiculoCliente #####
vehiculosClientes = []
for i in range(len(clientes)-1):
    vehiculo = random.choice(vehiculos).matricula
    cliente = clientes[i].id_cliente

    vc = VehiculoCliente(vehiculo, cliente)
    db.session.add(vc)
    vehiculosClientes.append(vc)

db.session.commit()

```

```

vehiculos = []
vehiculoBueno = Vehiculo("2450GDF", 100, '500e Cabrio electrico')
vehiculoBueno2 = Vehiculo("4950KZK", 100, 'Taycan electrico')
vehiculoBueno3 = Vehiculo("5134FFJ", 100, 'Leaf electrico')
vehiculoBueno4 = Vehiculo("LU50KZK", 100, 'EV6 electrico')
db.session.add(vehiculoBueno)
db.session.add(vehiculoBueno2)
db.session.add(vehiculoBueno3)
db.session.add(vehiculoBueno4)

```

## .27. Creación de Vehículos

La creación de las reservas se basó en crear fechas y horas falsas con el faker para poder asociarlas a un cargador en concreto, un vehículo y un cliente:

```

##### RESERVAS #####
reservas = []
for i in range(100):

    fecha_entrada = fake.date_time_between(start_date="-2y", end_date="now")
    fecha_salida = fake.date_time_between(start_date="-2y", end_date="now")

    procetnaje_carga = fake.random_int(min=0, max=100)

    precio_carga_completa = round(random.uniform(1.0, 100.0), 3)
    precio_carga_actual = round(random.uniform(1.0, 100.0), 3)

    estado = fake.pybool()
    tarifa = round(random.uniform(1.0, 100.0), 3)
    asistida = fake.pybool()
    estado_pago = fake.pybool()

    id_cargador = random.choice(cargadores).id_cargador
    id_vehiculo = random.choice(vehiculos).matricula
    id_cliente = random.choice(clientes).id_cliente

    r1 = Reserva(fecha_entrada, fecha_salida, procetnaje_carga, precio_carga_completa, precio_carga_actual,
                 estado, tarifa, asistida, estado_pago, id_cargador, id_vehiculo, id_cliente)
    db.session.add(r1)
    reservas.append(r1)

db.session.commit()

res1 = Reserva(datetime.fromisoformat("2022-06-11 10:30:01"), datetime.fromisoformat("2021-03-31 08:48:49"), 100, 50.754, 54.066,
               False, 28.309, False, True, 55, "QR1RQR", 14)
db.session.add(res1)
reservas.append(res1)

```

## 27. Creación de Reservas

Para la creación de Avisos se usó ids de las reservas ya creadas y de sus correspondientes clientes agregando algún mensaje:

```

##### AVISO #####
for i in range(20):
    idreserva=random.choice(reservas).id_reserva
    iddelcliente=random.choice(reservas).id_cliente
    fechareserva=random.choice(reservas).fecha_entrada
    a = Aviso("Cancelación","motomamiiiii",fechareserva, idreserva, iddelcliente)#c.id_cliente
    db.session.add(a)
    db.session.commit()

```

## 28. Creación de Avisos

Los tickets están relacionados con los avisos pero más concretamente con las averías ocurridas con al APP o con una estación en concreto:

```

##### TICKET #####
tcickets = []
for i in range(15):
    idcliente=random.choice(clientes).id_cliente
    fecha = fake.date_time_between(start_date="-2y", end_date="now")
    ticket = Ticket(fecha, "Error App",
                    "No me deja reservar en la estacion de Rambla Exposicio, no se que le pasa", "Pendiente", idcliente)
    db.session.add(ticket)
    tcickets.append(ticket)
db.session.commit()

sop1 = Ticket(datetime.fromisoformat("2022-06-15 13:00:08"), "No me carga el coche", "No me carga el coche, no se por que",
              "Pendiente", 30)

```

## 29. Creación de Tickets

Los mensajes son parte de los tickets de asistencia con la app por donde el Trabajador, usando la Página Web, puede comunicarse con el cliente directamente:

```

##### MENSAJE #####
    for i in range(15):
        trabj = random.choice(trabajadores).id_trabajador
        tick = tcickets[i].id_ticket
        fecha = fake.date_time_between(
            start_date="-2y", end_date="now")
        m = Mensaje("Me parece que lo haceis todo mal, salu2",fecha,
                    trabj, tick)
        db.session.add(m)
        db.session.commit()

```

```

Mens21 = Mensaje("Que tenga buen dia", datetime.fromisoformat("2022-06-15 13:30:08"), 2, 1)
db.session.add(Mens21)

```

### 30. Creación de Mensajes

Las averías son problemas registrados por un trabajador en una estación en concreto que pueden ser asociada a los tickets:

```

##### AVERIA #####
    for i in range(3):
        fecha = fake.date_time_between(start_date="-2y", end_date="now")
        estac=random.choice(estaciones).id_estacion
        trabj=random.choice(trabajadores).id_trabajador

        av = Averia(fecha, "Pendiente",
                    "No funciona cargador 8 de la estación por mantenimiento", trabj, estac)
        db.session.add(av)
        db.session.commit()

```

### 31. Creación de Averías

Las sesiones son registros de el ingreso a la web de un trabajador en concreto:

```

##### SESIONES #####
    for i in range(10):
        trabj=random.choice(trabajadores).id_trabajador
        fecha_1 = fake.date_time_between(
            start_date="-2y", end_date="now")
        fecha_2 = fake.date_time_between(
            start_date="-2y", end_date="now")
        s = Sesiones(fecha_1, fecha_2, trabj)
        db.session.add(s)
        db.session.commit()

```

### 32. Creación de Sesiones

Los cupones son descuentos únicos que un cliente puede usar y no pueden ser compartidos con otros clientes, es decir, son únicos por usuario:

```

##### CUPONES #####
estado = ['Activa', 'Desactiva']
for i in range(10):
    idcliente = random.choice(clientes).id_cliente
    cupon = fake.password(
        Length=8, special_chars=False, digits=True, upper_case=True, lower_case=False)
    est = random.choice(estado)
    c = Cupon(cupon, idcliente, est)
    db.session.add(c)
    db.session.commit()

cup0 = Cupon(16, "M28202246CP", 89, "Usado")
db.session.add(cup0)
cup1 = Cupon(17, "Z63480769RH", 34, "No usado")
db.session.add(cup1)
cup2 = Cupon(23, "G73313210TQ", 10, "Usado")
db.session.add(cup2)
cup3 = Cupon(29, "M68706352NF", 60, "Usado")
db.session.add(cup3)

```

### 33. Creación de Cupones