ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Trabajo Fin de Grado

Sistema de monitorización telemetría de una estación de recarga de vehículos eléctricos usando estándar OCPP 1.6

AUTOR: Francisco Sánchez Sánchez
DIRECTOR: Alejandro Santos Martínez Sala
Junio / 2018

A mi director de Proyecto, a mi familia y a mis amigos, sin todos ellos no habría sido posible.

Índice

1. Motivación para la realización de este trabajo

Hoy en día, surgen nuevas necesidades para mejorar el mundo en el que vivimos y tener una mejor calidad de vida, y una de ellas es reducir la contaminación que producen los vehículos con motor de combustión.

Es por ello por lo que nacen los vehículos propulsados con motor eléctrico, ya sea de forma completa o *compartiendo lugar* con el motor de combustión (lo que se denomina *híbrido*). ¿Qué hay de interés en esto? Es sencillo: tenemos la posibilidad de contribuir a un mundo más limpio y energético.

No obstante, cualquier vehículo, propulsado por el sistema que sea, necesita *repostar* para seguir circulando. En el presente existen multitud de *gasolineras* para los vehículos de combustión pero, ¿existen también *gasolineras* para vehículos eléctricos?

La respuesta es *sí*. El término con el que se conoce a estas *particulares gasolineras* es *electrolineras*. Es una analogía respecto a las gasolineras, pero en este caso, en vez de surtidores de gasolina, disponen de surtidores eléctricos para la recarga de vehículos tanto *100% eléctricos* como *híbridos, ambos enchufables*.

Vamos a profundizar un poco más. ¿Y si se pudieran *gestionar* dichas electrolineras para, por ejemplo, informar a los usuarios cuál es la más utilizada, la más barata o la más rápida en cargar nuestro vehículo o, sencillamente, que los administradores de las mismas puedan *saber a cada momento, en tiempo real, en qué estado está la electrolinera*?

Aquí es donde nace la necesidad de crear un sistema, un prototipo de aplicación, que gestione dichas electrolineras *en tiempo real*, o como hemos llamado al presente trabajo, ***crear un sistema de telemetría en tiempo real*** basado en el protocolo OCPP.

Y para finalizar este punto, ¿qué es el protocolo OCPP? *A grosso modo*, son las siglas de Open Charge Point Protocol (Protocolo abierto de punto de carga). Es un canal de comunicación entre la electrolinera y un sistema central que las gestiona. En el tercer capítulo se explica más detalladamente su funcionalidad.

2. Introducción. Objetivos del TFG.

2.1. Objetivos TFG

El presente Trabajo Fin de Grado tiene por objetivos:

- Introducir el concepto de *electrolineras*. Qué son, cómo funcionan y qué relación tienen con este proyecto.

- Relacionado con el primer punto, introducir y analizar el protocolo OCPP. Estado del arte del mismo, codificación del mismo en los puntos de recarga para crear redes de cargadores eléctricos.

- Introducir el concepto de *Websockets*. Funcionamiento *JSON over Websockets*. Diferencias entre *JSON* y *XML*.

- Introducir productos comerciales que satisfacen las necesidades de este proyecto. Fabricante, modelo, características principales.

- En colaboración con la empresa ***Solvent - Ingeniería y ahorro energético***, se ha diseñado una solución que se ha dividido en las siguientes fases:

    - Investigación acerca de los modelos comerciales existentes en el mercado.

    - Toma de decisión de la estación a elegir.

    - Por último y no menos importante, una *reunión previa* para acordar los *requisitos de diseño y funcionalidades de la interfaz* a desarrollar en la plataforma de gestión *y una reunión posterior para muestra del funcionamiento del mismo*.

- Implementar un prototipo de sistema de monitorización de telemetría de una estación de recarga. Introducción a los lenguajes y software utilizados para el desarrollo de la plataforma, *brainstorming* (lluvia de ideas), idea seleccionada, primeros pasos, fase de desarrollo detallada y finalmente pruebas de testeo y funcionamiento con futuras líneas de investigación y desarrollo.

2.2. Lenguajes web y herramientas de desarrollo utilizadas.

Lenguajes utilizados

En este apartado, se documentan los lenguajes de programación web utilizados en este proyecto, explicando *brevemente* su funcionamiento y *estado del arte*. En el capítulo *Bibliografía* estarán explicados de forma más detallada.

HTML

Hyper**T**ext **M**arkup **L**anguage. Lenguaje de marcas que permite crear *contenido estático* en una web, es decir, determina el contenido en la web pero no su funcionalidad, para ello están los lenguajes *funcionales*, por ejemplo *JavaScript*. Emplea *marcas* para definir *elementos*, como por ejemplo *títulos, párrafos* y *listas*. En la actualidad, se utiliza la versión 5 del estándar.

CSS

Cascading **S**tyle **S**heets, Hojas de estilo en cascada. Lenguaje empleado para dotar al contenido estático de propiedades visuales / estéticas, o dicho de otra forma, para *retocar visualmente* el contenido estático de una web. En el presente, se está utilizando la versión 3, que está en camino de convertirse en estándar. Emplea las marcas HTML para asignarle propiedades, ya sea posicionamiento, color, etcétera.

JavaScript

JavaScript, *JS*, se define como un lenguaje de programación *dinámico*, con idéntica sintaxis básica a Java con intención de reducir el número de nuevos conceptos necesarios para aprender el lenguaje. Su dinamismo incluye construcción de objetos en tiempo real, lista de parámetros variables o recuperación de código fuente, entre otras características.

Su estándar es *ECMAScript*. Desde 2012, todos los navegadores modernos soportan al completo ECMAScript 5.1 (junio de 2011). Por comodidad, se utiliza actualmente ES5, aunque se puede utilizar la que queramos a partir de esta versión.

Bootstrap

Bootstrap es una librería de código abierto para combinar HTML, CSS y JavaScript en los diseños de páginas web, siendo su versión actual la 4.0. Permite diseñar páginas web adaptadas a cualquier dispositivo y a cualquier resolución y es por ello que es el framework front-end más utilizado, combinado con AngularJS, a la hora de diseñar una página web.

Herramientas

Editor web

Brackets es un framework de código abierto implementado en lenguaje JavaScript, constantemente apoyado y desarrollado por una comunidad activa de usuarios y desarrolladores.

Se creó como editor de código web para *front-end / back-end developers* y diseñadores web. Es un editor ligero, moderno y potente, con herramientas visuales adecuadas para facilitar una creación de código ágil. Su interfaz y diseño *amigables* permite al desarrollador crear proyectos de pequeña y gran envergadura con una buena organización.

Gestor de repositorios y versiones de proyectos

Para la gestión de versiones y actualización de repositorios, se ha utilizado la herramienta *SourceTree*, integrándose bien con Git / GitHub. Posee una interfaz limpia con botones que implementan las diferentes funciones que se pueden realizar con los repositorios (*commit, push, pull, fetch, branch, merge*,...). En el apartado *Bibliografía* existe un apartado dedicado a explicar más en detalle cómo funciona cada comando y qué impacto tiene en el repositorio.

Desarrollo Back-end

- *XAMPP*

Como primera instancia de servidor web, se ha utilizado ***XAMPP***, un programa que ofrece múltiples módulos y opciones para configurar un ordenador y utilizarlo **como servidor**, para alojar aplicaciones web. Entre los módulos disponibles, se han instalado *Apache*, que permite crear un servidor de forma fácil y sencilla, un módulo *MySQL* para interactuar con bases de datos, inclusive con un gestor de bases de datos muy completo, *PHPMyAdmin*.

- *NodeJS*

Como segunda instancia, y más enfocado al objetivo principal de la arquitectura a implementar, es un framework por excelencia para dotar a las aplicaciones web de tecnologías *server-side* (lado del servidor). Permite al usuario *modular* el servidor en múltiples rutas independientes, cada una realizando una función distinta, permitiendo crear así un servidor completo y multifuncional. Se puede ejecutar a través de cualquier terminal disponible en el equipo (*PowerShell*, *Git Bash*, *CMD,...*) o a través de la suya propia.

Incluye un instalador/gestor de paquetes, *npm (Node Package Manager)*. Con la opción *-S* se guarda en un archivo, *package.json*, para facilitar la exportación de las dependencias a otro ordenador e instalar los paquetes necesarios para continuar con el desarrollo del proyecto.

Simulación de peticiones al servidor

Una vez que se ha desarrollado el servidor, ¿cómo se puede comprobar que funciona todo lo que se ha implementado? Existen multitud de programas, por ejemplo *XAMPP* (explicado anteriormente), *LAMP* (igual que *XAMPP* pero dedicado al S.O. Linux), etc… pero si *sólamente se quiere comprobar si el servidor opera como es debido a través de los conocidos métodos GET, PUT, POST y DELETE del protocolo HTTP*, existe una gran utilidad, llamada *Postman*.

Este programa permite crear y simular peticiones GET, POST, PUT y DELETE sin necesidad de instalar ningún programa externo, ya que, mediante la terminal *PowerShell* de Windows o bien la propia de *Node*, se puede arrancar un simple servidor web con pocas líneas de código.

3. Protocolo OCPP. Definición y usos.

El protocolo OCPP, cuyas siglas significan *Open Charge Point Protocol*, es un protocolo libre / abierto para puntos de recarga, o dicho de otra forma, protocolo *open-source* para gestionar, a través de un web manager o sistema de gestión en la nube (CPMS, *Charge Point Management System*), estaciones de recarga de vehículos eléctricos, conocidos con el término *electrolineras*, cuya función es idéntica al de un surtidor convencional de combustible con la diferencia de que suministra energía eléctrica a vehículos que dispongan de sistema de baterías, ya sean *híbridos* (motor de combustión + motor eléctrico) o *puramente eléctricos*.

Definiciones

A lo largo del documento, existen una serie de términos que es mejor indicar su significado a principio de este capítulo para una mejor lectura del documento:

- **OCPP**: Open Charge Point Protocol. Protocolo abierto de puntos de recarga.
- **EV**: Electrical Vehicle. Vehículo eléctrico.
- **CPMS**: Charge Point Management System. Sistema central de gestión de puntos de recarga.
- **OCA**: Open Charge Alliance.
- **JSON**: JavaScript Object Notation. Objeto construido en lenguaje JavaScript.
- **CS**: Charge Station. Estación de recarga.
- **SOAP/XML**: Simple Object Access Protocol / eXtensible Markup Language.

3.1. ¿Qué es el protocolo OCPP?¿Cómo y para qué se utiliza este protocolo?

OCPP fue una iniciativa de una fundación holandesa, *e-laad*, cuyos comienzos datan del año 2009, en el que lanzó *e-loading* junto con algunas empresas asociadas con OCPP. Estableció una red de más de 3000 estaciones de recarga públicas para coches eléctricos por todo Países Bajos entre dicho año y el comienzo de 2014, con el objetivo de estimular e incentivar a los vehículos eléctricos. Hay más de 440 participantes de 40 países, incluyendo EE.UU, Singapur, Alemania, Francia, Gran Bretaña y Rusia.

¿Cuál es el futuro de OCPP? De momento, todo el mundo puede unirse al uso y desarrollo del mismo, ya que es *100% open-source* (gratuito). Sin embargo, se está buscando la manera de convertirlo en una fórmula de coste neutro para así poder continuar trabajando en el desarrollo del protocolo OCPP, lo que quiere decir que el protocolo seguirá siendo *open-source*, pero estando más dispuestos a crear *colaboraciones entre empresas* para el desarrollo del mismo.

Versiones del protocolo OCPP

OCPP cuenta con varias versiones. Los primeros desarrollos datan del año 2010, año en el que la compañía holandesa *e-laad* lanzó la primera versión comercial del protocolo (1.2). Posteriormente, en el año 2013, *OCPP Forum* (actualmente *OCA*) lanzó la segunda versión del protocolo (1.5), mejorando la implementación del primero. Poco después, la actual OCA (Open Charge Alliance) lanzó la versión 1.6 que es la mejora de la anterior incluyendo soporte JSON vía WebSockets. En la siguiente ilustración se muestra el avance del desarrollo de dicho protocolo desde sus inicios hasta la actualidad:

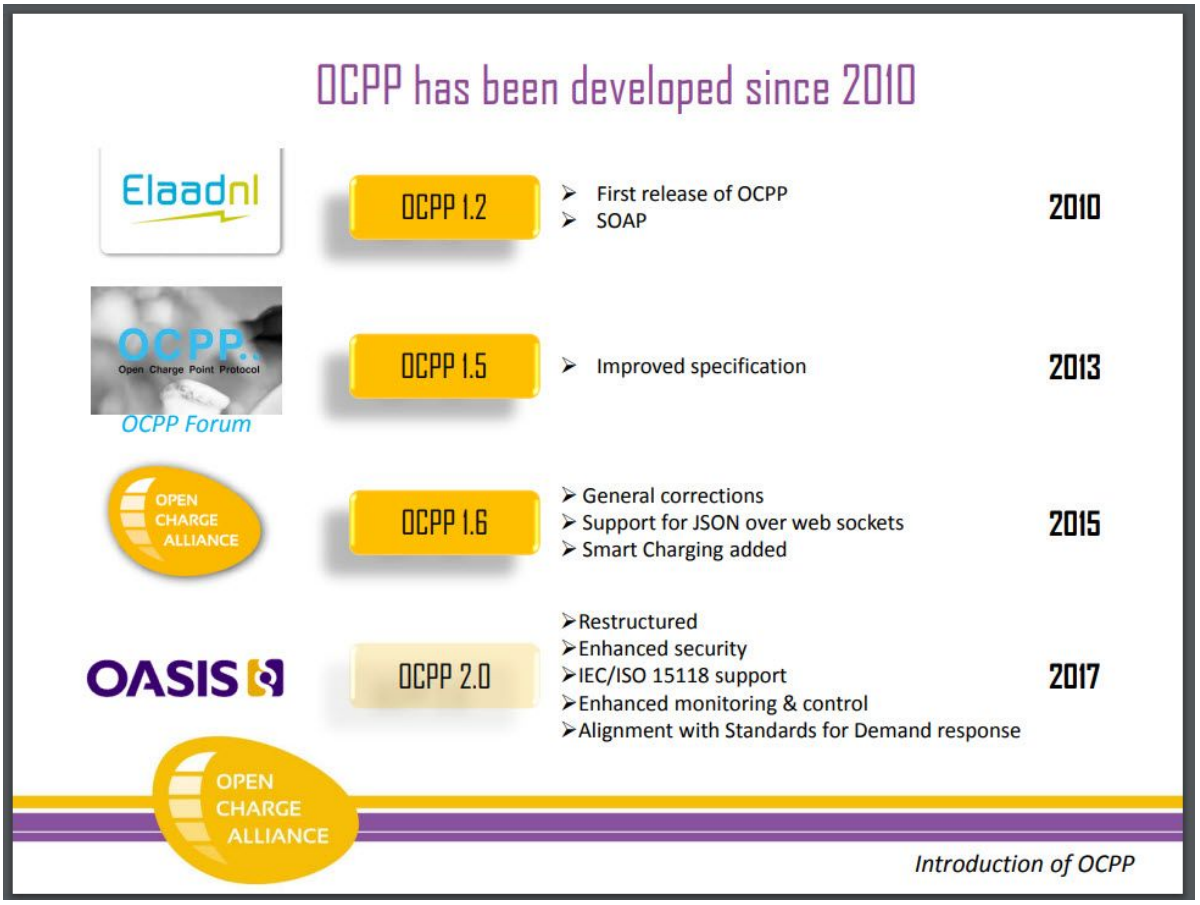


Ilustración 1: A fecha 27/12/2017, la última versión comercial de este protocolo es la 1.6 con soporte para SOAP y JSON

El *14 de diciembre de 2017* se hace público el borrador de la nueva versión del protocolo, la 2.0, la cual incluye diversas mejoras respecto a la anterior versión:

- *Gestión de dispositivos*. Tiene con objetivo proporcionar diversas funcionalidades para los operarios de CS:
  - Reporte de inventarios.
  - Mejora en el reporte de errores y estados.
  - Configuración mejorada.
  - Monitorización personalizable.

- Mejoras en el manejo de grandes cantidades de transacciones:
    - Agrupación de mensajes relacionados con las transacciones.
    - Reducción del consumo de ancho de banda.
- Mejoras de seguridad:
    - 3 niveles de perfiles de seguridad.
    - Gestión de claves para certificados de clientes (*client-side*).
    - Asegurando actualizaciones de seguridad.
    - Registro de eventos de seguridad.
- Carga inteligente extendida (mejorada):
- Estándar *ISO/IEC 15118 Plug & Charge*:
    - Nuevo protocolo de comunicación entre vehículo eléctrico y el equipo suministrador de vehículos eléctricos que permitirá comunicaciones más seguras.
- Mejoras en la experiencia de usuario:
    - Más opciones de autorización: terminales de pago, smartphones, etc…
    - Muestra de mensajes.
    - Idiomas preferidos para los conductores de EV.
    - Tarifas y costos. El usuario podrá ver el coste (€/kWh) antes de comenzar la transacción, ver el coste en tiempo real de la misma y/o ver el coste final de la transacción.
- Mejoras en el protocolo *OCPP over JSON (OCPP-J)*.
    - Enrutamiento simple de mensajes.
    - Mensajes extendidos.
    - ***Se elimina el protocolo SOAP como transporte de datos****.*
- Pequeños cambios / implementaciones.
    - Mensajes renombrados.
    - Generación del *TransactionId* por las CSs.
    - Numeraciones extendidas.
    - Marcado de transacciones offline.
    - Dar al usuario una razón por no estar autorizado a cargar.

La característica más reseñable de la versión 2.0 es la ***eliminación*** del protocolo ***SOAP*** como método de transporte de datos, dejando como único protocolo ***JSON***. Es un paso *importante*, ya que la principal mejora de JSON respecto a SOAP/XML es la reducción de cabeceras y por tanto de tamaño de mensajes, optimizando el consumo de ancho de banda en la red.

Otro concepto que introduce el protocolo es ***OCPP Endpoint***: es una analogía a una *puerta de enlace* (*gateway*) de un router. Se trata de la dirección websocket del sistema central al que apunta la estación de recarga. A través de esta dirección, la estación de recarga y el sistema central se comunican para enviarse datos entre sí:

Ilustración 2: Estructura dirección WebSocket

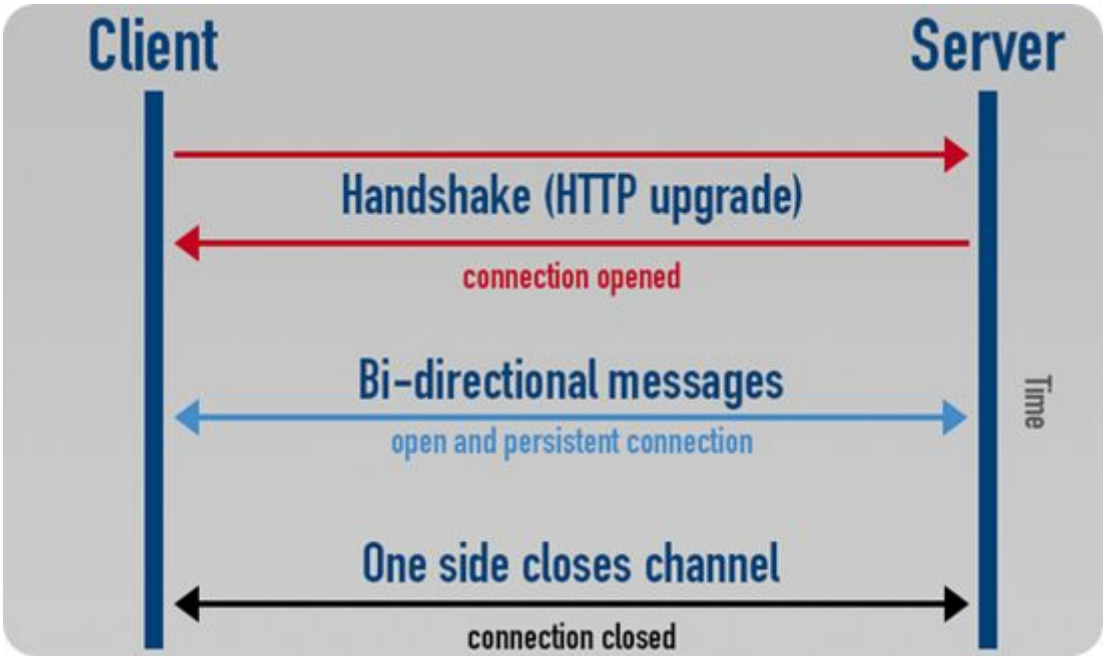


Ilustración 3: Cronograma cliente y servidor WebSocket

Obtención de los datos de interés

Los parámetros y sus valores, indicados en el apartado *Objetivos TFG*, se obtienen a través de un mensaje, ***MeterValues***, el cual se configura para que se envíe periódicamente cada intervalo de tiempo X la sesión o sesiones de telemetría registradas, y por consiguiente, las transacciones entre la estación y los vehículos que carguen de ella (mensajes (Start/Stop)Transaction).

MeterValues contiene los parámetros y datos *que el fabricante haya decidido implementar*, siendo recomendable la totalidad de los mismos. A través de él, cuando llegue una sesión de telemetría al sistema central (prototipo), se registrará y se mostrará por pantalla (la interfaz web se refrescará cada X segundos para mostrarla).

Según la especificación OCPP, el mensaje *MeterValues* consta de la siguiente estructura:

- connectorId: Id del conector usado en la carga del vehículo.
- transactionId: Id de la carga realizada
- meterValue: los parámetros registrados de la carga del vehículo, acompañados de fecha y hora. ***No todos los valores son registrados***, ya que cada fabricante decide cuáles quiere implementar. No obstante, para la especificación solicitada por Solvent, habiendo realizado un análisis de las electrolineras comerciales disponibles, **la mayoría de parámetros están disponibles**.

6.31. MeterValues.req

This contains the field definition of the MeterValues.req PDU sent by the Charge Point to the Central System. See also Meter Values

Field Name	Field Type	Card.	Description
connectorId	integer connectorId >= 0	1..1	Required. This contains a number (>0) designating a connector of the Charge Point.'0' (zero) is used to designate the main powermeter.
transactionId	integer	0..1	Optional. The transaction to which these meter samples are related.
meterValue	MeterValue	1..*	Required. The sampled meter values with timestamps.

7.33. MeterValue

Class

Collection of one or more sampled values in MeterValues.req. All sampled values in a MeterValue are sampled at the same point in time.

Field Name	Field Type	Card.	Description
timestamp	dateTime	1..1	Required. Timestamp for measured value(s).
sampledValue	SampledValue	1..*	Required. One or more measured values

7.43. SampledValue

Class

Single sampled value in MeterValues. Each value can be accompanied by optional fields.

Field Name	Field Type	Card.	Description
value	String	1..1	Required. Value as a "Raw" (decimal) number or "SignedData". Field Type is "string" to allow for digitally signed data readings. Decimal numeric values are also acceptable to allow fractional values for measurands such as Temperature and Current.
context	ReadingContext	0..1	Optional. Type of detail value: start, end or sample. Default = "Sample.Periodic"
format	ValueFormat	0..1	Optional. Raw or signed data. Default = "Raw"
measurand	Measurand	0..1	Optional. Type of measurement. Default = "Energy.Active.Import.Register"

7.31. Measurand

Enumeration

Allowable values of the optional "measurand" field of a Value element, as used in MeterValues.req and StopTransaction.req messages. Default value of "measurand" is always "Energy.Active.Import.Register"

Value	Description
Current.Export	Instantaneous current flow from EV
Current.Import	Instantaneous current flow to EV
Current.Offered	Maximum current offered to EV
Energy.Active.Export.Register	Energy exported by EV (Wh or kWh)
Energy.Active.Import.Register	Energy imported by EV (Wh or kWh)

Value	Description
Energy.Reactive.Export.Register	Reactive energy exported by EV (varh or kvarh)
Energy.Reactive.Import.Register	Reactive energy imported by EV (varh or kvarh)
Energy.Active.Export.Interval	Energy exported by EV (Wh or kWh)
Energy.Active.Import.Interval	Energy imported by EV (Wh or kWh)
Energy.Reactive.Export.Interval	Reactive energy exported by EV. (varh or kvarh)
Energy.Reactive.Import.Interval	Reactive energy imported by EV. (varh or kvarh)
Frequency	Instantaneous reading of powerline frequency
Power.Active.Export	Instantaneous active power exported by EV. (W or kW)
Power.Active.Import	Instantaneous active power imported by EV. (W or kW)
Power.Factor	Instantaneous power factor of total energy flow
Power.Offered	Maximum power offered to EV
Power.Reactive.Export	Instantaneous reactive power exported by EV. (var or kvar)
Power.Reactive.Import	Instantaneous reactive power imported by EV. (var or kvar)
RPM	Fan speed in RPM
SoC	State of charge of charging vehicle in percentage
Temperature	Temperature reading inside Charge Point.
Voltage	Instantaneous AC RMS supply voltage

El parámetro medido por defecto es *Energy Active Import Register* (registro de energía activa importada), que es el principal del mensaje ***StopTransaction*** (cuando el usuario realiza una carga a su vehículo eléctrico (*EV, Electrical Vehicle*)).

El esquema *MeterValues* sigue la misma estructura que todos los mensajes disponibles, tal y como se especifica en la especificación OCPP:

- ID que indica el tipo de mensaje
- ID de la transacción (único para cada par de mensajes *request-response*).
- Tipo de mensaje
- *Payload*, que hace referencia al contenido del tipo de mensaje.

Para más claridad, ***un ejemplo de cómo sería un mensaje MeterValues con los parámetros solicitados por Solvent***:

```
[2, "524633", "MeterValues", {
  "connectorId": 2,
  "transactionId": 0,
  "values": [
    {
      "timestamp": "2013-03-07T16:52:16Z",
      "values": [
        {
          "value": "0",
          "unit": "Wh",
          "measurand": "Energy.Active.Import.Register"
        },
        {
          "value": "0",
          "unit": "varh",
          "measurand": "Energy.Reactive.Import.Register"
        },
        {
          "value": "0",
          "unit": "V",
          "measurand": "Voltage"
        },
        {
          "value": "0",
          "unit": "kW",
          "measurand": "Power.Offered"
        },
        {
          "value": "0",
          "unit": "A",
          "measurand": "Current.Offered"
        },
        {
          "value": "0",
          "unit": "Wh",
          "measurand": "Energy.Active.Export.Register"
        },
        {
          "value": "0",
          "unit": "varh",
          "measurand": "Energy.Reactive.Export.Register"
        },
        {
          "value": "50",
          "unit": "Percent",
          "measurand": "SoC"
        },
```

```
    {
      "value": "0",
      "unit": "kWh",
      "measurand": "Power.Active.Export"
    },
    {
      "value": "0",
      "unit": "kWh",
      "measurand": "Power.Active.Import"
    },
    {
      "value": "0",
      "unit": "kWh",
      "measurand": "Power.Reactive.Export"
    },
    {
      "value": "0",
      "unit": "kWh",
      "measurand": "Power.Reactive.Import"
    }
  ]
},
{
  "timestamp": "2013-03-07T21:52:16Z",
  "values": [
    {
      "value": "100",
      "unit": "Wh",
      "measurand": "Energy.Active.Import.Register"
    },
    {
      "value": "150",
      "unit": "varh",
      "measurand": "Energy.Reactive.Import.Register"
    },
    {
      "value": "20",
      "unit": "V",
      "measurand": "Voltage"
    },
    {
      "value": "80",
      "unit": "kW",
      "measurand": "Power.Offered"
    },
    {
      "value": "120",
      "unit": "A",
```

```
      "measurand": "Current.Offered"
     },
     {
      "value": "110",
      "unit": "Wh",
      "measurand": "Energy.Active.Export.Register"
     },
     {
      "value": "150",
      "unit": "varh",
      "measurand": "Energy.Reactive.Export.Register"
     },
     {
      "value": "99",
      "unit": "Percent",
      "measurand": "SoC"
     },
     {
      "value": "10",
      "unit": "kWh",
      "measurand": "Power.Active.Export"
     },
     {
      "value": "20",
      "unit": "kWh",
      "measurand": "Power.Active.Import"
     },
     {
      "value": "30",
      "unit": "kWh",
      "measurand": "Power.Reactive.Export"
     },
     {
      "value": "40",
      "unit": "kWh",
      "measurand": "Power.Reactive.Import"
     }
    ]
   }
  ]
}]
```

Esquemas / Cronogramas del protocolo

Como todo protocolo, OCPP dispone de esquema o esquemas en los que se describe su funcionamiento. Como ejemplo, la siguiente ilustración muestra el intercambio de mensajes cuando se produce un evento *Authorize*:

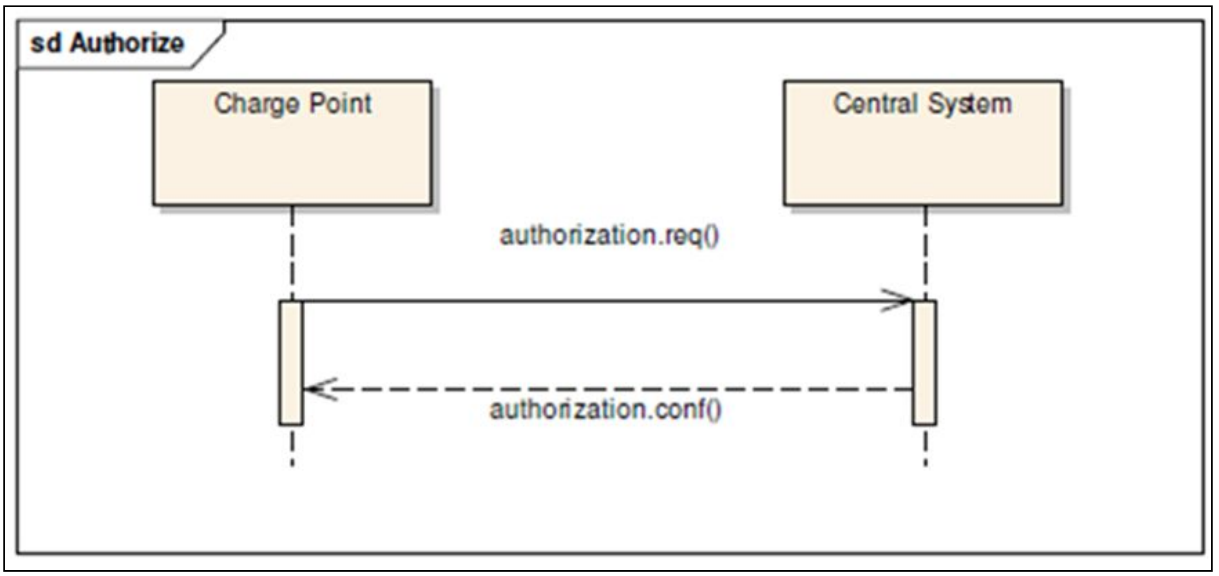


Ilustración 4: Cronograma mensaje Authorize

Este cronograma es el mismo para el resto de mensajes *iniciados por el punto de recarga*, que es lo habitual, ya que en el esquema de jerarquías, el "esclavo" (slave) es la estación de recarga y el *master* es el sistema central.

A continuación, se detalla *a grosso modo* el funcionamiento de cada mensaje implementado. En la sección *Pruebas y conclusiones* se explica su funcionamiento de forma más detallada:

- **Authorize**: comprueba si el usuario está autorizado a cargar su vehículo.
- **Heartbeat**: enviado por la estación de recarga para marcar que sigue operativa.
- **(Start/Stop)Transaction**: indican comienzo y final de una transacción (carga de vehículo).
- **MeterValues**: sesión (o sesiones) de telemetría que envía la estación de recarga de forma periódica.
- **StatusNotification**: mensajes informativos acerca de eventos ocurridos en la estación de recarga.
- **BootNotification**: mensaje inicial de la estación de recarga al sistema central indicando que ya está operativa.

RFID

RFID, cuyas siglas hacen referencia a *Radio Frequency IDentification*, es una tecnología que permite transmitir la identidad de un objeto mediante ondas de radio.

Un ejemplo son las etiquetas RFID, similares a una pegatina, son pequeños dispositivos que pueden ser incorporados a cualquier objeto, animal o persona que, a través de las antenas que lleva integradas, les permiten recibir y responder a peticiones por radiofrecuencia a través de un RFID emisor-receptor.

¿Cuál es el motivo de este apartado? *En primera instancia*, se llegó a la confusa conclusión de que esta técnica permitía acceder a la configuración de la electrolinera, lo cual es erróneo, ya que el usuario, a través de una tarjeta RFID, puede acceder únicamente a cargar el vehículo eléctrico si dicha tarjeta RFID estaba en una lista interna de autorización, *LocalAuthorizationList*, que es donde se registran las tarjetas que tendrán permisos para cargar vehículos eléctricos.

La administración de la electrolinera se hace a través de un panel de administración web interno realizado por el fabricante y la tarjeta RFID sirve para identificar usuarios y permitirles cargar vehículos eléctricos si se encuentran en la lista de identificadores autorizados.

3.2. Productos comerciales.

Tenemos una gran variedad de cargadores eléctricos que permiten la implementación del protocolo OCPP. A través de una búsqueda, con colaboración de la empresa *Solventie*, se construyó una lista con 4 potenciales candidatos de los fabricantes *Fenie*, *Ingeteam*, *Wallbox y Efimob.*

A continuación, vamos a evaluar las características *de interés* para el estudio, análisis y desarrollo del prototipo de gestión OCPP:

- *Fabricante Fenie, modelo MER-2-S*

Es un punto de recarga para un vehículo eléctrico, con conector *Mennekes* (tipo 2) que dispone de una conexión trifásica de 22 kW con salida de 32 Amperios máximo. Su funcionamiento es visible a través de 3 luces LED (rojo, azul y verde) y es compatible con el protocolo OCPP.

El conector *Mennekes* (o *tipo 2*) es un conector alemán de 55 cm de diámetro que dispone de 7 bornes (4 para corriente (trifásica), 2 para comunicaciones (modo 3) y 1 para toma a tierra):

Conector Mennekes tipo 2

- *Fabricante Fenie, modelo MER-D-32*

Similar al anterior modelo. Tiene 2 entradas para 2 vehículos, cada entrada con conector *Schuko* y *Mennekes* respectivamente. Dispone de conexión monofásica (7.4 kW, salida de 32 Amperios máximo). Funcionamiento visible a través de 3 luces LED (rojo, azul y verde), y tiene *de forma opcional* compatibilidad con protocolo OCPP.

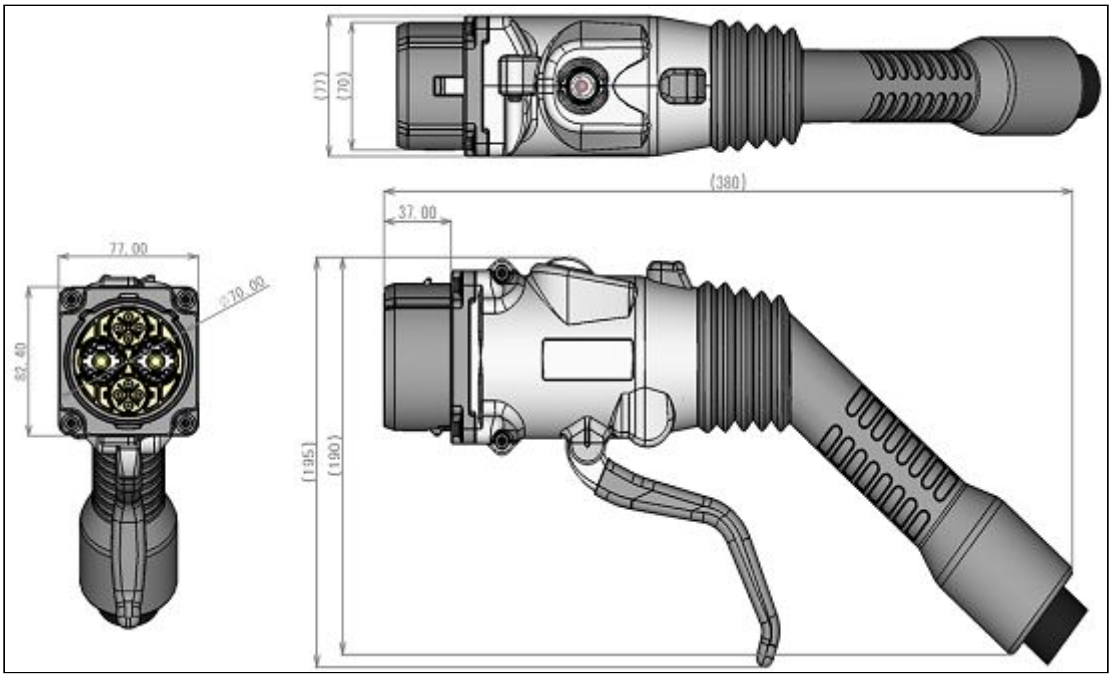
El conector *Schuko* lo podemos encontrar en múltiples aparatos electrónicos (tales como electrodomésticos, televisiones, microondas,...). Compatible con las tomas de corriente europeas, tiene 2 bornes para corriente y una para la toma a tierra. Soporta hasta 16 A, sólo para recarga lenta y sin comunicación integrada (modo 1).

Conector Schuko

Modelos Fenie MER-2-S y MER-D-32

Los modos de carga tienen que ver con la comunicación entre el vehículo y la infraestructura de recarga, en este caso una electrolinera. *A grosso modo*, existen cuatro modos:

- ***Modo 1***: aplica a cualquier toma de corriente con conector *Schuko*.
- ***Modo 2***: como el modo 1, pero dispone de control intermedio que verifica la correcta conexión del vehículo a la red de recarga.
- ***Modo 3***: los controles y las protecciones se encuentran dentro de la propia electrolinera, y el cable incluye incluye un hilo de comunicación integrado (como ejemplo de conector, *Mennekes*).
- ***Modo 4***: Existe un conversor a corriente continua y sólo se aplica a recarga rápida (como ejemplo está el conector CHAdeMO)



Conector CHAdeMO

- *Fabricante Ingeteam*

Este fabricante posee 4 modelos, 2 con conexión monofásica con potencias de 3.7 y 7.4 kW ofreciendo salidas de 16 y 32 Amperios respectivamente, y otros 2 con conexión trifásica con potencias de 11 kW y 22 kW ofreciendo idénticas salidas que la anterior. Compatible con el protocolo OCPP y la posibilidad de integrarlo, además de en el software del fabricante, en sistemas externos de control.

Se consultó con el equipo técnico del fabricante *cómo habían realizado la integración* del protocolo. Lo soporta *nativamente* adquiriendo, de forma opcional, un *módem de comunicaciones*, ya sea *ethernet* o *ethernet + conexión 3G*. Sin embargo, uno de los objetivos de este proyecto, que es comprobar *el funcionamiento de OCPP bajo WebSockets y estructuras JSON*, no se cumpliría, ya que el fabricante dispone de las versiones 1.2, 1.5 y en actual validación de 1.6 versión *SOAP*.

Ingeteam INGEREV GARAGE

- *Fabricante Wallbox, modelo Tipo 1*

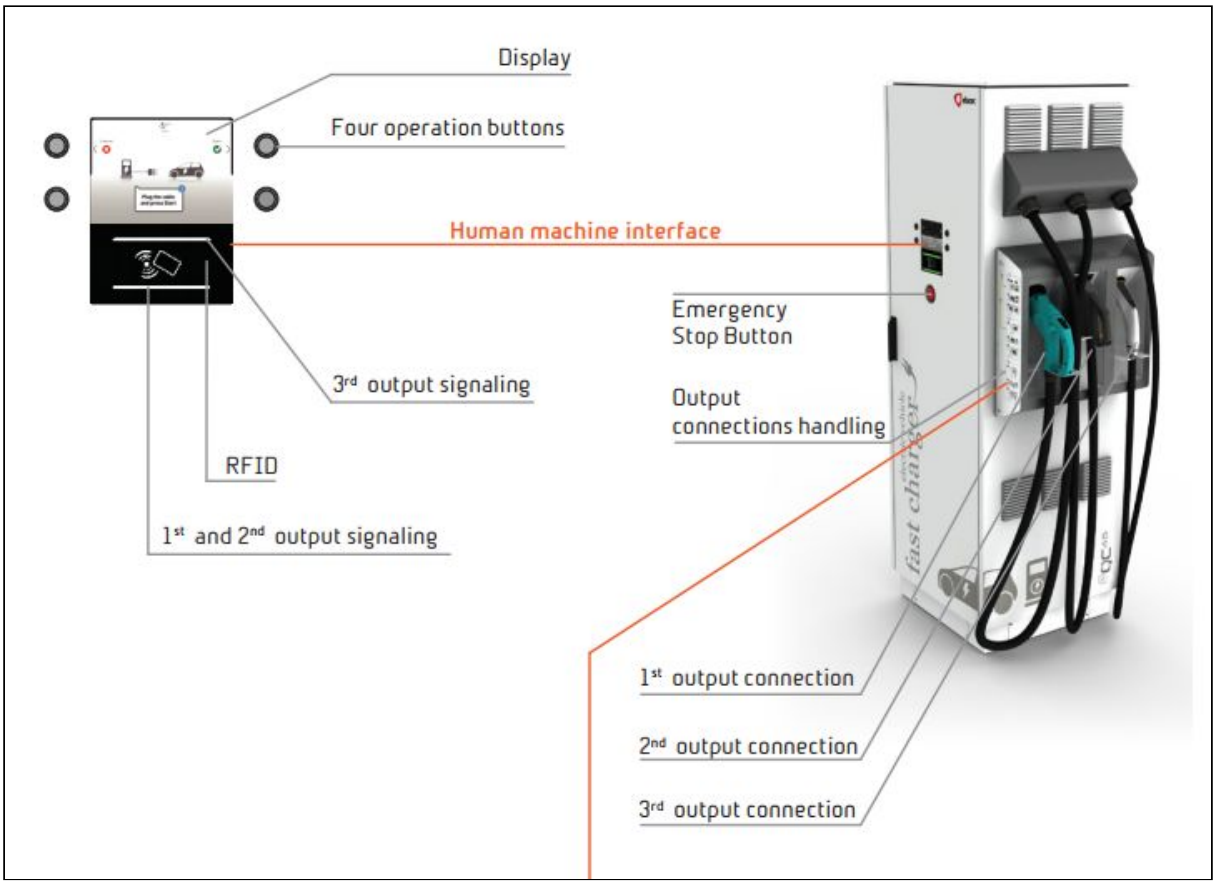
Ofrece posibilidad de gestionar la estación vía WiFi, con la posibilidad de actualizar *gratuitamente* el software de la misma de forma gratuita, utilizando la pantalla táctil que incorpora para dar facilidad al usuario a la hora de utilizarla, y a través de una aplicación para smartphones. Permite la integración vía OCPP con otros sistemas externos de gestión.

Wallbox Commander

Se mantuvieron conversaciones vía telemática y telefónica para comprobar compatibilidad con dicha estación, llegando a la conclusión de que el protocolo OCPP que implementan **no es 100% nativo**, ya que para que la estación se comunique con nuestro sistema central, dicho sistema central debe apuntar al portal *myWallbox* (analizado más adelante) para que se comuniquen la ER (Estación de Recarga) con el SC (Sistema Central), actuando de esta forma como *intermediario* entre la ER y el SC.

- *Fabricante Efimob*

Posee varios modelos comerciales, entre los que destacan *un cargador específico para autobuses híbridos / eléctricos* y cargadores rápidos que permiten cargar la batería de la mayoría de coches eléctricos *hasta un 80% en menos de 30 minutos*.



Efimob Efacec QC45

Se consultó al fabricante la implementación del protocolo (soporte nativo). En efecto, lo tiene, pero el problema, al igual que con Ingeteam, es que el fabricante trabaja con la versión 1.5 de OCPP SOAP, siendo el software de sus estaciones actualizable **manteniendo la misma versión de protocolo implementada**.

Disponen de cargadores que soportan hasta la versión 1.5 y otros que ya parten de la versión 1.6 (según información del fabricante, disponen de modelos AC para carga pública de pared, con 1 o 2 sockets de 7.4 o 22 kW y cargadores para vía pública de 1 o 2 sockets de mismas potencias).

Conclusiones del análisis anterior

Se llega a la conclusión, tras el análisis realizado en el anterior punto, que ninguna de las estaciones de recarga cumple con el requisito básico: **soporte nativo de OCPP versión JSON 1.6**.

No obstante, más adelante, se realizó una prueba para validar el correcto funcionamiento de este proyecto, ***quedando así un prototipo preparado para realizar una prueba con una electrolinera física que cumpla el requisito básico comentado***.

En los siguientes capítulos de esta memoria, se realiza un análisis de la implementación, primeramente con la toma de requisitos de la plataforma de gestión, pasando por el desarrollo y los cambios de arquitectura en la aplicación y terminando con el resultado final con evidencias y prueba de validación.

4. Arquitectura plataforma telemática de monitorización y gestión de electrolineras.

4.1. Especificación de requisitos de la empresa Solvent

Para el desarrollo de la plataforma, previamente tenemos que establecer unos objetivos para implementar una metodología ágil y unas fases para su posterior desarrollo.

Primeramente, se decidió realizar una reunión inicial de toma de contacto con la empresa *Solvent* para realizar un estudio y diversas pruebas iniciales ***con el objetivo de investigar el funcionamiento del protocolo OCPP***, así como las distintas opciones de gestión y administración de la estación a través del mismo.

En la primera reunión, se detalló realizar una primera visita a la electrolinera situada en la avenida de Los Pinos, situada en Murcia cerca del centro comercial Zig Zag. En dicha visita, acompañado por el CEO de *Solvent*, Javier Rincón, se realizó una toma de contacto con la electrolinera, analizando la interfaz web diseñada por el fabricante *Efacec* y realizando toma de ideas acerca de la misma. En la misma visita se intentó conectar con la electrolinera para realizar una prueba básica, la cual terminó sin éxito debido **al desconocimiento de la tecnología utilizada** por la electrolinera.

La siguiente reunión fue con *Raúl Franco*, componente del departamento técnico dentro de *Solvent* y supervisor del proyecto, en el cual se detallaron funcionalidades a añadir en la futura plataforma. En el siguiente apartado se explican las funcionalidades implementadas.

Funcionalidad básica del software

El software realizado para comunicación y representación de parámetros leídos de la estación de recarga, deberá cumplir al menos las siguientes funcionalidades:

1. Comunicación con 1 estación de recarga punto a punto o vía GSM para lectura de protocolo abierto OCPP (Open Charge Point Protocol o protocolo abierto de punto de recarga).
2. Diseño de programa para PC con las siguientes funcionalidades:
    a. Identificación del usuario que ha activado la estación de recarga a través de tarjetas RFID.
    b. Lectura en tiempo real y representación de las siguientes parámetros enviados por la estación de recarga: V, I, kWh, KVA, KVAr, cosy, %BAT, DTC (código de diagnóstico), tipo de manguera utilizada.
    c. Base de datos para archivo y lectura de los parámetros históricos anteriores con posibilidad de exportación a software comercial (en formato Excel).
    d. Posibilidad de imprimir informes de históricos o tickets de última recarga.
    e. Consultar en tiempo real el estado del cargador (encendido/cargando/apagado).
    f. Introducción de parámetros por teclado como €/kWh.

ESPECIFICACIÓN INTERFAZ DE USUARIO- METODOLOGÍA ORIENTADA EN USUARIO

Uno de los puntos en los cuales se dividió la reunión inicial fue la especificación de los requisitos principales de la interfaz de usuario de la plataforma web de gestión. A continuación se detallan las funcionalidades mínimas especificadas:

- Distinción entre usuarios que consultan recargas y administradores de electrolineras.
- Una **vista dedicada** a la gestión de la electrolinera.
- Incluir en el perfil de usuario, además de los datos principales, **la opción de añadir su coche eléctrico** de forma opcional.
- Parámetro fijo que puede ser **modificable**, en la pestaña de gestión de la electrolinera, que sirva al operario de la electrolinera para realizar el cobro de las cargas que realizan los usuarios.
- Vista **dedicada a la visualización de telemetría (incluida en la vista de gestión)**.
- Vista **dedicada a la consulta de datos de telemetría**, pudiendo realizar una consulta **completa** (de todos los parámetros disponibles), **especifica** (un parámetro en concreto), o acerca de la última sesión de telemetría, todas con opción de exportarlas a software comercial, por ejemplo Excel.
- Vista **dedicada a la consulta y/o adición de estaciones de recarga**.

4.2. Fase de desarrollo. Primeras pruebas

En este apartado, explicaremos paso por paso las fases que se han ido siguiendo para desarrollar la plataforma. El portátil de desarrollo utilizado funciona bajo el sistema operativo *Windows 10*.

Al aplicar la tecnología *JSON over WebSockets*, la conexión entre la electrolinera y el sistema central (CPMS) es **punto a punto, directa**, por lo que la solución a implementar es un ***CPMS con tecnología JSON over WebSockets*** que, a través de una correcta configuración del *OCPP Endpoint*, establezca una conexión *directa* con la electrolinera.

Primero, tenemos que configurar el entorno de trabajo.

- **Brackets**: Explicado anteriormente, es un editor de textos que nos permite crear aplicaciones web aplicando diferentes tecnologías/lenguajes web.

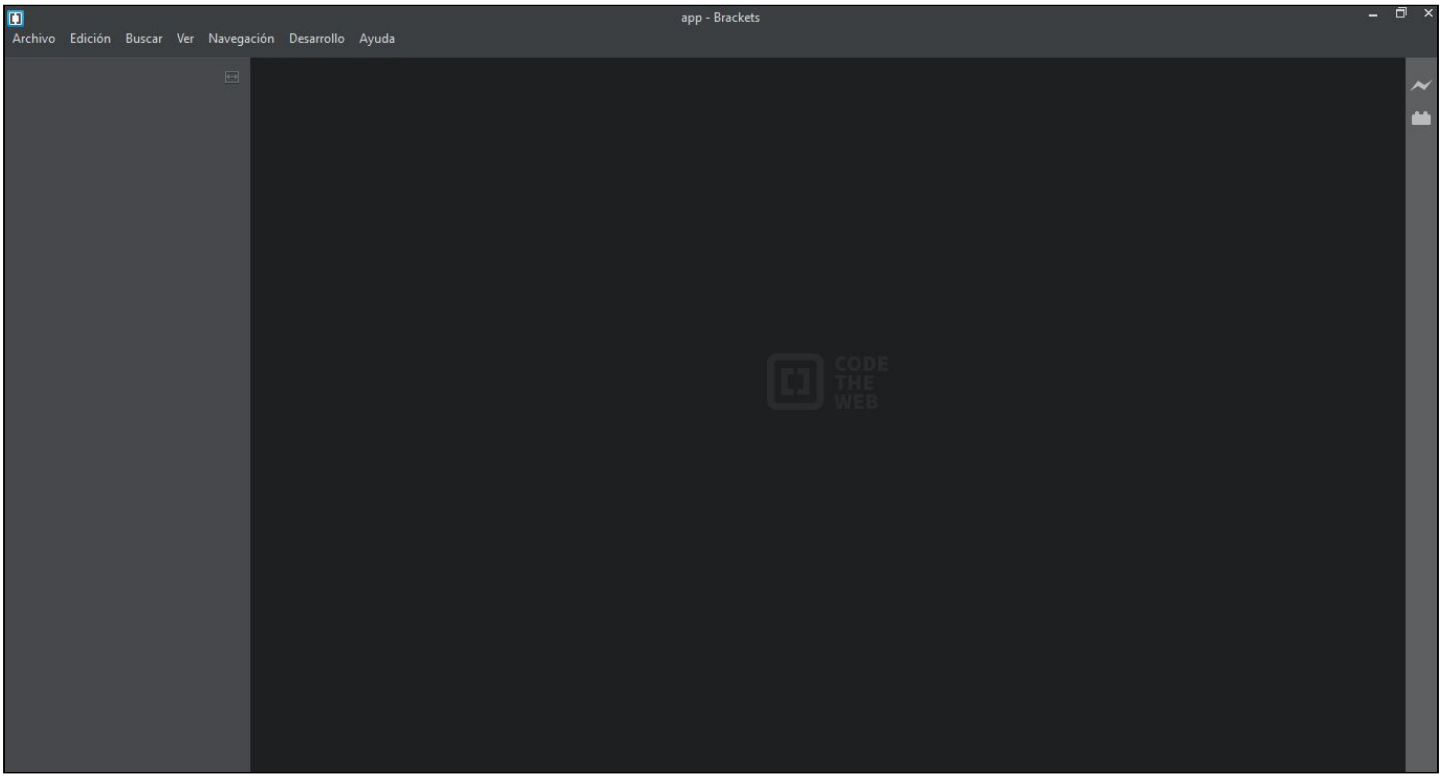


Ilustración 5: Carga Brackets

Una vez descargado e instalado lo anterior, empezamos a crear un nuevo proyecto. Para ello, nos dirigimos a la ruta donde tengamos instalado *XAMPP*, a la carpeta *htdocs*, y borramos el contenido que se ha instalado por defecto:

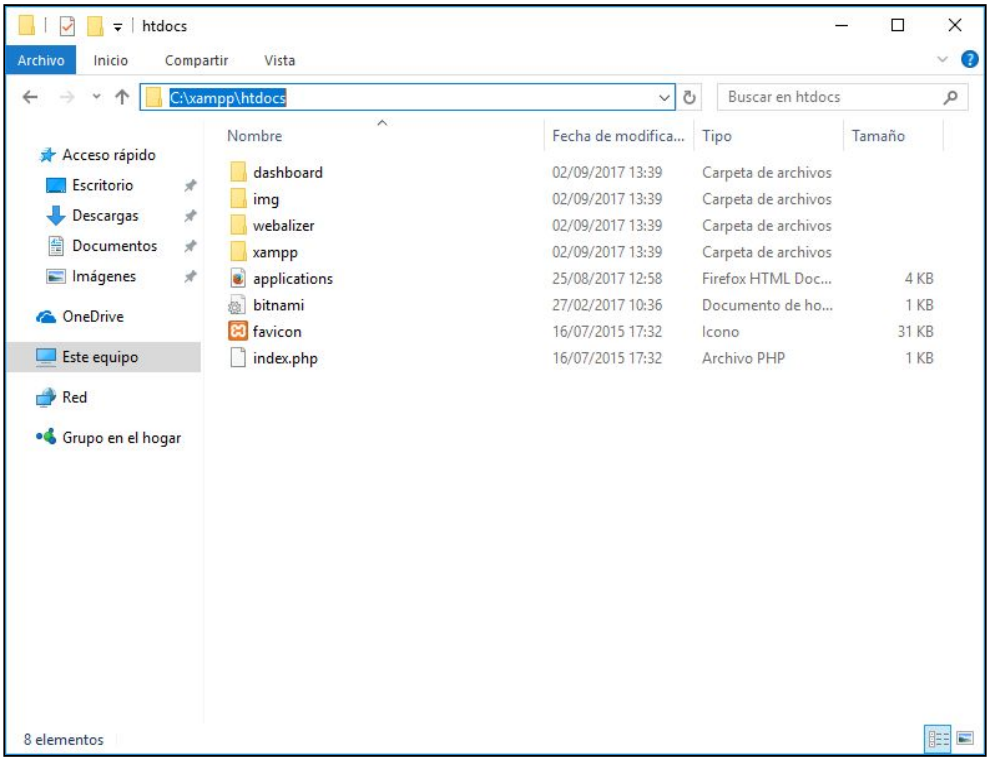


Ilustración 6: Ruta instalación XAMPP + archivos preinstalados

El siguiente paso es crear una carpeta dentro de esta misma ruta y seleccionarla en *Brackets*, que será donde irán ubicados todos los recursos de la aplicación web. En nuestro caso, partimos de una base, en concreto de una semilla de AngularJS (angular-seed) obtenida en una asignatura optativa de 4º curso, ideal si queremos que nuestra aplicación esté basada en el framework *AngularJS*, el cual se documenta en el anexo de la memoria.

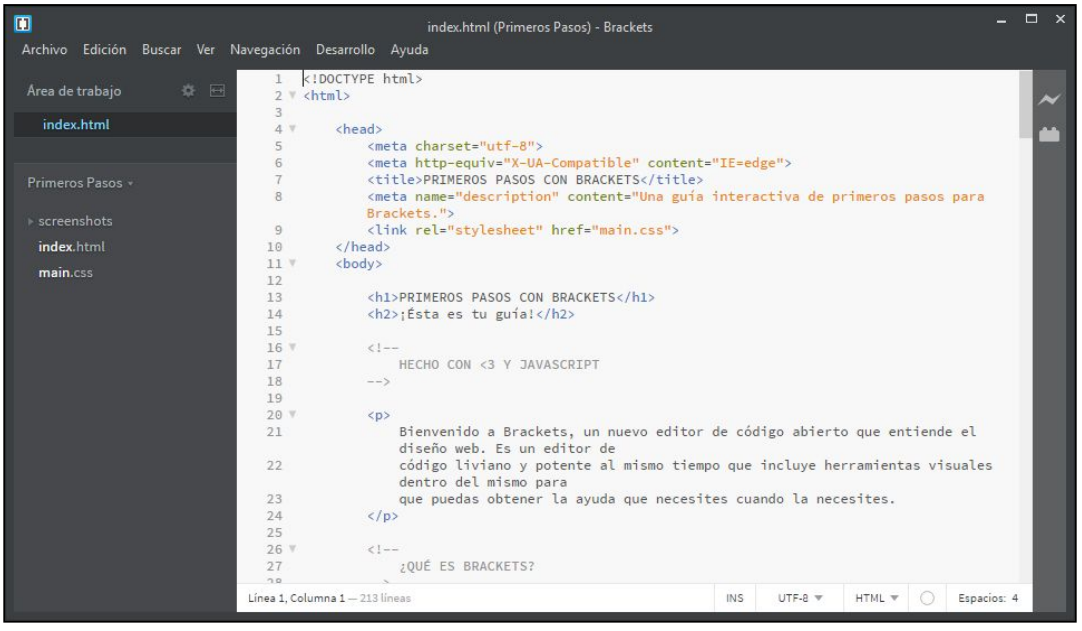


Ilustración 7: Aplicación demo al ejecutar por primera vez Brackets

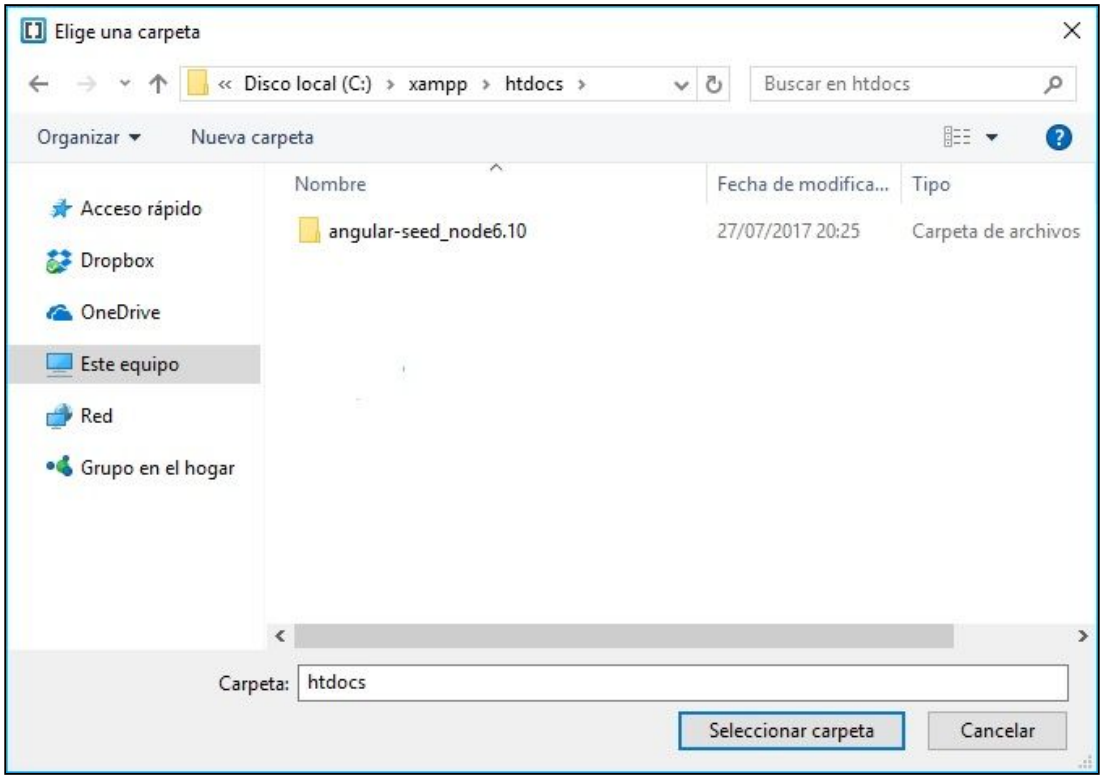


Ilustración 8: Doble clic en la carpeta y clic en "Seleccionar carpeta"

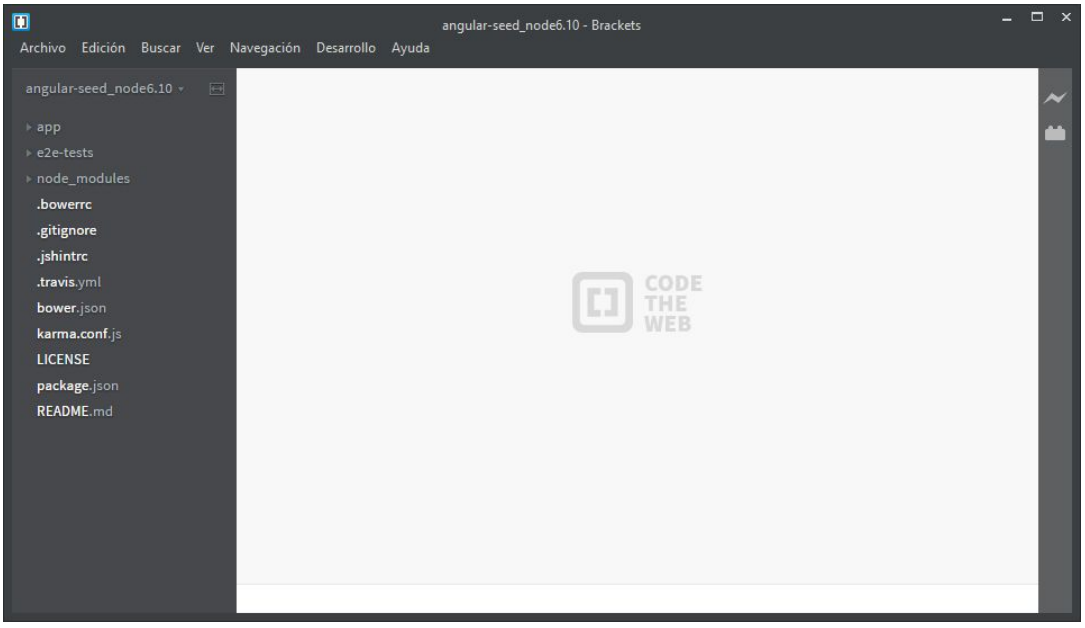


Ilustración 9: Entorno desarrollo preparado para empezar

Una vez todo preparado, se puede empezar con la fase de desarrollo iterativo de la aplicación. Siguiendo una estructura iterativa, por cada "pestaña" (*vista*) que se quiera añadir a la web, se creará una carpeta que contendrá la lógica (*backend*) de la aplicación, así como el contenido de la misma visualizada en plantillas (*frontend*).

Para implementar ese *backend* y *frontend*, esta plataforma se construirá en base al patrón de diseño MVC (Modelo-Vista-Controlador), en el que está basado *AngularJS* y frameworks derivados. Este patrón separa la lógica de la aplicación de la lógica de la vista en una aplicación. La siguiente ilustración explica muy bien la funcionalidad del mismo.

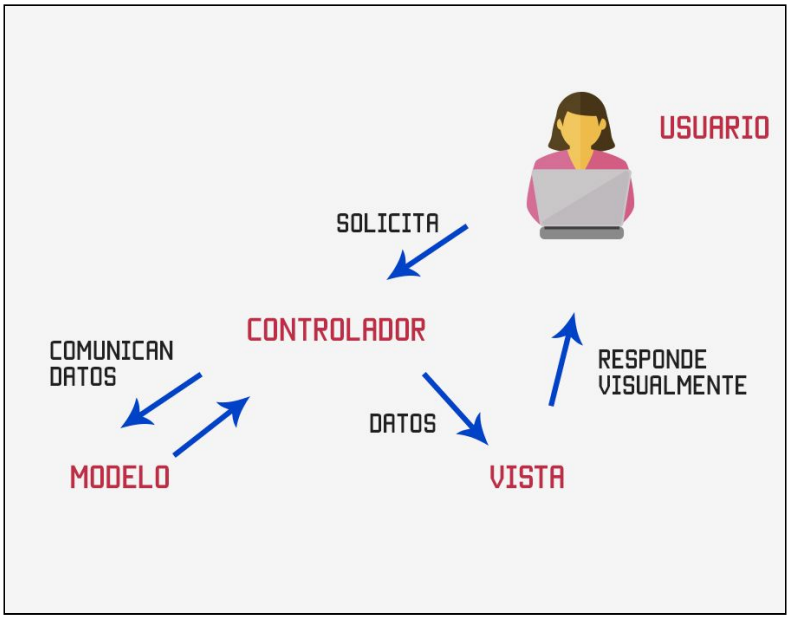


Ilustración 10: Patrón Modelo-Vista-Controlador (MVC)

Pero, ¿qué significan Modelo, Vista y Controlador? Se puede hacer la siguiente analogía:

- Modelo: Es el encargado de consultar a las bases de datos, crearlos, filtrarlos actualizarlos y/o eliminarlos. Como ejemplos podemos poner PHP o NodeJS, de los más utilizados en cuanto a tratamiento de bases de datos.

- Vista: Es lo que el usuario ve en pantalla navegando en la web, es decir, la interfaz gráfica de la aplicación, el cómo se visualizan los datos, siendo ésta una función exclusiva de la vista.

- Controlador: Es el que, a través de peticiones del usuario, se encarga de solicitar los datos al modelo, y una vez obtenidos entregarlos a la vista. Se puede definir como un *intermediario* entre *Modelo* y *Vista*.

Uno de los primeros pasos es decidir las librerías iniciales a utilizar en el servidor para un funcionamiento eficiente, aunque después se irán escogiendo algunas más conforme vaya avanzando el desarrollo. Además, se van a utilizar **componentes funcionales** de una web creada en una asignatura optativa de 4º curso, dedicada al desarrollo de software.

La lista de componentes que va a heredar la plataforma consta de *servicios*, *directivas* y *controladores*. Dentro de los mismos, la plataforma heredará:

- *Directiva* y servicio de logueo y deslogueo de la plataforma con chequeo de estado de la sesión actual.
- *Controladores* de *perfil*, *registro*, *logueo*, *deslogueo* y *home* (inicio de la web).

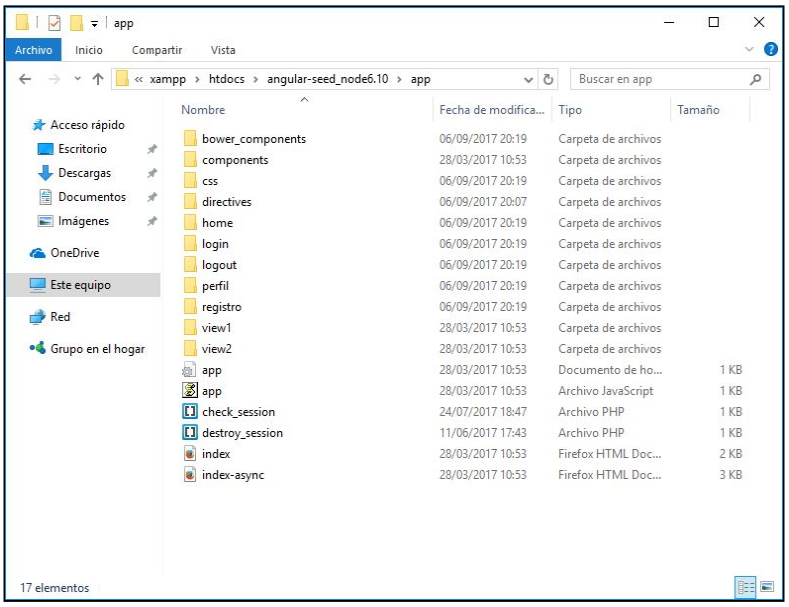


Ilustración 11: Lista de archivos y carpetas tras herencia de componentes

La funcionalidad que tienen los distintos componentes iniciales son:

- *Directives*: tendremos una directiva, encargada de gestionar y controlar la sesión del usuario (logueado / no logueado, gestión permisos,...), ***sessionService***.

- *Home*: contiene la página de inicio de la plataforma. Tiene un componente dinámico que cambia según el estado de la sesión de usuario, el cual se explica más adelante.

- *Login*: lógica y vista para loguearse en la aplicación.

- *Logout*: *enlace* en la barra superior de pestañas para desloguearse de la misma.

- *Perfil*: visualización y modificación de los datos registrados del usuario.

- *Registro*: creación de usuarios, con especial distinción entre *usuarios que consultan* y *administradores*, con datos de interés.

Por otra parte, los archivos externos:

- *app.css*: contiene una parte del estilo de la web. Va complementado con los archivos *css* contenidos en la carpeta *css*, que dan lugar a la plantilla utilizada para la misma.

- *app.js*: **archivo importante**, aquí se indicarán los módulos externos que utilizará la aplicación, así como una gestión global del estado de sesión actual del usuario, para restringir a qué vistas (pestañas) tiene acceso. Los módulos externos se pueden encontrar en las carpetas *bower_components*, *components* y *js*.

- *index.html*: se especificará a la plataforma los recursos que debe cargar en el navegador para que estén disponibles a todos los controladores y vistas de la aplicación. Mediante la directiva *ng-view*, se renderizarán las vistas con sus controladores asociados.

Este es el punto de partida de la plataforma. A continuación, se explica las fases de desarrollo de la misma. **(Nota: a fecha de 16/05/2018)**

En la siguiente ilustración de *Brackets* se observan qué librerías han sido utilizadas:

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.11/angular.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/ngStorage/0.3.10/ngStorage.min.js"></script>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.11/angular-route.min.js"></script>
<script src="components/version/version.js"></script>
<script src="components/version/version-directive.js"></script>
<script src="components/version/interpolate-filter.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.0.3/jquery.min.js"></script>
<script src="https://netdna.bootstrapcdn.com/bootstrap/3.3.4/js/bootstrap.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/ng-dialog/1.4.0/js/ngDialog.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/alasql/0.4.2/alasql.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/angular-xeditable/0.8.0/js/xeditable.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.18.1/moment-with-locales.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/xlsx/0.11.3/xlsx.core.min.js"></script>
```

Ilustración 12: Librerías empleadas

Se puede comprobar que todas están *externalizadas*, es decir, se realizan peticiones *http* a servidores CDN (*Center Delivery Network*) para obtenerlas pero, ¿por qué es mejor que *alojarlas en el servidor*? Existen varias ventajas de cargar los recursos de éstos servidores respecto a alojarlos en el servidor:

- **Mejora de latencia**: estos recursos están distribuidos en servidores por todo el mundo, así que cuando un usuario realice una petición, se hará al servidor más cercano de donde esté, *mejorando la respuesta de la web*.

- **Menor nº de peticiones al servidor propio**: al externalizar los recursos, estamos eliminando peticiones a nuestro servidor. Si multiplicamos el número de usuarios potenciales que visiten la web simultáneamente, se está ahorrando una buena cantidad de ancho de banda.

- **Mejor caché**: relacionada con la anterior, los usuarios no tendrían por qué descargarse la misma librería una y otra vez, ya que estaría *cacheada* del mismo CDN que usan la mayoría de webs, lo que proporciona a las plataformas web de *una buena experiencia de usuario* y *fluidez en la navegación por la misma*.

No obstante, para comprobar qué solución *minimiza el tiempo de carga y ancho de banda consumido*, se realizó una prueba de rendimiento que consistía en cargar la página principal referenciando librerías *en local* y *en servidores CDN*. Para hacerla *más realista*, hemos simulado la carga mediante un perfil de conexión 2G creado en el *Developer Mode* del navegador *Google Chrome*, herramienta que permite *simular* conexiones *reales* en diferentes condiciones.

GPRS	50 kb/s	20 kb/s	500ms
Regular 4G	4.0 Mb/s	3.0 Mb/s	20ms
Regular 2G	250 kb/s	50 kb/s	300ms

Ilustración 13: Distintos perfiles de simulación. Parámetros Descarga / Subida / Latencia

Se han realizado *4 pruebas*:

- Carga de librerías *locales* mediante *conexión doméstica* (cable/WiFi).

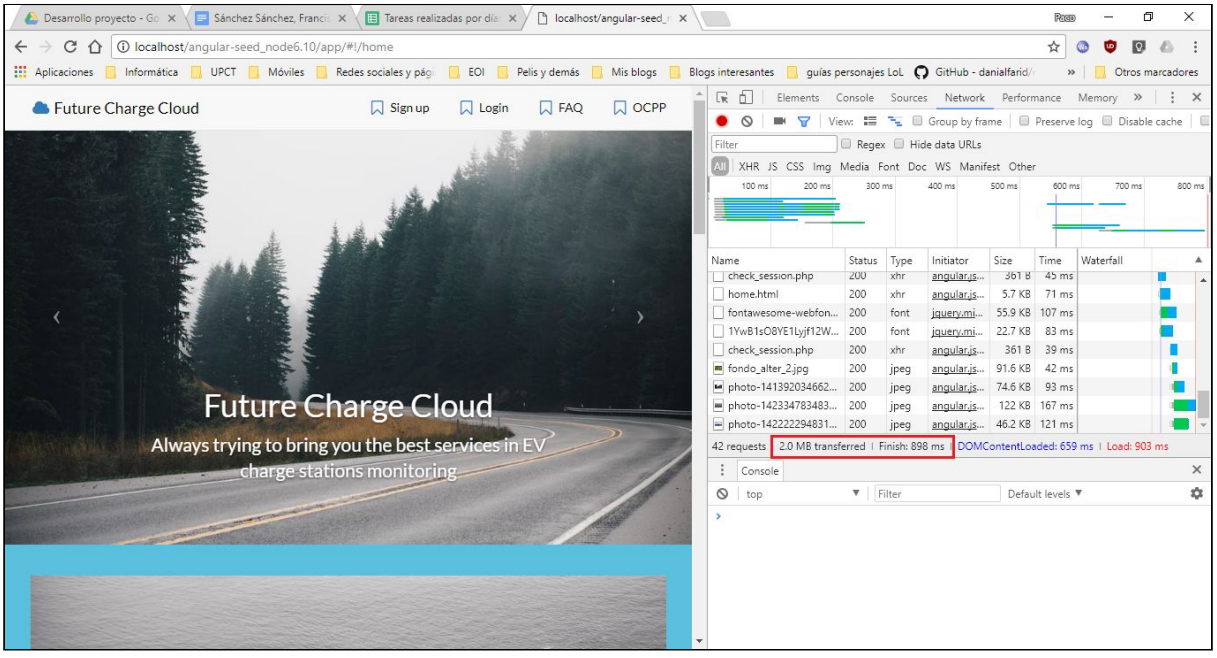


Ilustración 14: Prueba 1. Librerías locales - Conexión WiFi / Ethernet

- Carga de librerías *locales* mediante *conexión 2G simulada*.

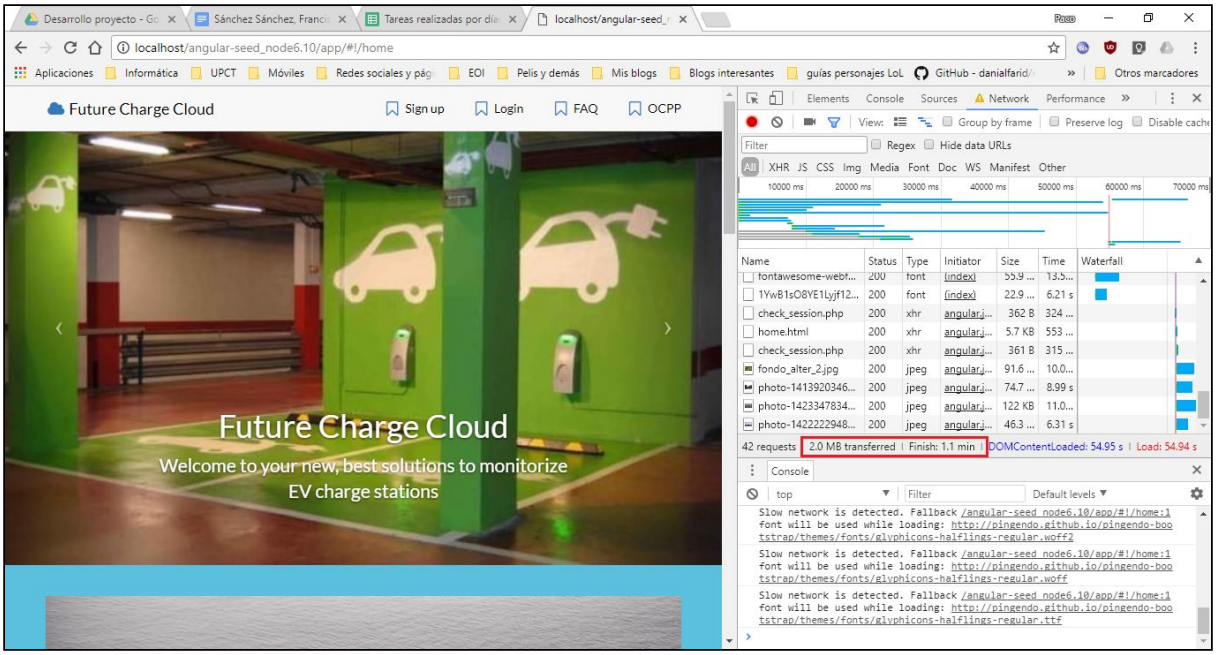


Ilustración 15: Prueba 2. Librerías locales - Conexión 2G simulada

- Carga de librerías desde *servidores CDN* con *conexión doméstica* (cable/Wi-Fi).

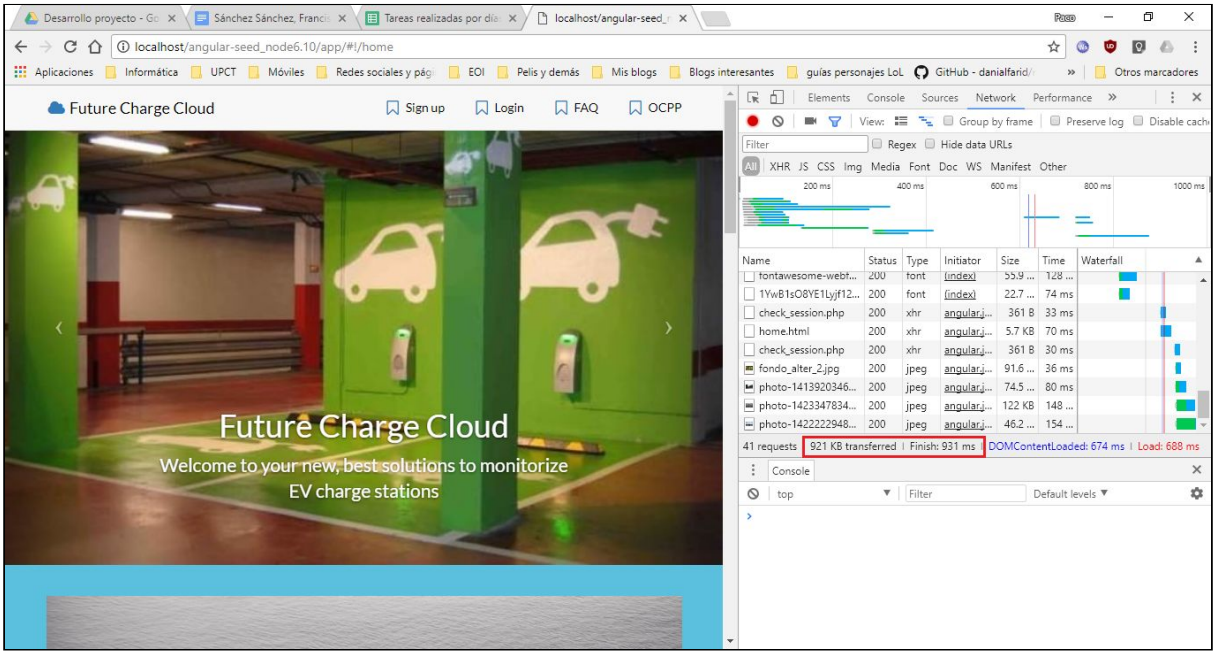


Ilustración 16: Prueba 3. Librerías desde CDN - Conexión WiFi / Ethernet

- Carga de librerías desde *servidores CDN con conexión 2G simulada*.

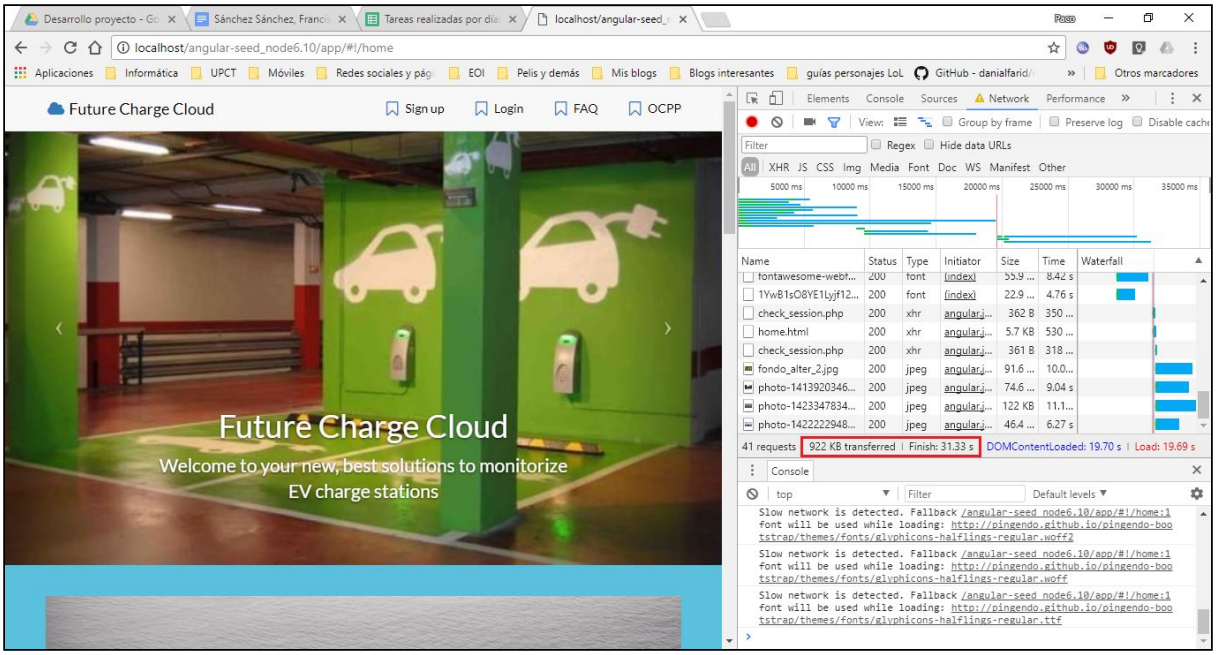


Ilustración 17: Prueba 4: Librerías desde CDN - Conexión 2G simulada

Como se puede observar, se reduce *de forma considerable* la carga de la plataforma si obtenemos las librerías desde *servidores CDN*, ya que se está ahorrando un 50% de ancho de banda en ambos casos, pero en especial en el caso de la conexión 2G, en el que, además, se ahorra *el mismo porcentaje respecto al tiempo de carga gracias a las ventajas que se han comentado anteriormente*.

Se llega a la conclusión de que, para conexiones *reales* y *simuladas,* la web responde **con soltura**, no siendo el tiempo de carga total *elevado* para unas condiciones en las que la conexión a la red es de *baja calidad*, situación que será habitual en la *mayoría de ocasiones*, para que sea ***accesible desde cualquier parte del mundo sin importar la condición o congestión de la red***. Por tanto, la elección será *cargar las librerías desde servidores CDN*.

A continuación, se explica el funcionamiento de cada librería:

- ***angular.min.js***: es la librería *core* del *front-end* de la plataforma. De ella dependen todas las posteriores funcionalidades/librerías *front-end* que se añadan. Se encarga de interpretar sentencias, directivas y servicios basados en este framework de JavaScript.

- ***ngStorage.min.js***: con esta librería, podremos guardar sesiones de usuario y almacenar datos localmente para su posterior uso. Su principal función será *almacenar el estado de la sesión de los usuarios*.

- ***angular-route.min.js***: proporciona servicios de enrutamiento de directivas y servicios. Está formado por varios componentes:

- ***ngView***: esta directiva complementa al servicio *$route* a través de la inclusión de la plantilla renderizada de la ruta actual a la vista principal (*index.html*). Cada vez que la ruta actual cambia, la vista incluida cambia con ella de acuerdo a la configuración del servicio *$route*.

```
<div ng-view></div>
```

Ilustración 18: Directiva ngView

- ***$routeProvider***: el encargado de configurar las rutas.
- ***$route / $routeParams***: el primer servicio trata de enlazar URLs a controladores y vistas. El segundo permite obtener la configuración actual de los parámetros de las rutas.

- ***version.js***, ***version-directive.js***, ***interpolate-filter.js***: sirven para mostrar la versión de la aplicación Angular.

- ***jquery-min.js***: librería Javascript, dependencia de *Bootstrap*, que simplifica la manera de usar JavaScript, siendo más intuitivo y sencillo.

- ***popper-min.js***: librería suministrada por *Pingendo*, un editor online con una larga lista de módulos para diseñar plantillas web, dependiente de *Bootstrap*.

- ***bootstrap-min.js***: librería que contiene la lógica de *Bootstrap* que se combina con la hoja de estilo de la misma (CSS) para funcionar adecuadamente.

- ***ngDialog-min.js***: librería que proporciona ventanas *modales*, mejorando la experiencia de usuario.
- *alasql-min.js*: se utiliza para la exportación de datos a formato CSV, XLS, XLSX y TXT.

- ***xeditable-min.js***: proporciona modificación de formularios *on-the-go*, es decir, no tenemos que abrir otra ventana para modificarlos, simplemente haciendo clic en *Editar*, se podrán modificar los campos incluidos en los mismos.

- ***moment-with-locales-min.js***: librería para mostrar y formatear fechas. En nuestro caso, se utilizará para mostrar fechas en formato *europeo*.

Nota: todas las librerías, a excepción de algunas, se están utilizando en su última versión y todas tienen el atributo *min*, que significa que están *minificadas*, es decir, se han suprimido los saltos de línea, espacios y caracteres derivados para reducir el tamaño de las mismas.

Una vez se ha llegado a este punto, *se mantuvieron conversaciones* con el fabricante de la estación para *obtener información sobre cómo obtener los datos de interés*. Para obtener acceso a dichos datos, el fabricante proporcionó la lista de requisitos para poder realizar *pruebas de conexión y testeo frente a la aplicación*. Los requisitos solicitados fueron:

- Número de serie del cargador.
- URL del servidor en el que se gestionará la lógica del cargador OCPP.
- Puerto asociado a la dirección WebSocket.
- Ruta del cargador dentro del servidor.
- Identificador del cargador.
- Identificador del usuario anónimo a quien registrar las sesiones.

Después de analizar los requisitos, el objetivo es crear una nueva pestaña en la que irá la lógica del cargador, y la dirección WebSocket irá asociada a esta pestaña, a la que después se le harán las respectivas peticiones para mostrar los datos de interés. Esta información proporcionada por Wallbox **ha resultado indispensable** a la hora de **comprender cómo funciona OCPP bajo JSON/Websockets** y establecer la comunicación con **la electrolinera de Los Pinos**.

¿Qué es un WebSocket? El mismo término lo indica: un socket web (canal de comunicación web) por el cual se transporta la información de **punto a punto**:

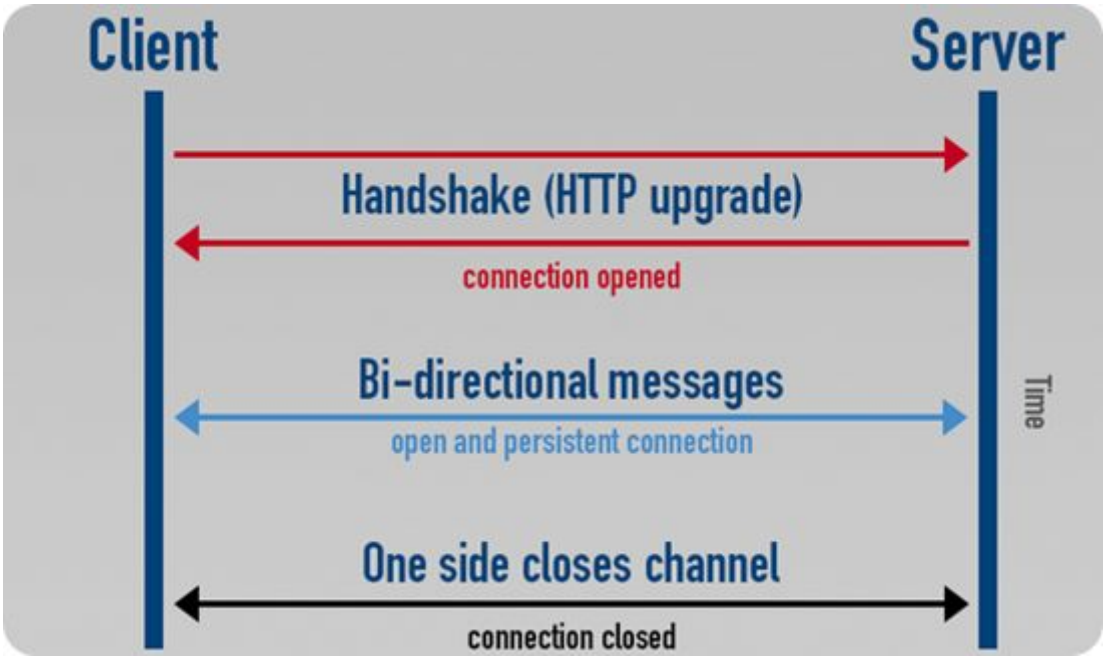


Ilustración 19: Un cliente (electrolinera) se conecta con un servidor web (CPMS)

Un punto positivo de esta tecnología es que la conexión punto a punto **puede acabar cuando un extremo decida cerrar la conexión en cualquier momento por cualquier motivo**. Para mantener la conexión activa, es necesario aplicar un método *keep-alive*, *WebSocket Ping-Pong messages*, o lo que es lo mismo, implementar pequeños mensajes de intercambio **que van en paralelo con la comunicación principal** para evitar que la conexión **se cierre por inactividad**.

La estructura que debe tener la dirección WebSocket de la plataforma es la siguiente. ***NOTA***: es importante utilizar *wss* al principio de la ruta para dotar de seguridad a la conexión websocket (wss, *WebSocket Secure connection*)

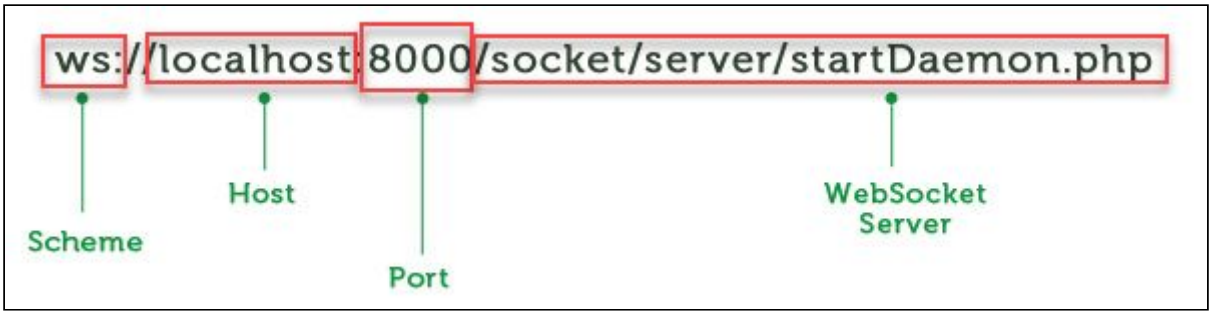


Ilustración 20: Componentes de una URL WebSocket

No obstante, **durante el desarrollo**, se observó que no era posible crear un servidor WebSocket mediante *AngularJS*, ya que este framework es usado **principalmente** para *front-end* y solamente permite conectarse a servidores WebSocket ya existentes, por lo que *se debe crear un servidor que permita combinar tanto front-end como back-end*. En este caso, utilizaremos el framework ***NodeJS***, el cual está orientado al *back-end* **pudiendo combinarlo con otro framework que lleve el *front-end***, como puede ser *AngularJS* o similares.

En definitiva, se debe crear un servidor *back-end* mediante ***NodeJS*** y combinarlo con ***AngularJS*** en el *front-end*. El funcionamiento será como el de hasta ahora, solo que el servidor tendrá un módulo *independiente*, encargado de gestionar todas las peticiones entre una estación con soporte nativo OCPP-JSON versión 1.6 y el servidor WebSocket de la plataforma.

Exportar funcionalidades a NodeJS y bases de datos a MongoDB

NodeJS es una tecnología *del lado del servidor (back-end)* capaz de llevar a cabo tareas *sin intervención del usuario, ya que se ejecutan a nivel interno del servidor*. Permite *dividirlo* en *módulos* que son asignados a *rutas* para ofrecer determinadas funcionalidades y diversidad, así como dotarlo de concepto *modular*.

También se debe *exportar* la estructura de la base de datos *MySQL* a una *NoSQL* para permitir una correcta gestión/integración de los datos en la aplicación. Se utilizará *MongoDB*, un gestor de base de datos *NoSQL*, idéntico a *MySQL* con la diferencia de que Mongo no tiene interfaz gráfica para gestionar las BBDD. Como solución, se incorporará un programa que proporciona una GUI con terminal integrada para gestionarlas, llamado *Robo 3T*.

El objetivo ahora es convertir la aplicación en una *MEAN stack application*:

- **M**ongoDB: gestor de bases de datos.
- **E**xpressJS: gestor de módulos, dependencias y rutas.
- **A**ngularJS: *front-end* de la plataforma.
- **N**odeJS: *back-end* de la plataforma que combinará los 3 puntos anteriores.

Las tablas de las que consta la aplicación *MEAN* son las siguientes

- Usuarios: para gestionar los usuarios registrados en la aplicación.
- Estaciones: para gestionar las estaciones que se suscriban a la aplicación.
- Telemetría: **la tabla principal**. En esta tabla se irán almacenando todos los registros de telemetría que provenga de las estación o estaciones.
- Transacciones: se registrarán las transacciones que realicen los usuarios que carguen su vehículo para el posterior cobro del operario de la misma.

En las tablas de base de datos indicadas, **se guardará toda la información** de la plataforma:

- En la tabla *Usuarios*, los usuarios/administradores que se registren en la plataforma (nombre de usuario, contraseña (almacenada previamente pasando por un algoritmo de cifrado), nombre, apellidos, edad, sexo y rol del usuario (user/admin).
- En la tabla *Estaciones*, las estaciones registradas, así como los puntos de recarga (nombre, dirección, localidad (y contraseña maestra si el usuario es un administrador, pasando también previamente por un algoritmo de cifrado).
- En la tabla *Telemetría,* las sesiones de telemetría que se reciban desde la estación de recarga. Esta y la anterior servirán para mostrar los datos especificados por *Solvent* en pantalla en tiempo real ***en el mismo instante en que estén disponibles***.
- En la tabla *Transacciones*, las transacciones que se produzcan entre la misma y el usuario que realice una carga a su vehículo (también registradas)
- Adicionalmente, una tabla *ChargeBill* (facturación), en la que se almacena el precio del kWh para su posterior cobro por parte del operario administrador.

Lo siguiente será definir las rutas de la aplicación y accesos a las mismas a través de **middlewares** implementados con NodeJS. Posteriormente, se le asignará un controlador a cada ruta con su correspondiente funcionalidad. Por último, mediante AngularJS, se creará la lógica de las vistas que verá el usuario final.

Comenzamos con las *pruebas iniciales.* Se contactó con Wallbox para comenzar las pruebas básicas de conexión OCPP.

Ilustración 21: Prueba conexiones OCPP con Wallbox

El resultado es el esperado, ya que se ha recibido una solicitud de arranque del cargador (*BootNotification*), el servidor WebSocket la recibe, *codifica el mensaje para comprobar qué es cada campo* y se le emite una respuesta. *Éste es el procedimiento* que se ha seguido en cada par de mensajes solicitud-respuesta.

¿Qué acciones parten una ER (Estación de Recarga) y cuáles del sistema central? Es una analogía para cualquier sistema que se implemente. Son las siguientes:

***Acciones emitidas por el cargador (CP, Charge Point)*:**

- **Authorize**: Cuando un usuario va a comenzar una sesión de carga, previamente se consulta el identificador (ya sea proporcionado por tarjeta RFID, NFC o derivados) en la base de datos para comprobar si está autorizado a cargar.

- **BootNotification**: *el cargador* envía al sistema un mensaje indicando que acaba de arrancar, indicando sus parámetros principales, tales como *fabricante, modelo* y derivados.

- **StartTransaction**: mensaje que indica el comienzo de una transacción.

- **StopTransaction**: análogo al anterior y siguiente, el parámetro *meterStop* indica los kWh que se han suministrado al vehículo del usuario.

- **MeterValues**: *probablemente el mensaje de mayor interés*. Este mensaje contiene la sesión de carga realizada en el cargador. Consta de fecha inicio, valores iniciales, fecha final y valores finales, acompañado del identificador del conector con el que se ha realizado dicha carga y un identificador de la carga (transacción).

- **Heartbeat**: Este mensaje es enviado cada X minutos para indicarle al sistema que el cargador sigue operativo. Lo normal es configurarlo para que se envíe 1-2 veces por hora.

En cuanto a las *acciones* que puede llevar a cabo el *sistema central*, de las distintas opciones que existen, las más resaltables para este caso de estudio son:

- **UnlockConnector**: se le indica al punto de recarga el conector que debe desbloquear. Este mensaje es el que seguiría a *Authorize*, ya que primero se comprueba autorización, y si es OK lo siguiente es desbloquear el conector en el que se va a realizar la carga.

- **Reset**: se le envía un mensaje de *reseteo* al punto de recarga. Puede ser *Soft* (suave, reinicio simple) o bien *Hard* (duro, reseteo a los valores de fábrica). Se suele utilizar cuando la estación no responde como debería, o lo que es lo mismo, existe un *problema de software*, y dependiendo de la dimensión del mismo se utilizaría una u otra opción.

El resultado de lo anteriormente comentado, a fecha 16/05/2018, es el siguiente:

Ilustración 22: Home (1)

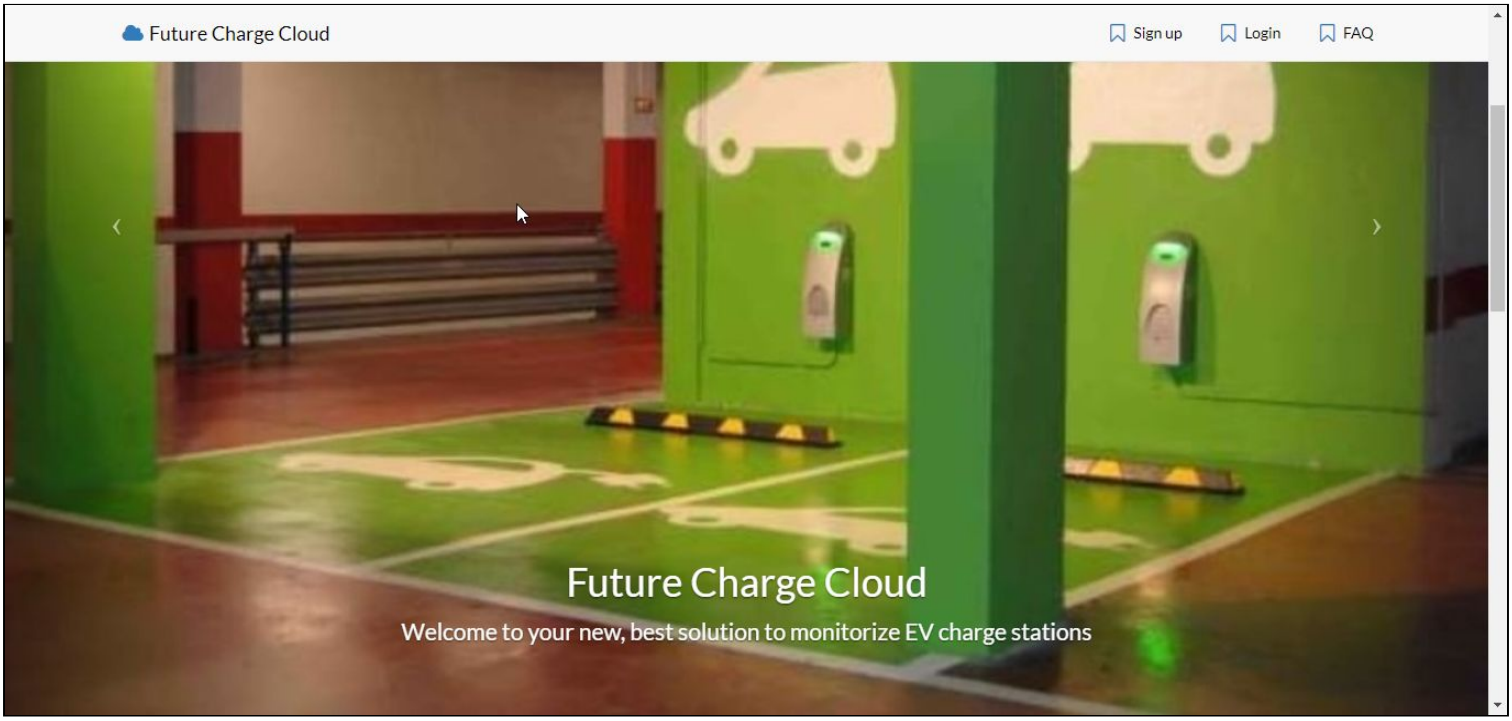


Ilustración 23: Home (2)

Las dos anteriores ilustraciones muestran la pantalla principal que ve el usuario nada más introducir la dirección web del portal. *Una buena página principal es una buena carta de presentación*, es por ello que se ha diseñado de forma minimalista e intuitiva. *Ésta será la tónica en cada pestaña del portal*.

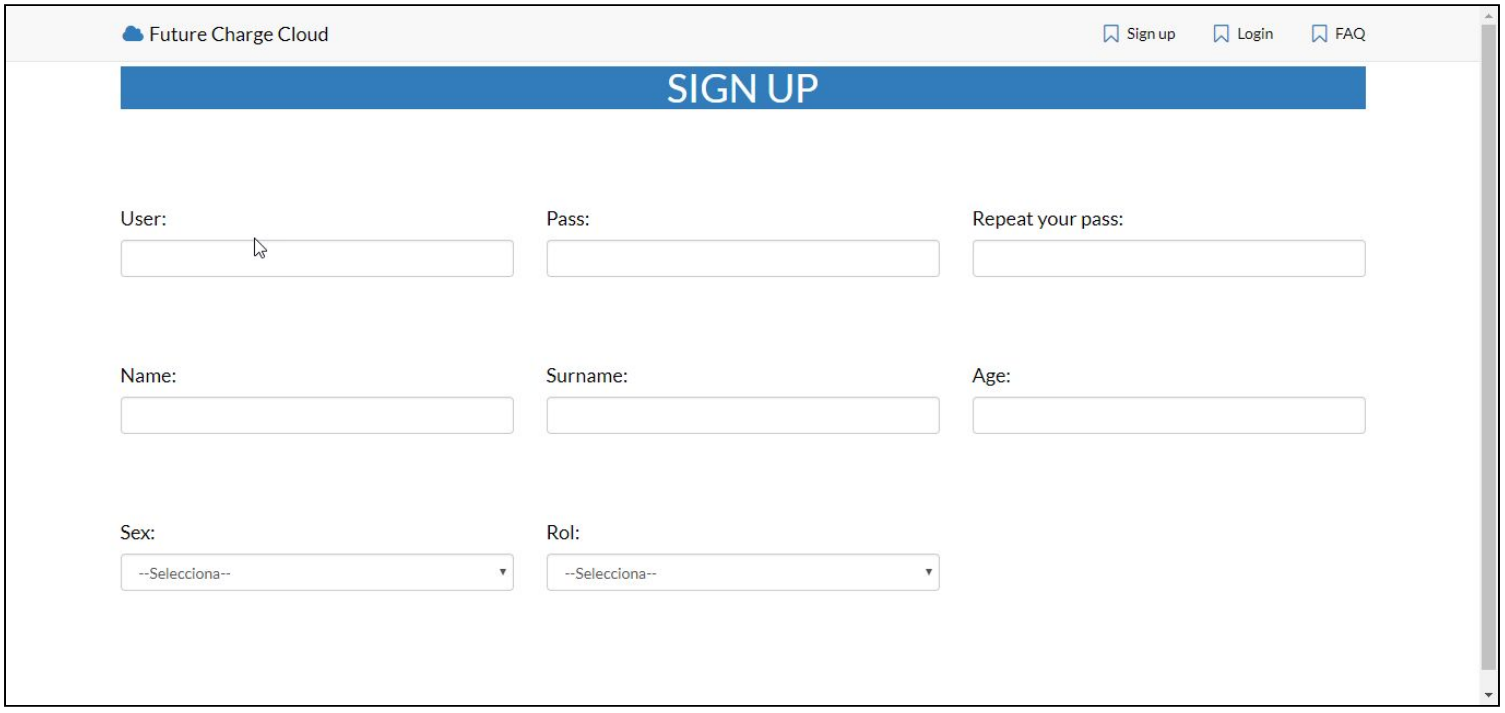


Ilustración 24: Sign up

En esta pestaña, el usuario puede registrarse y elegir el rol que desempeña, esto es, si va a ser administrador de una o varias electrolineras, tendrá que escoger la opción *Admin*. Sin embargo, solamente realizará consultas sobre datos de telemetría o derivados, la opción más adecuada es *User*.

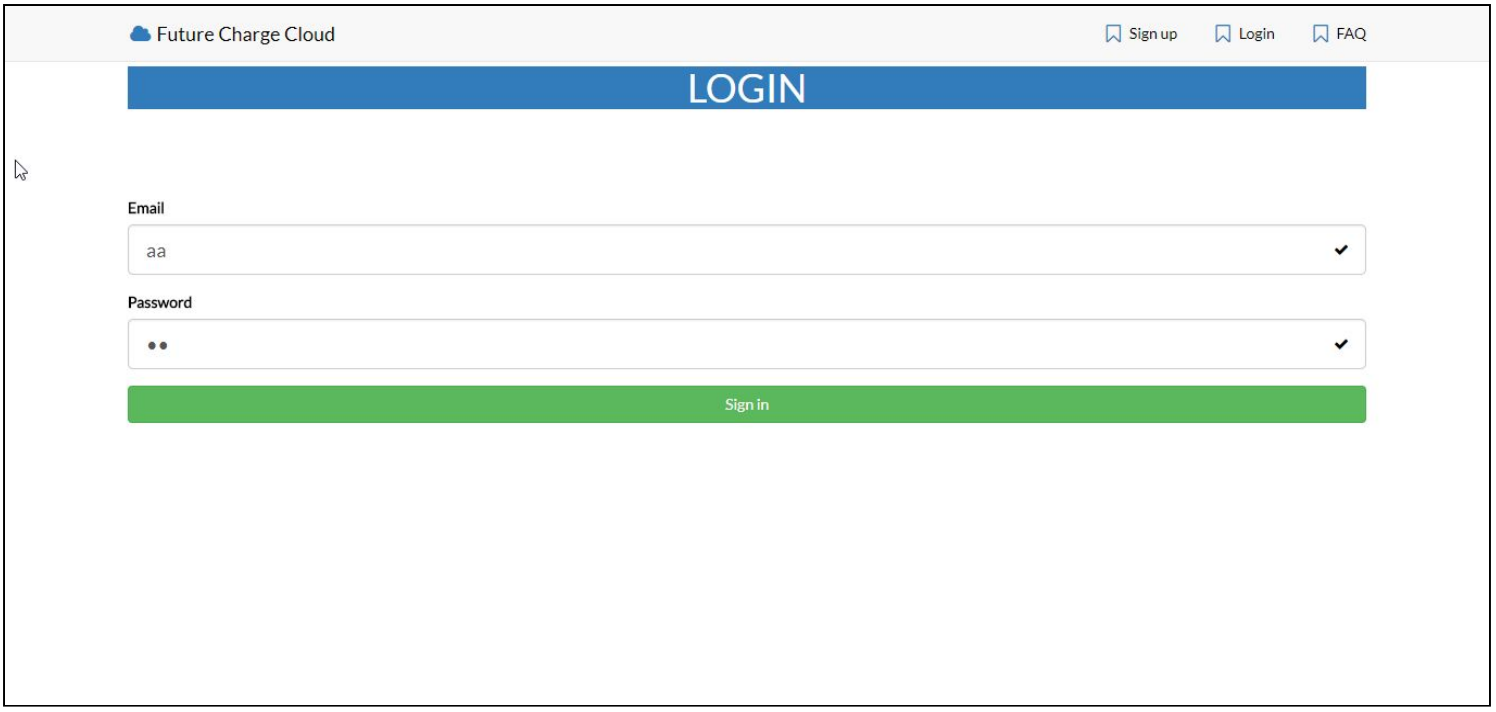


Ilustración 25: Login

Tras haberse registrado, el usuario puede dirigirse a la pestaña *Login* para loguearse en la plataforma y acceder a determinadas pestañas. Según el rol del usuario al loguearse, verá las siguientes pantallas:

- User: **consultas y perfil**.
- Admin: **todas las disponibles**.

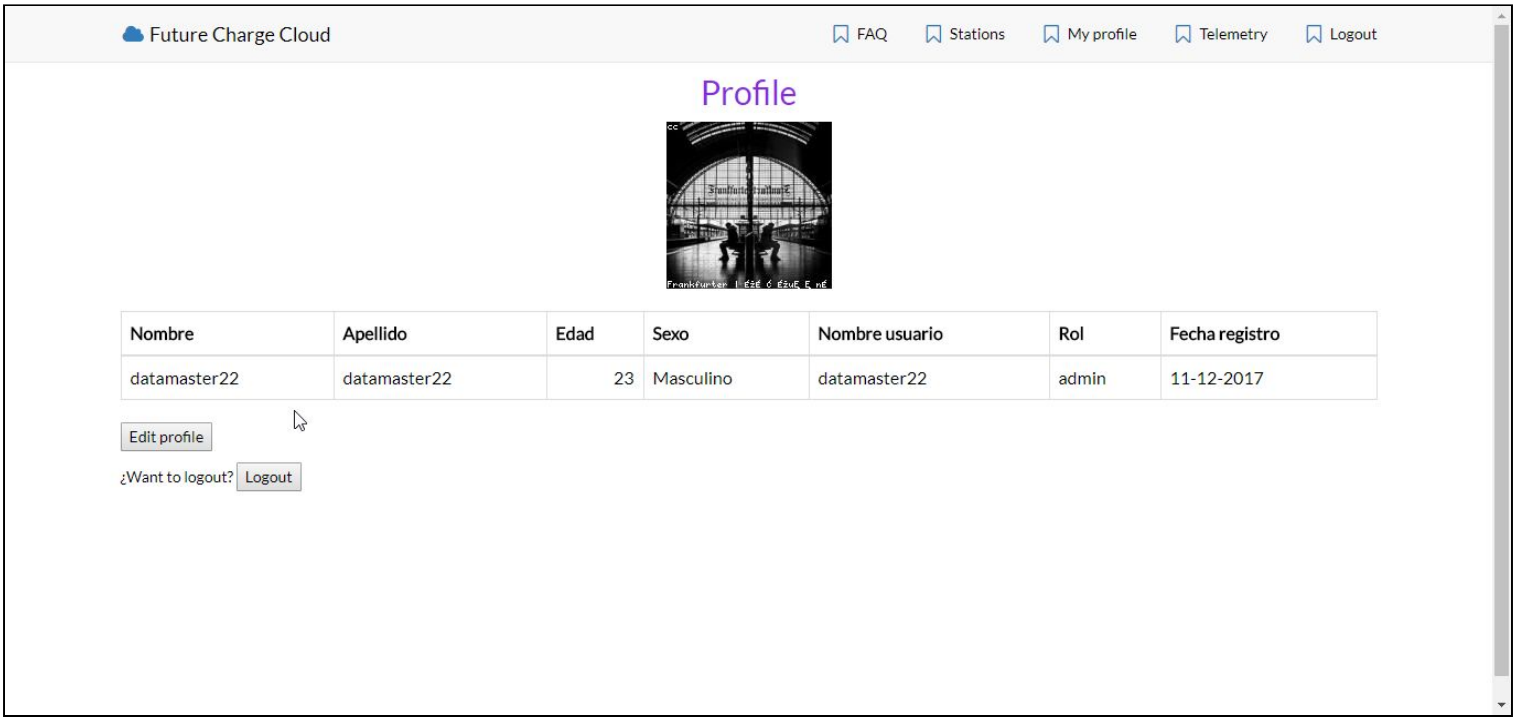


Ilustración 26: Perfil rol administrador

En esta pestaña, el usuario puede visualizar sus datos personales con posibilidad de modificarlos. En la barra superior disponemos de las pantallas *accesibles* para el usuario *en función* de su rol. Al ser administrador, tiene acceso a todas las disponibles.

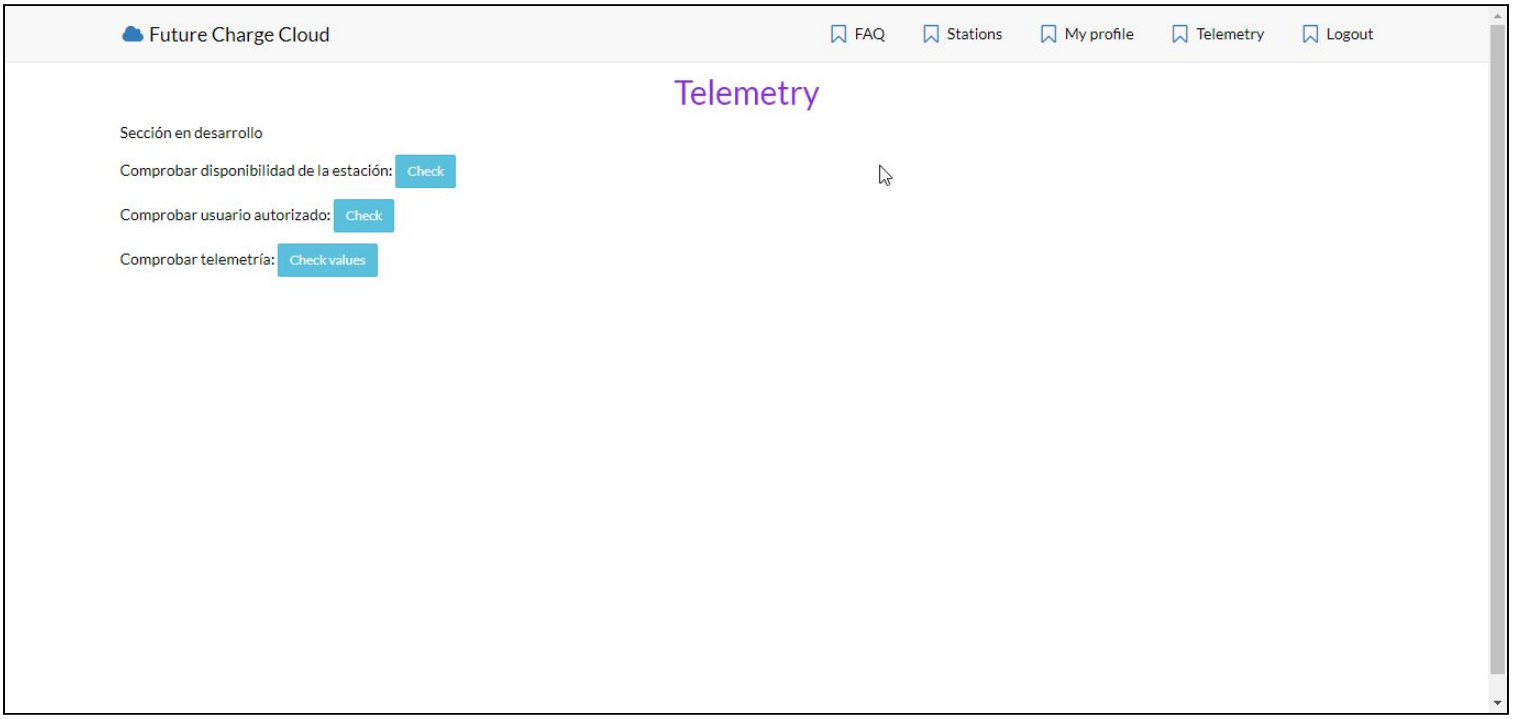


Ilustración 27: Telemetría

Una de las pestañas de más interés. En ella, el administrador podrá visualizar las últimas cargas o últimas sesiones de recargas *a tiempo real* que se produzcan en la electrolinera. ***NOTA***: El primer botón, *Check*, permite comprobar si existe comunicación con la electrolinera, sin embargo, no está desarrollado, ya que el mensaje OCPP para que esto sea posible es *TriggerMessage* con la operación *Heartbeat*, el cual no está implementado en la mayor, si no en la totalidad de la electrolineras comercializadas.

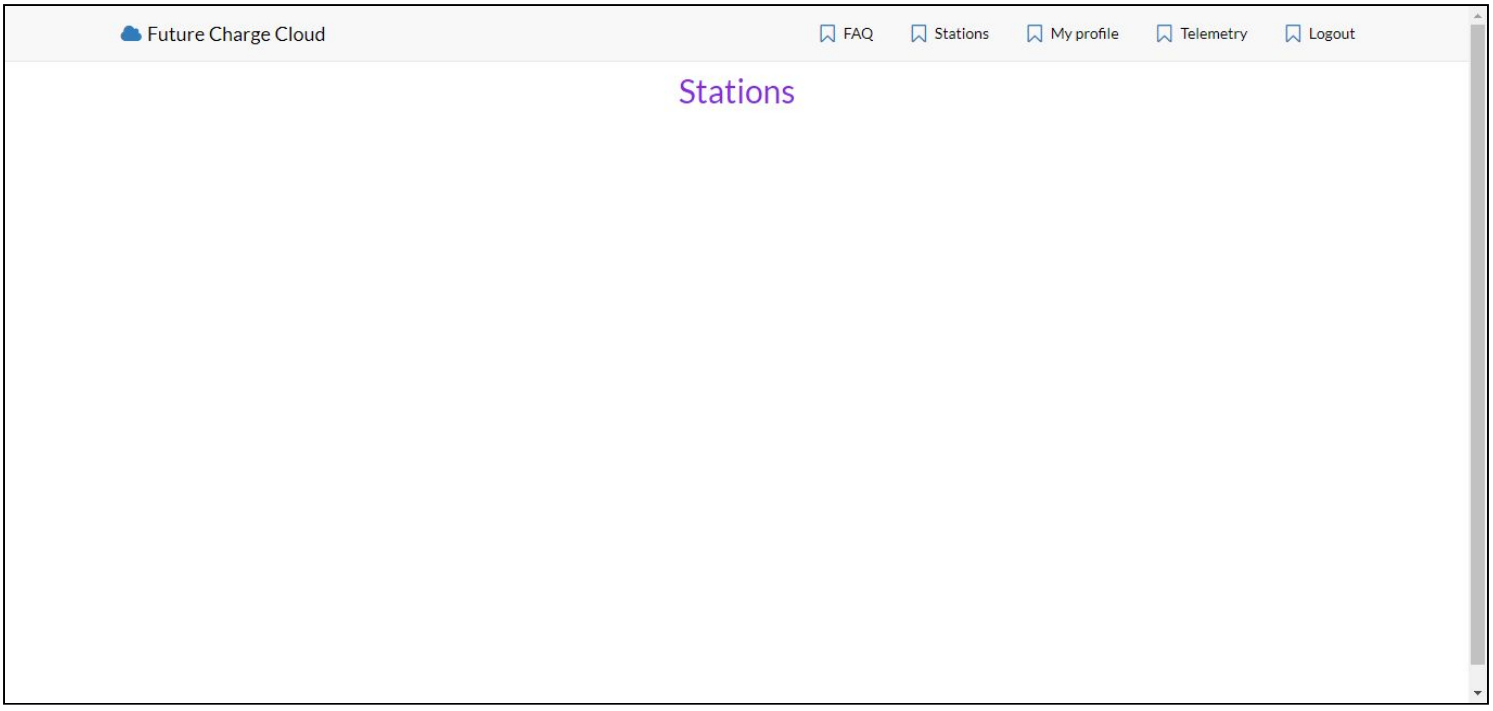


Ilustración 28: Estaciones

En esta pestaña, el administrador podría añadir las estaciones de recarga que necesite, así como modificar los datos de las mismas (nombre, localidad y dirección). Futuramente, se desarrollaría un panel de administración para gestionar parámetros internos de las mismas.

Ilustración 29: FAQ

Sección informativa con las preguntas más frecuentes acerca de la plataforma, tales como el registro, modificación de datos personales o visualización de datos de telemetría. Por el momento no se ha desarrollado, ya que la **navegación por la plataforma es intuitiva y sencilla**, cada sección está diseñada de forma que sea fácil navegar por las mismas y quede la menor duda.

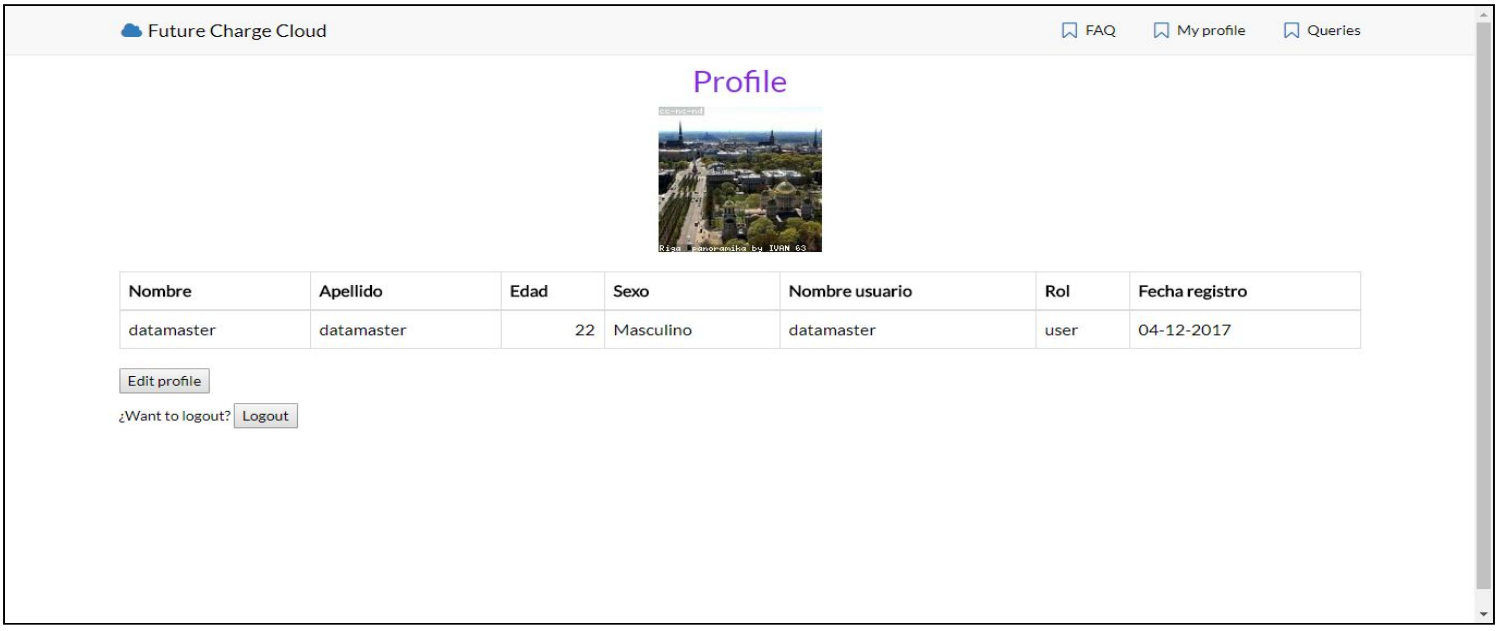


Ilustración 30: Perfil rol usuario

Vemos que el panel de perfil de usuario es prácticamente igual que al del administrador, pero con ligera diferencia en la barra superior de pestañas. El usuario tendrá acceso *únicamente* a realizar consultas de telemetría.

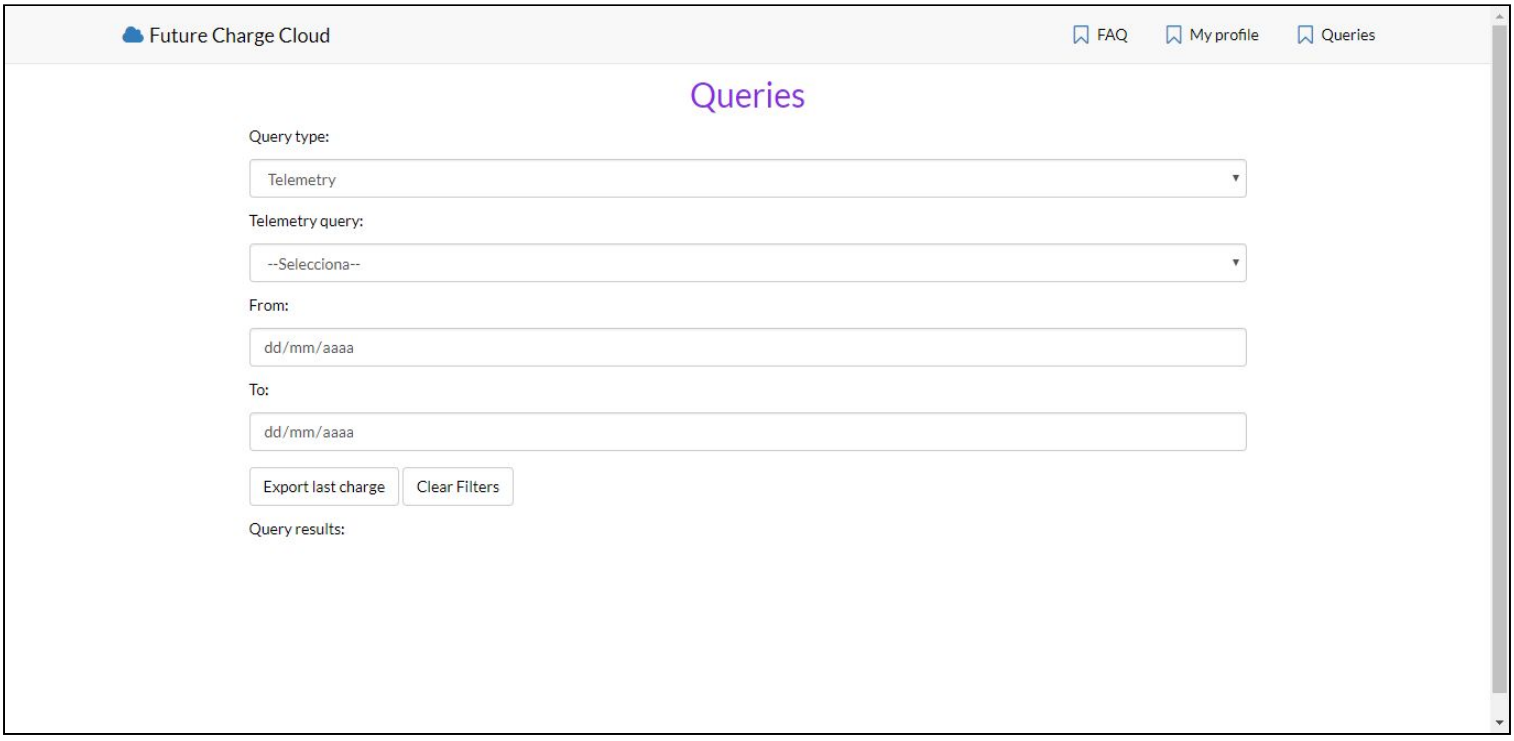


Ilustración 31: Consultas

Panel a través del cual tanto administrador como usuario pueden realizar consultas acerca de la estación que necesite. Además de realizar una consulta general, se pueden realizar consultas a parámetros específicos (voltaje, potencia ofrecida, batería al comienzo y final de la carga, etc…). Las siguientes capturas muestran cómo debe hacerse una consulta y su posterior descarga (opcional) de la propia consulta en formato Excel:

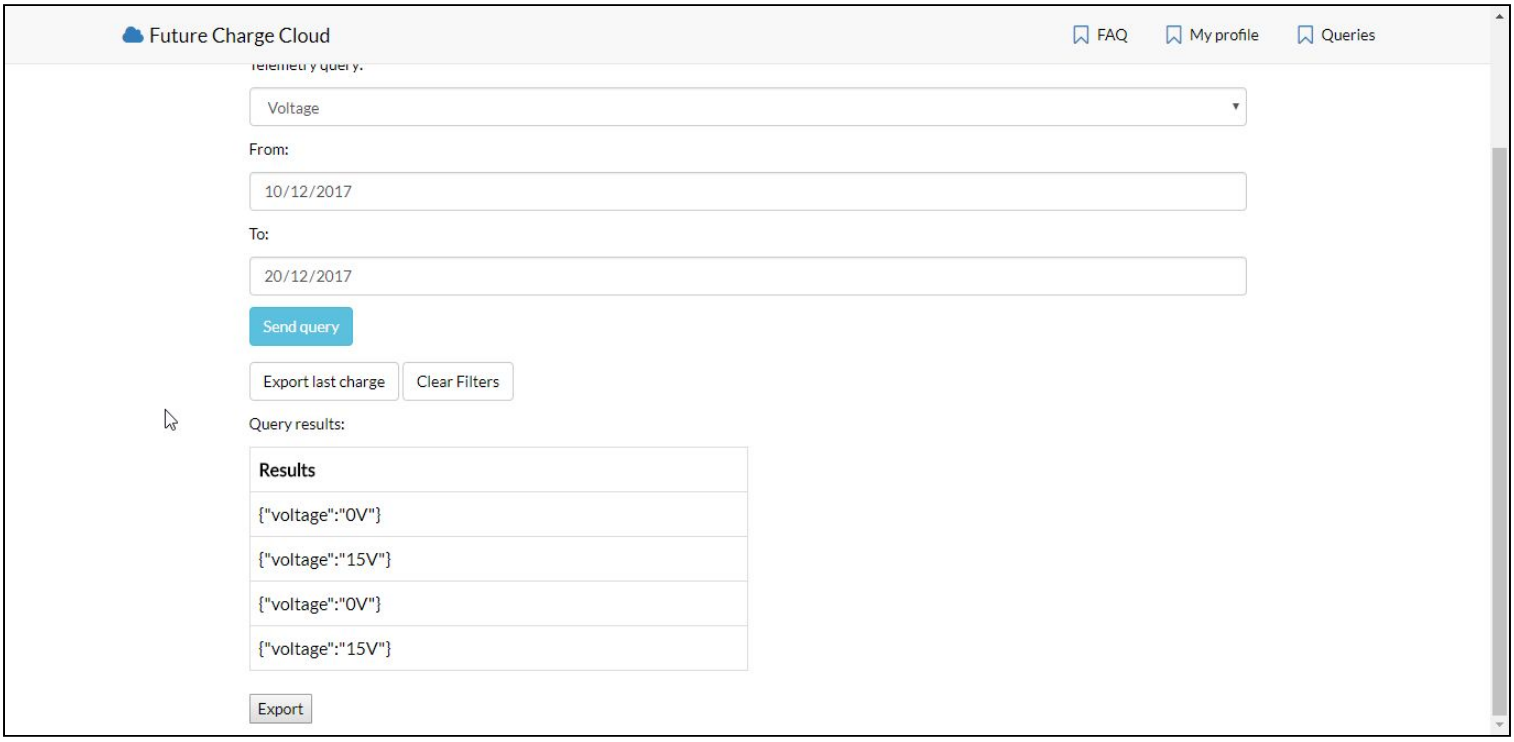


Ilustración 32: Ejemplo de consulta

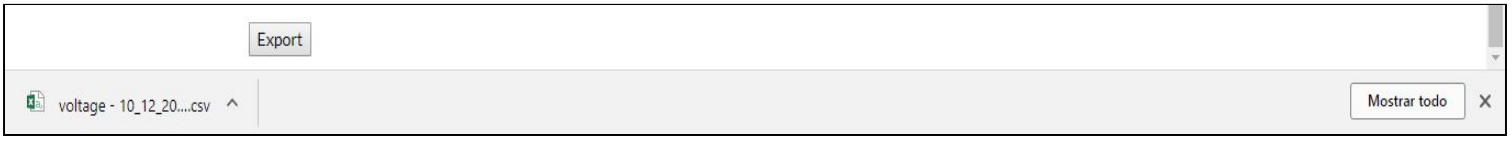


Ilustración 33: Consulta exportada

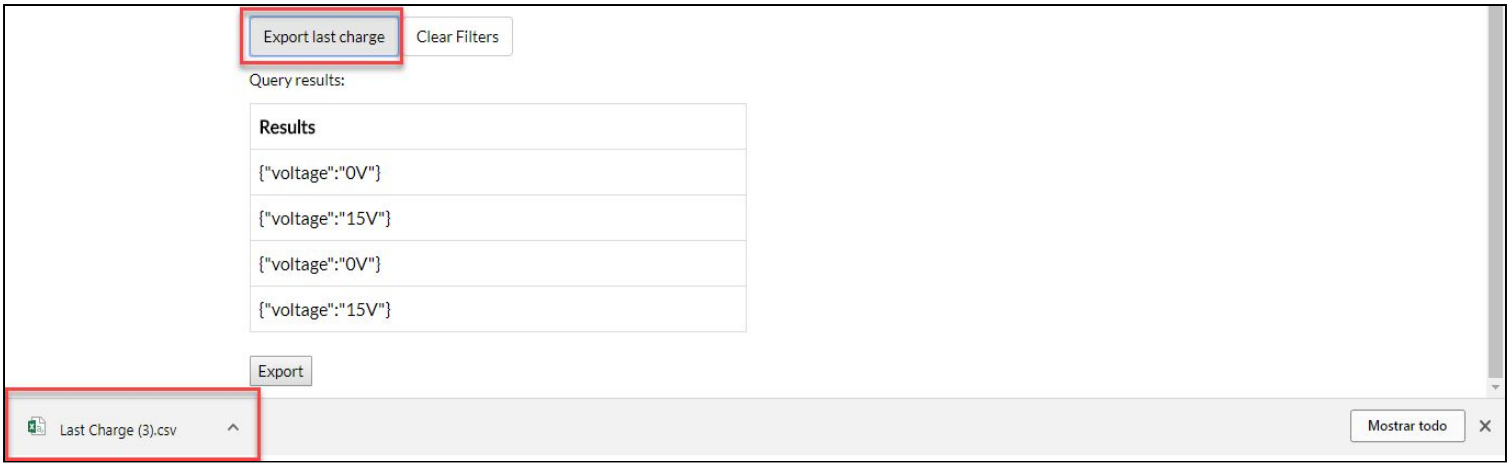


Ilustración 34: Última consulta exportada

Primero seleccionamos el parámetro o parámetros (consulta general), lo siguiente es elegir el rango de fechas de la consulta y por último ***Send query***. La consulta aparecerá en formato JSON para facilitar la lectura de los valores. En la parte inferior, se habilitará un botón para dar opción de descargar la consulta. Así mismo, existe una opción para descargar la última carga registrada.

5. Pruebas y conclusiones

5.1. Pruebas de validación

Una vez desarrollada la aplicación objetivo de este proyecto, se comenzaron a realizar una serie de pruebas para validar su correcto funcionamiento:

Primera prueba

El programa se ejecuta desde un ordenador de la universidad conectado a la red (proporcionado por el director del proyecto), emulando el comportamiento de un sistema central, es decir, de un sistema gestor de estaciones de recarga, y desde el PC de desarrollo realizar una conexión vía WebSocket al endpoint Websocket definido en el sistema gestor, emulando el comportamiento de una estación de recarga.

Para validar el correcto funcionamiento de los mensajes, se han utilizado los siguientes programas:

- **Wireshark**: se captura el tráfico intercambiado de la conexión entre el pc de desarrollo y el ordenador que actúa de sistema central.
- **Snagit**: Programa para realizar capturas de pantalla detalladas.
- **SimpleWebSocketClient**: extensión del navegador Google Chrome que permite realizar conexiones a servidores WebSocket.

Primeramente, se establece la conexión Websocket entre el gestor y la estación de recarga. Para comprobar que la conexión es correcta, mediante el comando *netstat*:

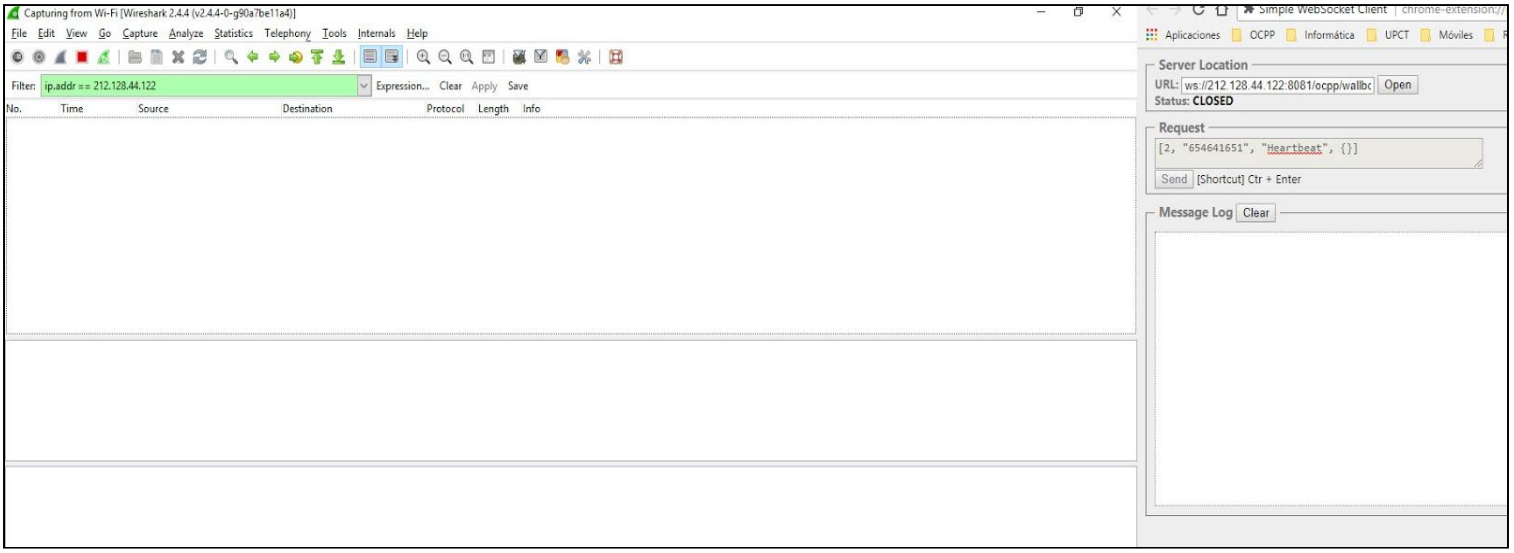


Ilustración 35: Estado inicial

```
C:\Users\psanc>netstat

Conexiones activas

  Proto  Dirección local        Dirección remota       Estado
  TCP    127.0.0.1:49984        activation:49986       ESTABLISHED
  TCP    127.0.0.1:49984        activation:49988       ESTABLISHED
  TCP    127.0.0.1:49984        activation:49990       ESTABLISHED
  TCP    127.0.0.1:49984        activation:49992       ESTABLISHED
  TCP    127.0.0.1:49984        activation:49994       ESTABLISHED
  TCP    127.0.0.1:49984        activation:49996       ESTABLISHED
  TCP    127.0.0.1:49984        activation:49998       ESTABLISHED
  TCP    127.0.0.1:49984        activation:50005       ESTABLISHED
  TCP    127.0.0.1:49984        activation:50011       ESTABLISHED
  TCP    127.0.0.1:49984        activation:50013       ESTABLISHED
  TCP    127.0.0.1:49984        activation:50021       ESTABLISHED
  TCP    127.0.0.1:49986        activation:49984       ESTABLISHED
  TCP    127.0.0.1:49988        activation:49984       ESTABLISHED
  TCP    127.0.0.1:49990        activation:49984       ESTABLISHED
  TCP    127.0.0.1:49992        activation:49984       ESTABLISHED
  TCP    127.0.0.1:49994        activation:49984       ESTABLISHED
  TCP    127.0.0.1:49996        activation:49984       ESTABLISHED
  TCP    127.0.0.1:49998        activation:49984       ESTABLISHED
  TCP    127.0.0.1:50005        activation:49984       ESTABLISHED
  TCP    127.0.0.1:50011        activation:49984       ESTABLISHED
  TCP    127.0.0.1:50013        activation:49984       ESTABLISHED
  TCP    127.0.0.1:50021        activation:49984       ESTABLISHED
  TCP    127.0.0.1:50364        activation:50365       ESTABLISHED
  TCP    127.0.0.1:50365        activation:50364       ESTABLISHED
  TCP    192.168.1.134:50360    wl-in-f188:https       ESTABLISHED
  TCP    192.168.1.134:50367    hk2sch130021046:https  ESTABLISHED
  TCP    192.168.1.134:50486    ad:https               ESTABLISHED
  TCP    192.168.1.134:50687    ec2-52-212-187-118:13936  ESTABLISHED
  TCP    192.168.1.134:50927    pcamartinez:8081       ESTABLISHED
  TCP    192.168.1.134:50989    52.109.88.44:https     TIME_WAIT
  TCP    192.168.1.134:50995    mad06s25-in-f131:https  ESTABLISHED
  TCP    192.168.1.134:51011    2.20.88.97:http        TIME_WAIT
  TCP    192.168.1.134:51012    a92-123-73-11:http     TIME_WAIT
  TCP    192.168.1.134:51074    a-0001:https           ESTABLISHED
  TCP    192.168.1.134:51079    13.107.42.254:https    ESTABLISHED
  TCP    192.168.1.134:51080    93.184.221.200:https   ESTABLISHED
  TCP    192.168.1.134:51081    13.107.6.254:https     ESTABLISHED
```

Ilustración 36: Conexión TCP con el SC emulado exitosa

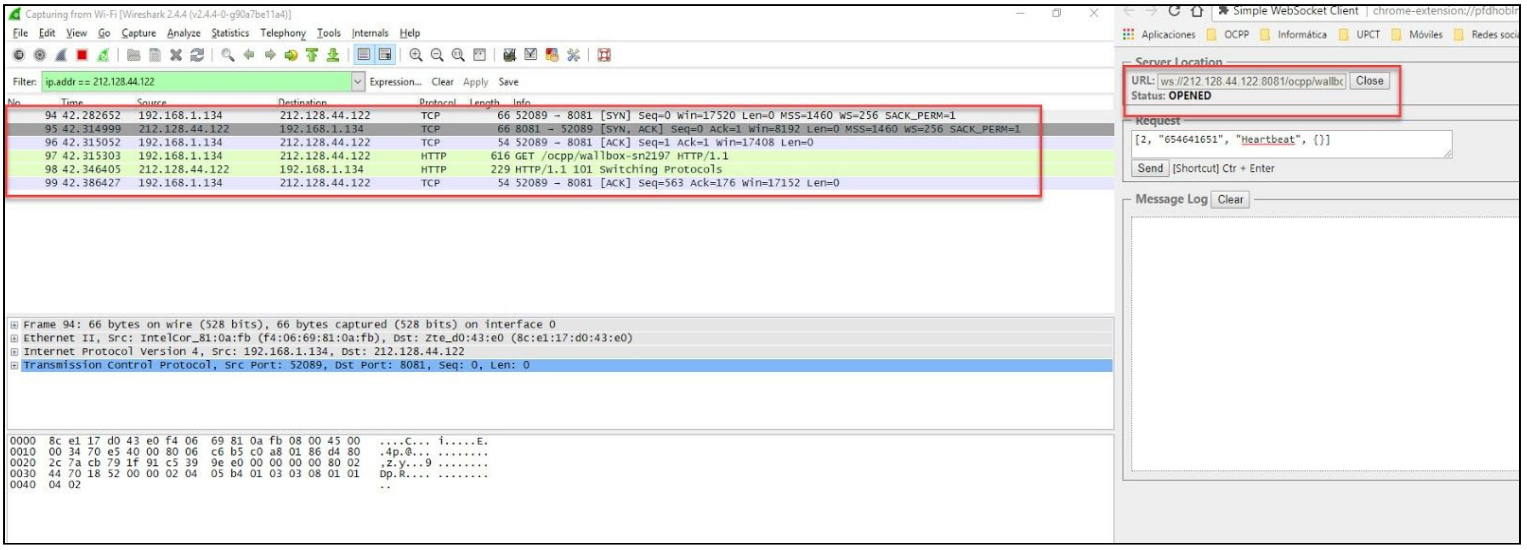


Ilustración 37: Captura Wireshark 1. Conexión con SC exitosa

Una vez establecida la conexión, en las siguientes ilustraciones se muestran los mensajes validados en esta prueba:

Authorize

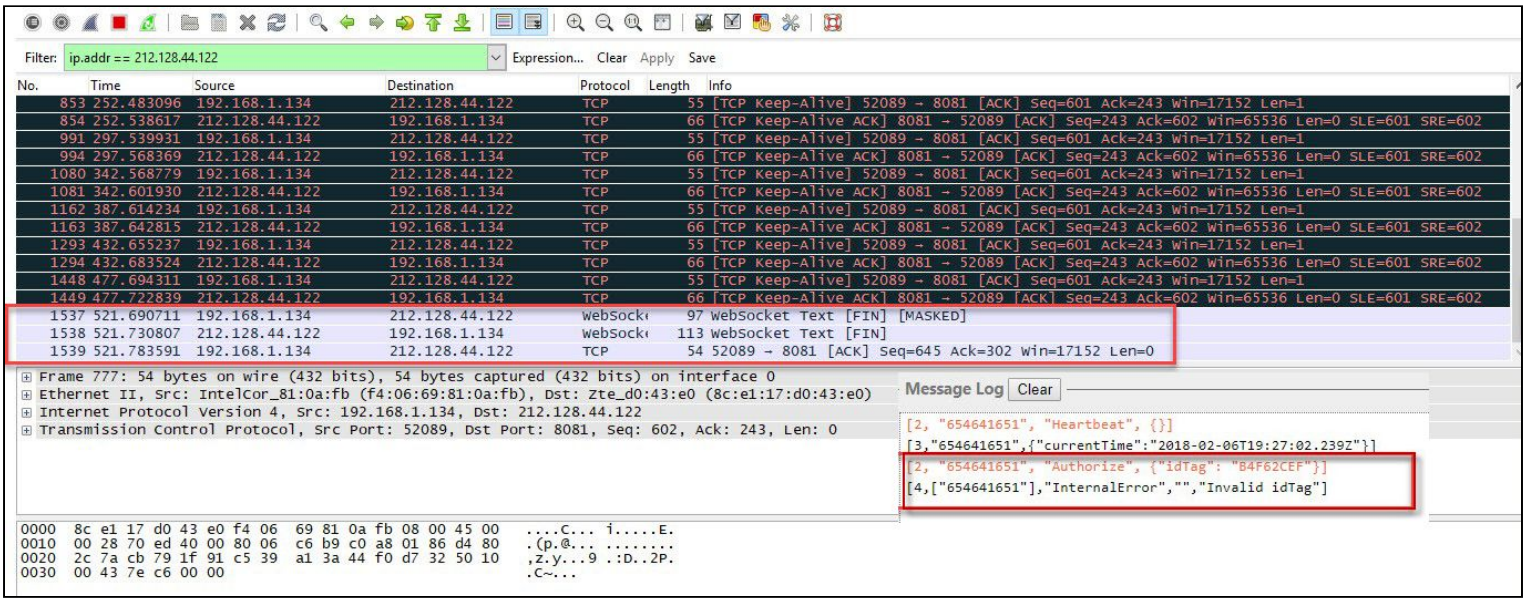
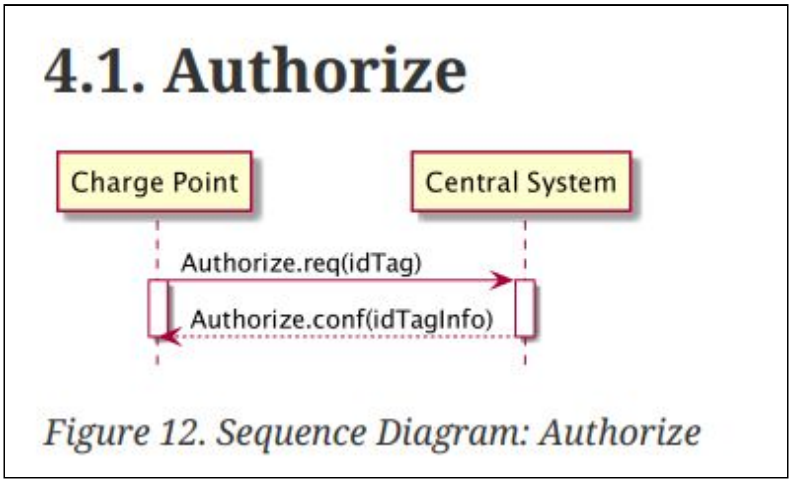


Ilustración 38: Captura Wireshark 2. Authorize con idTag no permitido

Este mensaje permite comprobar si el usuario está autorizado a cargar su vehículo. En este caso, como el identificador no está registrado, no se autoriza la carga.

Diagrama de bloques:



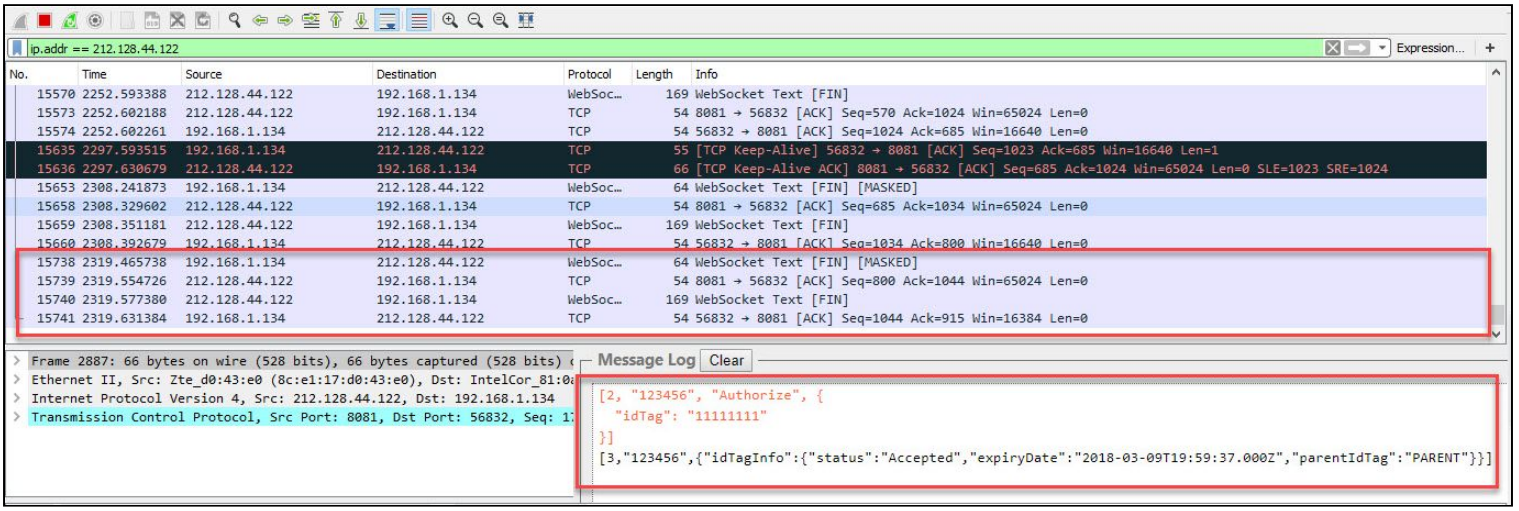


Ilustración 39: Captura Wireshark 3. Authorize con idTag permitido

Como el caso anterior, pero esta vez el identificador está registrado, por lo que el usuario está autorizado a cargar su vehículo.

BootNotification

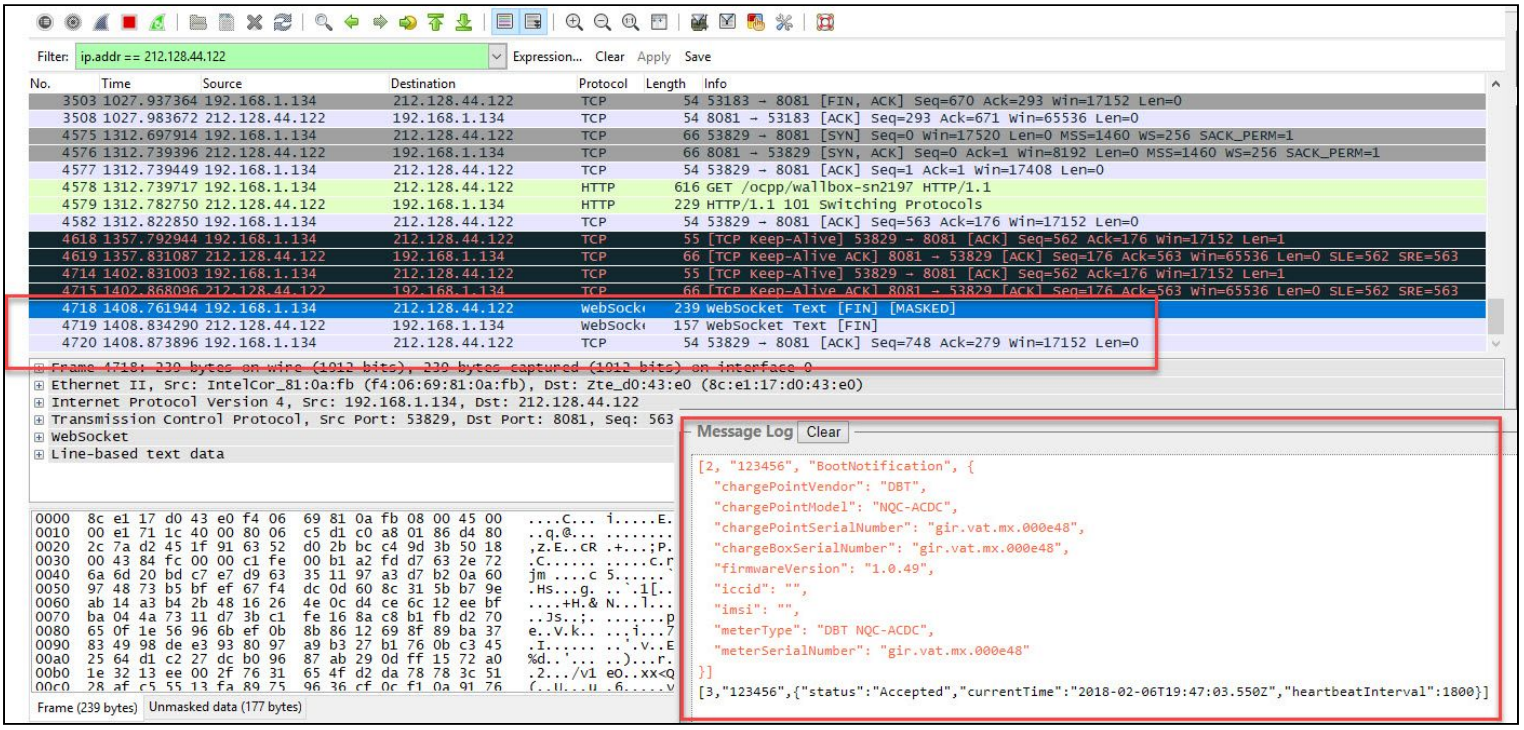


Ilustración 40: Captura Wireshark 4. BootNotification

Mensaje de arranque de la estación. Es el primer mensaje que se mandará entre la estación y el sistema central (ya sea al inicio de una conexión o tras una reconexión). Indica parámetros básicos tales como marca, modelo, versión de software, etc…) y el intervalo de tiempo cada cual debe mandar un mensaje *Heartbeat* (latido de corazón, operatividad).

Diagrama de bloques:

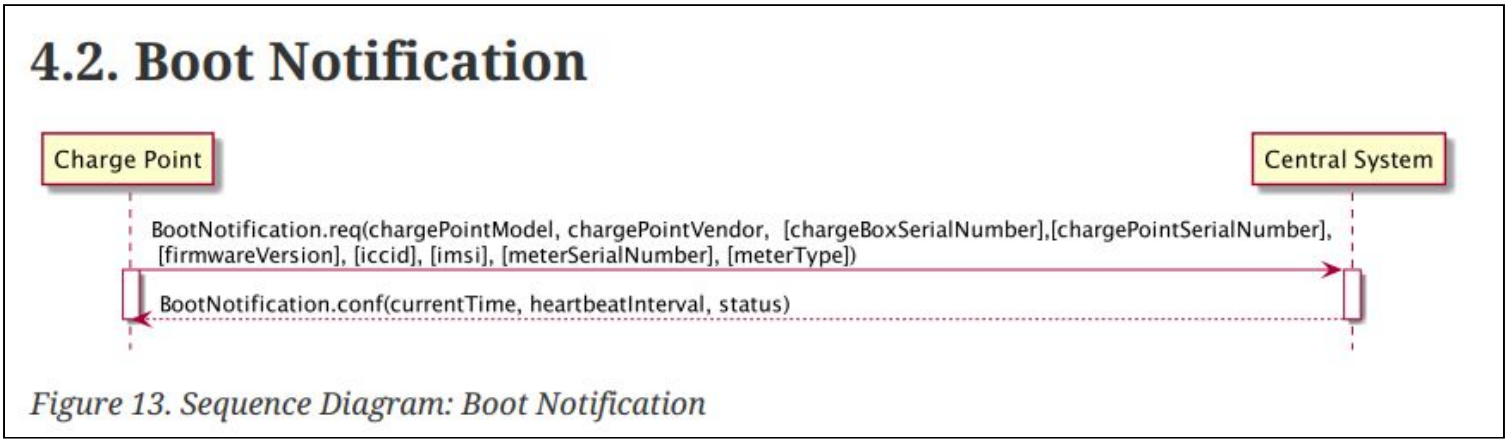


Figure 13. Sequence Diagram: Boot Notification

Heartbeat

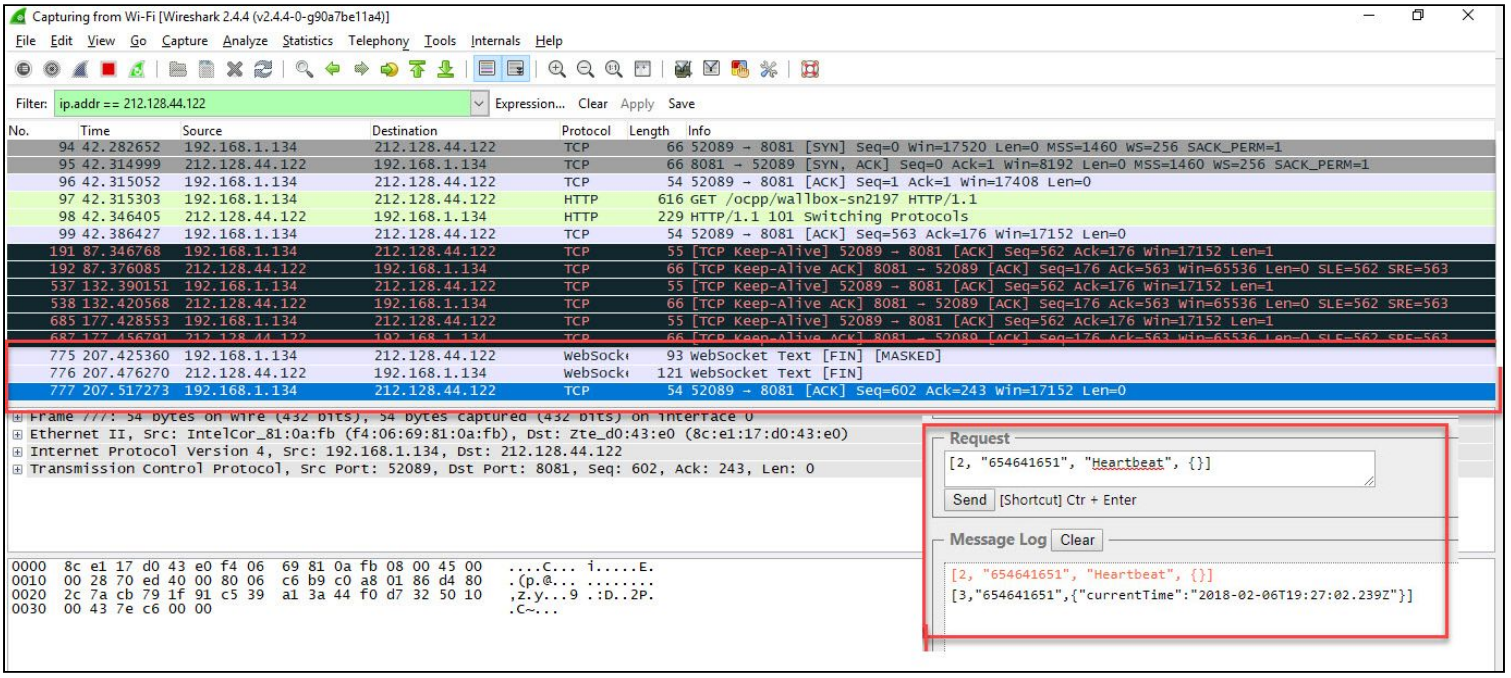
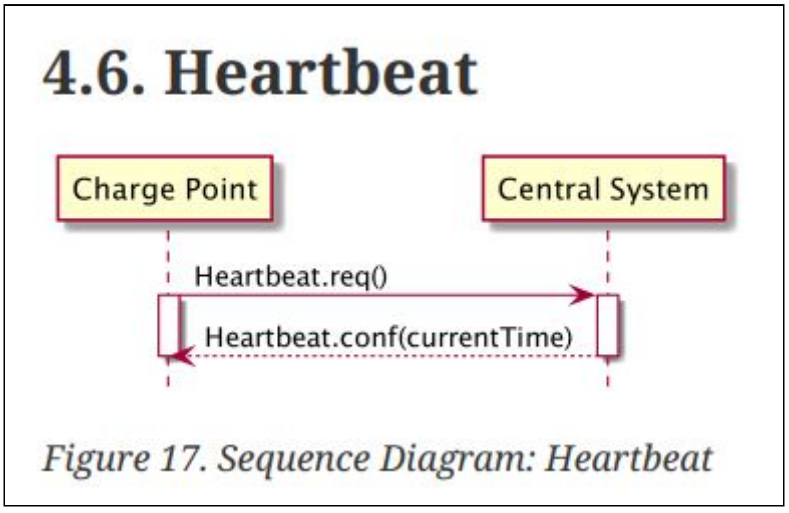


Ilustración 41: Captura Wireshark 5: Heartbeat

Mensaje de *operatividad*: mensaje que se enviará cada X tiempo (indicado por el mensaje *BootNotification*) conteniendo fecha y hora actual, para indicarle a la sistema central que está operativa.

Diagrama de bloques:



MeterValues

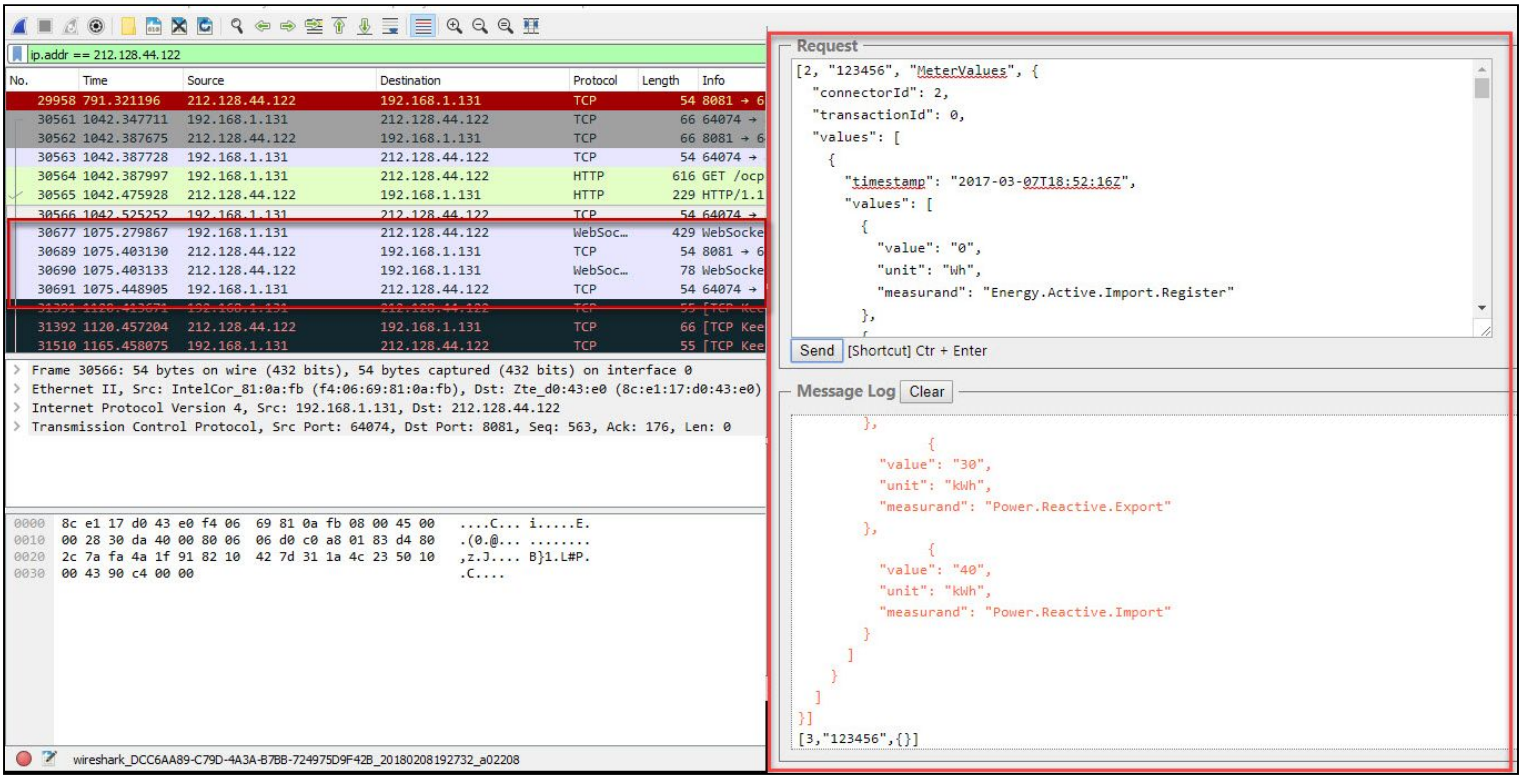


Ilustración 42: Captura Wireshark 6. MeterValues

Mensaje (opcional para los fabricantes) en el cual la estación emite las últimas sesiones de telemetría registradas al sistema central.

Diagrama de bloques:

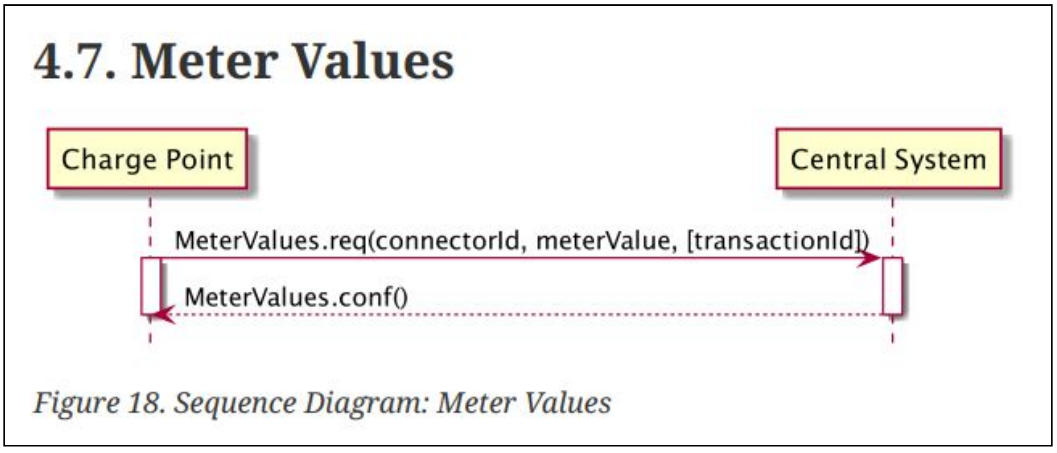


Figure 18. Sequence Diagram: Meter Values

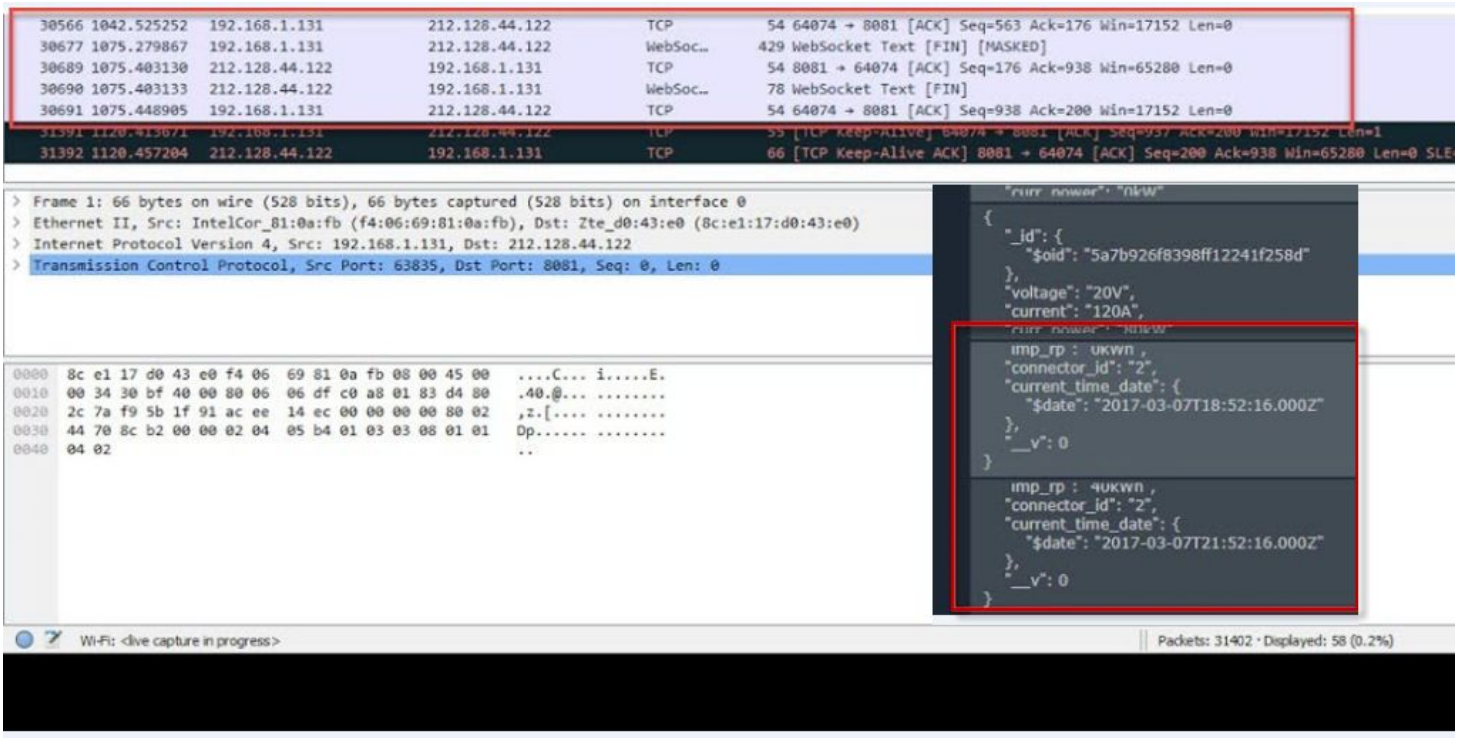


Ilustración 43: Captura 6_2. MeterValues con registro en la BBDD

Las sesiones de telemetría que le vayan llegando (cada X tiempo) al sistema central, se guardarán en el sistema para posteriores consultas.

StartTransaction

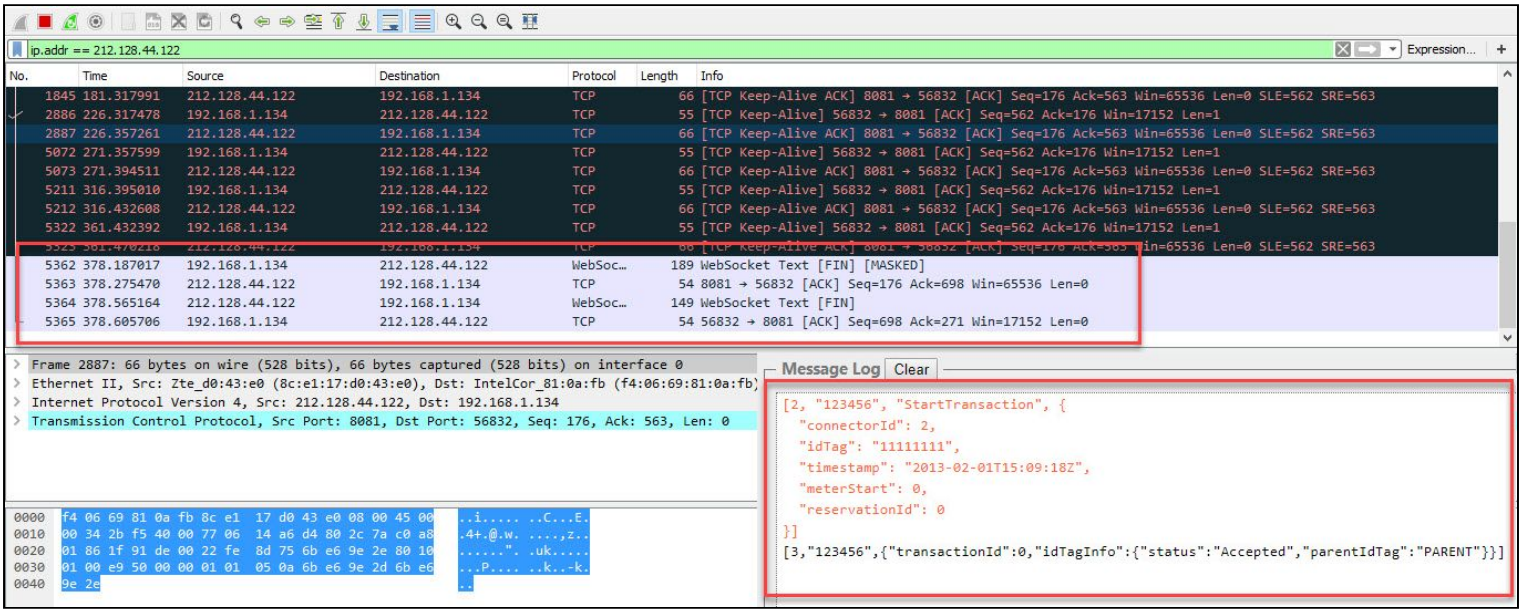


Ilustración 44: Captura Wireshark 7. StartTransaction

Mensaje que indica el comienzo de una transacción (carga), el cual contiene el conector con el que se realiza, fecha y hora del comienzo, *idTag* asociado (el del usuario), *meterStart* (comienzo medida) e id de reserva de carga.

Diagrama de bloques:

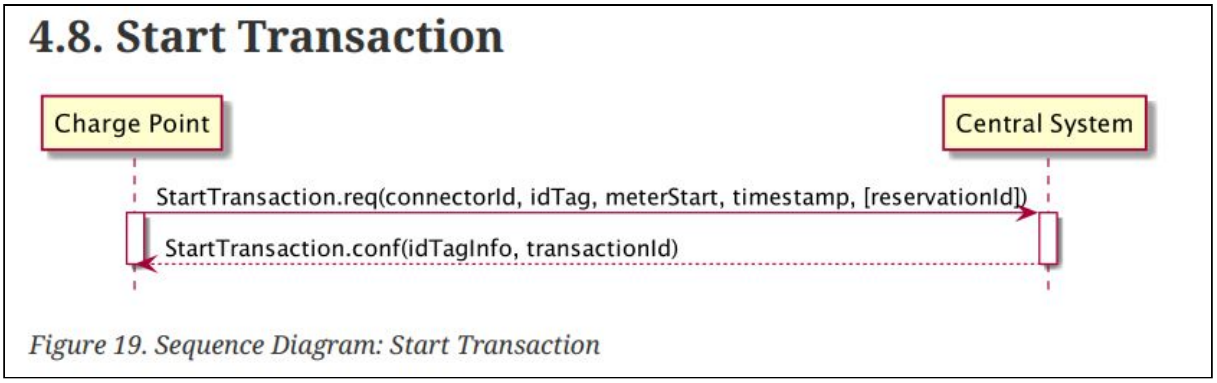


Figure 19. Sequence Diagram: Start Transaction

StopTransaction

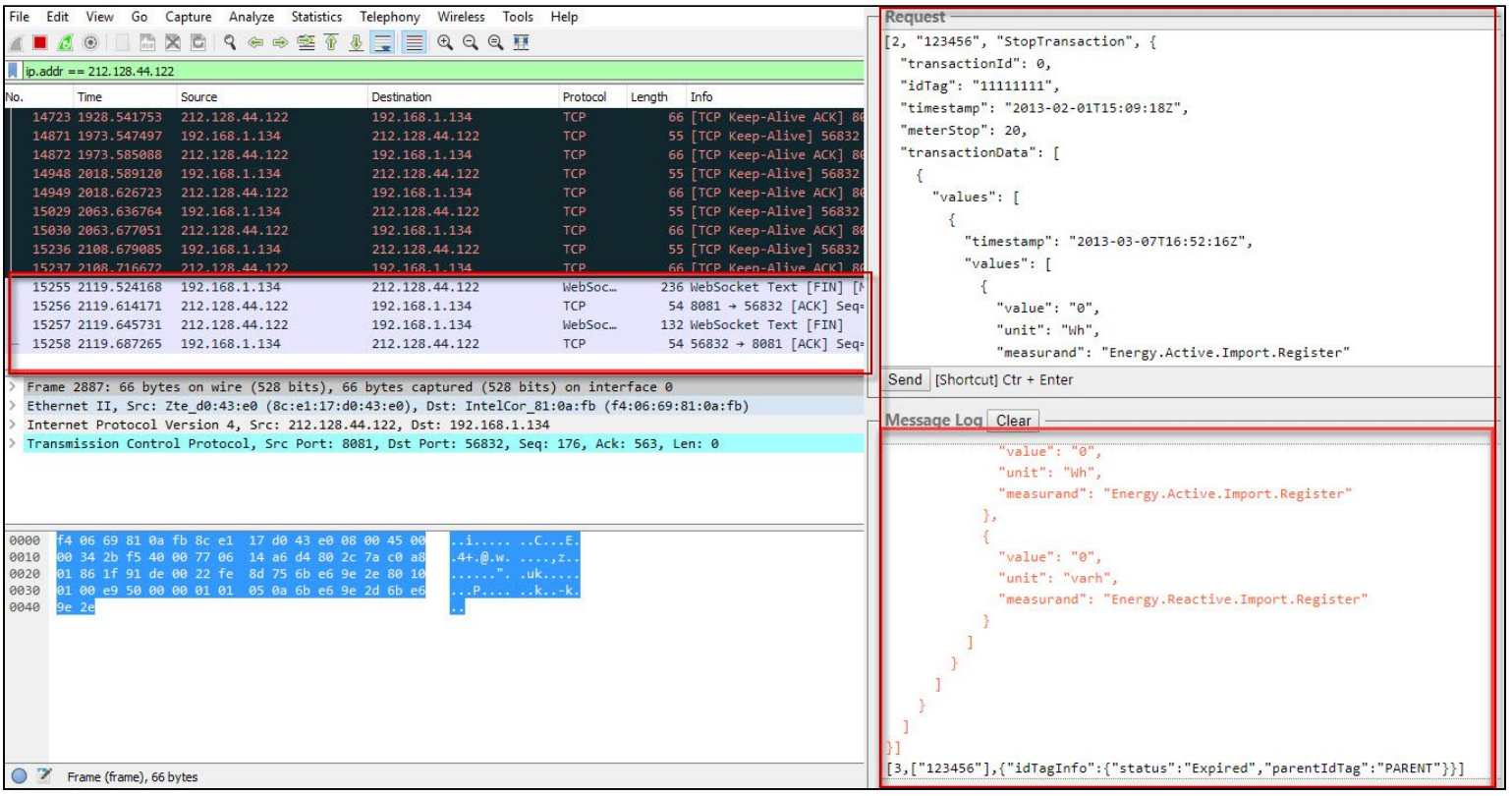
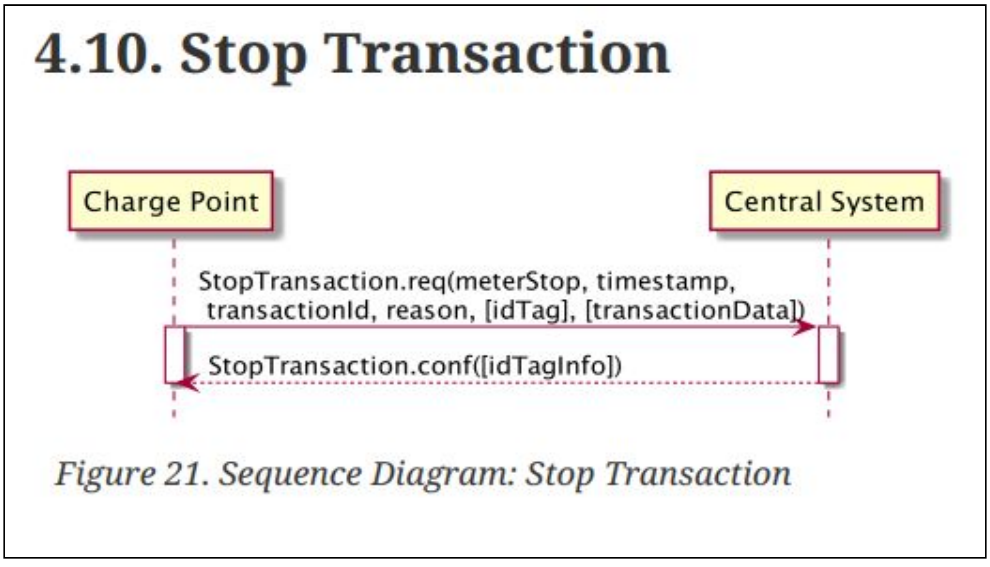


Ilustración 45: Captura Wireshark 8. StopTransaction

Este mensaje indica el fin de una transacción. El parámetro de interés es *meterStop*, el cual utilizará el operario de la electrolinera para cobrar al usuario por la carga realizada.

Diagrama de bloques:



StatusNotification

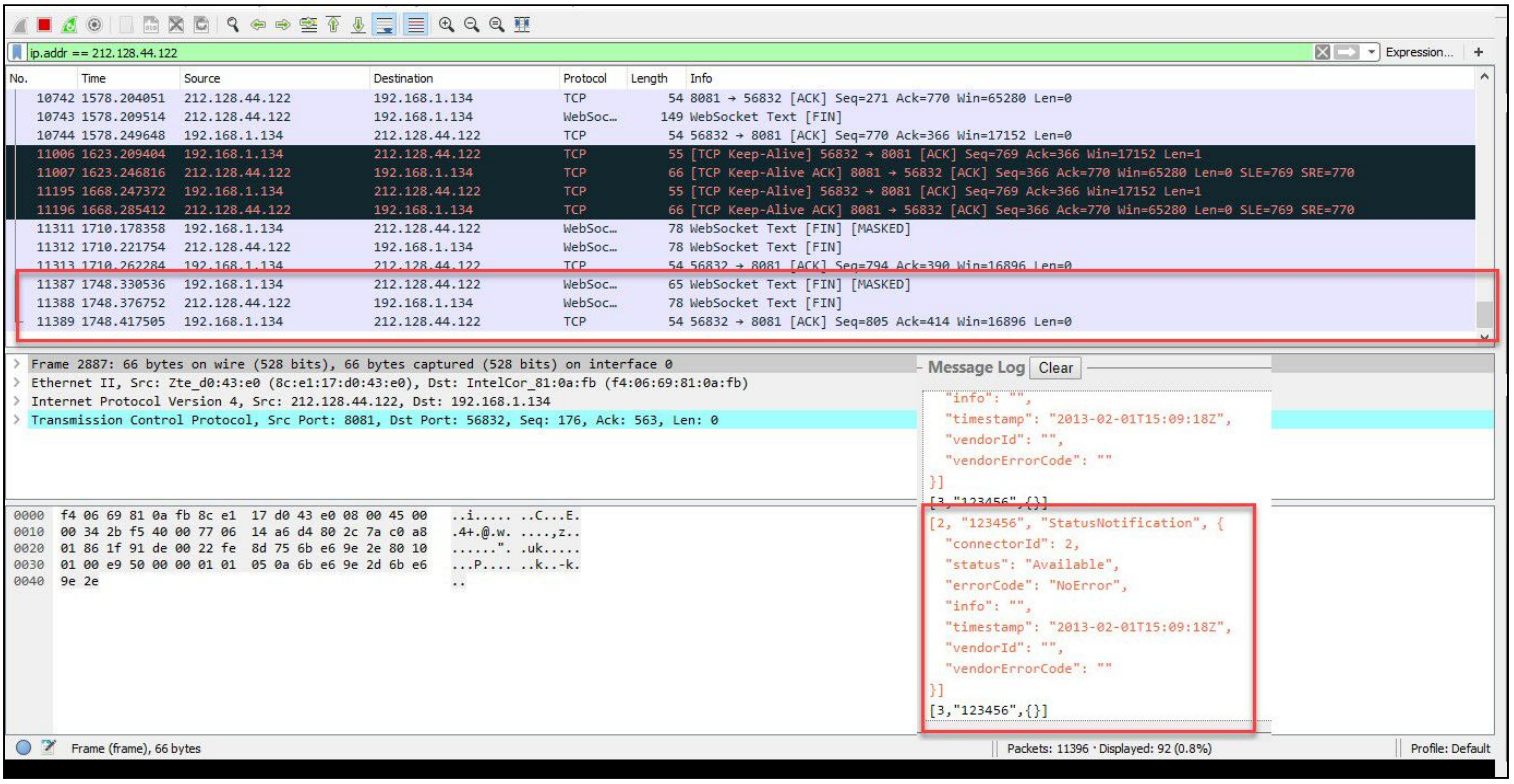


Ilustración 46: Captura Wireshark 9. StatusNotification

Mensaje informativo de eventos de interés ocurridos en la electrolinera. Por ejemplo, informar de la indisponibilidad de un conector en la fecha y hora ocurrida.

Diagrama de bloques:

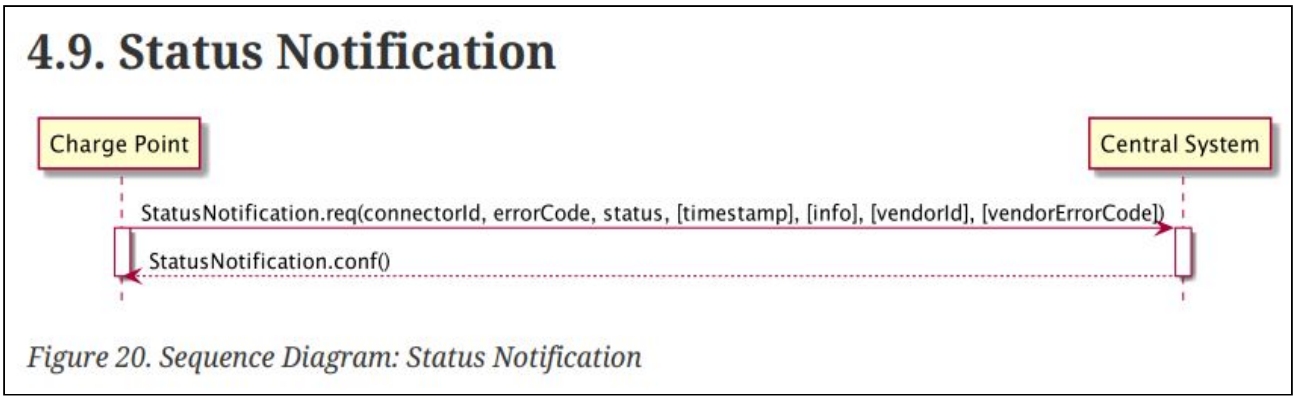


Figure 20. Sequence Diagram: Status Notification

5.2. Líneas futuras de investigación y/o desarrollo. Conclusiones.

En este punto se pueden unificar las líneas de investigación de este proyecto y las conclusiones a las que se han llegado con la realización del mismo:

- **Nuevas y futuras versiones del protocolo OCPP**: el prototipo está preparado para funcionar con la versión OCPP-J 1.6, así como, en buena medida, con la nueva versión 2.0, que estará basada exclusivamente en comunicación vía *JSON / WebSockets*. Si se continuara con el desarrollo del mismo, habría que estar *pendientes de futuras versiones del protocolo* para intentar hacerlo más versátil.

  Este protocolo es joven, con solo 8 años de antigüedad, por lo que habrán mejoras constantes en un futuro no muy a largo plazo, ya que está previsto que se produzca un aumento en la compra de vehículos eléctricos para reducir las emisiones de CO2 a la atmósfera, lo que genera una necesidad de *mejorar la comunicación entre los vehículos eléctricos y las estaciones de recarga* para ofrecer una solución que gestione la mayor cantidad de estaciones de recarga, ofreciendo a los usuarios, en tiempo real, toda la información de interés para ellos (estaciones libres, la más rápida, la más usada, etc…).

- **Es solo un prototipo**: el presente proyecto es un prototipo. Como bien dice su nombre, es una ***versión preliminar***, es decir, una versión inicial de la aplicación, creada con fines para la investigación del protocolo, que tiene amplio margen de mejora, ***pero estaría preparado para realizar pruebas con una estación de recarga real*** y llevar a cabo los objetivos asociados al mismo.

- **Modelos homólogos**: a este prototipo se le podrían realizar versiones que utilicen otro tipo de esquema (en vez de JSON, por ejemplo SOAP/XML). No obstante, este último tipo de esquema quedará obsoleto con el paso de los años, ya que la tendencia de OCPP será el uso exclusivo de JSON a la hora de transportar información.

- **Profundizaje en la investigación de OCPP**: la única información disponible para investigar el funcionamiento de dicho protocolo son los manuales y esquemas que ofrece la OCA (Open Charge Alliance), así como diversos sitios web (incluidos en el anexo). *Lo que se echa en falta* es un prototipo *compatible con ambas tecnologías (JSON / XML) que sirva de referencia para aprender más sobre su funcionamiento*.

6. Anexos

ANEXO I: Wallbox. Cargador y uso de la plataforma.

Posee una pantalla táctil de 7 pulgadas, desde la cual podremos visualizar datos de interés como, por ejemplo, la potencia suministrada al vehículo, el coste total de la carga, etc…, programar cargas al vehículo, por ejemplo si no estamos en casa, y queremos que el vehículo se cargue mientras no estamos, sólamente tenemos que indicar hora de inicio y final y confirmamos.

Ilustración 47: Pantalla de bienvenida

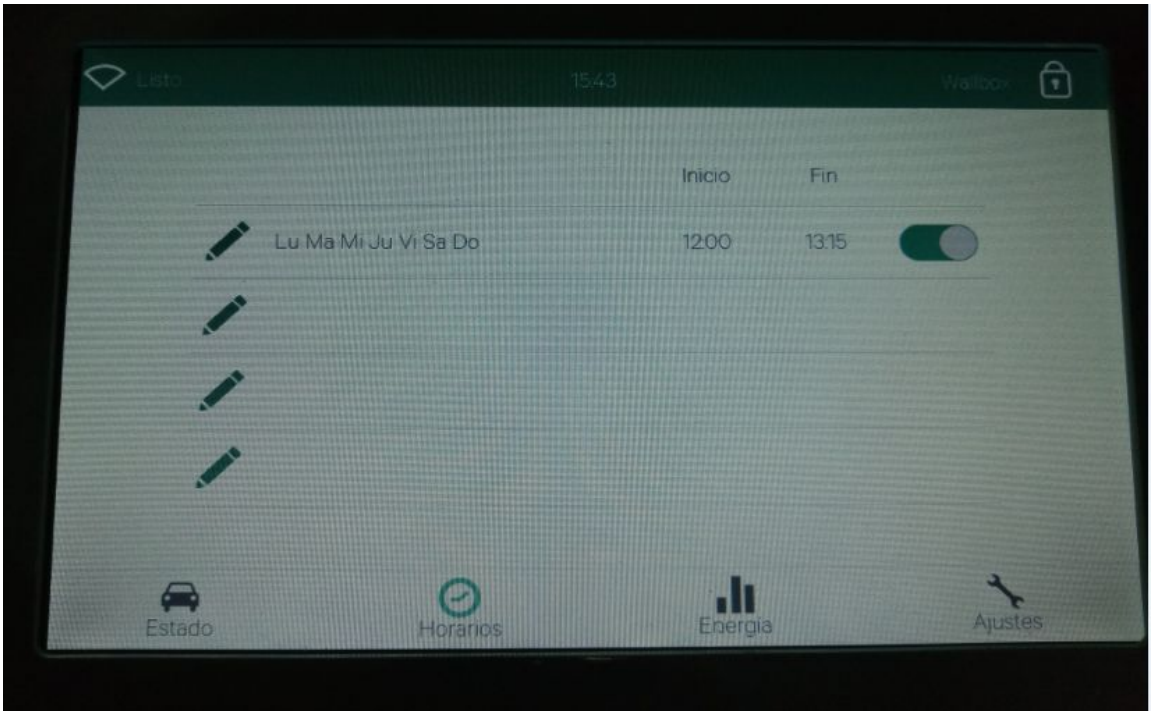


Ilustración 48: Programador de cargas

Otra opción que nos ofrece es la visualización, **por intervalo de fechas**, de las cargas que hemos realizado por días, meses y años, observando el gráfico en función de la potencia suministrada o del coste de la carga de ese día/mes/año.

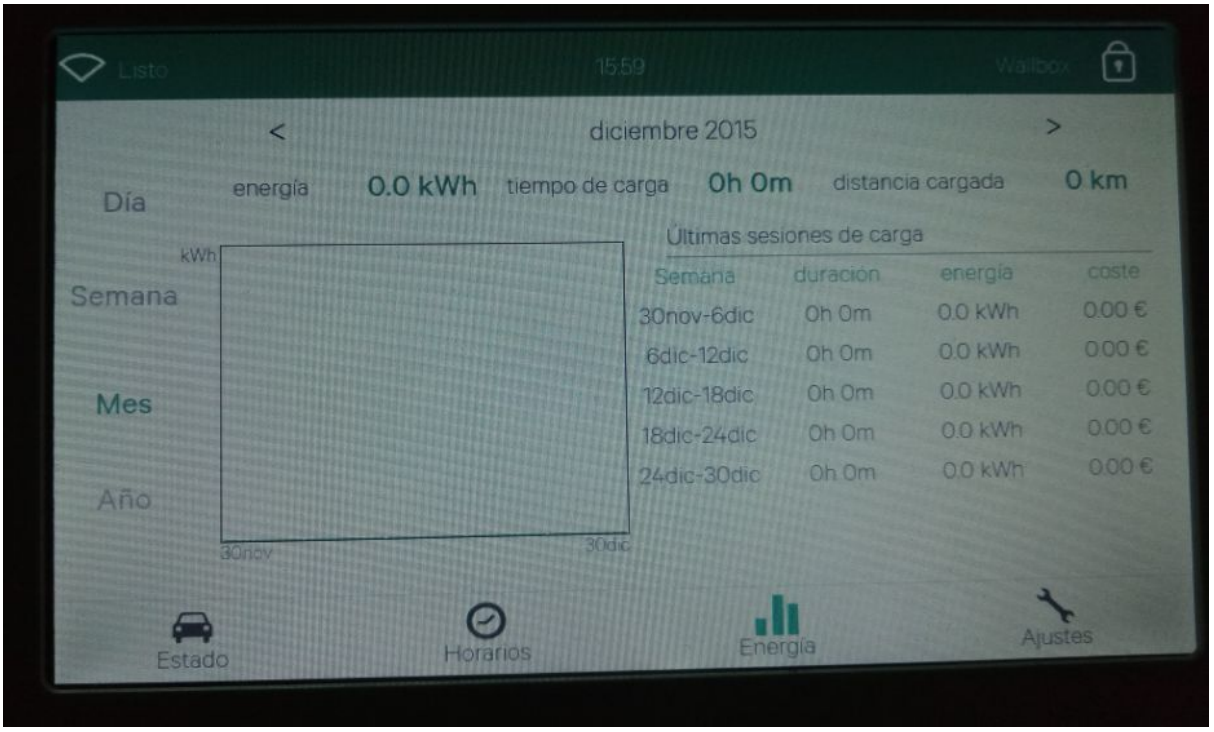


Ilustración 49: Gráfico de consumo de energía mensual

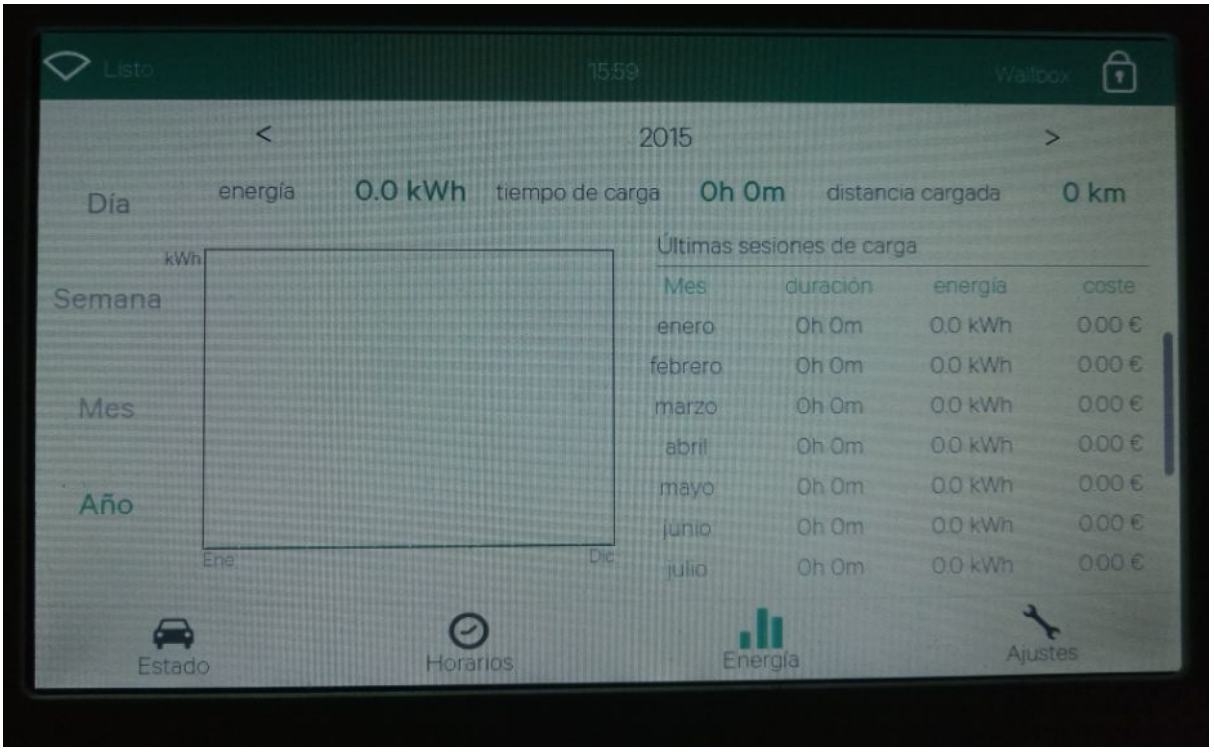


Ilustración 50: Gráfico consumo de energía anual

En *Apariencia*, podremos configurar la estación como queramos, indicando el nombre de la misma, el modelo de coche que vamos a cargar, esto es, para mostrar datos más realistas al usuario, y pudiendo escoger una imagen personalizada de bienvenida al arrancar la estación.

Ilustración 51: Ajustes -> Apariencia

En *Energía*, podremos configurar el tipo de energía, personalizar su nombre y modificar el coste/kW de potencia para obtener un resultado lo más *realista* posible.

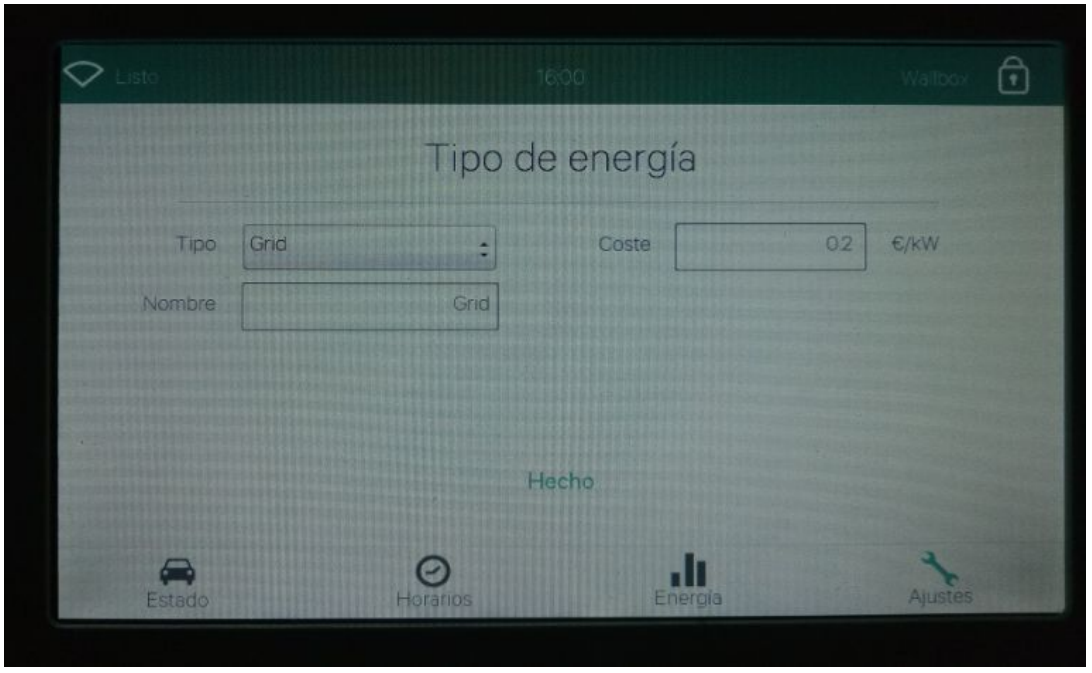


Ilustración 52: Ajustes -> Energía

En el siguiente, podremos configurar una conexión WiFi para poder acceder, bien sea desde la aplicación Android o desde el portal web *mywallbox*, a los datos en tiempo real de la estación, programar cargas o bien comprobar el registro de cargas que hayamos realizado.

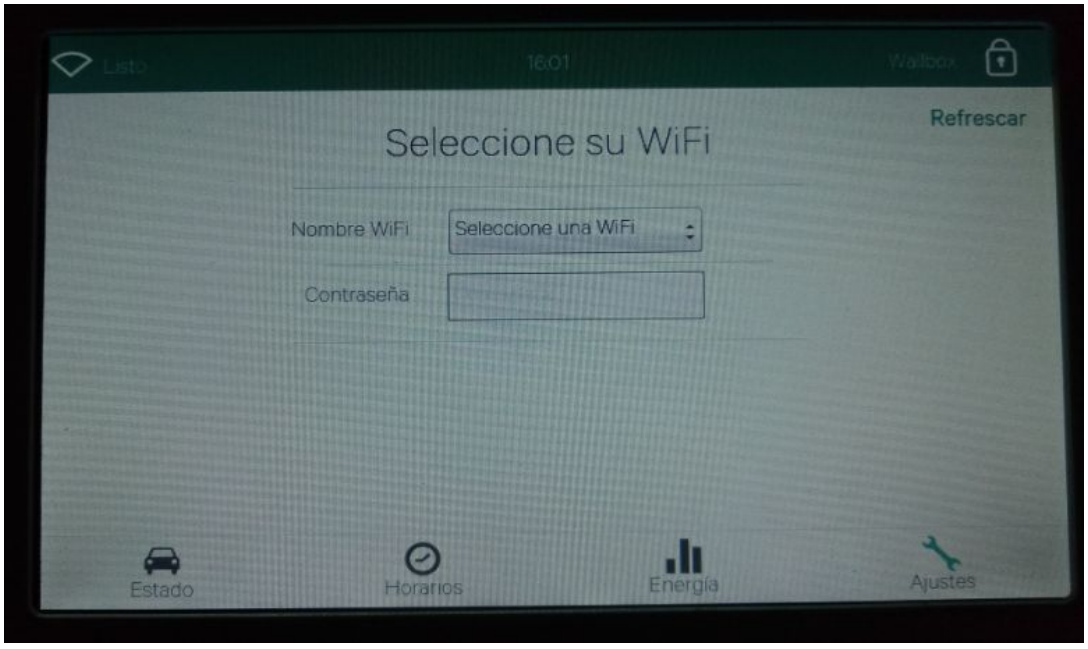


Ilustración 53: Ajustes -> WiFi

En *Bloqueo*, podremos configurar un PIN para bloquear la estación, pudiendo hacerlo automáticamente, para una mayor seguridad a la hora de gestionar su uso.

Ilustración 54: Ajustes -> Bloqueo

En *Día y Hora*, podremos configurar zona horaria, fecha, día y hora actuales, bien de forma manual o automática (actualizándose a través de conexión WiFi/Ethernet).

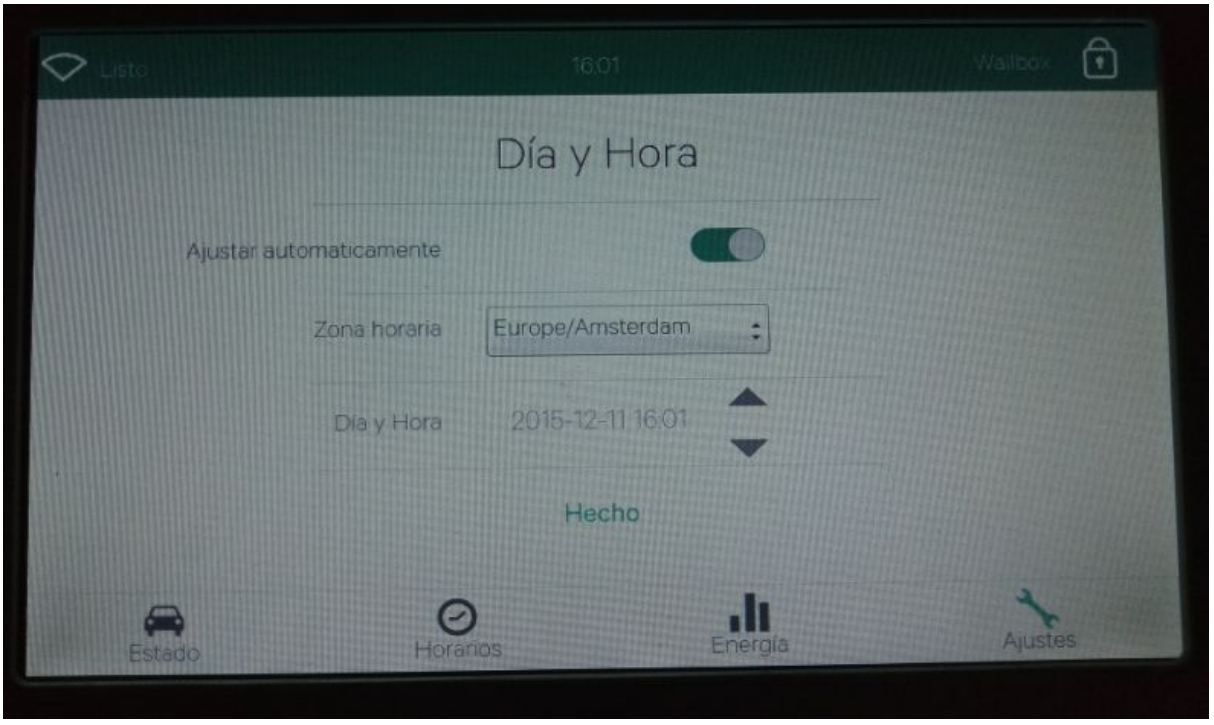


Ilustración 55: Ajustes -> Día/Hora

En *Actualizaciones*, podremos comprobar si existen nuevas actualizaciones para la estación. Éstas son ***de por vida y totalmente gratuitas***.

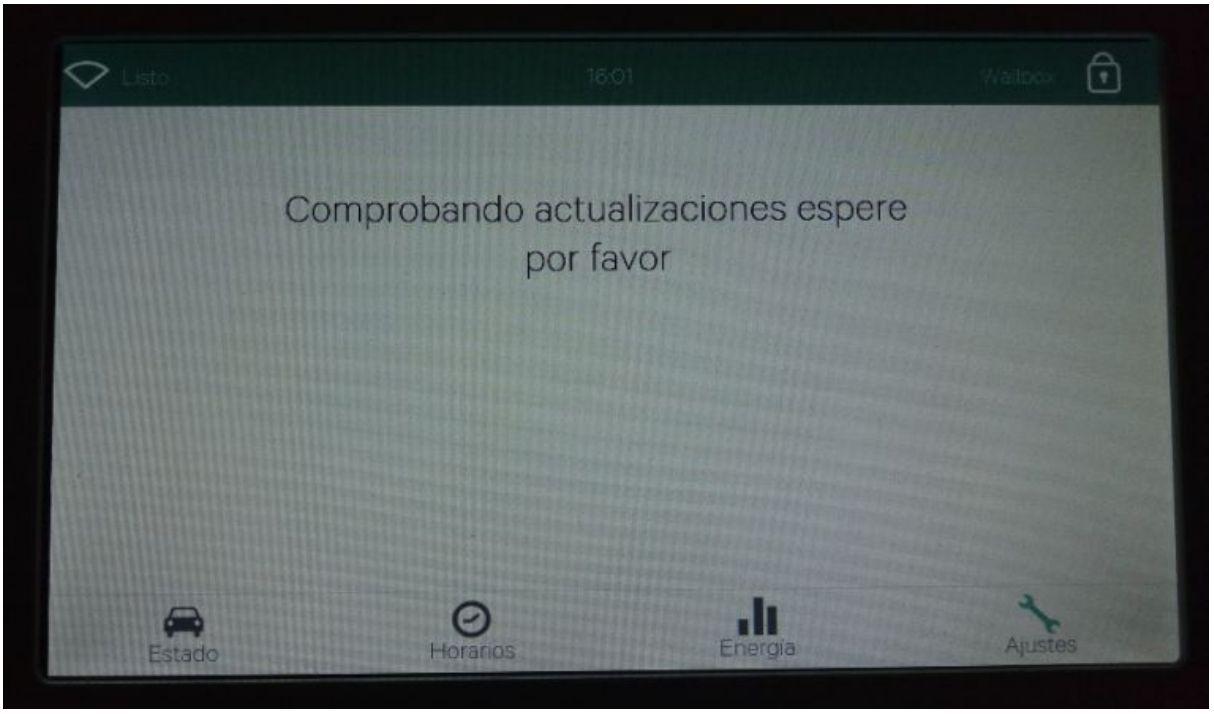


Ilustración 56: Ajustes -> Actualizar

En *Sistema*, podremos reiniciar el sistema, o bien restaurarlo si detectamos que la versión del software instalada en la estación no funciona como debería.

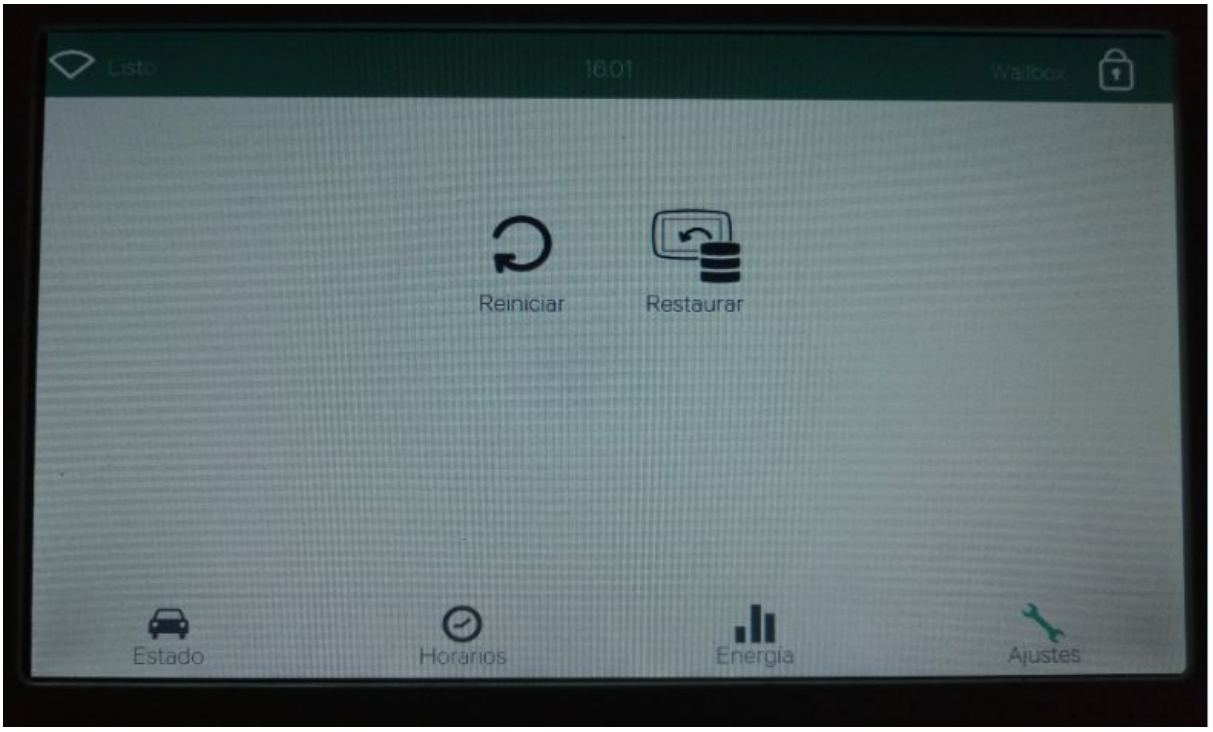


Ilustración 57: Ajustes -> Sistema

Por último, en *Acerca*, podemos observar el número de serie de la estación, la versión de software instalada, y las direcciones MAC de las tarjetas WiFi y Ethernet. ***Por seguridad***, ambas MAC se han omitido en la ilustración.

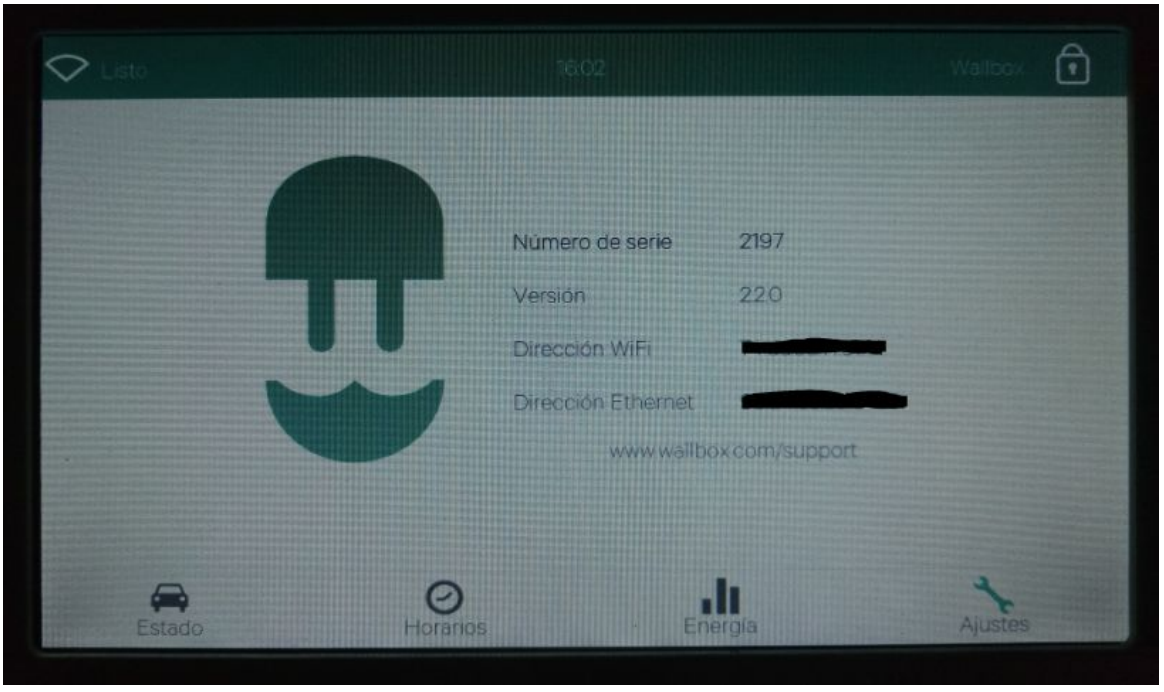


Ilustración 58: Ajustes -> Acerca

6.1.1. Portal web myWallbox.

Para empezar a gestionar la estación, primero tenemos que registrarnos en el portal myWallbox y dar de alta la estación adquirida introduciendo el UID (Unique ID) y número de serie correspondiente. Para ello, debemos seguir los siguientes pasos:

1. Accedemos a https://www.wallbox.com/es/mywallbox/ y escogemos la opción *Crear una cuenta*.

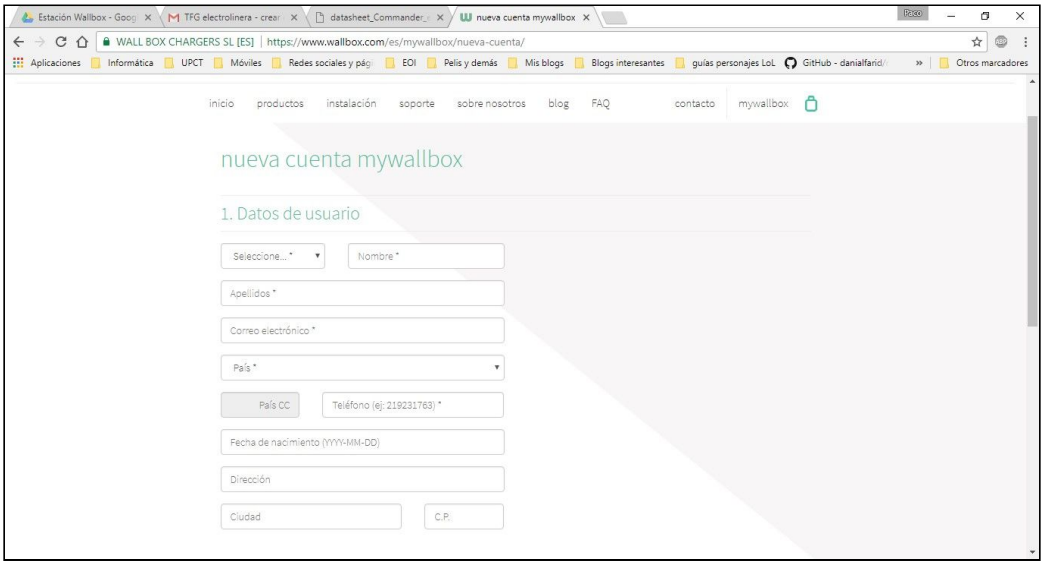


Ilustración 59: Registro en MyWallbox

2. A continuación, introducimos todos los datos obligatorios (marcados con *). *Importante*: no hace falta rellenar los datos de negocio, ya que el equipo se ha adquirido por cuenta propia (gracias a la colaboración con la empresa *Solventie*), por lo que marcamos *Omitir datos de negocio*.

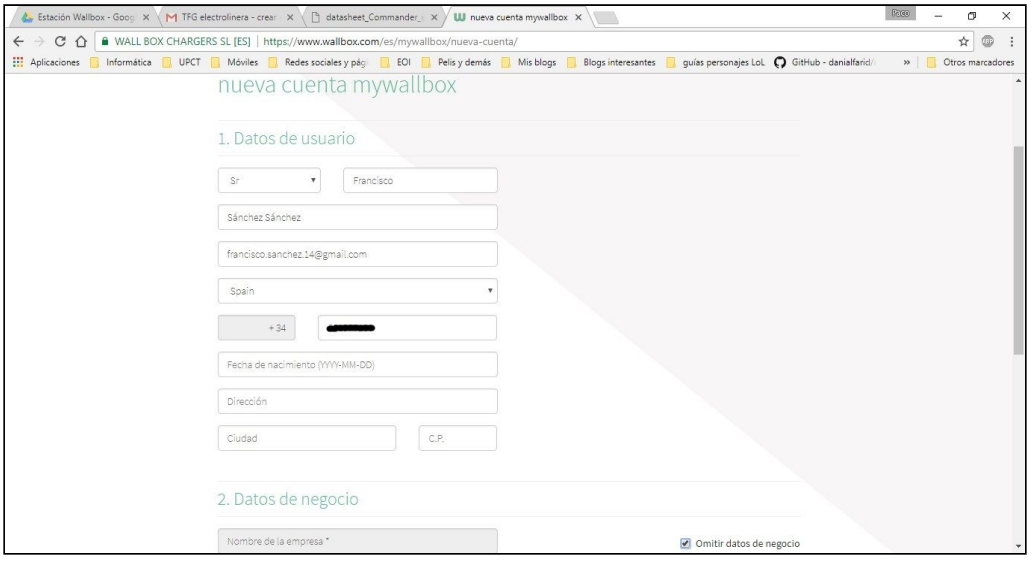


Ilustración 60: Completando registro en MyWallbox

3. Una vez está el formulario rellenado, seleccionamos *Continuar*, y se nos mostrará un aviso de que se nos ha enviado un email para confirmar la cuenta.

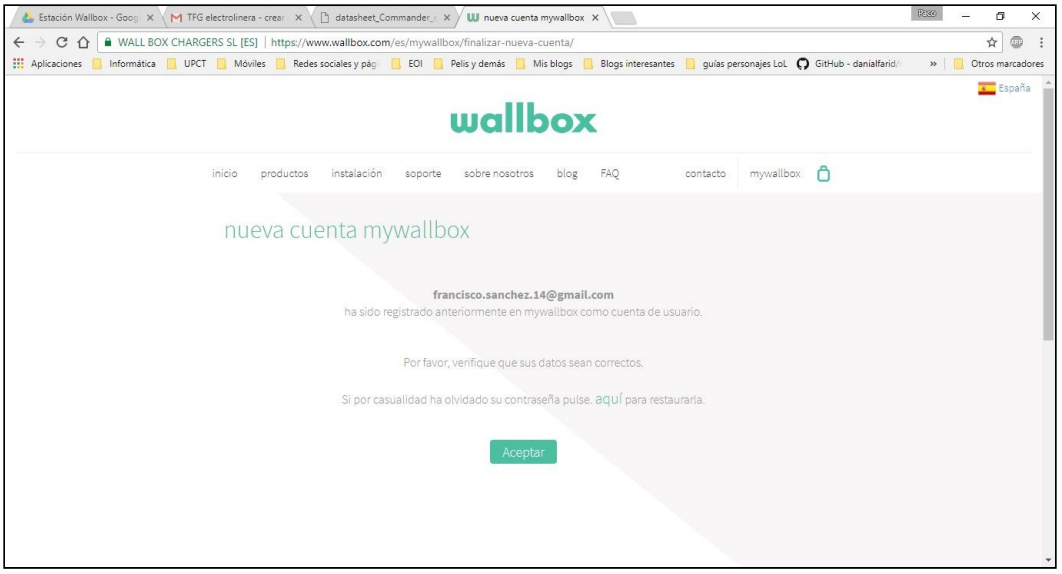


Ilustración 61: Registro completado

4. El siguiente paso es confirmar el registro y establecer una contraseña de acceso al portal. Para ello:

    a) Entramos en la cuenta de correo asociada y hacemos clic en *Activar cuenta de usuario*.

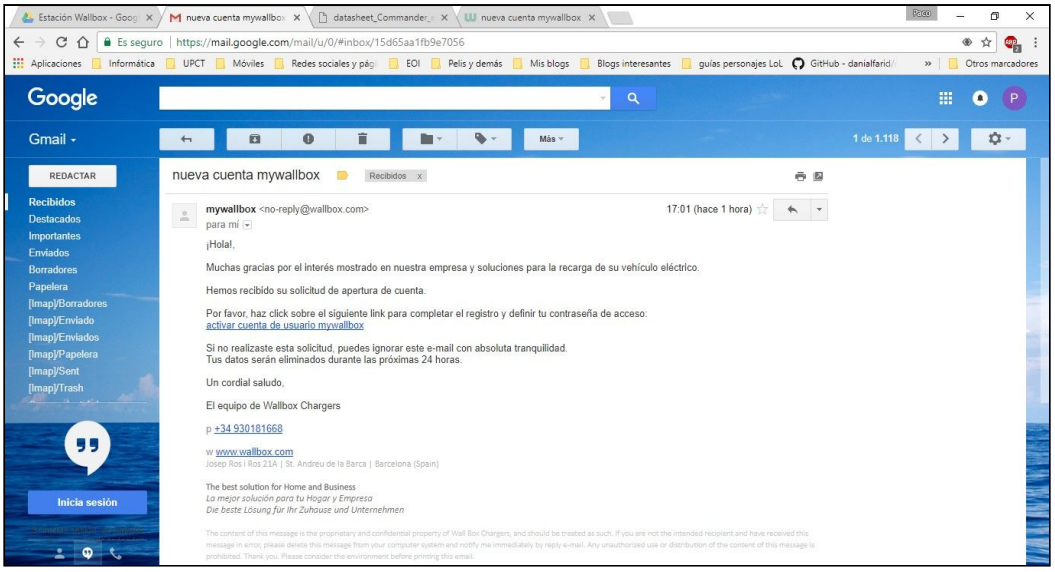


Ilustración 62: Correo confirmación registro en MyWallbox

b) Establecemos una contraseña para la cuenta y ya estaremos en disposición de añadir el nuevo cargador.

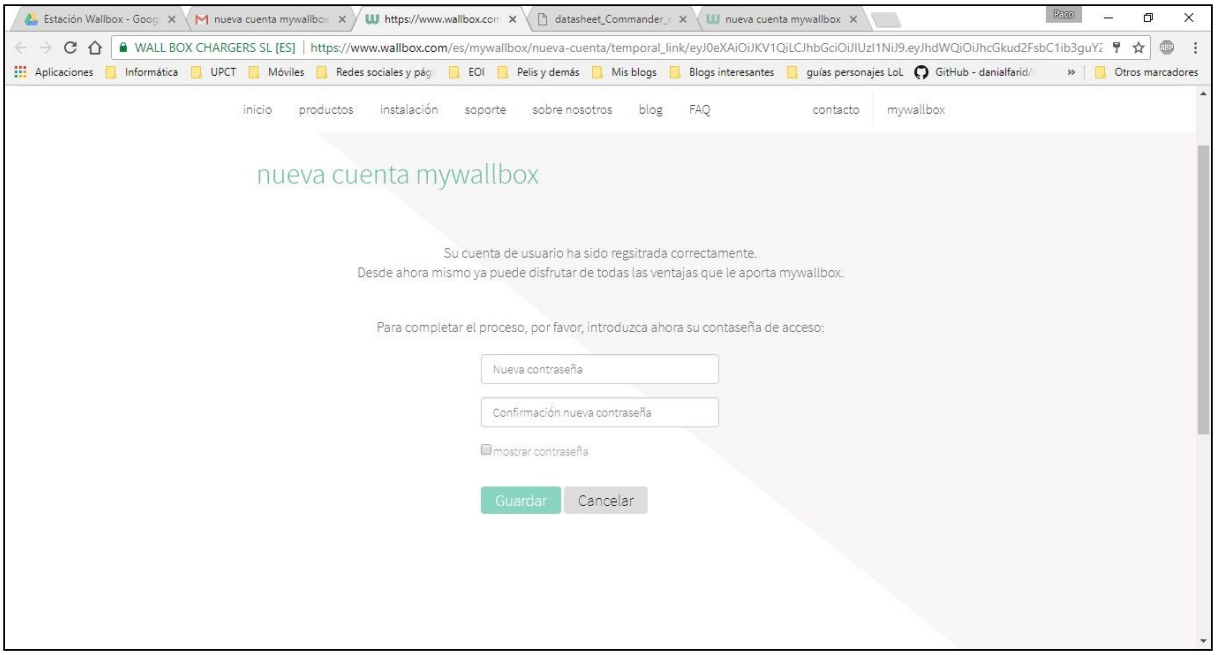


Ilustración 63: Logueo en MyWallbox

5. Ahora podemos acceder con la cuenta creada al portal *myWallbox*. Una vez logueados, se nos muestra el panel de bienvenida. A la derecha, tenemos un panel para añadir nuevos cargadores.

Ilustración 64: Panel registro de cargadores

6. Introducimos tanto el UID como el número de serie del equipo. Una vez hecho, se nos mostrará el cuadro de mando (página por defecto al loguearnos) con toda la información disponible de nuestra estación de recarga, así como la posibilidad de añadir más cargadores, controlar la corriente suministrada a nuestro vehículo, bloquear el acceso a la estación mediante un PIN o bien visualizar un panel resumen con los datos de la última recarga realizada. También incluye un panel de novedades con las noticias más relevantes de la empresa Wallbox Chargers.

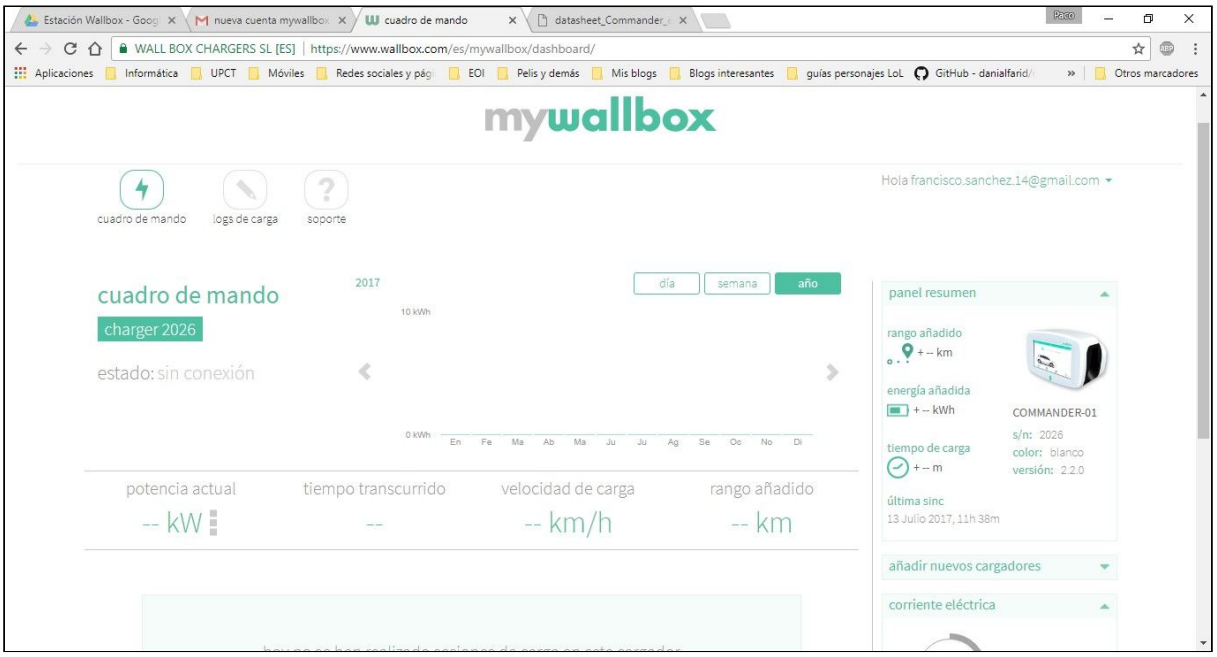


Ilustración 65: Panel usuario MyWallbox (1)

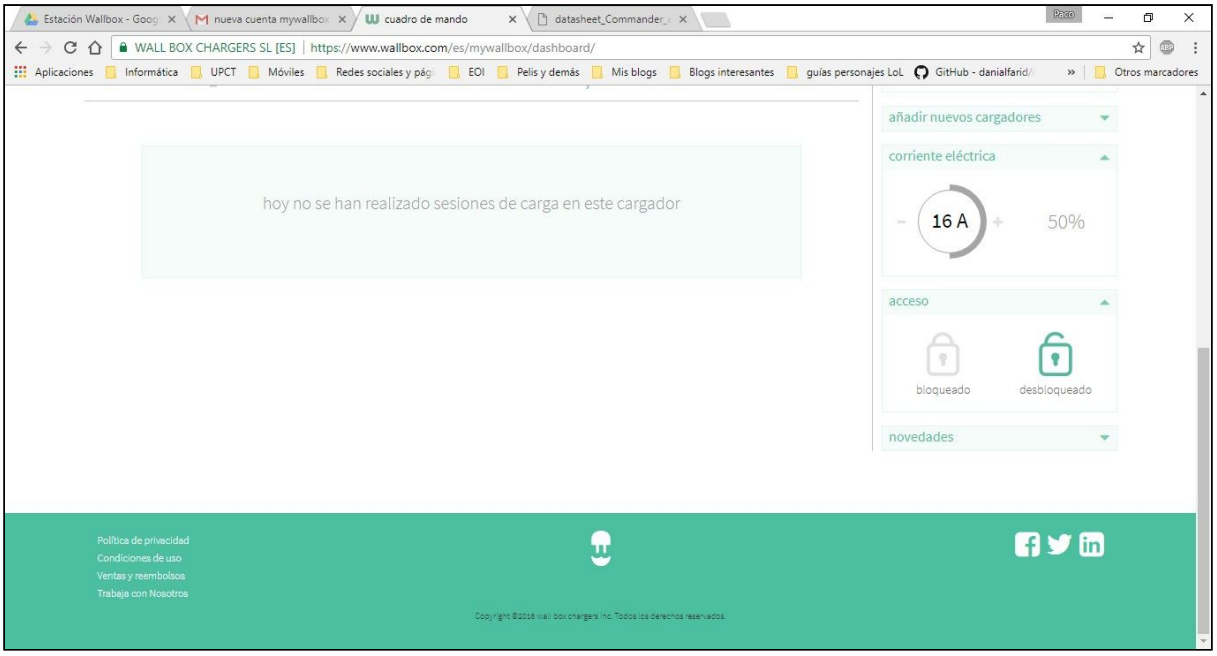


Ilustración 66: Panel usuario MyWallbox (2)

6.1.2. App Android Wallbox.

Otra de las posibilidades que nos ofrece el fabricante es gestionar la estación a través de su aplicación ***Wallbox***, disponible para smartphones con sistema operativo *Android* e *iOS*.

Nos encontramos con una aplicación compacta, enfocada en darle al usuario un rendimiento óptimo con una interfaz limpia y fácil de usar. Para este proyecto, el smartphone utilizado opera bajo el S.O. *Android*, por lo que las siguientes ilustraciones corresponden a la aplicación de *Wallbox para Android*:

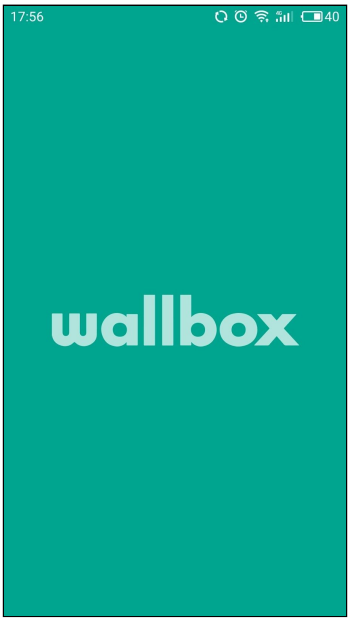


Ilustración 67: Animación de bienvenida a la aplicación.

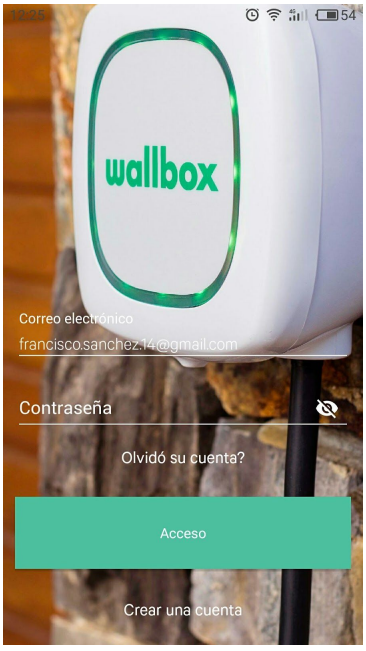


Ilustración 68: Panel de logueo.

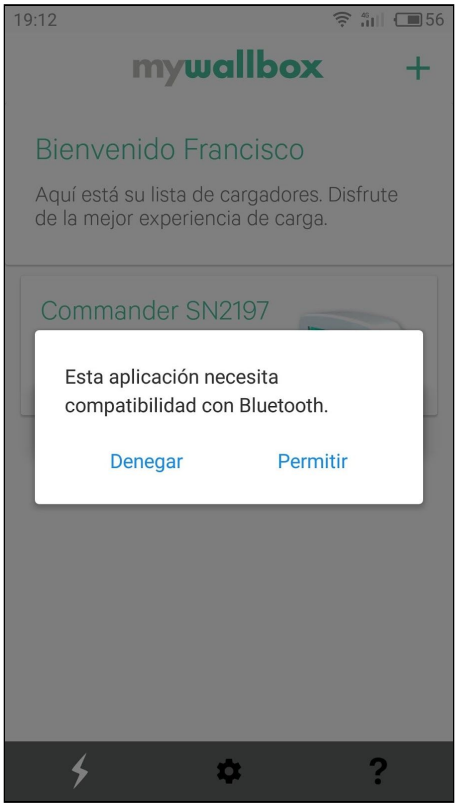


Ilustración 69: Aviso para activar bluetooth y poder programar cargas

Ilustración 70: Panel principal. Estado

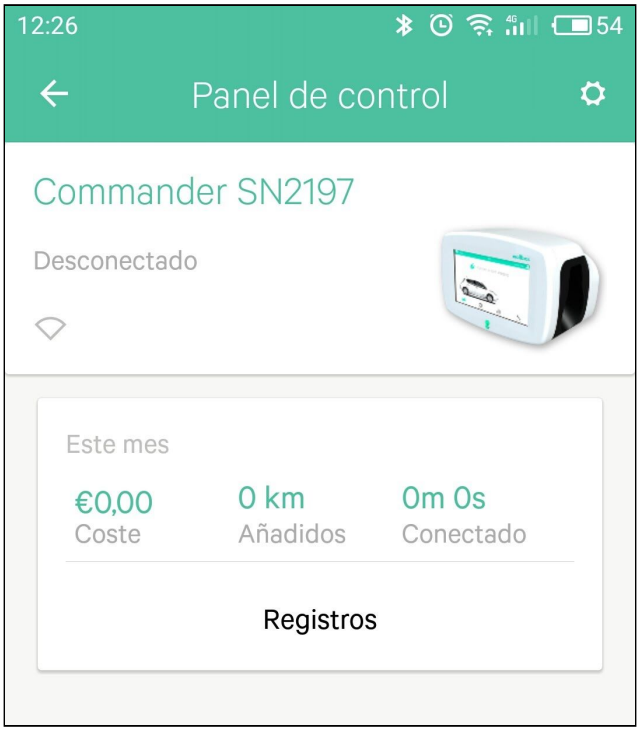


Ilustración 71: Panel de registro/logs de cargas.

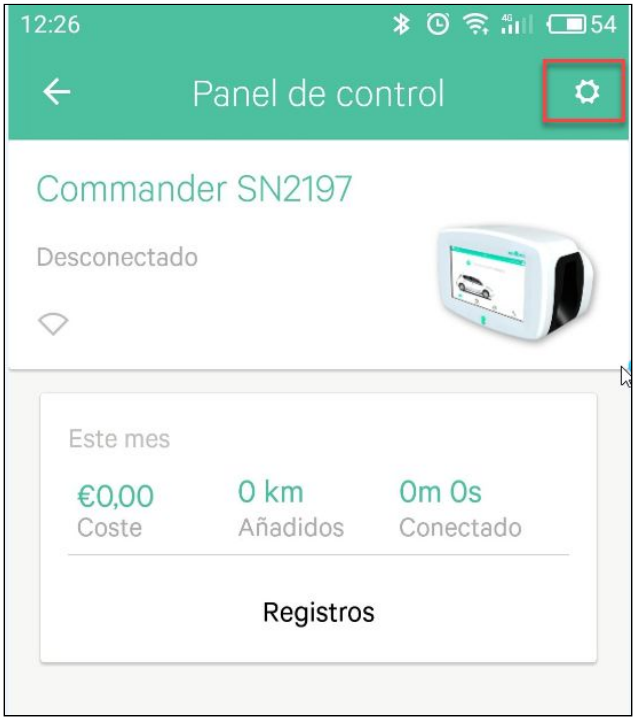


Ilustración 72: Acceso opciones en panel de control

Ilustración 73: Panel de ajustes del cargador.

Ilustración 74: Acceso información básica cargador

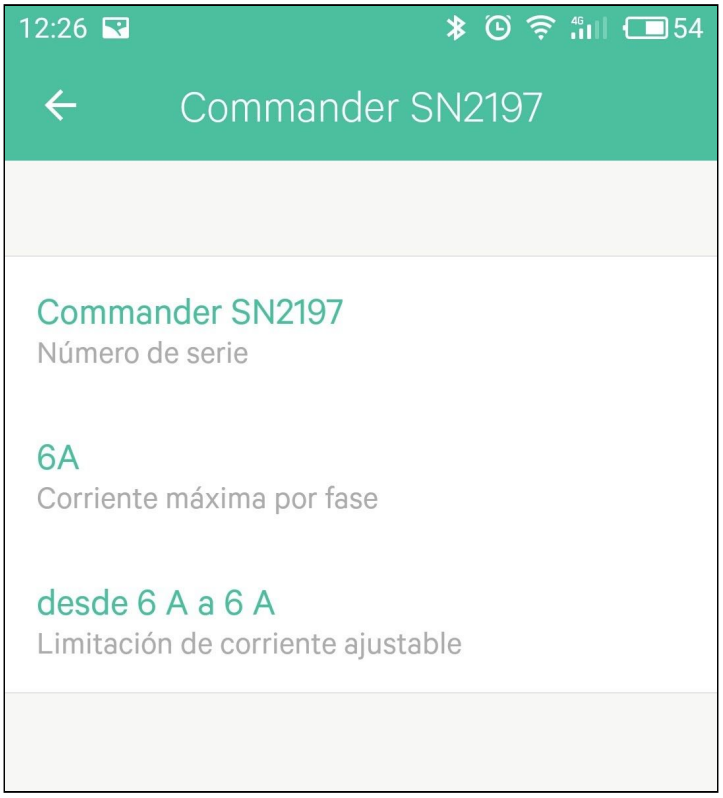


Ilustración 75: Información del cargador

Ilustración 76: En esta opción se puede cambiar el nombre del cargador

Ilustración 77: Se puede ajustar el precio del kWh para obtener gráficos más realistas.

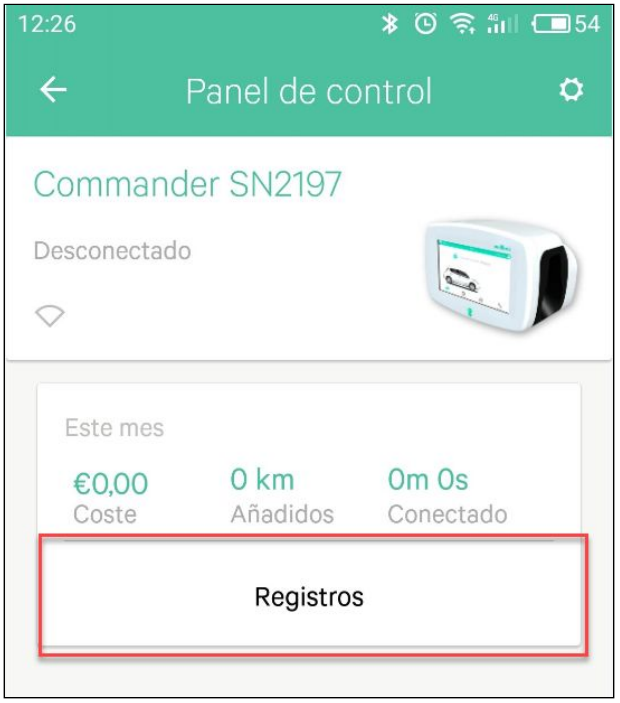


Ilustración 78: Acceso a registros de cargas

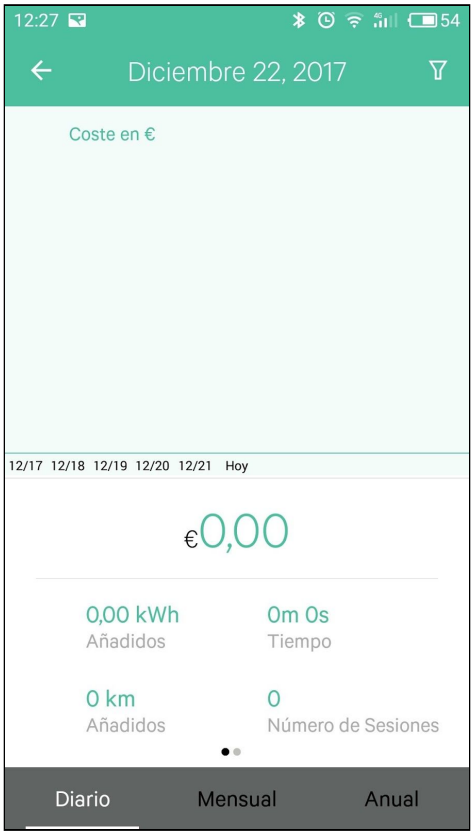


Ilustración 79: Registros diarios de cargas.

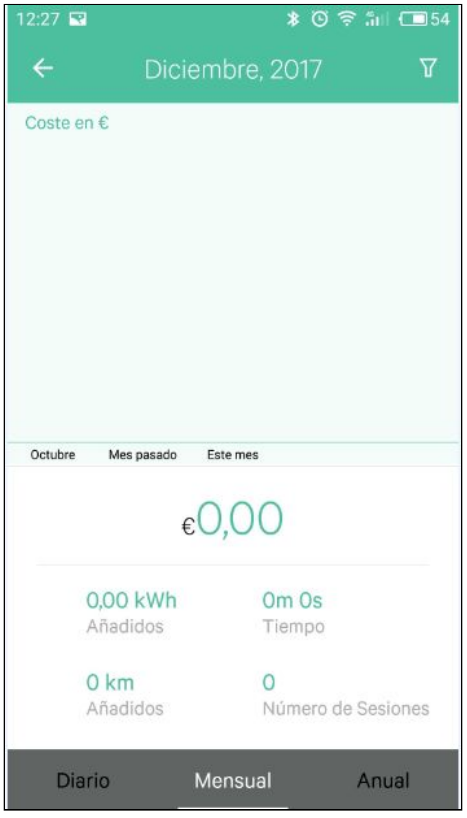


Ilustración 80: Registro mensuales de carga

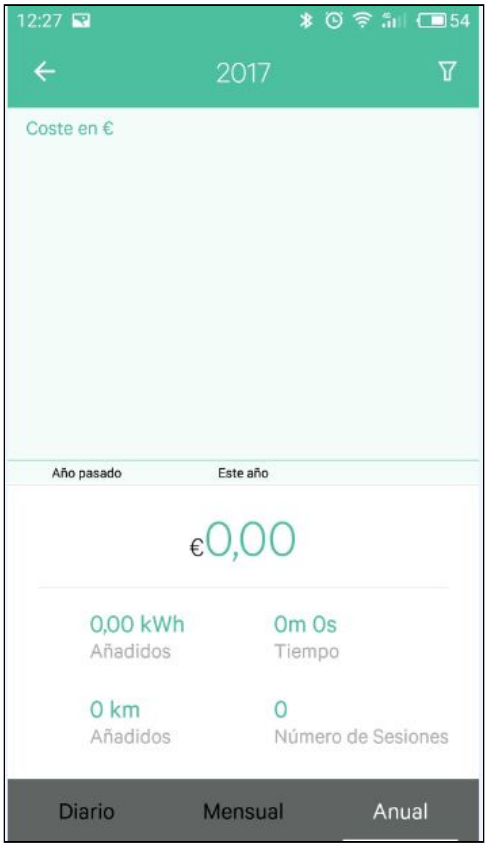


Ilustración 81: Registro anual de cargas

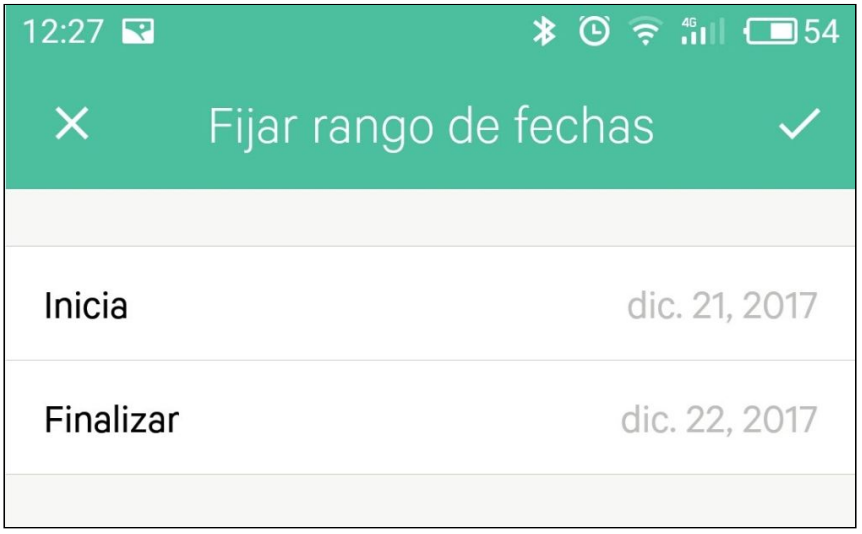


Ilustración 82: Filtrado de registros de cargas por fecha

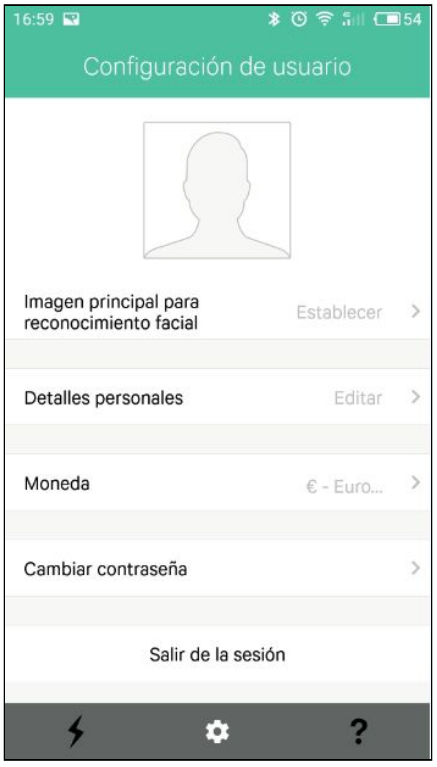


Ilustración 83: Configuración de los datos de usuario

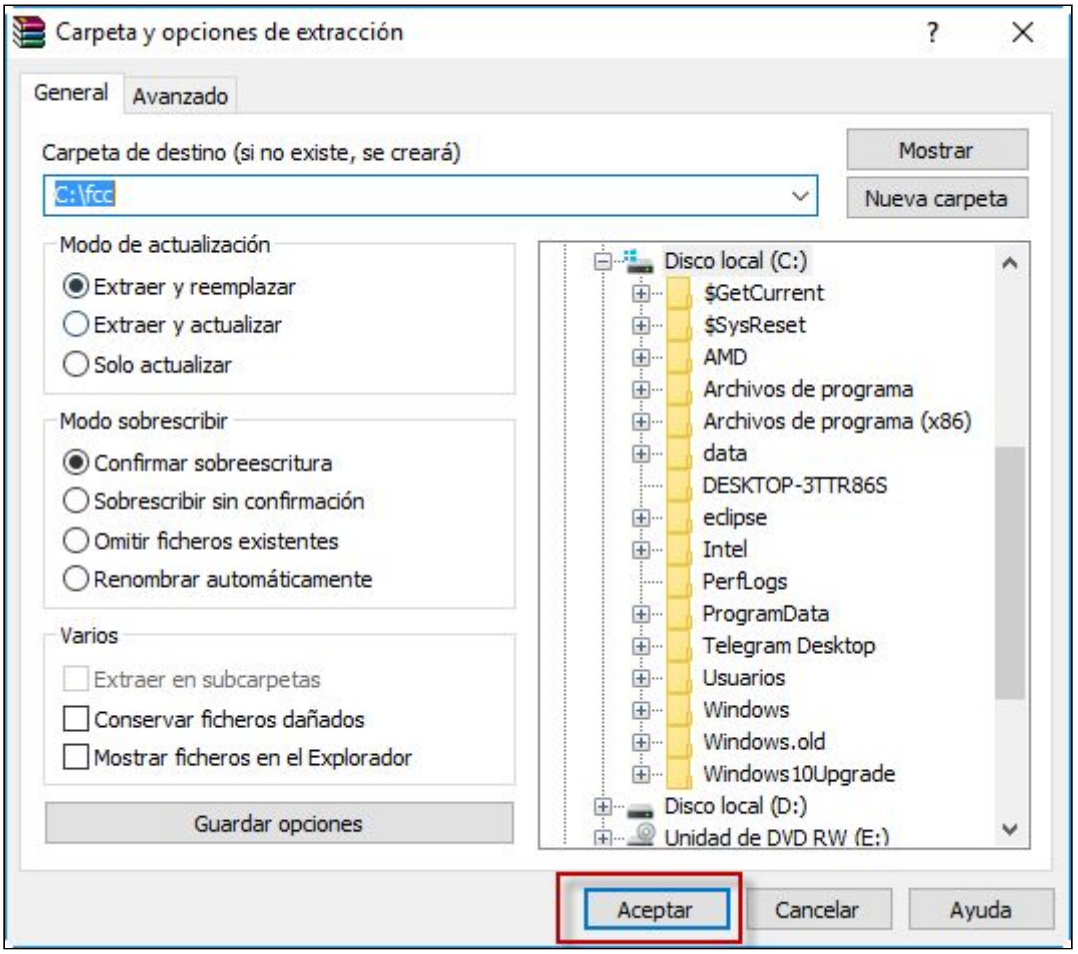
Ilustración 84: Sección de ayuda

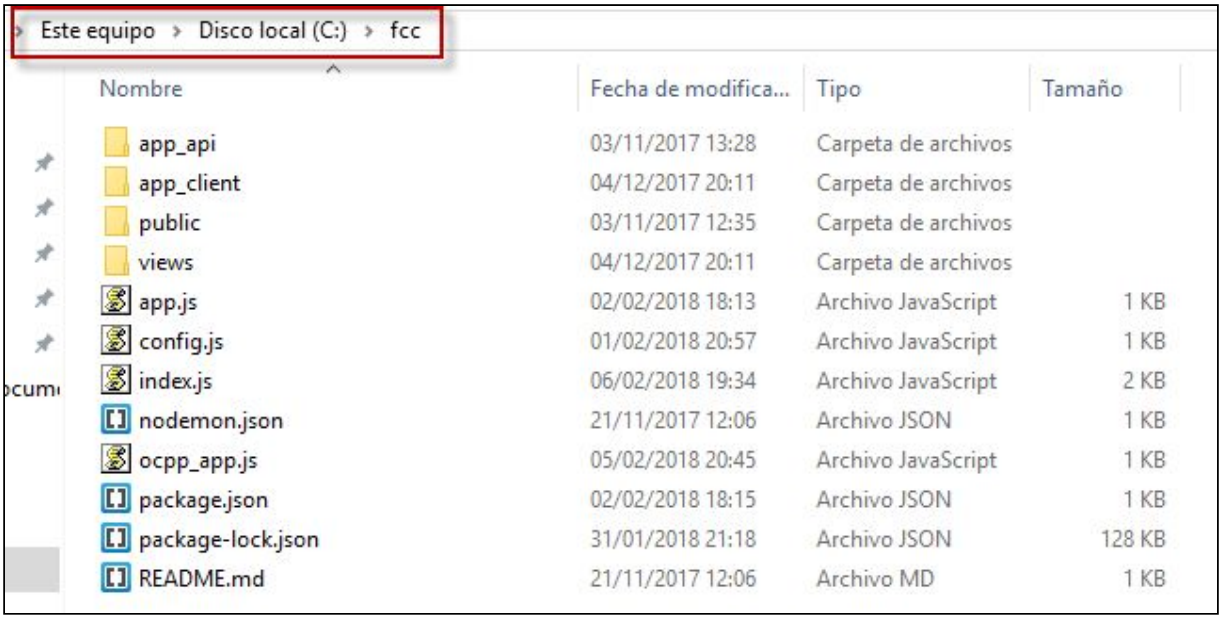
Todas las ilustraciones de la aplicación pertenecen a la versión 3.0 que data del día 21/12/2017.

ANEXO II: Instalación plataforma para continuación desarrollo.

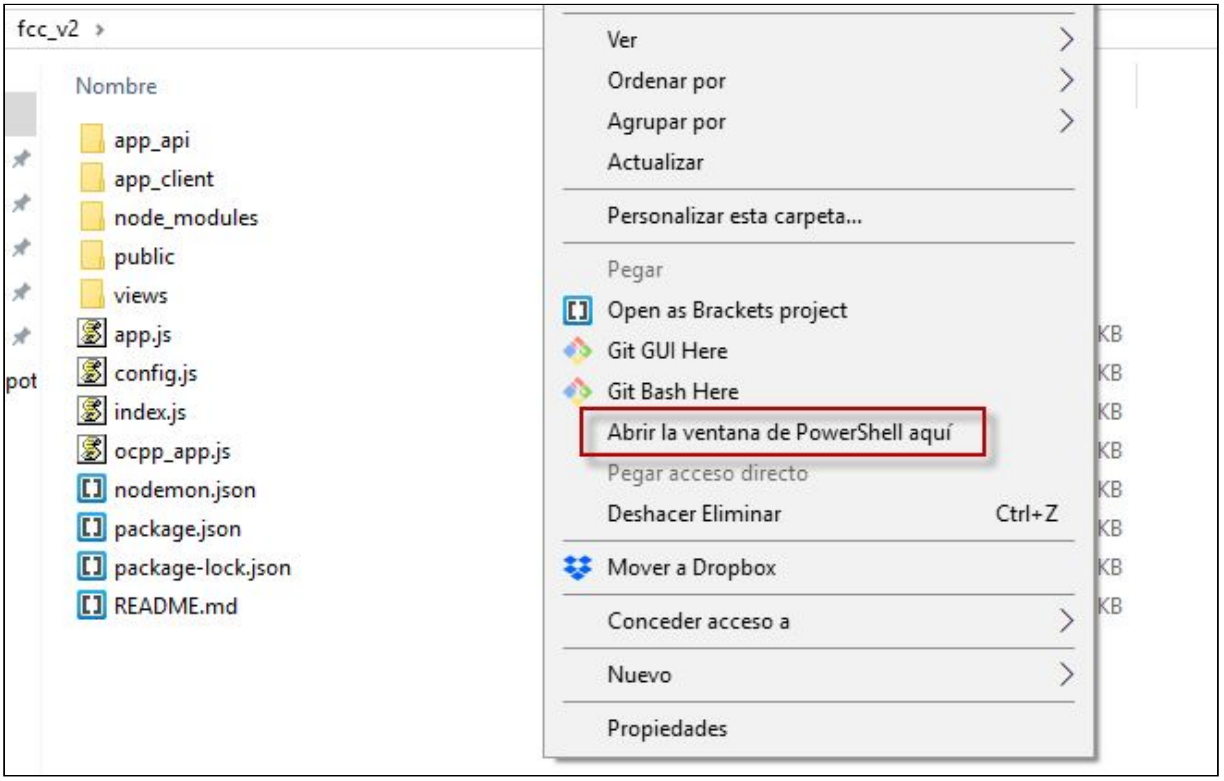
Se deben seguir los siguientes pasos:

- Descargamos:
  - NodeJS en su última versión.
  - MongoDB.
- Una vez se instale, se descomprime el código en cualquier carpeta (por ejemplo, en C:):

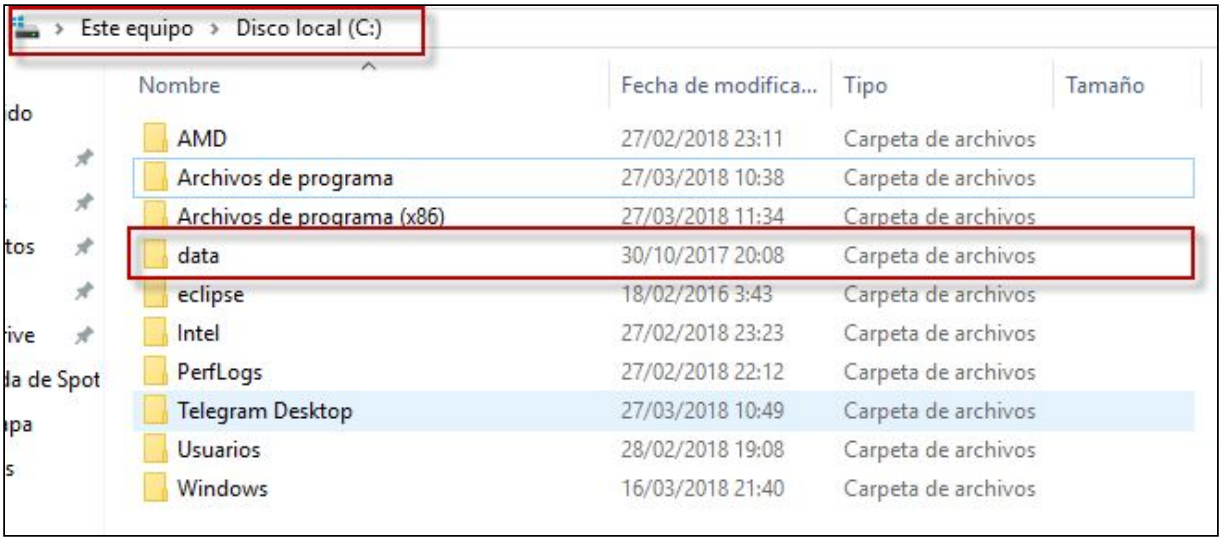




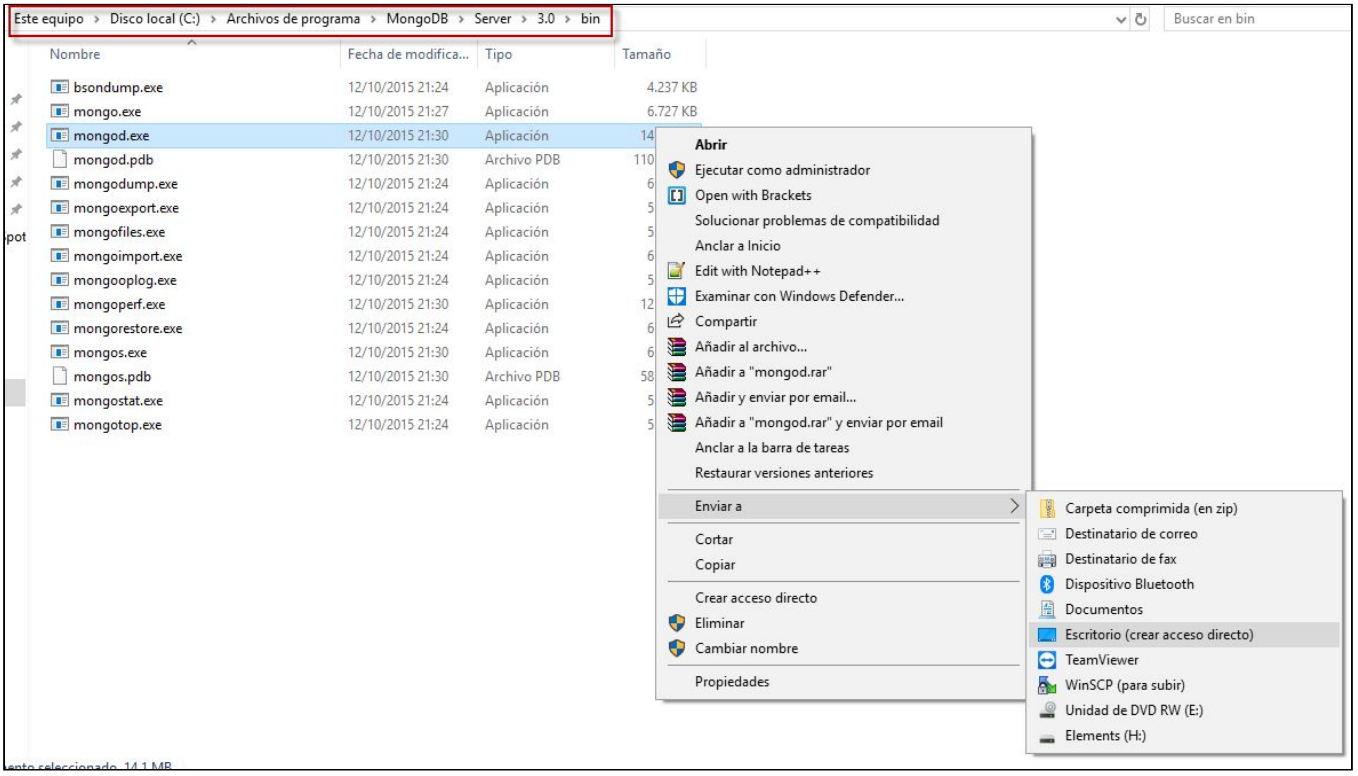
- Mediante la combinación (Mayús + clic derecho), abrimos una ventana de comandos en la carpeta en la que se ha descomprimido el código.



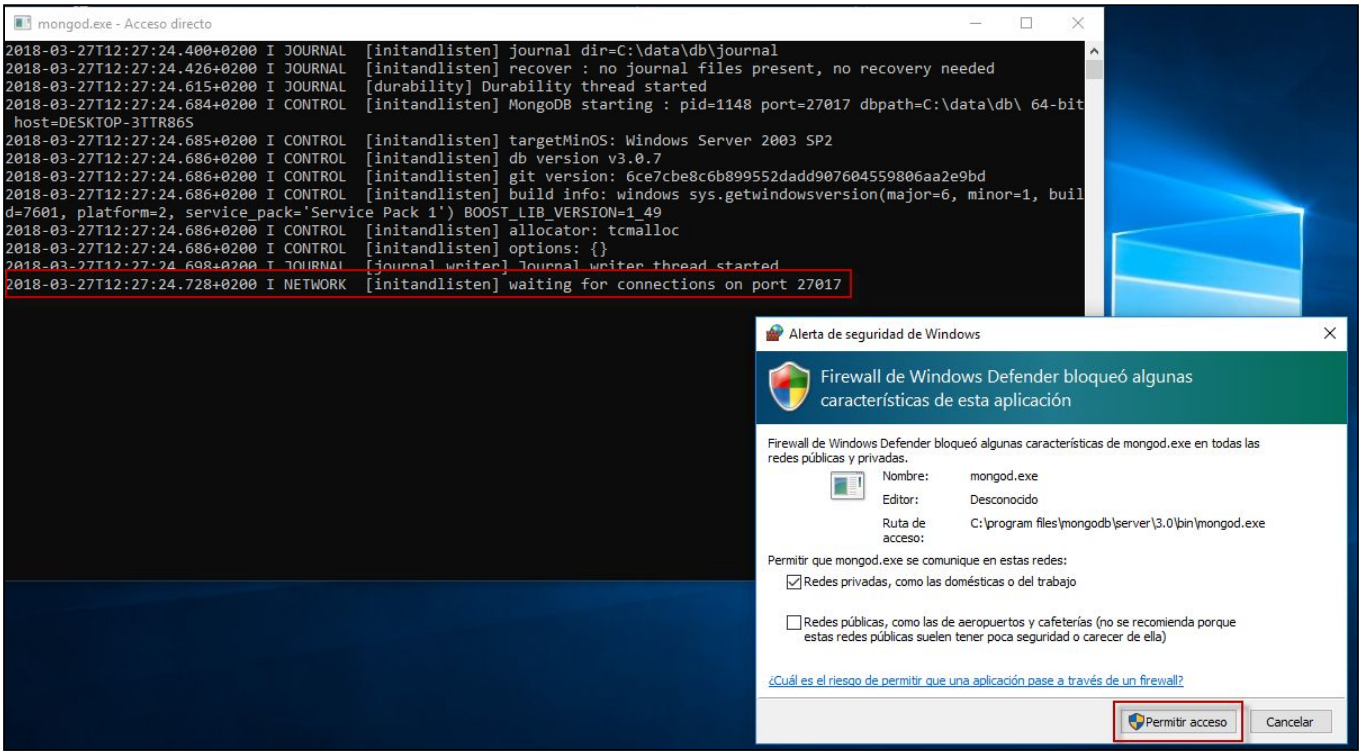
- Ahora, extraemos la BBDD que se ha utlizado para el proyecto y la descomprimos en el disco C:\



- Nos vamos a la ruta donde se ha instalado MongoDB y hacemos un acceso directo en el Escritorio de *mongod.exe*:



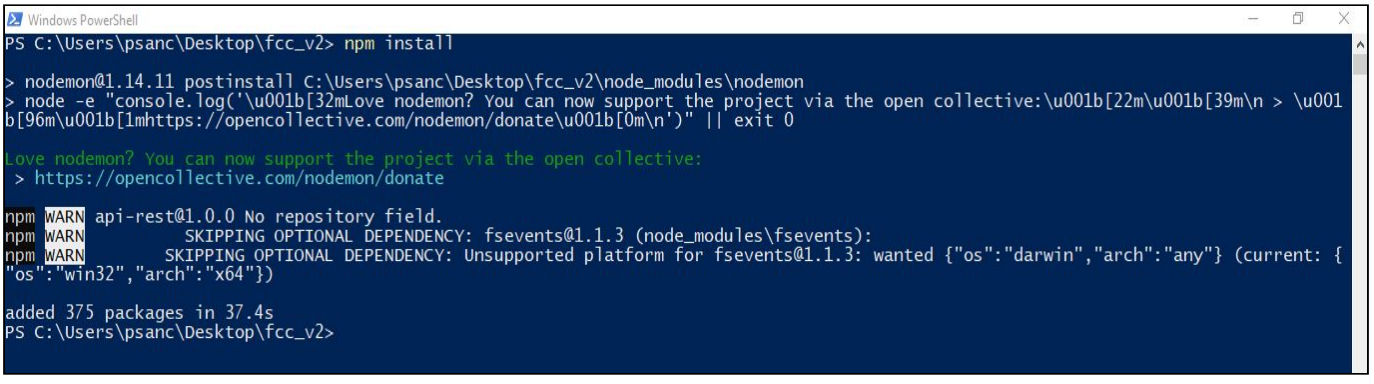
- Seguidamente, hacemos doble clic en el acceso directo creado en el Escritorio:

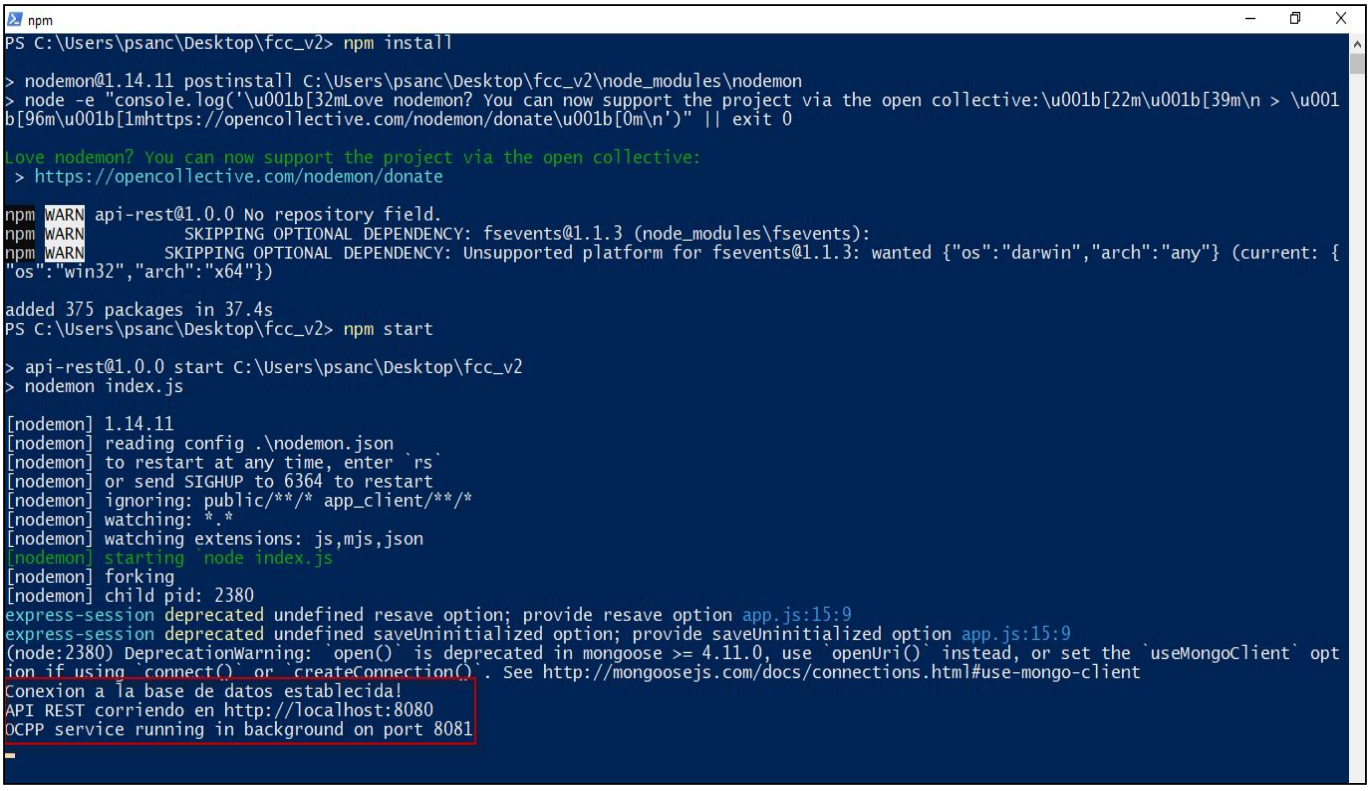


- Lo siguiente es introducir los siguientes comandos en la ventana *PowerShell* (o por defecto *ventana de comandos*, que es como se llama en el S.O. *Windows 7*):

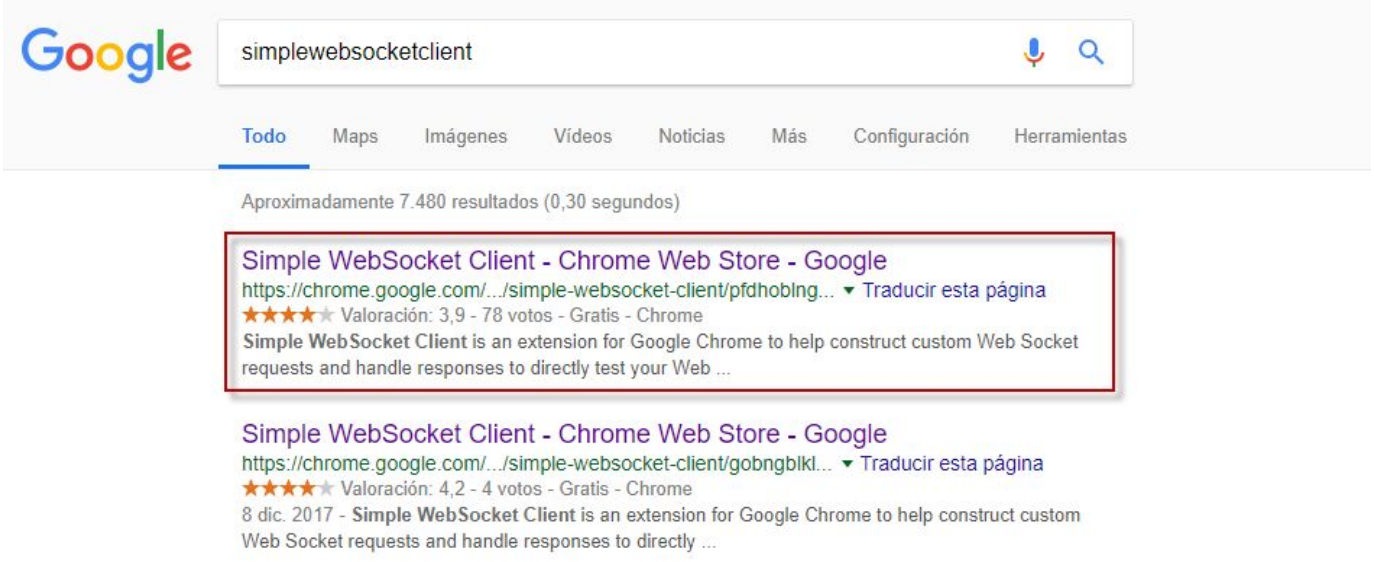
  - **npm install**
  - **npm start**

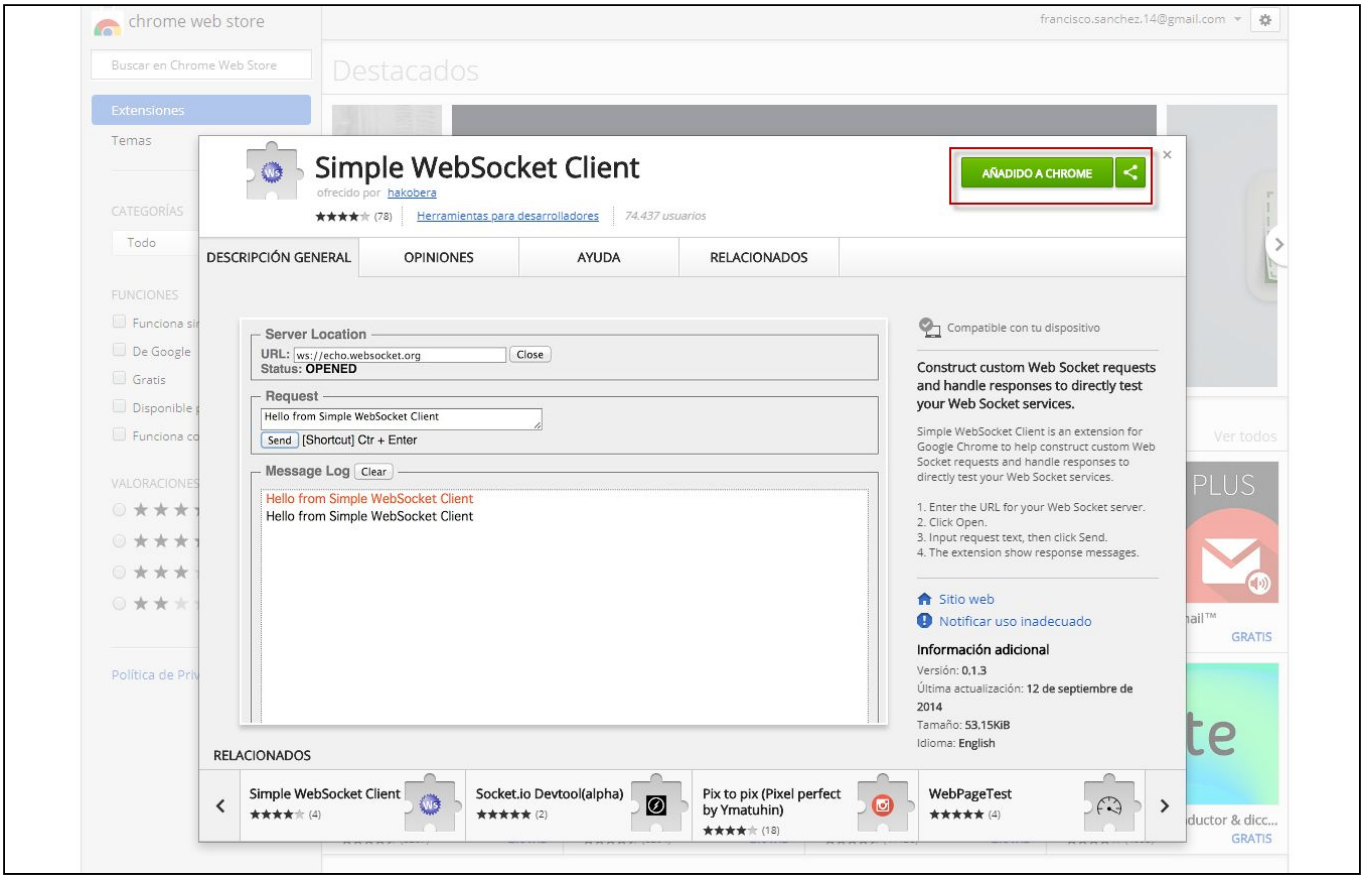
  El primero instala todas las dependencias del proyecto. El segundo arrancará la aplicación (por defecto la dirección es ***localhost:8080*** para la aplicación y ***localhost:8081/ocpp/wallbox-sn2197*** para el módulo OCPP). Cuando queramos pararlo, simplemente presionamos las teclas Ctrl+C a la vez y después tecleamos "S".





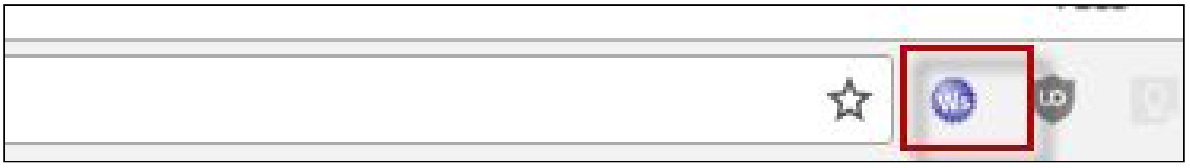
- Una vez que aparezcan los mensajes de que tanto la aplicación como servicio OCPP están activos, se instala la extensión ***SimpleWebSocketClient***. Para ello, entramos en el navegador *Google Chrome* y la buscamos en la página principal:



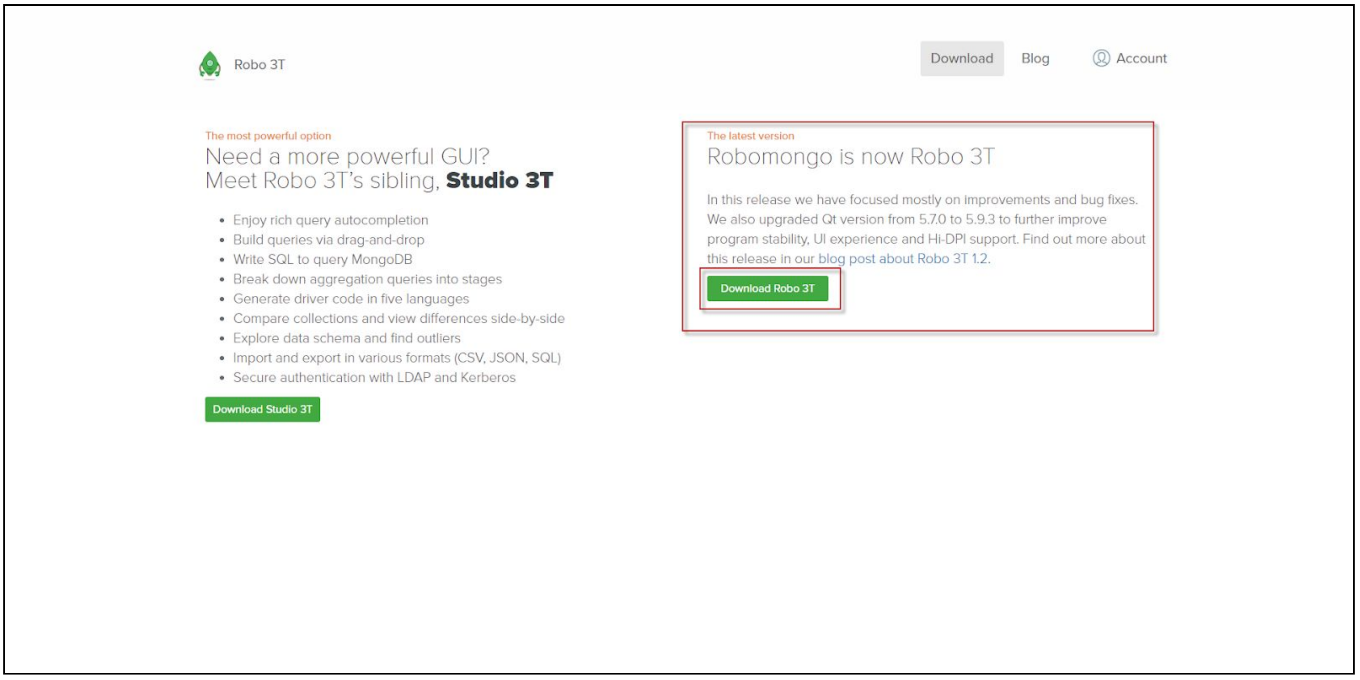


En este caso ya está instalada. No obstante, se pulsa en *Instalar* y esperamos a que se instale con éxito.

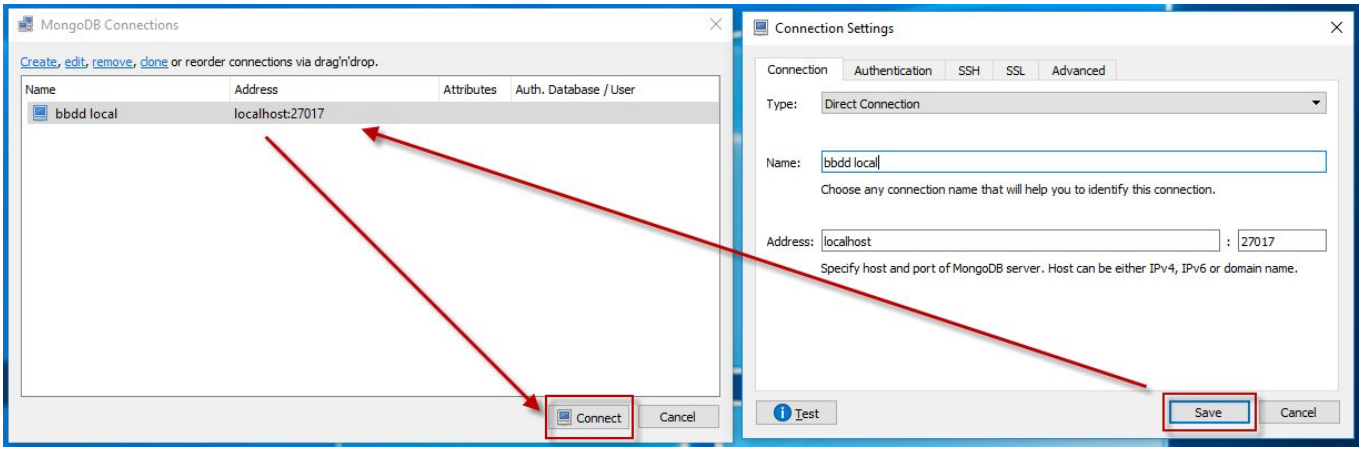
Una vez instalada, hacemos clic en el icono de la barra de navegación para comprobar que se puede acceder a la extensión instalada:



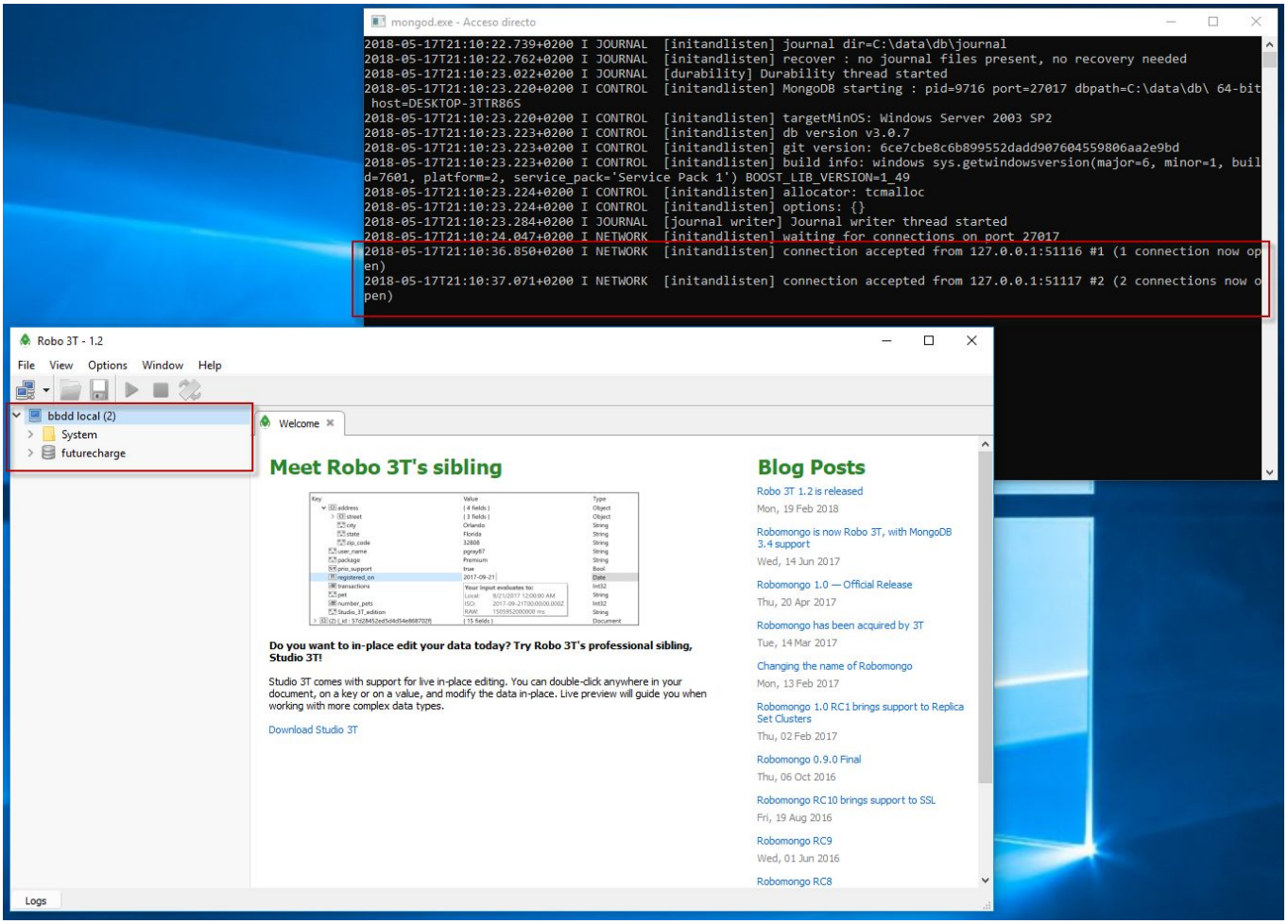
Si queremos un programa para visualizar el contenido de la base de datos, se puede instalar ***Robo3T***. Descargamos la versión adecuada (recomendada 64 bits por mejor rendimiento, en caso contrario la de 32 bits es compatible con cualquier ordenador) y la instalamos:



Una vez instalado, se abrirá de forma automática un panel donde podremos configurar las conexiones a las bases de datos que queramos utilizar (pueden ser locales o remotas (alojadas de forma externa como un *Database as a Service*, *DaaS*). Para nuestro caso, realizamos una configuración sencilla para nuestra base de datos local:



Al conectarnos, debe aparecer en la ventana de comandos que se han abierto varias conexiones a la base de datos, así como que en *Robo3T* podemos visualizar el contenido de la misma. A partir de este punto, se puede *probar* el funcionamiento.



7. Bibliografía

Empresa Solvent: **https://www.solventie.es/**

Conectores:
https://www.motorpasion.com/coches-hibridos-alternativos/tipos-de-conectores-tipos-de-recarga-y-modos-de-carga

DataSheets

Ingeteam:
https://www.ingeteam.com/es-es/sectores/movilidad-electrica/p15_58_163/ingerev-garage.aspx

Wallbox: **https://www.wallbox.com/es/productos/WBCOMM01.html**

Fenie:
https://www.fenieenergia.es/movilidad-electrica/#infraestructura-de-carga

Software y librerías utilizadas para el desarrollo

XAMPP. Montaje servidores locales: **https://www.apachefriends.org/es/index.html**

Robo 3T. Interfaz gráfica para base de datos: **https://robomongo.org/download**

Editor Brackets: **http://brackets.io/**

Postman, API Development: **https://www.getpostman.com/**

mLab, DaaS (*Database as a Service*) para MongoDB: **https://mlab.com/**

NodeJS. Potente plataforma para crear servidores backend: **https://nodejs.org/es/**

- Versión utilizada: **https://nodejs.org/dist/v8.9.4/node-v8.9.4-x64.msi**

AngularJS. Tecnología *front-end* para servidores *backend*: **https://angular.io/**

MongoDB. Plataforma para crear bases de datos: **https://www.mongodb.com/**

- Versión utlizada**:**
  https://www.dropbox.com/s/ito6ly19dnltouj/mongodb-win32-x86_64-3.0.7-signed.msi?dl=0

Git. Programa para gestión de repositorios: **https://git-scm.com/downloads**

Protocolo OCPP

http://www.openchargealliance.org/ → Open Charge Alliance (OCA)

http://www.openchargealliance.org/protocols/ocpp/ocpp-15/ → OCPP 1.5

http://www.openchargealliance.org/protocols/ocpp/ocpp-16/ → OCPP 1.6

http://www.openchargealliance.org/protocols/ocpp/ocpp-20/ → OCPP 2.0

http://www.openchargealliance.org/referenties/elaadnl/ → Elaad-NL

https://www.elaad.nl/nieuws/wat-is-ocpp/ → OCPP según Elaad-NL

http://www.etecnic.es/noticias/breve-resumen-sobre-el-protocolo-de-comunicacio-ocpp/ → Breve introducción al protocolo OCPP

http://www.frontandback.org/back_end/ocpp_protocolo_abierto_puntos_carga → Breve introducción al protocolo OCPP (2)

https://es.wikipedia.org/wiki/Open_Charge_Point_Protocol → Definición protocolo OCPP

Estado del arte de los lenguajes de programación web y bases de datos

https://es.wikipedia.org/wiki/SGML → Previo a HTML

https://www.w3.org/standards/techs/html#w3c_all → HTML

https://www.w3.org/standards/webdesign/htmlcss → HTML & CSS

https://developer.mozilla.org/es/docs/Web/ → HTML, CSS, JavaScript

https://getbootstrap.com/docs/4.0/getting-started/introduction/ → Bootstrap

https://angular.io/guide/quickstart → Angular

https://nodejs.org/es/docs/ → NodeJS

http://expressjs.com/es/ → ExpressJS

https://docs.mongodb.com/ → MongoDB

https://www.beeva.com/beeva-view/tecnologia/interfaces-graficas-para-gestion-y-administracion-de-mongodb/ → MongoDB. Interfaces gráficas.

https://www.w3schools.com/js/js_json_xml.asp → Diferencias XML / JSON

General

RFID: **https://es.wikipedia.org/wiki/RFID**