Grado en ingeniería informática

SISTEMAS DISTRIBUIDOS

EVCharging Network

(grupo 8 de prácticas)

Universitat d'Alacant
Universidad de Alicante

Pablo Bejarano Escolano 74445240J

Abdallah Shaitani Y7525414N

2025/2026

ÍNDICE

Introducción

En esta memoria vamos a abordar diversos puntos de nuestra práctica, desde cómo se han manejado los datos, pasando por la estructura de topics usada, clases involucradas con sus respectivos métodos, y finalmente una guía para desplegar esta práctica en los laboratorios del aula.

Base de datos

Nuestra base de datos será simple, constará de dos “tablas” que se manejarán cada una mediante un archivo JSON, acceciendo a sus datos mediante la central.
Las “tablas” serán las siguiente:

charging_points (representa los diferentes puntos de carga):

- **id:** identificador del punto de carga
- **status:** condición operativa general del punto de carga
- **posX:** posición X del punto de carga
- **posY:** posición Y del punto de carga
- **priceEurKwh:** precio del kilovatio hora
- **state:** estado funcional en tiempo real
- **lastSeen:** última comunicación con el punto de carga
- **connectedVehicleId:** id del conductor que está cargando su coche en este punto de carga
- **currentPowerKw:** capacidad de suministro actual del punto de carga
- **totalEnergySuppliedKwh:** electricidad total suministrada en la sesión
- **currentChargingCost:** coste total de la sesión en función de la energía suministrada y el precio del kilovatio hora

DR (representa los diferentes conductores):

- **id:** identificador del conductor
- **gmail:** correo electrónico del conductor por donde recibe la factura de central
- **name:** nombre del conductor

Topics

Por una parte, tenemos los siguientes topics:
charging.requests:
Producto/esr: EV_Driver
Consumidor/es: EV_Central
Propósito: El conductor solicita a central un punto de carga.

charging.authorizations:
Productor: EV_Central
Consumidor: EV_CP_Engine
Propósito: Central autoriza a cierto punto de carga a cargar el coche del driver solicitante.

charging.telemetry:
Productor/es: EV_CP_Engine
Consumidor/es: EV_Central, EV_Driver
Propósito: Se transmiten los datos de telemetría en tiempo real durante la carga

charging.notifications:
Productor/es: EV_Central
Consumidor/es: EV_Driver
Propósito: Se notifica al conductor acerca del estado de su solicitud.

cp.status:
Productor/es: EV_CP_Engine
Consumidor/es: EV_Central
Propósito: Actualiza el estado de los puntos de carga en central

Por otra parte, el flujo de comunicación sería el siguiente:
Primero el conductor solicita a central que le permita cargar su coche mediante el topic “charging.requests”. Una vez central ha hecho las comprobaciones necesarias le indica a un punto de carga mediante el topic “charging.authorizations” que puede proceder a la carga del vehículo solicitante. Simultáneamente, central usa el topic “charging.notifications” para notificar al conductor si su solicitud puede atenderse correctamente o no. Durante este proceso, el punto de carga usa el topic “charging.telemetry” para mantener informados tanto al conductor como a la central acerca del proceso de carga. Adicionalmente, el punto de carga va usando el topic “cp.status” para mantener actualizada a central acerca de su estado. Por último, central usa el topic “charging.notifications” para informar al conductor de que ya ha terminado de cargar, y le envía la correspondiente factura.

Clases

CentralUI

Esta clase nos proporcionará la interfaz gráfica de la central. Está compuesta por los siguientes métodos:
setupUI(): Configura la interfaz completa: barra de estado, tabla de CPs, área de logs y botones.
appendLog(): Añade mensajes al log con auto-scroll.
updateCPCount(): Actualiza el contador de puntos de carga conectados.
updateDriverCount(): Actualiza el contador de conductores registrados.
addOrUpdateCP(): Añade o actualiza filas de CP en la tabla, manejando IDs únicos.
removeCP(): elimina un CP de la tabla, dado su ID.
CPTableCellRenderer: clase interna que personaliza los colores de las filas según el estado del CP.

DriverUI

Esta clase nos proporcionará la interfaz gráfica del conductor. Está compuesta por los siguientes métodos:
setupUI(): Configura la interfaz: barra de estado, área de logs, panel de carga (progreso, energía, costo) y botones de control.
requestCharging(): Solicita carga al CP ingresado en el campo de texto. Valida que el ID no esté vacío.
loadFile(): Abre diálogo para cargar archivo con múltiples solicitudes de carga. Habilita el botón "Start Queue".
startQueue(): Inicia el procesamiento secuencial de todas las solicitudes cargadas desde el archivo.
appendLog(): Añade mensajes al área de logs con auto-scroll.
updateStatus(): Actualiza la etiqueta de estado principal.
updateChargingInfo(): Actualiza el progreso de carga en tiempo real: barra de progreso y energía consumida.
updateFinalTicket(): Muestra el ticket final con la energía total y el coste cuando termina la carga.
resetChargingInfo(): Reinicia todos los indicadores de carga a valores por defecto.
enableQueueButton(): Re-habilita el botón "Start Queue" cuando termina una cola de solicitudes.

CPEngineUI

Esta clase nos proporcionará la interfaz gráfica del monitor del CP. Está compuesta por los siguientes métodos:
setupUI(): Configura interfaz: información del CP, panel de sesión (conductor, potencia, energía), área de logs y botones de control.
setStopButtonListener(): Conecta el botón "Stop Charging" al motor del CP.
setFaultButtonListener(): Conecta el botón "Simulate Fault" al motor del CP.
appendLog(): Añade mensajes al área de logs con auto-scroll.
updateStatus(): Actualiza estado del CP con icono y color: Disponible(verde), Cargando(azul), Roto(rojo).
startCharging(): Inicia sesión de carga: muestra al conductor, potencia y reinicia contadores.
updateChargingProgress(): Actualiza progreso de carga: energía suministrada y porcentaje en barra.
stopCharging(): Finaliza la sesión: reinicia la interfaz al estado disponible.
showFault(): Muestra estado de falla: reinicia todos los valores y se marca como "BROKEN".

Driver

Clase simple que representa a un driver. El objetivo de esta clase es almacenar los drivers en la central, con la información extraída del JSON. Sus métodos son básicamente simples getters y setters principalmente por lo que no los comentaremos.

ChargingPoint

Clase simple que representa un charging point. El objetivo de esta clase es almacenar los charging points en la central, con la información extraída del JSON. Sus métodos son básicamente simples getters y setters principalmente por lo que no los comentaremos.

EV_Central

Clase que coordina todos los componentes, accediendo a la base de datos, comunicando los CP con los drivers y haciendo las validaciones pertinentes. Está compuesta por los siguientes métodos:
setUI(): Establece referencia a la UI para actualizaciones visuales.
log(): Registra mensajes en consola y los envía a la UI si está disponible.
logError(): Registra errores en consola y los envía a la UI con prefijo "[ERROR]".
loadChargingPointsFromJSON(): Carga todos los puntos de carga desde el archivo JSON a memoria y a la UI.
loadDriversFromJSON(): Carga todos los conductores desde el archivo JSON y actualiza contador en el UI.
saveChargingPointsToJSON(): Guarda todos los puntos de carga al archivo JSON.
updateChargingPoint(): Actualiza punto de carga en memoria y guarda los cambios en el JSON y el UI.
authenticateChargingPoint(): Valida que un punto de carga exista en la base de datos.
authenticateDriver(): Valida que un conductor exista en la base de datos.
getChargingPointById(): Busca punto de carga por ID en la base de datos.
initializeKafka(): Configura producer y consumer de Kafka para comunicación asíncrona.
startKafkaConsumerThread(): Hilo que consume mensajes Kafka de solicitudes, telemetría y estados.

handleKafkaMessage(): Dirige mensajes Kafka al manejador correspondiente según el topic.
handleChargingRequest(): Procesa solicitudes de carga: valida al conductor y CP, envía autorizaciones.
handleTelemetry(): Procesa telemetría: actualiza el estado, energía, coste y notifica a conductores.
handleCPStatus(): Actualiza el estado de puntos de carga recibido vía Kafka.
sendToKafka(): Envía mensajes a topics Kafka de forma genérica.
sendNotification(): Envía notificaciones a conductores vía Kafka.
startAutosaveThread(): Hilo que guarda automáticamente la base de datos cada 30 segundos.
start(): Inicia el servidor: carga datos, configura Kafka, abre Socket y arranca hilos.
stop(): Detiene el servidor: guarda la base de datos y cierra todas las conexiones.
ClientHanlder: Clase interna que maneja las conexiones Socket de puntos de carga: autenticación y procesamiento de comandos. Está compuesta por los siguientes métodos:

- **run():** Método principal del hilo: maneja autenticación y procesa mensajes continuos.
- **handleMessage():** procesa los comandos del punto de carga.
- **disconnect():** Limpia recursos al desconectar: actualiza estado y cierra conexión.

EV_Driver

Es el backend de la aplicación del conductor. Maneja toda la comunicación con Kafka: envía solicitudes de carga y recibe notificaciones/telemetría. Está compuesta por los siguientes métodos:
startWithUI(): Inicia la aplicación: configura producer/consumer Kafka, inicia el hilo consumidor y enlaza con UI.
startConsumerThread(): Crea un hilo que consume mensajes Kafka de notificaciones y telemetría en bucle.
handleMessage(): Dirige mensajes al manejador correspondiente según el topic.
handleNotification(): Procesa notificaciones: autorizaciones, denegaciones, inicio/fin de carga.
handleTelemetry(): Procesa la telemetría: actualiza progreso de carga en el UI en tiempo real.
scheduleNextRequest(): Programa la siguiente solicitud después de 4 segundos (para colas).
requestChargingInternal(): Envía solicitud de carga a central vía Kafka.
requestChargingManual(): Método público para el UI: solicita cargar en un CP específico.
loadRequestsFromFile(): Carga la lista de solicitudes desde un archivo para procesarlas por cola.
startRequestQueue(): Inicia el procesamiento secuencial de todas las solicitudes cargadas.
log(): Envía logs a UI o consola.
stop(): Detiene aplicación: cierra hilos y conexiones Kafka.

EV_ChargingPoint

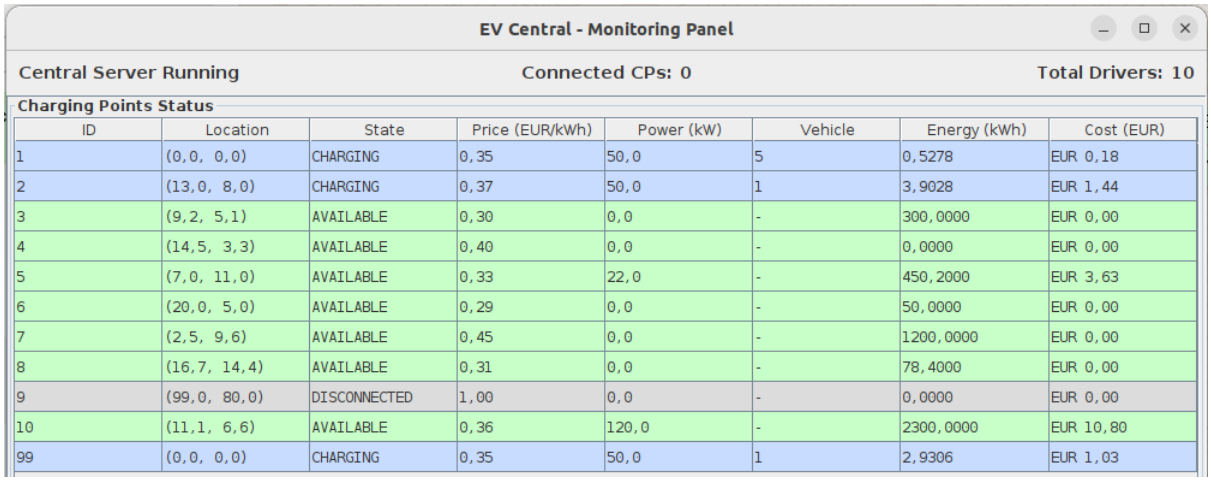
Cliente de consola que simula un punto de carga físico, conectándose al servidor central vía Socket TCP. Está compuesta por los siguientes métodos:
connectAndAuthenticate(): Conecta vía Socket y autentica con el servidor central.
sendMessage(): Envía mensajes al servidor central a través del Socket.

startListening(): Inicia hilo que escucha respuestas del servidor central en tiempo real.
startChargingSimulation(): Simula carga: envía actualizaciones de energía cada segundo al servidor.
stopChargingSimulation(): Detiene simulación de carga y reinicia contadores de energía.
runInteractiveMenu(): Menú interactivo por consola para probar todas las funcionalidades del punto de carga.
handleStartCharging(): Maneja inicio de carga: solicita el ID del vehículo y potencia.
handleStopCharging(): Maneja parada de carga: detiene la simulación y envía.
handleUpdatePosition(): Actualiza las coordenadas X,Y del punto de carga.
handleUpdatePrice(): Modifica precio por kWh.
handleChangeState(): Cambia estado funcional.
handleUpdateStatus(): Actualiza descripción de estado operativo.
handleSimulateFault(): Simula una falla: para la carga y se marca como BROKEN.
disconnect(): Cierra la conexión Socket con el servidor central.

EV_CP_Engine

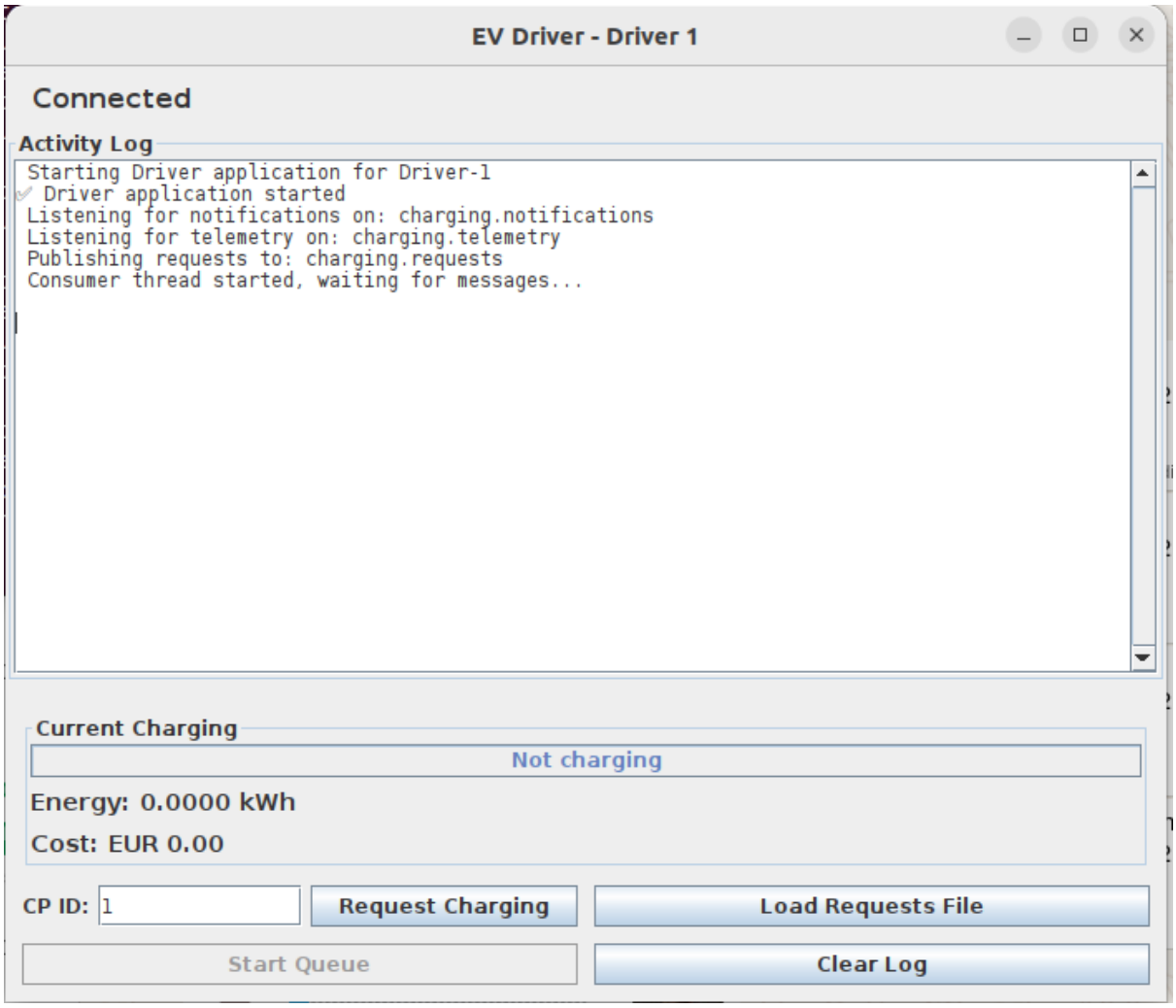
Backend automatizado del punto de carga que se comunica con el servidor central via Kafka. Es la versión automatizada de EV_ChargingPoint. Está compuesta por los siguientes métodos:
setUI(): Establece referencia a la UI para actualizaciones visuales.
start(): Inicia el motor: configura producer/consumer Kafka, envía el estado inicial y arranca hilos.
startConsumerThread(): Hilo que consume mensajes de autorización del central en bucle continuo.
handelAuthorizationMessage(): Procesa autorizaciones: comprueba si es para este CP e inicia carga automáticamente.
startCharging(): Inicia sesión de carga: actualiza el estado, envía START y arranca hilo de telemetría.
startTelemetryThread(): Hilo que envía datos de telemetría cada segundo durante la carga.
stopCharging(): Detiene sesión de carga: envía STOP, actualiza estado y reinicia variables.
sendTelemetry(): Envía datos de telemetría al topic “charging.telemetry”.
sendStatusUpdate(): Reporta el estado actual del CP al topic “cp.status”.
simulateFault(): Simula una falla: para la carga y se marca como BROKEN.
log(): Envía logs a UI o consola.
stop(): Detiene motor: cierra hilos y conexiones Kafka de forma segura.

Front-end

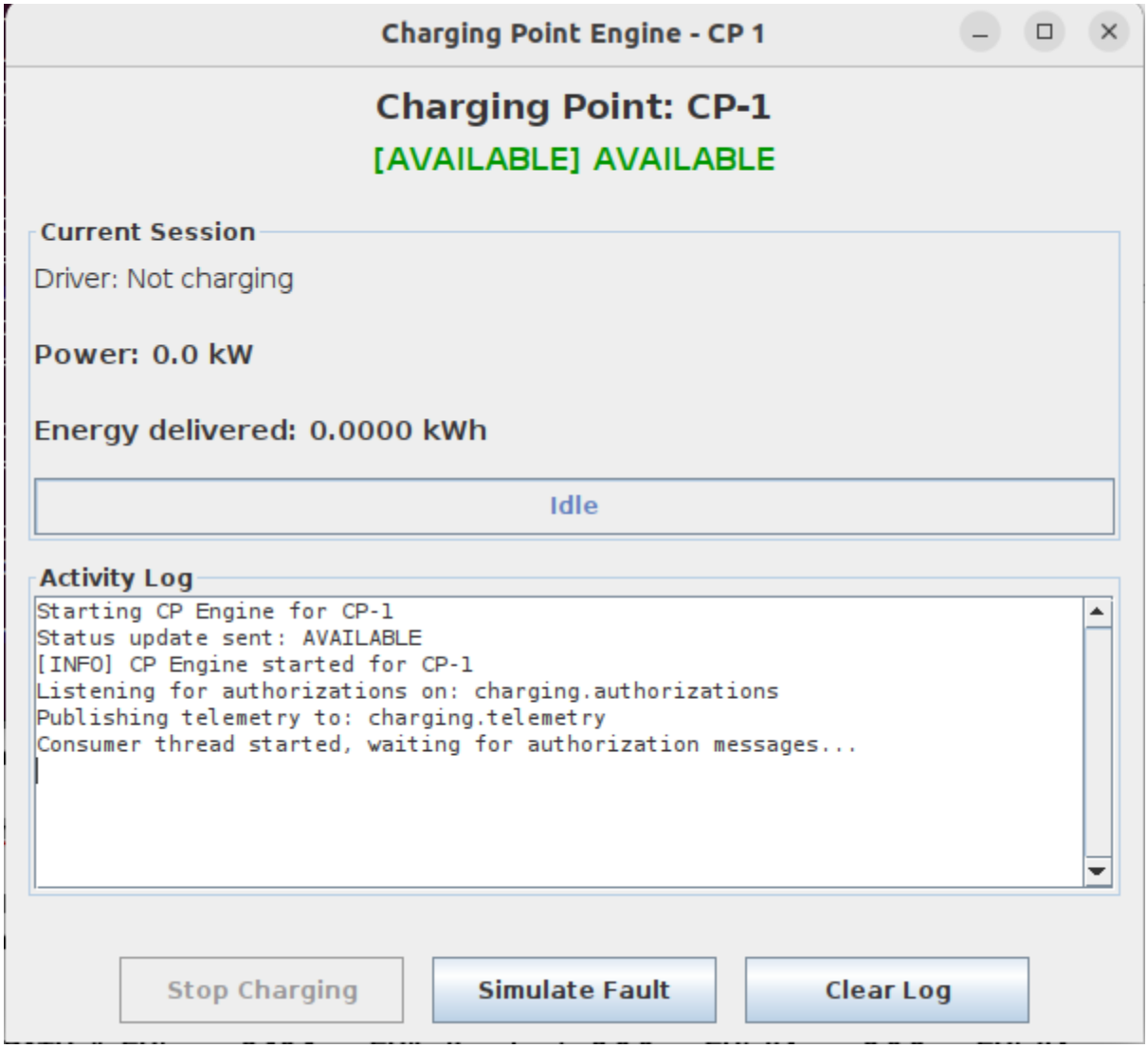


ID	Location	State	Price (EUR/kWh)	Power (kW)	Vehicle	Energy (kWh)	Cost (EUR)
1	(0,0, 0,0)	CHARGING	0,35	50,0	5	0,5278	EUR 0,18
2	(13,0, 8,0)	CHARGING	0,37	50,0	1	3,9028	EUR 1,44
3	(9,2, 5,1)	AVAILABLE	0,30	0,0	-	300,0000	EUR 0,00
4	(14,5, 3,3)	AVAILABLE	0,40	0,0	-	0,0000	EUR 0,00
5	(7,0, 11,0)	AVAILABLE	0,33	22,0	-	450,2000	EUR 3,63
6	(20,0, 5,0)	AVAILABLE	0,29	0,0	-	50,0000	EUR 0,00
7	(2,5, 9,6)	AVAILABLE	0,45	0,0	-	1200,0000	EUR 0,00
8	(16,7, 14,4)	AVAILABLE	0,31	0,0	-	78,4000	EUR 0,00
9	(99,0, 80,0)	DISCONNECTED	1,00	0,0	-	0,0000	EUR 0,00
10	(11,1, 6,6)	AVAILABLE	0,36	120,0	-	2300,0000	EUR 10,80
99	(0,0, 0,0)	CHARGING	0,35	50,0	1	2,9306	EUR 1,03

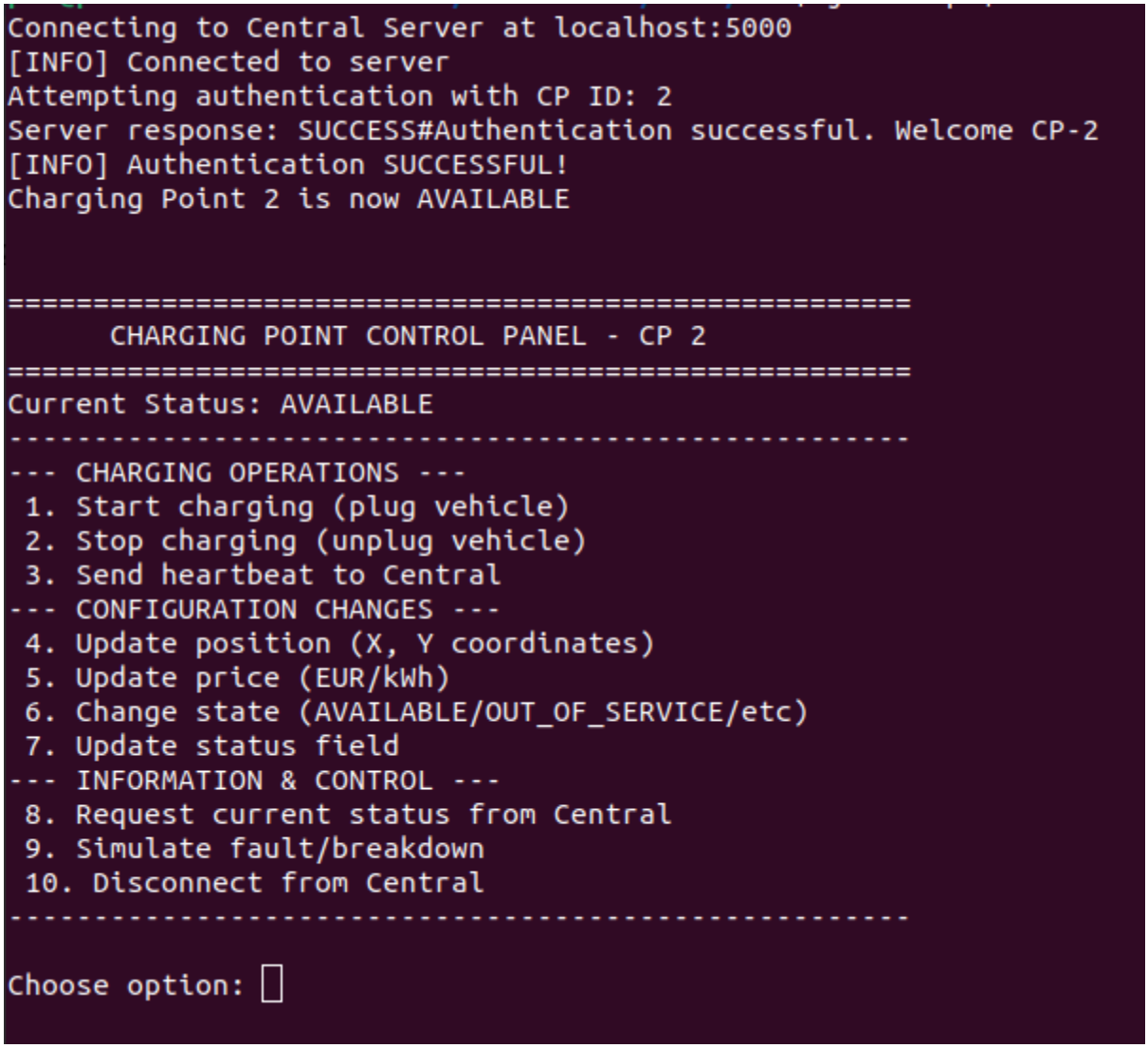
Aquí tenemos un ejemplo del front-end de la central.



Este sería un ejemplo del front-end que usaríamos para los drivers.



Esta parte del front-end correspondería a los charging points.



Por último, esta parte aunque se muestre por terminal la usaríamos para modificar ciertos aspectos de los charging points.

Despliegue

Para desplegar nuestra aplicación con éxito, deberemos seguir los siguientes pasos:
1 - en terminal 1, en src, ejecutar “**make all**”
2 - en terminal 1, en src, ejecutar “**sudo docker-compose up**”
3 - en terminal 2, en src, ejecutar
CPATH=".:EV/gson-2.10.1.jar:EV/kafka-clients-3.6.0.jar:EV/slf4j-api-2.0.9.jar:EV/slf4j-simple-2.0.9.jar"
4 - en terminal 2, en src, ejecutar
CPATH="java -cp $CPATH EV.EV_Central 5000 charging_points.json DR.json localhost:9092”
5 - en terminal 3, en src, ejecutar
CPATH=".:EV/gson-2.10.1.jar:EV/kafka-clients-3.6.0.jar:EV/slf4j-api-2.0.9.jar:EV/slf4j-simple-2.0.9.jar"
6 - en terminal 3, en src, ejecutar
java -cp $CPATH EV.DriverUI localhost:9092 1
7 - en terminal 4, en src, ejecutar
CPATH=".:EV/gson-2.10.1.jar:EV/kafka-clients-3.6.0.jar:EV/slf4j-api-2.0.9.jar:EV/slf4j-simple-2.0.9.jar"
8 - en terminal 4, en src, ejecutar
java -cp $CPATH EV.CPEngineUI localhost:9092 1
9 - en terminal 5, en src, ejecutar
CPATH=".:EV/gson-2.10.1.jar:EV/kafka-clients-3.6.0.jar:EV/slf4j-api-2.0.9.jar:EV/slf4j-simple-2.0.9.jar"
10 - en terminal 5, en src, ejecutar
java -cp $CPATH EV.EV_ChargingPoint localhost 5000 2