**EASYCAB**

SISTEMAS DISTRIBUIDOS

PRÁCTICA 2

Julio Corbalán Moreno

Álvaro Pedreño Rubio

# 

# **ÍNDICE**

[**EC\_CTC 3**](#_heading=h.lnxbz9)

[**Variables Globales 3**](#_heading=h.35nkun2)

[**Funciones 3**](#_heading=h.1ksv4uv)

[**EC\_Registry 5**](#_heading=h.6e20hvitthut)

[**Variables Globales 5**](#_heading=h.ndknob7uclfx)

[**Funciones 5**](#_heading=h.ohxu88n14hh9)

[**Sistema de Auditoría 7**](#_heading=h.rnlgyr45td2l)

[**FRONT 8**](#_heading=h.cpm6h6j574n2)

[**Funciones** 8](#_heading=h.33ej6bw2jrr)

## 

## EC\_CTC

Implementa un servidor que se conecta a la API de OpenWeather, resuelve si las condiciones climatológicas son óptimas para el funcionamiento de la aplicación y devuelve “OK” o “KO” dependiendo del resultado.

#### **Variables Globales**

* LISTEN\_PORT: Puerto de escucha donde un cliente podrá conectarse para conocer el estado climatologico actual de la ciudad.
* CITY\_JSON\_PATH: Ruta del JSON que contiene el nombre de la ciudad a buscar.
* API\_KEY: API KEY usada en OpenWeather, se encuentra en un archivo aparte para mantenerla aislada.

#### **Funciones**

**comprobarArgumentos(argumentos)** Verifica la cantidad de argumentos de línea de comandos al iniciar el programa.

* **Parámetros**:
  + argumentos: Lista de argumentos de línea de comandos.

Si el número de argumentos es incorrecto, muestra un mensaje de error y finaliza el programa.

**asignarConstantes(argumentos)** Asigna el valor del puerto de escucha, asignado usando los argumentos de línea de comandos.

* **Parámetros**:
  + argumentos: Lista de argumentos de línea de comandos.

**leerCiudad()** Abre el archivo donde se encuentra el nombre de la ciudad y lo devuelve.

**obtenerClima(city)** Se conecta a la API de OpenWeather, usando la variable global API\_KEY, haciendo una petición con el nombre de la ciudad y devuelve el resultado obtenido

* **Parámetros**:
  + city: Ciudad a consultar.

**consultarClima()** Maneja y crea una conexión para que un servicio, la central en este caso, pueda consultar el clima en la ciudad marcada. Devuelve “KO” o “OK” dependiendo del resultado obtenido.

* **URL**:
  + /consultarClima/

## EC\_Registry

Implementa un servidor que maneja el registro de los taxis en el sistema.

#### Variables Globales

* LISTEN\_PORT: Puerto de escucha donde un cliente podrá conectarse para conocer el estado climatologico actual de la ciudad.

#### Funciones

**comprobarArgumentos(argumentos)** Verifica la cantidad de argumentos de línea de comandos al iniciar el programa.

* **Parámetros**:
  + argumentos: Lista de argumentos de línea de comandos.

Si el número de argumentos es incorrecto, muestra un mensaje de error y finaliza el programa.

**asignarConstantes(argumentos)** Asigna el valor del puerto de escucha, asignado usando los argumentos de línea de comandos.

* **Parámetros**:
  + argumentos: Lista de argumentos de línea de comandos.

**registrarTaxi(taxi\_id)** Maneja y crea una conexión para que un servicio, en este caso un taxi, pueda registrar un taxi. Comprueba que ese taxi no este previamente registrado en base de datos y envía un mensaje de confirmación o rechazo.

* **Parámetros**:
  + taxi\_id: ID del taxi a registrar.
* **URL**:
  + /registrar/<taxi\_id>

**borrarTaxi(taxi\_id)** Maneja y crea una conexión para que un servicio, en este caso un taxi, pueda borrar un taxi. Comprueba que ese taxi este previamente registrado en base de datos y envía un mensaje de confirmación o rechazo.

* **Parámetros**:
  + taxi\_id: ID del taxi a borrar.
* **URL**:
  + /borrarTaxi/<taxi\_id>

## Sistema de Auditoría

Implementa un registro de todos los eventos que se producen en todos los sistemas de la aplicación de la manera que aquí se muestra.

Primero se ha implementado en el archivo EC\_Shared (servicio que contiene todos los métodos en común que usan todos los servicios que componen en el sistema) un método que escriba en un .TXT una string mandada por parámetros.

Un servicio, al usar printInfo, printWarning, etc (métodos en EC\_Shared) también estará haciendo que ese mensaje que se imprime por pantalla se guarde en los logs, guardando el evento, la ip dónde se produce y parámetros del mismo.

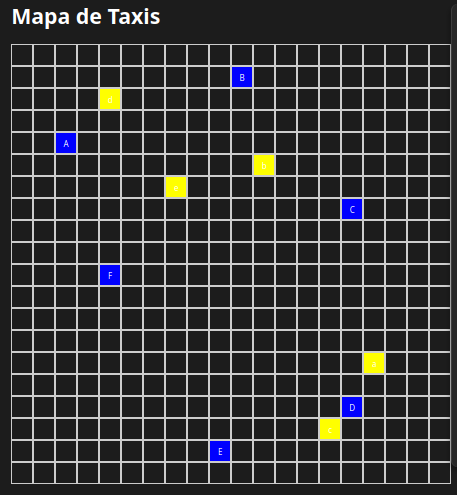
* Auditoría EC\_Central && EC\_Registry:
  + EC\_Central y EC\_Registry escriben en el mismo archivo de log, al estar ejecutados en el mismo servidor, y luego ese archivo de log es el que lee el front para mostrarlo.
* ¿Por qué sólo el FRONT lee ese LOG?
  + El front sólo lee ese log pues EC\_Central maneja todos los eventos del sistema, (drones, clientes, clientes, etc) excepto el Registry, pues son servicios completamente independientes, de ahí el hecho de que escriban en el mismo archivo.

## FRONT

Implementa un servidor usando Node y React para implementar una interfaz gráfica web.

#### Funciones

**Mapa** Se conecta a la API de la Central para consultar el estado. Al hacer la petición a la API, el front recibe el JSON con el estado actual del sistema y lo muestra en una cuadricula que se llamará desde el main.

****

**LOGS** Se conecta a la API de la Central para consultar el estado de los logs. Al hacer la petición a la API, el front recibe un TXT con todos los logs recopilados por la central y registry.

