EASYCAB

SISTEMAS DISTRIBUIDOS

PRÁCTICA 2

Julio Corbalán Moreno

Álvaro Pedreño Rubio

ÍNDICE

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>

EC_Central

Implementa el engine central de la solución del sistema.

Variables Globales

- LISTEN_PORT: Puerto de escucha donde escuchará las conexiones de los taxis
- FRONT_API_PORT: Puerto donde servirá el api para ser consumido por el front-end o otros interesados.
- BROKER_IP: IP donde se encontrará el broker del sistema.
- BROKER_PORT: Puerto de escucha del broker del sistema.
- WEATHER_IP: IP donde se encontrará el CTC del sistema.
- WEATHER_PORT: Puerto de escucha del CTC del sistema.

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.

---

leerBBDD() Lee el estado de la base de datos, para recuperar los datos en caso de una caída de central, se ejecuta al inicio del programa. Incluye una llamada a dbToJSON()

---

dbToJSON() Lee el estado de la base de datos y la envía por el topic TOPIC_ESTADOS_MAPA, para informar al frontend. Se llama cada vez que se realiza una sentencia SQL

---

exportDB() Exporta la base de datos cuando llega una solicitud por el API.

---

ejecutarSentenciaBBDD(sentencia, user, password) Ejecuta la sentencia sql suministrada, con el usuario y la contraseña indicada, y devuelve su resultado en caso de que se produzca uno.

---

gestionarBrokerClientes() Gestiona los servicios solicitados por los clientes, y asigna a los taxis los servicios a realizar.

---

gestionarBrokerTaxis() Gestiona los mensajes enviados por los taxis al broker, respecto a estado, movimiento, estado de los sensores, recogida de cliente, finalización de servicio, etc.

---

comprobarTaxi(idTaxi) Comprueba que el taxi está registrado y no conectado en el intento de conexión

---

verificarTokenTaxi(idTaxi, tokenTaxi) Verifica que el token de un taxi es el correcto cuando se recibe un mensaje.

---

autenticarTaxi(conexion, direccion) Realiza la gestión de la conexión de un taxi. Le envía tanto su último estado como el estado actual del sistema, y el token que deberá utilizar para el envío de mensajes.

---

gestionarTaxi(conexion, direccion) Gestiona la conexión del socket de un taxi una vez ya autenticado, lo que nos permite saber si se encuentra con vida o ha caído. En caso de caída actualiza la base de datos.

---

gestionarLoginTaxis() Escucha las nuevas conexiones por socket de taxis y les crea un hilo para gestionarlos.

---

dirigirABaseATodos() Envía todos los taxis a base, o cancela su envio.

---

dirigirTaxiABase(idTaxi) Envía el taxi indicado a la base, o cancela su envio.

---

inputBase() Gestiona los inputs del usuario para enviar los taxis a base.

---

verificarClima() Verifica que el estado del clima sea el correcto. Actúa según el estado de este.

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.

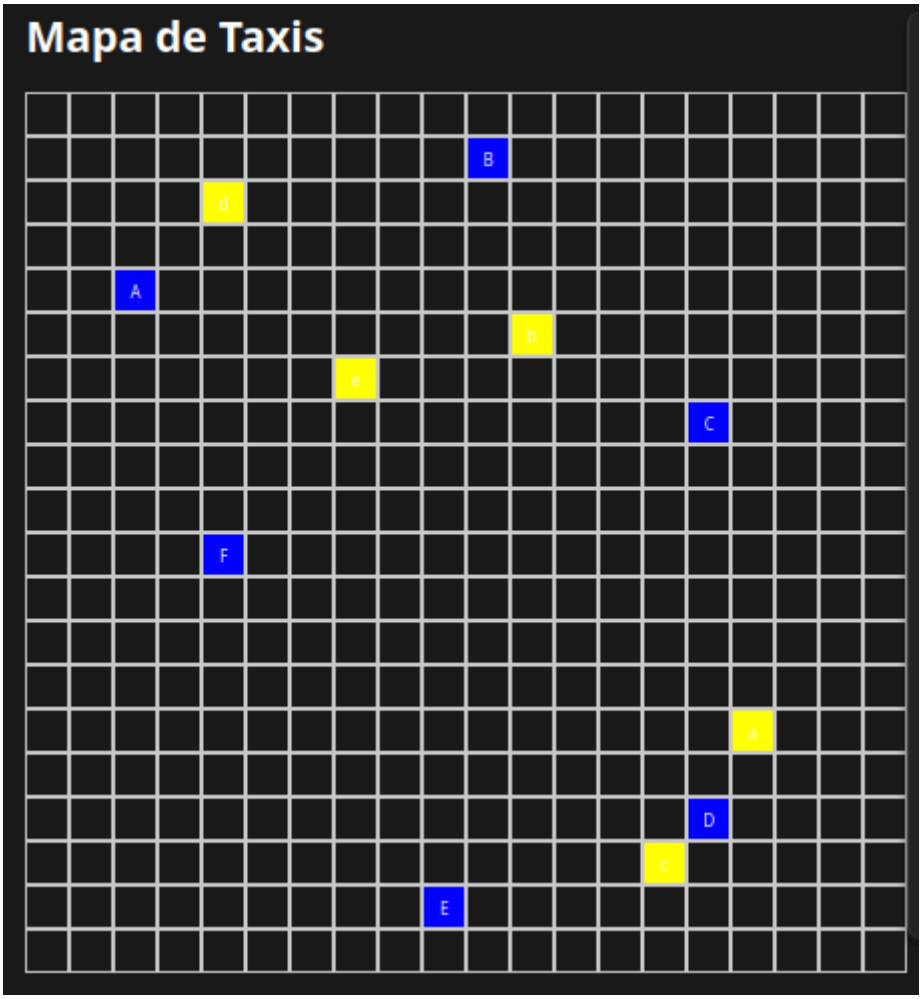
- Auditoría EC_Central && EC_Registry:
    - EC_Central y EC_Registry escriben en el mismo archivo de log, al estar ejecutados en el mismo servidor, y luego ese archivo de log es el que lee el front para mostrarlo.
- ¿Por qué sólo el FRONT lee ese LOG?
    - El front sólo lee ese log pues EC_Central 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.