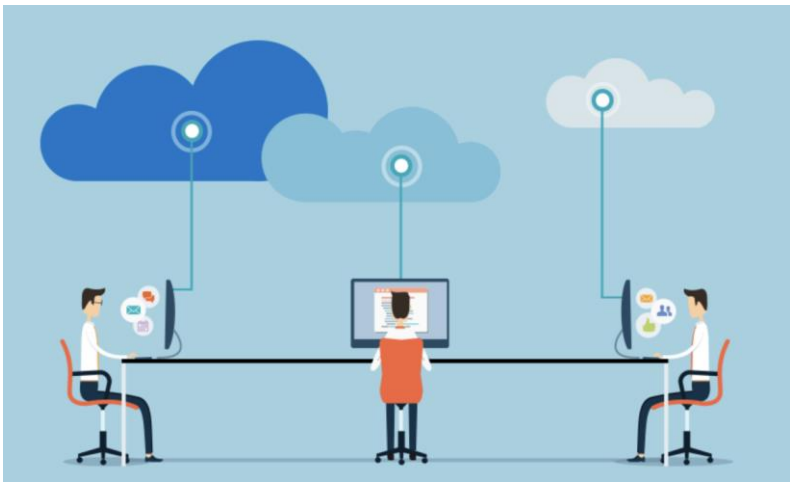


EASYCAB

SISTEMAS DISTRIBUIDOS

DESCRIPCIÓN BREVE

Desarrollo de un sistema distribuido que implemente una simulación de una solución para la gestión de un servicio de taxis de conducción autónoma.

Daniel Jenaro Fernández Juan
53979820Q – 2024/25

Índice

Componentes software

EC_Central

- **Descripción general:**

EC_Central es el núcleo del sistema EasyCab. Se encarga de gestionar el estado y la coordinación de todos los taxis y clientes conectados al sistema. Actúa como intermediario entre los clientes, los taxis, el EC_Registry y el EC_CTC. También interactúa con Apache Kafka como sistema de mensajería.

Su función principal es:

- Coordinar solicitudes de clientes.
- Asignar taxis.
- Enviar instrucciones a taxis.
- Recibir el estado de cada taxi.
- Monitorizar el estado general del tráfico (consultando el CTC).
- Mantener un mapa actualizado del entorno.
- Proporcionar una API HTTP para auditoría y monitorización.

- **Funcionalidades principales:**

1. Gestión de Taxis
   - Mantiene un registro de taxis disponibles y autenticados.
   - Actualiza el estado y posición de cada taxi.
   - Verifica los taxis en el EC_Registry antes de permitir su autenticación.

2. Procesamiento de Solicitudes de Clientes
   - Recibe solicitudes de transporte de los clientes vía Kafka.

- Asigna taxis disponibles.
- Informa a los clientes del resultado (aceptación, espera o rechazo).

3. Comunicación con el Centro de Tráfico (CTC)
   - Consulta periódicamente el estado del tráfico.
   - Si el estado es KO, obliga a todos los taxis a volver a base.

4. Canal de comunicación Kafka
   - Temas Kafka usados:
     - cliente_solicitud: Entrada de solicitudes de clientes.
     - cliente_respuesta: Respuesta a clientes.
     - taxi_instrucciones: Instrucciones enviadas a taxis.
     - taxi_estado: Estado reportado por taxis.
     - mapa_estado: Estado global del sistema.

5. Auditoría
   - Registra en un fichero .log todos los eventos significativos (conexiones, asignaciones, errores…).

6. API HTTP REST
   - Ofrece endpoints como /estado y /auditoria para consultar el estado del sistema y los logs.

7. Interfaz Gráfica (Tkinter)
   - Representación visual en tiempo real de:

- La posición de los taxis.
- La localización de los clientes.
- El estado del mapa (20x20).

- Estructura del código

Clase EC_Central

Contiene toda la lógica y gestión centralizada del sistema. Entre sus métodos más relevantes están:

- __init__()
  Inicializa los componentes del sistema: configuración de Kafka, carga de taxis, carga de mapa, creación de hilos para escuchar clientes, taxis y CTC.
- iniciar_servidor_taxis()
  Lanza un servidor SSL para aceptar conexiones seguras de taxis que quieran autenticarse.
- autenticacion_taxi()
  Gestiona la autenticación de los taxis. Verifica el LRC, consulta al EC_Registry si el taxi está registrado y gestiona el alta en el sistema.
- procesar_solicitudes_cliente()
  Escucha solicitudes de transporte enviadas por los clientes a través de Kafka, decide qué taxi asignar y envía la respuesta.
- unir_taxi_cliente()
  Asigna un taxi libre a un cliente que solicita un servicio. Marca el taxi como ocupado.
- llevar_taxi_a_cliente()
  Gestiona el proceso completo de un taxi yendo a recoger a un cliente y luego llevarlo a su destino. Se encarga también de enviar instrucciones de movimiento al taxi.

- enviar_instrucciones_taxi()
  Publica en Kafka el destino al que debe desplazarse un taxi. Tiene en cuenta el estado del tráfico antes de enviar.
- escuchar_estado_taxis()
  Escucha y procesa los mensajes que cada taxi envía con su estado y posición. Estos mensajes están cifrados con AES y deben ser descifrados.
- enviar_estado_global()
  Cada segundo, publica en Kafka el estado general del sistema (estado de taxis, clientes y mapa) para que otros componentes (por ejemplo, el Frontend) puedan visualizarlo.
- revisar_estado_ctc()
  Consulta cada cierto tiempo el estado del tráfico desde el CTC. Si el tráfico está en KO, da la orden a todos los taxis de volver a base.
- iniciar_http_server()
  Lanza un servidor HTTP (mediante Flask) que expone dos endpoints: /estado y /auditoria, para monitorización externa y consulta de logs.
- cargar_taxis()
  Carga el listado de taxis desde un fichero JSON de base de datos (por ejemplo, taxis.json).
- cargar_localizaciones()
  Carga las localizaciones predefinidas del mapa desde un fichero JSON (por ejemplo, EC_locations.json).
- calcular_lrc() y verificar_lrc()
  Funciones para calcular y validar el código LRC usado en la autenticación de taxis vía socket.
- iniciar_interfaz_grafica() y actualizar_grafico()
  Lanza una ventana Tkinter para visualizar en tiempo real el estado del mapa, taxis y clientes.
- procesar_comandos_arbitrarios()
  Permite al operador introducir comandos manuales (como parar un taxi, reanudarlo o enviarlo a destino) desde la consola.

EC_DE

- **Descripción general**

EC_DE representa el software que se ejecuta en cada taxi del sistema EasyCab. Actúa como un emulador del dispositivo físico del taxi.

Su misión principal es:

- Autenticarse frente a la central (EC_Central) mediante una conexión segura.
- Publicar periódicamente su estado (posición y estado del sensor) hacia la central.
- Escuchar instrucciones enviadas por la central (como moverse a un destino, detenerse o volver a base).
- Simular el movimiento del taxi sobre el mapa.
- Gestionar eventos como fallos del sensor o paradas del servicio.

- **Funcionalidades principales:**

1. Autenticación

- Al arrancar, el taxi se conecta al servidor SSL de la central (EC_Central).
- Envía un mensaje de autenticación usando el formato LRC con <STX>, <ETX> y <LRC>.
- Solo taxis autorizados por el EC_Registry son aceptados.

2. Envío periódico de estado

- Cada segundo, el taxi publica su estado al tópico Kafka taxi_estado.
- El estado incluye:
    - Posición (coordenadas X, Y en el mapa).

    - Estado del sensor (por defecto "OK").
- El mensaje se cifra usando AES en modo CBC, con una clave compartida por taxi.

3. Recepción de instrucciones

- Escucha el tópico taxi_instrucciones de Kafka.
- Responde a órdenes como:
    - IR_A_DESTINO: Se mueve paso a paso hasta la posición indicada.
    - VOLVER_BASE: Vuelve a la posición base (0,0).
    - PARAR: Detiene el envío de estado (simulando un fallo o parada).
    - REANUDAR: Reanuda el envío de estado.

4. Simulación de Movimiento

- Simula el desplazamiento del taxi sobre el mapa.
- Actualiza su posición en cada paso y envía el nuevo estado.

5. Cambio del estado del sensor

- Permite simular manualmente desde consola que el sensor entre en estado "ERROR" o vuelva a "OK".

- **Estructura del código:**

Clase EC_DE

Esta clase gestiona todo el comportamiento del taxi.

Principales métodos:

- __init__()
  Inicializa parámetros como el ID del taxi, IP del broker, IP de la central, y carga la clave compartida para cifrado.

- autenticarse_en_central()
  Realiza la conexión SSL con la central y ejecuta el proceso de autenticación LRC.
- generar_lrc()
  Calcula el código de redundancia LRC que valida la integridad del mensaje de autenticación.
- conectar_kafka()
  Establece la conexión con Apache Kafka. Inicializa los productores y consumidores necesarios para comunicación.
- enviar_estado_periodico()
  Cada segundo cifra y publica el estado del taxi (posición + estado del sensor) en el topic taxi_estado.
- cifrar_mensaje_estado()
  Cifra el JSON de estado del taxi usando AES-CBC y devuelve el IV y el ciphertext en base64.
- recibir_instrucciones()
  Escucha instrucciones para el taxi desde el topic Kafka taxi_instrucciones.
- procesar_instruccion()
  Dependiendo del comando recibido:
  - Mueve el taxi hacia un destino.
  - Lo para o lo reanuda.
  - Lo devuelve a la base.
- simular_movimiento()
  Calcula un movimiento gradual del taxi hacia el destino paso a paso (simula el trayecto físico).
- enviar_estado()
  Envia de forma inmediata el estado actual del taxi (por ejemplo, después de cada paso de movimiento).
- cambiar_estado_sensor()
  Permite manualmente cambiar el estado del sensor del taxi desde la consola.

Flujo general de ejecución de EC_DE:

1. Autenticación inicial contra la central.
2. Conexión a Kafka.
3. Arranca hilos paralelos para:
    - Enviar estado cada segundo.
    - Escuchar instrucciones desde la central.
4. Ejecuta movimientos e instrucciones según órdenes de la central.

EC_Customer

EC_Customer es el componente software que representa a un cliente del sistema EasyCab. Su propósito es simular el comportamiento de un pasajero que desea solicitar un taxi para desplazarse de un origen a un destino dentro del mapa del sistema.

- **Su función principal es:**
  - Generar y enviar solicitudes de transporte a la central (EC_Central) a través de Kafka.
  - Esperar y procesar la respuesta de la central (aceptación, espera o rechazo).
  - Simular el trayecto junto con el taxi (recibir actualizaciones de estado).
  - Mostrar información al usuario (en consola) sobre el estado de su viaje.

- **Funcionalidades principales:**
  1. Generación de solicitudes de transporte
  - El cliente selecciona (o se le asigna aleatoriamente) un origen y un destino válidos dentro del mapa.
  - La solicitud incluye:
    - Su ID de cliente.
    - Coordenadas de origen y destino.
    - Estado inicial ("OK").
  2. Comunicación vía Kafka
  - Publica su solicitud en el tópico Kafka cliente_solicitud.
  - Escucha el tópico cliente_respuesta para recibir el resultado.
  3. Recepción de respuestas del sistema
  - Posibles estados recibidos en Kafka:

- OK: La central ha asignado un taxi.
- ESPERA: No hay taxis disponibles. El cliente deberá esperar o volver a intentarlo.
- KO: Solicitud rechazada.
- RECOGIDO: El taxi ha llegado al origen del cliente.
- DESTINO: El cliente ha llegado a su destino final.
- ABANDONADO: La central ha notificado que el taxi ha abandonado al cliente por algún motivo (por ejemplo, fallo o baja de taxi).

4. Mostrar estado en consola

- El cliente recibe mensajes en tiempo real sobre el estado de su viaje y los imprime por consola.
- Ejemplos:
  - "Taxi asignado, en camino."
  - "Taxi ha llegado a recogerte."
  - "Has llegado a tu destino."
  - "El servicio fue cancelado por el sistema."

5. Configuración inicial

- Al arrancar, el cliente recibe parámetros como:
  - Su ID.
  - IP del broker Kafka.
  - Origen y destino (si no se especifican, pueden ser aleatorios).

- **Estructura del código:**

Clase EC_Customer

Contiene toda la lógica del comportamiento de un cliente.

Métodos principales:

- __init__()
  Inicializa el cliente, configurando ID, IP de broker, origen, destino y estado inicial. También arranca la conexión a Kafka.
- connect_to_kafka()
  Establece la conexión con Apache Kafka tanto para el producer como el consumer.
- send_request()
  Publica en el topic Kafka cliente_solicitud un JSON con los campos:
  - cliente_id
  - origen
  - destino
  - estado
- listen_response()
  Se suscribe al topic Kafka cliente_respuesta.
  Filtra y procesa solo los mensajes cuyo cliente_id coincida con el suyo.
  Va actualizando el estado del cliente según las respuestas de la central.
- process_response()
  Analiza el campo "estado" de las respuestas recibidas:
  - Si es "OK", espera ser recogido.
  - Si es "RECOGIDO", marca que está viajando.
  - Si es "DESTINO" o "ABANDONADO", termina el proceso.
- run()
  Método que coordina todo el ciclo de vida del cliente:
  Enviar la solicitud → Escuchar → Procesar → Finalizar.

Flujo general de ejecución de EC_Customer:

1. El cliente arranca y se conecta a Kafka.
2. Envía su solicitud de taxi.

3. Escucha respuestas en tiempo real.
4. Actualiza y muestra su estado según la central.
5. Finaliza una vez llega a destino o si es abandonado.

EC_CTC

EC_CTC es el componente encargado de simular el estado general del tráfico en el entorno EasyCab. Representa un sistema externo que indica a la central (EC_Central) si las condiciones del tráfico permiten o no el desplazamiento de los taxis.

- **Su función principal es:**

  - Mantener y exponer el estado actual del tráfico (OK o KO).
  - Permitir consultas HTTP por parte de la central.
  - Permitir al operador humano cambiar manualmente el estado del tráfico mediante una interfaz web sencilla.

Funcionalidades principales:

1. Gestión del estado del tráfico

- Tiene un único estado global:
  OK (tráfico funcionando) o KO (estado crítico que impide la circulación de taxis).

2. API HTTP REST

- Levanta un servidor Flask.
- Expone los siguientes endpoints:
  - /consulta (GET):
    La Central (EC_Central) consulta periódicamente este endpoint para saber el estado actual del tráfico.
  - /cambiar_estado (POST):
    Permite modificar el estado del tráfico enviando un JSON.

Log de eventos

- Cada cambio de estado queda registrado en la consola del servidor.
- Opcionalmente, podría añadirse logging a fichero si se desea auditar los cambios.

- Estructura del código:

Aplicación Flask - EC_CTC

El código de EC_CTC es sencillo y se basa principalmente en Flask.

Estructura y funciones principales:

- Variable Global: estado_actual = "OK"

Almacena el estado del tráfico.

- Endpoint /consulta (GET)
  - Permite a la central consultar el estado actual.
  - Devuelve un JSON con el estado.
- Endpoint /cambiar_estado (POST)
  - Permite cambiar el estado.
  - Recibe un JSON con el nuevo estado ("OK" o "KO").
  - Actualiza la variable global.
  - Devuelve confirmación en JSON.
- Arranque del servidor:

  app.run(host='0.0.0.0', port=5000, debug=False)

Escucha en el puerto 5000 para que la central pueda acceder.

Flujo de uso típico:

1. La EC_Central llama cada 10 segundos al endpoint /consulta.
2. Si el EC_CTC devuelve "KO", la central reacciona (por ejemplo, ordena a todos los taxis volver a base).
3. Un operador puede abrir Postman o un navegador, o usar curl, y hacer un POST a /cambiar_estado para forzar un cambio a "KO" o volver a "OK".

EC_Registry

EC_Registry actúa como el repositorio centralizado de confianza (registry) para el sistema EasyCab. Es responsable de almacenar y gestionar la lista de taxis válidos en el sistema junto con sus claves compartidas para cifrado.

- **Su función principal es:**

  - Mantener una base de datos de taxis registrados (ID de taxi y su clave).
  - Proporcionar a otros componentes (principalmente EC_Central) una forma de consultar o recuperar claves asociadas a cada taxi.
  - Evitar que taxis no registrados se autentiquen en el sistema.

- **Funcionalidades principales:**

  1. Gestión de Taxis Registrados

  - Almacena un listado interno de taxis válidos, junto con su clave secreta.
  - Esta lista puede estar almacenada en memoria o leída de un archivo JSON (ejemplo: taxis_registry.json).

  2. API HTTP REST
     Levanta un servidor web (Flask) con los siguientes endpoints:

- GET /list_taxis
  Devuelve la lista de IDs de taxis registrados.
  La EC_Central consulta este endpoint antes de permitir que un taxi se autentique.
- POST /get_taxi_key
  Permite a la EC_Central obtener la clave compartida de un taxi específico, necesaria para descifrar mensajes cifrados con AES.

3. Seguridad (SSL)

- EC_Registry funciona bajo HTTPS (con certificado TLS).
- Esto garantiza que el intercambio de claves entre EC_Central y EC_Registry sea seguro.
- Para ello, al iniciar el servidor Flask, se proporciona un certificado (por ejemplo registry.crt y registry.key).

4. Auditoría Básica

- Cada petición recibida (listado o consulta de clave) puede registrarse en consola.
- Opcionalmente, puede implementarse un log de auditoría en fichero.

- **Estructura del código:**

Aplicación Flask - EC_Registry

Principales partes del código:

- Base de datos de taxis
  Cargada desde un archivo JSON o definida como un diccionario en memoria.
- Endpoint /list_taxis (GET):
  - Devuelve un JSON con los IDs de todos los taxis registrados.
- Endpoint /get_taxi_key (POST):
  - Recibe un ID de taxi.
  - Si el taxi está registrado, devuelve la clave correspondiente.

    - Si el ID no existe, devuelve un error.
- Servidor HTTPS:

  app.run(host='0.0.0.0', port=5001, ssl_context=('registry.crt', 'registry.key'))

Flujo de uso típico:

1. EC_Central llama a /list_taxis para validar que un taxi existe antes de aceptar su autenticación vía socket.
2. EC_Central también llama a /get_taxi_key para recuperar la clave de cifrado de cada taxi antes de descifrar mensajes de estado (Kafka taxi_estado).

EC_Sensores

EC_Sensores es el módulo que simula y gestiona el estado del sensor físico de un taxi dentro del sistema EasyCab. Cada instancia de EC_Sensores representa el "sensor de hardware" embarcado en un taxi.

- **Su función principal es:**
  - Simular el estado operativo del taxi (por ejemplo: OK, KO, FALLA).
  - Enviar periódicamente el estado cifrado a la EC_Central a través de Kafka (tema taxi_estado).
  - Aplicar cifrado AES-CBC sobre el estado y la posición antes de enviarlo.

Funcionalidades principales:

1. Simulación de Estado del Taxi

- El sensor genera estados como:
  OK
  KO
  FALLA

- El estado puede ser aleatorio o controlado por el operador.

2. Envío de Estado a EC_Central

- Publica en Kafka el estado cifrado del taxi en el tema taxi_estado.
- El mensaje incluye:
    - El ID del taxi.
    - Un IV (vector de inicialización).
    - Los datos cifrados en base64.

3. Cifrado AES-CBC

- Cada taxi tiene una clave única que se obtiene del EC_Registry al iniciar.
- El estado y la posición se empaquetan como JSON y se cifran antes de enviarlos.

4. Conexión a Apache Kafka

- El sensor utiliza un KafkaProducer para publicar mensajes en el topic taxi_estado.
- La IP del broker Kafka se pasa como parámetro al iniciar.

5. Obtención de clave del Registry

- Antes de empezar a publicar, el sensor realiza una llamada HTTPS al EC_Registry para obtener su clave compartida.
- Endpoint usado:
  POST /get_taxi_key

6. Control por consola (opcional)

- Permite cambiar manualmente el estado del sensor a través de la entrada por consola mientras está en ejecución.

- **Estructura del código:**

Script EC_Sensores.py

Las partes más relevantes del código:

- Carga de configuración:
    - Se recibe como parámetros:
        - ID del taxi.
        - IP del broker Kafka.
        - IP del Registry.
- Obtener clave del taxi:
    - Solicitud POST a /get_taxi_key del Registry.
- Ciclo de envío de estado:
    - Genera o recibe el estado actual del sensor.
    - Obtiene la posición actual del taxi (generalmente recibida de EC_Central vía otro canal o simulada internamente).
    - Cifra con AES-CBC usando la clave y un IV aleatorio.
    - Codifica a base64 y publica en Kafka.
- Control de cambios de estado por consola:
    - Permite al operador forzar estados (ej: pasar a KO manualmente).
- Manejo de errores:
    - Controla fallos al obtener la clave, o al enviar mensajes.

Flujo de trabajo típico:

1. El taxi inicia el EC_Sensores.
2. Se conecta al Registry y obtiene su clave.
3. Cada segundo (o periodo definido):
    - Cifra su estado actual y posición.

    - Envía el mensaje cifrado por Kafka.
4. La EC_Central, al escuchar el topic taxi_estado, descifra el mensaje y actualiza el estado del taxi.

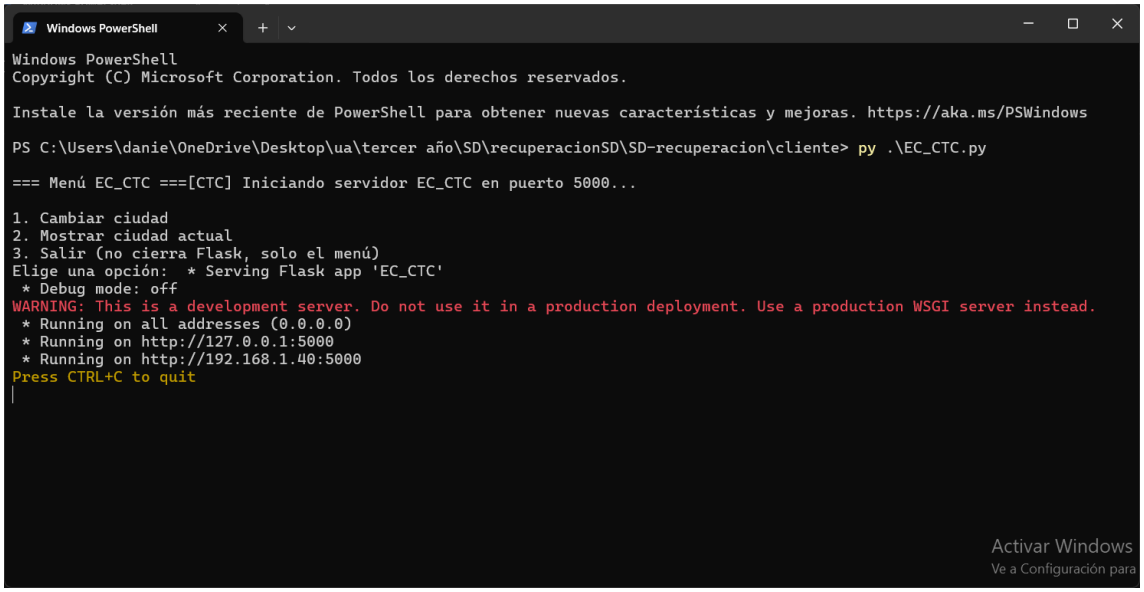
Guía de despliegue

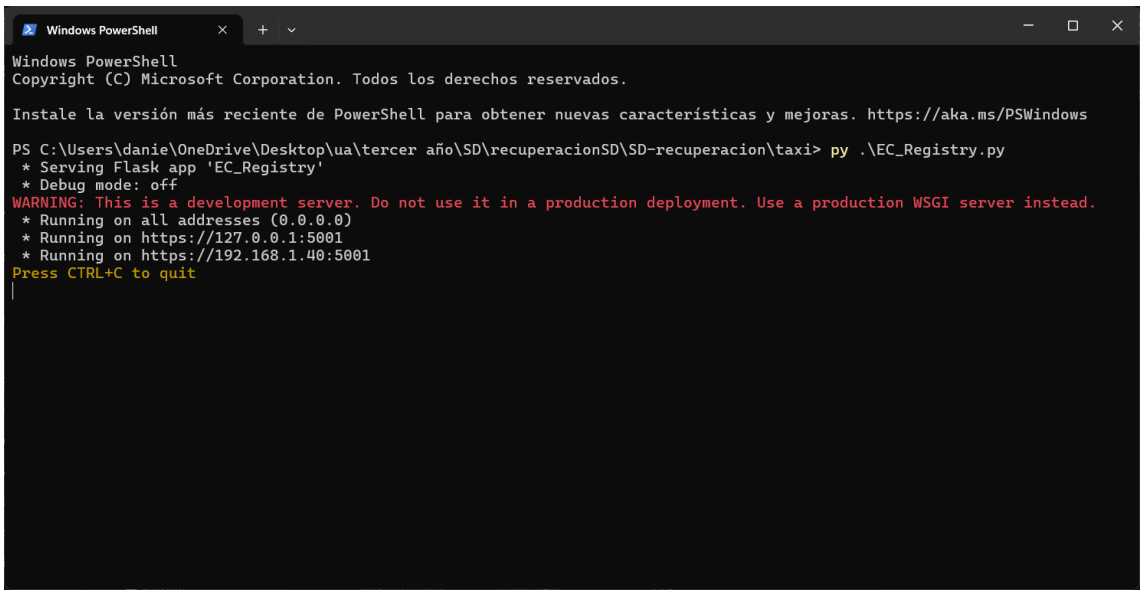
- **<u>Pasos para desplegar la aplicación de forma distribuida.</u>**

1. Instalar todos los archivos y ficheros necesarios en los 3 ordenadores. En el primero irán el gestor de colas, la central y el front, en el segundo, el CTC y los clientes, y en el tercero, los taxis y el Registry.
2. Modificaremos el Kafka, concretamente el fichero server.properties, especificando la ip de la máquina.
3. Modificaremos los scripts de despliegue, introduciendo las ip's de las máquinas en cuestión. Modificar los scripts y añadir la ip de la máquina que contiene Kafka.
4. Después en el código, cambiaremos los endpoints poniendo las ip's a las que se necesite acceder. Como puede ser para el front o registry.
5. Se generará los certificados correspondientes para permitir el cifrado entre máquinas.
6. Se encenderán primero los componentes CTC, Registry, luego la central, después el servidor web del front y por último los taxis y los clientes.
7. Para desplegar los taxis, se realizará con un script para ejecutar tantos taxis como se quiera y tantos sensores como se quiera. También hay otro script para desplegar tantos clientes como especifiquemos.

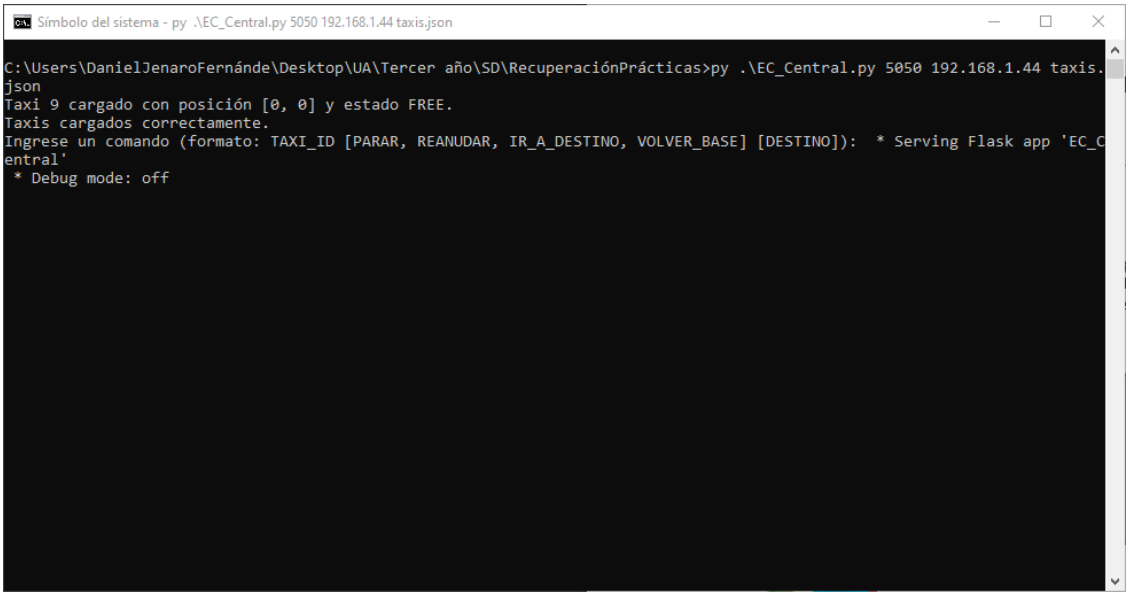
Funcionamiento de la aplicación

Primero tendremos que ejecutar el CTC y el Registry

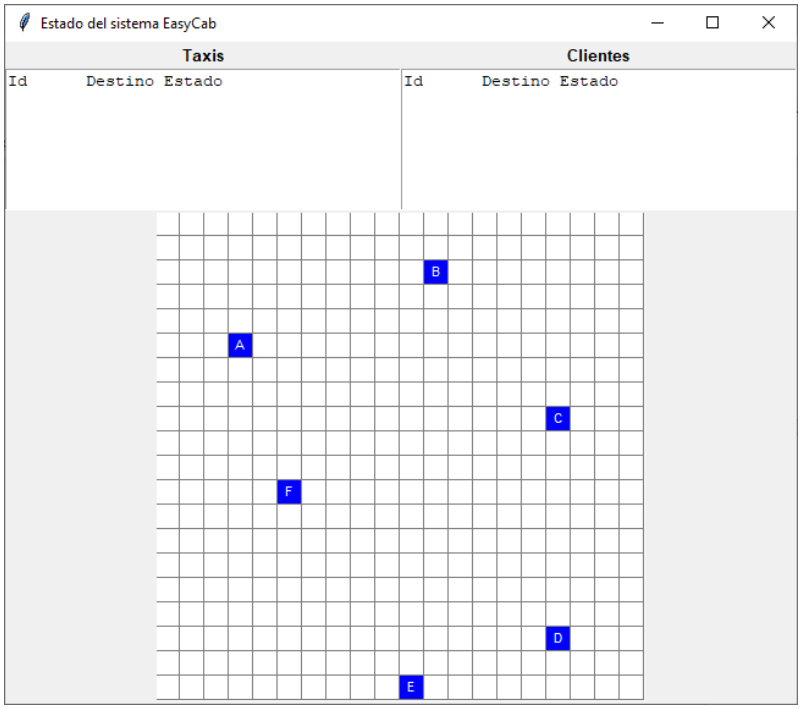




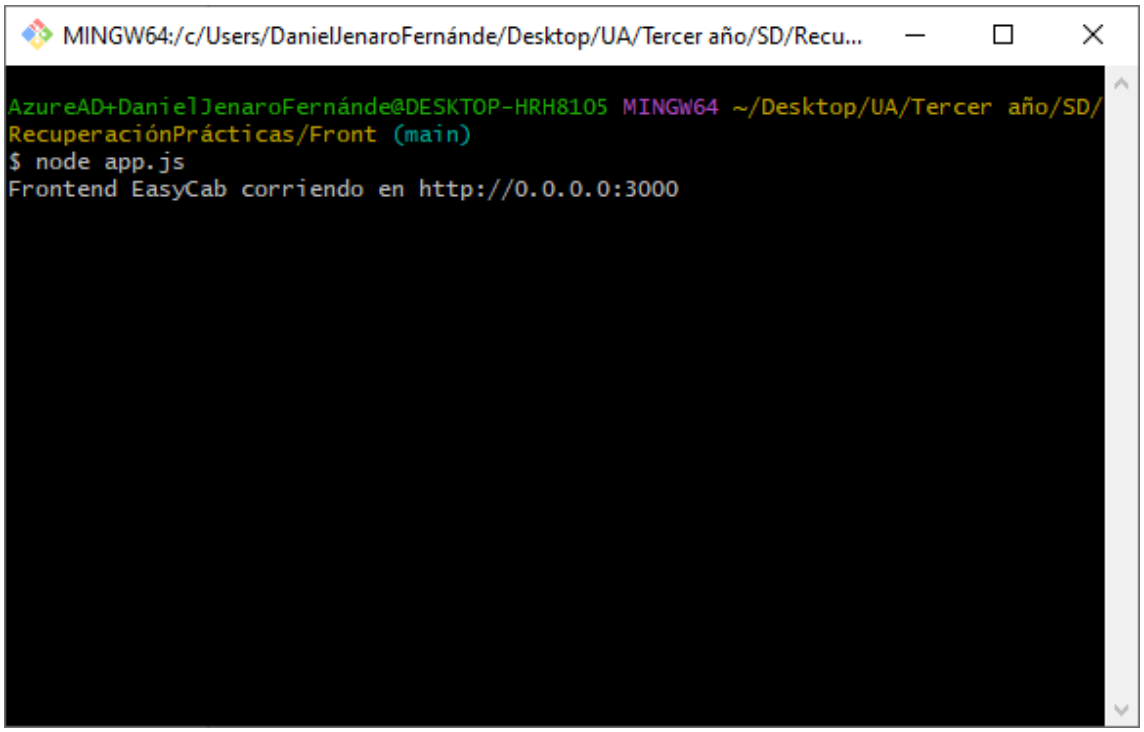
Una vez hecho esto, ya podremos iniciar la central



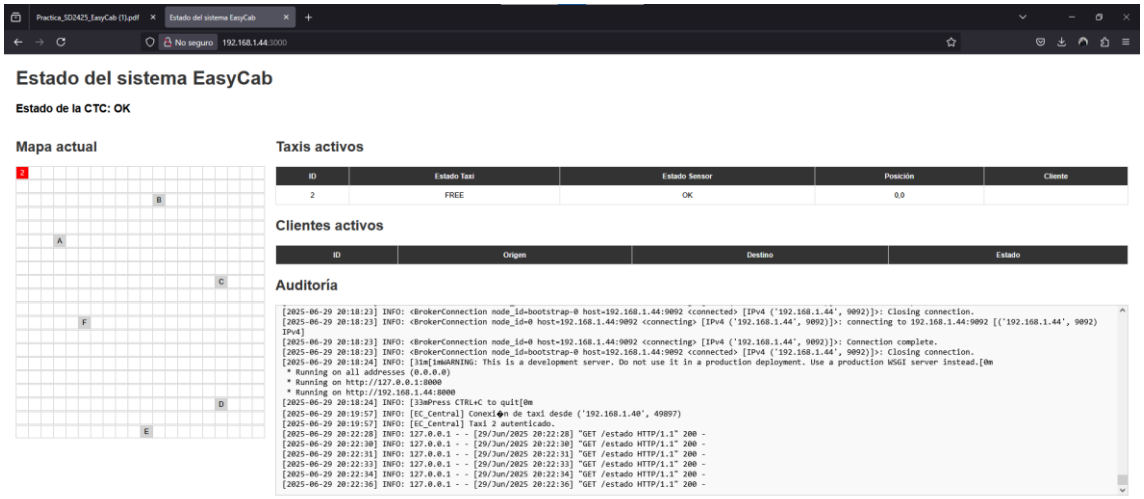
Que queda en espera hasta que se conecten los taxis y los clientes. Al iniciar la central se abrirá el mapa con las posiciones del tablero.



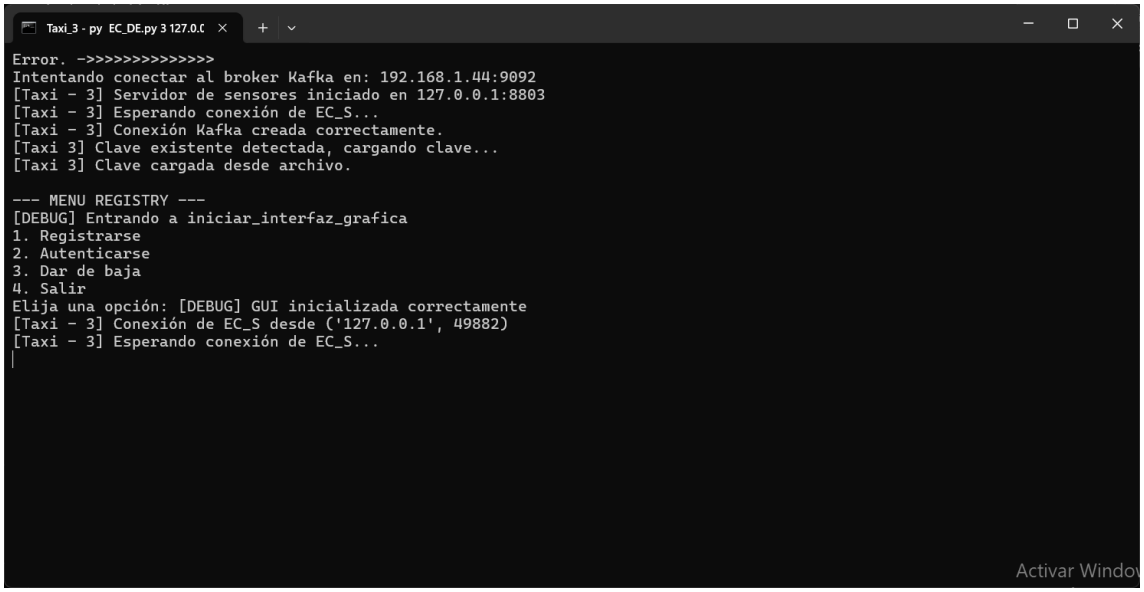
Ahora iniciamos el servidor web:



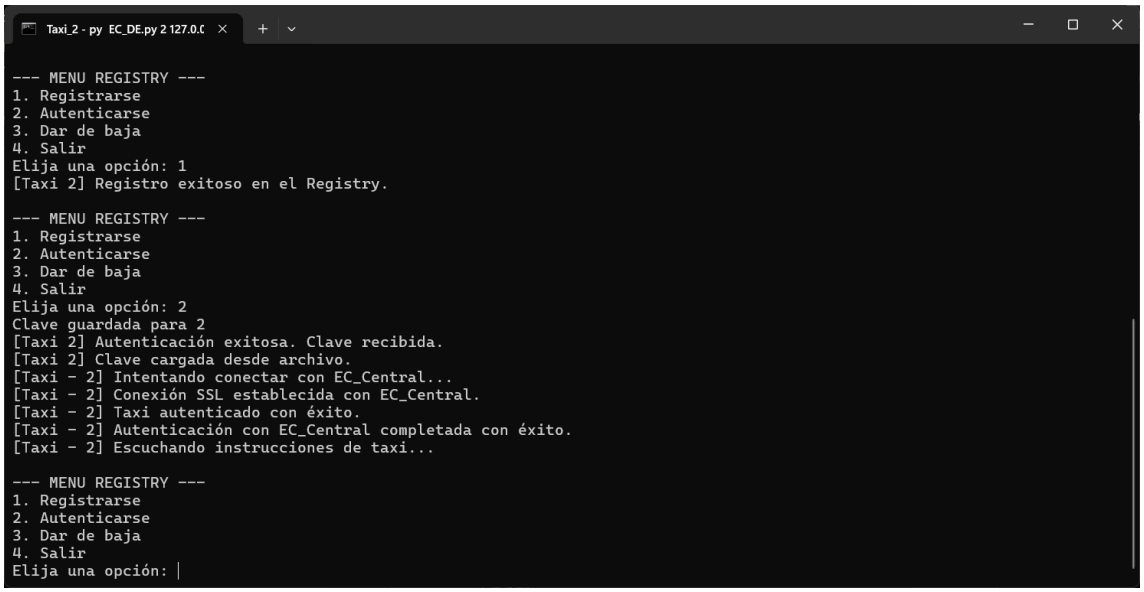
Que se ve de esta manera:



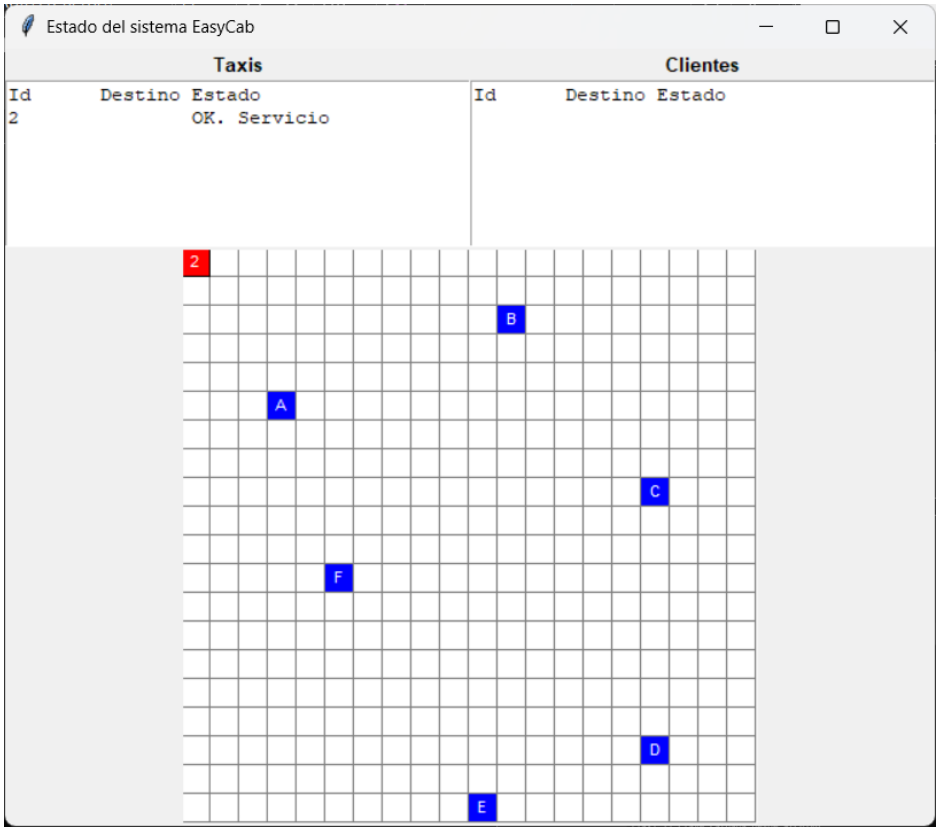
Ahora podemos iniciar los taxis, una vez iniciados, deberemos registrarnos en la base de datos y autenticarnos contra la central



Esto se hace con el menú de opciones.



Cuando nos autenticamos aparecerá el taxi en el tablero de todos los componentes y en la web. La web es accesible desde cualquier máquina.



Y se pondrá en ejecución el programa.