SISTEMAS DISTRIBUIDOS
PRÁCTICA 1

Alejandro García Gil 77042008N & Ginés Caballero Guijarro 54639204J

ÍNDICE

EC_CENTRAL

DESCRIPCIÓN DEL ARCHIVO:

La clase ECCentral gestiona la central del sistema de taxis. Esta central es el componente principal que gestiona las solicitudes de los clientes, asigna taxis disponibles y realiza el seguimiento de estos durante el servicio. La central se comunica tanto con los clientes como con los taxis a través de Kafka, y utiliza varios tópicos para coordinar estas interacciones.

DETALLES IMPORTANTES:

Constantes de Kafka:

TOPIC_SOLICITUDES_TAXIS: Recibe las solicitudes de los clientes. Cuando un cliente solicita un taxi, la central escucha este tópico para procesar la solicitud.

TOPIC_RESPUESTAS_TAXIS: Envía respuestas a los clientes indicando si hay taxis disponibles. La central responde con un mensaje que confirma si se ha encontrado un taxi o no.

TOPIC_ASIGNACION_TAXIS: Envía asignaciones a los taxis. Este tópico se utiliza para enviar la información de los clientes a los taxis que se encuentran disponibles para recogerlos.

TOPIC_TAXI_UPDATES: Recibe las actualizaciones de estado de los taxis. Estas actualizaciones incluyen la posición del taxi y su disponibilidad, lo cual permite que la central tenga una visión precisa del estado actual de la flota.

TOPIC_TAXI_END_CENTRAL: Recibe notificaciones de finalización de servicio. Cuando un taxi llega a su destino, envía una notificación a la central indicando que está nuevamente disponible.

Métodos principales:

listen_taxi_updates(): Escucha las actualizaciones de los taxis y actualiza su estado en el sistema. Este método es fundamental para mantener un registro de la ubicación y disponibilidad de cada taxi, lo cual facilita la asignación eficiente de recursos.

listen_customer_services(): Maneja las solicitudes de los clientes y asigna taxis según la disponibilidad. Al recibir una solicitud, este método busca un taxi disponible y le asigna la tarea, enviando una respuesta al cliente para confirmarle que su solicitud ha sido atendida.

listen_taxi_end(): Recibe notificaciones cuando un taxi completa un servicio y actualiza su disponibilidad. Este método permite actualizar el estado de un taxi a disponible después de completar un viaje, lo cual es esencial para garantizar que los taxis puedan ser asignados a nuevos clientes.

Gestión de Recursos y Optimización:

La central es responsable de optimizar la asignación de taxis, asegurando que los taxis más cercanos sean asignados a las solicitudes de los clientes para minimizar el tiempo de espera.

La clase también maneja errores y excepciones para evitar problemas en la asignación, como por ejemplo cuando un taxi no se encuentra en la base de datos o tiene un estado no válido.

Flujo de Datos:

La central (ECCentral) escucha las solicitudes de los clientes a través del tópico TOPIC_SOLICITUDES_TAXIS. Una vez que se recibe una solicitud, se busca un taxi disponible y se envía una asignación a través del tópico TOPIC_ASIGNACION_TAXIS. Los taxis envían actualizaciones de ubicación y estado a TOPIC_TAXI_UPDATES, que la central utiliza para gestionar la flota. Al finalizar un servicio, la central recibe notificaciones de los taxis mediante TOPIC_TAXI_END_CENTRAL y actualiza la disponibilidad del taxi, permitiendo que sea asignado a nuevos cliente.

EC_CUSTOMER

DESCRIPCIÓN DEL ARCHIVO:

Este archivo contiene la clase ECCustomer, que se encarga de simular el comportamiento de un cliente que solicita un taxi. Este cliente envía su solicitud a la central, recibe una confirmación y espera la llegada del taxi. La clase ECCustomer utiliza Kafka para enviar y recibir mensajes de forma asíncrona, lo que permite simular un entorno realista de solicitud y recepción de servicios de taxis.

FUNCIONALIDADES CLAVE:

Constantes de Kafka:

TOPIC_SOLICITUDES_TAXIS: Canal utilizado para enviar solicitudes de taxis. Cuando un cliente desea ir a un destino específico, esta solicitud se publica en este tópico.

TOPIC_RESPUESTAS_TAXIS: Canal utilizado para recibir las respuestas de la central sobre el estado de la solicitud. Las respuestas indican si un taxi ha sido asignado o si la solicitud no puede ser atendida.

TOPIC_TAXI_END_CLIENT: Canal utilizado para recibir el aviso de que el servicio ha finalizado. Esto permite al cliente saber cuándo ha llegado al destino.

Métodos principales:

handle_response(): Maneja la respuesta de la central, verificando que el mensaje recibido sea para el cliente actual. Si el cliente ha sido identificado correctamente, se procesa la respuesta, lo cual incluye comprometer el offset del consumidor de Kafka.

request_taxi(destination): Envía una solicitud de taxi para un destino específico. La solicitud incluye el ID del cliente y el destino deseado. La clase KafkaProducer se utiliza para enviar el mensaje al tópico TOPIC_SOLICITUDES_TAXIS.

wait_arrival(): Espera la confirmación de llegada del taxi. El cliente se mantiene a la escucha de mensajes en el tópico TOPIC_TAXI_END_CLIENT hasta que recibe un mensaje que indica que el taxi ha llegado al destino.

Uso del archivo:

Se ejecuta con los parámetros <KAFKA_IP_PORT> <CUSTOMER_ID> <REQUESTS FILE>. El cliente carga sus destinos desde el archivo JSON proporcionado (Customer_services.json) y procede a solicitar taxis para cada destino en la lista. Cada destino representa una ubicación específica a la que el cliente desea ir, simulando una interacción real con un servicio de transporte.

FLUJO DE DATOS:

El cliente (ECCustomer) comienza cargando la lista de destinos desde un archivo JSON. Luego, se envía una solicitud de taxi a través de Kafka utilizando el KafkaProducer, la cual es recibida por la central. La central procesa esta solicitud y envía una respuesta al cliente utilizando TOPIC_RESPUESTAS_TAXIS. Una vez que el cliente recibe la confirmación, se mantiene a la escucha de mensajes en TOPIC_TAXI_END_CLIENT hasta que el servicio finaliza y el taxi llega al destino.

EC_DE

DESCRIPCIÓN DEL ARCHIVO:

La clase DigitalEngine gestiona el comportamiento de un taxi. Este archivo representa la lógica de cada taxi, simulando su comportamiento al recibir asignaciones de clientes, desplazarse hacia el cliente, y luego llevar al cliente al destino deseado. DigitalEngine es responsable de gestionar la posición del taxi, así como su estado operativo.

FUNCIONALIDADES CLAVE:

Constantes de Kafka:

TOPIC_TAXI_UPDATES: Envía actualizaciones de ubicación y estado del taxi a la central.
Esto permite a la central conocer la posición y disponibilidad de los taxis en todo momento.

TOPIC_ASIGNACION_TAXIS: Recibe asignaciones de viajes desde la central. Cuando un cliente solicita un taxi y la central le asigna un vehículo, el mensaje se recibe en este tópico.

TOPIC_TAXI_END_CLIENT: Envía notificaciones al cliente al terminar el servicio, indicando que el taxi ha llegado al destino.

TOPIC_TAXI_END_CENTRAL: Envía notificaciones a la central al terminar el servicio, actualizando el estado del taxi a disponible.

Métodos principales:

connect_to_central(): Establece la conexión con la central y verifica el taxi. Envía el ID del taxi a la central para comprobar si el taxi está registrado en la base de datos de la central.

listen_asignacion_kafka(): Escucha asignaciones de clientes y actualiza la posición del cliente y destino. Este método se mantiene a la escucha en el tópico TOPIC_ASIGNACION_TAXIS y actualiza las coordenadas de destino y cliente cuando se recibe una nueva asignación.

handle_sensors(): Maneja los mensajes provenientes de los sensores para actualizar el estado y la ubicación del taxi. Los sensores envían continuamente información sobre la posición del taxi, y este método asegura que los datos se transmitan a la central para mantener el estado actualizado.

updateCoordinates(): Actualiza las coordenadas del taxi mientras se mueve hacia el destino del cliente. Este método ajusta las coordenadas paso a paso para simular el movimiento del taxi por el mapa.

Interacción con Sensores y Kafka:

DigitalEngine establece una conexión de socket con los sensores para recibir información del estado del vehículo (por ejemplo, si está operativo o tiene problemas).

También mantiene una comunicación constante con la central a través de Kafka, lo cual permite que el sistema tenga un flujo continuo de información sobre la ubicación y el estado de cada taxi.

FLUJO DE DATOS:

El DigitalEngine recibe asignaciones de la central a través del tópico TOPIC_ASIGNACION_TAXIS. Luego, comienza a moverse hacia la posición del cliente y actualiza constantemente su ubicación en el tópico TOPIC_TAXI_UPDATES. Los sensores informan sobre el estado del vehículo (por ejemplo, si está operativo) mediante el método handle_sensors(), y cuando el servicio ha finalizado, se envía una notificación tanto al cliente (TOPIC_TAXI_END_CLIENT) como a la central (TOPIC_TAXI_END_CENTRAL) para informar que el taxi está disponible nuevamente.

EC_S

DESCRIPCIÓN DEL ARCHIVO:

Este archivo simula un sensor asociado a un taxi. Los sensores se encargan de enviar información del estado del taxi, indicando si está operativo o tiene algún problema (estado "OK" o "KO"). Esta información es fundamental para la central, ya que permite decidir si un taxi está disponible para atender una nueva solicitud de cliente.

FUNCIONALIDADES CLAVE:

Métodos principales:

wait_input(status): Permite al usuario cambiar manualmente el estado del taxi a "OK" o "KO". Este método simula un sensor que permite a un operador indicar si el taxi tiene algún problema.

send_message(client_socket, message): Envía mensajes al taxi sobre su estado actual. Utiliza sockets para enviar estos mensajes, lo cual permite comunicar de manera directa el estado del vehículo a DigitalEngine.

run(): Establece la conexión con el motor digital (Digital Engine) y gestiona el flujo de mensajes del sensor. Este método permite que el sensor envíe actualizaciones periódicas sobre el estado del taxi, lo que mantiene la información del estado actualizada en la central y en DigitalEngine.

Simulación Manual del Estado del Taxi:

En este archivo se simula la posibilidad de que un operador pueda introducir manualmente el estado del taxi. De esta manera, se pueden simular fallas y verificar si la central es capaz de adaptarse a estas situaciones.

El sensor utiliza un hilo separado (threading) para manejar la entrada del usuario mientras sigue enviando mensajes al motor digital sobre el estado del taxi.

FLUJO DE DATOS:

El sensor (EC_S) se conecta al DigitalEngine utilizando sockets y envía mensajes periódicos que indican el estado del taxi. Estos mensajes incluyen el estado actual ("OK" o "KO"), que es utilizado por el DigitalEngine para actualizar la información del taxi en la central. Esto permite que el sistema reaccione en tiempo real a cualquier problema que pueda tener el vehículo.

ARCHIVOS JSON

Flujo de Datos:

Mapa.json: se carga en la central para determinar las posiciones de los clientes y destinos. La información de coordenadas se utiliza para planificar las rutas de los taxis y garantizar que los servicios se gestionen de manera eficiente. Cada vez que se asigna un taxi, la central consulta este archivo para determinar la ubicación exacta del cliente y el destino.

Customer_services.json: Define las solicitudes de los clientes, incluyendo los destinos a los que desean ir. Este archivo es cargado por los clientes (ECCustomer) para enviar solicitudes de taxis. Cada solicitud contiene un identificador de destino que se correlaciona con las ubicaciones definidas en Mapa.json. El archivo sigue un formato sencillo que permite a cada cliente especificar múltiples destinos que desea visitar durante la simulación.

.

taxis.json y taxisDefault.json: Estos archivos contienen la información de los taxis, incluyendo su estado, ubicación actual y si están disponibles para ser asignados a un cliente. El archivo taxis.json se utiliza para gestionar la disponibilidad en tiempo real, mientras que taxisDefault.json sirve como una copia de respaldo para restablecer los datos. Cada taxi tiene atributos como id, estado (verde o rojo), disponible (booleano que indica si está libre), y las coordenadas de su ubicación actual (coordenada_origen) y destino (coordenada_destino). Estos archivos son utilizados por la central para gestionar la disponibilidad de los taxis y su localización.

Estructura de Datos:

Los archivos JSON permiten estructurar la información de una manera accesible y fácil de interpretar por los diferentes componentes del sistema. Por ejemplo, el campo estado de cada taxi es esencial para que la central determine si un taxi puede ser asignado o no.

Además, el campo cliente en taxis.json y taxisDefault.json guarda información sobre el cliente actualmente asignado al taxi, lo cual facilita la gestión del recorrido y el monitoreo de los servicios.

DESPLIEGUE DE LA APLICACIÓN

Para desplegar nuestra práctica en los tres ordenadores de la universidad, seguimos un proceso estructurado que asegura la correcta interacción entre los distintos componentes del sistema. Aquí describimos los pasos en detalle:

Ordenador 1: Configuración de Kafka y Clientes

1. **Instalación de Kafka:**

→ Primero, instalamos Kafka en el ordenador, lo que incluye configurar Zookeeper y el servidor de Kafka. Esto es fundamental ya que Kafka actúa como el mediador de mensajes entre los diferentes componentes de nuestra práctica.

→ Modificamos el archivo `server.properties` de Kafka, descomentando las líneas necesarias para activar configuraciones específicas que mejoren el rendimiento o la seguridad según nuestras necesidades.

2. **Levantar Kafka:**

→ Iniciamos Zookeeper y el servidor de Kafka para asegurarnos de que el sistema de mensajería esté operativo antes de iniciar cualquier componente que dependa deél.

3. **Preparación de los Clientes:**

→ En este mismo ordenador, levantamos los procesos de los clientes (`customers`). Cada cliente se conectará con Kafka para enviar y recibir mensajes que coordinen las acciones dentro del sistema.

Ordenador 2: Configuración de la Central

1. **Levantar la Central:**

→ En el segundo ordenador, solo levantamos la aplicación de la central (`central`). Esta aplicación es crucial ya que procesa todas las solicitudes y comandos enviados por los clientes a través de Kafka y toma decisiones correspondientes sobre cómo manejar los taxis.

Ordenador 3: Configuración de los Taxis

Instalación de Dependencias:

→ Aseguramos que todas las librerías necesarias, como las de Confluent Kafka para la integración con Kafka y Matplotlib para cualquier visualización necesaria, estén instaladas y configuradas correctamente.

Levantar los Taxis:

→ Levantamos dos instancias de `Digital Engine`, cada uno representando un taxi. Cada `Digital Engine` incluye un proceso que simula los sensores del taxi, enviando y recibiendo mensajes a través de Kafka para actualizar su estado y posición según las instrucciones de la central.

Regreso al Ordenador 1

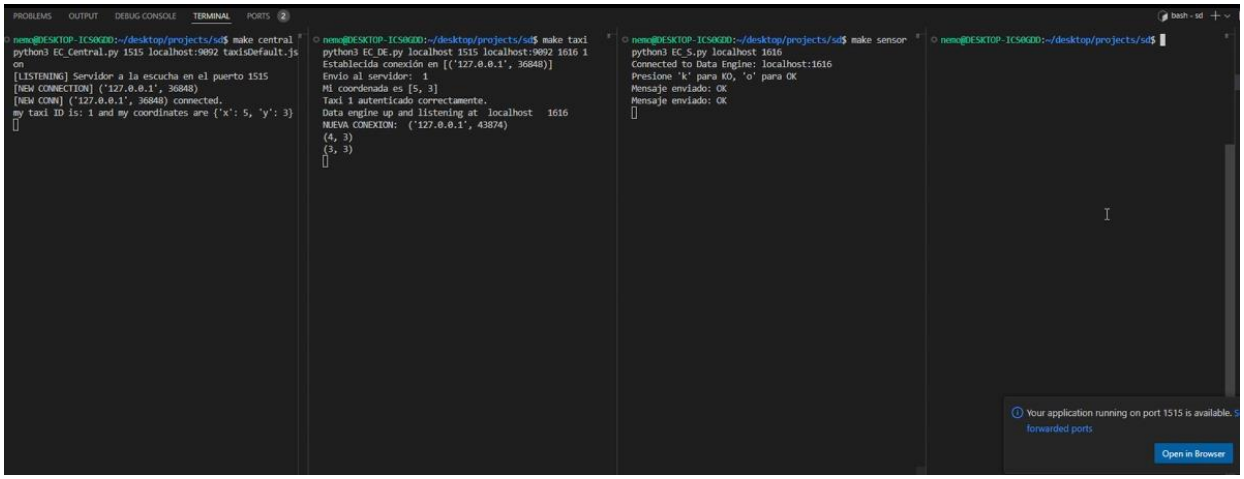
Activar los Clientes:

→ Finalmente, volvemos al primer ordenador para activar y monitorear los clientes. A este punto, todos los componentes están ya operativos y los clientes pueden comenzar a interactuar con el sistema, enviando solicitudes para servicios de taxi y recibiendo respuestas.

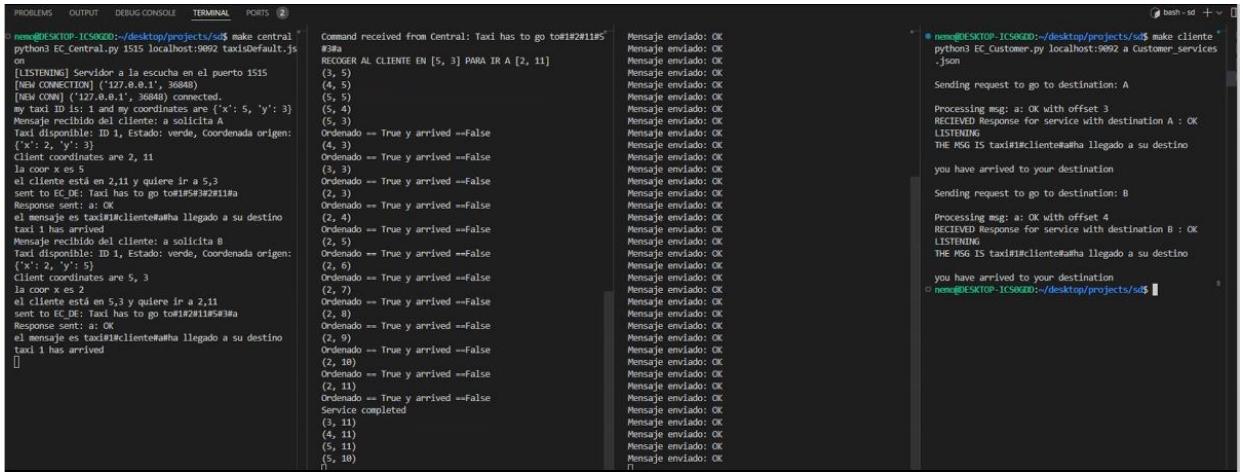
Para la ejecución se ha creado tanto un makefile como un .bat para la ejecución de Kafka y ejecución de los scripts para tener un poco mas de posibilidades de despliegue, el makefile usa apache dockerizado.

EJEMPLO DE USO Y EJECUCIÓN:

Iniciamos la central y añadimos un taxi junto a su sensor:



En esta imagen, ya se ha ejecutado el servicio de un cliente y se ha completado:



El mapa se ve de la siguiente manera: