




# SISTEMAS DISTRIBUIDOS

## PRÁCTICA 1

### EASYCAB

3º INGENIERÍA INFORMÁTICA  
IZAN ALMODÓVAR ACEVEDO 51773287B  
RAUL CHINCHILLA ALBERT 50383884S



Antes de abordar en los componentes vamos a explicar el uso de la lectura de parámetros:

Usaremos un .json

Aquí podemos cambiar los parámetros sin recompilar el código

```
{
  "central": {
    "ip": "127.0.0.1",
    "port": 8000
  },
  "taxi": {
    "taxi_id": 4,
    "sensors_port": 12004,
    "broker": "localhost:9092",
    "posicion_inicial": [1, 1]
  },
  "cliente": {
    "client_id": "C",
    "ubicacion_inicial": [3,7],
    "broker": "localhost:9092",
    "topic_request_taxi": "taxi_requests",
    "topic_confirmation": "taxi_confirmation",
    "Requests":[
      {
        "Id":"A"
      },
      {
        "Id":"F"
      }
    ]
  }
}
```

Usaremos dentro de cada componente una función de lectura de parámetros

(Ejemplo en EC\_Central aunque es muy similar en los demás)

```
# Función para cargar parámetros
def cargar_configuracion(file_path):
    try:
        with open(file_path, 'r') as config_file:
            config = json.load(config_file)
            return config
    except Exception as e:
        print(f"Error al cargar el archivo de configuración: {e}")
        return None

# Cargar la configuración desde un archivo JSON
config = cargar_configuracion('config.json')

# Parámetros de configuración de Kafka
TOPIC_REQUEST_TAXI = config["cliente"]["topic_request_taxi"]
TOPIC_CONFIRMATION = config["cliente"]["topic_confirmation"]
BROKER = config["taxi"]["broker"]
```

Procedemos a explicar cada componente software.

### EC\_CENTRAL:

Esta es la parte más importante de la aplicación ya que es la que maneja las peticiones de los clientes y la asignación de los taxis a los clientes.

Tenemos varios tópicos para conectarlos con Kafka:

Topic\_request\_taxi sirve para recibir peticiones a los clientes

Topic\_confirmation lo usamos para enviar al cliente mensajes con información

También tenemos los sockets para conectar los taxis con la central

Central\_socket\_ip

Central\_socket\_port

Inicia los sockets:

```
def iniciar_socket_taxi(taxis_activos):
    """Inicia el socket para recibir conexiones de taxis y autenticarlos."""
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((CENTRAL_SOCKET_IP, CENTRAL_SOCKET_PORT))
    server_socket.listen(5)
    print(f"Central esperando taxis en {CENTRAL_SOCKET_IP}:{CENTRAL_SOCKET_PORT}...")

    while True:
        connection, address = server_socket.accept()
        print(f"Conexión entrante de {address}")
        threading.Thread(target=manejar_conexion_taxi, args=(connection, taxis_activos)).start()
```

Comprueba que la id del taxi que quiere registrarse esté en la base de datos:

```
def autenticar_taxi(taxi_id, taxis_activos):
    """Autentica un taxi comparando su ID contra los datos de la base de datos."""
    if taxi_id in taxis_activos:
        print(f"Taxi {taxi_id} autenticado con éxito.")
        return True
    else:
        print(f"Taxi {taxi_id} rechazado. No está activo o no registrado.")
        return False
```

Disponemos de una base de datos:

```
1, no, rojo, -, -, -
2, no, rojo, -, -, -
3, no, rojo, -, -, -
4, no, rojo, -, -, -
5, no, rojo, -, -, -
```

En la que se basa en:

Id\_taxi, disponibilidad para los clientes, movimiento, destino del taxi, cliente subido

Tenemos esta función para leer la base de datos:

```
def leer_base_datos(file_path='bdd.txt'):
    """Lee el archivo de la base de datos y devuelve un diccionario con la información de cada taxi."""
    taxis = {}
    try:
        with open(file_path, 'r') as f:
            for line in f:
                taxi_id, libre, estado, coord_x_destino, coord_y_destino, cliente = line.strip().split(',')
                origen_taxi = config["taxi"]["posicion_inicial"]
                taxi_id = int(taxi_id)
                libre = libre.strip().lower() == 'si'
                taxis[taxi_id] = {
                    "libre": libre,
                    "estado": estado.strip(),
                    "origen": origen_taxi,
                    "destino": (None if coord_x_destino.strip() == '-' else int(coord_x_destino),
                               None if coord_y_destino.strip() == '-' else int(coord_y_destino)),
                    "cliente": None if cliente == '-' else cliente
                }
    except FileNotFoundError:
        print("El archivo de base de datos no existe.")
    except Exception as e:
        print(f"Error al leer la base de datos: {e}")
    return taxis
```

Lee la base de datos y se le asigna al array taxis

Y para actualizar la base de datos:

```
#Usar en casos que cambiemos el estado de un taxi
def actualizar_base_datos(taxis_activos, file_path='bdd.txt'):
    with open(file_path, 'w') as f:
        for taxi_id, datos in taxis_activos.items():
            estado = "si" if datos["libre"] else "no"
            destino_x = datos["destino"][0] if datos["destino"][0] is not None else '-'
            destino_y = datos["destino"][1] if datos["destino"][1] is not None else '-'
            cliente = datos["cliente"].strip() if datos["cliente"] else '-' # Usamos strip() para limpiar espacios extra
            f.write(f"{taxi_id}, {estado}, {datos['estado']}, {destino_x}, {destino_y}, {cliente}\n")
    print("Base de datos actualizada.")
```

Para asignar un taxi a un cliente:

```
def asignar_taxi(solicitud, taxis_activos):
    """Busca un taxi libre y lo asigna a la solicitud del cliente."""
    for taxi_id, datos in taxis_activos.items():
        if datos["libre"]:
            taxis_activos[taxi_id]["libre"] = False
            taxis_activos[taxi_id]["estado"] = "verde"
            taxis_activos[taxi_id]["destino"] = solicitud["destino"]
            #taxis_activos[taxi_id]["cliente"] = solicitud["client_id"] #Hacemos que se introduzca en la base de datos cuando ya esté montado

            #Confirmar asignación al cliente
            print(f"Asignando taxi {taxi_id} al cliente {solicitud['client_id']}")
            producer.send(TOPIC_CONFIRMATION, {"client_id": solicitud["client_id"], "mensaje": f"Taxi {taxi_id} asignado"})
            producer.flush()

            # Enviar ubicación de recogida y destino al taxi
            enviar_destino_a_taxi(taxi_id, solicitud["ubicacion_actual"], solicitud["destino"])

            # Actualizar base de datos después de la asignación
            actualizar_base_datos(taxis_activos)

            actualizar_mapa("taxi", taxi_id, estado=estado_taxi)
            return

    print("No hay taxis libres disponibles")
```

Nada más conectar el taxi tenemos esta función para manejar la conexión:

```
def manejar_conexion_taxi(connection, taxis_activos):
    """Maneja la conexión con un taxi a través de sockets."""
    try:
        data = connection.recv(1024).decode().strip()
        taxi_id = int(data)
        if autenticar_taxi(taxi_id, taxis_activos):
            # Cambiar el estado a disponible y color rojo al autenticarse
            taxis_activos[taxi_id]["libre"] = True
            taxis_activos[taxi_id]["estado"] = "rojo"
            taxis_activos[taxi_id]["destino"] = (None, None)
            taxis_activos[taxi_id]["cliente"] = None
            print(f"Taxi {taxi_id} autenticado con éxito y disponible en estado 'rojo'.")
            connection.send("Autenticación exitosa".encode())

            # Obtener posición inicial desde la base de datos o usar una por defecto
            posicion_inicial = taxis_activos[taxi_id].get("origen") # Posición por defecto en (1,1)
            actualizar_mapa("taxi", taxi_id, posicion_inicial, estado="rojo") # Actualiza el mapa con el taxi en rojo
            actualizar_base_datos(taxis_activos) # Actualizar base de datos
        else:
            print(f"Taxi {taxi_id} rechazado. No está activo o no registrado.")
            connection.send("Autenticación fallida".encode())
    except ValueError:
        print(f"Error: Se esperaba un número para el ID del taxi, pero se recibió: {data}")
    finally:
        connection.close()
```

Para mandar rumbo al taxi tenemos la siguiente función:

```
# Función para enviar el destino al taxi a través de Kafka
def enviar_destino_a_taxi(taxi_id, ubicacion_cliente, destino):
    topic_taxi_commands = f'central_commands_{taxi_id}'
    mensaje = {
        "ubicacion_cliente": ubicacion_cliente, # Ubicación de recogida como lista de enteros
        "destino_final": destino, # Destino final también en lista de enteros
        "asignado": True
    }
    producer.send(topic_taxi_commands, value=mensaje)
    producer.flush()
    print(f"Ubicación del cliente {ubicacion_cliente} y destino {destino} enviados al taxi {taxi_id} en el tópic {topic_taxi_commands}")
```

Y para liberarlo para cuando ya no haga falta que siga yendo a por un cliente:

```
def liberar_taxi(taxi_id, taxis_activos):
    """Libera un taxi cuando ha terminado su recorrido."""
    taxis_activos[taxi_id]["libre"] = True
    taxis_activos[taxi_id]["estado"] = "rojo"
    taxis_activos[taxi_id]["destino"] = (None, None)
    taxis_activos[taxi_id]["cliente"] = None
    print(f"Taxi {taxi_id} liberado y listo para nuevas asignaciones.")

    # Actualizar base de datos después de liberar el taxi
    actualizar_base_datos(taxis_activos)
```

Además, tenemos funciones para escuchar por los tópicos mensajes de actualización tanto de los clientes (peticiones) como de los taxis (actualizaciones del destino y clientes que llevan).

```

def escuchar_actualizaciones_taxi(taxis_activos):
    global pos_final
    """Escucha actualizaciones de posición de los taxis en Kafka y actualiza el mapa en tiempo real."""
    print("Central escuchando actualizaciones de posición de los taxis en Kafka...")

    # Consumidor de Kafka con patrón de tópicos
    consumer_status = KafkaConsumer(TOPIC_TAXI_STATUS_PATTERN)

    for mensaje in consumer_status:
        data = json.loads(mensaje.value.decode('utf-8'))
        print(f"Datos recibidos: {data}")
        taxi_id = data["taxi_id"]
        nueva_pos = data["pos"]

        if taxi_id and nueva_pos:
            estado_taxi = taxis_activos[taxi_id]["estado"]

            for client_id, ubicacion_cliente in solicitudes_pendientes.items():
                if nueva_pos == ubicacion_cliente:
                    #cliente se monta
                    taxis_activos[taxi_id]["cliente"] = client_id
                    actualizar_mapa("taxi", taxi_id, nueva_pos, estado=estado_taxi) # Actualiza el mapa

                    #del solicitudes_pendientes[client_id]
                    print(f"Cliente '{client_id}' subió al taxi {taxi_id}")
                    actualizar_base_datos(taxis_activos)

            actualizar_mapa("taxi", taxi_id, nueva_pos, estado=estado_taxi)
            print(f"Central actualizó posición del Taxi {taxi_id} a {nueva_pos}")

            if taxis_activos[taxi_id]["destino"] == nueva_pos:
                pos_final = nueva_pos
                manejar_llegada_destino(taxis_activos, taxi_id)

def escuchar_peticiones_cliente(taxis_activos):
    """Escucha continuamente las peticiones de los clientes en Kafka y procesa cada solicitud."""
    print("Central escuchando peticiones de clientes en Kafka...")
    for mensaje in consumer:
        solicitud = mensaje.value
        client_id = solicitud["client_id"]
        ubicacion_cliente = solicitud["ubicacion_actual"]

        print(f"Solicitud de cliente recibida: {solicitud}")
        actualizar_mapa("cliente", solicitud["client_id"], solicitud["ubicacion_actual"], None)

        solicitudes_pendientes[client_id] = ubicacion_cliente

        asignar_taxi(solicitud, taxis_activos)

```

Por último, tenemos todo lo relacionado con el mapa:

Pintar el mapa inicial:

```
def pintar_mapa_inicial():
    """Dibuja el mapa inicial y coloca las ubicaciones predefinidas desde `locations`."""
    print("\n" * 5)
    sys.stdout.write(LINE)
    sys.stdout.write(f'':<20} *** EASY CAB Release 1 ***\n")
    sys.stdout.write(LINE)

    # Encabezado de taxis y clientes
    sys.stdout.write(f'':<20} {'Taxis':<29} {'':<20} {'Clientes'}\n")
    sys.stdout.write(LINE)
    sys.stdout.write(f'':<8} {'Id.':<10} {'Destino':<15} {'Estado':<14} {'':<7} {'Id.':<10} {'Destino':<15} {'Estado':<15}\n")
    sys.stdout.write(LINE)

    sys.stdout.write(LINE + "\n")
    sys.stdout.write(" " + " ".join([f"{i:2}" for i in range(1, 21)]) + "\n") # Ajuste de ancho
    sys.stdout.write(LINE + "\n")

    # Limpiar el mapa y poner los puntos de interés
    for x in range(20):
        for y in range(20):
            mapa[x][y] = EMPTY # Cada celda como un punto

    # Colocar ubicaciones desde `locations`
    for loc_id, (x, y) in locations.items():
        mapa[x - 1][y - 1] = Fore.BLUE + loc_id + Style.RESET_ALL # Identificación de ubicación por ID

    # Imprimir filas del mapa con ubicaciones
    for row in range(20):
        sys.stdout.write(f"{row + 1:<2} ")
        for col in range(20):
            sys.stdout.write(f"{mapa[row][col]:<2} ")
        sys.stdout.write("\n") # Nueva línea después de cada fila

    sys.stdout.write(LINE + "\n")
    sys.stdout.flush()
```

Luego, tenemos la función para ir pintando el mapa:

```
def pintar_mapa():
    """Dibuja el mapa inicial y coloca las ubicaciones predefinidas desde `locations`."""
    print("\n" * 5)
    sys.stdout.write(LINE)
    sys.stdout.write(f'':<20} *** EASY CAB Release 1 ***\n")
    sys.stdout.write(LINE)

    # Encabezado de taxis y clientes
    sys.stdout.write(f'':<20} {'Taxis':<29} {'':<20} {'Clientes'}\n")
    sys.stdout.write(LINE)
    sys.stdout.write(f'':<8} {'Id.':<10} {'Destino':<15} {'Estado':<14} {'':<7} {'Id.':<10} {'Destino':<15} {'Estado':<15}\n")
    sys.stdout.write(LINE)

    sys.stdout.write(LINE + "\n")
    sys.stdout.write(" " + " ".join([f"{i:2}" for i in range(1, 21)]) + "\n") # Ajuste de ancho
    sys.stdout.write(LINE + "\n")

    # Imprimir filas del mapa con ubicaciones
    for row in range(20):
        sys.stdout.write(f"{row + 1:<2} ")
        for col in range(20):
            sys.stdout.write(f"{mapa[row][col]:<2} ")
        sys.stdout.write("\n") # Nueva línea después de cada fila

    sys.stdout.write(LINE + "\n")
    sys.stdout.flush()

taxi_activos = leer_base_datos() # variable global para leer la base de datos
```

Y luego para ir actualizándolo con los taxis, los clientes y los colores respectivos:

```
def actualizar_mapa(tipo, id_, posicion, estado=None):
    x, y = posicion
    if tipo == "taxi":
        # Determina el color en función del estado
        color = Fore.GREEN if estado == "verde" else Fore.RED

        # Limpia la posición anterior del taxi en el mapa
        for fila in range(20):
            for col in range(20):
                if mapa[fila][col] == f"{Fore.GREEN}{id_}{Style.RESET_ALL}" or mapa[fila][col] == f"{Fore.RED}{id_}{Style.RESET_ALL}":
                    mapa[fila][col] = EMPTY

        # Actualiza las ubicaciones en el mapa
        for loc_id, (a, b) in locations.items():
            mapa[a - 1][b - 1] = Fore.BLUE + loc_id + Style.RESET_ALL

        # Representa el taxi con el cliente si hay uno en la base de datos
        cliente = taxis_activos[id_].get("cliente")
        if cliente:
            for fila in range(20):
                for col in range(20):
                    if mapa[fila][col] == f"{color}{id_}{cliente}{Style.RESET_ALL}":
                        mapa[fila][col] = EMPTY

            mapa[x - 1][y - 1] = f"{color}{id_}{cliente}{Style.RESET_ALL}"
        else:
            mapa[x - 1][y - 1] = f"{color}{id_}{Style.RESET_ALL}"

    elif tipo == "cliente":
        color = Fore.YELLOW
        mapa[x - 1][y - 1] = f"{color}{id_}{Style.RESET_ALL}"

    # Llama a la función para mostrar el mapa actualizado
    pintar_mapa()
```

## EC\_DIGITAL:

Este componente se encarga de conectarse con el sensor y mandarle información a la central, además de recorrer el mapa:

Con lo relacionado a los sensores tenemos su conexión con el taxi (la cual hasta que no pasa no permite conectarse a la central) :

```
def conectar_con_sensor():
    """Espera y se conecta al sensor en el puerto configurado."""
    try:
        sensor_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sensor_socket.bind(('0.0.0.0', SENSORS_PORT))
        sensor_socket.listen(1)
        print(f"Esperando la conexión del sensor en el puerto {SENSORS_PORT}...")
        conn, addr = sensor_socket.accept()
        ready_message = conn.recv(1024).decode()
        if ready_message == str(TAXI_ID):
            print(f"Sensor del taxi {TAXI_ID} conectado.")
            return conn # Devuelve la conexión establecida
        else:
            print(f"Error: Sensor del taxi {TAXI_ID} no se pudo conectar.")
            conn.close()
            return None
    except Exception as e:
        print(f"Error al conectar con el sensor: {e}")
        return None
```

Y la escucha permanente a los sensores para enviar si sucede una incidencia (OK/KO):



```
def escuchar_sensores(conn):
    global taxi_status
    while True:
        try:
            sensor_data = conn.recv(1024).decode()
            if sensor_data:
                print(f'Sensor envió: {sensor_data}')
                taxi_status = sensor_data
                if taxi_status == 'KO':
                    actualizar_estado_en_central("rojo")
                elif taxi_status == 'OK': # Si vuelve a OK y está en movimiento
                    actualizar_estado_en_central("verde")
            except (ConnectionResetError, ConnectionAbortedError):
                print("Conexión con el sensor perdida. Intentando reconectar...")
                conn.close()
                time.sleep(1)
                conn = conectar_con_sensor()
                if not conn:
                    print("No se pudo reconectar al sensor. Terminando escucha.")
                    break
        except Exception as e:
            print(f"Error inesperado en la comunicación con el sensor: {e}")
            break
```

Necesitamos conectarnos con la central para posteriormente interactuar con ella:

```
def autenticar_con_central():
    """Autentica el taxi con la central después de conectar con el sensor."""
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.connect((CENTRAL_IP, CENTRAL_PORT))
            s.sendall(f"{TAXI_ID}".encode())
            respuesta = s.recv(1024).decode()
            if respuesta == "Autenticación exitosa":
                print(f"Taxi {TAXI_ID} autenticado con éxito en la central.")
                return True
            else:
                print(f"Taxi {TAXI_ID} autenticación fallida: {respuesta}")
                return False
    except Exception as e:
        print(f"Error de conexión con la central: {e}")
        return False

# Modificar la función mover_taxi hacia para enviar posición del taxi
```

Para escuchar los mandamientos de la central debemos tener una función de escucharla:

```
def escuchar_destino():
    """Escucha el destino desde Kafka y mueve el taxi hacia allí solo cuando recibe una solicitud de destino válida."""
    print(f"Esperando destino en el tópico {TOPIC_TAXI_COMMANDS}...")

    # Asegurarse de que el consumidor esté suscrito y tenga particiones asignadas
    consumer.subscribe([TOPIC_TAXI_COMMANDS])
    consumer.poll(0) # Forzar la suscripción inmediata

    # Espera hasta que se asignen particiones
    while not consumer.assignment():
        print("Esperando asignación de particiones...")
        time.sleep(0.5)

    # Limpia mensajes residuales para evitar movimientos no deseados
    consumer.seek_to_end() # Coloca el consumidor al final del tópico

    # Procesa solo mensajes nuevos y válidos
    for message in consumer:
        comando = message.value
        print(f"Mensaje recibido de la central: {comando}")

        # Verifica que el mensaje tenga la estructura esperada y la solicitud sea confirmada
        if (isinstance(comando, dict) and
            "ubicacion_cliente" in comando and
            "destino_final" in comando and
            comando.get("asignado", False) == True): # Verifica si está asignado

            ubicacion_cliente = comando["ubicacion_cliente"]
            destino_final = comando["destino_final"]

            # Asegúrate de que destino_final esté en el formato correcto
            if isinstance(destino_final, str):
                destino_final = list(map(int, destino_final.split(',')))

            # Mueve el taxi a la ubicación del cliente primero
            print(f"Destino de recogida recibido: {ubicacion_cliente}")
            mover_taxi_hacia(ubicacion_cliente[0], ubicacion_cliente[1])

            # Luego, al destino final
            print(f"Destino final recibido: {destino_final}")
            mover_taxi_hacia(destino_final[0], destino_final[1])
```

Y nosotros a ella le mandamos actualizaciones:

```
def actualizar_estado_en_central(color):
    """Envía el color actual del taxi (rojo o verde) a la central para actualización en la base de datos."""
    mensaje = {
        "taxi_id": TAXI_ID,
        "estado": color,
        "pos": taxi_pos
    }
    producer.send(TOPIC_TAXI_ESTADO, json.dumps(mensaje).encode())
    producer.flush()
```

Por último tenemos todo lo relacionado con el movimiento del taxi como la función de mover taxi, la cual le pasamos un destino y se mueve sobre un mapa circular para llegar al destino

```
# Modificar la función mover_taxi_hacia para enviar posición del taxi
def mover_taxi_hacia(destino_x, destino_y):
    """Función para mover el taxi hacia la posición de destino en un mapa circular de 20x20."""
    global taxi_pos, taxi_status

    print(f"Iniciando movimiento hacia el destino: [{destino_x}, {destino_y}]")
    print(f"Taxi inicia en posición: {taxi_pos}")

    # Cambiar estado a verde al iniciar el movimiento
    # actualizar_estado_en_central("verde")

    while True:
        actualizar_estado_en_central("verde")
        # Si el taxi está en KO, cambia el estado a "rojo" en la central y espera
        if taxi_status == 'KO':
            actualizar_estado_en_central("rojo") # Reflejar el estado en la central
            print(f'Taxi {TAXI_ID} en estado KO, esperando cambio a OK...')
            producer.send(TOPIC_TAXI_STATUS, f'TAXI_{TAXI_ID}_KO'.encode()) # Notificar KO a Kafka
            producer.flush()
            time.sleep(2)
            continue

        # Verificar si ha alcanzado el destino
        if taxi_pos == [destino_x, destino_y]:
            print(f'Taxi {TAXI_ID} ha llegado al destino {taxi_pos}')
            producer.send(TOPIC_TAXI_STATUS, f'Taxi {TAXI_ID} ha llegado al destino {taxi_pos}'.encode())
            producer.flush()
            actualizar_estado_en_central("rojo") # Cambiar a "rojo" al llegar al destino
            break

        # Calcular delta con movimiento circular en X
        delta_x = (destino_x - taxi_pos[0]) % 20
        if delta_x > 10:
            delta_x -= 20
```

(hay más pero eso se verá reflejado en el código)

Y luego la función de realizar un recorrido para llevar a los clientes:

```
def realizar_recorrido(ubicacion_cliente, destino_final):
    """Gestiona el proceso de recoger al cliente y luego llevarlo a su destino final."""
    # Asegurarse de que destino_final sea una lista de enteros
    if isinstance(destino_final, str):
        destino_final = list(map(int, destino_final.split(',')))

    print(f"Taxi {TAXI_ID} dirigiéndose a la ubicación del cliente en {ubicacion_cliente} para recogerlo.")

    # Ir a la ubicación del cliente
    mover_taxi_hacia(*ubicacion_cliente)
    print(f"Taxi {TAXI_ID} ha llegado a la ubicación del cliente en {ubicacion_cliente}")
    producer.send(TOPIC_TAXI_STATUS, f"Taxi {TAXI_ID} ha llegado a la ubicación del cliente en {ubicacion_cliente}".encode())
    producer.flush()

    # Indicar que el cliente ha subido al taxi
    print(f"Cliente ha subido al taxi {TAXI_ID}")
    producer.send(TOPIC_TAXI_STATUS, f"Cliente ha subido al taxi {TAXI_ID}".encode())
    producer.flush()

    # Llevar al cliente al destino final
    print(f"Taxi {TAXI_ID} llevando al cliente al destino {destino_final}")
    mover_taxi_hacia(*destino_final)

    # Notificar a la central que ha llegado al destino final
    print(f"Taxi {TAXI_ID} ha llegado al destino final {destino_final}")
    mensaje_llegada = {
        "taxi_id": TAXI_ID,
        "destino": destino_final,
        "llegada": True
    }
    producer.send(TOPIC_TAXI_STATUS, json.dumps(mensaje_llegada).encode()) # Enviar llegada a la central
    producer.flush()

    # Poner el taxi en estado libre y rojo
    actualizar_estado_en_central("rojo")
    taxi_pos = config["taxi"]["posicion_inicial"] # Reiniciar posición al punto inicial
```

## EC\_S:

Este componente es bastante simple, se basa en mandarle al taxi cada segundo (OK/KO) y la detección de una tecla para cambiar este mensaje

Tenemos la conexión con el taxi mediante sockets:

```
def conectar_con_taxi():
    """Establece la conexión con el taxi una vez y solo intenta reconectar si ocurre un error."""
    while True:
        try:
            taxi_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            taxi_socket.connect((CENTRAL_IP, SENSOR_PORT))
            taxi_socket.send(str(TAXI_ID).encode())
            print(f"Sensor del taxi {TAXI_ID} conectado.")
            return taxi_socket # Retorna el socket si la conexión es exitosa

        except (ConnectionRefusedError, ConnectionAbortedError):
            print("Taxi no disponible o conexión perdida. Reintentando en 2 segundos...")
            taxi_socket.close()
            time.sleep(2)
        except Exception as e:
            print(f"Error inesperado al conectar con el taxi: {e}")
            taxi_socket.close()
            break

    return None
```

El envío del mensaje cada segundo:

```
def mostrar_estado(taxi_socket):
    """Muestra el estado actual del sensor y lo envía al Digital Engine cada segundo."""
    global MENSAJE
    while True:
        print(f"Estado actual del taxi {TAXI_ID}: {MENSAJE}")

        # Enviar el estado actual al Digital Engine
        try:
            taxi_socket.sendall(MENSAJE.encode())
        except (ConnectionResetError, ConnectionAbortedError):
            print("Error al enviar estado al Digital Engine. Conexión cerrada.")
            break
        except Exception as e:
            print(f"Error al enviar estado al Digital Engine: {e}")
            break

        time.sleep(1) # Enviar el estado cada segundo
```

Y el cambio de mensaje mediante el pulso de una tecla (en este caso da igual cuál)

Para ello usamos la librería msvcrt:

```
def listen_for_key_press(taxi_socket):
    """Cambia el estado entre 'OK' y 'KO' al presionar una tecla."""
    global MENSAJE
    message = 'OK'
    while True:
        if msvcrt.kbhit():
            key = msvcrt.getch()
            if key:
                message = 'KO' if message == 'OK' else 'OK'
                MENSAJE = message # Actualizar el mensaje global
                print(f"Sensor del taxi {TAXI_ID} cambió el estado a {message}")
                try:
                    taxi_socket.sendall(message.encode())
                except (ConnectionResetError, ConnectionAbortedError):
                    print("Error al enviar mensaje al taxi. Conexión cerrada.")
                    break
                except Exception as e:
                    print(f"Error al enviar mensaje al taxi: {e}")
                    break

            time.sleep(0.1) # Reducir el tiempo de espera para detectar cambios de tecla rápidamente

def mostrar_estado(taxi_socket):
```

**EC\_Customer:**

Por último componente tenemos al customer, el cliente que enviará peticiones a la central y será recogido por los taxis, estos clientes van a tener una id y una lista de destinos a los que ir:

Primeramente tenemos la solicitud a la central para pedir un taxi, aquí llevará incluido el destino a donde quiere ir el cliente:

```
def solicitar_taxi(destino):
    """Envía una solicitud de taxi con la ubicación actual y el destino especificado."""
    solicitud = {
        "client_id": CLIENT_ID,
        "ubicacion_actual": UBICACION,
        "destino": destino
    }
    print(f"Cliente {CLIENT_ID} solicitando taxi desde {UBICACION} hacia {destino}")
    producer.send(TOPIC_REQUEST_TAXI, json.dumps(solicitud).encode('utf-8'))
    producer.flush()
```

Luego el cliente espera un mensaje de la central indicándole que ya ha llegado al destino y se actualiza la ubicación del cliente a donde está:

```
def esperar_confirmacion_llegada():
    global UBICACION
    """Espera la confirmación de que el taxi ha llegado al destino final."""
    print("Esperando confirmación de llegada al destino...")
    for message in consumer:
        confirmacion = json.loads(message.value.decode())

        if confirmacion.get("client_id") == CLIENT_ID:
            mensaje = confirmacion.get("mensaje", "")
            print(f"Cliente '{CLIENT_ID}' recibió mensaje: {mensaje}")

            if "ha llegado a su destino" in mensaje:
                ubicacion = confirmacion.get("ubicacion", "")
                UBICACION = ubicacion
                print("Cliente ha llegado a su destino. Solicitará el siguiente en 5 segundos.")
                break # Sale del bucle cuando recibe la confirmación de llegada
```

Por último tenemos la función, que como en realizar\_recorrido en EC\_DE, reorganiza todos los recorridos y hace el bucle de peticiones:

```
def solicitar_destinos():
    """Solicita taxis secuencialmente para los destinos en la lista 'Requests'."""
    requests = config["cliente"]["Requests"]
    for request in requests:
        destino_id = request["Id"]

        # Verificar si el destino existe en las ubicaciones definidas
        if destino_id in locations:
            destino = locations[destino_id]
            solicitar_taxi(destino) # Enviar solicitud al destino actual
            esperar_confirmacion_llegada() # Espera a que el taxi confirme la llegada
            time.sleep(5) # Espera 5 segundos antes de solicitar el siguiente destino
        else:
            print(f"Destino '{destino_id}' no encontrado en las ubicaciones.")
```

Procedemos a mostrar el funcionamiento del código:

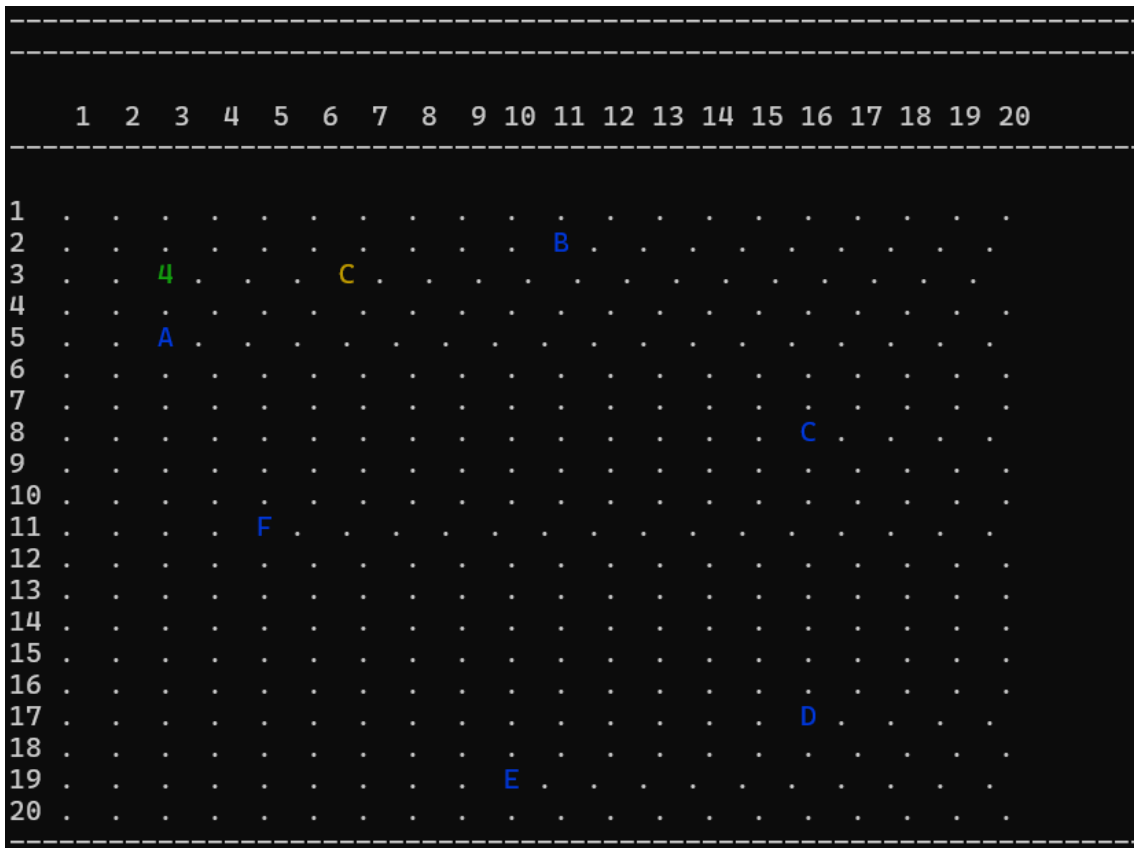
La central con los destinos puestos en él:

```
1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
-----
1  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
2  .  .  .  .  .  .  .  .  .  .  B  .  .  .  .  .  .  .  .
3  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
4  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
5  .  A  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
6  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
7  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
8  .  .  .  .  .  .  .  .  .  .  .  C  .  .  .  .  .  .
9  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
10 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
11 .  .  .  F  .  .  .  .  .  .  .  .  .  .  .  .  .  .
12 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
13 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
14 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
15 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
16 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
17 .  .  .  .  .  .  .  .  .  .  .  .  D  .  .  .  .  .
18 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
19 .  .  .  .  .  .  .  E  .  .  .  .  .  .  .  .  .  .
20 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
-----
Central escuchando peticiones de clientes en Kafka...
Central escuchando actualizaciones de posición de los taxis en Kafka...Central esperando taxis en 127.0.0.1:8000...
```

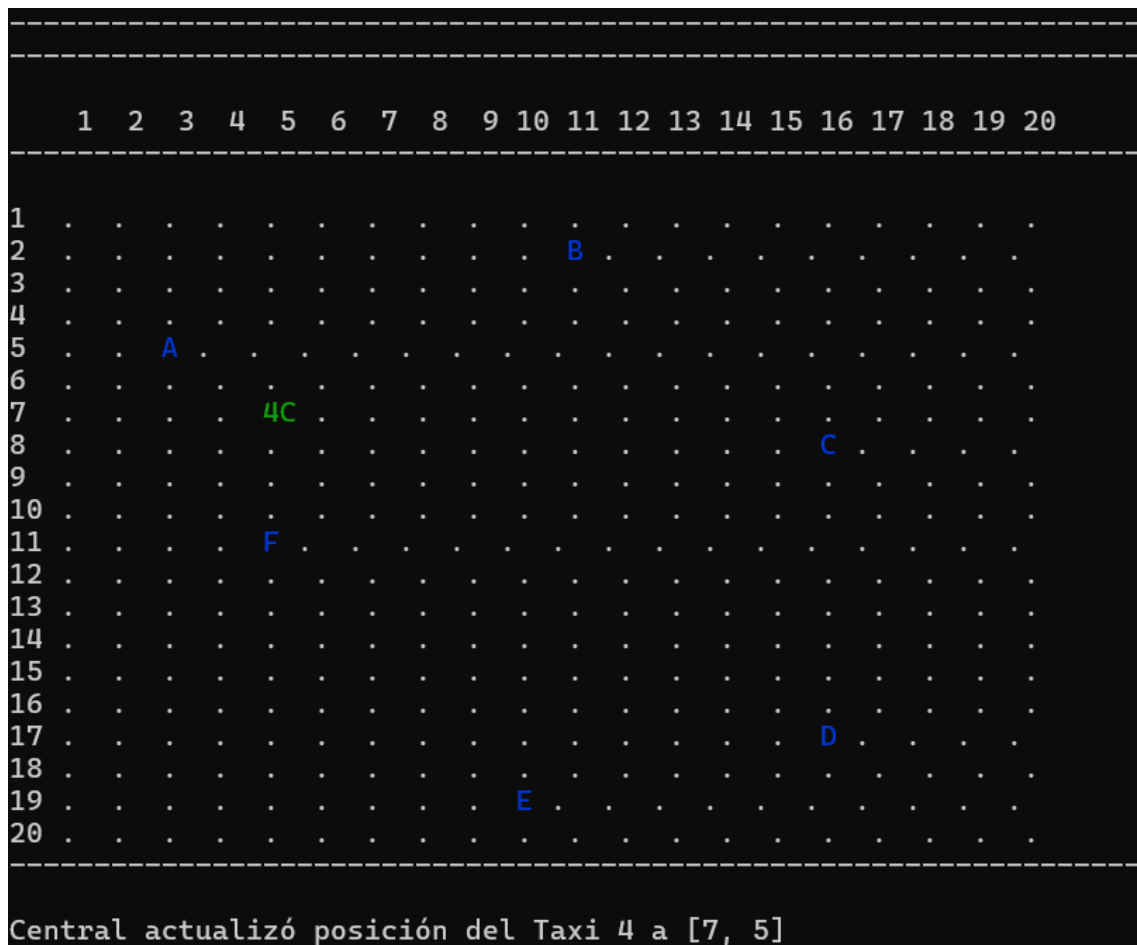
Un taxi:

```
1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
-----
1  4  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
2  .  .  .  .  .  .  .  .  .  .  B  .  .  .  .  .  .  .
3  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
4  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
5  .  .  A  .  .  .  .  .  .  .  .  .  .  .  .  .  .
6  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
7  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
8  .  .  .  .  .  .  .  .  .  .  .  C  .  .  .  .  .
9  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
10 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
11 .  .  .  F  .  .  .  .  .  .  .  .  .  .  .  .  .
12 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
13 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
14 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
15 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
16 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
17 .  .  .  .  .  .  .  .  .  .  .  .  D  .  .  .  .
18 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
19 .  .  .  .  .  .  .  E  .  .  .  .  .  .  .  .  .
20 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
-----
```

Un cliente:



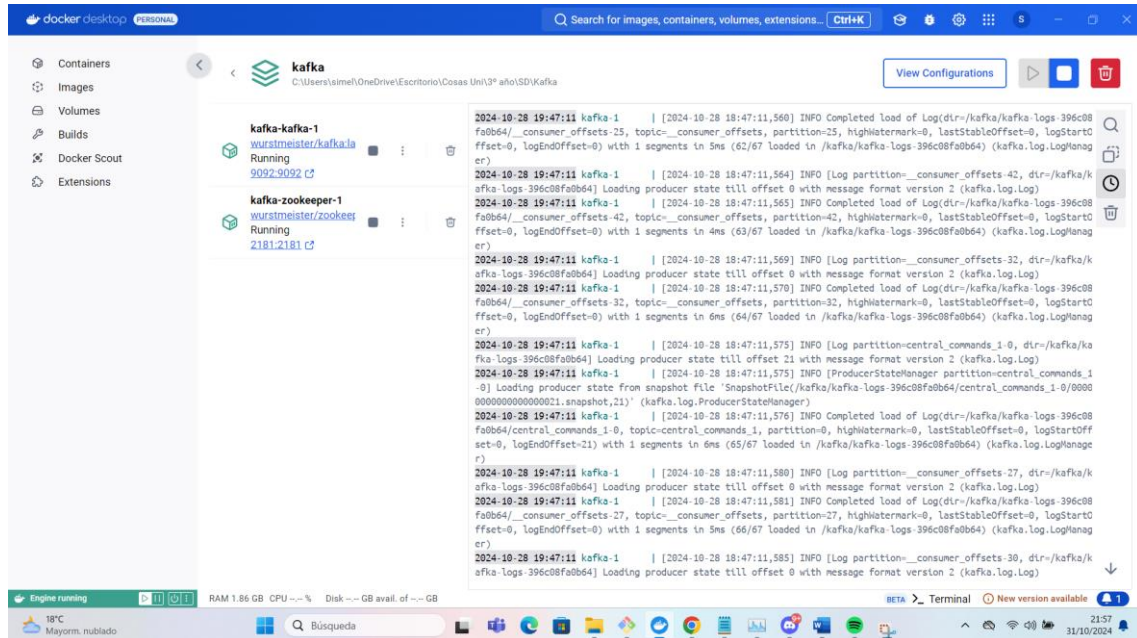
Un taxi con un cliente:



Despliegue de la aplicación:

Para el despliegue de nuestra aplicación se usará Docker Desktop y se hará en 3 portátiles:

Esta aplicación es bastante útil ya que con solo pulsar un botón se ejecuta tanto el Kafka como el Zookeeper



Uno corresponderá con la central, otro ordenador levantará los clientes y por último otro hará la función de los taxis.

Para ello dentro del config.json tenemos ip que se pueden cambiar si necesidad de recompilar.