**Memoria Resumen: Seminario 3 - Tráfico Inteligente en la SmartCity**

**1. Objetivo del Proyecto**

El objetivo de esta práctica ha sido desarrollar y simular un escenario de *Smart City* utilizando una arquitectura de Internet de las Cosas (IoT). La simulación implementa un flujo completo de eventos: un vehículo conectado (SmartCar) sufre un accidente, notifica el incidente a la infraestructura (SmartRoad y SmartCity) a través del protocolo MQTT, y desencadena una respuesta coordinada: la carretera alerta a otros vehículos y la ciudad envía una ambulancia mediante una llamada API REST.

**2. Entornos de Simulación**

Para la realización de la práctica, se han utilizado los siguientes entornos, siendo el **Entorno Virtual** el seleccionado para la implementación y pruebas finales.

|  |  |  |
| --- | --- | --- |
| **Componente** | **Entorno Virtual (Recomendado para pruebas remotas)** | **Entorno Físico (Accesible solo desde red UPV)** |
| **API REST** | http://tambori.dsic.upv.es:10082 | http://ttmi008.iot.upv.es:8182 |
| **Broker MQTT** | tcp://tambori.dsic.upv.es:10083 | tcp://ttmi008.iot.upv.es:1883 |

**3. Ejercicios de Prueba Iniciales (Fase de Exploración)**

Antes de la implementación, se realizaron las siguientes llamadas para familiarizarse con la API REST de la plataforma.

1. **Consultar longitud del segmento R5s1:**
   * **Método:** GET
   * **URL:** http://tambori.dsic.upv.es:10082/segment/R5s1
   * **Resultado:** Se obtuvo un JSON con los detalles del segmento, confirmando que su longitud ("length") es de 50 metros.
2. **Consultar información del vehículo virtual 21:**
   * **Método:** GET
   * **URL:** http://tambori.dsic.upv.es:10021/car
   * **Resultado:** Se recibió el estado completo del vehículo, identificando sus dispositivos: engine, steering, fdistance, etc.
3. **Controlar el motor del vehículo (hacerlo avanzar):**
   * **Método:** PUT
   * **URL:** http://tambori.dsic.upv.es:10021/car/engine
   * **Header:** Content-Type: application/json
   * **Body:** {"action": "forward", "speed": 80}
   * **Resultado:** Se envió el comando para poner en marcha el motor.

**4. Resumen de Endpoints Disponibles**

La plataforma ofrece un conjunto de endpoints REST y topics MQTT.

**Endpoints API REST:**

|  |  |  |
| --- | --- | --- |
| **Método** | **Endpoint** | **Descripción** |
| GET | /simulator | Obtiene el estado del simulador. |
| GET | /roads | Lista todas las carreteras. |
| GET | /road/{id} | Obtiene detalles de una carretera. |
| GET | /segment/{id} | Obtiene detalles de un segmento. |
| GET | /vehicles | Lista los vehículos virtuales activos. |
| POST | /vehicles | Crea un nuevo vehículo virtual. |
| GET, PUT, DELETE | /vehicle/{id} | Gestiona un vehículo virtual específico. |
| GET, PUT | /car, /car/engine, /car/steering, /navigator | Interactúa con un vehículo (físico o virtual). |

**Topics MQTT (Canales):**

|  |  |
| --- | --- |
| **Canal** | **Descripción** |
| .../alerts | Canal para notificar incidentes críticos (ej. accidentes). |
| .../info | Canal para difundir información general (ej. retenciones, retransmisión de alertas). |
| .../traffic | Canal para que los vehículos informen de su posición. |
| .../signals | Canal para notificar cambios en señales de tráfico. |

**5. Proceso de Implementación**

La implementación consistió en integrar la lógica de negocio proporcionada en las carpetas Paso2, Paso3a y Paso3b dentro de la estructura base del Paso1.

1. **Paso 2: Notificación de Accidente (Proyecto smartcar)**
   * Se **añadió** el fichero SmartCar\_InicidentNotifier.java al paquete smartcar.impl.
   * Se **reemplazaron** los ficheros SmartCar.java y SmartCarStarter.java.
   * **Resultado:** El SmartCar ahora tiene la capacidad de construir y publicar un mensaje de accidente en el topic .../alerts.
2. **Paso 3a: Retransmisión de Alerta (Proyecto smartroad)**
   * Se **añadió** SmartRoad\_InicidentNotifier.java al paquete smartroad.impl.
   * Se **reemplazaron** SmartRoad.java y SmartRoad\_RoadIncidentsSubscriber.java.
   * **Resultado:** El SmartRoad ahora escucha el topic /alerts y, al recibir un mensaje, lo retransmite al topic /info.
3. **Paso 3b: Envío de Ambulancia (Proyecto smartcity)**
   * Se **reemplazó** el fichero SmartCity\_RoadIncidentsSubscriber.java.
   * **Resultado:** La SmartCity ahora escucha el topic /alerts y, al recibir un mensaje, realiza una llamada API REST para activar un vehículo-ambulancia.
4. **Corrección de Incidencia:** Durante las pruebas, se detectó un error java.net.ConnectException: Connection timed out. Se diagnosticó que el smartcity intentaba contactar un vehículo **físico** (ttmi021.iot.upv.es) desde un entorno que solo tenía acceso a la red **virtual**.
   * **Solución:** Se modificó la URL en SmartCity\_RoadIncidentsSubscriber.java para que apuntara al vehículo **virtual** correspondiente: http://tambori.dsic.upv.es:10021/car/engine.

**6. Verificación Final del Sistema**

Para probar la solución completa, se siguió el siguiente protocolo:

1. **Preparación:** Se utilizó un cliente MQTT (MQTT.fx) suscrito al topic es/upv/pros/tatami/smartcities/traffic/PTPaterna/road/R5s1/# para monitorizar la comunicación.
2. **Ejecución (en orden estricto):**
   * Se lanzó la aplicación SmartCityStarter.
   * Se lanzó la aplicación SmartRoadStarter.
   * Finalmente, se lanzó la aplicación SmartCarStarter.
3. **Resultados Observados:**
   * La consola del **SmartCar** mostró la publicación de un mensaje JSON en el topic .../alerts.
   * La consola del **SmartRoad** mostró la recepción del mensaje en /alerts y su posterior retransmisión en /info.
   * La consola del **SmartCity** mostró la recepción del mensaje en /alerts y ejecutó la llamada API REST sin errores.
   * La consola del **SmartCar** (actuando como otro vehículo) recibió correctamente el mensaje retransmitido en el topic /info.
   * El cliente **MQTT.fx** mostró ambos mensajes (el de /alerts y el de /info), confirmando el flujo correcto de eventos.