



FUNDAMENTOS DE INTERNET DE LA COSAS

2024-2025

Universidad Carlos III de Madrid

Cuaderno 2- Ejercicio - 11

2024/2025

VISUALIZACIÓN DE DATOS IOT

Cuaderno 2 - Ejercicio 11

Universidad Carlos III de Madrid. Escuela Politécnica Superior

Objetivos

El propósito de este ejercicio consiste en:

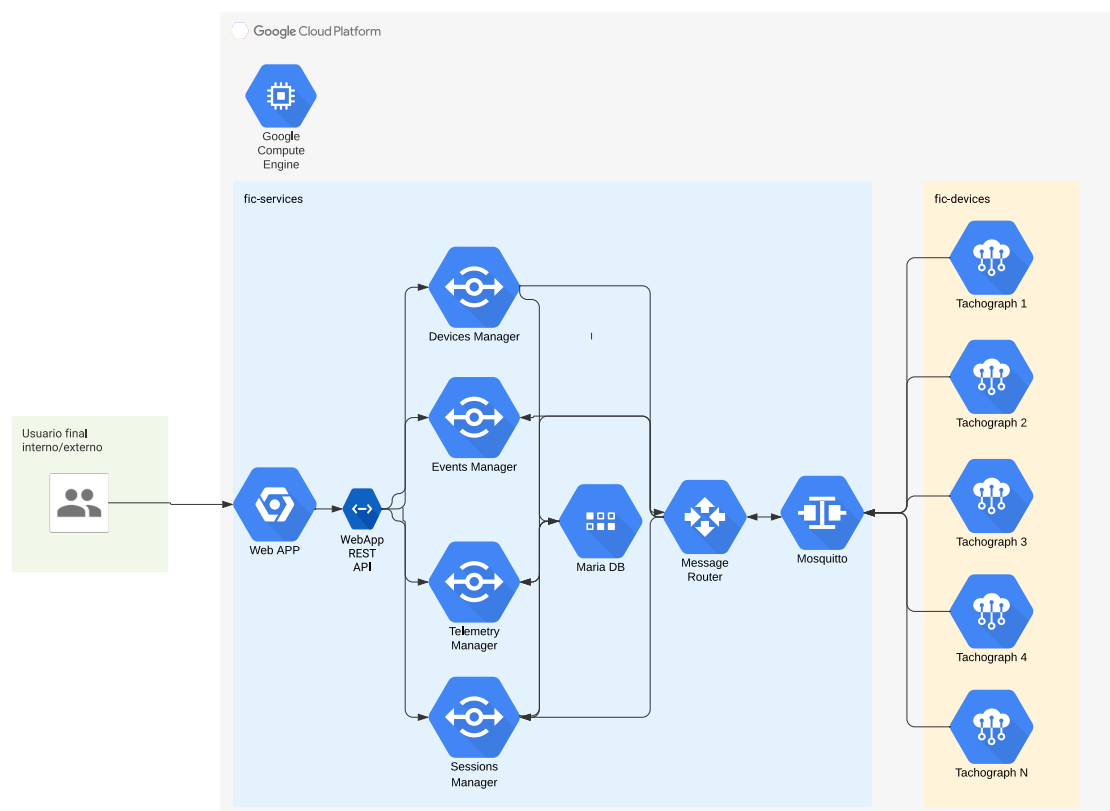
- Afianzar los conceptos relacionados con arquitecturas IoT.
- Desarrollar una API REST que encamine todas las peticiones que una aplicación de usuario pueda realizar sobre la arquitectura IoT.
- Desarrollar una interfaz de usuario simple que permita la visualización de datos y envío de configuraciones a los tacógrafos.

Introducción y pasos iniciales













































En el ámbito de este proyecto se partirá de la solución obtenida en la sesión 10.

Arquitectura y organización del proyecto

La arquitectura del proyecto es la siguiente:



La organización del código y los ficheros del proyecto será la siguiente:

- ▼  IoTCloudServices
 - ▼  dbservice
 -  Dockerfile
 -  initial_script.sql
 - ▼  message_router
 - >  code
 -  Dockerfile
 - ▼  microservices
 - ▼  devices_microservice
 - ▼  code
 -  devices_db_manager.py
 -  devices_manager_api.py
 -  requirements.txt
 -  Dockerfile
 - ▼  events_microservice
 - ▼  code
 -  events_db_manager.py
 -  events_manager_api.py
 -  requirements.txt
 -  Dockerfile
 - ▼  sessions_microservice
 - ▼  code
 -  requirements.txt
 -  sessions_db_manager.py
 -  sessions_manager_api.py
 -  Dockerfile
 - ▼  telemetry_microservice
 - ▼  code
 -  requirements.txt
 -  telemetry_db_manager.py
 -  telemetry_manager_api.py
 -  Dockerfile
 - ▼  mosquitto
 - >  code
 -  Dockerfile
 - ▼  webapp-backend
 - ▼  code
 -  webapp-backend-api.py
 -  Dockerfile
 - ▼  webapp-frontend
 -  code
 -  Dockerfile
 -  docker-compose.yml
 - >  VirtualTachograph

El código se estructura en dos carpetas: *IoTCloudServices* su código se ejecutará en la máquina *fic-cloud-services*) y *VirtualTachograph* (su código se ejecutará en la máquina *fic-devices*)

- En *IoTCloudServices* habrá cuatro carpetas:
 - *mosquitto* contendrá el Dockerfile y los ficheros de configuración necesarios para el despliegue de esta solución MQTT de código libre en un contenedor.
 - *message_router* contendrá el código fuente en Python para la implementación de este servicio (este código se colocará en la subcarpeta *code* donde también existirá un fichero *requirements.txt* con las dependencias necesarias para el funcionamiento del código desarrollado). Además, esta carpeta tiene el fichero Dockerfile para el despliegue mediante un contenedor del componente Message Router.
 - *dbService* contendrá el Dockerfile para la creación del servicio de base de datos que se utilizará para almacenar los datos operativos de la red de vehículos IoT. También contendrá un fichero con el script para la creación de la BD con el modelo definido para este caso práctico.
 - *microservices* contendrá el código de los microservicios que se desarrollarán en el ejercicio. En el ámbito de este ejercicio, los microservicios a desarrollar son:
 - *devices_microservice*. Esta carpeta contiene los elementos de este microservicio está estructurada de la siguiente manera: la carpeta *code* tiene los ficheros de código fuente que se utilizarán para la implementación del microservicio (*devices_manager_api.py* y *devices_db_manager.py*). Asimismo, en la carpeta raíz del microservicio se incluye el *Dockerfile* que define la imagen a partir de la cual se creará el contenedor para el despliegue del microservicio.
 - *sessions_microservice*. Esta carpeta contiene los elementos de este microservicio está estructurada de la siguiente manera: la carpeta *code* tiene los ficheros de código fuente que se utilizarán para la implementación del microservicio (*sessions_manager_api.py* y *sessions_db_manager.py*). Asimismo, en la carpeta raíz del microservicio se incluye el *Dockerfile* que define la imagen a partir de la cual se creará el contenedor para el despliegue del microservicio.
 - *telemetry_microservice*. Esta carpeta contiene los elementos de este microservicio está estructurada de la siguiente manera: la carpeta *code* tiene los ficheros de código fuente que se utilizarán para la implementación del microservicio (*telemetry_manager_api.py* y *telemetry_db_manager.py*). Asimismo, en la carpeta raíz del microservicio se incluye el *Dockerfile* que define la imagen a partir de la cual se creará el contenedor para el despliegue del microservicio.

- *events_microservice*. Esta carpeta contiene los elementos de este microservicio está estructurada de la siguiente manera: la carpeta *code* tiene los ficheros de código fuente que se utilizarán para la implementación del microservicio (*events_manager_api.py* y *events_db_manager.py*). Asimismo, en la carpeta raíz del microservicio se incluye el *Dockerfile* que define la imagen a partir de la cual se creará el contenedor para el despliegue del microservicio.
- *webapp-backend* contendrá el código de la interfaz API REST que se publicará para que las diferentes aplicaciones de usuario puedan integrarse con los servicios de la plataforma IoT. Esta carpeta estará estructurada de la siguiente manera: la carpeta *code* contendrá los ficheros de código fuente que se utilizarán para la implementación del backend. Asimismo, en la carpeta raíz del microservicio se incluye el *Dockerfile* que define la imagen a partir de la cual se creará el contenedor para su despliegue.
- *webapp-frontend* contendrá el código de la aplicación web que se desarrollará para interactuar con los servicios de la plataforma IoT desarrollada. Esta carpeta estará estructurada de la siguiente manera: la carpeta *code* contendrá los ficheros en javascript, html y css que son necesarios para la aplicación diseñada. Asimismo, en la carpeta raíz del microservicio se incluye el *Dockerfile* que define la imagen a partir de la cual se creará el contenedor para su despliegue.

Por último, en la carpeta *IoTCloudServices* se incluirá el fichero *docker-compose.yml* con la orquestación de los contenedores que se despliegan en la máquina *fic-cloud-services*.

- En *VirtualTachographs* habrá una carpeta *VehicleDigitalTwin* y contendrá el código fuente en Python para la implementación de este servicio (este código se colocará en la subcarpeta *code* donde también existirá un fichero *requirements.txt* con las dependencias necesarias para el funcionamiento del código desarrollado). Además, esta carpeta tiene el fichero *Dockerfile* para el despliegue mediante un contenedor de cada gemelo digital.

Por último, en la carpeta *VirtualTachographs* se incluirá el fichero *docker-compose.yml* con la orquestación de todos los gemelos digitales a contemplar en la máquina *fic-devices*.

Preparación del proyecto

En primer lugar, es necesario clonar el proyecto en gitlab (<https://teaching.sel.inf.uc3m.es>) correspondiente a la sesión 11.

VISUALIZACIÓN DE DATOS IoT

Se recomienda que, en este momento, se copien todos los ficheros de código y configuración desarrollados en la sesión 10 a la carpeta en la que se está desarrollando el código de la sesión 11, contemplando la estructura de carpetas presentada en el apartado anterior.

Desarrollo del backend

En la carpeta *IoTCloudServices/webapp-backend/code* se tienen que crear un fichero con el código fuente que será necesario desarrollar: *webapp-backend-api.py*.

Desarrollo de la interfaz API REST con Flask

La interfaz API REST que se implementará con Flask para el front-end de la arquitectura se incluirá en el fichero *webapp-backend-api.py*.

Los requisitos que se tienen que satisfacer para la implementación de esta interfaz son los siguientes:

1. Se tienen que importar todas las librerías correspondientes al framework de Flask y sus extensiones como con los microservicios desarrollados en el ejercicio 10.
2. Se tienen que crear el esqueleto de la aplicación Flask de acuerdo con el esquema presentado en el ejercicio 10. Sin embargo, en este caso, es necesario configurar la URL y el puerto por el cual se publicará la API REST. Las variables de entorno HOST y PORT nos indicarán cuál es la dirección y puerto en el que se publicará la API REST. HOST debe tomar el valor 0.0.0.0 y PORT debe tomar el valor 5005.
3. La API REST tendrá los siguientes métodos:
 - `@app.route('/tachographs/active/', methods=['GET'])`

Este método no recibe ningún parámetro como entrada.

Su comportamiento consiste en reenviar la petición al microservicio de telemetría para obtener la información de la última posición de los tacógrafos activos.

```
host = os.getenv('TELEMETRY_MICROSERVICE_ADDRESS')
port = os.getenv('TELEMETRY_MICROSERVICE_PORT')
result = requests.get('http://' + host + ':' + port +
    '/telemetry/positions/')
if result.status_code == 201:
    return result.json(), 201
else:
```

```
return {"result": "Error: Tachographs information is not available"}, 500
```

- @app.route('/tachographs/telemetry/', methods=['GET'])

Este método recibe como parámetro de entrada los datos del tacógrafo a consultar. Se consultarán las telemetrías recibidas en el último minuto.

```
http://<server_address>:5005/tachographs/telemetry/?tachograph_id=tachograph_control_unit-2
```

Su comportamiento consiste en reenviar la petición al microservicio de telemetría para obtener la información de las telemetrías de reportadas en el intervalo de tiempo establecido.

La información que se mostrará de las telemetrías consiste

```
port = os.getenv('TELEMETRY_MICROSERVICE_PORT')
host = os.getenv('TELEMETRY_MICROSERVICE_ADDRESS')
result = requests.get('http://' + host + ':' + port +
'/telemetry/', json=params)
```

- @app.route('/tachographs/events/', methods=['GET'])

Este método recibe como parámetro de entrada los datos del tacógrafo a consultar. Se consultarán los eventos recibidos en el último minuto.

```
http://<server_address>:5005/tachographs/events/?tachograph_id=tachograph_control_unit-2
```

Su comportamiento consiste en reenviar la petición al microservicio de eventos para obtener la información de los eventos producidos en el intervalo de tiempo establecido.

```
params = request.get_json()
host = os.getenv('EVENTS_MICROSERVICE_ADDRESS')
port = os.getenv('EVENTS_MICROSERVICE_PORT')
result = requests.get('http://' + host + ':' + port +
'/tachographs/params/', json=params)
```

Este método devuelve como resultado la lista de eventos proporcionada por el microservicio relacionado.

- @app.route('/tachographs/configuration/', methods=['GET'])

Este método recibe como parámetro de entrada los datos del tacógrafo a consultar.

```
http://<server_address>:5005/tachographs/configuration/?tachograph_id=tachograph_control_unit-2
```

Su comportamiento consiste en reenviar la petición al microservicio de dispositivos para obtener, en formato JSON, la configuración actual del tacógrafo.

```
host = os.getenv('DEVICES_MICROSERVICE_ADDRESS')
port = os.getenv('DEVICES_MICROSERVICE_PORT')
result = requests.get('http://' + host + ':' + port +
'/tachographs/params/', json=params)
```

Este método devuelve como resultado la información proporcionada por el microservicio de dispositivos con respecto a la configuración del tacógrafo solicitado.

- `@app.route('/tachographs/configuration/', methods=['POST'])`

Este método recibe como parámetro de entrada los datos de los nuevos valores a asignar a la configuración del tacógrafo considerado.

```
http://<server>:5005/tachographs/configuration/?tachograph_id=tachograph_control_unit-
2?rate=2?sampling=2
```

Su comportamiento consiste en reenviar la petición al microservicio de dispositivos para la actualización de la configuración del tacógrafo.

```
my_tachograph = request.args.get('tachograph_id')
my_sensor_sampling = request.args.get('sampling')
my_telemetry_rate = request.args.get('rate')
params = {"tachograph_id": my_tachograph,
"sensors_sampling_rate":my_sensor_sampling,
"telemetry_rate":my_telemetry_rate}
host = os.getenv('DEVICES_MICROSERVICE_ADDRESS')
port = os.getenv('DEVICES_MICROSERVICE_PORT')
r = requests.post('http://' + host + ':' + port +
'/tachographs/params/', json=params)
```

En caso de que todo haya ido bien, este método devolverá un mensaje informativo, junto con el código de éxito 201:

```
return {"result": "Success: Tachograph params updated"}, 201
```

En caso contrario, se debe proporcionar un mensaje de error y el código de error 500 de http:

```
return {"result": "Error: Tachograph params not updated"}, 500
```

Creación de la imagen con Dockerfile

Una vez que se ha probado que el código del webapp-backend funciona correctamente en el equipo de desarrollo, se procederá a la preparación para su despliegue con contenedores.

VISUALIZACIÓN DE DATOS IoT

Para ello, en la carpeta *IoTCloudServices/webapp-backend* se creará el fichero *Dockerfile*. El contenido de este fichero tendrá que permitir lo siguiente:

- Crear la imagen del contenedor a partir de la correspondiente a python:3.12
- Copiar el código que está en la carpeta *./code* a la carpeta */etc/usr/src/app*
- Establecer esta carpeta como directorio de trabajo.
- Instalar los paquetes necesarios especificados en *requirements.txt*. Los paquetes necesarios son: *requests*, *Flask* y *Flask-Cors*, para ello ejecutar `pip install <paquetes necesarios>`
- Ejecutar el código incluido en el fichero *webapp_backend_api.py*

Configuración de la orquestación del microservicio

Para ello, en la carpeta *IoTCloudServices* se modificará el fichero denominado *docker-compose.yml*. El nuevo contenido de este fichero debe incluir la definición de la orquestación del servicio *webapp_backend* de acuerdo con el siguiente código:

```
webapp_backend:
  build: ./webapp-backend
  ports:
    - '5005:5005'
  environment:
    - HOST=0.0.0.0
    - PORT=5005
    - TELEMETRY_MICROSERVICE_ADDRESS=telemetry_microservice
    - TELEMETRY_MICROSERVICE_PORT=5003
    - EVENTS_MICROSERVICE_ADDRESS=events_microservice
    - EVENTS_MICROSERVICE_PORT=5004
    - DEVICES_MICROSERVICE_ADDRESS=devices_microservice
    - DEVICES_MICROSERVICE_PORT=5001
  volumes:
    - "./webapp-backend/code:/etc/usr/src/app"
  depends_on:
    - devices_microservice
    - telemetry_microservice
    - events_microservice
```

Despliegue del microservicio

Para desplegar las versiones actualizadas del gemelo digital del simulador de vehículos en un contenedor Docker, se recomienda utilizar Docker Compose porque permitirá orquestar este servicio con los otros que se van a ir desplegando en la máquina virtual *fic-services*.

Posteriormente, es necesario conectarse por *ssh* a la máquina *fic-services*. A continuación, se debe clonar el repositorio correspondiente al código de esta sesión en esta máquina virtual.

VISUALIZACIÓN DE DATOS IoT

Una vez que se haya obtenido en la máquina virtual el código correspondiente a esta sesión, hay que desplazarse a la carpeta *IoTCloudServices* y lanzar los servicios de esta máquina ejecutando los comandos:

```
docker compose build  
  
docker compose up -d
```

Para verificar el despliegue, se pueden ejecutar los comandos siguientes:

```
docker ps -a  
  
docker logs <nombre del contenedor>
```

Prueba de la API REST del Webapp-Backend

Una vez que se haya finalizado el desarrollo, se debe probar el correcto funcionamiento del código desarrollado.

Las pruebas se desarrollarán con Postman de la misma manera que se han probado los microservicios en los ejercicios anteriores.

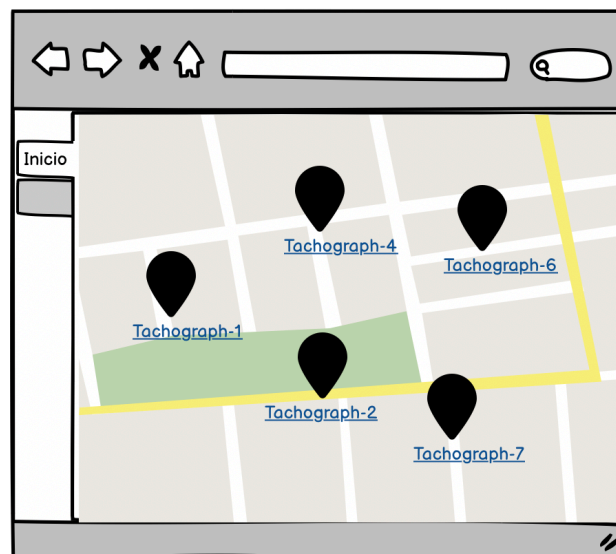
Cuando finalice la verificación de que el código funciona como se espera, el código desarrollado se debe publicar en el repositorio de control de versiones.

Desarrollo del frontend HTML y Javascript

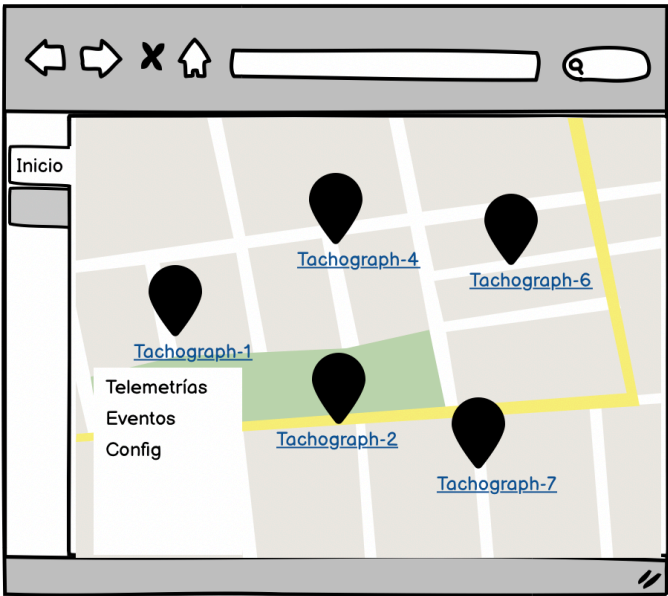
El front-end, cuyo desarrollo se propone a los estudiantes, mostrará un mapa con las ubicaciones de los tacógrafos activos en la flota de rutas. Cuando se pinche sobre el marcador que representa a un tacógrafo, se obtendrá su información detallada que consistirá en la información de las últimas telemetrías y los últimos proporcionados por el tacógrafo.

Mockup de la interfaz web

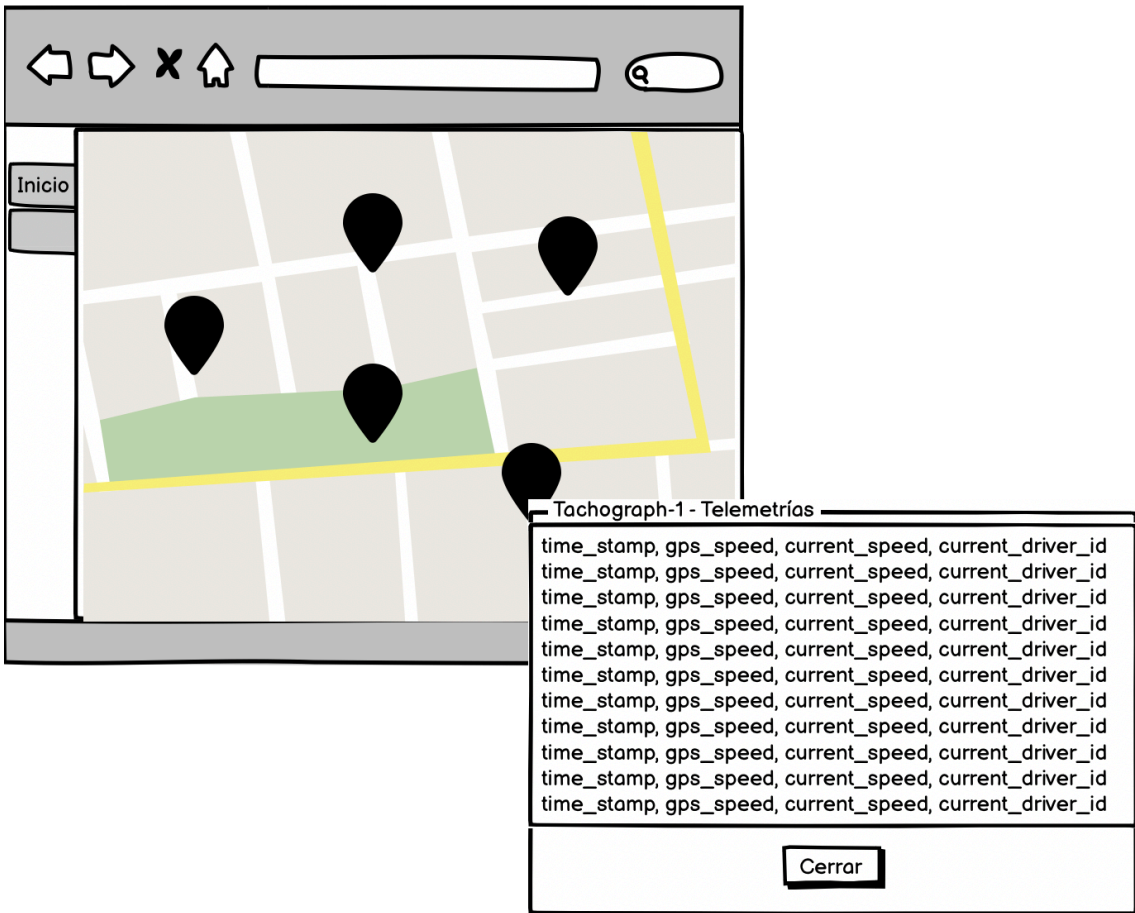
El mockup correspondiente a la interfaz web que se pretende desarrollar es la siguiente:



En la pantalla de inicio, se mostrará un mapa con las ubicaciones de los vehículos activos en la flota de rutas.



Si se pulsa en un tacógrafo, se mostrará un menú contextual para acceder a las distintas opciones disponibles. Estas opciones, son: Telemetrías, Eventos y Configuración.



VISUALIZACIÓN DE DATOS IoT

Al pulsar sobre la opción de Telemetrías, en una ventana emergente, se mostrarán las telemetrías recibidas en el último minuto (podrían configurarse varios minutos para fácil la prueba).



Al pulsar sobre la opción de Eventos, en una ventana emergente, se mostrarán los eventos recibidos en los últimos 5 minutos.

Al pulsar sobre la opción de Eventos, en una ventana emergente, se mostrarán los parámetros de configuración del tacógrafo.

VISUALIZACIÓN DE DATOS IoT

Posteriormente, si se desea actualizar la configuración del tacógrafo, se podrán actualizar los valores de la tasa de envío de telemetría y/o de muestreo de sensores. La orden de modificación de la configuración del tacógrafo se emitirá al pulsar el botón asignar.

Finalmente, pulsando el botón volver, se mostrará la página de inicio.

Desarrollo de la página de inicio

Los elementos que desarrollar para crear la página de inicio consisten en: una página html (*index.html*), un fichero de estilos css (*style.css*) y el código en javascript para gestionar el funcionamiento de la página (*index.js*).

Estos ficheros se deben incluir en la carpeta *IoTCloudServices/webapp-frontend/code*.

1. Los elementos que se deben incluir en la página de inicio *index.html* son los siguientes:
 - En la sección de cabecera se debe incluir:

Referencia a las librerías de javascript necesarias de soporte

```
<script
src="https://polyfill.io/v3/polyfill.min.js?features=default"></
script>
<script src="https://code.jquery.com/jquery-
3.6.0.min.js"></script>
```

Referencia a la guía de estilos

```
<link rel="stylesheet" type="text/css" href="./style.css" />
```

Referencia al código que se desarrollará para gestionar el funcionamiento de la página de inicio.

```
<script type="module" src="./index.js"></script>
```

- En el cuerpo de la página:

Se debe incluir un contenedor para representar el mapa:

```
<div id="map"></div>
```

En este último código, se debe colocar la API key para el acceso a la API de Google Maps y que se obtuvo durante el ejercicio 7 de este cuaderno.

```
<script>(g=>{var h,a,k,p="The Google Maps JavaScript
API",c="google",l="importLibrary",q="__ib__",m=document,b=window
;b=b[c]||(b[c]={});var d=b.maps||(b.maps={}),r=new Set,e=new
URLSearchParams,u=(()=>h||(h=new Promise(async(f,n)=>{await
(a=m.createElement("script"));e.set("libraries",[...r]+"");for(k
in g)e.set(k.replace(/[A-
Z]/g,t=>"_"+t[0].toLowerCase()),g[k]);e.set("callback",c+".maps.
"+q);a.src=`https://maps.${c}apis.com/maps/api/js?`+e;d[q]=f;a.o
nerror=(()=>h=n(Error(p+" could not
load."));a.nonce=m.querySelector("script[nonce]")?.nonce||"";m.h
ead.append(a)}));d[l]?console.warn(p+" only loads once.
Ignoring:",g):d[l]=(f,...n)=>r.add(f)&&u().then(()=>d[l](f,...n)
)}))
({key: "<API KEY>", v: "weekly"});</script>
```

2. Los elementos que se deben incluir en la hoja de estilos (*style.css*) se incluyen en el fichero que acompaña a este enunciado.
3. Los elementos que se deben programar en javascript (*index.js*) para determinar el funcionamiento esperado es:
 - Para la inicialización del mapa se debe incluir el código de inicio:

```
let map;

var server_address = "http://<dirección_ip>:5005/"

window.initMap = initMap;
setInterval(initMap,60000)
```

El código de la función `initMap`, creará un nuevo mapa centrado en Madrid, realizará la llamada a la API REST para obtener la ubicación de todos los vehículos que están activos.

```
async function initMap(callback) {
  // const { Map } = await google.maps.importLibrary("maps");
  const position = { lat: 40.33256, lng: -3.76516 };
}
```

```

const { Map } = await google.maps.importLibrary("maps");
const { AdvancedMarkerElement, PinElement } = await
google.maps.importLibrary("marker");

const map = new Map(document.getElementById("map"), {
  center: position,
  zoom: 12,
  mapId: "DEMO_MAP_ID", //Generar un map id
});

const infoWindow = new google.maps.InfoWindow();

const address = server_address + "tachographs/active/"
$.getJSON(address, function(result){

```

Para dibujar el marcador de cada vehículo se debe crear una un marcador avanzado por cada uno de los tacógrafos recuperados

```

$.getJSON(address, function(result){
  console.log(result);
  $.each(result, function(index, item){
    const m_position = { lat: item.Latitude, lng:
item.Longitude };
    const pinTextGlyph = new PinElement({
      glyph: "T",
      glyphColor: "white",
    });
    const marker = new AdvancedMarkerElement({
      map: map,
      position: m_position,
      content: pinTextGlyph.element,
      title: item.Tachograph_id,
      gmpClickable: true,
    });
  });

```

- Para la visualización del menú contextual que permita acceder a los valores de telemetrías, eventos y configuración es similar al siguiente:

```

// Add a click listener for each marker, and set up the info
window.
const contentString =
'<div id="content">' +
'<h3 id="firstHeading" class="firstHeading">' + marker.title +
'</h3>' +
'<div id="bodyContent">' +
'<p> <a href="./telemetry.html?tachograph=' + marker.title +
'">Telemetría</a></p>' +
'<p> <a href="./events.html?tachograph=' + marker.title +
'">Eventos</a></p>' +
'<p> <a href="./configuration.html?tachograph=' + marker.title +
'">Configuración</a></p>' +
'</div>' +
'</div>';
marker.addListener("click", ({domEvent, latLng}) => {
  const {target} = domEvent;
  infoWindow.close();
  infoWindow.setContent(contentString);
  infoWindow.open(marker.map, marker);

```

Desarrollo del formulario para la actualización de las configuraciones del tacógrafo

Los elementos que desarrollar para crear la página de inicio consisten en: una página html (*configuration.html*), un fichero de estilos css (*style.css*) y el código en javascript para gestionar el funcionamiento de la página (*configuration.js*).

Estos ficheros se deben incluir en la carpeta *IoTCloudServices/webapp-frontend/code*.

1. Los elementos que se deben incluir en la página *routes.html* son los siguientes:

- En la sección de cabecera se debe incluir:

Referencia a las librerías que se utilizarán para el procesamiento de la información procedente del webapp backend.

```
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
```

Referencia a la guía de estilos

```
<link rel="stylesheet" type="text/css" href="./style.css" />
```

Referencia al código que se desarrollará para gestionar el funcionamiento de la página de inicio.

```
<script type="module" src="./routes.js"></script>
```

- En el cuerpo de la página:

```
<div id="configuration_viewer">
  <label>Configuración del vehículo:</label>
</div>
<div id="configuration_assigner">
  <label for="samplingFrequencyInputText">Nueva Frecuencia de
Muestreo:</label><br>
  <input type="text" id="samplingFrequencyInputText"
name="samplingFrequencyInputText" value="2"><br>
  <label for="sendingFrequencyInputText">Nueva Frecuencia de
Envío:</label><br>
  <input type="text" id="sendingFrequencyInputText"
name="sendingFrequencyInputText" value="2"><br><br>
  <button type="button"
onclick=update_configuration()>Enviar</button>
</div>
```

2. Los elementos que se deben programar en javascript (*configuration.js*) para determinar el funcionamiento esperado es:

- Para la inicialización del selector de matrículas activas se debe incluir el código de inicio:

```
window.onload = search_tachograph_configuration;
```

- Para la búsqueda y visualización de la configuración del tacógrafo:

Recoger los parámetros del tacógrafo a consultar.

```
const queryString = window.location.search;
const urlParams = new URLSearchParams(queryString);
const tachograph = urlParams.get('tachograph')
const body = {
  tachograph_id: tachograph
};
```

Configurar la cabecera de la tabla para presentar los parámetros de configuración del tacógrafo.

```
let container = document.getElementById("configuration_viewer");
// Create the table element
let table = document.createElement("table");
let thead = document.createElement("thead");
let tr = document.createElement("tr");
let th1 = document.createElement("th");
th1.innerText = "Frecuencia de Muestreo"; // Set the column name
as the text of the header cell
tr.appendChild(th1); // Append the header cell to the header row

let th2 = document.createElement("th");
th2.innerText = "Frecuencia de Envío"; // Set the column name as
the text of the header cell
tr.appendChild(th2); // Append the header cell to the header row

thead.appendChild(tr); // Append the header row to the header
table.appendChild(tr) // Append the header to the table
```

Consultar los parámetros de configuración del tacógrafo y visualizarlos en la página web.

```
$.getJSON( address, body, function( result ) {
  var object = JSON.parse(result)
  let tr = document.createElement("tr");
  let td1 = document.createElement("td");
  td1.innerText = result['sensors_sampling_rate']; // Set
the value as the text of the table cell
  tr.appendChild(td1); // Append the table cell to the
table row
  let td2 = document.createElement("td");
  td2.innerText = result['telemetry_rate']; // Set the
value as the text of the table cell
  tr.appendChild(td2); // Append the table cell to the
table row
  table.appendChild(tr); // Append the table row to the table
});
table.setAttribute("border", "1");

container.appendChild(table) // Append the table to the
container element
```

- Para la modificación de la configuración del tacógrafo:

Obtener los valores de la página web.

```
const samplingFrequencyField =  
document.getElementById("samplingFrequencyInputText");  
const samplingFrequency = samplingFrequencyField.value;  
const sendingFrequencyField =  
document.getElementById("sendingFrequencyInputText");  
const sendingFrequency = sendingFrequencyField.value;
```

Realizar la petición de actualización mediante la webapp backend.

```
const body = {  
  tachograph_id: tachograph,  
  sampling: samplingFrequency,  
  rate: sendingFrequency  
};  
$.post(server_address+"tachographs/configuration/", body, (data,  
status) => {  
  console.log(data);  
});
```

Desarrollo de la página web para visualizar los datos de los eventos

Los elementos que desarrollar para crear la página de inicio consisten en: una página html (*events.html*), un fichero de estilos css (*style.css*) y el código en javascript para gestionar el funcionamiento de la página (*events.js*).

- 1) La estructura de la página web es la siguiente:

```
<div id="events_list">  
</div>  
<div id="events_list_controls">  
  <input type="button" onclick="location.href='./index.html';"  
value="Volver" />  
</div>
```

La cabecera de la página web será similar a la establecida para el fichero *configuration.html*.

- 2) El código a implementar en el módulo Javascript deberá tener las siguientes funcionalidades:
 - Recoger los parámetros del tacógrafo a consultar.
 - Configurar la cabecera de la tabla para presentar los eventos.
 - Consultar los eventos del tacógrafo y visualizarlos en la página web.

Estas funcionalidades se implementarán de manera similar a las establecidas en la página web de consulta de la configuración del tacógrafo.

Desarrollo de la página web para visualizar los datos de las telemetrías

Los elementos que desarrollar para crear la página de inicio consisten en: una página html (*telemetry.html*), un fichero de estilos css (*style.css*) y el código en javascript para gestionar el funcionamiento de la página (*telemetry.js*).

3) La estructura de la página web es la siguiente:

```
<div id="telemetry_list">
</div>
<div id="telemetry_list_controls">
  <input type="button" onclick="location.href='./index.html';"
value="Volver" />
</div>
```

La cabecera de la página web será similar a la establecida para el fichero *configuration.html*.

4) El código a implementar en el módulo Javascript deberá tener las siguientes funcionalidades:

- Recoger los parámetros del tacógrafo a consultar.
- Configurar la cabecera de la tabla para presentar las telemetrías.
- Consultar las telemetrías del tacógrafo y visualizarlas en la página web.

Estas funcionalidades se implementarán de manera similar a las establecidas en la página web de consulta de la configuración del tacógrafo.

Creación de la imagen con Dockerfile

A continuación, se procederá a la preparación para su despliegue con contenedores.

Para ello, en la carpeta *IoTCloudServices/webapp-frontend* se creará el fichero Dockerfile. El contenido de este fichero tendrá que permitir lo siguiente:

- Crear la imagen del contenedor a partir de la correspondiente a [httpd:2.4](#)
- Copiar el código que está en la carpeta *./code* a la carpeta */usr/local/apache2/htdocs/*

Orquestación de los servicios con Docker Compose

Para ello, en la carpeta *IoTCloudServices* se modificará el fichero denominado *docker-compose.yml*. El nuevo contenido de este fichero debe incluir la definición de la orquestación de la interfaz web de acuerdo con el siguiente código:

VISUALIZACIÓN DE DATOS IoT

```
webapp_frontend:  
  build: ./webapp-frontend  
  ports:  
    - '80:80'
```

Despliegue de la imagen con Docker Compose

Para desplegar el componente `message_router` es necesario conectarse por *ssh* a la máquina *fic-services* creada anteriormente.

A continuación, se debe obtener la última versión del código de esta sesión que se encuentra en el repositorio en esta máquina virtual mediante un comando *git pull*.

Una vez que se haya obtenido en la máquina virtual el código correspondiente a esta sesión, hay que desplazarse a la carpeta *IoTCloudServices* y lanzar los servicios de esta máquina ejecutando los comandos:

```
docker compose stop  
  
docker compose build  
  
docker compose up -d
```

Para verificar que WebApp Frontend se encuentra en ejecución, se pueden ejecutar los comandos siguientes:

```
docker ps -a  
  
docker logs <nombre del contenedor del WebApp Frontend>
```


Criterios de evaluación y procedimiento de entrega



Criterios de Evaluación

- Webapp Backend – Consulta de vehículos activos – 1 punto
- Webapp Frontend – Visualización de vehículos en mapas – 2 puntos
- Webapp Backend – Consulta de telemetrías – 1 punto
- Webapp Frontend – Visualización de telemetrías – 1 punto
- Webapp Backend – Consulta de eventos – 1 punto
- Webapp Frontend – Visualización de eventos – 1 punto
- Webapp Backend – Consulta de configuraciones de vehículos – 1 punto
- Webapp Frontend – Consulta y asignación de configuraciones – 2 puntos



Entrega de la solución del ejercicio guiado

La entrega se realizará de dos maneras:

- 1) **Código Fuente.** En el repositorio de código de la asignatura tendrá en el proyecto Sesión11 correspondiente al grupo de prácticas los ficheros de código con la organización en directorios y nomenclatura de los ficheros que se han indicado a lo largo de este guion.
- 2) **Video demostrativo.** En una tarea Kaltura Capture Video habilitada para este ejercicio se entregará un vídeo que demuestre el correcto funcionamiento de la solución elaborada.

Para este programa, se debe proporcionar una breve explicación del código generado (código en Python, dockerfile y docker compose) para el Backend y las funcionalidades implementadas del Frontend mostrar el lanzamiento de la ejecución de la solución en la GCP y la demostración de las funcionalidades desarrolladas para el frontend.

La fecha límite para la entrega del ejercicio es 08/05/2025 a las 23:59 h.

VISUALIZACIÓN DE DATOS IoT

De acuerdo con las normas de evaluación continua en esta asignatura, si un estudiante no envía la solución de un ejercicio antes de la fecha límite, el ejercicio será evaluado con 0 puntos.



Sugerencias

Cada estudiante debe guardar una copia de la solución entregada hasta la publicación de las calificaciones finales de la asignatura.