

# Vehículos conectados a la nube con IoT

Luis Coco Enríquez



**Luxoft**  
A DXC Technology Company



# Luxoft: coches definidos por software ya están aquí!

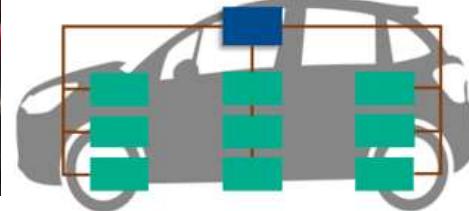


[Luxoft's remote repair of software-defined vehicles](#)

# Cuadro de mando y arquitectura tradicional de un coche



Distributed E/E Architecture



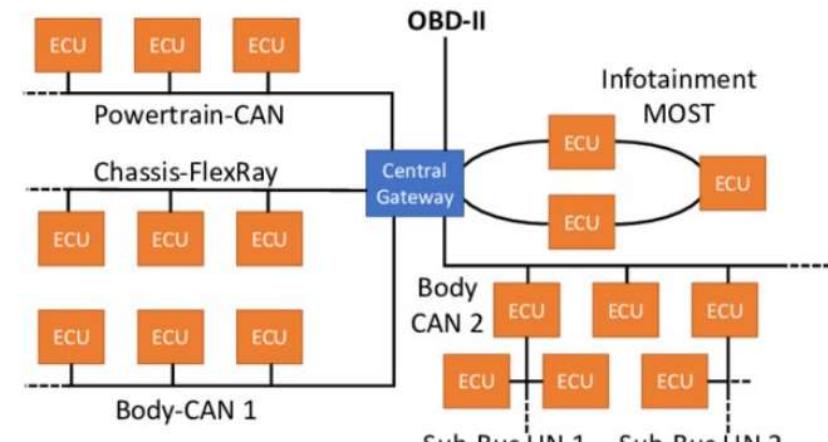
Optional ECUs e.g., central gateway



Automotive ECUs (function-specific)



OBD2  
On-Board Diagnostic II



**ECU** Electronic Control Units conectadas mediante el protocolo CAN Bus

# Cuadros de mando digitales

Los **coches de última generación** disponen de: control de **navegación**, “**infotainment**” (Sistema de entretenimiento), **conducción autónoma** y **sistemas seguridad avanzados**, **controles digitales** del vehículo (aire acondicionado, consumo eléctrico, etc).

El ordenador de a bordo de los coches **Tesla** combina tres ordenadores con funcionalidades diferenciadas pero interconectados entre ellos: **Media unit**, **Telematic board**, **Full Self-Driving computer**.



[Mercedes EQE 350+ cockpit](#)



[2023 Tesla Model S Plaid video](#)



[Renault Austral 2023 Multimedia & Cockpit](#)



 [2023 BMW X1 Multimedia & Digital Cockpit](#) [Nissan X-TRAIL 2023 cockpit and interior](#) [AUDI Q6 e-tron 2024 digital cockpit and infotainment](#)



# Pantalla del ordenador de abordo de un Tesla Model 3

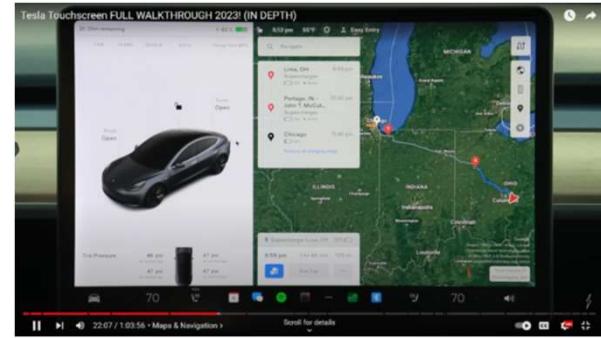
[Touchscreen | Model 3 and Model Y](#)

[Tesla Touchscreen FULL WALKTHROUGH 2023! \(IN DEPTH\)](#)

## Controles de los **sistemas** del vehículo

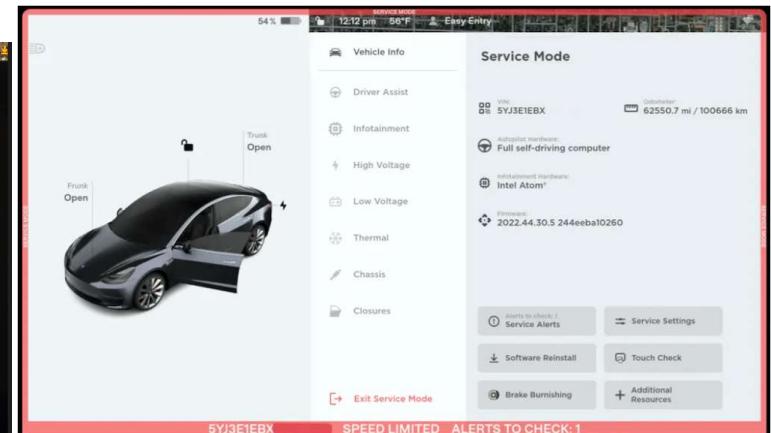
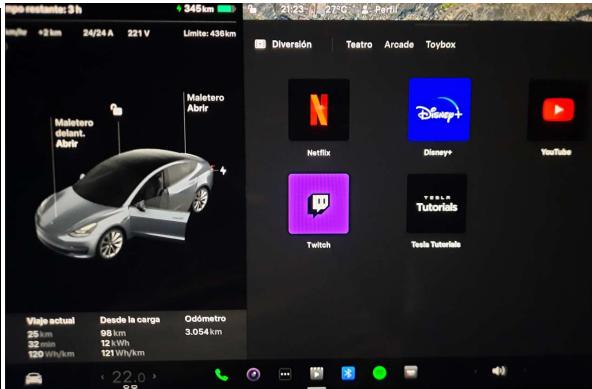
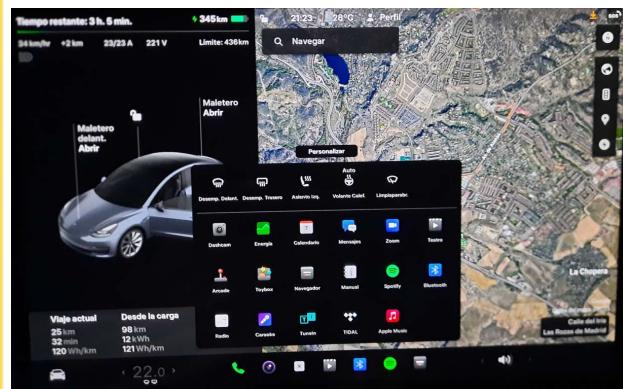


## Sistema de navegación con Google Maps

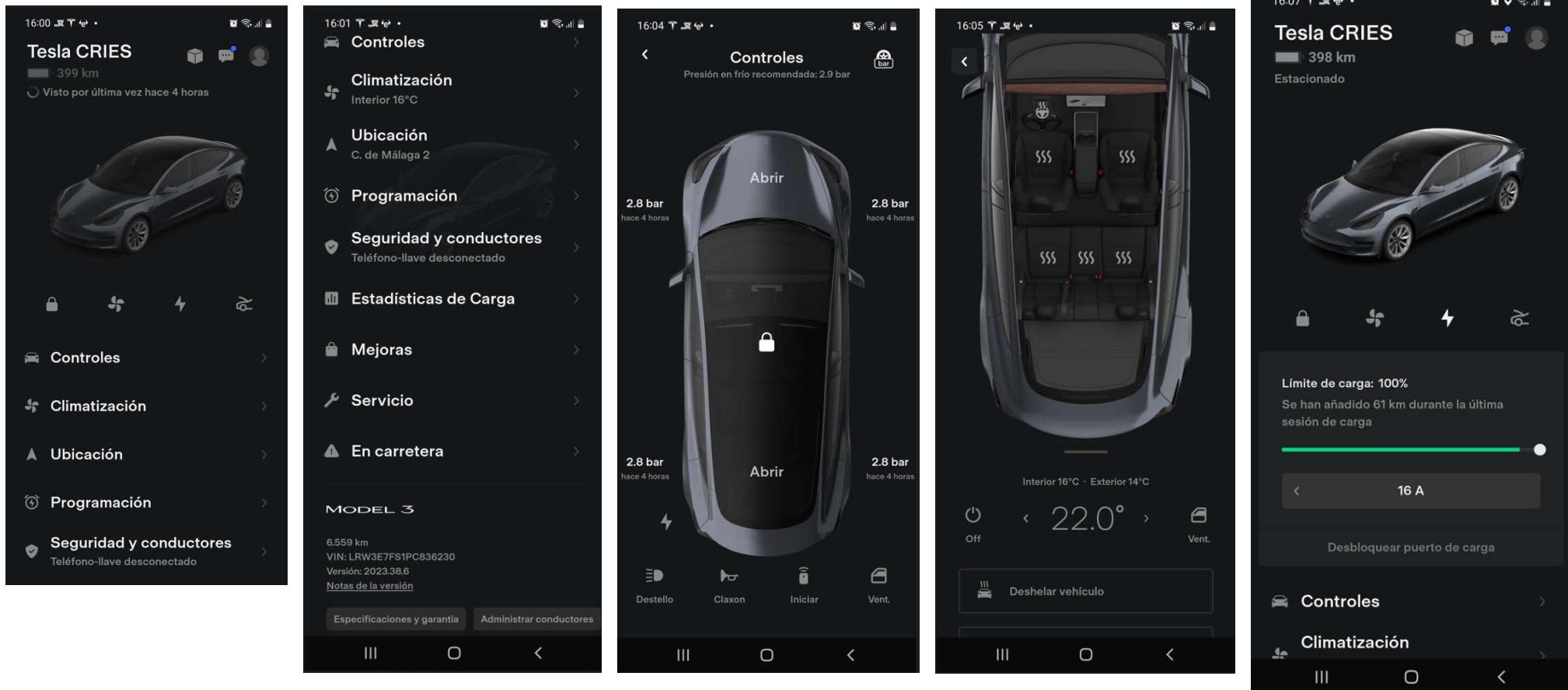


## Menú para diagnóstico y reparación

## Aplicaciones de infoentretenimiento



# Aplicaciones móvil para manejo remoto del vehículo



# ¿Qué es un vehículo conectado?

Transformación digital de los vehículos mediante ordenadores a bordo con conexión a internet.

Electronic Control Units **ECUs** vs **IVC** In-Vehicle-Computer y el **IVI** In-Vehicle-Infotainment, ambos con capacidad **IoT**, mediante tarjeta **SIM**.

**Protocolos de comunicación:** LTE, 4G, 5G, etc.

Comunicación del vehículo con la infraestructura, los peatones, con otros vehículos, con satélites, etc.

**V2I** Vehicle to Infrastructure

**V2N** Vehicle to Network

**V2P** Vehicle to Pedestrian

**V2V** Vehicle to Vehicle

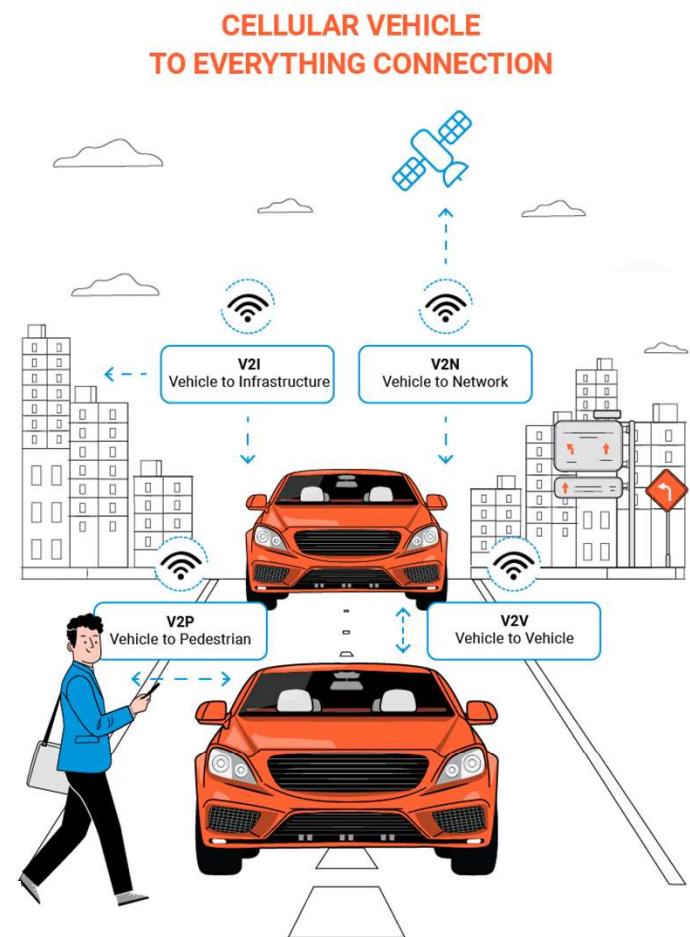
**V2G** Vehicle to Grid

**V2X** Vehicle connected to everithing

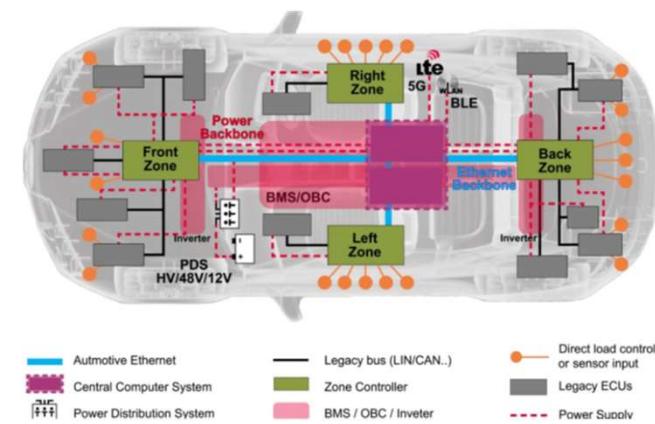
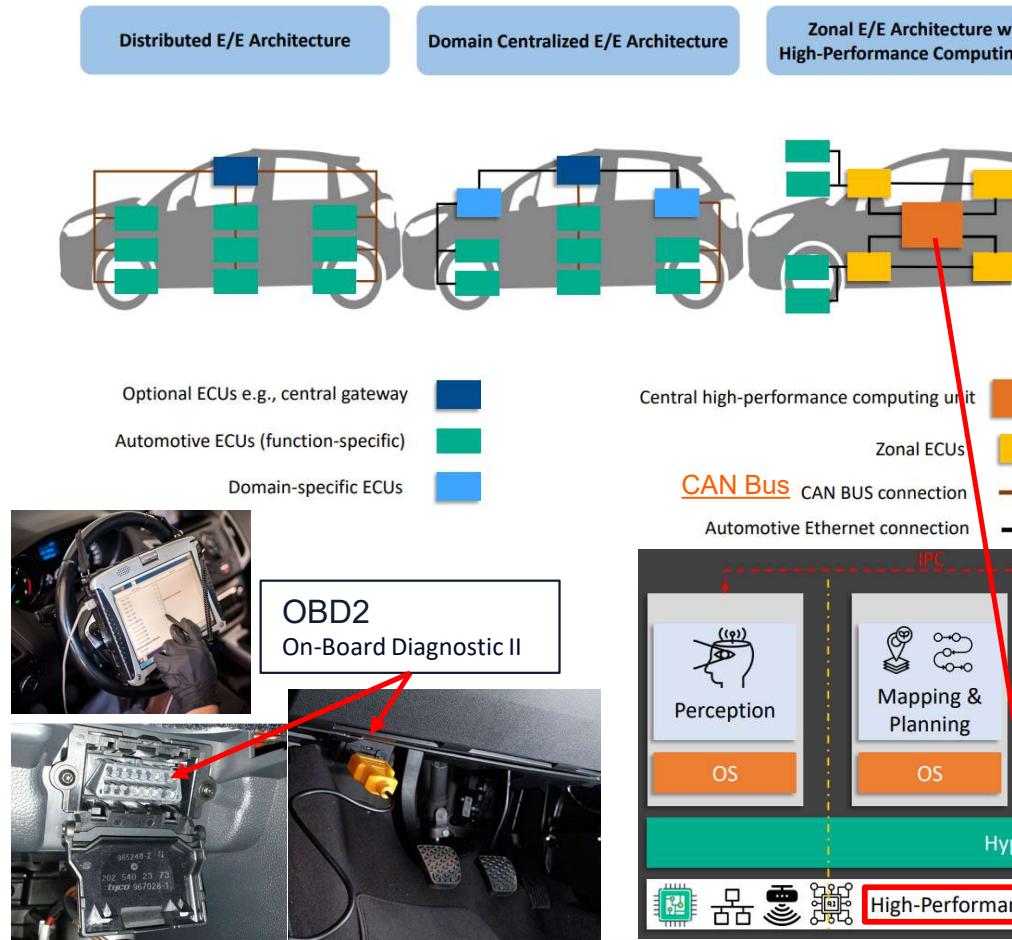
[Ericsson Connected Vehicle](#)

[Eviden - Connected Vehicle](#)

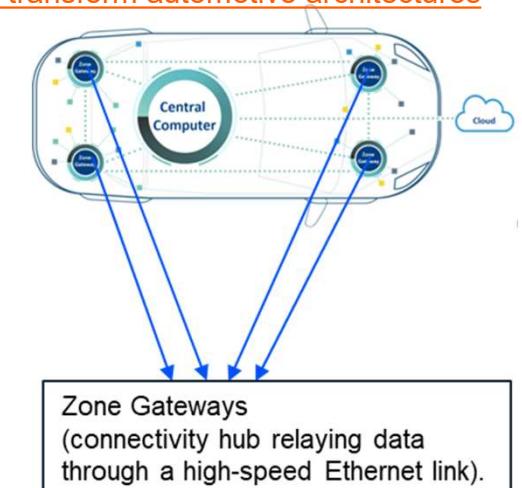
[Connected vehicles video](#)



# Evolución arquitectura de control de un coche



Zonal E/E will transform automotive architectures



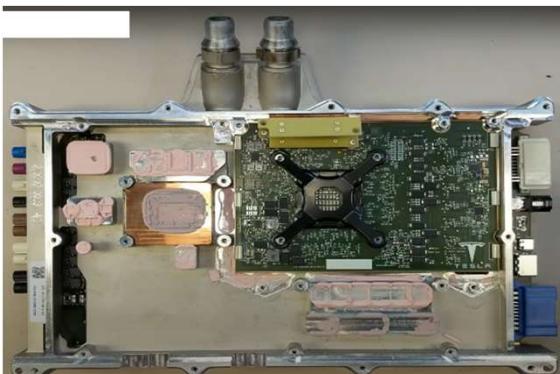
# Hardware de un vehículo conectado Tesla

[Tesla's new car on-board computer](#)

[Tesla's NEW HW4 Car Computer](#)

[Tesla Model S Plaid | ADAS Breakdown](#)

[Tesla's new self-driving \(HW4\) computer leaks: Here's a teardown](#)



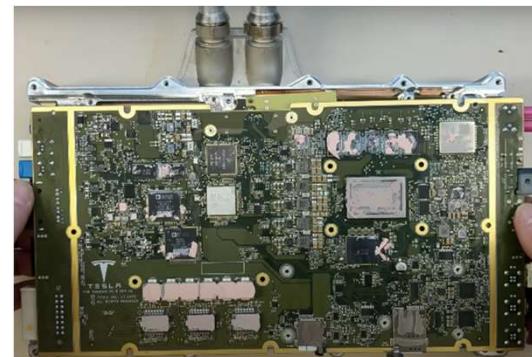
Graphic card



[Tesla Model 3 Y S X EU LTE  
3G Modem, Adapter,  
Connectivity, QUECTEL  
AG525R PCBA](#)



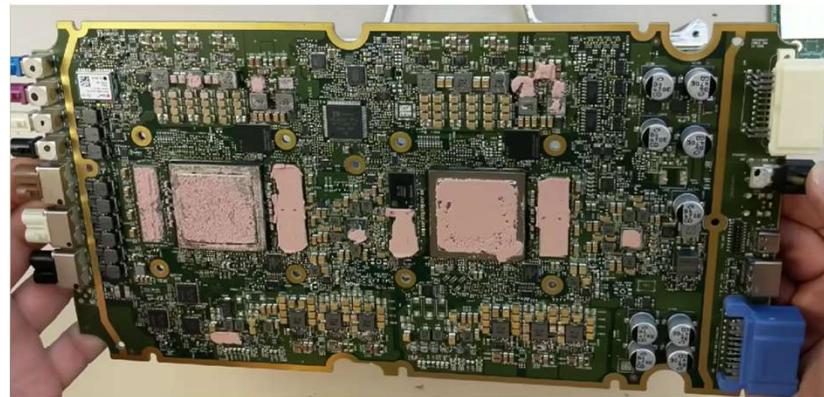
Computer back-side



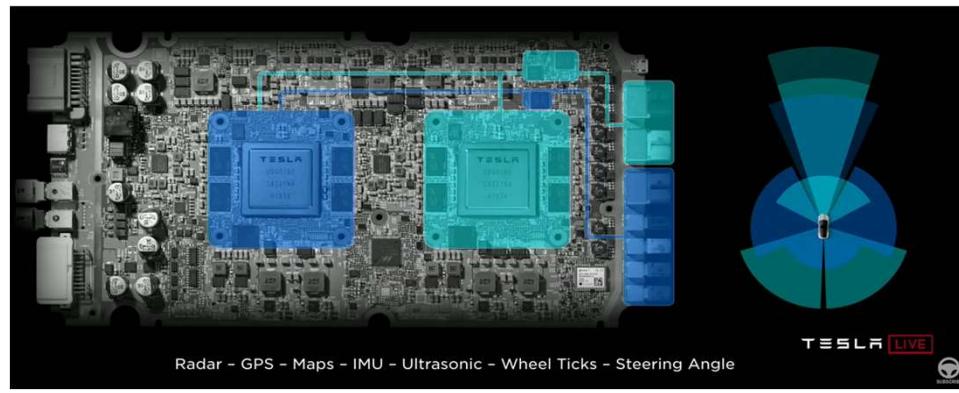
# Hardware de un vehículo conectado



AutoPilot Computer back-side

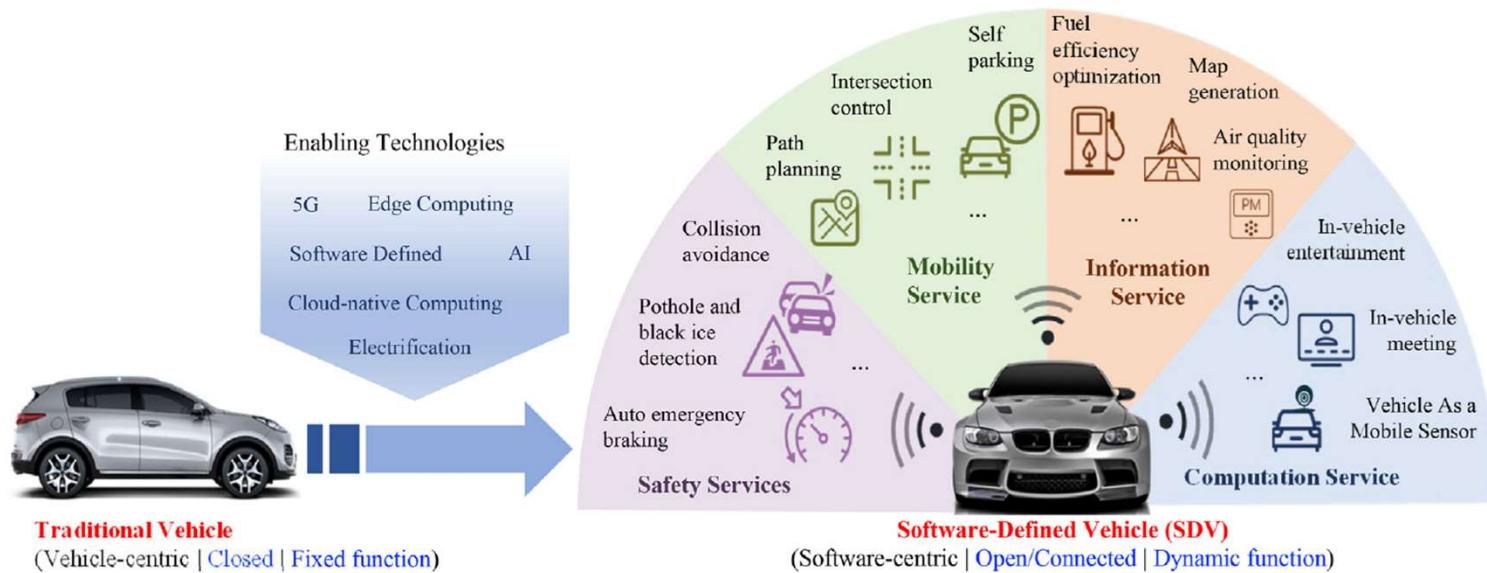


AutoPilot Computer front-side



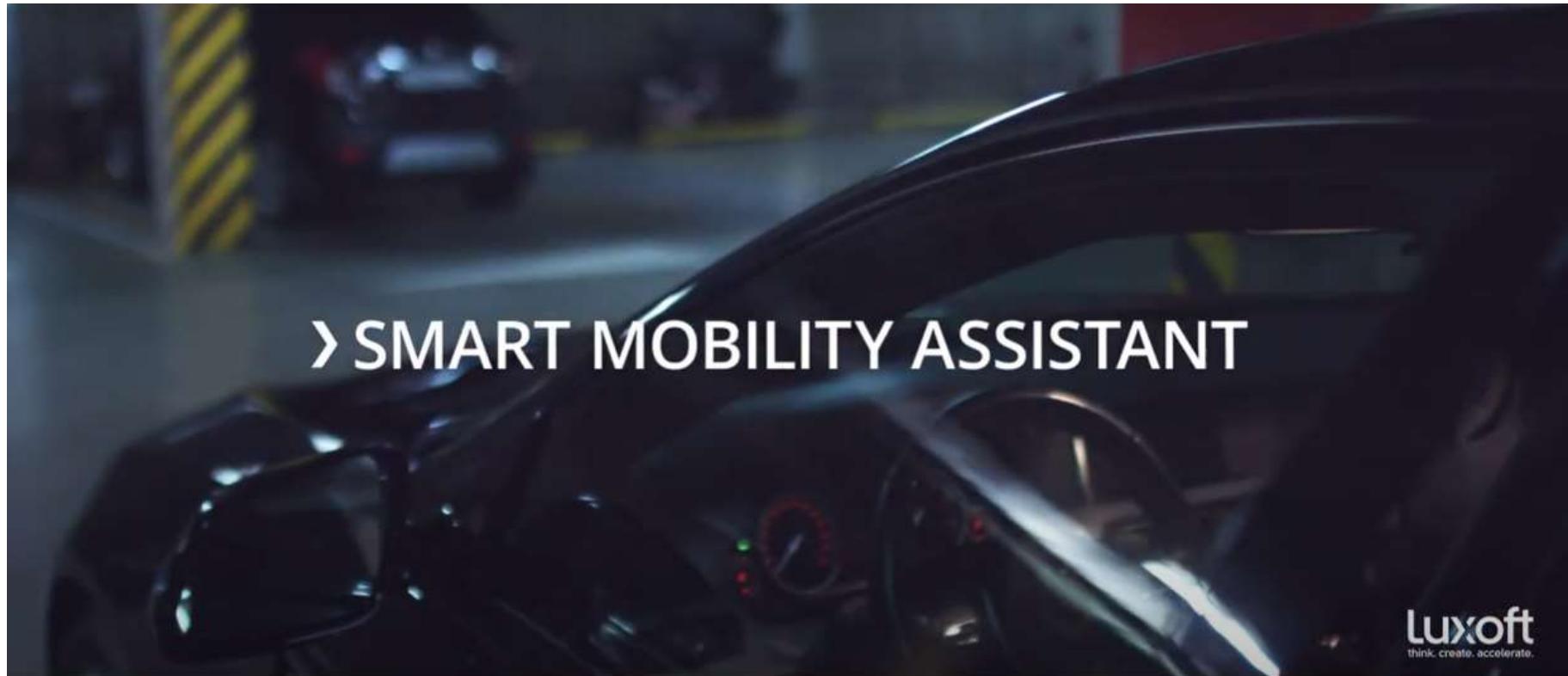
# Vehículos definidos por software (SDV)

- La **arquitectura central o zonal** forma la columna vertebral de los SDV (multiple ECUs vs central computer) [Unveiling the Transformation of Software-Defined Vehicles \(eetimes.eu\)](#) [The Eclipse Foundation](#)
- Sinergias/conexión con los **servicios en la nube**: AWS, Azure, Google Cloud, etc.
- Actualizaciones inalámbricas (OTA).



<https://www.sciencedirect.com/science/article/pii/S2949715922000038>  
[BlackBerry-SDV explained](#)

## Luxoft: asistente de movilidad

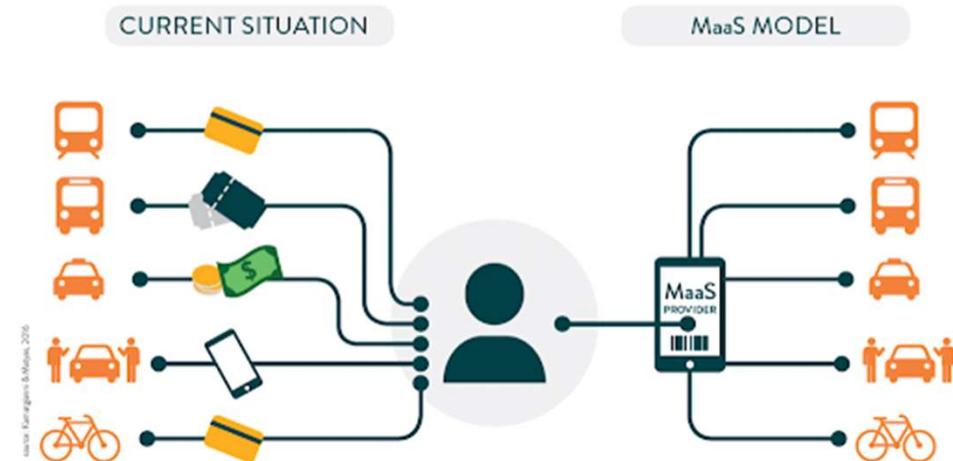


**luxoft**  
think. create. accelerate.

[Smart Mobility Assistant | Luxoft Videos](#)

# ¿Qué es Mobility as a Service (MaaS)?

- Plataforma inteligente de vehículos conectados.
- Planificar rutas. Optimizar flujos de transporte.
- Sustituir compra de vehículos por **Pay as you drive**, y promover el **carsharing**.
- Reducir las superficies para **aparcamientos**.
- “Ventanilla única” para diferentes medios de transporte. Una aplicación y un sistema de pago.
- Vehículos autónomos, self-driving bus.
- Compartir datos en tiempo real: tráfico, meteorológicas, disponibilidad de vehículos.
- Experiencia de viaje **personalizada**: preferencias y hábitos de los usuarios.
- Optimizar costes y reducir la huella de carbono.



[Mobility as a Service | URBAN MOBILITY SIMPLY EXPLAINED](#)

[Mobility as a Service Components](#)

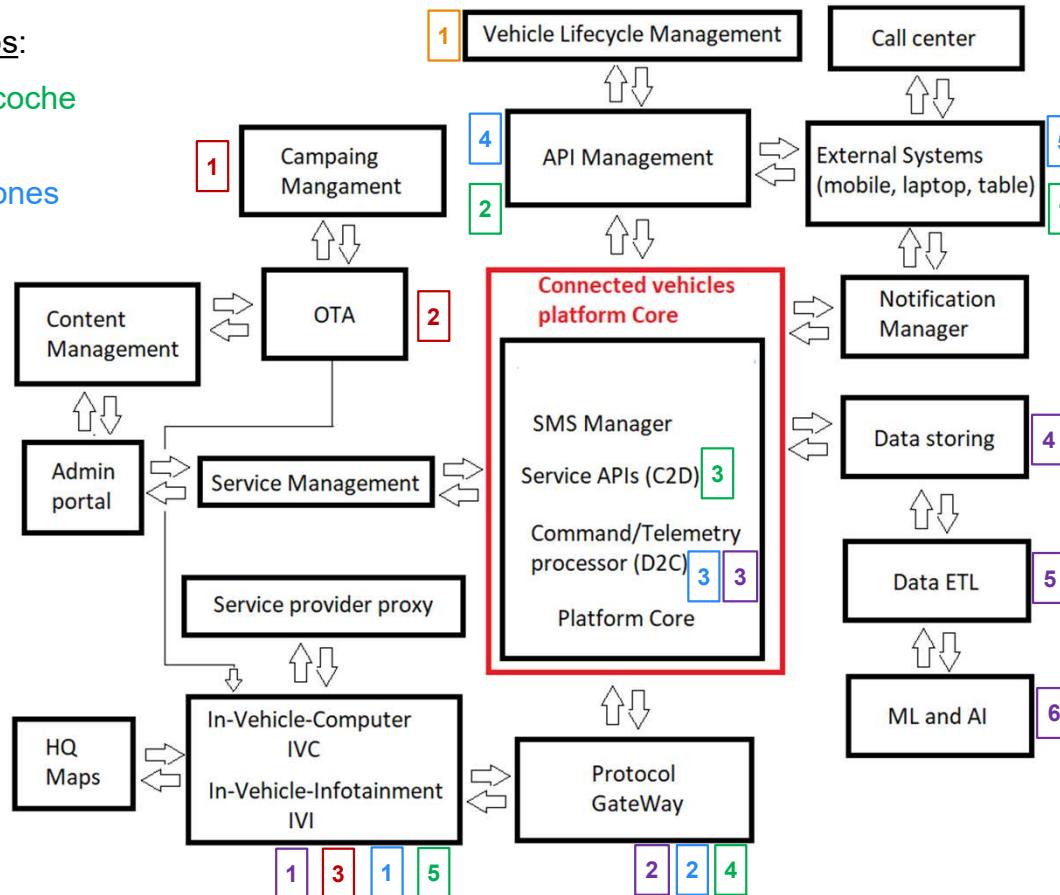
# Arquitectura de una Plataforma de vehículos conectados

## Flujos principales de datos:

- Envío de Comandos al coche (Cloud to Device C2D)
- Telemetrías y Notificaciones (Device to Cloud D2C)

## Otros flujo de datos:

- Gestión de vida del vehículo
- Gestión de campaña + OTA
- Servicios de ML e AI
- Acceso a servicios en internet

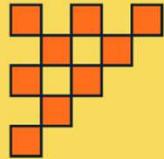


## Requisitos de las plataformas de vehículos conectados:

- Escalabilidad
- Fault tolerant (redundancia)
- Optimización de coste
- Baja latencia
- Ciberseguridad

Building and Modernizing Connected Vehicle platforms with AWS IoT

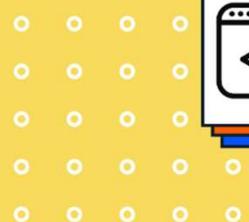
Automotive messaging, data & analytics reference architecture - Azure Event Grid | Microsoft Learn



# Cómo crear una Web API con .NET 8 para conectarnos a nuestro coche eléctrico Tesla Model 3



<https://github.com/luiscoco/codemotion-Barcelona-2023-public-repo>



# Paso 1. Instalación del entorno de desarrollo

1.1. Instalación de **.NET 8 SDK**: <https://dotnet.microsoft.com/es-es/download/dotnet/thank-you/sdk-8.0.100-windows-x64-installer>

Comprobar la instalación ejecutando el comando: **dotnet --version**

1.2. Instalación de **Visual Studio 2022 Community Edition**:  
<https://visualstudio.microsoft.com/es/vs/community/>

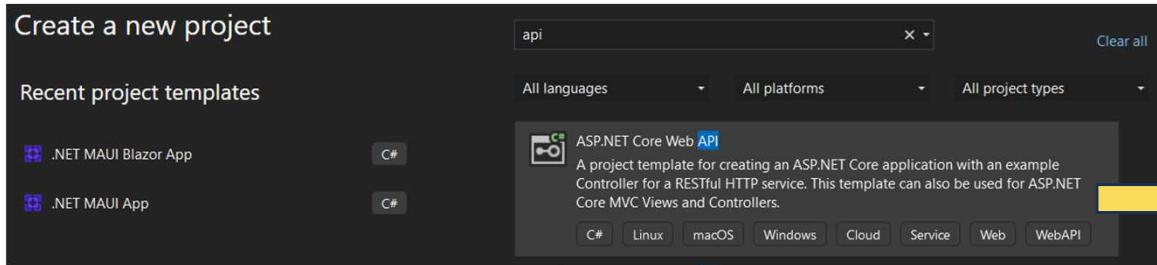
1.3. Instalación de **VSCode**: <https://code.visualstudio.com/download>

1.4. Instalación de **Docker Desktop**:  
<https://docs.docker.com/desktop/install/windows-install/>

1.5. Instalación de **Studio 3T Free for MongoDB**: <https://studio3t.com/download/>

## Paso 2. Creamos una aplicación .NET API

### Opción 1. Utilizamos Visual Studio 2022



#### Additional information

##### ASP.NET Core Web API

###### Framework

.NET 7.0 (Standard Term Support)

###### Authentication type

None

Configure for HTTPS

Enable Docker

###### Docker OS

Linux

Use controllers (uncheck to use minimal APIs)

Enable OpenAPI support

Do not use top-level statements

### Opción 2. Utilizamos commandos dotnet CLI

```
Command Prompt
C:\>dotnet --version
7.0.402

C:\>md codemotionTeslaAPI

C:\>cd codemotionTeslaAPI

C:\codemotionTeslaAPI>dotnet new webapi --framework net7.0
The template "ASP.NET Core Web API" was created successfully.

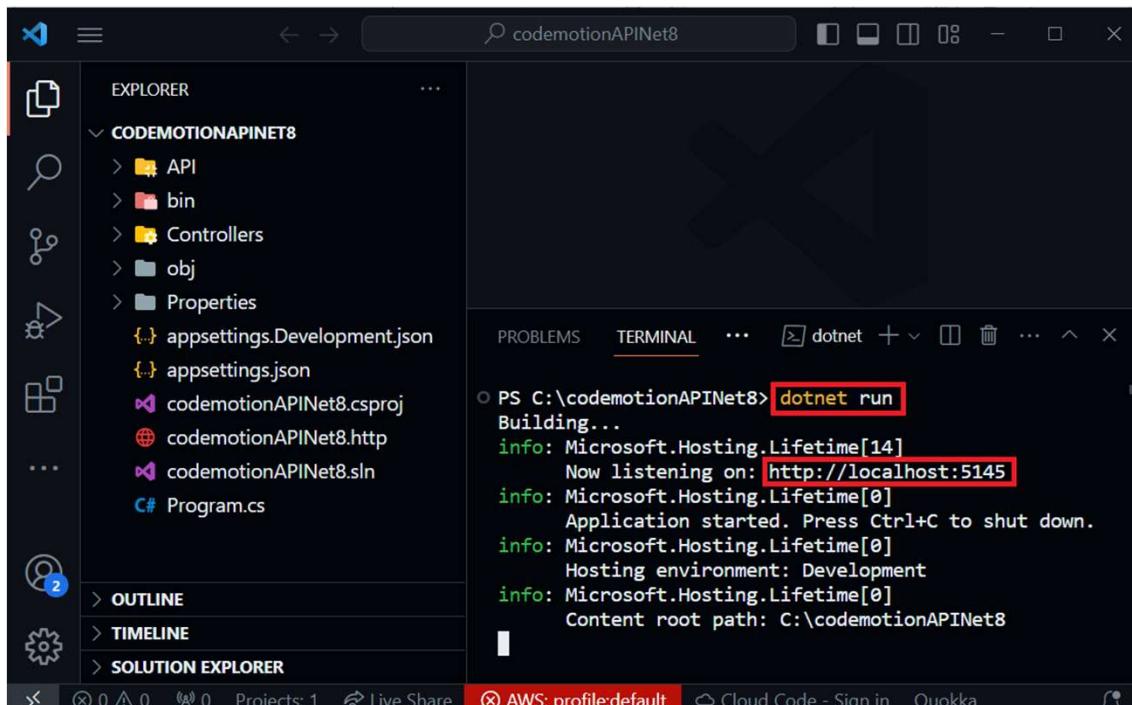
Processing post-creation actions...
Restoring C:\codemotionTeslaAPI\codemotionTeslaAPI.csproj:
  Determining projects to restore...
  Restored C:\codemotionTeslaAPI\codemotionTeslaAPI.csproj (in 2.48 sec).
Restore succeeded.
```

dotnet --version  
dotnet new list  
dotnet new --help webapi  
md codemotiondotnet8api  
cd codemotiondotnet8api  
dotnet new webapi --framework net8.0  
code .

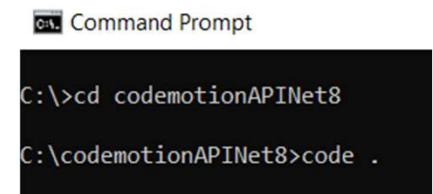
## Paso 3. Desarrollar nuestra aplicación con VSCode

Después de crear la aplicación con **dotnet CLI** ejecutamos el comando: **code .**

Para ejecutar la Web API ejecutamos el comando: **dotnet run**



<http://localhost:5136/swagger/index.html>



The screenshot shows a Command Prompt window with two commands entered: `cd codemotionAPINet8` and `code .` The second command is highlighted with a red box.

```
C:\>cd codemotionAPINet8
C:\codemotionAPINet8>code .
```

## Paso 4. Instalación de dependencias (VSCode)

Instalamos las siguientes librerías (paquetes Nuget) ejecutando los siguientes comandos:

```
dotnet add package Microsoft.AspNetCore.Cors (Enable Cross-Origin Requests CORS)
dotnet add package Microsoft.Extensions.Configuration (Read and write in appSetting.json file)
dotnet add package MongoDB.Driver (MongoDb communication)
dotnet add package Newtonsoft.Json (JSON Serialization and Deserialization)
```

Para añadir los paquetes al proyecto ejecutamos el comando: **dotnet restore**

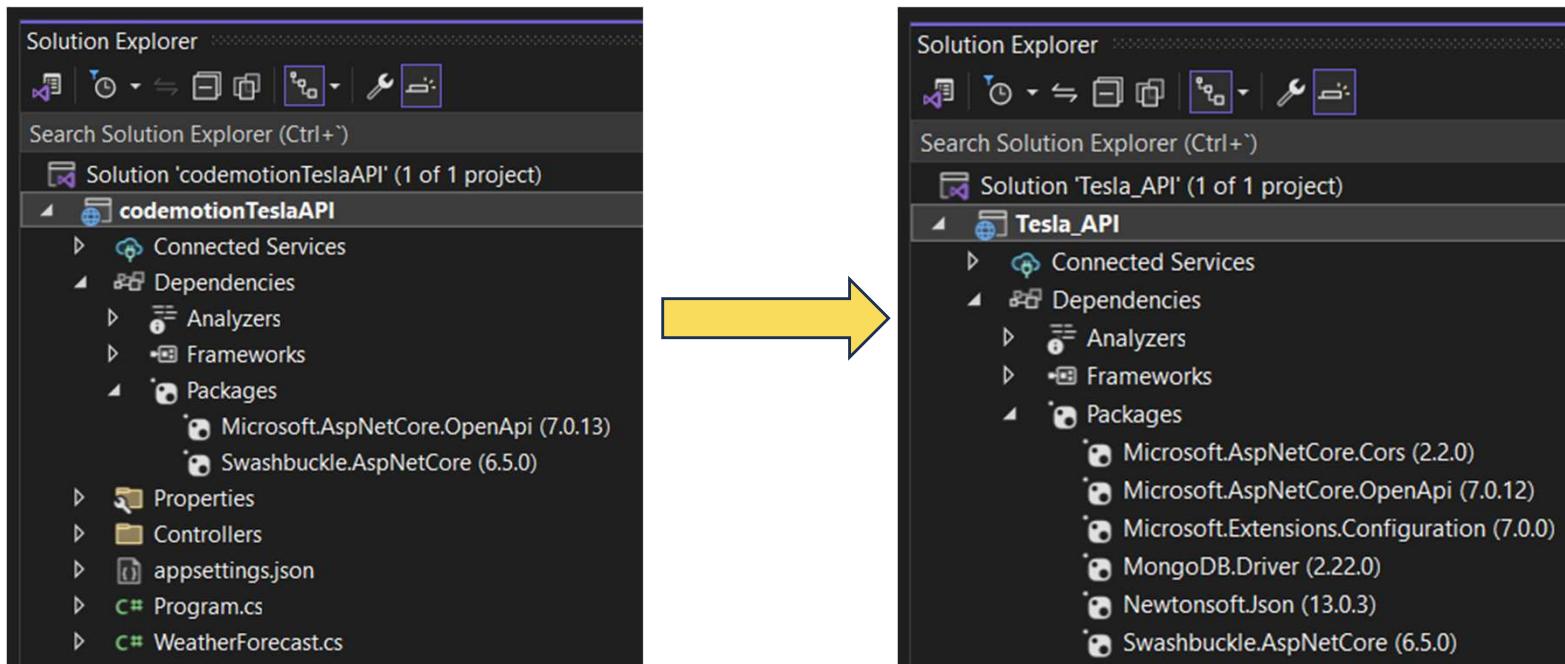
Verificamos que la instalación de la librerias ha sido correcta, consultando el archive **csproj**.

```
dotnet add package Microsoft.AspNetCore.Cors
dotnet add package Microsoft.Extensions.Configuration
dotnet add package MongoDB.Driver
dotnet add package Newtonsoft.Json
```

## Paso 5. Instalación de dependencias

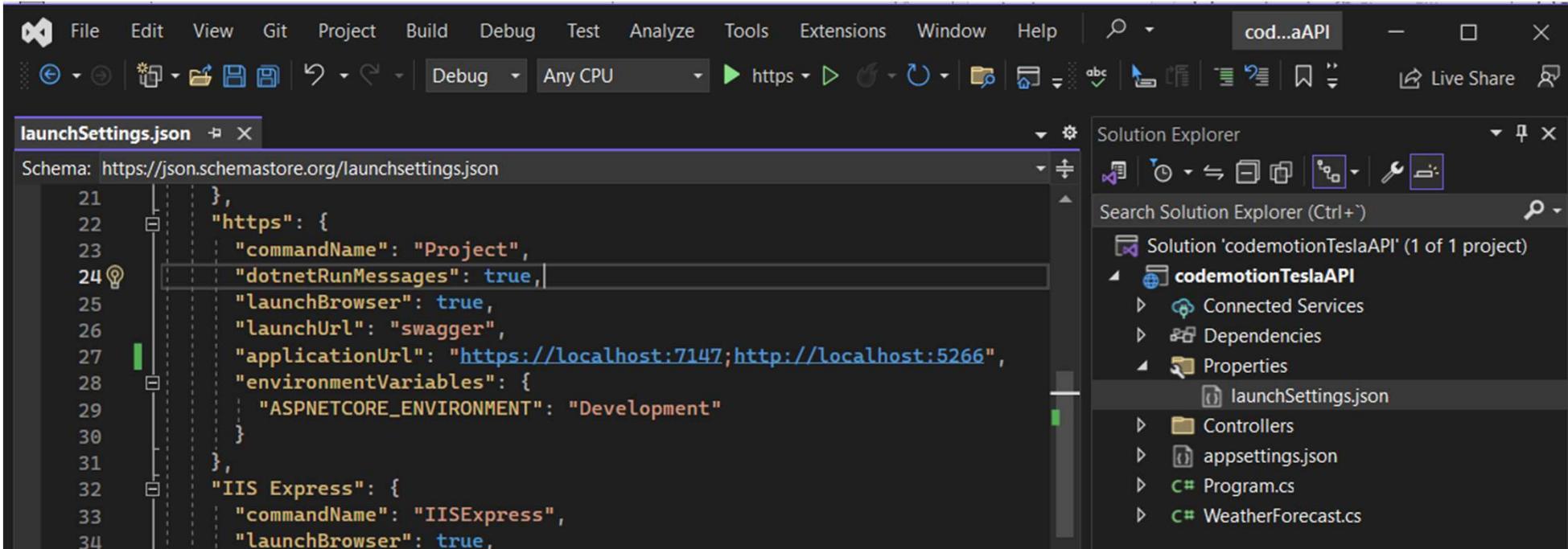
Instalamos las siguientes librerías con ayuda de NuGet:

Microsoft.AspNetCore.Cors    Microsoft.Extensions.Configuration    MongoDB.Driver  
Newtonsoft.Json



## Paso 6. Modificamos el archivo “launchSettings.json”

Fijamos el número de puerto (7147) para ejecutar la aplicación API



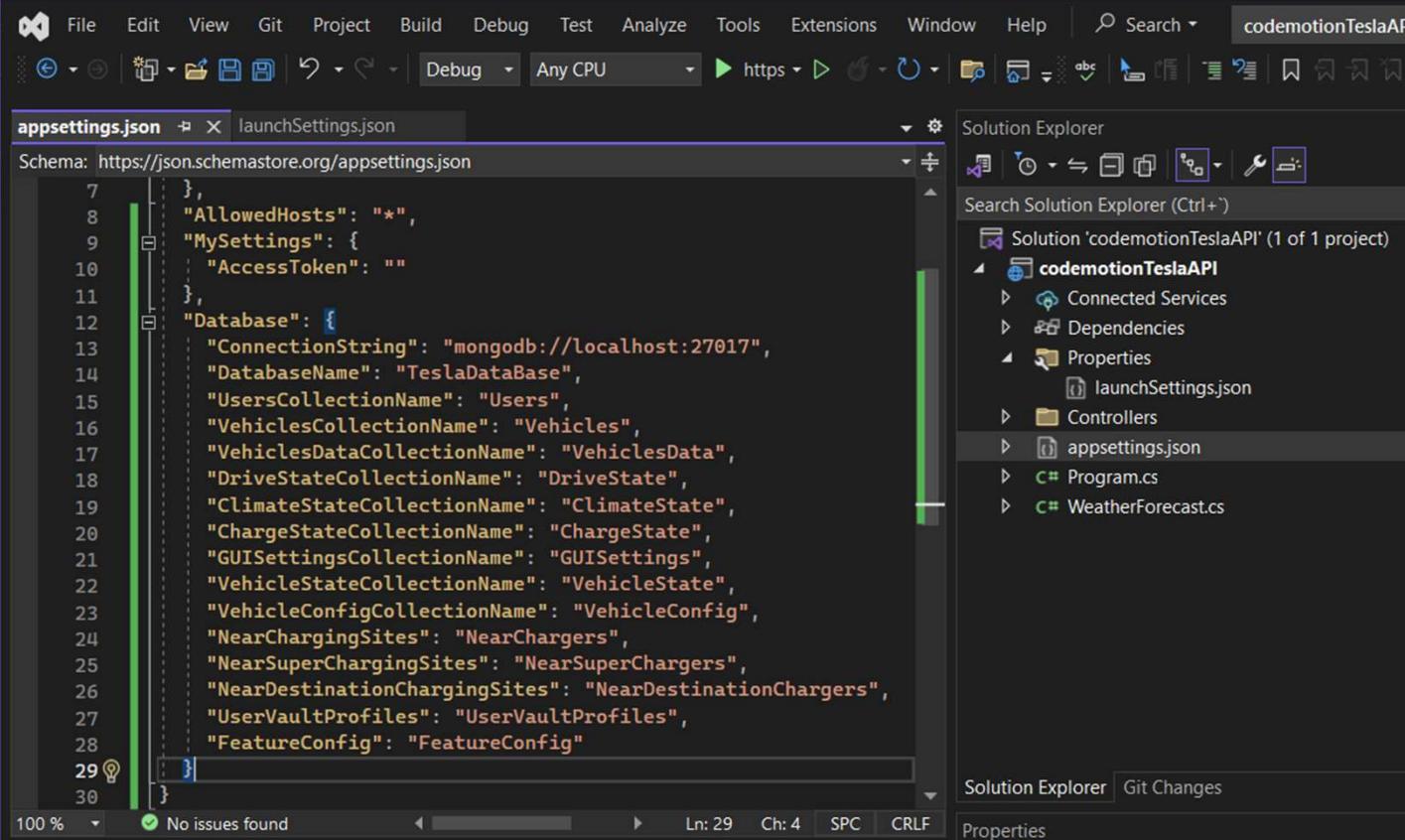
The screenshot shows the Visual Studio IDE interface. The title bar says "cod...aAPI". The menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help. The toolbar has various icons for file operations like Open, Save, Find, and Run. The status bar shows "Debug Any CPU https". The main area is an editor titled "launchSettings.json" with the schema "https://json.schemastore.org/launchsettings.json". The code content is:

```
Schema: https://json.schemastore.org/launchsettings.json
21     },
22     "https": {
23         "commandName": "Project",
24         "dotnetRunMessages": true,
25         "launchBrowser": true,
26         "launchUrl": "swagger",
27         "applicationUrl": "https://localhost:7147;http://localhost:5266",
28         "environmentVariables": {
29             "ASPNETCORE_ENVIRONMENT": "Development"
30         }
31     },
32     "IIS Express": {
33         "commandName": "IISExpress",
34         "launchBrowser": true,
```

To the right is the Solution Explorer window showing the project "codemotionTeslaAPI" with files like "launchSettings.json", "Controllers", "appsettings.json", "Program.cs", and "WeatherForecast.cs".

## Paso 7. Modificamos el archivo “appSettings.json”

Introducimos los datos de la base de datos MongoDB y el AccessToken



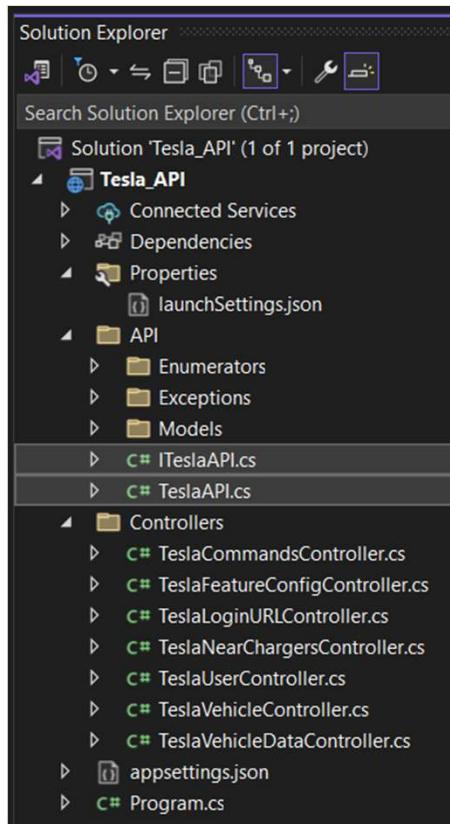
The screenshot shows the Visual Studio IDE interface. The title bar says "codemotionTeslaAPI". The main window displays the "appsettings.json" file in the editor. The code content is as follows:

```
Schema: https://json.schemastore.org/appsettings.json
    7   },
    8   "AllowedHosts": "*",
    9   "MySettings": {
   10     "AccessToken": ""
   11   },
   12   "Database": {
   13     "ConnectionString": "mongodb://localhost:27017",
   14     "DatabaseName": "TeslaDataBase",
   15     "UsersCollectionName": "Users",
   16     "VehiclesCollectionName": "Vehicles",
   17     "VehiclesDataCollectionName": "VehiclesData",
   18     "DriveStateCollectionName": "DriveState",
   19     "ClimateStateCollectionName": "ClimateState",
   20     "ChargeStateCollectionName": "ChargeState",
   21     "GUISettingsCollectionName": "GUISettings",
   22     "VehicleStateCollectionName": "VehicleState",
   23     "VehicleConfigCollectionName": "VehicleConfig",
   24     "NearChargingSites": "NearChargers",
   25     "NearSuperChargingSites": "NearSuperChargers",
   26     "NearDestinationChargingSites": "NearDestinationChargers",
   27     "UserVaultProfiles": "UserVaultProfiles",
   28     "FeatureConfig": "FeatureConfig"
   29   }
  30 }
```

The Solution Explorer on the right shows the project structure with files like "Connected Services", "Dependencies", "Properties", "Controllers", "appsettings.json", "Program.cs", and "WeatherForecast.cs".

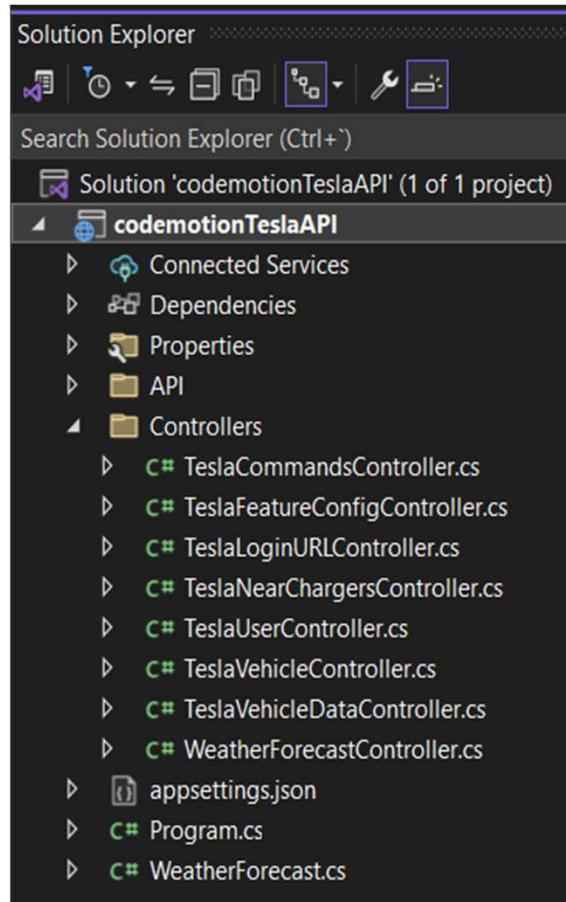
## Paso 8. Creamos los modelos de datos y servicio “TeslaAPI”

Creamos una nueva carpeta “**API/Models**” donde ubicaremos los archivos incluyen los modelos de datos de la aplicación. En los archivos “**ITeslaAPI.cs**” y “**TeslaAPI.cs**” definimos los Servicios.



## Paso 9. Creamos los controladores de la aplicación

Creamos una nueva carpeta “**Controllers**” donde ubicamos los controladores de nuestra aplicación.



## Paso 10 (opcional). Creamos los controladores con inyección de dependencia del servicio "TeslaAPI"

```
using TeslaAPI;
...
public class TeslaVehicleController : ControllerBase
{
    ...
    private readonly ITeslaAPI _teslaAPIService;

    public TeslaVehicleController(IConfiguration configuration, IMongoCollection<Vehicle> vehiclesCollection,
        ILogger<TeslaVehicleController> logger, ITeslaAPI teslaAPIService)
    {
        ...
        _teslaAPIService = teslaAPIService;
        ...
    }

    [HttpPost("getVehicle", Name = "GetVehicle")]
    public async Task<Vehicle> GetVehicle()
    {
        Vehicle vehicle = new Vehicle();
        vehicle = await _teslaAPIService.WakeUpAsync("vehicleNumber");
        vehicle = await _teslaAPIService.GetVehicleAsync("vehicleNumber ");

        return vehicle;
    }
}
```

## Paso 10 (opcional). Creamos los controladores con inyección de dependencia del servicio "TeslaAPI"

Recomiendo la opción con inyección de dependencia, es más **extensible** y **mantenible** a largo plazo.

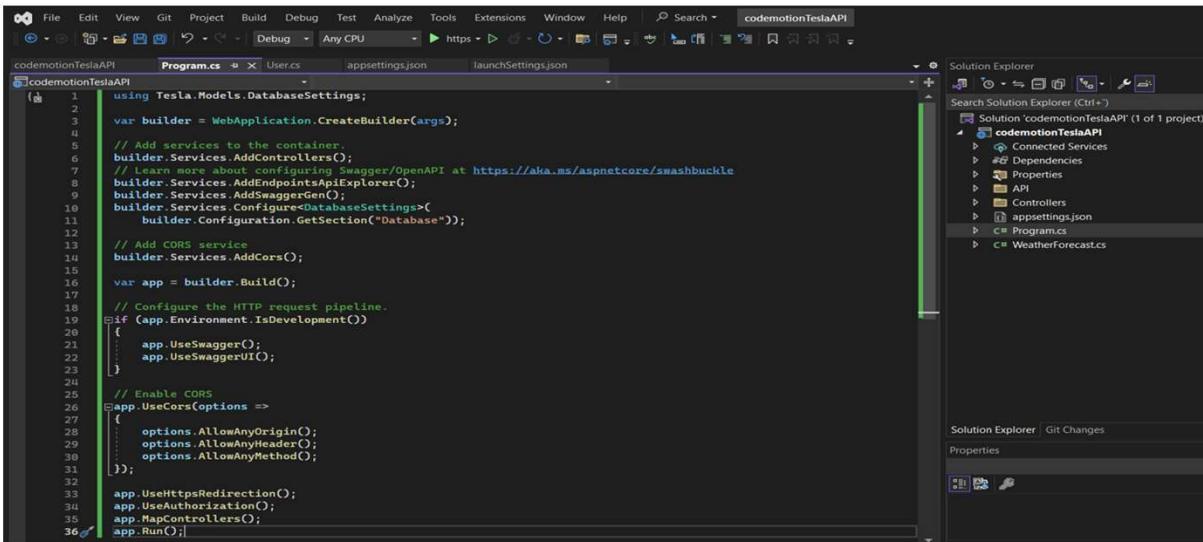
**Capacidad de prueba:** mejor capacidad de prueba al injectar la interfaz ITeslaAPIService, puede sustituirla fácilmente con una implementación simulada durante las pruebas unitarias.

**Desacoplamiento:** el controlador sólo depende de la interfaz ITeslaAPIService, lo que lo hace más flexible y fácil de reemplazar o ampliar en el futuro.

**Mantenibilidad:** si necesita cambiar la implementación de la API de Tesla o agregar nuevas funciones, puede hacerlo sin modificar el código del controlador.

## Paso 11. Modificar el archivo program.cs

Incluimos los servicios de la base de datos MongoDB y CORS



```
codemotionTeslaAPI Program.cs User.cs appsettings.json launchSettings.json
codemotionTeslaAPI
1  using Tesla.Models.DatabaseSettings;
2
3  var builder = WebApplication.CreateBuilder(args);
4
5  // Add services to the container.
6  builder.Services.AddControllers();
7  // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
8  builder.Services.AddEndpointsApiExplorer();
9  builder.Services.AddSwaggerGen();
10 builder.Services.Configure<DatabaseSettings>(
11     builder.Configuration.GetSection("Database"));
12
13 // Add CORS service
14 builder.Services.AddCors();
15
16 var app = builder.Build();
17
18 // Configure the HTTP request pipeline.
19 if (app.Environment.IsDevelopment())
20 {
21     app.UseSwagger();
22     app.UseSwaggerUI();
23 }
24
25 // Enable CORS
26 app.UseCorsOptions =>
27 {
28     options.AllowAnyOrigin();
29     options.AllowAnyHeader();
30     options.AllowAnyMethod();
31 };
32
33 app.UseHttpsRedirection();
34 app.UseAuthorization();
35 app.MapControllers();
36 app.Run();
```

Si preferimos realizar la **injeción de dependencias** del servicio **TeslaAPI** en los constructores de los controladores, entonces incluir el siguientes código en el archive **program.cs**:

(opción 1)

**services.AddScoped<ITeslaAPI, TeslaAPI>();**

(opción 2)

**services.AddTeslaApi();**

## Paso 12. Ejecutar contenedor de MongoDB

### Ejecutar Docker Desktop

Opcion 1: Descargar y ejecutar el contenedor docker de MongoDb :

```
docker run -d --rm -p 27017:27017 --name mongo --network mongoCluster mongo:latest mongod --bind_ip_all
```

```
docker ps -a
```

or

Opcion 2: Descargar y ejecutar los contenedores docker de la MongoDb ReplicaSet:

```
docker run -d --rm -p 27017:27017 --name mongo1 --network mongoCluster mongo:latest mongod --replicaSet myReplicaSet --bind_ip localhost,mongo1
```

```
docker run -d --rm -p 27018:27017 --name mongo2 --network mongoCluster mongo:latest mongod --replicaSet myReplicaSet --bind_ip localhost,mongo2
```

```
docker run -d --rm -p 27019:27017 --name mongo3 --network mongoCluster mongo:latest mongod --replicaSet myReplicaSet --bind_ip localhost,mongo3
```

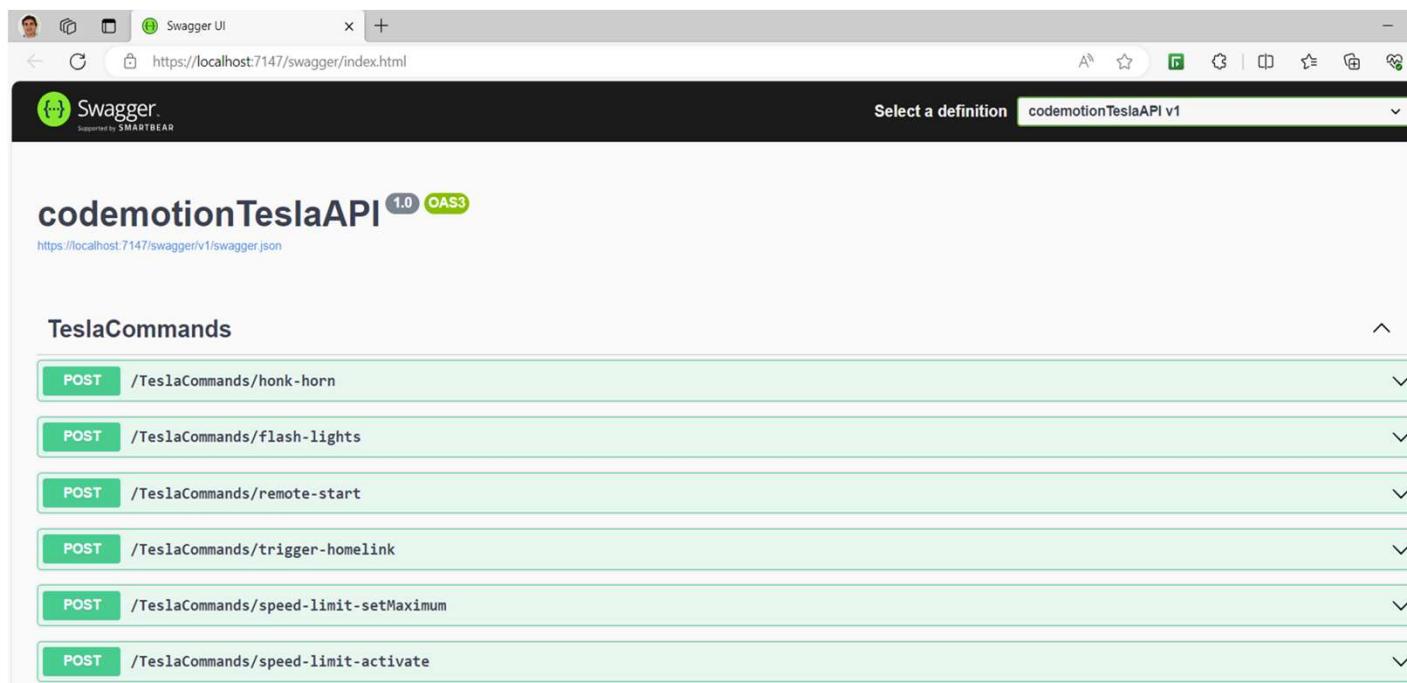
```
docker exec -it mongo1 mongosh --eval "rs.initiate({ _id: 'myReplicaSet', members: [{_id: 0, host: 'mongo1', priority: 3}, {_id: 1, host: 'mongo2', priority: 2}, {_id: 2, host: 'mongo3', priority: 1}]}))"
```

```
docker ps -a
```

## Paso 13. Compilar y Ejecutar la aplicación

Antes de compilar y ejecutar la aplicación eliminamos los dos archivos “**WeatherForecast**” del ejemplo(template) original de las aplicaciones .NET Web API

Compilamos y ejecutamos la aplicación .NET Web API con el comando: `dotnet run`



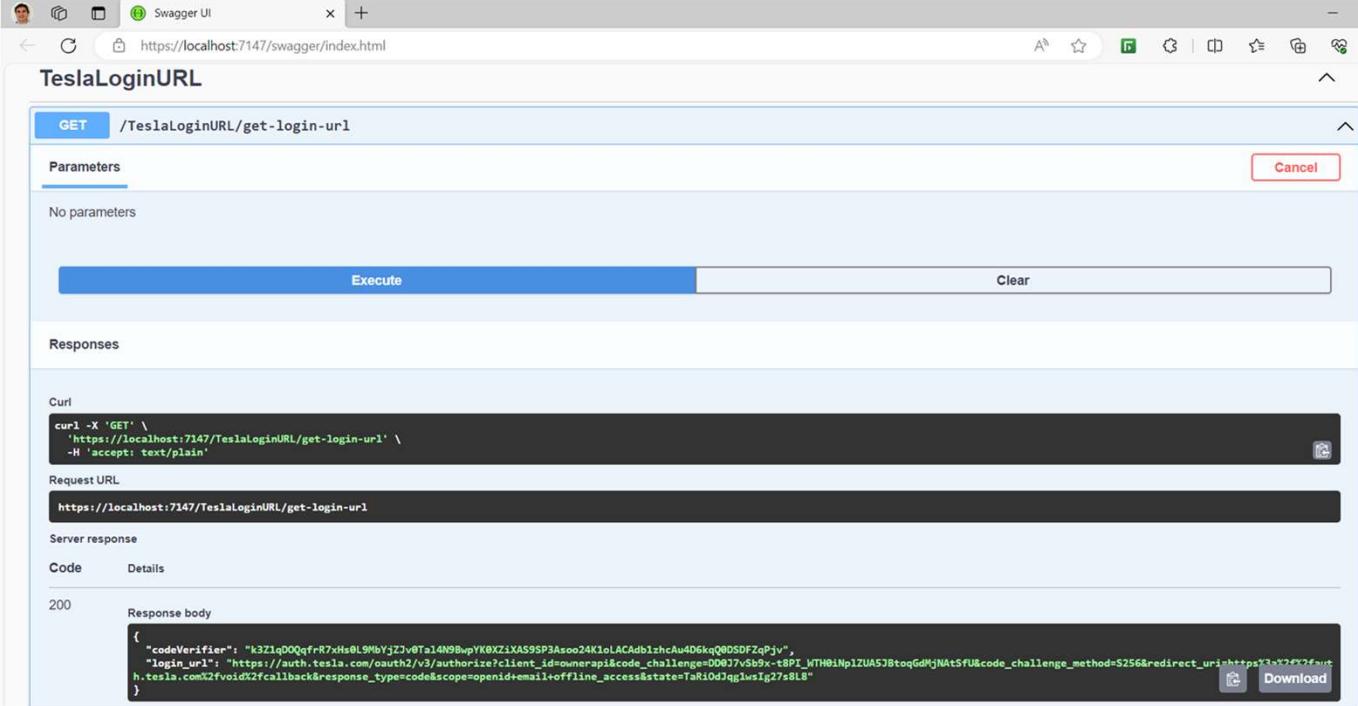
The screenshot shows a web browser window displaying the Swagger UI for the `codemotionTeslaAPI v1`. The title bar reads "Swagger UI". The address bar shows the URL `https://localhost:7147/swagger/index.html`. The main content area is titled "codemotionTeslaAPI 1.0 OAS3" and includes a link to `https://localhost:7147/swagger/v1/swagger.json`. A dropdown menu labeled "Select a definition" shows "codemotionTeslaAPI v1". Below the title, there is a section titled "TeslaCommands" containing six POST methods:

- `POST /TeslaCommands/honk-horn`
- `POST /TeslaCommands/flash-lights`
- `POST /TeslaCommands/remote-start`
- `POST /TeslaCommands/trigger-homelink`
- `POST /TeslaCommands/speed-limit-setMaximum`
- `POST /TeslaCommands/speed-limit-activate`

## Paso 14. Login

En este paso temenos que ejecutar la API “**/TeslaLoginURL/get-login-url**”. Como respuesta recibidos el “**codeVerifier**” y el “**login\_url**”.

Copiamos el valor del “**login\_url**” y lo pegamos en la barra de nuestro buscador de internet para acceder a la **página de Login de TESLA**.

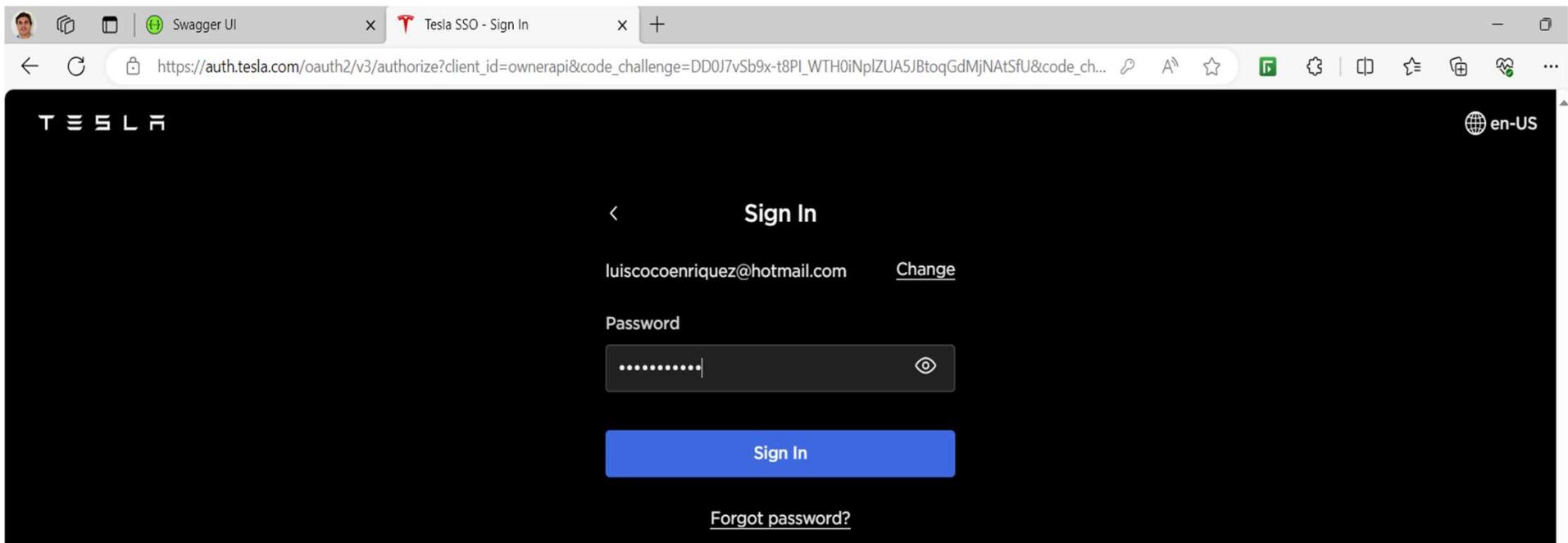


The screenshot shows the Swagger UI interface for the **TeslaLoginURL** API. The endpoint **/TeslaLoginURL/get-login-url** is selected. The "Parameters" section indicates "No parameters". The "Responses" section shows a "Curl" command and a "Request URL" of **https://localhost:7147/TeslaLoginURL/get-login-url**. The "Code" tab is active, displaying a response body with the following JSON:

```
{
  "codeVerifier": "k3ZlqD0QqfrR7xHs0L9MbYjZJv0Ta14N9BwpYK0XZiXAS9SP3Asoo24K1oLACAdblzhcAu4D6kqQ0DSDFzqPjv",
  "login_url": "https://auth.tesla.com/oauth2/v3/authorize?client_id=ownerapi&code_challenge=DD037vb5x-t8PI_WTH0iNp1ZUA5JtoqGdMjNatSfU&code_challenge_method=S256&redirect_uri=https%3A%2F%2Fut.h.tesla.com%2fvoid%2fcallback&response_type=code&scope=openid+email+offline_access&state=TaRi0dJqglwsIg27s8L8"
}
```

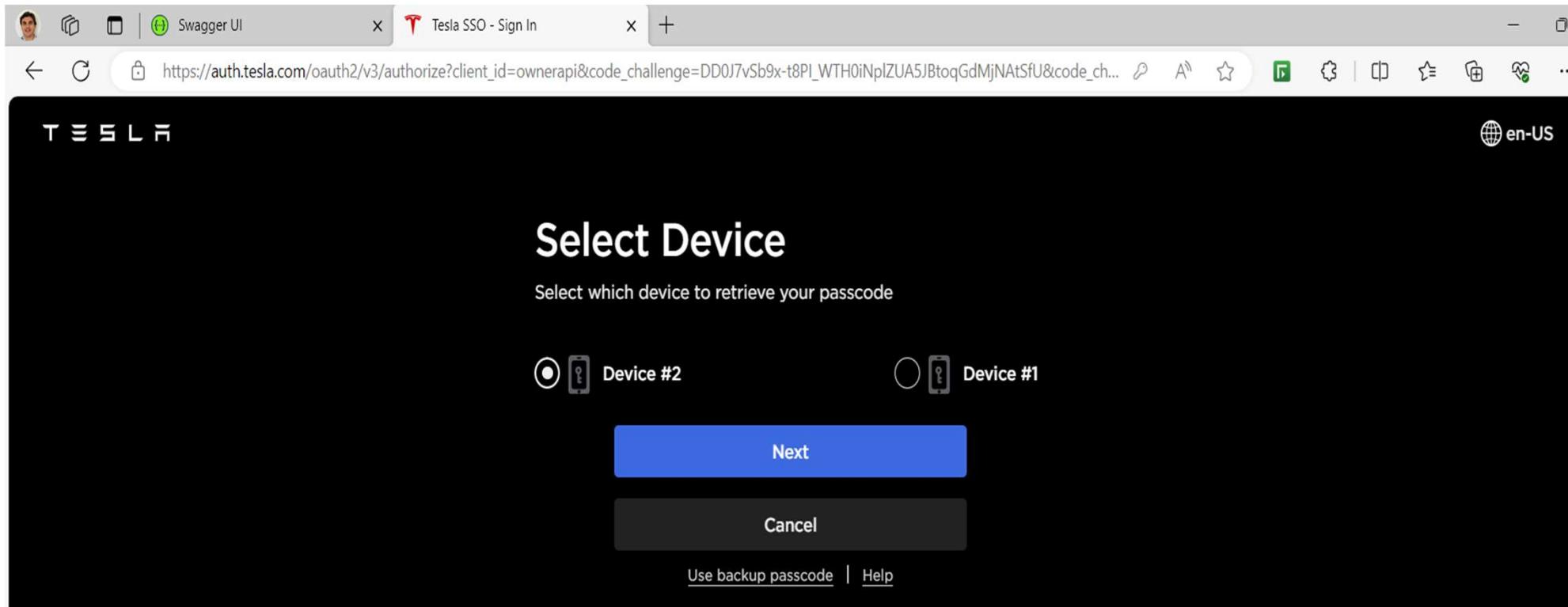
## Paso 15. Continuamos con el Login

Introducimos nuestro nombre de usuario (dirección de correo electrónico) y el password de nuestra cuenta TESLA.



## Paso 16. MFA(Multi-Factor Authentication)

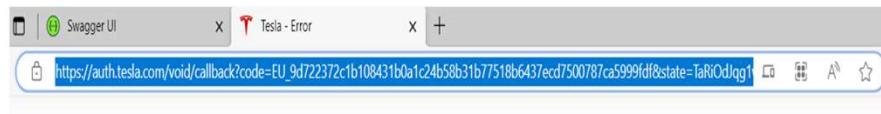
En caso de que hayamos activado el MFA en nuestra cuenta de TESLA, tenemos que realizar la autenticación MFA. En mi caso activé el MFA con la aplicación "**Authenticator**" en mi teléfono móvil.



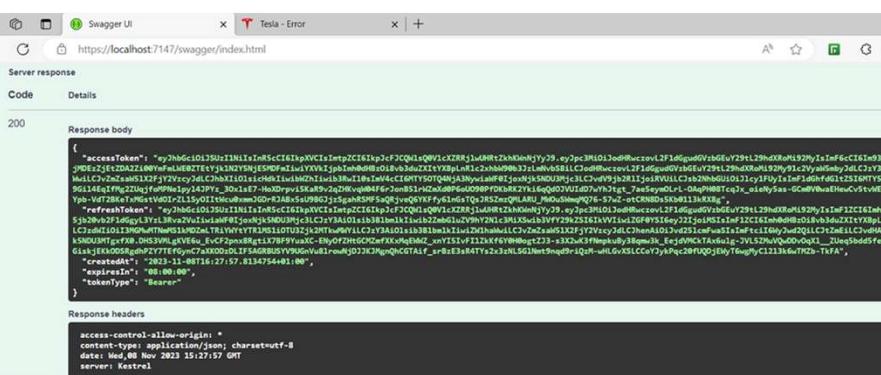
## Paso 17. Obtenemos el access token

Copiamos la URL a la que nos ha redirigido la página de Login de TESLA y la pegamos en el campo “**redirectUrl**” y también cumplimentamos el campo “**codeverifier**”, obtenido en el paso 14 de esta presentación. Una vez introducidos ambos campos “**redirectUrl**” y “**codeverifier**” ejecutamos la API “**TeslaloginURL/get-token-after-login**”









Esta función guarda en la variable **AccessToken** del **appSettings.json** el token que temenos que incluir en el Header de las APIs para tener acceso a la nube TESLA. Este token va a ser leido por todas las acciones de los controladores para solicitar la autorización de acceso a la API de Tesla.

## Paso 18. Comprobamos los datos escritos en MongoDB con Studio 3T

Studio 3T Free for MongoDB

File Edit Database Collection Index Document View Help

Connect IntelliShell Export Import Feedback Want More?

Search open connections (Ctrl+F) Quickstart

**Connection Manager**

New Connection New Folder Edit Delete Duplicate

Click here to filter connections

Name	DB Server
mongoConnection	localhost:27017

Open connections

mongoConnection localhost:27017 [direct]

TeslaDatabase

- Collections (1)
  - ChargeState
- System (0)
- Views (0)
- admin
- config
- local

Operations

Quickstart | IntelliShell: mongoConnection

localhost:27017 > TeslaDatabase

Raw shell output Find Query (line 1) ↵

```
1 / db.getCollection("ChargeState").find({})
```

Raw shell output Find Query (line 1) ↵

```
1 {
  "id": ObjectId("65dbb2e7648072e4471e8ff0"),
  "BatteryIserton": false,
  "BatteryLevel": NumberInt(79),
  "BatteryRange": 212.91,
  "ChargeAmps": NumberInt(16),
  "ChargeCurrentRequest": NumberInt(16),
  "ChargeCurrentRequestMax": NumberInt(16),
  "ChargeEnableRequest": true,
  "ChargeEnergyAdded": 22.11,
  "ChargeLimitStateOfCharge": NumberInt(100),
  "ChartlimitStateOfChargeMaximum": NumberInt(100),
  "ChargeLimitStateOfChargeMinimum": NumberInt(50),
  "ChargeLimitStateOfChargeStandard": NumberInt(80),
  "ChargeMilesAddedTotal": 100.0,
  "ChargeMilesAddedRated": 100.0,
  "ChargePortColdWeatherMode": false,
  "ChargePortColor": "invalid",
  "ChargePortDoorOpen": false,
  "ChargePortLatch": "Engaged",
  "ChargeRate": 0.0,
  "ChargeToMaxRange": null,
  "ChargerActualCurrent": NumberInt(0),
  "ChargerPhases": null,
  "ChargerPilotCurrent": NumberInt(16),
```

1 document selected

Count Documents 00:00:00.008

## Paso 19. Comprobamos los datos escritos en MongoDB con comandos

```

mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
C:\Users\LEnriquez>docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
10878a94b059 mongo:latest "docker-entrypoint.s..." About an hour ago Up About an hour 0.0.0.0:27017->27017/tcp mongo

C:\Users\LEnriquez>docker exec -it mongo bash
root@10878a94b059:/# mongosh
Current Mongosh Log ID: 654bb3a1915464d60d7d7578
Connecting to: mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.0.1
Using MongoDB: 7.0.2
Using Mongosh: 2.0.1

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

-----
The server generated these startup warnings when booting
2023-11-08T14:55:01.175+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
2023-11-08T14:55:02.538+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
2023-11-08T14:55:02.538+00:00: /sys/kernel/mm/transparent_hugepage/enabled is 'always'. We suggest setting it to 'never'
2023-11-08T14:55:02.538+00:00: vm.max_map_count is too low
-----

test> show dbs
TeslaDataBase 40.00 KiB
TeslaDataBase> use TeslaDataBase
already on db TeslaDataBase
TeslaDataBase> db.ChargeState.findOne()

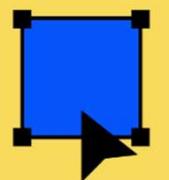
```

**docker exec -it mongo bash**  
**mongosh**  
**show dbs**  
**use TeslaDataBase**  
**show collections**  
**db.VehiclesData.findOne()**  
**db.VehiclesData.find()**  
**db.Users.find()**

# PREGUNTAS?



## REPOSITORIOS DE CÓDIGO Y OTROS...



# Repositorio de códigos

- <https://www.teslaapi.io/>
- <https://www.tesla.com/developer-docs>
- <https://developer.tesla.com/docs/fleet-api>
- [GitHub - Azure/azure-iot-protocol-gateway:](#)  
[Azure IoT protocol gateway enables protocol translation for Azure IoT Hub](#)
- <https://github.com/aws/aws-iot-fleetwise-edge>

## Otros casos de estudio Luxoft

- **Coches que se curan solos:** <https://www.luxoft.com/videos/luxofts-remote-repair-of-software-defined-vehicles>
- <https://www.luxoft.com/files/pdfs/case-studies/Public-Transportation-Smart-Card.pdf>
- **Conectividad en tiempo real para seguros:** <https://www.luxoft.com/case-studies/enabling-effective-monetization-of-connected-car-data>
- **CX en movilidad urbana:** <https://www.luxoft.com/case-studies/an-intuitive-customer-experience-that-is-reshaping-the-future-of-urban-mobility-from-research-to-realization>
- **Gamification en coches:** <https://www.luxoft.com/blog/gamification-could-revolutionize-automotive-industry>

# ¡Contacta conmigo!



[@GitHub](#)



[@Linkedin](#)

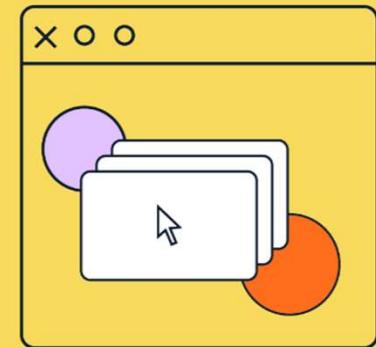


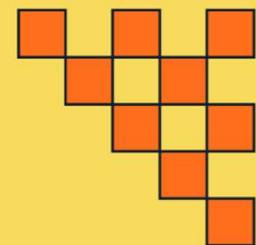


</>

## {e} WORKSHOP FEST

¡Gracias!





## {e} WORKSHOP FEST

**¡Recuerda puntuar  
el Workshop!**

