

Trabajo Práctico 5 - Herramientas de construcción de software

1- Ejemplo con C# y .NET Core

Instalar el SDK de .NET Core: Asegúrate de tener el SDK de .NET Core instalado en tu sistema. Puedes descargarlo desde el sitio web oficial de .NET:

<https://dotnet.microsoft.com/download>

Crear un Proyecto de Web API:

Abre una terminal y navega hasta la ubicación donde deseas crear tu proyecto. Luego, ejecuta el siguiente comando para crear un nuevo proyecto de Web API:

`dotnet new webapi -n MiProyectoWebAPI`

Esto creará un nuevo proyecto de Web API llamado "MiProyectoWebAPI" en un directorio con el mismo nombre.

Navegar al Directorio del Proyecto: Ve al directorio del proyecto que acabas de crear:

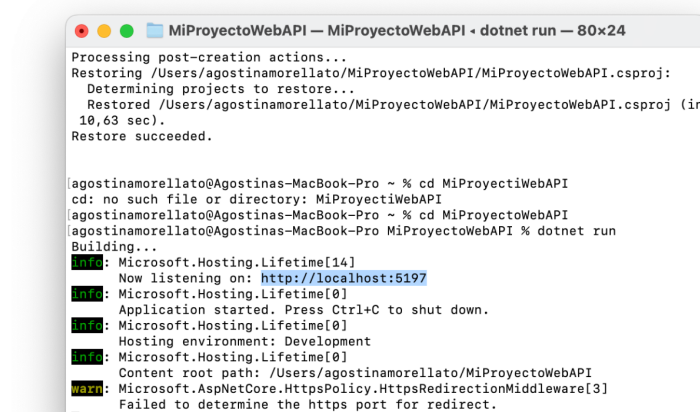
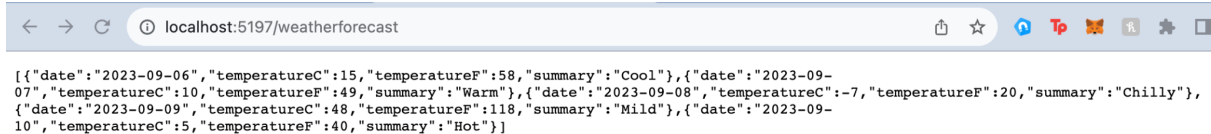
`cd MiProyectoWebAPI`

Ejecutar la Aplicación: Para ejecutar la aplicación de Web API, ejecuta el siguiente comando:

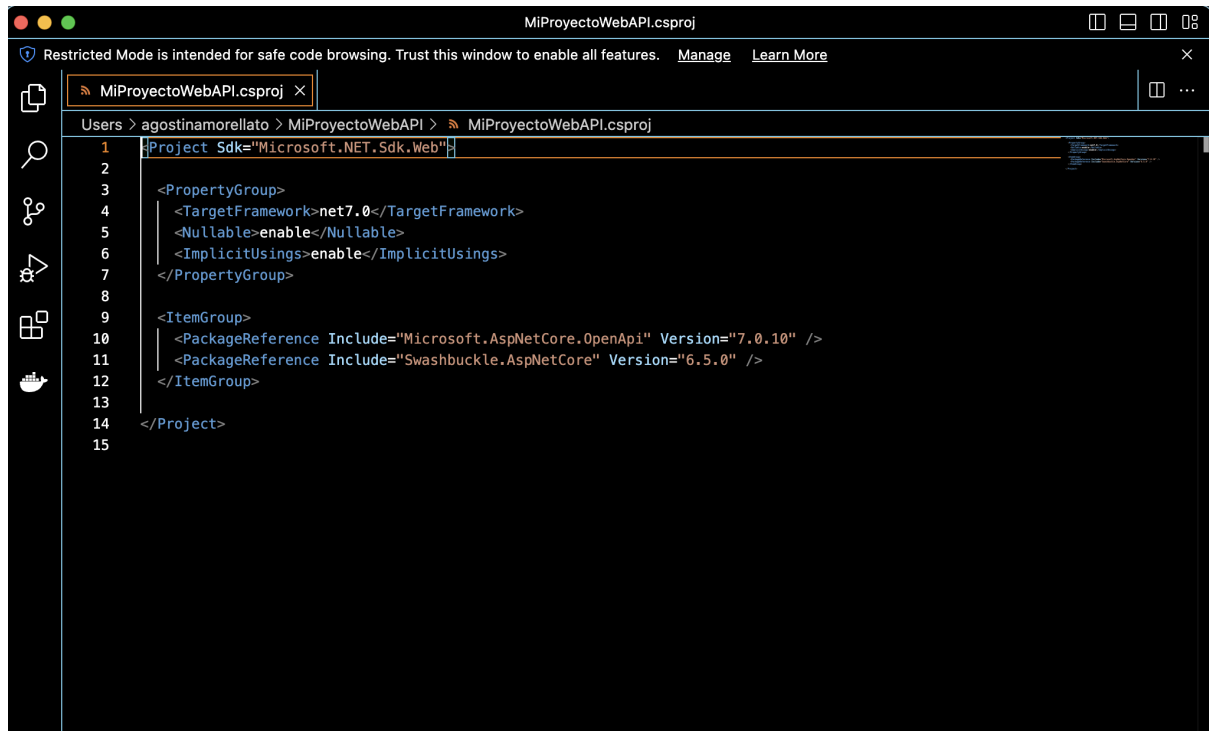
`dotnet run`

Esto iniciará la aplicación y la hará disponible en una URL local.

Navegar a la url indicada en el mensaje recibido por consola añadiendo /weatherforecast



Revisar el archivo MiProyectoWebAPI.csproj:



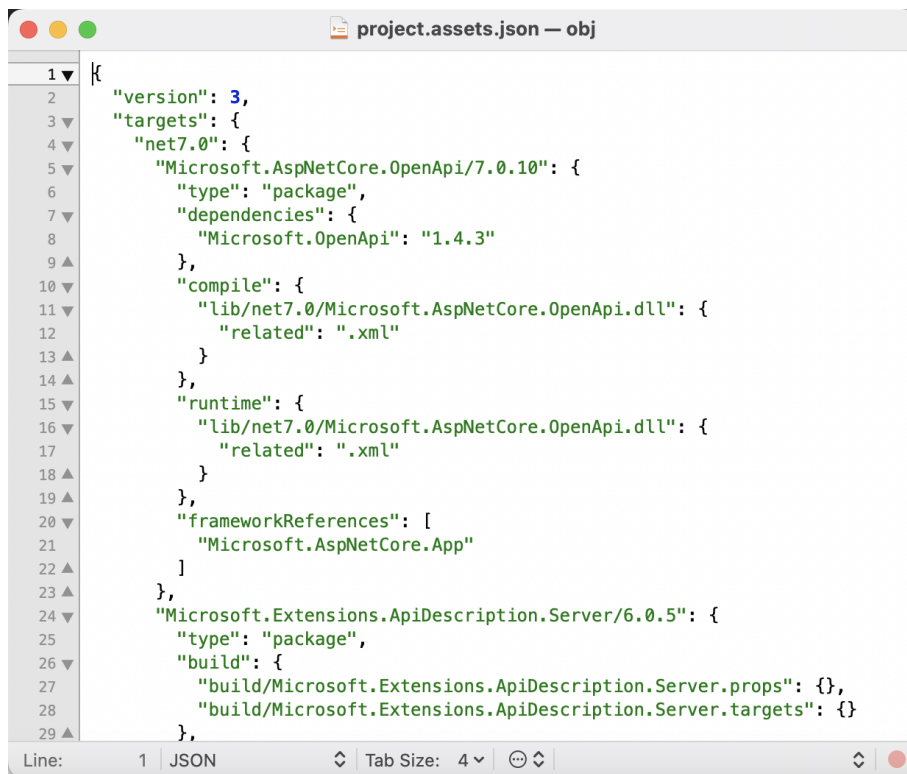
```
1 <?xml version='1.0' encoding='utf-8'>
2 <Project Sdk="Microsoft.NET.Sdk.Web">
3
4   <PropertyGroup>
5     <TargetFramework>net7.0</TargetFramework>
6     <Nullable>enable</Nullable>
7     <ImplicitUsings>enable</ImplicitUsings>
8   </PropertyGroup>
9
10  <ItemGroup>
11    <PackageReference Include="Microsoft.AspNetCore.OpenApi" Version="7.0.10" />
12    <PackageReference Include="Swashbuckle.AspNetCore" Version="6.5.0" />
13  </ItemGroup>
14
15 </Project>
```

Revisar el archivo obj/debug/project.assets.json

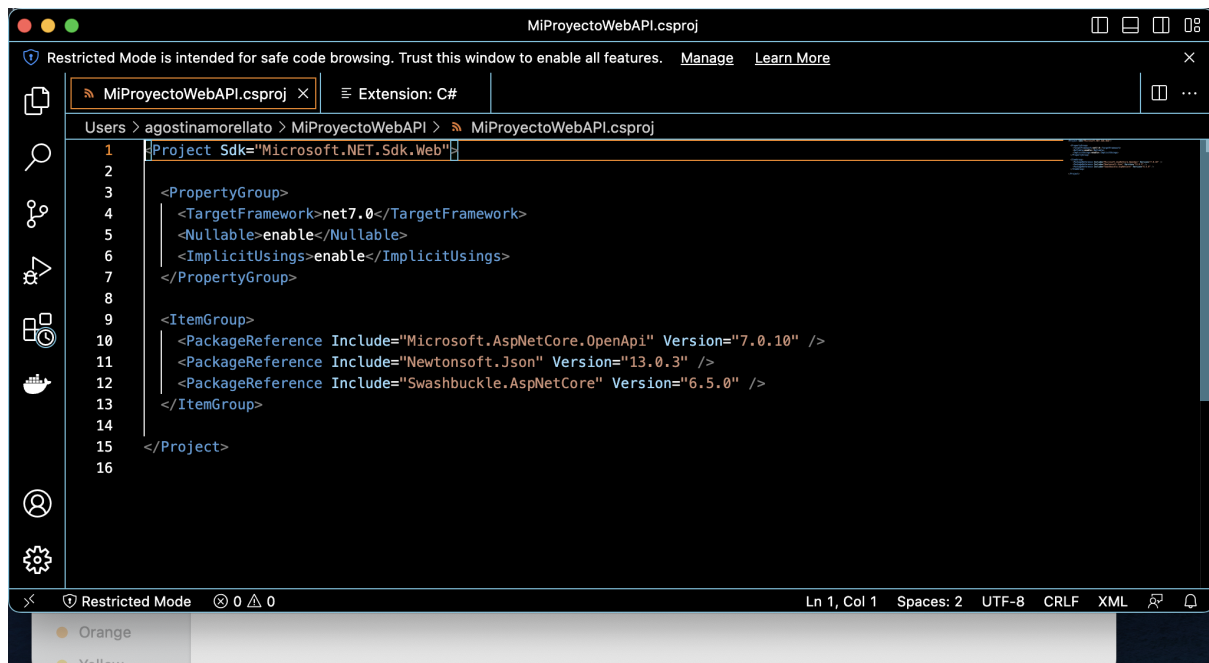
Borrar directorios bin y obj

Agregar una nueva referencia a librería Newtonsoft:

dotnet add package Newtonsoft.Json



```
1 {
2   "version": 3,
3   "targets": {
4     "net7.0": {
5       "Microsoft.AspNetCore.OpenApi/7.0.10": {
6         "type": "package",
7         "dependencies": {
8           "Microsoft.OpenApi": "1.4.3"
9         },
10        "compile": {
11          "lib/net7.0/Microsoft.AspNetCore.OpenApi.dll": {
12            "related": ".xml"
13          }
14        },
15        "runtime": {
16          "lib/net7.0/Microsoft.AspNetCore.OpenApi.dll": {
17            "related": ".xml"
18          }
19        },
20        "frameworkReferences": [
21          "Microsoft.AspNetCore.App"
22        ]
23      },
24      "Microsoft.Extensions.ApiDescription.Server/6.0.5": {
25        "type": "package",
26        "build": {
27          "build/Microsoft.Extensions.ApiDescription.Server.props": {},
28          "build/Microsoft.Extensions.ApiDescription.Server.targets": {}
29        }
30      }
31    }
32  }
```



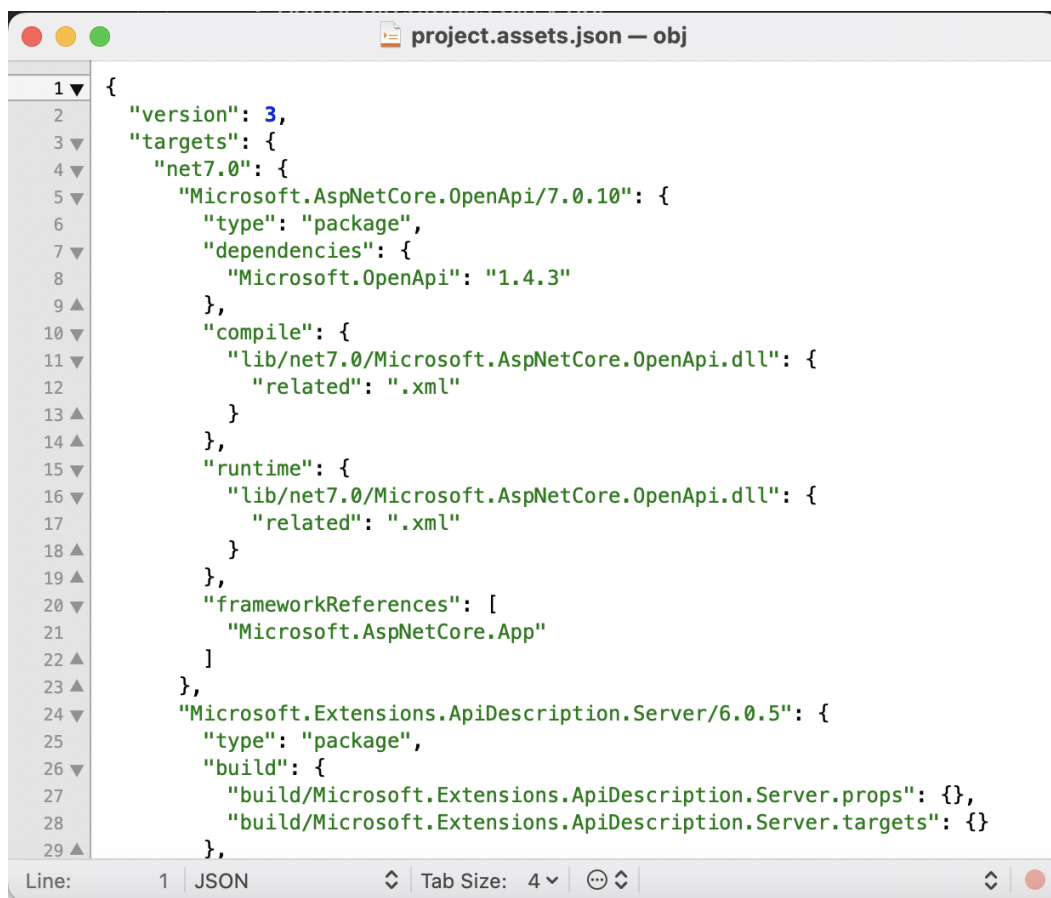
```
1 <?xml version='1.0' encoding='utf-8'>
2 <Project Sdk="Microsoft.NET.Sdk.Web">
3
4   <PropertyGroup>
5     <TargetFramework>net7.0</TargetFramework>
6     <Nullable>enable</Nullable>
7     <ImplicitUsings>enable</ImplicitUsings>
8   </PropertyGroup>
9
10  <ItemGroup>
11    <PackageReference Include="Microsoft.AspNetCore.OpenApi" Version="7.0.10" />
12    <PackageReference Include="Newtonsoft.Json" Version="13.0.3" />
13    <PackageReference Include="Swashbuckle.AspNetCore" Version="6.5.0" />
14  </ItemGroup>
15
16 </Project>
```

Revisar nuevamente los archivos MiProyectoWebAPI.csproj y obj/debug/project.assets.json

Borrar directorios bin y obj

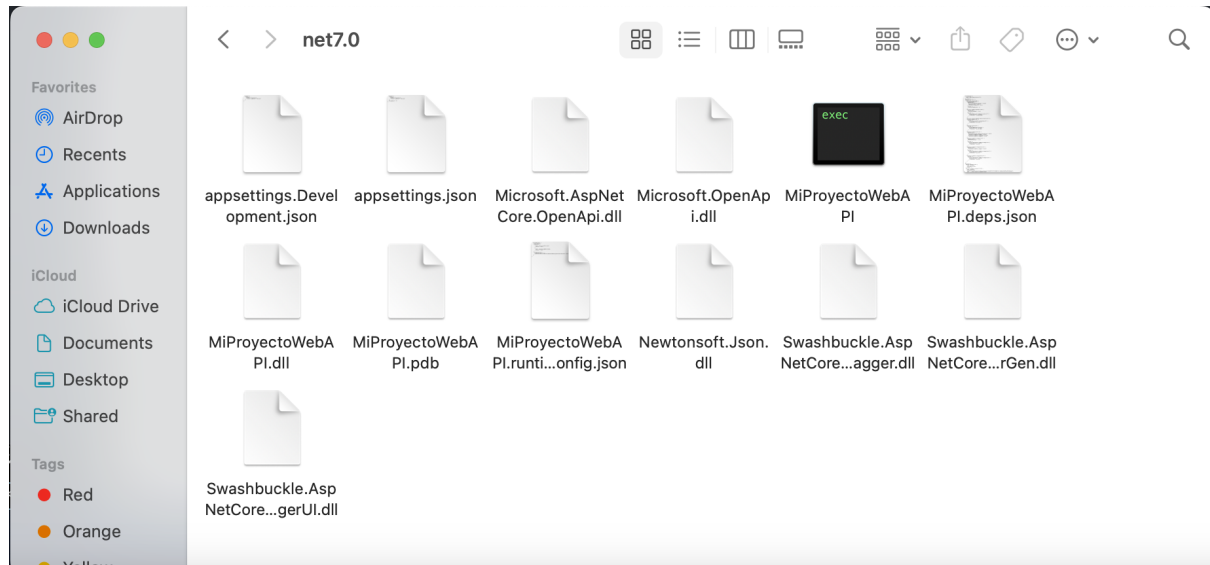
Ejecutar nuevamente:

dotnet run



```
1 {
2   "version": 3,
3   "targets": {
4     "net7.0": {
5       "Microsoft.AspNetCore.OpenApi/7.0.10": {
6         "type": "package",
7         "dependencies": {
8           "Microsoft.OpenApi": "1.4.3"
9         },
10        "compile": {
11          "lib/net7.0/Microsoft.AspNetCore.OpenApi.dll": {
12            "related": ".xml"
13          }
14        },
15        "runtime": {
16          "lib/net7.0/Microsoft.AspNetCore.OpenApi.dll": {
17            "related": ".xml"
18          }
19        },
20        "frameworkReferences": [
21          "Microsoft.AspNetCore.App"
22        ]
23      },
24      "Microsoft.Extensions.ApiDescription.Server/6.0.5": {
25        "type": "package",
26        "build": {
27          "build/Microsoft.Extensions.ApiDescription.Server.props": {},
28          "build/Microsoft.Extensions.ApiDescription.Server.targets": {}
29        }
30      }
31    }
32  }
```

Revisar contenido de directorio bin/debug/net7.0:



2- Ejemplo con nodejs

Instalar Nodejs: <https://nodejs.org/en/>

Crear una nueva aplicación

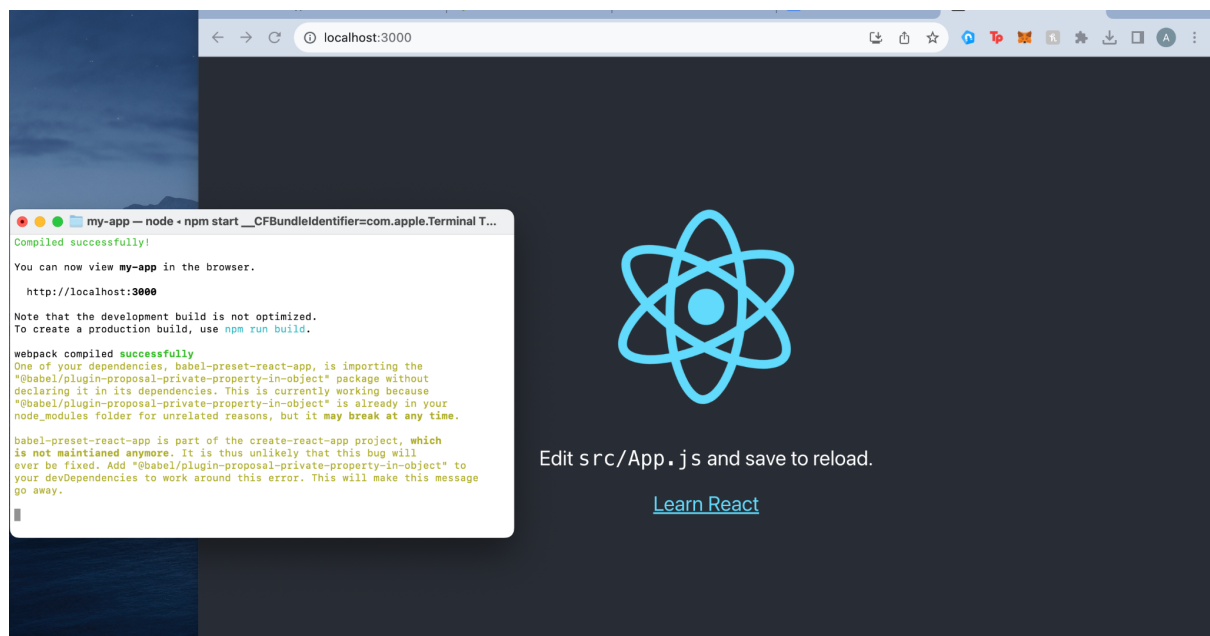
npx create-react-app my-app

Ejecutar la aplicación

cd my-app

npm start

La aplicación web estará disponible en **http://localhost:3000**



Analizar el manejo de paquetes y dependencias realizado por npm.

- Instalación de Node.js: Lo primero que se debe hacer es instalar Node.js, un entorno de tiempo de ejecución de JavaScript que incluye el gestor de paquetes npm (Node Package Manager).

- Creación de una nueva aplicación React:

Una vez que Node.js y npm están instalados, se utiliza el comando npx para crear una nueva aplicación de React. En este caso el comando fue ***npx create-react-app my-app***. Generando una estructura de directorios y archivos para la aplicación de React, además de instalar todas las dependencias necesarias.

- Instalación de dependencias: Cuando se ejecuta el comando create-react-app, se genera un archivo package.json que contiene una lista de dependencias requeridas para la aplicación de React. npm se encargará de descargar e instalar estas dependencias automáticamente.

Se instalan todas las dependencias, navegando al directorio de la aplicación (cd my-app) y se ejecuta:

npm install

Esto descarga todas las dependencias mencionadas en el archivo package.json y las almacenará en una carpeta llamada node_modules.

- Ejecución de la aplicación: Una vez que todas las dependencias se instalaron, inicia la aplicación con el comando:

npm start

Ejecutando un servidor de desarrollo y abriendo la aplicación en el navegador web.

La aplicación estará disponible en <http://localhost:3000>.

3- Build tools para otros lenguajes

Hacer una lista de herramientas de build (una o varias) para distintos lenguajes, por ejemplo (Rust -> cargo)

Rust:

- Cargo: La herramienta de construcción oficial de Rust. Se utiliza para compilar, administrar dependencias y ejecutar tareas de desarrollo en proyectos de Rust.

Java:

- Apache Maven: Una popular herramienta de construcción y administración de proyectos Java que utiliza un modelo de proyecto basado en XML (POM).

Python:

- setuptools: Un paquete de Python que proporciona funcionalidad para construir, empaquetar e instalar proyectos de Python.

JavaScript/Node.js:

- npm (Node Package Manager): La herramienta de gestión de paquetes y construcción estándar para proyectos de Node.js y JavaScript.

C/C++:

- Make: Una herramienta ampliamente utilizada para la construcción de proyectos en C y C++.
- CMake: Una herramienta de construcción multiplataforma que genera archivos de construcción para diferentes sistemas y entornos.

Go:

- go build: La herramienta de construcción integrada de Go que se utiliza para compilar programas Go directamente desde el código fuente.