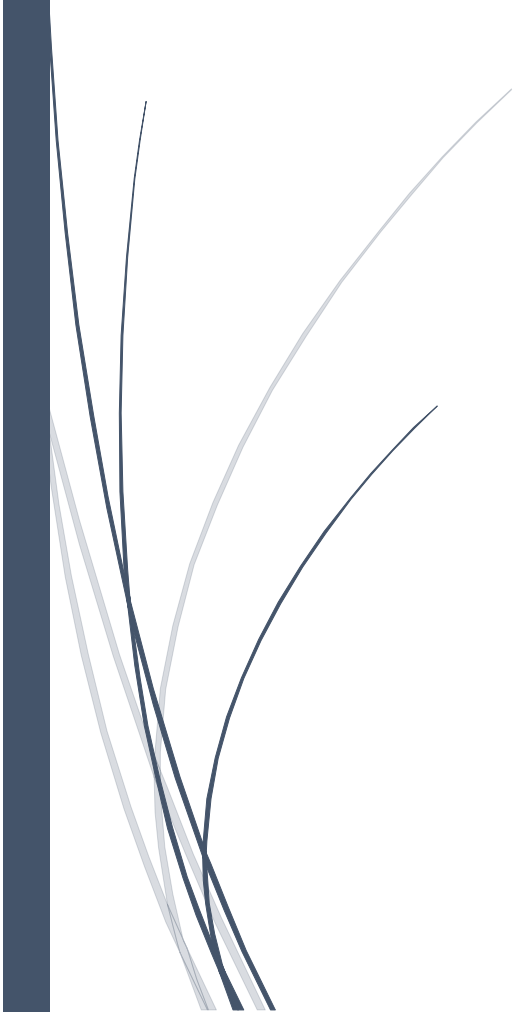
19-3-2020

Instructivo para la creación de Microservicios utilizando .NetCore



Arquitectura de Software

PORTALES DISTRIBUTORS.INC

Contenido

Inyección de Dependencias.

La inyección de dependencias (DI, por sus siglas en inglés) es un patrón usado en el diseño orientado a objetos de una aplicación. Es parte de uno de los cinco principios de diseño de clases conocido como S.O.L.I.D.

Como todo patrón de diseño, DI tiene como finalidad solucionar un problema común que los programadores encuentran en la construcción de aplicaciones. Este es, mantener los componentes o capas de una aplicación lo más desacopladas posible. Busca que sea mucho más sencillo reemplazar la implementación de un componente por otro. Así, evitar un gran cambio o impacto en la aplicación que pudiera originar que deje de funcionar por completo.

Para cumplir con dicho objetivo, DI nos permite inyectar comportamientos a componentes haciendo que nuestras piezas de software sean independientes y se comuniquen únicamente a través de una interface. Esto extrae responsabilidades a un componente para delegarlas en otro, estableciendo un mecanismo a través del cual el nuevo componente puede ser cambiado en tiempo de ejecución. Para lograr esta tarea DI se basa en un patrón más genérico llamado Inversión de Control (Inversion of Control).

Es necesario realizar ciertas modificaciones en el código fuente. Por ejemplo: el uso de interfaces que expongan la firmas de los métodos y operaciones de los que es responsable un componente, la eliminación de la instanciación de objetos o la necesidad de un modo de configuración que indique qué clases se instanciarán en el caso de solicitarlo.

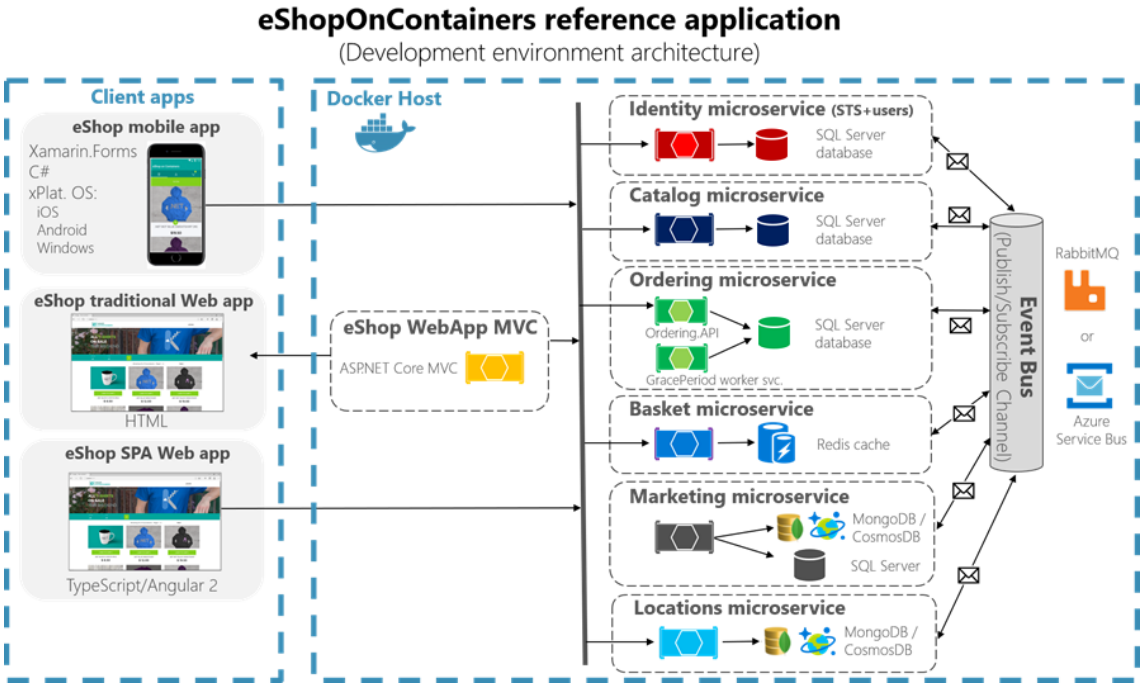
Al igual que cualquier técnica o herramienta, DI tiene sus ventajas e inconvenientes. Un uso indebido, pensando que nos va a solucionar nuestros problemas, se puede volver en nuestra contra creando una arquitectura compleja e innecesaria.

Microservicios

Una “arquitectura de microservicios” es un enfoque para desarrollar una aplicación software como una serie de pequeños servicios, cada uno ejecutándose de forma autónoma y comunicándose entre sí, por ejemplo, a través de peticiones HTTP a sus API. Normalmente hay un número mínimo de servicios que gestionan cosas comunes para los demás (como el acceso a base de datos), pero cada microservicio es pequeño y corresponde a un área de negocio de la aplicación.

Además, cada uno es independiente y su código debe poder ser desplegado sin afectar a los demás. Incluso cada uno de ellos puede escribirse en un lenguaje de programación diferente, ya que solo exponen la API (una interfaz común, a la que le da igual el lenguaje de programación en la que el microservicio esté programado por debajo) al resto de

microservicios.



.NET Core

.NET Core es una plataforma de desarrollo de uso general de código abierto de cuyo mantenimiento se encargan Microsoft y la comunidad .NET en GitHub. Es multiplataforma, admite Windows, macOS y Linux y puede usarse para compilar aplicaciones de dispositivo, nube e IoT.

Características

- **Multiplataforma**: se ejecuta en los sistemas operativos Windows, macOS y Linux.
- **Coherente entre arquitecturas**: el código se ejecuta con el mismo comportamiento en varias arquitecturas, como x64, x86 y ARM.
- **Herramientas de línea de comandos**: incluye herramientas de línea de comandos sencillas que se pueden usar para el desarrollo local y en escenarios de integración continua.
- **Implementación flexible:** Se pueden incluir en la aplicación o se pueden instalar de forma paralela (instalaciones a nivel de usuario o de sistema). Se puede usar con contenedores de Docker.
- **Compatible**: .NET Core es compatible con .NET Framework, Xamarin y Mono mediante .NET Standard.
- **Código abierto**: la plataforma .NET Core es de código abierto, con licencias de MIT y Apache 2. .NET Core es un proyecto de .NET Foundation.
- **Compatible con Microsoft**: .NET Core incluye compatibilidad con Microsoft, como se indica en .NET Core Support (Compatibilidad de .NET Core).

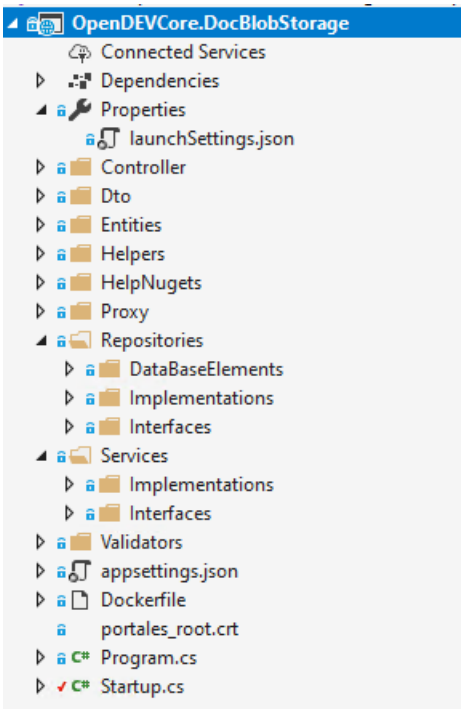
Herramientas Necesarias

Para realizar el desarrollo de microservicios es necesario al menos de las siguientes herramientas:

- SDK .net Core 3.1.101
- Visual Studio 16.4.X o superior

Estructura del Proyecto.

El proyecto contendrá la siguiente estructura de directorios.



Cada uno de esto directorios contendrá los siguientes archivos:

- **Controller:** se colocan todos los controladores, que son encargados de exponer los servicios para que estos sean consumidos desde otro microservicio
- **Dto:** Se colocan los Dtos que los mismos que interactúan con los servicios.
- **Entities:** Se colocan todas las clases que representan a cada una de las tablas de la BDD estas clases pueden ser generadas mediante el proceso de Scaffold recordando que llevaremos el esquema “Database First”.
- **Helpers:** Se colocan las clases de Automappers y los manejadores de Excepciones
- **Proxy:** Si el microservicio se integra con otros para compartir información dentro de este directorio se colocan las interfaces con las cuales se integrará
- **Repositories:**
    - **DataBaseElements:** Se coloca el context de la base de datos que se generó del scafold este archivo contiene la configuración para cada una de las tablas.
    - **Implementations:** Se colocan las implementaciones de las interfaces de repositorio estas implementaciones contienen la lógica de interacción

con la base de datos. Recordar que estas clases interactúan con las clases que están dentro del directorio de Entities
    - **Interfaces:** Contiene las interfaces para cada uno de los repositorios.
- **Services**
    - **Implementations:** Se colocan las implementaciones de las interfaces de servicios estas clases tienen interacción con los Dtos por tanto hay que realizar el mapeo correspondiente además se debe realizar la inyección de dependencias de los repositorios y Automapper
    - **Interfaces:** Contiene las interfaces de cada uno de los servicios dentro de cada interfaz se definen las firmas de los métodos
- **Validators:** Se registra los controles que van a tener cada Dto's de tenerlos.

Archivos de importancia

- **launchSettings.json:** archivo de Propiedades está en el apartado de properties el icono que tiene una llave en este archivo se define el ambiente de ejecución el cual puede ser QA, DEVELOPMENT, LOCAL trabajar con development si el microservicio está en desarrollo aquí también se define en que IP y puerto el servicio será ejecutado.
- **appsetings.json**: tiene toda la configuración del servicio como el nombre de la app configuración de conexión hacia la BDD, Camunda, rabitMQ y restease este último almacena la información de los servicios con los que se integra el microservicio.
- **Program.cs:** encargado de levantar todo no se lo debe modificar a no ser que requiera alguna configuración extra.
- **Startup.cs:** es el archivo donde se registrara configuración necesaria del contenedor de .NetCore además es donde se registra la inyección de dependencias se lo puede realizar de la siguiente manera .

INYECCION DE DEPENDENCIAS:

services.AddScoped<INTERFAZ, IMPLEMENTACION>();

REGISTRAR VALIDADORES

services.AddSingleton<IValidator<NOMBREDTO>, CLASEDELVALIDADOR>();

REGISTRAR INTEGRACION CON OTROS SERVICIOS

services.RegisterServiceForwarder<nombreInterfazEnCarpetaIproxy>("nombre de la app definido en restEase(appsetingsjson");

Name Spaces

Considerar la siguiente estructura de namespaces que se debe llevar para cada uno de las clases creadas

- namespace OpenDEVCore.nombreProyecto: Para el context de la base.

- namespace OpenDEVCore.nombreProyecto.Repositories: Para los repositorios tanto interfaces como implementations
- namespace OpenDEVCore.nombreProyecto.Services : Para los repositorios tanto interfaces como implementations
- namespace OpenDEVCore.nombreProyecto.Entities: Para todas las entidades
- namespace OpenDEVCore.nombreProyecto.Dtos: Para todos los dtos