# Preparación del ambiente

Aplicaciones y tareas a realizar para comenzar a trabajar.

## Instalación de aplicaciones

Para facilitar la instalación (y explicación) dejo lo necesario para instalar con Chocolatey. O pueden instalar manualmente.

<https://chocolatey.org/install>

Instalar:

* **Git**: choco install git.install
* **Visual Studio a elección**:
  + **VS2019 Community**: choco install visualstudio2019community
  + **VS2019 Profesional**: choco install visualstudio2019professional
  + **VS2019 Enterprise**: choco install visualstudio2019enterprise
* **SqlServer2019 Express**: choco install sql-server-express
* **Nuget CLI**: choco install nuget.commandline

## Configuraciones

Para bajar dependencias de malone.Core que están alojadas en Github Package Registry hay que generar un Token como propietario que van a tener que registrar en nuget CLI con el siguiente comando (cerrar el VS):

nuget sources Add -Name "github" -Source "<https://nuget.pkg.github.com/malonejv/index.json>" -UserName malonejv -Password TOKEN

Ese Token queda guardado en:

%userprofile%\AppData\Roaming\NuGet\NuGet.Config

# Estructura del proyecto

Servicios:

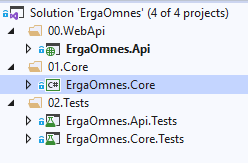
* WebApi

Core de la Aplicación

* Componentes de negocio
* Componentes de acceso a datos
* Componentes trasversales: Manejo de Excepciones, Login, etc.

Proyectos de pruebas unitarias

* Tests de Web Api
* Tests de Core



## Proyecto Core

### Inicializadores

Aquí se va a detallar la configuración para el framework de inyección de dependencias, en este caso Unity. Bajo el namespace Initializers vamos a tener las siguientes clases:

* CLInitializer 🡪 Configura los componentes trasversales.
* ELInitializer 🡪 Configura las entidades de negocio.
* DALInitializer 🡪 Configura el contexto y repositorios.
* BLInitializer 🡪 Configura los componentes de negocio y validadores.
* ErgaOmnesInitializer 🡪 Dispara la inicialización de cada capa.

### Service Layer (SL)

Texto

### Business Layer (BL)

Texto

### Data Access Layer (DAL)

### Entities Layer (EL)

### Common Layer (CL)

## Proyecto Web API

### Configuración

1. //Descomentar si no queremos devolver XML nunca
2. //config.Formatters.Remove(config.Formatters.XmlFormatter);
4. //Definimos el formato JSON por defecto
5. config.Formatters.JsonFormatter.SerializerSettings.Formatting = Newtonsoft.Json.Formatting.Indented;
6. config.Formatters.JsonFormatter.SerializerSettings.ContractResolver = **new** CamelCasePropertyNamesContractResolver();

#### Cross Domain Request

Por seguridad los browsers no permiten las consultas por Ajax desde distintos dominios.

Para resolver este problema existen 2 alternativas:

* Usar JSONP
* Habilitar CORS

Para usar JSONP hay que:

1. Instalar este paquete de nuget: Install-package WebApiContrib.Formatting.Jsonp
2. Agregar esta configuracion en el archivo WebApiConfig.cs:

var jsonpFormatters = **new** JsonpMediaTypeFormatter(config.Formatters.JsonFormatter);

config.Formatters.Insert(0, jsonpFormatters);  

1. En la consulta Ajax poner dataType = ‘jsonp’ en lugar de json.

Para habilitar CORS hay que:

1. Instalar este paquete de nuget: Install-package Microsoft.AspNet.WebApi.Cors
2. Agregar esta configuracion en el archivo WebApiConfig.cs:

//EnableCorsAttribute enableCorsAttribute = new EnableCorsAttribute("http://url-frontend", "\*", "\*");

EnableCorsAttribute enableCorsAttribute = **new** EnableCorsAttribute("\*", "\*", "\*");

config.EnableCors(enableCorsAttribute);

Cual usar entonces:

CORS puede usarse como una alternativa moderna al patrón JSONP . Los beneficios de CORS son:

* Si bien JSONP solo admite el método de solicitud GET, CORS también admite otros tipos de solicitudes HTTP.
* CORS permite que un programador web use XMLHttpRequest, lo cual admite un mejor manejo de errores que JSONP.
* Si bien JSONP puede causar problemas de secuencias de comandos entre sitios (XSS) cuando el sitio externo se ve comprometido, CORS permite que los sitios web analicen manualmente las respuestas para aumentar la seguridad.

La principal ventaja de JSONP es su capacidad para trabajar en navegadores heredados que son anteriores a la compatibilidad con CORS ( Opera Mini e Internet Explorer 9 y versiones anteriores). CORS ahora es compatible con la mayoría de los navegadores web modernos.

### Global.asax

Agregar:

AppInitializer<UnityWebApiActivator, IUnityContainer, ErgaOmnesInitializer>.Initialize();

### Microsoft.Aspnet.Identity

Cuando se crea un proyecto con Microsoft.Aspnet.Identity, el template genera un archivo IdentityConfig.cs y IdentityModels.cs que contiene distintas clases que utiliza el framework para administrar autenticación.

En malone.Core se están reutilizando esas clases en el componente malone.Core.Identity y malone.Core.Identity.EntityFramework. Con lo cual hay que excluir esos archivos y modificar los archivos Startup.Auth.cs, AccountController.cs y ManageController.cs.  
En todos estos archivos hay que modificar las ocurrencias de ApplicationUserManager por UserBusinessComponent, ApplicationSignInManager por SignInBusinessComponent y ApplicationUser por CoreUser. Además ApplicationUser utilizar un Id de tipo string, por lo que hay que modificar User.Identity.GetUserId() por User.Identity.GetUserId<int>().

## Proyectos de pruebas unitarias

# Ejemplo de uso

Creo entidad Ejemplo en ErgaOmnes.Core.EL.Model. La entidad debe implementar IBaseEntity o IBaseEntity<TKey>.

Si quiero habilitar el borrado lógico, implementar también ISoftDelete.