**IDMS vNext code set up and asp.net zero best practices**

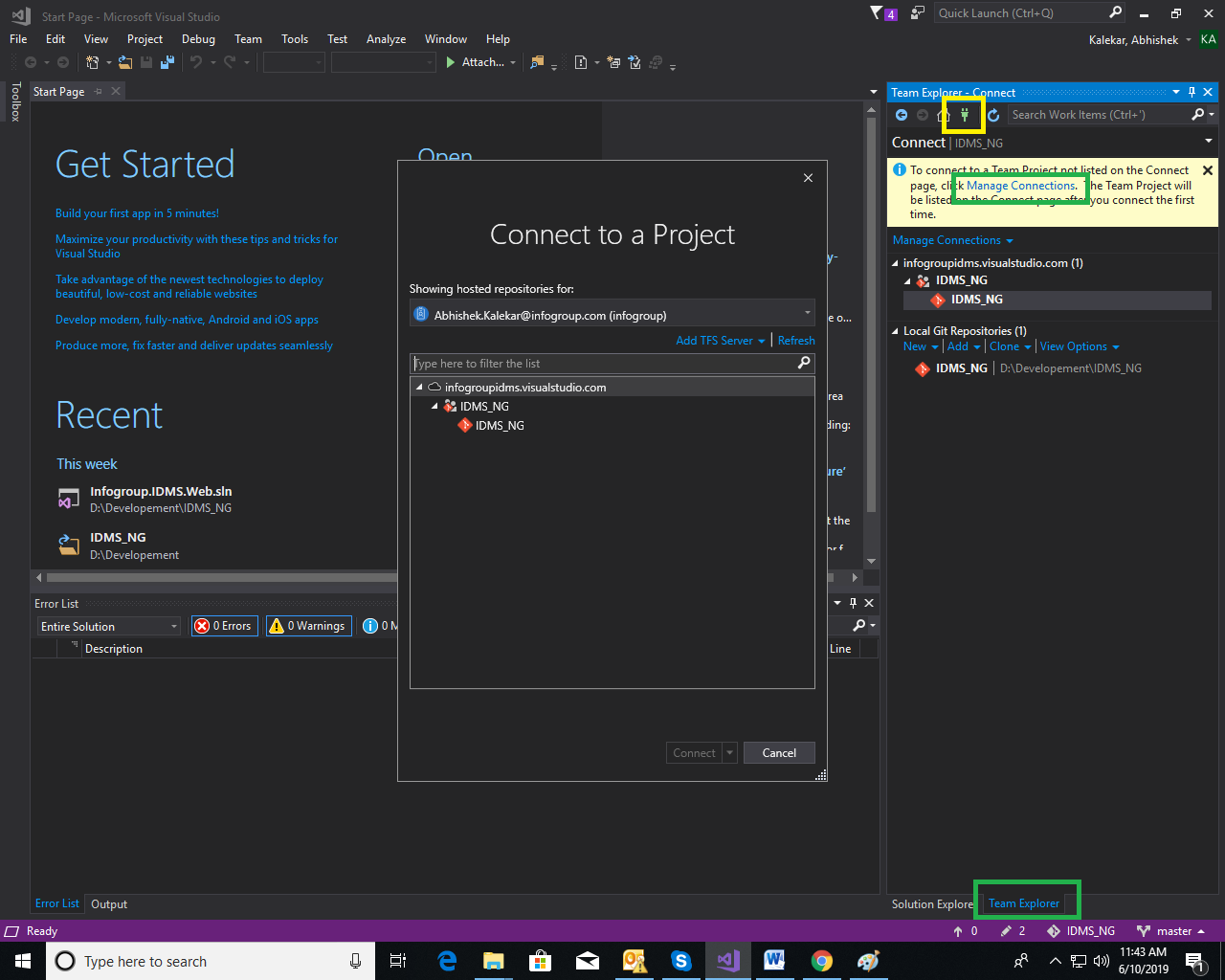
**Prerequisites:**

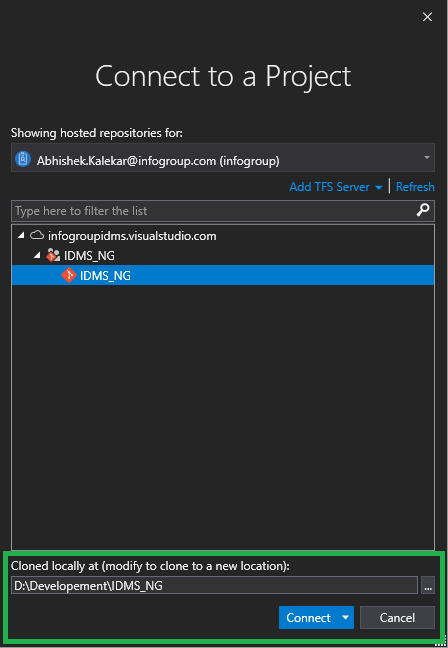
1. Install Visual studio 2017.
2. Install VIP Access.
3. Install Cisco AnyConnect
4. Install Node JS
5. Install Angular CLI
6. Install Yarn package.
7. If not installed already, install latest .Net core version.

**Code setup:**

1. Create folder “**Development**” under “**D:**” drive. Then, create “**IDMS\_NG**” folder under this folder.
2. Open **Visual Studio 2017**. Head to “**Team Explorer**” tab. Then click on “**Manage Connections**” button.

Then, open “[**https://infogroupidms.visualstudio.com/\_git/IDMS\_NG**](https://infogroupidms.visualstudio.com/_git/IDMS_NG)”.

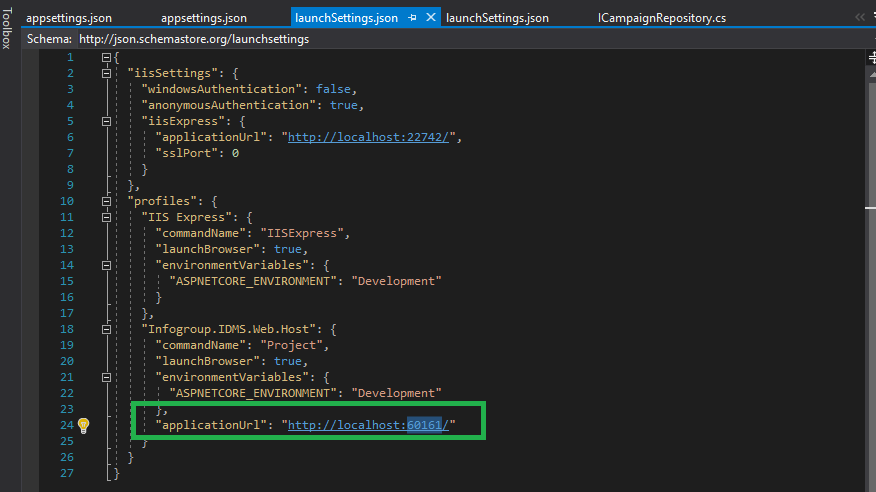




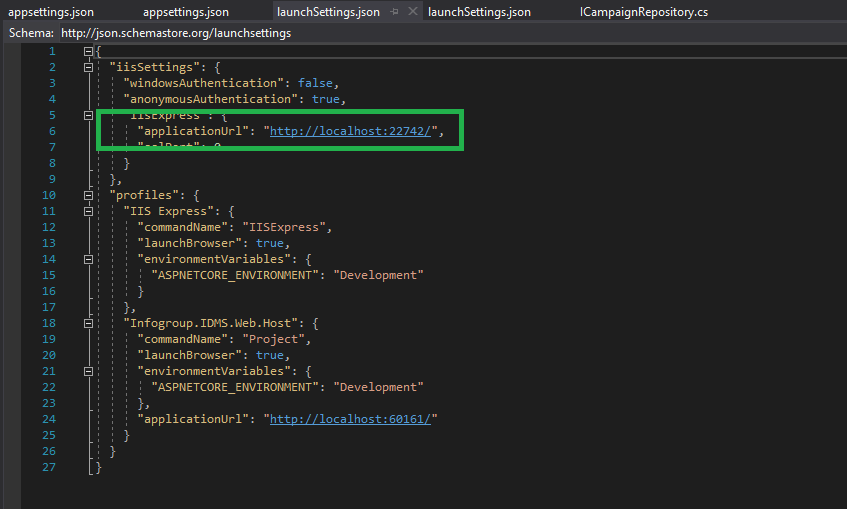
1. Login with valid credentials and click on “**Connect**” button. This will connect to code repository of **IDMS\_NG**.
2. After successful connection, clone code on local directory from repository.
3. Select local path as “**D:\Development\IDMS\_NG**” in Azure DevOps popup and click on connect.
4. For running the “**Web API**” project, open the “**Infogroup.IDMS.Web.sln**” in VS2017. Set the “**Infogroup.IDMS.Web.Host**” project as startup project.

Also, we can run “Web API” project using command i.e. “**dotnet run**”. Go to command prompt set directory as “**D:\Developement\IDMS\_NG\src\Infogroup.IDMS.Web.Host**” and execute “**dotnet run**” command.

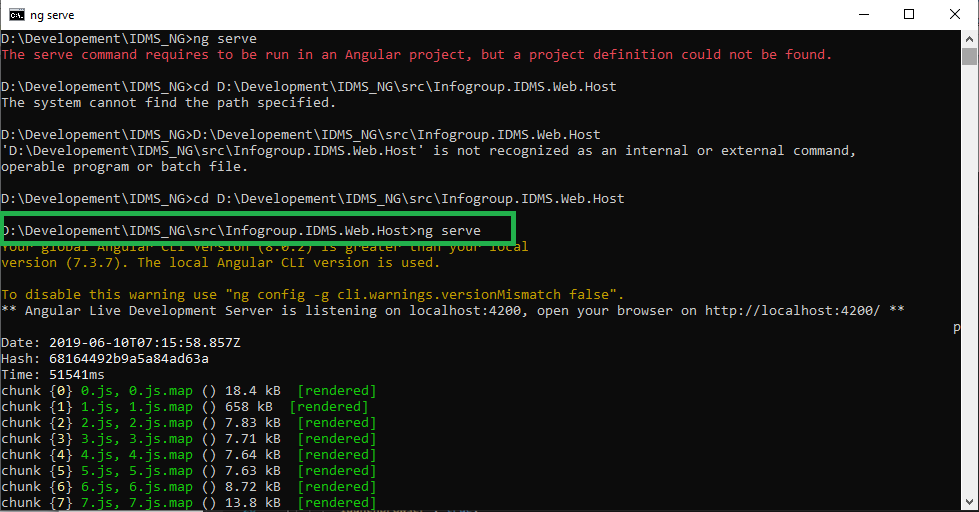
In case to change port for running application using “dotnet run”, update port in “**launchSettings.json”.**

****

1. Before running project, please install following on machine.
   1. Install **Node JS** from “[**https://nodejs.org/en/**](https://nodejs.org/en/)” website. Install **LTS** version of Node Js.
   2. Then open command prompt. Set path as "**D:\Development\IDMS\_NG**". Install Angular CLI by running given command “**npm install -g @angular/cli**”
   3. Install “**Yarn**” by executing given command “**npm install yarn –g**” and then run “**Yarn**” by executing command “**yarn**”.
2. Build code. If any .net core related errors are found then install latest .net core.
3. Make sure port in “**appsettings.json**” in “**Infogroup.IDMS.Web.Host**” matches with “**launchSettings.json**”. If not update the port in “**launchSettings.json**”.



1. Run host application. It will open in desired browser with link “[**http://localhost:22742/**](http://localhost:22742/)”
2. For running Angular project, open project “**D:\Development\IDMS\_NG\src\Infogroup.IDMS.Web.Host**” in command prompt and type “**ng serve**” command.



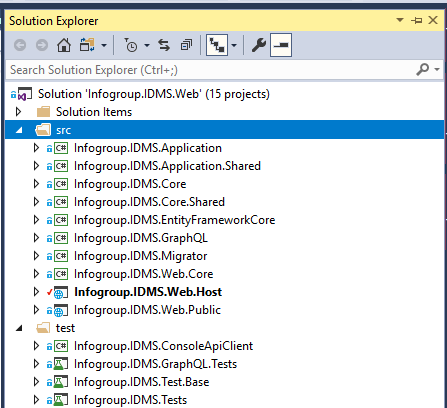
1. Run angular application. It will open in desired browser with link “[**http://localhost:4200/**](http://localhost:4200/)”

Aspnetzero over view and project structure:

The Angular-asp.net core version of Aspnetzero framework contains 3 solutions. The \*.Web.sln solution contains all the web development projects. The \*.Mobile.sln contains only Xamarin application development projects and the \*.All.sln contains both mobile and web development projects.

For IDMS\_vNext we shall be currently focusing only on the Infogroup.IDMS.Web.sln solution.

Below is the Infogroup.IDMS.Web.sln structure:



There are 12 projects in the solution:

* **Core.Shared** project contains consts, enums and helper classes used both in mobile & web projects.
* **Core** project contains domain layer classes (like [entities](https://aspnetboilerplate.com/Pages/Documents/Entities) and [domain services](https://aspnetboilerplate.com/Pages/Documents/Domain-Services)).
* **Application.Shared** project contains [application service interfaces](https://aspnetboilerplate.com/Pages/Documents/Application-Services#DocIApplicationServiceInterface) and [DTO](https://aspnetboilerplate.com/Pages/Documents/Data-Transfer-Objects)s.
* **Application** project contains application logic (like [application services](https://aspnetboilerplate.com/Pages/Documents/Application-Services)).
* **EntityFrameworkCore** project contains your DbContext, [repository](https://aspnetboilerplate.com/Pages/Documents/Repositories)implementations, database migrations and other Entity Framework Core specific concepts.
* **Web.Host** project does not contain any web related files like HTML, CSS or JS. Instead, it just serves the application as remote API. So, any device can consume your application as API. For more information see [Web.Host Project](https://docs.aspnetzero.com/documents/aspnet-core-angular/latest/Features-Mvc-Core-Web-Host-Project)
* **Web.Core** project contains common classes used by MVC and Host projects.
* **Web.Public** project is a separated web application that can be used to create a public web site or a landing page for your application. For more information see [Public Website](https://docs.aspnetzero.com/documents/aspnet-core-angular/latest/Public-Website).
* **Migrator** project is a console application that runs database migrations. For more information see [Migrator Console Application](https://docs.aspnetzero.com/documents/aspnet-core-angular/latest/Migrator-Console-Application)
* **ConsoleApiClient** project is a simple console application for performing API requests to the application authenticated via IdentityServer4.
* **Tests** project contains unit and integration tests.

Applications

ASP.NET Zero solution contains three applications:

* **Back End API** (Web.Host): An application to only serve the main application as REST API and does not provide any UI.
* **Public Web Site** (Web.Public): This can be used to create a public web site or a landing page for your application.
* **Migration Executer** (Migrator): Console application that runs database migrations.

Basic Configuration

appsettings.json in Web.Host project contains many settings but **ServerRootAddress**, **ClientRootAddress** and **CorsOrigins** are required to run the application:

"ServerRootAddress": "http://localhost:22742/",

"ClientRootAddress ": "http://localhost:4200/",

"CorsOrigins": "http://localhost:4200/"

**ServerRootAddress** is the URL of the server side Web.Host application, **ClientRootAddress** is the URL of the client side Angular application. **CorsOrigins** contains the URLs allowed to make cross-origin requests to Web.Host application.

### Angular Solution

Entry point of the Angular solution is src/**main.ts**. It simply bootstraps the root [Angular Module](https://angular.io/docs/ts/latest/guide/ngmodule.html): **RootModule**. Fundamental modules of the solution are shown below:

Angular 2 modules

* **RootModule** is responsible to bootstrap the application.
* **AccountModule** provides login, two factor authentication, register, password forget/reset, email activation, etc... It's [lazy loaded](https://angular.io/docs/ts/latest/guide/router.html).
* **AppModule** is just to group application modules and provide a base layout. It contains two sub modules:
  + **AdminModule** contains pages like user management, role management, tenant management, language management, settings and so on. It's lazy loaded.
  + **MainModule** is the main module to develop your own application. It only contains a demo dashboard page which you can modify or delete. It's suggested to divide your application into smaller modules like we did in the startup project, instead of adding all functionality into the main module. This is also lazy loaded.

Fundamental modules have their own **routes**. For example; AccountModule views start with "**/account**" (like "/account/login"), AdminModule views starts with**/app/admin**" (like "/app/admin/users").

Angular's router lazy loads modules based on their url. For instance, when you request a url starts with "app/admin", the AdminModule and all it's components are loaded. They are not loaded if you don't request those pages. That brings better startup time (and also better development time since they are independently splitted to chunks).

In addition to those fundamental modules, there are some shared modules:

* app/shared/common/**app-common.module**: a common module used by main and admin modules as shared functionality.
* shared/common/**common.module**: A common module used by account and app modules (and their sub modules).
* shared/utils/**utils.module**: Another common module used by all modules (and their sub modules). We tried to collect general purpose code here those can be used even in different applications.
* shared/service-proxies/**service-proxy.module**: Auto generated nswagcode. It's used to communicate to backend ASP.NET Core API. We will see "how to generate automatic proxies" later.

#### Configuration

Angular solution contains src/assets/**appconfig.json** file which contains some fundamental settings for the client side:

* **remoteServiceBaseUrl**: Used to configure base address of the server side APIs. Default value: http://localhost:22742
* **appBaseUrl**: Used to configure base address of the client application. Default value: http://localhost:4200
* **localeMappings**: Used to configure localizations of third-party libraries those are incompatible with existing localizations.

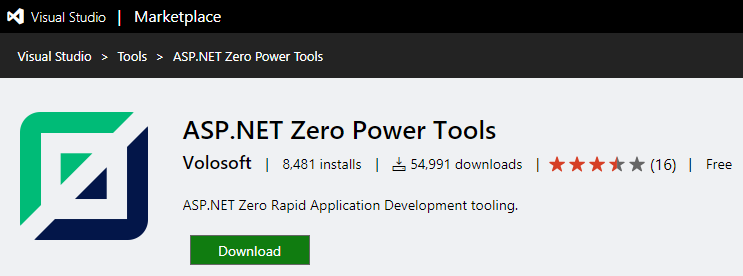
**appBaseUrl** is configured since we use it to define format of our URL.

#### AppComponentBase

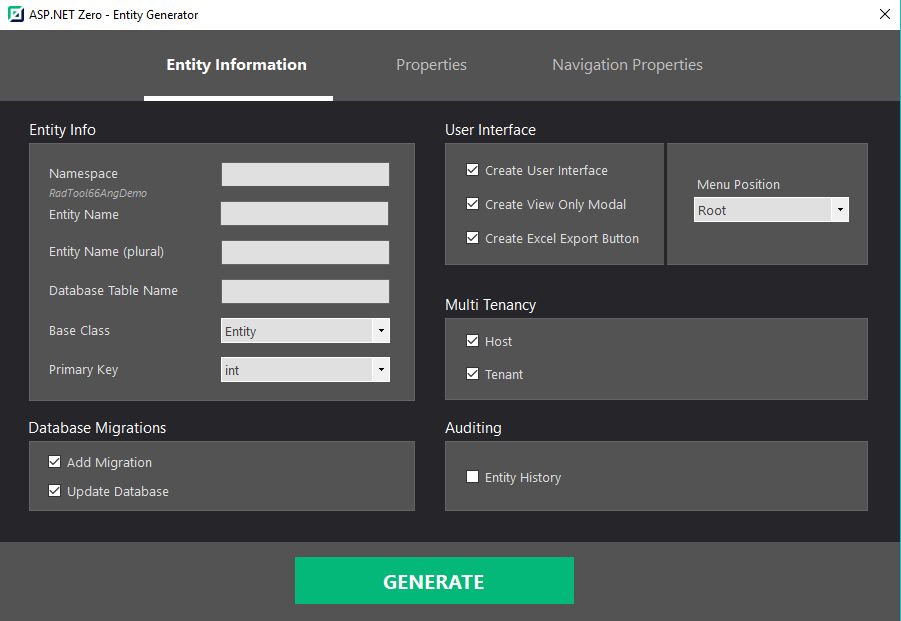
If you inherit your components from **AppComponentBase** class, you can get many commonly used services as pre-injected (like localization, permission checker, feature checker, UI notify/message, settings and so on...). For example; you can just use **this.l(...)** function in component classes for localization. In views, you can use **localize** pipe. See pre-built components for example usages.

Aspnet RAD tools:

[ASP.NET Zero Power Tools](https://marketplace.visualstudio.com/items?itemName=Volosoft.AspNetZeroPowerTools) is a Visual Studio Extension that can automatically create an entity and its layers from the database to the user interface. So, you can create a fully working, production ready CRUD page in seconds.

[](https://marketplace.visualstudio.com/items?itemName=Volosoft.AspNetZeroPowerTools)

The extension can be found inside the **Tools** menu (Tools -> Asp.Net Zero -> Create An Entity). When you run it, you will see the interface for creating an entity. After carefully filling out the fields, press the **Generate** button to start the code generation process.



A simple console will appear and give you information about the process. If there is no warning or failure, run your project to see the results. If you don't see the new page on UI, grant yourself the required **permissions** in the application.

Warning: If you are working on ASP.NET Core & Angular template, after generating the entity via Power Tools, run your \***.Web.Host** project and then run "**./angular/nswag/refresh.bat**" to update **service-proxies.ts**.

Warning: Be sure that you have saved your work before running this tool since it will add new files and modify some of the existing files. We strongly recommend using a source control system (like Git). Otherwise, backup your project.

## Generated Files

Here is the full list of the files that are created or modified by the tool, if you give a basic "Campaign" entity as input.

### Server Side

**Created**

* Campaign.cs
* Campaign Dto.cs
* LookupDto.cs
* GetAllForLookupTableInput.cs
* GetCampaignForEditOutput.cs
* GetAllCampaign sOutput.cs
* CreateOrEditCampaign Dto.cs
* GetAllCampaign sInput.cs
* CampaignConsts.cs
* ICampaign AppService
* CampaignAppService
* CampaignsExcelExporter.cs
* ICampaignsExcelExporter.cs

**Modified**

* AppAuthorizationProvider.cs
* AppPermissions.cs
* ProjectNameDbContext.cs
* CustomDtoMapper.cs
* ProjectName.xml (English localization file)

(Optionally, adds a database migration and updates the database.)

### Client Side

#### Angular

**Created**

* campaigns.component.ts
* campaigns.component.html
* create-or-edit-campaign-modal.component.ts
* create-or-edit-campaign-modal.component.html
* Lookup-Table-modal.component.ts
* Lookup-Table-modal.component.html
* Lookup-Table-modal.component.less

**Modified**

* app-navigation.service.ts
* service-proxy.module.ts
* (Main or Admin)-routing.module.ts
* (Main or Admin).module.ts

Testing the API using swagger:

We can enable and test the created api’s by setting up swagger.

1. GOTO Infogroup.IDMS.Web.Core\Common\WebConts.cs and check if SwaggerUiEnabled is set to true
2. GOTO Infogroup.IDMS.Web.Host\StartUp.cs and check the Configure and ConfigureServies method for the swagger UI settings.
3. Complie the code and run the Host application
4. That's it! Let's browse to **/swagger**

