Implement Sitecore 9 Project from scratch Using Helix

Helix was introduced by Sitecore in 2016 as a set of official guidelines and recommended practices for Sitecore Development. Since then, the community has been supporting the initiative and many websites have been built using this approach. A well-known website that was built to demonstrate in practice the Helix principles and guideline was the Habitat.

This blog walks you through how to Setup a Sitecore Project from scratch using Helix principles and Habitat resources. The main goal it's to show how we can start from a new Sitecore Project from scratch using existing resources (*Scripts*, *Configs files*, and some *Foundations* and *Feature* modules) already being implemented. This blog does not detailed topics of the foundation layer/modules such as *SitecoreExtensions*, *Metadata*, *Dictionary* or gulp scripts. The goal of this blog is to show how to take advantage of those pre-built scripts and modules to start your own Sitecore project using Helix. Then you are good to go ahead and develop any module in any those layers to support your business model.

Environment Specifications	
Website host name	XCNine.local
Project root	C:\projects\XCNine
Sitecore Version	9.0.171002
Instance root folder	C:\\inetpub\\wwwroot\\XCNine.Local
Publish / Website URL	http://www.xcnine.local

If you need help on how to install Sitecore, visit my blog Sitecore 9 - Setup Environment before Installation. Assuming at this point that you already have installed a clean Sitecore 9 instance, I will describe step-by-step how to set up your project.

LAYERS: XCNine Physical & Solution Folder Structure

Open the Visual Studio (VS 2017) and create a blank solution inside of the root folder (ex. C:\projects\XCNine). Then, create the ***Src*** folder and inside of it create the three layers: ***Feature***, ***Foundation,*** and ***Projects***. From the Visual Studio (VS) create four (4) solution folder ***Configurations***, ***Feature***, ***Foundation***, ***Project***. At this point, I am

expecting you to have the base structure in order to start adding your first project/module.

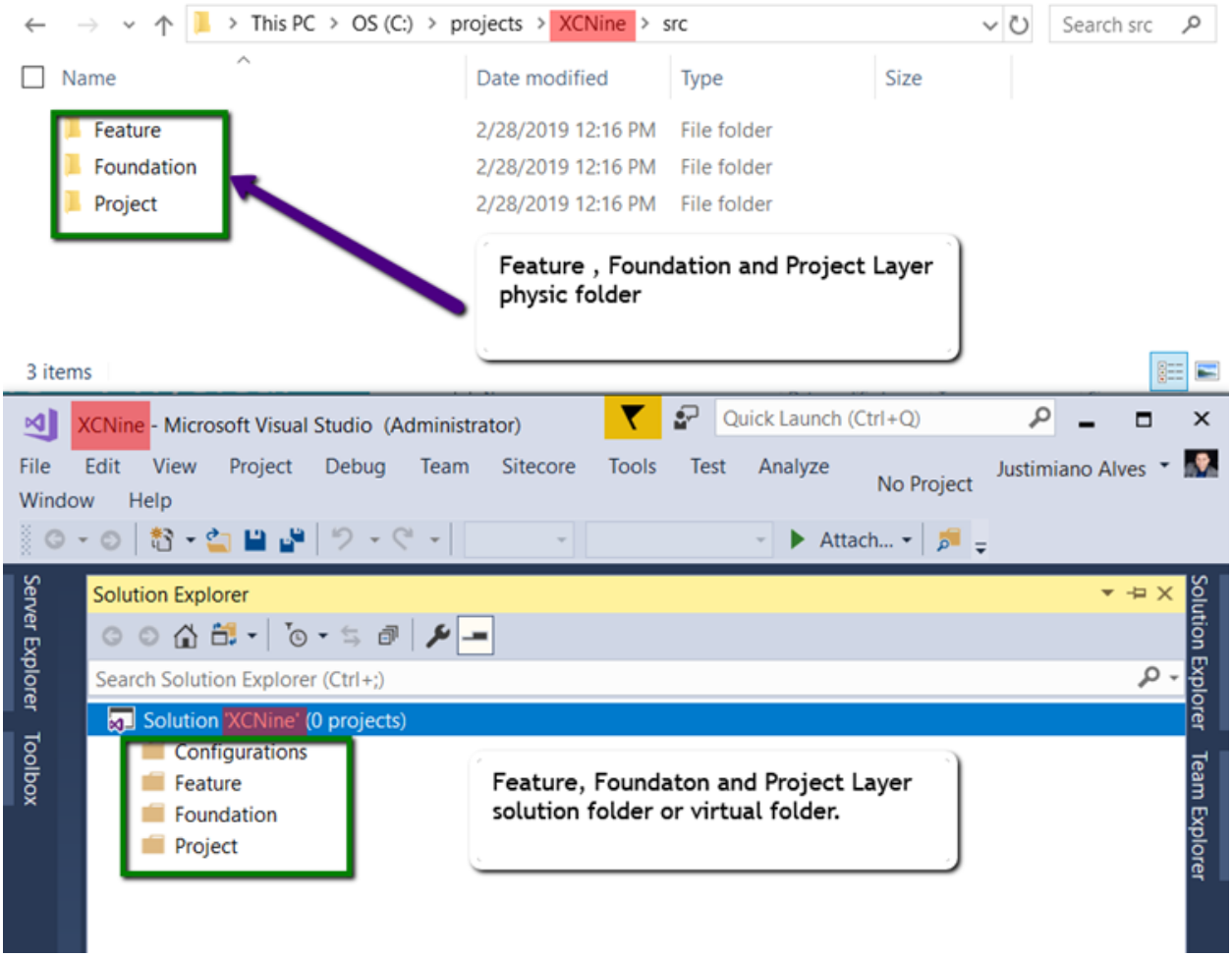

Figure 1: Physical and Virtual Structure folder

MODULE: Add Project Module to the Layer

In order to create your first module, you need first to create a solution folder that will contain the project module, as an example – ***Dictionary***. Then, right click on the folder and choose from the list of options *add a new project*. On the first screen choose the *.NET Framework 4.6.2*, and then use the VS browser to navigate to the *Foundation* layer and create a ***Dictionary*** folder.

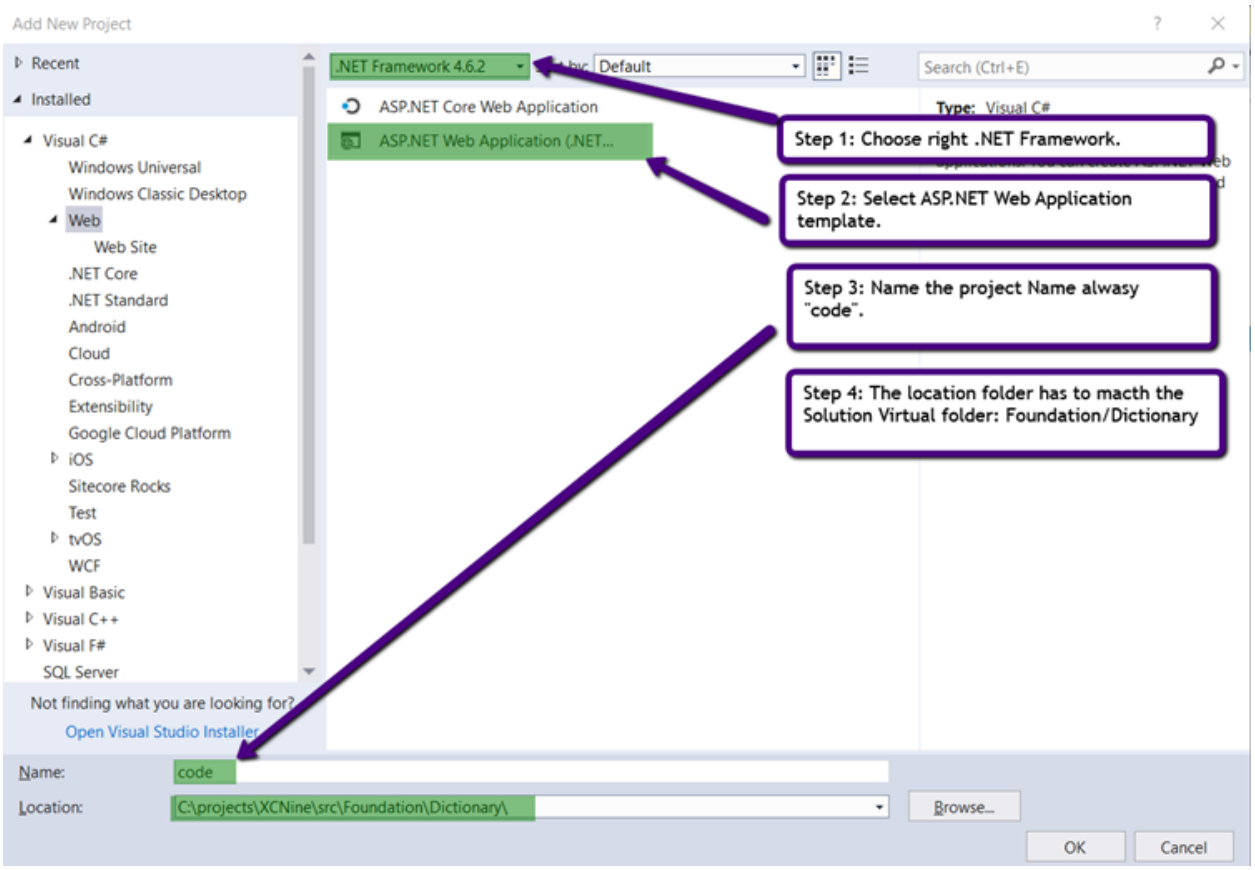

Figure 2: Foundation Module Dictionary – Select Framework, template, folder and name.

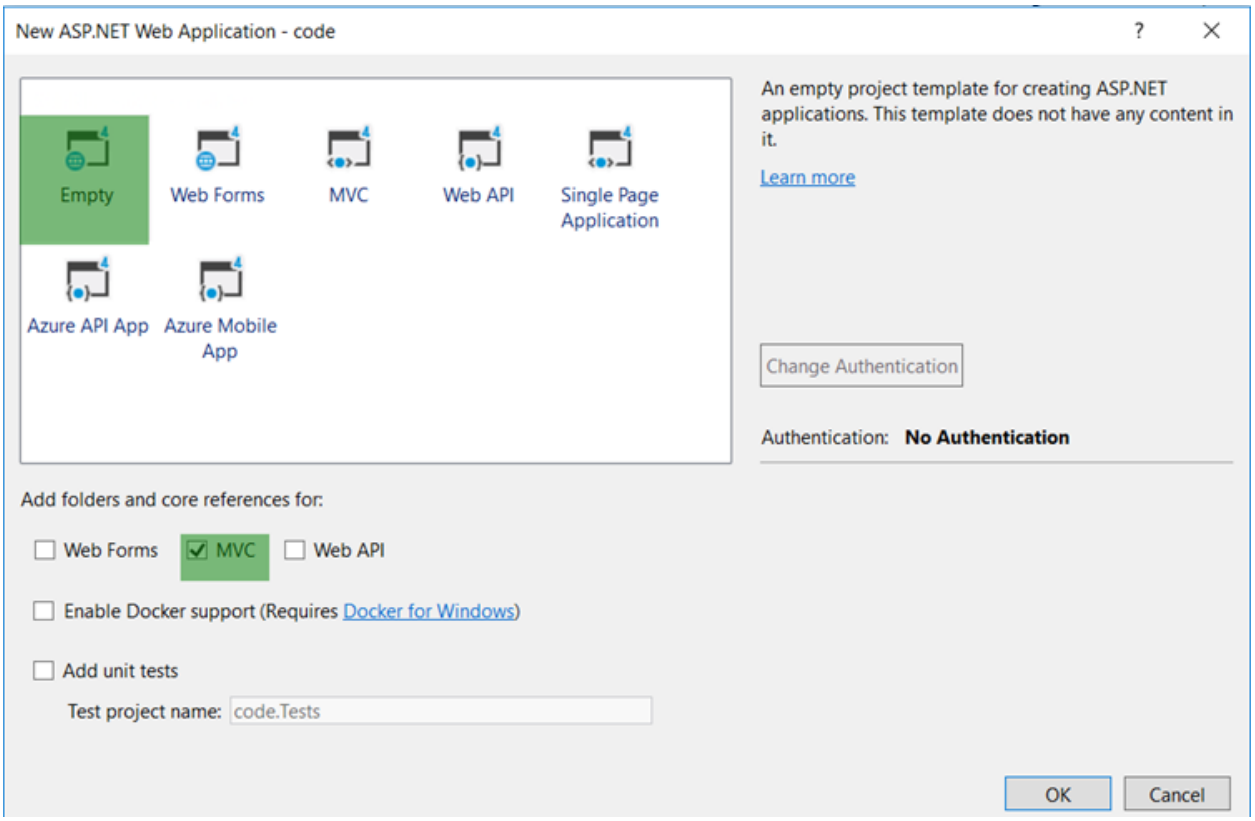

Figure 3: Dictionary Module Dictionary- Project template and reference

Following the Helix naming and convention, the module name will be ***XCNine.Foundation.Dictionary***. [See Partner, Principles, and Conventions]. Right click on the project and open the property to change the assembly and default namespace name.

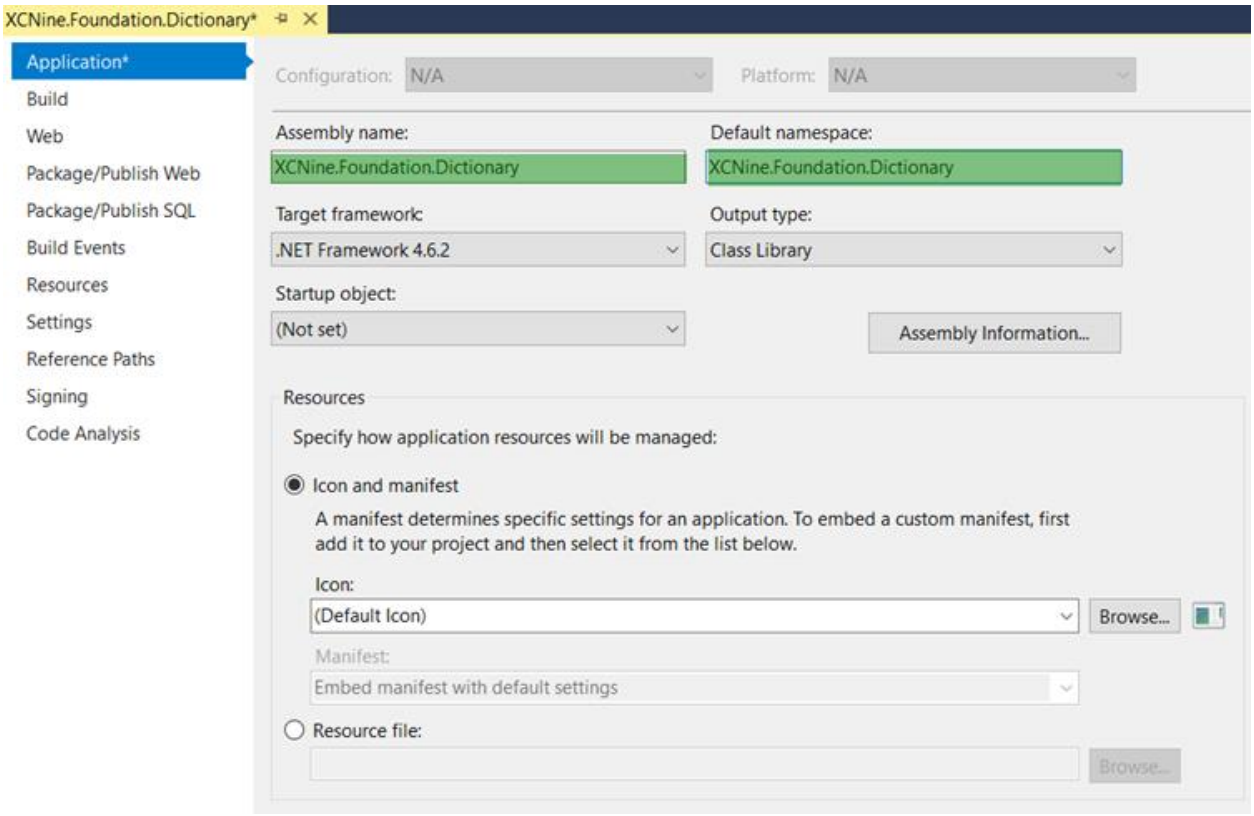

Figure 4: Update Namespace and Assembly

As the last step before kicking off the implementation of the module, you might consider doing some cleanup such as:

- Delete Global.asax file
- Delete *Views, App_Data, Controllers, App_Start* folder
- Update *web.config* build action property field to none in order to avoid being deployed and override Sitecore *web.config*.
    - In the `Views/web.config` file of your Visual Studio project, add the following Sitecore namespaces to the list of namespaces:
    - `<add namespace="Sitecore.Mvc" />`
    - `<add namespace="Sitecore.Mvc.Presentation" />`
-

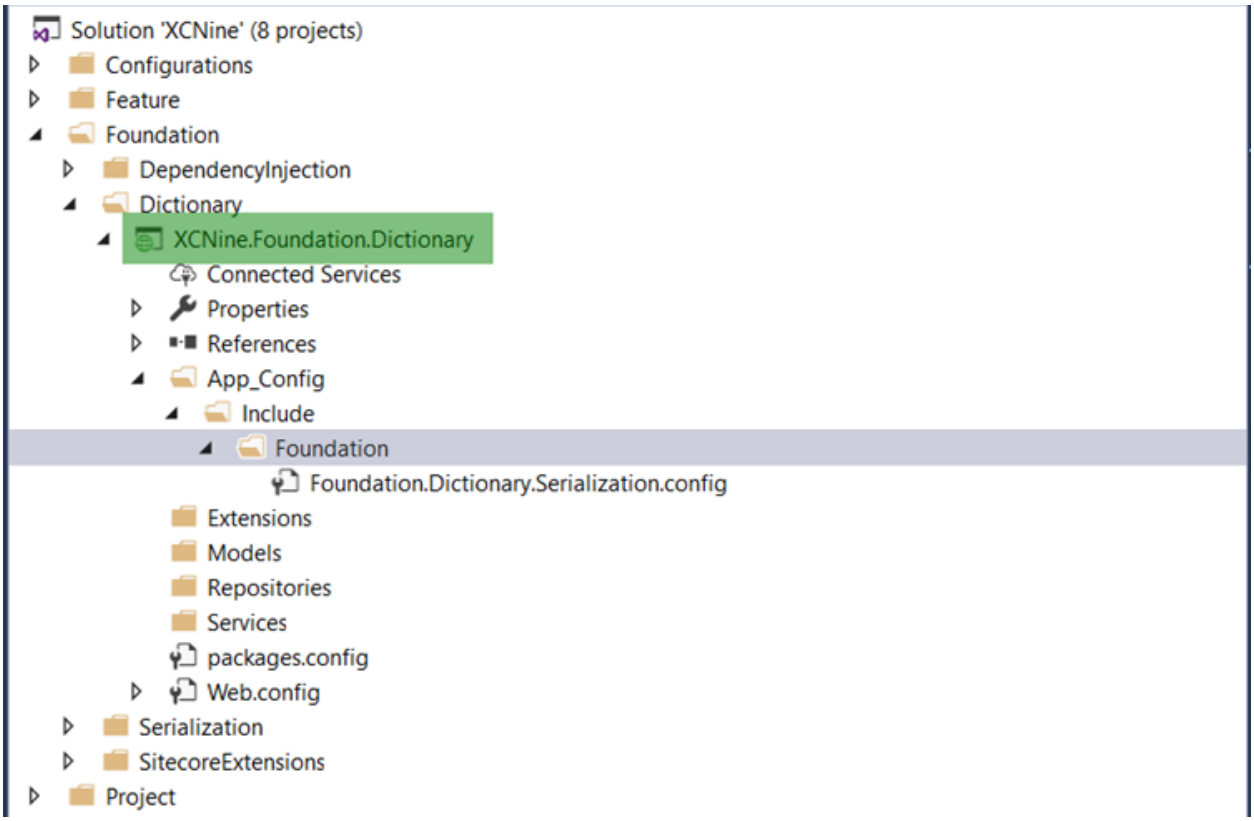

Figure 5: Foundation Dictionary - Clean projects

Add Sitecore kernel and Sitecore MVC dlls through the NuGet package and ensure you choose the correct dll version.

For Sitecore 9.2, we need below Nuget reference for 9.2

- Microsoft.AspNet.Mvc -> 5.2.4
  - Below references may be automatic updated with above reference
    - Microsoft.AspNet.Razor -> 3.2.4
    - Microsoft.AspNet.WebPages -> 3.2.4
- Sitecore.Kernel -> 9.2.0
- Sitecore.Mvc -> 9.2.0
- Sitecore.ContentSearch -> 9.2.0

In the **Solution Explorer** in Visual Studio, set the Copy Local property of each DLL reference the **NuGet Package Manager** added to `False`.

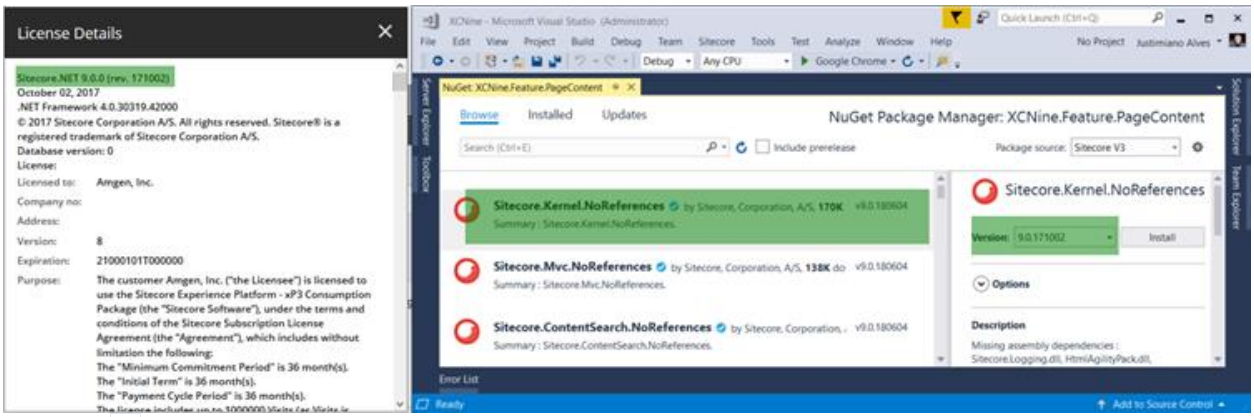

Figure 6: Sitecore Kernel Dll

DEPLOYMENT: Gulp + Unicorn

For the deployment, you are going to use Unicorn to manage Sitecore Items and Contents. Gulp will be used to run the task that deploys files, Platform files, Configurations files and more.

Install Unicorn

You can install Unicorn through NuGet package manager as you usually do for any package. However, before you install unicorn ensure you already have created the ***Serialization*** module project under ***Foundation*** Layer [
See ***MODULE: Add Project Module to the Layer***]. Download resource regarding Serialization module from the Habitat project.

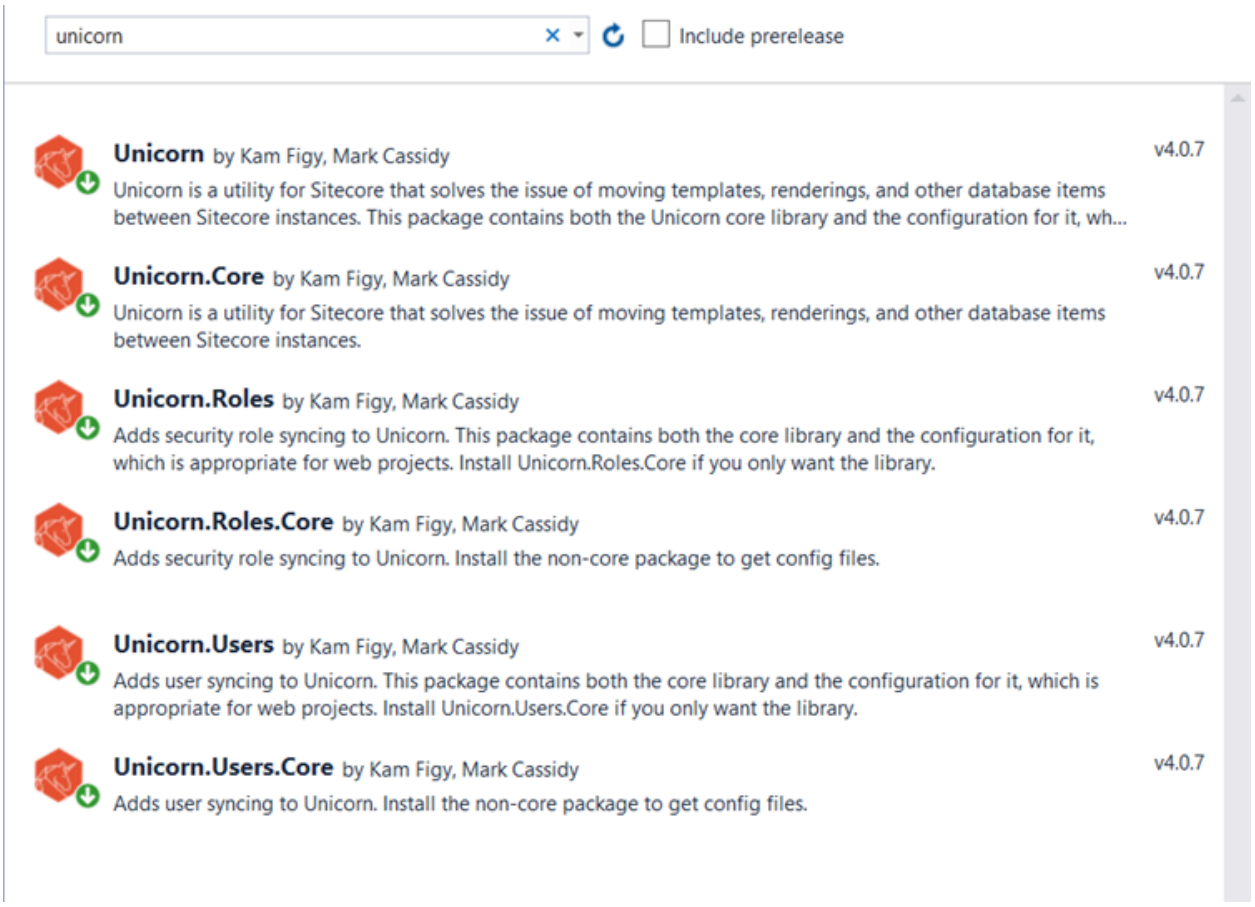

Figure 7: Unicorn References

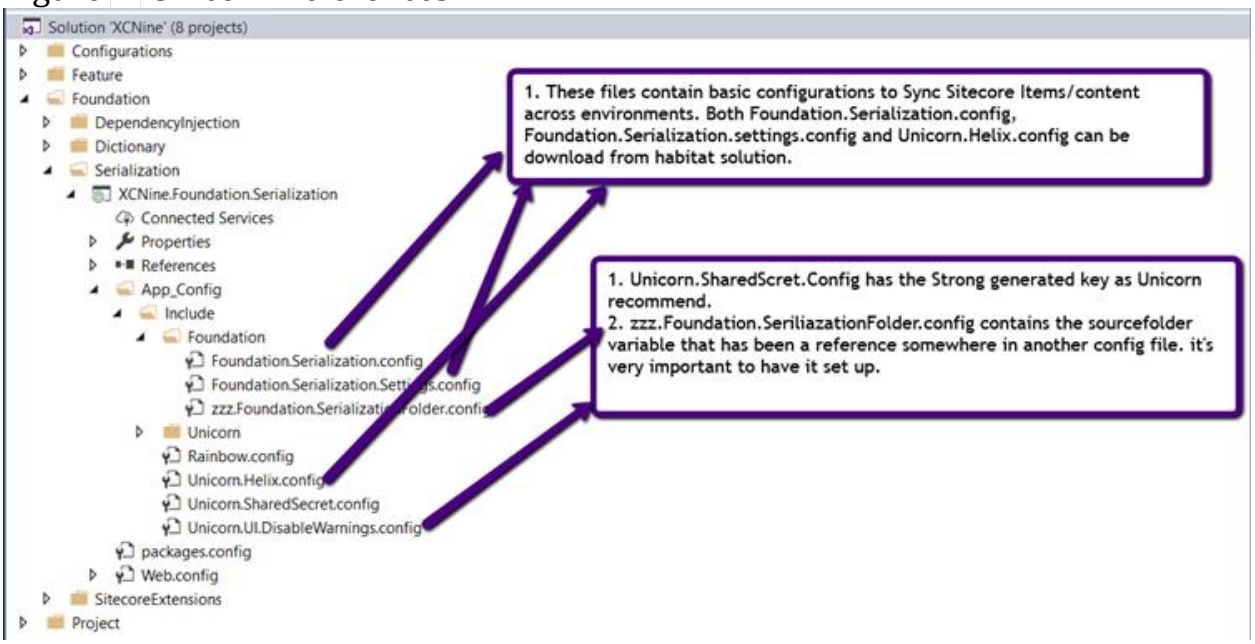

Figure 8: Unicorn - Config files

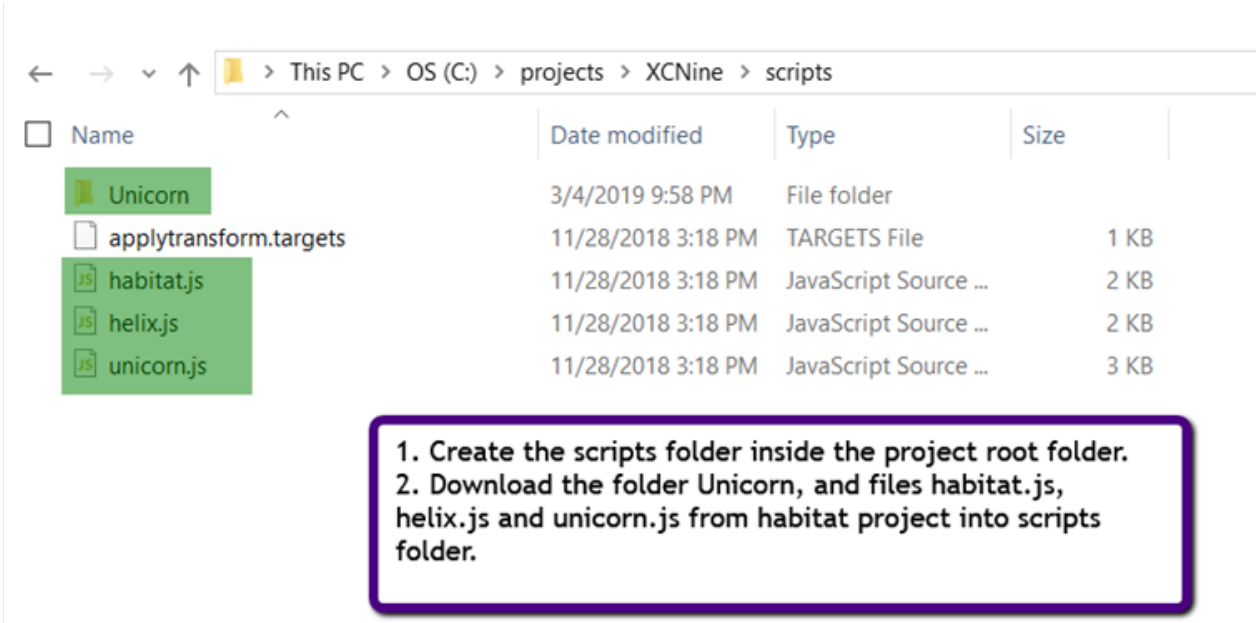

Figure 9: Unicorn scrips for Gulp Unicorn-Sync task

Install Gulp and Deploy

Download or copy from Habitat project the following files *package.json*, *gulpfile.js,* and *gulp-config.js* into your project root folder. Then open the command line as administrator and ensure that npm and node are installed by checking their version. To install the gulp, execute the following command:

npm install gulp –save-dev
npm install gulp-msbuild
npm install gulp-debug
npm install gulp-foreach

From Visual Studio, add the *Configurations* solution folder and then move the following existent files *gulpfile.js*, *gulp-config.js*, and *publishsettings.targets* into it. Open the *gulp-config.js* and update variables to match your local environments.

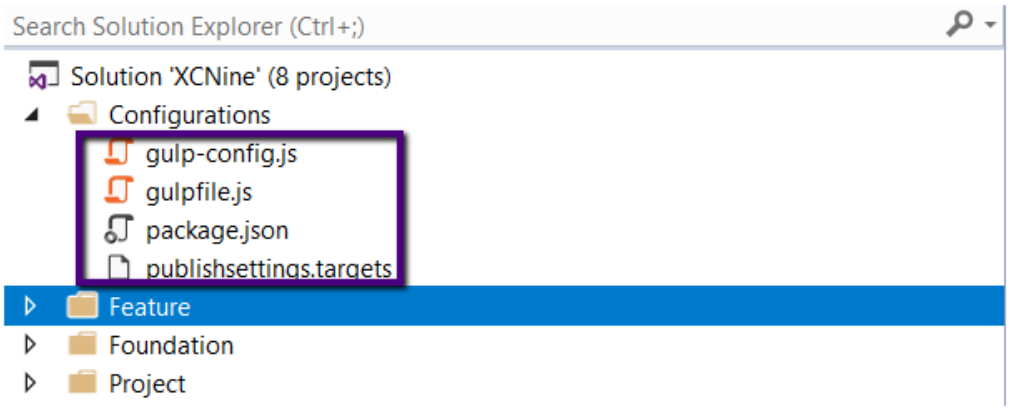

Figure 10: Configurations folder and config files

At this point, you are all set to deploy your project to the Sitecore website. Gulp *Sync-unicorn* will fail for the first deployment because you need to create all the root items in the Sitecore content tree that were referenced in the configurations file and the first sync needs to be done through the admin portal - http://xcnine.local/unicorn.aspx. Below is the list all Sitecore root items where the Layers (Foundation, Feature, and Project) should be created.

- Layouts - /sitecore/layout/Layouts
- Models - /sitecore/layout/Models
- Renderings - /sitecore/layout/Renderings
- Placeholder Settings - /sitecore/layout/Placeholder Settings
- Templates - /sitecore/templates
- Settings - /sitecore/system/Settings
- Media Library - /sitecore/media library

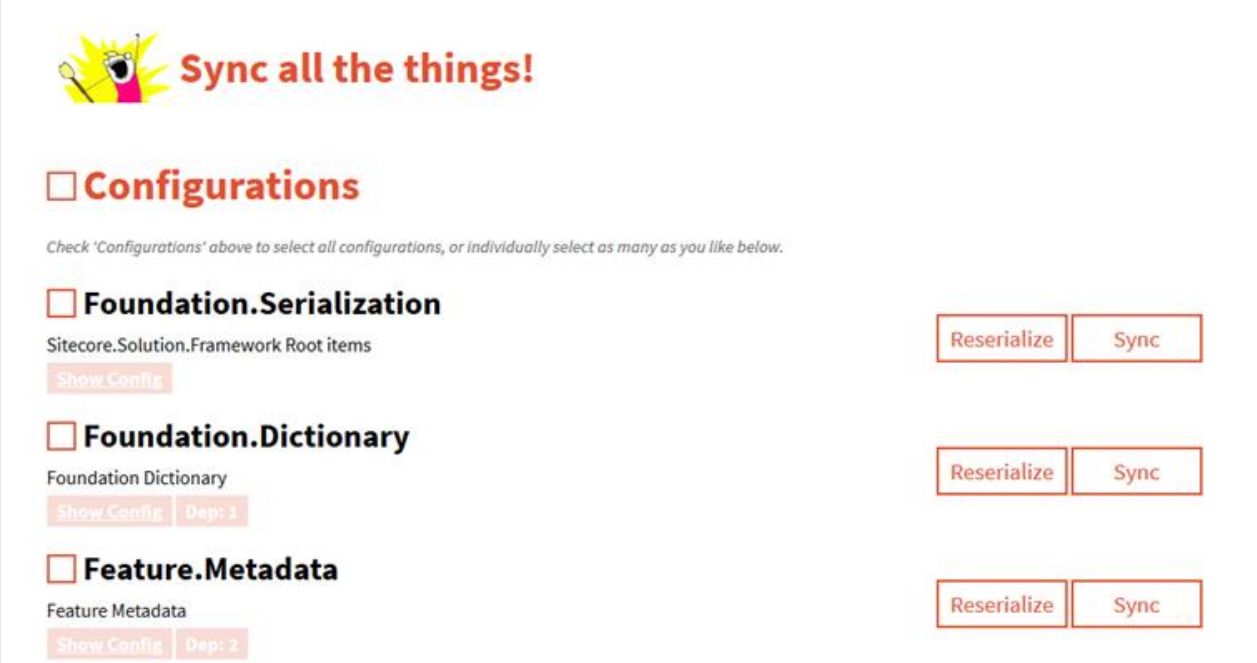

Figure 11: Unicorn Admin Page

DEMO: Metadata Module

Follow the steps illustrated on section - ***MODULE: Add Project Module to the Layer*** to create the Metadata module under Feature layer. You can get the Metadata source code from the Habitat project, which is reason why I won't cover in this blog. However, I will be sharing some screenshots related to Sitecore content items such as Metadata templates, Page template, and Home item. As a result of this demo, you should be able to see metadata information you have entered in the home Item using any browser. Login into your CMS (XCNine.local/sitecore) and create the following Metadata templates *_Keyword*, *_OpenGraph*, *_PageMetadata*, *_SiteMetadata* and the Page (/sitecore/templates/Project/Public/Pages/Page) template that will implement those metadata templates. Change the Home item template in order to be based on the Page template.

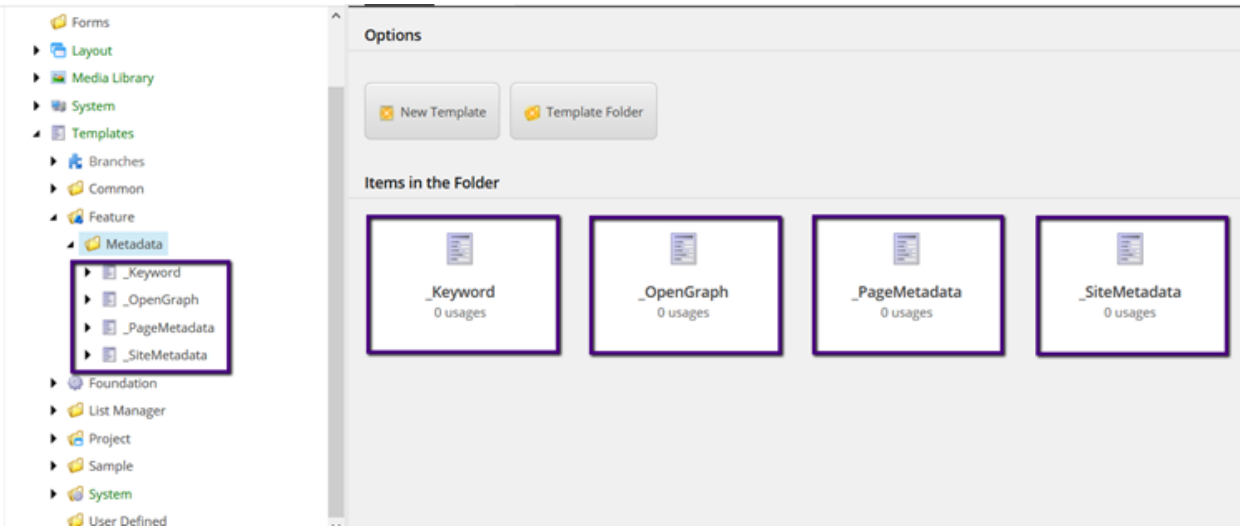

Figure 12: Metadata Module – Template

Next, change the Home Item template you will see all the new sections and fields to enter the metadata information.

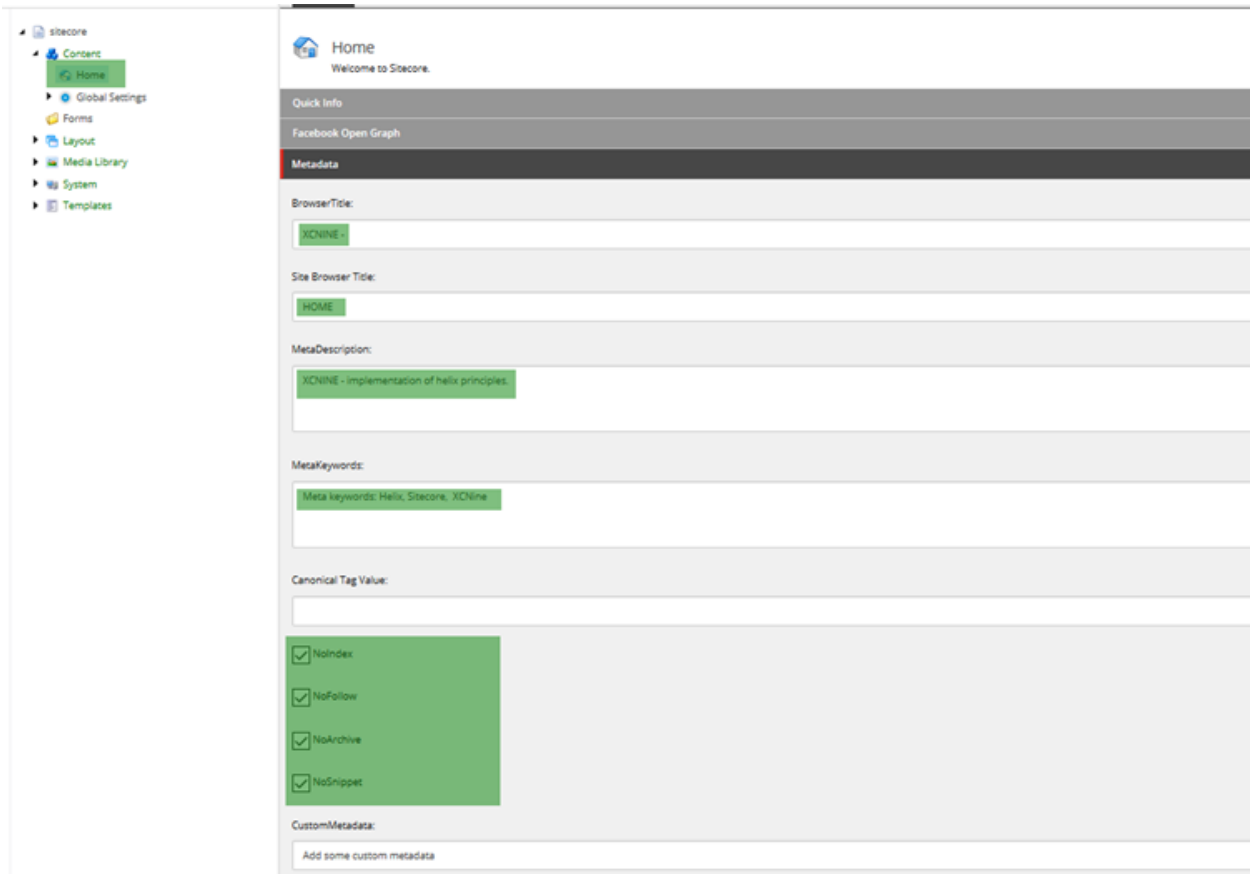

Figure 13: Metadata Module - Home Content

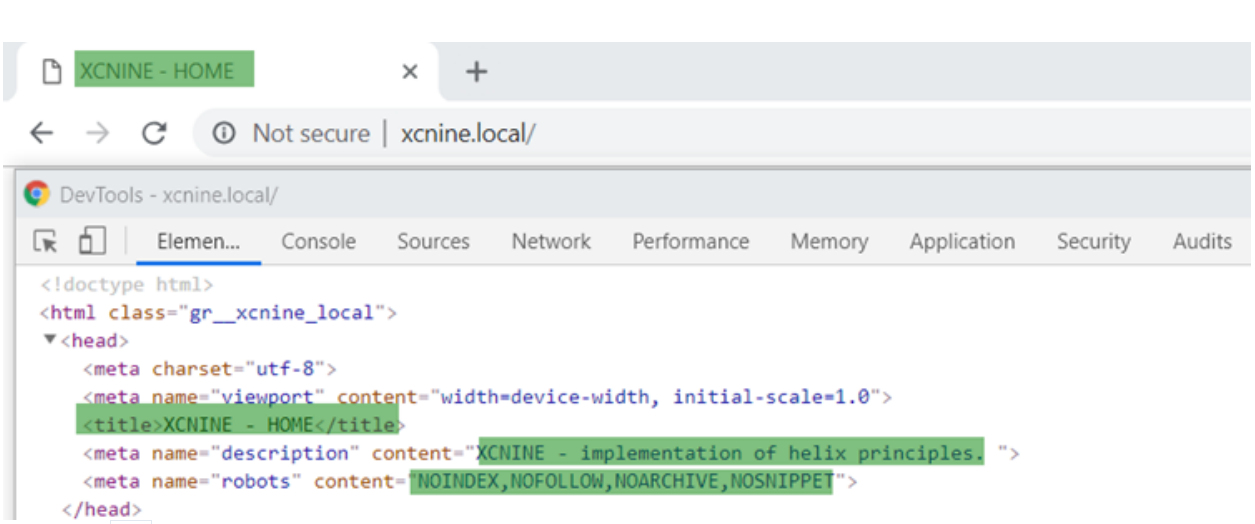

Figure 14: Metadata Module - Home Page

Few more useful links

How to create a Sitecore Helix solution from scratch

- https://www.youtube.com/watch?v=7MSaTFYBVLM

Create Sitecore Helix Solution in Less than 10 minutes

- https://www.youtube.com/watch?v=22TIDHBlLFM