Asp.Net MVC 5 Starter Kit Documentation

Release 1.0.0

Sarto Research, Thiago A. Schneider

Jul 24, 2019

Contents

Asp.Net MVC 5 Starter Kit is a basic, clean, globalized and S.O.L.I.D template with all the necessary boilerplate is ready to go.

Attention: *Docs are still a work in progress*

Source Code:

> https://github.com/akasarto/aspnet-mvc5-starter-template

Contents:

CHAPTER 1

Getting Started

Get the project up and running with a few simple steps.

1.1 Minimum Requirements

- Windows 10+ machine
- Visual Studio 2017 Community v15.8.3+ or
- SQL Server Express LocalDB or .

SQL Server Express LocalDB is included in Visual Studio 2017 editions.

1.2 Running locally

Default credentials for the initial user:

- **Username:** `admin`
- **Password:** `password`

Using Script (Recommended)

- Open a new `cmd` or `powershell` console window.
- Navigate to the project root folder (where it was extracted or cloned).
- Execute the following command to setup the app: `app install` or `./app install`.
  - The command above will perform the the following tasks:
    - Check for compatible PowerShell and .NET Framework versions.
    - Restore both server and client side dependent libraries (from *Nuget* and *LibMan*).
    - Compile the application after all required dependencies are properly restored.

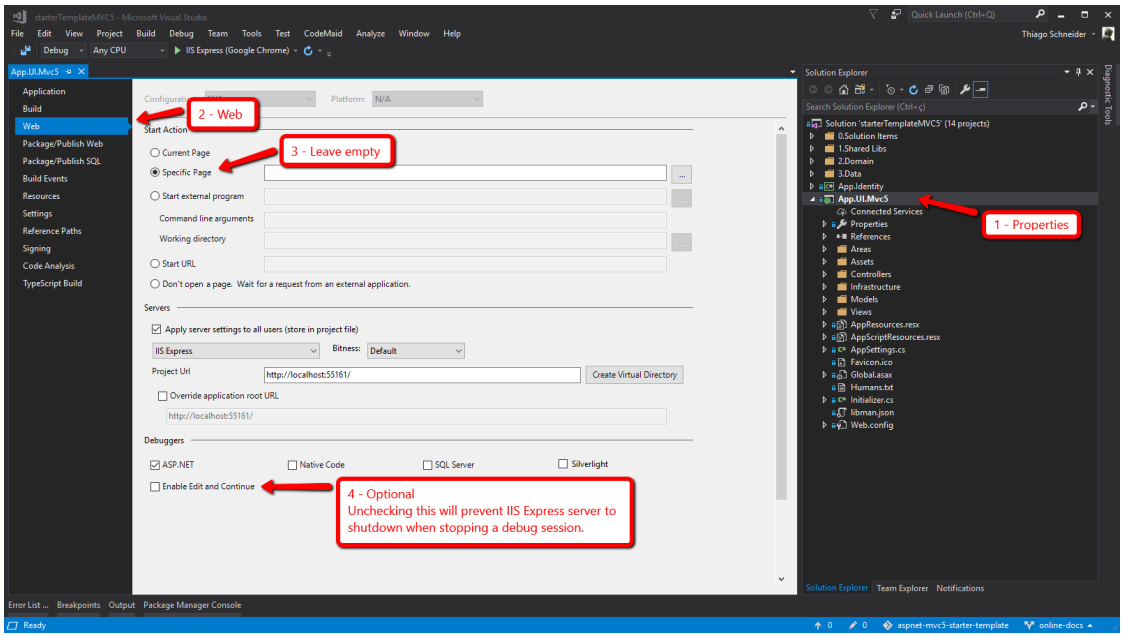
        * Create the **starterTemplateMVC5** database on your **LocalDB** instance.
- If necessary, change the **web.config** connection string to point to your desired SQL Server edition.
- Open the `starterTemplateMVC5.sln` solution file under the `sources` folder.
- If necessary, set **App.UI.Mvc5** as the startup project.
- Hit F5 to start the application.

Manual Setup (if the above fails for some reason)

- Create a database named `starterTemplateMVC5`
- Execute the script under `/sources/Data.Tools.Migrator/SqlServerScripts/Create_Initial_Db_Structure.sql` to create the db objects.
- Execute the script under `/sources/Data.Tools.Migrator/SqlServerScripts/Create_Initial_SuperUser_Account.sql` to create the default user.
- If necessary, change the **web.config** connection string to point to your SQL Server.
- Open the `starterTemplateMVC5.sln` solution file under the `sources` folder.
- If necessary, set **App.UI.Mvc5** as the startup project.
- Restore required *Nuget* and *LibMan* dependencies.
- Compile and you should be good to go.
- Hit F5 to start the application.

1.3 Optional Settings

The configuration bellow will ensure your project will always start in the home page and that the II Express instance will not shutdown after stopping the debugger.



1.4 Solution .editorconfig

Under the **0.Solution Items** solution folder, there is a global `.editorconfig` file that will ensure consistency between certain aspects of the code. This will apply to all users and will override their indivisual settings for this project.

More details can be found .

CHAPTER 2

Solution Details

Detailed information about the solution architecture, conventions, design choices and used tech.

2.1 Projects

Solution Folder / Project	Description
0.Solution Items	This folder contains globally available scripts, docs and configuration files.
1.Shared Libs	This folder contains projects that are independent from the main model and could be event converted to nuget packages to be used in another projects. These projects provide several functionality such as email and sms messaging, storage, image processing and helful extensions.
2.Domain	This folder contains the projects that can be considered the **core** of the onion architecture concept. This will contain the main code for the problem your project is solving.
3.Data	This folder contains the projects that will handle the data your application will produce and/or consume.
App.Identity	This is the project that will handle the ASP.Net Identity functionality for the users. It should be self contained and has its own repository and services to handle the app identities.
App.UI.Mvc5	This is the main *User Interface* project and will be the project that your end users will do most of their interaction with.

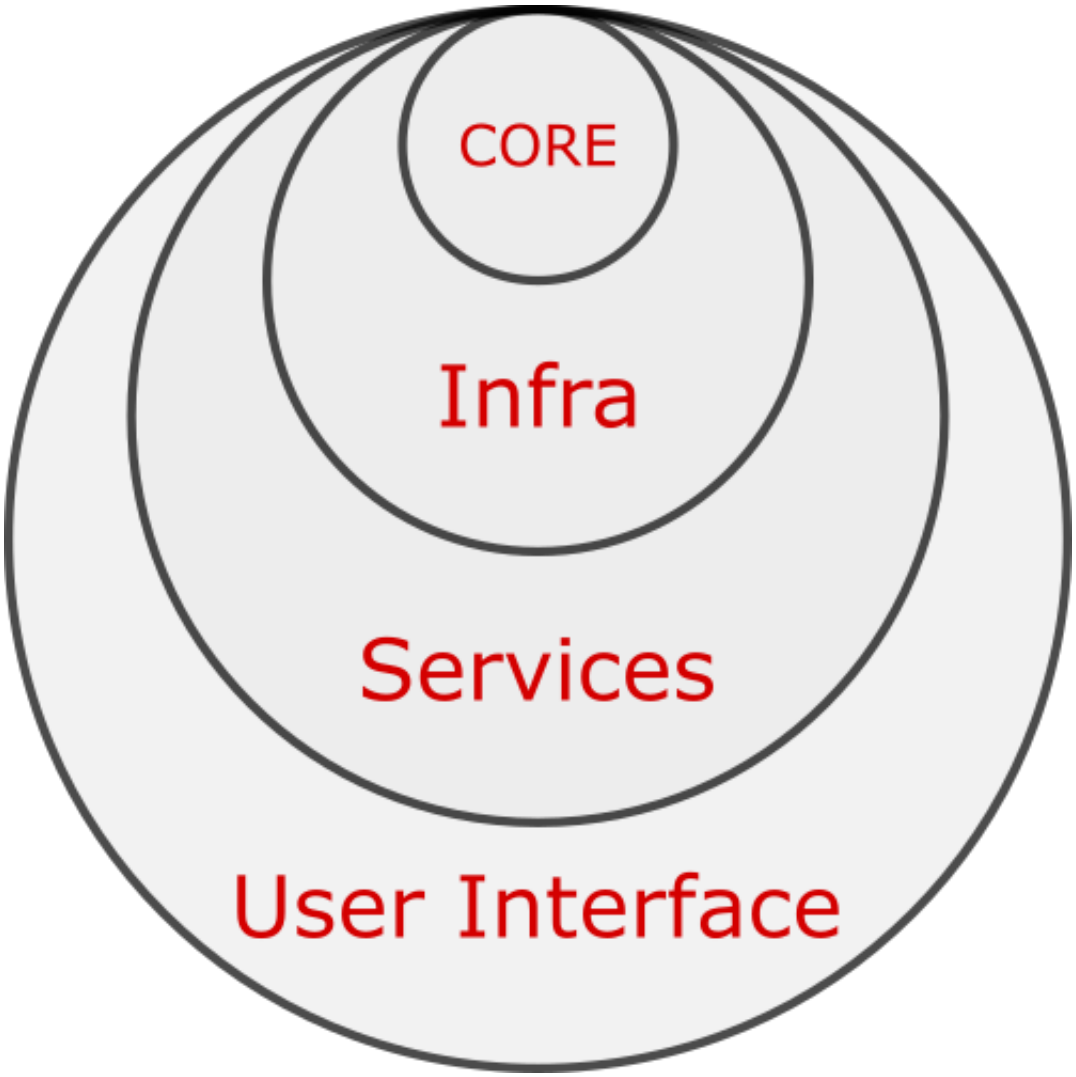
2.2 Architecture

The solution was built following the **Onion Architecture** concept.

The overall philosophy of the Onion Architecture is to keep your business logic and model in the middle (Core) of your application and push your dependencies as far outward as possible.

Basically, it states that code from inner layers should not depend on code from outer layers. It is very simple and help keeping things organized.

Check the image below:



Using the diagram above as an example, we can say that all code in the application can depend on code from the *core* layer, but code in the *Infra* layer can **not** depend on code from *Services* or *User Interface* layers.

For more information regarding this topic, please check the original definition post at http://jeffreypalermo.com/blog/the-onion-architecture-part-1/.

2.3 Folder Conventions

The application folders use a simple structure that has been proven to keep things handy and organized. Take the snippet bellow:

```
folder
├── subfolder
|   └── file.ext
└── file.ext
```

Every major feature has its own folder. Files shared by that feature will be kept in the root folder and possible subfolders will follow the same rules. Taking *part* of the **Infrastructure** folder from the **App.UI.Mvc5** project, we have the following result:

```
Infrastructure
├── Blobs
|   ├── BlobService.cs
|   ├── BlobServiceConfigs.cs
|   ├── BlobUploadResult.cs
|   └── IBlobService.cs
├── Cookies
|   ├── GetCookie.cs
|   └── SetCookie.cs
├── Realtime
|   ├── Configs
|   |   ├── HubActivator.cs
|   |   ├── SignalRCamelCaseJsonResolver.cs
|   |   └── UserIdProvider.cs
|   ├── Hubs
|   |   └── DatabusHub.cs
|   ├── IRealtimeService.cs
|   └── RealtimeService.cs
├── UrlExtensions
|   ├── BlobThumbnail.cs
|   └── GetHomeUrl.cs
├── AppAreas.cs
├── AppJsonResult.cs
├── AppJsonResult.cs
├── AppViewEngine.cs
├── AppWebViewPage.cs
├── AuthorizeAttribute.cs
├── DirectRouteProvider.cs
└── SetLayout.cs
```

2.4 Namespace Conventions

Namespaces are defined keeping the folder structure above in mind, but they will go no farther then the first subfolder level for an specific project. Using the **App.UI.Mvc5** project as example, the namespaces will be as follows:

```
App.UI.Mvc5
├── Controllers
|   └── (Namespace: App.UI.Mvc5.Controllers)
├── Infrastructure
|   ├── Configs
```

(continues on next page)

(continued from previous page)

```
|   └── (Namespace: App.UI.Mvc5.Infrastructure)
└── (Namespace: App.UI.Mvc5.Infrastructure)
├── Models
|   └── (Namespace: App.UI.Mvc5.Models)
└── (Namespace: App.UI.Mvc5)
```

2.5 Tech Used and Third-Party Libraries

A list with the main tech and libraries that are used throughout the solution, for further information.

Microsoft

- Latest released Visual Studio Community edition or higher (https://www.visualstudio.com).
- ASP.Net MVC 5 (http://www.asp.net/mvc).
- ASP.Net Identity 2.0 (http://www.asp.net/identity).
- SQL Server Express and Tools (https://www.microsoft.com/en-us/sql-server/sql-server-editions-express/).
- PowerShell - for script execution and automation (https://docs.microsoft.com/en-us/powershell/scripting/overview).

Third party libraries

- Image Resizer ** - for local image manipulation (http://imageresizing.net/plugins/editions/free).
- Dapper .Net Micro ORM - for data manipulation (https://github.com/StackExchange/Dapper).
- Simple Injector - for IoC and Dependency injection (https://simpleinjector.org).
- Serilog - for logging (https://serilog.net).
- Json.NET - for json data manipulation (http://www.newtonsoft.com/json).
- FluentValidation - for client and server data validation (https://fluentvalidation.net).
- ValueInjecter - for class mapping (https://github.com/omuleanu/ValueInjecter).
- FluentMigrator - for database robust versioning and manipulation (https://fluentmigrator.github.io).

** Free and paid version available.*

CHAPTER 3

Client Side Scritps

This solution does not use nuget packages to reference client side scripts and libraries. Instead, we use the newly available Library Manager (LibMan) that has become available since the release of Visual Studio 2017 Community v15.8.3+ or higher.

Referencing new libraries is pretty simple, just edit the `libman.json` file under the **App.UI.Mvc5** project root providing the library name and the destination to where it should be restored and you'll be good to go:

```
{
    "version": "1.0",
    "defaultProvider": "cdnjs",
    "libraries": [
        {
            "library": "jquery@3.3.1",
            "destination": "Assets/vendor/jquery"
        }
    ]
}
```

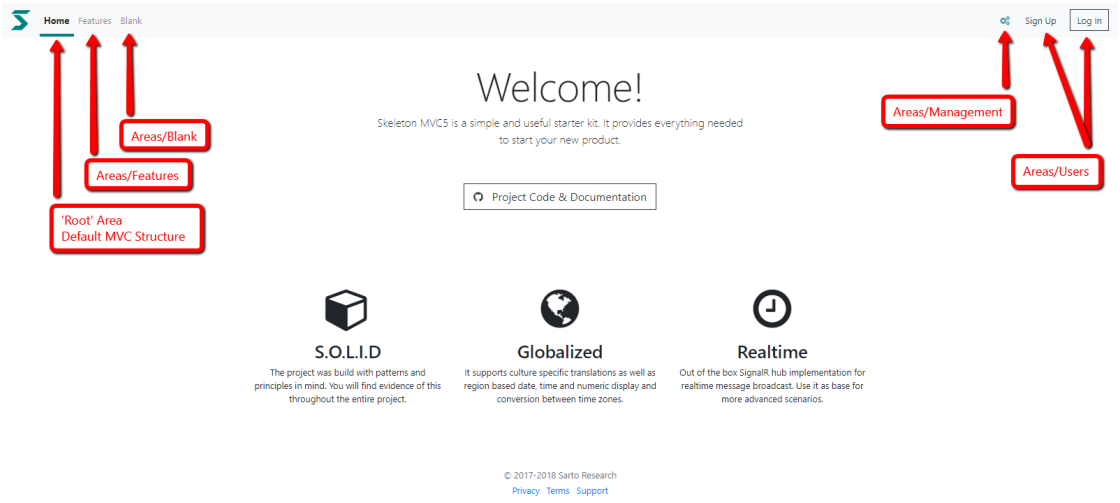
You can browse available libraries from the https://cdnjs.com/ catalog or add your preferred source.

For more details, check the official docs at https://docs.microsoft.com/en-us/aspnet/core/client-side/libman.

CHAPTER 4

Asp.NET MVC Areas

It is not very common to see areas being used in MVC applications, but when properly configured, they can be very handy. Providing good modularity options to the solution:



Each area can be considered a 'mini mvc' website inside the main app and need just a few adjustments to get up and running:

- Area name and namespaces
- Area controllers
- Views and models

- Client side scripts

The easiest way to create a new area is copying the `Blank` area and renaming the required classes as needed.

Areas **must** have their own base controller that inherits from the main base controller (`__BaseController.cs`). For convention, the area base controller is named `__AreaBaseController.cs`.

Considering you may want to create a new area named `Products`, your new area base controller will look like the following:

```
using App.UI.Mvc5.Controllers;
using System.Web.Mvc;

namespace App.UI.Mvc5.Areas.Products.Controllers
{
    [RouteArea("Products", AreaPrefix = "products")]
    public abstract class __AreaBaseController : __BaseController
    {
    }
}
```

Notice above that we also set the `AreaPrefix` in there to make sure all routes that bellong to the new area will start with `procucts/`.

Another controller that is required is the `_LandingController.cs`. The sole responsibility for this controller is to redirect the request to the primary controller of the area. It was designed like this so all areas can be called in a standardized way. The landing controller for the example above will look like the following:

```
using System.Web.Mvc;

namespace App.UI.Mvc5.Areas.Products.Controllers
{
    [RoutePrefix("")]
    public class _LandingController : __AreaBaseController
    {
        [HttpGet]
        [Route(Name = "Products_Landing_Index_Get")]
        public ActionResult Index() => RedirectToAction("Index", "Overview");
    }
}
```

With that setup, it means that each time some request is made to the route `.../products`, the request will be redirected to the `Index` action in the `OverviewController.cs` class:

```
using App.UI.Mvc5.Infrastructure;
using App.UI.Mvc5.Models;
using System.Web.Mvc;

namespace App.UI.Mvc5.Areas.Products.Controllers
{
    [RoutePrefix("overview")]
    [TrackMenuItem("products.overview")]
    public partial class OverviewController : __AreaBaseController
    {
        [Route(Name = "Products_Overview_Index_Get")]
        public ActionResult Index()
        {
            var model = new EmptyViewModel();
```

(continues on next page)

(continued from previous page)

```
            return View(model);
        }
    }
}
```

And for the final required step when setting up a new area, we must create a `_MenuController.cs` class as follows:

```
using App.UI.Mvc5.Models;
using System.Web.Mvc;

namespace App.UI.Mvc5.Areas.Products.Controllers
{
    [RoutePrefix("menu")]
    public class _MenuController : __AreaBaseController
    {
        [Route("top-menu-item", Name = "Products_Menu_TopMenuItem")]
        public ActionResult TopMenuItem()
        {
            var model = new EmptyPartialViewModel();

            return PartialView(model);
        }
    }
}
```

Notice that it will return a partial view named `TopMenuItem.cshtml` that will be available under the area `Views` folder:

```
@using App.UI.Mvc5.Areas.Products

<!-- Page Contents -->

<li class="nav-item @Menu.IfActiveItem("products.*", "active")">
    <a class="nav-link" href="@Url.Action("Index", "_Landing")">@(GetLocalizedString
↪<AreaResources>("Products"))</a>
</li>
```

The partial view can then be called anywhere in the main website to render the area menu entry (normally in the root `TopMenu.cshtml` file):

```
<ul class="nav navbar-nav mr-auto">

    <li class="nav-item @Menu.IfActiveItem("root.landing", "active", string.Empty)">
        <a class="nav-link" href="@Url.GetHomeUrl()">@GetLocalizedString("Home")</a>
    </li>

    @Html.Action("TopMenuItem", "_Menu", new { area = AppAreas.GetAreaName(Area.
↪Features) })

    @Html.Action("TopMenuItem", "_Menu", new { area = AppAreas.GetAreaName(Area.
↪Blank) })

    @Html.Action("TopMenuItem", "_Menu", new { area = AppAreas.GetAreaName(Area.
↪Products) })

</ul>
```

One last thing to notice is that, when using areas, all website links MUST know to which area the route is supposed to belong. To facilitate that process, the system provide a helper class named `AppAreas.cs` that can be found under the `Infrastructure` folder. Just add the new area name to the `Areas` enumerator and, when creating links, call the method as show above.

```
namespace App.UI.Mvc5.Infrastructure
{
    public enum Area : int
    {
        Root,
        Blank,
        Features,
        Management,
        Users,
        Products
    }

    public class AppAreas
    {
        public static string GetAreaName(Area area)
        {
            if (area == Area.Root)
            {
                return string.Empty;
            }

            return area.ToString();
        }
    }
}
```

CHAPTER 5

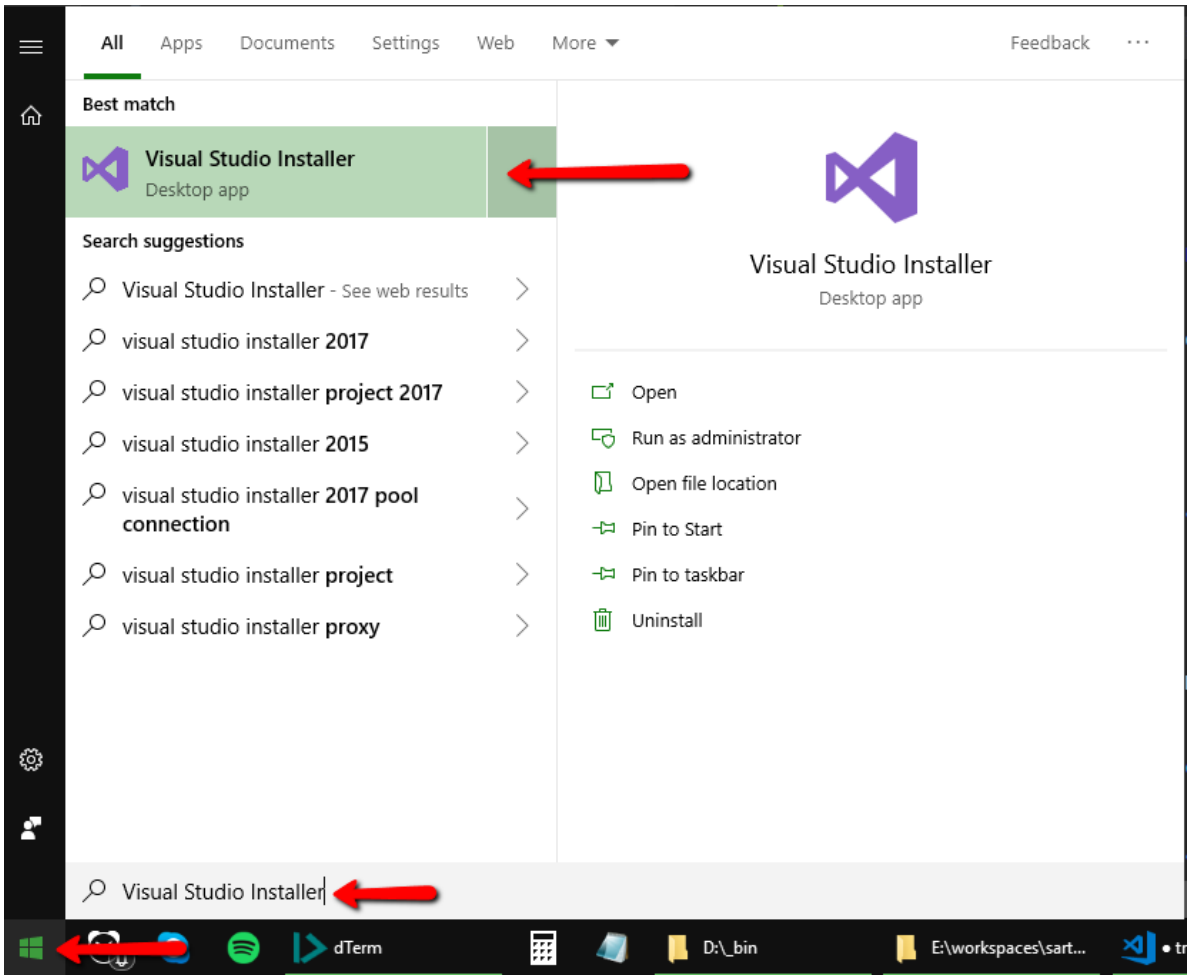
Troubleshooting

Possible problems that may occur during the development lifecycle.

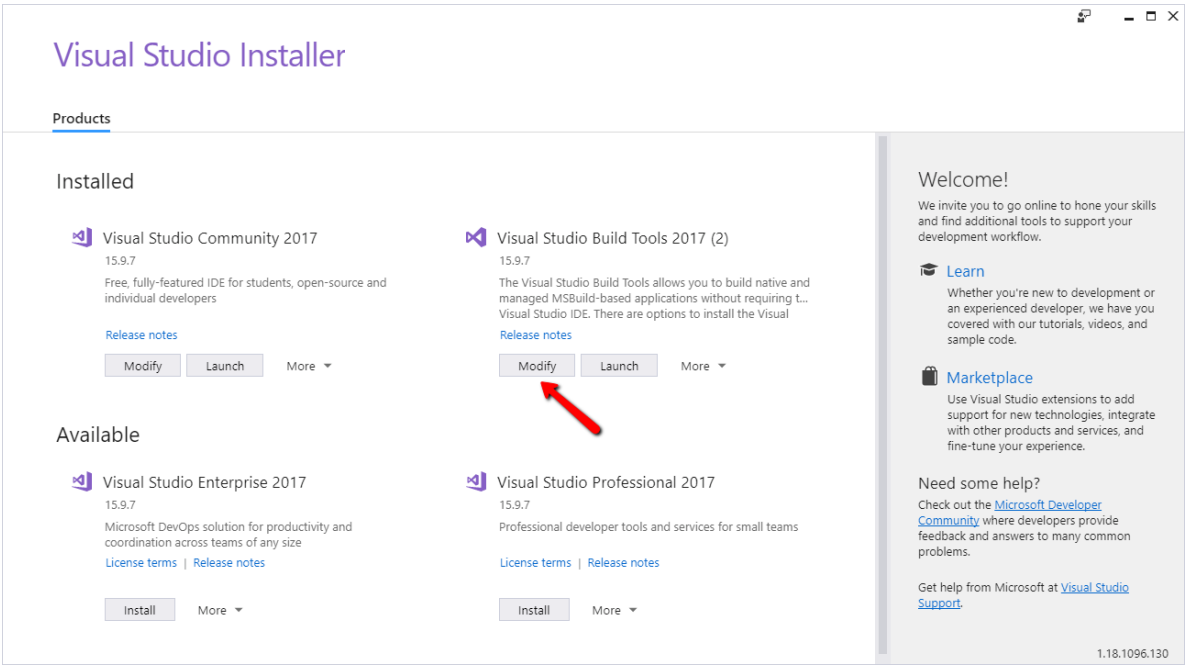
5.1 Scripted Build Errors

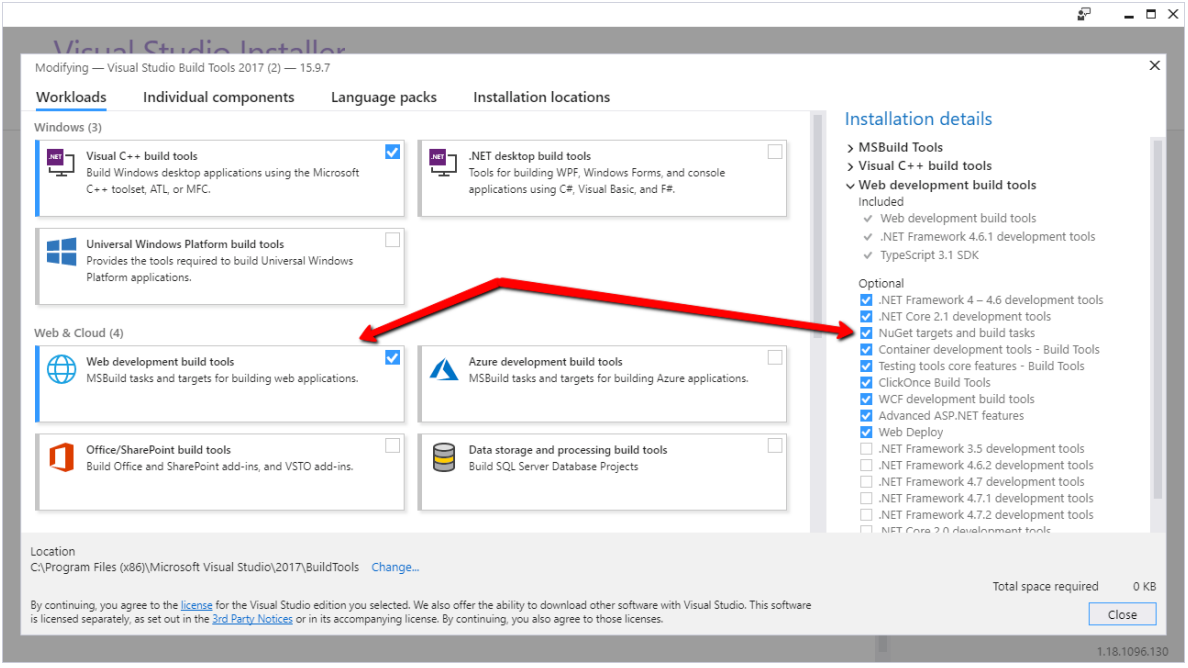
In some specific cases, the command `app.cmd install` may fail. If you're facing problems with that, try the following steps:

- On your **development** machine, look for the `Visual Studio Installer` application:



- Make sure that you have the **Nuget targets and build tasks** selected:


Visual Studio Installer
Products
Installed
Visual Studio Community 2017
15.9.7
Free, fully-featured IDE for students, open-source and individual developers
Release notes
Modify
Launch
More
Visual Studio Build Tools 2017 (2)
15.9.7
The Visual Studio Build Tools allows you to build native and managed MSBuild-based applications without requiring t... Visual Studio IDE. There are options to install the Visual
Release notes
Modify
Launch
More
Available
Visual Studio Enterprise 2017
15.9.7
Microsoft DevOps solution for productivity and coordination across teams of any size
License terms | Release notes
Install
More
Visual Studio Professional 2017
15.9.7
Professional developer tools and services for small teams
License terms | Release notes
Install
More
Welcome!
We invite you to go online to hone your skills and find additional tools to support your development workflow.
Learn
Whether you're new to development or an experienced developer, we have you covered with our tutorials, videos, and sample code.
Marketplace
Use Visual Studio extensions to add support for new technologies, integrate with other products and services, and fine-tune your experience.
Need some help?
Check out the Microsoft Developer Community where developers provide feedback and answers to many common problems.
Get help from Microsoft at Visual Studio Support.
1.18.1096.130

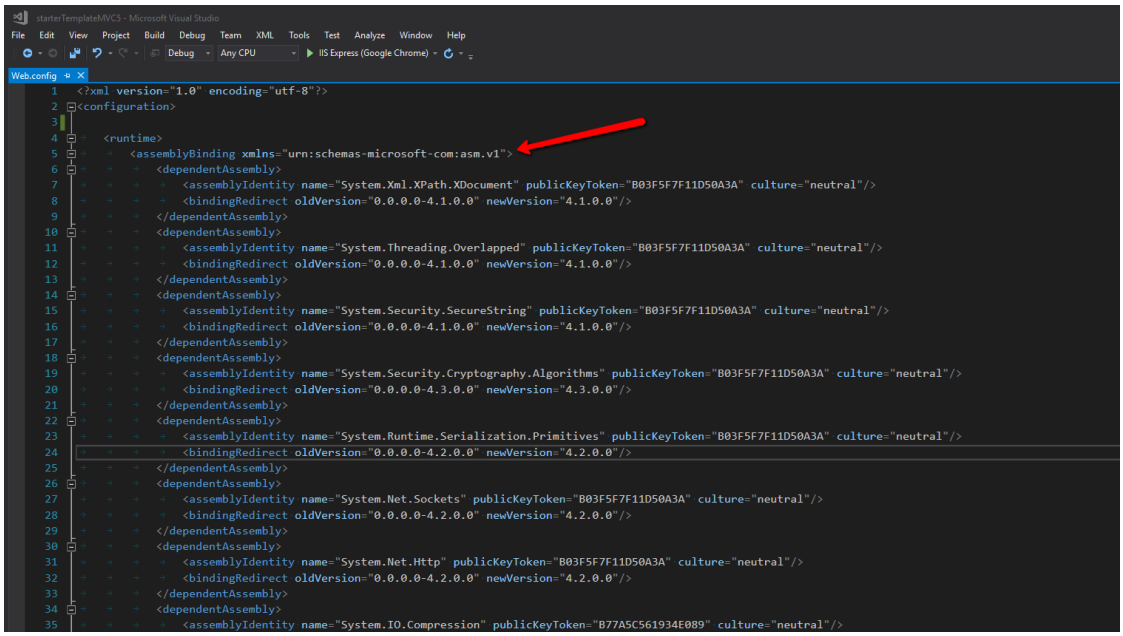

Modifying — Visual Studio Build Tools 2017 (2) — 15.9.7
Workloads
Individual components
Language packs
Installation locations
Windows (3)
Visual C++ build tools
Build Windows desktop applications using the Microsoft C++ toolset, ATL, or MFC.
.NET desktop build tools
Tools for building WPF, Windows Forms, and console applications using C#, Visual Basic, and F#.
Universal Windows Platform build tools
Provides the tools required to build Universal Windows Platform applications.
Web & Cloud (4)
Web development build tools
MSBuild tasks and targets for building web applications.
Azure development build tools
MSBuild tasks and targets for building Azure applications.
Office/SharePoint build tools
Build Office and SharePoint add-ins, and VSTO add-ins.
Data storage and processing build tools
Build SQL Server Database Projects
Installation details
MSBuild Tools
Visual C++ build tools
Web development build tools
Included
Web development build tools
.NET Framework 4.6.1 development tools
TypeScript 3.1 SDK
Optional
.NET Framework 4 – 4.6 development tools
.NET Core 2.1 development tools
NuGet targets and build tasks
Container development tools - Build Tools
Testing tools core features - Build Tools
ClickOnce Build Tools
WCF development build tools
Advanced ASP.NET features
Web Deploy
.NET Framework 3.5 development tools
.NET Framework 4.6.2 development tools
.NET Framework 4.7 development tools
.NET Framework 4.7.1 development tools
.NET Framework 4.7.2 development tools
Location
C:\Program Files (x86)\Microsoft Visual Studio\2017\BuildTools
Change...
Total space required
0 KB
By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.
Close
1.18.1096.130

5.2 Assembly mappings

Specially after updating nuget packages, you may experience runtime errors like the following:

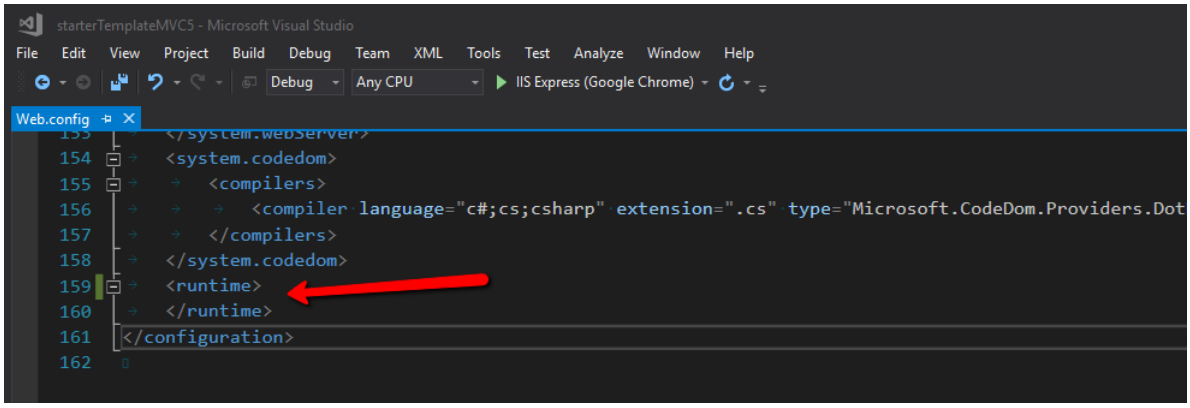
```
Could not load file or assembly 'xxx' or one of its dependencies. The
located assembly's manifest definition does not match the assembly reference.
(Exception from HRESULT: 0x80131040)
```

That is usually caused by assembly bindings that were not properly updated, along with the packages, in your **web.config** file, under the `configuration/runtime/assemblyBinding` node:

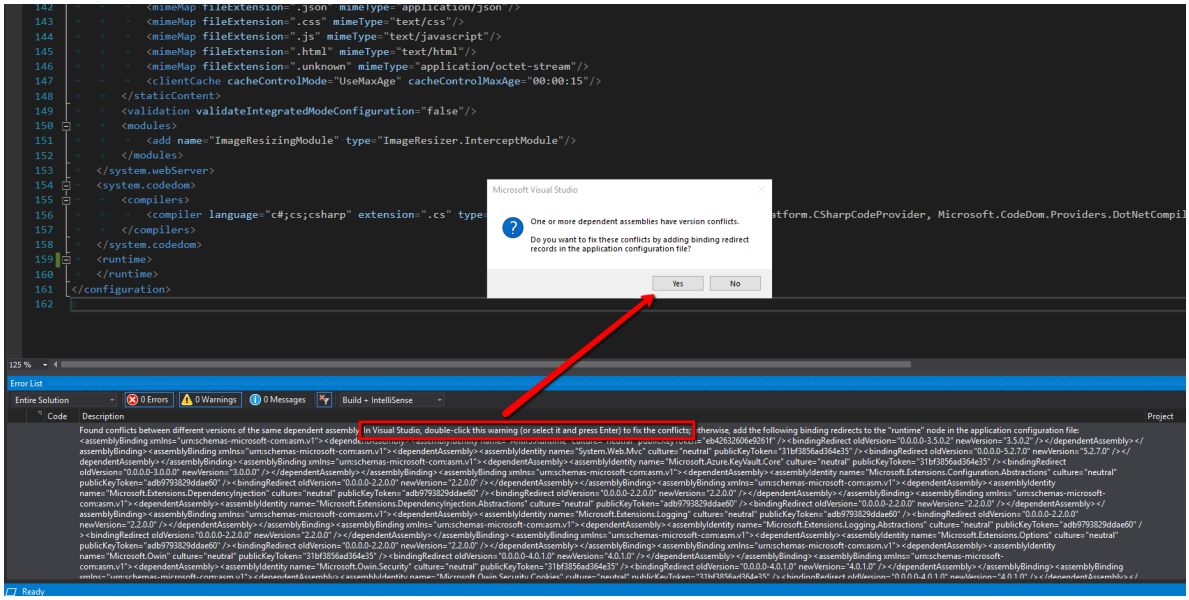


You can manually fix that by comparing the package versions with the ones being redirected to or let **Visual Studio** handle it for you by doing the following steps:

- Completely delete the `assemblyBinding` node from the **web.config** file.



- Recompile the application using Visual Studio.
- Click on the compilation warning as instructed and click 'Yes' to the action box that is shown.



Your assembly bindings should now be pointing to the correct versions and the runtime errors will be gone.

For more details, check the official docs at https://docs.microsoft.com/en-us/dotnet/framework/configure-apps/redirect-assembly-versions.