Contents

[Running the solution as Multiple Startup Projects 2](#_Toc15249437)

[Architecture 3](#_Toc15249438)

[Framework 4](#_Toc15249439)

[Dependency Injection 4](#_Toc15249440)

[CORS Policy 4](#_Toc15249441)

[AKQAConversion.Business 4](#_Toc15249442)

[AKQAConversion.Model 5](#_Toc15249443)

[AKQAConversion.API 5](#_Toc15249444)

[AKQAConversion.UI 5](#_Toc15249445)

[AppSettings.JSON 6](#_Toc15249446)

[Unit Tests and Mocking Framework 7](#_Toc15249447)

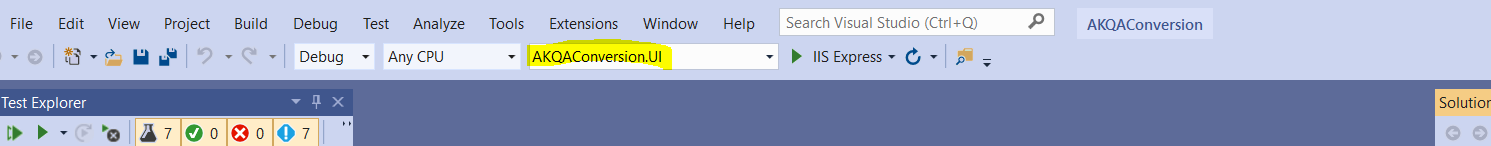
[Nuget 7](#_Toc15249448)

[Errors while building project 8](#_Toc15249449)

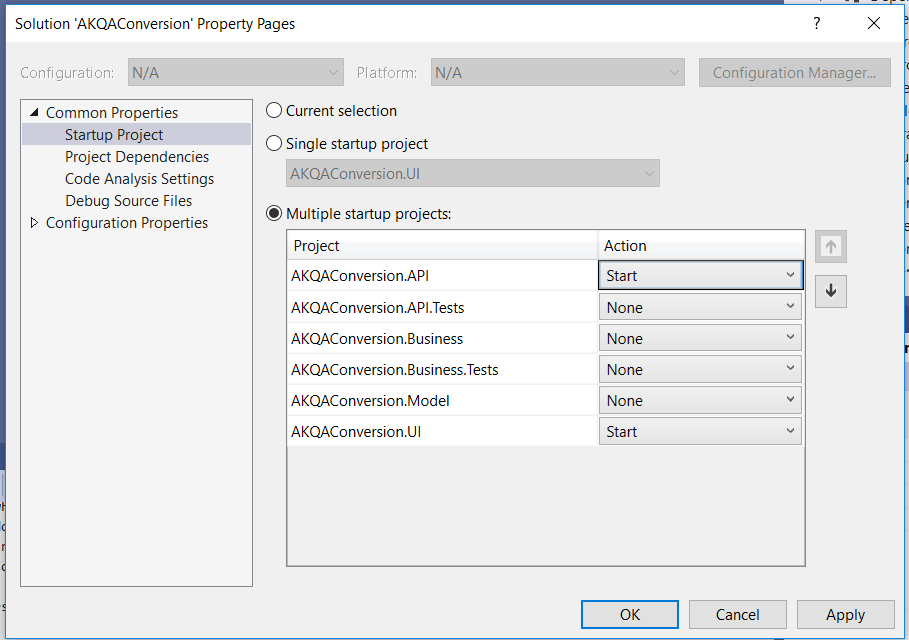
# Running the solution as Multiple Startup Projects

The solution was initially created in Visual Studio 2017. However, after encountering licensing issues, the developer downloaded VS 2019 to continue the coding exercise.

If the solution looks like this on loading:-



Please set the solution properties to Multiple startup.



Right click on AKQAConversion Solution 🡪 go to properties 🡪 set multiple startup projects as indicated in screenshot.

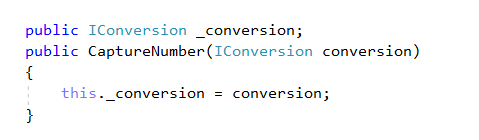
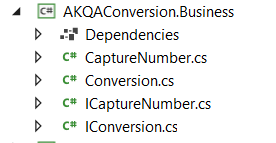
Action on AKQAConversion.API and AKQAConversion.UI should be set to Start

The API service runs at <http://localhost:58592/> and

UI runs at <http://localhost:64551/>

# Architecture

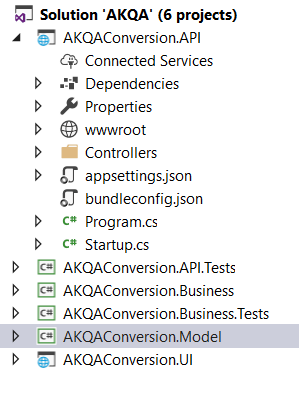
This application follows the service-oriented architecture where every method is accessed using an interface. Classes are accessed and instantiated by means of injecting dependencies



Advantages of this approach:-

1. Adding level of abstraction and reducing the interdependencies between the classes.
2. Promotes reusability – services can be reused
3. Flexibility – Services can be added/ split as application becomes more complex.

The solution is structured as follows: -



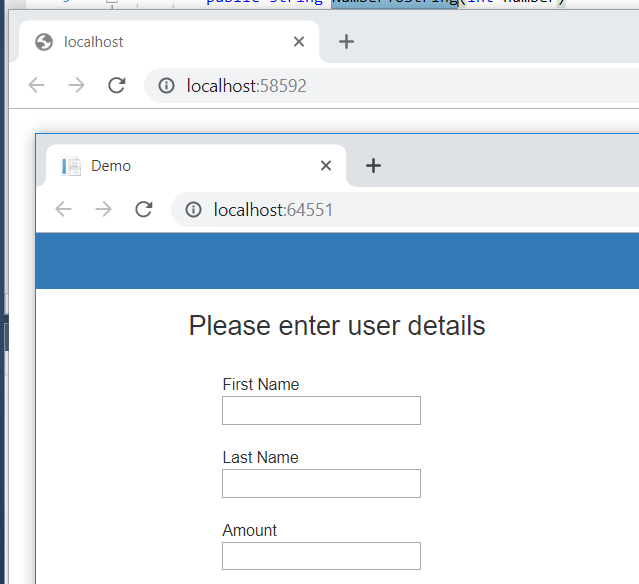
1. AKQAConversion.UI – Javascript UI which captures user input, calls the API service GET method and displays results on the browser.
2. AKQAConversion.API – The API service, with HTTP Get method – that calls the business layer method to compute and return amount value in words. This API results are consumed by UI.
3. AKQAConversion.Model – User Model which is utilized by the controller method
4. AKQAConversion.Business – Implements with the conversion logic by calling a third-party Utility.
5. Unit Test Projects:
   1. AKQAConversion.API.Tests
   2. AKQAConversion.Business.Tests

# Framework

1. UI, API and Test projects are running in Visual Studio Core 2.0 framework
2. Business and Model projects are of Library Project types and are utilizing .NET Standard 2.0

# Dependency Injection

.NET core supports dependency injection. This application uses the inbuilt DI.

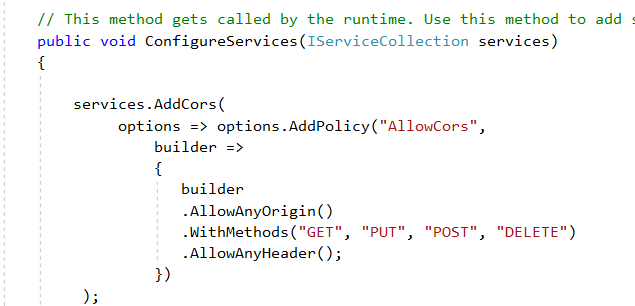


# CORS Policy

The service and UI run on different URL’s (i.e. different origins).

Single origin policy is very restrictive and thus CORS had to be enabled within Startup.cs of AKQAConversion.API project, to allow the UI to gain access to a service of different origin.

.NET Core CORS Middleware handles cross-origin requests:-



# AKQAConversion.Business

Business Layer handles the application logic.

There are 2 classes

1. Conversion.cs – Responsible for interacting with third party Utility dll. NumberToString method accepts an Integer input and returns string value of the number as output
2. CaptureNumber.cs – Responsible for composing a complete text, by splitting number on its decimal place and determining the dollar and cent value. This class consumes interface to access Conversion.NumberToString method.

# AKQAConversion.Model

This has been designed as an independent project, so that any project can utilize classes from the Model project.

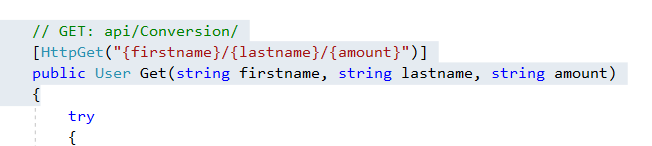
The app has one User class – with FirstName, LastName, Amount (double) and AmountInText properties. In this solution, the API project is using the Model class and returning the corresponding json output for UI consumption.

{"firstName":"Test","lastName":"User","amountinText":"one hundred dollars","amount":100.0}

# AKQAConversion.API

The API controller ConversionController has a get method to return response to the UI. Route parameters are being passed to this method and they can be encrypted for security.

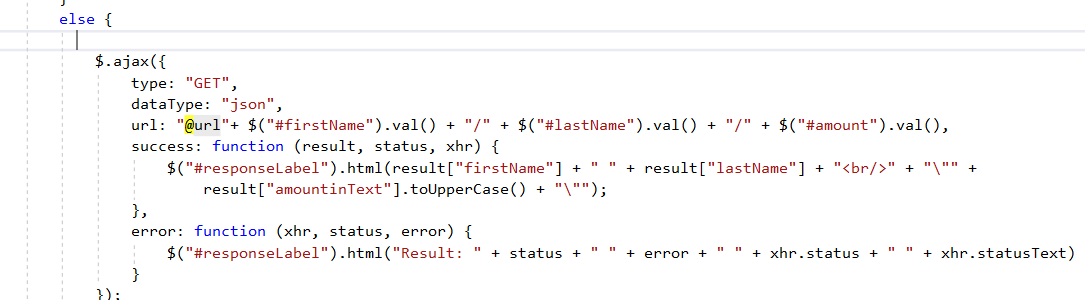
<http://localhost:58592/api/Conversion/Test/User/100.00>



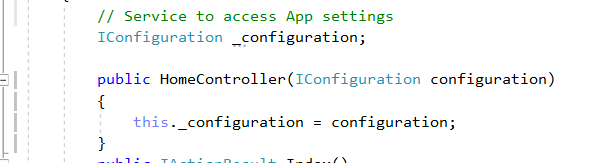
# AKQAConversion.UI

UI project calls Index.cshtml.

UI Access API by means of AJAX call



HomeController – uses IConfiguration to reach API URL from appsettings.json



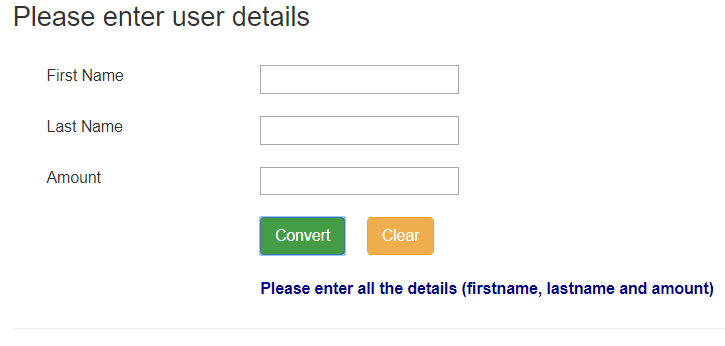
# AppSettings.JSON

The API service URL has been defined in UI projects appsettings.json file

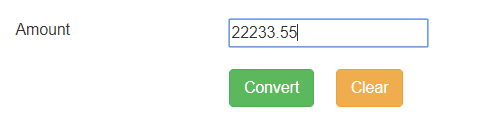


UI Validations

Basic JavaScript validations ensure that all inputs have been filled before calling the API.



Validations have been added to enter amount upto 2 decimal places

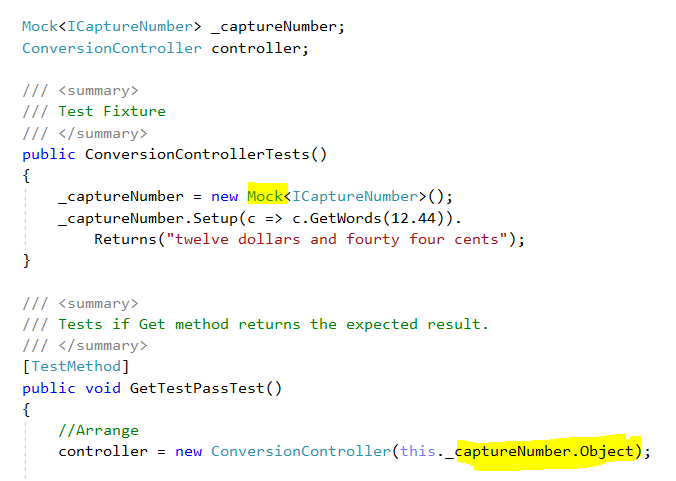


What’s missing – Validation to disallow 0 as the first digit after a decimal point.

# Unit Tests and Mocking Framework

Business and API projects are backed with Unit tests.

Interface are mocked within unit tests, using Moq Mocking library.



**Please note:-**

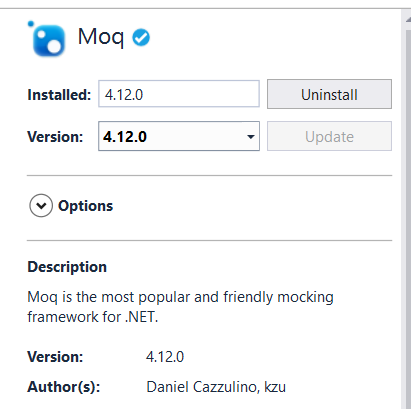
Classes inheriting from Interfaces have their own test cases.

Interfaces are mocked for classes utilizing interface via dependency injection.

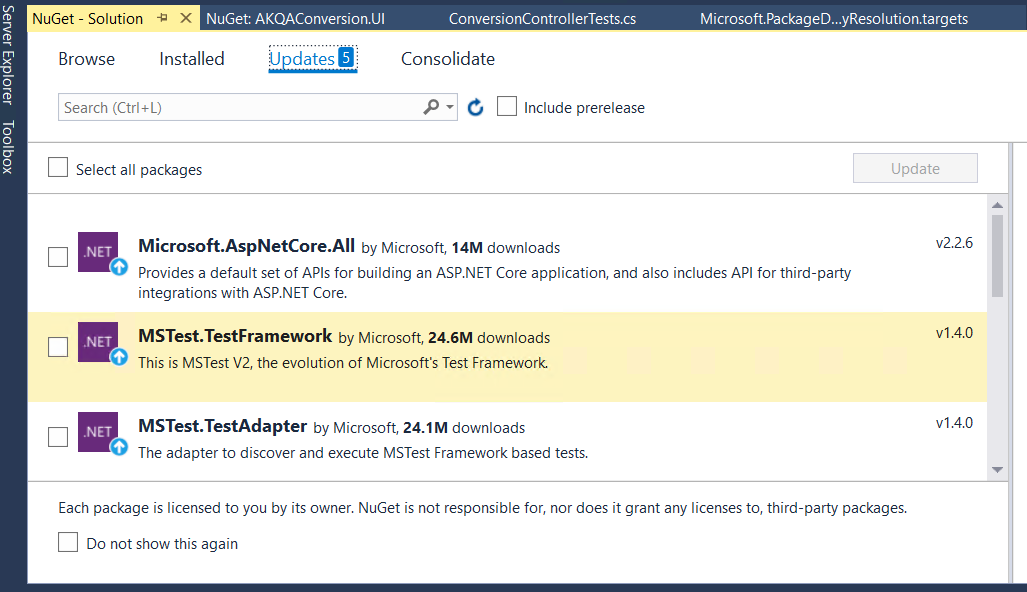
# Nuget

Nuget packages used in this project:-

1. Utility – Third party number to word conversion tool.
2. Moq – For Unit Testing Framework



# Errors while building project



In case of errors while building the solution

Click on Solution🡪 Manage Nuget and install all Updates.