How to use Github Copilot and Visual Studio Code to write and test a C# app

A beginner's guide to using AI-powered code suggestions and a popular code editor

# Introduction

Github Copilot is a new AI-powered tool that can help you write code faster and smarter. It works as an extension for Visual Studio Code, a popular and versatile code editor. In this guide, you will learn how to use Github Copilot and Visual Studio Code to write a sample application in C# that performs a series of mathematical functions and then write and run Xunit tests to test the app.

# Prerequisites

Before you start, you will need the following:

* A Github account and access to Github Copilot. You can sign up for the waitlist here: https://copilot.github.com/
* Visual Studio Code installed on your computer. You can download it here: https://code.visualstudio.com/
* The C# extension for Visual Studio Code. You can install it from the Extensions tab in Visual Studio Code or from here: <https://marketplace.visualstudio.com/items?itemName=ms-dotnettools.csharp>  
  A screenshot of a computer

  Description automatically generated
* The .NET 7/8 SDK installed on your computer.
* The Xunit extension for Visual Studio Code. You can install it from the Extensions tab in Visual Studio Code or from here: <https://marketplace.visualstudio.com/items?itemName=formulahendry.dotnet-test-explorer>  
  A screenshot of a computer

  Description automatically generated

# Prompt

1. Let’s ask Copilot how to get started. Prompt for instructions on how to build a solution and run some xunit tests. Type something like this: **Give step-by-step instructions how to build a solution containing a unit test project and source code project in .NET Core using dotnet test and xunit. Create a Service called NumberService that contains methods named IsPrime, CalculateAverage, IsEven, and IsDivisibleByThree. Create tests in xunit to test each of the methods.**  
     
   A black screen with white text

   Description automatically generated
2. Copilot responds with instructions on how to get started. They should look something like this:  
     
   A screenshot of a computer

   Description automatically generated  
   A screenshot of a computer program

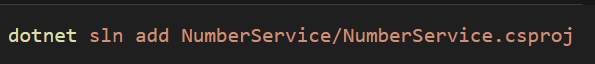
   Description automatically generated  
   A screenshot of a computer program

   Description automatically generated
3. Let’s follow the steps that Copilot provided.

# Creating a C# project

1. Open Visual Studio Code and create a new folder for your project.
2. First, we want to create a new solution. Open the terminal in VS Code and enter the following command to create a new solution: **dotnet new sln -n MySolution  
   A black screen with white text

   Description automatically generated**
3. Now we will create a new .NET Core class library project. Run the following command: **dotnet new classlib -n NumberService.** The new class library will contain the code to be tested  
   A black background with white text

   Description automatically generated
4. Add the **NumberService** project to the solution by running **dotnet sln add NumberService/NumberService.csproj  
   **
5. Next, we will create NumberService. **Open** the Class.cs file in the NumberService project and replace the existing code with the code that Copilot Suggested in step #3. You can hover over the code suggested in Copilot Chat and click “**Copy**”.
6. **Rename** Class.cs to NumberService.cs and **Save.** The file should look something like this:  
     
   A screenshot of a computer program

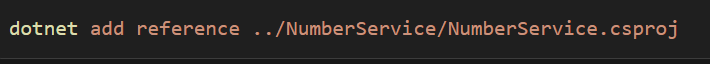
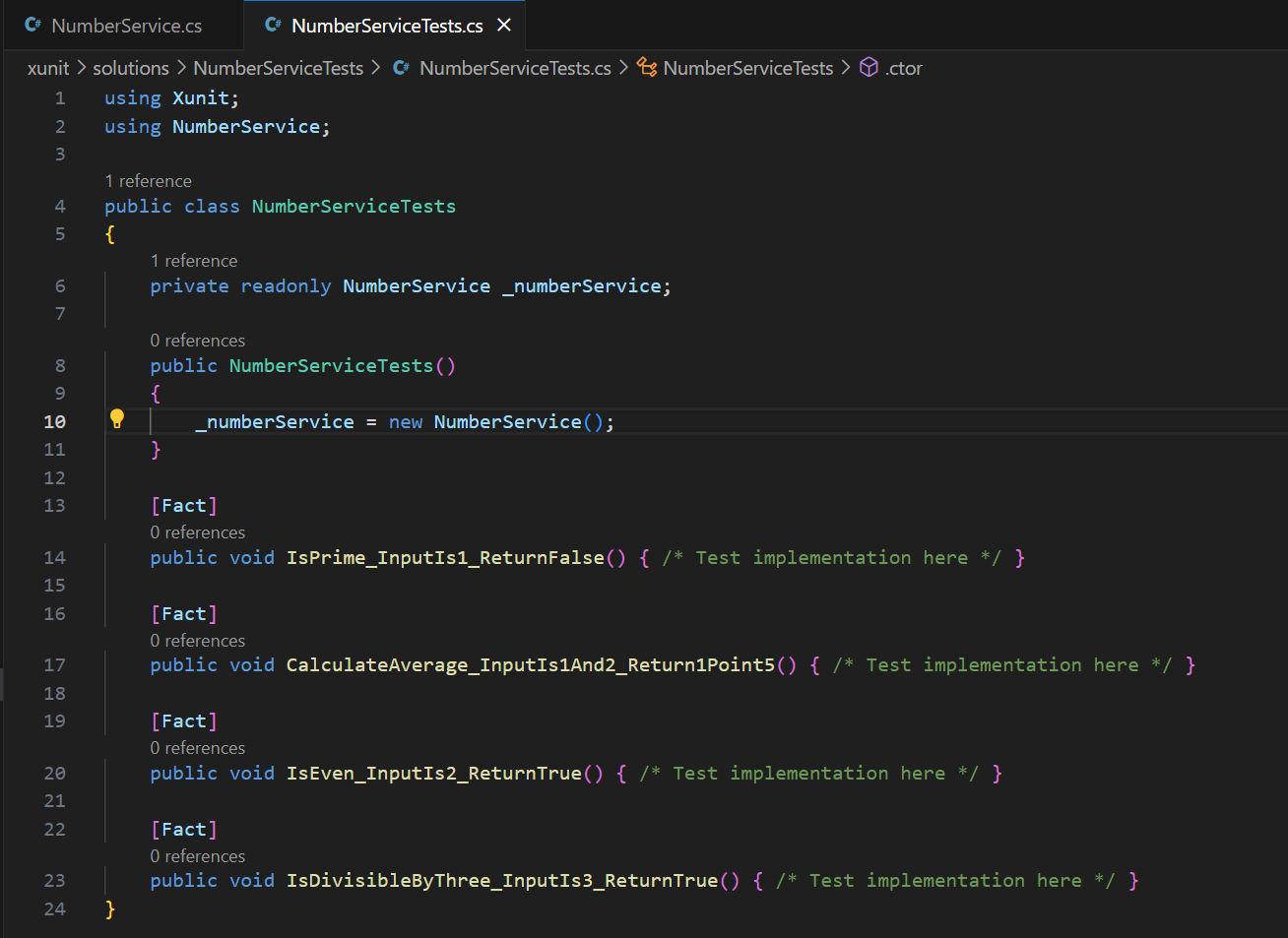
   Description automatically generated
7. Next, we want to implement each of the methods according to their requirements. Let’s ask Copilot for some help with this. **Click** in the IsPrime method and prompt copilot to implement the method by typing something like: **// Return true if the number is a prime number.** Enter a **Return + Tab** a few times and Copilot will autocomplete some new lines. You may need to prompt Copilot to get it going with the implementation. I had to type: **if (number**. Play around with it, and you should see something similar to this:  
     
   A computer screen shot of a program

   Description automatically generated
8. Repeat this process for the CalculateAverage, IsEven, and IsDivisibleByThree methods. By the end, you should have something like this:  
     
   A computer screen shot of a program

   Description automatically generated

# Writing and running Xunit tests

1. Next, we will create a new .NET Core xUnit test project. Run the command **dotnet new xunit -n NumberServiceTests**A black background with white and blue text

   Description automatically generated
2. Add the NumberServiceTests project to the solution with the command **dotnet sln add NumberServiceTests/NumberServiceTests.csproj**
3. **Add** a reference to the NumberService project. **Navigate** to the NumberServiceTests directory and run the following command: **dotnet add reference ../NumberService/NumberService.csproj**
4. Now let’s create the tests. Open the UnitTest1.cs file in NumberServiceTests and replace the code with the code that Copilot Suggested in step #6. You can hover over the code suggested in Copilot Chat and click “**Copy**”.
5. **Rename** UnitTest1.cs to NumberServiceTests.cs and **Save.** The file should look something like this:  
   
6. Now, we need to implement the Unit Tests according to their requirements. Let’s ask Copilot for some help with this. **Click** in the IsPrime\_InputIs1\_ReturnFalse method and enter **Return + Tab** a few times and Copilot will autocomplete some new lines. You may need to prompt Copilot to get it going with the implementation. Play around with it, and you should see something similar to this:  
   A computer screen shot of text

   Description automatically generated
7. Repeat this process for the CalculateAverage\_InputIs1And2\_Return1Points, IsEven\_InputIs2\_ReturnTrue, and IsDivisibleByThree\_InputIs3\_ReturnTrue methods. By the end, you should have something like this:  
   A screenshot of a computer program

   Description automatically generated
8. Finally, it’s time to run the tests. Navigate back to the root directory of your solution and run **dotnet test** to run the tests. This will build the solution and run all the tests.  
   A black background with orange letters

   Description automatically generated
9. Hopefully, your tests run! The first time I ran this command, I got an error:  
   A screen shot of a computer

   Description automatically generated
10. Fix the error(s) if you have any. For me, the fix was to replace references to NumberService with NumberService.NumberService to refer to the class within the namespace to resolve ambiguity between the namespace and the class.
11. When you can successfully run the command, the results will be displayed in the terminal and look something like this:  
    A screen shot of a computer program

    Description automatically generated
12. Fix any failing tests and repeat this process to add new tests as needed.

# Conclusion

In this guide, you learned how to use Github Copilot and Visual Studio Code to write a sample application in C# that performs a series of mathematical functions and then write and run Xunit tests to test the app. You also learned how to create a test project, write test methods, and run them using Xunit. You can use these skills to write and test your own C# applications using Github Copilot and Visual Studio Code.