Christopher Evans (5829035234)

Construction document

Contents

[Introduction 1](#_Toc83983790)

[Roadblocks 1](#_Toc83983791)

[Setting up Continuous Integration 2](#_Toc83983792)

[Creating documentation with Sandcastle 6](#_Toc83983793)

[Limitations 8](#_Toc83983794)

[Future Improvements 8](#_Toc83983795)

[Summary 9](#_Toc83983796)

# Introduction

The purpose of this document is to detail the roadblocks of my synoptic project, the process of setting up appropriate continuous integration and summarize the results of the work produced.

# Roadblocks

One issue I ran into when setting up my Unit Test project was that Visual Studio was unable to set up a project reference, displaying the following error:

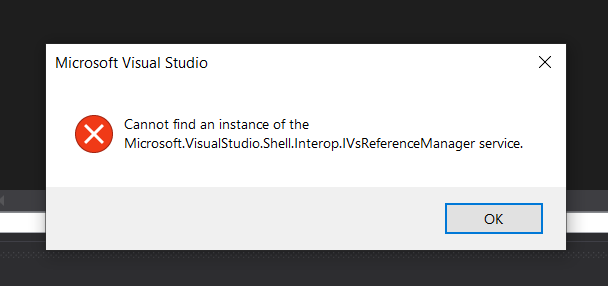


Figure : Error when trying to add project reference

Unsure what this meant, I looked up the error message typing it word for word into my search engine. After digging around, I found an article that detailed the steps to add the missing dll. I opened a developer command prompt and ran the following command:

*gacutil -i Microsoft.VisualStudio.Shell.Interop.11.0.dll*

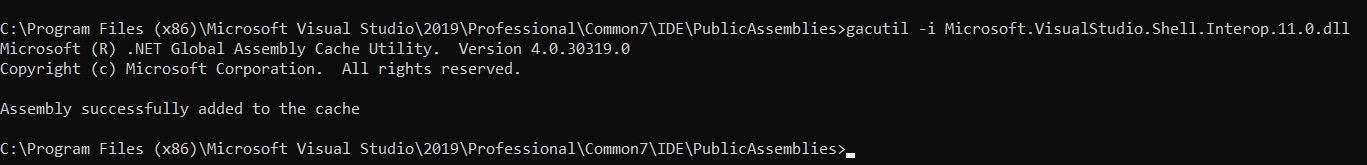


Figure : Loading assembly from command line

I then restarted visual studio and tried again and I was able to add the project reference.

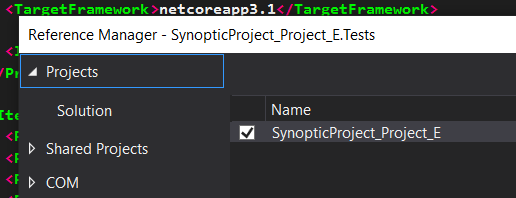


Figure : Successfully adding project reference

# Setting up Continuous Integration

Because I’m hosting the source code on Github, I’m able to set up Github Workflows to allow me to build and test my source code as I push it to the repository. This allows me to continually integrate my code, ensuring it builds on other machines.

Setting it up was easy, I just clicked on the “Actions” tab of the repository and selected my default workflow. The .NET one suited my purposes best.

Text

Description automatically generated

Figure : Github default .NET workflow

I accepted the default file and committed it to a separate branch named “implement-workflows”. I then created a pull request into the “master” branch to trigger the first build.

A screenshot of a computer

Description automatically generated with medium confidence

Figure : Build triggers in pull request

The build was queued and I was able to see it in the Pull request as an amber icon, with a link to view the details and watch the build. After a few seconds, the build failed at the “Restore dependencies” step.

A screenshot of a computer

Description automatically generated with medium confidence

Figure : Build error when restoring dependencies

I read the error message and saw that it couldn’t find a solution file. This is because the build working directory was set to the root of the repository, and the solution file is in the “Source” directory.

To amend this, I created an environment variables for the solution file:

Text

Description automatically generated

Figure : Adding environment variable

I then specified it in my build and committed the changes.

A screenshot of a computer

Description automatically generated with medium confidence

Figure : Using environment variable in build scripts

This time, it got past the “Restore dependencies” step and built successfully, but the unit tests failed. Once again, I inspected the error:

Text

Description automatically generated

Figure : Build error on unit tests

This time it specified that it could not find .NET version 3.1.0 but was able to find 5.0.10. I knew that I was working with .NET core 3.1 so .NET 5.0 was the wrong version. I returned to the YAML configuration file and updated the .NET version:

A screenshot of a computer

Description automatically generated with medium confidence

Figure : Changing .NET version

I committed once more, and this time the build succeeded.

A screenshot of a computer

Description automatically generated with medium confidence

Figure : Successful build

From now on, I can create pull requests into the “master” branch and have them checked via Github workflows to ensure everything is in working order.

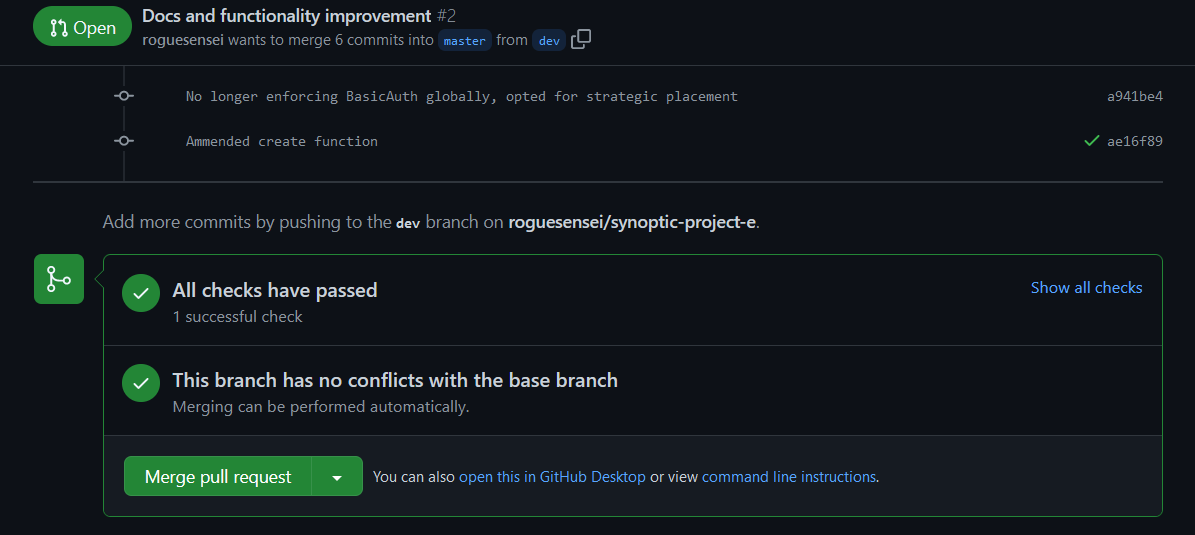


Figure : Passing check in Pull Request

# Creating documentation with Sandcastle

Visual studio contains support for XML comments, which allow developers to document classes and methods.

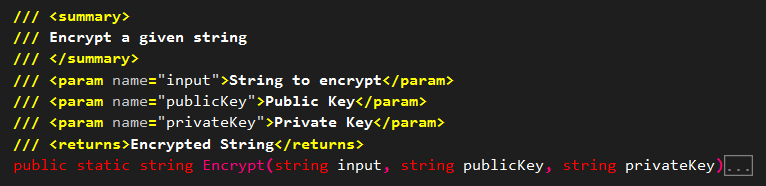


Figure : XML comments in visual studio

These then get picked up by the in intellisense to help other developers use and understand:

* The definition of the method – i.e. what does it do
* Parameters – a short description of the parameters required
* The return value – what the return value is and what it means

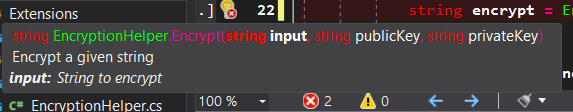


Figure : Documentation picked up by intellisense

While these are helpful, it can also be helpful to have a website or word document containing this documentation. This can allow developers to see, in once place:

* The methods available in a class
* Links to classes
* Class relationships

Visual studio provides functionality to export the XML comments on build under the project properties, in the build tab.

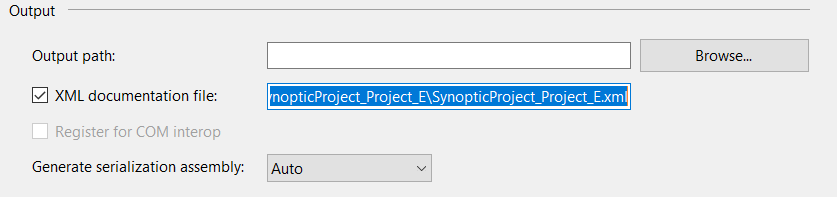


Figure : Generating XML documentation with visual studio

This file can be read by the sandcastle program to generate documentation from the XML data and the output DLL of the project.

Inside sandcastle, I specify the output type and the language(s) to optimize for. Sandcastle supports HTML generation so the documentation can be hosted as a website. For this project though, I will create an open document (.docx).

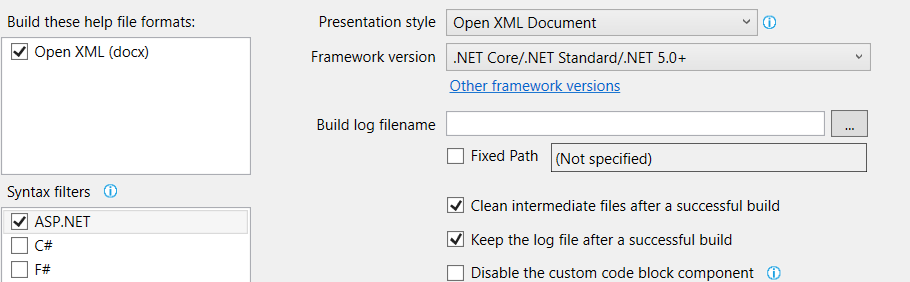


Figure : Configuring sandcastle output

Next, under “Documentation Sources”, I can add a source by navigating to the project output directory and selecting the output DLL. The generated XML comments will automatically be picked up.

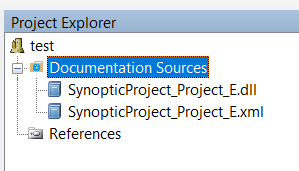


Figure : Setting sandcastle documentation sources

Finally, I can build the project with CTRL+SHIFT+B and the documentation will be generated.

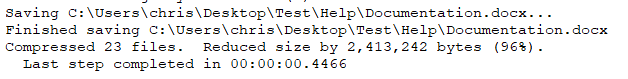


Figure : Successful build output from sandcastle

I can then view the output documentation and make any styling changes/manual edits.

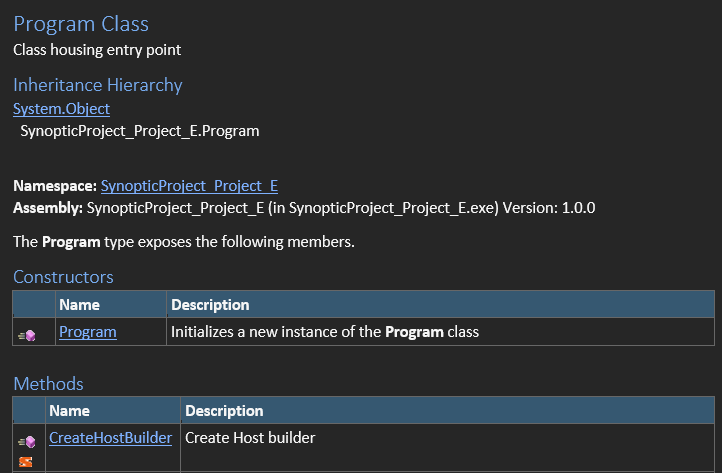


Figure : Snippet of default sandcastle generation

# Limitations

One limitation of the API is the inability to update or delete users despite laying it out in the design document. This means that as it currently stands, once a user is created, it is permanent, so mistakes in the credentials cannot be amended. These were not implemented due to the time constraints of the project.

Another limitation is that all of the user fields are “required” fields, making email addresses/mobile numbers mandatory. This could be amended in the future.

# Future Improvements

On the contrary, despite the lack of the ability to edit or delete users, I have been able to write the code in a way that would make implementing these features easy as it would be as simple as adding the methods to the DAL.

Another future improvement would be the ability to store user credits and place orders for food via the API. This could save the user’s having to use their own money to purchase food and credits could be automatically added via a microservice of manually edited by an admin user.

Finally, while unit testing is present in the project, it would be an improvement to unit test the endpoints via mock databases.

# Summary

In summary, I am happy with the final project and acknowledge that despite it’s shortcomings and limitations, I can identify the areas for improvement to expand upon the project beyond the initial requirements, thus iterating on the projects life cycle.