Christopher Evans (5829035234)

Test document

Contents

[Introduction 1](#_Toc83984366)

[Test Plan 1](#_Toc83984367)

[Test Methods 1](#_Toc83984368)

[Postman 1](#_Toc83984369)

[Unit Testing 3](#_Toc83984370)

[Github Workflows 4](#_Toc83984371)

[Tests 5](#_Toc83984372)

# Introduction

The purpose of this document is to lay out my test plan, detail my test methods and list the results of each individual test made including:

* The expected result and actual result
* Whether or not the test passed
* In the case of a failed test, the action taken in a following test

# Test Plan

I will be testing the REST API in the following 3 ways:

* Manual Testing using Postman
* Unit Testing using Visual Studio
* Build Testing using Github Workflows

# Test Methods

## Postman

Postman is a piece of software used to develop and test API’s. It allows you to build the requests and test each individual endpoint. To test, I run the application as usual and start an ngrok tunnel using Visual Studio’s ngrok extension.

Graphical user interface, application

Description automatically generated

Figure : Starting ngrok tunnel

Ngrok allows me to make a secure connection to localhost, though postman will still accept localhost in the URL field. I can then use the supplied URLs to use as the base url.

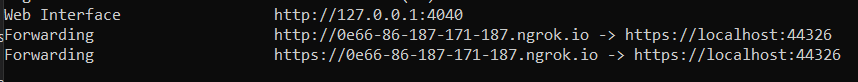


Figure : ngrok output in command line window

I can then plug a request into postman like as follows, and send a request for a result.

Graphical user interface, text, application, email

Description automatically generated

Figure : Sending a request with Postman

I can then make amendments, such as the headers to test each request.

Graphical user interface, text, application, email

Description automatically generated

Figure : Adding Authorization header to postman

## Unit Testing

Using visual studio, I was able to set up a Unit Test project to test units of code, typically the ones that don’t require a database connection such as the “Helper” classes.

Graphical user interface, text, application, chat or text message

Description automatically generated

Figure : Unit tests as they appear in Visual Studio

The unit tests follow the industry standard pattern of Arrange, Act, Assert. For example, consider the following test method:

Text

Description automatically generated

Figure : Example Unit Test

The “Arrange” step, is where I declare my variable(s) to be used as part of the test. The “Act” step is where I Act upon the test. Note that I have named the test “EncryptThenDecryptTest”, thus I base the name of my methods on what I’m testing in the “Act” stage. In this case, I’m encrypting the declared string, then decrypting it to get a result. Finally, the “Assert” stage is where I validate the test. In this case, I’m testing if the decrypted string matches the original string.

## Github Workflows

Since the code is hosted on Github, I’m able to utilize Github’s “Workflows”. This allows me to build and test my code on a virtual machine to ensure that the solution isn’t just buildable on my machine. The workflow is detailed in the .github/workflows/dotnet.yml file:

Graphical user interface, text

Description automatically generated

Figure : Github workflow YAML configuration

The workflow is triggered every time I push or make a pull request to the “master” branch. Note that the solution is being built on ubuntu rather than Windows, my development OS. This doubly ensures that my code won’t just deploy on another machine, but another operating system entirely.

The rest of the file details the steps:

1. Setup .NET – specify version of .NET
2. Restore dependencies – Restore NuGet packages
3. Build – Compile the source code
4. Test – Run unit tests

With workflows in place, it’s easy to see from the commit history which ones passed and which ones failed.

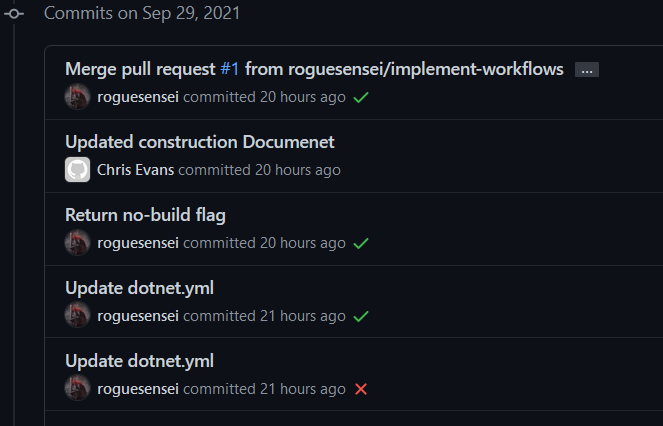


Figure : Git commit history on Github

# Tests

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Number | Description | Type | Expected Result | Actual Result | Pass/Fail |
| 1 | Acquire User information without Authorization Header | Postman | Server to respond with 401 | As expected | Pass |
| 2 | Encrypt a given string, then decrypt it. | Unit | Decrypted string to match the original string. | As expected | Pass |
| 3 | Decrypt a given string, then re-encrypt it. | Unit | Encrypted string to match the original string. | As expected | Pass |
| 4 | Convert string to base64 and back again using extension method | Unit | String converted back from base64 to match original string. | As expected | Pass |
| 5 | Acquire User information with Authorization header without logging in. | Postman | Server to respond with 401 | As expected | Pass |
| 6 | Login a user with empty credentials | Postman | Server to respond 400 and an error message | As expected | Pass |
| 7 | Login a user with invalid credentials | Postman | Server to respond with 401 | As expected | Pass |
| 8 | Login a user with correct credentials | Postman | Server to respond with 200 and a welcome message including the user’s name | As expected | Pass |
| 9 | Acquire user information after logging in | Postman | Server to respond with User information | Server responded 401 (Please Log in) | Fail |
| 10 | Action on test no. 9: modified code in BaseController – made the dictionary housing authenticated users static. | Debug | As with Test no. 9 | As expected | Pass |
| 11 | Send logout message | Postman | Server to respond with 200 and a goodbye message | As expected | Pass |
| 12 | Send logout message while already logged out | Postman | Server to respond with “You are already logged out” | As expected | Pass |
| 13 | Login with user that doesn’t exist | Postman | Server to respond with 401 and to request the user registers | As expected | Pass |
| 14 | Register a new user with missing fields | Postman | Server to respond with 400 and an error message | Responded with 401 and asked to login or register | Fail |
| 15 | Action on test no. 14: modified code in BaseContoller to no longer enforce BasicAuth for every endpoint. Now enforcing on UserController and the Login/Logout methods in the Authentication controller | Debug | As with Test no. 14 | As expected | Pass |
| 16 | Register a new user | Postman | Server to respond with 200 | As expected | Pass |
| 17 | Attempt to register a user that already exists | Postman | Server to respond with 400 | As expected | Pass |
| 18 | Login with newly created user | Postman | As with Test no. 8 | As expected | Pass |
| 19 | Build with Github workflows | Workflows | Build to pass | Build failed | Fail |
| 20 | Action on Test no. 19: specify solution file in the build script | Workflows | As with Test no. 19 | Build succeeded but unit tests failed | Fail |
| 21 | Action on Test no. 20: Specify the correct .NET version | Workflows | Build to still pass but unit tests to also pass | As expected | Pass |
| 22 | Create an admin user as a non-admin | Postman | Server to respond with 403 | As expected | Pass |
| 23 | Create an admin user as an admin | Postman | Server to respond with 200 | Server responded with 400 saying user already existed | Fail |
| 24 | Action on Test no. 23: changed “GetUser()” for existing user validation to be checked if it is null rather than not equal to null | Postman | As with test no. 23 | As expected | Pass |
| 25 | Login with the newly created admin user | Postman | Server to respond with welcome message including admin’s name | As expected | Pass |