😊 Code Coverage – Adding Unit test

In the previous Lab, we discussed how to incorporate **unit testing** into a project within a **Visual Studio Solution**. We also explored how, after checking the code into the repository and running it through the **Azure DevOps pipeline**, we could see a summary of the test execution as part of the pipeline.

Now, in this Lab, I want to introduce another important aspect: **code coverage**.

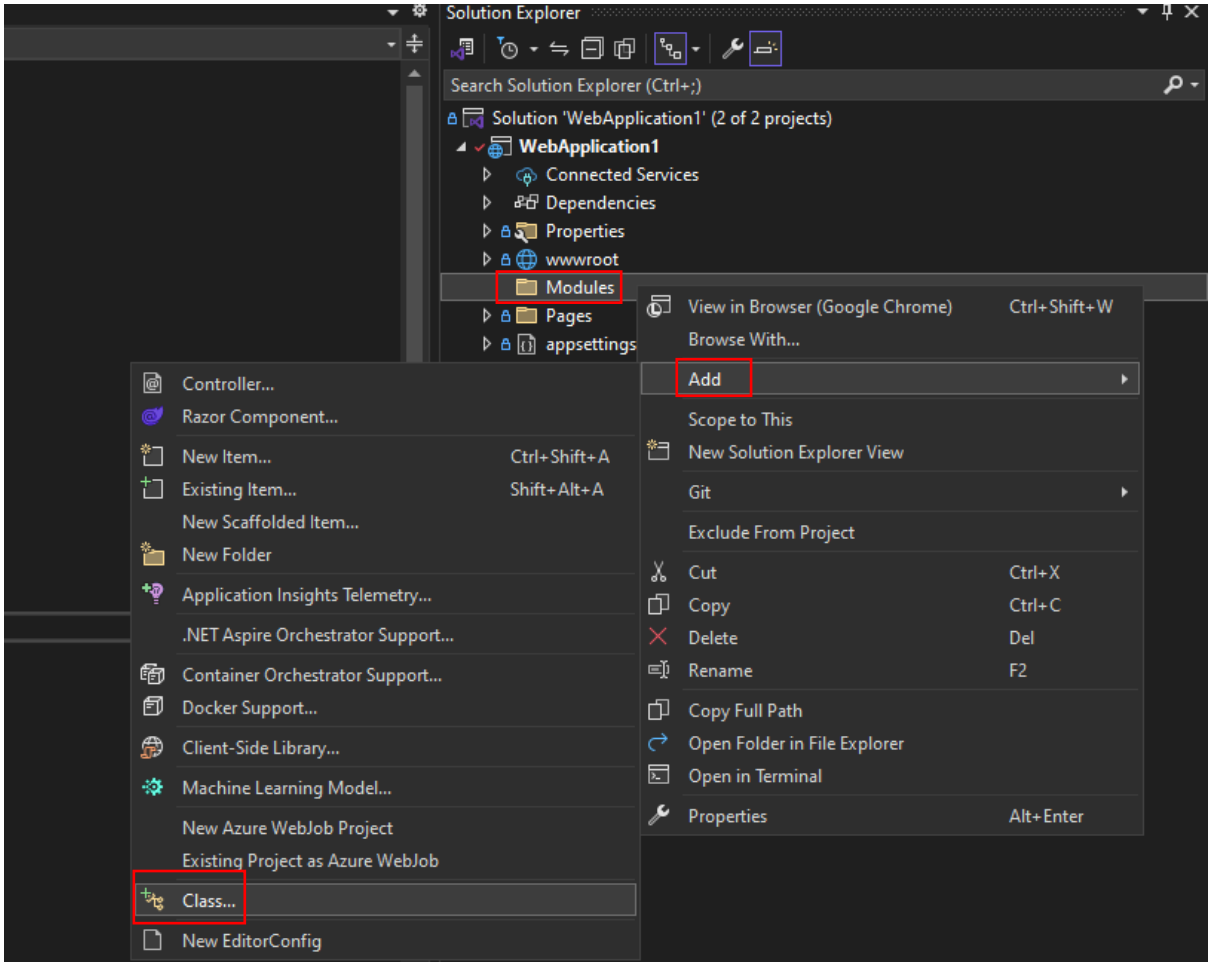
What is Code Coverage?

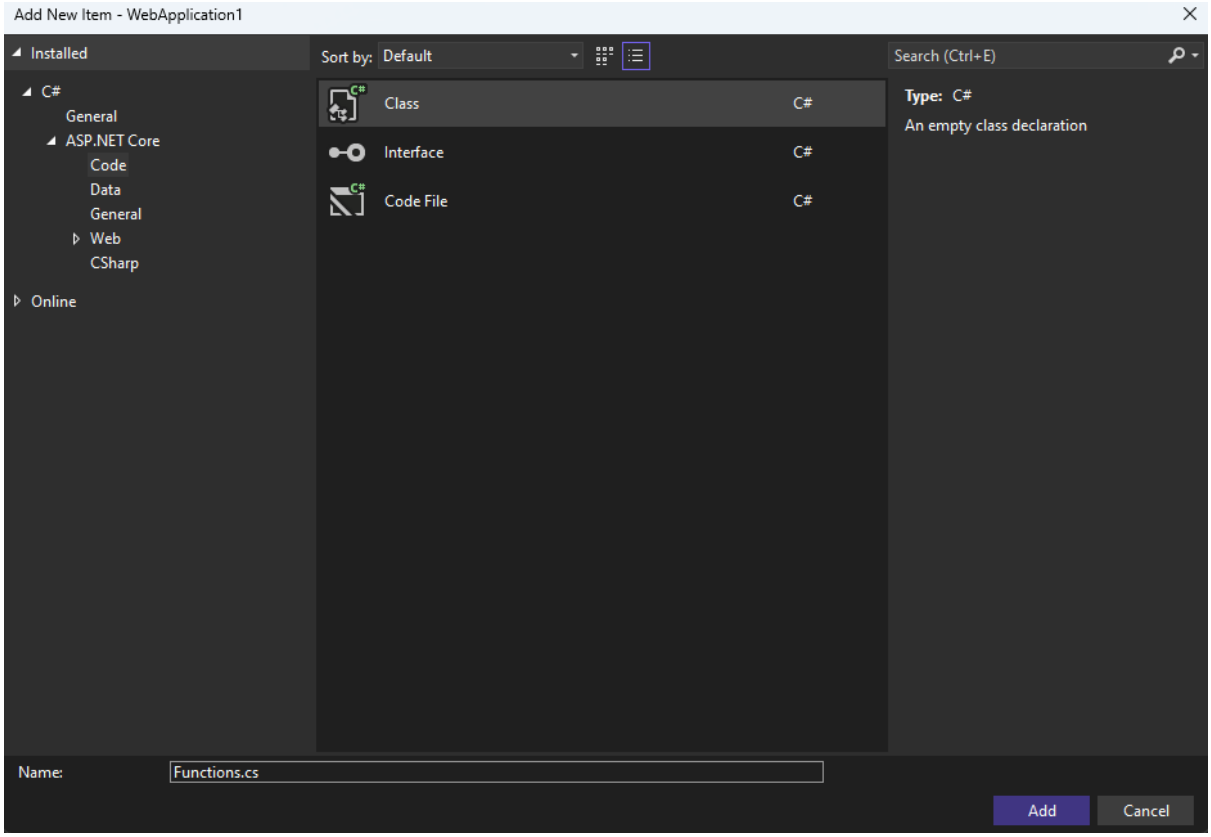
When writing unit tests, the goal is to test different modules of the application. **Code coverage** is a key measure that tells us how much of our actual application code is being tested by these unit tests. In other words, it helps determine the percentage of code that is covered by test cases.

In our previous example, the **web test** we created was very basic and did not actually test any functionality from our **web application**. So, let's now look at a simple example of writing a unit test that verifies functionality within the main web application.

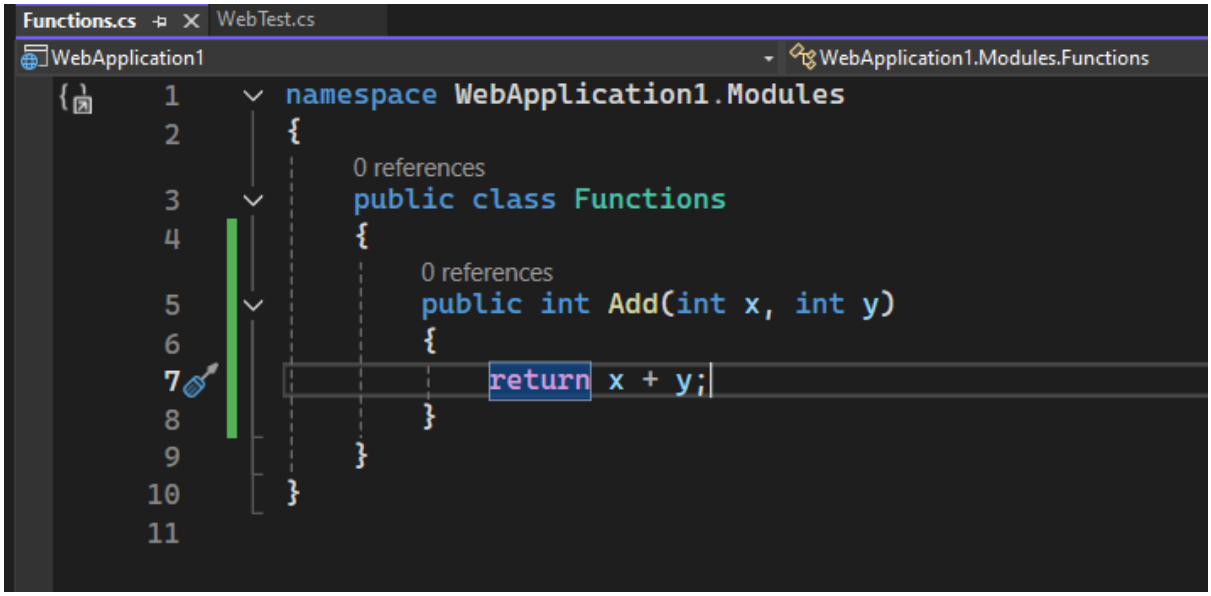
Adding a Function for Testing

1. In the **web application**, I will first **right-click** and create a new folder named **Modules**.
2. Inside this **Modules** folder, I will **add a new class** named Functions.





3. In the Functions class, I will implement a simple method that **adds two numbers together**. This keeps the example straightforward and easy to understand.

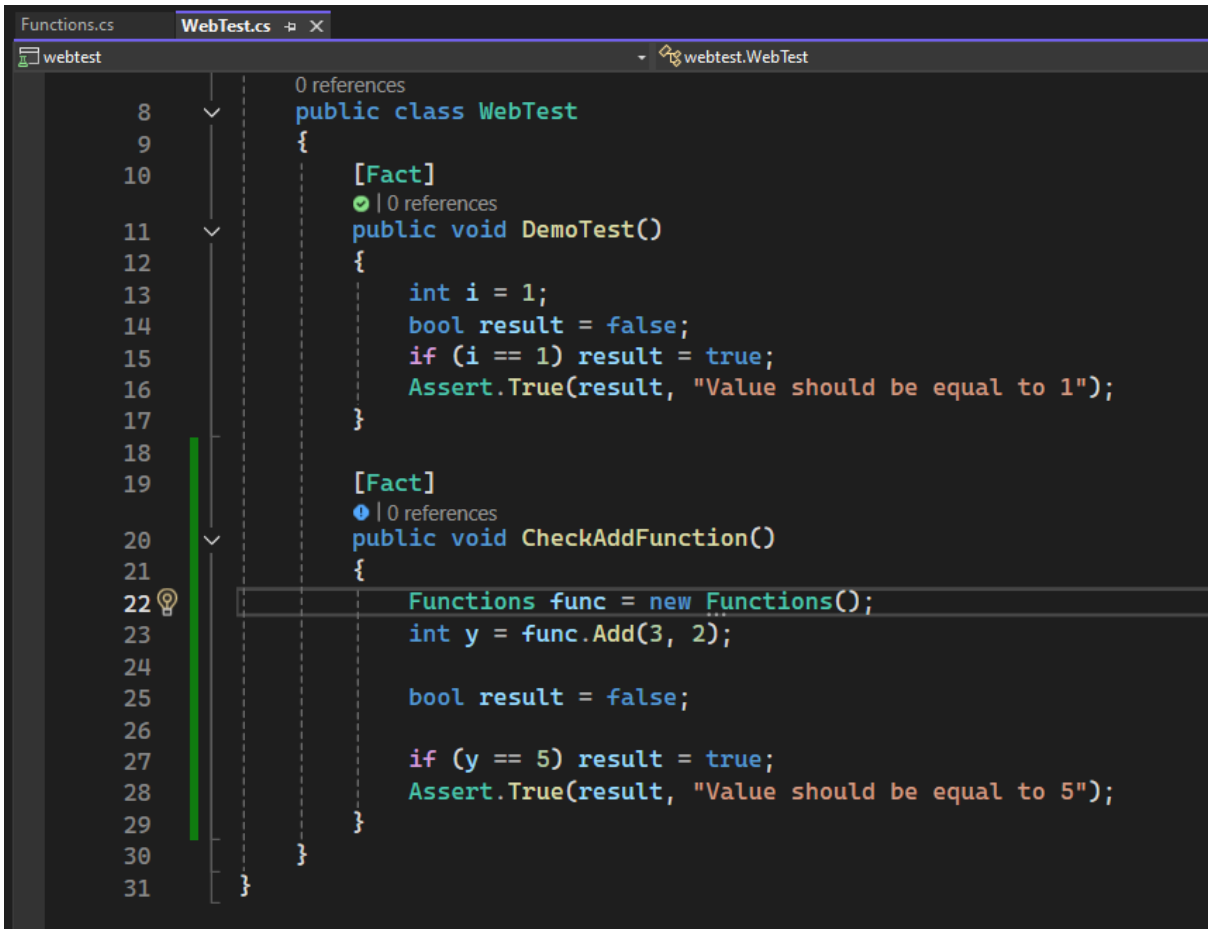


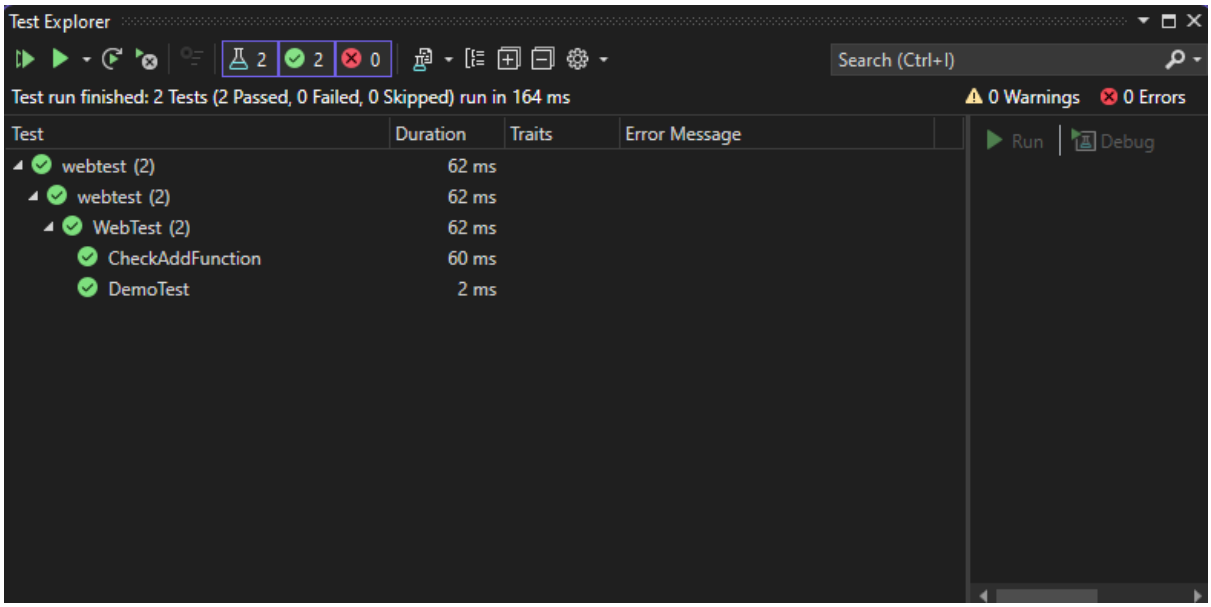
Writing a Unit Test for the Function

1. Now, I will add a **unit test** to verify this method.
2. In the **WebTest** project, I will create a new test method called **CheckAddFunction**.
3. Within this test method:
    - I will **reference** the Functions class from the web application.

- I will **invoke the Add function**, passing 3 and 2 as arguments.
- Since 3 + 2 = 5, I will assert that the returned value is 5.
- If the result is 5, the test will pass. Otherwise, it will fail.

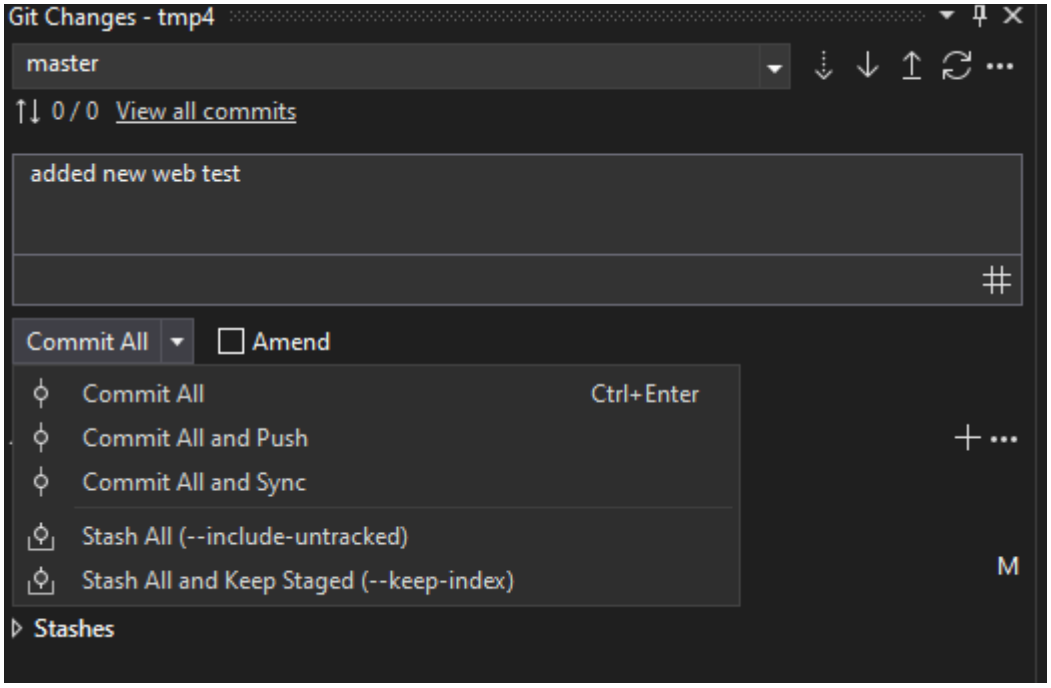
After implementing this, I will **save all changes**, run the test locally, and confirm that it works correctly.



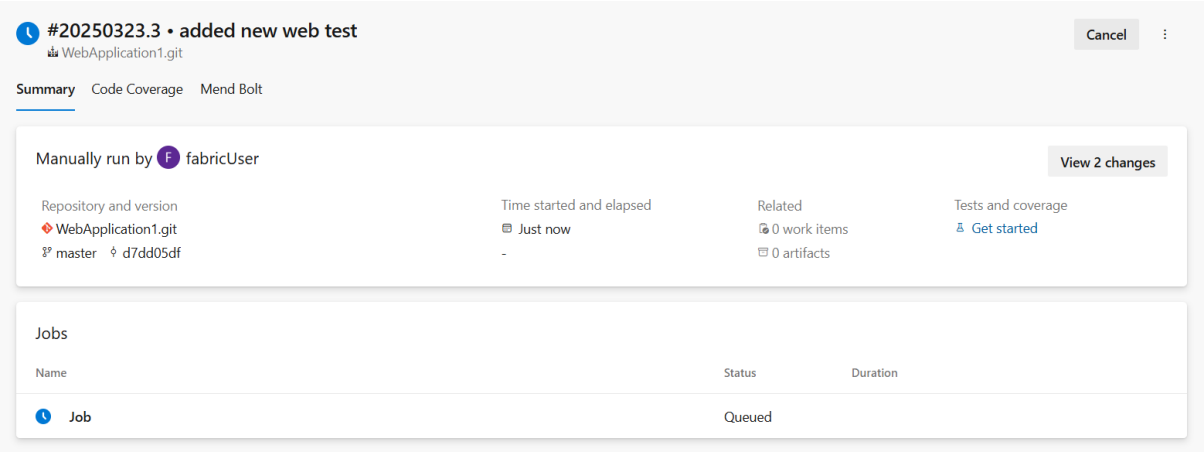


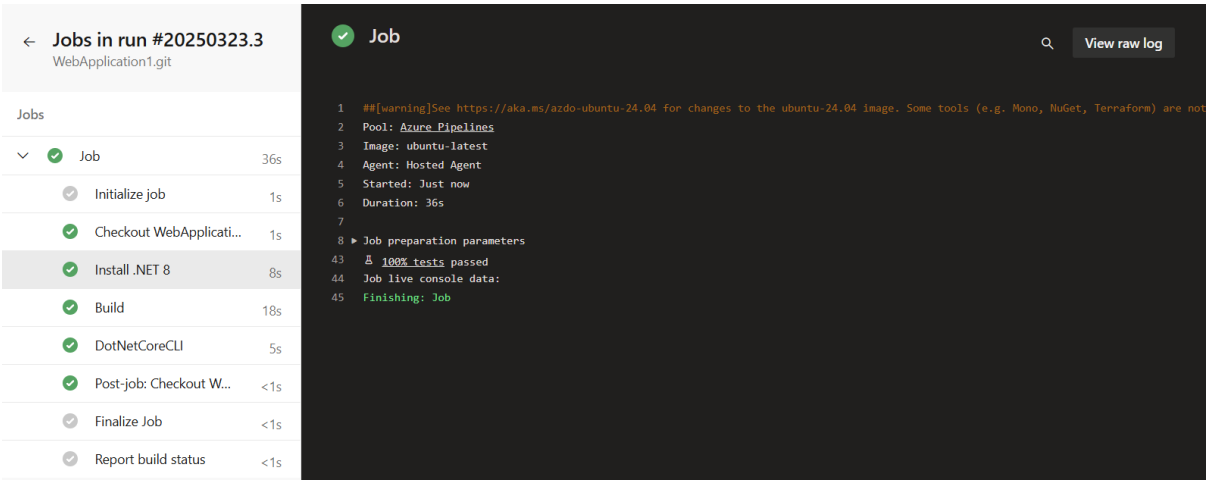
Pushing the Changes and Running Tests in the Pipeline

1. After verifying that the test works locally, I will **commit and push** my changes to the **Git repository**.

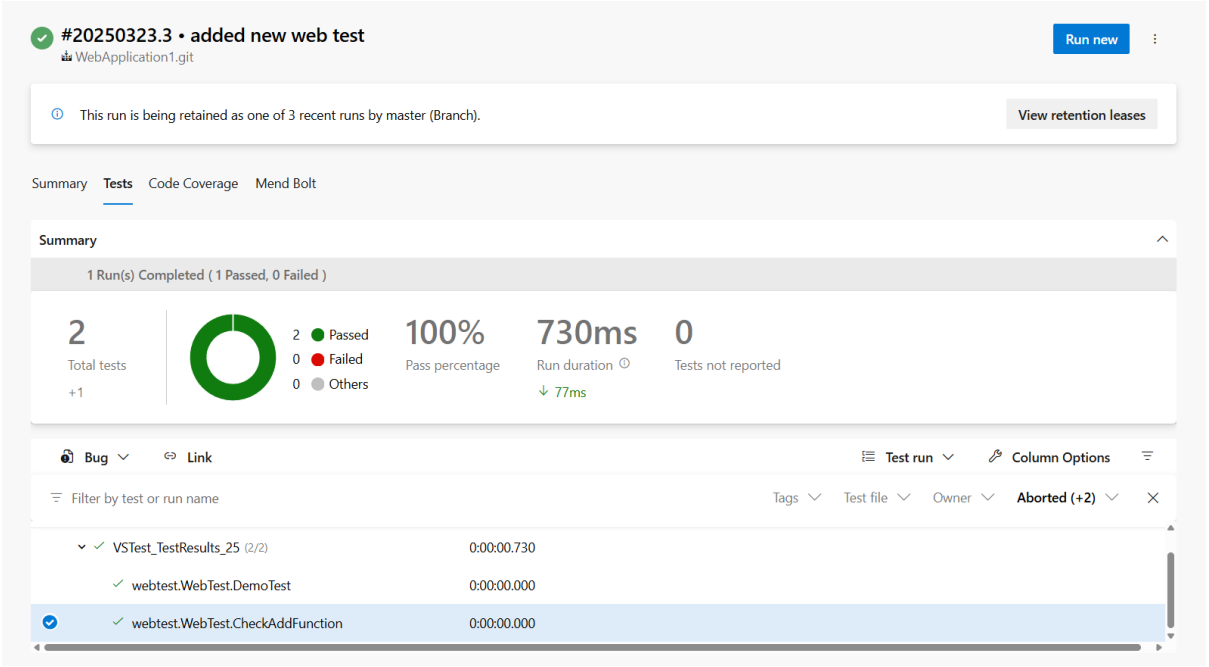


2. This will **trigger a build** in the Azure DevOps pipeline.





3. Once the build is complete, I will navigate to the **test results** in the pipeline.
4. I should see that both my unit tests have passed.



5. By filtering the results to show **passed tests**, I can confirm that both tests were executed successfully.

Conclusion

This is the **first step** in achieving better **code coverage**—by ensuring that **unit tests cover actual functionality** from the main web application. Going forward, increasing test coverage will help improve the reliability and robustness of the application.