بررسی تست نویسی در نرم افزار

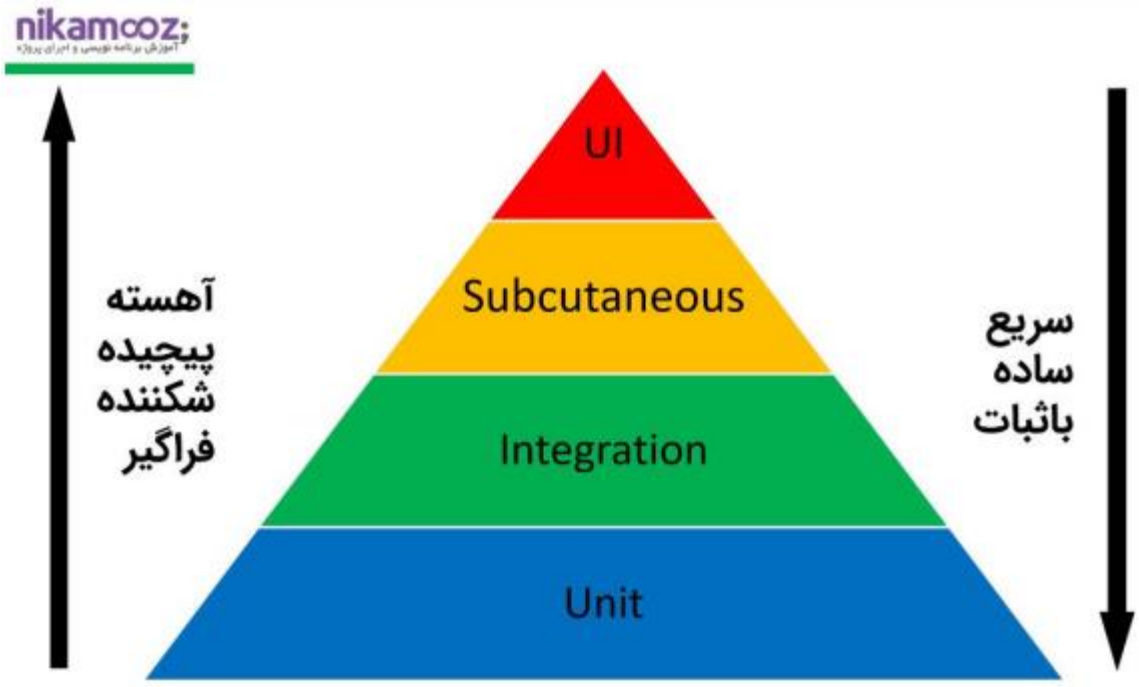
انواع تست ها:

۱. Unit Test
۲. Integration Test
۳. Code Coverage
۴. End-To-End Test
۵. Contract Test
۶. Non-Functional Test

توضیحات:

- Performance testing is a testing method used to determine the speed of a computer, network or devices.
- Load testing simulates real-world load on any application or website.
- Stress testing determines the stability and robustness of the system
- Performance testing helps to check the performance of website servers, databases, networks.
- Load testing is used for the Client/Server, Web-based applications.
- Stress testing is done unexpected test traffic of your website.

هرم تست:



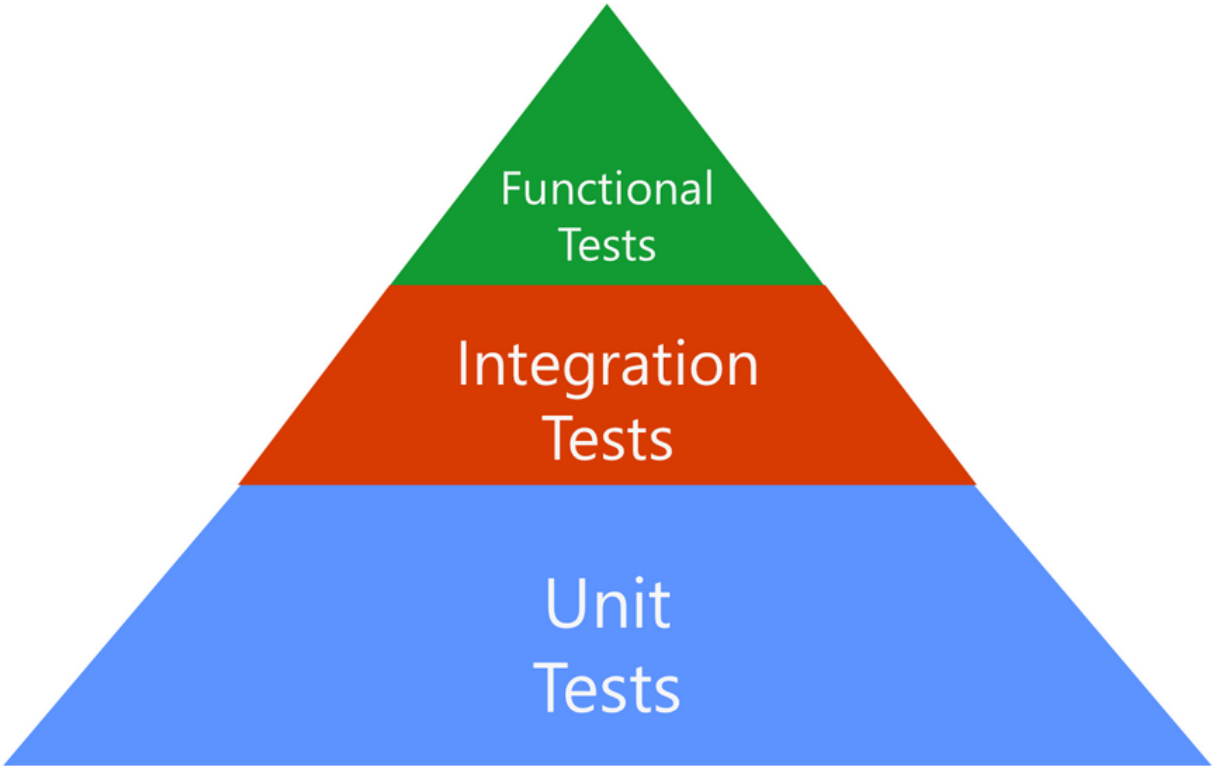


Figure 9-1. Testing Pyramid

۱. بررسی Unit Test

به مثال کدنویسی شده رجوع شود.

۲. بررسی Integration Test

در این سناریو Repository های Product تست شده است:

چون ما یک وابستگی به دیتابیس داریم، یک کلاس SharedDatabaseFixture ایجاد میکنیم تا بتوانیم با DbContext کار کنیم. روال کار بدین صورت است که از دیتابیس Sqlite استفاده میکنیم تا داده‌ها را در زمان اجرای تست نگهداری کنیم. جهت تولید دیتای Fake از پکیج Bogus استفاده شده است.

کلاس SharedDatabaseFixture یکبار در حین اجرای تست توسط IClassFixture نمونه سازی میشود به صورت Shared استفاده میشود تا زمانی که اجرای تست به اتمام برسد.

https://www.c-sharpcorner.com/article/implementing-unit-and-integration-tests-on-net-with-xunit/

تست Application های دارای EF Core:

https://learn.microsoft.com/en-us/ef/core/testing/#the-fixture

۳. تولید Coverage Report:

در ابتدا باید ابزار **dotnet-reportgenerator-globaltool** **را نصب نماییم. برای این منظور در CMD فرمان زیر را اجرا میکنیم:**

Dotnet tool install –g dotnet-reportgenerator-globaltool

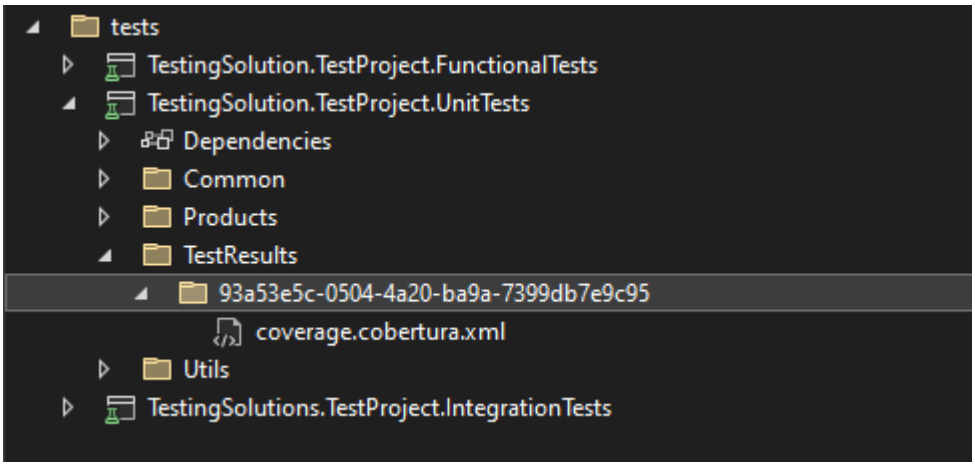
```
C:\WINDOWS\system32>Dotnet tool install -g dotnet-reportgenerator-globaltool
You can invoke the tool using the following command: reportgenerator
Tool 'dotnet-reportgenerator-globaltool' (version '5.1.22') was successfully installed.

C:\WINDOWS\system32>
```

فرمان زیر را در مسیر پروژه Test اجرا میکنیم تا اطلاعات Code Coverage جمع آوری شود:

Dotnet test –collect:”XPlat Code Coverage”

```
C:\Users\Raven\source\repos\Testing\StoreCleanArchitecture-NET-2.1.0\tests\Store.UnitTests>dotnet test --collect:"XPlat Code Coverage"
  Determining projects to restore...
  All projects are up-to-date for restore.
  Store.ApplicationCore -> C:\Users\Raven\source\repos\Testing\StoreCleanArchitecture-NET-2.1.0\src\Store.ApplicationCore\bin\Debug\netstandard2.1\Store.ApplicationCore.dll
  Store.UnitTests -> C:\Users\Raven\source\repos\Testing\StoreCleanArchitecture-NET-2.1.0\tests\Store.UnitTests\bin\Debug\net6.0\Store.UnitTests.dll
Test run for C:\Users\Raven\source\repos\Testing\StoreCleanArchitecture-NET-2.1.0\tests\Store.UnitTests\bin\Debug\net6.0\Store.UnitTests.dll (.NETCoreApp,Version=v6.0)
Microsoft (R) Test Execution Command Line Tool Version 17.5.0 (x64)
Copyright (c) Microsoft Corporation.  All rights reserved.

Starting test execution, please wait...
A total of 1 test files matched the specified pattern.

Passed!  - Failed:     0, Passed:    13, Skipped:     0, Total:    13, Duration: 120 ms - Store.UnitTests.dll (net6.0)

Attachments:
  C:\Users\Raven\source\repos\Testing\StoreCleanArchitecture-NET-2.1.0\tests\Store.UnitTests\TestResults\53345c3d-b41c-45eb-957c-41a9ebfc6a37\coverage.cobertura.xml

C:\Users\Raven\source\repos\Testing\StoreCleanArchitecture-NET-2.1.0\tests\Store.UnitTests>
```

بعد از اجرای دستور یک پوشه در مسیر پروژه تست ایجاد میشود:

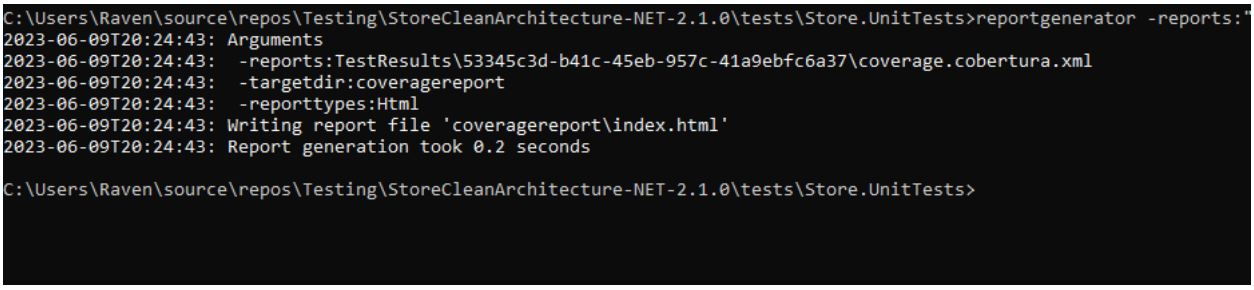


حالا که اطلاعات جمع اوری شد برای تولید گزارش از فرمان زیر استفاده میکنیم:

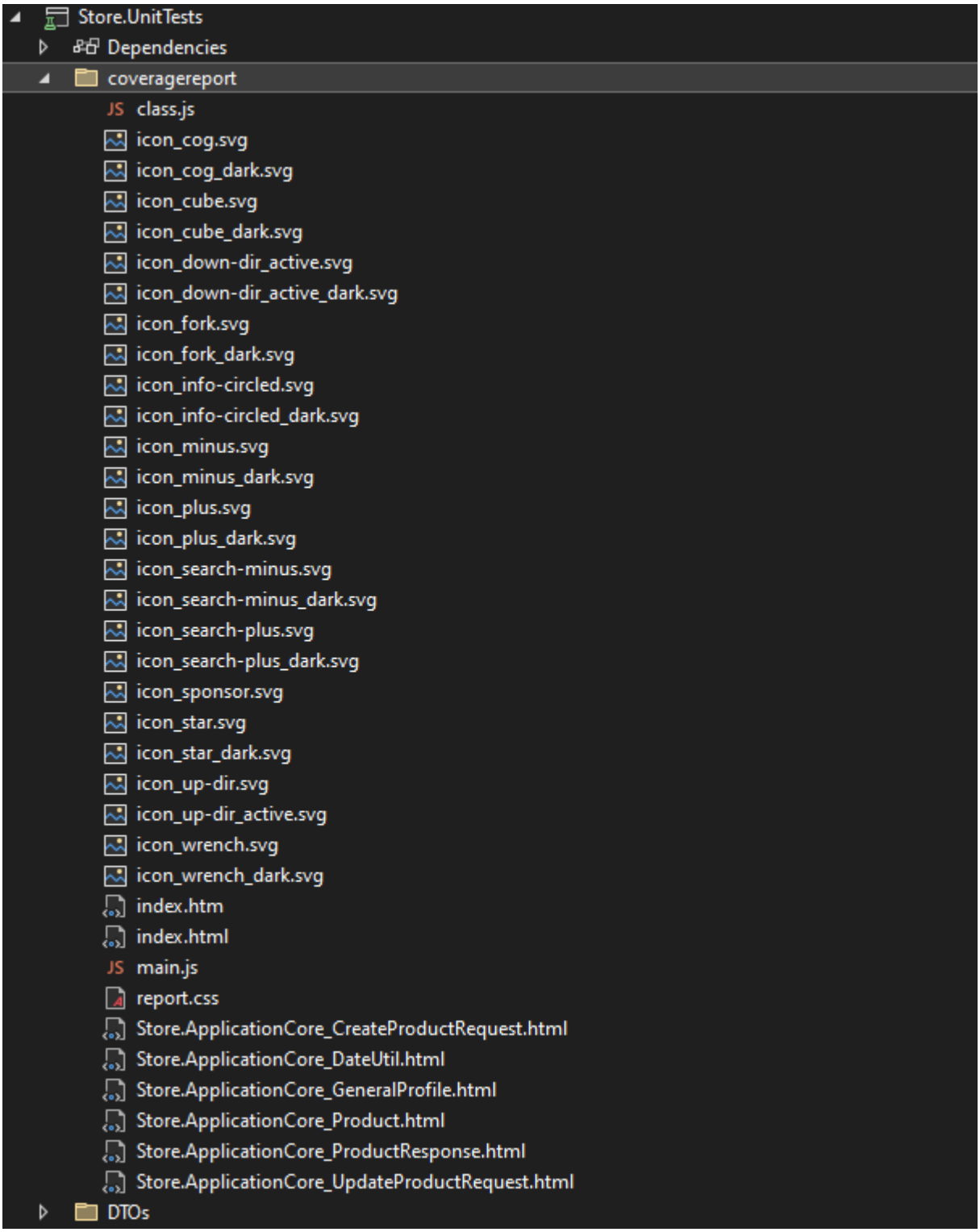
نکته: Guid پوشه ی ساخته شده را باید در فرمان جایگذاری نماییم.

```
reportgenerator -reports:"TestResults\53345c3d-b41c-45eb-957c-41a9ebfc6a37\coverage.cobertura.xml" -targetdir:"coveragereport" -reporttypes:Html
```

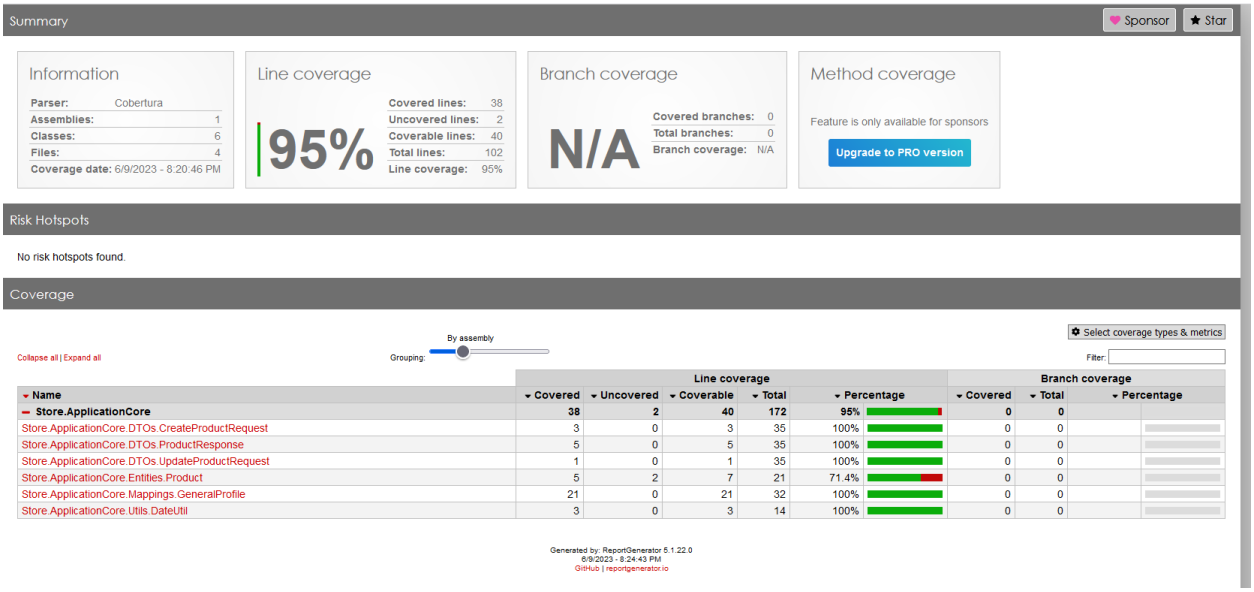
```
reportgenerator -reports:"TestResults\<GUID>\coverage.cobertura.xml" -
targetdir:"coveragereport" -reporttypes:Html
```



بعد از اجرا یک پوشه ایجاد خواهد شد که حاوی گزارش Test Coverage ما است:



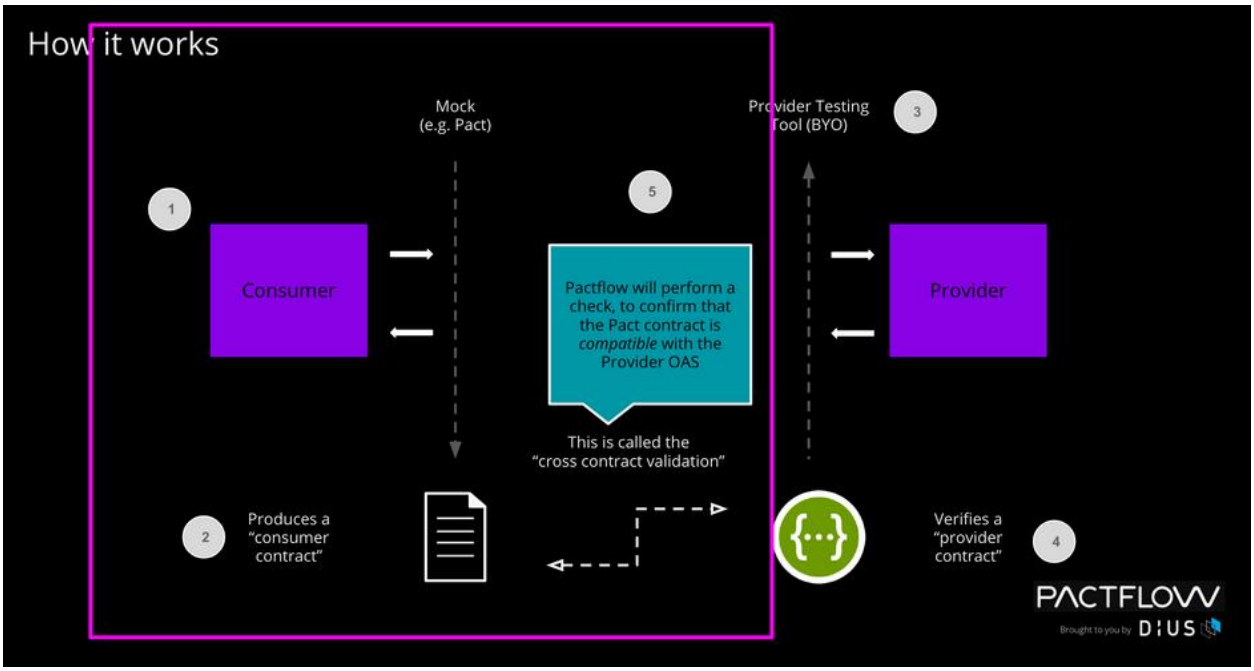
اگر فایل Index.html را اجرا کنیم نتیجه گزارش نمایش داده خواهد شد:



End-To-End Test .۴

https://blog.devgenius.io/asp-net-core-end-to-end-testing-۵۲۲۳۸۷e۲۸e۵

Contract Test .۵



۶. Non-Fuctional Test

- Performance Testing
- Load Testing
- Failover Testing
- Compatibility Testing
- Usability Testing
- Stress Testing
- Maintainability Testing
- Scalability Testing
- Volume Testing
- Security Testing
- Disaster Recovery Testing
- Compliance Testing
- Portability Testing
- Efficiency Testing
- Reliability Testing
- Baseline Testing
- Endurance Testing
- Documentation Testing
- Recovery Testing
- Internationalization Testing
- Localization Testing

https://www.guru۹۹.com/non-functional-testing.html

Unit Test vs Integration Test vs Functional Test

https://www.softwaretestinghelp.com/the-difference-between-unit-integration-and-functional-testing/

UI Test:

Selenium نصب

Install-Package Selenium.WebDriver

Install-Package Selenium.WebDriver.ChromeDriver

https://code-maze.com/selenium-aspnet-core-ui-tests/

Api Test:

تست Api در دو سمت قابل انجام است:

۱- کلاینت (Api Consumer)

۲- سرور (Api Producer)

Contract Testing:

https://tomdriven.dev/.net/core/contract-testing/pact/test/testing-with-pact-in-net-core.html/۱۳/۰۳/۲۰۱۸

https://github.com/pact-foundation/pact-workshop-dotnet-core-v۱

Consumer:

With an instance of our Mock HTTP Server in our test class, we can add the first test. All the Pact tests added during this workshop will follow the same three steps:

۱. Mock out an interaction with the Provider API.
۲. Interact with the mocked out interaction using our Consumer code.
۳. Assert the result is what we expected.

Provider:

The Pact tests for the Provider API will need to do two things:

۱. Manage the state of the Provider API as dictated by the Pact file.

٢. Communicate with the Provider API to verify that the real responses for HTTP requests defined in the Pact file match the mocked ones.