



* Changed the notations from test to fact
* Eliminates the dependencies between different test methods: **xUnit** creates a **new instance of the test class** for each test method. This means that **each test method runs in its own instance of the test class**, ensuring that changes made to class-level variables or objects in one test do not affect another test
* In **xUnit**, there is no need for [SetUp] and [TearDown] attributes like in NUnit or MSTest because **xUnit’s constructor** and **IDisposable interface** serve these purposes.
* While each test method has its own instance, xUnit also provides **class fixtures** to share objects between tests in a class, but only when explicitly needed. This gives you control over when to share resources while maintaining isolation by default i.e. test methods still run in separate class instance
* xUnit allows you to inject dependencies into test classes via constructors, making it easy to provide different instances of dependencies for each test. This ensures that tests remain isolated even when using external resources like databases or web services
* xUnit supports **parallel test execution**, meaning tests can run concurrently, but only if they are truly independent. This reinforces the importance of isolated tests since any tests that have side effects on shared resources will fail when run in parallel
* In scenarios where test data must be shared across tests, xUnit provides ways to isolate this through features like **collections** or **class fixtures**, but by default, each test runs in isolation unless explicitly shared, reducing the risk of unintentional data leakage

# Packages required:

1. .net test sdk
2. Xunit
3. Xunit runner visual studio
4. Selenium WebDriver
5. WebDriver Manager (Works combine with Firefox, chrome driver)
6. Xunit.SkippableFact
7. Dotnet Selenium Extra WebHelpers

# Resources:

* <https://xunit.net/#documentation>
* <https://github.com/xunit/samples.xunit>

# Xunit Attributes:

* [Fact]: to mark test cases
* [Fact (Skip=” reason to skip”)]: to skip a test case
* [SkippableFact(typeof(<Exception Class>))]: Skip the test if the specified exception is faced. It is Skippable Fact library.
* [Trait (“attribute”, “value”)]: you can any trait to filter you tests and use them when running from command line i.e. **dotnet test –filter “attribute=value**”
* [RetryFact(Maxretries = <int>)]: Re-run test, if failed. It can be used from samples.xunit repo as below:
  + Go to [GitHub repo](https://github.com/xunit/samples.xunit) from resources above
  + Go to retry folder
  + Copy all .cs file except for example and sample
  + Change the namespace
  + In “RetryFactAttribute.cs” update below attribute with namespace
    - [XunitTestDiscoverer(“<namespace>.RetryFactDiscoverer”, “<namespace>”)]
  + Note: It cannot be used in parallel with [Fact] attribute
* [UseCulture(“<culture>”)]: useful to handle dates as per the location. It binds your test to follow culture
  + Go to GitHub repo from resources
  + Go to culture folder
  + Copy .cs files to your project
  + Change the namespace and its ready to use
* [TestPriority(<int>)]: Order you tests based on the priority:
  + Copy the files from repo
  + Update the namespace where required

# To Run Test in parallel:

* Copy the customization in github from NamespaceParallelization to your project
  + Rename the namespaces where required
* Add xunit.runner.json file with below code:
  + "parallelizeAssembly": true,
  + "parallelizeTestCollections": true
  + Set the properties of file to “Copy always”
* Add below code to the namespace:
  + [assembly: CollectionBehavior(CollectionBehavior.CollectionPerClass, MaxParallelThreads = 2)]

# To Run Multiple Browsers:

* Pre-Req:
  + You have driver fixture with all of the below initialization already in it
* Implement Adapter design pattern
  + To use Webdirver interface as another interface
  + Allows interface of existing class as another interface
  + Create Adapter class
  + Implement Idisposible
* Define private compositions
  + Wait for element constant
  + Driver
  + Wait
* In adapter class, use factory method to initialize web driver
  + Create Start() method to initialize Driver
  + In same method, initialize the web driver wait as well
* Create an Enum of browser types to use in factory method
* Move the commonly used methods to newly created adapter class as public because driver fixture will be marked abstract after
* Mark the driver fixture as abstract
  + Define Adapter class as a property in it
  + Modify WAIT\_FOR\_ELEMENT property to abstract so it can be override. Also define some default time in case it is not override
* Implement template design patter in driver fixture class to initialize it
  + Create class for each browser, inherit from Driver fixture abstract class implement the abstract methods
  + Override properties if necessary
* Update references in your project