Section 15: xUnit [MVC and Web API]

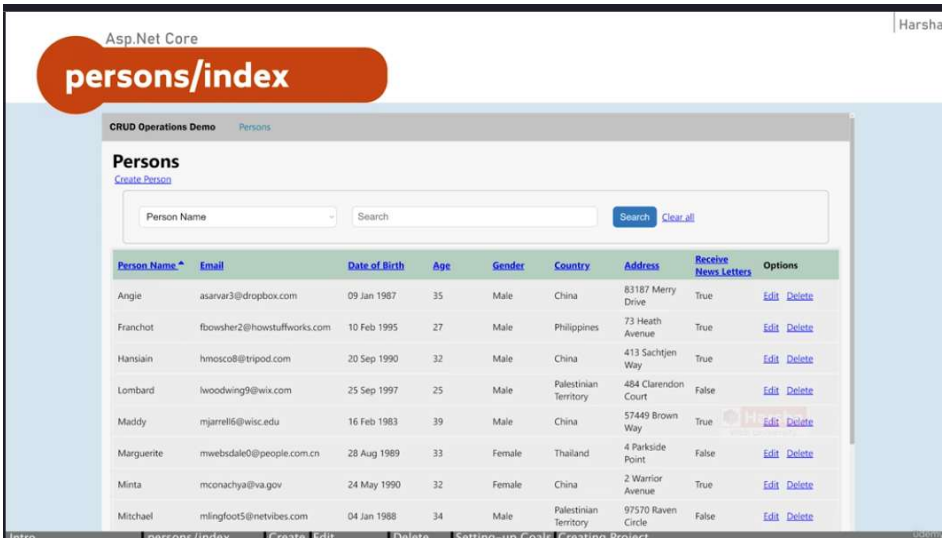
Thursday, January 23, 2025 7:51 PM

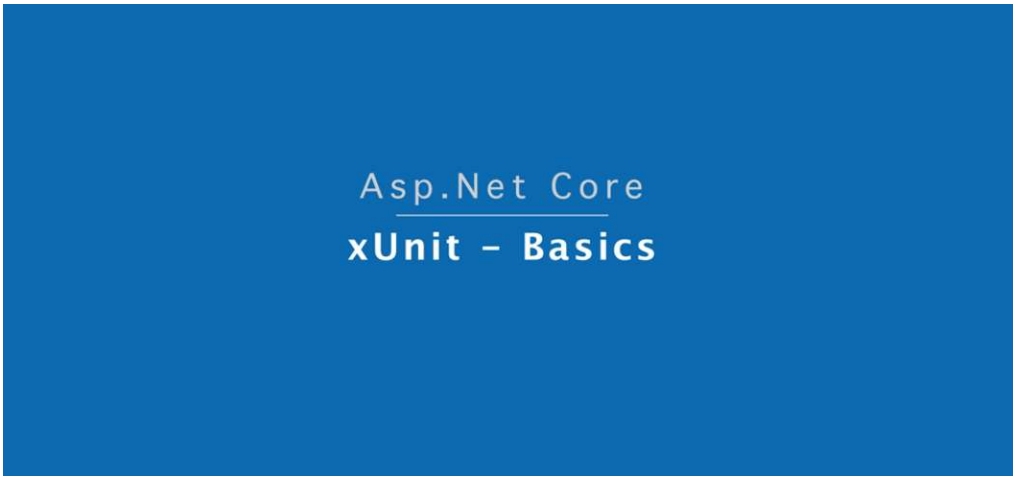
The next four sections will be highly hands-on. We will create a web application with CRUD operations, where you will practically implement Create, Retrieve, Update, and Delete functionalities.

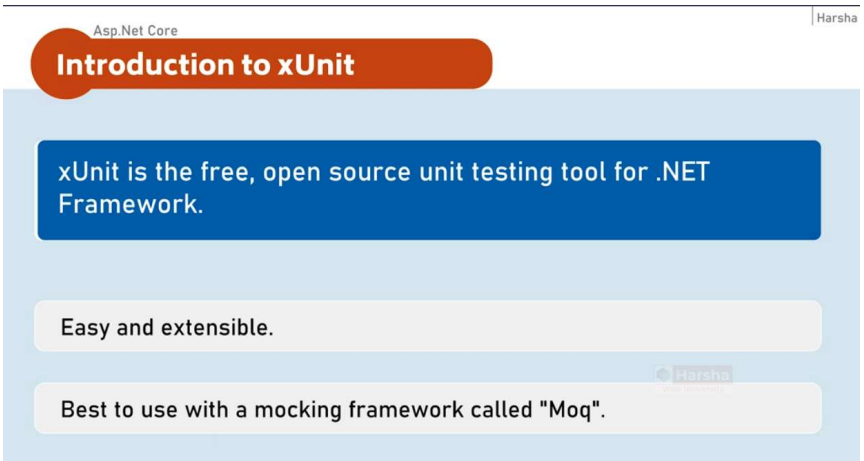
Initially, we will focus on creating unit tests, views, and controllers. Afterward, we will integrate the application with a database using Entity Framework.

This approach will give us an opportunity to test all the knowledge we've gained from the beginning of the course. We will utilize concepts such as:

- Razor views
- Layout views
- Dependency injection
- Partial views
- And other essential features

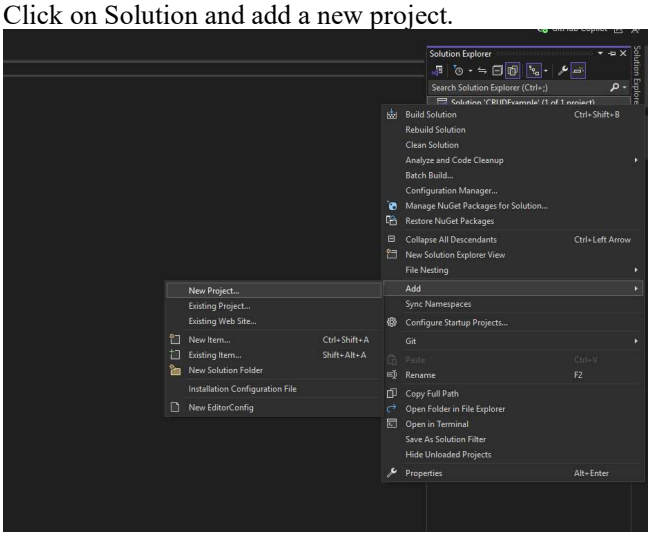


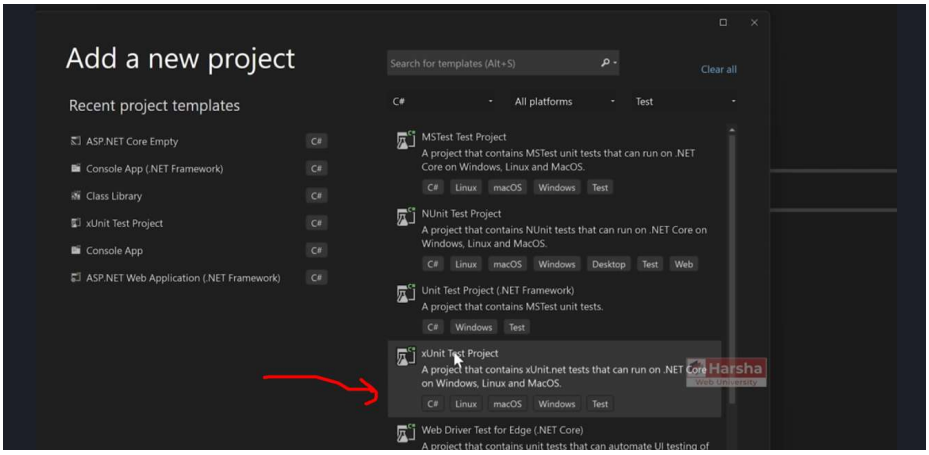




XUnit is a third-party package commonly used for unit testing controllers, services, and other classes in ASP.NET Core. It is one of the most popular and widely used unit testing frameworks in the .NET ecosystem.
While .NET provides an inbuilt testing framework called MSTest, XUnit offers greater extensibility, ease of use, and a shorter learning curve. It is also well-suited to work with the mocking framework Moq.
The primary reason for XUnit's popularity in .NET unit testing is its extensibility, allowing developers to adapt or extend it to meet complex requirements.
In this lecture, we'll get started with XUnit!

Click on Solution and add a new project.

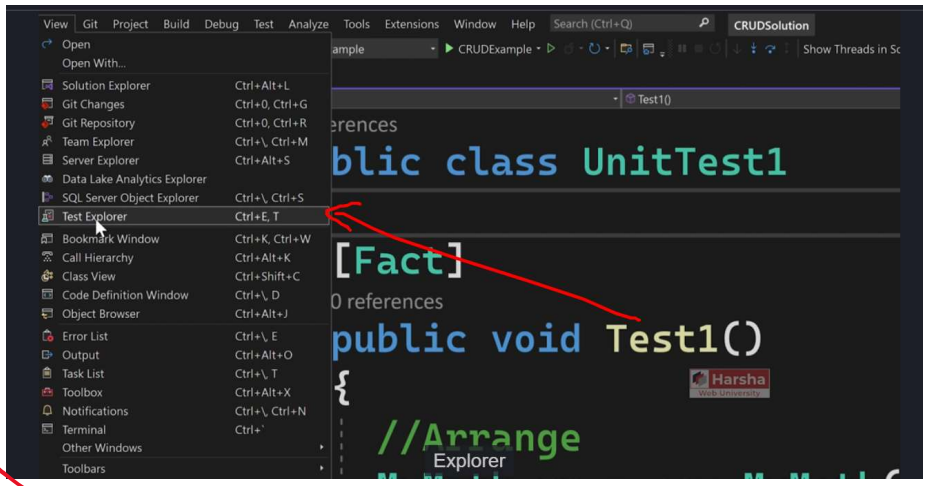


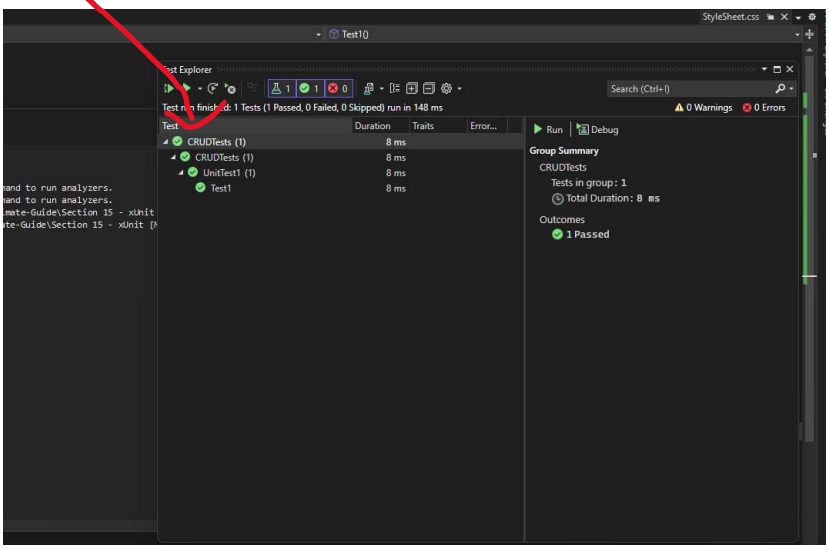


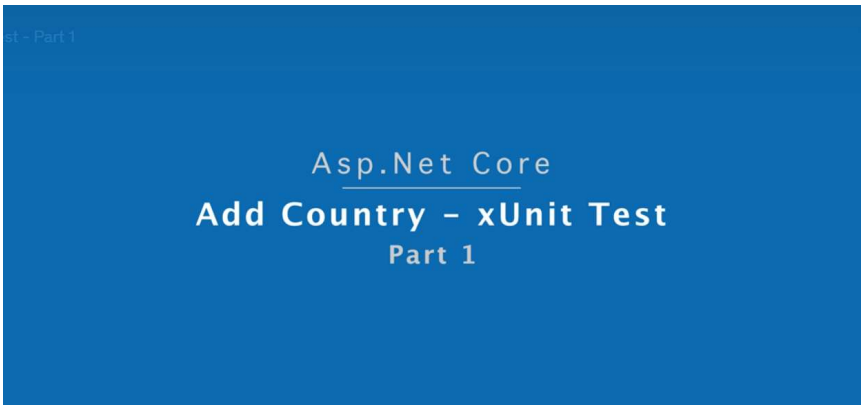

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespac
{
    //just for understanding purposes we have added this class here, otherwise it would be a service cla
    internal class MyMath
    {
        //we want to test this 'Add' method, whether it works correctly.
        public int Add(int a, int b)
        {
            return a + b;
        }
    }
}
```

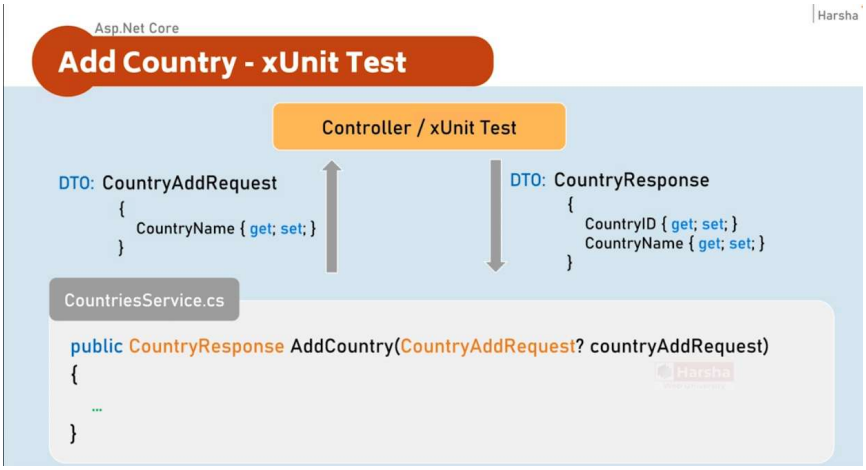
```
namespace CRUDTests
{
    public class UnitTest1
    {
        [Fact] // Fact means, you are goint to write one or two unit test in this method
        public void Test1()
        {
            //Arrange
            // means, the declaration of variables and collecting the inputs
            MyMath myMath = new MyMath();
            int input1 = 10, input2 = 5;
            int expected = 15;

            //Act
            //act means, calling the method, which method you would like to test
            int actual =  myMath.Add(input1,input2);

            //Assert
            //means comparing the expected value with the actual value.
            //if the expected and actual value are same then test case is pass otherwise it is fail
            Assert.Equal(expected,actual);
        }
    }
}
```

Now, in order to test the value, go to the view menu and click on 'Test Explorer'


View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
CRUDSolution
Open
Open With...
Solution Explorer Ctrl+Alt+L
Git Changes Ctrl+0, Ctrl+G
Git Repository Ctrl+0, Ctrl+R
Team Explorer Ctrl+\, Ctrl+M
Server Explorer Ctrl+Alt+S
Data Lake Analytics Explorer
SQL Server Object Explorer Ctrl+\, Ctrl+S
Test Explorer Ctrl+E, T
Bookmark Window Ctrl+K, Ctrl+W
Call Hierarchy Ctrl+Alt+K
Class View Ctrl+Shift+C
Code Definition Window Ctrl+\, D
Object Browser Ctrl+Alt+J
Error List Ctrl+\, E
Output Ctrl+Alt+O
Task List Ctrl+\, T
Toolbox Ctrl+Alt+X
Notifications Ctrl+\, Ctrl+N
Terminal Ctrl+`
Other Windows
Toolbars
CRUDExample
Show Threads in Sc
Test1()
blic class UnitTest1
[Fact]
0 references
public void Test1()
{
//Arrange
Harsha
Web University
Explorer


StyleSheet.css
Test1()
Test Explorer
Search (Ctrl+I)
Test run finished: 1 Tests (1 Passed, 0 Failed, 0 Skipped) run in 148 ms
0 Warnings
0 Errors
Test
Duration
Traits
Error...
CRUDTests (1) 8 ms
CRUDTests (1) 8 ms
UnitTest1 (1) 8 ms
Test1 8 ms
Run
Debug
Group Summary
CRUDTests
Tests in group: 1
Total Duration: 8 ms
Outcomes
1 Passed


Asp.Net Core
Add Country – xUnit Test
Part 1



Overview of the AddCountry Method and TDD Approach

The AddCountry method in the **Country Service** is responsible for receiving a Country object and adding it to the list of countries, which acts as a data store (e.g., a collection or database table).
Whenever a client (e.g., a controller or another class) calls this method and provides a CountryAddRequest object, the method will:

1. Validate the request.
2. Check for duplicate entries or any other business validations.
3. Generate a unique CountryId internally (it will not be supplied in the request).
4. Add the valid country to the list of countries.
5. Return a CountryResponse object containing the newly generated CountryId and the CountryName.

Following Test-Driven Development (TDD)

In this lecture, we will implement **unit tests** for the AddCountry method first, before writing its actual implementation. TDD ensures the developer accurately implements functionality that adheres to the expected behavior without missing important edge cases.

Data Transfer Objects (DTOs)

The method will use **DTOs (Data Transfer Objects)** for exchanging data:

1. **CountryAddRequest**:
   - Used for sending the country name as input.
   - Does not include the CountryId (as it is generated internally).
2. **CountryResponse**:
   - Used for returning the CountryId and CountryName back to the client.

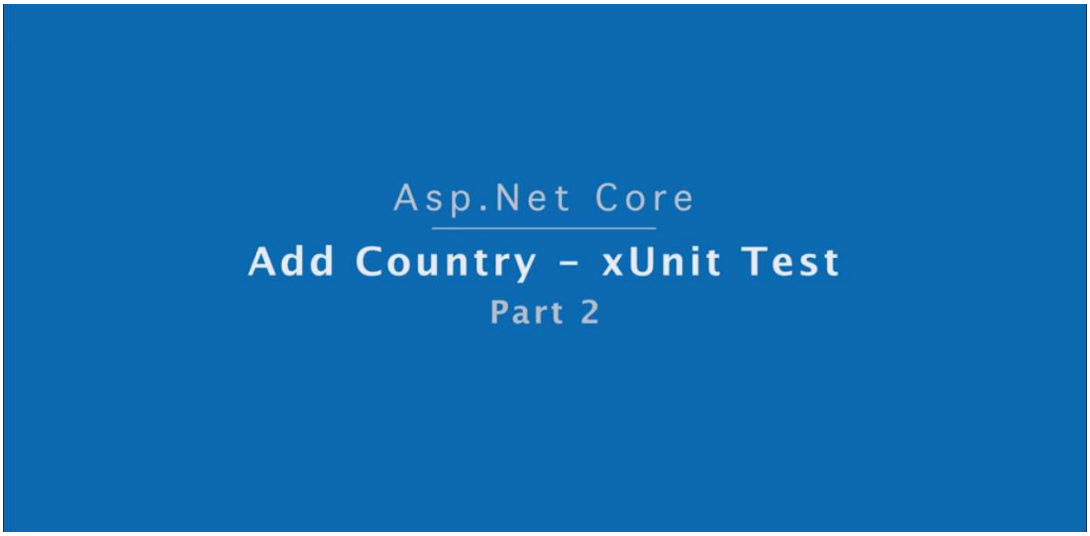
Key Points about DTOs:

- They facilitate communication between the **Controller** and **Service** layers.
- They help encapsulate request and response data for operations.
- In this context, the CountryAddRequest is the argument for the AddCountry method, and the CountryResponse is its return type.

Implementation Plan

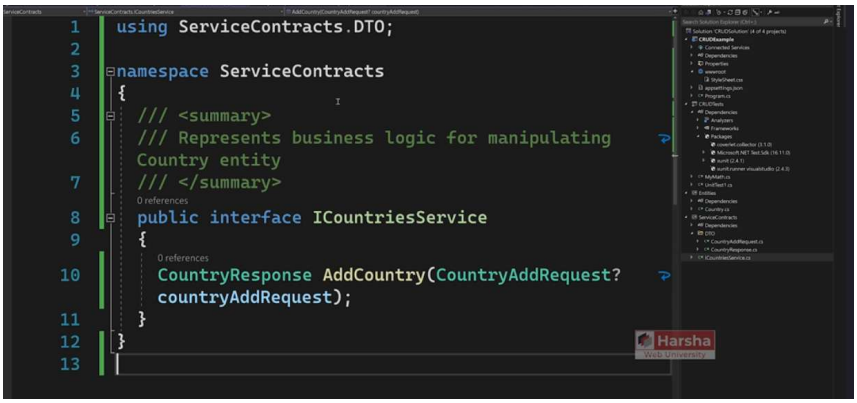
1. Write unit tests for the AddCountry method in the Country Service.
2. Simulate supplying a CountryAddRequest object and validate:
   - Proper handling of input.
   - Prevention of duplicate country entries.
   - Accurate response object structure (CountryResponse).
3. Implement the method in the next lecture, ensuring it passes all written unit tests.

Let’s move forward and write the unit tests for the AddCountry method!

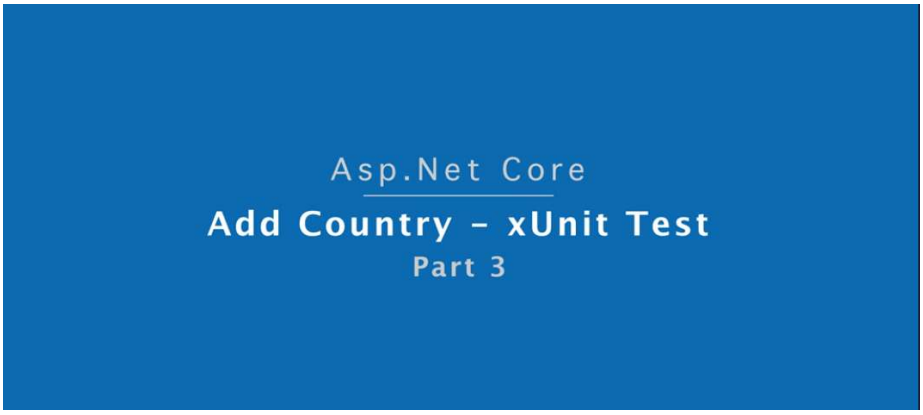


We have declared the interface but we will not implement first. As per the TDD, we will write test cases first then we will implement the interface. Let the unit test fail first. That means, the implementation should follow the unit test.

The unit test should not follow the implementation. (Unit test first then implementation later.)



```
using ServiceContracts.DTO;

namespace ServiceContracts
{
    /// <summary>
    /// Represents business logic for manipulating
    Country entity
    /// </summary>
    0 references
    public interface ICountriesService
    {
        0 references
        CountryResponse AddCountry(CountryAddRequest?
        countryAddRequest);
    }
}
```



Asp.Net Core

Add Country – Implementation

Asp.Net Core

Harsha

Add Country - xUnit Test

Controller / xUnit Test

DTO: CountryAddRequest
{
CountryName { get; set; }
}

DTO: CountryResponse
{
CountryID { get; set; }
CountryName { get; set; }
}

CountriesService.cs

```
public CountryResponse AddCountry(CountryAddRequest? countryAddRequest)
{
    ...
}
```

In the last lecture, we have written unit test cases for 'AddCountry' method. Now let's implement our 'AddCountry()' method to pass our unit test cases.

Asp.Net Core

Harsha

Add Country - xUnit Test

```
public CountryResponse AddCountry(CountryAddRequest? countryAddRequest)
{
    //Check if "countryAddRequest" is not null.
    //Validate all properties of "countryAddRequest"
    //Convert "countryAddRequest" from "CountryAddRequest" type to "Country".
    //Generate a new CountryID
    //Then add it into List<Country>
    //Return CountryResponse object with generated CountryID
}
```

Asp.Net Core

Get All Countries – xUnit Test

Asp.Net Core

Get All Countries - xUnit Test

Controller / xUnit Test

DTO: CountryResponse

```
{
    CountryID { get; set; }
    CountryName { get; set; }
}
```

CountriesService.cs

```
public List<CountryResponse> GetAllCountries()
{
    ...
}
```

Asp.Net Core

Get All Countries – Implementation

Asp.Net Core

Get All Countries - xUnit Test

Controller / xUnit Test

DTO: CountryResponse

```
{
    CountryID { get; set; }
    CountryName { get; set; }
}
```

CountriesService.cs

```
public List<CountryResponse> GetAllCountries()
{
    ...
}
```

Asp.Net Core

Harsha

Get All Countries - xUnit Test

```
public List<CountryResponse> GetAllCountries()
{
    //Convert all countries from "Country" type to "CountryResponse" type.
    //Return all CountryResponse objects
}
```

Asp.Net Core

Get Country by CountryID – xUnit Test

Asp.Net Core

Harsha

Get Country by Country ID - xUnit Test

Controller / xUnit Test

countryID

DTO: CountryResponse

```
{
    CountryID { get; set; }
    CountryName { get; set; }
}
```

CountriesService.cs

```
public CountryResponse? GetCountryByCountryID(Guid? countryID)
{
    ...
}
```

Asp.Net Core

Get Country by CountryID – Implementation

Asp.Net Core

Harsha

Get Country by Country ID - xUnit Test

Controller / xUnit Test

countryID

DTO: CountryResponse

```
{
   CountryID { get; set; }
   CountryName { get; set; }
}
```

CountriesService.cs

```
public CountryResponse? GetCountryByCountryID(Guid? countryID)
{
   ...
}
```

Asp.Net Core

Harsha

Get Country by Country ID - xUnit Test

```
public CountryResponse? GetCountryByCountryID(Guid? countryID)
{
   //Check if "countryID" is not null.
   //Get matching country from List<Country> based countryID.
   //Convert matching country object from "Country" to "CountryResponse" type.
   //Return CountryResponse object
}
```

Asp.Net Core

Add Person – Creating Models

Part 1

Asp.Net Core

Harsha

Add Person - xUnit Test

Controller / xUnit Test

DTO: PersonAddRequest

```
{
   PersonName { get; set; }
   Email { get; set; }
   ...
}
```

DTO: PersonResponse

```
{
   PersonID { get; set; }
   PersonName { get; set; }
   ...
}
```

PersonsService.cs

```
public PersonResponse AddPerson(PersonAddRequest? personAddRequest)
{
   ...
}
```

Asp.Net Core

Add Person – Creating Models

Part 2

Asp.Net Core

Add Person – xUnit Test

Harsha

Asp.Net Core

Add Person - xUnit Test

Controller / xUnit Test

DTO: PersonAddRequest

```
{
    PersonName { get; set; }
    Email { get; set; }
    ...
}
```

DTO: PersonResponse

```
{
    PersonID { get; set; }
    PersonName { get; set; }
    ...
}
```

PersonsService.cs

```
public PersonResponse AddPerson(PersonAddRequest? personAddRequest)
{
    ...
}
```

Intro | AddPerson() method in interface | Test Case 1 | Test Case 2 | Test Case 3

Asp.Net Core

Add Person – Implementation

Asp.Net Core

Add Person - xUnit Test

Controller / xUnit Test

DTO: PersonAddRequest

```
{
    PersonName { get; set; }
    Email { get; set; }
    ...
}
```

DTO: PersonResponse

```
{
    PersonID { get; set; }
    PersonName { get; set; }
    ...
}
```

PersonsService.cs

```
public PersonResponse AddPerson(PersonAddRequest?  personAddRequest)
{
    ...
}
```

Intro | AddPerson() - Implementation | Summary

Asp.Net Core

Add Person - xUnit Test

```
public PersonResponse AddPerson(PersonAddRequest? personAddRequest)
{
    //Check if "personAddRequest" is not null.
    //Validate all properties of "personAddRequest".
    //Convert "personAddRequest" from "PersonAddRequest" type to "Person".
    //Generate a new PersonID.
    //Then add it into List<Person>.
    //Return PersonResponse object with generated PersonID.
}
```

Asp.Net Core

Add Person – Validation

Asp.Net Core

Get Person by Person ID – xUnit Test

Harsha

Asp.Net Core

Get Person by Person ID - xUnit Test

Controller / xUnit Test

personID

DTO: PersonResponse

```
{
    PersonID { get; set; }
    PersonName { get; set; }
}
```

PersonsService.cs

```
public PersonResponse GetPersonByPersonID(Guid? personID)
{
    ...
}
```

Asp.Net Core

Get Person by Person ID – Implementation

Asp.Net Core

Harsha

Get Person by Person ID - xUnit Test

Controller / xUnit Test

personID

DTO: PersonResponse

```
{
   PersonID { get; set; }
   PersonName { get; set; }
}
```

PersonsService.cs

```
public PersonResponse GetPersonByPersonID(Guid? personID)
{
   ...
}
```

Asp.Net Core

Harsha

Get Person by Person ID - xUnit Test

```
public PersonResponse GetPersonByPersonID(Guid? personID)
{
   //Check if "personID" is not null.
   //Get matching person from List<Person> based personID.
   //Convert matching person object from "Person" to "PersonResponse" type.
   //Return PersonResponse object
}
```

Asp.Net Core

Get All Persons – xUnit Test

Asp.Net Core

Harsha

Get All Persons - xUnit Test

Controller / xUnit Test

DTO: PersonResponse

```
{
   PersonID { get; set; }
   PersonName { get; set; }
}
```

PersonsService.cs

```
public List<PersonResponse> GetAllPersons()
{
   ...
}
```

Asp.Net Core

Get All Persons – Implementation

Asp.Net Core

Harsha

Get All Persons - xUnit Test

Controller / xUnit Test

DTO: PersonResponse

```
{
    PersonID { get; set; }
    PersonName { get; set; }
}
```

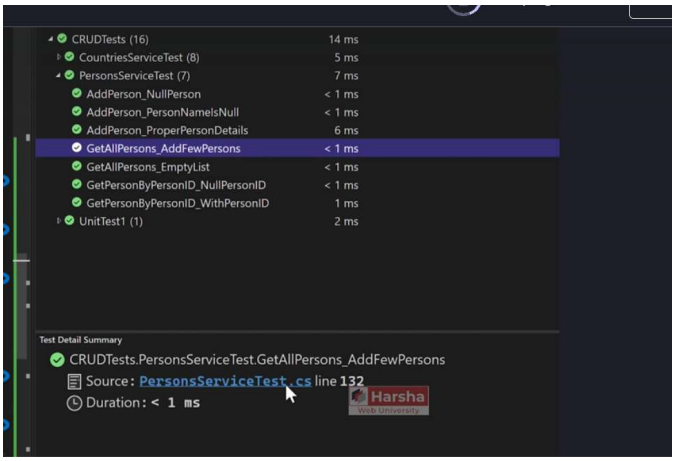
PersonsService.cs

```
public List<PersonResponse> GetAllPersons()
{
    ...
}
```

Asp.Net Core

TestOutputHelper

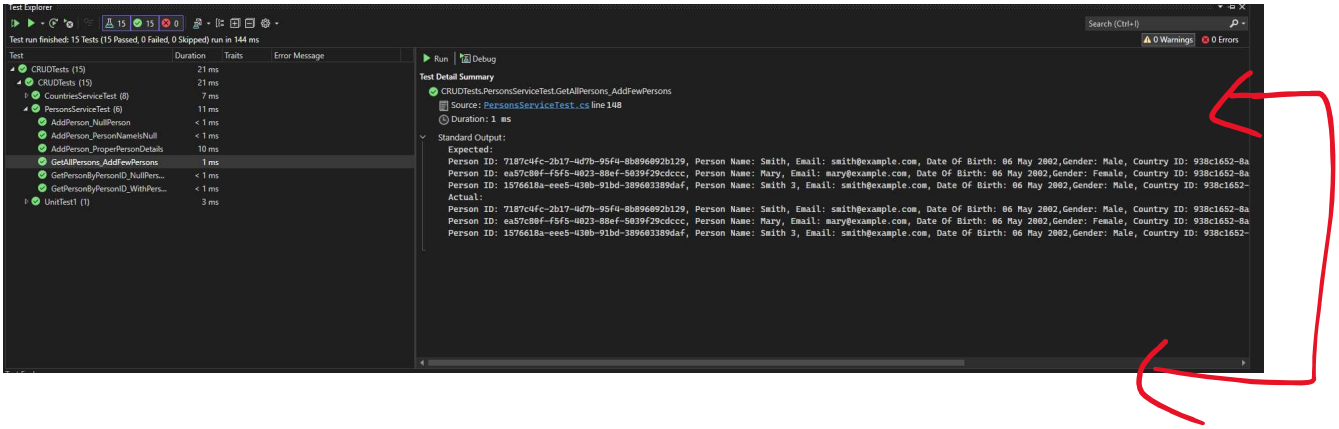
Whenever the test cases are passed, it will not show the actual value or expected value in the details.



Buy if you want to see the 'expected' or 'actual' value, when the test cases are passed or failed, then you can use the pre-defined service called 'ITestOutputHelper'

```
namespace CRUDTests
{
    public class PersonsServiceTest
    {
        // private fields
        private readonly IPersonsService _personsService;
        private readonly ICountriesService _countriesService;
        private readonly ITestOutputHelper _testOutputHelper;

        //constructor
        public PersonsServiceTest(ITestOutputHelper testOutputHelper)
        {
            _personsService = new PersonsService();
            _countriesService = new CountriesService();
            _testOutputHelper = testOutputHelper;
        }

        AddPerson
```

```
        //print person_response_list_from_add
        _testOutputHelper.WriteLine("Expected:");

        foreach (PersonResponse person_response_from_add in person_response_list_from_add)
        {
            _testOutputHelper.WriteLine(person_response_from_add.ToString());
        }

        //Act
        List<PersonResponse> person_list_from_get =  _personsService.GetAllPersons();

        //print person_list_from_get
        _testOutputHelper.WriteLine("Actual:");

        foreach (PersonResponse person_from_get in person_list_from_get)
        {
            _testOutputHelper.WriteLine(person_from_get.ToString());
        }
```



Asp.Net Core

Get Filtered Persons – xUnit Test

Asp.Net Core

Harsha

Get Filtered Persons - xUnit Test

Controller / xUnit Test

```
string searchBy
string searchString
```

```
DTO: PersonResponse
{
    PersonID { get; set; }
    PersonName { get; set; }
}
```

PersonsService.cs

```
public List<PersonResponse> GetFilteredPersons(string searchBy, string? searchString)
{
    ...
}
```

Asp.Net Core

Get Filtered Persons – Implementation

Asp.Net Core

Harsha

Get Filtered Persons - xUnit Test

Controller / xUnit Test

string searchBy
string searchString

DTO: PersonResponse
{
PersonID { get; set; }
PersonName { get; set; }
}

PersonsService.cs

```
public List<PersonResponse> GetFilteredPersons(string searchBy, string? searchString)
{
    ...
}
```

Asp.Net Core

Harsha

Get Filtered Persons - xUnit Test

```
public List<PersonResponse> GetFilteredPersons(string searchBy, string? searchString)
{
  //Check if "searchBy" is not null.
  //Get matching persons from List<Person> based on given searchBy and searchString.
  //Convert the matching persons from "Person" type to "PersonResponse" type.
  //Return all matching PersonResponse objects
}
```

Asp.Net Core

Get Sorted Persons – xUnit Test

Asp.Net Core

Harsha

Get Sorted Persons - xUnit Test

Controller / xUnit Test

```
List<PersonResponse> allPersons
string sortBy
SortOrderEnum sortOrder
```

DTO: PersonResponse

```
{
    PersonID { get; set; }
    PersonName { get; set; }
}
```

PersonsService.cs

```
public List<PersonResponse> GetSortedPersons(List<PersonResponse> allPersons,
                                             string sortBy, SortOrderEnum sortOrder)
{
    ...
}
```

Asp.Net Core

Get Sorted Persons – Implementation

Asp.Net Core

Harsha

Get Sorted Persons - xUnit Test

Controller / xUnit Test

```
List<PersonResponse> allPersons
string sortBy
SortOrderEnum sortOrder
```

DTO: PersonResponse

```
{
    PersonID { get; set; }
    PersonName { get; set; }
}
```

PersonsService.cs

```
public List<PersonResponse> GetSortedPersons(List<PersonResponse> allPersons,
                                             string sortBy, SortOrderEnum sortOrder)
{
    ...
}
```

Asp.Net Core

Update Person – Creating DTO

Asp.Net Core

Harsha

Update Person - xUnit Test

Controller / xUnit Test

DTO: PersonUpdateRequest

```
{
    PersonID { get; set; }
    PersonName { get; set; }
    ...
}
```

DTO: PersonResponse

```
{
    PersonID { get; set; }
    PersonName { get; set; }
    ...
}
```

PersonsService.cs

```
public PersonResponse UpdatePerson(PersonUpdateRequest? personUpdateRequest)
{
    ...
}
```

Asp.Net Core

Update Person – xUnit Test

Asp.Net Core

Harsha

Update Person - xUnit Test

Controller / xUnit Test

DTO: PersonUpdateRequest

```
{
    PersonID { get; set; }
    PersonName { get; set; }
    ...
}
```

DTO: PersonResponse

```
{
    PersonID { get; set; }
    PersonName { get; set; }
    ...
}
```

PersonsService.cs

```
public PersonResponse UpdatePerson(PersonUpdateRequest? personUpdateRequest)
{
    ...
}
```

Asp.Net Core

Update Person – Implementation

Asp.Net Core

Update Person - xUnit Test

Controller / xUnit Test

DTO: PersonUpdateRequest

```
{
    PersonID { get; set; }
    PersonName { get; set; }
    ...
}
```

DTO: PersonResponse

```
{
    PersonID { get; set; }
    PersonName { get; set; }
    ...
}
```

PersonsService.cs

```
public PersonResponse UpdatePerson(PersonUpdateRequest?  personUpdateRequest)
{
    ...
}
```

Asp.Net Core

Update Person - xUnit Test

```
public PersonResponse UpdatePerson(PersonUpdateRequest? personUpdateRequest)
{
    //Check if "personUpdateRequest" is not null.
    //Validate all properties of "personUpdateRequest"
    //Get the matching "Person" object from List<Person> based on PersonID.
    //Check if matching "Person" object is not null
    //Update all details from "PersonUpdateRequest" object to "Person" object
    //Convert the person object from "Person" to "PersonResponse" type
    //Return PersonResponse object with updated details
}
```

Asp.Net Core

Delete Person - xUnit Test

Controller / xUnit Test

PersonID

true / false

PersonsService.cs

```
public bool DeletePerson(Guid?  personID)
{
    ...
}
```

Asp.Net Core

Delete Person – Implementation

Asp.Net Core

Delete Person - xUnit Test

```
public bool DeletePerson(Guid? personID)
{
   //Check if "personID" is not null.
  //Get the matching "Person" object from List<Person> based on PersonID.
  //Check if matching "Person" object is not null
  //Delete the matching "Person" object from List<Person>
  //Return Boolean value indicating whether person object was deleted or not
}
```