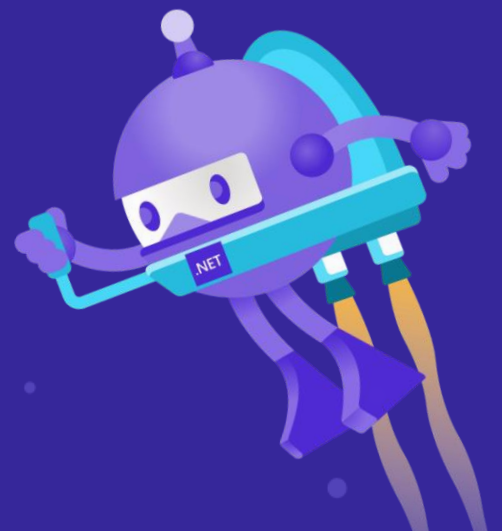
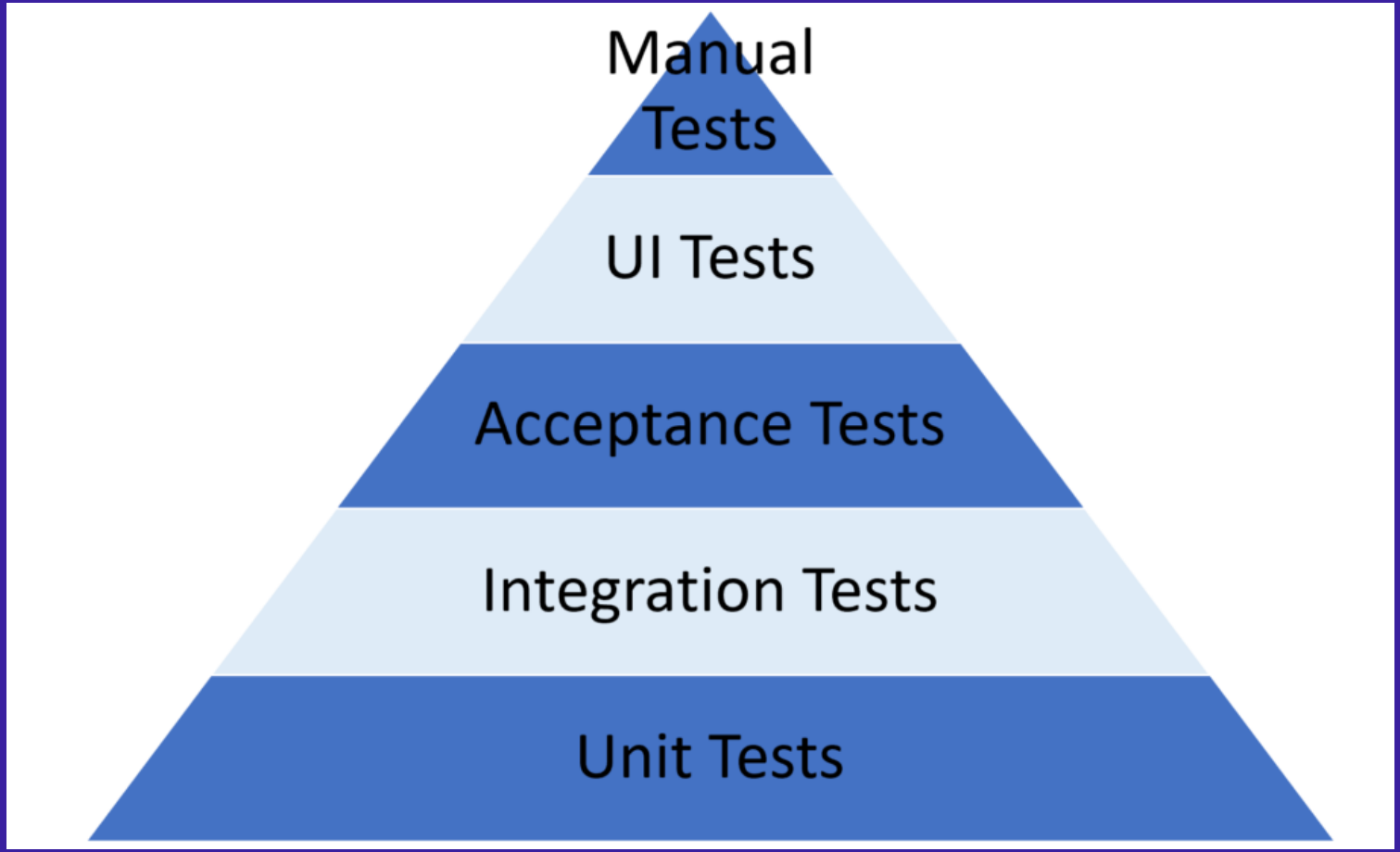
Day 16: Testing in ASP.NET Core



Testing

- Is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not

Testing Pyramid



Unit Tests

- Is a testing technique using which individual modules are tested for functional correctness
- Main aim is to isolate each unit of the system to identify, analyse and fix the defects
- Are smallest sort of tests
- Unit testing tools are divided into
  - Test Frameworks
  - Test Runners
  - Assertion Libraries

Integration Tests

- Are larger than Unit tests
- Typically, cross over boundaries between the modules of your application
- More time is needed for running the tests
- Are normally written from the perspective of a developer

Acceptance Tests

- Is also known as Functional tests
- Crosses module boundaries just like integration tests
- These tests are written from a user’s point of view
- Tests describes behavior of the system rather than the functioning

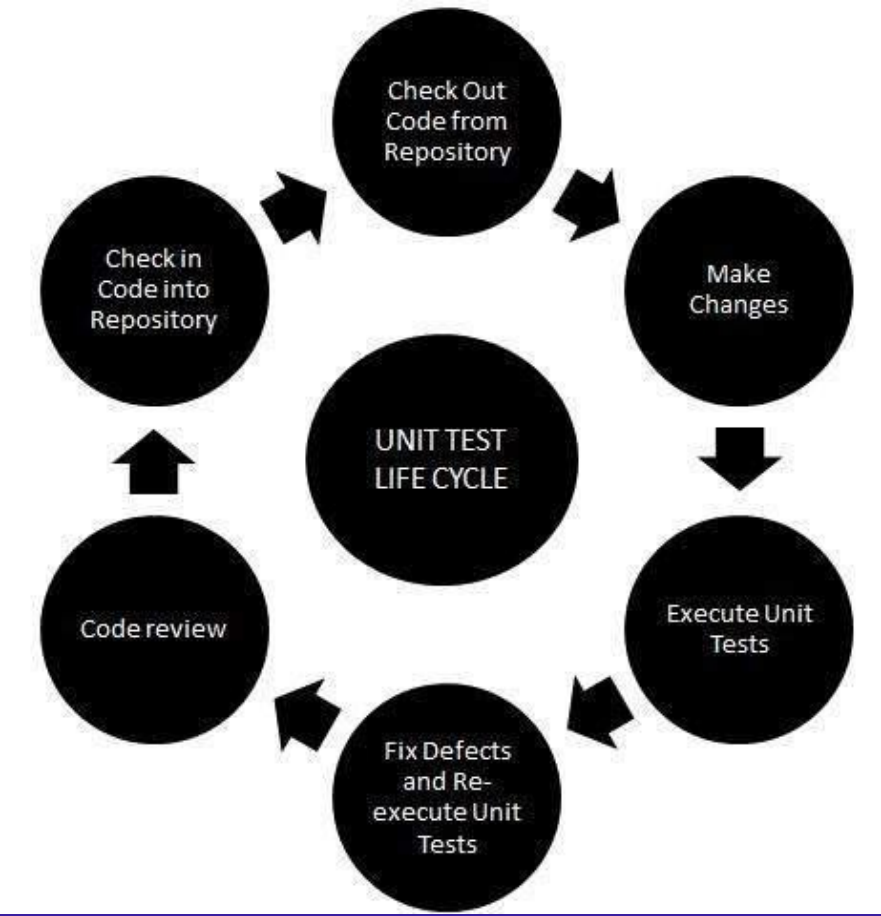
UI Tests

- Highest level of automated tests
- Tests drives the browser in an automated manner performing clicks, typing texts and navigation
- UI Testing tools are portable and can be used with any web application

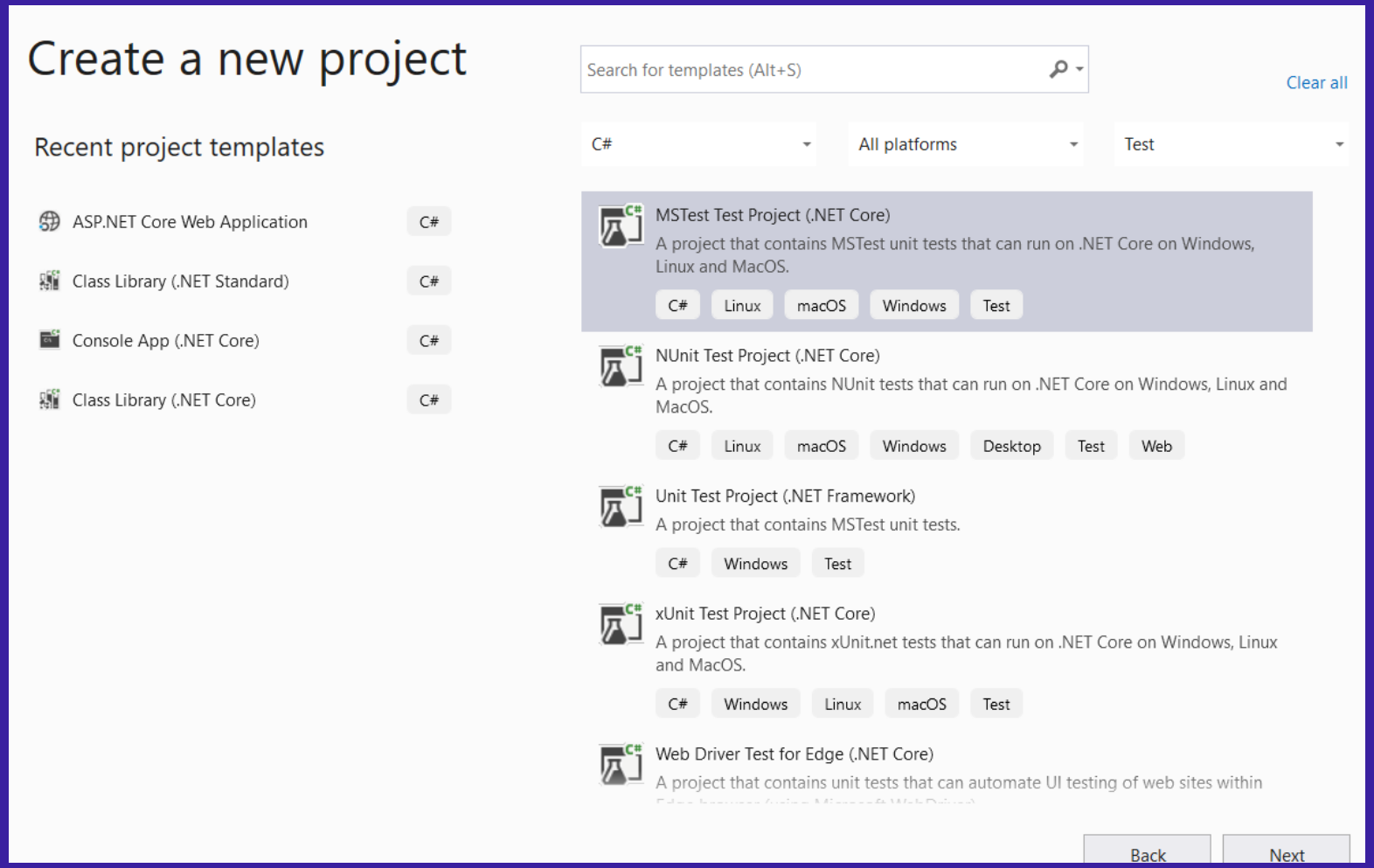
Manual Tests

- Highest level of automated tests
- Tests drives the browser in an automated manner performing clicks, typing texts and navigation
- UI Testing tools are portable and can be used with any web application

Unit Testing – Life Cycle



Unit Testing ASP.NET Core Apps



Supported Frameworks

- xUnit
  - a free, open source, community-focused unit testing tool for .NET
  - Written by the original inventor of NUnit v2
- Nunit
  - a unit-testing framework for all .NET languages.
  - Initially ported from JUnit, the current production release has been rewritten with many new features and support for a wide range of .NET platforms
- MSTest
  - Is the Microsoft test framework for all .NET languages.
  - It's extensible and works with both .NET CLI and Visual Studio

AAA Principle

- For unit testing, the practice is to follow AAA Principle
  - Arrange
    - where you would typically prepare everything for the test
  - Act
    - where the method we are testing is executed
  - Assert
    - where we compare what we expect to happen with the actual result of the test method execution

Executing Tests

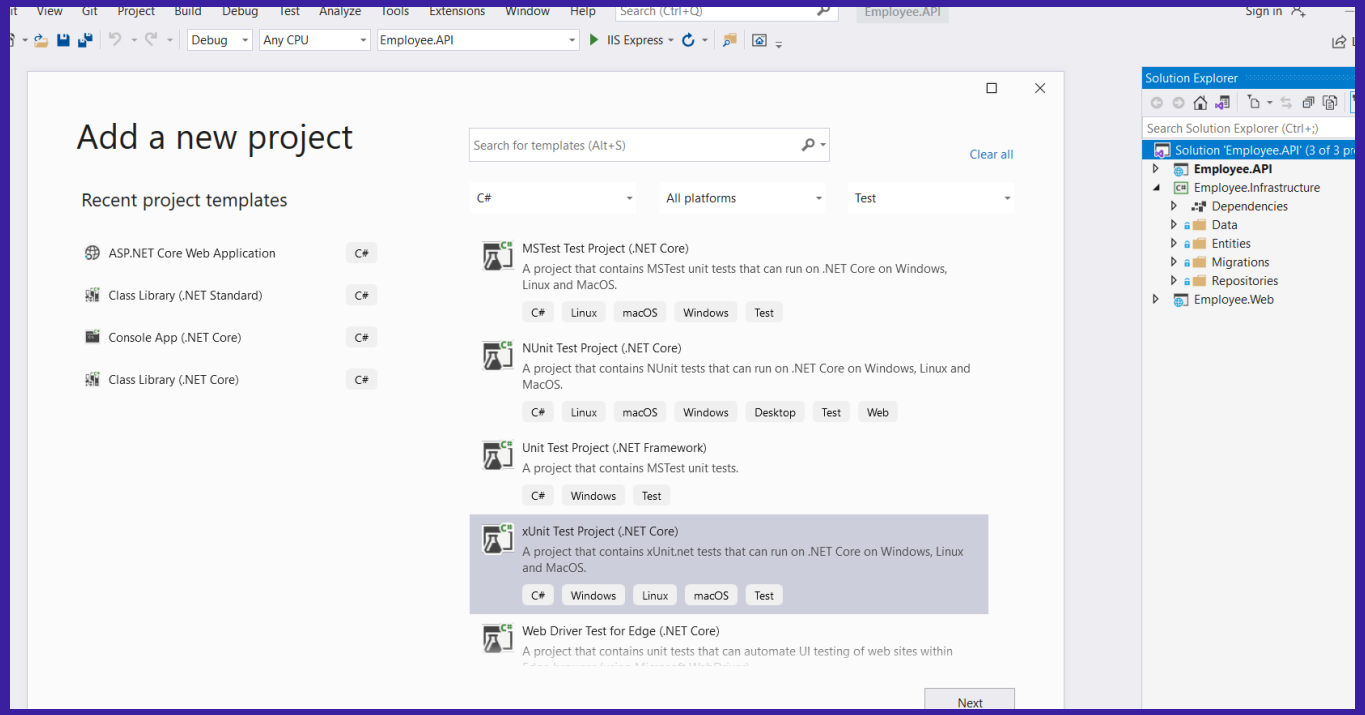
- .NET CLI
  - Uses *dotnet test* command to run unit tests in a solution
  - Available to use as part of CI/CD pipelines

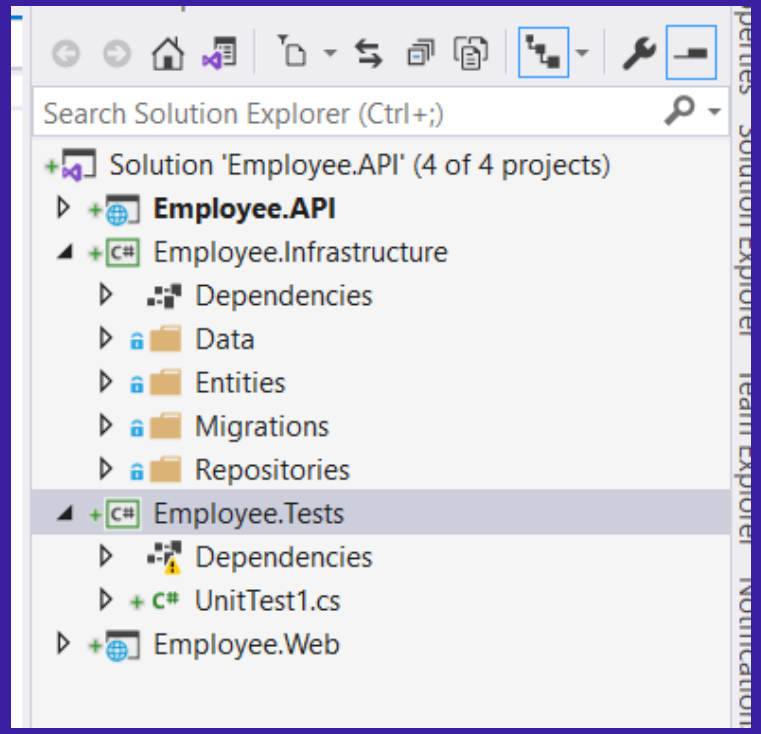
- IDE
  - Available as a graphical user interface in Visual Studio, Visual Studio for Mac and Visual Studio Code
  - More features are available in IDE when compared to CLI. Eg: Live Unit Testing

xUnit - Overview

- Is essentially a testing framework which provides a set of attributes and methods we can use to write the test code for our applications.
- Important Attributes
  - [Fact] – attribute states that the method should be executed by the test runner
  - [Theory] – attribute implies that we are going to send some parameters to our testing code.
  - [InlineData] – attribute provides those parameters we are sending to the test method.

xUnit – Creating Test Project





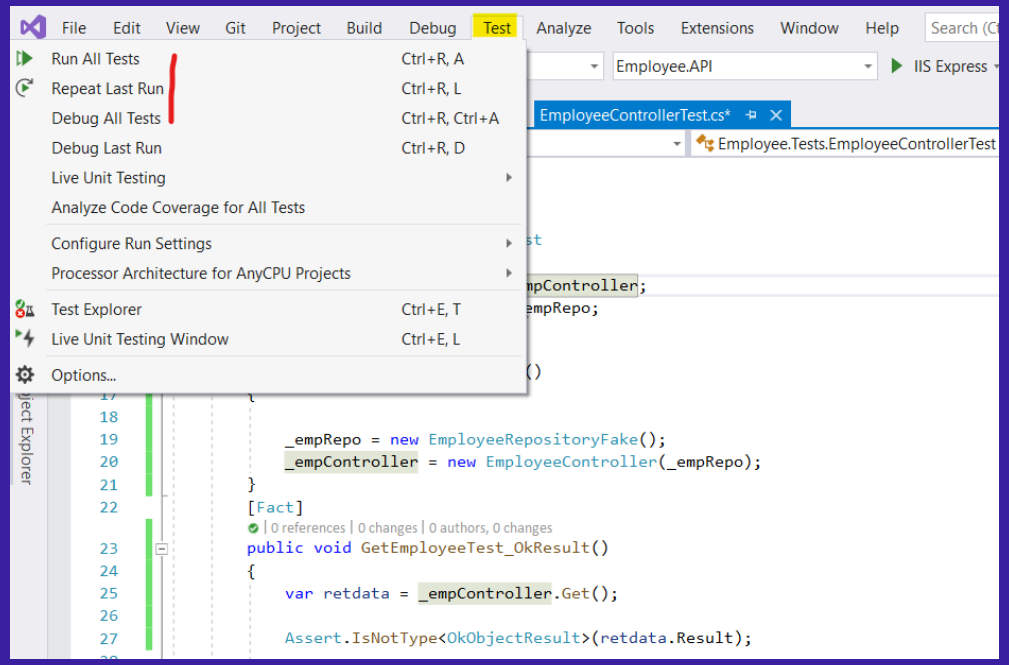
xUnit – Setting up Dependency

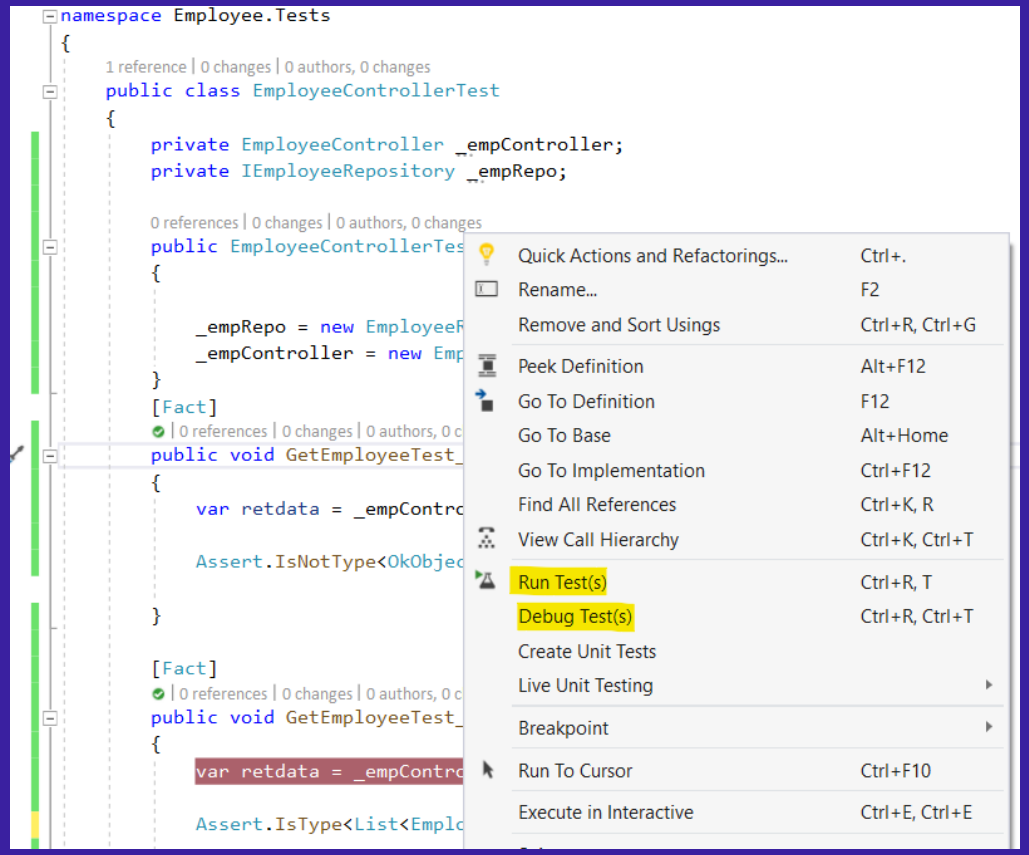
```
namespace Employee.Tests
{
    1 reference | 0 changes | 0 authors, 0 changes
    public class EmployeeRepositoryFake : IEmployeeRepository
    {
        List<EmployeeEntity> empFakeStore;
        1 reference | 0 changes | 0 authors, 0 changes
        public EmployeeRepositoryFake()
        {
            empFakeStore= new List<EmployeeEntity>
            {
                new EmployeeEntity { Id =1 , Name = "Amal", City = "Trvindrum", Worklocation = "India"},
                new EmployeeEntity { Id =2 , Name = "Dev", City = "Kochi", Worklocation = "India"},
                new EmployeeEntity { Id =3 , Name = "Tony", City = "New York", Worklocation = "US"},
                new EmployeeEntity { Id =4 , Name = "Allan", City = "London", Worklocation = "UK"},
                new EmployeeEntity { Id =5 , Name = "Sankar", City = "Delhi", Worklocation = "India"},
            };
        }
        4 references | 0 changes | 0 authors, 0 changes
        public int AddEmployee(EmployeeEntity Employee)
        {
            throw new NotImplementedException();
        }

        3 references | 0 changes | 0 authors, 0 changes
        public bool DeleteEmployee(int employeeId)
        {
            throw new NotImplementedException();
        }

        4 references | 0 changes | 0 authors, 0 changes
        public List<EmployeeEntity> EmployeeList()
        {
            return empFakeStore;
        }
```

xUnit – Writing Unit Tests

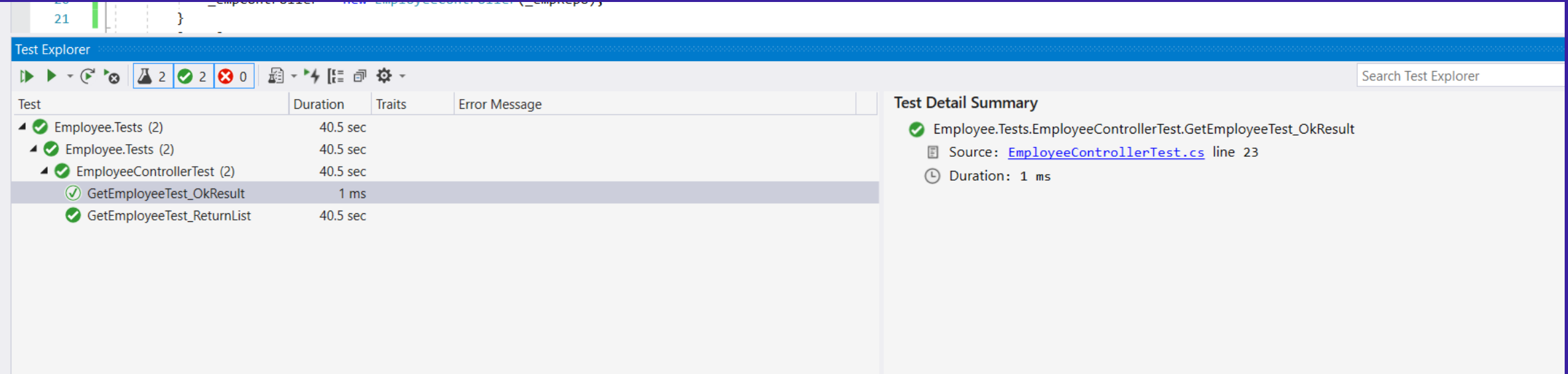
```
namespace Employee.Tests
{
    1 reference | 0 changes | 0 authors, 0 changes
    public class EmployeeControllerTest
    {
        private EmployeeController _empController;
        private IEmployeeRepository _empRepo;

        0 references | 0 changes | 0 authors, 0 changes
        public EmployeeControllerTest()
        {

            _empRepo = new EmployeeRepositoryFake();
            _empController = new EmployeeController(_empRepo);
        }
        [Fact]
        | 0 references | 0 changes | 0 authors, 0 changes
        public void GetEmployeeTest_OkResult()
        {
            var retdata = _empController.Get();

            Assert.IsNotType<OkObjectResult>(retdata.Result);

        }

        [Fact]
        | 0 references | 0 changes | 0 authors, 0 changes
        public void GetEmployeeTest_ReturnList()
        {
            var retdata = _empController.Get();

            Assert.IsType<List<EmployeeEntity>>(retdata.Value);

        }
    }
```

xUnit – Executing Unit Tests





xUnit – View Test Results



Live Unit Testing

```
15  public class EmployeeController : ControllerBase
16  {
17      private readonly IEmployeeRepository _empRepo;
        0 references | 0 changes | 0 authors, 0 changes
18      public EmployeeController(IEmployeeRepository EmpRepo)
19      {
20          _empRepo = EmpRepo;
21
22      }
23
24      [HttpGet]
        0 references | 0 changes | 0 authors, 0 changes
25      public ActionResult<EmployeeEntity> Get()
26      {
27          return _empRepo.EmployeeList().FirstOrDefault();
28      }
29
30      [HttpGet]
31      [Route("{id}")]
        0 references | 0 changes | 0 authors, 0 changes
32      public ActionResult<EmployeeEntity> GetEmployee([FromRoute] int Id)
33      {
34          return _empRepo.GetEmployeeDetails(Id);
35      }
36
37      [HttpPost]
38
39
        1 reference | 0 changes | 0 authors, 0 changes
40      public IActionResult Post(  EmployeeEntity Employee)
41      {
42          if (!ModelState.IsValid)
43          {
44              return BadRequest(ModelState);
```

Thanks for joining!