



# Integration Testing

---

.NET CORE

***Integration tests*** ensure that an app's components function correctly at a level that includes the app's supporting infrastructure, such as the database, file system, and network.

[HTTPS://DOCS.MICROSOFT.COM/EN-US/ASPNET/CORE/TEST/INTEGRATION-TESTS?VIEW=ASPNETCORE-3.1](https://docs.microsoft.com/en-us/aspnet/core/test/integration-tests?view=aspnetcore-3.1)

# Integration Testing - Overview

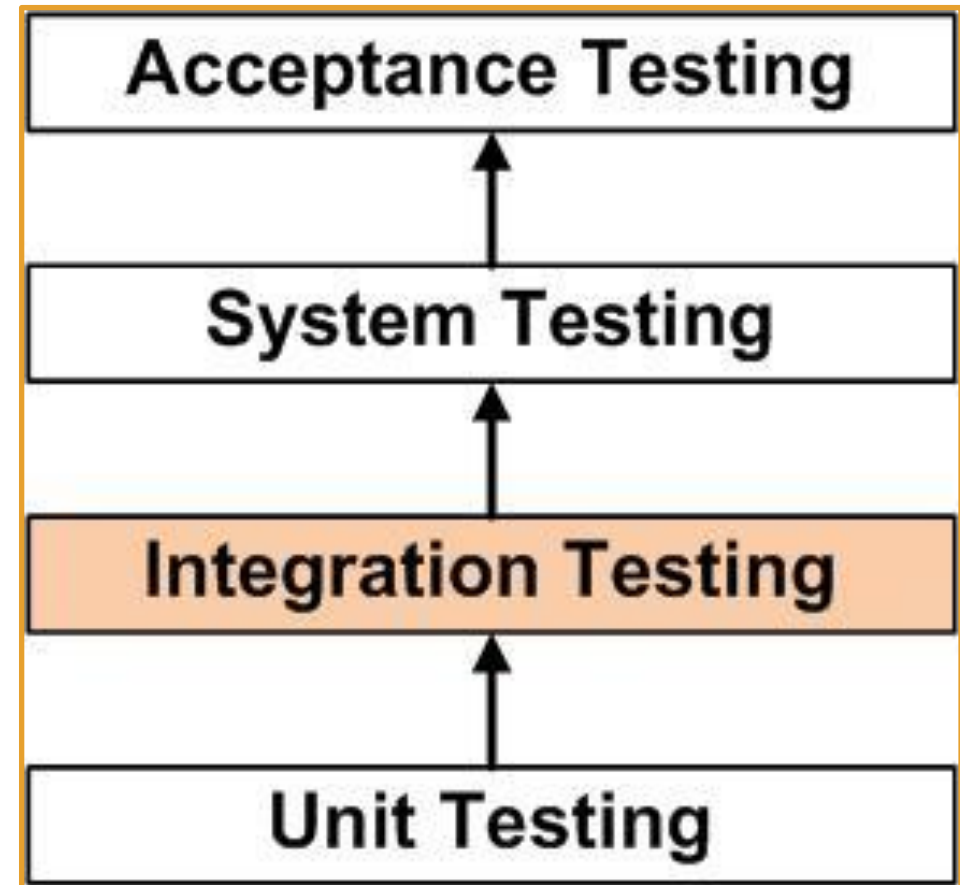
<https://docs.microsoft.com/en-us/aspnet/core/test/integration-tests?view=aspnetcore-3.1#introduction-to-integration-tests>

**Integration tests** are the second level of testing. They evaluate an app's components on a broader level than **unit tests**.

**Unit tests** are used to test isolated software components, like methods.

**Integration tests** confirm that two or more app components work together to produce an expected result.

Broader tests of the app's infrastructure and framework can include everything from UI to DB. A different database or different app settings might be used for the tests.



# Integration Tests – Cons

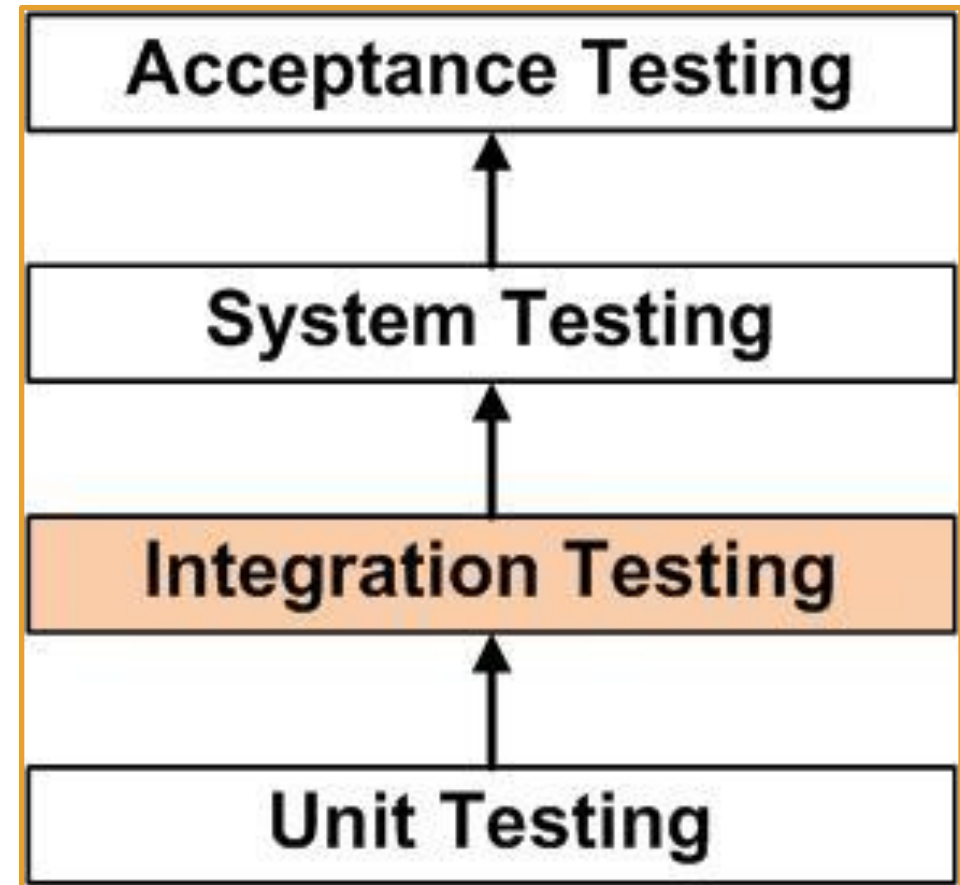
<https://docs.microsoft.com/en-us/aspnet/core/test/integration-tests?view=aspnetcore-3.1#aspnet-core-integration-tests>

Integration tests:

- use production components,
- require more code and data processing,
- take longer to run.

Limit the use of *integration tests* to the most important infrastructure scenarios.

If a behavior can be tested using both a unit test and an integration test, choose the unit test.



# Integration Testing – Requirements/Setup

<https://docs.microsoft.com/en-us/aspnet/core/test/integration-tests?view=aspnetcore-3.1#aspnet-core-integration-tests>

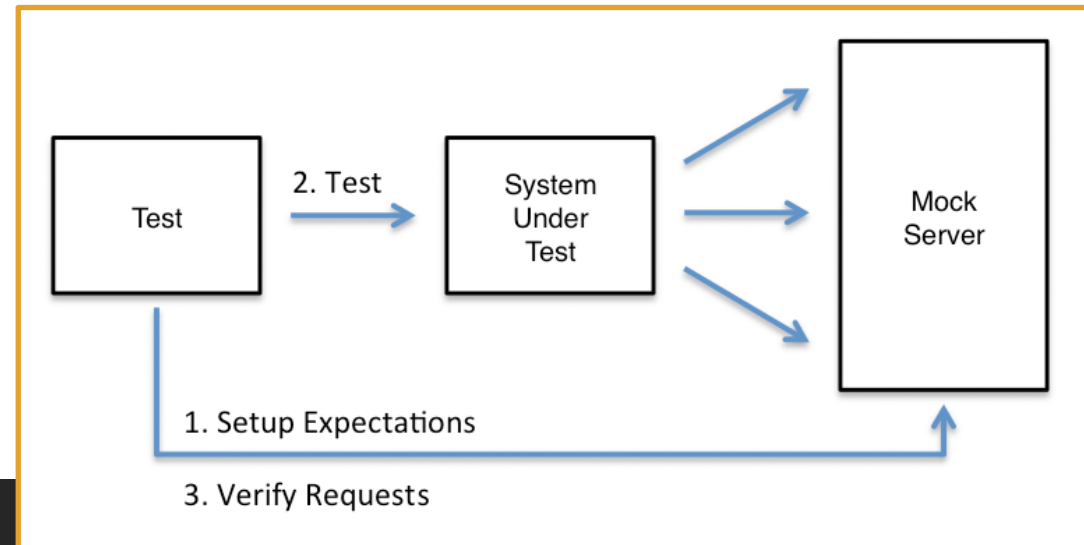
To complete Integration tests in ASP.NET Core, you need:

- a test project containing the tests and having a reference to the **SUT (System Under Test)**.
- a test web host (created by the test project) for the **SUT** that uses a test server client to handle requests and responses with the **SUT**.
- a test runner is used to execute the tests and report the test results.
- the test app to specify the **Web SDK** in its project file (`<Project Sdk="Microsoft.NET.Sdk.Web">`).
- the test app to reference the **Microsoft.AspNetCore.Mvc.Testing** package

The workflow of Integration Testing follows the familiar Arrange/Act/Assert process.

Each test:

1. configures a web host
2. Creates a test server client
3. Arrange...
4. Act...
5. Assert...





# Integration Testing – Requirements/Setup

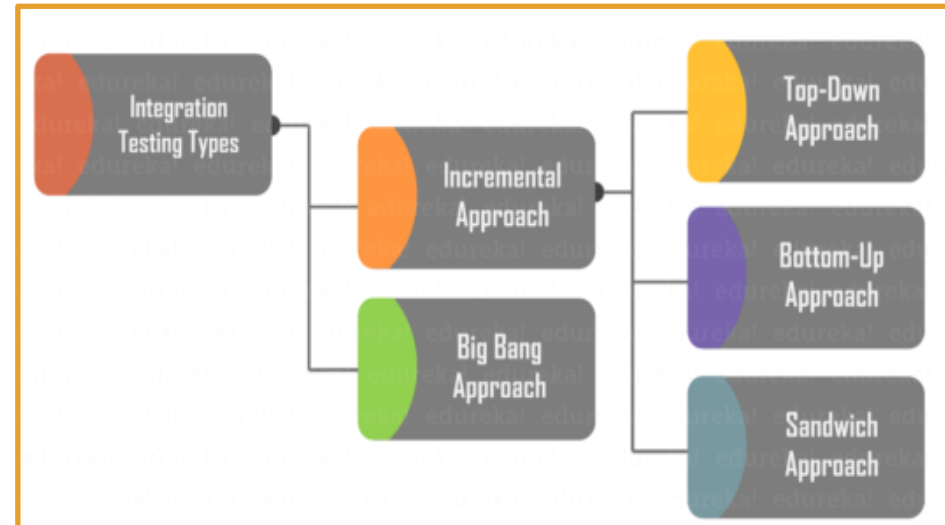
<https://docs.microsoft.com/en-us/aspnet/core/test/integration-tests?view=aspnetcore-3.1#aspnet-core-integration-tests>

The *Microsoft.AspNetCore.Mvc.Testing* package handles the following tasks:

- Copies the dependencies file (.deps) from the SUT into the test project's bin directory.
- Sets the **content root** to the SUT's project root so that static files and pages/views are found when the tests are executed.
- Provides the *WebApplicationFactory* class to streamline bootstrapping the SUT with TestServer.

You can [use a mock database](#) for testing.

In an **MVC** app, tests are usually organized by **Controller** classes and named after the **Controllers** they test.



# Integration Testing Setup (Simple) – Step By Step

<https://docs.microsoft.com/en-us/aspnet/core/test/integration-tests?view=aspnetcore-3.1#aspnet-core-integration-tests>

<https://docs.microsoft.com/en-us/aspnet/core/test/integration-tests?view=aspnetcore-3.1#basic-tests-with-the-default-webapplicationfactory>

1) The test project inherits

- `IClassFixture<WebApplicationFactory<TEntryPoint>>`.
- “TEntryPoint” is your app entry file, usually `Startup.cs`.

2) create a property:

```
private readonly WebApplicationFactory<TEntryPoint> _factory;
```

3) Inject the `WebApplicationFactory<TEntryPoint>` into the test project constructor.

4) In each test,

- `var client = _factory.CreateClient();` //Arrange
- `var response = await client.MethodToTest(url);` //Act
- `response.EnsureSuccessStatusCode();` //200-299 Assert
- `Assert.Equal(verify the response was as expected)` //Assert

```
public class BasicTests
    : IClassFixture<WebApplicationFactory<RazorPagesProject.Startup>>
{
    private readonly WebApplicationFactory<RazorPagesProject.Startup> _factory;

    public BasicTests(WebApplicationFactory<RazorPagesProject.Startup> factory)
    {
        _factory = factory;
    }

    [Theory]
    [InlineData("/")]
    [InlineData("/Index")]
    [InlineData("/About")]
    [InlineData("/Privacy")]
    [InlineData("/Contact")]
    public async Task Get_EndpointsReturnSuccessAndCorrectContentType(string url)
    {
        // Arrange
        var client = _factory.CreateClient();

        // Act
        var response = await client.GetAsync(url);

        // Assert
        response.EnsureSuccessStatusCode(); // Status Code 200-299
        Assert.Equal("text/html; charset=utf-8",
            response.Content.Headers.ContentType.ToString());
    }
}
```