Module 11

Testing and Troubleshooting

infoSupport
Solid Innovator

Module Overview

- Testing Web Applications
- Implementing an Exception Handling Strategy
- Logging Web Applications

Lesson 1: Testing Web Applications

- Why Perform Unit Tests?
- Principles of Test-Driven Development
- Writing Loosely Coupled Web Components
- Writing Unit Tests for Web Components
- Demonstration: How to Run Unit Tests
- Using Mocking Frameworks

Why Perform Unit Tests?

- Types of Tests:
  - Unit tests
  - Integration tests
  - Acceptance tests
- Unit tests verify that small units of functionality work as designed
  - Arrange. This phase of a unit test arranges data to run the test on
  - Act. This phase of the unit test calls the methods you want to test
  - Assert. This phase of the unit test checks that the results are as expected
- Any unit test that fails is highlighted in Visual Studio whenever you run the test or debug the application
- Once defined, unit tests run throughout development and highlight any changes that cause them to fail

Principles of Test-Driven Development

Write the Test

- Understand the problem
- Specify the desired behavior
- Run the test
- Test fails

Pass the Test

- Write application code
- Run the test
- Test passes

Refactor

- Clean the code and remove assumptions
- Test passes

Writing Loosely Coupled Web Components

- Loose coupling means that each component in a system requires few or no internal details of the other components in the system
- A loosely coupled application is easy to test because it is easier to replace a fully functional instance of a class with a simplified instance that is specifically designed for the test
- Loose coupling makes it easier to replace simple components with more sophisticated components
- Dependency injection inherently supports loose coupling

Writing Unit Tests for Web Components

- You can test an ASP.NET Core Web web application project by adding a test project to the solution
- Model classes can be tested by instantiating them in-memory, arranging their property values, acting on them by calling a method, and asserting that the result was as expected

Testing a Page

You can test a page by:

- Creating a service
- Implementing and using a service in the application
- Implementing a test double service
- Using a test double to test a page

Using a Test Double in a Unit Test

```
[TestMethod]
public void IndexModelShouldBeListOfProducts()
{
  // Arrange
  var productsService = new FakeProductsService();
  var expectedProducts = new[] { new Product(), new Product(), new Product()
}.AsQueryable();
  productsService.Products = expectedProducts;

  var pageModel = new ProductPageModel(productsService);

  // Act
  pageModel.OnGet();

  // Assert
  var actualProducts = Assert.IsAssignableFrom<List<Product>>(pageModel.Products);
Assert.Equal(
    expectedProducts.OrderBy(m => m.Id).Select(m => m.Text),
    actualProducts.OrderBy(m => m.Id).Select(m => m.Text));
}
```

Demonstration: How to Run Unit Tests

In this demonstration, you will see how to:

- Add a new test project, **ProductsWebsite.Tests**, to a solution to test an ASP.NET Core web application
- Create code for two unit tests
- Observe the results of the unit tests – one of them fails and the other one passes
- Fix the code
- Observe the results of the unit tests – both of them pass

Using Mocking Frameworks

- A mocking framework automates the creation of mock objects during tests
  - You can automate the creation of a single object
  - You can automate the creation of multiple objects of the same type
  - You can automate the creation of multiple objects that implement different interfaces
- The mocking framework saves time when writing unit tests

Lesson 2: Implementing an Exception Handling Strategy

- Raising and Catching Exceptions
- Working with Multiple Environments
- Configuring Error Handling
- Demonstration: How to Configure Exception Handling

Raising and Catching Exceptions

- The most common method to catch an exception is to use the **try/catch** block
- You can add custom exceptions or use existing ones

```
throw new ArgumentNullException();
…
try
{
            price = product.GetPriceWithTax(-20);
}
catch (InvalidTaxException ex)
{
    return Content("Tax cannot be negative");
}
```

Working with Multiple Environments

- Use the environment variable **ASPNETCORE_ENVIRONMENT** to determine application environment
- The **IHostingEnvironment** interface exposes useful methods:
  - **IsDevelopment**
  - **IsStaging**
  - **IsProduction**
  - **IsEnvironment**(*Environment Name*)

Using Environments in Page Content

Use the environment tag helper to differentiate between environments inside page content

```
<environment include="Development">
    <script src="~/Scripts/jquery.js"></script>
    <script src="~/Scripts/popper.js"></script>
    <script src="~/Scripts/bootstrap.js"></script>
</environment>
<environment include="Production,Staging">
    <script src="~/Scripts/vendor.min.js"></script>
</environment>
```

Configuring Error Handling

In ASP.NET Core applications, there are many ways to handle errors including:

- Using the developer exception page
- Using an exception handler to direct to a custom error page
- Using status code pages
- Using exception filters to catch exceptions in specific actions and controllers

Configuring Error Handling Example

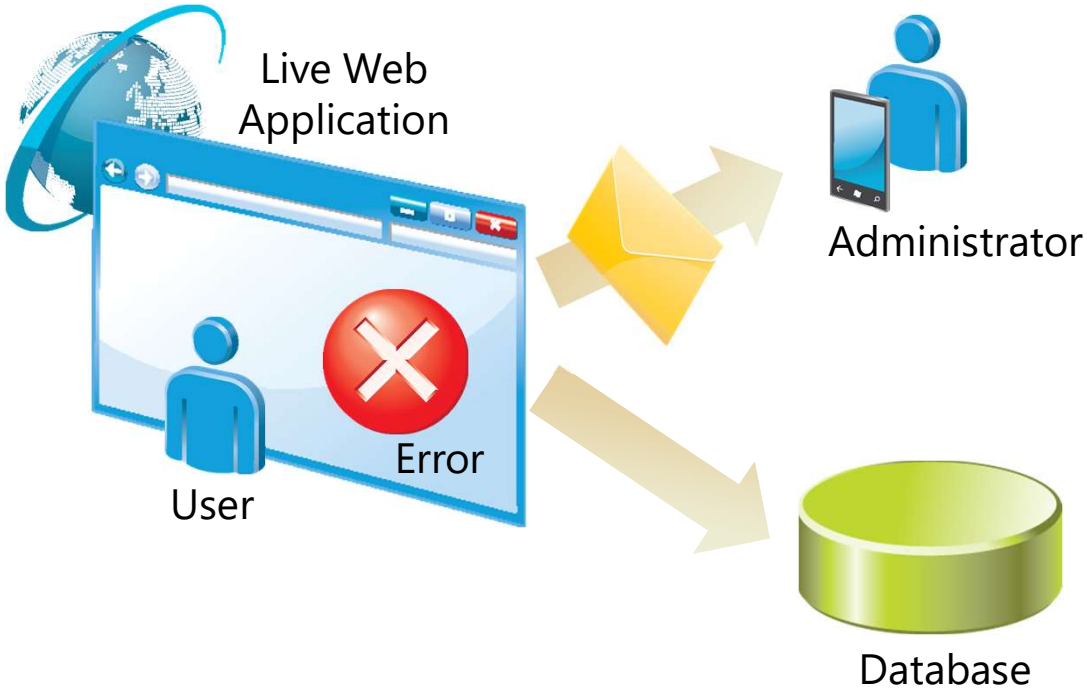
```
//Program.cs

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment()) {
    app.UseExceptionHandler("/Error");
}
```

Lesson 3: Logging in Web Applications

- Logging Exceptions
- Logging in ASP.NET Core
- Demonstration: How to Log in a Web Application

Logging Exceptions



When an exception occurs, the application sends an email message to the administrators, and logs full details of the exception to a database.

Logging in ASP.NET Core

```
public ActionResult OnGet()
{
    _logger.LogDebug("Index Page was entered");
    try
    {
        int x = 3;
        x -= 3;
        int result = 30 / x;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "An error occured while dividing!");
    }
    return Content(“Result from Page");
}
```

Demonstration: How to Log in a Web Application

In this demonstration, you will see how to:

- Configure logging of an ASP.NET Core application
- Write log messages to a file and to the console
- Investigate and solve problems in an ASP.NET Core application using log messages

Lab: Testing and Troubleshooting

- Exercise 1: Testing a Page using a Fake Service
- Exercise 2: Adding Logging

Estimated Time: 60 minutes