go deploy

Screenshots

Module 10: Testing and Troubleshooting

Lab: Testing and Troubleshooting

Scenario

To improve the quality of a web application for a shirt store, your development team has decided to add testing and troubleshooting to the web application. You have been asked to add unit tests to test a model and a controller. You have also been told that when an error occurs the browser should display a detailed exception page on the development environment, and a custom error page on the production environment. In addition, you are required to add logging to the web application.

Exercise 1: Testing a Model

Scenario

You are required to develop an ASP.NET Core application in a test-driven environment. In this exercise, you will create an MSTest testing project and add it to the solution, add the ASP.NET Core website application to its list of dependencies, and then test the Shirt model.

The main tasks for this exercise are as follows:

- Add a testing project
- Write a test for a model
- Run the unit test - it should fail
- Implement the model class so the test will pass
- Run the unit test - it succeeds

Task 1: Add a Testing Project

1. Navigate to **D:\Allfiles\Mod10\Labfiles\01_ShirtStore_begin**, and then double-click **ShirtStore.sln**.

   Note: If a **Security Warning for ShirtStore** dialog box appears, verify that the **Ask me for every project in this solution** check box is cleared, and then click OK.

2. In the **ShirtStore -- Microsoft Visual Studio** window, in Solution Explorer, right-click **Solution**

3. In the **Add a new project** dialog box, enter **test** in the search box.

4. In the **Add a new project** dialog box, in the result pane, click **MSTest Test Project**.

5. In the **Add a new project** dialog box, in the **Name** box, type ShirtStoreWebsite.Tests and click **Next**.

6. In the **Additional information** dialog, ensure that **.NET 6.0 (Long-term support)** is selected, and then click **Create**.

7. In Solution Explorer, under **ShirtStoreWebsite.Tests**, right-click **Dependencies**, and then click **Add Project Reference**.

8. In the **Reference Manager - ShirtStoreWebsite.Tests** dialog box, in the result pane, select the **ShirtStoreWebsite** check box, and then click **OK**.

Task 2: Write a test for a model

1. In Solution Explorer, right-click **ShirtStoreWebsite.Tests**, point to **Add**, and then click **New Folder**.

2. In the **NewFolder** box, type Models, and then press Enter.

3. In Solution Explorer, under **ShirtStoreWebsite.Tests**, right-click **UnitTest1.cs**, and then click **Rename**.

4. In the **UnitTest1** box, type ShirtTest, and then press Enter.

5. In the **Microsoft Visual Studio** dialog box, click **Yes** to the question on renaming references.

6. In Solution Explorer, right-click **ShirtTest.cs**, and then click **Cut**.

7. In Solution Explorer, right-click **Models**, and then click **Paste**.

8. In the **ShirtTest.cs** code window, check the name of the class, and change it to ShirtTest if needed, so it looks like the following:

```
namespace ShirtStoreWebsite.Tests.Models
{
   [TestClass]
   public class ShirtTest
   {
```

9. In the **ShirtTest.cs** code window, select the following code:

```
public void TestMethod1()
{
}
```

```
public void IsGetFormattedTaxedPriceReturnsCorrectly()
{
   Shirt shirt = new Shirt
   {
         Price = 10F,
         Tax = 1.2F
   };

   string taxedPrice = shirt.GetFormattedTaxedPrice();

   Assert.AreEqual("$12.00", taxedPrice);
}
```

Note: You can verify that Visual Studio has automatically added a reference to the **ShirtStoreWebsite.Models** namespace.

- [ ] 11. In the **ShirtStore -- Microsoft Visual Studio** window, on the **File** menu, click **Save All**.

Task 3: Run the unit test -- it should fail

- [ ] 1. In the **ShirtStore - Microsoft Visual Studio** window, on the **Test** menu, click on **Run All Tests**.

Note: The **Test Explorer** displays one failed test: **IsGetFormattedTaxedPriceReturnsCorrectly**.

Task 4: Implement the model class so the test will pass

- [ ] 1. In the **ShirtStore - Microsoft Visual Studio** window, in Solution Explorer, under **ShirtStoreWebsite**, expand **Models**, and then click **Shirt.cs**.

- [ ] 2. In the **Shirt.cs** code window, select the following code:

```
return Price.ToString($"C2", CultureInfo.GetCultureInfo("en-US"));
```

- [ ] 3. Replace the selected code with the following code:

```
return (Price * Tax).ToString($"C2", CultureInfo.GetCultureInfo("en-US"));
```

- [ ] 4. In the **ShirtStore -- Microsoft Visual Studio** window, on the **File** menu, click **Save All**.

Task 5: Run the unit test -- it succeeds

- [ ] 1. In the **ShirtStore - Microsoft Visual Studio** window, on the **Test** menu, point to **Run**, and then click **All Tests**.

IsGetFormattedTaxedPriceReturnsCorrectly.

✓ **Results**: After completing this exercise, you will have created a test project and tested a model while fixing its code, as an example of Test-driven Development.

Exercise 2: Testing a Controller using a Fake Repository

Scenario

After creating a test project and testing the model, you are now required to test the controller. To do this, you will need to create a repository interface as a dependency for the controller to gain access to the data. To test the controller, you will create a fake repository as a substitute, then provide it to the controller via its constructor.

The main tasks for this exercise are as follows:

- Create an interface repository
- Implement the interface repository by using a fake repository
- Pass the fake repository to the constructor of a controller
- Write a test for a controller
- Run the unit test - it should fail
- Implement the controller class so the test will pass
- Run the unit test - it succeeds

Task 1: Create an interface repository

1. In the **ShirtStore - Microsoft Visual Studio** window, in Solution Explorer, right-click **ShirtStoreWebsite**, point to **Add**, and then click **New Folder**.

2. In the **NewFolder** box, type **Services**, and then press Enter.

3. In the **ShirtStore - Microsoft Visual Studio** window, in Solution Explorer, right-click **Services**, point to **Add**, and then click **New Item...**.

4. In the **Add New Item -- ShirtStoreWebsite** dialog box, click **Interface**.

5. In the **Add New Item -- ShirtStoreWebsite** dialog box, in the **Name:** box, type **IShirtRepository**, and then click **Add**.

6. In the **IShirtRepository.cs** code window, select the following code:

```
}
```

- [ ] 7. Replace the selected code with the following code:

```
public interface IShirtRepository
{
   IEnumerable<Shirt> GetShirts();
   bool AddShirt(Shirt shirt);
   bool RemoveShirt(int id);
}
```

Task 2: Implement the interface repository using a fake repository

- [ ] 1. In the **ShirtStore - Microsoft Visual Studio** window, in Solution Explorer, right-click **ShirtStoreWebsite.Tests**, point to **Add**, and then click **New Folder**.
- [ ] 2. In the **NewFolder** box, type FakeRepositories, and then press Enter.
- [ ] 3. In the **ShirtStore - Microsoft Visual Studio** window, in Solution Explorer, right-click **FakeRepositories**, point to **Add**, and then click **Class...**.
- [ ] 4. In the **Add New Item - ShirtStoreWebsite.Tests** dialog box, in the **Name** box, type FakeShirtRepository, and then click **Add**.
- [ ] 5. In the **FakeShirtRepository.cs** code window, select the following code:

```
 internal class FakeShirtRepository
```

- [ ] 6. Replace the selected code with the following code:

```
internal class FakeShirtRepository
{
}
```

- [ ] 7. In the **FakeShirtRepository.cs** code window, locate the following code:

```
        public IEnumerable<Shirt> GetShirts()
        {
            return new List<Shirt>()
            {
                    new Shirt { Color = ShirtColor.Black, Size = ShirtSize.S, Price = 11F }
                    new Shirt { Color = ShirtColor.Gray, Size = ShirtSize.M, Price = 12F },
                    new Shirt { Color = ShirtColor.White, Size = ShirtSize.L, Price = 13F }
            };
        }

        public bool AddShirt(Shirt shirt)
        {
            return true;
        }

        public bool RemoveShirt(int id)
        {
            return true;
        }
    }
```

Task 3: Pass the fake repository to the constructor of a controller

☐ 1. In the **ShirtStore - Microsoft Visual Studio** window, in Solution Explorer, expand **Controllers**, and then click **ShirtController.cs**.

☐ 2. In the **ShirtController.cs** code window, locate the following code:

```
using ShirtStoreWebsite.Models;
```

☐ 3. Place the cursor at the end of the located code, press Enter, and then type the following code:

```
using ShirtStoreWebsite.Services;
```

☐ 4. In the **ShirtController.cs** code window, locate the following code:

```
public class ShirtController : Controller
{
```

☐ 5. Place the cursor after the **{** (opening brace) sign, press Enter, and then type the following code:

```
private IShirtRepository _repository;

public ShirtController(IShirtRepository repository)
{
   _repository = repository;
}
```

- [ ] 2. In the **NewFolder** box, type Controllers, and then press Enter.
- [ ] 3. In the **ShirtStore - Microsoft Visual Studio** window, in Solution Explorer, right-click **Controllers**, point to **Add**, and then click **Class...**.
- [ ] 4. In the **Add New Item - ShirtStoreWebsite.Tests** dialog box, in the **Name** box, type ShirtControllerTest, and then click **Add**.
- [ ] 5. In the **ShirtControllerTest.cs** code window, select the following code:

```
internal class ShirtControllerTest
{
}
```

- [ ] 6. Replace the selected code with the following code:

```
[TestClass]
public class ShirtControllerTest
{
   [TestMethod]
   public void IndexModelShouldContainAllShirts()
   {
         IShirtRepository fakeShirtRepository = new FakeShirtRepository();
         ShirtController shirtController = new ShirtController(fakeShirtRepository)
         ViewResult viewResult = shirtController.Index() as ViewResult;
         List<Shirt> shirts = viewResult.Model as List<Shirt>;
         Assert.AreEqual(shirts.Count, 3);
   }
}
```

- [ ] 7. In the **ShirtStore -- Microsoft Visual Studio** window, on the **File** menu, click **Save All**.

Task 5: Run the unit test -- it should fail

- [ ] 1. In the **ShirtStore - Microsoft Visual Studio** window, on the **Test** menu, point to **Run**, and then click **All Tests**.

> **Note**: The **Test Explorer** displays one failed test: **IndexModelShouldContainAllShirts**, and one passed test: **IsGetFormattedTaxedPriceReturnsCorrectly**.

Task 6: Implement the controller class so the test will pass

- [ ] 1. In the **ShirtStore - Microsoft Visual Studio** window, in Solution Explorer, under **ShirtStoreWebsite**, under **Controllers**, click **ShirtController.cs**.

go deploy ☰

```
return View();
```

- [ ] 3. Replace the selected code with the following code:

```
IEnumerable<Shirt> shirts = _repository.GetShirts();
return View(shirts);
```

- [ ] 4. In the **ShirtController.cs** code window, locate the following code:

```
public IActionResult AddShirt(Shirt shirt)
{
```

- [ ] 5. Place the cursor at the end of the located code, press Enter, and then type the following code:

```
_repository.AddShirt(shirt);
```

- [ ] 6. In the **ShirtController.cs** code window, locate the following code:

```
public IActionResult Delete(int id)
{
```

- [ ] 7. Place the cursor at the end of the located code, press Enter, and then type the following code:

```
_repository.RemoveShirt(id);
```

- [ ] 8. In the **ShirtStore -- Microsoft Visual Studio** window, on the **File** menu, click **Save All**.

Task 7: Run the unit test -- it succeeds

- [ ] 1. In the **ShirtStore - Microsoft Visual Studio** window, on the **Test** menu, point to **Run**, and then click **All Tests**.

> **Note**: The **Test Explorer** displays two passed tests: **IndexModelShouldContainAllShirts** and **IsGetFormattedTaxedPriceReturnsCorrectly**.

✓ **Results**: After completing this exercise, you have tested a controller by using a fake repository.

Exercise 3: Implementing a Repository in the MVC Project

Scenario

ShirtRepository repository, which will get data from a database and update a database. The ShirtRepository repository will be registered in the ConfigureService method.

The main tasks for this exercise are as follows:

- Implement the interface repository in a repository class
- Register the repository as a service
- Run the MVC application

Task 1: Implement the interface repository in a repository class

1. In the **ShirtStore - Microsoft Visual Studio** window, in Solution Explorer, right-click **Services**, point to **Add**, and then click **Class....**

2. In the **Add New Item - ShirtStoreWebsite** dialog box, in the **Name** box, type ShirtRepository, and then click **Add**.

3. In the **ShirtRepository.cs** code window, select the following code:

```
public class ShirtRepository
```

4. Replace the selected code with the following code:

```
public class ShirtRepository : IShirtRepository
```

5. In the **ShirtRepository.cs** code window, locate the following code:

```
public class ShirtRepository : IShirtRepository
{
```

Note: You'll get a warning that the interface is not implemented. You can ignore it - you'll add the implementation in the next steps.

6. Ensure that the cursor is at the end of the located code, press Enter, and then type the following code:

```
public ShirtRepository(ShirtContext context)
{
   _context = context;
}

public IEnumerable<Shirt> GetShirts()
{
   return _context.Shirts.ToList();
}

public bool AddShirt(Shirt shirt)
{
   _context.Add(shirt);
   int entries = _context.SaveChanges();
   if(entries > 0)
   {
         return true;
   }
   else
   {
         return false;
   }
}

public bool RemoveShirt(int id)
{
   var shirt = _context.Shirts.SingleOrDefault(m => m.Id == id);
   _context.Shirts.Remove(shirt);
   int entries = _context.SaveChanges();
   if (entries > 0)
   {
         return true;
   }
   else
   {
         return false;
   }
}
```

Task 2: Register the repository as a service

- [ ] 1. In the **ShirtStore - Microsoft Visual Studio** window, in Solution Explorer, click **Program.cs**.
- [ ] 2. In the **Program.cs** code window, locate the following code:

```
builder.Services.AddDbContext<ShirtContext>(options =>
   options.UseSqlServer(_configuration.GetConnectionString("DefaultConnection")));
```

- [ ] 3. Ensure that the cursor is at the end of the located code, press Enter twice, and then type the go deploy ☰

```
builder.Services.AddScoped<IShirtRepository, ShirtRepository>();
```

☐ 4. In the **ShirtStore -- Microsoft Visual Studio** window, on the **File** menu, click **Save All**.

Task 3: Run the MVC application

☐ 1. In the **ShirtStore - Microsoft Visual Studio** window, on the **Debug** menu, click **Start Without Debugging**.

> **Note**: The browser displays the **Shirt Store** page.

☐ 2. In Microsoft Edge, click **Close**.

> **Results**: After completing this exercise, you have developed a repository to support a functional MVC application.

Exercise 4: Adding Exception Handling

Scenario

You have been asked to add exception handling to the web application. If an error occurs while running the application, two use cases must be implemented: In the case that the application is running in a development environment and an error occurs, the user would see a detailed error page with information on where to find the error. In the case that the application is running in a production environment, a custom none-informative page would be displayed claiming there was an error. You are required to add exception handling to each of the use cases.

The main tasks for this exercise are as follows:

- Add exception handling in Program.cs
- Create a temporary exception for testing
- Run the application in the development environment
- Run the application in the production environment
- Remove the temporary exception

Task 1: Add exception handling in Program.cs

☐ 1. In the **ShirtStore - Microsoft Visual Studio** window, in Solution Explorer, in the **Program.cs** code window, locate the following code:

```
var app = builder.Build();
```

go deploy

```
if (builder.Environment.IsDevelopment())
{
   app.UseDeveloperExceptionPage();
}
else
{
   app.UseExceptionHandler("/error.html");
}
```

Task 2: Create a temporary exception for testing

- [ ] 1. In the **ShirtStore - Microsoft Visual Studio** window, in Solution Explorer, under **Controllers**, click **ShirtController.cs**.
- [ ] 2. In the **ShirtController.cs** code window, select the following code:

```
_repository.RemoveShirt(id);
```

- [ ] 3. Replace the selected code with the following code:

```
_repository.RemoveShirt(-1);
```

- [ ] 4. In the **ShirtStore -- Microsoft Visual Studio** window, on the **File** menu, click **Save All**.

Task 3: Run the application in the development environment

- [ ] 1. In the **ShirtStore - Microsoft Visual Studio** window, on the toolbar, click the arrow next to the **Start Debugging** button, and then click **Development**.
- [ ] 2. In the **ShirtStore - Microsoft Visual Studio** window, on the **Debug** menu, click **Start Without Debugging**.
- [ ] 3. In Microsoft Edge, at the top of the **Shirts in stock** table, click the **Delete** link.

> ⚠ **Note**: The browser displays the **detailed exception** page, which shares a lot of information about the application.

- [ ] 4. In Microsoft Edge, click **Close**.

Task 4: Run the application in the production environment

- [ ] 1. In the **ShirtStore - Microsoft Visual Studio** window, on the toolbar, click the arrow next to the **Start Debugging** button, and then click **Production**.
- [ ] 2. In the **ShirtStore - Microsoft Visual Studio** window, on the **Debug** menu, click **Start Without Debugging**.

⚠ **Note**: The browser displays a custom error page **error.html**, which is located in the **wwwroot** folder.

- [ ] 4. In Microsoft Edge, click **Close**.

Task 5: Remove the temporary exception

- [ ] 1. In the **ShirtStore - Microsoft Visual Studio** window, in Solution Explorer, under **Controllers**, click **ShirtController.cs**.

  - [ ] 2. In the **ShirtController.cs** code window, select the following code:

```
_repository.RemoveShirt(-1);
```

- [ ] 2. Replace the selected code with the following code:

```
_repository.RemoveShirt(id);
```

- [ ] 3. In the **ShirtStore -- Microsoft Visual Studio** window, on the **File** menu, click **Save All**.

✓ **Results**: After completing this exercise, you have added exception handling to an MVC application, by displaying a custom error page or the developer exception page if an exception is thrown.

Exercise 5: Adding Logging

Scenario

You are required to provide logging to the ASP.NET Core application by using the Serilog library, while configuring the logging separately by using appsettings.json files to the different environments. Any trace log level logs in development would be displayed to the console, while any warning level logs in production would be written to its dedicated file. This would also require injecting the ILogger to the controller, thus would require to update the controller's test.

The main tasks for this exercise are as follows:

- Add logging to the MVC application
- Test the controller by using a mocking framework
- Run the unit test
- Run the application in the development environment
- Run the application in the production environment

Task 1: Add logging to the MVC application

go deploy

2. In the **Add New Item -- ShirtStoreWebsite** dialog box, in the navigation pane, under **Installed**, expand **ASP.NET Core**, and then click **Web**.

3. In the **Add New Item -- ShirtStoreWebsite** dialog box, in the result pane, click **App Settings File**.

4. In the **Add New Item - ShirtStoreWebsite** dialog box, in the **Name** box, type appsettings.development.json, and then click **Add**.

5. In the **appsettings.development.json** code window, select the following code:

```
"ConnectionStrings": {
   "DefaultConnection": "Server=(localdb)\\MSSQLLocalDB;Database=_CHANGE_ME;Trusted
}
```

6. Replace the selected code with the following code:

```
"Logging": {
   "LogLevel": {
         "Default": "Trace"
   }
}
```

7. In the **ShirtStore - Microsoft Visual Studio** window, in Solution Explorer, right-click **ShirtStoreWebsite**, point to **Add**, and then click **New Item...**.

8. In the **Add New Item -- ShirtStoreWebsite** dialog box, in the navigation pane, under **Installed**, expand **ASP.NET Core**, and then click **Web**.

9. In the **Add New Item -- ShirtStoreWebsite** dialog box, in the result pane, click **App Settings File**.

10. In the **Add New Item - ShirtStoreWebsite** dialog box, in the **Name** box, type appsettings.production.json, and then click **Add**.

11. In the **appsettings.production.json** code window, select the following code:

```
"ConnectionStrings": {
   "DefaultConnection": "Server=(localdb)\\MSSQLLocalDB;Database=_CHANGE_ME;Trusted
}
```

12. Replace the selected code with the following code:

go deploy

```
      "LogLevel": {
      "Default": "Warning"
      }
   }
```

- [ ] 13. In the **ShirtStore - Microsoft Visual Studio** window, in Solution Explorer, click **Program.cs**.

- [ ] 14. In the **ShirtStore - Microsoft Visual Studio** window, in Solution Explorer, in the **Program.cs** code window, locate the following code:

```
var app = builder.Build();
```

- [ ] 15. **Before** the located code, insert the following code:

```
builder.Logging.ClearProviders();
var config = builder.Configuration.GetSection("Logging");
if (builder.Environment.IsDevelopment())
{
   builder.Logging.AddConfiguration(config);
   builder.Logging.AddConsole();
}
else
{
   builder.Logging.AddFile("shirt_store_logs.txt");
}
```

- [ ] 16. In Solution Explorer, under **Controllers**, click **ShirtController.cs**.

- [ ] 17. In the **ShirtController.cs** code window, locate the following code:

```
private IShirtRepository _repository;
```

- [ ] 18. Place the cursor at the end of the located code, press Enter, and then type the following code:

```
private ILogger _logger;
```

- [ ] 19. In the **ShirtController.cs** code window, select the following code:

```
public ShirtController(IShirtRepository repository)
```

- [ ] 20. Replace the selected code with the following code:

```
public ShirtController(IShirtRepository repository, ILogger<ShirtController> logger
```

- [ ] 21. In the **ShirtController.cs** code window, locate the following code:

go deploy

☐ 22. Place the cursor at the end of the located code, press Enter, and then type the following code:

```
_logger = logger;
```

☐ 23. In the **ShirtController.cs** code window, locate the following code:

```
_repository.AddShirt(shirt);
```

☐ 24. Place the cursor at the end of the located code, press Enter, and then type the following code:

```
_logger.LogDebug($"A {shirt.Color.ToString()} shirt of size {shirt.Size.ToString()}
```

☐ 25. In the **ShirtController.cs** code window, select the following code:

```
_repository.RemoveShirt(id);
return RedirectToAction("Index");
```

☐ 26. Replace the selected code with the following code:

```
try
{
   _repository.RemoveShirt(id);
   _logger.LogDebug($"A shirt with id {id} was removed successfully.");
   return RedirectToAction("Index");
}
catch (Exception ex)
{
   _logger.LogError(ex, $"An error occured while trying to delete shirt with id of
   throw;
}
```

☐ 27. In the **ShirtStore -- Microsoft Visual Studio** window, on the **File** menu, click **Save All**.

Task 2: Test the controller by using a mocking framework

☐ 1. In the **ShirtStore - Microsoft Visual Studio** window, on the **Tools** menu, point to **NuGet Package Manager**, and then click **Package Manager Console**.

☐ 2. In the **Package Manager Console** window, type the following text, and then press Enter.

```
Install-Package Moq -Version 4.9.0 -ProjectName ShirtStoreWebsite.Tests
```

☐ 3. In the **ShirtStore - Microsoft Visual Studio** window, in Solution Explorer, under **ShirtStoreWebsite.Tests**, click **ShirtControllerTest.cs**.

go deploy

```
ShirtController shirtController = new ShirtController(fakeShirtRepository);
```

5. Replace the selected code with the following code:

```
Mock<ILogger<ShirtController>> mockLogger = new Mock<ILogger<ShirtController>>();
ShirtController shirtController = new ShirtController(fakeShirtRepository, mockLogg
```

6. In the **ShirtStore -- Microsoft Visual Studio** window, on the **File** menu, click **Save All**.

Note: ShirtControllerTest must be updated because we added a parameter to the ShirtController constructor **ILogger logger**; following the update the test will pass correctly.

Task 3: Run the unit test

1. In the **ShirtStore - Microsoft Visual Studio** window, on the **Test** menu, point to **Run**, and then click **All Tests**.

Note: The **Test Explorer** displays two passed tests: **IndexModelShouldContainAllShirts** and **IsGetFormattedTaxedPriceReturnsCorrectly**.

Task 4: Run the application in the development environment

1. In the **ShirtStore - Microsoft Visual Studio** window, on the toolbar, click the arrow next to the **Start Debugging** button, and then click **Development**.
2. In the **ShirtStore - Microsoft Visual Studio** window, on the **Debug** menu, click **Start Debugging**.
3. In the **ShirtStore - Microsoft Visual Studio** window, on the **Output** tab, in the **Show output from** list, select **ASP.NET Core Web Server**, and then click the **Clear All** button.
4. In Microsoft Edge, on the **Size** list, select **M**.
5. In Microsoft Edge, on the **Color** list, select **Yellow**.
6. In Microsoft Edge, on the **Price** box, type **10**.
7. In Microsoft Edge, on the **Tax** box, type **1.2**.
8. In Microsoft Edge, click **Add Shirt to Stock**.

Note: The new shirt was added to the **Shirts in Stock** table.

```
A Yellow shirt of size M with a price of $12.00 was added successfully.
```

- [ ] 10. In the **ShirtStore - Microsoft Visual Studio** window, on the **Output** tab, click **Clear All**.
- [ ] 11. In Microsoft Edge, on the **Shirts in stock** table, click the top **Delete** link.
- [ ] 12. In the **ShirtStore - Microsoft Visual Studio** window, in the **Output** tab, press **ctrl + F**, and then locate the following text:

```
A shirt with id 1 was removed successfully.
```

- [ ] 13. In Microsoft Edge, in the address bar, type http://localhost:[port]/Shirt/Delete/-1, and then press Enter.

> **Note**: You will need to continue execution after the application breaks. The browser displays the **DeveloperException** page.

- [ ] 14. In Microsoft Edge, click **Close**.

Task 5: Run the application in the production environment

- [ ] 1. In the **ShirtStore - Microsoft Visual Studio** window, on the toolbar, click the arrow next to the **Start Debugging** button, and then click **Production**.
- [ ] 2. In the **ShirtStore - Microsoft Visual Studio** window, on the **Debug** menu, click **Start without Debugging**.
- [ ] 3. In Microsoft Edge, in the address bar, type http://localhost:[port]/Shirt/Delete/-1, and then press Enter.

> **Note**: The browser displays **error.html** content, located in **wwwroot**, under **ShirtStoreWebsite**.

- [ ] 4. In Microsoft Edge, click **Close**.
- [ ] 5. In File Explorer, navigate to **D:\Allfiles\Mod10\Labfiles\01_ShirtStore_begin\ShirtStoreWebsite**, and then double-click **shirt_store_logs-XXXXXXXX.txt**.

> **Note**: Inspect the **ArgumentNullException** stack trace.

- [ ] 6. In the **shirt_store_logs-XXXXXXXX - Notepad** window, click **Close**.

go deploy

- ☐ 8. In Microsoft Edge, on the **Size** list, select **M**.
- ☐ 9. In Microsoft Edge, on the **Color** list, select **Yellow**.
- ☐ 10. In Microsoft Edge, on the **Price** box, type **10**.
- ☐ 11. In Microsoft Edge, on the **Tax** box, type **1.2**.
- ☐ 12. In Microsoft Edge, click **Add Shirt to Stock**.

  ⚠ **Note**: The new shirt was added to the bottom of **Shirts in Stock** table.

- ☐ 13. In Microsoft Edge, click **Close**.
- ☐ 14. In File Explorer, navigate to **D:\Allfiles\Mod10\Labfiles\01_ShirtStore_begin\ShirtStoreWebsite**, and then double-click **shirt_store_logs-XXXXXXXX.txt**.

  ⚠ **Note**: The log file does not contain another message because the action was successful and there are no errors.

- ☐ 15. In **shirt_store_logs-XXXXXXXX - Notepad** window, click **Close**.
- ☐ 16. In the **ShirtStore - Microsoft Visual Studio** window, on the **File** menu, click **Exit**.

✓ **Results**: At the end of this exercise, you have added logging in different logging levels in different environments, displaying errors or information by writing into a log file or a console output in the desired format. You also created a mock substitute by using a mocking framework.