Labs:

Para mejorar la calidad de una aplicación web para una tienda de camisas, su equipo de desarrollo ha decidido agregar pruebas y solución de problemas a la aplicación web. Se le ha pedido que agregue pruebas unitarias para probar un modelo y un controlador. También le han dicho que cuando se produce un error, el navegador debe mostrar una página de excepción detallada en el entorno de desarrollo y una página de error personalizada en el entorno de producción. Además, debe agregar el registro a la aplicación web.

Después de completar este laboratorio, podrá:

- Manejar Pruebas una aplicación ASP.NET Core MVC.

- Agreguar manejo de excepciones para los diferentes entornos.

- Agregar el registro a una aplicación ASP.NET Core MVC.

Testing a Model

Debe desarrollar una aplicación \*\* ASP.NET Core MVC \*\* en un entorno controlado por pruebas.

En este ejercicio, creará un proyecto de prueba \*\* MSTest \*\* y lo agregará a la solución, agregará la aplicación del sitio web \*\* ASP.NET Core MVC \*\* a su lista de dependencias y luego probará la \*\* Shirt \*\* model.

Las tareas principales para este ejercicio son las siguientes:

1. Agregar un proyecto de prueba

2. Escribe una prueba para un modelo

3. Ejecute la prueba de la unidad: debería fallar

4. Implemente la clase de modelo para que la prueba pase

5. Ejecute la prueba de la unidad: tiene éxito

#### Task 1: Add a testing project

Abrir

01\_ShirtStore\_begin\*\*, and then open the \*\*ShirtStore.sln\*\*.

2. In the \*\*ShirtStore\*\* solution, add a new project with the following information:

- Project name : \*\*ShirtStoreWebsite.Tests\*\*

- Project template : \*\*MSTest Test Project (.NET Core)\*\*

3. In the \*\*ShirtStoreWebsite.Tests\*\* project, add a reference of \*\*ShirtStoreWebsite\*\* to its dependencies.

#### Task 2: **Write a test for a model**

1. Create a new folder with the following information:

- Folder name: \*\*Models\*\*

- Project: \*\*ShirtStoreWebsite.Tests\*\*

2. In the \*\*ShirtStoreWebsite.Tests\*\* project, rename the \*\*UnitTest1\*\* class to \*\*ShirtTest\*\*.

3. Move the \*\*ShirtTest.cs\*\* file to the \*\*Models\*\* folder.

4. In the \*\*ShirtTest\*\* class, add a \*\*USING\*\* statement for the following namespace:

- \*\*ShirtStoreWebsite.Models\*\*

5. In the \*\*ShirtTest\*\* class code block, rename \*\*TestMethod1\*\* to \*\*IsGetFormattedTaxedPriceReturnsCorrectly\*\*.

6. In the \*\*IsGetFormattedTaxedPriceReturnsCorrectly\*\* method, add a new variable with the following information:

- Type: \*\*Shirt\*\*

- Name: \*\*shirt\*\*

- Value: \*\*new Shirt { Price = 10F, Tax = 1.2F }\*\*

**Shirt shirt = new Shirt**

**{**

**Price = 10F,** **Tax = 1.2F**

**};**

7. Add a new variable with the following information:

- Type: \*\*string\*\*

- Name: \*\*taxedPrice\*\*

- Value: \*\*shirt.GetFormattedTaxedPrice()\*\*

**string taxedPrice = shirt.GetFormattedTaxedPrice();**

8. Call the static \*\*AreEqual\*\* method of the \*\*Assert\*\* class, pass \*\*"$12.00"\*\* and \*\*taxedPrice\*\* as a parameters to the \*\*AreEqual\*\* method.

9. Save all the changes.

#### Task 3: Run the unit test – it should fail

1. 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 \*\*Shirt\*\* class, replace \*\*GetFormattedTaxedPrice\*\* return value by using the following information:

- Value: \*\*(Price \* Tax).ToString($"C2", CultureInfo.GetCultureInfo("en-US"))\*\*

2. Save all the changes.

#### Task 5: Run the unit test – it succeeds

1. Run all tests.

>\*\*Note\*\*: The \*\*Test Explorer\*\* displays one passed test: \*\*IsGetFormattedTaxedPriceReturnsCorrectly\*\*.

>\*\*Results\*\*: After completing this exercise, you will be able to create a test project, and test a model while fixing its code, as in a test-driven development environment.

### Exercise 2: Testing a Controller using a Fake Repository

#### Scenario

Después de crear un proyecto de prueba y probar el modelo, ahora debe probar el controlador.

Para hacer esto, necesitará crear una interfaz de repositorio como una dependencia para que el controlador obtenga acceso a los datos. Para probar el controlador, creará un repositorio falso como sustituto y luego se lo proporcionará al controlador a través de su constructor.

Las tareas principales para este ejercicio son las siguientes:

1. Crear un repositorio de interfaz

2. Implemente el repositorio de la interfaz utilizando un repositorio falso

3. Pase el repositorio falso al constructor de un controlador

4. Escribe una prueba para un controlador

5. Ejecute la prueba de la unidad: debería fallar

6. Implemente la clase de controlador para que la prueba pase

7. Ejecute la prueba de la unidad: tiene éxito

#### Task 1: Create an interface repository

1. Create a new folder with the following information:

- Folder name: \*\*Services\*\*

- Project: \*\*ShirtStoreWebsite\*\*

2. Create a new interface with the following information:

- Folder: \*\*Services\*\*

- Name: \*\*IShirtRepository\*\*

- Scope: \*\*public\*\*

3. In the \*\*IShirtRepository\*\* class, add a \*\*USING\*\* statement for the following namespace:

- \*\*ShirtStoreWebsite.Models\*\*

4. Declare a method with the following information:

- Return type: \*\*IEnumerable&lt;Shirt&gt;\*\*

- Name: \*\*GetShirts\*\*

5. Declare a method with the following information:

- Return type: \*\*bool\*\*

- Name: \*\*AddShirt\*\*

- Parameter:

- Type: \*\*Shirt\*\*

- Name: \*\*shirt\*\*

6. Declare a method with the following information:

- Return type: \*\*bool\*\*

- Name: \*\*RemoveShirt\*\*

- Parameter:

- Type: \*\*int\*\*

- Name: \*\*id\*\*

public interface IShirtRepository

{

IEnumerable<Shirt> GetShirts();

bool AddShirt(Shirt shirt);

bool RemoveShirt(int id);

}

#### Task 2: Implement the interface repository by using a fake repository

1. Create a new folder with the following information:

- Folder name: \*\*FakeRepositories\*\*

- Project: \*\*ShirtStoreWebsite.Tests\*\*

2. Create a new class with the following information:

- Folder: \*\*FakeRepositories\*\*

- Name: \*\*FakeShirtRepository\*\*

- Scope: \*\*internal\*\*

3. In the \*\*FakeShirtRepository\*\* class, add the \*\*USING\*\* statements for the following namespaces:

- \*\*ShirtStoreWebsite.Services\*\*

- \*\*ShirtStoreWebsite.Models\*\*

4. To implement the \*\*IShirtRepository\*\* interface, modify the \*\*FakeShirtRepository\*\* class.

5. Add a method with the following information:

- Scope: \*\*public\*\*

- Return Type: \*\*IEnumerable&lt;Shirt&gt;\*\*

- Name: \*\*GetShirts\*\*

**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 }**

**};**

**}**

6. In the \*\*GetShirts\*\* method, return the \*\*IEnumerable&lt;Shirt&gt;\*\* result by using the following information:

- Value:<br/>

\*\*new List&lt;Shirt&gt;()<br>

{<br>

new Shirt { Color = ShirtColor.Black, Size = ShirtSize.S, Price = 11F },<br>

new Shirt { Color = ShirtColor.Gray, Size = ShirtSize.M, Price = 12F },<br>

new Shirt { Color = ShirtColor.White, Size = ShirtSize.L, Price = 13F }<br>

}\*\*

7. Add a method with the following information:

- Scope: \*\*public\*\*

- Return Type: \*\*bool\*\*

- Name: \*\*AddShirt\*\*

- Parameter:

- Type: \*\*Shirt\*\*

- Name: \*\*shirt\*\*

**public bool AddShirt(Shirt shirt)**

**{**

**return true;**

**}**

8. In the \*\*AddShirt\*\* method, return \*\*TRUE\*\*.

9. Add a method with the following information:

- Scope: \*\*public\*\*

- Return Type: \*\*bool\*\*

- Name: \*\*RemoveShirt\*\*

- Parameter:

- Type: \*\*int\*\*

- Name: \*\*id\*\*

10. In the \*\*RemoveShirt\*\* method, return \*\*TRUE\*\*.

**public bool RemoveShirt(int id)**

**{**

**return true;**

**}**

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

1. In the \*\*ShirtController\*\* class, add a \*\*USING\*\* statement for the following namespace:

- \*\*ShirtStoreWebsite.Services\*\*

2. In the \*\*ShirtController\*\* class, create a new field with the following information:

- Scope: \*\*private\*\*

- Type: \*\*IShirtRepository\*\*

- Name: \*\*\_repository\*\*

private IShirtRepository \_repository;

3. Add a constructor with the following parameters:

- Parameter:

- Type: \*\*IShirtRepository\*\*

- Name: \*\*repository\*\*

4. In the \*\*ShirtController\*\* constructor, initialize the \*\*\_repository\*\* field with the value of the \*\*repository\*\* parameter.

public ShirtController(IShirtRepository repository)

{

\_repository = repository;

}

#### Task 4: Write a test for a controller

1. Instalar Microsoft.AspNetCore.Mvc 2.1.1 en ShirtStoreWebsite.Tests\*\*.

3. Create a new class with the following information:

- Project: \*\*ShirtStoreWebsite.Tests\*\*

- Name: \*\*ShirtControllerTest\*\*

- Scope: \*\*public\*\*

3. In the \*\*ShirtControllerTest\*\* class, add the \*\*USING\*\* statements for the following namespaces:

Microsoft.VisualStudio.TestTools.UnitTesting

Microsoft.AspNetCore.Mvc

ShirtStoreWebsite.Controllers

ShirtStoreWebsite.Models

ShirtStoreWebsite.Services

ShirtStoreWebsite.Tests.FakeRepositories

using ShirtStoreWebsite.Controllers;

using ShirtStoreWebsite.Models;

using ShirtStoreWebsite.Services;

using ShirtStoreWebsite.Tests.FakeRepositories;

4. Above the \*\*ShirtControllerTest\*\* class, add a \*\*TestClass\*\* attribute.

5. Add a method with the following information:

- Scope: \*\*public\*\*

- Return Type: \*\*void\*\*

- Name: \*\*IndexModelShouldContainAllShirts\*\*

6. Above the \*\*IndexModelShouldContainAllShirts\*\* method, add a decorador \*\*TestMethod\*\* attribute.

7. Create a variable named \*\*fakeShirtRepository\*\* of type \*\*IShirtRepository\*\*.

8. Initialize the \*\*fakeShirtRepository\*\* variable by using the \*\*fakeShirtRepository\*\* constructor.

9. Create a variable named \*\*shirtController\*\* of type \*\*ShirtController\*\*.

10. Initialize the \*\*shirtController\*\* variable by using the \*\*shirtController\*\* constructor, and then pass the following parameter:

- \*\*fakeShirtRepository\*\*

11. Create a variable named \*\*viewResult\*\* of type \*\*ViewResult\*\*.

12. Initialize the \*\*viewResult\*\* variable by using the \*\*Index\*\* method of the \*\*shirtController\*\* variable, and then store the return value as a \*\*ViewResult\*\* type by using the \*\*as\*\* operator.

13. Create a variable named \*\*shirts\*\* of type \*\*List&lt;Shirt&gt;\*\*.

14. Initialize the \*\*shirts\*\* variable by using the \*\*Model\*\* property of the \*\*viewResult\*\* variable, and then store the value as \*\*List&lt;Shirt&gt;\*\* type by using the \*\*as\*\* operator.

15. Call the static \*\*AreEqual\*\* method of the \*\*Assert\*\* class, and then pass the \*\*Count\*\* property of the \*\*shirts\*\* variable and the integer \*\*3\*\*.

**[TestMethod]**

**public void IndexModelShouldContainAllShirts()**

**{**

**IShirtRepository fakeShirtRepository = new FakeShirtRepository();**

**Mock<ILogger<ShirtController>> mockLogger = new Mock<ILogger<ShirtController>>();**

**ShirtController shirtController = new ShirtController(fakeShirtRepository, mockLogger.Object);**

**ViewResult viewResult = shirtController.Index() as ViewResult;**

**List<Shirt> shirts = viewResult.Model as List<Shirt>;**

**Assert.AreEqual(shirts.Count, 3);**

**}**

16. Save all the changes.

#### Task 5: Run the unit test – it should fail

1. Run all tests.

>\*\*Note\*\*: The \*\*Test Explorer\*\* displays 1 failed test: \*\*IndexModelShouldContainAllShirts\*\*, and 1 passed test: \*\*IsGetFormattedTaxedPriceReturnsCorrectly\*\*.

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

1. In the \*\*ShirtController\*\* class, in the \*\*Index\*\* action, remove its contents.

2. Add a variable named \*\*shirts\*\* of type \*\*IEnumerable&lt;Shirt&gt;\*\*, and assign it the value of the \*\*GetShirts\*\* method of the \*\*\_repository\*\* field.

3. Return the \*\*ViewResult\*\* result by using the \*\*View\*\* method.

4. Pass the \*\*shirts\*\* variable as a parameter to the \*\*View\*\* method.

5. In the \*\*AddShirt\*\* action, at the start of the method's scope, call the \*\*AddShirt\*\* of the \*\*\_repository\*\* field.

6. Pass the \*\*shirt\*\* as a parameter to the \*\*AddShirt\*\* method.

7. In the \*\*Delete\*\* action, at the start of the method's scope, call the \*\*RemoveShirt\*\* of the \*\*\_repository\*\* field.

8. Pass the \*\*id\*\* as a parameter to the \*\*RemoveShirt\*\* method.

9. Save all the changes.

#### Task 7: Run the unit test – it succeeds

1. Run all tests.

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

>\*\*Results\*\*: After completing this exercise, you will be able to test a controller using a fake repository.

### Exercise 3: Implementing a Repository in the MVC Project

#### Scenario

After the model and the controller have been tested, you have been asked to add a repository class so that the application will be able to run. In this exercise, you will implement a \*\*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:

1. Implement the interface repository in a repository class

2. Register the repository as a service

3. Run the MVC application

#### Task 1: Implement the interface repository in a repository class

1. Create a new class with the following information:

- Folder: \*\*Services\*\*

- Name: \*\*ShirtRepository\*\*

- Scope: \*\*public\*\*

2. In the \*\*ShirtRepository\*\* class, add the \*\*USING\*\* statements for the following namespaces:

- \*\*ShirtStoreWebsite.Models\*\*

- \*\*ShirtStoreWebsite.Data\*\*

3. Modify the \*\*FakeShirtRepository\*\* class to implement the \*\*IShirtRepository\*\* interface.

4. In the \*\*ShirtRepository\*\* class, create a new field with the following information:

- Scope: \*\*private\*\*

- Type: \*\*ShirtContext\*\*

- Name: \*\*\_context\*\*

5. Add a constructor with the following parameters:

- Parameter:

- Type: \*\*ShirtContext\*\*

- Name: \*\*context\*\*

6. In the \*\*ShirtRepository\*\* constructor, initialize the \*\*\_context\*\* field with the value of the \*\*context\*\* parameter.

7. Add a method with the following information:

- Scope: \*\*public\*\*

- Return Type: \*\*IEnumerable&lt;Shirt&gt;\*\*

- Name: \*\*GetShirts\*\*

8. In the \*\*GetShirts\*\* method, return the \*\*IEnumerable&lt;Shirt&gt;\*\* result by using the \*\*\_context.Shirts.ToList();\*\* method.

9. Add a method with the following information:

- Scope: \*\*public\*\*

- Return Type: \*\*bool\*\*

- Name: \*\*AddShirt\*\*

- Parameter:

- Type: \*\*Shirt\*\*

- Name: \*\*shirt\*\*

10. In the \*\*AddShirt\*\* method, call the \*\*Add\*\* method of the \*\*\_context\*\* field.

11. Pass \*\*shirt\*\* as a parameter to the \*\*Add\*\* method.

12. Add a variable named \*\*entries\*\* of type \*\*int\*\*.

13. In the \*\*AddShirt\*\* method, initialize the \*\*entries\*\* variable by using the \*\*SaveChanges\*\* method of the \*\*\_context\*\* field.

14. Create an \*\*IF\*\* statement that checks that the value of \*\*entries\*\* is greater then \*\*0\*\*.

15. Inside the \*\*IF\*\* statement code block, return \*\*TRUE\*\*.

16. After the \*\*IF\*\* statement code block, add an \*\*ELSE\*\* statement.

17. Inside the \*\*ELSE\*\* statement code block, return \*\*FALSE\*\*.

18. Add a method with the following information:

- Scope: \*\*public\*\*

- Return Type: \*\*bool\*\*

- Name: \*\*RemoveShirt\*\*

- Parameter:

- Type: \*\*int\*\*

- Name: \*\*id\*\*

19. In the \*\*RemoveShirt\*\* method, add a variable named \*\*shirt\*\* of type \*\*var\*\*.

20. Initialize the \*\*shirt\*\* variable by using the \*\*\_context.Shirts.SingleOrDefault(m => m.Id == id)\*\* method.

21. Call the \*\*Remove\*\* method of the \*\*\_context.Shirts\*\* property.

22. Pass \*\*shirt\*\* as a parameter to the \*\*Remove\*\* method.

23. Add a variable named \*\*entries\*\* of type \*\*int\*\*.

24. Initialize the \*\*entries\*\* variable by using the \*\*SaveChanges\*\* method of the \*\*\_context\*\* field.

25. Create an \*\*IF\*\* statement that checks that the value of \*\*entries\*\* is greater then \*\*0\*\*.

26. Inside the \*\*IF\*\* statement code block, return \*\*TRUE\*\*.

27. After the \*\*IF\*\* statement code block, add an \*\*ELSE\*\* statement.

28. Inside the \*\*ELSE\*\* statement code block, return \*\*FALSE\*\*.

#### Task 2: Register the repository as a service

1. In the \*\*Startup\*\* class, add a \*\*USING\*\* statement for the following namespace:

- \*\*ShirtStoreWebsite.Services\*\*

2. In the \*\*ConfigureServices\*\* method, call the \*\*AddScoped\*\* method of \*\*services\*\* parameter with the following information:

- Interface: \*\*IShirtRepository\*\*

- Implementation: \*\*ShirtRepository\*\*

3. Save all the changes.

#### Task 3: Run the MVC application

1. Start the application without debugging.

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

2. Close Microsoft Edge.

>\*\*Results\*\*: After completing this exercise, you have developed a repository to have 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:

1. Add exception handling in Startup.cs

2. Create a temporary exception for testing

3. Run the application in the development environment

4. Run the application in the production environment

5. Remove the temporary exception

#### Task 1: Add exception handling in Startup.cs

1. In the \*\*Startup\*\* class, at the beginning of the \*\*Configure\*\* method code block, create an \*\*IF\*\* statement that checks the return value of the \*\*env.IsDevelopment\*\* method.

2. Inside the \*\*IF\*\* statement code block, call the \*\*UseDeveloperExceptionPage\*\* method of the \*\*app\*\* parameter.

3. After the \*\*IF\*\* statement code block, add an \*\*ELSE\*\* statement.

4. Inside the \*\*ELSE\*\* statement code block, call the \*\*UseExceptionHandler\*\* method of the \*\*app\*\* parameter.

5. Pass \*\*"/error.html"\*\* as a parameter to the \*\*UseExceptionHandler\*\* method.

#### Task 2: Create a temporary exception for testing

1. In the \*\*ShirtController\*\* class, in the \*\*Delete\*\* method, change the parameter passed to the \*\*\_repository.RemoveShirt\*\* method to \*\*-1\*\*.

2. Save all the changes.

#### Task 3: Run the application in the development environment

1. Select the development environment, and then start the application without debugging.

2. In Microsoft Edge, click the \*\*Delete\*\* link of the first shirt.

>\*\*Note\*\*: The browser displays the detailed exception page.

3. Close Microsoft Edge.

#### Task 4: Run the application in the production environment

1. Select the production environment, and then start the application without debugging.

2. In Microsoft Edge, click the \*\*Delete\*\* link of the first shirt.

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

3. Close Microsoft Edge.

#### Task 5: Run the application in the production environment

1. In the \*\*ShirtController\*\* class, in the \*\*Delete\*\* method, change the parameter passed to the \*\*\_repository.RemoveShirt\*\* method to the \*\*id\*\* parameter.

2. Save all the changes.

>\*\*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 MVC 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:

1. Add logging to the MVC application

2. Test the controller by using a mocking framework

3. Run the unit test

4. Run the application in the development environment

5. Run the application in the production environment

#### Task 1: Add logging to the MVC application

1. In the \*\*ShirtStoreWebsite\*\* project, create a new top-level \*\*App Setting File\*\* with the following information:

- File Name: \*\*appsettings.development.json\*\*

- Folder: \*\*/\*\*

2. Delete the contents of the \*\*appsettings.development.json\*\* file.

3. In the \*\*appsettings.development.json\*\* file, add new values by using the following information:

- Property: \*\*"Logging"\*\*

- Type: \*\*object\*\*

- Value:

- Property: \*\*"LogLevel"\*\*

- Type: \*\*object\*\*

- Value:

- Property: \*\*"Default"\*\*

- Type: \*\*string\*\*

- Value: \*\*"Trace"\*\*

4. In the \*\*ShirtStoreWebsite\*\* project, create a new top-level \*\*App Setting File\*\* with the following information:

- File Name: \*\*appsettings.production.json\*\*

- Folder: \*\*/\*\*

5. Delete the content of the \*\*appsettings.production.json\*\* file.

6. In the \*\*appsettings.production.json\*\* file, add new values by using the following information:

- Property: \*\*"Logging"\*\*

- Type: \*\*string\*\*

- Value:

- Property 1:

- Name: \*\*"PathFormat"\*\*

- Type: \*\*string\*\*

- Value: \*\*"shirt\_store\_logs.txt"\*\*

- Property 2:

- Name: \*\*"LogLevel"\*\*

- Type: \*\*object\*\*

- Value:

- Property: \*\*"Default"\*\*

- Type: \*\*string\*\*

- Value: \*\*"Warning"\*\*

7. In the \*\*Program\*\* class, chain the \*\*ConfigureLogging\*\* method after the \*\*CreateDefaultBuilder\*\* method.

8. Pass a lambda expression as a parameter to the \*\*ConfigureLogging\*\* method with the following information:

- Lambda Expression: \*\*(hostingContext, logging) => { }\*\*

9. In the lambda expression, add a variable named \*\*env\*\* of type \*\*var\*\*.

10. Initialize the \*\*env\*\* variable by using the \*\*HostingEnvironment\*\* property of the \*\*hostingContext\*\* parameter.

11. In the lambda expression, add a variable named \*\*config\*\* of type \*\*var\*\*.

12. Initialize the \*\*config\*\* variable by using the \*\*GetSection\*\* method of the \*\*hostingContext.Configuration\*\* property.

13. Pass \*\*"Logging"\*\* as a parameter to the \*\*GetSection\*\* method.

14. Call the \*\*ClearProviders\*\* method of the \*\*logging\*\* parameter.

15. Create an \*\*IF\*\* statement that checks the return value of the \*\*IsDevelopment\*\* method of the \*\*env\*\* parameter.

16. Inside the \*\*IF\*\* statement code block, call the \*\*AddConfiguration\*\* method of the \*\*logging\*\* parameter.

17. Pass the \*\*config\*\* variable to the \*\*AddConfiguration\*\* method.

18. Inside the \*\*IF\*\* statement code block, call the \*\*AddConsole\*\* method of the \*\*logging\*\* parameter.

19. After the \*\*IF\*\* statement code block, add an \*\*ELSE\*\* statement.

20. Inside the \*\*ELSE\*\* statement code block, call the \*\*AddFile\*\* method of the \*\*logging\*\* parameter.

21. Pass the \*\*config\*\* variable to the \*\*AddFile\*\* method.

22. In the \*\*ShirtController\*\* class, add a \*\*USING\*\* statement for the following namespace:

- \*\*Microsoft.Extensions.Logging\*\*

23. Create a new field with the following information:

- Scope: \*\*private\*\*

- Type: \*\*ILogger\*\*

- Name: \*\*\_logger\*\*

24. In the constructor, accept an additional parameter by using the following information:

- Type: \*\*ILogger&lt;ShirtController&gt;\*\*

- Name: \*\*logger\*\*

25. In the \*\*ShirtController\*\* constructor, initialize the \*\*\_logger\*\* field with the value of the \*\*logger\*\* parameter.

26. In the \*\*AddShirt\*\* method, after calling the \*\*AddShirt\*\* method of the \*\*\_repository\*\* variable, call the \*\*LogDebug\*\* method of the \*\*\_logger\*\* variable.

27. Pass \*\*$"A {shirt.Color.ToString()} shirt of size {shirt.Size.ToString()} with a price of {shirt.GetFormattedTaxedPrice()} was added successfully."\*\* as a parameter to the \*\*LogDebug\*\* method.

28. Delete the contents of the \*\*Delete\*\* method.

29. In the \*\*Delete\*\* method, create a \*\*TRY\*\* statement.

30. Inside the \*\*TRY\*\* statement code block, call the \*\*RemoveShirt\*\* method of the \*\*\_repository\*\* parameter.

31. Pass the \*\*id\*\* parameter to the \*\*RemoveShirt\*\* method.

32. Inside the \*\*TRY\*\* statement code block, call the \*\*LogDebug\*\* method of the \*\*\_logger\*\* parameter.

33. Pass \*\*$"A shirt with id {id} was removed successfully."\*\* as a parameter to the \*\*LogDebug\*\* method.

34. Inside the \*\*TRY\*\* statement code block, return \*\*RedirectToActionResult\*\* by using the \*\*RedirectToAction\*\* method.

35. Pass \*\*"Index"\*\* as a parameter to the \*\*RedirectToAction\*\* method.

36. After the \*\*TRY\*\* statement code block, add a \*\*CATCH\*\* statement with the following information:

- Parameter:

- Type: \*\*Exception\*\*

- Name: \*\*ex\*\*

37. Inside the \*\*CATCH\*\* statement code block, call the \*\*LogError\*\* method of the \*\*\_logger\*\* parameter.

38. Pass the \*\*ex\*\* parameter and \*\*$"An error occured while trying to delete shirt with id of {id}."\*\* as a parameters to the \*\*LogError\*\* method.

39. Inside the \*\*CATCH\*\* statement code block, throw the \*\*ex\*\* parameter.

40. Save all the changes.

#### Task 2: Test the controller by using a mocking framework

1. Open \*\*Package Manager Console\*\*.

2. In the \*\*Package Manager Console\*\* tab, type the following command:

- \*\*Install-Package Moq -Version 4.9.0 -ProjectName ShirtStoreWebsite.Tests\*\*.

3. In the \*\*ShirtControllerTest\*\* class, add the \*\*USING\*\* statements for the following namespaces:

- \*\*Microsoft.Extensions.Logging\*\*

- \*\*moq\*\*

4. In the \*\*IndexModelShouldContainAllShirts\*\* method, after initializing the \*\*fakeShirtRepository\*\* variable, add a new variable with the following information:

- Type: \*\*Mock&lt;ILogger&lt;ShirtController&gt;&gt;\*\*

- Name: \*\*mockLogger\*\*

- Value: \*\*new Mock&lt;ILogger&lt;ShirtController&gt;&gt;()\*\*

5. In the initialization of the \*\*shirtController\*\* variable, pass \*\*mockLogger.Object\*\* as a second parameter to its constructor.

6. Save all the changes.

#### Task 3: Run the unit test

1. Run all tests.

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

#### Task 4: Run the application in the development environment

1. Select the development environment, and then start the application without debugging.

2. In the \*\*Output\*\* tab, clear all text.

3. In Microsoft Edge, add a new \*\*Shirt\*\* to the stock by using the following information:

- Size: \*\*M\*\*

- Color: \*\*Yellow\*\*

- Price: \*\*10\*\*

- Tax: \*\*1.2\*\*

4. In the \*\*Output\*\* tab, locate the following text:

- \*\*A Yellow shirt of size M with a price of $12.00 was added successfully.\*\*

5. In the \*\*Output\*\* tab, clear all text.

6. In Microsoft Edge, delete the first shirt in stock.

7. In the \*\*Output\*\* tab, locate the following text:

- \*\*A shirt with id 1 was removed successfully.\*\*

8. In Microsoft Edge, access the following relative path:

- Path: \*\*http://localhost:[port]/Shirt/Delete/-1\*\*

>\*\*Note\*\*: The browser displays the \*\*DeveloperException\*\* page.

9. Close Microsoft Edge.

#### Task 5: Run the application in the production environment

1. Select the production environment, and then start the application without debugging.

2. In Microsoft Edge, access the following relative path:

- Path: \*\*http://localhost:[port]/Shirt/Delete/-1\*\*

3. Close Microsoft Edge.

4. In File Explorer, navigate to \*\*[Repository Root]\Allfiles\Mod10\Labfiles\01\_ShirtStore\_begin\ShirtStoreWebsite\*\*, and open the \*\*shirt\_store\_logs-XXXXXXXX.txt\*\* file.

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

5. Close \*\*shirt\_store\_logs-XXXXXXXX - Notepad\*\*.

6. Select the \*\*production\*\* environment, and then start the application without debugging.

7. In Microsoft Edge, add a new \*\*Shirt\*\* to the stock by using the following information:

- Size: \*\*M\*\*

- Color: \*\*Yellow\*\*

- Price: \*\*10\*\*

- Tax: \*\*1.2\*\*

8. In File Explorer, navigate to \*\*[Repository Root]\Allfiles\Mod10\Labfiles\01\_ShirtStore\_begin\ShirtStoreWebsite\*\*, and then open the \*\*shirt\_store\_logs-XXXXXXXX.txt\*\* file.

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

9. Close \*\*shirt\_store\_logs-XXXXXXXX - Notepad\*\*.

10. Close Microsoft Visual Studio.