* 1. 

Hands-On Lab

ASP.NET MVC 3 Testing

Lab version: 1.1.0

Last updated: 1/10/2011

Contents

[Overview 3](#_Toc282180591)

[Exercise 1: Testing simple controller actions 5](#_Toc282180592)

[Task 1 – Creating a Testing Project 5](#_Toc282180593)

[Task 2 – Running the tests 8](#_Toc282180594)

[Task 3 – Implementing the test for the Index action method 11](#_Toc282180595)

[Task 4 – Implementing the test for the Browse action method 12](#_Toc282180596)

[Task 5 – Implementing the test for the Details action method 13](#_Toc282180597)

[Exercise 2: Testing CRUD actions 14](#_Toc282180598)

[Task 1 – Reviewing the provided solution 14](#_Toc282180599)

[Task 2 – Adding transaction capabilities to the Test Methods 15](#_Toc282180600)

[Task 2 – Testing Create Action Method 18](#_Toc282180601)

[Task 3 – Testing Delete Action Method 19](#_Toc282180602)

[Task 4 – Testing Edit Action Method 21](#_Toc282180603)

[Task 5 – Testing Index Action Method 22](#_Toc282180604)

[Task 6 – Testing Details Action Method 23](#_Toc282180605)

[Task 7 – Testing Browse Action Method for StoreController 24](#_Toc282180606)

[Exercise 3: Testing Cart Actions 26](#_Toc282180607)

[Task 1 – Running ShoppingCartController Tests 26](#_Toc282180608)

[Task 2 – Implementing the test for the AddToCart action method 27](#_Toc282180609)

[Task 3 – Implementing the test for the Index action method 29](#_Toc282180610)

[Task 4 – Implementing the test for the RemoveFromCart action method 31](#_Toc282180611)

[Summary 33](#_Toc282180612)

Overview

* 1. **Note:** This Hands-on Lab assumes you have basic knowledge of ASP.NET MVC. If you have not used ASP.NET MVC before, we recommend you to go over ASP.NET MVC Fundamentals Hand-on Lab.

Testing an application is an automated way of verifying it is always behaving the way it is supposed to. That way, you can later make changes to the code and refactor components without breaking something else somewhere in the application.

This Lab will take you through the process of testing controllers in an ASP.NET MVC application, used to verify that actions are working properly.

Unit testing is a particular method where each test isolates and verifies a single part of the application, in an isolated way, with the possibility of automating tests. A unit test is the smallest testable part of an application. In procedural programming may be a procedure, in object oriented programming, a method.

Unit tests are created by programmers, and their objective is to isolate each part of a program and verify that individual parts are correct. Typically in unit testing, substitutes like method stubs, mock objects, fakes and test harnesses can be used to guarantee that isolation.

In this Hands-on Lab, each test is independent from the others, but instead of stubs or mocks, you will be testing the controllers against a well-known database. If needed, that database will be protected by transactions and its initial state will be copied for each test execution, making sure that each test runs with a controlled data set.

You will first focus on testing simple controller actions, then CRUD actions and finally the shopping cart actions implemented in the ASP.NET MVC Membership, Authorization & Store Checkout Hands-on Lab.

* 1. **Disclaimer:** Test Driven Development (TDD) is a software development process where failing unit tests that define an improvement/function are written before the actual code that implements that behavior. The test passes after the implementation is done according to the test specifications. Finally, the code is refactored to match standards, improve efficiency, etc.
  2. Since you will be testing the MVC Music Store code built throughout the past Hands-on labs, you will not be strictly practicing TDD. Else, you will create tests to verify the behavior of already-written code.

# Objectives

* 1. In this Hands-On Lab, you will learn how to:
  + Test simple controller actions
  + Test actions that interact with a database
  + Test actions that use the user’s session

# System Requirements

* 1. You must have the following items to complete this lab:
  + ASP.NET and ASP.NET MVC 3
  + Visual Studio 2010 Professional or superior
  + SQL Server Database (Express edition or above)
  1. **Note:** You can install the previous system requirements by using the Web Platform Installer 3.0: <http://go.microsoft.com/fwlink/?LinkID=194638>.

# Setup

* 1. For convenience, much of the code used in this hands-on lab is available as Visual Studio code snippets. To check the prerequisites of the lab and install the code snippets:
  2. Open a Windows Explorer window and browse to the lab’s **Source\Setup** folder.
  3. Double-click the **Dependencies.dep** file in this folder to launch the Dependency Checker tool and install any missing prerequisites and the Visual Studio code snippets.
  4. If the User Account Control dialog is shown, confirm the action to proceed.
  5. **Note:** This process may require elevation. The *.dep* extension is associated with the Dependency Checker tool during its installation. For additional information about the setup procedure and how to install the Dependency Checker tool, refer to the **Setup.docx** document in the **Assets** folder of the training kit.

# Exercises

* 1. This Hands-On Lab is comprised by the following exercises:
  2. Exercise 1: Testing simple controller actions
  3. Exercise 2: Testing CRUD actions
  4. Exercise 3: Testing cart actions
  5. Estimated time to complete this lab: **35 minutes**.
  6. **Note:** Each exercise is accompanied by an **End** folder containing the resulting solution you should obtain after completing the exercises. You can use this solution as a guide if you need additional help working through the exercises.

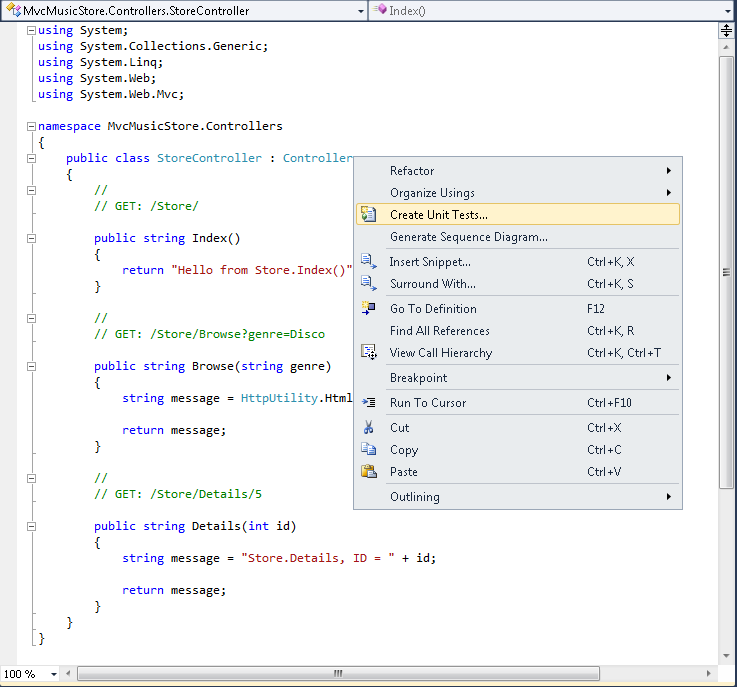
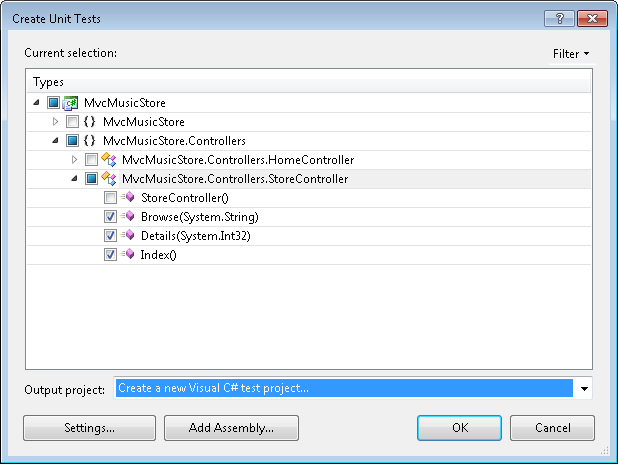
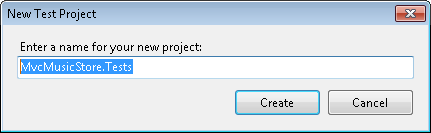
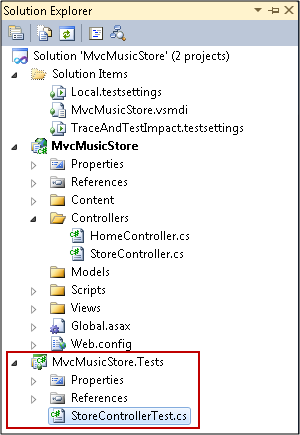
# Next Step

Exercise 1: Testing simple controller actions

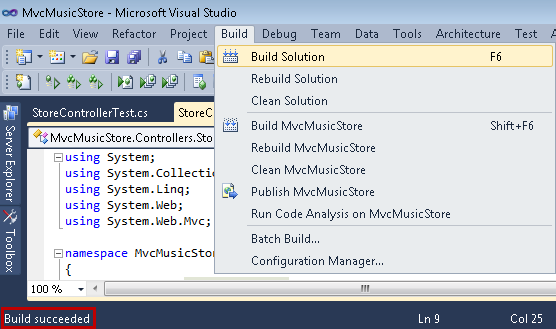
Exercise 1: Testing simple controller actions

* 1. In this exercise, you will learn how to test your controllers by adding testing methods to verify the behavior of its method actions.

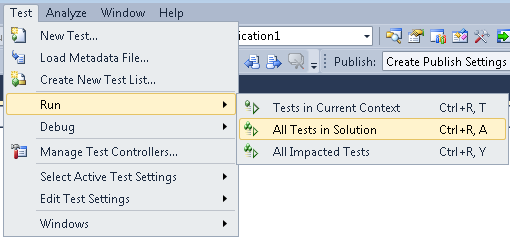
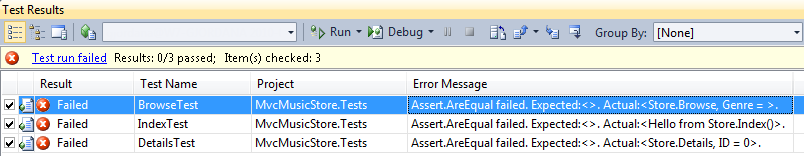
Task 1 – Creating a Testing Project

* 1. In this task you will create a testing project for your tests.
  2. Start Microsoft Visual Studio 2010 Professional from **Start** | **All Programs** | **Microsoft Visual Studio 2010** | **Microsoft Visual Studio 2010**.
  3. In the **File** menu, choose Open **Project**. In the Open Project dialog, browse to Source\Ex01-Testing simple controller actions\Begin, select **MvcMusicStore.sln** and click **Open**.
  4. In the **Solution Explorer** expand the **Controllers** folder and double-click the **StoreController.cs** class to open that file.
  5. Right-click at any point of the **StoreController** class code and then select **Create Unit Tests**.
     1. 
     2. Figure 1
     3. Creating Unit Tests
  6. A window appears with a tree to select the methods to test. Uncheck **StoreController()** because you will not add any test to the class constructor, and verify that **Browse(System.String)**, **Details(System.Int32)** and **Index()** methods are selected. Verify that the option **Create a new Visual C# test project is selected** in the **Output project** select box. Finally, click **OK.**
     1. 
     2. Figure 2
     3. Selecting the tests to create
     4. **Note:** This process creates a project, adds a reference to the project to be tested and creates skeleton methods that you will complete to test your controller actions.
  7. In the New Test Project window change the project name to **MvcMusicStore.Tests** and click **Create.**
     1. 
     2. Figure 3
     3. Creating a New Test Project
  8. A new project is created.
     1. 
     2. Figure 4
     3. Solution Explorer with the new Test Project

Task 2 – Running the tests

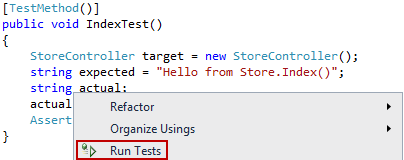
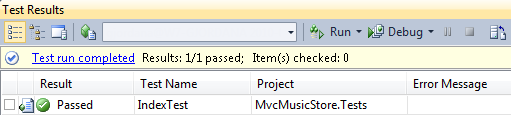
* 1. In this task you will run the tests to verify the system’s behavior. These tests were marked as Inconclusive: an **Assert.Inconclusive** method is invocated, raising an error. This is a way to verify that your tests did not pass because they were not implemented.
  2. Build the Solution to verify everything is in order. To do this, In the **Build** Menu, select **Build Solution**. The Solution should build succesfully.
     1. 
     2. Figure 5
     3. Building the project
  3. The tests were created in a default ASP.NET environment. In order to test the controllers but without the need to run the web application, you must delete the following attributes before each of the 3 test methods.
     1. C#
     2. [TestMethod()]
     3. ~~[HostType("ASP.NET")]~~
     4. ~~[AspNetDevelopmentServerHost("D:\\ Source\\Ex01-Testing simple controller actions\\Begin\\MvcMusicStore", "/")]~~
     5. ~~[UrlToTest("http://localhost:1045/")]~~
     6. **Note:** These attributes avoid you to configure these tests to run through ASP.NET. Unit tests typically run under the VSTest host process. ASP.NET unit tests must run under the ASP.NET host process. This is configured by the **HostType** attribute. Additionally, you should specify the path to the web application by the **AspNetDevelopmentServerHost** attribute. Finally, you have to indicate Visual Studio the URL to test by defining the **UrlToTest** attribute.
     7. In your scenario, you will test your controller action’s methods and you don’t need to configure this.
     8. Tests should look like the following:
     9. C#
     10. **[TestMethod()]**
     11. **public void IndexTest()**
     12. C#
     13. **[TestMethod()]**

**public void DetailsTest()**

* + 1. C#
    2. **[TestMethod()]**
    3. **public void BrowseTest()**
  1. Run the tests. To do this, in **Test** Menu select **Run**, and then **All Tests in Solution**.
     1. 
     2. Figure 6
     3. Running the tests
  2. Verify that all tests fail in the **Test Results** window.
     1. 
     2. Figure 7
     3. Tests Results

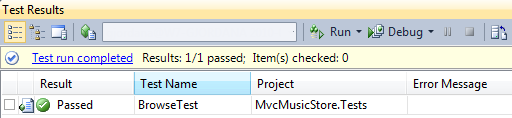
Task 3 – Implementing the test for the Index action method

Now that the tests fail, you will start making them pass one by one. In this task you will implement the test for the Index action method of the Store Controller. The Index method provided in the Begin solution only returns this text: "Hello from Store.Index()", hence that is the text that your test method should verify.

* 1. In the StoreControllerTest.cs file, replace the **IndexTest** method. Note that the **Assert.Inconclusive** method was removed with the real implementation for the test.
     1. (Code Snippet – ASP.NET MVC 3.0 Testing – Ex1 IndexTestMethod – CSharp)
     2. C#
     3. **/// <summary>**
     4. **///A test for Index**
     5. **///</summary>**
     6. **[TestMethod()]**
     7. **public void IndexTest()**
     8. **{**
     9. **StoreController target = new StoreController();**
     10. **string expected = "Hello from Store.Index()";**
     11. **string actual;**
     12. **actual = target.Index();**
     13. **Assert.AreEqual(expected, actual);**
     14. **}**
     15. Note: The **Assert.AreEqual** is a method that verifies that two objects are equal. This overload in particular compares two strings.
  2. Run exclusively this test to verify the results. To do this, right-click inside the **IndexTest** method and then select **Run Tests**.
     1. 
     2. Figure 8
     3. Running tests in Current Context
  3. The **Test Result** window should show that the test passes.
     1. 
     2. Figure 9
     3. Running test IndexTest

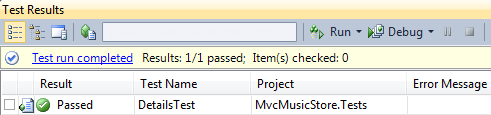
Task 4 – Implementing the test for the Browse action method

In this task you will implement the test for the **Browse** action method of the Store Controller. The Browse method provided in the Begin solution returns a text concatenating the genre received as parameter. So, your method should verify that text is returned.

* 1. In the StoreControllerTest.cs file, replace the **BrowseTest** method. Note that the **Assert.Inconclusive** method was removed with the real implementation for the test.
     1. (Code Snippet – ASP.NET MVC 3.0 Testing – Ex1 BrowseTestMethod – CSharp)
     2. C#
     3. **/// <summary>**
     4. **///A test for Browse**
     5. **///</summary>**
     6. **[TestMethod()]**
     7. **public void BrowseTest()**
     8. **{**
     9. **StoreController target = new StoreController();**
     10. **string genre = "Disco";**
     11. **string expected = "Store.Browse, Genre = Disco";**
     12. **string actual;**
     13. **actual = target.Browse(genre);**
     14. **Assert.AreEqual(expected, actual);**
     15. **}**
  2. Run exclusively this test to verify the results. To do this, right-click inside the **BrowseTest** method and then select **Run Tests**.
  3. The **Test Result** window should show that the test passes.
     1. 
     2. Figure 10
     3. Running test IndexTest

Task 5 – Implementing the test for the Details action method

In this task you will implement the test for the **Details** action method of the Store Controller. The Details method provided in the Begin solution returns a text concatenating the id received as a parameter. So, your method should verify that text is returned.

* 1. In the StoreControllerTest.cs file, replace the **DetailsTest** method. Note that the **Assert.Inconclusive** method was removed with the real implementation for the test.
     1. (Code Snippet – ASP.NET MVC 3.0 Testing – Ex1 DetailsTestMethod – CSharp)
     2. C#
     3. **/// <summary>**
     4. **///A test for Details**
     5. **///</summary>**
     6. **[TestMethod()]**
     7. **public void DetailsTest()**
     8. **{**
     9. **StoreController target = new StoreController();**
     10. **int id = 5;**
     11. **string expected = "Store.Details, ID = 5";**
     12. **string actual;**
     13. **actual = target.Details(id);**
     14. **Assert.AreEqual(expected, actual);**
     15. **}**
  2. Run exclusively this test to verify the results. To do this, right-click inside the **DetailsTest** method and then select **Run Tests**.
  3. The **Test Result** window should show that the test passes.
     1. 
     2. Figure 11
     3. Running test IndexTest

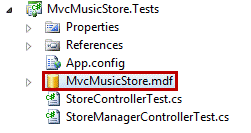
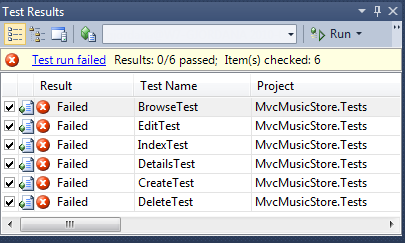
# Next Step

* 1. Exercise 2: Testing CRUD actions

Exercise 2: Testing CRUD actions

* 1. In this exercise, you will learn how to test CRUD actions that modify the database. One of the premises to take into account is that tests don’t necessarily run in a specific order. Another one is that every test could be executed as many times as needed. So your tests should not depend on previous tests executions. For instance, if your test is going to delete a record in the database, the setup of the test should provide the record to be deleted. This way, the test could be executed repeatedly.

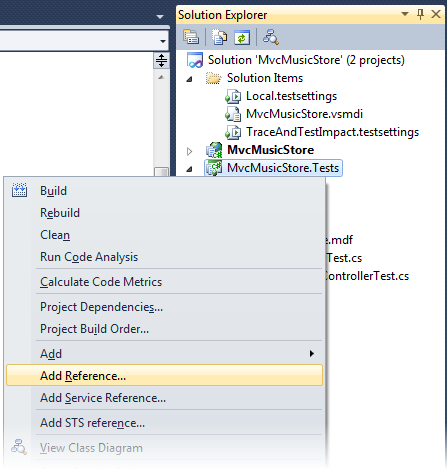
Task 1 – Reviewing the provided solution

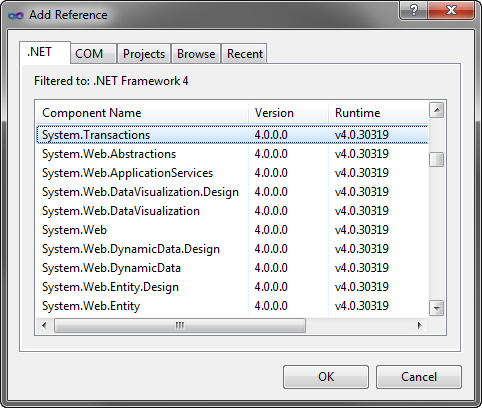
* 1. Some changes where performed to the end solution of the last exercise in order to prepare it for this one.
  + A test project is provided to hold the tests for this exercise
  + The CRUD tests were created using the approach followed in the previous exercise (right-click on the class to be tested and then **Create Unit Tests**)
  + The test project has its own database for testing purposes: since in this exercise you are testing actions that rely on a database, a well-known database for testing purposes was included inside the testing project. This database includes the tables used by the entity classes and has data pre-loaded to simplify test execution.
    1. 
    2. Figure 12
    3. Well-known database as part of the testing project
  + A web.config was added to the project to connect the solution to this database in order to avoid modifying data in the project to be tested.
  1. In this task, you will review those changes before moving on, in the following tasks, with the particular additions for testing CRUD actions.
  2. Start Microsoft Visual Web Developer 2010 Express from **Start** | **All Programs** | **Microsoft Visual Studio 2010** | **Microsoft Visual Studio 2010**.
  3. In the **File** menu, choose Open **Project**. In the Open Project dialog, browse to Source\Ex02-Testing CRUD actions\Begin, select **MvcMusicStore.sln** and click **Open**. Based on the changes explained, you should use this solution instead of the one you ended exercise 1 with.
  4. In the **Test** menu, select **Run** and then **All Tests in Solution**.
  5. The tests are executed and fail. In this exercise, you will modify these tests to make them succeed.
  6. 
  7. Figure 13
  8. Running all the tests

Task 2 – Adding transaction capabilities to the Test Methods

In this task, you will learn how to use transactions to limit the scope of the test. This way, you could execute the test as many times as you want, without altering the database content.

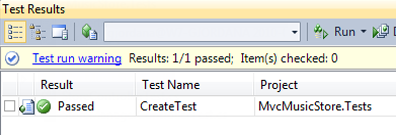
* 1. In the **Solution Explorer** right-click the project **MvcMusicStore.Tests** andselect **Add Reference**.



* + 1. Figure 14
    2. Adding a Reference
  1. In the **Add Reference** window select the **Net** tab and locate the **System.Transactions** component and click **OK.**
     1. 
     2. Figure 15
     3. Adding System.Transactions reference
     4. **Note:** The **System.Transactions** classes provide explicit and implicit transaction programming model. In the implicit model, when you use **TransactionScope** the transaction manager determines if it will subscribe to a current transaction or if it will instantiate a new one. This depends on two factors: whether an ambient transaction already exists and also the **TransactionScopeOption** parameter in the constructor.
     5. When you use **TransactionScope** in a **using** block, the whole block is transactional. This avoids you from having to focus on the transaction scope. If you were using a SQL Server database (not a local one, as now) and you wanted to commit the changes to the database, you would have toinvoke **TransactionScope**’s **Complete** method. Since the intention in this exercise is to maintain the database intact for the next test, you will not see that invocation in the code. Nevertheless you should remember to use it appropriately when your intention is to commit the changes of the transaction.
  2. Add the following **using** directives in the **StoreManagerControllerTest.cs** class. To do this, add the following references before the namespace definition:
     1. (Code Snippet – ASP.NET MVC 3.0 Testing – Ex2 Using directives – CSharp)
     2. C#
     3. using MvcMusicStore.Controllers;
     4. using Microsoft.VisualStudio.TestTools.UnitTesting;
     5. using System;
     6. using Microsoft.VisualStudio.TestTools.UnitTesting.Web;
     7. using MvcMusicStore.Models;
     8. using System.Web.Mvc;
     9. **using System.Transactions;**
     10. **using System.Linq;**
     11. **using MvcMusicStore.ViewModels;**
     12. namespace MvcMusicStore.Tests
     13. {

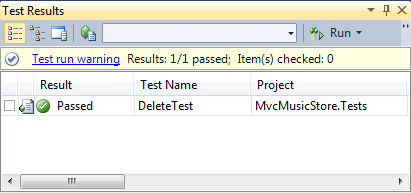
Task 2 – Testing Create Action Method

In this task, you will learn how to test the creation of new records in the database by adding new albums.

* 1. Open **StoreManagerControllerTest.cs** class and locate the **CreateTest** method.
  2. Test the addition of a new Album, then retrieve it from the database and compare its attributes to verify that it was correctly persisted. To do this, replace the **CreateTest** method with the following code.
     1. (Code Snippet – ASP.NET MVC 3.0 Testing – Ex2 CreateTestMethod – CSharp)
     2. C#
     3. [TestMethod()]
     4. [DeploymentItem("MvcMusicStore.mdf")]
     5. [DeploymentItem("MvcMusicStore\_log.ldf")]
     6. public void CreateTest()
     7. {
     8. **using (TransactionScope ts = new TransactionScope())**
     9. **{**
     10. **StoreManagerController target = new StoreManagerController();**
     11. **Album album = new Album()**
     12. **{**
     13. **GenreId = 1,**
     14. **ArtistId = 1,**
     15. **Title = "New Album",**
     16. **Price = 10,**
     17. **AlbumArtUrl = "/Content/Images/placeholder.gif"**
     18. **};**
     19. **ActionResult actual;**
     20. **actual = target.Create(album);**
     21. **Assert.IsTrue(album.AlbumId != 0);**
     22. **MusicStoreEntities storeDB = new MusicStoreEntities();**
     23. **var newAlbum = storeDB.Albums.SingleOrDefault(a => a.AlbumId == album.AlbumId);**
     24. **Assert.AreEqual(album.GenreId, newAlbum.GenreId);**
     25. **Assert.AreEqual(album.ArtistId, newAlbum.ArtistId);**
     26. **Assert.AreEqual(album.Title, newAlbum.Title);**
     27. **Assert.AreEqual(album.Price, newAlbum.Price);**
     28. **Assert.AreEqual(album.AlbumArtUrl, newAlbum.AlbumArtUrl);**
     29. **}**
     30. }
     31. **Note:** You will notice that before the test method two new **DeploymentItem** attributes were added. This two attributes refer to the database and its log file.
     32. The **DeploymentItem** attribute is used in test methods to alert to the test engine of files that will be needed to execute tests. Visual Studio run tests in a different folder where it copies the built assemblies. This folder is created for each test execution and when a test is run, code-under-test assemblies, and the deployment items are placed in a test deployment folder specific for that test run. Since these files are copied every time the test is executed, the database used for the test execution always starts with the same set of data.
  3. Run the test. On the **StoreManagerControllerTest.cs** class right-click the **CreateTest** method and select **Run Tests**
     1. 
     2. Figure 16
     3. Running the CreateTest test

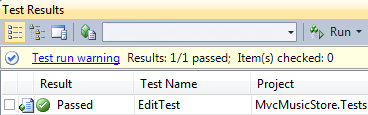
Task 3 – Testing Delete Action Method

In this task, you will learn how to test the deletion of albums.

* 1. Open **StoreManagerControllerTest.cs** class and locate the **DeleteTest** method.
  2. Test the deletion of an Album and then verify it no longer exists in the database. To do this, replace the **DeleteTest** method with the following code.
     1. (Code Snippet – ASP.NET MVC 3.0 Testing – Ex2 DeleteTestMethod – CSharp)
     2. C#
     3. /// <summary>
     4. /// A test for Delete
     5. ///</summary>
     6. [TestMethod()]
     7. [DeploymentItem("MvcMusicStore.mdf")]
     8. [DeploymentItem("MvcMusicStore\_log.ldf")]
     9. public void DeleteTest()
     10. {
     11. **using (TransactionScope ts = new TransactionScope())**
     12. **{**
     13. **StoreManagerController target = new StoreManagerController();**
     14. **int id = 669;**
     15. **ActionResult actual;**
     16. **actual = target.Delete(id, null);**
     17. **Assert.IsNotNull(actual);**
     18. **MusicStoreEntities storeDB = new MusicStoreEntities();**
     19. **var album = storeDB.Albums.SingleOrDefault(a => a.AlbumId == id);**
     20. **Assert.IsNull(album);**
     21. **}**
     22. }
  3. Run the test. On the **StoreManagerControllerTest.cs** class right-click the **DeleteTest** method and select **Run Tests**
     1. 
     2. Figure 17
     3. Running the DeleteTest test

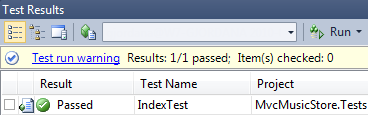
Task 4 – Testing Edit Action Method

In this task, you will learn how to test the HTTP-GET edit action method.

* 1. Open **StoreManagerControllerTest.cs** class and locate the **EditTest** method.
  2. Test the edit action by obtaining an Album’s information. To do this, replace the **EditTest** method with the following code.
     1. (Code Snippet – ASP.NET MVC 3.0 Testing – Ex2 EditTestMethod – CSharp)
     2. C#
     3. /// <summary>
     4. ///A test for Edit
     5. ///</summary>
     6. [TestMethod()]
     7. [DeploymentItem("MvcMusicStore.mdf")]
     8. [DeploymentItem("MvcMusicStore\_log.ldf")]
     9. public void EditTest()
     10. {
     11. **StoreManagerController target = new StoreManagerController();**
     12. **int id = 669;**
     13. **ActionResult actual;**
     14. **actual = target.Edit(id);**
     15. **Assert.IsNotNull(actual);**
     16. **Assert.IsInstanceOfType(actual, typeof(ViewResult));**
     17. **ViewResult viewResult = (ViewResult)actual;**
     18. **Assert.IsInstanceOfType(viewResult.ViewData.Model, typeof(StoreManagerViewModel));**
     19. **StoreManagerViewModel model = (StoreManagerViewModel)viewResult.ViewData.Model;**
     20. **Assert.AreEqual(id, model.Album.AlbumId);**
     21. **Assert.AreEqual("Ring My Bell", model.Album.Title);**
     22. **Assert.AreEqual(10, model.Genres.Count());**
     23. }
     24. **Note:** In this code block you first verify that the **Edit** action method returns an **ActionResult** object and that is of type **ViewResult**. This conversion is necessary to reach the model defined in the **StoreManagerViewModel**. Finally, the album obtained is compared by its **ID** and **Title** attributes.
  3. Run the test. On the **StoreManagerControllerTest.cs** class right-click the **EditTest** method and select **Run Tests**
     1. 
     2. Figure 18
     3. Running the EditTest test

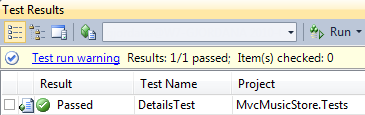
Task 5 – Testing Index Action Method

In this task, you will test the Index action method that obtains the list of genres from the store.

* 1. In the **Solution Explorer** expand the project **MvcMusicStore.Tests** anddouble click on the **StoreControllerTest.cs** class.
  2. Add the following namespaces at the beginning of the **StoreControllerTest.cs** file:
     1. (Code Snippet – ASP.NET MVC 3.0 Testing – Ex2 StoreControllerTestUsing – CSharp)
     2. C#
     3. using MvcMusicStore.Controllers;
     4. using Microsoft.VisualStudio.TestTools.UnitTesting;
     5. using System;
     6. using Microsoft.VisualStudio.TestTools.UnitTesting.Web;
     7. using System.Web.Mvc;
     8. **using MvcMusicStore.ViewModels;**
     9. **using MvcMusicStore.Models;**
     10. namespace MvcMusicStore.Tests
     11. {
  3. Test getting all the Genres for the Music Store by invoking the **Index** action method and verify the amount of genres retrieved. To do this, replace the **IndexTest** method with the following code.
     1. (Code Snippet – ASP.NET MVC 3.0 Testing – Ex2 IndexTestMethod – CSharp)
     2. C#
     3. /// <summary>
     4. /// A test for Index
     5. ///</summary>
     6. [TestMethod()]
     7. [DeploymentItem("MvcMusicStore.mdf")]
     8. [DeploymentItem("MvcMusicStore\_log.ldf")]
     9. public void IndexTest()
     10. {
     11. **StoreController target = new StoreController();**
     12. **ActionResult actual;**
     13. **actual = target.Index();**
     14. **Assert.IsNotNull(actual);**
     15. **Assert.IsInstanceOfType(actual, typeof(ViewResult));**
     16. **ViewResult viewResult = (ViewResult)actual;**
     17. **Assert.IsInstanceOfType(viewResult.ViewData.Model, typeof(StoreIndexViewModel));**
     18. **StoreIndexViewModel model = (StoreIndexViewModel)viewResult.ViewData.Model;**
     19. **Assert.AreEqual(10, model.Genres.Count);**
     20. **Assert.AreEqual(10, model.NumberOfGenres);**
     21. }
  4. Run the test. On the **StoreControllerTest.cs** class right-click the **IndexTest** method and select **Run Tests**.
     1. 
     2. Figure 19
     3. Running the IndexTest test

Task 6 – Testing Details Action Method

In this task, you will test the Details method. This method obtains a specific album based on its ID.

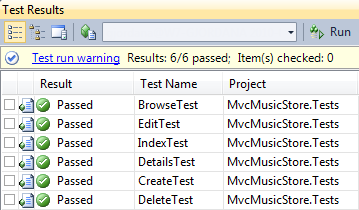
* 1. Open **StoreControllerTest.cs** class and locate the **DetailsTest** method.
  2. Test getting the Album’s information for a record stored in the provided database. To do this, replace the **DetailsTest** method with the following code.
     1. (Code Snippet – ASP.NET MVC 3.0 Testing – Ex2 DetailsTestMethod – CSharp)
     2. C#
     3. /// <summary>
     4. ///A test for Details
     5. ///</summary>
     6. [TestMethod()]
     7. [DeploymentItem("MvcMusicStore.mdf")]
     8. [DeploymentItem("MvcMusicStore\_log.ldf")]
     9. public void DetailsTest()
     10. {
     11. **StoreController target = new StoreController();**
     12. **int id = 669;**
     13. **ActionResult actual;**
     14. **actual = target.Details(id);**
     15. **Assert.IsNotNull(actual);**
     16. **Assert.IsInstanceOfType(actual, typeof(ViewResult));**
     17. **ViewResult viewResult = (ViewResult)actual;**
     18. **Assert.IsInstanceOfType(viewResult.ViewData.Model, typeof(Album));**
     19. **Album album = (Album)viewResult.ViewData.Model;**
     20. **Assert.AreEqual(id, album.AlbumId);**
     21. **Assert.AreEqual("Ring My Bell", album.Title);**
     22. }
  3. Run the test. On the **StoreControllerTest.cs** class right-click the **DetailsTest** method and select **Run Tests**
     1. 
     2. Figure 20
     3. Running the DetailsTest test

Task 7 – Testing Browse Action Method for StoreController

In this task, you will learn how to test the Browse method. This method returns a list of albums for a specific genre.

* 1. Open **StoreControllerTest.cs** class and locate the **BrowseTest** method.
  2. Test getting the Albums list for a provided genre. To do this, replace the **BrowseTest** method with the following code.
     1. (Code Snippet – ASP.NET MVC 3.0 Testing – Ex2 BrowseTestMethod – CSharp)
     2. C#
     3. /// <summary>
     4. /// A test for Browse
     5. ///</summary>
     6. [TestMethod()]
     7. [DeploymentItem("MvcMusicStore.mdf")]
     8. [DeploymentItem("MvcMusicStore\_log.ldf")]
     9. public void BrowseTest()
     10. {
     11. **StoreController target = new StoreController();**
     12. **string genre = "Disco";**
     13. **ActionResult actual;**
     14. **actual = target.Browse(genre);**
     15. **Assert.IsNotNull(actual);**
     16. **Assert.IsInstanceOfType(actual, typeof(ViewResult));**
     17. **ViewResult viewResult = (ViewResult)actual;**
     18. **Assert.IsNotNull(viewResult.ViewData.Model);**
     19. **Assert.IsInstanceOfType(viewResult.ViewData.Model, typeof(StoreBrowseViewModel));**
     20. **StoreBrowseViewModel model = (StoreBrowseViewModel)viewResult.ViewData.Model;**
     21. **Assert.AreEqual("Disco", model.Genre.Name);**
     22. **Assert.AreEqual(3, model.Albums.Count);**

}

* 1. This is the last test of the exercise where you will verify that all tests pass. To do this, in the **Test** menu, select **Run** and then **All Tests in Solution**.
     1. 
     2. Figure 21
     3. Running all testsx in the solution

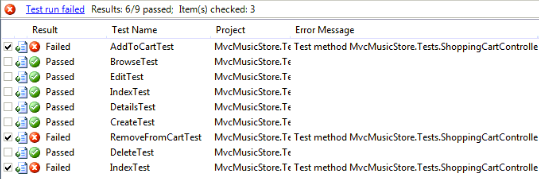
# Next Step

Exercise 3: Testing cart actions

Exercise 3: Testing Cart Actions

* 1. In this exercise, you will learn how to test the main functionality of the ShoppingCartController: adding items to the cart, removing items and displaying the cart’s content. In order to simulate those actions, the tests you will implement will use a TestCartIdProvider instance that will return a new Guid object to simulate the ID of the cart to handle for the test. The MvcMusicStore project, the one used for the application itself, behaves differently: another provider works with the user’s session to store and retrieve the cart ID.

Task 1 – Running ShoppingCartController Tests

* 1. In this task you will run the 3 tests you will be implementing later during this Exercise. Since they are not implemented with the correct content yet, they will fail.
  2. Start Microsoft Visual Studio 2010 Professional from **Start** | **All Programs** | **Microsoft Visual Studio 2010** | **Microsoft Visual Studio 2010**.
  3. In the **File** menu, choose Open **Project**. In the Open Project dialog, browse to Source\Ex03-Testing Cart actions\Begin, select **MvcMusicStore.sln** and click **Open**.
  4. In the **Solution Explorer** expand the **MvcMusicStore.Tests** project and double-click the **ShoppingCartControllerTest.cs** class to open that file.
  5. You will find 3 test methods included as part of the Begin solution of this exercise: **IndexTest**, **AddToCartTest** and **RemoveFromCartTest**. Run all tests in the solution to verify that those 3 are failing. To do this, click on the following button in the Test toolbar or press *CTRL+R, A*.
     1. 
     2. Figure 22
     3. Running all tests in the solution
  6. The 3 test methods mentioned run and fail.
     1. 
     2. Figure 23
     3. Tests failing

Task 2 – Implementing the test for the AddToCart action method

* 1. In this task you will implement the test for the AddToCart action method of the ShoppingCart Controller. The AddToCart method receives an album ID that is used to retrieve the album from the database, add it to the shopping cart and then returns the Index View template that will display that album as part of the cart.
  2. Open the ShoppingCartControllerTest.cs file and replace the using directives with the following, needed to run the tests of this Exercise:
     1. (Code Snippet – ASP.NET MVC 3.0 Testing – Ex3 using directives – CSharp)
     2. C#
     3. **using MvcMusicStore.Controllers;**
     4. **using Microsoft.VisualStudio.TestTools.UnitTesting;**
     5. **using System;**
     6. **using System.Linq;**
     7. **using Microsoft.VisualStudio.TestTools.UnitTesting.Web;**
     8. **using System.Web.Mvc;**
     9. **using MvcMusicStore.Models;**
     10. **using System.Collections.Generic;**
     11. **using System.Transactions;**
     12. **using MvcMusicStore.ViewModels;**
  3. Replace the **AddToCartTest** method. Note that the **Assert.Inconclusive** method was removed with the real implementation for the test.
     1. (Code Snippet – ASP.NET MVC 3.0 Testing – Ex3 AddToCartTest – CSharp)
     2. C#
     3. /// <summary>
     4. ///A test for AddToCart
     5. ///</summary>
     6. [TestMethod()]
     7. [DeploymentItem("MvcMusicStore.mdf")]
     8. [DeploymentItem("MvcMusicStore\_log.ldf")]
     9. public void AddToCartTest()
     10. {
     11. **using (TransactionScope ts = new TransactionScope())**
     12. **{**
     13. **ICartIdProvider provider = new TestCartIdProvider();**
     14. **ShoppingCartController target = new ShoppingCartController(provider);**
     15. **int id = 669;**
     16. **ActionResult actual;**
     17. **actual = target.AddToCart(id);**
     18. **Assert.IsNotNull(actual);**
     19. **ShoppingCart cart = ShoppingCart.GetCart(provider);**
     20. **List<Cart> items = cart.GetCartItems();**
     21. **Assert.IsNotNull(items);**
     22. **Assert.AreEqual(1, items.Count);**
     23. **Assert.AreEqual(id, items[0].AlbumId);**
     24. **Assert.AreEqual(1, items[0].Count);**
     25. **}**
     26. }
     27. **Note:**  In order to simulate the addition of an album to the shopping cart, this test will need to invoke the **ShoppingCartController**’s **AddToCart** action. You will find that when creating the **ShoppingCartController** object, a **TestCartIdProvider** instance is passed as parameter.
     28. **TestCartIdProvider** implements **ICartIdProvider** and will return a new **Guid** object to simulate the ID of the cart where you will add a new album. In the **MvcMusicStore** project, the one used for the application itself, the provider used is the **CartIdProvider**, which also implements **ICartIdProvider** but instead uses the user’s session to store and retrieve the cart ID. So, if you run the web application, its default behavior is to use the session id provider, as in previous exercise.
     29. Once the **ShoppingCartController** object is created, a well-known album id (669) is used to invoke the **AddToCart** action. Afterwards, the cart items are retrieved and the test asserts that:
     30. 1. Result of **AddToCart** invocation is not null
     31. 2. The cart’s items list is not null
     32. 3. There is only one cart
     33. 4. The cart’s first item’s ID is the same used on the **AddtoCart** action
     34. 5. The cart contains only one item (the album recently added)
  4. Run exclusively this test to verify the results. To do this, right-click inside the **AddToCartTest** method and then select **Run Tests**.
  5. The **Test Result** window should show that the test passes.
     1. 
     2. Figure 24
     3. Running test AddToCartTest

Task 3 – Implementing the test for the Index action method

* 1. In this task you will implement the test for the Index action method of the ShoppingCart Controller. The Index method populates a **ShoppingCartViewModel** object with the cart items and its total cost in order to display them with the corresponding View template.
  2. In the ShoppingCartControllerTest.cs file, add the **GetAlbum** private method, needed to test the Index action method.
     1. (Code Snippet – ASP.NET MVC 3.0 Testing – Ex3 GetAlbum method – CSharp)
     2. C#
     3. **private Album GetAlbum(int id)**
     4. **{**
     5. **MusicStoreEntities storeDB = new MusicStoreEntities();**
     6. **return storeDB.Albums.Single(a => a.AlbumId == id);**
     7. **}**
  3. Replace the **IndexTest** method. Note that the **Assert.Inconclusive** method was removed with the real implementation for the test.
     1. (Code Snippet – ASP.NET MVC 3.0 Testing – Ex3 IndexTest – CSharp)
     2. C#
     3. /// <summary>
     4. ///A test for Index
     5. ///</summary>
     6. [TestMethod()]
     7. [DeploymentItem("MvcMusicStore.mdf")]
     8. [DeploymentItem("MvcMusicStore\_log.ldf")]
     9. public void IndexTest()
     10. {
     11. **using (TransactionScope ts = new TransactionScope())**
     12. **{**
     13. **Album album1 = this.GetAlbum(669);**
     14. **Album album2 = this.GetAlbum(668);**
     15. **ICartIdProvider provider = new TestCartIdProvider();**
     16. **ShoppingCart cart = ShoppingCart.GetCart(provider);**
     17. **cart.AddToCart(album1);**
     18. **cart.AddToCart(album2);**
     19. **cart.AddToCart(album2);**
     20. **ShoppingCartController target = new ShoppingCartController(provider);**
     21. **ActionResult actual;**
     22. **actual = target.Index();**
     23. **Assert.IsNotNull(actual);**
     24. **Assert.IsInstanceOfType(actual, typeof(ViewResult));**
     25. **ViewResult viewResult = (ViewResult)actual;**
     26. **Assert.IsInstanceOfType(viewResult.ViewData.Model, typeof(ShoppingCartViewModel));**
     27. **ShoppingCartViewModel model = (ShoppingCartViewModel)viewResult.ViewData.Model;**
     28. **Assert.AreEqual(2, model.CartItems.Count);**
     29. **Assert.AreEqual(3, model.CartItems.Sum(it => it.Count));**
     30. **}**
     31. }
     32. **Note:** In order to simulate the correct display of the shopping cart content, this test will:
     33. 1. Retrieve from the database 2 well-known albums
     34. 2. Add them to a new shopping cart: the first album once and the second album twice; hence making the cart to hold 3 items.
     35. 3. Invoke the **Index** action method of the **ShoppingCartController.**
     36. 4. Assert that:

- Result of that invocation is not null

- Result of that invocation is actually a View

- The view’s model is of type **ShoppingCartViewModel**

- The view’s model holds 2 cart items which are different

- The view’s model holds 3 cart items in total

* 1. Run exclusively this test to verify the results. To do this, right-click inside the **IndexTest** method and then select **Run Tests**.
  2. The **Test Result** window should show that the test passes.
     1. 
     2. Figure 25
     3. Running test IndexTest

Task 4 – Implementing the test for the RemoveFromCart action method

* 1. In this task you will implement the test for the RemoveFromCart action method of the ShoppingCart Controller. The RemoveFromCart receives the ID of the album to remove, used for invoking the RemoveFromCart method of the ShoppingCart Model class and then it displays a confirmation message.
  2. In the ShoppingCartControllerTest.cs file, replace the **RemoveFromCartTest** method. Note that the **Assert.Inconclusive** method was removed with the real implementation for the test.
     1. (Code Snippet – ASP.NET MVC 3.0 Testing – Ex3 RemoveFromCartTest – CSharp)
     2. C#
     3. /// <summary>
     4. ///A test for RemoveFromCart
     5. ///</summary>
     6. [TestMethod()]
     7. [DeploymentItem("MvcMusicStore.mdf")]
     8. [DeploymentItem("MvcMusicStore\_log.ldf")]
     9. public void RemoveFromCartTest()
     10. {
     11. **using (TransactionScope ts = new TransactionScope())**
     12. **{**
     13. **Album album1 = this.GetAlbum(669);**
     14. **Album album2 = this.GetAlbum(668);**
     15. **ICartIdProvider provider = new TestCartIdProvider();**
     16. **ShoppingCart cart = ShoppingCart.GetCart(provider);**
     17. **cart.AddToCart(album1);**
     18. **cart.AddToCart(album2);**
     19. **cart.AddToCart(album2);**
     20. **ShoppingCartController target = new ShoppingCartController(provider);**
     21. **int id = cart.GetCartItems().First().RecordId;**
     22. **ActionResult actual;**
     23. **actual = target.RemoveFromCart(id);**
     24. **Assert.IsNotNull(actual);**
     25. **Assert.IsInstanceOfType(actual, typeof(JsonResult));**
     26. **JsonResult jsonResult = (JsonResult)actual;**
     27. **Assert.IsInstanceOfType(jsonResult.Data, typeof(ShoppingCartRemoveViewModel));**
     28. **ShoppingCartRemoveViewModel model = (ShoppingCartRemoveViewModel)jsonResult.Data;**
     29. **Assert.AreEqual(2, model.CartCount);**
     30. **Assert.AreEqual(id, model.DeleteId);**
     31. **}**
     32. }
     33. **Note:** In order to simulate the correct removal of an album from the shopping cart, this test will:
     34. 1. Retrieve from the database 2 well-known albums
     35. 2. Add them to a new shopping cart: the first album once and the second album twice; hence making the cart to hold 3 items.
     36. 3. Retrieve the ID of the first album added to the cart
     37. 4. Invoke the **RemoveFromCart** action method of the **ShoppingCartController**, passing the ID retrieved in the last step.
     38. 5. Assert that:
     39. - Result of that invocation is not null
     40. - Result of that invocation is actually Json
     41. - Json’s result data is of type **ShoppingCartRemoveViewModel**
     42. - Json’s result data now holds 2 cart items in total (instead of 3)
     43. - Json’s result data’s ID of the removed item matches the ID of the first album added to the cart, which was removed from the cart later.
  3. Run exclusively this test to verify the results. To do this, right-click inside the **RemoveFromCartTest** method and then select **Run Tests**.
  4. The **Test Result** window should show that the test passes.
     1. 
     2. Figure 26
     3. Running test RemoveFromCartTest

# Next Step

Summary

Summary

* 1. By completing this Hands-On Lab you have learned how to test controllers that interact with a database and the user session in an ASP.NET MVC application. Some of the key elements that you learned to use are:
     + Testing Project
     + Unit tests
     + Assert statements
     + Transactions
     + DeploymentItem