# Lab: Web Services Testing

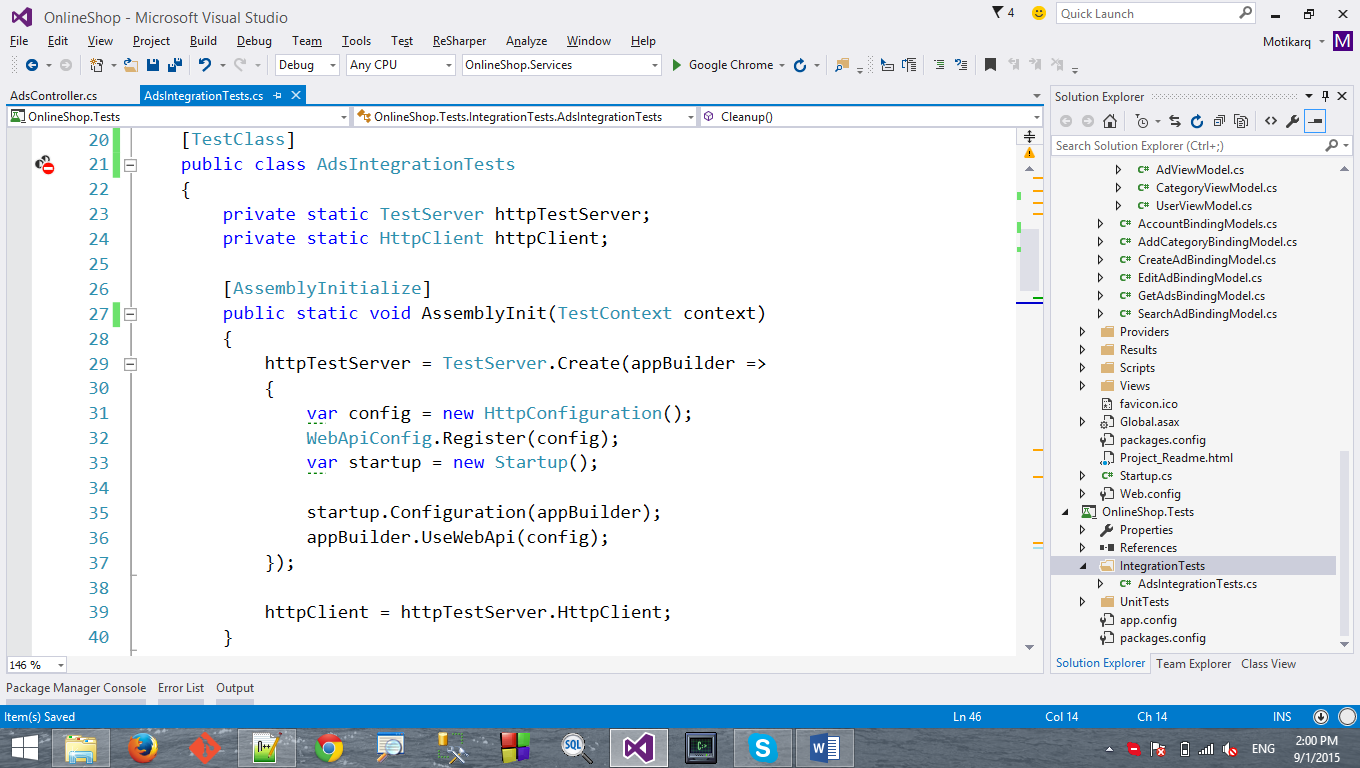
This document defines the lab assignment from the ["Web Services and Cloud" Course @ Software University](https://softuni.bg/courses/web-services-and-cloud/).

This lab is a continuation of the **previous** [**lab**](https://softuni.bg/downloads/svn/web-services-and-cloud/August-2015/5.%20Web-Services-Testing-Exercise.docx) **assignment** from the **Web Services Testing** topic. The goal is to practice writing **integration tests**.

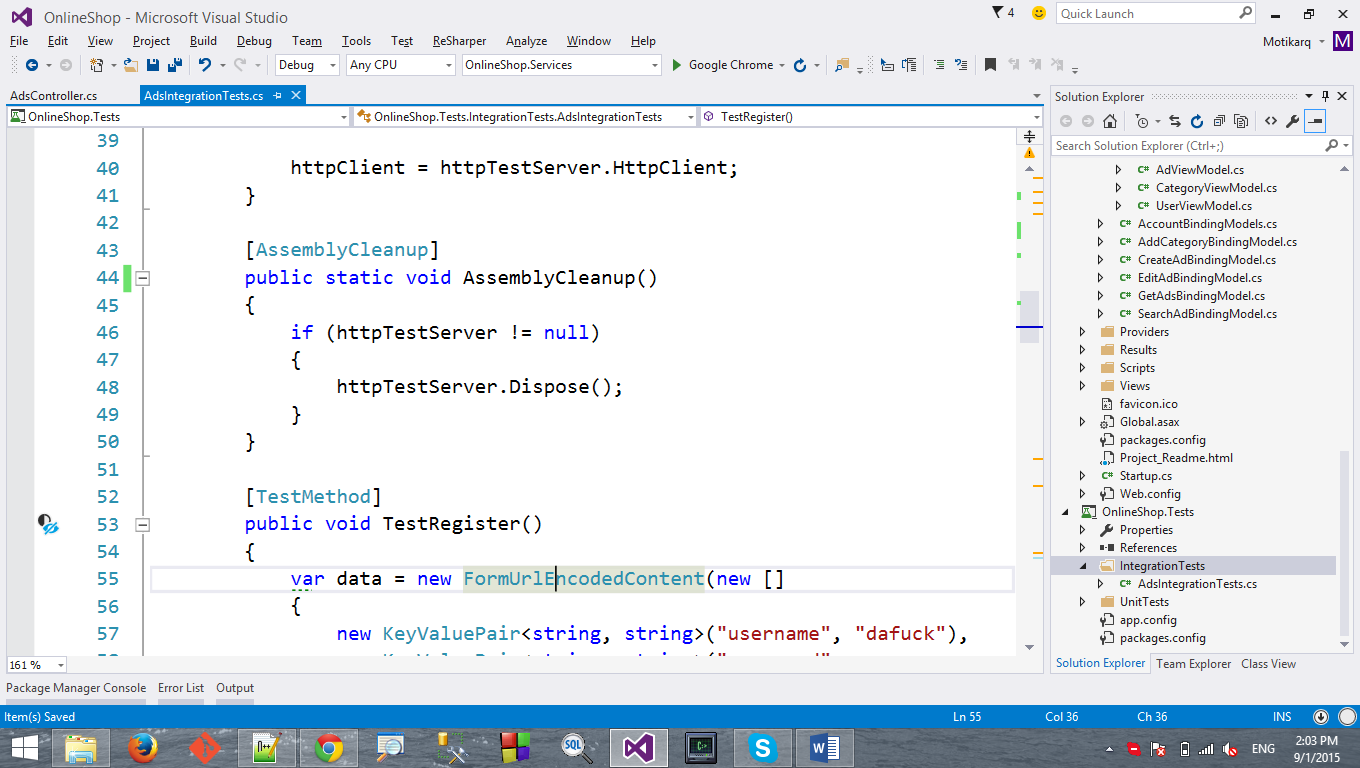
# 6. Integration Tests

**Integration** tests (or **end-to-end** tests) test the entire application.

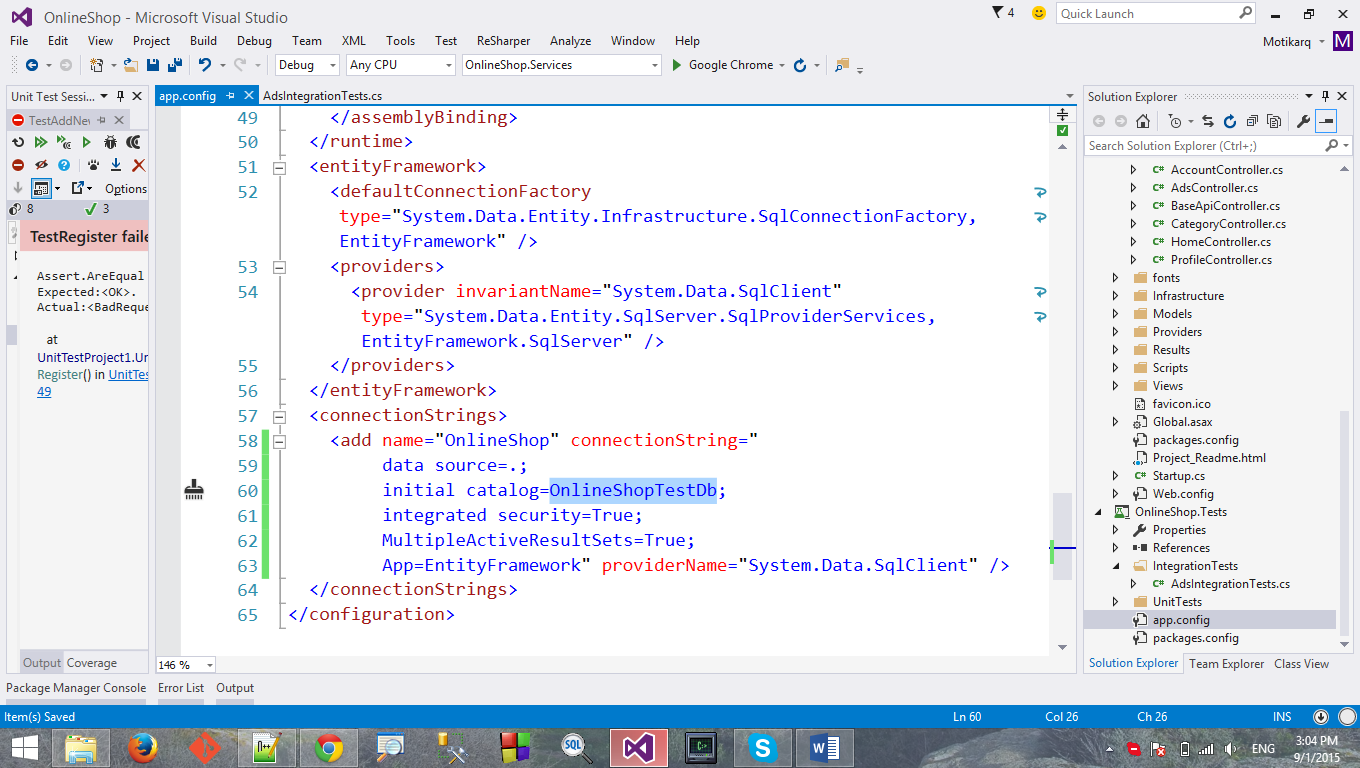
1. Install the **Microsoft.Owin.Testing** and **Microsoft.AspNet.WebApi.OwinSelfHost** from NuGet. Those packages will allow us to **self-host our Web API services** in an **in-memory server**.
2. Create a new **AdsIntegrationTests** class. It will hold the integrations tests for ads endpoints, so mark it with the **[TestClass]** attribute.
3. Create a static void method called **AssemblyInit**. Mark it with the **[AssemblyInitialize]** attribute - by doing so we tell the unit testing framework (VSTT) to **execute this method only once** when the assembly is loaded for execution.
4. Setup the test server as shown below:



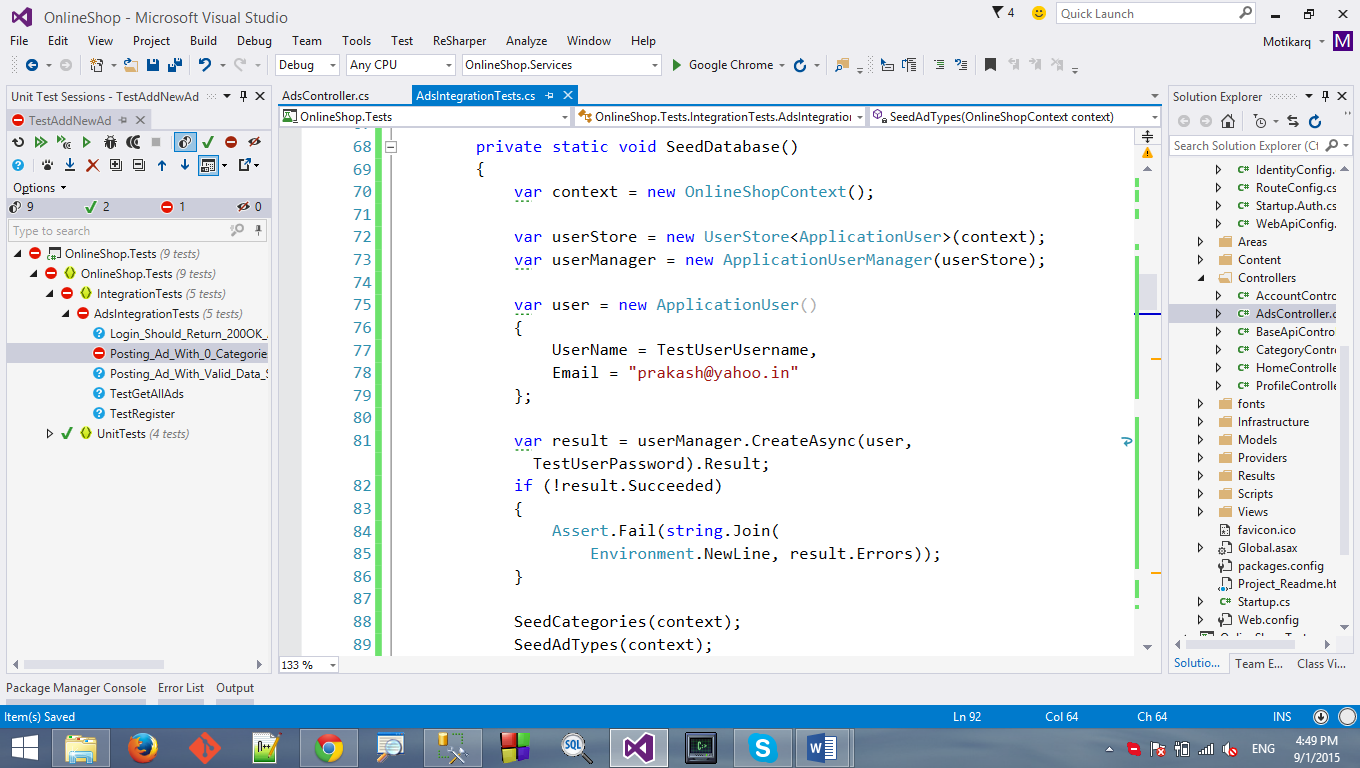
1. The above code loads our Online shop services into in-memory server.
2. Create a static void method and mark it with the **[AssemblyCleanup]** attribute. It will also be executed only once, but after all tests have been run. It basically closes the test server.



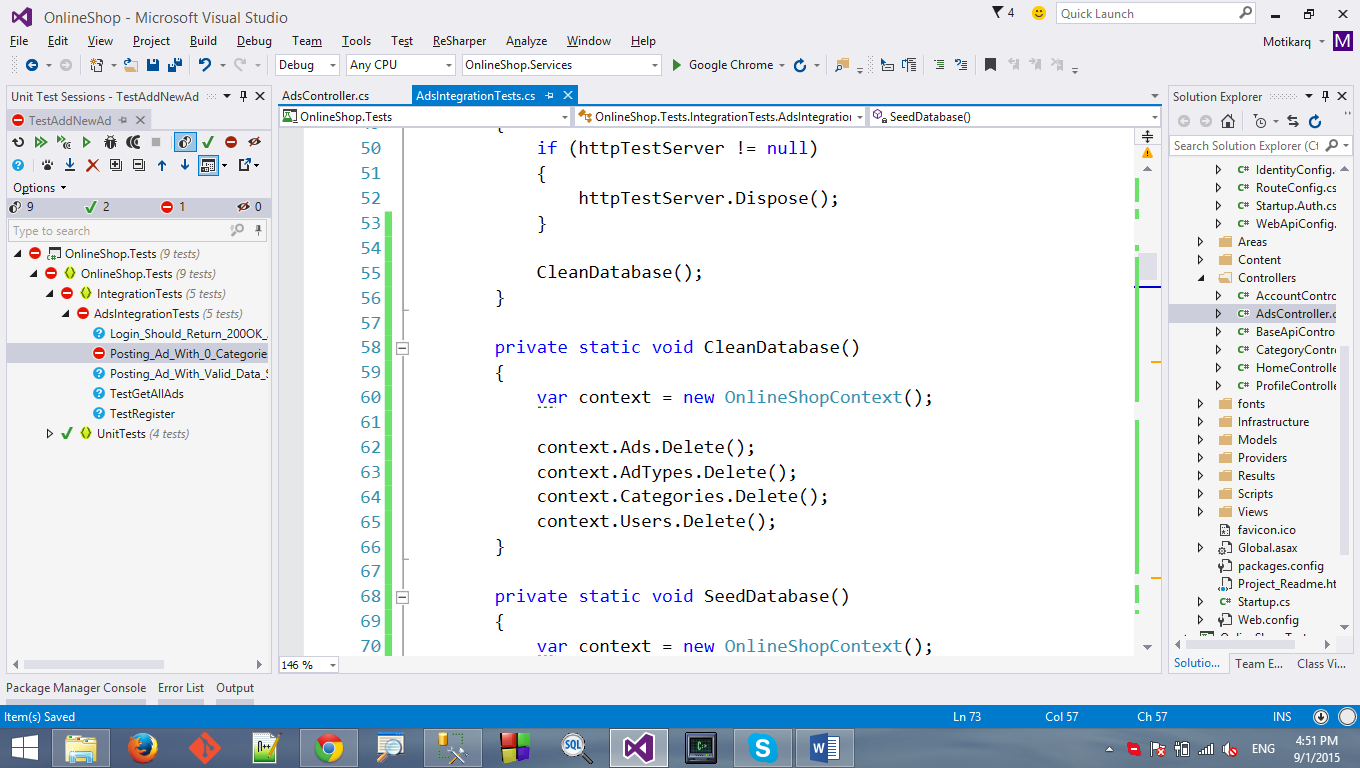
1. Install **Entity Framework** and setup a connection string in the **App.config** file.
   * Make sure the connection string's **name** corresponds to the OnlineShopContext's.
   * Set the **initial catalog** (database name) to "**OnlineShopTestDb**".



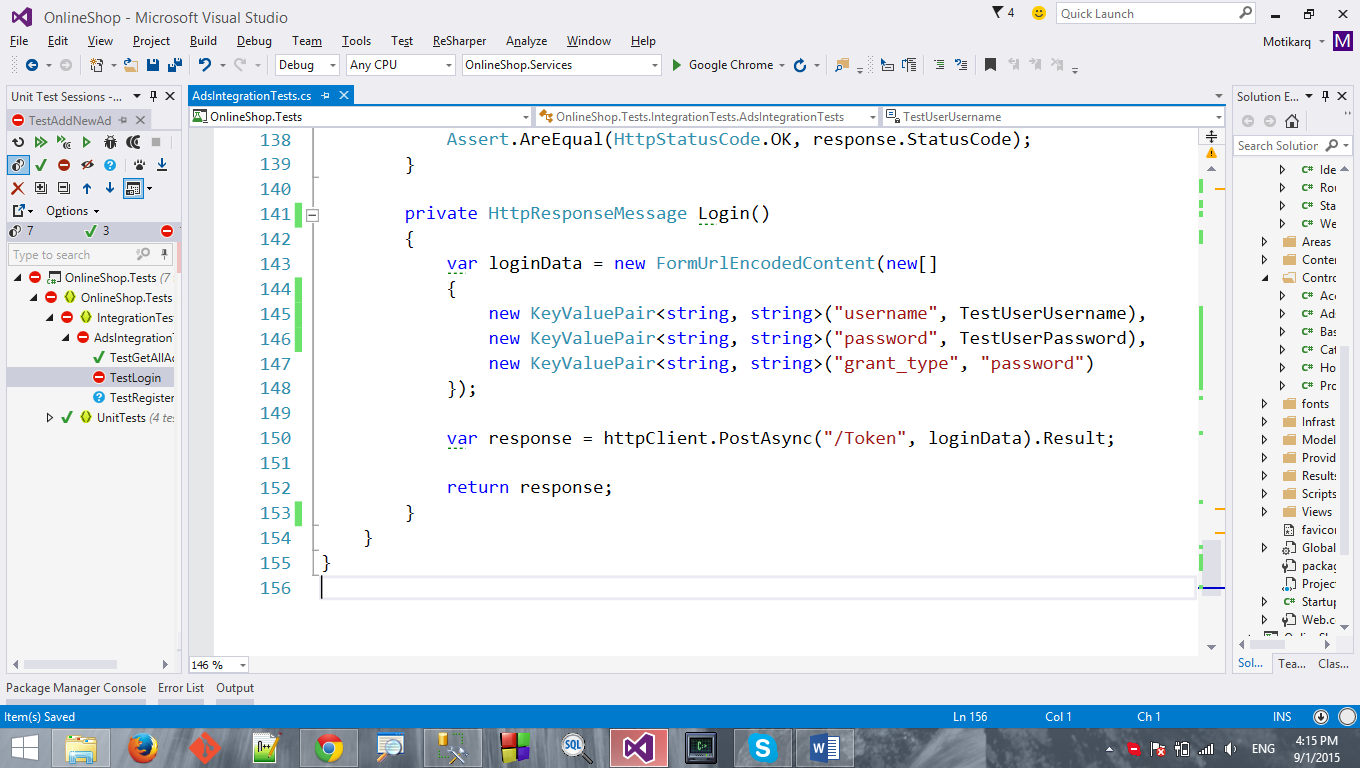
1. Now let's create a **test initialize method**. It will **seed** data into the database after the test server is started.
   * Extract the username and password into constants so we can reuse them in some tests.
   * Call the method in assembly initialize.

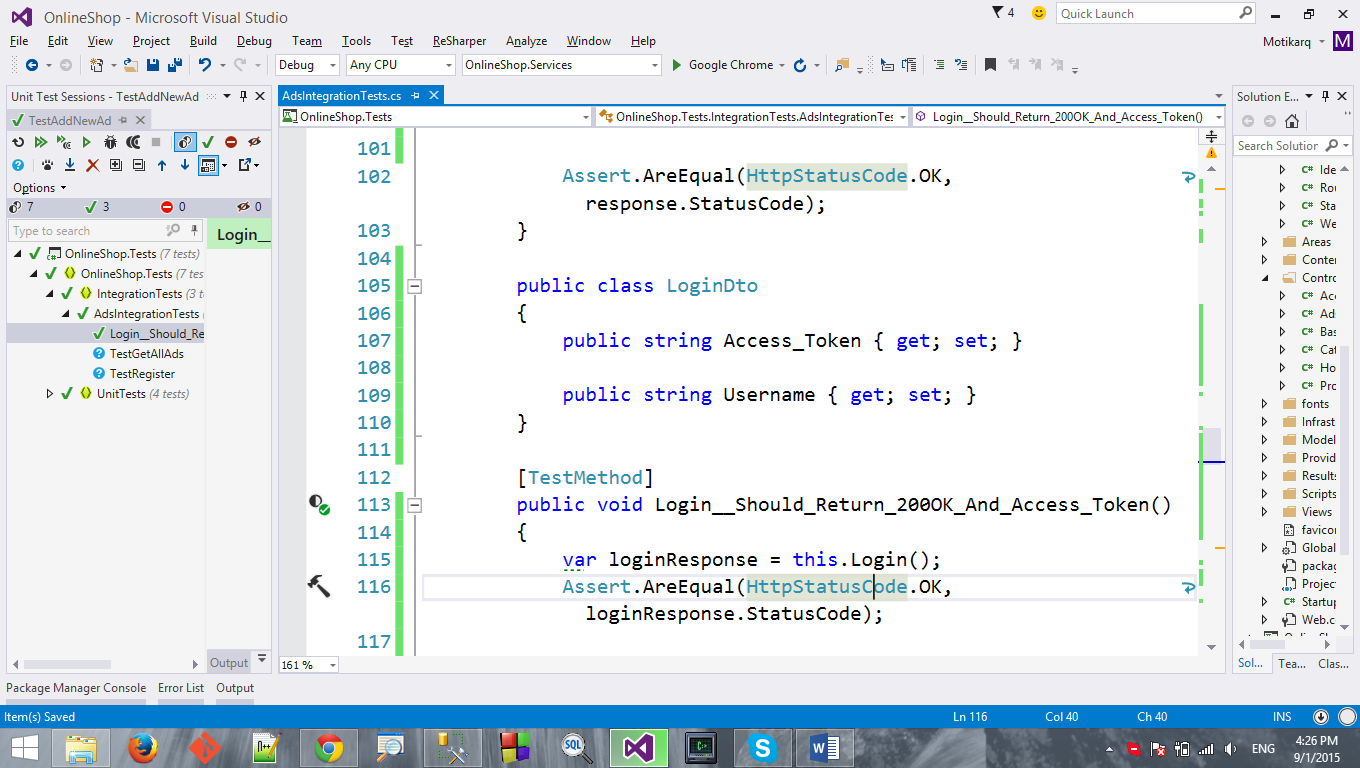


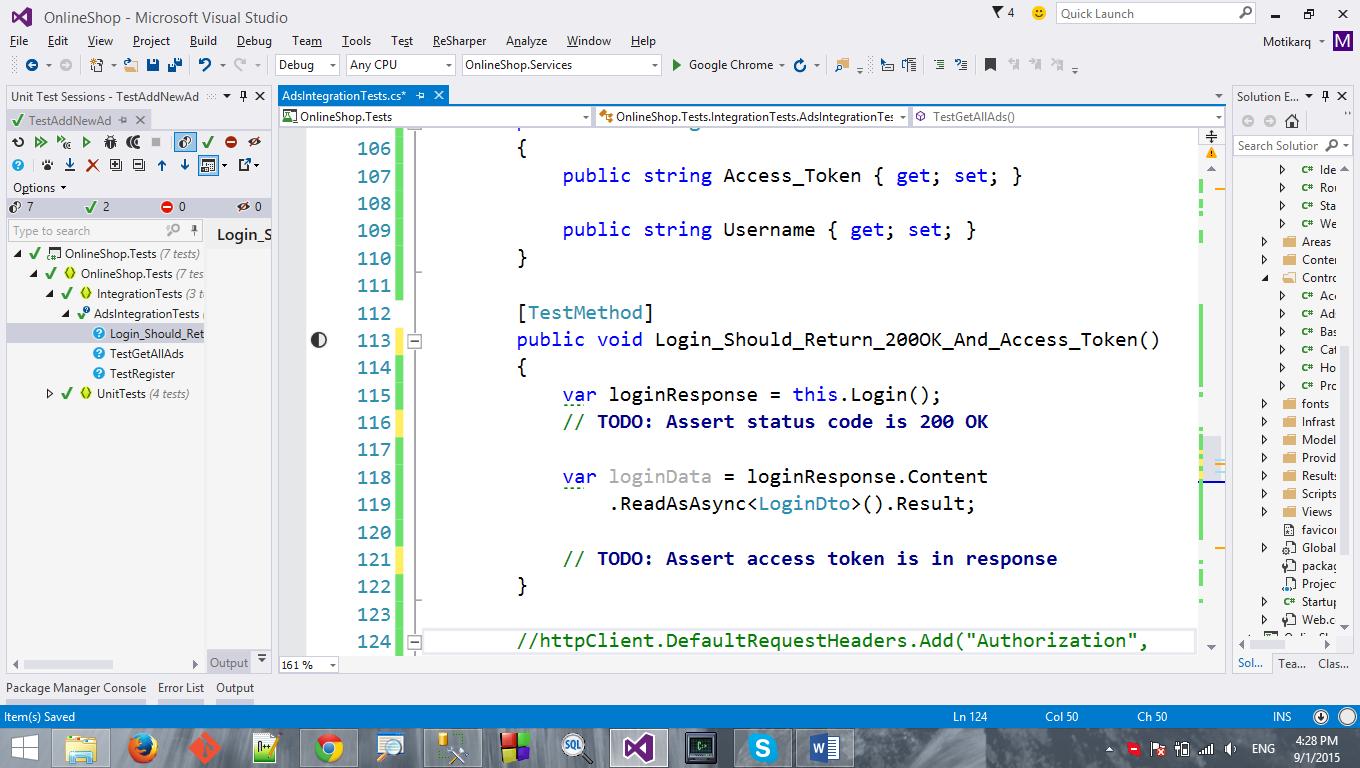
1. Create a **test cleanup method** for clearing the database of all data after all test have been run.
   * Install the **EntityFramework.Extended** package so we can use the one-line **Delete()** extension method.
   * Call the method in assembly cleanup.



### 1. Login







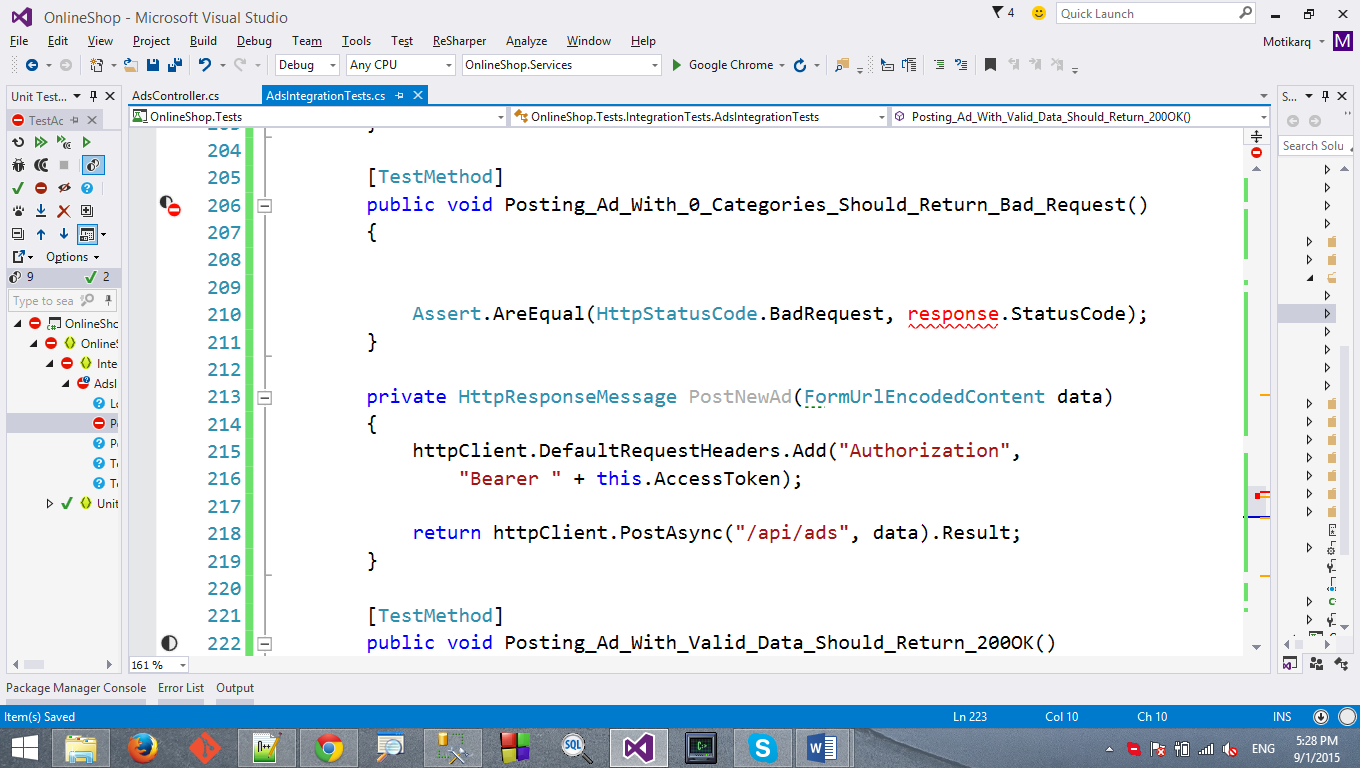
### 2. Create Ad

We want to write integration tests for the following scenarios:

* Valid ad binding model -> 200 OK
* Missing ad name (required) -> 400 Bad Request
* Non-existing ad type (invalid id) -> 400 Bad Request
* No ad categories (must be in range 1..3) -> 400 Bad Request
* More than 3 ad categories (must be in range 1..3) -> 400 Bad Request

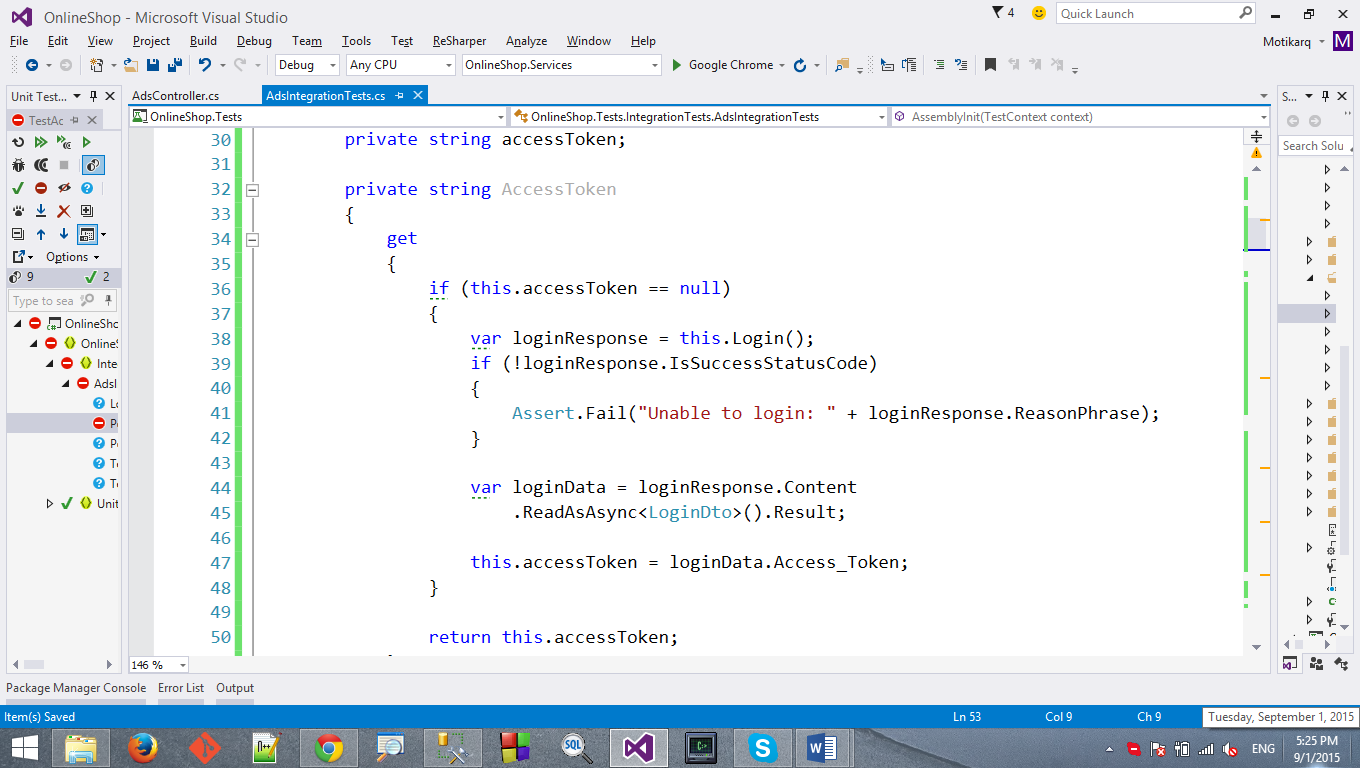
Now, we can easily write integration test for those cases. But let's do it **without repeating** any code. What's common in all 4 tests? Everything, except the data we send.

1. Extract the main logic into a method which accepts the data and sends a request:

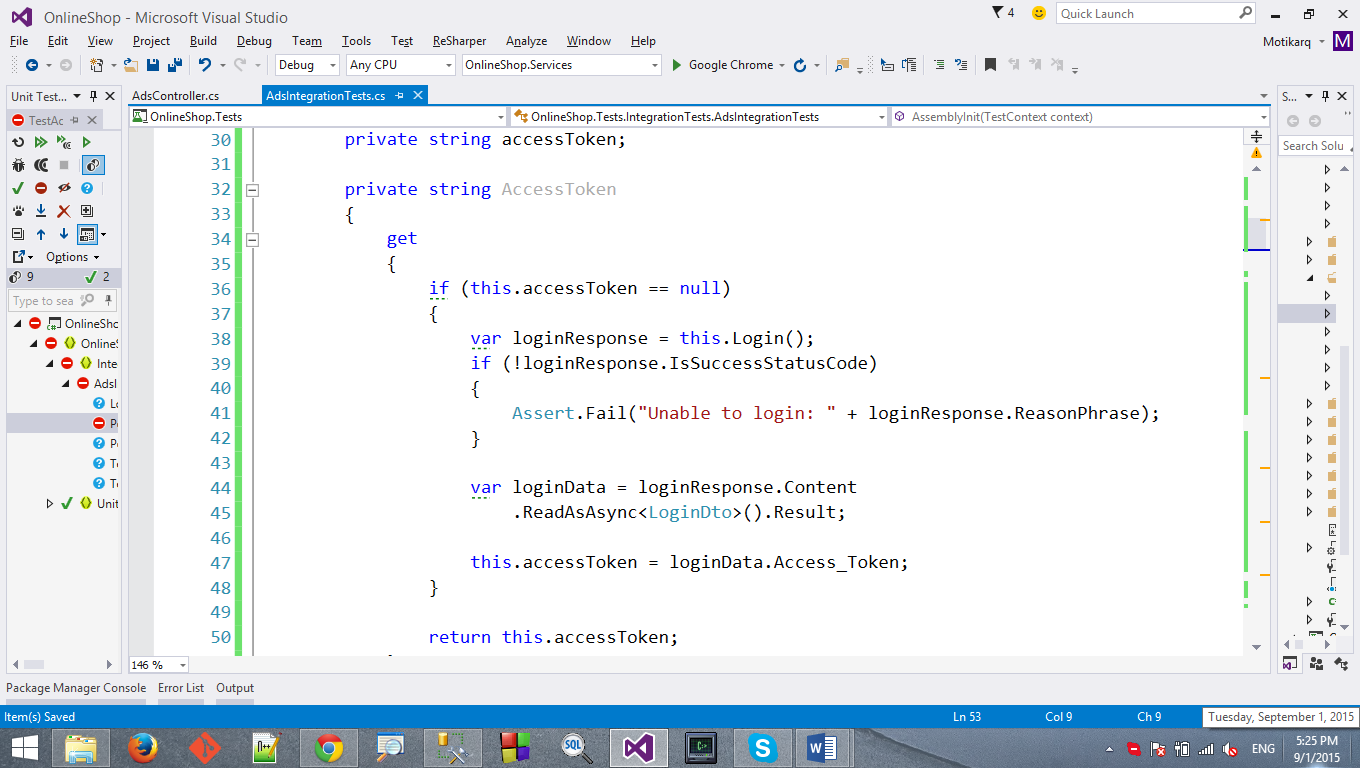


Instead of performing a login for each request that requires authorization, let's keep the access token after the first login. How?

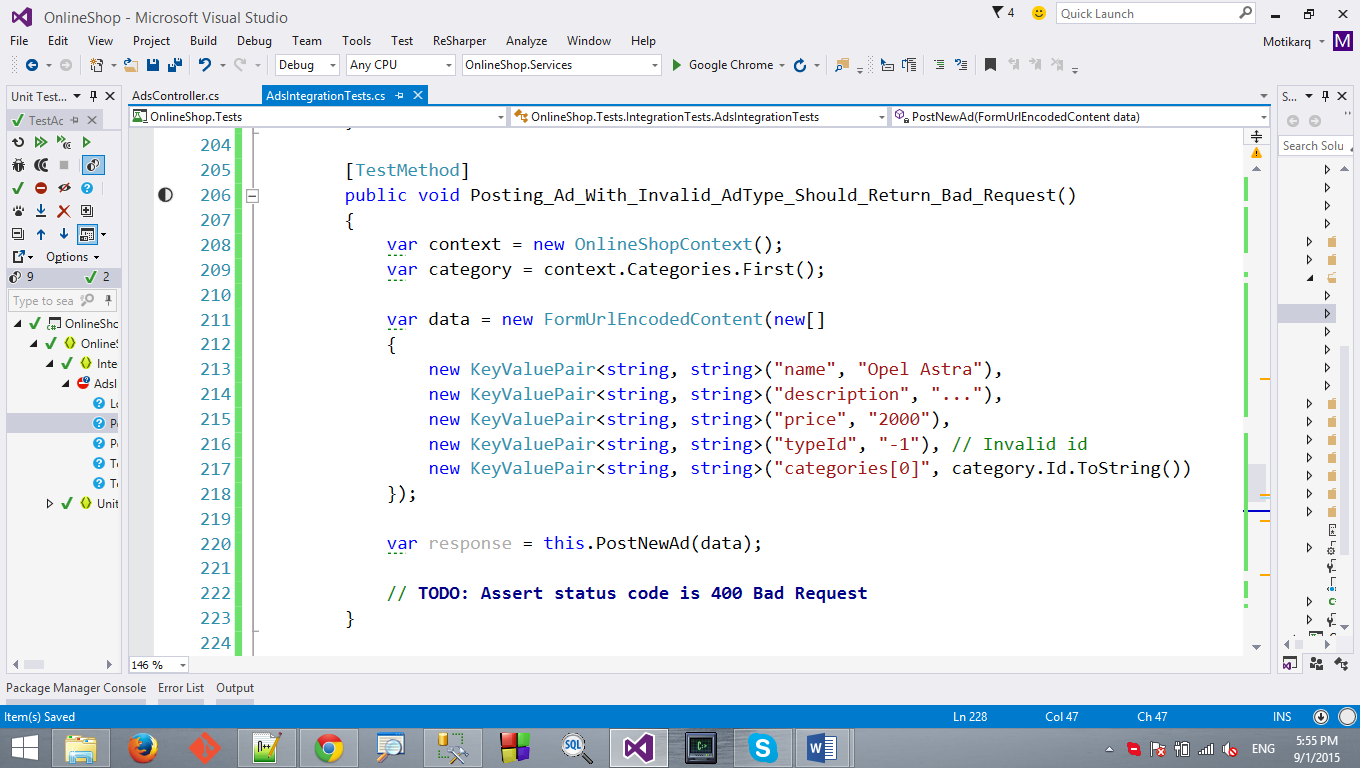
1. Define a **private field** for keeping the token:



1. Define a **private property** with a **getter** only. It should return the access token.
   * If it's null, it should call **Login()** and set the token from the response.



1. Create a test method. Test adding a new ad with an **invalid** ad type id.
   * Assert that the status code is **400 Bad Request**.



1. Do the same for the other cases. Write the following tests by **reusing** the **PostNewAd** method we wrote earlier.

