**BCIT Comp 4952 HCI for Application Development**

**Technical Programming Option**

**Option Head Mirela Gutica**

**Fall 2022**

**Implement Authentication and Authorization (2) - Identity**

1. **Introduction**

ASP.NET (all flavors Windows Forms, MVC, MVC Core, etc.) is provided with three types of authentication:

* Windows-based authentication
  + Causes a login dialog box
  + Supported by most browsers
  + Configured through the IIS management console
  + Uses Windows accounts
* Individual user authentication
  + Developers add a login page
  + Encrypts the username and password
  + Doesn’t use the Windows user account
* Third-party authentication (e.g., Facebook, Google, Twitter, etc.)
  + Provided by third-party services
  + Allows users to use their existing accounts
  + Can use two-factor authentication

ASP.NET Core Identity is an API that supports user interface (UI) login functionality. Manages users, passwords, profile data, roles, claims, tokens, email confirmation, and more.

Users can create an account with the login information stored in Identity or they can use an external login provider. Supported external login providers include Third-party authentication including Facebook, Google, Microsoft Account, and Twitter.

1. **Add Identity in ASP.NET Core MVC 3.1**

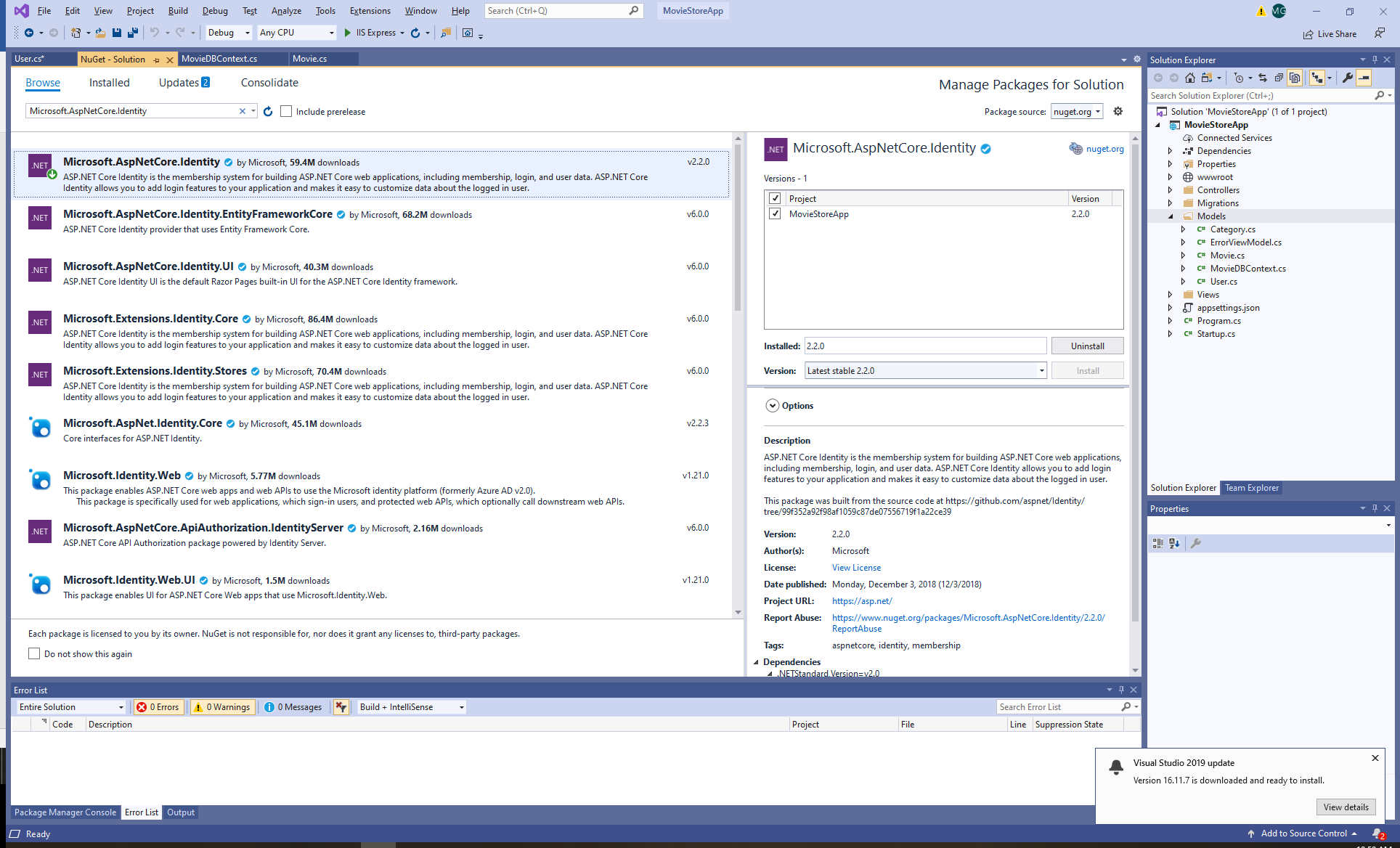
The following steps are needed:

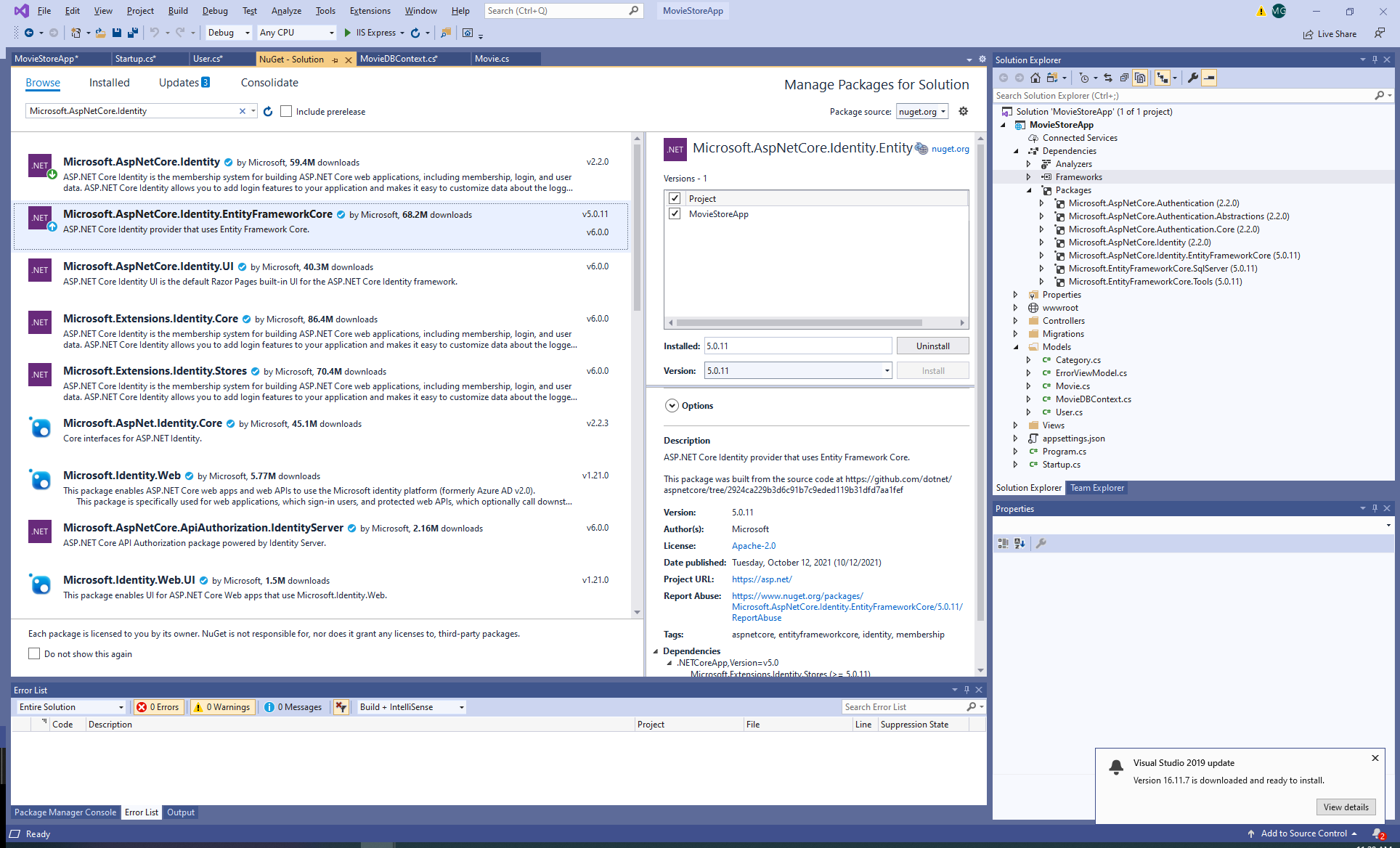
1. Add the Identity Package
2. Modify the DBContext class to include the IdentityUser
3. Add a User model
4. Use Migrations to update the DB
5. Configure middleware to allow authorization and authentication (Startup.cs)
6. Add ViewModels for Register and Login
7. Add a Controller named Account and add methods for Login(), Register(), Logout(), AccessDenied().
8. Add Vies for Register, Login and AccesDenied.
   1. Logout doesn’t need a View
9. Modify the shared View \_Layout.cshtml by adding a check for authentication and Log In/Out and Registration buttons.
10. **Add the Identity Package**

Use the NuGet package to your project:

Microsoft.AspNetCore.Identity.EntityFrameworkCore

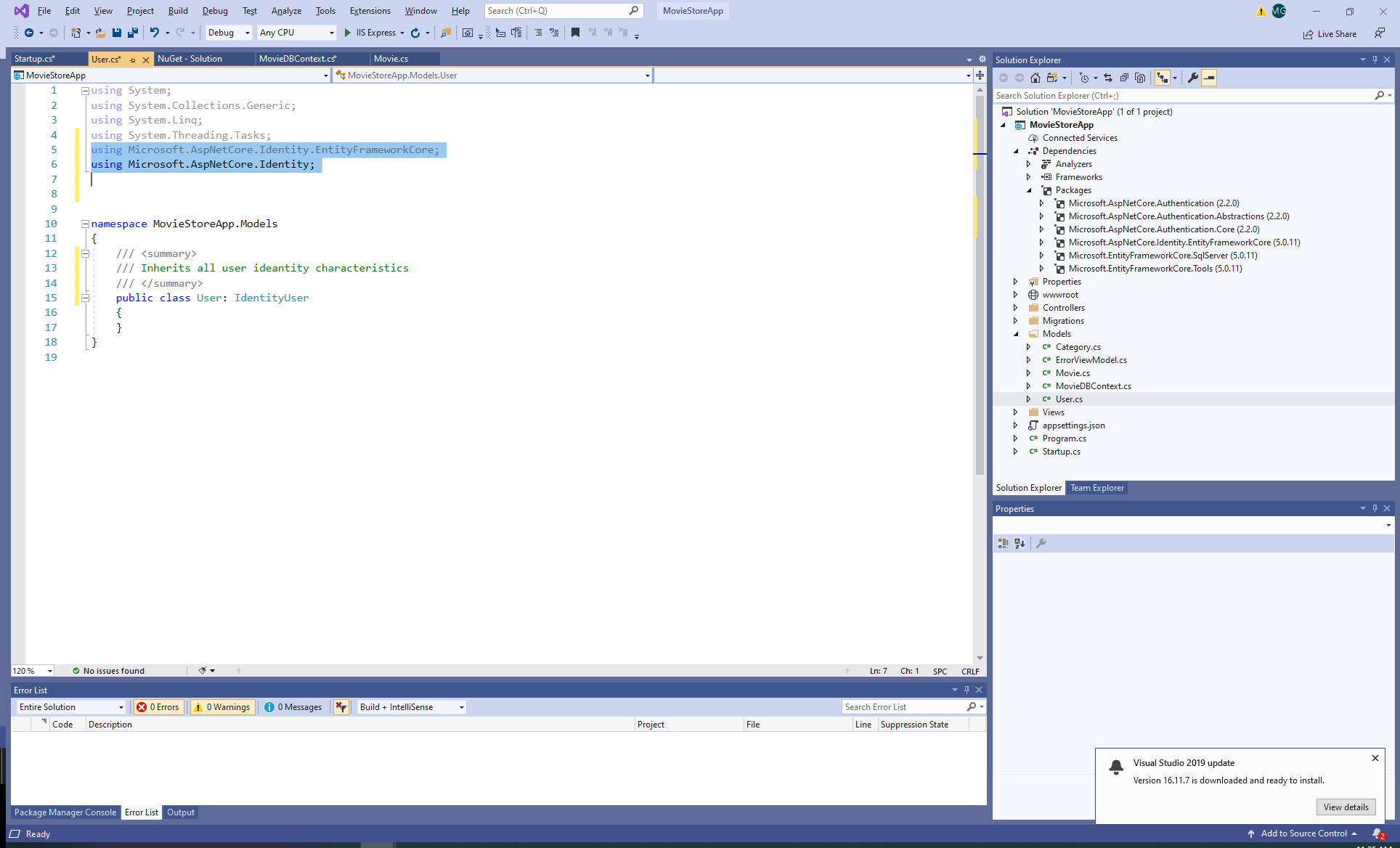
Important: Version 6.0 is not working with .NET Core 3.1. Therefore, install the version 5.0.11





1. Create the User Identity Class

Create a new Model class User. Include the *Identity* and the *Identity.EntityFrameworkCore* namespaces:

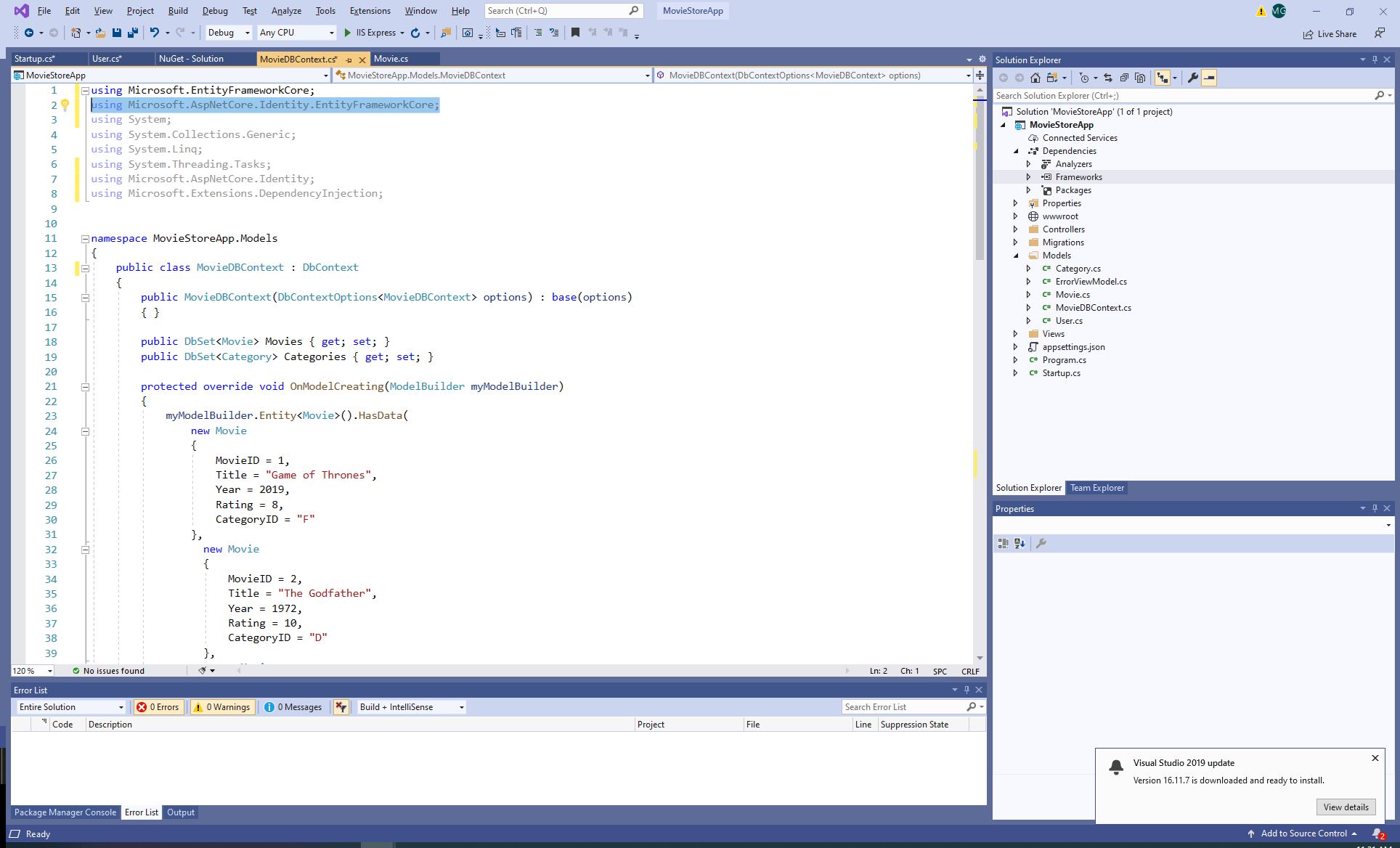


1. Add the Identity to the application context class:

using Microsoft.AspNetCore.Identity;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;



1. Add User Identity to the DBContext Class:

* In the DBContext class inherit from IdentityDbContext<IdentityUser> and add the ModelBuilder:

public class MovieDBContext : IdentityDbContext<IdentityUser>

protected override void OnModelCreating(ModelBuilder myModelBuilder)

{

base.**OnModelCreating**(myModelBuilder);…



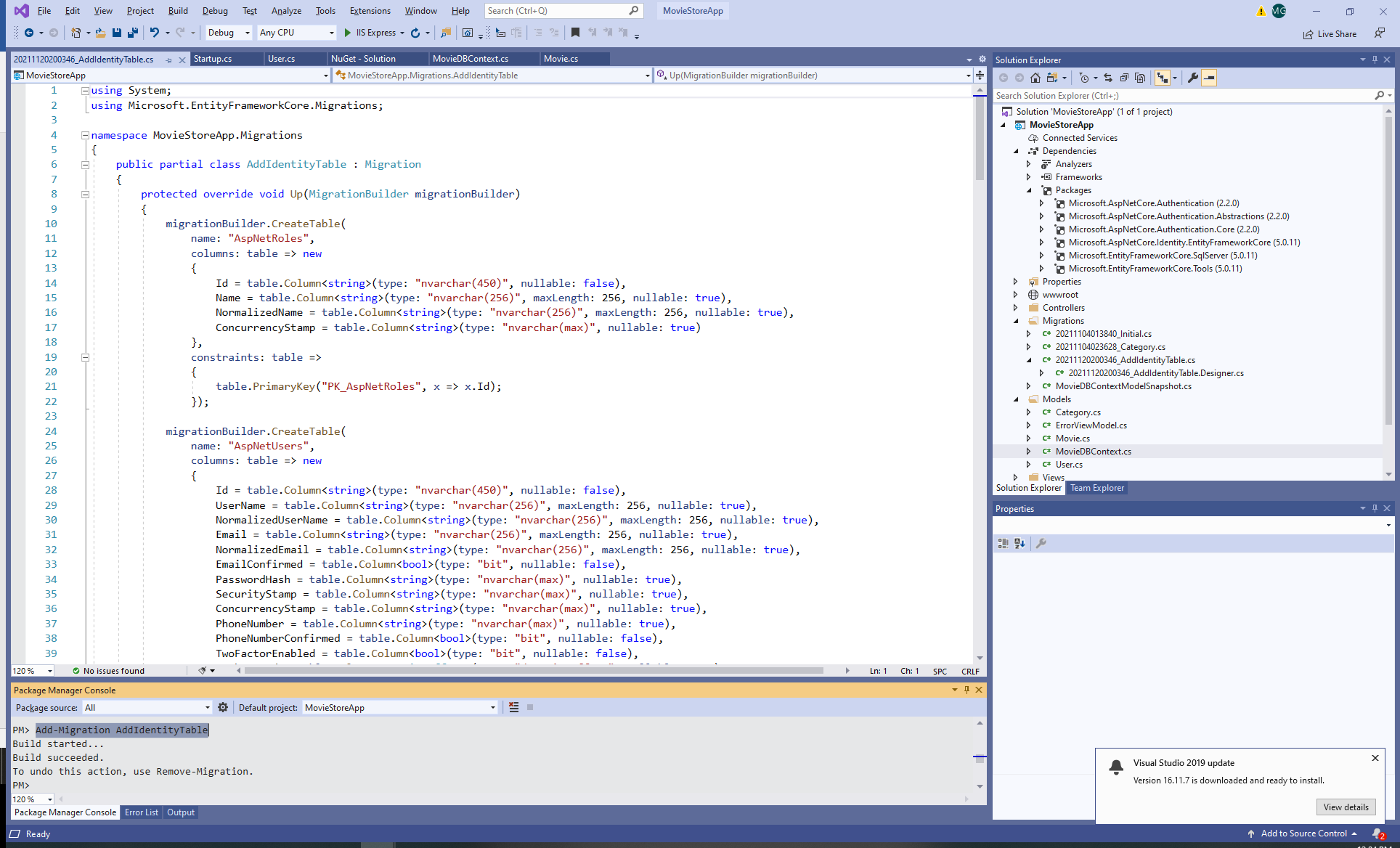
1. Add the Identity table to the Database

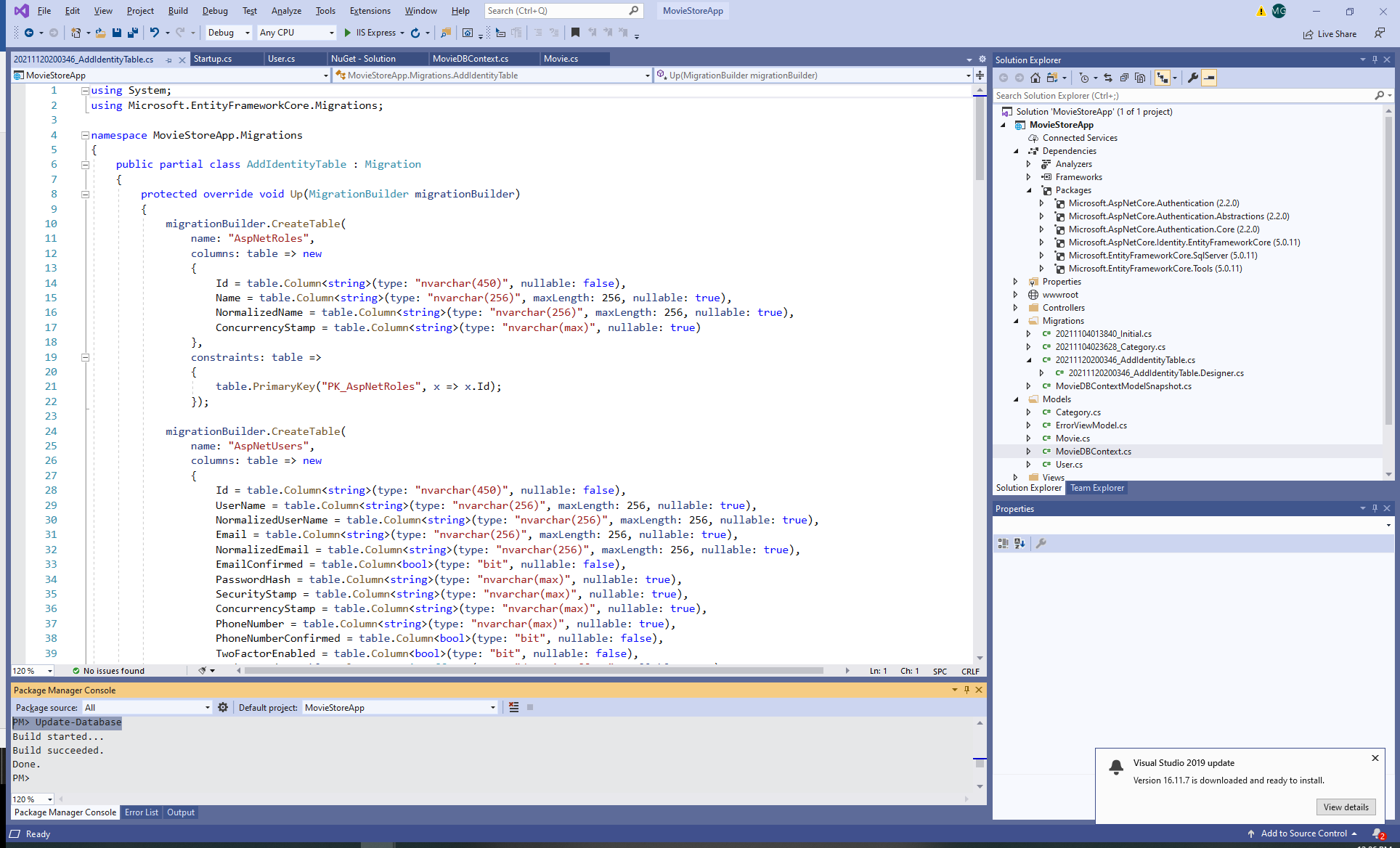
Use migrations to create the user tables in the database and update the existing database.

In the Package Manager Console (PM) run:

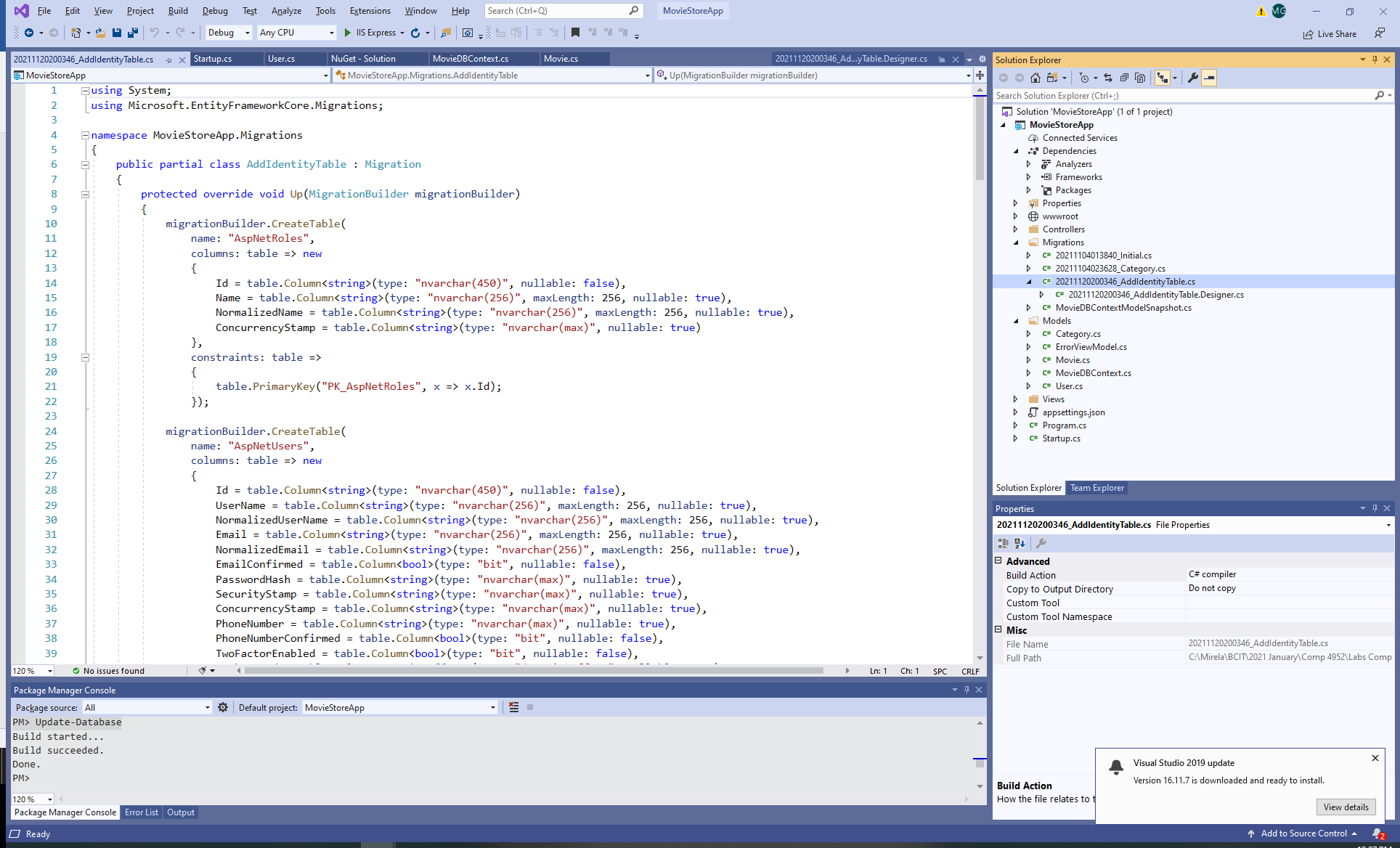
Add-Migration AddIdentityTable

Update-Database

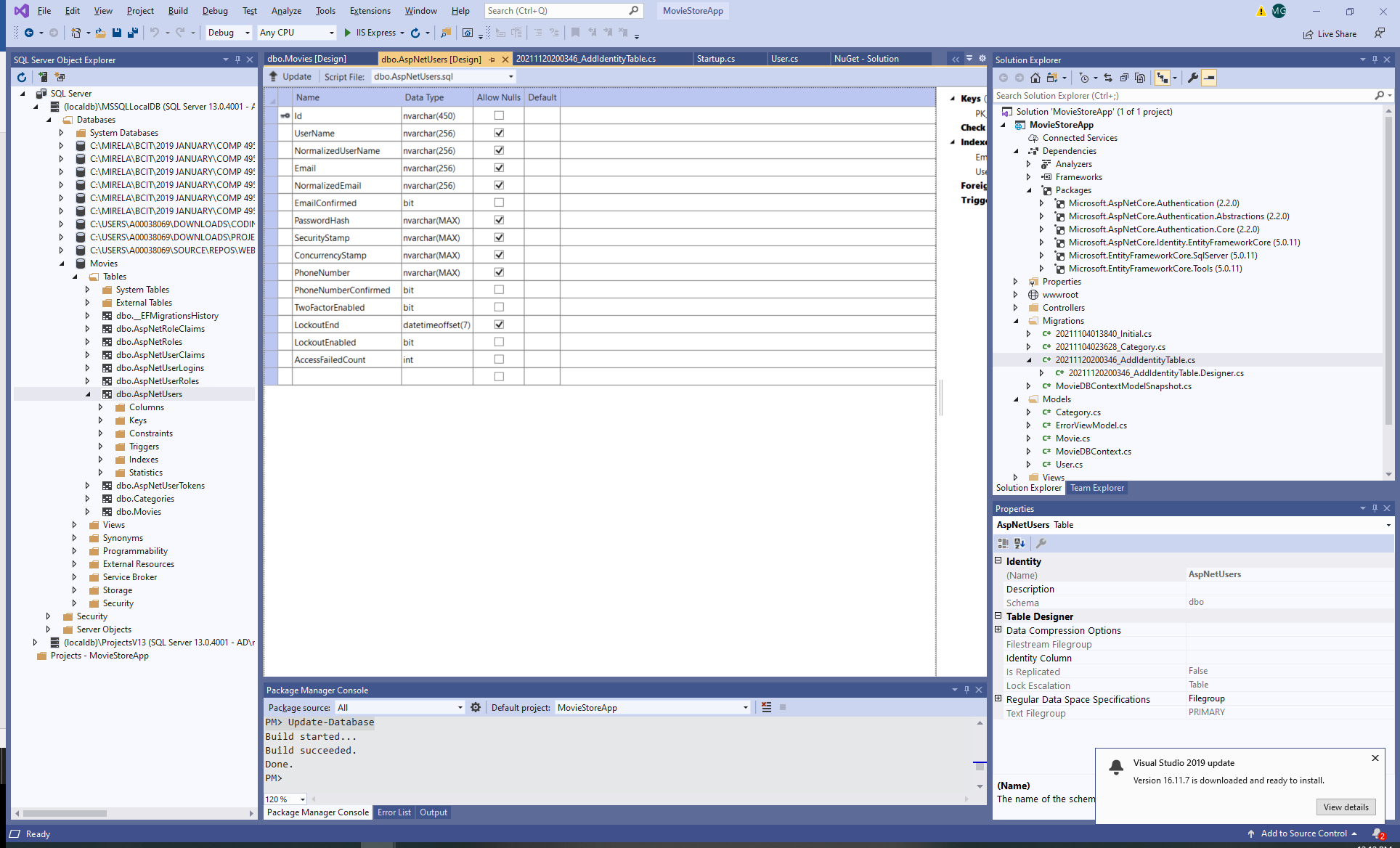




1. Observe the new DB migrations:

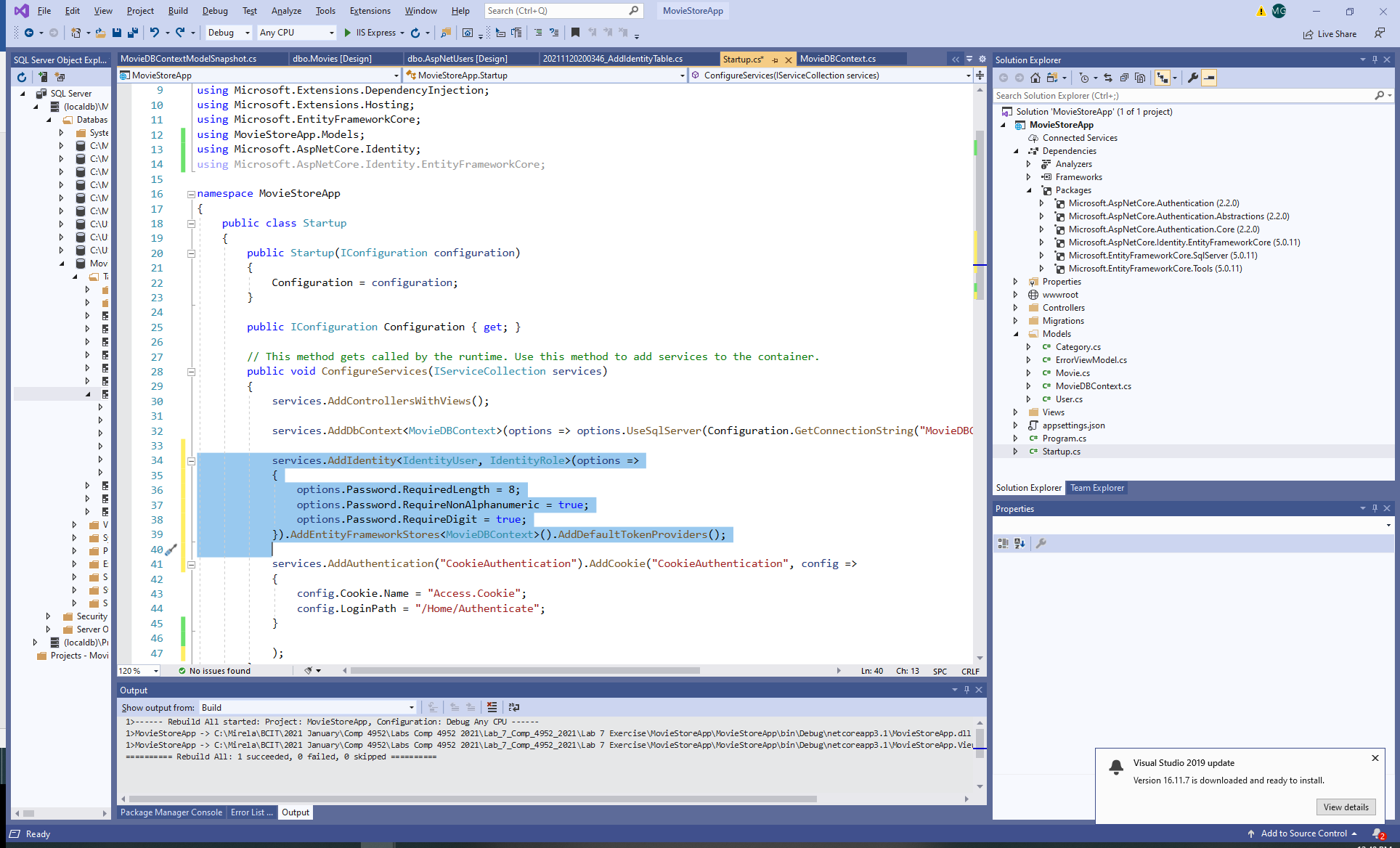


1. Observe the new created tables: user, logins, claims, roles, etc.



1. Configure middleware to allow authorization and authentication

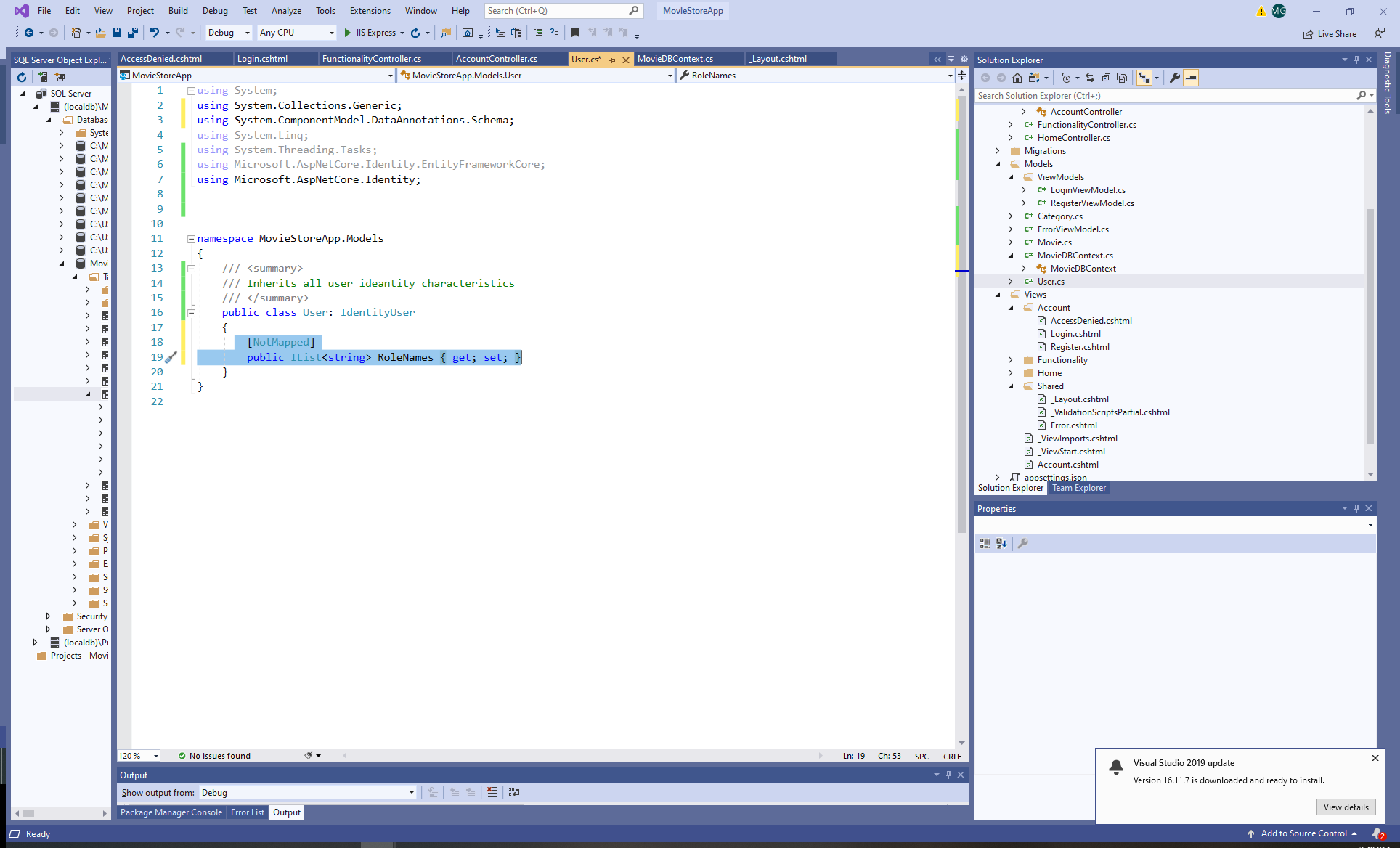
In the Startup.cs add the Identity namespaces and the password identity service and set up the desired rules. Make sure that this service is included before the AddAuthentication service.



1. Make sure that you include the use of authorization and authentication (We already included these in the previous assignment).

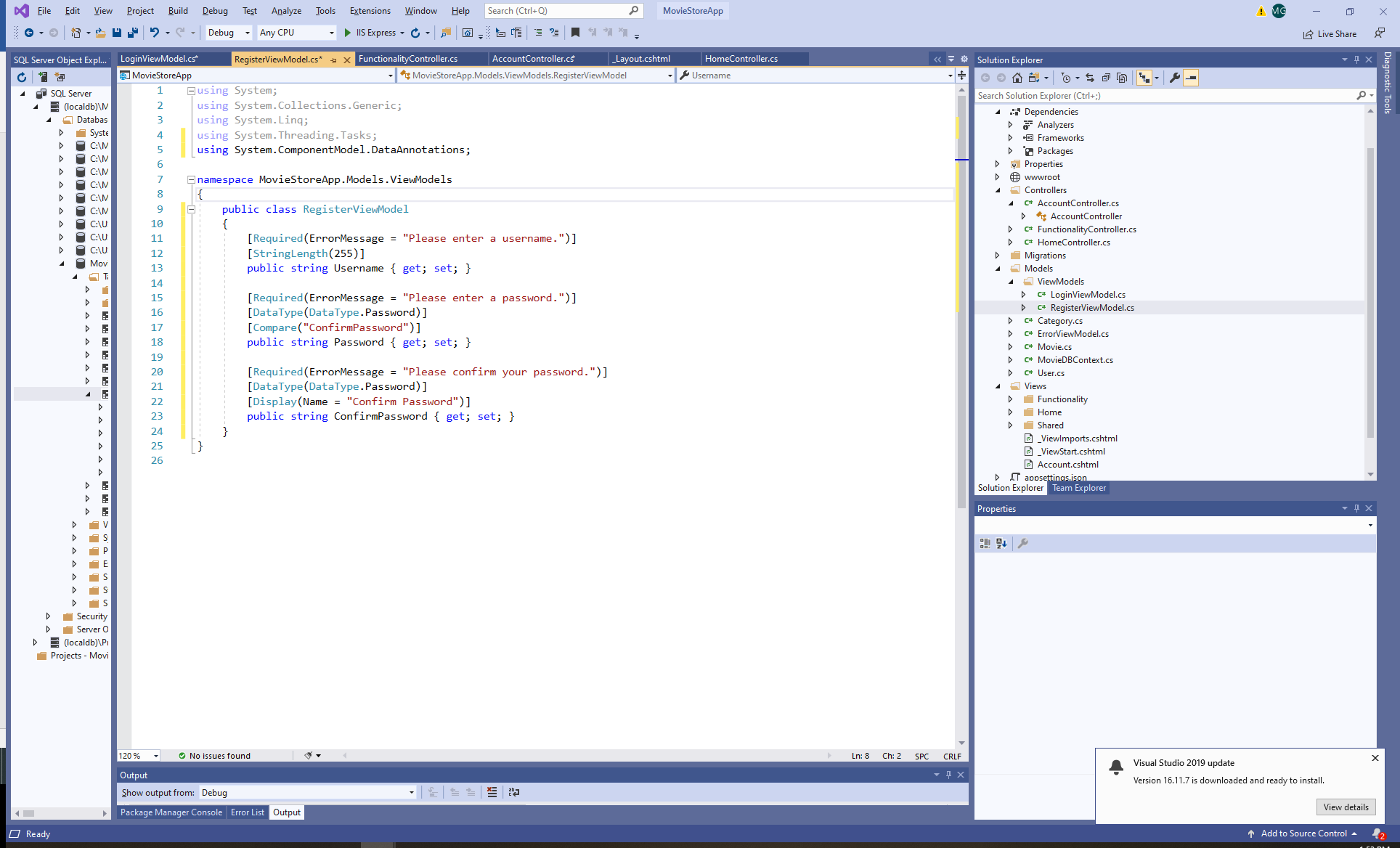


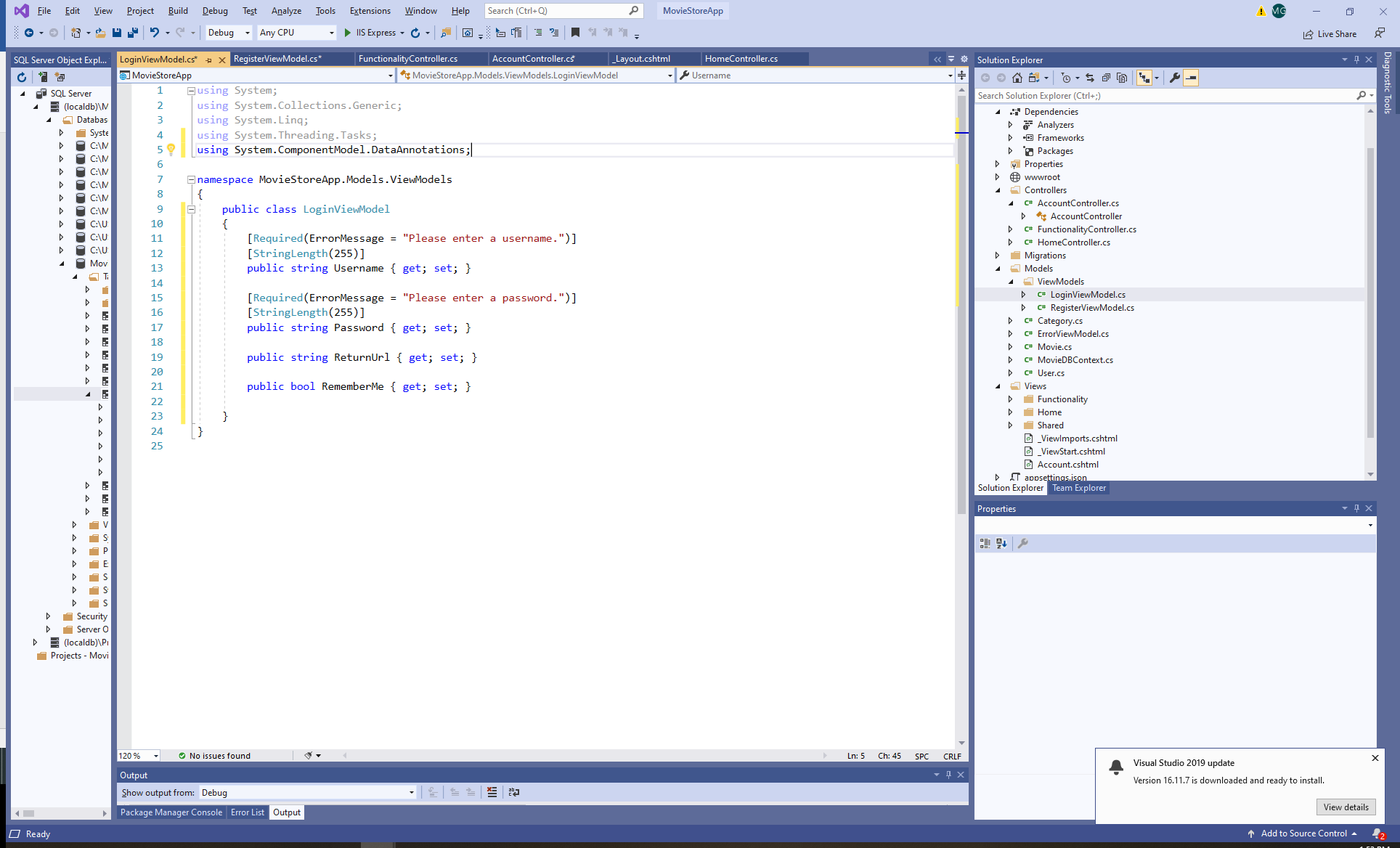
1. Update the User Model:



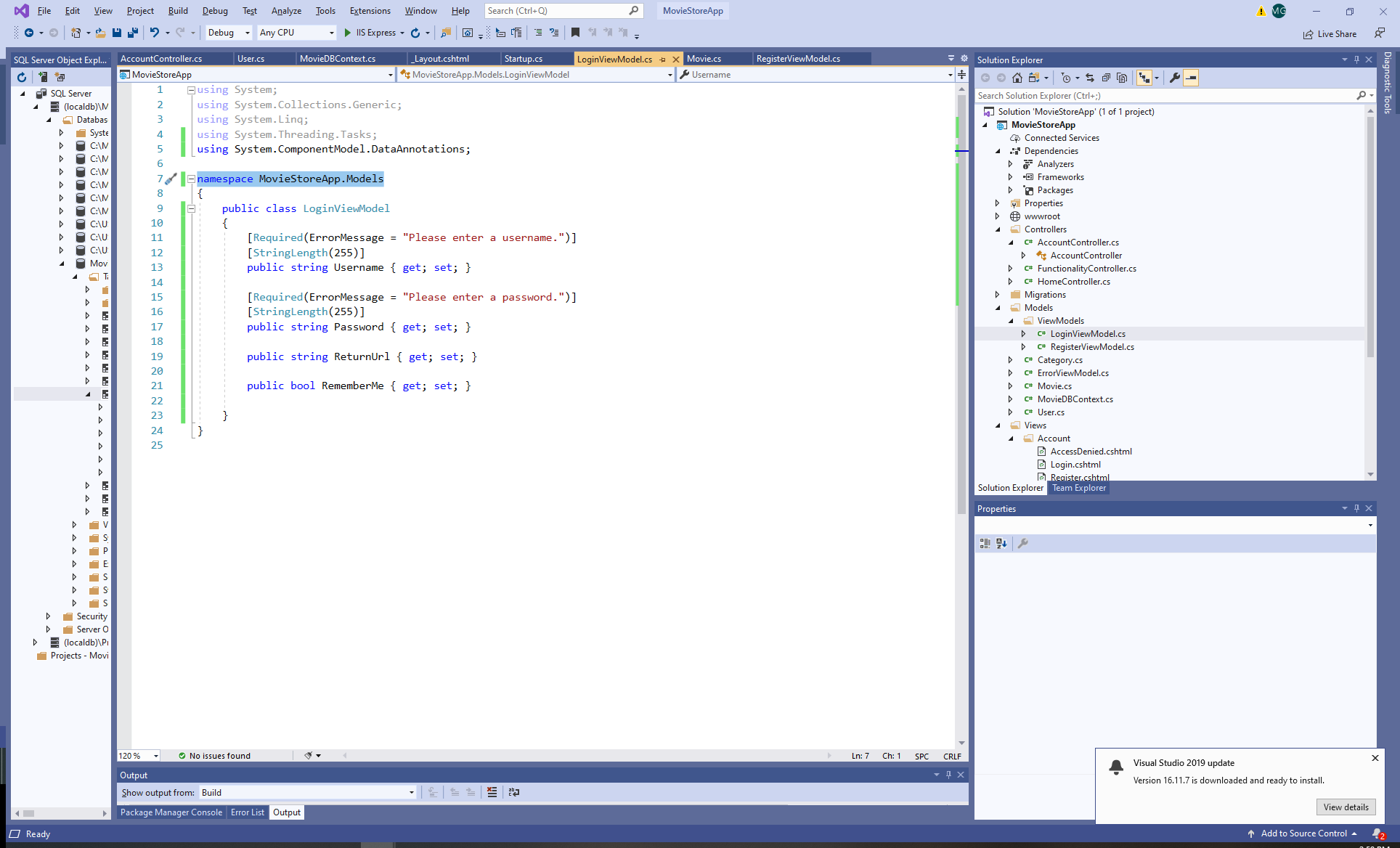
1. Add models for Authentication

Add a ViewModel folder (for better organization) and add ViewModels Login and Register:



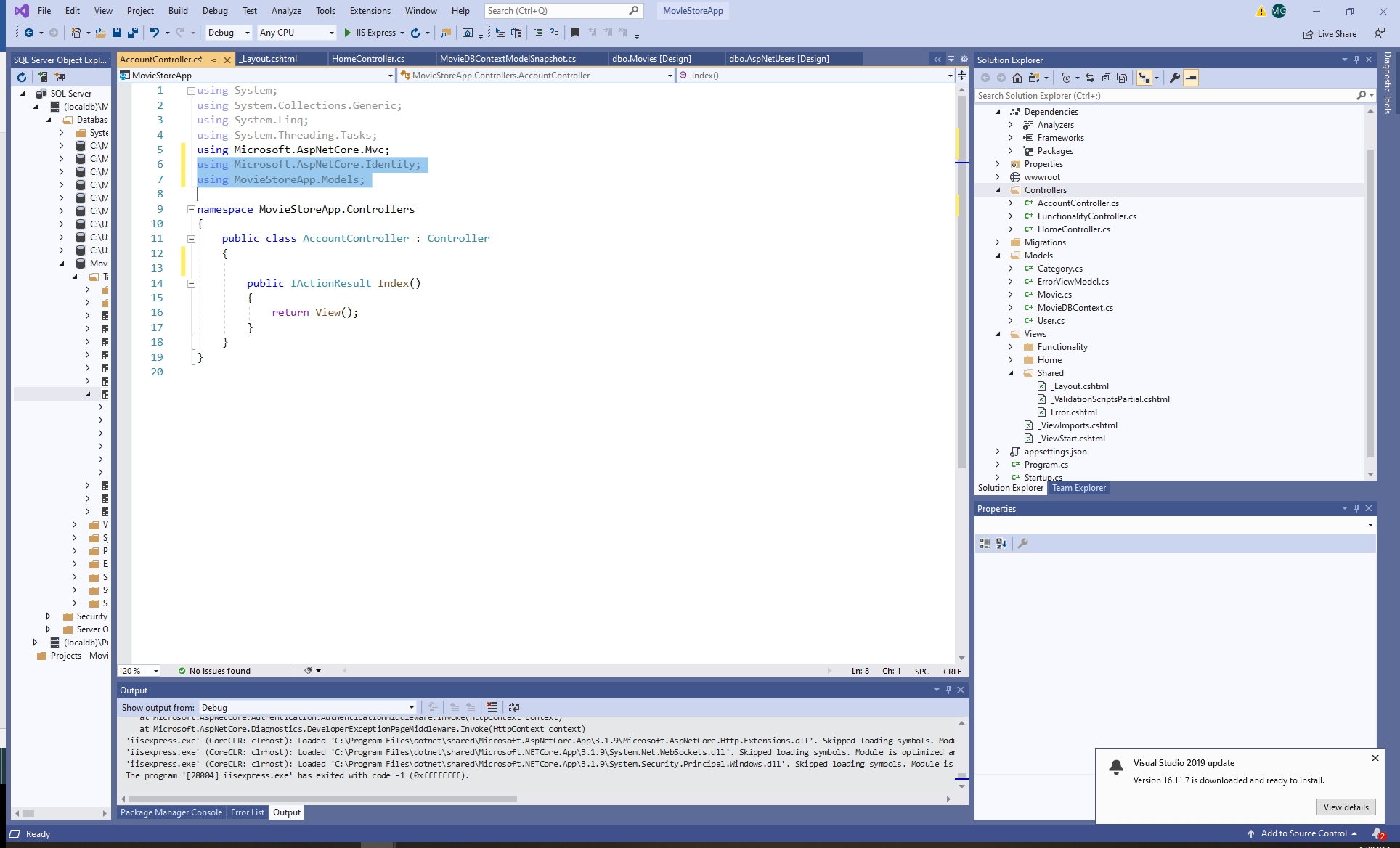


Make sure that you delete the .ViewModel from the name space (or you create the views first and move them in a folder later):

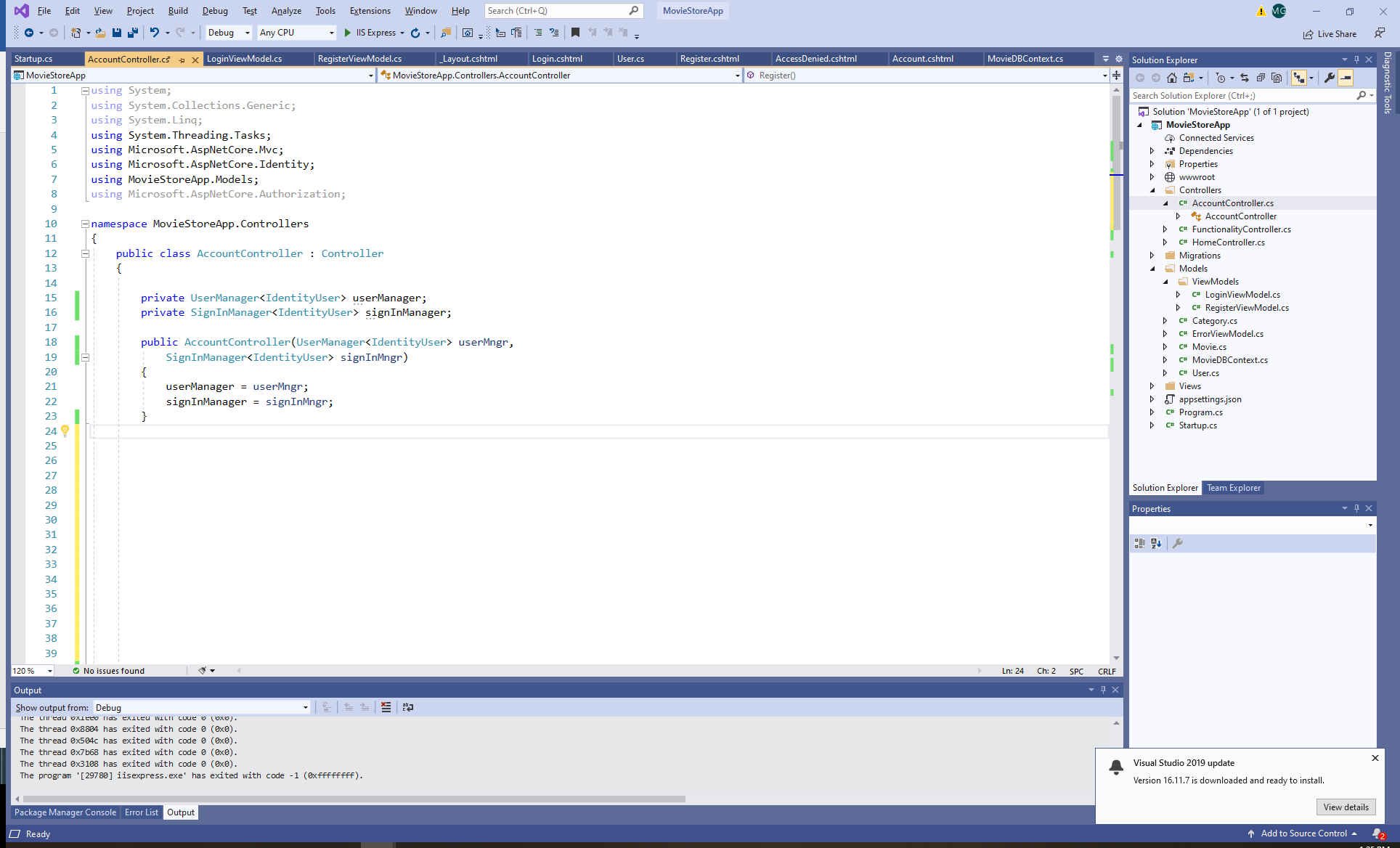




1. Add a Controller Account

Add a new Controller Account to allow the authentication functionality. Add the required namespaces. 

1. Add functionality to the Account Controller. Implement a constructor:



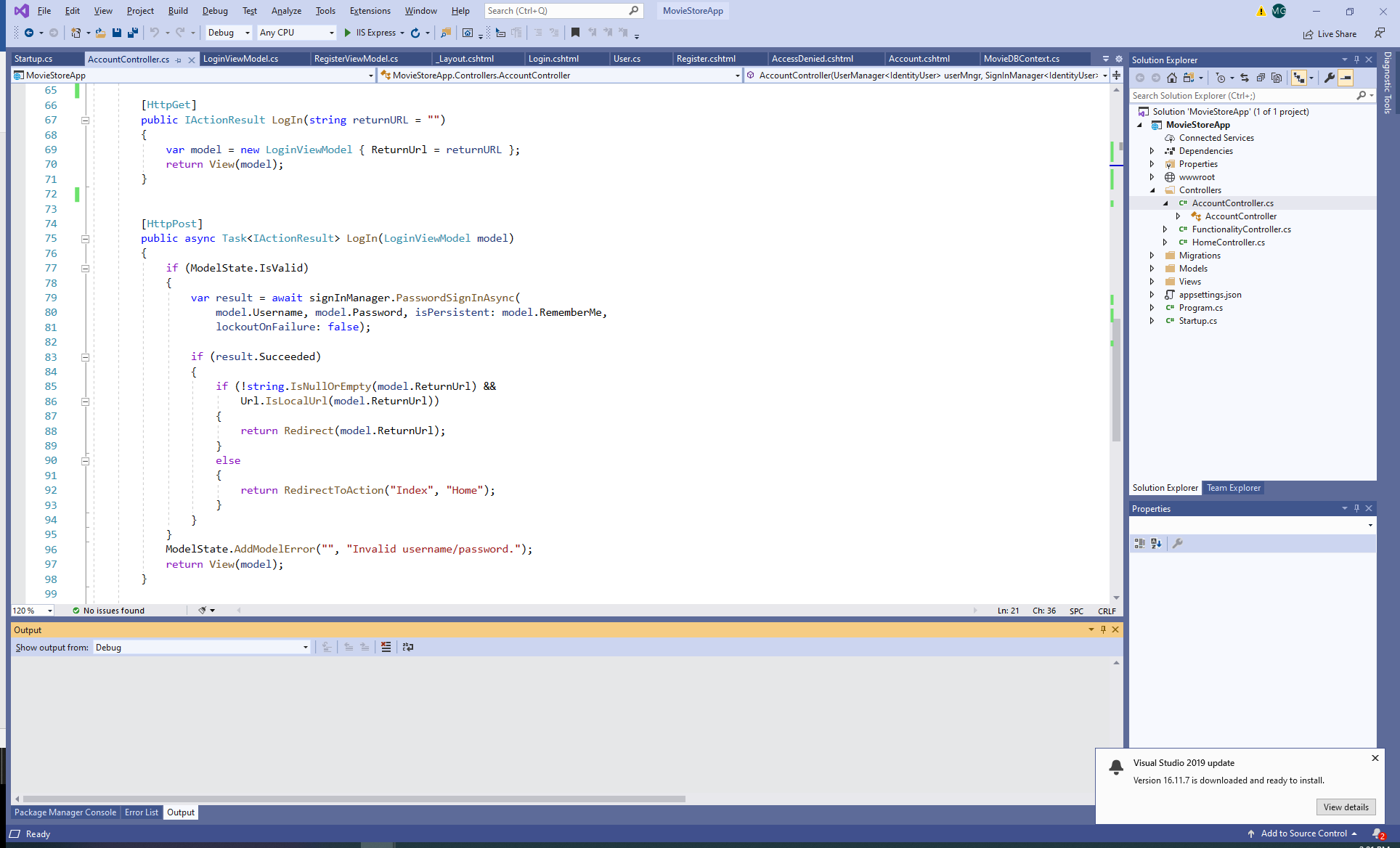
1. Add methods for Login(), Register(), Logout(), AccessDenied():

Each method is implemented for Get and Post.

For HttpGet: The methods are implemented to return an IActionResult.

For HttpPost: The methods are implementing an asynchronous operation that returns the result of an action method.

Example for Login:



Here is the full code for the Account Controller:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Identity;

using MovieStoreApp.Models;

using Microsoft.AspNetCore.Authorization;

namespace MovieStoreApp.Controllers

{

public class AccountController : Controller

{

private UserManager<IdentityUser> userManager;

private SignInManager<IdentityUser> signInManager;

public AccountController(UserManager<IdentityUser> userMngr,

SignInManager<IdentityUser> signInMngr)

{

userManager = userMngr;

signInManager = signInMngr;

}

[HttpGet]

public IActionResult Register()

{

return View();

}

[HttpPost]

public async Task<IActionResult> Register(RegisterViewModel model)

{

if (ModelState.IsValid)

{

var user = new User { UserName = model.Username };

var result = await userManager.CreateAsync(user, model.Password);

if (result.Succeeded)

{

bool isPersistent = false;

await signInManager.SignInAsync(user, isPersistent);

return RedirectToAction("Index", "Home");

}

else

{

foreach (var error in result.Errors)

{

ModelState.AddModelError("", error.Description);

}

}

}

return View(model);

}

[HttpGet]

public async Task<IActionResult> Logout()

{

await signInManager.SignOutAsync();

return RedirectToAction("Index", "Home");

}

[HttpGet]

public IActionResult LogIn(string returnURL = "")

{

var model = new LoginViewModel { ReturnUrl = returnURL };

return View(model);

}

[HttpPost]

public async Task<IActionResult> LogIn(LoginViewModel model)

{

if (ModelState.IsValid)

{

var result = await signInManager.PasswordSignInAsync(

model.Username, model.Password, isPersistent: model.RememberMe,

lockoutOnFailure: false);

if (result.Succeeded)

{

if (!string.IsNullOrEmpty(model.ReturnUrl) &&

Url.IsLocalUrl(model.ReturnUrl))

{

return Redirect(model.ReturnUrl);

}

else

{

return RedirectToAction("Index", "Home");

}

}

}

ModelState.AddModelError("", "Invalid username/password.");

return View(model);

}

public ViewResult AccessDenied()

{

return View();

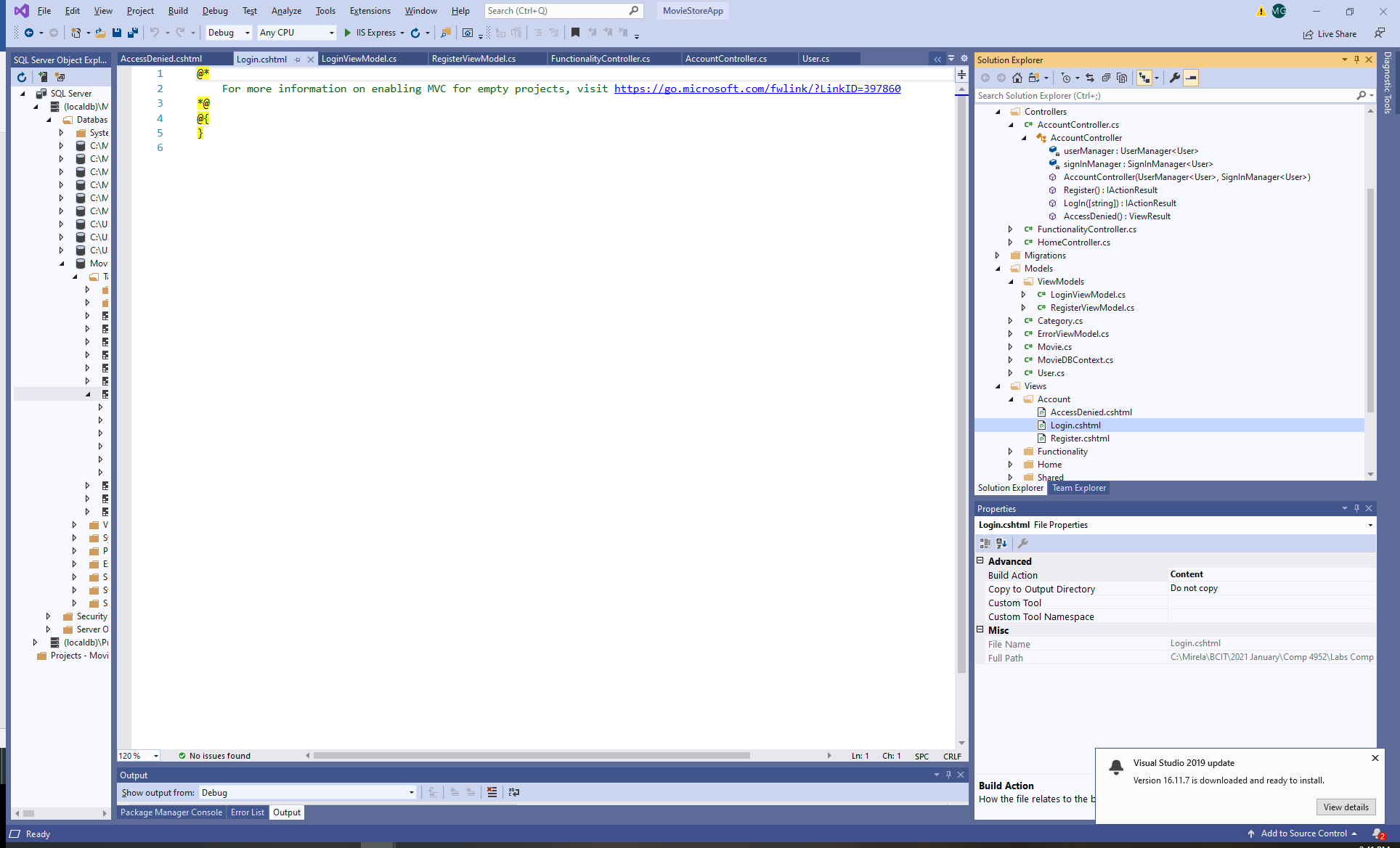
}

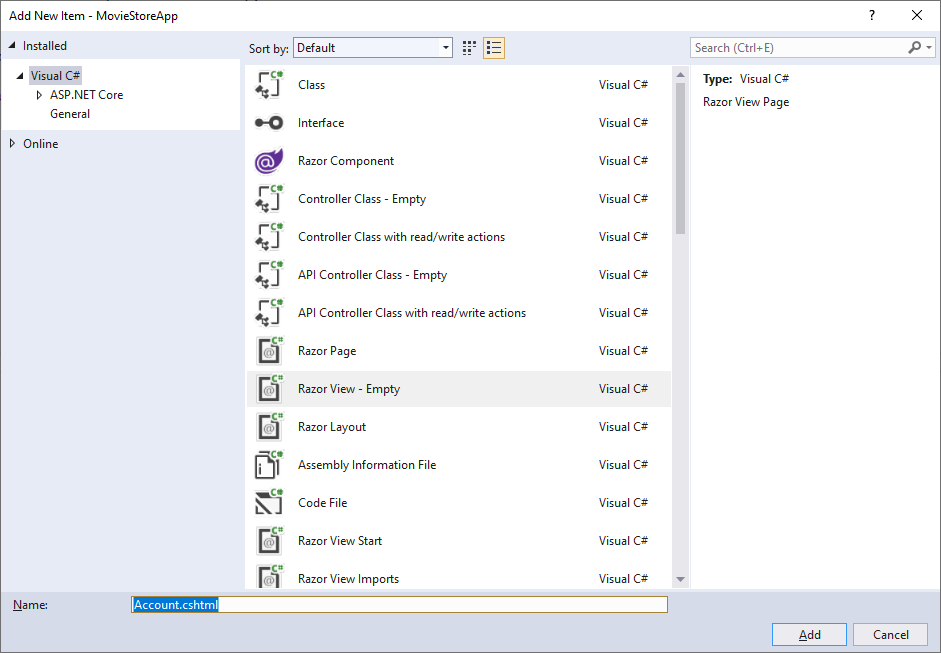
}

}

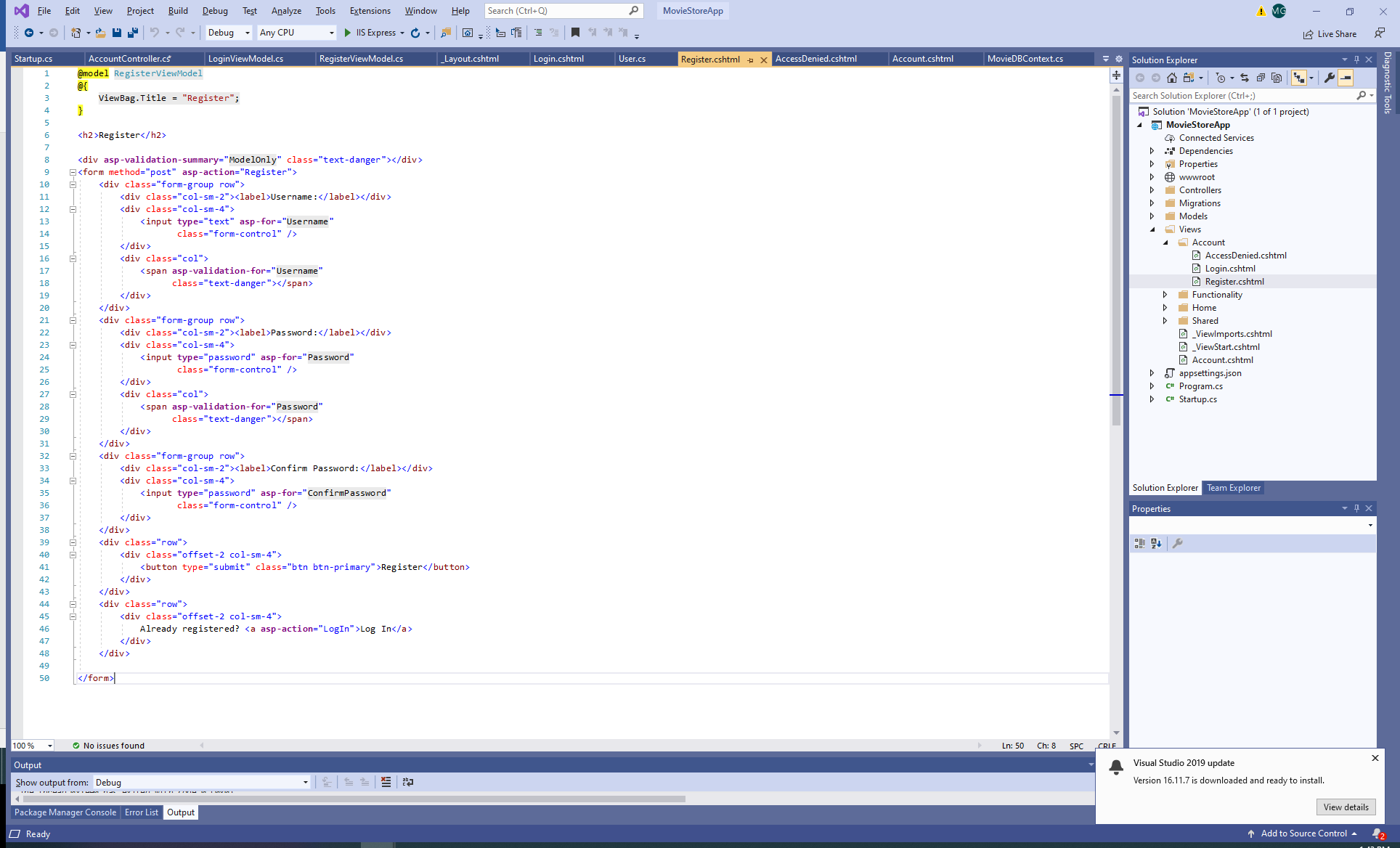
1. Add an Account View

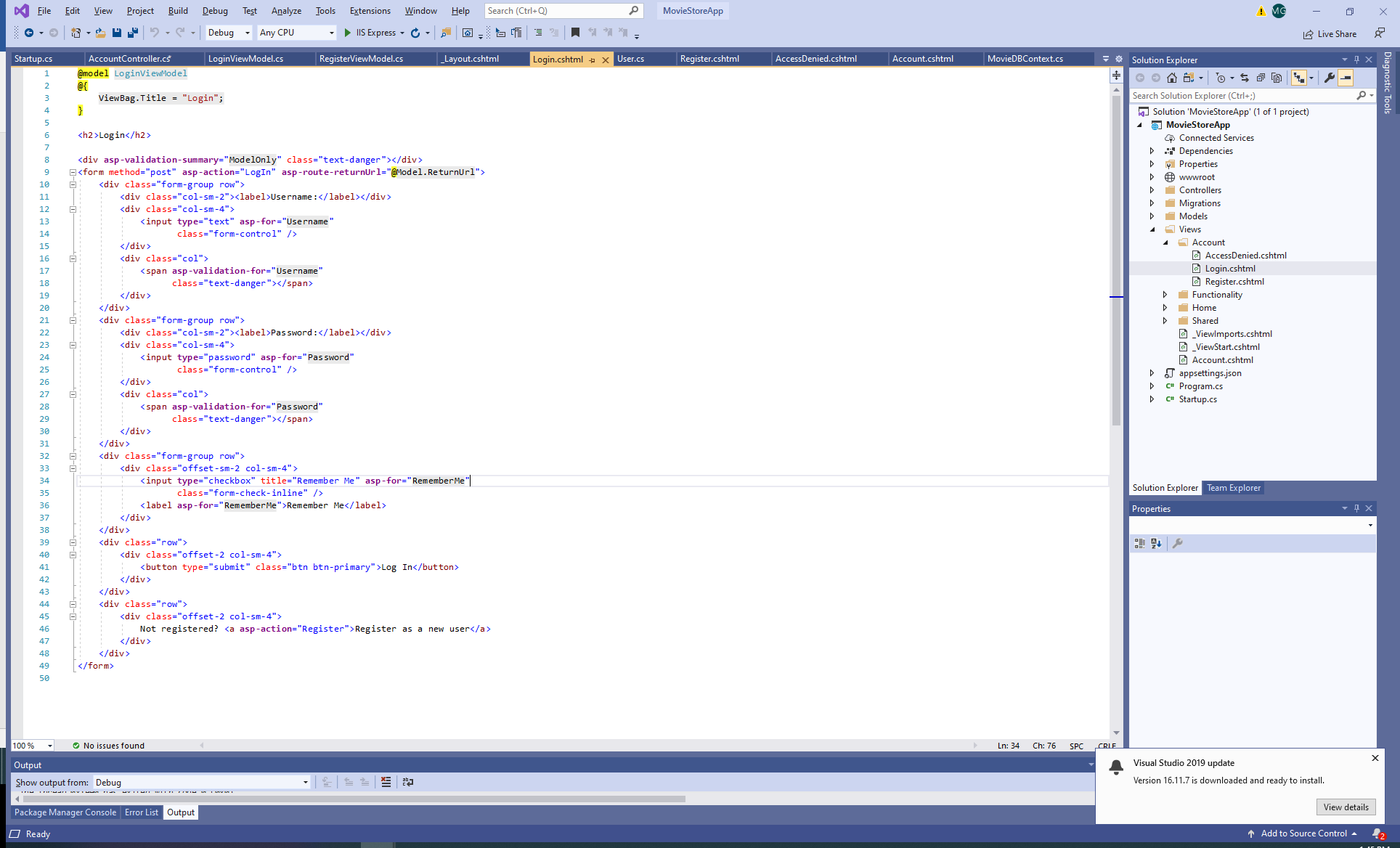
Add a folder Account and three Views: Login, Register, AccessDenied:





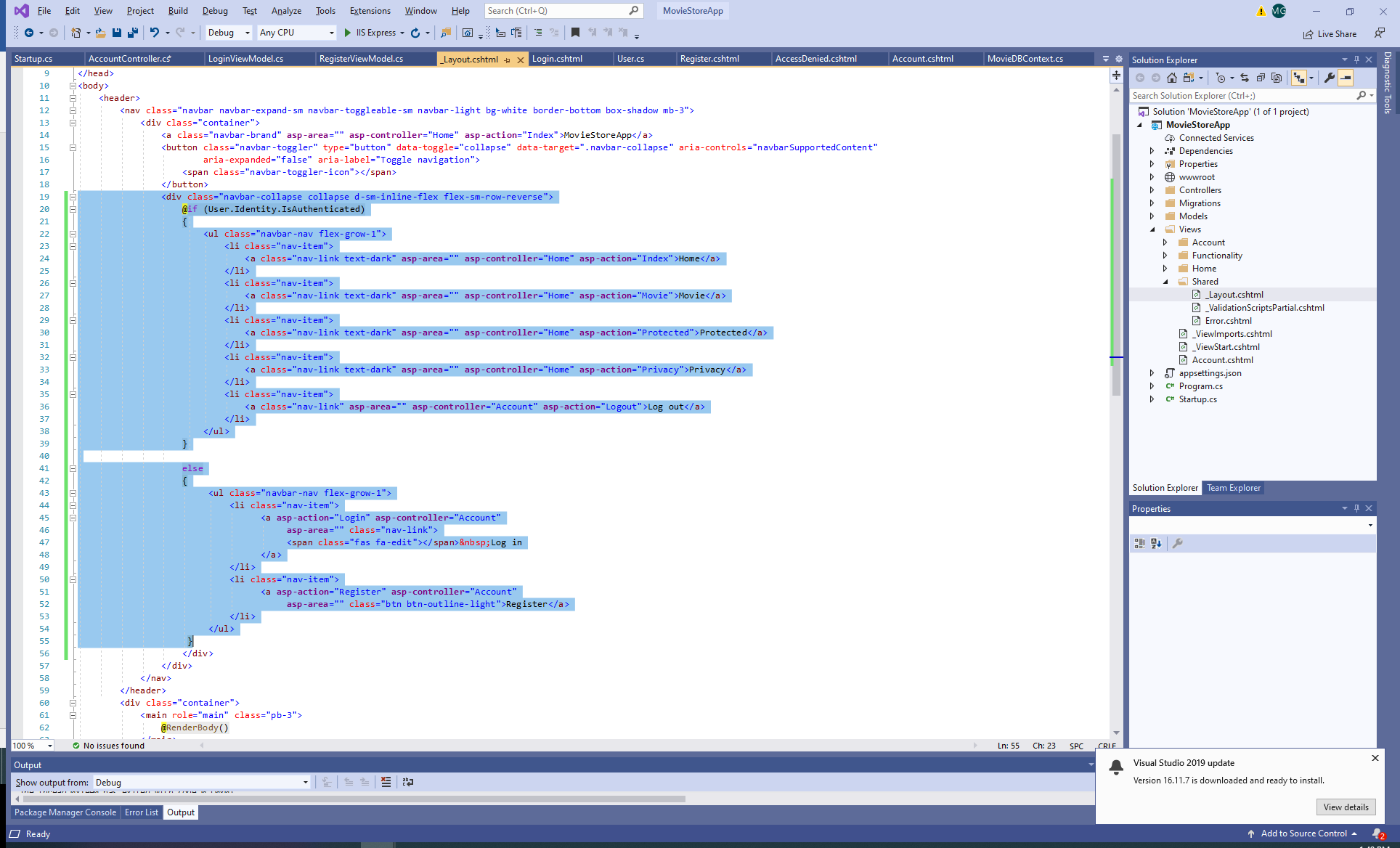
1. Add Views for Register, Login and AccesDenied:





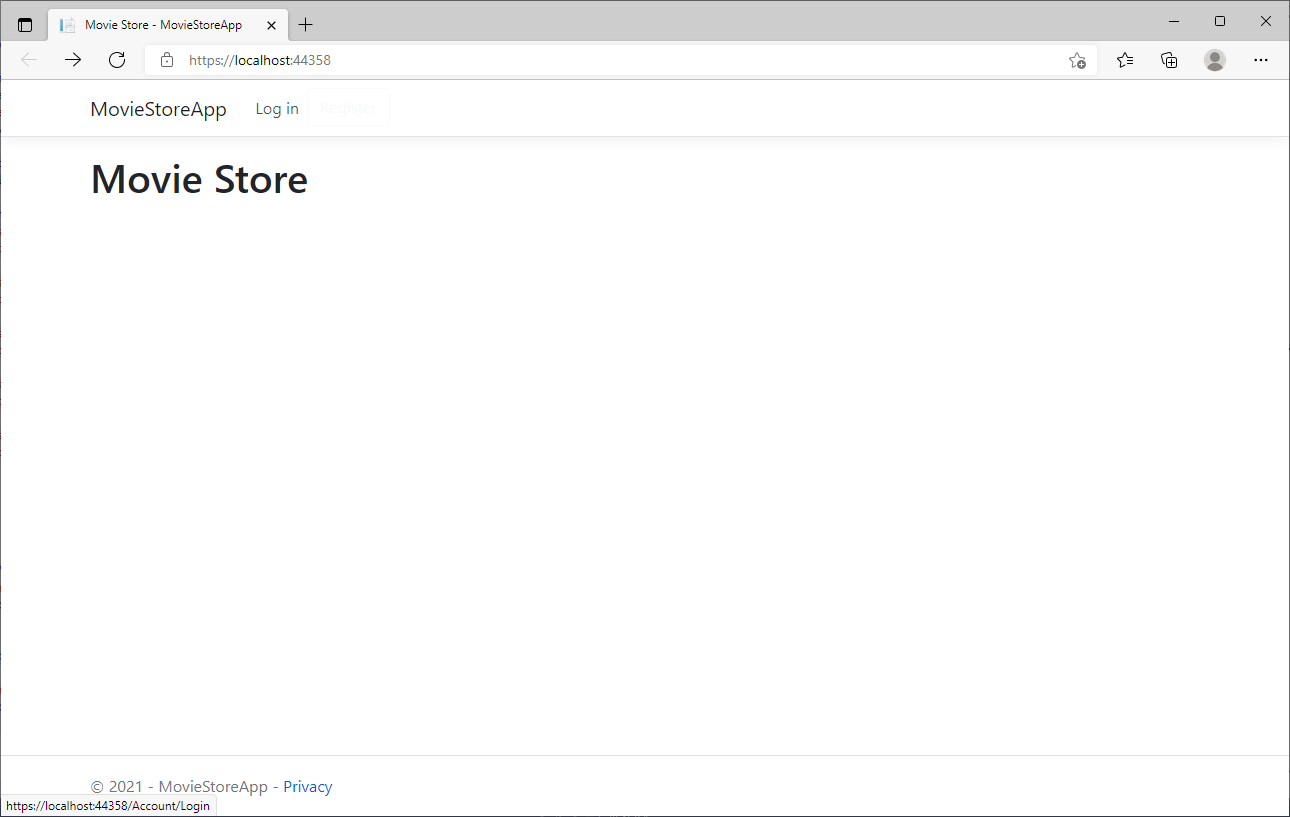


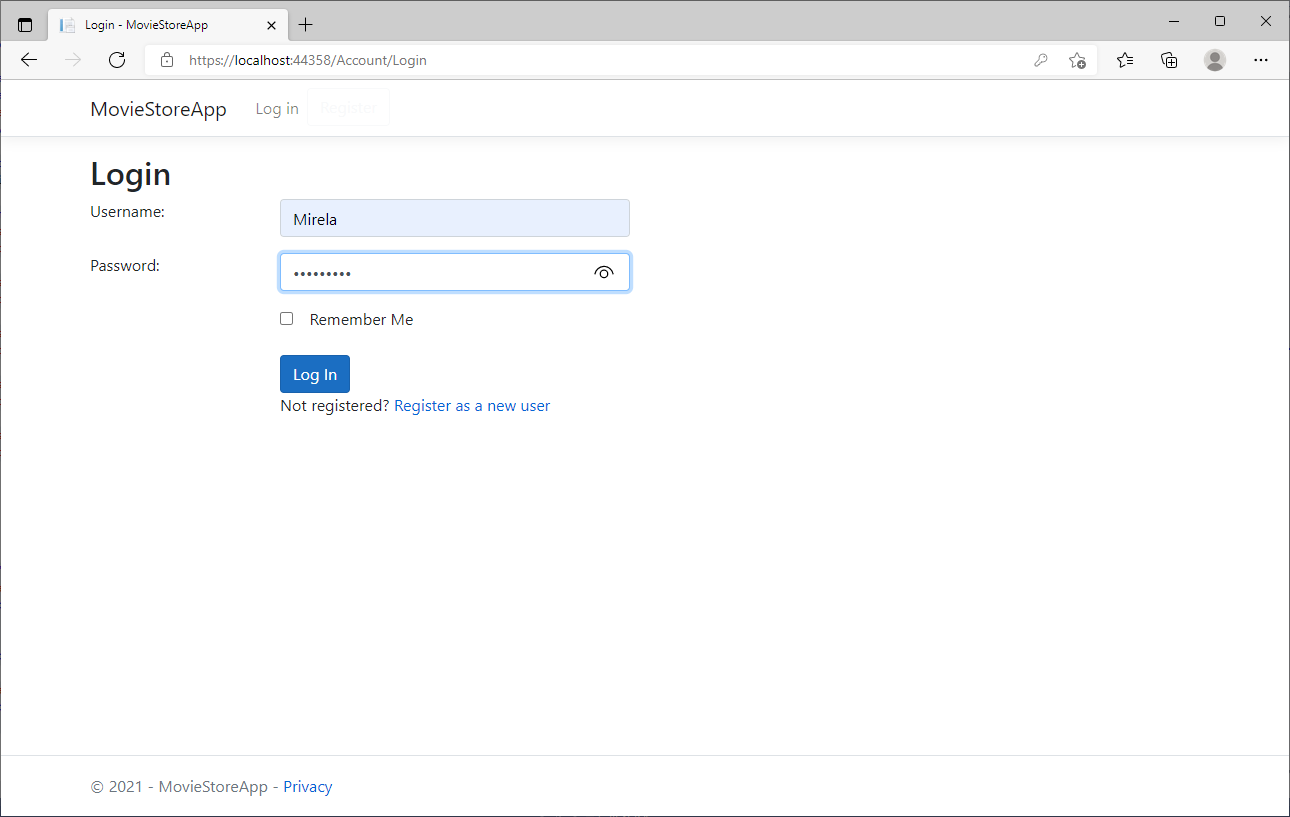
1. Modify the shared View \_Layout.cshtml by adding a check for authentication and Log In/Out and Registration buttons.

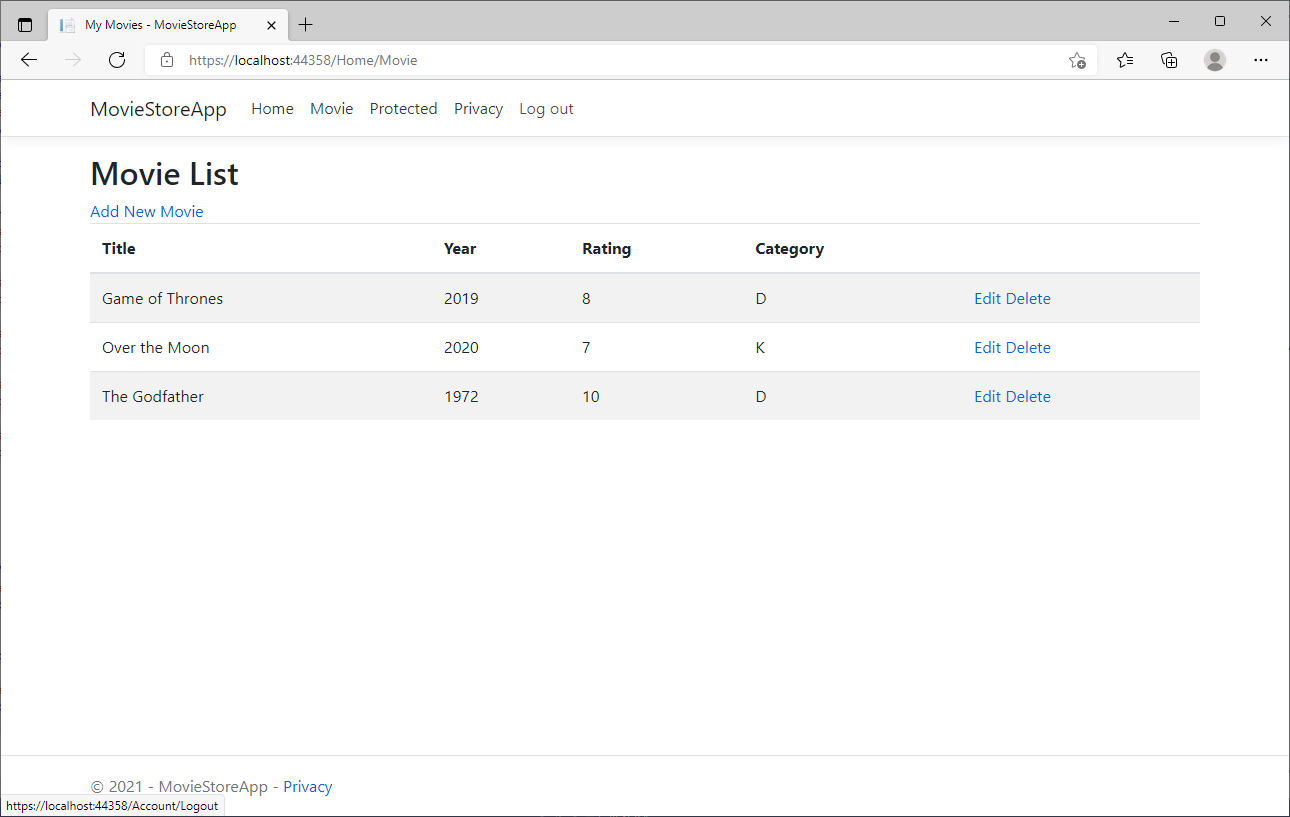


1. Run the application:

Now, the Login is visible on the navigation bar:







Register is on the navigation bar, too.

