

(420-PS4-AB)

Developing ASP .NET MVC Authentication and Authorization

Summer 2018

Outline

- Authentication in MVC
- Restricting Access
- Role Base Access
- Social Logins

Authentication in MVC

The Problem

- Restrict main capabilities to logged-in users.
 - Example: register customers, rental, returns, etc.
- Define specific roles to a groups or users to perform administrative tasks.
 - Example: Store manager has capabilities to add, delete or update the medias.

Authentication Options

New ASP.NET Project - WebApplication4

Select a template:

ASP.NET 4.5.2 Templates

Empty Web Forms **MVC** Web API Single Page Application

Azure API App (Preview) Azure Mobile Service


ASP.NET 5 Templates


Get ASP.NET 5 RC

A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.
[Learn more](#)

Change Authentication

Authentication: **Individual User Accounts**

 **Microsoft Azure**

 ☐ Host in the cloud

Web App

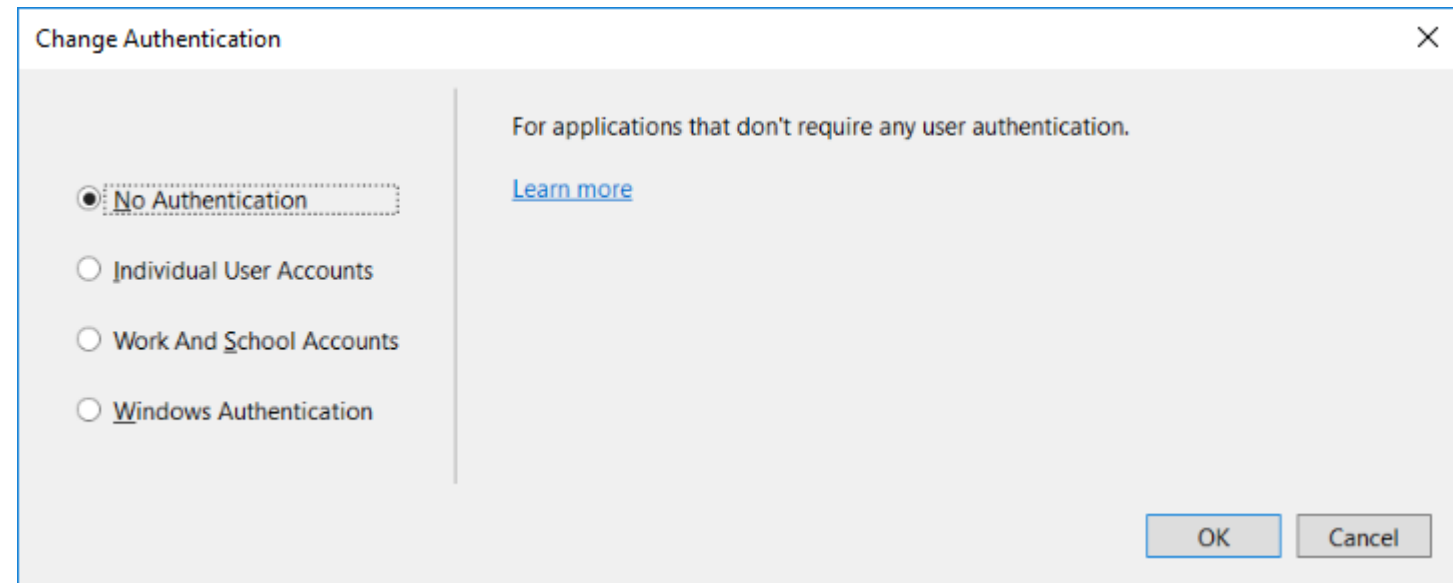
☐ Add unit tests

Test project name: WebApplication4.Tests

OK Cancel

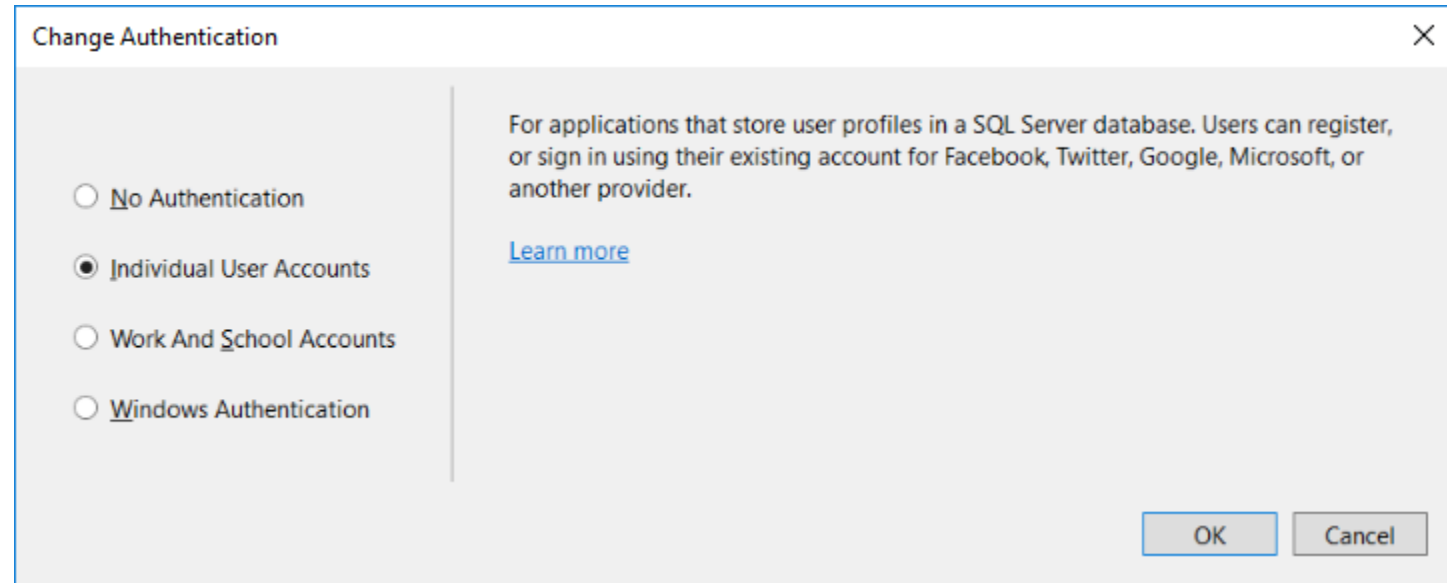
Option: 1- No Authentication

- Used in the application the need any authentication services.
- Such application are accessible to anonymous users.



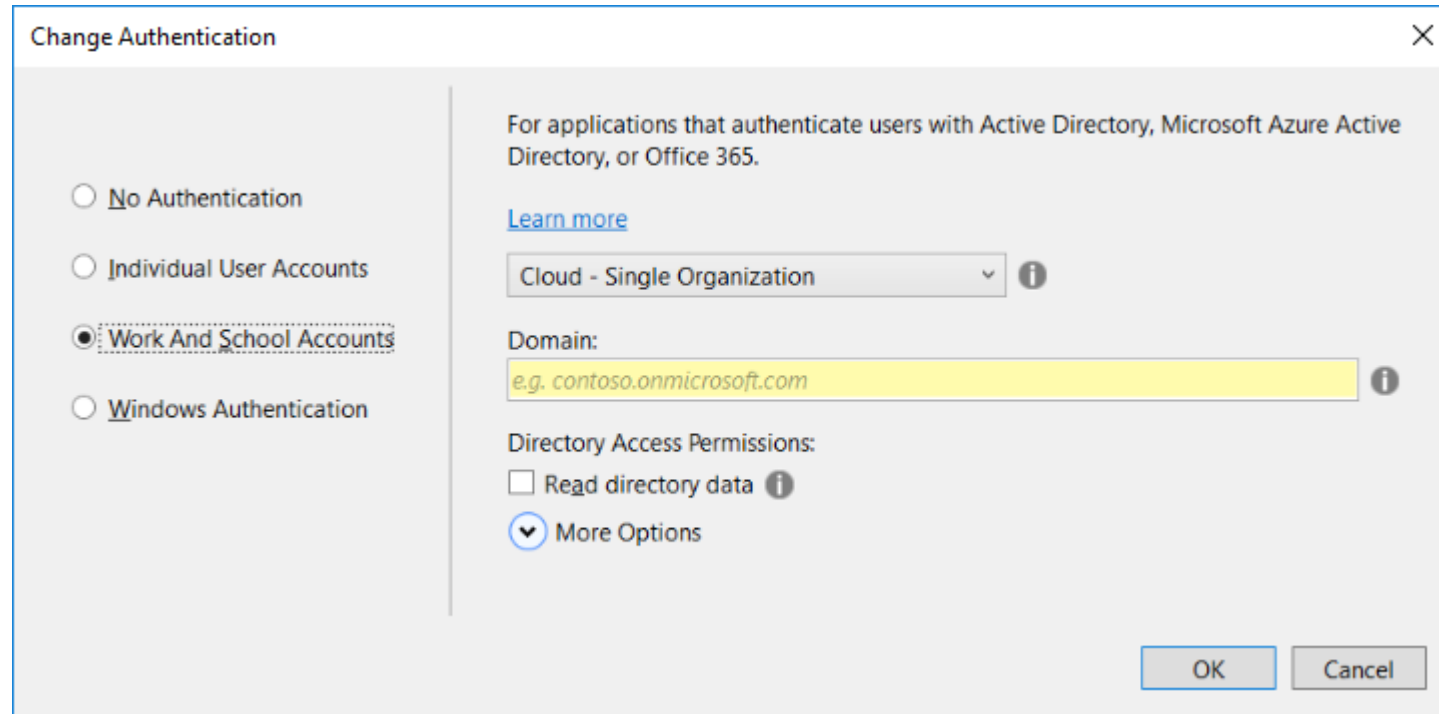
Option: 2- Individual User Accounts

- Suitable to for internet web applications where a form is required for the user to log in.
- Can use social logins.
 - Facebook, Google, ..etc.
- Most common.



Option: 3- Organizational Accounts

- Useful with single sign-on for internal and cloud apps using an active directory.



Change Authentication

For applications that authenticate users with Active Directory, Microsoft Azure Active Directory, or Office 365.

[Learn more](#)

Cloud - Single Organization ⓘ

Domain:
e.g. contoso.onmicrosoft.com ⓘ

Directory Access Permissions:

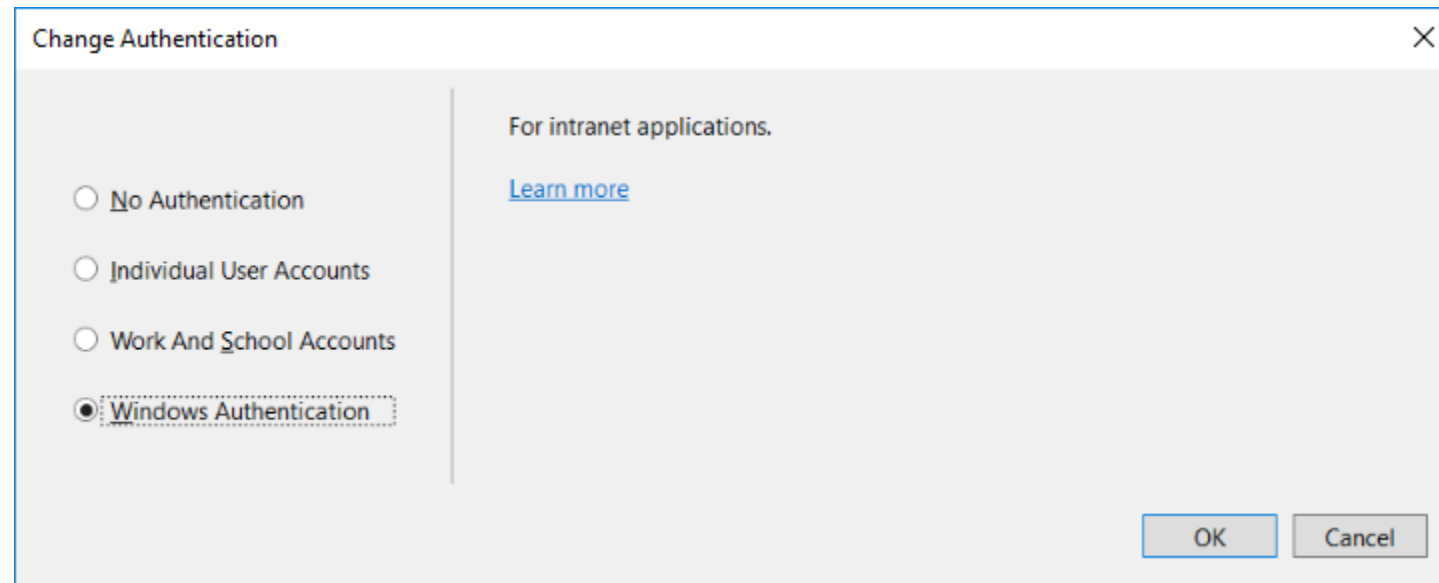
☐ Read directory data ⓘ

▼ More Options

OK Cancel

Option: 4- Windows Authentication

- Useful in intranet applications.
- When a user logs in to their PCs on the network, they will be authenticated to the web application automatically using their credentials.



Demo: Register / Log In



Log in.

Use a local account to log in.

Email

Password

☐ Remember me?

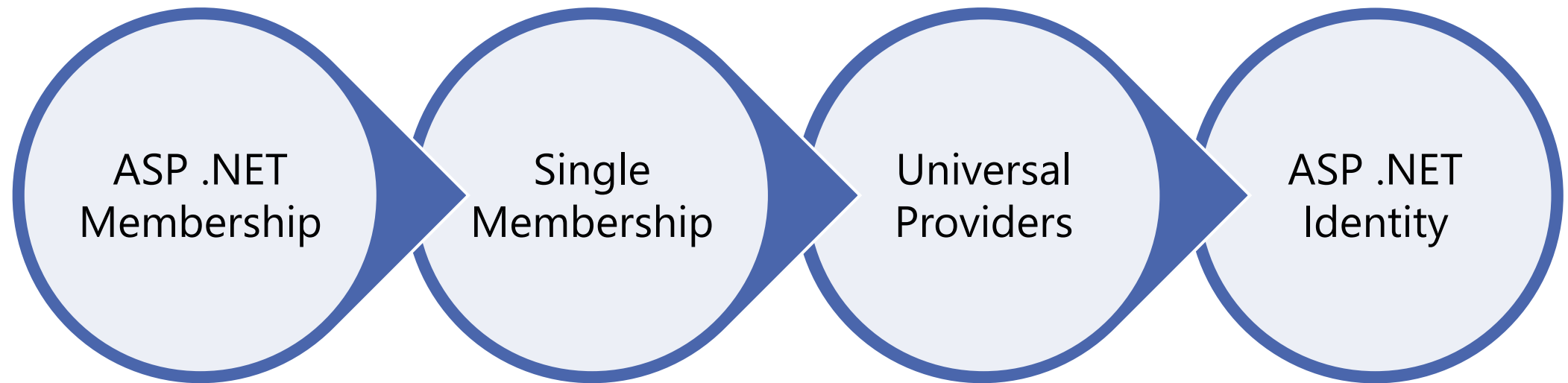
Log in

[Register as a new user](#)

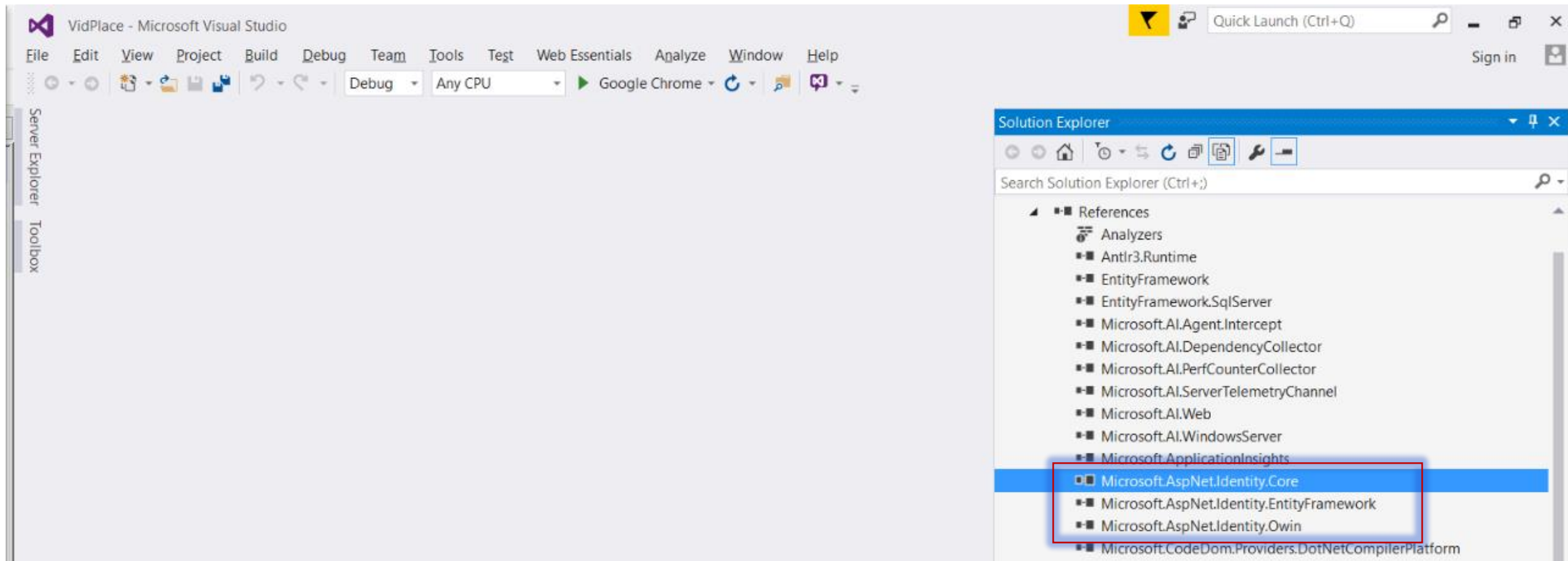
Use another service to log in.

There are no external authentication services configured.
See [this article](#) for details on setting up this ASP.NET
application to support logging in via external services.

ASP .NET Identity



Assemblies



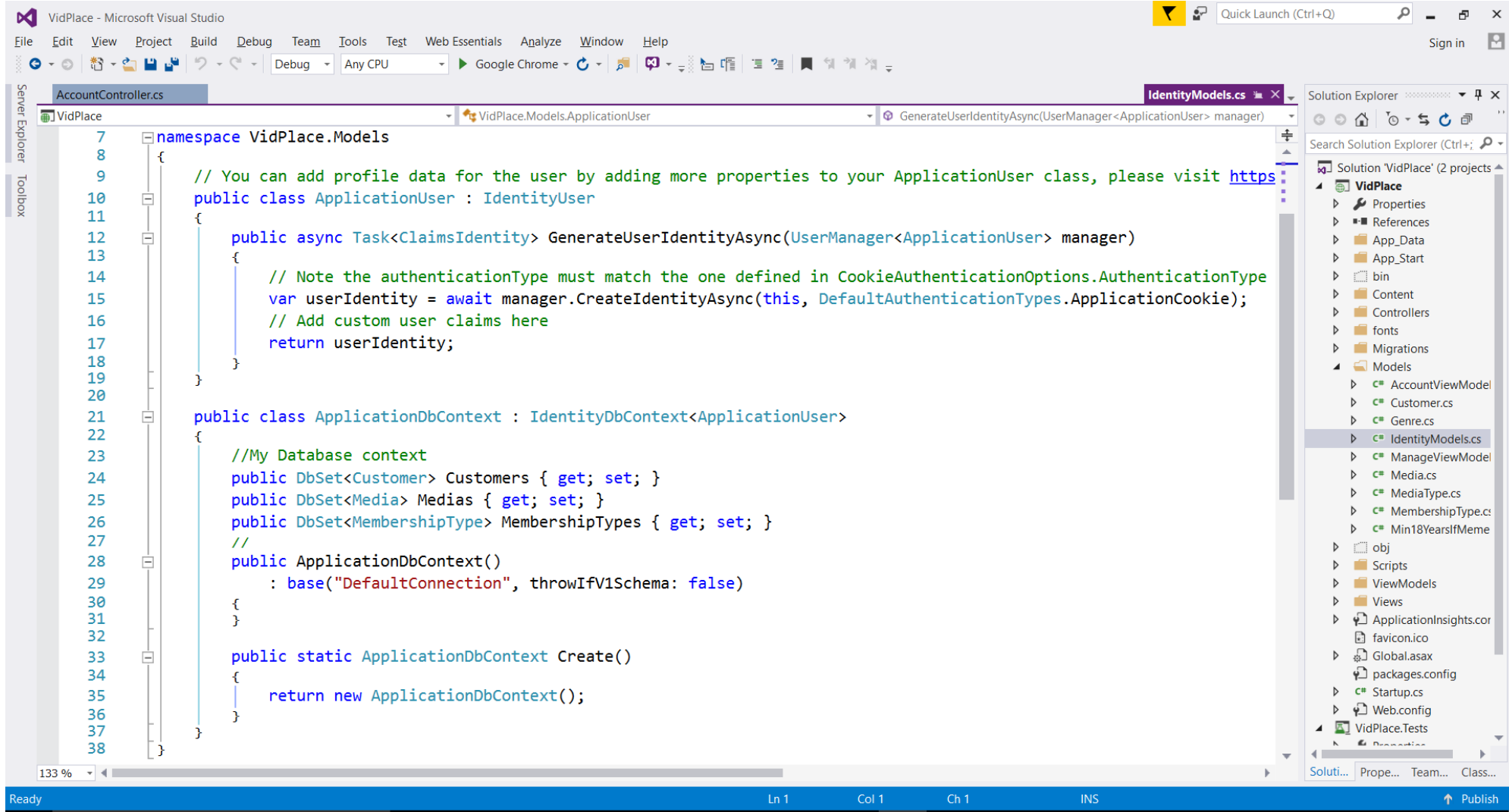
ASP .NET Identity – Technicality

- The architecture of the ASP .NET Identity has a number of domain classes. Example:
 - *IdentityUser*
 - *Role*
- An simple API is provided to work with these classes
 - *UserManager*
 - *RoleManager*
 - *SignInManager*

ASP .NET Identity – Technicality

- The API classes internally talk to another group of classes like *UserStore* and *RoleStore* which represent the "Persistent" store for ASP.NET identity.
- A *persistent store* is a repository in which managed objects may be stored.
- ASP .NET Identity provides an implementation of this persistent store using entity framework and a relational SQL database
 - Other implementations of a persistent store may be used.

Identity Model

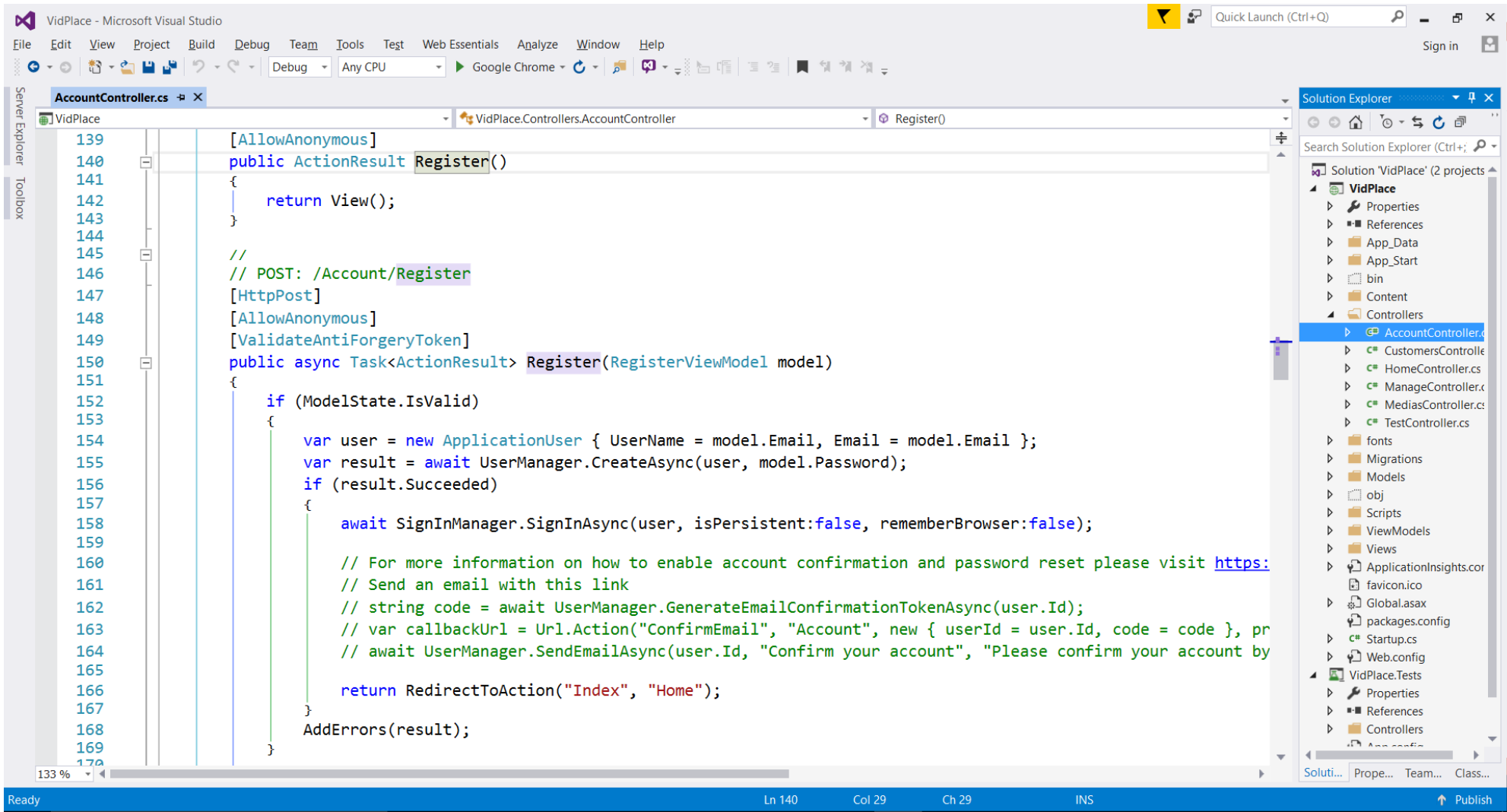


The screenshot displays the Microsoft Visual Studio IDE with the following components:

- File Explorer:** Shows the project structure for 'VidPlace' (2 projects). The 'Models' folder is expanded, showing files like 'AccountViewModel', 'Customer.cs', 'Genre.cs', 'IdentityModels.cs' (selected), 'ManageViewModel', 'Media.cs', 'MediaType.cs', 'MembershipType.cs', and 'Min18YearsIfMeme'.
- Code Editor:** Displays the 'IdentityModels.cs' file. The code defines the 'ApplicationUser' class and the 'ApplicationDbContext' class.

```
7 namespace VidPlace.Models
8 {
9     // You can add profile data for the user by adding more properties to your ApplicationUser class, please visit https
10    public class ApplicationUser : IdentityUser
11    {
12        public async Task<ClaimsIdentity> GenerateUserIdentityAsync(UserManager<ApplicationUser> manager)
13        {
14            // Note the authenticationType must match the one defined in CookieAuthenticationOptions.AuthenticationType
15            var userIdentity = await manager.CreateIdentityAsync(this, DefaultAuthenticationTypes.ApplicationCookie);
16            // Add custom user claims here
17            return userIdentity;
18        }
19    }
20
21    public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
22    {
23        //My Database context
24        public DbSet<Customer> Customers { get; set; }
25        public DbSet<Media> Medias { get; set; }
26        public DbSet<MembershipType> MembershipTypes { get; set; }
27        //
28        public ApplicationDbContext()
29            : base("DefaultConnection", throwIfV1Schema: false)
30        {
31        }
32
33        public static ApplicationDbContext Create()
34        {
35            return new ApplicationDbContext();
36        }
37    }
38 }
```
- Output Window:** Shows the status 'Ready'.
- StatusBar:** Displays 'Ln 1 Col 1 Ch 1 INS'.

Account Controller



```
139 [AllowAnonymous]
140 public ActionResult Register()
141 {
142     return View();
143 }
144
145 //
146 // POST: /Account/Register
147 [HttpPost]
148 [AllowAnonymous]
149 [ValidateAntiForgeryToken]
150 public async Task<ActionResult> Register(RegisterViewModel model)
151 {
152     if (ModelState.IsValid)
153     {
154         var user = new ApplicationUser { UserName = model.Email, Email = model.Email };
155         var result = await UserManager.CreateAsync(user, model.Password);
156         if (result.Succeeded)
157         {
158             await SignInManager.SignInAsync(user, isPersistent:false, rememberBrowser:false);
159
160             // For more information on how to enable account confirmation and password reset please visit https://go.microsoft.com/fwlink/?LinkID=320755
161             // Send an email with this link
162             // string code = await UserManager.GenerateEmailConfirmationTokenAsync(user.Id);
163             // var callbackUrl = Url.Action("ConfirmEmail", "Account", new { userId = user.Id, code = code }, protocol: Request.Url.Scheme);
164             // await UserManager.SendEmailAsync(user.Id, "Confirm your account", "Please confirm your account by clicking on this link");
165
166             return RedirectToAction("Index", "Home");
167         }
168         AddErrors(result);
169     }
170 }
```


Restricting Access

Restricting Access

- ASP .NET uses filter over actions to provide restrictions.
 - Filter are embedded on of an action, like attributes.
 - Once a filter is applied to an action, it will be called before and after the action or its result are executed.
- [Authorize]
 - When applied to an action, it will check if the user is logged in or not before execution.
 - If not, the user will redirected to the login page.

Demo: [Authorize] Filter

- Apply [Authorize] filter
 - On Customers Controller, Index action.
 - Above action.
 - Apply on the whole controller
 - Above class name.
 - Apply globally: will be applied to controllers in the web application and their actions.
 - Under App_Start: FilterConfig
`filters.Add(new AuthorizeAttribute());`
 - Good practice: if most of the application functionality required protections from anonymous users.

Global Authorization

- Highest level of restriction.
- Can not even reach home page.
- Making certain controllers/Actions accessible to anonymous users:
 - Add filter [AllowAnonymous]

Demo: Anonymous Authorizations

- Add an anonymous authorization to all the home controller
 - But keep all other page will still need authorization.

Role Base Access

Restricting Specific Actions in Controllers

- Creating accounts with special permission to pursue specific tasks.
 - Create a special role in the application.
 - Very Important: Upon deployment of the application, a user should be created and assigned to this role.
- Example:
 - Restrict media management operations to store manager.
 - A store manager can start adding media and then delegate other tasks to the staff.

Create a "Guest" Account

- Create a guest account:
 - Use the "Register" link to create a guest account
 - Guest@vidplace.com

Creating Roles

Account Controller → Register action

- Create a role store.
 - Create a role manager
 - Create the required role
 - Assign role to the user.
- One time process, to create initial admin.

```
var roleStore = new RoleStore<IdentityRole>(new ApplicationDbContext());  
var roleManager = new RoleManager<IdentityRole>(roleStore);  
await roleManager.CreateAsync(new IdentityRole("CanManageMedia"));  
await UserManager.AddToRoleAsync(user.Id, "CanManageMedia");
```

Create an "Admin" account

- Create a new account for an admin
 - Use the "Register" link to create an admin account
 - Admin@vidplace.com
- Disable the role code we added.

Seeding Users and Roles

- To have the default user accounts and role be deployed with developed web application:
 - Populate the database with the account using a migration.
- Create a new migration seedUsers
 - Export the records in the following tables:
 - AspNetUsers
 - AspNetRoles
 - AspNetUseRoles
 - Hint: use the Script
- Using migrations on other databases (testing or production) will produce the same exact setup.

Other Seeding Options

3. Call the new method from the `Seed` method:

```
protected override void Seed(ContactManager.Models.ApplicationDbContext context)
{
    AddUserAndRole(context);
    context.Contacts.AddOrUpdate(p => p.Name,
    // Code removed for brevity
}
```

The following images shows the changes to `Seed` method:

```
internal sealed class Configuration : DbMigrationsConfiguration<ContactManager>
{
    public Configuration()
    {
        AutomaticMigrationsEnabled = false;
    }

    bool AddUserAndRole(ContactManager.Models.ApplicationDbContext context)
    {
```

Demo: Seed vidPlace with a Guest and Admin Accounts

- Create a guest account.
- Create a role "CanManageMedia"
 - Create Admin account.

Using Roles

- Special roles have special access privileges in some views.
 - All based on the business logic.
- Implementation options:
 - Show/Hide different elements of view depending on the user's privileges.
 - Complex views will have a lot of cases → a lot of "if" statements.
 - Create an entirely new view for user with less or different privileges.

Demo: Applying Roles.

- Create a new view for regular staff to only have read only access on the media index view.
- Add logic to index action (Media) using
`User.IsInRole("CanManageMedia")`
 - Display all list if admin
 - Display read-only list if guest.

Security Hole

- Is UserRole enough?
 - Try URL address to access the "New" action in Media
- Solution
 - Apply on action the "Authorize" attribute with a role:
[Authorize(Roles="CanManageMedia")]
 - To add multiple roles to a single action add a comma ","
 - Need to apply of all needed actions in a controller.

Demo: Authorize with Role

- Apply Authorize attribute with role on the following actions on Medias:
 - New
 - Edit-Update
 - Delete

Demo: Remove Magic String

- Create a class in Models
 - Static class
 - Name: RoleName
 - Add public constant value

```
public const string  
    C anManageMedia = "CanManageMedia"
```

Social Logins

Social Logins



- Social logins are getting more popular with diverse social media services.
 - It is easier for users to create logins
 - Easier for developers to authenticate users.
- To use social logins:
 - Enable SSL
 - Register application with social media provider.

Demo: Link vidPlace with Facebook (1)

- Enable SSL in the project properties.
 - Copy SSL URL
- Add SSL URL in solution properties.
 - Web → Project URL
- Force application to use only SSL
 - App_Start → FilterConfig

```
filters.Add(new RequireHttpsAttribute());
```

Demo: Link vidPlace with Facebook (2)

- Go to developers.facebook.com
 - Register
 - Add new application
 - Log In Service
 - Add URL
 - Go to dashboard to get the details.
- In VS → App_Start → Start_Auth.cs
 - Go to Facebook Authorization and provide required information
`app.UseFacebookAuthentication(appId: "...", appSecret: "...");`
- Test add a user with Facebook.
 - Check the database: AspNetUserLogins table.

Q & A

