Table of Contents

[Authentication 2](#_Toc498940315)

[Individual User Accounts 2](#_Toc498940316)

[Model 3](#_Toc498940317)

[Database Context Class 3](#_Toc498940318)

[The Connection String 3](#_Toc498940319)

[Service Configuration 4](#_Toc498940320)

[Generated Database 5](#_Toc498940321)

[Register Users 5](#_Toc498940322)

[Log In/Out User 7](#_Toc498940323)

[Authorization 9](#_Toc498940324)

[[Authorize] 9](#_Toc498940325)

[[AllowAnonymous] 10](#_Toc498940326)

[Accessing User Information 10](#_Toc498940327)

[User Object 10](#_Toc498940328)

[References 12](#_Toc498940329)

# Authentication

(Rastogi, et al., 2017)

Authentication is “the process or action of verifying the identity of a user or process.”[[1]](#footnote-1)

In ASP.NET core, authentication is managed through the Identity system. The following dependencies are needed:

* Microsoft.AspNetCore.Identity.EntityFrameworkCore - Contains the required types to use Identity with Entity Framework Core.
* Microsoft.EntityFrameworkCore.SqlServer - Entity Framework Core is Microsoft's recommended data access technology for relational databases like SQL Server. For testing, you can use Microsoft.EntityFrameworkCore.InMemory.
* Microsoft.AspNetCore.Authentication.Cookies - Middleware that enables an app to use cookie-based authentication.

# Individual User Accounts

Individual user accounts store profiles in a SQL Server database. This is the traditional enter username and password use case. When this is selected during program creation, Visual Studio adds any needed dependencies and generates quite a bit of boilerplate starting code.

|  |
| --- |
|  |

## Model

The framework generates an ApplicationUser class derived from IdentityUser. You can add any additional profile properties here.

|  |
| --- |
| public class ApplicationUser : IdentityUser  {  // Any additional properties for your application here  } |

The IdentityUser class already has the following properties (plus others):

* Id, Email, PasswordHash, PhoneNumber, Roles, UserName

## Database Context Class

The framework generates an ApplicationDbContext class derived from IdentityDbContext<ApplicationUser>. You can rename this class to suit your application. You should add your application’s entity properties to this class.

|  |
| --- |
| public class ApplicationDbContext : IdentityDbContext<ApplicationUser>  {  public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)  : base(options)  {  }  protected override void OnModelCreating(ModelBuilder builder)  {  base.OnModelCreating(builder);  // Customize the ASP.NET Identity model and override the defaults if needed.  // For example, you can rename the ASP.NET Identity table names and more.  // Add your customizations after calling base.OnModelCreating(builder);  }  } |

## The Connection String

The framework generates the connection string. You should change the database name to suit your application.

|  |
| --- |
| {  "ConnectionStrings": {  "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=aspnet-W13Authentication-e0123608-efb0-47f1-870d-3adf23b0632e;Trusted\_Connection=True;MultipleActiveResultSets=true"  },  "Logging": {  "IncludeScopes": false,  "LogLevel": {  "Default": "Warning"  }  }  } |
| "ConnectionStrings": {  "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=MyDatabaseDb;Trusted\_Connection=True;MultipleActiveResultSets=true"  }, |

## Service Configuration

|  |
| --- |
| ***Startup.cs***  public void ConfigureServices(IServiceCollection services)  {  // Add framework services.  services.AddDbContext<ApplicationDbContext>(options =>  options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));  **services.AddIdentity<ApplicationUser, IdentityRole>()**  **.AddEntityFrameworkStores<ApplicationDbContext>()**  **.AddDefaultTokenProviders();**  services.AddMvc();  // Add application services.  services.AddTransient<IEmailSender, AuthMessageSender>();  services.AddTransient<ISmsSender, AuthMessageSender>();  }  public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)  {  loggerFactory.AddConsole(Configuration.GetSection("Logging"));  loggerFactory.AddDebug();  if (env.IsDevelopment())  {  app.UseDeveloperExceptionPage();  app.UseDatabaseErrorPage();  app.UseBrowserLink();  }  else  {  app.UseExceptionHandler("/Home/Error");  }  app.UseStaticFiles();  **app.UseIdentity();**  app.UseMvc(routes =>  {  routes.MapRoute(  name: "default",  template: "{controller=Home}/{action=Index}/{id?}");  });  } |

## Generated Database

|  |
| --- |
|  |

## Register Users

|  |
| --- |
|  |

|  |
| --- |
| public class AccountController : Controller  {  **private readonly UserManager<ApplicationUser> \_userManager;**  private readonly SignInManager<ApplicationUser> \_signInManager;  private readonly IEmailSender \_emailSender;  private readonly ISmsSender \_smsSender;  private readonly ILogger \_logger;  private readonly string \_externalCookieScheme; |
| …  [HttpGet]  [AllowAnonymous]  public IActionResult Register(string returnUrl = null)  {  ViewData["ReturnUrl"] = returnUrl;  return View();  }  [HttpPost]  [AllowAnonymous]  [ValidateAntiForgeryToken]  public async Task<IActionResult> Register(**RegisterViewModel model**, string returnUrl = null)  {  ViewData["ReturnUrl"] = returnUrl;  if (ModelState.IsValid)  {  **var user = new ApplicationUser { UserName = model.Email, Email = model.Email };**  **var result = await \_userManager.CreateAsync(user, model.Password);**  if (result.Succeeded)  {  await \_signInManager.SignInAsync(user, isPersistent: false);  \_logger.LogInformation(3, "User created a new account with password.");  return RedirectToLocal(returnUrl);  }  AddErrors(result);  }  // If we got this far, something failed, redisplay form  return View(model);  } |
| public class **RegisterViewModel**  {  [Required]  [EmailAddress]  [Display(Name = "Email")]  public string Email { get; set; }  [Required]  [StringLength(100, ErrorMessage = "The {0} must be at least {2} and at max {1} characters long.", MinimumLength = 6)]  [DataType(DataType.Password)]  [Display(Name = "Password")]  public string Password { get; set; }  [DataType(DataType.Password)]  [Display(Name = "Confirm password")]  [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]  public string ConfirmPassword { get; set; }  } |

|  |
| --- |
|  |
|  |

## Log In/Out User

|  |
| --- |
|  |
|  |

|  |
| --- |
| public class AccountController : Controller  {  private readonly UserManager<ApplicationUser> \_userManager;  **private readonly SignInManager<ApplicationUser> \_signInManager;**  private readonly IEmailSender \_emailSender;  private readonly ISmsSender \_smsSender;  private readonly ILogger \_logger;  private readonly string \_externalCookieScheme; |
| [HttpGet]  [AllowAnonymous]  public async Task<IActionResult> Login(string returnUrl = null)  {  // Clear the existing external cookie to ensure a clean login process  **await HttpContext.Authentication.SignOutAsync(\_externalCookieScheme);**  ViewData["ReturnUrl"] = returnUrl;  return View();  }  [HttpPost]  [AllowAnonymous]  [ValidateAntiForgeryToken]  public async Task<IActionResult> Login(**LoginViewModel model**, string returnUrl = null)  {  ViewData["ReturnUrl"] = returnUrl;  if (ModelState.IsValid)  {  **var result = await \_signInManager.PasswordSignInAsync(model.Email, model.Password, model.RememberMe, lockoutOnFailure: false);**  if (result.Succeeded)  {  \_logger.LogInformation(1, "User logged in.");  return RedirectToLocal(returnUrl);  }  if (result.RequiresTwoFactor)  {  return RedirectToAction(nameof(SendCode), new { ReturnUrl = returnUrl, RememberMe = model.RememberMe });  }  if (result.IsLockedOut)  {  \_logger.LogWarning(2, "User account locked out.");  return View("Lockout");  }  else  {  ModelState.AddModelError(string.Empty, "Invalid login attempt.");  return View(model);  }  }  // If we got this far, something failed, redisplay form  return View(model);  } |
| public class **LoginViewModel**  {  [Required]  [EmailAddress]  public string Email { get; set; }  [Required]  [DataType(DataType.Password)]  public string Password { get; set; }  [Display(Name = "Remember me?")]  public bool RememberMe { get; set; }  } |
| [HttpPost]  [ValidateAntiForgeryToken]  public async Task<IActionResult> Logout()  {  **await \_signInManager.SignOutAsync();**  \_logger.LogInformation(4, "User logged out.");  return RedirectToAction(nameof(HomeController.Index), "Home");  } |

# Authorization

(Unnamed, 2016)

Authorization refers to the process that determines what a user is able to do. For example, an administrative user is allowed to create a document library, add documents, edit documents, and delete them. A non-administrative user working with the library is only authorized to read the documents.

Restricting access to web resources is done by using the ‘[Authorize]’ annotation.

## [Authorize]

The user must be logged in to access this resource. It may be applied to the controller or to the action methods. When applied to the controller, all action methods in the controller are restricted.

|  |
| --- |
| [Authorize]  public class HomeController : Controller  {  …  }  [Authorize]  public IActionResult Restricted()  {  return Content("This is restricted.");  } |

**[Authorize(Users="Roach,jeff@abc.com")]** – Authorizes these users by their usernames

**[Authorize(Roles="admin")]** – Authorizes users with these roles.

## [AllowAnonymous]

Allows access to non-authenticated users.

|  |
| --- |
| [AllowAnonymous]  public IActionResult About()  {  ViewData["Message"] = "Your application description page.";  return View();  } |

## Accessing User Information

## User Object

The framework maintains the currently logged in user using the ‘User’ object. Which maps to an authentication cookie.

|  |  |
| --- | --- |
| **User.Identity.Name** | Gets the name of the user |

|  |
| --- |
| if(**User.IsInRole("Special")**)  {  …  } |

Using the ‘User’ object reference:

|  |
| --- |
| public IActionResult Index()  {  return Content(User.Identity.Name);  } |

|  |
| --- |
| public interface IUserRepository  {  ApplicationUser ReadUser(String email);  } |
| public class DbUserRepository : IUserRepository  {  private ApplicationDbContext \_db;  public DbUserRepository(ApplicationDbContext db)  {  \_db = db;  }  public ApplicationUser ReadUser(string email)  {  var user = \_db.Users.FirstOrDefault(u => u.Email == email);  return user;  }  } |
| public IActionResult UserId()  {  var user = \_userRepo.ReadUser(User.Identity.Name);  return Content(user.Id);  } |

# References

Rastogi, P., Anderson, R., Dykstra, T., Galloway, J., Reitan, E., & Smith, S. (2017, July 7). *Introduction to Identity on ASP.NET Core*. Retrieved November 20, 2017, from Microsoft Docs: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?tabs=visual-studio%2Caspnetcore2x

Unnamed. (2016, October 14). *Authorization in ASP.NET Core: Simple, role, claims-based, and custom*. Retrieved November 20, 2017, from Microsoft Docs: https://docs.microsoft.com/en-us/aspnet/core/security/authorization/

1. http://www.dictionary.com/browse/authentication?s=t [↑](#footnote-ref-1)