ASP.NET Identity System

Users, Roles, Authorization, Registration, Login, Logout, …

Table of Contents

AUTHENTICATION AND AUTHORIZATION

What's the Difference?

Authentication vs. Authorization

- Authentication
  - The process of verifying the identity of a user or computer
  - Questions: Who are you? How you prove it?
  - Credentials can be password, smart card, external token, etc.
- Authorization
  - The process of determining what a user is permitted to do on a computer or network
  - Questions: What are you allowed to do? Can you see this page?

ASP.NET IDENTITY SYSTEM

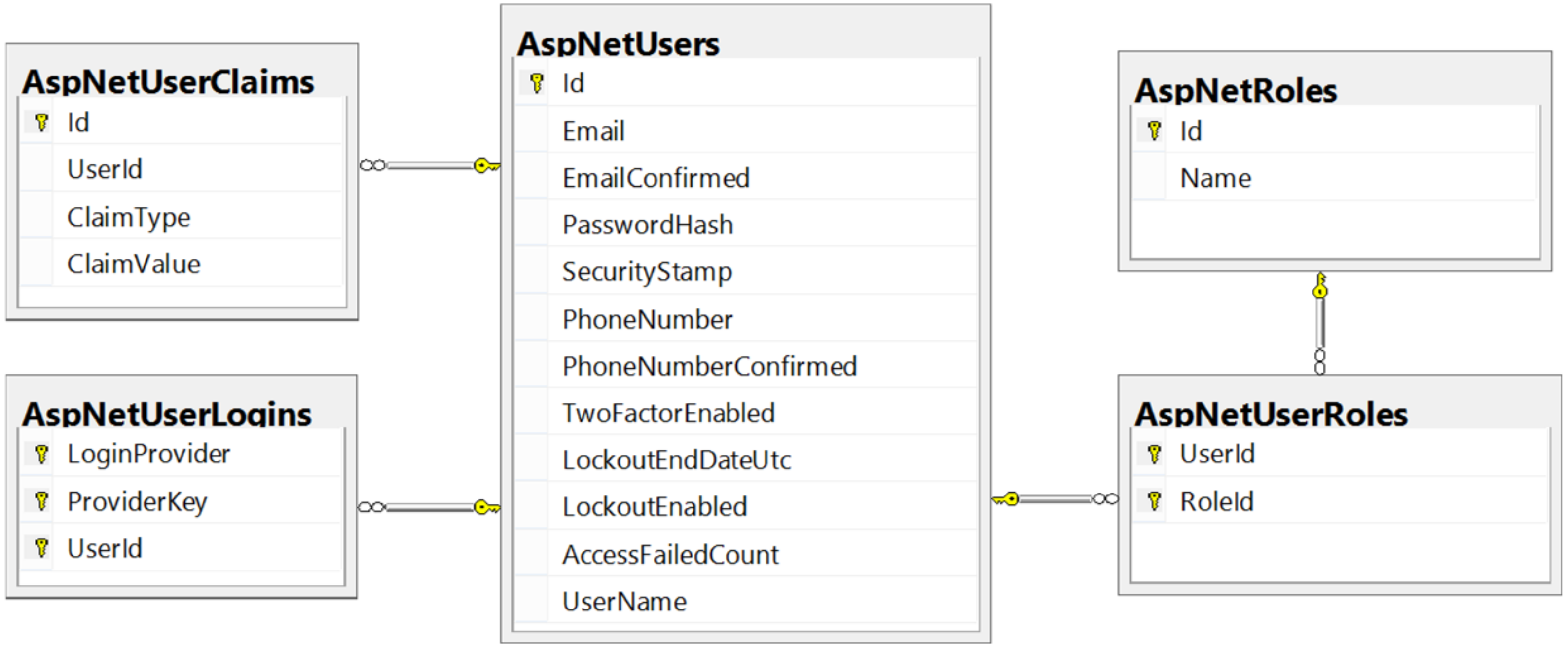
Overview

ASP.NET Identity

- The ASP.NET Identity system
  - Authentication and authorization system for ASP.NET Web apps
    - Supports ASP.NET MVC, Web API, Web Forms, SignalR, Web Pages
  - Handles users, user profiles, login / logout, roles, etc.
    - Keeps the user accounts in local database or in external data store
  - External login (through OAuth)
    - Supports Facebook, Google, Microsoft, Twitter accounts
  - Based on the OWIN middleware (can run outside of IIS)
  - Available through the NuGet package manager

ASP.NET Identity and Entity Framework

- Typically, the ASP.NET identity data (users, passwords, roles) is stored in relational database through EF Code First
  - You have some control over the internal database schema



UNIVERSITY of GREENWICH

Alliance with FPT Education

ASP.NET IDENTITY API

Setup, Registration, Login, Logout

ASP.NET Identity System Setup

- Ways to setup ASP.NET Identity based authentication in MVC
  - Using the ASP.NET project templates from Visual Studio
  - By hand: install NuGet packages, manual configuration, create EF mappings (models), view models, controllers, views, etc.
- Required NuGet packages
  - `Microsoft.AspNet.Identity.Core`
  - `Microsoft.AspNet.Identity.Owin`
  - `Microsoft.AspNet.Identity.EntityFramework`

ASP.NET Project Template Authentication

- `IdentityConfig.cs` – holds the user manager configuration
  - `ApplicationUserManager : UserManager<ApplicationUser>`
    - The main class for managing users in the ASP.NET Identity system
    - Can define the user and password validation rules
  - `ApplicationSignInManager : SignInManager`
    - Implements the user login / logout
    - Supports external login, e.g. Facebook login
    - Two-factor authentication (email confirm)

ASP.NET Project Template Authentication (2)

- **`IndentityModels.cs`** – holds user class and EF DB context
- **`ApplicationUser : IdentityUser`**
  - Holds the user information for the ASP.NET application
  - **`Id`** (unique user ID, string holding a GUID)
    - E.g. **`313c241a-29ed-4398-b185-9a143bbd03ef`**
  - **`Username`** (unique username), e.g. **`maria`**
  - **`Email`** (email address – can be unique), e.g. **`mm@gmail.com`**
  - May hold additional fields, e.g. first name, last name, date of birth

ASP.NET Project Template Authentication (3)

- **`ApplicationDbContext : IdentityDbContext<ApplicationUser>`**
  - Holds the EF data context with all database mapped entities
  - May define database initializer to specify DB migration strategy
- **`Startup.Auth.cs`**
  - Configures OWIN to use identity authentication
  - Usually enables cookie-based authentication
  - May enable external login (e.g. Facebook login)

User Registration

```
var newUser = new ApplicationUser
{
    UserName = "maria",
    Email = "mm@gmail.com",
    PhoneNumber = "+359 2 981 981"
};
var userManager = HttpContext.GetOwinContext().
    GetUserManager<ApplicationUserManager>();
var result = userManager.Create(newUser, "S0m3@Pa$$");

if (result.Succeeded)
    // User registered
else
    // result.Errors holds the error messages
```

User Login

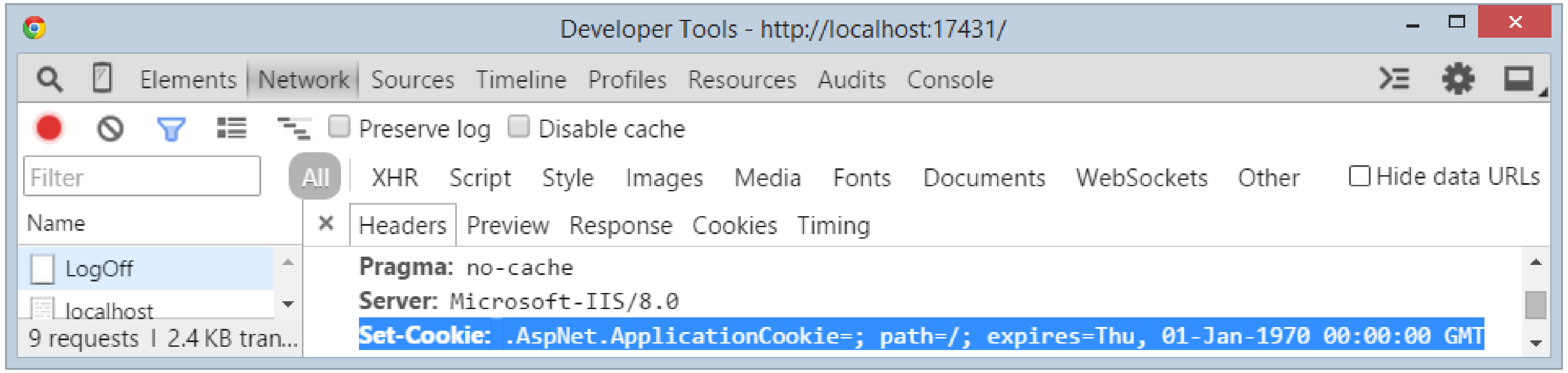
```
var signInManager = HttpContext.GetOwinContext().
    Get<ApplicationSignInManager>();
bool rememberMe = true;
bool shouldLockout = false;
var signInStatus = signInManager.PasswordSignIn(
    "maria", "S0m3@Pa$$", rememberMe, shouldLockout);
if (signInStatus == SignInStatus.Success)
    // Sucessfull login
else
    // Login failed
```

User Logout

- Performs local / external logout logout (log off / sign out):

```
var authenticationManager =
    HttpContext.GetOwinContext().Authentication;
authenticationManager.SignOut();
// Redirect to home screen or login screen
```

  - The logout clears the authentication cookies


Developer Tools - http://localhost:17431/
Elements Network Sources Timeline Profiles Resources Audits Console
Preserve log Disable cache
Filter
All XHR Script Style Images Media Fonts Documents WebSockets Other Hide data URLs
Name
Headers Preview Response Cookies Timing
LogOff
localhost
9 requests | 2.4 KB tran...

```
Pragma: no-cache
Server: Microsoft-IIS/8.0
Set-Cookie: .AspNet.ApplicationCookie=; path=/; expires=Thu, 01-Jan-1970 00:00:00 GMT
```

Change Password

- Logged-in user changes his password:

```
var currentUser = User.Identity.GetUserId();
var userManager = HttpContext.GetOwinContext().
    GetUserManager<ApplicationUserManager>();
var result = userManager.ChangePassword(
    currentUser, "old pass", "new pass");
if (result.Succeeded) …
```

- Administrator resets some user's password:

```
string token = userManager.GeneratePasswordResetToken (userId);
var result = userManager.ResetPassword(
   userId, token, "new password");
```

Extending the User Profile

- To extend the user profile
  - Just add properties to **`ApplicationUser`** class

```
public class ApplicationUser : IdentityUser
{
  [Required]
  public string Name { get; set; }
  …
}
```

- Enable migrations for the project / data layer
  - E.g. in **`Global.asax`** set the database initializer

AUTHORIZATION AND USER ROLES

ASP.NET Authorization

- Use the **`[Authorize]`** and **`[AllowAnnonymous]`** attributes to configure authorized / anonymous access for controller / action

```
[Authorize]
public class AccountController : Controller
{
  // GET: /Account/Login (annonymous)
  [AllowAnonymous]
  public ActionResult Login(string returnUrl) { … }

  // POST: /Account/LogOff (for logged-in users only)
  [HttpPost]
  public ActionResult LogOff() { … }
}
```

Check the Currently Logged-In User

```
// GET: /Account/Roles (for logged-in users only)
[Authorize]
public ActionResult Roles()
{
    var currentUserId = this.User.Identity.GetUserId();
    var userManager = HttpContext.GetOwinContext().
        GetUserManager<ApplicationUserManager>();
    var user = userManager.FindById(currentUserId);
    ViewBag.Roles = user.Roles;
    return this.View();
}
```

Create a New Role

- Identity roles group users to simplify managing permissions
  - ASP.NET MVC controllers and actions could check the user role
- Creating a new role:

```
var roleManager = new RoleManager<IdentityRole>(
  new RoleStore<IdentityRole>(new ApplicationDbContext()));
var roleCreateResult =
  roleManager.Create(new IdentityRole("Administrator"));
if (! roleCreateResult.Succeeded)
{
  throw new Exception(string.Join("; ", roleCreateResult.Errors));
}
```

Add User to Role

- Adding a user to existing role:

```
var userManager = HttpContext.GetOwinContext().
    GetUserManager<ApplicationUserManager>();
var addAdminRoleResult =
  userManager.AddToRole(adminUserId, "Administrator");
if (addAdminRoleResult.Succeeded)
{
  // The user is now Administrator
}
```

Require Logged-In User in Certain Role

- Give access only to users in role "Administrator":

```
[Authorize(Roles="Administrator"])
public class AdminController : Controller
{ … }
```

- Give access if user's role is "User" or "Student" or "Trainer":

```
[Authorize(Roles="User, Student, Trainer")]
public ActionResult Roles()
{
    …
}
```

Check the Currently Logged-In User's Role

```
// GET: /Home/Admin (for logged-in admins only)
[Authorize]
public ActionResult Admin ()
{
    if (this.User.IsInRole("Administrator"))
    {
        ViewBag.Message = "Welcome to the admin area!";
        return View();
    }

    return this.View("Unauthorized");
}
```