# 1: Create the Project

* Create a new ASP.NET Core MVC web application named ‘Lab12Authentication’ with authentication set to ‘Individual User Accounts’
* Use NuGet to update the installed packages to the latest builds of Core version 1.1.
* Create the DbContexts, Entities, and ViewModels folders in Models
* Create the Interfaces folder in Services
* In the Data folder, rename ApplicationDbContext to AuthenticationDbContext
* In appsettings.json, update the name of the database to Lab12AuthenticationDb
* Update the database

|  |
| --- |
| PM> update-database |

* ***Milestone 1: Show the created tables to the instructor. They should look something like the following:***

|  |
| --- |
|  |

# 2: Seeding Roles and an Admin User

* In the Services folder, create a class called Initializer with the following code. This class will be used to seed the database with 3 roles and the Admin user. Note the injections.

|  |
| --- |
| public class Initializer  {  private AuthenticationDbContext \_context;  private RoleManager<IdentityRole> \_roleManager;  private UserManager<ApplicationUser> \_userManager;  public Initializer(  AuthenticationDbContext context,  RoleManager<IdentityRole> roleManager,  UserManager<ApplicationUser> userManager)  {  \_context = context;  \_roleManager = roleManager;  \_userManager = userManager;  }  public async Task SeedAsync()  {  \_context.Database.EnsureCreated();  if (!\_context.Roles.Any(r => r.Name == "Admin"))  {  await \_roleManager.CreateAsync(new IdentityRole { Name = "Admin" });  }  if (!\_context.Roles.Any(r => r.Name == "Teacher"))  {  await \_roleManager.CreateAsync(new IdentityRole { Name = "Teacher" });  }  if (!\_context.Roles.Any(r => r.Name == "Student"))  {  await \_roleManager.CreateAsync(new IdentityRole { Name = "Student" });  }  if (!\_context.Users.Any(u => u.UserName == "admin@test.com"))  {  var user = new ApplicationUser  {  Email = "admin@test.com",  UserName = "admin@test.com"  };  await \_userManager.CreateAsync(user, "Pass1!");  await \_userManager.AddToRoleAsync(user, "Admin");  }  }  } |

* In Startup.ConfigureServices, add the Initializer service:

|  |
| --- |
| services.AddSingleton<Initializer>(); |

* Modify Program.cs to call Seed()

|  |
| --- |
| public class Program  {  public static void Main(string[] args)  {  var host = new WebHostBuilder()  .UseKestrel()  .UseContentRoot(Directory.GetCurrentDirectory())  .UseIISIntegration()  .UseStartup<Startup>()  .UseApplicationInsights()  .Build();  **using (var scope = host.Services.CreateScope())**  **{**  **var services = scope.ServiceProvider;**  **try**  **{**  **var initializer = services.GetRequiredService<Initializer>();**  **initializer.SeedAsync().Wait();**  **}**  **catch (Exception)**  **{**  **var logger = services.GetRequiredService<ILogger<Program>>();**  **logger.LogError("An error occurred while seeding the database.");**  **}**  **}**  host.Run();  }  } |

* Build the application to ensure there are no errors.
* Start the application
* ***Milestone 2: Show to the instructor the roles and user that were added to the database.***
  + ***AspNetRoles***
  + ***AspNetUsers***

# 3: Viewing the Roles

* Add the repository interface: IAuthenticationRepository

|  |
| --- |
| public interface IAuthenticationRepository  {  IQueryable<IdentityRole> ReadAllRoles();  } |

* Fill in the blanks for the repository implementation: DbAuthenticationRepository

|  |
| --- |
| public class DbAuthenticationRepository : IAuthenticationRepository  {  private AuthenticationDbContext \_db;  public DbAuthenticationRepository(AuthenticationDbContext db)  {  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;  }  public IQueryable<IdentityRole> ReadAllRoles()  {  return \_\_\_\_\_\_\_\_\_\_\_\_\_;  }  } |

* Configure the repository service in StartUp
* Add an empty controller called ‘RolesController’
* Inject the repository into the RolesController
* In \_ViewImports.cshtml, add the following @using. This is for IdentityRole.

|  |
| --- |
| @using Microsoft.AspNetCore.Identity.EntityFrameworkCore |

* In the ‘RolesController’ class, add the Index action method. Its model is the Roles from the repository.
* Add the ‘Index’ view by using the empty template. Here is the view code:

|  |
| --- |
| @model IEnumerable<IdentityRole>  @{  ViewData["Title"] = "Roles";  }  <h2>@ViewData["Title"]</h2>  <table class="table">  <tr>  <th>  @Html.DisplayNameFor(model => model.Name)  </th>  </tr>  @foreach (var item in Model)  {  <tr>  <td>  @Html.DisplayFor(modelItem => item.Name)  </td>  </tr>  }  </table> |

* Navigate the browser to ‘/roles.’ You should see the following output:

|  |
| --- |
|  |

* ***Milestone 3: Show the output to the instructor***

# 4: Viewing the Users

* Add ReadAllUsers to the repository. A user’s type is ApplicationUser. Implement ReadAllUsers. You should eager load the roles.
* Add the ‘UsersController’ and then code the Index action method and view. It should produce the following output. The Assign Roles link should send a request to /users/assignroles.

|  |
| --- |
|  |

* Click on the Register link to use the application to register a few users and then navigate back to ‘/users.’ Here’s mine:

|  |
| --- |
|  |
|  |

* Ensure you are logged out.
* Modify the ‘UsersController’ so that only users with the ‘Admin’ role may see the users. Use [Authorize(Roles="Admin")]
* Navigate to /users/ you should see the Log in page.
* Examine the Seed method and note the email and password of the Admin user.
* Log in as the Admin.
* ***Milestone 4: Demonstrate your application and code to the instructor.***

# 5: Assign Roles

* Create a view model AssignRoleViewModel that encapsulates two string collections: UserNames and RoleNames and two string properties UserName and RoleName.
* In ‘UsersController’, add the AssignRole action method. The model to the view is AssignRoleViewModel, which is populated with the user names and role names from the repository. The AssignRole action method should respond with. The form should POST to AssignRole.

|  |
| --- |
|  |

* Add bool AssignRole(string username, string rolename) to IAuthenticationRepository. Update the DbAuthenticationRepository implementation accordingly.
* Similar to the Initializer class, inject the UserManager into DbAuthenticationRepository.
* Complete the implementation of AssignRole in the repository. Here is the code:

|  |
| --- |
| public bool AssignRole(string username, string rolename)  {  var user = ReadAllUsers().FirstOrDefault(u => u.UserName == username);  if(user != null)  {  var roles = ReadAllRoles();  // Get the rolenames by joining user.Roles to roles  var q = from ur in user.Roles  join r in roles on ur.RoleId equals r.Id  select new  {  RoleName = r.Name  };  var role = q.FirstOrDefault(o => o.RoleName == rolename);  if(role == null)  {  \_userManager.AddToRoleAsync(user, rolename).Wait();  return true;  }  }  return false;  } |

* Add the POST AssignRole in the UsersController. Call the repository’s AssignRole and then redirect to the Index.
* Test the application by assigning roles to users. After assigning a new role, the role count should be updated on the user’s page.
* ***Milestone 5: Demonstrate the application to the instructor.***

**End of Lab**