DOCUMENT 8
CREATE ROLES AND SEED ADMIN USER

Goal

Create two roles, Admin and User.
Create a default admin account automatically.
Assign the Admin role to that admin account.

Step 1. Create the seeding class

Where it goes
Place this file inside your DbServices or Data folder.

File name
RolesAndSeedAdminUser.cs

```
using Microsoft.AspNetCore.Identity;
using RTWebApplication.Models;

namespace RTWebApplication.DbServices
{
    public class RolesAndSeedAdminUser
    {
        public static async Task RolesAndSeedAdminUserAsync(
            UserManager<ApplicationUser> userManager,
            RoleManager<IdentityRole> roleManager)
        {
            // Seed Roles
            if (!await roleManager.RoleExistsAsync("Admin"))
            {
                await roleManager.CreateAsync(new IdentityRole("Admin"));
            }
            if (!await roleManager.RoleExistsAsync("User"))
            {
                await roleManager.CreateAsync(new IdentityRole("User"));
            }

            // Seed Default Admin User
            string adminUserEmail = "admin123@tlevel.co.uk";
            string Password = "Password123!";

            if (await userManager.FindByEmailAsync(adminUserEmail) ==
null)
            {
                ApplicationUser adminUser = new ApplicationUser
                {
                    UserName = adminUserEmail,
                    Email = adminUserEmail,
                    FirstName = "Admin",
                    LastName = "User",
                    Address = "123 Admin St, Admin City, AD1 2BC",
                    CreatedAt = DateTime.Now
                };

                IdentityResult result = await
userManager.CreateAsync(adminUser, Password);

                if (result.Succeeded)
                {
                    await userManager.AddToRoleAsync(adminUser, "Admin");
                }
            }
        }
    }
}

```

Line by line explanation

Namespace and class
The class is inside RTWebApplication.DbServices.
That keeps it separate from controllers and models, which is clean organisation.

Method signature
RolesAndSeedAdminUserAsync is static.
You do not need to create an object to run it.
It is async because Identity calls are async.

Parameters
UserManager<ApplicationUser> manages users.
RoleManager<IdentityRole> manages roles.

Role seeding
RoleExistsAsync checks if the role already exists.
If not, CreateAsync inserts the role into AspNetRoles.

This is important
Without RoleExistsAsync, you risk duplicate role creation errors.

Admin user seeding
adminUserEmail is the lookup key.
FindByEmailAsync checks if the admin user exists in AspNetUsers.
If null, the user does not exist, so you create them.

Creating the admin user object
UserName and Email are set to the email. This keeps login simple.
FirstName, LastName, Address are your custom ApplicationUser fields.
CreatedAt stores a timestamp.

CreateAsync(adminUser, Password)
This inserts the user into AspNetUsers.
It also hashes and stores the password securely.
It returns IdentityResult so you can check success or failure.

AddToRoleAsync(adminUser, "Admin")
This links the user to the Admin role via AspNetUserRoles.

Step 2. Call your seeding method in Program.cs

Goal
Run the seeding method once on app start.

What you need in Program.cs
You must get UserManager and RoleManager from dependency injection.

Add these usings at the top if needed

```
1. using Microsoft.AspNetCore.Identity;
2. using RTWebApplication.DbServices;
3. using RTWebApplication.Models;
4.
```

After you build the app, but before app.Run(), add this block

```
 1. using (var scope = app.Services.CreateScope())
 2. {
 3.     var services = scope.ServiceProvider;
 4.
 5.     var userManager = services.GetRequiredService<UserManager<ApplicationUser>>();
 6.     var roleManager = services.GetRequiredService<RoleManager<IdentityRole>>();
 7.
 8.     await RolesAndSeedAdminUser.RolesAndSeedAdminUserAsync(userManager,
roleManager);
 9. }
10.
```

Explanation
CreateScope creates a safe scope for scoped services.
UserManager and RoleManager are scoped services.
GetRequiredService retrieves them from DI.
Then you call your seeding method.

Common mistake
Trying to call your method without a scope.
That often fails because scoped services need a scope.

Step 3. Confirm it worked

Open SQL Server Object Explorer.
Check these tables.

AspNetRoles
Should contain Admin and User.

AspNetUsers
Should contain admin123@tlevel.co.uk.

AspNetUserRoles
Should contain a link between that user and Admin.

If admin does not appear
Common reasons
Identity not installed or not configured
Migration not applied
Seed method not called in Program.cs

Step 4. Protect admin pages using the Admin role

In your admin controller, add this.

```
1. using Microsoft.AspNetCore.Authorization;
2.
3. [Authorize(Roles = "Admin")]
4. public class AdminProductController : Controller
5. {
6. }
7.
```

Explanation
Only users in the Admin role can access the controller actions.
This is the point of seeding Admin.

Extra exam points and improvements you should know

1. Use DateTime.UtcNow instead of DateTime.Now
   Utc avoids time zone issues.

If you want to change it

CreatedAt = DateTime.UtcNow

End result checklist

Roles exist. Admin and User.
Admin user exists. admin123@tlevel.co.uk.
Admin user is assigned the Admin role.
Admin pages are protected using Authorize with Roles = Admin.