WEB PROGRAMMING ASP.NET MVC CORE

SPORTS STORE PROJECT

IMPLEMENTING SEARCH (FILTER) CONTROLLER

```
public IActionResult Index(string name = null, int page = 1) {
    var pagination = new PagingInfo {
        CurrentPage = page,
        PageSize = PagingInfo.DEFAULT_PAGE_SIZE,
        TotalItems = repository.Products.Where(p => name == null || p.Name.Contains(name)).Count()
    };

    return View(new ProductsListViewModel {
        Products = repository.Products.Where(p => name == null || p.Name.Contains(name))
            .OrderBy(p => p.Price).Skip((page - 1) * pagination.PageSize).Take(pagination.PageSize),
        Pagination = pagination,
        SearchName = name
    });
}
```

IMPLEMENTING SEARCH (FILTER) VIEWMODEL

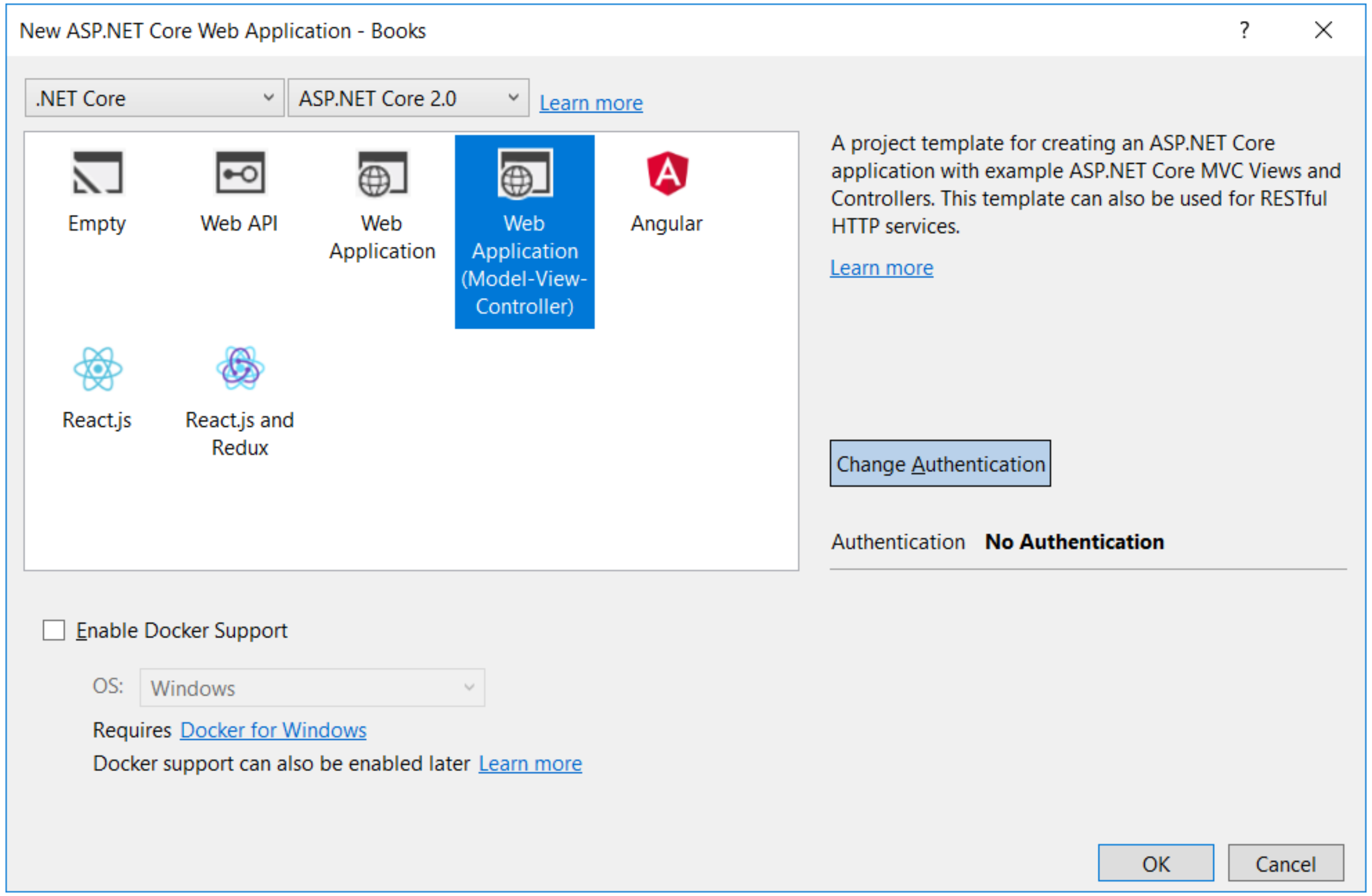
```
public class ProductsListViewModel {
        public IEnumerable<Product> Products { get; set; }
        public PagingInfo Pagination { get; set; }
        public string SearchName { get; set; }
}
```

IMPLEMENTING SEARCH (FILTER) VIEW

```
<div class="card mt-3 bg-info">
    <div class="card-header">
        <div class="card-title font-weight-bold">Search</div>
    </div>
    <div class="card-body">
        <form asp-action="Index" method="get">
            <label for="name">Name</label>
            <input name="name" type="search" class="form-control" value="@Model.SearchName" />
            <input name="page" type="hidden" value="1" />
            <div class="mt-3">
                <input type="submit" value="Search" class="btn btn-primary" />
                <a asp-action="Index" class="btn btn-secondary">Clear</a>
            </div>
        </form>
    </div>
</div>
```

BOOKS PROJECT

AUTHENTICATION


New ASP.NET Core Web Application - Books

.NET Core | ASP.NET Core 2.0 | Learn more

A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

Learn more

Change Authentication

Authentication **No Authentication**

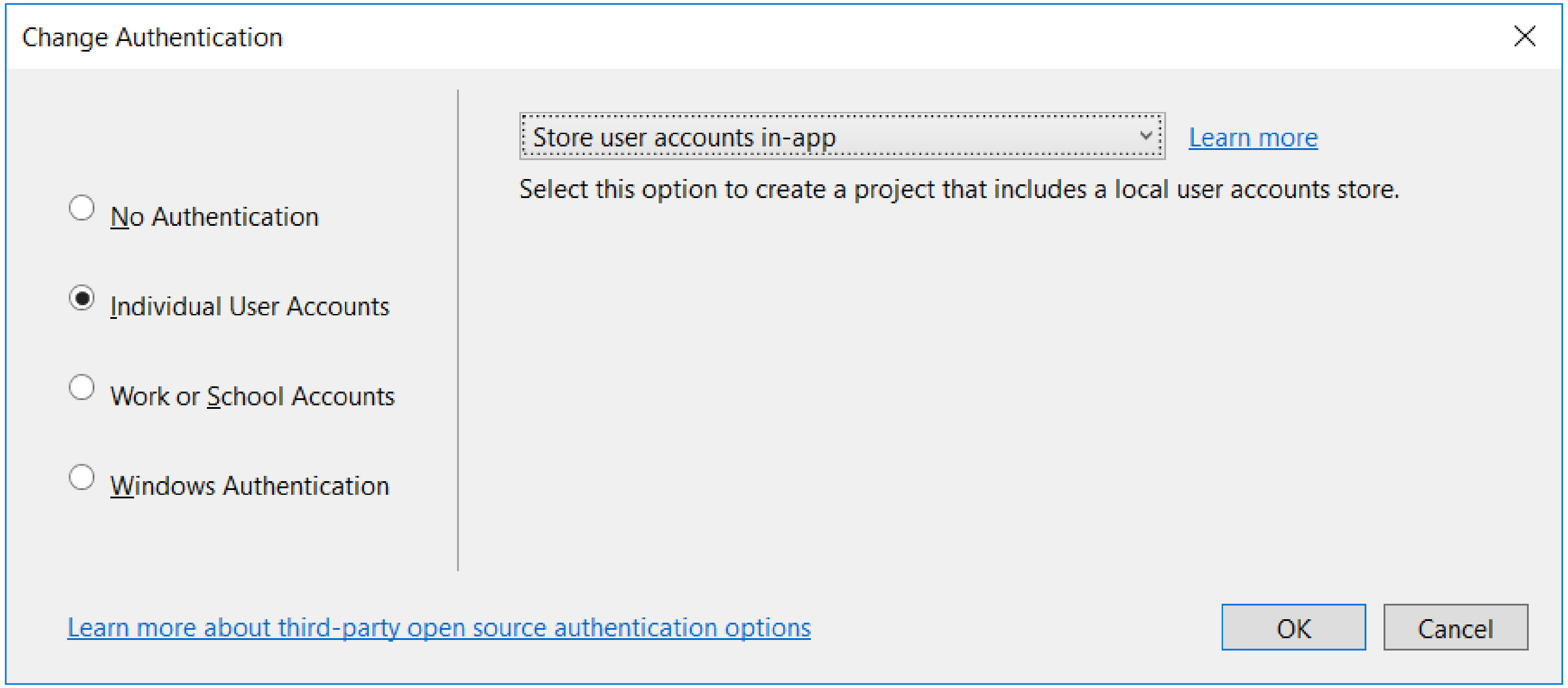
Enable Docker Support

OS: Windows

Requires Docker for Windows

Docker support can also be enabled later Learn more

AUTHENTICATION


Change Authentication

( ) No Authentication
(●) Individual User Accounts
( ) Work or School Accounts
( ) Windows Authentication

Store user accounts in-app

Learn more

Select this option to create a project that includes a local user accounts store.

Learn more about third-party open source authentication options

OK

Cancel

ASP.NET CORE IDENTITY

HTTPS://DOCS.MICROSOFT.COM/ASPNET/CORE/SECURITY/AUTHENTICATION/IDENTITY

- Authentication and authorization are provided by the ASP.NET Core Identity system.
- ASP.NET Core Identity is a membership system which allows you to add login functionality to your application. Users can create an account and login with a user name and password or they can use an external login provider such as Facebook, Google, Microsoft Account, Twitter or others.
- You can configure ASP.NET Core Identity to use a SQL Server database to store user names, passwords, and profile data. Alternatively, you can use your own persistent store, for example Azure Table Storage.

CONFIGURE IDENTITY SERVICES

```
public void ConfigureServices(IServiceCollection services) {
    services.AddDbContext<SportsStoreDbContext>(options =>    options.UseSqlServer(
        Configuration.GetConnectionString("ConnectionStringBooksUsers")));

    services.AddIdentity<IdentityUser, IdentityRole>(options => {
        // Sign in
        options.SignIn.RequireConfirmedAccount = false;
        // ...
    }).AddEntityFrameworkStores<ApplicationDbContext>().AddDefaultUI();

    // ...
}
```

CONFIGURE IDENTITY SERVICES

```
services.Configure<IdentityOptions>(options => {
     // Sign in
     options.SignIn.RequireConfirmedAccount = false;

     // Password
     options.Password.RequireDigit = true;
     options.Password.RequireLowercase = true;
     options.Password.RequiredLength = 8;
     options.Password.RequiredUniqueChars = 6;
     options.Password.RequireNonAlphanumeric = true;
     options.Password.RequireUppercase = true;

     // Lockout
     options.Lockout.AllowedForNewUsers = true;
     options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(30);
     options.Lockout.MaxFailedAccessAttempts = 5;
});
```

ADDING MIGRATIONS

- `dotnet restore`
- `dotnet ef migrations add Initial --context BooksDbContext`
- `dotnet ef migrations add Initial --context BooksIdentityDbContext`
- `dotnet ef database update --context BooksDbContext`
- `dotnet ef database update --context BooksIdentityDbContext`

SEED USERS DATA

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env,
    BooksDbContext db,
    UserManager<IdentityUser> userManager // Dependency injection
) {
    // ...

    SeedData.SeedDefaultAdminAsync(userManager).Wait();
    if (env.IsDevelopment()) {
        SeedData.SeedDevData(db);
        SeedData.SeedDevUsersAsync(userManager).Wait();
    }
}
```

SEED USERS DATA

```
public class SeedData {
    private const string DEFAULT_ADMIN_USER = "admin@ipg.pt";
    private const string DEFAULT_ADMIN_PASSWORD = "Secret123$";

    internal static async Task SeedDefaultAdminAsync(UserManager<IdentityUser> userManager) {
        await EnsureUserIsCreated(userManager, DEFAULT_ADMIN_USER, DEFAULT_ADMIN_PASSWORD);
    }

    private static async Task EnsureUserIsCreated(UserManager<IdentityUser> userManager,
                                string username, string password) {
        IdentityUser user = await userManager.FindByNameAsync(username);

        if (user == null) {
            user = new IdentityUser(username);
            await userManager.CreateAsync(user, password);
        }
    }
}
```

AUTHORIZE

```
public class BooksController : Controller {
    // ...

    [Authorize]
    public IActionResult Create() {
        return View();
    }

    // ...
}
```

ROLES

- A role is just an arbitrary label that you define to represent permission to perform a set of activities within an application.
- Almost every application differentiates between users who can perform administration functions and those who cannot. In the world of roles, this is done by creating an Administrators role and assigning users to it.
- Users can belong to many roles, and the permissions associated with roles can be as coarse or as granular as you like, so you can use separate roles to differentiate between administrators who can perform basic tasks, such as creating new accounts, and those who can perform more sensitive operations, such as accessing payment data.
- ASP.NET Core Identity takes responsibility for managing the set of roles defined in the application and keeping track of which users are members of each one. But it has no knowledge of what each role means; that information is contained within the MVC part of the application, where access to action methods is restricted based on role membership.

SEED ROLES

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env,
    BooksDbContext db,
    UserManager<IdentityUser> userManager,
    RoleManager<IdentityRole> roleManager) {
    // ...
    SeedData.SeedRolesAsync(roleManager).Wait();
    SeedData.SeedDefaultAdminAsync(userManager).Wait();

    if (env.IsDevelopment()) {
        SeedData.SeedDevData(db);
        SeedData.SeedDevUsersAsync(userManager).Wait();
    }
}
```

SEED ROLES

```
public class SeedData {
    // ...

    private const string ROLE_ADMINISTRATOR = "Admin";
    private const string ROLE_PRODUCT_MANAGER = "ProdutManager";
    private const string ROLE_CUSTOMER = "Customer";

    internal static async Task SeedRolesAsync(RoleManager<IdentityRole> roleManager) {
        await EnsureRoleIsCreated(roleManager, ROLE_ADMINISTRATOR);
        await EnsureRoleIsCreated(roleManager, ROLE_PRODUCT_MANAGER);
        await EnsureRoleIsCreated(roleManager, ROLE_CUSTOMER);
    }

    private static async Task EnsureRoleIsCreated(RoleManager<IdentityRole> roleManager, string role) {
        if (!await roleManager.RoleExistsAsync(role)) {
            await roleManager.CreateAsync(new IdentityRole(role));
        }
    }
}
```

ASSIGN ROLES

```
internal static async Task SeedDefaultAdminAsync(UserManager<IdentityUser> userManager) {
    await EnsureUserIsCreated(userManager, DEFAULT_ADMIN_USER, DEFAULT_ADMIN_PASSWORD, ROLE_ADMINISTRATOR);
}

private static async Task EnsureUserIsCreated(UserManager<IdentityUser> userManager, string username,
        string password, string role) {
    IdentityUser user = await userManager.FindByNameAsync(username);
    if (user == null) {
        user = new IdentityUser(username);
        await userManager.CreateAsync(user, password);
    }

    if (!await userManager.IsInRoleAsync(user, role)) {
        await userManager.AddToRoleAsync(user, role);
    }
}
```

ASSOCIATE ROLES TO ENTITIES

```
internal static async Task SeedDevUsersAsync(UserManager<IdentityUser> userManager) {
    await EnsureUserIsCreated(userManager, "john@ipg.pt", "Secret123$", ROLE_PRODUCT_MANAGER);
    await EnsureUserIsCreated(userManager, "mary@ipg.pt", "Secret123$", ROLE_CUSTOMER);
}

internal static void SeedDevData(BooksDbContext db) {
    if (db.Customer.Any()) return;

    db.Customer.Add(new Customer {
        Name = "Mary",
        Email = "mary@ipg.pt"
    });

    db.SaveChanges();
}
```

ROLES

```
public class BooksController : Controller {
    // ...
    [Authorize(Roles = "Administrator")]
    public IActionResult Create() {
        return View();
    }

    [Authorize(Roles = "Customer")]
    public string Buy() {
        return "The option for customers to buy books will be added soon !!!";
    }
}
```