Sales Board Application Technical Documentation

Migrations

The application was built with identity for its built-in login functionality. The web application was started by creating a ASP.NET Web Application project with Individual User Accounts. The initial migration ‘Initial’ set up Identity. The second migration ‘CustomUserData’ set up a model ‘SalesBoardAppUser’ which derived from IdentityUser class. Additional properties were added for the user model such as first name and last name of a user. The third migration ‘DatabaseSchema’ set up the tables and their relationships for the app functionality as a sales board. More information on the schema can be found in the section about the database. The fourth and fifth migration added additional information needed as the application progressed, such as product date, product availability and product image.

Database Schema

This basic sales board application has 3 main tables; User, Product and Purchase. A user can have many products for sale but a product for sale can only belong to one user. This is a one-to-many relationship from User to Product.

A user can be a customer or a seller or both. A customer can purchase products from many different users. A seller can sell products to many different users. This is a many to many self-join relationship on the User table and represented by the Purchase table, which tracks who is a seller and who is a customer with each purchase.

Fluent API in Entity Framework Core was used to configure the schema. The ‘OnModelCreating’ method was overrided and the ‘ModelBuilderAPI’ was used in the SalesBoardAppContext. This allowed more flexibility in creating the relationships between the models manually.

Authorization

This application uses a combination of role-based authorization and policy-based authorization. There is only one role that a user can be assigned to. This is the administrator role. If a user does not have this role then they are considered a regular user. Each controller also contains a simple authorize attribute to limit every page to those that have an account. As this is a sales board, you must be signed in to be able to sell or purchase items. The policy is a simple authorization handler based on the product and user. The HandleRequirementAsync method of the ProductAuthorizationHandler marks the UserAccessRequirement as successful if the current user id of a user matches the user id of a product or if the user has an administrator role. The requirement is an empty class because there are no additional data parameters for the policy to use. The policy uses already existing parameters; user id’s, product id’s and user roles.

For example, in the products controller, to prevent users from editing products that are not their own, this policy is used within both the GET and POST edit methods. The policy is also used in the views to only display certain buttons if a product is owned by the current user.

Sessions & Shopping Cart

Sessions are used to store user data. The shopping cart is based on a user session. ISession is used to get and set string and integer values. Extension methods from the Microsoft.AspNetCore.Http namespace give access to the session as an ISession. Get and set methods are used to serialize and deserialize session data. In this application the session data that needs to be stored are objects. In order to do this, an object must be stored by converting to JSON first. When data is to be retrieved it is retrieved from JSON and inflated as an object again.

The cart controller deals with the cart-based requests coming from the user. User data is stored or retrieved using the ISession extension get and set methods as noted above. For example, an ‘add to cart’ request from the server will add a product and quantity to the session as a list of items. An Item is a product and a quantity combined. When viewing the shopping cart, all the items stored in the session are retrieved and displayed.

Models

**Item:**

An item has two properties; a Product and a Quantity. This is used in the implementation for the shopping cart. Items are stored in lists and serialized as strings in JSON form.

**Product:**

The product model defines the properties to be stored in the product table of the database. These properties are:

* ProductId
* Name
* Description
* Image
* Price
* Available
* Posted

The product model has a foreign key of UserId, establishing the many-to-one relationship with users.

**ProductSearchViewModel:**

This model allows a user to search for products via a search string. The result of a search for products is stored in the Products property of a ProductSearchViewModel, which is sent to the index view for display.

**Purchase:**

The purchase model defines the properties to be stored in the purchase table of the database. The properties include PurchaseId and Amount. It has two foreign keys of SellerId and Customer Id both representing users from the User table. The purchase model represents the many-to-many self-join on the User table.

**SeedData:**

Seeding is done through this class. Users and their roles are created, and the database is populated with data to get the application started for use.

**ShoppingCartViewModel:**

This model has two properties; Items and TotalPrice. This allows for the creation of an object that summarizes a shopping cart with all the items in it and a grand total price of all the items.

**UserInfoViewModel:**

This model has two properties; Products and Purchases. This allows for the creation of an object that summarizes user data, including what products they have for sale and who their customers are.

Alerts

Bootstrap 4 alerts are added through extension methods. These alerts provide important information to the user, such as login success/failure, access restriction messages and general feedback application use. The method used was implemented from the following source:

<https://www.trycatchfail.com/2018/01/22/easily-add-bootstrap-alerts-to-your-viewresults-with-asp-net-core/>