Project 1: ASP.NET Core MVC Store Web Application

Due June 30, 2021 - Arlington .NET / Mark Moore

functionality

- place orders to store locations for customers
- add a new customer
- search customers by name
- display details of an order
- display all order history of a store location
- display all order history of a customer
- client-side validation
- server-side validation
- exception handling
- CSRF prevention
- persistent data (no prices, customers, order history, etc. hardcoded in C#)
- logging where appropriate
- (+5: order history can be sorted by earliest, latest, cheapest, most expensive, etc)
- (+5: get a suggested order for a customer based on his order history)
- (+5: display statistics based on order history)
- (+5: asynchronous network)

design

- use EF Core (either database-first approach or code-first approach)
- use a SQL DB in third normal form (3NF)
- Use the most restricted access possible. (i.e. don't use public fields, instead use protected fields or public properties with private backing fields)
- define and use at least one interface
- (+10: Deploy to publicly available Azure App Service website)
- (+10: implement Authorization. DO NOT use the automatic authorization feature of Visual Studio.)

core / domain / business logic

- class library
- contains all business logic

- contains domain classes (customer, order, store, product, etc.)
- documentation with
- XML comments on all public types and members (optional: and)

customer

- has first name, last name, etc.
- Every newly created customer can choose a default store location to order from

order

- has a store location
- has a customer
- has an order time (when the order was placed)
- can contain multiple products in the same order
- (+5: additional business rules, like special deals, bundles)

location

- each location has an inventory.
- inventory decreases when orders are accepted
- rejects orders that cannot be fulfilled with remaining inventory
- (+5: for a product, more than one inventory item decrements when ordering that product)

product (Same expectation as above.)

- has a description, price, etc.

user interface

- ASP.NET Core MVC web application
- must separate request processing and presentation projects using the MVC pattern
- minimize logic in Views
- customize the default styling
- keep 'NamesLikeThis' out of the visible UI. (use Model annotations to give [DisplayName()])

data access

- contained in separate class library project
- use Dependency Injection
- contains EF Core DbContext and entity classes
- contains data access logic but no business logic

- use Repository Pattern for Separation of Concerns

test

- at least 20 test methods
- focus on unit testing business logic
- data access tests to the app's actual database