Tasks

Backlog

Address separation of concerns

Address unit testing in XUnit : min 10 unit tests.

Set up try-catch for exception handling at ALL points of data entry ON business logic

Improve logging

XML Comment all code.

Separation of Concerns for Data\_Management

Note on the backlog plan: vertical integration will involve getting ONE feature implemented at a time. This means getting ONE page up and running and making sure it integrates with the rest of the website.

Get An MVC Example running.

Dissect the Pokémon code and attach to business logic.

Integrate Console App and Business Logic into ASP.NET MVC Project

Create the pages.

Create the forms

Create the queries

Improvements over Project 0

Business Logic

Business Management

Query Customer’s:

Account Recovery

Inventory Management

Customer Experience

Password Hint

Design docs: Flowchart

ERD

A Script for Order of Operation during 5min presentation.

Power Point Presentation

Project 1

**project 1: store web application**

Sept 30 2019 Arlington .NET / Nick Escalona

**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 (SQL, 3NF database); no products, prices, customers, etc. hardcoded in C#
* logging
* (optional: order history can be sorted by earliest, latest, cheapest, most expensive)
* (optional: get a suggested order for a customer based on his order history)
* (optional: save some or all data to disk in JSON format)
* (optional: load some or all data from disk)
* (optional: display some statistics based on order history)

**structure**

**business logic**

* class library
* contains all business logic
* contains domain classes (customer, order, store, product, etc.)
* documentation with <summary> XML comments (optional: <params> and <return>)
* has no dependency on UI or any input/output considerations

**user interface**

* ASP.NET MVC web application
* strongly-typed views
* minimize logic in views
* use dependency injection
* customize the default styling to some extent

**data access**

* class library
* contains EF DbContext and entity classes
* contains data access logic but no business logic
* use repository pattern for separation of concerns

**test**

* at least 10 test methods
* use TDD for some of the application
* focus on unit testing business logic
* data access tests (if present) should not impact the app's actual database

**object model**

**customer**

* has first name, last name, etc.
* (optional: has 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 product types in the same order
* rejects orders with unreasonably high product quantities
* must have some additional business rules

**location**

* has an inventory
* inventory decreases when orders are accepted
* rejects orders that cannot be fulfilled with remaining inventory
* (optional: more than one inventory item decrements for a given product order, for at least one product)

**product**

**technologies**

* C#/.NET
* ASP.NET MVC
* Entity Framework
* Azure SQL Database
* xUnit, NUnit, or MSTest
* Serilog or NLog
* (optional: SonarLint extension in VS or VS Code for static analysis)