Store App Console Version

Let's sell work appropriate stuff.

Overview:

The store manager is a console application used to help ease your shopping! You decide what type of store you'll be selling (safe appropriate) and any other functionalities that makes sense with the items you are selling (age restricted items or other regulations).

App Architecture

Models

- The models project would contain all the data structures that you will be utilizing in your BL, DL, and UI.
- These are the data structures you'll be processing in the BL, DL, & UI.
- Ex: A restaurant would need an employee model to hold any employee information that restaurant has hired.
- Project type: Class Library

Business Logic

- The business project would contain the logic to compute and process the data needed for transactions.
- Project type: Class Library

Data Access Logic

- This data access project would contain the logic to perform CRUD operations on the necessary models to some repository.
- Contains repository specific logic on storing and accessing data from different kinds of data storage systems.
- Project type: Class Library

User Interface

- The User Interface (UI) is responsible for displaying information to the end-user.
- Can interact with the end-user to ask for certain information or what is the next step to take.
- The end-user should only interact with the UI and nothing else.
- Project type: Console

Models

- You can add more models as you see fit.

Customer

- The customer model is supposed to hold the data concerning a customer.

Properties:
- Name
- Address
- Email/Phone number
- List of Orders

StoreFront
- The store front contains information pertaining the various store locations.

Properties:
- Name
- Address
- List of Products
- List of Orders

Orders
- The orders contain information about customer orders.

Properties:
- List of Line Items
- StoreFront's Location (that the order was placed).
- Total price

Line Items
- The line items contain information about a particular product and its quantity.

Properties:
- Product
- Quantity

Products
- The Product model is supposed to hold the data concerning a customer.

Properties:
- Name
- Price
- Desc. (optional)
- Category (optional)

Functionalities
Add customer
Logic to be implemented:
UI
- User interface that validates customer information based on model properties.

- UI should ask the end user relevant information about the customer they're creating such as the name, address, etc.
- Hint: Use console methods such as readline or writeline (Check Microsoft Documents for more information) and looping construct to help with validation.

BL

- Come up with logic to add a customer. Call the appropriate DL methods to accomplish this. Throw necessary exceptions if needed.

DL

- Write data storage specific logic to add a customer to some form of repository. Over time, repository will develop from file system to database. (If you don't know how to do either, focus on doing the UI and having the validation all setup).

Search for a customer

Logic to be implemented:

UI

- User interface that takes in user input (name, address, etc.) and find that customer in the database.
- Make sure appropriate messages are given if a customer is not found and not an exception.

BL

- Call the appropriate DL methods to accomplish this.
- If DL cannot find a customer, be sure your BL has the appropriate response to give to your UI.

DL

- Create logic that can go through the database and find the appropriate customer.
- Hint: Use LINQ methods

View Store Front inventory

Logic to be implemented:

UI

- User interface that searches and gets store front information based on some search parameter.
- User can select the store to buy products from.
- Make sure appropriate messages are given if a Store is not found and not an exception.
- If the location has no products left, give appropriate message of an empty store.

BL

- Call the appropriate DL methods to accomplish this.
- Logic that user has selected the store and can start viewing its inventory.

DL

- Create logic that can go through the database and find the appropriate store front.

- Create logic that can view that store’s inventory once selected.

Place order

Logic to be implemented:

UI

- User interface that uses customer’s information to place an order.
- User can select appropriate products that exists in that store.
- User cannot buy more than the current quantity of the product.
- User can buy multiple items within one order.
- Other functionalities that make sense for that item (age restriction or other regulations).
- User should see the total price of the order.
- Nice things to have but not required: remove existing items in the order, update items to have more quantity if the user wants more, etc.

BL

- Search customer and store front based on some parameter and assign them to the current order.
- Update the inventory of the store after the order has been placed.
- Validate that the current store front’s inventory can satisfy the order’s quantity.
- Calculate total price.

DL

- Use pre-existing DL that you already made to search the customer and store front.
- Create logic to add the order in the database that is attached to the customer and store front.

View order history

- You should be able to view the order history of both the customer and the store front.

Logic to be implemented:

UI

- User interface that gives the order history of a customer or store front.
- Nice to have: Being able to sort the order history in some way (total price, alphabetical, date, etc.).

BL

- Call appropriate DL methods to retrieve order history from a customer/store front.
- Nice to have: Logic to sort the order history.

DL

- Create logic to find all the order history of a customer/store front from the database.

Replenish inventory

Logic to be implemented:

UI

- User interface can select a store front and view its inventory.
- User can replenish the quantity of the product in that store.

BL

- Call the appropriate DL methods to retrieve store front's inventory and update the quantity.

DL

- Create logic to update the quantity of the store front's inventory.

Logging

- Your app should store meaningful logging information (Such as exceptions, user's action, what table is being accessed to, etc.).
- You can use any logging framework, but I recommend using Serilog or NLog.

Unit Testing

- Your app should have at least **10** meaningful unit test that checks if validation is working as intended in your application.
- You can also unit test your DL if it is able to access the proper data from your database using SQLite.

Tips

- Start as soon as you get this SDD even if you don't know how to do everything. You only have two weeks to create it and you need to utilize as much as you can pour in to have a working application.
- Don't cheat! Copying other people's repository in Github is grounds for immediately mutual release.
- Try to prioritize which functionality makes sense in a working store app. If you can't buy anything in your app then the whole purpose of a store app is ruined.
- Try to finish as much functionality as possible since the next project will build upon your existing P0.
- If you have any unclear understanding of the requirements listed above, ask your batchmates first. If they also don't know, then you can ask me. First thing I'll ask for you if you asked other people first.
    - Get comfortable talking with them since they will be your future group mates on working one enterprise project.