Assignment 1 Sales Board

Contents

[Introduction 2](#_Toc50847972)

[Phase 1: Creating MVC Application and Data 2](#_Toc50847973)

[Setting up Application 2](#_Toc50847974)

[Modifying Identity User 2](#_Toc50847975)

[Database Modelling 2](#_Toc50847976)

[Scaffolding 3](#_Toc50847977)

[Phase 2: Site Functionality and Shopping Cart 4](#_Toc50847978)

[Appendix 5](#_Toc50847979)

# Introduction

Github Link: <https://github.com/ZephyrGitCode/HIT339_Assign1_Salesboard>

# Phase 1: Creating MVC Application and Data

This encompasses setting up the application from scratch, creating database model files and scaffolding items.

## Setting up Application

I started by setting up a github repository for the application. Next, I created the MVC application, ensuring to use “User Authentication” functionality.

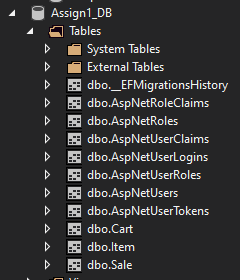
## Modifying Identity User

I then decided that I would start with modelling the database by creating empty classes for Items, Sales and Cart. I then started the Items class, during which I got side-tracked by trying to include multiple items under 1 user and add custom fields to the current user database. Therefore, I performed the following:

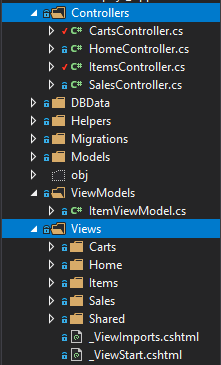
1. I created an applicationUser.cs class for my new custom fields.
2. I then modified the ApplicationContext.cs to support custom user data from the applicationUser class above.
   1. Made 1 custom field with no entry yet – to be added later.
3. Now I have my Identity, I then started to add some custom fields to the Identity.
   1. I added the following custom fields to the identity user: first name, last name, age, state, city, postcode, street, images. See [Figure 1.1](#_Appendix) For an image of the identity fields.
4. Having successfully made a migration with new custom user fields I then added a foreign key to users in Items and migrated.
   1. After I fixed Items class to reference Zephyr\_ApplicationUser I then migrated again.
   2. Then it worked and Item class is properly referencing the User.

## Database Modelling

Now that the Identity user table and Items table was able to reference it, then begun developing all the classes.

1. Next is to do the Items table.
   1. See [Figure 1.2](#_Appendix) for contents of Items class.
2. Then Sales table.
   1. See [Figure 1.3](#_Appendix) for contents of Sales class.
3. Finally, Cart. This table holds temporary records purchases with cart.
   1. See [Figure 1.4](#Figure1_4) for contents of Carts class.
   2. Time for migrations. I migrated the content in the following order: Identity, Items, Sales, Carts.   
      
4. I then tested all the tables with two users, added some items and included some sales. This was achieved using the in-built database tool as I have not scaffolded my data yet, which is the next step.

## Scaffolding

1. Lastly, I performed scaffolding of my database classes to generate CRUD pages for them.
   1. CRUD for Items, Sales and Carts.  
      

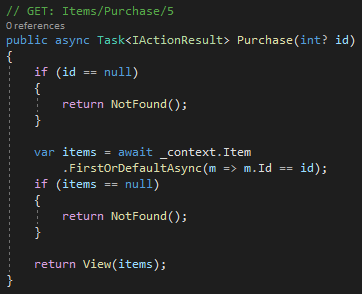
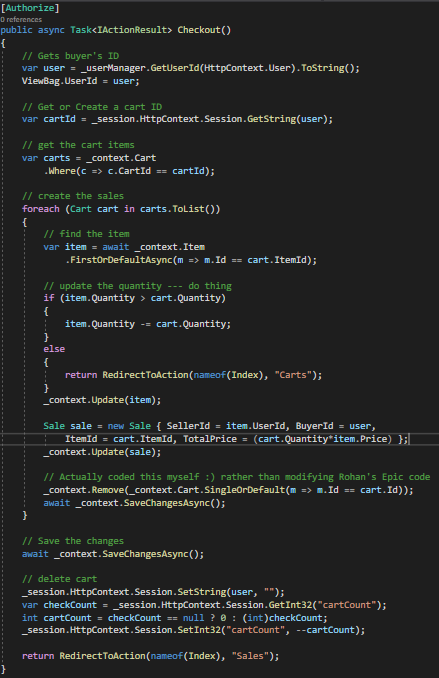
# Phase 2: Site Functionality and Shopping Cart

This phase encompasses the creation of filtered pages, the addition of add to cart functionality, creating and managing the shopping cart then lastly checking out.

## Filtering

1. First, I modified the scaffolded views to remove “Edit” or “Delete” options for users whom did not create the items.
   1. To Do - Apply logic that allows Admins to modify all content.
   2. Next, I modified the Items controller index page to filter out items that had less than one quantity left.
   3. Also, when the item is owned by the current user it does not show the purchase button
2. Also, for the Carts and Sales index pages I filtered out data that did not match the current user
   1. Additionally, the Sales are ordered by descending.

## Shopping Cart

1. I created a new views page named “Purchase” to be interacted with in the Items controller.
   1. I then modified the Items index with a link to the purchase page for a particular item.
2. Next, I added the following actions to the Items controller
   1. GET: Purchase – Loads the purchase view page with the item data.  
      
3. POST: PurchaseConfirmed– Inserts a new cart record into the DB.  
   
4. I then modified the carts index page with a checkout action then created the action in the carts controller.
   1. POST: Checkout – For each item in cart, create a new sale.  
      
5. With this functionality complete, users are able to add items to their cart and checkout their items.

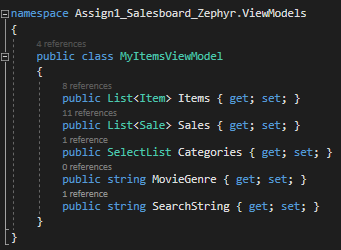
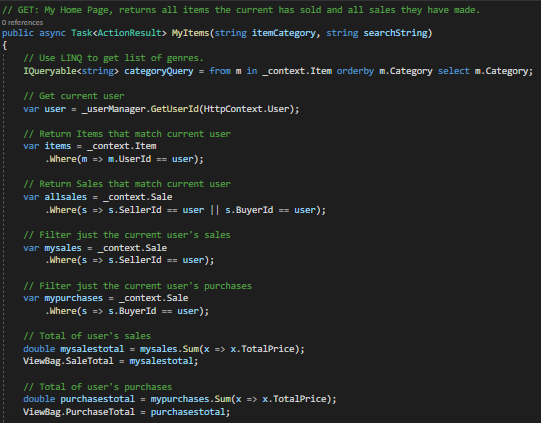
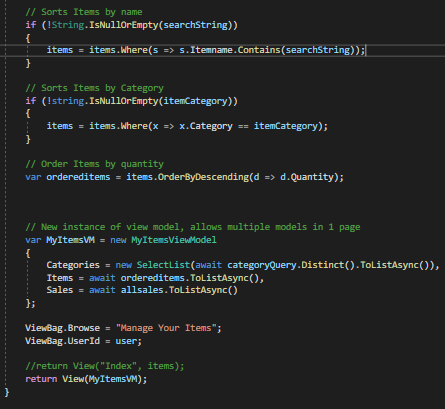
# Phase 3: Usability and HTML improvements and View Models

This encompasses many improvements to html and classes and the creation and use of View Models for data.

1. First, I went through and removed all unnecessary functions such as:
   1. Carts: Creating, editing and details.
   2. Sales: Creating, editing and deleting.
2. Next improvement was a search function for all items.

## View Models

Next, I wanted to make the user home page screen with all their own items, sales and purchases.

1. I created a new MyItemsViewModel. This VM (View Model) will serve as the model for passing many models to the view.  
   
2. Next, I created a MyItems action that accepts item category and search string.
   1. This action creates a new instance of the VM and supplies it multiple models, containing Items, Sales and a list of categories. It then passes this VM to a created MyItems html page.  
        
      
   2. Additionally, in the above action it gets a total of the current user’s sales and purchases.

# Appendix

Figure 1.1

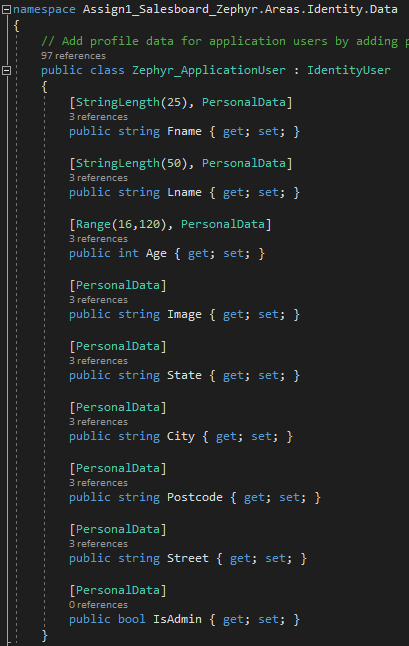


Figure 1.2

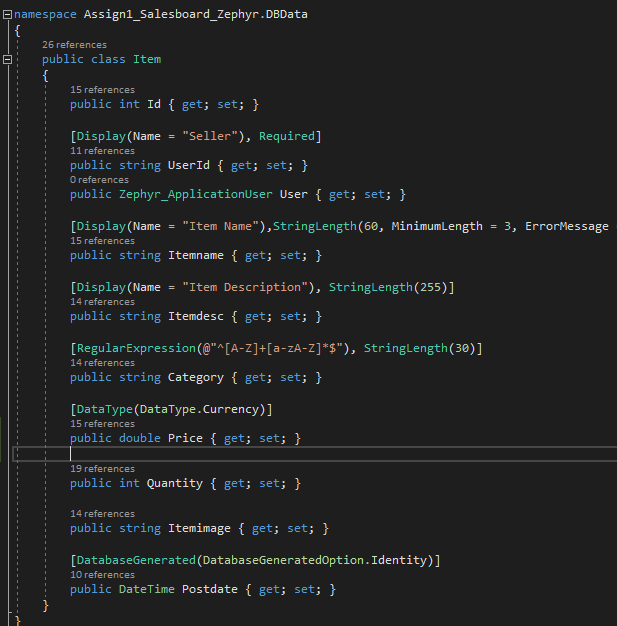


Figure 1.3

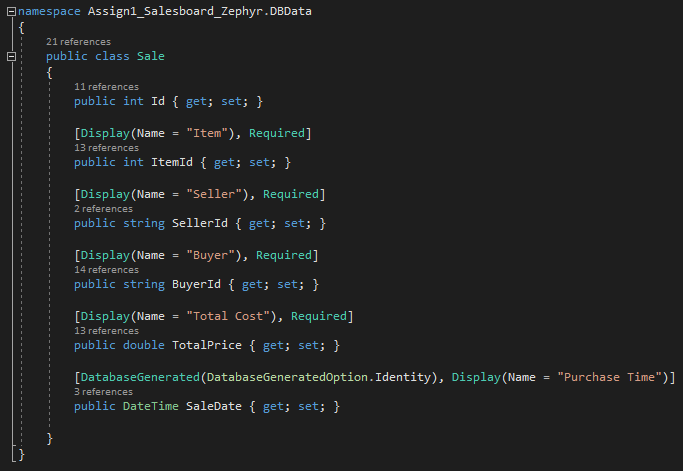


Figure 1.4

