Tanner Croom, Eric Franks, Cal Furgal

Team 4; ISTM 415-501

Spring 2022

Project Final Report

**Key Business Process**

The Key Business Process (KBP) that we focused on in our creation of the code was the

Recording and tracking of the restaurant's sales. We consider this to be the KBP due to the fact that it is highly interconnected with the other processes within the restaurant’s system. It is also considered our KBP due to it being necessary for keeping track of sales at an individual level. This way, we can learn if our daily orders are dropping or growing over time, and possibly analyze the drops or growth the business has to see the cause and how to adjust to them. We also believe that it will help tremendously in the purchase of inventory for the company. Finally, tracking sales will help to show which items should be advertised to customers more, depending on how well they sell.

We developed our code to handle the relationship between the multiple tables by creating unique databases, each with a unique primary key. Perhaps the most obvious place to start for a system focused on sales would be the Order table. The Order table has the OrderID primary key, which uniquely identifies each individual sale for the purposes that were listed in the previous paragraph. The Order table is connected to three different tables, the Customer table, the Order Detail table, and the Employee table. The Customer table is attached to the Order table in a zero-to-many relationship, with there being zero to many orders per customer. It has the CustomerID attribute as a unique identifying primary key, which functions as a foreign key for the Order table to keep track of which customers ordered what. The Customer table is useful for keeping track of customers to see which ones are “regulars” and which come only once in a while. The Order table connects to another table that holds information about the employees of the restaurant. This table would be useful because it allows the restaurant to keep track of what customers the employees have served. The primary key for this table is the EmployeeID, which functions as another foreign key for the Order table. Their relationship is zero-to-many, with there being zero to many orders that a single employee could take. Finally, the Order table connects to the Order Detail table in a one-to-many relationship. However, this relationship has one to many order details per single order. This is important to the order entity as it identifies what inventory items the customer ordered. The Order Detail table has a primary key of OrderDetailID, with foreign keys connecting to the Order and Inventory tables. The Inventory table is one more important part of the database, as it keeps track of the unique food items that they cook and serve. The ItemID is the primary key that keeps track of all of the food that they store, cook and sell. This would help the restaurant with understanding which menu items have been selling well and which have not. The Inventory table is connected in a zero-to-many relationship with the Order Detail table as an item may be part of zero to many order details. It is connected in the same way as the Order table is with the Order Detail table, in that there are many order details to a single item. Each of these tables connect in a unique way, and all of it has been accounted for in our code.

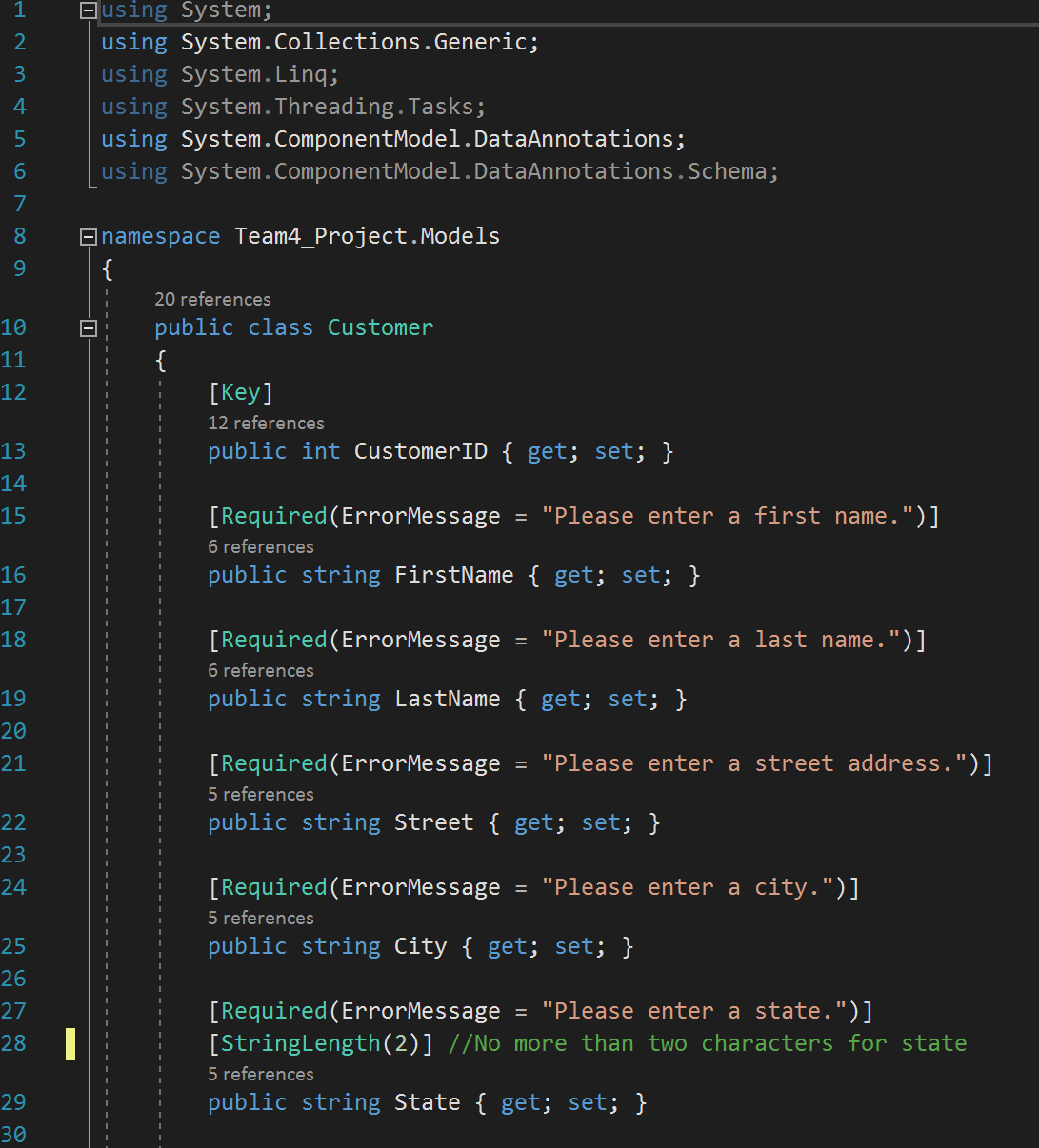
**Code Documentation & Explanation**

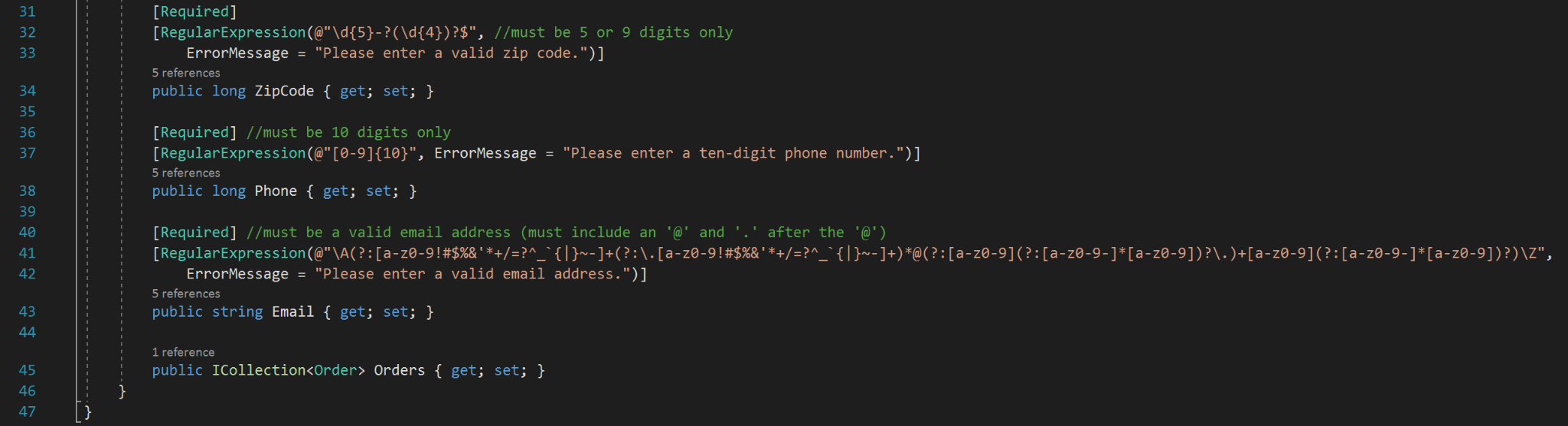
***Models***:

Our application contains twelve total models. Five of these models contain individual properties and definitions for each of the items in the database tables, and another five contain initial seed data for said tables. In addition, one model details the database context, and another was built-in to the application upon initial creation (ErrorViewModel.cs). A couple of these models are described in further detail below:

Customer.cs

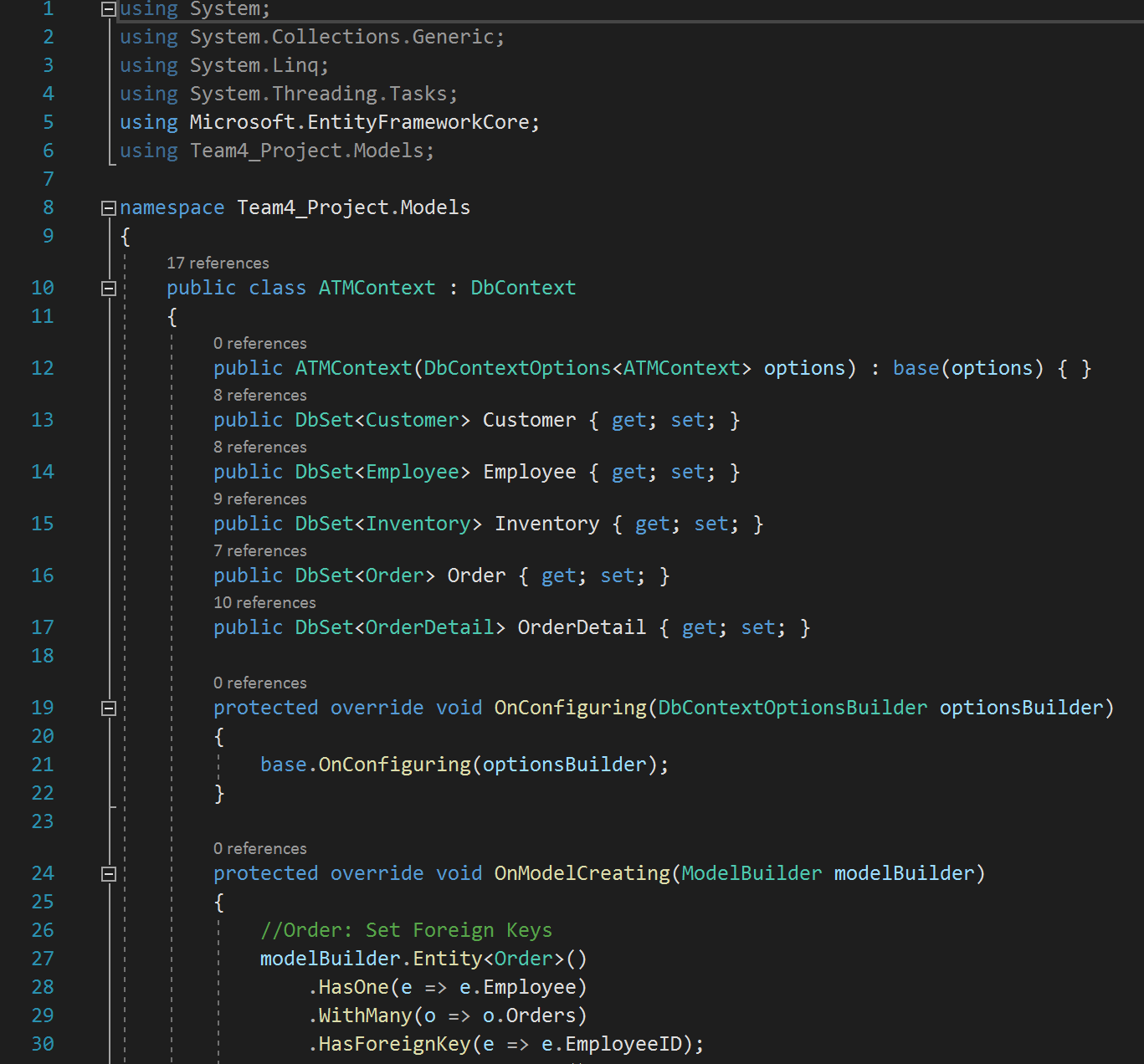
The Customer model (Customer.cs) contains the individual properties and definitions for each customer in the designated Customer table. Customers can be added, modified, and deleted, with each attribute properly validated for user entry. The ZipCode, Phone, and Email attributes also include special validation via the RegularExpression validator to allow for 5 or 9 numeric characters (ZipCode), only 10 numeric characters (Phone), and a properly formatted email address, with an ‘@’ character succeeded by a ‘.’ character. (Email) The model also contains a foreign key navigation property to the Order table, where CustomerID is a foreign key.





ATMContext.cs

The ATMContext model (ATMContext.cs) contains the database context for the SQL Database included in our application, named ATMDB. This context designates each of the five tables for the database, connects them via foreign keys, and establishes a constraint where OrderDetail items are deleted via cascade when an Order is deleted. The context also calls upon the five seed models, which simply inject initial seed data into each database table upon creation.



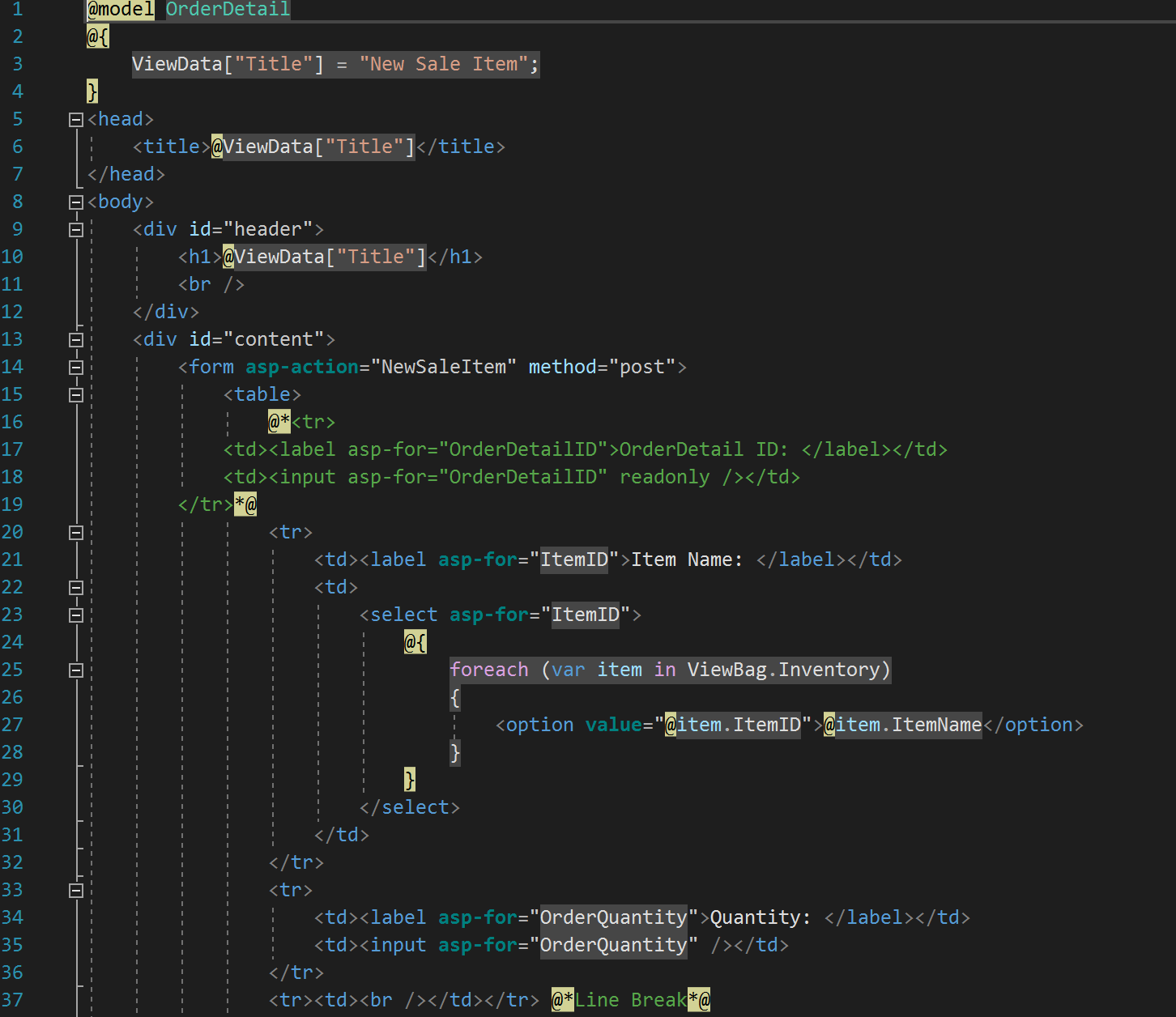


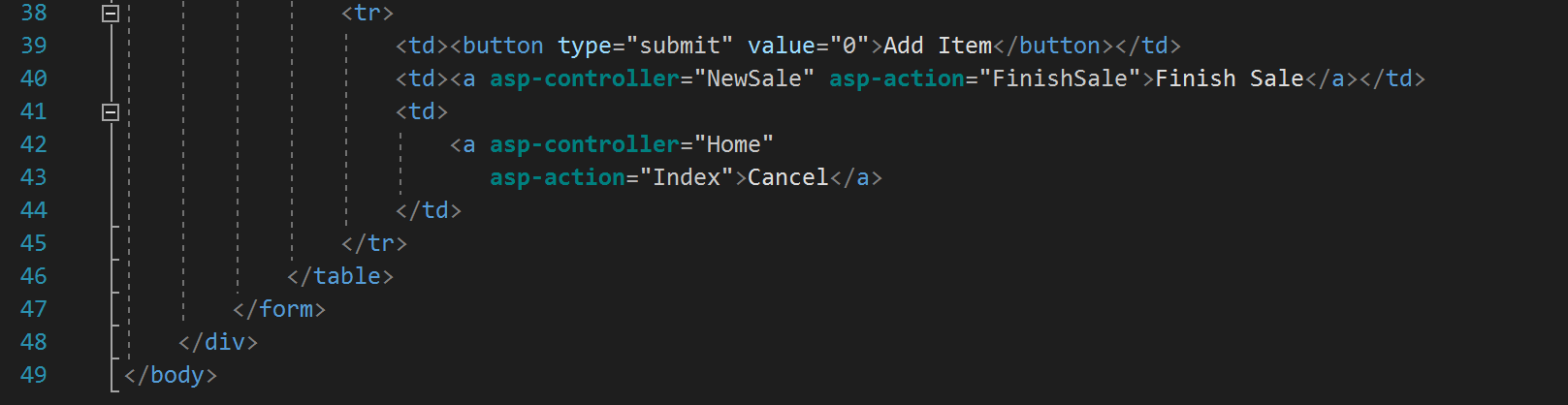
***Views***:

Our application contains eighteen (18) total views. Each of the 5 tables have three corresponding views that are used to interact with their respective table: one that displays the current table, an add/edit view, and a delete view. In addition, there are two views that directly enable the user to execute the Key Business Process, and an Index view that serves as the application’s main menu. A few of these views are described in further detail below:

NewSale/NewSaleItem.cshtml

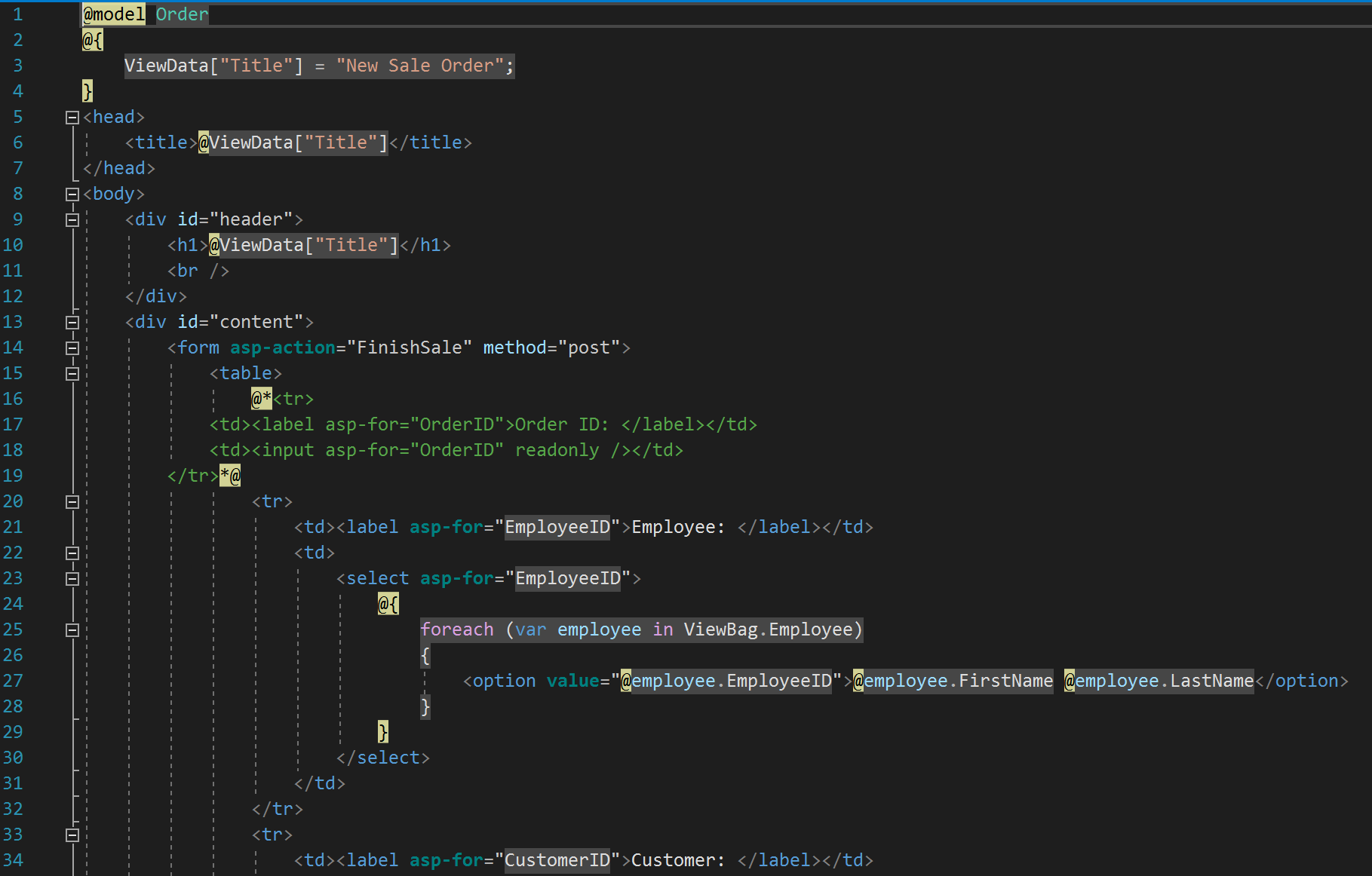
The New Sale Item view (NewSaleItem.cshtml) returns an input screen for the user to add a new order detail (line item) to a new sale/order. On this screen, The user is presented with a dropdown box to select an item from the Inventory table, and an input field to enter the quantity of the item the customer requests. After clicking Add Item, the order detail is passed to the controller and added to the OrderDetail table, with the OrderID field null. In addition, the quantity of the Inventory item specified in the order detail is decremented by the quantity ordered as specified in the New Sale Item view. When Finish Sale is clicked, the controller returns the Finish Sale view.

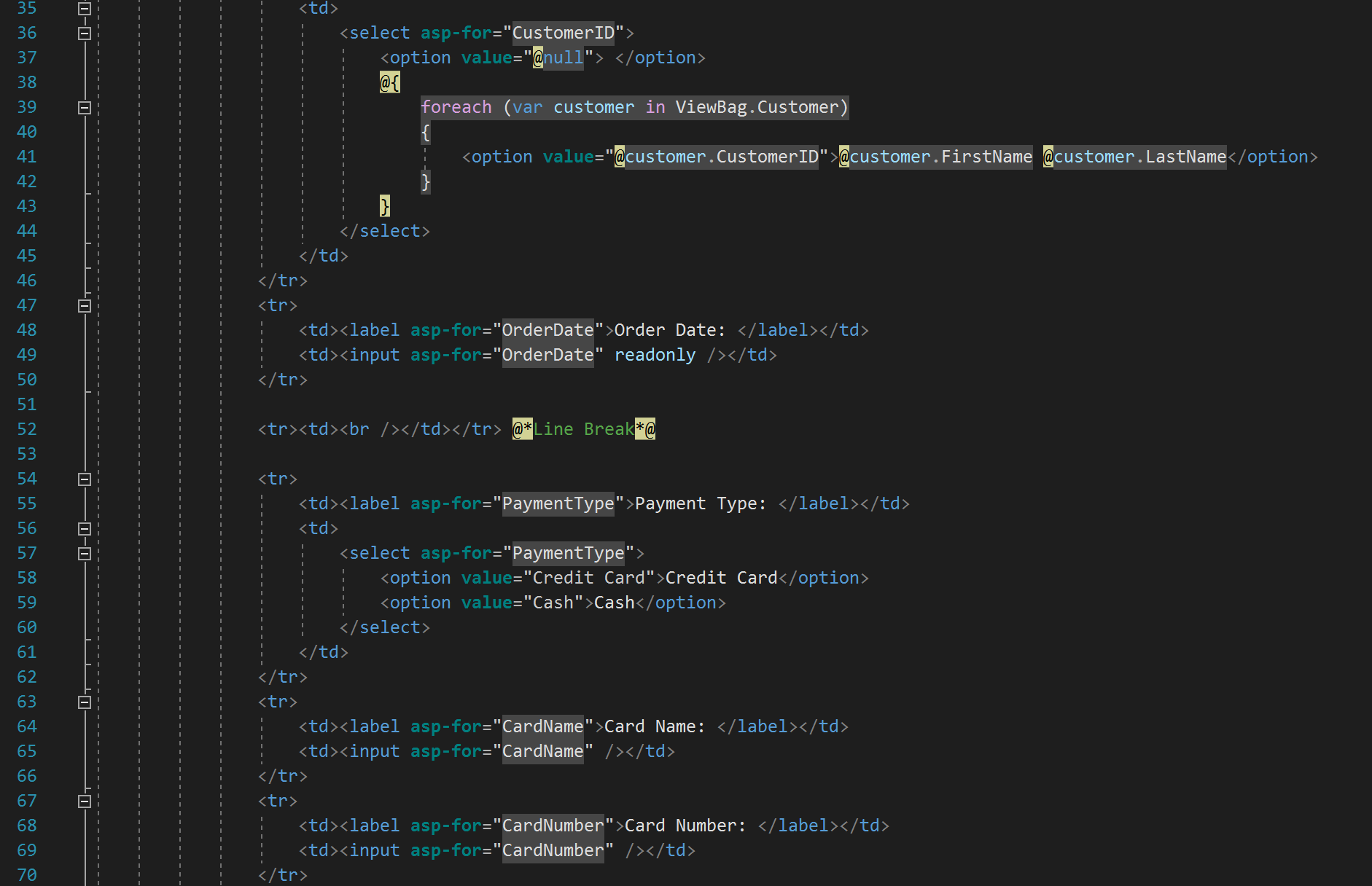


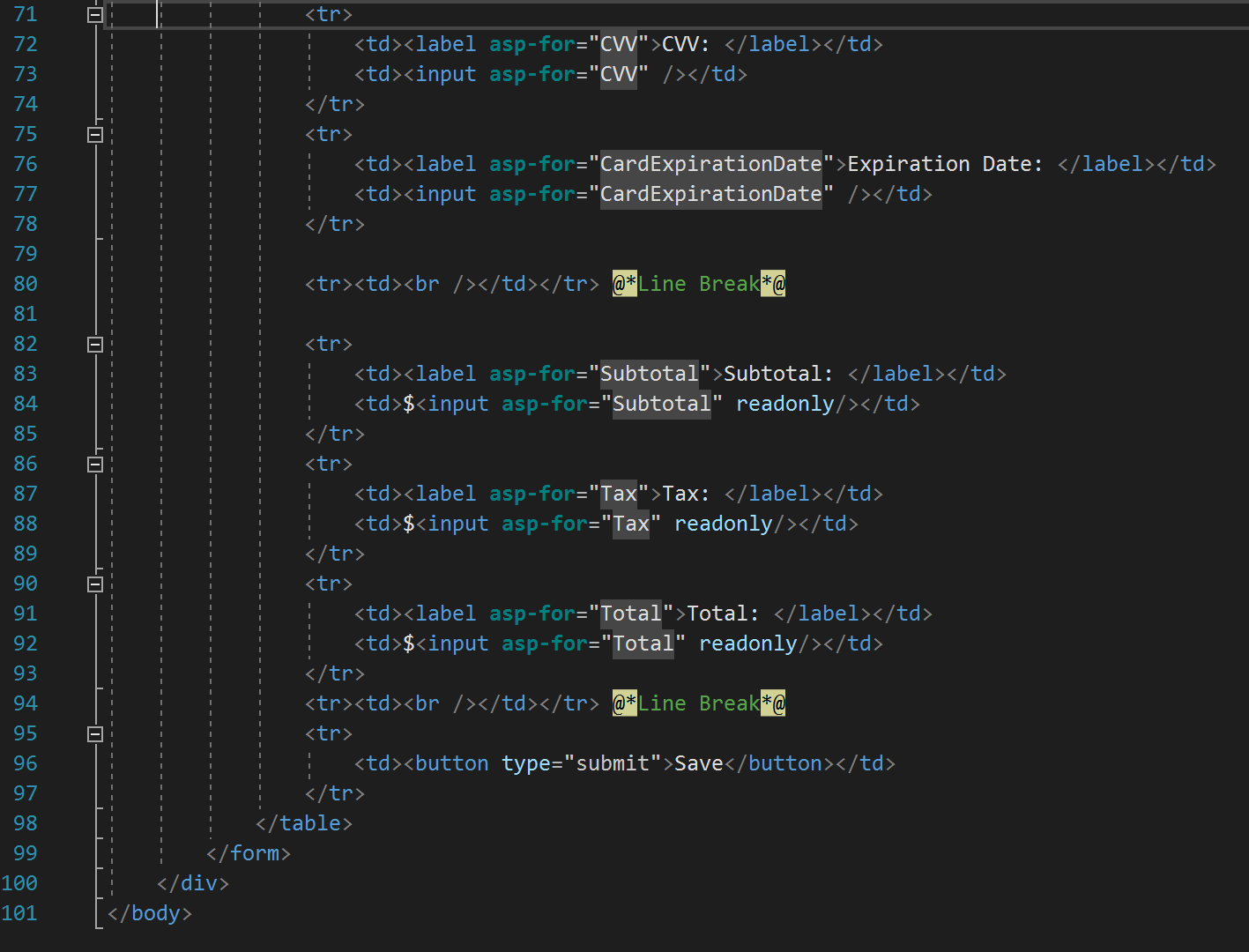


NewSale/FinishSale.cshtml

The Finish Sale (FinishSale.cshtml) returns an input screen for the user to add a new order to the Order table, with the subtotal, tax, and total fields pre-populated based on the subtotal of all order details created on the previous view. The user selects their name from the employee dropdown list, then can optionally select a customer if they are registered in the system. Otherwise, the field can be left null. If the customer is paying for the order with a credit card, that information can be filled in, and if the customer is paying with cash, the user can select that payment type and leave the card fields null. When saved, a new order is created, all order details related to the order are updated with the OrderID of the new said order, and the user is returned to the Order view, showing all completed orders in the table, including the one the user just completed.

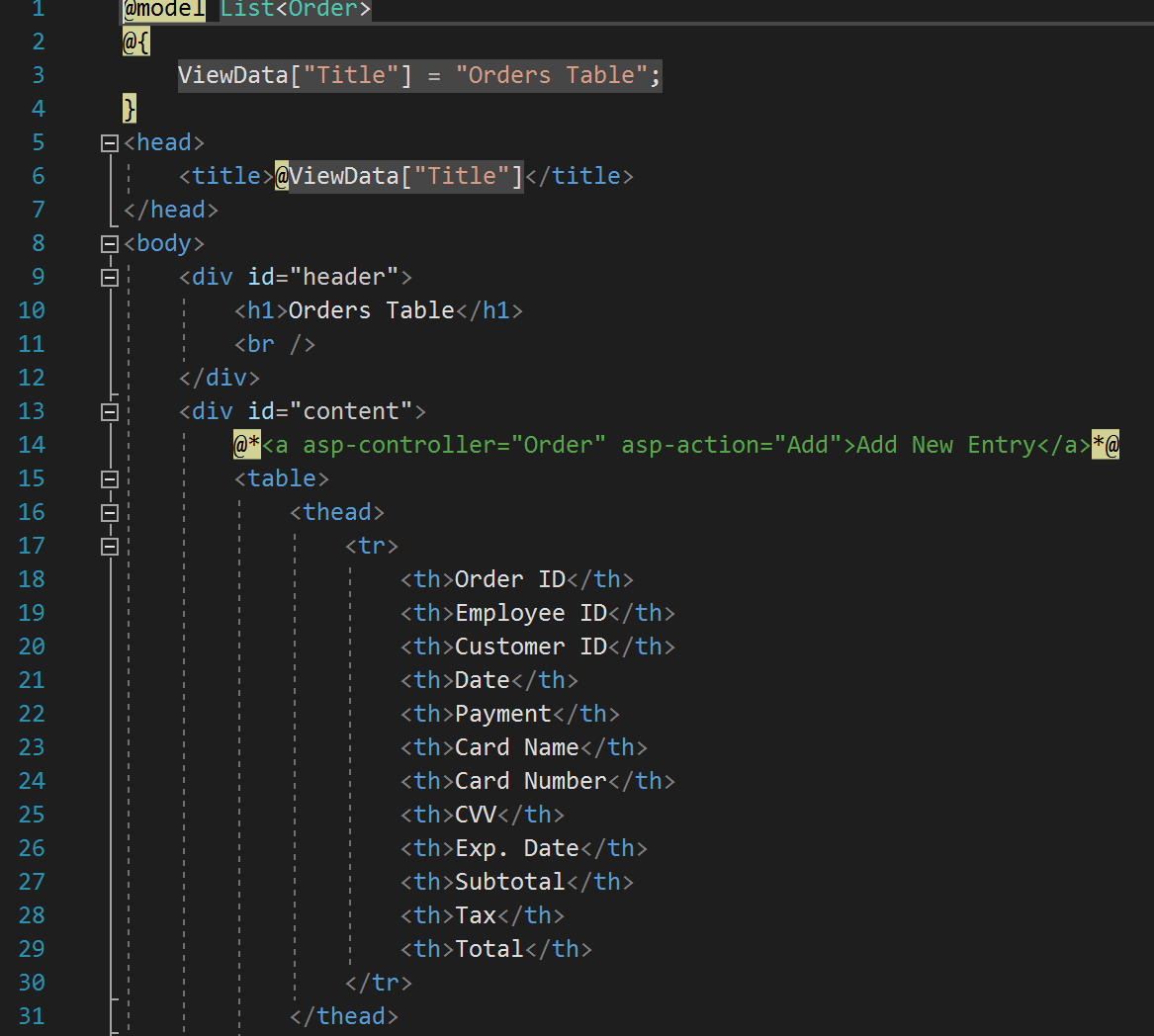






Order/Order.cshtml

The Order view (Order.cshtml) returns the full Order table of the ATMDB database, showing each entry in the table and all associated attributes/properties. From this view, there are links to views for each entry to allow the user to manually edit or delete an individual order if necessary.





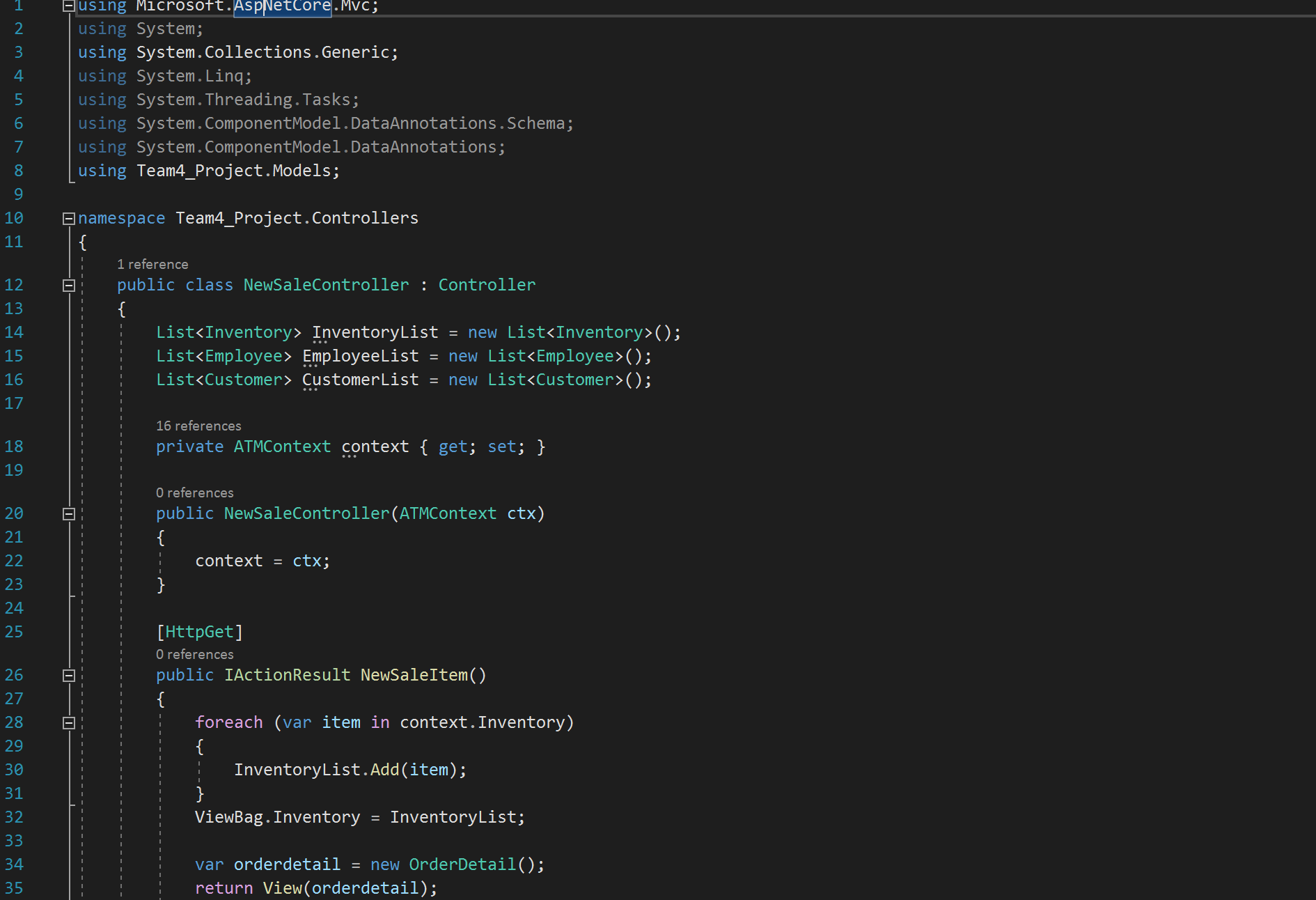
***Controllers***:

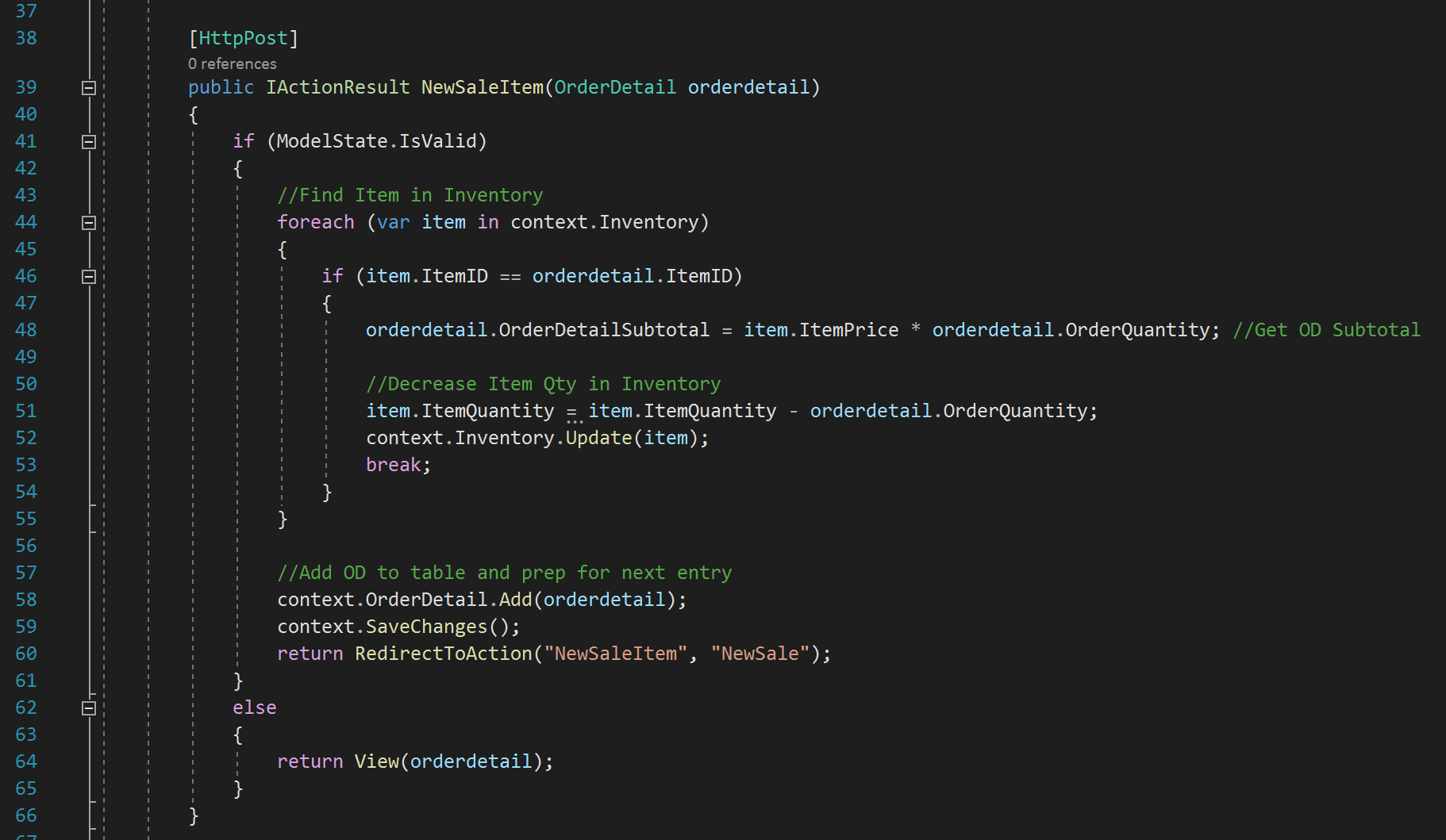
Our application contains seven (7) total controllers. Each controller is responsible for a specific set of views; either for a singular database table, the Key Business Process, or the Index of the application. The most pertinent of these controllers to our Key Business Process, the New Sale Controller, is described in further detail below:

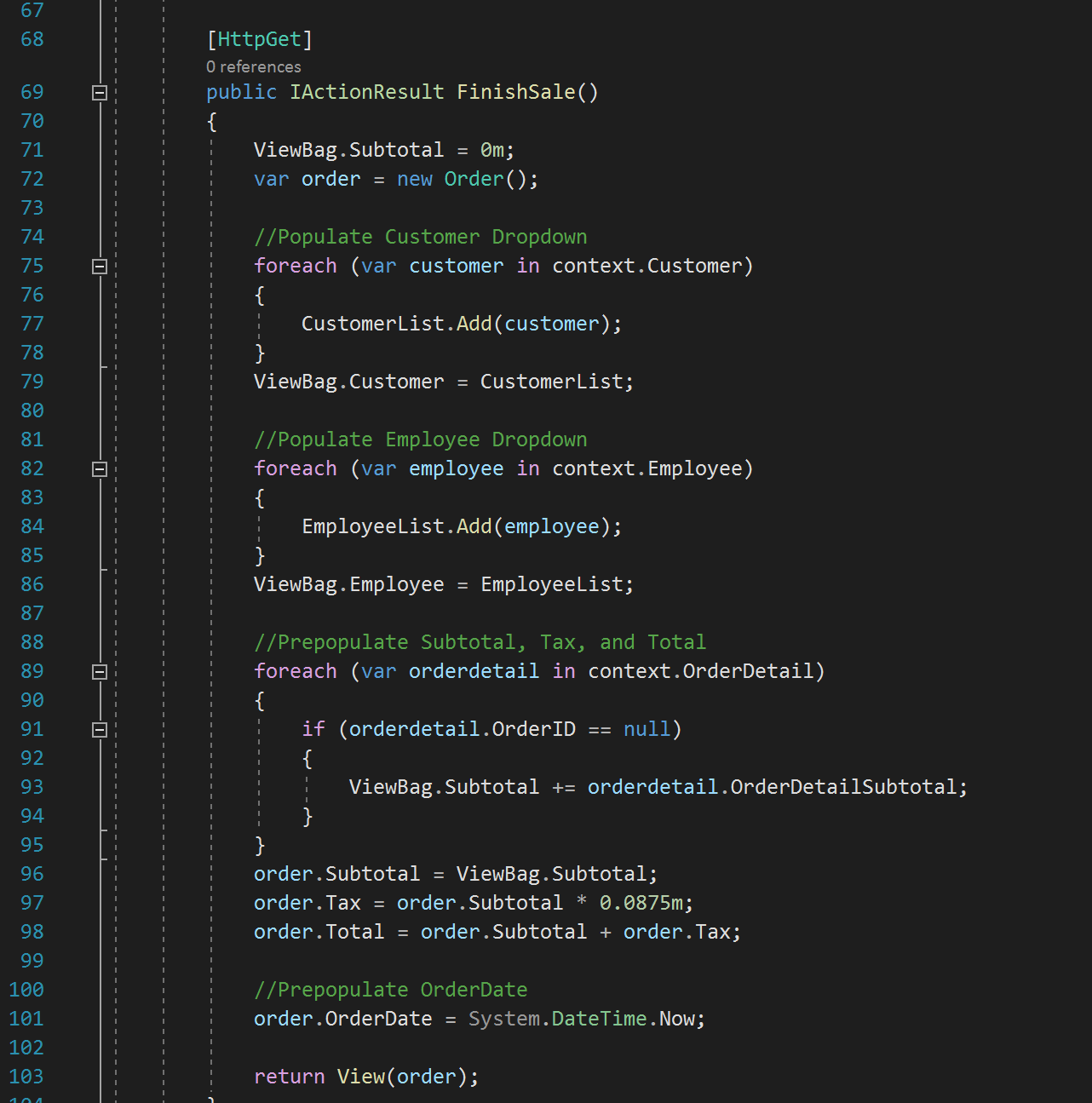
NewSaleController.cs

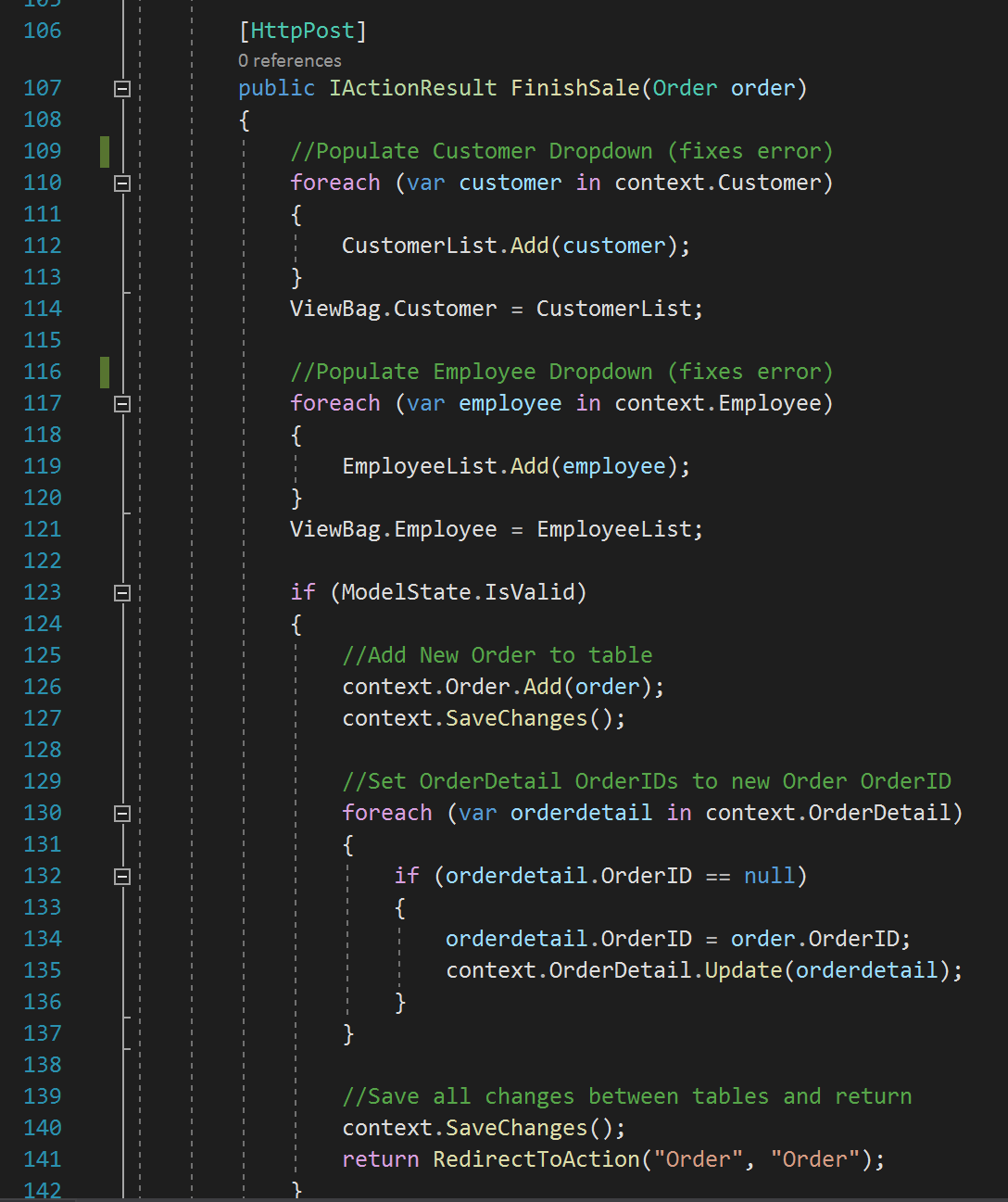
The New Sale Controller (NewSaleController.cs) is the controller for all views and actions related to the Key Business Process of creating a new order/sale. The controller is first used to return the New Sale Item view (NewSaleItem.cshtml) for the user to input and create new order details corresponding to the new order being placed in the system. When the user clicks “Add Item” on the New Sale Item view, the controller uses the user’s input to create a new order detail (line item) to be placed in the Order Detail table (per the OrderDetail.cs model) with the information entered by the user through the view, leaving the OrderID field null, and compounds the order subtotal variable to be used at the Finish Sale view. This also decrements the quantity of the corresponding Inventory item selected by the user for the new order detail, by the specified quantity.

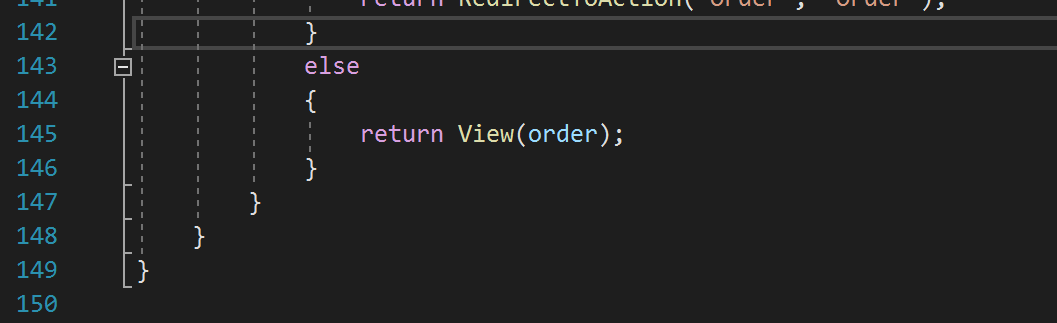
Once the user clicks “Finish Sale” on the New Sale Item view, the Finish Sale view is returned by the controller, with the subtotal, tax, total, and order date fields pre-populated. After the user enters valid input in the view and clicks “Save”, the controller uses the user’s input to create a new order to be placed in the Order table (per the Order.cs model), and updates the order details associated with the order with the OrderID of the new, completed order.











**Final Thoughts**

At our group’s inception, we decided to meet in person once a week outside of class time.

This goal proved difficult given our differences in schedule, and we encountered a number of issues establishing a good time to meet for the entire group. Despite our efforts, this produced lackluster results prior to the due date of the project proposal. We managed to put together a strong outline for the project, however, we were aware the minimal contact between our team would not be enough to accomplish our goal. During our review of the first deliverable, we diagnosed the issue and decided on increasing the frequency of our meetings to twice a week. Furthermore, we eliminated the in-person requirement to accommodate each member’s schedule. Immediately, our team improved communication, consistency, and most importantly, progress towards the final goal. Even if we only talked briefly to establish next week’s goals, our group found that consistently meeting was one of the most important factors in our success.

Evidently, establishing guidelines for group meetings was the biggest hurdle early in the project. However, we faced another issue that was coding specific at roughly the halfway point in the first prototype. Instead of building one database with five tables, we had created five databases with only one table. After realizing our mistake, we came to the conclusion that starting from scratch was the best way to make things right. Although it was the most work of our options, our team took the problem head-on and created a more solid foundation for the rest of the application. Through this mistake, we learned the importance of a solid foundation in programming. It’s always better to get things right at the beginning, no matter how long it takes, as opposed to working around bad code for an entire project. After finding a pair of solutions for our two main issues, our team was ready to tackle the remainder of the project. We divided the material by identifying the group’s strengths and weaknesses from information learned during the proposal. It was clear that Tanner was the most advanced programmer, and served as the chief problem solver during the development of the application. Eric took the lead with mapping out how we would tackle the Key Business process and the relationships for our application, and Calum was able to help build the databases and validate code throughout the application.

During construction of the application, our team was consistently sharing our progress on google drive. More specifically, we would share our work prior to scheduled meetings and talk through the progress we had made. As the most knowledgeable programmer, Tanner would often liveshare his screen through visual studio and lead our team meetings. This allowed all team members to edit and make notes within the application while reviewing the week’s progress.In hindsight, our team did well to identify our issues within the early stages of the project. The earlier you can spot an issue, the earlier you can fix it and move forward. Had we not figured out how to be efficient, the later stages of the application would have presented more challenges than necessary. I can speak for our whole team when I say this was a valuable lesson to learn relative to the team building process. Furthermore, we discovered that prioritizing flexibility and communication was more important than working together through conversations in person.