Thoughts:

When I first approached this challenge, I thought of it in terms of a template that you could add features onto in the future. The first thing I did was sit down with 2 sheets of paper. The first one I used to plan out my classes and some data structures that would be contained in the library. The second sheet of paper I used to sketch out the layout. Although it's basic I am a visual person so I would rather see and picture what I am doing and working with.

Library

I wanted to make a library that could separate out the backend from the front end. This is how software can accomplish things on multiple platforms without becoming a disaster in a rollout of the software. For instance, this is how Microsoft rolls Office for mobile, cloud, and physical application software. That is also why APIs exist as well to give more flexible functionality so you are not constrained to just one front end interface layout.

Since I decided I wanted to have a receipt type layout I thought about what a receipt would have inside it. I came up with different objects that a receipt would contain. The receipt would have a store item and a store item would belong to a store category for existing items. Since this coding challenge isn't extensive in detail I decided that newly added items would go into a miscellaneous category. A future feature could be to place it into the appropriate category bucket. I next thought about what would go onto a receipt such as company, city, etc. I also decided to give items added to a receipt a number like you would see on an actual receipt. I used a random number generator to give me those and a hashset to ensure the numbers were unique. I next thought about how I might need some distinction for each category and created an enum for various categories. I knew this would come in handy later when I needed to either add existing items to the receipt or add existing items to my drop down. Since I decided to incorporate a state sales tax I found the state taxes per state and stored them in a static class in a dictionary. I decided to populate my state dropdown with the dictionary keys (US states) so I could easily find the value instantaneously for each state tax. In the Kitchen and Misc categories, I decided to give them a description, title, and list of items for that particular category. My thinking in this was that it could be used in a future report to pull existing order receipts.

Class Specifics

Category

This has my enum of categories which I randomly chose Miscellaneous, Grocery, Electronic, Clothing, Kitchen

Store Item Interface

I used this to show that different items such as MiscItem and KitchenItem would need to implement methods and properties. For this assignment, I decided to include just properties because a future expansion could utilize methods to implement. This also could have been done as a base class for items but since I was going through different categories this could be expanded upon later and implemented in different ways in different departments. So if the interface was built upon to include other unique methods for each category this is where an interface would be useful.

UniqueInventoryNum

Since this coding challenge was not required to go into extensive features, I used this static class to generate numbers like an inventory number for some item. I used this for newly added items and existing items. I also created a hashset because I didn't want to generate the same number on a receipt. That wouldn't make sense in a real world situation. So I check the hashset and if the inventory item exists I recursively call the method again.

StateSalesTax

This is my static class which contains the dictionary of states and their surge tax rate. In a real world situation this would be stored in a database to retain. There would also be a distinction on items and whether a sales tax was applied or not. For this assignment I just have the sales tax apply to the items on the receipt. I thought a dictionary was a good representation to grab the rates since it grabs it in O(1) based off of the key.

Kitchen Category and Kitchen Item

I thought if I was going to create the categories then a real system might need a description, title, and list of those particular item types. In the constructor, I made random Kitchen items for fun because in a real world situation these categories and items could be populated from adding new items to a transaction or from looking into a previous transaction.

MiscCategory and MiscItem

This is the similar to the Kitchen classes where I store a list of MiscItems in the category. The only difference is since I use miscellaneous only for new items I add, I generate a unique number upon the constructor. I made this property a get so it is generated on the constructor and a user can view and use the number but not change it. That makes sense because in a real life situation a user shouldn't be allowed to edit an inventory number.

Receipt

This would be a heavy class possibly if it was expanded upon further. It contains the company, city, phone number, expiration of the receipt, subtotal, fees, state tax, total, and would contain various category objects. I made this class thinking that in a real world situation it would have some unique id attached to it for adding a new transaction or viewing an existing order.

Front End

I decided not to do anything too crazy since I know it was a limited amount of time and effort to see my thinking and problem solving approach. I decided to make 2 different areas to add items. One area would be to add new items and one to add existing items. I made the new item section textboxes to type in an item and price and choose a quantity. The other section would be existing items which would be items from a theoretical inventory. I chose to use ajax cascading dropdowns for this section to show that I was aware of it and how to use it. The state dropdown at the top needs to be chosen to add the state sales tax in. I decided the section to contain the items would be the receipt. I didn't want it to be plain and boring so I decided to look at a receipt like a Target receipt and generate something with a receipt type font, company, barcode, a random expiration date 3 months from the current date. I also decided to leave in the about and contact pages which are really basic and just a placeholder that a real company would use.

The only extra plugin I installed was facebox and .net projects come with Bootstrap and jQuery so I already had those. So if you try to add an item before choosing a state a facebox should appear informing you to choose a state first. Also, more popups should show if you try to add an existing item before selecting a category, item, or quantity. I made the footer similar to the header because for this it challenges it should still be simple but consistent.

I added a default.css. You will see a mix of css classes, a handle on elements using '#element' and inline css on default.aspx. I wanted to show that I understood the differences and uses in both. I also added in if you make the window half screen, the sections fill up the screen. You receipt and state will be on top and the add sections on the bottom. I didn't do anything extensive because of the constraints of the coding challenge time.

Web Services files and classes

CascDropDown.asmx

This is the class I used to populate the cascading drop downs. I manually filled in the categories. Then for the items I take the category chosen from the first drop down and parse it to the enum defined in my library for categories and use a switch statement to decide which items to add. Afterwards the quantity dropdown populates.

About

Simple about page I used just to fill in that this is where info would go about a company.

Contact

Just has my contact info.

Default

This is the main page of functionality. I chose to keep the state dropdown at the top because that is very important to the functionality since it uses the state tax. I have an ajax update panel for the receipt info so we don't need to do a full postback each time we add items to the receipt. I have the triggers as the button clicks as well as the selected index change for a state. If you change the state it will update on the receipt. I also update the total if you change a state to a different state.

For the new item section, I added newer elements such as the pattern, title, and placeholder. If it doesn't match the regex defined, you can't continue adding the item. I did this for the name so the name will only accept characters upper and lower a space. The price expects a decimal with a period like ‘2.00’ or ‘150.65’.

I also decided to create a simple js file with the state check to show I am aware of internal js and external js files which you could serve and minify/compress. On the button clicks to add an existing or new item the function runs to check if a state is selected. If not, you get a facebox message. On the new item if you don't put a name or price I just did an alert window to show I’m aware of those as well.

If there was more time to spend on this, I would have included the js file and my css file in the bundle and rendered it minified.

Since I was limited on time I have static tables and a receipt object. If I wanted to persist the receipt and items through a post back, I could have cached the object or could even store it in the viewstate if I serialized the classes. Although I know the larger viewstate the more it would hinder performance. I also added a page init with gzip compression. I know this could be done through server side settings but in this case I may not have a handle on that so I decided to add it programmatically. In my pageload, on the initial load, I populate the state dropdown and initialize my tables for storing the items. On the postbacks I re-add the category tables to their placeholders so no items go missing between postbacks. I also create a receipt object and add the data to it. I didn't continue going down this path because it would have taken a lot longer to persist this object through postbacks but in a real world situation I would continue to add the proper objects to my receipt.

On the add new item method, I first check if the boxes contain a name and price. If not, I display an alert. If yes, I create a new miscellaneous object which populates from my library class. I populate the object from what I entered, add it to the receipt, update the subtotal, then total.

On the add existing items, I check which category was selected then go to the appropriate add method. These have a check if the item and quantity is selected as well. Since I do not have the types implemented for these yet I just created strings and decimals and then called the method to add them to the receipt. Next I update the subtotal and then the total. Although I have a kitchen object in the library I decided to stay consistent with what I did for the other categories since they are not implemented yet.

Overall Goal: I wanted to show various aspects of coding including ajax, some OO, uses of different data structures, and a mix of the front end as well. My over goal was to create classes and a layout template that could be built upon later. That is the reason I created the library of classes which separate out from the front end. My goal was to show okay here is what to start with and afterwards be able to say okay I can continue building the categories, item types, and modify the classes to use a database. Additional functionality would be required to persist the receipt object through the postbacks. If someone were to take this and build on to scale it up they would need to do things like add thread safety, maybe build some background processes, add pages, figure out which architecture between a data center or cloud based deployment. For a data center based scaled deployment maybe the backend could use HAproxy for a TCP/HTTP load balancer. Caching of objects is important so maybe a company would want to use some Redis or Memcache to cache objects. They could scale a full text search for items using elastic search too. They also have the ability to take the library and use it with MVC, web forms, etc. They would also want to minify the js, css, maybe even have some cdns host and give static content. I only decided to do a web form project because I knew I would be able to display different types of skills: interface, static classes, enums, class library, ajax, jQuery, use different data structures, using class library in my front end.