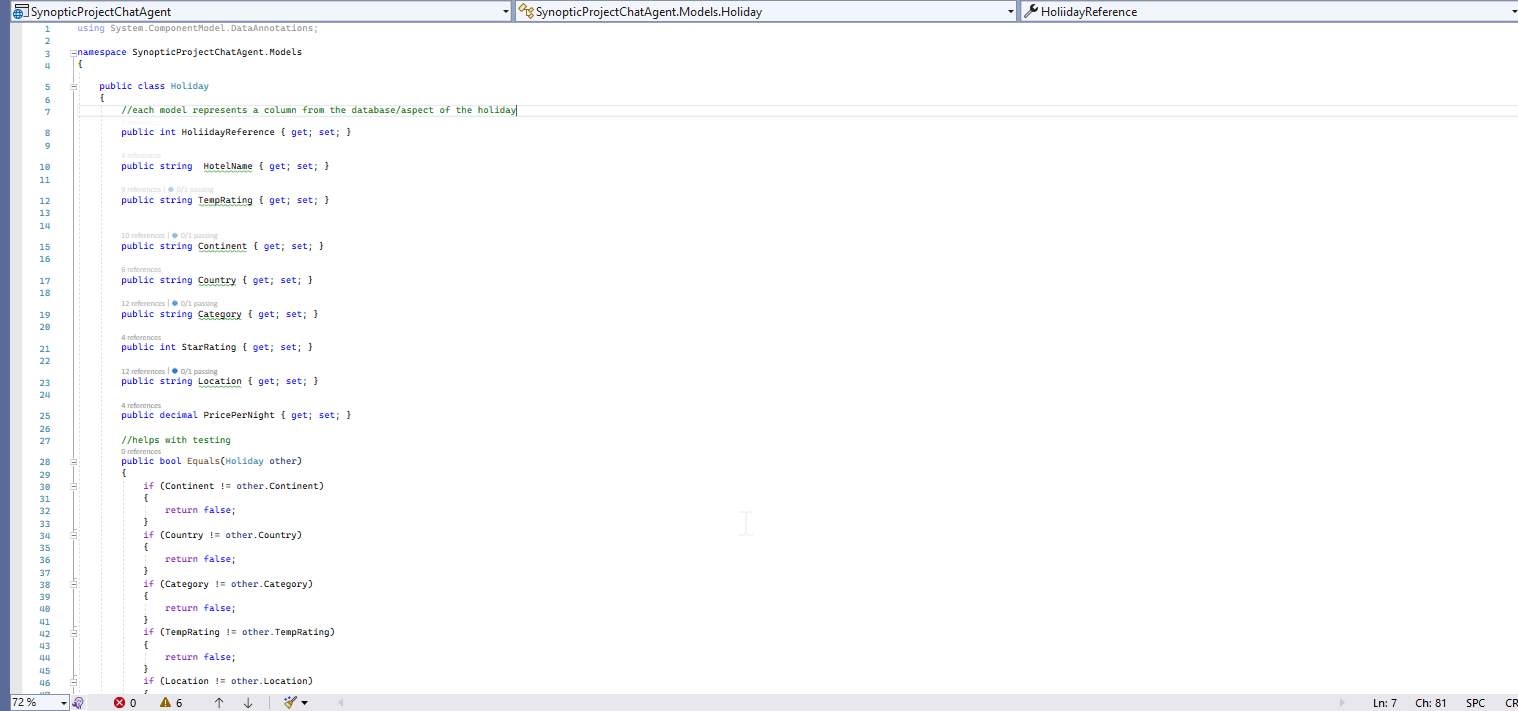
Synoptic Project

I have created an MVC app as the synoptic project.

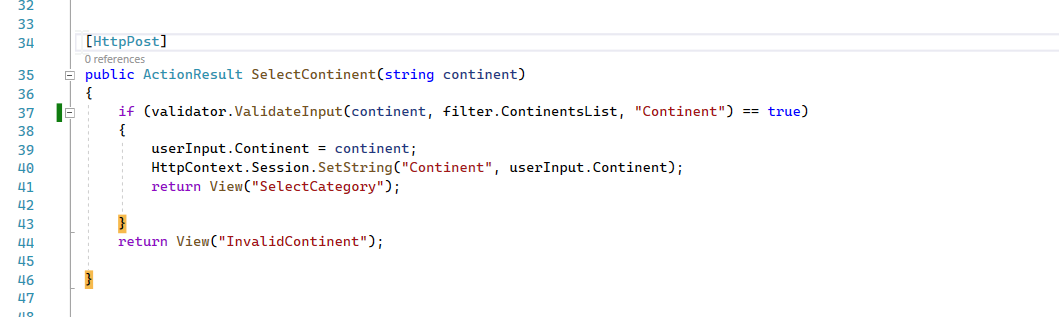
Model

This is my model class of ’Holiday’. It is public, so it is easily accessible outside of its class. The properties are the same as the columns in the data document. I use this model to assign data to its properties, filter and display data.



Controller

I have used the home controller to map my website. When the user opens the page, they are brought straight to the ‘Select Continent’ page.



I’ve used [HttpPost] here because I want to post the data from the view into my saved session. On the front end, the user enters their input into a textbox on the webpage and then presses submit. If the Boolean statement on line 37 returns true, HttpContext.Session.SetString(“continent”) will save the continent in a session that can be accessed in another method.

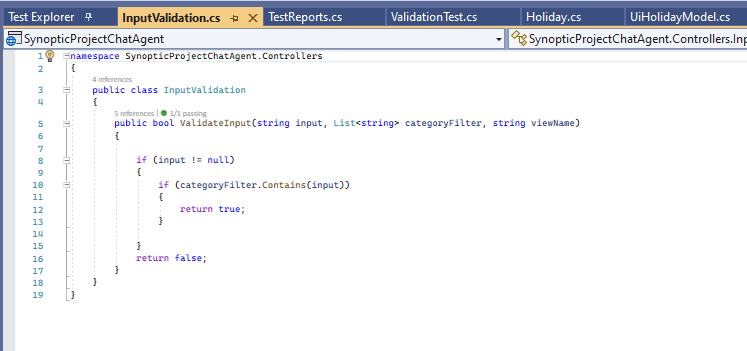


I have a similar method for each page in my View folder. I assign variables foreach filter by using the session state I created in each method. The variables are then passed into my GetFilteredHolidays method, which returns the data.

Model

Validation

The ValidationInput method is in its class. It takes in an input, a list of valid answers, and a view name.



I was happy that I was able to create a method that would successfully do what I needed. On retrospect, I wished that I could have created my own custom data annotation that wouldn’t allow the users to enter that particular input at all. In addition to this, it takes a third ‘View’ parameter but doesn’t use it. This I would remove in future versions.

Below is the list of valid answers.

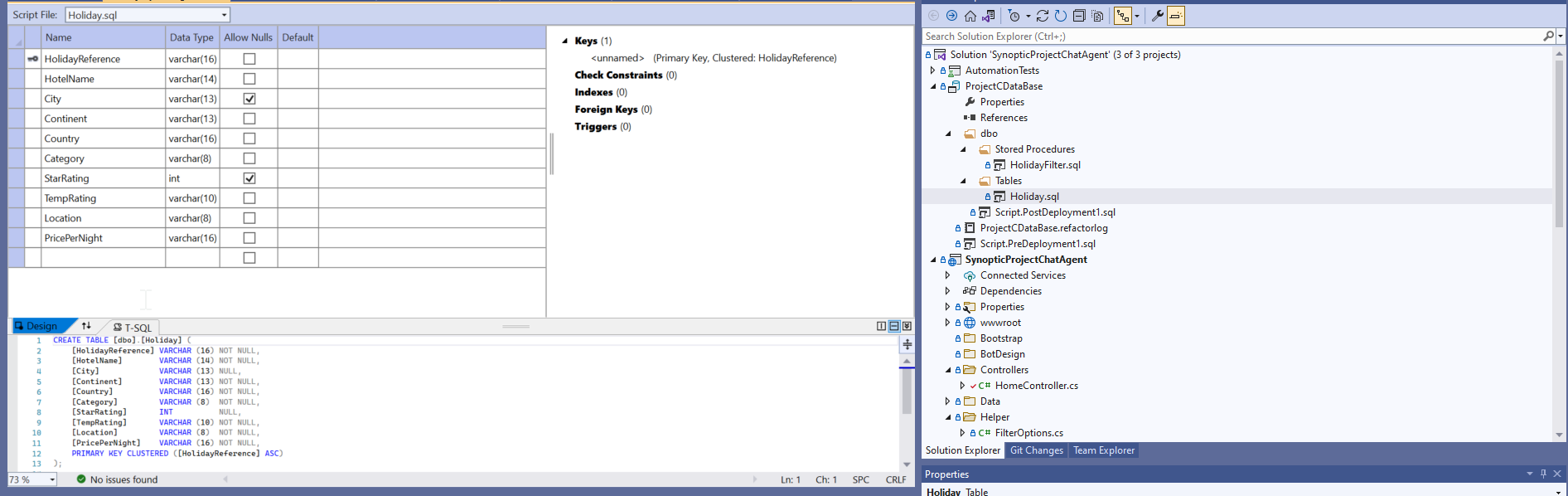


The validation method checks the input against each of these strings. This could be improved by maybe using a get request to look at the internal database to see which holidays we cater for and then return results based on that. The way I have implemented it is weak because the user could potentially spell the word correctly but with some capitalisation and the system would reject them.

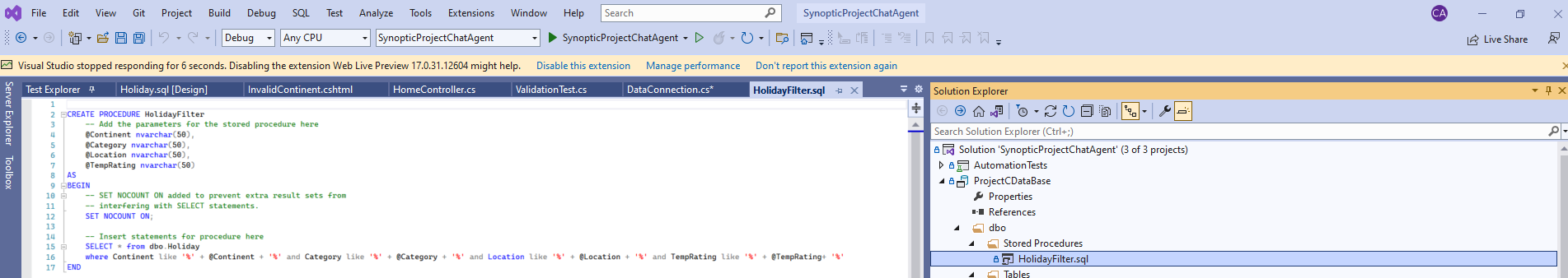
Model and Data

I am going to explain how the app fetches the data.

After creating the database, I added a table with columns that replicate the columns that were in the excel data file provided to me at the start of the project. This is how it looks.



After inserting the data, I created a stored procedure that would filter out the data based on the parameters that it was fed.



This stored procedure accepts the parameters of the continent, category, location, and templates. Then it selects all holidays where the parameters are exactly like what is passed through.

After that, I published the database, so that it was on my computer to use.

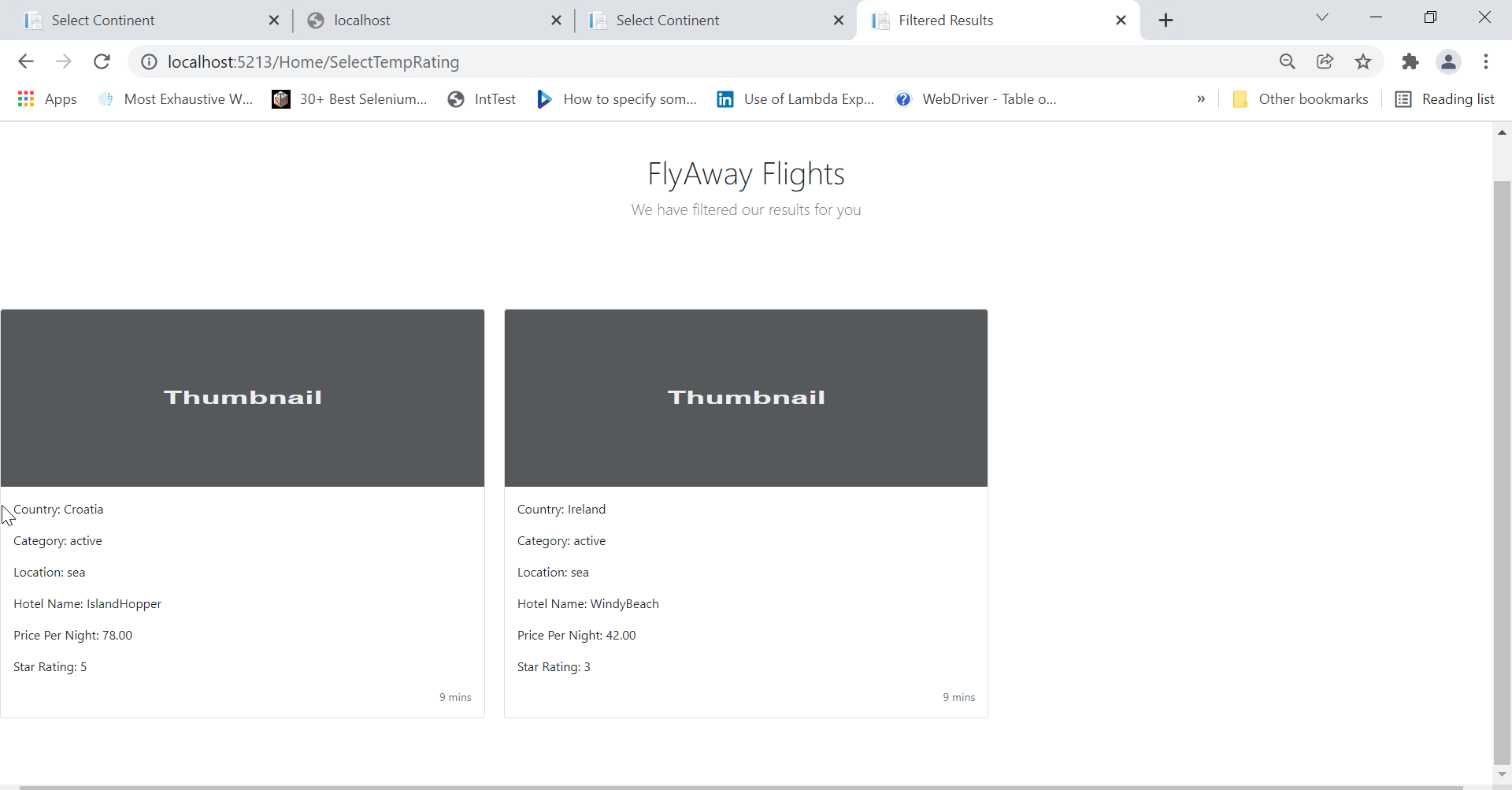
I created a separate class called ‘data connection’. This held the connection string and a method called ‘GetFilteredHolidays’ that would return a list of Holidays. Inside of a using statement, the code connects to my database, using the connection string, and is able to execute my stored procedure to return the list of holidays, ‘recordscreated’. I have used a library called Dapper to help out with the passing of data.



The method above is then called in my HomeController once all of the parameters are returning values.



In which I pass the list of the holidays that were created into my view named ‘FilteredResults’ where it is displayed to the user.



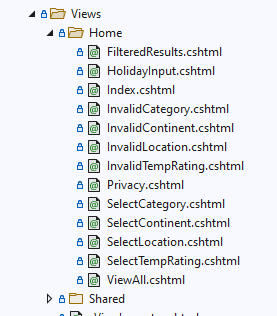
Data – On reflection

I did not think about implementing security measures within this proof of concept. If it was taken further, I would need to be cautious of SQL injection.

This met the criteria in returning results, so I am happy about that. I wouldn’t have changed much, the only thing would be filtering via continent, purely because it limits the number of results that the user will see- however, is a full-scale application, this may not be a problem.

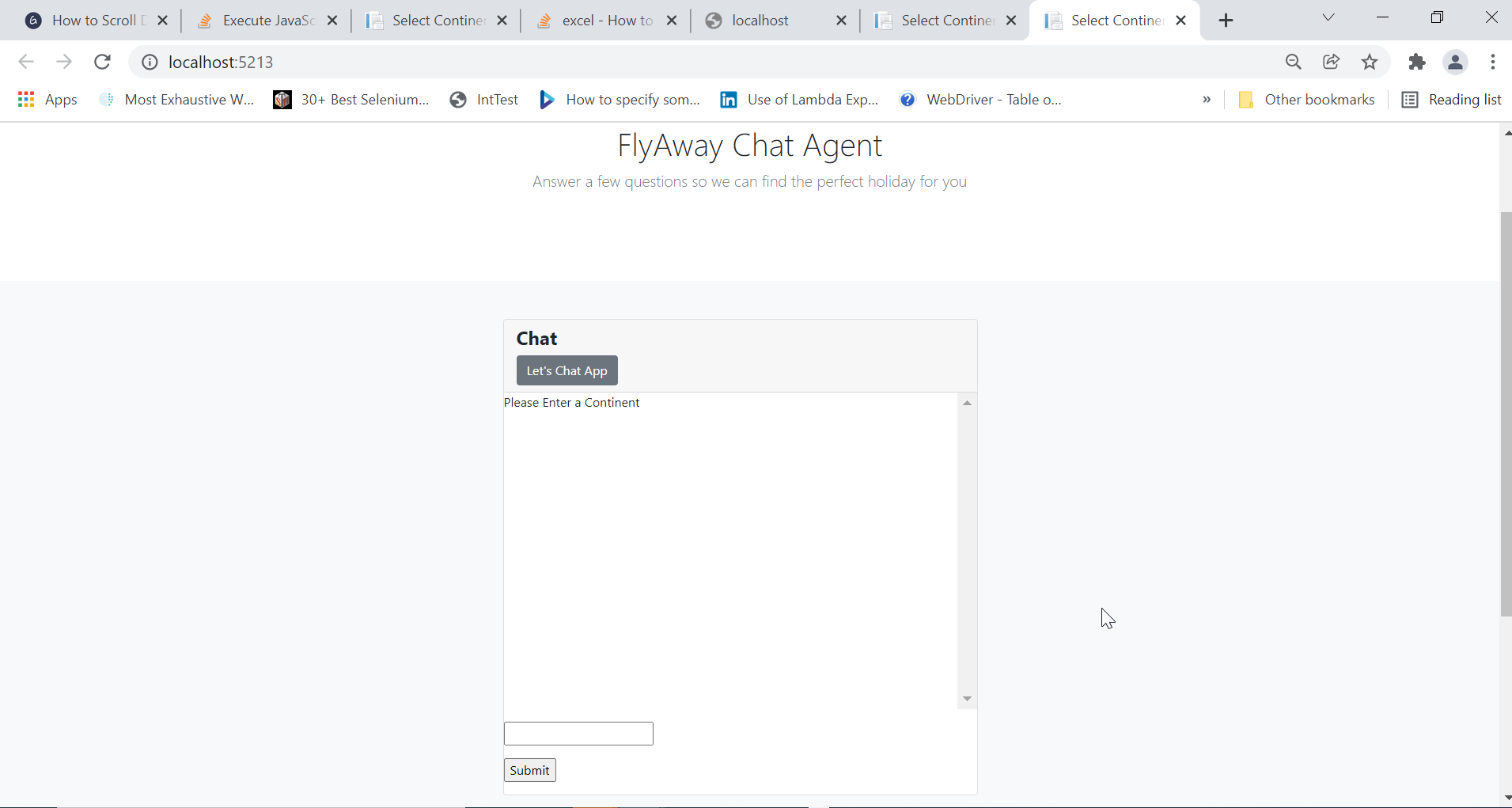
User Interfaces

An MVC app contains ‘views’ that are called from the home controller. I have created a view for each of the pages that the user will experience. In the view all and Filtered Results view, I have passed through the Holiday model that I had created prior.



When developing the user interface, I used HTML and referenced a cascading style sheet (css) from Bootstrap. I took inspiration from a pre-existing template that I found online.

This is a screenshot of how the user will see the interface

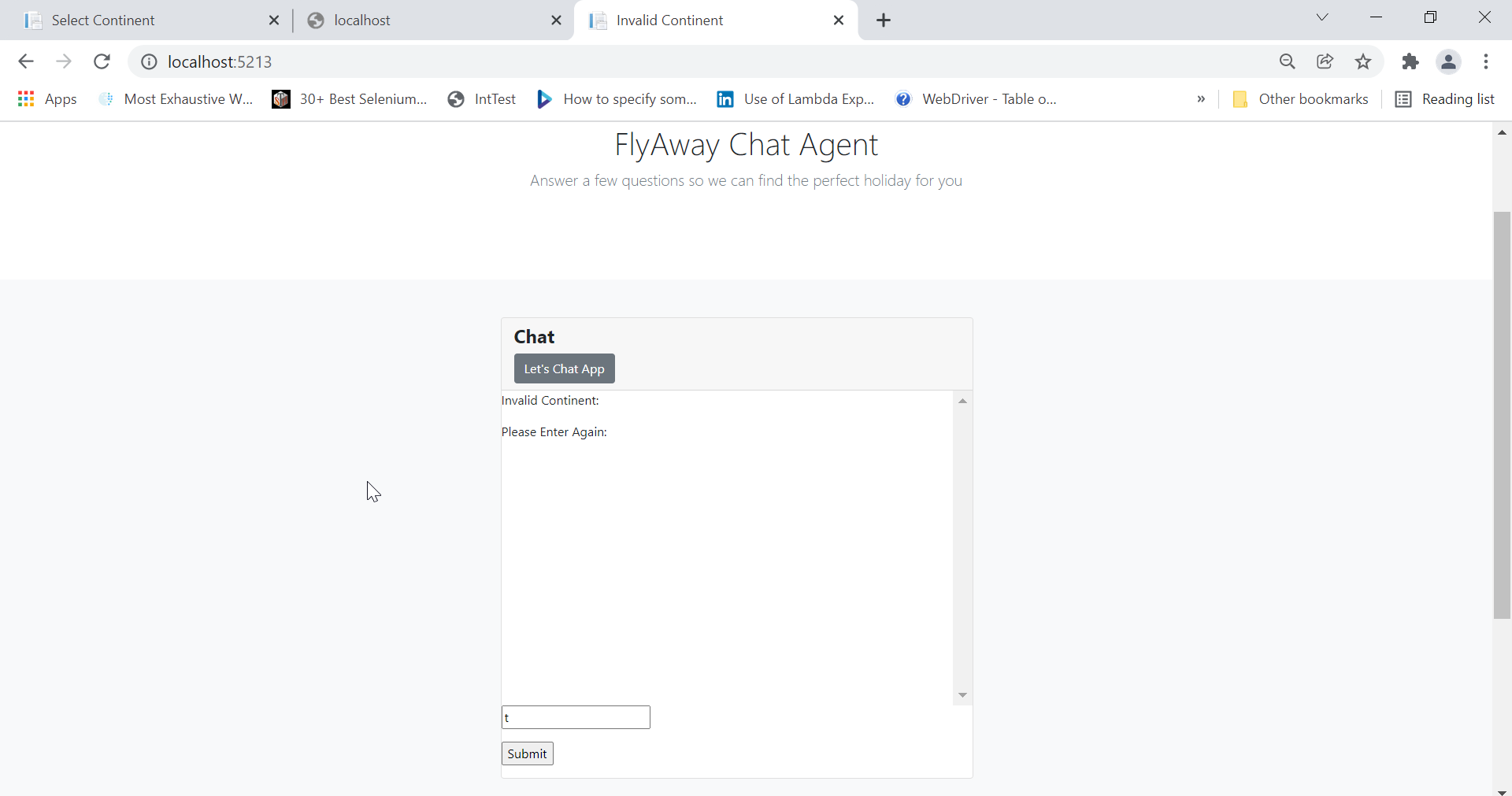


All of the text and user input details are wrapped inside a container to anchor it to the same position on the screen.

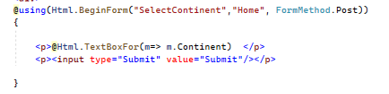


I have used *“@using(HTML.beginform(“selectcontinent”,”home”,FormMethod.Post)”* to allow my user to input a string inside the text box and send that string data, via a post request, so that it is stored in memory. I’ll come back to this later.

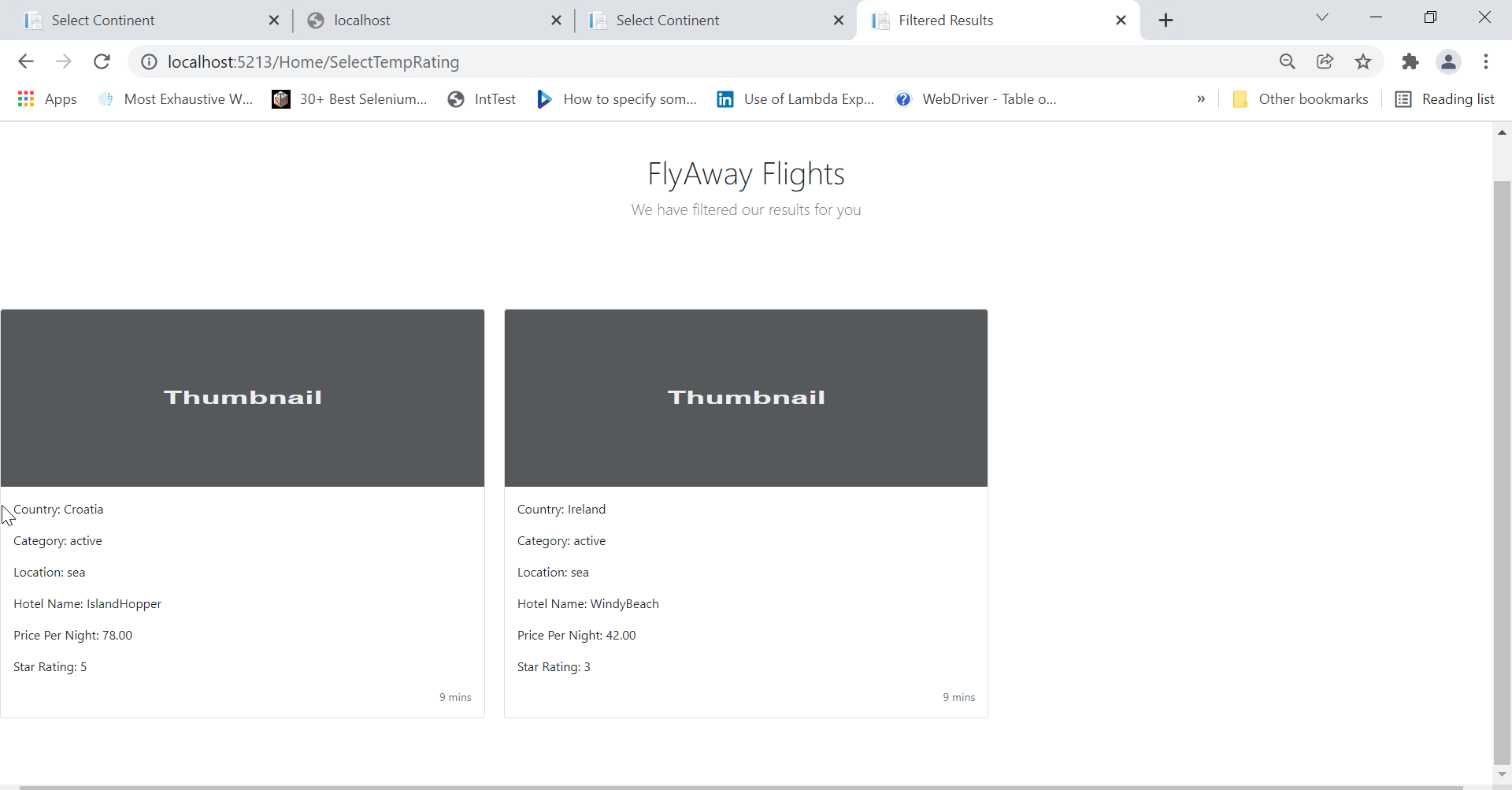
I have restricted what inputs are valid. When an invalid input is entered, the user will be redirected to a page saying ‘Invalid’ in the title. Like below.



The invalid continent page has the same ‘BeginForm’ method as below because that is the method that validates whether an input was correct or not.

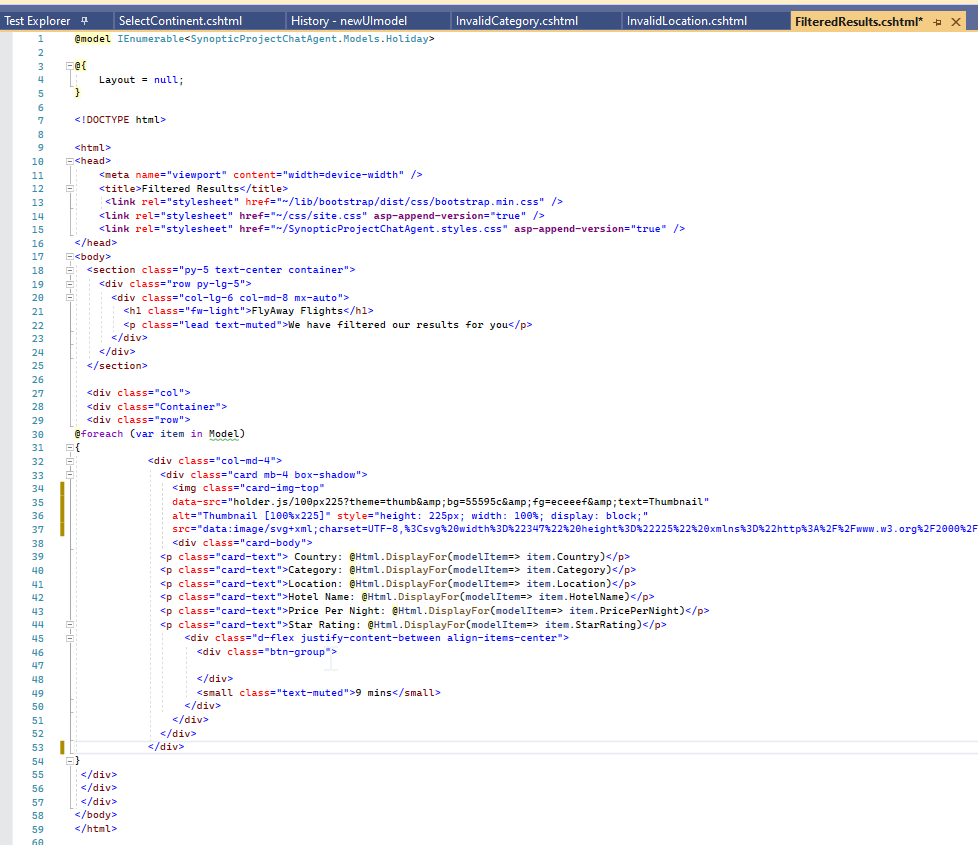


I have used another type of view to display the results that the user will see. Here is a screenshot



The user gets to see the Holidays that are returned in a card format. Inside each card contains the properties from my model class and the data that has been returned from the stored procedure.

Here is the cshtml from this page. I have passed through my Holiday model as an ‘Ienumerable’ of type ‘Holiday’ because I want the page to show the list of holidays that were returned from my data connection.



I have written a foreach loop that will iterate through each item in the IEnumerable of ‘Holiday’ that I passed through as a @model. This code will produce a new card for each holiday in the list.

User Interface – On reflection

I’m happy that I have a functioning user interface that is easy for the user to understand, however, there are many things that I would change about this if I had the time to do so. There are options on Asp.net core to create a ‘Layout’. This is HTML that you can reuse and import into the project.

I have little experience with User Interfaces and was unable to successfully create my layout and reuse them. I would consider copying the code in each view to be bad practice when it comes to development.

Another thing I was unhappy with was that I was unable to implement a way to show the user what was input into the chatbot previously. I tried to use the HttpSession.Context.GetString to show what was stored in the session state, but ultimately failed to implement it without damaging the code. I’m disappointed in this fact because this was highlighted as a ‘Should’ under the acceptance criteria.

Finally, If I created this again, I would have looked at implementing javascript. This would make the page dynamic and more like a standard chatbot. At work, I only use C# so this is something that I am excited to learn about.