MyZerodhaTask Documentation

Task Performed:

a) Web Scrap
b) Extracting and file reading operation
c) Design using MVC (**M**odel – **V**iew – **C**ontroller) structure
d) Web search operation

While going through the task I've encountered with the following *constraints*:

a) File to be downloaded prior to current date
b) File unavailability during weekends i.e. *Saturday* and *Sunday*

The above constraints has been smartly handled while performing schedule download as well as proper user message on the web page during the file unavailability.

This documentation contains brief explanation about the tasks i.e. executed in *Dot Net* platform with the application of C# programming and MVC.

Web Scrap

The objective is to hit the chrome browser with url, and download the *Bhavcopy* equity file prior to current date which is being achieved with *selenium web driver* package.

Fig 1: Web Scrap Code

Extracting and File Reading Operation

A. Unzipping the file

Before we go with extracting the file, we need a library i.e. *IO.Compression* as well as need to ensure location for both the path for zip file and unzipping the *.zip* file to extract the *.csv* file.

```
1 reference
public static void ExtractFile(string fileName)
{
    Console.WriteLine("Extracting");
    try
    {
        string zipPath = @"C:\Users\HP\Downloads\" + fileName;
        string extractPath = @"C:\Users\HP\Downloads";
        ZipFile.ExtractToDirectory(zipPath, extractPath);
        Console.WriteLine("Process complete..");
    }
    catch (Exception)
    {
        Console.WriteLine("Some error occurred");
    }
}
```

zip file location

location for unzipping the file.

file extraction performed.

Fig 2: Extraction Operation

The above two operations is consolidated to a C# console application and scheduled in the Windows Task scheduler with a timer 6:00pm for every *Tuesday*, *Wednesday*, *Thursday*, *Friday* and *Saturday*. Two days of the week i.e. *Sunday* and *Monday* has been ignored because considering the second constraint, the files aren't available on the web.

B. Reading the File

After we extract the *.csv* file successfully, it's time to refresh the page and proceed with the file reading and populate data with required equity details. Before reading the file we need to package i.e. *System.IO* to support the *StreamReader* class and read the file.

```
1 reference
public static List<Equity> GetEquityList()
{
    List<Equity> equityList = new List<Equity>();
    DateTime myDate = DateTime.Now.AddDays(-1);
    string fileName = "EQ" + myDate.ToString("dd") + myDate.ToString("MM") + myDate.ToString("yy") + ".csv";
    using (StreamReader myInput = new StreamReader(@"C:\Users\HP\Downloads\" + fileName))
    {
        //We ignore the first header line of csv file, hence had read before getting into the loop
        string headerLine = myInput.ReadLine();

        string row;
        while ((row = myInput.ReadLine()) != null)
        {
            string eqId = row.Split(',')[0].Trim();
            string eqName = row.Split(',')[1].Trim();
            double openVal = Convert.ToDouble(row.Split(',')[4].Trim());
            double highVal = Convert.ToDouble(row.Split(',')[5].Trim());
            double lowVal = Convert.ToDouble(row.Split(',')[6].Trim());
            double closeVal = Convert.ToDouble(row.Split(',')[7].Trim());
            double lastVal = Convert.ToDouble(row.Split(',')[8].Trim());
            equityList.Add(new Equity
            {
                EquityId = eqId,
                EquityName = eqName,
                OpenValue = openVal,
                HighValue = highVal,
                LowValue = lowVal,
                CloseValue = closeVal,
                LastValue = lastVal
            });
        }
    }
    return equityList;
}
```

reading .csv file using an object of *StreamReader* class

getting the required data from the .csv into the variables

adding the data into the list.

returning the list.

Fig 3: Reading the file

Design using MVC structure

Finally after arranging the file it is required to be produced to the web page which is achieved by using the MVC i.e. **M**odel **V**iew **C**ontroller structure

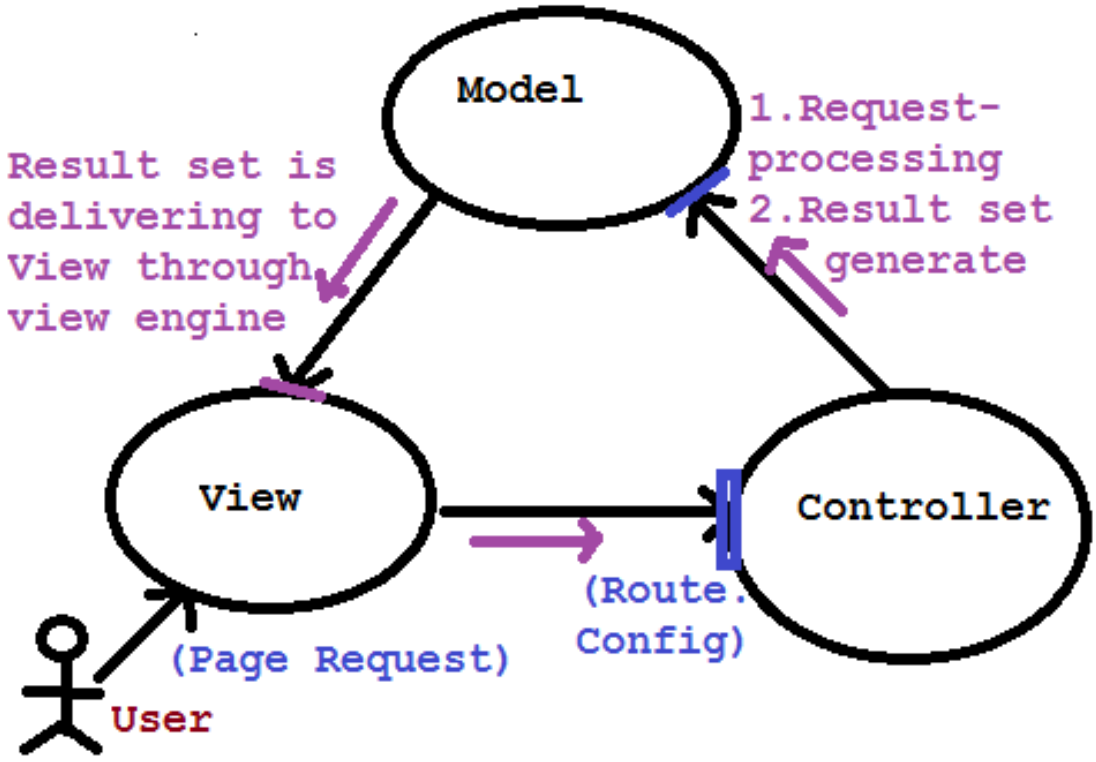

Fig 4: MVC structure

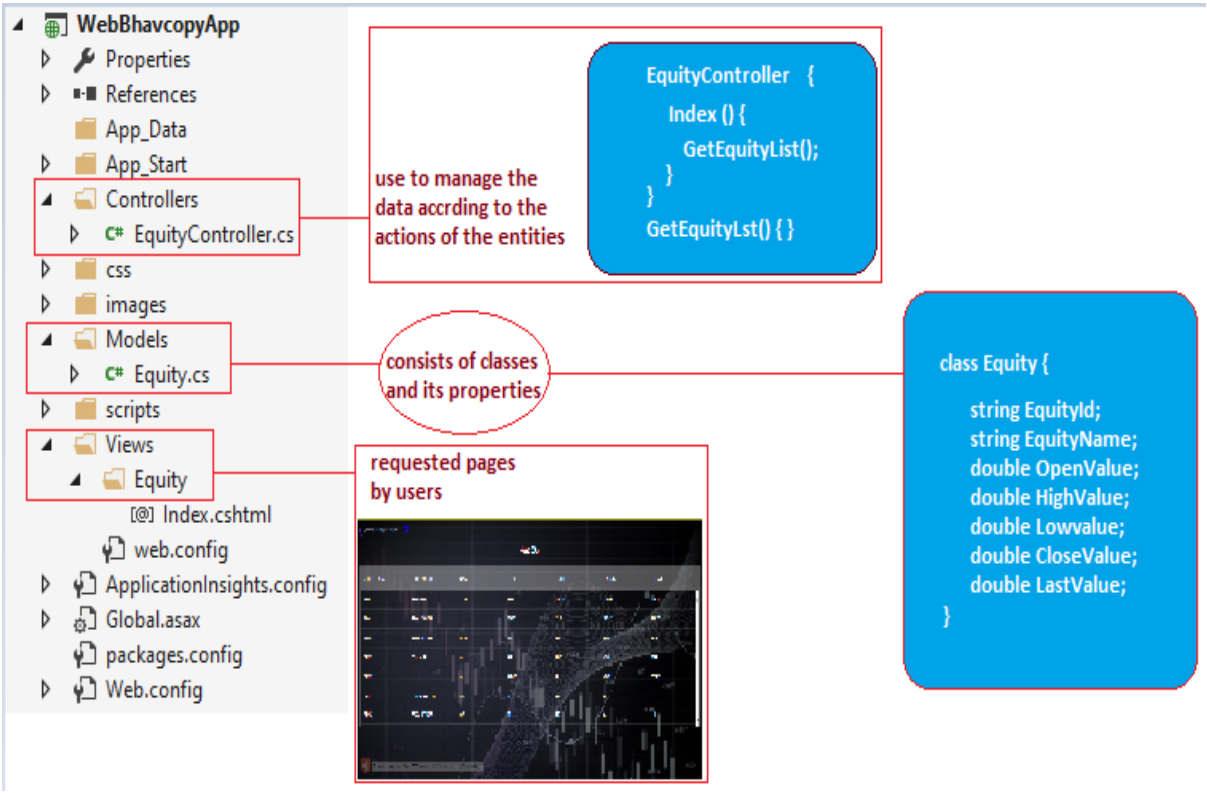

Fig 5: MyTask Application Structure

Web Search Operations

A simple web search operation is done using a *JavaScript* code embedded into the page, which is triggered during *onkeyup* operation over the search field. The *JavaScript* function get all the details of the row element of the table and match with the index value of the word typed by a user.

```
<script>
    function myFunction() {
        var input = document.getElementById("myInput");
        var filter = input.value.toUpperCase();
        var table = document.getElementById("myContent");
        var tr = table.getElementsByTagName("tr");
        for (var i = 0; i < tr.length; i++) {
            td = tr[i].getElementsByTagName("td")[1];
            if (td) {
                var txtValue = td.textContent || td.innerText;
                if (txtValue.toUpperCase().indexOf(filter) > -1) {
                    tr[i].style.display = "";
                }
                else {
                    tr[i].style.display = "none";
                }
            }
        }
    }
</script>
```

returns 1 when value matche, returns -1 when value doesn't match

true result displays the value

false result hides the display

Fig 6: JavaScript code for web search

Screenshots:

Picture 1: Simple display after successful file read.

Picture 2: Web Search Operation for *hdfc* and the contents in display

Picture 3: Message to user while file not found.

Future Work: Hosting the application