**Obsidian Developer Test**

**Introduction:**

The goal of this test is to evaluate the compatibility of the developer in terms of the technology and skill sets that are required for Obsidian’s ongoing projects.

Obsidian is looking for developers with the following skillsets:

* Strong development experience with .NET C# web projects
* Strong understanding of REST WCF services
* Experience with jQuery / JS front-end coding and data manipulation
* Experience with HTML/CSS layout handing
* Knowledge of mathematics and ability to grasp and program mathematical concepts

**What’s included?**

You will find a basic .NET web project included with all required packages for this test, as well as a data file that will be used for processing.

* ObsidianDeveloperTest -> .NET 4.5 web solution where code will be written
* Newtonsoft.dll -> JSON parsing library, already referenced in the project
* Foundation Zurb -> front-end framework used for mobile responsive design/development
* jQuery -> front-end framework
* DevExtreme Charting -> front-end graph/charting framework
* PerformanceData.csv -> data to be used for this test

**Part I – Web service configuration initiation and configuration**

* 1. Create a Ajax Enabled WCF web service under the handlers folder
  2. Inside the service, create a WebGet method called ‘getFundPerformance’ that will be used for Ajax tie in.
  3. Modify the Web.Config file to ensure that your new service can be accessed via SSL

**Part II – Back end coding & basic maths:**

* + - 1. Using C# read the text of the PerfomranceData.csv into your web service
      2. Create a class to act as a container for daily returns (e.g. PerformanceDate, MonthlyReturn, TotalReturn)
      3. Read the CSV data from the PerforamanceData.csv file
      4. Convert the PerformanceData.csv into a list of performance values and sort by date
    1. Calculate **total** fund performance for **each** data point; When given the monthly returns of an investment portfolio, you can calculate the total return for each period as follows:

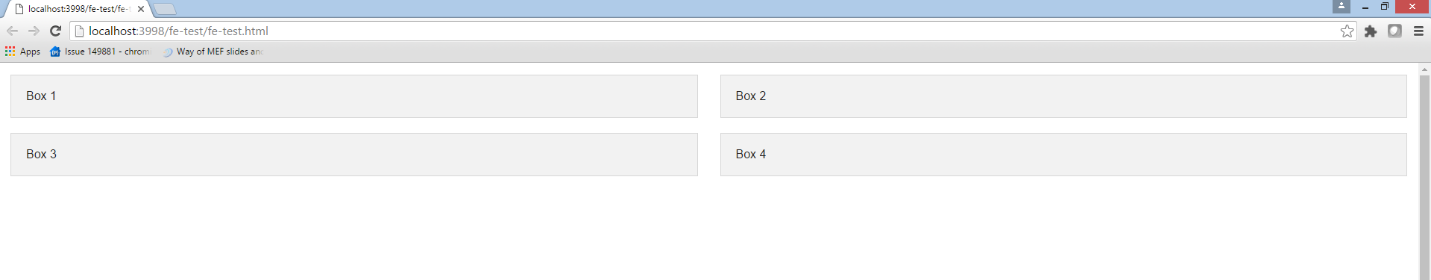
Total Return @ data point N = ((1 + R1) \* (1 + R2) \* (1 + R3) … \* (1 + Rn)) - 1

* + 1. Notice that the data in the text file isn’t sorted by Date. You will need to sort the list **prior** to running the above calculation. To sort the data you must use a **Linq** expression.

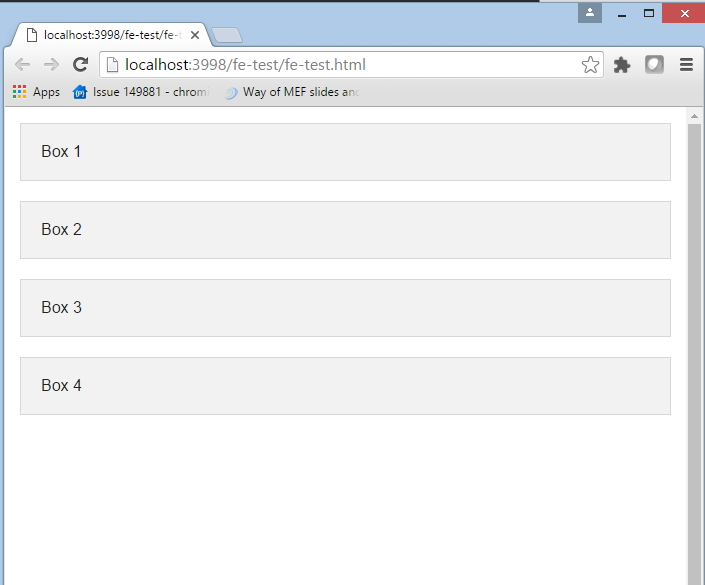
Part III – front end layout, and responsive grids.

* 1. In the fe-test.html page that already has all of the required scripts and css create a responsive grid based on Foundation Zurb. The grid should behave as follows:
* It should have 2 ROWS
* Each row should have 2 COLUMNS
* Each column should take 100% of the screen width when the view size is “small” or “medium”, and 50% of the screen width when the size is “large” or above.
* Each COLUMN should contain a div with a class “Panel” so that the result looks like the examples below

Example. large size screen:



Example small / medium size screen:



* 1. Implement a new JavaScript file that has a method that on load call your REST web service and request the fund performance for the fund.
  2. In Box 1 create a table that shows the following columns: Date, Return %, Total Return to Date, and using javascript populate the table with the performance data of the fund returns from the file delivered via your web service.
  3. In Box 2 create a DIV container with ID “graphObject”
     1. Create a javsacript method to draw a dx line chart into this div based on the performance numbers provided
     2. The chart should display the “TotalReturn” as calculated in the above section, against the PerformanceDate on the X axis. The chart should only have one series.
     3. You can get documentation on a sample DX chart here; <http://js.devexpress.com/Demos/WidgetsGallery/#demo/charts-line_and_point_series-line>
     4. Format the X axis to show date as “yyyy-M-d”
     5. Format the Y axis to show the values as %