**Introduction to KnockoutJS**

**KnockoutJS** is an open source JavaScript library built upon the MVVM pattern that makes it easier to create rich and interactive web application user interfaces with JavaScript and HTML, using knockout observables to make your UI automatically stay in sync with an underlying data model. It provides a high level way to link a view model with UI.

**Feature of Knockout:**

* Declaritive Bindings: Easily provides a way to associate DOM elements with view models using a readable and easily understandable syntax.
* Automatic UI Refresh: Easily updates the UI on the change of your view model properties.
* Dependency Tracking: Sets up dependent relationship within the properties of your view model.
* Templating: Easy to make reusable and nested HTML templates.

**Model-View-ViewModel (MVVM)**

MVVM is a design pattern for building user interfaces. It describes how you can keep a potentially sophisticated UI simple by splitting it into three parts:

* A model: your application’s stored data. This data represents objects and operations in your business domain
* A view model: a pure-code representation of the data and operations on a UI. View model is responsible for handling all the logical computations. Your view model would be an object holding a list of items, and exposing methods to bind and interact with View.
* A view: a visible, interactive UI representing the state of the view model. It displays information from the view model, sends commands to the view model (e.g., when the user clicks buttons), and updates whenever the state of the view model changes.

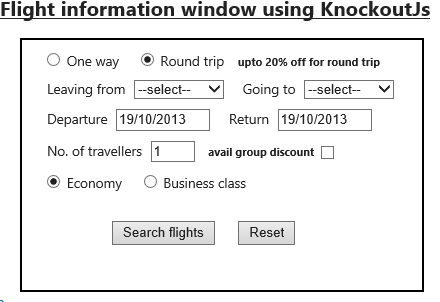
**Let’s Get Started**

In this document we will be developing two very simple and interactive Web Application using some basic concepts of KnockoutJs, jQuery, HTML and CSS.

1. **Flight Information Window:**

This is a very simple application which enables the user to search for the flights according to the criteria selected by him. It allows the user to modify his search criteria based on different conditions for each field.

Main motive behind using Knockout for developing this application is to show that how a large number of validations check and event handling can be easily achieved by two-way Knockout data-bindings without writing huge amount of code.

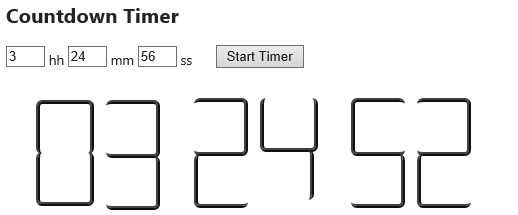


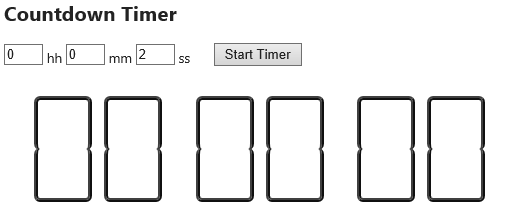
Here is the [**Link for the demo of this app**](http://jsfiddle.net/G96bp/) **.** Unfortunately the search flight functionality will not work on JsFiddle as it involve server side data handling.

**Page 7-15 covers all the details of this application.**

1. **Digital Countdown Clock Timer:**

This is a super cool countdown timer that is developed using the concepts of Knockout CSS bindings. The DOM elements are modified in such a manner that it will give the look and feel of retro style Digital clock. Based on the start time entered by the user, the countdown timer starts till it reaches 00:00:00. A number of validations are also applied on enabling/disabling Start Timer button based on the time entered by the user.





Here is the [**Link for demo of this app**](http://jsfiddle.net/ksVWz/3/)**.**

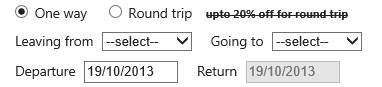
**Page 16-20 covers all the details of this application.**

**Flight Information Window**

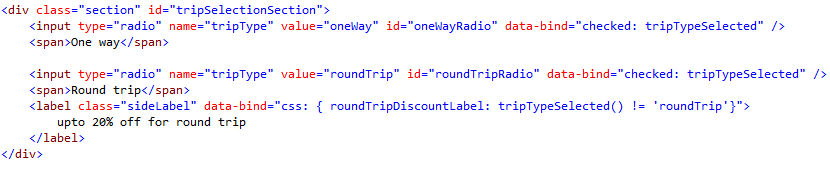
We will create four files where we will create a cshtml view for DOM elements, a view model having all the knockout observables, a controller action for returning data to view model and a CSS class for having all the styles.

Let’s divide the UI of the application into different sections for better understanding:

1. **Trip Selection Section:**



Based on the One way or Round trip radio button selection other controls like the discount label and Return date picker will be updated.



Here tripTypeSelected is ko.observable(), so based on the select/unselect of radio button tripTypeSelected will be updated with the value “oneWay” or “roundtrip” of radio button. And at the same time the discount label is bind to the value of tripTypeSelected. If the value is not roundtrip then class roundTripDiscountLabel gets applied to the label.

.roundTripDiscountLabel {

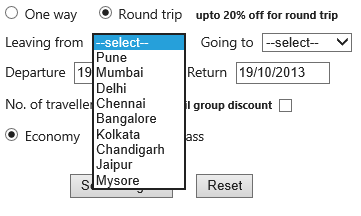
text-decoration: line-through;

}

For disabling the return date picker we will bind the control like attr: {disabled: tripTypeSelected() != 'roundTrip'}. We will look into more details in date picker section.

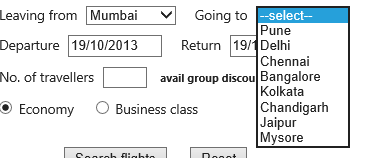
1. **City Selection Section:**

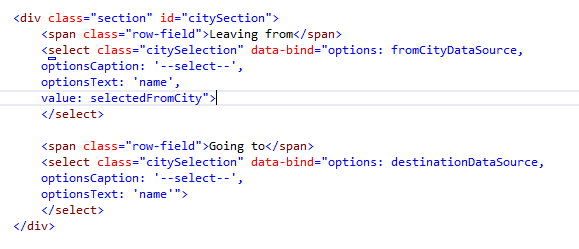
Leaving from and Going to dropdowns will be populated with the list of cities and user will choose his current and destination city.



Here the destination city drop down is dependent on the selection of Leaving from dropdown.

For Ex. If user selects Mumbai as his current city then the destination drop down gets populated by all cities except Mumbai.

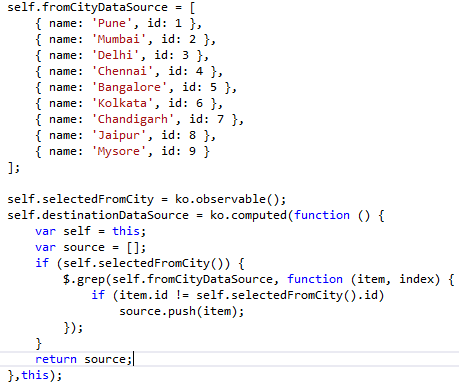




* Options: will have the data source.
* optionCaption: is the default text that will be displayed.
* optionsText: is the property of our data source that we want to get displayed. In our case the data source is in ‘Name’, ‘Id’ format and we want to display Name.
* value: Selected item will get stored in the value.

Here fromCityDataSource will be having the list of cities that will be shown on ‘Leaving from’ drop down. I have just hardcoded it for simplicity but this can be also be fetched from XML or JSON data source. City selected by the user will get stored and updated in selectedFromCity observable.

It is important here to see that we have not populated the destination drop down with some predefined list. But it will gets computed based on the change in observable selectedFromCity and fromCityDataSource.



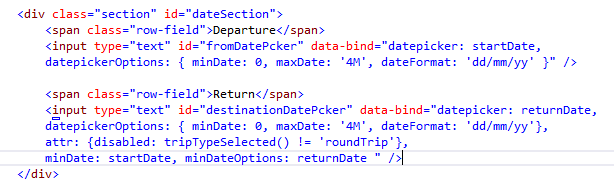
**3. Date Picker Section:**

In this section we will be covering the date picker controls that we have used for selecting start and return dates.

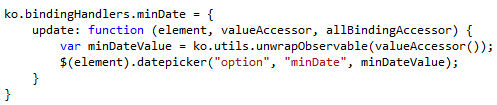
Here flight start and return dates have dependency between them, like return date will have to be greater than start date. So, we will be updating the return date on the selection of start date if it is not in accordance to the validations. To attach the date picker control and tracking the update we will use Knockout Custom Handlers.



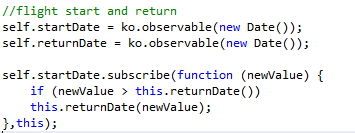
We have created a custom handler called datepicker and will bind our control with this handler. This handler will attach the jQuery datepicker with our DOM elements. We will also be passing some options to this handler to match our needs.

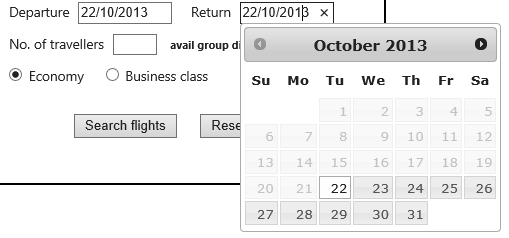


* startDate and returnDate are the observable that we have defined in our view model to track the respective date selected by the user.
* minDate and maxDate are options which will fix the range of our date picker to maximum of four months from current date
* return date picker will be disabled if the trip type is one way.
* Now in order to set the minDate of return date picker we have created another custom handler which will track the current value of startDate and initialize return date picker with startDate as its minimum date.



In the end in our view model we will be writing this piece of code to check if user has changed the start date to be greater than the current return date then it should update the returnDate.

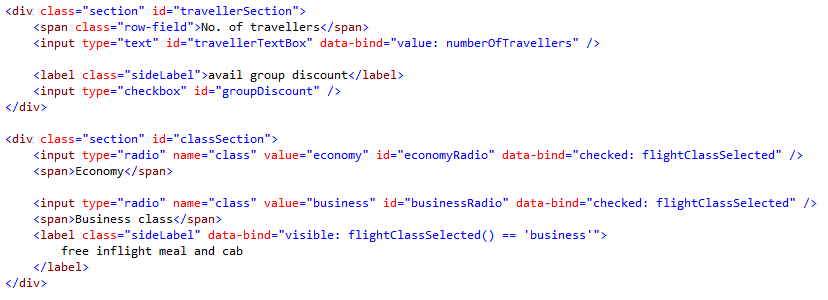


**4. Travelers and Flight Class Selection:**

In this section user will enter the number of travelers and selects if he want to travel in economy or business class.

C:\Users\Shobhit Ghai\Desktop\ksshop images\economy radio button.PNG



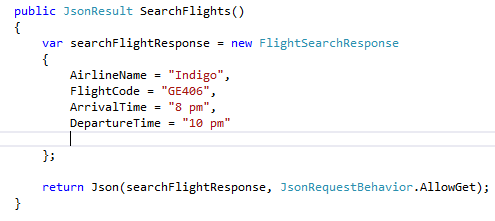
As you can see the Flight Class radio buttons are data-bind with the visible property of the side label for Business Class.

**5. Search Section:**

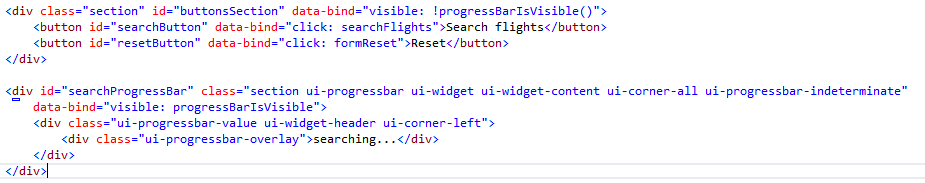
In this section user will be able to search for the flights after selecting his matching criteria and also reset the form to its default state.

For this we will write a Controller action which in our case return some dummy flight data. We will trigger an Ajax call to this controller on the click of Search button and update our view model observables which are data-bind to the DOM elements. Till the time this data will be fetched and passed to UI we will show a progress bar which signifies that the search call is still in incomplete state.

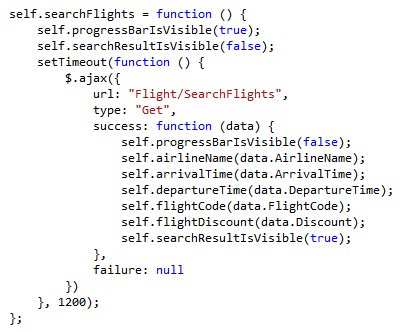
**SearchFlights Controller action**

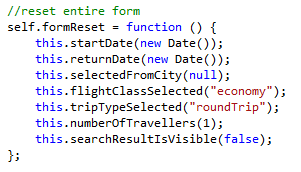


**View for search/reset section**

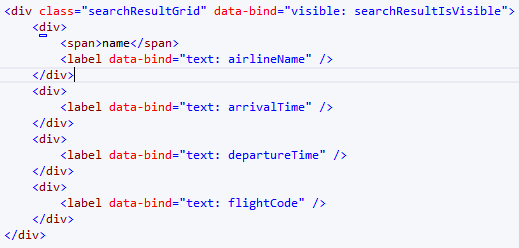


Here we have data-bind the click event of searchButton with our view model searchFlights() method and reset button with formReset(). Also the progress bar visible property is data-bind to progressBarIsVisible observable.

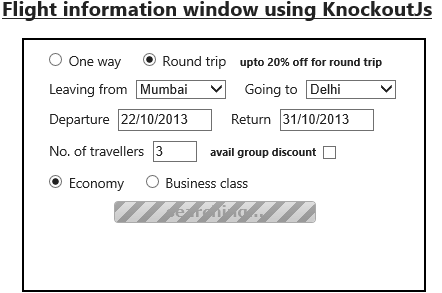




The search response is data-bind with this view and will update these elements with the values.



This is the UI when the search request is in progress.

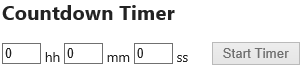
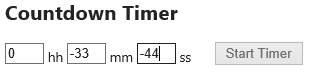


More detailed code can be viewed [here](http://jsfiddle.net/G96bp/) : http://jsfiddle.net/G96bp/

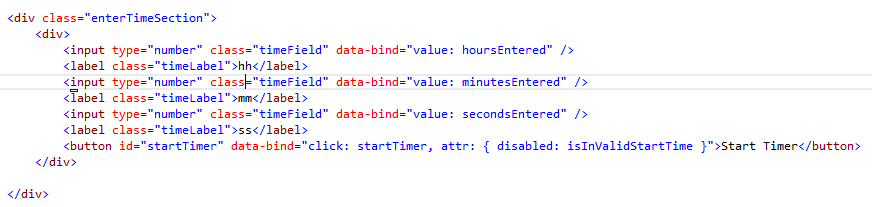
**Digital Countdown Timer**

In this application we explore Knockout CSS binding in details. The basic concept here will be like after the user enter the start time in hours, minutes or seconds the view-model will convert this is in totalSeconds. Now we will break down this total seconds in such a manner that after the interval of every one second the cshtml DOM elements will get updated with the specific CSS classes to give a look of a retro digital clock.

Also there will be validation on the StartTimer enabling/disabling on the basis of the time entered by the user. Start button will be disabled if the time entered is –ve or it is greater than 24 hours.

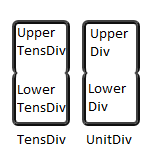
 

This will be the **view** for the controls in enter time section:



**How the clock works:**

This image is let’s say hourDiv which acts as a parent Div for the child Divs that are inside it. Digit on the right is UnitsDiv and digit on the left is TensDiv. Each of this digit Div is also sub divided into upper and lower div.



So to get it work we will update the CSS classes by Knockout CSS binding in such a way that each of the Div will change and together give a style of a Digit Clock.

The clock section is divided in three parent Div hoursDiv, minutesDiv, secondsDiv. The templates for all these Div’s are very much same except the CSS classes which will be applied to the sub sections within these three parent Divs.

Let’s take example of **Hour DIv** template**:**



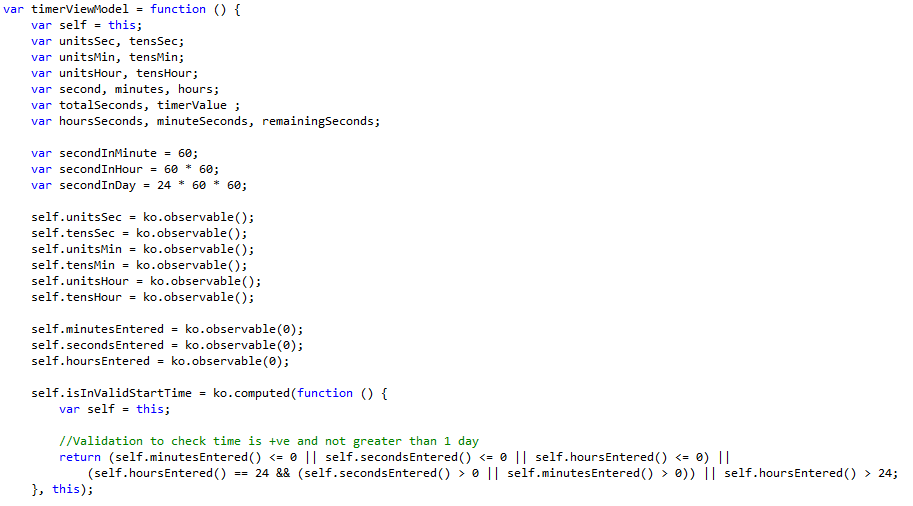
The structure of the other two parent Div for minutes and seconds will be same except they are data-bind with other observables.

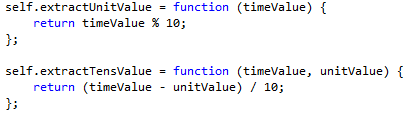
**Timer ViewModel:**

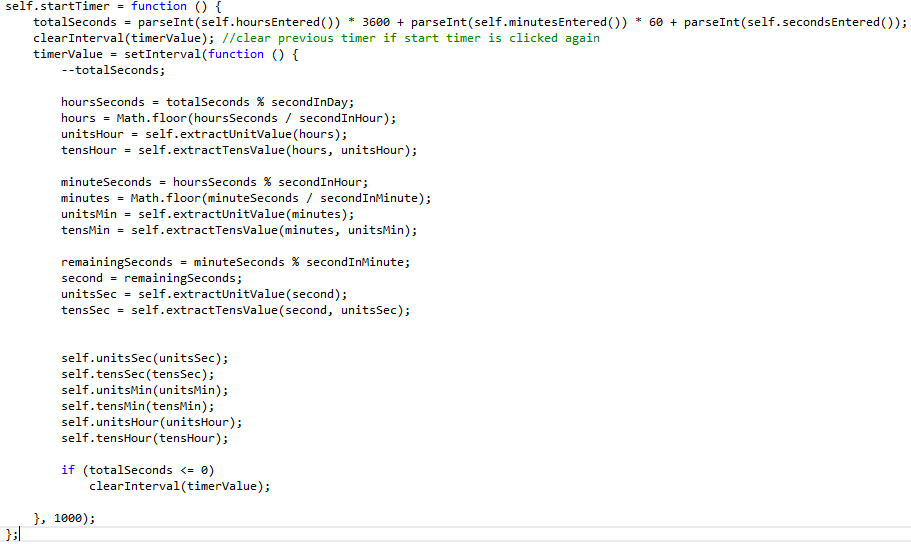
We have declared and define all the observables in our view model that we have bind with our view.

IsInvalidStartTime() is a Knockout Computed which gets called whenever any of the second, minute or hour is entered by the user.

StartTimer() will calculate all the time related fields after the interval of every one second as specified in setInterval method and will clear the interval after the time reached zero.

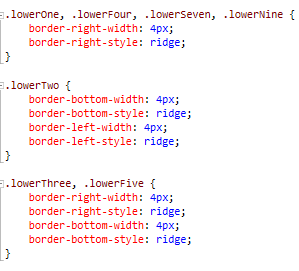
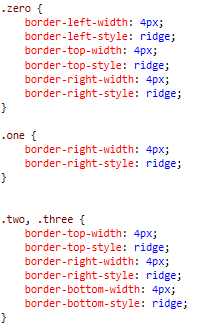






**Styling**

These are some of the CSS classes that are associated with the DOM elements. As you can see its nothing fancy, we are just playing with the border of the Div’s to get it applied and unapplied on the element.



There are lots of other classes, see the full details in the Project Code [link](http://jsfiddle.net/ksVWz/3/): http://jsfiddle.net/ksVWz/3/

**Running application:**

