Backbone.js TODO Application

RAY BUSINESS TECHNOLOGIES PVT LTD

Backbone.js

What is Backbone.js ?

Backbone.js is a Javascript Library, which allows user to build single page web applications and enable them separate his code into Models, Views and Routers.

Why Backbone.js ?

Building single-page web applications or complicated user interfaces will get extremely difficult by simply using Jquery or other tools. Because these java script libraries will not give you any structure for developing web applications. But backbone.js gives you a structure for developing single-page web applications.

Backbone's core consists of four major classes:

1 Model
2 Collection
3 View
4 Router

1. Model

In Backbone.js, a Model represents a singular entity (or) a record in a database . Models are the heart of any JavaScript application. Models hold the data and presents the data to the view for rendering data.

We can set and get data from model using modelobj.set() and modelobj.get() methods which are predefined. **E.g.**

```
Book = Backbone.Model.extend({});

    var bookObj = new Book({Name:'Learn Jquery' , Price:500 , Stock:'Available'});

    bookObj.get('Name'); // Learn Jquery

    bookObj.set({Price:400}); // Updates book price from 500 to 400

   while defining a Model we can set predefined values for the attributes like following.

   Book = Backbone.Model.extend({

           defaults:{

            Stock:'Not Available'

          }

          })
```

The following is an example for a Todo nModel created with default attribute values

----

```
    var Todo = Backbone.Model.extend({
         defaults:
          {
             title: 'New Todo',
            done: false
          },
        urlRoot: '/MyTodo/TodoService.svc/web/CallTodo'
});
```

- In the above example "Todo" is the Model name which is a Todo table in the database with columns title and done.
- 'defaults' is block of code used to set predefined values for the model object.
- 'urlRoot' is the property which holds the server side Code (Wcf Service Url (or) Php File)
- 'initialize' is a predefined function which is like a Constructor for a Class, means whenever a model object is created the code in this initialize function will execute.

2. Collection

As its name indicates, collections are ordered sets of models, where you can get and set models in the collection, listen for events when any element in the collection changes, and fetching for model's data from the server.

Collections are group of models and a collection can be defined using Backbone.Collection.extend(). Collections are the results of a query where the results consists of a number of records [models]. So model holds the data of one record like one student or one book or one Todo Item where as a collection holds the data of number of models like class students or books data or Todo List.

Ex: Example for a Simple Collection

```
    Books = Backbone.Collection.extend({
        model: Book

      });
```

→ The 'model' property tells, this collection belongs to which model.

The following code represents Todo Collection used to get the data from server.

```
          window.TodoList = Backbone.Collection.extend({

          model: Todo,

          url: "/MyTodo/TodoService.svc/web/CallTodo",

          done: function () {

            return this.filter(function (todo) {

              return todo.get('done');

            });

          },

          remaining: function () {

            return this.without.apply(this, this.done());

           }

        });
```

→ The url property contains the server side code Url for fetching data from Database. Here I have given WCF Restful service url created by me, which can perform Create, Retrieve, Update and Delete operations on the Todo table which is present in the Sql Server Database.

3. Model View

Backbone views are used to reflect what your application's data models look like. They are also used to listen to events and react accordingly. A view handles two duties fundamentally:

- Listen to events thrown by the DOM and models/collections.
- Represent the application's state and data model to the user.

Ex: Example for creating a simple View

```
        BookView = Backbone.View.extend({
            tagName:'li',
            className:'book-Item',
            render:function() {
                $(this.el).html(this.model.toJSON());
            }
        });
```

The following Code represents a single Todo item View and its Events

----

```
var TodoView = Backbone.View.extend({
 tagName: "li",
 // getting main template
 template: _.template($('#item-template').html()),
 // Events for the list view
 events: {
    "click .toggle": "toggleDone",
    "dblclick .view": "edit",
    "click a.destroy": "clear",
    "keypress .edit": "updateOnEnter",
    "blur .edit": "close"
 },
 // initialize function will be called first
```

```
initialize: function () {
    this.model.bind('change', this.render, this);
    this.model.bind('destroy', this.remove, this);
},
// to render the template
render: function () {
    this.$el.html(this.template(this.model.toJSON()));
    this.$el.toggleClass('done', this.model.get('done'));
    this.input = this.$('.edit');
    return this;
},
// Toggle the classes
toggleDone: function () {
    this.model.toggle();
},

edit: function () {
    this.$el.addClass("editing");
    this.input.focus();
},
// To close the item
close: function () {
    var value = this.input.val();
    if (!value) this.clear();
    this.model.save({
```

```
                title: value
            });
            this.$el.removeClass("editing");
        },
        // update item on Enter key pressed
        updateOnEnter: function (e) {
            if (e.keyCode == 13) this.close();
        },
        // To delete a perticular item
        clear: function () {
            this.model.destroy();
        }
    });
```

- The "el" property references the DOM object created in the browser. Every Backbone.js view has an "el" property, and if it not defined, Backbone.js will construct its own, which is an empty div element .
- To attach a listener to our view, we use the "events" attribute of Backbone.View. Remember that event listeners can only be attached to child elements of the "el" property.
- We can bind events to the model using the following code snippet.
  this.model.bind('change', this.render, this); // the render function will be called when the data in model changes

4. Collection View

For retrieving collection from server we need to create instance for the collection class and call fetch method.

Ex: *var bookList = new Books();*

bookList.fetch(); // Sends request for the server url and gets data from server

Code for fetching Todos from Server and render to the view:

```
var AppView = Backbone.View.extend({

  el: $("#todoapp"),

  // Geting the template

  statsTemplate: _.template($('#stats-template').html()),

  // Events for the application

  events: {

    "keypress #new-todo": "createOnEnter",

    "click #clear-completed": "clearCompleted",

    "click #toggle-all": "toggleAllComplete"

  },

  initialize: function () {

    this.input = this.$("#new-todo");

    this.allCheckbox = this.$("#toggle-all")[0];

    Todos.bind('add', this.addOne, this);

    Todos.bind('reset', this.addAll, this);

    Todos.bind('all',  this.render, this);

    this.footer = this.$('footer');

    this.main = $('#main');

    Todos.fetch();

  },

  render: function () {

    var done = Todos.done().length;

    var remaining = Todos.remaining().length;
    if (Todos.length) {
      this.main.show();
      this.footer.show();
      this.footer.html(this.statsTemplate({
        done: done,
        remaining: remaining
      }));
    } else {
      this.main.hide();
      this.footer.hide();
    }
    this.allCheckbox.checked = !remaining;
  },
  addOne: function (todo) {
    var view = new TodoView({
      model: todo
    });
    this.$("#todo-list").append(view.render().el);
  },
  addAll: function () {
    Todos.each(this.addOne);
  },
  createOnEnter: function (e) {
    if (e.keyCode != 13) return;
    if (!this.input.val()) return;
```

```
            Todos.create({
                title: this.input.val()
            });
            this.input.val('');
        },
        clearCompleted: function () {
            _.each(Todos.done(), function (todo) {
                todo.destroy();
            });
            return false;
        },
        toggleAllComplete: function () {
            var done = this.allCheckbox.checked;
            Todos.each(function (todo) {
                todo.save({
                    'done': done
                });
            });
        }
    });
    var App = new AppView;
});
```

----

5. Routers

Backbone routers are used for routing your applications URL's when using hash tags(#). A Backbone "router" is very useful for any application/feature that needs URL routing/history capabilities: **Ex:**

```
------------------------------------------------------

        var BookRouter = Backbone.Router.extend({

                routes: {

                    "about" : "showAbout",

                    "Books":"GetBooks"

                    "Books/:id" : "GetbookById",

                  },

                   showAbout: function(){

                       // Show About us Page

                     },

                   GetBooks : function(){

                    // get all Books from database and display to user

                     },

                   GetbookById: function(id){

               // get Book with id from Database and Show the record to user in edit mode

                  }

                  })

 ------------------------------------------------------
```

<u>What's Happening in Background</u> :

HTTP GET:

When home page was loaded, we are calling **TodoCollection.fetch()** method, which sends **GET** request to the WCF service url: "/MyTodo/TodoService.svc/web/CallTodo" , and then the service will return List of Todo items in JSON array format and we are binding these JSON data to the Todo Collection view.

Data returned by Wcf Service url :

```
[{"done":false,"id":10,"title":"Create Model"},
{"done":true,"id":11,"title":"Create Collection"},
{"done":false,"id":14,"title":"Create View"}]
```

The WCF service is a Restful service, hence the data is returned as a JSON Array. And we are binding this data to the Todo Application's View.

HTTP POST:

When the user enters a new item in the input box and after pressing enter, we are calling **modelobj.save()** method, which will **POST** the Todo item to the WCF service in JSON format, and the wcf service will insert these Todo item in database.

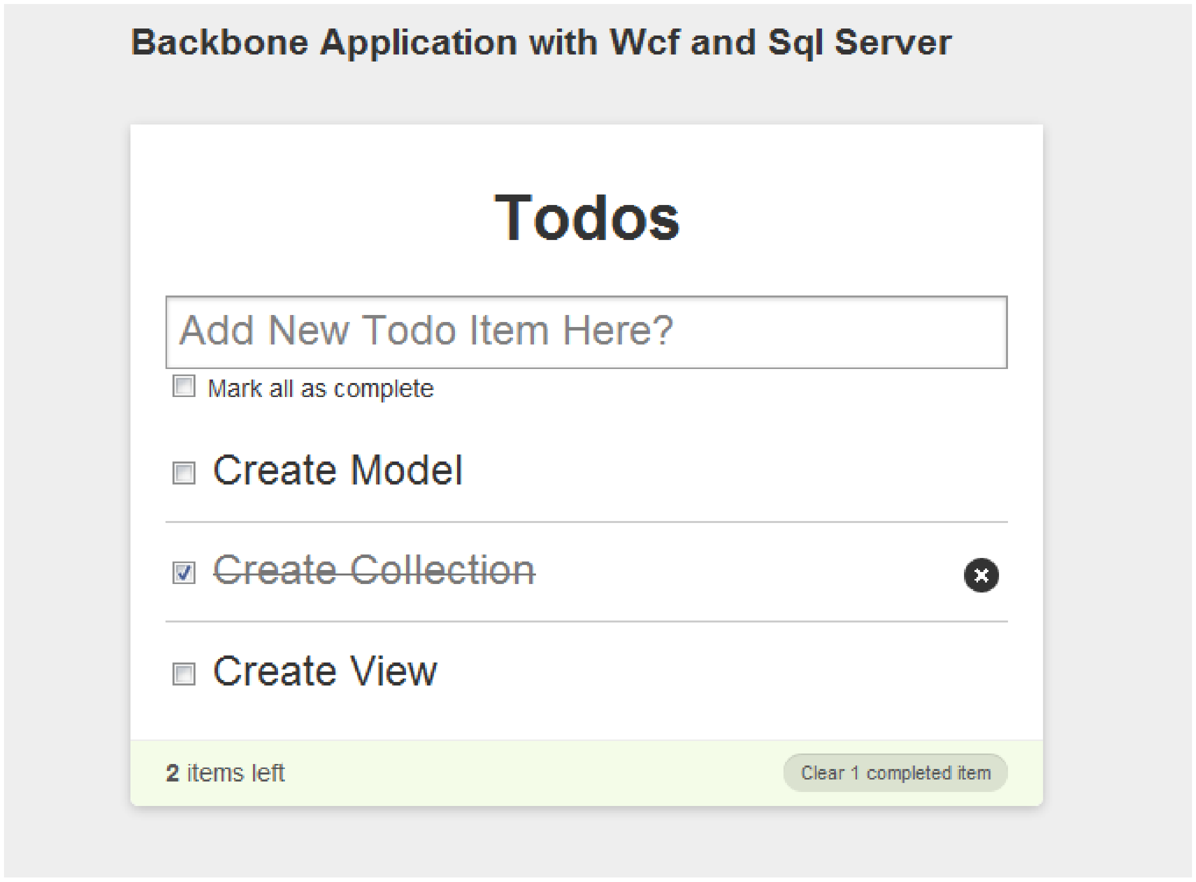
HTTP PUT:

When the user modifies a todo Item and after pressing enter ,we are calling **modelObj.save()** method, which will send the modified Todo item with its id to the server and this request will be considered as **PUT**, and the wcf service will update the data in database.

HTTP DELETE:

When the user clicks on delete image, we are calling **modelObj.destroy()** method, which makes a **DELETE** request to the wcf service and sends the Id. Based on this id we are removing the Todo Item from database.

Application working Screen-shots


Backbone Application with Wcf and Sql Server

Todos

Add New Todo Item Here?

Mark all as complete

Create Model

~~Create Collection~~

Create View

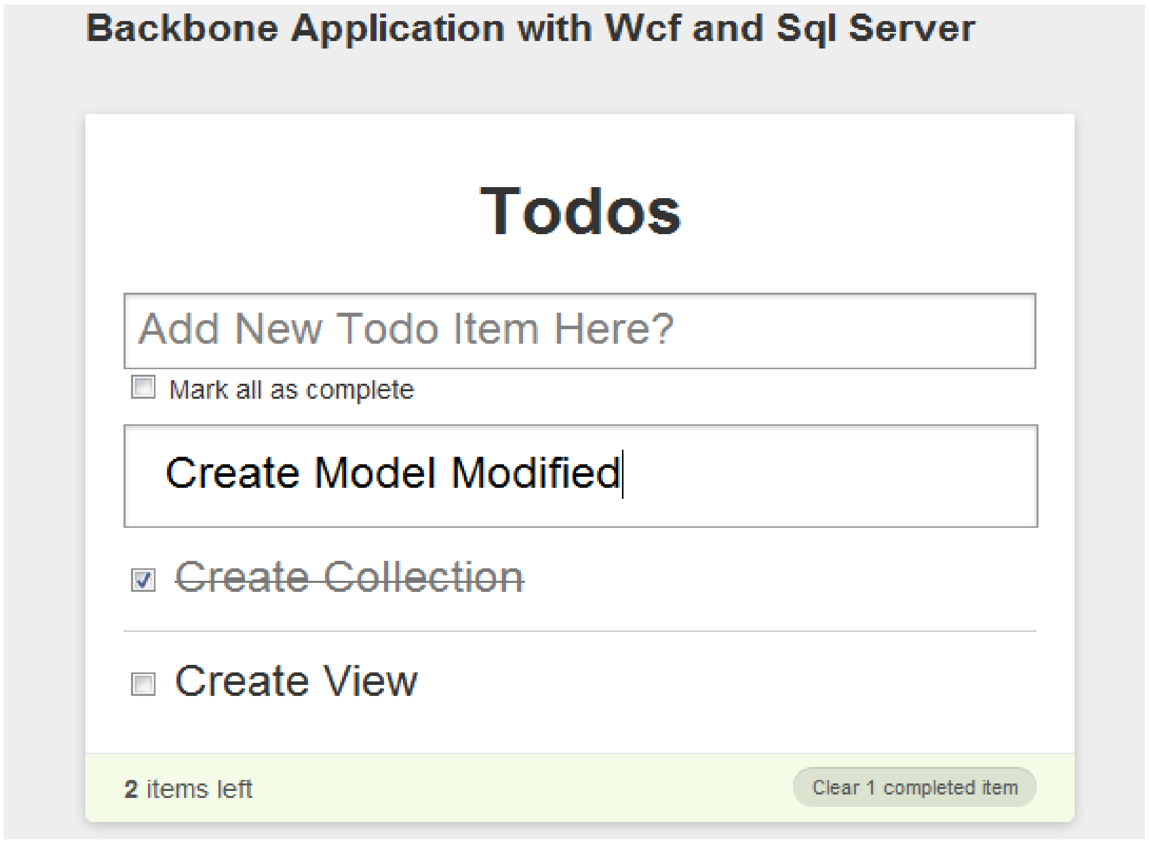
2 items left

Clear 1 completed item

The above image shows the output of the Todos application. The user can add new Todo item in the Add New Todo item input box.

When we keep our mouse pointer on any Todo ite, we can see a delete image and By clicking on that image we can remove that Todo Item.

By double clicking on any Todo item, the application will change the item into edit-mode and then user can modify the Todo item. The following screen shows editing an existing Todo item.



Backbone Application with Wcf and Sql Server

Todos

Add New Todo Item Here?

Mark all as complete

Create Model Modified

~~Create Collection~~

Create View

2 items left

Clear 1 completed item

In the above image we have changed the “Create Model” to “Create Model Modified” the the cursor is showing there, this means the Todo Item is in edit-mode.

About Ray Business Technologies Pvt. Ltd.

Ray Business Technologies, www.raybiztech.com, is a leading IT Services and Solutions Provider to Fortune 500 enterprises worldwide. Raybiztech is partnered with top technology companies and is a Member of NASSCOM. Having offices worldwide, Raybiztech has matured systems and processes, working on diversified technologies and develops turnkey solutions.

Contact us

Ray Business Technologies Pvt Ltd

India
Ray Business Technologies Pvt. Ltd.
Plot No. 204, Block B, Kavuri Hills,
Next to IGNOU, Madhapur, Hyderabad - 500033
Tel: +91 - 40 – 2311 8011 / 22
Email: info@raybiztech.com

USA
19720 Ventura Blvd., Suite 325
Woodland Hills, CA 91364
USA Direct: 786-600-1743
Email: usa@Raybiztech.com

Visit us: www.raybiztech.com