BACKBONEJS - OVERVIEW

What is Backbone.js?

Backbone.js is a light weight JavaScript library that allows to develop and structure client side applications that run in a web browser. It offers MVC framework which abstracts data into models, DOM into views and bind these two using events.

History

Backbone.js was developed by *Jeremy Ashkenas* and initially released on *October 13, 2010.*

When to use Backbone

- Consider you are creating a application using tons of lines code using JavaScript or jQuery. You add or replace DOM elements to the application make some requests or show animation in the application or add more number of lines to your code, the application may become complicated.
- If you want better design and tons of code, then make use of Backbone.js library that provides good functionality, well organized and structured manner for developing your application.
- Backbone communicates via events, so that you won't end up the application in mess. Your code will be cleaner, nicer and more maintainable.

Features

- Backbone.js allows to develop applications and front-end much easier and better using JavaScript functions.
- Backbone provides various building blocks such as models, views, events, routers and collections for assembling client side web applications.
- When model changes, it automatically updates the HTML of your application.
- Backbone.js is a simple library for separating business and user interface logic.
- It is free and open source library and contains over 100 available extensions.
- It acts like a backbone for your project and helps to organize your code.
- It manages the data model which includes the user data and display that data at the server side with the same format written at client side.
- It has soft dependency with *jQuery* and hard dependency with *Underscore.js*.
- It allows to create client side web applications or mobile applications in well structured and organized format.

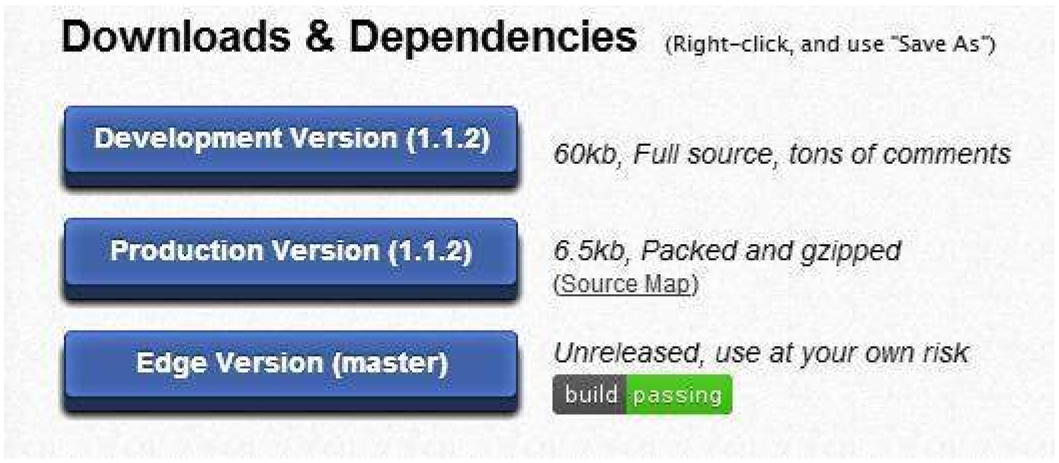
BACKBONEJS - ENVIRONMENT SETUP

Backbone.js is very easy to setup and work. This chapter will discuss about download and setup of Backbone.js library. Backbone.js can be used in two ways:

- Downloading UI library from its official website.
- Downloading UI library from CDNs

Downloading UI library from its official website

When you open the link http://backbonejs.org/, you will get to see a screen as below:


Downloads & Dependencies (Right-click, and use "Save As")
Development Version (1.1.2)
60kb, Full source, tons of comments
Production Version (1.1.2)
6.5kb, Packed and gzipped
(Source Map)
Edge Version (master)
Unreleased, use at your own risk
build passing

As you can see, there are three options for download of this library:

- **Development Version** - Right click on this button and save as and you get full source JavaScript library.
- **Production Version** - Right click on this button and save as and you get Backbone-min.js library file which is packed and gzipped.
- **Edge Version** - Right click on this button and save as and you get an unreleased version i.e development is going on, hence you need to use it at your own risk.

Dependencies

Backbonejs depends on the following javascript files:

- *Underscore.js* : This is the only hard dependency which needs to be included. You can get it from here
- *jQuery.js* : Include this file for RESTful persistence, history support via Backbone.Router and DOM manipulation with Backbone.View. You can get it from here
- *json2.js* : Include this file for older Internet Explorer support. You can get it from here

Download UI Library from CDNs

A CDN or Content Delivery Network is a network of servers designed to serve files to users. If you use a CDN link in your web page, it moves the responsibility of hosting files from your own servers to a series of external ones. This also offers an advantage that if the visitor to your webpage has already downloaded a copy of Backbone.js from the same CDN, it won't have to be re-downloaded.

As said above, Backbone.js has dependency of following javascript:

- jQuery
- Underscore

Hence CDN for all the above is as follows:

```
<script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.5.2/jquery.min.js"></script>
<script type="text/javascript"
src="http://ajax.cdnjs.com/ajax/libs/underscore.js/1.1.4/underscore-min.js"></script>
<script type="text/javascript"
src="http://ajax.cdnjs.com/ajax/libs/backbone.js/0.3.3/backbone-min.js"></script>
```

> *We are using the CDN versions of the library throughout this tutorial.*

Example

Let's create a simple example using Backbone.js.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
  <title>Hello World using Backbone.js</title>
</head>
<body>
  <!-- ========= -->
  <!-- Your HTML -->
  <!-- ========= -->

  <div >Loading...</div>

  <!-- ========= -->
  <!-- Libraries -->
  <!-- ========= -->
  <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
  <script src="http://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.3.3/underscore-
min.js" type="text/javascript"></script>
  <script src="http://cdnjs.cloudflare.com/ajax/libs/backbone.js/0.9.2/backbone-min.js"
type="text/javascript"></script>


  <!-- =============== -->
  <!-- Javascript code -->
  <!-- =============== -->
  <script type="text/javascript">
    var AppView = Backbone.View.extend({
      // el - stands for element. Every view has an element associated with HTML content,
will be rendered.
      el: '#container',
      // It's the first function called when this view is instantiated.
      initialize: function(){
        this.render();
      },
      // $el - it's a cached jQuery object (el), in which you can use jQuery functions to
push content. Like the Hello TutorialsPoint in this case.
      render: function(){
        this.$el.html("Hello TutorialsPoint!!!");
      }
    });

    var appView = new AppView();
  </script>

</body>
</html>
```

The code comments are self explanatory. Few more details as below:

- There's a html code at the start of *body* tag

  ```
  <div >Loading...</div>
  ```

  This prints *Loading...*

- Next, we have added the following CDNs

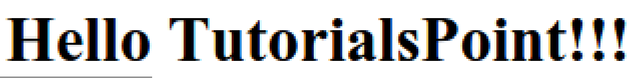
  ```
   <script src="https://code.jquery.com/jquery-2.1.3.min.js"
  type="text/javascript"></script>
    <script src="http://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.3.3/underscore-
  min.js" type="text/javascript"></script>
    <script src="http://cdnjs.cloudflare.com/ajax/libs/backbone.js/0.9.2/backbone-
  min.js" type="text/javascript"></script>
  ```

- Next we have the following script:

  ```
   var AppView = Backbone.View.extend({
        // el - stands for element. Every view has an element associated with HTML
  content, will be rendered.
        el: '#container',
        // It's the first function called when this view is instantiated.
        initialize: function(){
          this.render();
        },
        // $el - it's a cached jQuery object (el), in which you can use jQuery
  functions to push content. Like the Hello World in this case.
        render: function(){
          this.$el.html("<h1>Hello TutorialsPoint!!!</h1>");
        }
      });

      var appView = new AppView();
  ```

  The comments are self explanatory. The last line, we are initializing *new AppView*. This will print the "Hello TutorialsPoint" in the *div* with

Save this page as myFirstExample.html. Open this in your browser and a screen as below would be seen:

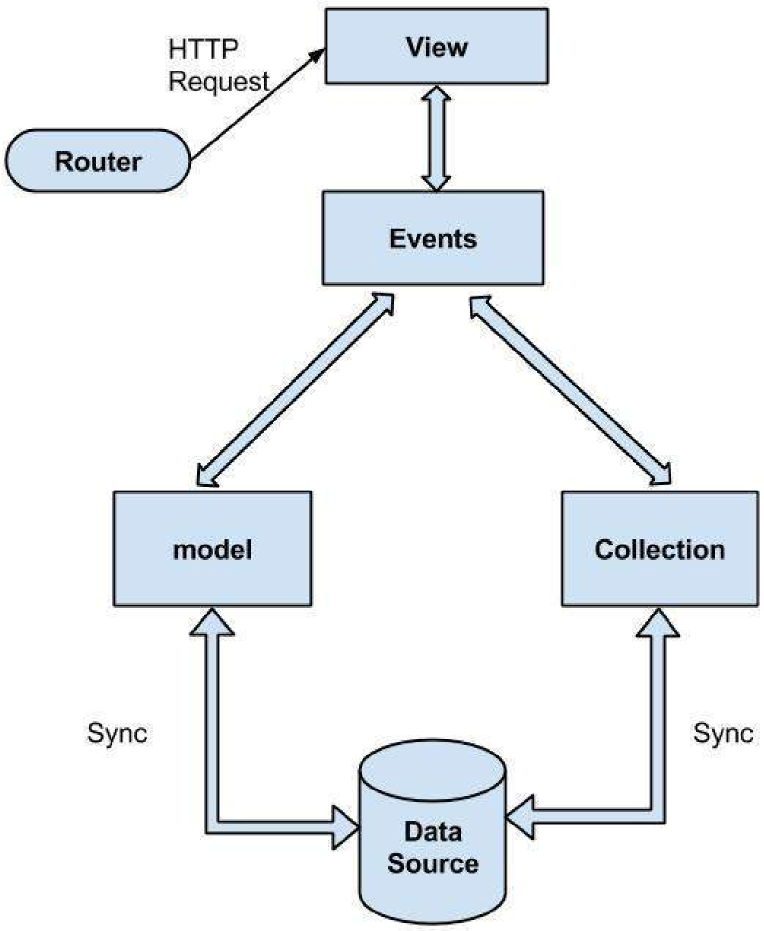


Hello TutorialsPoint!!!

Loading [MathJax]/jax/output/HTML-CSS/jax.js

BACKBONEJS - APPLICATIONS

The Backbone.js gives a structure to the web applications that allows to separate business logic and user interface logic. In this chapter, we are going to discuss the architectural style of Backbone.js application, for implementing user interfaces. The following daigram shows architecture of Backbone.js:



The architecture of Backbone.js contains following modules:

- **HTTP Request**
- **Router**
- **View**
- **Events**
- **Model**
- **Collection**
- **Data Source**

HTTP Request

The HTTP client sends a HTTP request to a server in the form of request message where web browsers, search engines etc acts like HTTP clients. User requests for a file such as documents, images etc using HTTP request protocol. In the above daigram, you could see that the HTTP client uses the *router* to send the client request.

Router

It is used for routing client side applications and connects them to actions and events using URL's. It is an URL representation of application's objects. The URL is changed manually by the user. The URL is used by the backbone so that it can understand what application state to be sent or present to the user. Router is a mechanism which can copy the URL's to reach the view. Router is required when web applications provide linkable, bookmarkable, and shareable URL's for important locations in the app.

In the above daigram, the router sending HTTP request to the view. It is an useful feature when an application needs routing capability.

View

Backbone.js views are responsible for how and what to display from our application and they don't contain HTML markup for the application. It specifies an idea behind the presentation of the model's data to the user. Views are used to reflect "how your data model looks like". The view classes do not know anything about the HTML and CSS and each view can be updated independently when the model changes without reloading the whole page. It represents the logical chunk of UI in the DOM.

As shown in the above architecture, views represents the user interface which is responsible for displaying the response for user request done by using the *router*.

Events

Events are main parts of an application. It binds user's custom events to an application. They can be mixed into any object and are capable of binding and triggerring custom events. You can bind the custom events by using desired name of your choice. Typically, events are handled synchronously with their program flow. In the above architecture, you could see when an event occurs, it represents the model's data by using the *view*.

Model

It is the heart of the JavaScript application that retrieves and populates the data. Models contain data of an application and logic of the data and represents basic data object in the framework. Models represents business entities with some business logic and business validations. It's mainly used for data storage and business logic. It can be retrieved from and saved to data storage. Model takes the HTTP request from the *events* passed by the *view* using the *router* and synchronizes the data from database and send the response back to the client.

Collection

Collection is a set of models which binds events, when the model has been modified in the collection. Collection contains list of models that can be processed in the loop and supports sorting and filtering. When creating a collection, we can define what type of model that collection is going to have along with the instance of properties. Any event triggered on a model, which will also trigger on the collection in the model.

It also takes the request from the *view*, bind *events* and synchronizes the data with requested data and send response back to the HTTP client.

Data Source

It is the connection set up to a database from a server and contains the information which is requested from the client. The flow of the Backbone.js architecture can be described as shown in the below steps:

- User requests for the data using *router*, which routes the applications to the *events* using URL's.
- The *view* represents model's data to the user.
- The *model* and *collection* retrieves and populates the data from the database by binding custom events.

BACKBONEJS - EVENTS

Events are capable of binding objects and trigger custom events i.e. you can bind the custom events by using desired name of our choice.

Following table lists down all the methods which you can use to manipulate the BackboneJS-Events:

S.N.	Methods & Description
1	**on** It binds an event to an object and executes the callback whenever an event is fired.
2	**off** It removes callback functions or all events from an object.
3	**trigger** It invokes the callback functions for the given events.
4	**once** It extends *backbone.Model* class while creating your own backbone *Model*.
5	**listenTo** It informs one object to listen an event on another object.
6	**stopListening** It can be used to stop listening to events on the other objects.
7	**listenToOnce** It causes the *listenTo* occur only once before the callback function is being removed.

Catalog of Built-in Events

BackboneJS allows use of global events wherever necessary in your application.It contains some of the built-in events with arguments as shown in the below table:

S.N.	Events & Description
1	**"add"***model, collection, options* It used when model is added to the collection.
2	**"remove"***model, collection, options* It removes the model from the collection.
3	**"reset"***collection, options* It is used to resets the collection contents.
4	**"sort"***collection, options* It is used when collection needs to resorted.
5	**"change"***model, options* It is used when changes in model's attributes.
6	**"change:[attribute]"***model, value, options* It is used when there is an update in an attribute.
7	**"destroy"***model, collection, options* It fires when model is destroyed.

8	**"request"**$model_o r_c ollection, xhr, options$ It is used model or colection starts requesting to the server.
9	**"sync"**$model_o r_c ollection, resp, options$ It is used when model or collection synced successfully with server.
10	**"error"**$model_o r_c ollection, resp, options$ It activates when there is an error in requesting to the server.
11	**"invalid"**$model, error, options$ When there is a fail in model validation, it returns invalid.
12	**"route:[name]"**$params$ When there is a specific route match, this event can be used.
13	**"route"**$route, params$ It is used when there is a match with any route.
14	**"route"**$router, route, params$ It is used by history when there is a match with any route.
15	**"all"** It fires for all triggered event by passing event name as first argument.

BACKBONEJS - MODEL

Models contain dynamic data and its logic. Logic such as conversions, validations, computed properties, and access control fall under Model. As it contains all application data, model is also called as *heart of JavaScript* application.

Following table lists down all the methods which you can use to manipulate the BackboneJS-Model:

S.N.	Methods & Description
1	**extend** It extends *backbone.Model* class while creating your own backbone *Model*.
2	**initialize** When model instance is created, the class's constructor gets called and it is invoked by defining *initialize* function when model is created.
3	**get** It gets value of an attribute on the model.
4	**set** It sets the value of an attribute in the model.
5	**escape** It is similar to *get* funtion, but returns the HTML-escaped version of a model's attribute.
6	**has** Returns true, if attribute value defined with non-null value or non-undefined value.
7	**unset** It removes an attribute from a backbone model.
8	**clear** Remove all attributes, including id attribute from a backbone model.
9	**id** It uniquely identify the model entity, that might be manually set when model is created or populated when model is saved on the server.
10	**idAttribute** Defines model's unique identfier which contains the name of the member of the class which will be use as id.
11	**cid** It is auto generated client id by Backbone which uniquely identify the model on the client.
12	**attributes** Attributes defines property of a model.
13	**changed** Changes all the attributes that have changed after setting the attributes using *set* method.
14	**defaults** Sets a default value to a model, that means if user doesn't specify any data, the model won't fall with empty property.
15	**toJSON** Returns copy of the attributes as an object for JSON stringification.

16	**sync** It is used to communicate with the server and to represents state of a model.
17	**fetch** Accept the data from the server by delegating *sync* method in the model.
18	**save** Saves the data of the model by delegating to *sync* method which reads and save the model every time when Backbone calls it.
19	**destroy** Destroys or removes the model from the server by using the *Backbone.sync* method which delegates the HTTP "delete" request.
20	**validate** If input is invalid, it returns specified error message or if input is valid, it doesn't specify anything and simply display the result.
21	**validationError** It display validation error, if validation fails or after the *invalid* event is triggered.
22	**isValid** It checks the model state by using *validate* method and also checks validations for each attributes.
23	**url** It is used for the instance of the model and returns url where model's resource is located.
24	**urlRoot** Enables the url function by using the model id to generate the URL.
25	**parse** Returns the model's data by passing the through the response object and represents the data in JSON format.
26	**clone** It is used to create deep copy of a model or to copy one model object to another object.
27	**hasChanged** Returns true, if attribute gets changed since the last *set*.
28	**isNew** Determines whether the model is a new or existing one.
29	**changedAttributes** It returns the model's attributes that have changed since the last *set* or else becomes false, if there are no attributes.
30	**previous** It determines the previous value of the changed attribute.
31	**previousAttributes** Returns state of the all attributes prior to last change event.

Underscore Methods

There are six *Underscore.js* methods which provides their functionality to be used on the *Backbone.Model*.

S.N.	Methods & Description
1	**_.keys***object* It is used to access the object's enumerable properties.

2	**_.values***object* It is used to get values of object's properties.
3	**_.pairs***object* It describes the object's properties in terms of key value pairs.
4	**_.invert***object* It returns the copy of object, in which keys have become the values and vice versa.
5	**_.pick***object, * keys* It returns the copy of object and indicates which keys to pick up.
6	**_.omit***object, * keys* It returns the copy of object and indicates which keys to omit.

Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js

BACKBONEJS - COLLECTION

Collections are ordered sets of *Models*. We just need to extend the backbone's collection class to create your own collection. Any event that is triggered on a model in a collection will also be triggered on the collection directly. This allows you to listen for changes to specific attributes in any model in a collection.

Following table lists down all the methods which you can use to manipulate the BackboneJS-Collection:

S.N.	Methods & Description
1	**extend** Extends the backbone's collection class to create an own collection.
2	**model** To specify the model class, we need to override the model property of the collection class.
3	**initialize** When model instance is created, it is invoked by defining initialize function when the collection is created.
4	**models** Array of models which are created inside of the collection.
5	**toJSON** Returns the copy of the attributes of a model using JSON format in the collection.
6	**sync** It represents the state of the model and uses Backbone.sync to display the state of the collection.
7	**add** Add a model or array of models to the collection.
8	**remove** Removes a model or array of models from the collection.
9	**reset** It resets the collection and populates with new array of models or will empty the entire collection.
10	**set** It is used to update the collection with set of items in a model. If any new model is found, the items wiil be added to that model.
11	**get** It is used to retrieve the model from a collection by using idor cid.
12	**at** Retrieve the model from a collection by using specified index.
13	**push** It is similar to add method which take array of models and push the models to the collection.
14	**pop** It is similar to remove method which take array of models and remove the models from the collection.

15	**unshift** Add specified model at the beginning of a collection.
16	**shift** It removes the first item from the collection.
17	**slice** Displays the shallow copy of the elements from the collection model.
18	**length** Counts the number of models in the collection.
19	**comparator** It is used to sort the items in the collection.
20	**sort** Sorts the items in the collection and uses *comparator* property in order to sort the items.
21	**pluck** Retrieves the attributes from the model in the collection.
22	**where** It is used to display the model by using the matched attribute in the collection.
23	**findWhere** It returns the model, that matches the specified attribute in the collection.
24	**url** It creates an instance of the collection and returns where resource is located.
25	**parse** Returns the collection's data by passing the through the response object and represents the data in JSON format.
26	**clone** It returns the shallow copy of the specified object.
27	**fetch** It extracts the data from the model in the collection using the *sync* method.
28	**create** It creates new instance of the model in the collection.

Underscore Methods

The below table list down the *Underscore.js* methods which provides their functionality to be used on the *Backbone.Collection*.

S.N.	Methods & Description
1	**_.each*list, iteratee, [context]*** Iterates each of the elements in the collection using *iteratee* function.
2	**_.map*list, iteratee, [context]*** It maps the each value and display them in a new array of values using *iteratee* function.
3	**_.reduce*list, iteratee, memo, [context]*** It reduces list of values into single value and it also known as *inject* and *foldl*.
4	**_.reduceRight*list, iteratee, memo, [context]*** It is right associative version of *reduce*.
5	**_.find*list, predicate, [context]***

	It finds each value and returns the first one which passes the predicate or test.
6	**_.filter***list, predicate,* [*context*] It filters each value and returns the array of values which passes the predicate or test.
7	**_.reject***list, predicate,* [*context*] It returns the rejected elements in the list which doesnot pass the predicted values.
8	**_.every***list, predicate,* [*context*] It returns true, if elements in the list which pass the predicted values.
9	**_.some***list, predicate,* [*context*] It returns true, if elements in the list which pass the predicted values.
10	**_.contains***list, value,* [*fromIndex*] It returns true, if value present in the list.
11	**_.invoke***list, methodName, ∗ arguments* It invokes the method name using *methodName* on each value in the list.
12	**_.max***list,* [*iteratee*], [*context*] It specifies the maximum value in the list.
13	**_.min***list,* [*iteratee*], [*context*] It specifies the minimum value in the list.
14	**_.sortBy***list,* [*iteratee*], [*context*] It returns the sorted elements in ascending order by using *iteratee* in the list.
15	**_.groupBy***list,* [*iteratee*], [*context*] It divides the collection values into sets, grouped by using *iteratee* in the list.
16	**_.shuffle***list* It returns shuffled copy of the list.
17	**_.toArray***list* It defines an array of the list.
18	**_.size***list* It defines the number of values in the list.
19	**_.first***array,* [*n*] It specifies the first element of the array in the list.
20	**_.initial***array,* [*n*] It returns everything, but specifies the last entry of the array in the list.
21	**_.last***array,* [*n*] It specifies the last element of the array in the list.
22	**_.rest***array,* [*index*] It defines rest of the elements in the array.
23	**_.without***array, ∗ values* It returns values of all instances which are removed in the list.
24	**_.indexOf***array, value,* [*isSorted*] It returns value if it found at specified index or returns -1, if it is not found.
25	**_.indexOf***array, value,* [*fromIndex*] It returns last occurrence of the value in the array or returns -1, if it is not found.
26	**_.isEmpty***object* It returns true if there are no values in the list.
27	**_.chain***obj*

	It returns a wrapped object.

Loading [MathJax]/jax/output/HTML-CSS/jax.js

BACKBONEJS - HISTORY

It keeps track of the history, matches the appropriate route, fires callbacks to handle events and enables the routing in the application.

start

This is the only method which can be used to manipulate the BackboneJS-History. It starts listening to routes and manages the history for bookmarkable URL's.

Syntax

```
Backbone.history.start(options)
```

Parameters:

- *options:* The options include the parameters such as *pushState* and *hashChange* used with history.

Example

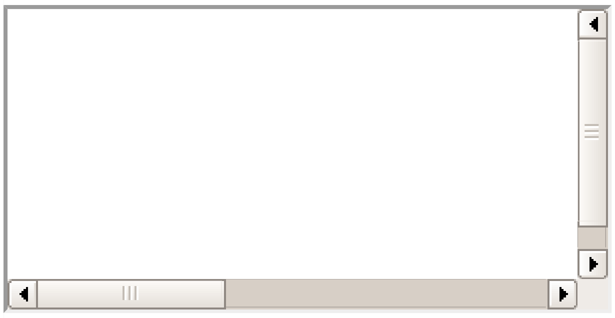
```
<!DOCTYPE html>
   <head>
      <title>History Example</title>
         <script src="https://code.jquery.com/jquery-2.1.3.min.js"
type="text/javascript"></script>
         <script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.2/underscore-min.js"
type="text/javascript"></script>
         <script src="https://cdnjs.cloudflare.com/ajax/libs/backbone.js/1.1.2/backbone-
min.js" type="text/javascript"></script>
   </head>
   <script type="text/javascript">
       //'Router' is a name of the router class
      var Router = Backbone.Router.extend({

         //The 'routes' maps URLs with parameters to functions on your router
         routes: {
            "myroute" : "myFunc"
         },

         //'The function 'myFunc' defines the links for the route on the browser
         myFunc: function (myroute) {
            document.write(myroute);
         }
      });

      //'router' is an instance of the Router
      var router = new Router();

      //Start listening to the routes and manages the history for bookmarkable URL's
      Backbone.history.start();
  </script>
  <body>

     <a href="#route1">Route1 </a>
     <a href="#route2">Route2 </a>
```

```
      <a href="#route3">Route3 </a>
   </body>
</html>
```

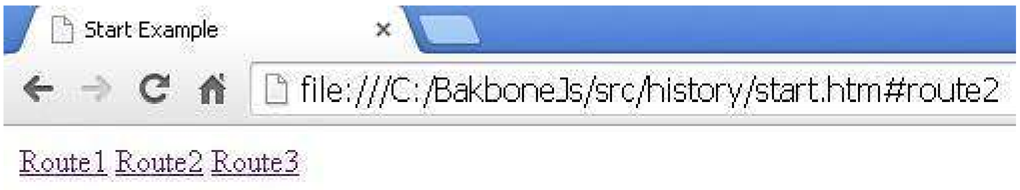
Output

Let's carry out the following steps to see how above code works:

- Save above code in **start.htm** file
- Open this HTML file in a browser.



> *NOTE: That above functionality is related to address bar so when you will open above code in actual browser then it will show result as follows*



[Click here for the demo]

BACKBONEJS - ROUTER

Router is used for routing client side applications and defines URL representation of application's object. Router is required when web applications provide linkable, bookmarkable, and shareable URL's for important locations in the app.

Following table lists down the methods which can be used to manipulate the BackboneJS - Router:

S.N.	Methods & Description
1	**extend** It extends the backbone's router class.
2	**routes** It defines URL representation of applications objects.
3	**initialize** It creates new constructor for the router instantiation.
4	**route** It creates route for the router.
5	**navigate** It is used to update the URL in the applications.
6	**execute** It is used when a route matches its corresponding callback.

BACKBONEJS - SYNC

It is used to persist the state of the model to the server.

Following table lists down the methods which can be used to manipulate the BackboneJS-Sync:

S.N.	Methods & Description
1	**Backbone.sync** It persist the state of the model to the server.
2	**Backbone.ajax** It defines the custom ajax function.
3	**Backbone.emulateHTTP** If your web server does not support, REST or HTTP approach then turn on the Backbone.emulateHTTP.
4	**Backbone.emulateJSON** It is used to handle the requests encoded with *application/json* by setting the method to *true*.

BACKBONEJS - UTILITY

The utility class defines set of methods used for implementing backbone utility.

Following table lists down the methods which you can use to manipulate the BackboneJS-Utility:

S.N.	Methods & Description
1	**Backbone.noConflict** It displays the original value of Backbone object and allows to store reference to a backbone.
2	**Backbone.$** It allows Backbone to use particular object as DOM library.

BACKBONEJS - VIEW

Views are used to reflect "how your data model looks like". They represents model's data to the user. They provide idea behind the presentation of the model's data to the user. It handles the user input events, binds events and methods, renders model or collection and interacts with user.

Following table lists down the methods which can be used to manipulate the BackboneJS-Views:

S.N.	Methods & Description
1	**extend** It extends the *Backbone.View* class to create a custom view class.
2	**initialize** It instantiate the view by using *new* keyword.
3	**el** It defines which element to be used as the view reference.
4	**$el** It represents the jQuery object for the view's element.
5	**setElement** It specifies existing dom element to a different dom element.
6	**attributes** They can be used as DOM element attributes on the view class.
7	**$(jQuery)** It is used as selector that contains $ function and runs queries within the view's element.
8	**template** While rendering the view, template creates reusable copies of markup and provides access to instance data.
9	**render** It contains the logic for rendering a template.
10	**remove** Removes a view from the DOM.
11	**delegateEvents** Binds elements to the specified DOM elements with callback methods to handle events.
12	**undelegateEvents** It removes delegate events from the view.