BackboneJS

BackboneJS 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 (Document Object Model) into views and bind these two using events. This tutorial covers most of the topics required for a basic understanding of BackboneJS and to get a feel of how it works.

BackboneJS is a **lightweight JavaScript library** that allows to develop and structure the 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 – BackboneJS was developed by Jeremy Ashkenas and was initially released on October 13th, 2010.

When to use Backbone

- Consider you are creating an application with numerous lines of code using JavaScript or jQuery. In this application, if you –
  - add or replace DOM elements to the application or
  - make some requests or
  - show animation in the application or
  - add more number of lines to your code,

  then your application might become complicated.

- If you want a better design with less code, then it is better to use the BackboneJS library that provides good functionality, is well organized and in a structured manner for developing your application.
- BackboneJS communicates via events; this ensures that you do not mess up the application. Your code will be cleaner, nicer and easy to maintain.

Features

The following are a list of features of BackboneJS –

- BackboneJS allows developing of applications and the frontend in a much easier way by using JavaScript functions.

- BackboneJS provides various building blocks such as models, views, events, routers and collections for assembling the client side web applications.
- When a model changes, it automatically updates the HTML of your application.
- BackboneJS is a simple library that helps in 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 displays that data at the server side with the same format written at the client side.
- BackboneJS has a soft dependency with **jQuery** and a hard dependency with **Underscore.js**.
- It allows to create client side web applications or mobile applications in a wellstructured and an organized format.

BackboneJS - Environment Setup

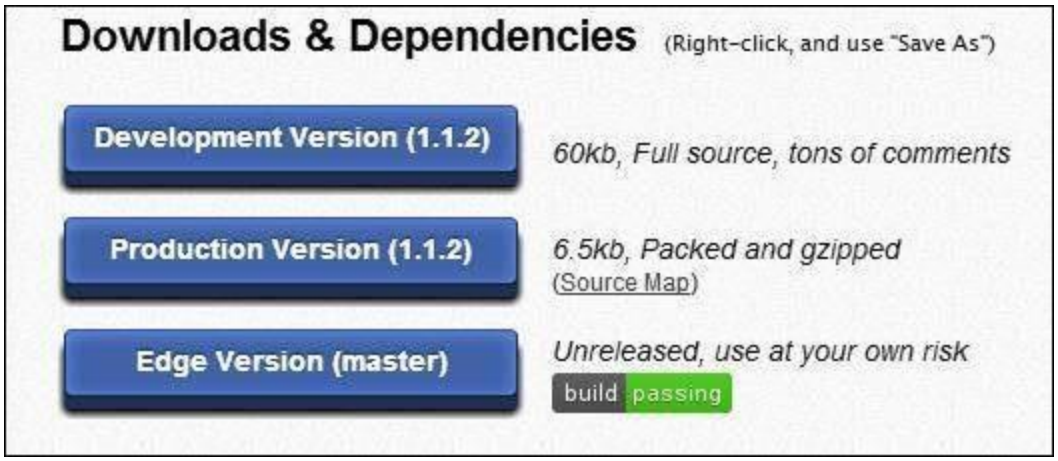
BackboneJS is very easy to setup and work. This chapter will discuss the download and setup of the **BackboneJS Library**.

BackboneJS can be used in the following two ways –

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

Downloading the UI library from its official website

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



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 the full source **JavaScript library**.
- **Production Version** – Right click on this button and save as and you get the **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 BackboneJS from the same CDN, it won't have to be re-downloaded.

As said above, BackboneJS has a dependency of the 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 = "https://ajax.cdnjs.com/ajax/libs/underscore.js/1.1.4/underscore-min.js"></script>
<script type = "text/javascript"
   src = "https://ajax.cdnjs.com/ajax/libs/backbone.js/0.3.3/backbone-min.js"></script>
```

Note – We are using the CDN versions of the library throughout this tutorial.

Example

Let's create a simple example using BackboneJS.

```
<!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 id = "container">Loading...</div>
      <!-- ========= -->
      <!-- Libraries -->
      <!-- ========= -->
      <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/1.3.3/underscoremin.js"
         type = "text/javascript"></script>
```

```
        <script src = "https://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. A few more details are given below –

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

```
<div id = "container">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 = "https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.3.3/underscore-min.js"
    type = "text/javascript"></script>
<script src = "https://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();
```

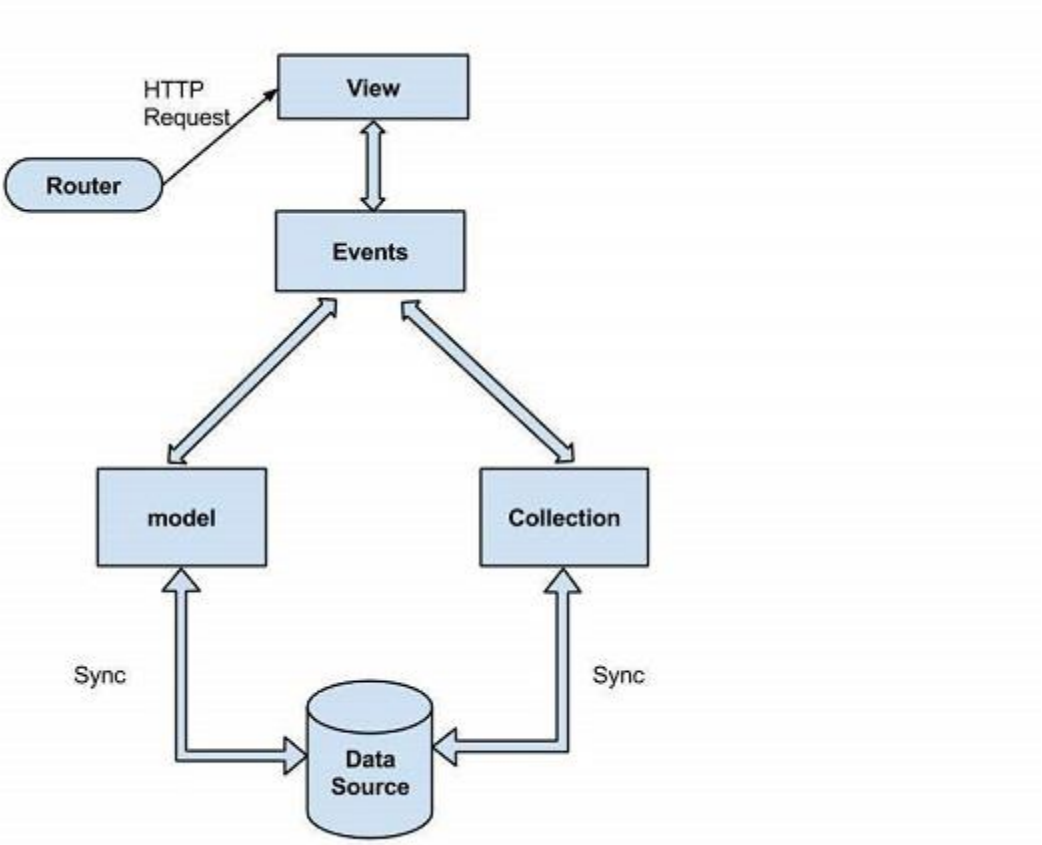
The comments are self-explanatory. In the last line, we are initializing ***new AppView()***. This will print the "Hello TutorialsPoint" in the **div with id = "container"**

Save this page as **myFirstExample.html**. Open this in your browser and the screen will show the following text.

Hello TutorialsPoint!!!

BackboneJS - Applications

The BackboneJS 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 the BackboneJS application for implementing user interfaces. The following diagram shows the architecture of BackboneJS –



The architecture of BackboneJS contains the following modules –

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

Let us now discuss all the modules in detail.

HTTP Request

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

Router

It is used for routing the client side applications and connects them to actions and events using URL's. It is a URL representation of the application's objects. This 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.

The router is a mechanism which can copy the URL's to reach the view. The Router is required when web applications provide linkable, bookmarkable, and shareable URL's for important locations in the app.

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

View

BackboneJS 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 the UI in the DOM.

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

Events

Events are the main parts of any application. It binds the user's custom events to an application. They can be mixed into any object and are capable of binding and triggering custom events. You can bind the custom events by using the 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, logic of the data and represents the basic data object in the framework.

Models represents business entities with some business logic and business validations. They are mainly used for data storage and business logic. Models can be retrieved from and saved to data storage. A Model takes the HTTP request from the Events passed by the View using the Router and synchronizes the data from the database and sends the response back to the client.

Collection

A Collection is a set of models which binds events, when the model has been modified in the collection. The collection contains a 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 will also trigger on the collection in the model.

It also takes the request from the view, bind events and synchronizes the data with the requested data and sends the 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 BackboneJS architecture can be described as shown in the following steps –

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

In the next chapter, we will understand the significance of Events in BackboneJS.

BackboneJS - Events

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

The following table lists down all the methods which you can use to manipulate the BackboneJS-Events –

S.No.	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 the backbone.Model class while creating your own backbone Model.
5	listenTo It informs one object to listen to 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 the use of global events wherever necessary in your application. It contains some of the built-in events with arguments as shown in the following table –

S.No.	Events & Description
1	**"add"(model, collection, options)** It used when a model is added to the collection.
2	**"remove"(model, collection, options)** It removes a model from the collection.
3	**"reset"(collection, options)** It is used to reset the collection content.
4	**"sort"(collection, options)** It is used when a collection needs to resorted.
5	**"change"(model, options)** It is used when changes are to be made to a 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 the model is destroyed.
8	**"request"(model_or_collection, xhr, options)** It is used when a model or a collection starts requesting to the server.
9	**"sync"(model_or_collection, resp, options)** It is used when a model or a collection is synced successfully with the server.
10	**"error"(model_or_collection, 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 the triggered events by the passing event name as the first argument.

BackboneJS - Model

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

The following table lists down all the methods which you can use to manipulate the BackboneJS-Model –

S.No.	Methods & Description
1	extend It extends the **backbone.Model** class while creating your own backbone Model.
2	initialize When a model instance is created, the class's constructor gets called and it is invoked by defining the initialize function when the model is created.
3	get It gets the value of an attribute on the model.
4	set It sets the value of an attribute in the model.
5	escape

It is like the **get** function, but returns the HTML-escaped version of a model's attribute.

has

6

Returns true, if attribute value defined with non-null value or non-undefined value.

unset

7

It removes an attribute from a backbone model.

clear

8

Removes all attributes, including id attribute from a backbone model.

id

9

It uniquely identifies the model entity, that might be manually set when a model is created or populated or when a model is saved on the server.

idAttribute

10

Defines a model's unique identifier which contains the name of the member of the class which will be used as id.

cid

11

It is an auto generated client id by Backbone which uniquely identifies the model on the client.

attributes

12

Attributes defines property of a model.

changed

13

Changes all the attributes that have changed after setting the attributes using the **set()** method.

defaults

14

Sets a default value to a model, that means if the user doesn't specify any data, the model won't fall with an empty property.

toJSON

15

Returns a copy of the attributes as an object for JSON stringification.

sync

16

It is used to communicate with the server and to represent the state of a model.

fetch

17

Accept the data from the server by delegating **sync()** method in the model.

save

18

Saves the data of the model by delegating to **sync()** method which reads and saves the model every time when a Backbone calls it.

destroy

19

Destroys or removes the model from the server by using the **Backbone.sync** method which delegates the HTTP "delete" request.

validate

20

If the input is invalid, it returns a specified error message or if the input is valid, it doesn't specify anything and simply displays the result.

validationError

21

It displays the validation error, if the validation fails or after the **invalid** event is triggered.

isValid

22

It checks the model state by using the **validate()** method and also checks validations for each attribute.

url

23

It is used for the instance of the model and returns the url to where the model's resource is located.

urlRoot

24

Enables the url function by using the model id to generate the URL.

parse

25

Returns the model's data by passing through the response object and represents the data in the JSON format.

	clone	
26		It is used to create a deep copy of a model or to copy one model object to another object.
	hasChanged	
27		Returns true, if the attribute gets changed since the last **set**.
	isNew	
28		Determines whether the model is a new or an existing one.
	changedAttributes	
29		It returns the model's attributes that have changed since the last **set** or else becomes false, if there are no attributes.
	previous	
30		It determines the previous value of the changed attribute.
	previousAttributes	
31		Returns the state of the all the attributes prior to the last change event.

Underscore Methods

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

S.No.	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.

BackboneJS - Collection

Collections are ordered sets of Models. We just need to extend the backbone's collection class to create our 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.

The following table lists down all the methods which you can use to manipulate the BackboneJS-Collection –

S.No.	Methods & Description
1	extend Extends the backbone's collection class to create a collection.
2	model To specify the model class, we need to override the model property of the collection class.
3	initialize When a model instance is created, it is invoked by defining the initialize function when the collection is created.
4	models Array of models which are created inside the collection.
5	toJSON Returns the copy of the attributes of a model using the JSON format in

the collection.

6 sync

It represents the state of the model and uses the 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 a set of items in a model. If any new model is found, the items will be added to that model.

11 get

It is used to retrieve the model from a collection by using the idor cid.

12 at

Retrieve the model from a collection by using specified index.

13 push

It is similar to the add() method which takes the array of models and pushes the models to the collection.

14 pop

It is similar to the remove() method which takes the array of models and removes the models from the collection.

15 unshift

Add a specified model at the beginning of a collection.

16 shift

It removes the first item from the collection.

slice

17

Displays the shallow copy of the elements from the collection model.

length

18

Counts the number of models in the collection.

comparator

19

It is used to sort the items in the collection.

sort

20

Sorts the items in the collection and uses comparator property in order to sort the items.

pluck

21

Retrieves the attributes from the model in the collection.

where

22

It is used to display the model by using the matched attribute in the collection.

findWhere

23

It returns the model, that matches the specified attribute in the collection.

url

24

It creates an instance of the collection and returns where resources are located.

parse

25

Returns the collection's data by passing through the response object and represents the data in JSON format.

clone

26

It returns the shallow copy of the specified object.

fetch

27

It extracts the data from the model in the collection using the sync

method.

28 create

It creates a new instance of the model in the collection.

Underscore Methods

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

S.No.	Methods & Description
1	**_.each(list, iteratee, [context])** Iterates each of the elements in the collection using the iteratee function.
2	**_.map(list, iteratee, [context])** It maps each value and displays them in a new array of values using the **iteratee** function.
3	**_.reduce(list, iteratee, memo, [context])** It reduces the list of values into a single value and it also known as **inject** and **foldl**.
4	**_.reduceRight(list, iteratee, memo, [context])** It is the 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 do not pass the predicted values.

8 **_.every(list, predicate, [context])**

It returns true, if elements in the list pass the predicted values.

9 **_.some(list, predicate, [context])**

It returns true, if elements in the list pass the predicted values.

10 **_.contains(list, value, [fromIndex])**

It returns true, if a value is 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 the ascending order by using iteratee in the list.

15 **_.groupBy(list, [iteratee], [context])**

It divides the collection values into the sets, grouped by using the **iteratee** in the list.

16 **_.shuffle(list)**

It returns the 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 the remaining elements in the array.

23 _.without(array, *values)

It returns the values of all instances which are removed in the list.

24 _.indexOf(array, value, [isSorted])

It returns the value if it is found at a specified index or returns -1, if it is not found.

25 _.indexOf(array, value, [fromIndex])

It returns the 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.

BackboneJS - Router

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

The following table lists down the methods which can be used to manipulate the **BackboneJS - Router** –

S.No.	Methods & Description
1	extend

It extends the backbone's router class.

routes

2

It defines the URL representation of applications objects.

initialize

3

It creates a new constructor for the router instantiation.

route

4

It creates a route for the router.

navigate

5

It is used to update the URL in the applications.

execute

6

It is used when a route matches its corresponding callback.

BackboneJS - History

It keeps a 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>
<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 us carry out the following steps to see how the above code works –

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

NOTE – The above functionality is related to the address bar. So, when you open the above code in the browser, it will show the result as follows.

Click here for the demo

BackboneJS - Sync

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

The following table lists down the methods which can be used to manipulate the **BackboneJS-Sync** –

S.No.	Methods & Description
1	Backbone.sync It persists 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**.