Introduction to Backbone.js

[Backbone.js setup and dependencies 2](#_Toc79098614)

[Backbone View: 3](#_Toc79098615)

[Templates in Backbone.js 4](#_Toc79098616)

[Backbone.js Events 5](#_Toc79098617)

[Models 6](#_Toc79098618)

[Pass Model Data to Views 7](#_Toc79098619)

[Pass Model data to Underscore.js templates 7](#_Toc79098620)

[Syntax for using the Model variables in Templates 10](#_Toc79098621)

[Model methods: 10](#_Toc79098622)

[Model Methods for Validations 11](#_Toc79098623)

[Model Change Events in Backbone 11](#_Toc79098624)

[More Model Events 12](#_Toc79098625)

[Model Inheritance in Backbone.js 13](#_Toc79098626)

[Collections in Backbone.js 13](#_Toc79098627)

[Collection Methods of Backbone 14](#_Toc79098628)

[Passing Collections to Underscore.js Templates 14](#_Toc79098629)

[Underscore.js Methods 15](#_Toc79098630)

[Routing in Backbone.js 16](#_Toc79098631)

[Parameterized Routing in Backbone.js 17](#_Toc79098632)

[Handlebars Templating Engine 18](#_Toc79098633)

[Resources: 19](#_Toc79098634)

## Backbone.js setup and dependencies

Backbone.js is a library available for simplifying single page/multi-page web applications. It has dependencies on

* Jquery.js
* Underscore.js

Hence, programs that intend to use backbone.js should first include these resources in the same sequence.

1. Jquery.js
2. Underscore.js
3. Backbone.js

Text

Description automatically generated

Diagram

Description automatically generated

## Backbone View:

Renders the content

Responds to DOM events (Clicks/Drag drops etc.)

More like Controllers in MVC

Have a DOM Element

View has to be extended by using Backbone.View.extend() method. This method takes input property set. Which indicates what is expected of this view.

initialize: indicates what needs to be done when this view is loaded for the first time

render: indicates what should happen when this view is rendered

this.$el by refers to the jQuery object that refers to the DOM element. Backbone creates by default a Div element unless explicitly specified as otherwise, when a view is initialized.

this.el refers to the DOM element itself.

Initialize method can return the default DOM element created by Backbone to the calling function. Backbone.js. this default element properties can be changed in the initialize function, like id, tagName, className etc.

The DOM element created by initialize can be added to the other DOM elements to modify the view behaviour.

**Example: message.html**

## Templates in Backbone.js

What are templates in Backbone.js? Reusable Markups used in Backbone.js

Why do we need templates in Backbone.js? To be able to re-use the same components multiple places

How do we use templates in Backbone.js?

Backbone.js supports multiple template engines:

* Underscore templates
* Handlebars templating engine

Text

Description automatically generated

Inline templates

Define template using the script tag with type = “text/template” or “text/html” attribute value, Id to identify the template uniquely, and use the template in the backbone views using the Id.

**Example: Backbone-template.html**

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

## Backbone.js Events

Text

Description automatically generated with low confidence

Built in Events of Backbone.js:

on, off, trigger, once, listento, stoplistening, listentoonce – Default event provided by backbone.js library

Custom events can also be fired from the views

1. Define the template to be used for the event: like a button to be clicked on the html page
2. Render the template using render method of the view
3. Define the **events** property on the view to indicate what is the event name, for which selector and what is the callback function to be invoked with the event occurs

Ex:

events:{

“click #eventBtnClick”:”onBtnClick”

}

1. Define the callback function onBtClick as a property of the backbone view

Ex:

onBtnClick:function(){

alert(“Button Clicked”);

}

## Models

**Building blocks of Backbone.js application**

Model manages the data that is to be used in the web application. Keeps data on the UI and in the Data Server in sync.

Model will also be capable of applying business logic

Graphical user interface, text, application

Description automatically generated

Model can be created by extending the Backbone.Model class

Initialize method needs to be added as a property for the model that is being created, just like initialize/render methods for Backbone Views

Graphical user interface, text

Description automatically generated

Use set(Name,value) function to set attributes on to the Model

Use get(Name) function to get the values of the attributes

toJSON() function returns JSON converted definition of the model

Graphical user interface, text, application, email

Description automatically generated

## Pass Model Data to Views

Create a model and view using Backbone.View.extend, Backbone.Model.extend methods

While instantiating the View, we can pass the model as the parameter

Example: Backbone-ModelView.html

## Pass Model data to Underscore.js templates

1. Define a Template using one of the two ways:

<script type = "text/template" id="tmpl-datatmpl">

            <P><%=Name%> authored Title: <%=Title%></P>

   </script>

template:\_.template($("#tmpl-datatmpl").html())

OR

template = \_.template("<p><%=Name%> authored Title: <%=Title%></p>");

1. Define a Model

let MyModel = Backbone.Model.extend({

                initialize:function(){

                }

            });

            let myModel = new MyModel({

                Name:"Bhaskar Kotha",

                Title:"Backbone Models and Templates"

            });

OR

let myModel = new MyModel();

myModel.set({

Name: "Bhaskar",

Title: "Model View and Templates"

});

1. Define a View

Pass the Template and Model to the view as shown below in the initialize function:

let MyView = Backbone.View.extend({

                el:"#content",

                template:\_.template($("#tmpl-datatmpl").html()),

                model:myModel,

                initialize:function(){

                    this.render();

                },

                render:function(){

                    console.log("In render function");

                    let template = \_.template("<Button>Click Me</Button><br/><p><%=Name%> authored Title: <%=Title%></p>");

                    console.log(this.model.toJSON());

                    this.$el.html(this.template(this.model.toJSON()));

                    this.$el.append(template(this.model.toJSON()));

                }

            });

In the extend function of the Backbone.View Object, observe the below two lines:

template:\_.template($("#tmpl-datatmpl").html()),

model:myModel

with this passed, the render:function() can access the values of the Model attributes using

this.model.get(“Name”), this.model.get(“Title”) functions.

OR

As shown in this example, this.model.toJSON() returns the attribute name value pairs as JSON to the template being called:

this.$el.html(this.template(this.model.toJSON()));

template is using dynamic field replacement notion <%=variableName%>, which will read the keys from the JSON key value pairs to fetch and replace the respective values to show the content on the web page.

OR

Pass the model variable name as parameter to the template function like below in the render method:

this.$el.html(this.template({

                        model:this.model.toJSON()

                    }));

Change the HTML template accordingly to read the attribute values from the passed Model data using get() method

<P><%=this.model.get('Name')%> authored Title: <%=this.model.get('Title')%></P>

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Example: Backbone-ModelTemplate.html

## Syntax for using the Model variables in Templates

<% %> notion can be used to execute logic in the template

Ex: <% var a = 10; if(a==2){…} %>

Use an ‘=’ operator to assign variables to be displayed directly on the template, like in the example:

<%= Name %>

Example: Backbone-ModelTemplate.html

## Model methods:

* Initialize – constructor for the Model
* defaults – can be used to set default values for each instantiation of the Object. Default values will be overridden when an instantiation passes the values for the default attributes
* get – gets the value of the attribute from the Model list of attributes
* set – sets the value of an attribute on the Model
* toJSON – returns the attribute list as a JSON object
* escape – converts by escaping HTML content
* has – checks if the element is present or not true/false Boolean value is returned
* unset – removes the attribute from the Model list of attributes
* clear – removes all attributes from the Model object

## Model Methods for Validations

**Graphical user interface, text, application, email

Description automatically generated**

validate() – defined in the Model definition. Attributes JSON object of the Model from which isValid() is called will be available as input argument to this function.

validationError – is the property of the Model instance, which will hold the validation error message for the Model instance. can be accessed using any model instance

isValid() – can be accessed using any model instance. Calling this method will execute the validate function defined on the Model definition, with current model instance attribute values passed as a JSON Object

Example: Backbone-ModelValidations.html

## Model Change Events in Backbone

Events that get triggered with there is a change in Model Data

Graphical user interface, text, application

Description automatically generated

bind() is used to let the Backbone framework know that we are interested in changes made to the current Model data. This function should be added in the initialize() method of the model.

When the model attributes/data is changed, it will automatically trigger the callback function associated with the bind method. Syntax for bind:

initialize:function(){

                    console.log("Model Constructor");

                    this.bind("change",function(model){

                        console.log("Model data changed");

                    });

                    this.bind("change:Company",function(model){

                        console.log("Company Name changed");

                    });

            }

First parameter indicates that we are binding the “change” event to this call back function, which will have the instance of the model where the change had happened as input parameter

Binding can also be done specific to an attribute of the model than entire Model, as shown in the example, where "change:Company" indicates that this is to identify any changes that are happening to the “Company” attribute of the Model instance.

Bind() function can also be used in View definition by using an instance of the Model object.

Instead of using this.bind(), one should use modelInstancevariable.bind(). It works same as this.bind() with a model definition

on() function on the Model instance is used to indicate actions to be performed by the **View** when the change is triggered on the Model. This function is associated with the **render** function of the View Object.

Syntax:

this.model.on(“change”,function(model){

//actions to be performed by the View when the change in Model data is identified

});

Example: Backbone-ModelChangeEvents.html

## More Model Events

clone() – creates duplicate copy of the Model instance

hasChanged() – indicates if the model attributes have been changed or not – true/false

changedAttribtues() – returns the JSON object that contains changed attributes and their values

previous(<AttributeName>) – returns the previous value of the attribute that has been changed. If there is no value change, it will still return the current value

previousAttributes() – returns all attribute value pairs as JSON object before the recent changes to the Model attributes

If an attribute is changed more than once, the previousAttributes() will return the attribute value pairs just before the most recent changes. Not the original values

listenTo() – listens to any changes on the Model attribute values and calls the specified callback function

This is a function available on View. It takes 3 parameters:

1. Model which needs to be listened to
2. What model changes should be listened to (like “change” event)
3. Callback function

Ex: Backbone-ModelChangeEvents2.html

\* this file also has backbone js code stripped out from HTML and moved to an external file.

## Model Inheritance in Backbone.js

Use extend() method to inherit the attributes/functions of an existing Method

If same function is present in both parent and child object definitions, the child method will override the parent method when the function is called using a child object instance

Use **<ParentModel>.prototype.<functionName>.apply(this)** from the child model definition to invoke the Parent Objects function.

When bothe PModel and cModel has a function playingMethod() defined

Calling cModel.playingMethod() calls the child model function definition

PModel.prototype.playingMethod.apply(this) will invoke the parent models function definition

Ex: Backbone-Inheretance.html

## Collections in Backbone.js

Collection is ordered set of Models in Backbone

Used to deal with group of related models (ex: list of Students can be represented as Collection of Students in Backbone)

It handles loading and saving of new models to the server

Provides helper functions to perform aggregation and computation against a list of models

Collections can be created by extending the Collection class in Backbone.js

Collection can be defined like any other backbone object, but needs the type of model that it is going to store:

let CollectionName = Backbone.Collection.extend({

Model:ModelName

});

Instantiate the Collection obeject like any other object:

Let collectionName = new CollectionName(modelName1, modelName2, ..modelNameN);

Ex: Backbone-Collections.html

## Collection Methods of Backbone

push – adds the element at the end of the collection, {at:i} notation can be used to insert at specified positions as second parameter of the method

unshift – adds the element at the first position of collection, {at:i} notation can be used to insert at specified positions as second parameter of the method

add – adds the element at the end of the collection, {at:i} notation can be used to insert at specified positions as second parameter of the method

pop – removes the last element of the collection

shift – removes the first element of the collection

remove – removes the specified model from the collection

## Passing Collections to Underscore.js Templates

Pass the Template, Collection and HTML element to the View and use these three to pass the information to the HTML element just like passing model data to html template and using that in the HTML element

In the view definition, use

let PView = Backbone.View.extend({

    el:"#content",//passing the HTML element to fill data

    tmpl:\_.template($("#tmpl-collection").html()),//passing the template

    coll:collPlayers,//passing the collection

    initialize:function(){

        this.render();

    },

render:function(){

        console.log("In Render Method");

        this.$el.html(this.tmpl({//set the content of the HTML element from

            collection:this.coll//template html element. Template will have to

        }));//handle the data appropriately, Loop through each model in the

    }//collection

});

Template should have been defined something like below:

<script type = "text/template" id="tmpl-collection">

            <% collection.forEach((each)=>{ %>

                Name: <%= each.toJSON().Name %> is from Company: <%= each.toJSON().Company %> and of age: <%= each.toJSON().Age%><br/>

            <% }) %>

        </script>

Example: Backbone-PassCollectionsToView.html

## Underscore.js Methods

\_.template() – for creating a new instance of template

\_.each – for looping through each modelJSON from the collectionJSON

\_.where – to fliter the Models based on the a condition, uses the collectionJSON as input,{name:value} as second parameter. Returns all collection elements satisfying the condition

\_.findwhere – returns the first model of the collection which satisfies the condition specified

\_.find – returns the first model that satisfies the condition(s) as specified in the callback function, which is passed as the second argument

\_.filter – returns the collection of models that satisfy the condition(s) as specified in the callback function which is passed as the second argument

\_.pluck – returns a list/Array of the specified attribute value from each model of the collection

let collPlayers = new Players();

collPlayers.add([p1,p2,p3,p4,p5]);

\_.each(this.coll.toJSON(),function(each){

            console.log(each.Name+", "+each.Team);

        });

        let whereColl = \_.where(this.coll.toJSON(),{Team:"India"});

        console.log(whereColl);

        let findWhereColl = \_.findWhere(this.coll.toJSON(),{Team:"India"});

        console.log(findWhereColl);

        let findResult = \_.find(this.coll.toJSON(),function(model){

            if(model.Age>7){

                return model;

            }

        });

        console.log(findResult);

        let filterResult = \_.filter(this.coll.toJSON(),function(model){

            if(model.Age>7){

                return model;

            }

        });

        console.log(filterResult);

        let filterResult1 = \_.filter(this.coll.toJSON(),(model)=>{return (model.Age>7 ? model : null); });

        console.log(filterResult1);

        let pluckResult = \_.pluck(this.coll.toJSON(),"Name");

        console.log(pluckResult);

Example: Backbone-UnderscorMethods.html

## Routing in Backbone.js

Text

Description automatically generated

Router object is used to show different Views on the webpage based on the parameter passed to the webpage URL by adding **#/<paramName>** at the end of the webpage.

Example:

[https://mysite.com/mywebpage**#/First**](https://mysite.com/mywebpage#/First)

[https://mysite.com/mywebpage**#/Second**](https://mysite.com/mywebpage#/Second)

Router class should extend the Backbone Router class, like other Backbone call extensions and it takes the **routes** parameter in the definition of the class.

This parameter takes set of parameter, function as name value pairs

let MyRouter = Backbone.Router.extend({

    routes:{

        "":"showFirstView",

        "First":"showFirstView",

        "Second":"showSecondView",

        "Third":"showThirdView"

    },

    showFirstView:function(){

        console.log("First View");

        let fView = new FirstView();

    },

    showSecondView:function(){

        console.log("Second View");

        let sView = new SecondView();

    },

    showThirdView:function(){

        console.log("Third View");

        let tView = new ThirdView();

    }

});

let myRouter = new MyRouter();

Backbone.history.start();

In this example, if the parameter **First** is passed, or no parameter is passed, it will display the HTML content from **FIrstView**

**Example:** Backbone-Routing.html

**Note**: for Routing to work, history should be enabled on the browser, by using following statement:

**Backbone.history.start();**

## Parameterized Routing in Backbone.js

Include the parameter expected in the URL as /:param in the routes parameter as shown:

routes:{

        "":"showFirstView",

        "First/:param":"showFirstView",

        "Second/:param1(/:param2"):"showSecondView",

        "Third":"showThirdView"

    }

Keeping the param in parantheis ‘(..)’ makes it optional

Later the function can take the parameters and access these parameter values from the URL

showFirstView:function(param){

        console.log("First View");

        console.log(param);

        let fView = new FirstView();

    },

    showSecondView:function(param1,param2){

        console.log("Second View");

console.log(param1+'-'+param2);

        let sView = new SecondView();

    }

Using <a href> tags in html we can show the routing options as links on the web page

        <div id="nav">

            <ul>

                <li><a href ="#/First/Bhaskar/Madhuri">First</a></li>

                <li><a href ="#/Second/Bhaskar">Second</a></li>

                <li><a href ='#/Third'>Third</a></li>

            </ul>

        </div>

Example: Backbone-ParameterizedRouting.html

## Handlebars Templating Engine

Download the handlebars.js file from the handlebarsjs.com website and include that the html file as a reference js file.

Define the template using Handlebars.compile() function like:

template:Handlebars.compile(<HTMLCONTENT TO GO HERE>);

use {{}} notation to indicate the variable in the template as shown below for inline templates

template:Handlebars.compile("{{Name}} and Team: {{Team}}")

for External templates, use the type as text/x-handlebars-template in the <script> tag

<script type="text/x-handlebars-template" id="tmpl-hbtmpl">

...

...

</script>

## Resources:

Youtube videos: [Learn backbone.js tutorial from scratch for beginners](https://www.youtube.com/watch?v=HOAU-nfy5Sc&list=PLT9miexWCpPUfPUGeQUMXQ9WS9rrbxI2D)

Backbone.js Official site: <https://backbonejs.org/>