Backbone Views

Views are the interface, in both directions, between your HTML document, and your Backbone models and collections. For most applications, views will form the majority of your Backbone.js code.

Views provide the "glue" between models and the document. They handle model events (i.e. change events), as well as DOM events. They depend on models, and models trigger events that the view can handle.

A Backbone View is:

- An object that manages the presentation of a model
- Not just the HTML that gets rendered to the DOM
- Manages events for its markup

Defining new View types: you define new View types by extending Backbone.View

```
var UserView = Backbone.View.extend({ /* properties */ });
```

All views have an associated DOM element at all times (a .el) The element is either passed to the View's constructor, or it is created by the View.

```
var UserView = Backbone.View.extend({});
var userView = new UserView({ el: 'body' });
```

Views that create new elements: the new element is defined by the id, tagName, className, and attributes:

```
var UserView = Backbone.View.extend({
    tagName: 'li',
    id: 'test',
    className: 'active foo',
    attributes: {
        'data-value': 12345
    }
});
var userView = new UserView()
$('.user-list').prepend(userView.el);
```

Views that attach to existing elements: pass an 'el' property to the View's constructor

```
var UserView = Backbone.View.extend({});
var userView = new UserView({ el: '#test' });
userView.$el.css('background-color', 'CornflowerBlue');
```

Instantiating Views - to create a new view object call its constructor function with the 'new' operator

```
// the simplest way to create a Backbone view instance
var view = new Backbone.View();
```

Usually you will want to instantiate instances of your own view type

```
var UserView = Backbone.View.extend({});
var userView = new UserView();
```

Often you will pass a model to the view constructor. The model contains the data that is used to render the view.

```
var userModel = new Backbone.Model();
var userView = new UserView({ model: userModel });
```

There are a set of properties that, if supplied to the view's constructor, will be copied to the view object - model, collection, el, id, className, tagName, attributes.

```
var myModel = new Backbone.Model();
myModel.set('content', 'this is some content');

var myView = new Backbone.View({
    model: myModel,
    className: 'model-object'
});
$('body').prepend(myView.el);
```

el: all views have an 'el' property that references the view's DOM element - every view maps to exactly one DOM element that may or may not have been added to the document

```
var view = new Backbone.View({ el: 'body' });
view.el // <body></body>
```

$el: this is a cached jQuery (or Zepto) wrapper around el - this avoids repeated use of $(this.el) inside of view methods

```
var view = new Backbone.View({ el: 'body' });
view.$el // [<body></body>]
```

this.$: the jQuery (or Zepto) function scoped to the current view. this.$(selector) is equivalent to this.$el.find(selector)

```
// selects a child element of the view's el with the class 'item'
this.$('.item')
```

render: the function that renders the view's element (.el)
Usually based on the view's model data. The default implementation is a no-op.

```
var UserView = Backbone.View.extend({
    render: function () {
        this.$el.html('some content');

        // render methods should return 'this' by convention
        return this;
    }
});
```

Combining views and models: generating markup from models and binding views to the model's change events is the core of Backbone. To set a View's model, pass the model to the View's constructor.

```
var myModel = new Backbone.Model();
var view = new View({
    model: myModel
});
```

Bind the View's render method: this used to be done calling .on on the model or collection, but instead, you should use this.listenTo

```
this.listenTo(this.model, 'change', function () {
    $('body').append(this.render().el);
});
```

remove: a shortcut method to remove the view from the DOM. It also calls stopListening() to remove the bound events the view has called listenTo on. This is equivalent to calling jQuery's remove method on the views $el.

```
// calling remove on Views no longer used is important to prevent memory leaks
var view = new Backbone.View();
view.remove();
```

events: Backbone Views use a declarative syntax to register handlers for DOM events

```
var FormView = Backbone.View.extend({
    events: {
        'click .clickable': 'handleClick'
    },
    handleClick: function () {}
});

// equivalent to
initialize: function () {
    this.$('.clickable').click(handleClick);
}
```

Backbone View guidelines:

- Views should render self-contained DOM elements
- Do not attach to existing elements other than the elements passed to the view's constructor
- Do not access DOM elements the View does not own.
- Single responsibility principle: keep your views granular and devolve rendering tasks as much as possible even within a given view's render method