moxie-js-client documentation

Release 1.0

Mobile Oxford team, IT Services, University of Oxford

December 16, 2016

CHAPTER 1

Developer

1.1 Overview

Documentation for developers on good practices when coding on an app / a view.

1.1.1 When creating a new app

Make sure that you have a default empty route pointing to a page that should be the home screen for your app.

1.1.2 When creating a new view

Each view should have a correct title describing the page. This is particularly important for views that will be put in favorites.

```
Backbone.trigger('domchange:title', "My title");
```

1.2 Core Moxie Modules

When developing with Backbone.js we identify common code and move that into the `core` module.

1.2.1 core/collections/MoxieCollection

Our base Collection providing some common methods we're using throughout our applications. To create a new collection using `MoxieCollection` simply extend it.

`MoxieCollection.`**`getAsync`**`(`*key*[, *options*, *retry*]`)`

We found that a common usecase for us was waiting for a "fetch" or somesuch async call to be made before we could confidently call `get` to get our object. Since we were doing this quite often we created `getAsync`. This allows you to wait for an event to fire on your collection before calling `get` on the key. Passing a success callback which will receive the object requested as the first argument.

Arguments

- **`key`** – The `id` of the model you want to access from the collection. See `Backbone.Model.idAttribute`.

- **`options`** (*`object`*) – The following optional arguments can be passed in the options argument
  - `success`: Called when the get has succeeded, this function will be called with the object returned from the collection as the argument.
  - `failure`: Called if the `key` cannot be found in the collection.
  - `pendingEvent`: default `reset` - The event you want getAsync to wait for before calling `success`.
- **`retry`** (*`boolean`*) – Used to prevent repeated callbacks occuring.

1.2.2 core/media

Simple module which presents an API to determine which media queries are active on the `document` at a particular time.

`media.`**`isTablet`**

Does our media query suggest the `document` is being rendered in a tablet style layout. This also applies for desktop's and any device with significant width.

`media.`**`isPhone`**

Are we being rendered on a device with a small width suggesting a phone.

1.2.3 core/views/MapView

Base view for rendering a Map with a collection of Points of Interest (POICollection) on it.

class **`MapView`**(*[options]*)

Accepts all the usual `Backbone.View` arguments. As well as:

Arguments

- **`options.fullScreen`** – Should this Map have the full-screen class to render at 100% height?
- **`options.interactiveMap`** – Should the map allow user interaction e.g. touch and drag the map about. If this is falsy or the map is rendered in phone view then click events on the map will be fired as `mapClick` on the `MapView` object.

`MapView.`**`setCollection`**(*collection*)

Update the MapView, removing any points currently rendered and place pointers for the new `collection`.

Arguments

- **`collection`** – The new `Backbone.Collection` to be rendered. Models within this collection should have `lat` and ‘‘lon‘ attribute in order for points to be placed.

1.3 Favourites

Moxie has a customisable home view which responds to users making favourites of resources in the application. For example a favourited bus stop will appear on the today view.

1.3.1 Responding to items being "favourited"

When a user clicks the favourite button and that model is successfully saved we fire an event on the `Backbone` object with the event type `favourited`. Here's an example of capturing the event and setting the `type` attribute on a favourite model:

```
Backbone.on('favourited', function(favourite) {
    favourite.set('type', 'poi:tram-station');
    favourite.save();
});
```

This event can be captured anywhere so be sure to call `Backbone.off('favourited')` once you're done (e.g. on `cleanup`).

Now your Today view "cards" can respond to favourites with your specified type.

1.4 Handling Geolocation

Any components in moxie-js-client can access the users location data through two different APIs found in the `moxie.position` module.

1.4.1 Follow/Unfollow - subscribe to updates

`UserPosition.`**`follow`**`(`*callback*`)`

Arguments

- **`callback`** (*`function`*) – a callback to be called each time the user position updates.

Start listening to user position updates.

`UserPosition.`**`unfollow`**`(`*callback*`)`

Arguments

- **`callback`** (*`function`*) – the callback already registered you want to remove from the listener

Stop listening to user position updates.

1.4.2 getLocation - one shot accurate position

`UserPosition.`**`getLocation`**`(`*cb*[, *options*]`)`

Arguments

- **`callback`** (*`function`*) – a callback to be called once a good enough position has returned from the navigator APIs or the timeout has fired.
- **`options`** (*`object`*) – Optional paramters passed in this object include, `errorMargin` specify in meters how accurate a response you want returned by getLocation. Also a `timeout` in ms how long should getLocation wait before returning the most recent result which didn't meet the `errorMargin` criteria.

Uses the phones most accurate capabilities to get a good position result within the specified paramaters.

1.5 Infinite Scrolling

To add infinite scrolling to your views simply extend `core/views/InfiniteScrollView`, a simple example of this can be found in `core/views/specs/infinite`.

1.5.1 InfiniteScrollView

`InfiniteScrollView.`**`initScroll`**`(`[*options*]`)`

Arguments

- **`options`** (*`object`*) – The following optional arguments can be passed in the options argument
  - `windowScroll`: default `false` - should be a boolean saying if we want to listen to window.scroll events
  - `scrollElement`: default `undefined` - DOM element we want to listen to scroll events for
  - `intervalPeriod`: default 250ms - time in ms which we should check if the user has scrolled
  - `scrollThreshold`: default `undefined` a floating point integer between 0 and 1 - The ratio representating how far down a page scroll should the `scrollCallbacks` be called. If left undefined `scrollCallbacks` are called whenever the scroll event fires.

`InfiniteScrollView.`**`scrollCallbacks`**

Array like object of functions to be called when the user scrolls down the page.

1.6 Real-time Information

Moxie presents RTI (real-time information) for different resources, currently these are all points of interest however in the future that could change.

The JS client has a standard way of handling RTI for a POI which is inline with how Backbone.js is structured. Each different RTI `type` has its own `Backbone.Model` and `Backbone.View`.

1.6.1 Adding a new RTI type

The modules we're interested in for adding a new RTI type are `places/models/RTIModels.js` and `places/views/RTIViews.js`. These modules *export* objects which take the following format (RTIViews.js):

```
{
    'rti-type': Backbone.View,
    'another-rti-type': Backbone.View
}
```

Adding support for a new RTI type should be a matter of extending these objects with your View and Model respectively.

To get this rendering on a particular POI (assuming the API is serving the RTI correctly, see Moxie RTI docs for that) the RTI type needs to be added to `DEFAULT_RTI_TYPES` in `places/models/POIModel.js`.

1.7 Testing

Moxie JS client is tested using Jasmine a behaviour-driven development framework. The specs (BDD lingo for Test-Suite) for the common Moxie JavaScript can be found in `app/tests/specs/*`, specs for each applicaiton should be located alongside the application in a folder called *specs*. This is done with a view to possibly breaking applications into separate repositories if ever possible.

1.7.1 Running the tests

The easiest way to run the tests is to open your browser to the `SpecRunner.html` file. However it's also possible to run the tests with phantomjs:

```
$ phantomjs run-jasmine.js SpecRunner.html
6 specs, 0 failures in 0.018s
```

The test runner, `run-jasmine.js` will handle setting the correct exit code and output the details of any failing specs.

1.7.2 Adding specs

When adding new spec files to be run as part of the test suite it's important to add the path of the spec file in `app/tests/main.js`.

1.8 Today view "cards"

Our Today view is composed of many individual "cards", for example a card for displaying the image from a webcam in Oxford. Another showing the current term date, weather etc.

This allows users to explore different applications within Moxie from the today screen. Users are also able to hide/show certain cards from the Today view by selecting them in the Today settings view.

1.8.1 Creating a card

The best way to see how to implement a card is to look at an example:

```
// From today/models/RiverStatus.js
define(['backbone', 'underscore', 'app/moxie.conf', 'app/today/views/RiversCard'], function(Backbone,
    var RiverStatus = Backbone.Model.extend({
        url: conf.urlFor('rivers'),
        View: RiversCard
    });
    return RiverStatus;
});

// From today/views/RiversCard.js
define(['app/today/views/CardView', 'hbs!app/today/templates/rivers'], function(CardView, riversTempl
    var RiversCard = CardView.extend({
        weight: 70,
        manage: true,
        id: 'rivers_status',
        attributes: {'class': 'today'},
        serialize: function() {
            return this.model.toJSON();
        },
        template: riversTemplate
    });
    return RiversCard;
});
```

The `RiverStatus` model provides an attribute `View` which points to our `RiversCard` view. When the `RiversCard` is rendered it is placed into the Today view depending on the value of the `weight` attribute. Views with a higher weight appear at the top of the page and those with low weights appear at the bottom. This functionality is provided through the `CardView` which each card should extend.

Note: Further details on how the weighting effects the page render can be found in `app/today/views/CardView.js`.

Models which represent cards and are **enabled** in the `TodaySettings` collection are added to the `TodayItems` collection, which calls `fetch()` on each model. Once that Model fetches its default value the View is rendered and the card is **inserted** into the page.

1.8.2 Settings

Cards on the Today view can be enabled and disabled by each user. These user settings are stored in `localStorage` currently as the `TodaySettings` collection.

Note: Default configuration is currently stored in `app/today/collections/TodaySettings.js` and defaults to having all cards enabled. This must be updated for all future cards.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search