Introduction to Backbone.js collections

In most cases we would like to work with an array of items, not with a single item. This is where Backbone.js **collections** come into play. Think of them as of ordered sets of models, an array of models.

Collections typically work with one type of models and provide method to load an array of models from the server at once, and also present some methods like where and create that mimic querying functionality. Collections in Backbone.js are really useful as you can also bind various events to them to watch for model changes and we will use collections extensively.

Create a collection

Create a new *events.js* file inside the *collection* directory and include it into the project:

```
<script src="js/collections/events.js"></script>
```

Define a new collection with a basic extending mechanism:

```
Organizer.Events = Backbone.Collection.extend({

});
```

Generally you'd want to specify with which model this collection will work

```
Organizer.Events = Backbone.Collection.extend({
  model: Organizer.Event
});
```

We also have to specify where to get an array of events from. If we had a back-end server, we would provide a `url` property with a relative path like this:

```
url: '/events'
```

This way Backbone knows where to fetch our events by sending GET request there. However, we have no back-end server and therefore will use local storage instead.

```
localStorage: new Backbone.LocalStorage('events')
```

`events` is the name of our local storage and now our collection knows where to get events.

Open up console and reload the page. We can instantiate a collection like this:

```
var events = new Organizer.Events();
```

However if I use

```
events.toJSON();
```

I'll get an empty array even though my local storage does have some data in it. To actually load data from the server (or from local storage in our case), you have to use `fetch` method:

```
events.fetch();
```

You can use `models` method that also allows raw access to your models:

```
events.models
```

`toJSON` returns only models' properties whereas `models` return models with all the corresponding methods and attributes.

If you wish to remove all models from a collection, use `reset`:

```
events.reset();
```

By passing an array to reset, you can replace the current set of models with a new one:

```
events.reset([ {title: 'new'} ]);
```

To add one model to a collection, use `add` method:

```
events.add( {title: 'added'} );
```

Instead of `add` you may use `push` that has the same functionality.

If you wish to find a specific model in the collection, use `get` method. It accepts either model's id or cid if no id is assigned yet.

```
var event = events.get('c201');
```

You can use this object to remove a specific model from the collection by using `remove`:

```
events.remove(event);
```

If for some reason you can't use `get` method, use `at` to fetch model at a specific position in the collection.

```
events.at(0);
```

Now add some more models to the collection:

```
events.add( {title: 'special', body: 'test1'} );
events.add( {title: 'special', body: 'test2'} );
```

Now suppose we want to find all models with a title "special". That is really easy to do:

```
events.where( {title: 'special'} );
```

If you wish to fetch only the first model matching the condition, use `findWhere`:

```
events.findWhere( {title: 'special'} );
```