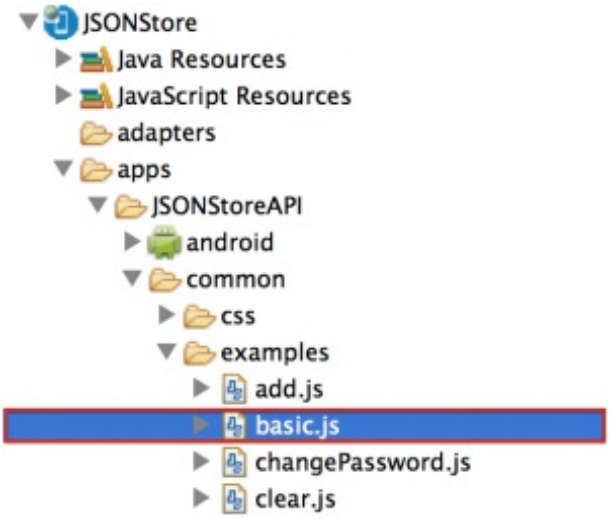
JSONStore - JavaScript API

fork and edit tutorial (https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/foundation/6.3/working-offline/jsonstore/jsonstore-javascript-api.html) | report issue (https://github.ibm.com/MFPSamples/DevCenter/issues/new)



Follow along with the code sample

1. Download the compressed file with the code sample that is associated with this tutorial.
2. Open the `basic.js` "JSONStore/apps/JSONStoreAPI/common/ examples/basic.js" file. The sample image provides context.
3. Run the application on one of the environments listed (for example: Android, iPhone). The "Setting up your environment" and "Hello World" Getting Started Tutorials cover this procedure in detail.

> **Note:** The code sample uses a JavaScript™ Unit Test framework that is called QUnit (qunitjs.com). Explaining how it works is beyond the scope of this tutorial.

Code Sample Walkthough

```
    WL.JSONStore.destroy()
    .then(function () {
        var collections = {
                people : {
                    searchFields: {name: 'string', age: 'integer'}
                }
        };
        return WL.JSONStore.init(collections);
    })
    .then(function () {
        var data = [{name: 'carlos', age: 20},
                {name: 'mike', age: 30}];
        return WL.JSONStore.get('people').add(data);
    })
    .then(function () {
        return WL.JSONStore.get('people').findAll();
    })
    .then(function (res) {
        deepEqual(res, [{_id: 1, json: {name: 'carlos', age: 20}},
                    {_id: 2, json: {name: 'mike', age: 30}}], 'check find all result
');
        start();
    })
    .fail(function (err) {
        ok(false, 'Got failure: ' + err.toString());
        start();
    });
```

The `destroy` API removes all JSONStore content from the application. It is used here to start with no data. Doing it this way ensures that the output is predictable in the code sample.

To persist data, you must first define at least one collection. These collections are entities that hold data. Shown here is the definition of a collection that is called `people`.

Search fields are fields that are indexed inside a collection. You can use those fields when you search for data that is inside a collection. You can see here the definition of two search fields:

- `name` (string)
- `age` (integer)

The data types, such as `string`, `integer`, `number`, `Boolean`, are used to better store input data.

The `init` API is used to open one or more collections. If the collection was never opened before, a file is created on the file system to persist data that is inside the collection. Before the operation finishes, an accessor to that file is created.

The accessor allows the caller to call collection-level APIs such as `add` and `findAll`, which are shown later in this code sample walkthrough.

The data that is stored inside the `people` collection is defined here. Notice that the data is a hardcoded array of two JSON objects with key value pairs for `name` and `age`. This data can be acquired from multiple sources (for example: Network Request, File I/O, User Input).

The accessor that is created by the completion of the `init` API is accessed via the `get` API. That accessor provides access to store data inside the people collection. The input data must be in JSON format.

You can find documents inside a JSONStore collection in different ways. For example: `find`, `findById`. The easiest way, and the way that is shown here, is by using the `findAll` API. This method returns all the data that is stored inside a collection.

Data that is stored inside a collection is called a document. Documents have `_id` and `json` key value pairs. The `_id` pair is an internal identifier that is added automatically when data is added. The `json` pair contains all the data that was added.

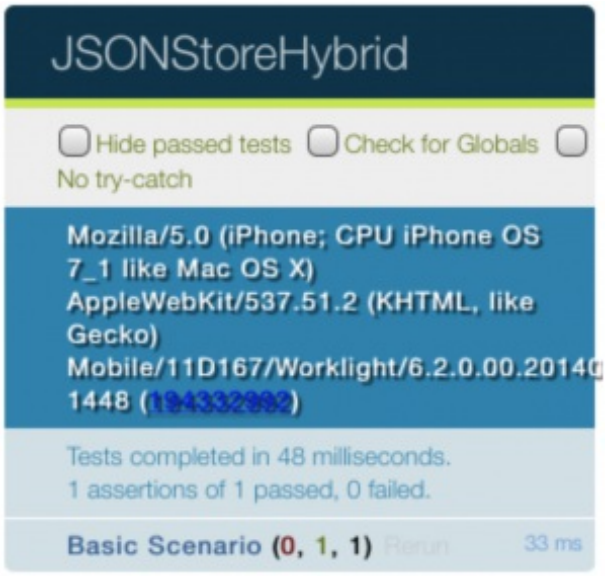
If one of the API calls fails (for example: `init`, `add`, `findAll`) the function inside `.fail` is called with an error object. Make sure that the error object contains enough information for the user to understand what went wrong. For example, it can contain:

- `src` - source of the error, for example: `find`
- `msg` - message that is related to the error, for example: `BAD_PARAMETER_EXPECTED_STRING`

Sample application

Click to download (http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v630/JSONStoreAPIBasicsProject.zip) Studio project.

Expected Output



When the application is executed in one of the supported environments, the output looks similar to the sample image. The green line indicates that everything is working as expected.

For more information

For more information about JSONStore, see the product user documentation.