Getting Started with PouchDB - Part 1

As more and more users interact with web applications on their mobile devices, it is becoming increasingly important for us to allow them to work offline. There are many cases where users need to work offline, such as on an airplane, in a remote location where there is no cellular access, or perhaps on board a large ship where Wi-Fi is not available. If you can store data local to your web application, the user can continue to work even without a connection.

You may be tempted to use local storage, but this approach is very unstructured, can be slow to access, and does not allow the storage of a lot of data. Another route is to use WebSQL, which is based on SQLLite. However, this specification has been deprecated and will eventually be phased out of browsers. Another route is IndexedDB, a very popular JavaScript database designed for local storage in a web application. However, the API for IndexedDB can vary slightly from browser to browser, and is callback-based, instead of using the more modern promise-based approach.

Enter PouchDB, which provides a thin wrapper on top of IndexedDB, makes all calls consistent between browsers, and is promise-based. This first part of a series of blog posts shows teaches you the basics of working with PouchDB.

What is PouchDB

PouchDB is an open-source JavaScript NoSQL database designed to run offline within a browser. There is also a PouchDB server version that can be used when online. These two databases synchronize from one to another using a simple API call. You may also use CouchDB on the server to synchronize your data.

A NoSQL database is storage where there is no fixed table structure as in a relational database. There are a few different methods NoSQL databases store data: column, document, Graph, and key-value pair. Of these, the most common are column and document. PouchDB supports document-oriented where data in the model is stored as a series of JSON objects with a key value assigned to each document.

Each document in PouchDB must contain a property called *_id*. The value in the _id field must be unique per database. You may use any string value you want for the *_id* field. In this article, I am going to use a value that is very simple. However, for a

real-world application you might consider using a GUID as this will ensure the values are completely unique if you synchronize with a server-side database.

You may insert new documents into your database by using the post() or the put() method, and pass in a JSON object. Once inserted, you can retrieve the document very quickly by searching for the value in the *_id* field, or you can perform other searches by using views and creating indexes.

To modify or delete an existing document, you must first locate the specific document, and load the full document into memory. Make changes to any field you want using JavaScript, then save the entire document back into the database. A field named *_rev* is created/updated with a new, unique value to help keep track of which is the most up-to-date version of the document.

PouchDB Basics

To illustrate the basics of working with PouchDB, create a HTML project and include the Bootstrap CSS framework (www.getbootstrap.com), PouchDB (www.pouchdb.com) and jQuery (www.jquery.com). Add an HTML page and add the two statements. The first statement is used to display messages, and the other displays the JSON returned from our PouchDB operations. You can add some HTML buttons to call the various functions illustrated in this post.

```
<div id="messageArea" class="alert alert-danger hidden">
  <span id="message"></span>
</div>

<div id="jsonArea" class="alert alert-info hidden">
  <textArea id="json" cols="100" rows="30"></textArea>
</div>
```

Common JavaScript

Create a JavaScript file named **PouchDBSamples-common.js** and add a closure with the following code. The code in this closure is used in each of our samples to display messages, display JSON, and to clear the messages and JSON display areas.

```
let pouchDBSamplesCommon = (function () {
  //************************************
  //* Private Functions
  //************************************
  function displayMessage(msg) {
    $("#messageArea").removeClass("hidden");
    $("#message").text(msg);
    console.log(msg);
  }

  function displayJSON(data) {
    $("#jsonArea").removeClass("hidden");
    $("#json").text(JSON.stringify(data, undefined, 2));
  }

  function hideMessageAreas() {
    $("#messageArea").addClass("hidden");
    $("#message").text("");
    $("#jsonArea").addClass("hidden");
    $("#json").text("");
  }

  //************************************
  //* Public Functions
  //************************************
  return {
    displayMessage: displayMessage,
    displayJSON: displayJSON,
    hideMessageAreas: hideMessageAreas
  }
})();
```

Most of the methods in this closure are self-explanatory, but the displayJSON() method might need a little more explanation. I'm sure you are familiar with JSON.stringify(), but I added two additional parameters to it. The second parameter is not used, so just pass an undefined. If you pass a number as the third parameter, it includes the specified number of white space in the stringified JSON object. This gives us nicely formatted JSON string for display on our web page.

Open/Create a Database

On the HTML page you created, add links to PouchDB, jQuery and the JavaScript file you just created.

```
<script src="../Scripts/pouchdb-6.4.3.min.js"></script>
<script src="../Scripts/jquery-3.3.1.min.js"></script>
<script src="../Scripts/pouchDBSamplesCommon.js"></script>
```

Add a <script></script> tag just below the script tags you just created. Within these script tags create a variable named db. Create a function named openCreateDatabase() and write code within this function to create a new instance

of a PouchDB database called 'handyman'. Respond to the 'created' event by using the on() method on the PouchDB class. If the PouchDB database is opened, or created successfully, the database name is passed to this event. Use the closure to display a message that the database is now ready to be used.

```
<script>
  let db = null;

  function openCreateDatabase() {
    db = new PouchDB('handyman');
    PouchDB.on('created', function (dbName) {
      pouchDBSamplesCommon.displayMessage("Database: '" + dbName +
"' opened successfully.");
    });
  }
</script>
```

Add an HTML button on the page to call the openCreateDatabase() function.

```
<button class="btn btn-primary" onclick="openCreateDatabase();">
  Open/Create Database
</button>
```

Add a Document

Now that the database is open, call the put() method on the database object to inject a new JSON object into the database. Always use the put() method instead of the post() method, as put() either adds or inserts a document and sets a revision id (*_rev* field). The *_rev* field is needed when synchronizing client-side data to the server.

```
function addTechnician() {
  db.put({
    _id: 'psheriff',
    firstName: 'Pal',
    lastName: 'Sheriss',
    docType: 'technician'
  }).then(function (response) {
    pouchDBSamplesCommon.displayJSON(response);
    pouchDBSamplesCommon.displayMessage("Technician added.");
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });
}
```

Get a Document

Once a document is in the database, you may extract it by passing the *_id* value to the get() method. This method returns the complete document found or throws an error if the document is not found.

```
function getTechnician() {
  db.get("psheriff")
    .then(function (response) {
      pouchDBSamplesCommon.displayJSON(response);
    }).catch(function (err) {
      pouchDBSamplesCommon.displayMessage(err);
    });
}
```

Update a Document

In order to update a document, you need to retrieve the entire document first using the get() method. Once the document is retrieved, set any of the properties you want to modify, the invoke the put() method, passing in the changed document. Be sure to return the output from the put() method so this new promise can be caught and you check for the success or failure of the update operation.

```
function updateTechnician() {
  db.get("psheriff")
    .then(function (doc) {
      // Change the document
      doc.firstName = "Paul";
      doc.lastName = "Sheriff";
      // Update the document
      return db.put(doc);
    }).then(function (response) {
      pouchDBSamplesCommon.displayJSON(response);
      pouchDBSamplesCommon.displayMessage("Technician updated.");
    }).catch(function (err) {
      pouchDBSamplesCommon.displayMessage(err);
    });
}
```

Delete a Document

You need to retrieve a document prior to deleting it. Pass the *_id* value you wish to locate to delete. In the then() function, call the remove() method on the PouchDB object. Pass the complete document to the remove() method. Alternatively, you may pass the values for the *_id* and *_rev* fields to the remove() method. Return the output from the remove() method, which is a promise. This way you can check the success or failure of the remove operation.

```
function deleteTechnician() {
  // Get technician
  db.get("psheriff")
    .then(function (doc) {
      // Delete the technician
      return db.remove(doc);
    }).then(function (response) {
      pouchDBSamplesCommon.displayJSON(response);
      pouchDBSamplesCommon.displayMessage("Technician deleted.");
    }).catch(function (err) {
      pouchDBSamplesCommon.displayMessage(err);
    });
}
```

Compact Database

As you modify documents, a new revision of the document is stored each time. This means that over time, you have a lot of old versions of the same document. Not all of these are needed, especially after you have synchronized your data to the server. Call the compact() method to remove the old data periodically to keep your database size reasonable.

```
function compactDB() {
  if (db) {
    db.compact().then(function (response) {
      pouchDBSamplesCommon.displayJSON(response);
      pouchDBSamplesCommon.displayMessage("Database compacted");
    }).catch(function (err) {
      pouchDBSamplesCommon.displayMessage(err);
    });
  }
  else {
    pouchDBSamplesCommon.displayMessage("Please open the database
first.");
  }
}
```

Delete a Database

If you are finished using a database, you can remove it completely using the destroy() method. This option removes all data, views, and other meta-data associated with this database. There is no way to recover from this operation, so be careful!

```
function destroyDatabase() {
  if (db) {
    db.destroy().then(function (response) {
      pouchDBSamplesCommon.displayJSON(response);
      pouchDBSamplesCommon.displayMessage("Database deleted.");
    }).catch(function (err) {
      pouchDBSamplesCommon.displayMessage(err);
    });
  }
  else {
    pouchDBSamplesCommon.displayMessage("Please open the database
first.");
  }
}
```

Summary

In this blog post you were introduced to PouchDB. You learned to create, update, delete and read documents from within a database. You also saw how to compact and delete a database. In the next several installments of this blog post series, you are going to learn to bulk insert and read documents, filter and count documents, use Mango queries, use map and reduce functions.

Sample Code

You can download the complete sample code at my website. http://www.pdsa.com/downloads. Choose "PDSA/Fairway Blog", then "Getting Started with PouchDB - Part 1" from the drop-down.