Working With IndexedDB - Part 2

by Raymond Camden 21 Oct 2013 Difficulty: Intermediate Length: Quick Languages: English

Web Development IndexedDB JavaScript

Welcome to the second part of my IndexedDB article. Istrongly recommend reading the first article in this series, as I'll be assuming you are familiar with all the concepts covered so far. In this article, we're going to wrap up the CRUD aspects we didn't finish before (specifically updating and deleting content), and then demonstrate a real world application that we will use to demonstrate other concepts in the final article.

Updating Records

Let's start off by discussing how to update a record with IndexedDB. If you remember, adding data was pretty simple:

```
//Define a person
var person = {
    name:name,
    email:email,
    created:new Date()
}

//Perform the add
var request = store.add(person);
```

Updating a record is just as simple. Assuming that you have defined a property called `id` as your key for your object store, you can simply use the `put` method instead of `add`.

```
var person = {
    name:name,
    email:email,
    created:new Date(),
    id:someId
}

//Perform the update
var request = store.put(person);
```

Like the `add` method, you can assign methods to handle the asynchronous results of the operation.

Deleting Records

Deleting records is done via the delete method. (Big surprise there.) You simply pass in the unique identifer of the record you want to remove. Here is a simple example:

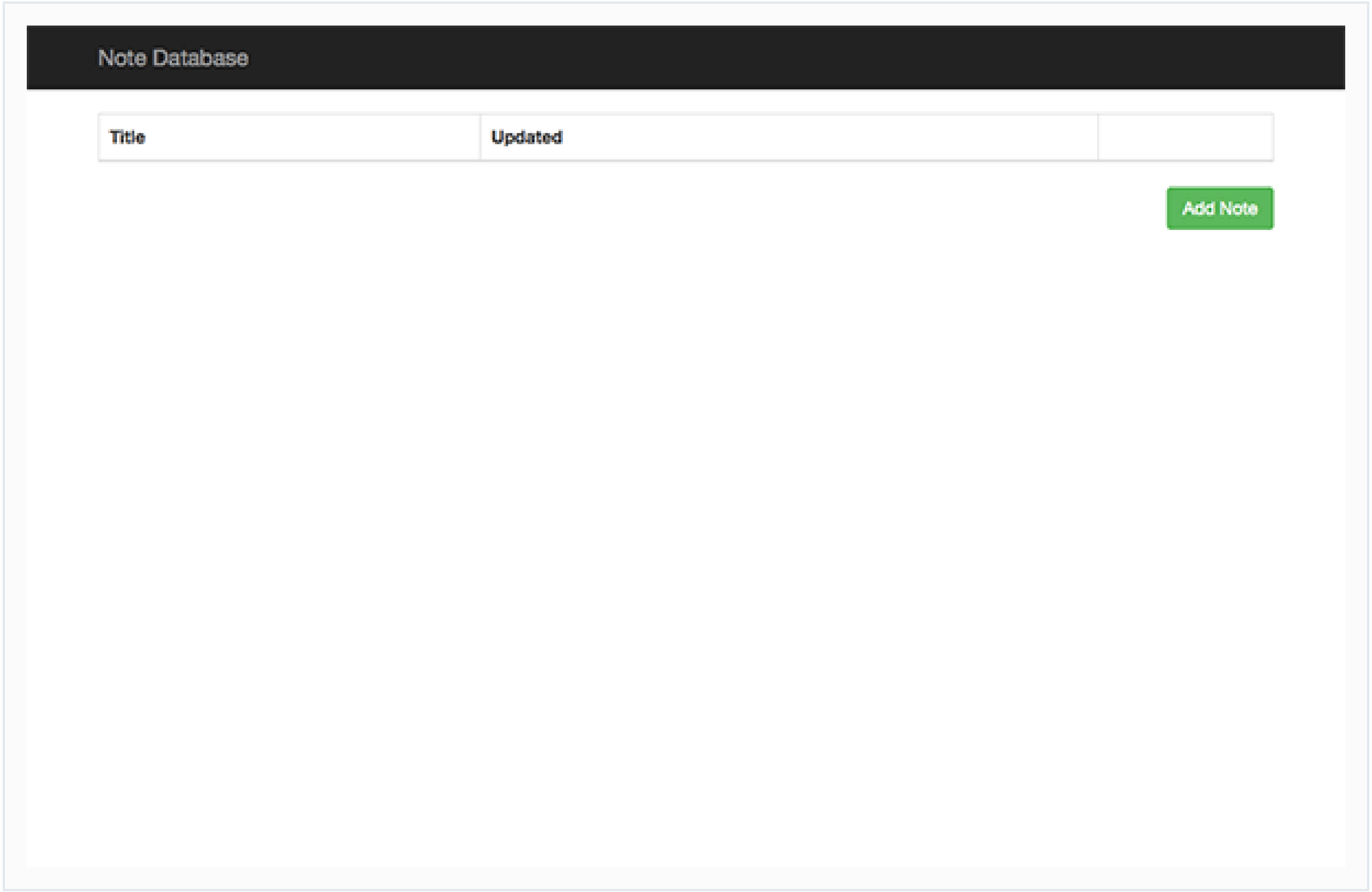
```
var t = db.transaction(["people"], "readwrite");
var request = t.objectStore("people").delete(thisId);
```

And like every other aspect of IndexedDB, you can add your handles for the asynchronous results.

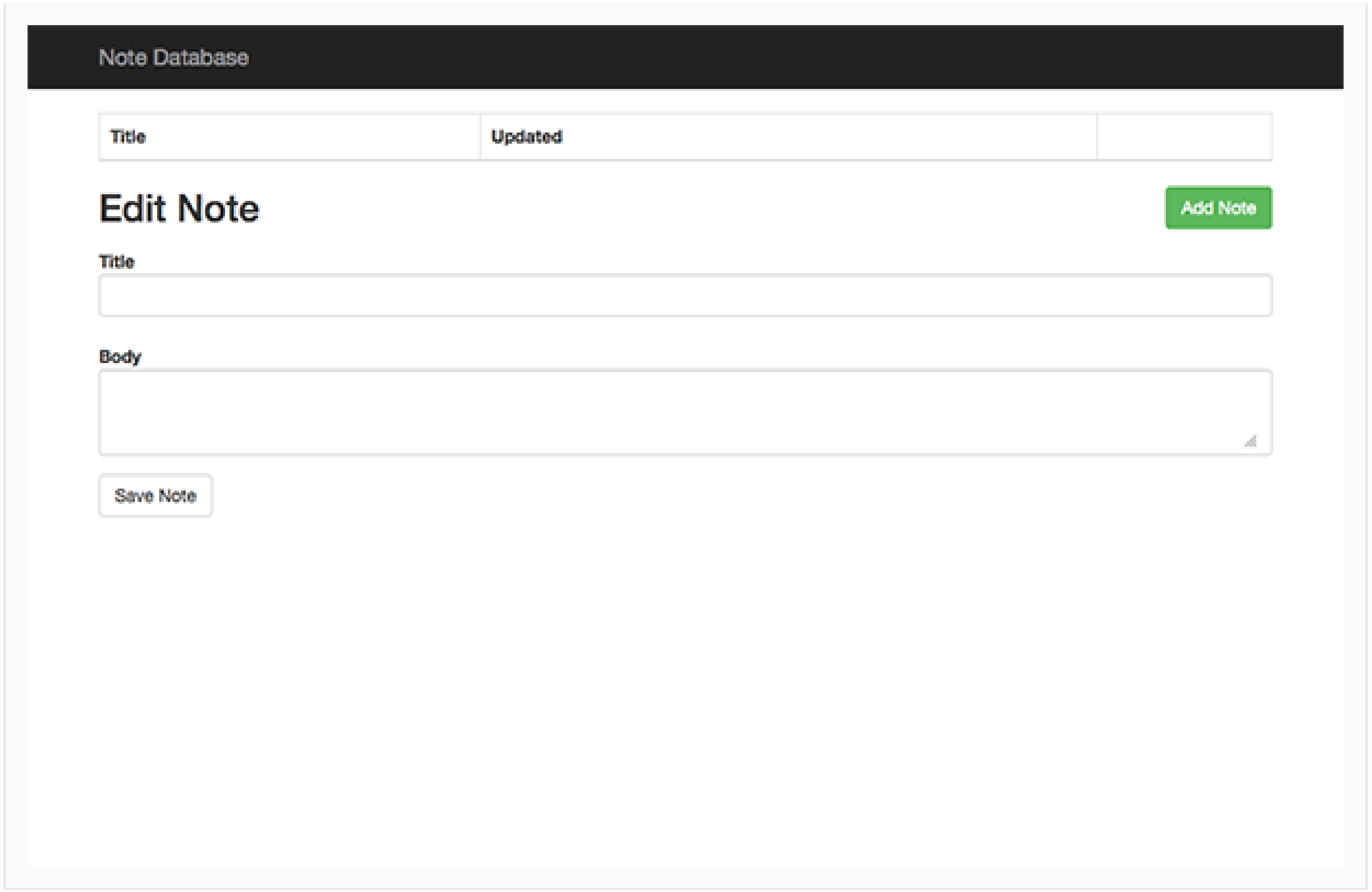
So, as I said, not terribly exciting, which is probably good. You want your APIs simple, boring, and unsurprising. Now let's take what we've learned and bring it together to create a real, if simple, application.

The Note App

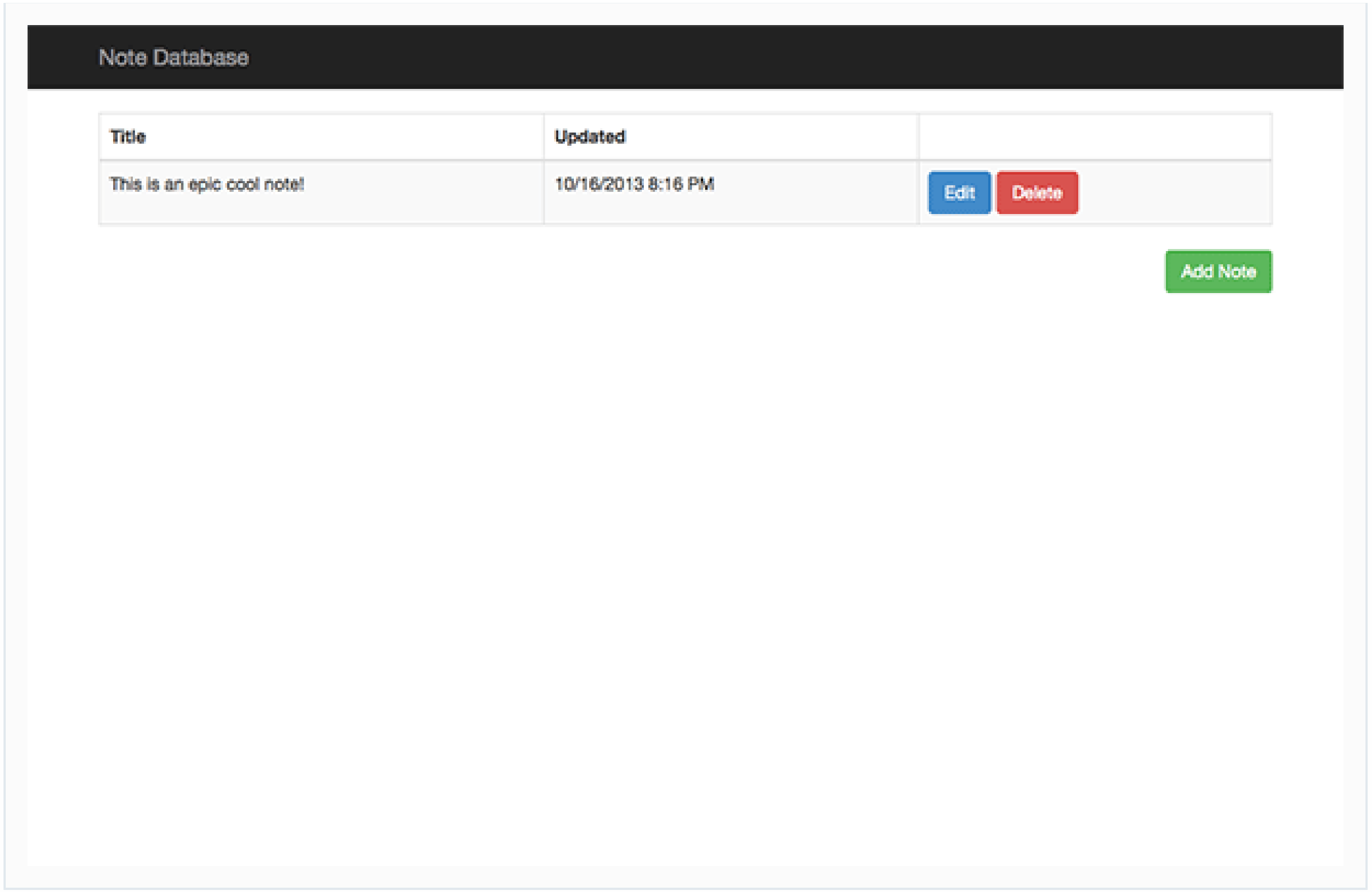
Ok, finally we have all (well, most) of the parts we need to build a real application. Since it hasn't been done before (ahem), we are going to build a simple note taking application. Let's look at a few screen shots and then I'll show you the code behind it. On launch, the application initializes an IndexedDB for the application and renders an empty table. Initially, all you can do with the application is add a new note. (We could make this a bit more user friendly perhaps.)



Clicking the **Add Note** button opens a form:

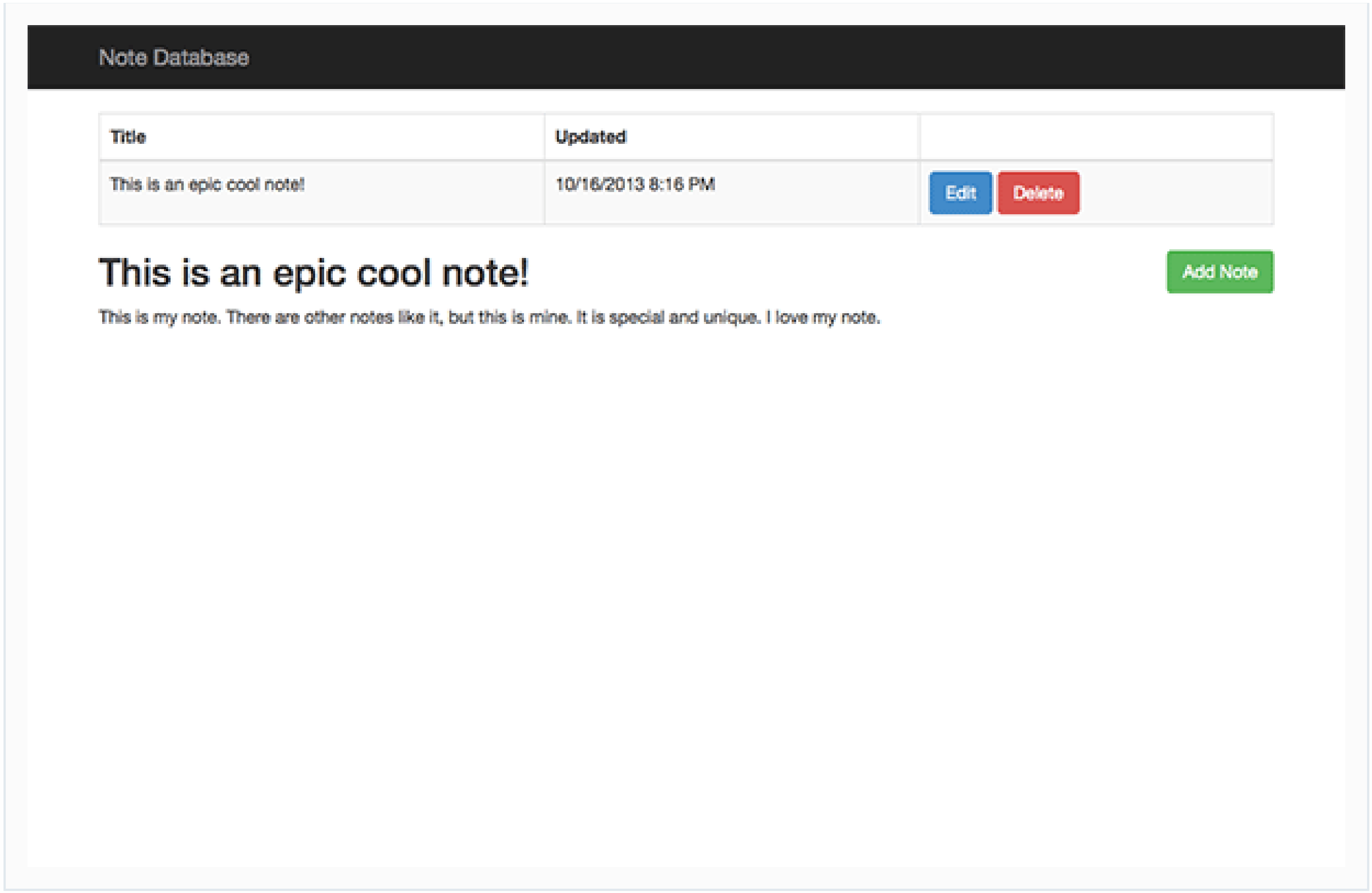


After entering some data in the form, you can then save the note:



Title	Updated	
This is an epic cool note!	10/16/2013 8:16 PM	Edit Delete

As you can see, you have the option to edit and delete notes. Finally, if you click the row itself, you can read the note:


Note Database

Title	Updated	
This is an epic cool note!	10/16/2013 8:16 PM	Edit Delete

This is an epic cool note!

This is my note. There are other notes like it, but this is mine. It is special and unique. I love my note.

So not exactly rocket science, but a full working example of the IndexedDB specification. The notes written here will persist. You can close your browser, restart your machine, take a few years off to contemplate life and poetry, and when you open the browser again your data will still be there. Let's take a look at the code now.

> First - a disclaimer. This application would have been a perfect candidate for one of the many JavaScript frameworks. I'm sure those of you who use Backbone or Angular can already imagine how you would set this up. However - I made the bold decision here to not use a framework. I was worried both about the people who may use a different framework and those who use none. I wanted our focus here to be on the IndexedDB aspects alone. I fully expect some people to disagree with that decision, but let's hash it out in the comments.

Our first template is the HTML file. We've only got one and most of it is boilerplate Bootstrap:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1

    <title>Note Database</title>

    <link href="bootstrap/css/bootstrap.css" rel="stylesheet">
    <link href="css/app.css" rel="stylesheet">

  </head>

  <body>

    <div class="navbar navbar-inverse navbar-fixed-top">
      <div class="container">
        <div class="navbar-header">
          <a class="navbar-brand" href="#">Note Database</a>
        </div>
      </div>
    </div>

    <div class="container">

        <div id="noteList"></div>
        <div class="pull-right"><button id="addNoteButton" class="btn bt
        <div id="noteDetail"></div>

        <div id="noteForm">
            <h2>Edit Note</h2>
            <form role="form" class="form-horizontal">
            <input type="hidden" id="key">
            <div class="form-group">
                <label for="title" class="col-lg-2 control-label">Title
                <div class="col-lg-10">
                <input type="text" id="title" required class="form-cont
                </div>
            </div>
            <div class="form-group">
                <label for="body" class="col-lg-2 control-label">Body</l
                <div class="col-lg-10">
                <textarea id="body" required class="form-control"></text
        </div>
      </div>
      <div class="form-group">
        <div class="col-lg-offset-2 col-lg-10">
          <button id="saveNoteButton"class="btn btn-default"
        </div>
      </div>
      </form>
    </div>

  </div>

  <script src="js/jquery-2.0.0.min.js"></script>
  <script src="bootstrap/js/bootstrap.min.js"></script>
  <script src="js/app.js"></script>
 </body>
</html>
```

As mentioned above, a good size portion of this file is template code for Bootstrap. The parts we care about are the `noteList` div, the `noteDetail` div, and the `noteForm`. You can probably guess that these are the DIVs we'll be updating as the user clicks around in the application.

Coding Our Core App File

Now let's take a look at `app.js`, the core file that handles the logic for our application.

```
/* global console,$,document,window,alert */
var db;

function dtFormat(input) {
    if(!input) return "";
    var res = (input.getMonth()+1) +"/" + input.getDate() +"/" + input
    var hour = input.getHours();
    var ampm = "AM";
    if(hour === 12) ampm ="PM";
    if(hour > 12){
        hour-=12;
        ampm = "PM";
    }
    var minute = input.getMinutes()+1;
    if(minute < 10) minute ="0" + minute;
    res += hour +":" + minute + " " + ampm;
    return res;
}
```

You can ignore the first function as it is simply a format utility for dates. Let's skip ahead to the jQuery document ready block.

Checking for Browser Support

```
$(document).ready(function() {

    if(!("indexedDB" in window)) {
        alert("IndexedDB support required for this demo!"
        alert("IndexedDB support required for this demo");
        return;
    }

    var $noteDetail = $("#noteDetail");
    var $noteForm = $("#noteForm");

    var openRequest = window.indexedDB.open("nettuts_notes_1",1);

    openRequest.onerror =function(e) {
        console.log("Error opening db");
        console.dir(e);
    };

    openRequest.onupgradeneeded =function(e) {

        var thisDb = e.target.result;
        var objectStore;

        //Create Note OS
        if(!thisDb.objectStoreNames.contains("note")) {
            console.log("I need to make the note objectstore");
            objectStore = thisDb.createObjectStore("note", { keyPath: "i
        }

    };

    openRequest.onsuccess =function(e) {
        db = e.target.result;

        db.onerror = function(event) {
          // Generic error handler for all errors targeted at this da
          // requests!
```

```
            alert("Database error: "+ event.target.errorCode);
            console.dir(event.target);
          };

          displayNotes();

      };
```

Our very first action is to check for IndexedDB support. If the user's browser isn't compatible, we use an alert and abort the function. It would probably be better to relocate them to a page that fully explains why they can't use the application. (And to be clear, we could also build an application that made use of WebSQL as a backup. But again - my focus here is on simplicity.)

After caching a few jQuery selectors, that we'll use throughout the app, we then open up our IndexedDB database. The database is fairly simple. In the `onupgradeneeded` handler you can see one object store called `notes` being created. Once everything is done, the `onsuccess` handler will fire off a call to `displayNotes`.

The `displayNotes` Function

```
function displayNotes() {

    var transaction = db.transaction(["note"], "readonly");
    var content="<table class='table table-bordered table-striped'><th

    transaction.oncomplete = function(event) {
        $("#noteList").html(content);
    };

    var handleResult = function(event) {
      var cursor = event.target.result;
      if (cursor) {
        content += "<tr data-key=\""+cursor.key+"\"><td class=\"notetit
        content += "<td>"+dtFormat(cursor.value.updated)+"</td>";

        content += "<td><a class=\"btn btn-primary edit\">Edit</a> <a
        content += "</tr>";
        cursor.continue();
      }
      else {
        content += "</tbody></table>";
      }
    };

    var objectStore = transaction.objectStore("note");

    objectStore.openCursor().onsuccess = handleResult;

}
```

The `displayNotes` function does what you expect - get all the data and display it. We discussed how to get all rows of data in the previous entry, but I want to point out something slightly different about this example. Note that we have a new event handler, `oncomplete`, that we've tied to the transaction itself. Previously, we've used events just within the actions, inside the transaction, but IndexedDB lets us do it at the top level as well. This becomes especially useful in a case like this. We have a giant string, our HTML table, that we build up over each iteration of our data. We can use the transaction's `oncomplete` handler to wrap up the display portion and write it out using a simple jQuery call.

The `Delete`, `Edit`, and `Add` Functions

```
$("#noteList").on("click", "a.delete", function(e) {
    var thisId = $(this).parent().parent().data("key");

    var t = db.transaction(["note"], "readwrite");
    var request = t.objectStore("note").delete(thisId);
    t.oncomplete = function(event) {
        displayNotes();
        $noteDetail.hide();
        $noteForm.hide();
    };
    return false;
});

```

```
$("#noteList").on("click", "a.edit", function(e) {
    var thisId = $(this).parent().parent().data("key");

    var request = db.transaction(["note"], "readwrite")
                    .objectStore("note")
                    .get(thisId);
    request.onsuccess =function(event) {
        var note = request.result;
        $("#key").val(note.id);
        $("#title").val(note.title);
        $("#body").val(note.body);
        $noteDetail.hide();
        $noteForm.show();
    };

    return false;
});

$("#noteList").on("click", "td", function() {
    var thisId = $(this).parent().data("key");
    var transaction = db.transaction(["note"]);
    var objectStore = transaction.objectStore("note");
    var request = objectStore.get(thisId);

    request.onsuccess =function(event) {
        var note = request.result;
        $noteDetail.html("<h2>"+note.title+"</h2><p>"+note.body+"</p>").
        $noteForm.hide();
    };
});

$("#addNoteButton").on("click", function(e) {
    $("#title").val("");
    $("#body").val("");
    $("#key").val("");
    $noteDetail.hide();
    $noteForm.show();
});
```

Our next two methods (`delete` and `edit`) is another example of this same principal. Since none of the IndexedDB calls here are new, we won't bother going over them. Most of the "meat" here ends up being simple DOM manipulation to handle the particular actions. The handler for clicking the add button is exactly that, so we'll skip over that as well.

The `Save` Function

```
$("#saveNoteButton").on("click",function() {

    var title = $("#title").val();
    var body = $("#body").val();
    var key = $("#key").val();

    var t = db.transaction(["note"], "readwrite");

    if(key === "") {
        t.objectStore("note")
                        .add({title:title,body:body,updated:new Dat
    } else {
        t.objectStore("note")
                        .put({title:title,body:body,updated:new Dat
    }

    t.oncomplete = function(event) {
        $("#key").val("");
        $("#title").val("");
        $("#body").val("");
        displayNotes();
        $noteForm.hide();
    };

    return false;
});
});
```

The next interesting tidbit is the `save` method. It has to use a bit of logic to determine if we are adding or updating, but even that is rather simple. And that's it! A complete, if simple, IndexedDB application. You can play around with this demo yourself by downloading the attached source code.

In Conclusion

That's it for part two! The third article will take this application and begin adding additional features including search and array based properties.

Raymond Camden

Raymond Camden is a senior developer evangelist for Adobe. His work focuses on web standards, mobile development and ColdFusion. He's a published author and presents at conferences and user groups on a variety of topics. He loves kittens and Star Wars - not necessarily in that order.

Download Attachment

Translations

Envato Tuts+ tutorials are translated into other languages by our community members—you can be involved too!

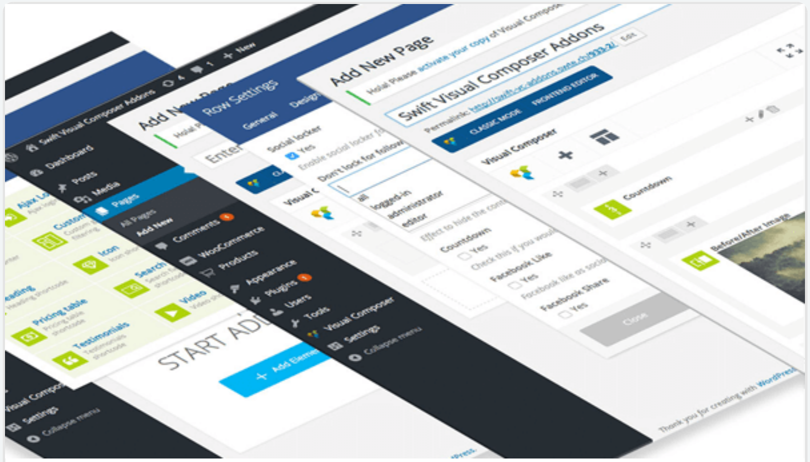
Translate this post

Powered by native

tuts+
BUILD
YOUR OWN CUSTOM
GOOGLE MAPS
Step by step video course.
START NOW

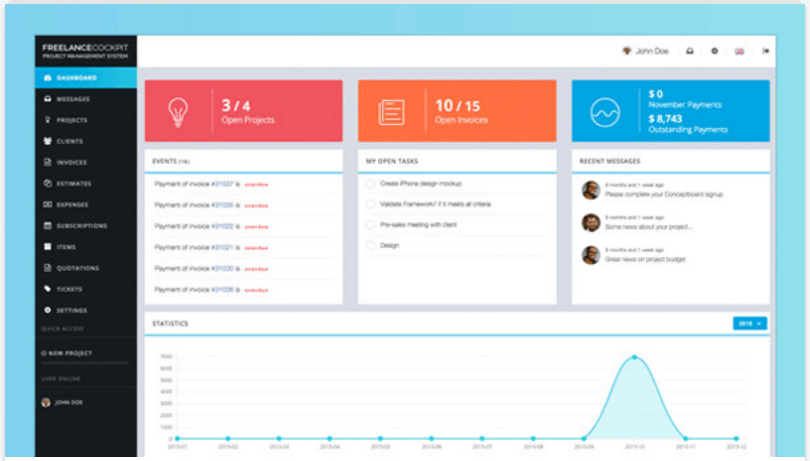
Looking for something to help kick start your next project?

Envato Market has a range of items for sale to help get you started.



WordPress Plugins

From $4



PHP Scripts

From $1

envatotuts+

Teaching skills to millions worldwide.

22,497 Tutorials **900** Video Courses

Meet Envato

About Envato
Explore our Ecosystem
Careers

Join our Community

Teach at Envato Tuts+
Translate for Envato Tuts+
Forums
Community Meetups

Help and Support

FAQ
Help Center
Terms of Use
About Envato Tuts+
Advertise

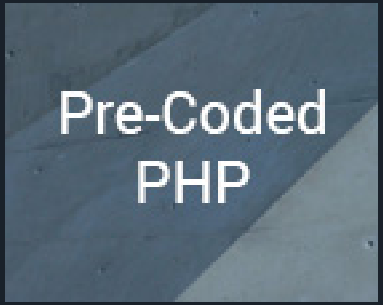
Email Newsletters

Get Envato Tuts+ updates, news, surveys & offers.

Email Address

Subscribe

Privacy Policy



Follow Envato Tuts+