app-developer-tutorial

nil

October 1, 2012

Contents

1 App Developer Tutorial

1.1 Introduction

This guide is meant for web developers, who wish to write applications using remotestorage / remoteStorage.js.

It explains basic concepts of both remotestorage and remoteStorage.js and provides an introduction to the JavaScript API of remoteStorage.js.

1.2 Terms

You don't need to know all of these, to develop an app.

- **unhosted** - When we say "Unhosted Web App", we mean applications written in javascript, that only run within the browser. For more information on the word unhosted, see it's website.
- **remotestorage** - remotestorage is a specification for the interaction between storage servers and web applications.
- **remoteStorage.js** - This is the client side implementation of the remotestorage specification. Put simply, this is the file you include in your webapp, to make it talk remotestorage.
- **CORS** - A cool new feature of the web platform, that lets an application from one server (origin actually), send AJAX requests to another server, without being prevented by the browser to do so (while adhering to restrictions given by that server being queried). As an app developer that uses remoteStorage.js you don't need to no anything about this, except that prevents you from worrying in many ways.

1.3 remotestorage - storage for Unhosted Web Apps

This is not an in-depth explaination of remotestorage. Just a quick glance to see what we'll be dealing with. If you want to know the details of remotestorage, I encourage you to read the specification.

But before I explain any of what remotestorage is made of, let's take a look at how it's used.

1.3.1 From a users perspective...

- ... it may look like this:
  - A friend sends me a link to a cool new app.
  - That app has a neat orange cube in the top right corner, designating the support for remotestorage in the application.
  - I click the cube, enter my useraddress (in the form of user@host) into the appearing textfield, then click "connect"
  - Now I have been redirected to the authorization site of my storage provider.

  - If I haven't already logged into my storage, it will ask me to do so.
  - Now my storage provider shows me information on what permissions the app requests, once I accept that, I'm back at the app.
  - Now the app can read and write to my storage, restrained only through the permissions it requested.
  - Any other app can read and write the same data and hence interoperate.

  The scenario just described is what we call "app-first".

- ... or it may look like this:
  - A friend sends me a link to a cool new app.
  - I visit my storage provider's dashboard (which may already be open).
  - There I paste the link to the app, the app then gets installed on my dashboard.
  - Now I click the app's icon and it opens.
  - My storage provider has provided the app with the necessary information, it needs to know where my storage is.
  - So, as above: from now on the app can read and write to my storage.

  This is what we call the "storage-first" scenario.

1.3.2 From a developers perspective

- user addresses
  User addresses look like email addresses: user@host.

  A user address is what the user in the "app-first" scenario provided to the app to make her storage provider known.

  The app (or rather remoteStorage.js) discovers both storage and authentication server based on the user address through a mechanism called Webfinger.

  In order to discover the storage of a user alice@wonderland.lit, the server at wonderland.lit is asked for information on the user alice. wonderland.lit then provides her profile, which contains links. One of these links is called "remotestorage" and contains the information the app needs.

- authorization
  The remotestorage specification chooses a subset (the "implicit grant flow") of OAuth2 for authorization of apps to access storage. During that process, the app claims a set of permissions it wishes to have on the storage, the user is prompted to confirm those, then the authorization provider issues a bearer token and passes it back to the app by redirect.

- exchanging data: GET, PUT, DELETE
  The storage is accessed through regular HTTP requests:

  - GET requests load data from the storage
  - PUT requests put data into the storage
  - DELETE requests destroy data on the storage

  The path can be any valid HTTP path. There is one exception though: Paths ending with a forward slash (/) represent "directories". A GET request to a directory always results in a JSON object, listing the direct child-nodes of that directory, and their respective mtime. Both PUT and DELETE requests to directory paths are illegal and have no effect.

  The permissions mentioned above, place a few restrictions on what paths exactly an app may GET, PUT or DELETE to. A single permissions statement consists of a **path** and a **mode**. The **path** is relative to the root of the storage and must represent a directory. The **mode** may either be "r" for "read", or "rw" for read-write. When access to a given **path** has been requested and confirmed, only requests to that path and nodes below it can be performed by the app.

- public data
  An exception to the permission model described in the last paragraph applies to all paths starting with the string "public/". Non-directory nodes (aka files, i.e. those paths not ending in a slash) can be read (GET) by anyone without authorization. Directory nodes on the other hand can only be read, with sufficient permissions. Also PUT and DELETE requests require authorization.

  By definition, when granted permission to read / write at a given path, permission to write at the same path prefixed with public/ as well is granted implicitly. That is, if an app requests to read-write to "tasks/", it can also read-write to "public/tasks/".

  That way sharing of data can be implemented.

1.4 remoteStorage.js - accessing remotestorage from an App

remoteStorage.js is a client side implementation of remotestorage in JavaScript.

It provides the following things for an app developer:

- A widget, through which the user controls it's connection to remotestorage.
- A set of modules to interact with common kinds of data (such as list of friends/contacts, locations, photos, ...).
- An interface to write a module which models the data your app works with.

As the rest of this document is about remoteStorage.js, I won't say any more about it here, but instead get you set up with remotestorage first, in the next paragraph.

1.5 Setting up remotestorage

1.5.1 Get developer access

- heahdk.net
  heahdk.net is the staging instance for remotestorage-ruby. You can simply signup, and you're good to go. Your **user address** will be <login>@heahdk.net. remotestorage-ruby also supports the storage-first scenario.
- 5apps.com
  5apps provides storage for developers. Sign up regularly on 5apps, then ask in the #5apps IRC channel for remotestorage developer access.
- owncube.com
  owncube is a hosted owncloud service. It uses the owncloud plugin for remotestorage. This might not work with remoteStorage.js 0.7, see this issue for status.

1.5.2 Set up your own

The State of the movement page has a list of compatible storage software.

1.6 Getting remoteStorage.js

1.6.1 Include the script

As remoteStorage.js is still a moving target, as of this writing I suggest you use the HEAD version, directly from github:

```
<script src="https://raw.github.com/RemoteStorage/remoteStorage.js/master/build/0.7.0-
```

or the debug version:

```
<script src="https://raw.github.com/RemoteStorage/remoteStorage.js/master/build/0.7.0-
```

1.6.2 Using git

You can clone remoteStorage.js like this:

```
git clone git://github.com/RemoteStorage/remoteStorage.js
```

1.7 Let's hit the code!

1.7.1 A skeleton

So, after a lot of preface, now some code:

```
<!DOCTYPE html>
<html>
  <head>
    <script src="https://raw.github.com/RemoteStorage/remoteStorage.js/master/build/0.
    <script src="app.js"></script>
  </head>
  <body>
    <div id="remotestorage-widget"></div>
  </body>
</html>
```

what you see above, is the basic skeleton. It contains the bare minimum for a remotestorage app:

- the remoteStorage.js library
- a placeholder for the widget
- the app's code, within app.js

I haven't shown you the app's code yet, so here it goes:

```
window.onload = function() {

  remoteStorage.claimAccess({ 'tasks' : 'rw' });
  remoteStorage.displayWidget('remotestorage-widget');

}
```

This does two things:

- Set permissions the app wishes to have (in this case it wishes to "read" (GET) and "write" (PUT, DELETE) data in the "tasks" category)
- Display the widget, within the placeholder div.

As soon as the widget is visible, the user may interact with it, so at that point all permissions must have been requested. In other words, all calls to **claimAccess** MUST come before the call to **displayWidget**.

1.7.2 Common data: tasks