Lessons from (unsuccessful) play with the files in HTML5....

...during the HTML5 hackathon in August

Loose objective...

- To learn something about working with files in HTML5

Some more thoughts...

- Try to do something with the Google Storage API and the Filesystem APIs
  - Make some Dropbox like client...

Naivete...

- Thought we could do a too much in the browser...

Problems...

- Problems talking to Google domain from content served from non-google domain
  - Doh!
- Filesystem API only relevant for single domain
  - Cannot be used across multiple domain
    - Doh again!

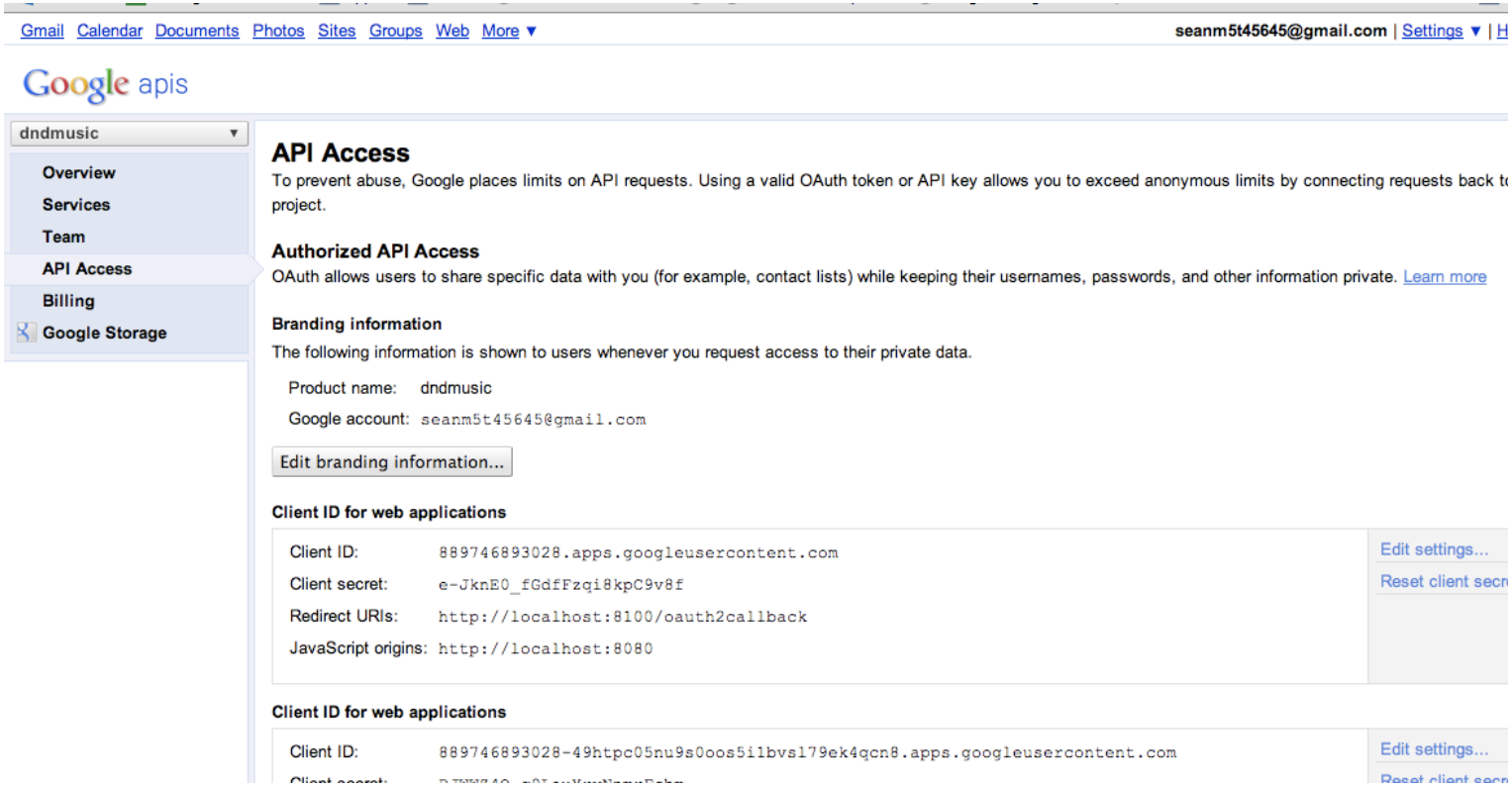
Regroup...

- Implement a filesharing type application for a group based on Google Storage
  - server side mechanism to talk to GS
  - use Filesystem API on client side to store content accessed from server
  - had a music component to story
    - intended use case was music sharing...

Talking to Google Storage

- Create new application using the Google APIs console
  - For OAuth purposes - create client id, secret and specify callback
- Give access to user's GS account via OAuth
  - OAuth link on page
  - Parameters include - client id, redirect, permission required (devstorage.fullcontrol)
    - read_only and read_write also permitted

Google API console



Complications with access...

- Access permitted to two types of users:
  - Registered Google Storage users
    - Use OAuth flow
  - Non Google Storage users
    - Use a cookie based mechanism with specific URI
      - sandbox.google.com/storage

Struggling to get single user working...

- Ignored multi-user use case...
- Used Boto library...
  - Had some issues with this and ended up using S3 in a basic fashion...

Filesystem API

- Support file storage locally for webapps
  - Cannot be accessed by different domain
- More sophisticated than key-based Local Storage mechanism
  - Can support hierarchies of files of different types

Filesystem API

- Two types
  - Temporary - browser can remove content as necessary
  - Persistent - browser will only remove content on approval (user or app)

Initialization

- `window.requestFileSystem(type, size, successCallback, opt_errorCallback)`
-

Error Handler...

```
function errorHandler(e) {
  var msg = '';

  switch (e.code) {
    case FileError.QUOTA_EXCEEDED_ERR:
      msg = 'QUOTA_EXCEEDED_ERR';
      break;
    case FileError.NOT_FOUND_ERR:
      msg = 'NOT_FOUND_ERR';
      break;
    case FileError.SECURITY_ERR:
      msg = 'SECURITY_ERR';
      break;
    case FileError.INVALID_MODIFICATION_ERR:
      msg = 'INVALID_MODIFICATION_ERR';
     break;
    case FileError.INVALID_STATE_ERR:
      msg = 'INVALID_STATE_ERR';
      break;
    default:
      msg = 'Unknown Error';
      break;
  };

  console.log('Error: ' + msg);
}
```

FileEntry & DirectoryEntry

- Flexible interfaces which permits standard file operations and supports typical properties
- FileEntry supports read, write, create, delete, etc
- DirectoryEntry supports directory navigation

Listing Directory...

```
window.requestFileSystem(window.PERSISTENT, 5*1024*1024,
                  function(filesystem) {
                  fs = filesystem;
                  console.log('filesystem got!');

                  var dirReader = fs.root.createReader();
                  var entries = [];

                  // Call the reader.readEntries() until no more
                  // results are returned.
                  var readEntries = function() {
                      dirReader.readEntries (function(results) {
                      if (!results.length) {
                          listResults(entries.sort());
                      } else {
                          entries = entries.concat(toArray(results));
                          readEntries();
                      }
                      }, errorHandler);
                  };

                  readEntries(); // Start reading dirs.

                  }, errorHandler);
```

FileEntry - isDirectory

```
function listResults(entries) {
  // Document fragments can improve performance
  // since they're only appended
  // to the DOM once. Only one browser reflow occurs.
  var fragment = document.createDocumentFragment();

    entries.forEach(function(entry, i) {
     var img = entry.isDirectory ? '<img src="/img/folderIcon.gif"
    width="20" height="20">' : '<img src="img/file_icon.jpg"
    width="20" height="20">';
     var li = document.createElement('li');
     li.innerHTML = [img, '<span>', entry.name, entry.size,
      '</span>'].join('');
     fragment.appendChild(li);
    });

    document.querySelector('#filelist').appendChild(fragment);
}
```

Gotcha...

- Had problems with Chrome not providing Filesystem API
  - needed to run with `unlimited-quota-for-files` with flag

Conclusions, observations...

- Filesystem API still very raw
  - very limited support right now
- Google Storage has a lot of complexity
  - Much of which related to access control
- Need more clarity on project when doing hackathon ;-)