Databases Final Project

David Katz
Harrison Zhao
Alejandro Acosta

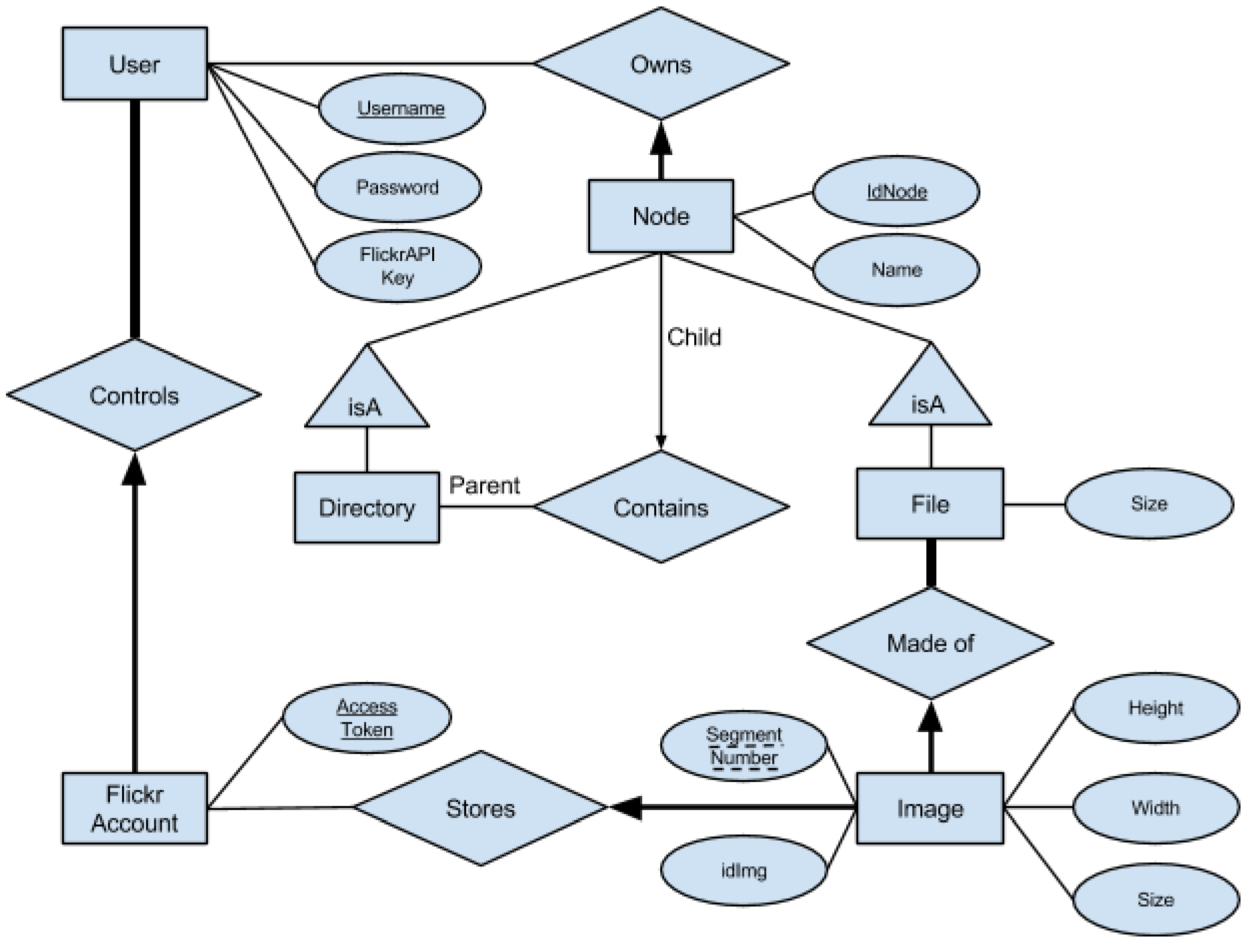
Infinite Cloud Storage

Overview:

Our project is a distributed file system based built on top of Flickr. Files are read byte by byte, encoded into images, and uploaded to Flickr. Files are decoded by downloading the proper image from Flickr, decoding the image into its original format, and rewriting it into a file. The image files were encoded as Portable Network Graphics (PNG). Because the PNG format uses lossless compression, perfect restoration of the file is guaranteed. Our system has a dropbox like interface which allows users to upload, download and delete their files.

Our primary concern was scalability, both for users and for the system as a whole. To address this, we gave users the ability to attach multiple flickr accounts to a single account on our system. Since each account provides 1 TB of storage, attaching 3 accounts would provide 3 TB of storage. We also require each user to provide their own API key. Flickr limits each key to 3600 API calls an hour. Flickr also imposes a limit on the number of requests a machine with a given IP address can make within a 24 hour window. With many users this would not scale, but again a reasonable user should not be able to make nearly this many calls. Finally, the amount of computational load on our server is extremely low. While the server signs the keys and requests, the client handles the encoding, uploading, and downloading of images.

Our software stack was built primarily in Node.js using the Express framework. We also used MySQL as our database and Angular.js as a frontend framework. A variety of other Javascript libraries were also used to assist in API calls, database interactions and other miscellaneous tasks.



Updated ER diagram:

Our ER diagram was expanded from our previous proposals to support the many to one relationship between Flickr Accounts and users. Each user has at least one Flickr Account associated with it. Each Flickr account is associated to exactly one user, and will add 1 TB to the total available storage for that user. When files are uploaded a query checks which of the user's Flickr accounts has the most free space, and uploads there. This association of the image to that specific Flickr account is then stored within the database.

Registration:

When a user registers, they provide a username, password, and their Flickr API keys. Before they can use the virtual file system the user must then attach a Flickr account to these keys. To do so, a user selects the button in the top right corner to attach a new account. This will open a page where the user must login to their Flickr account and agree to give our application permission to upload and download files on behalf of the user. Doing so generates an authorization token associated with that account an API key. We store that key in the Flickr Accounts table, and can use it to create upload/download requests for that Flickr account. An added benefit is that the user does not need to memorize the password for their Flickr account as the authentication tokens stored on the database do not expire.

Upload:

To upload a file, we used a javascript library to load the file's contents into memory as a base64 string. From there, the string was padded to produce the proper number of bytes for an uncompressed image of a given size, and then encoded into a PNG binary. This was then uploaded to Flickr via their API, and upon success our database and the user's current session was updated with the new file. Our server signs the request. However the client performs the upload. The file that then appears on the client is available for download upon click.

Directories and Traversal:

Directories can be created by simply adjusting the name of the "new directory" field and clicking create a directory. Upon doing so, the directory is added to the database and user's session. The user can then click the directory to traverse into it or the go up button to return to its parents. These actions do not affect the database, but use a query to extract the contents of the new directory they are moving into. When attempting to move up from the root, the user remains in their current directory.

Download:

All files are available to be downloaded when clicked. Via the imageId and access token stored in the server the server retrieves the raw url of where the image is stored. The client can then (after some difficulty explained later) retrieve the file as a base64 PNG string, decode it into its original file format, and write the file back to the user's filesystem. The bit rate appears to be in the megabytes range for download.

Delete:

Delete is performed recursively. If a directory is deleted all of its children nodes are also recursively deleted.

Issues Encountered:

One issue that we encountered was client side downloading of files. In order for the browser to retrieve data from another website via AJAX requests, the remote server must implement cross origin resource sharing (CORS). In our case, the static website where Flickr stores each file does not implement CORS. As a result, we initially used the server as a proxy and streamed the file over from the server to the client. This caused downloading files to take longer as the server would have to first download the file and store it in memory before streaming it to the client. To overcome this issue, we used the static website's url to load the PNG stored on Flickr into an HTML img element. We then serialized that img element into a base64 string and decoded the image.

Another issue encountered is that most browsers will kill threads which last too long. For this reason extremely large uploads are prone to failing depending on connection speed. Flickr supports images of up to 200 MB. Thus, even without the splitting of images this 200 MB is likely larger than the browser will support uploading. A concern here would be the fragmented read of files. The application currently is able to download files because it stores the entire file in memory and then downloads the file as a binary. However a problem occurs with downloading fragments of files as we would then need to write to the filesystem. This would require us to request permissions from the user to be able to write to their filesystem. For this reason, we opted to push the splitting of images into multiple parts as a future goal.

Future Work:

File Sharing:
Allows users to mark files as public, and retrieve the file's Id. Create another page which will allow users to insert a file Id and download it if it is public. We could also expand this feature to support only sharing files to specific users.

File Syncing:
Create a desktop application that creates a folder that automatically syncs files stored on Flickr to a user's local file system.

File Renaming:
Having the ability to rename files.

File Preview:
For small enough files be able to preview them.

File Upload and Download Progress Bar:
Progress bar for upload and download percentage

File Sharding:
Uploading takes a long time and the browser might kill a thread if it takes too long. Sharding also enables larger uploads.