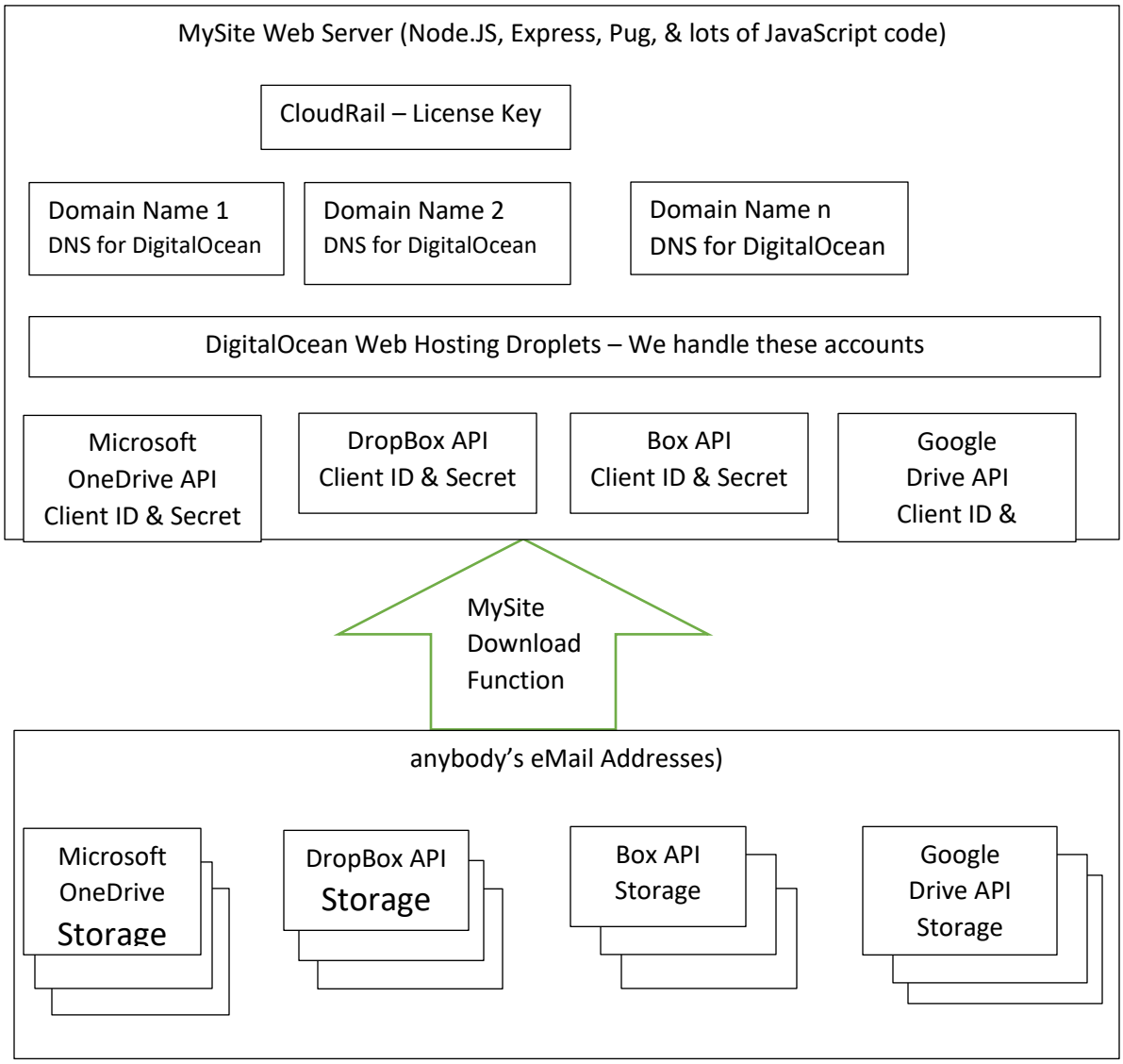
MySite- Architecture

This Document defines the general architecture of this website.

Accounts used to create MySite## websites



A simple single Microsoft OneDrive MySite - One website & one OneDrive cloud service account:

- Create a Microsoft email address (doing this automatically gives you):
    - Access to 5GBs free OneDrive storage
    - Allows you to create a OneDrive API
- Using your Microsoft email address:
    - Acquire CloudRail License Key (www.cloudrail.com)
    - Acquire a Microsoft OneDrive API (http://www.signup.live.com/
    - Store your photos and videos on your Microsoft OneDrive storage

MySite Free Demo Site

The figure above is the maximum configuration I use to allow anybody to use MySite##.com using their cloud service storage. A person would create a folder (myWebsite) on the root of their cloud service. Then create the folders “Photos” and “Videos” under “myWebsite”. Then store their photos in the Photos folders, and videos under the Videos folders. They create additional folder (which turn into menu items on the website). Then they perform a Download function in MySite10 and enter their email address and their password. MySite##.com downloads and builds the website using their cloud storage files.

The Document Object Model (DOM)

Reference: https://en.wikipedia.org/wiki/Document_Object_Model

Service Side Web Application

Reference: https://en.wikipedia.org/wiki/Server-side

Node.JS

Reference: https://en.wikipedia.org/wiki/Node.js

I choose Node.JS because of the asynchronous operations it supports, and the need to dynamically alter the Photos and Videos pages with the proper menu items.

Node.js is an open-source, cross-platform JavaScript run-time environment for executing JavaScript code server-side. Node.js enables JavaScript to be used for server-side scripting, and runs scripts server-side to produce dynamic web page content *before* the page is sent to the user's web browser.

Node.js brings event-driven programming to web servers, enabling development of fast web servers in JavaScript. Developers can create highly scalable servers without using threading, by using a simplified model of event-driven programming that uses callbacks to signal the completion of a task. Node.js was created because concurrency is difficult in many server-side programming languages, and often leads to poor performance. Node.js connects the ease of a scripting language (JavaScript) with the power of Unix network programming.

Node.js applications are events-based and run asynchronously. Code built on the Node platform does not follow the traditional model of receive, process, send, wait, receive. Instead, Node processes incoming requests in a constant event stack and sends small requests one after the other without waiting for responses.

Node.js is a runtime environment and library for running JavaScript applications outside the browser.

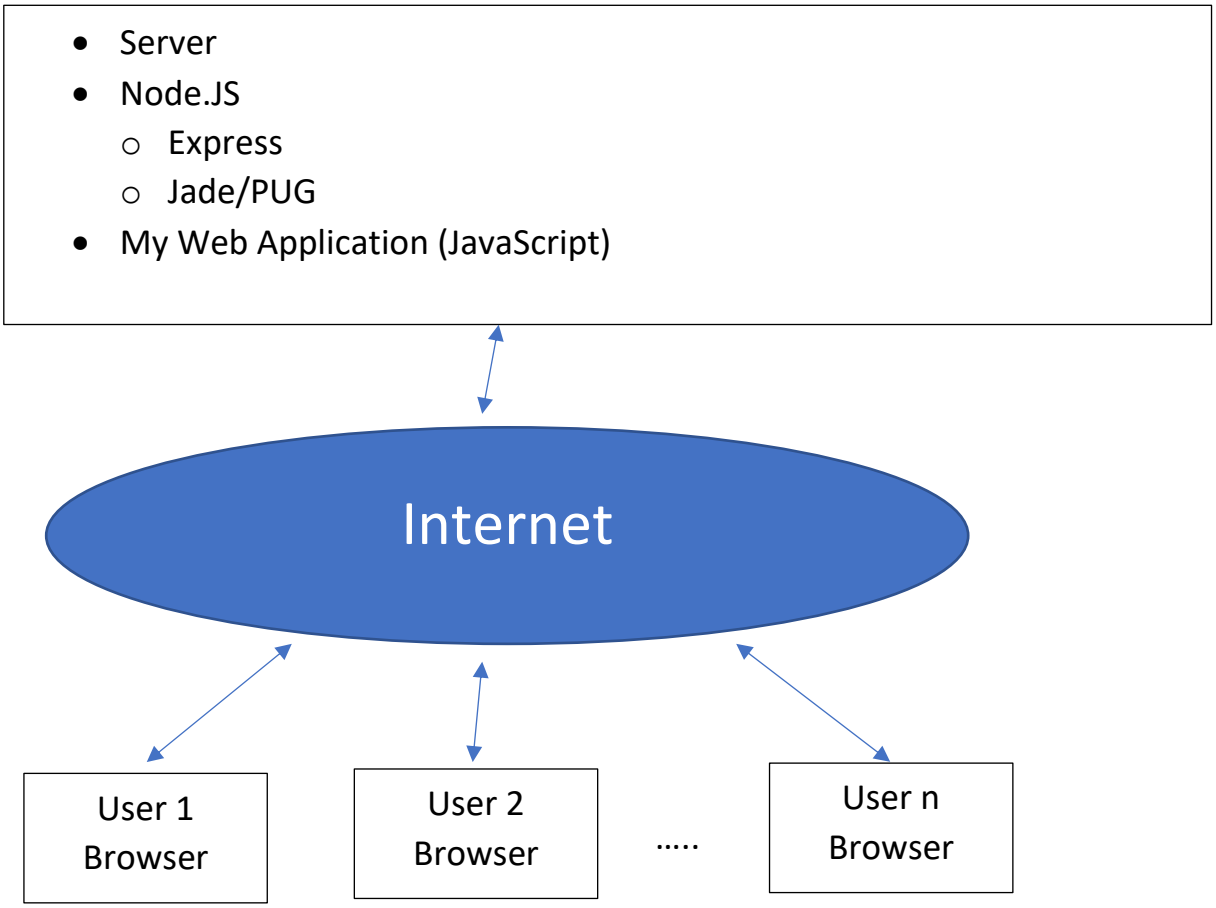
Node.js is mostly used to run real-time server applications and shines through its performance using non-blocking I/O and asynchronous events.

When to use Node

- Node is great for streaming or event-based real-time applications like:

    - Chat Applications
    - Real time applications and collaborative environments
    - Game Servers
    - Ad Servers
    - Streaming Servers
- Node is great for when you need high level of concurrency but little dedicated CPU time.
- Great for writing JavaScript code everywhere!

Component Overview



General Page Layout

Logo & Banner

Logo | Banner

Slider

Static Images that slide to left

Main Menu Bar

HOME ABOUT PHOTOS VIDEOS CONTACT DOWNLOAD

Alternate Navigation Menu Bar

Photos Family & Friends Family

Leftside Navigation Menu Bar

- Grandparents
- Great Grandparents
- Mother & Father

Photos or Videos Page

Photos Galleries, Photos, Video Player displayed here

Footer / Copyright

Logo & Banner

This is where my website Logo and Banner is displayed. This is displayed on all pages.

Slider

These are 6 or 8 images that represent my website and are rotated to the left. This Slider is show in the HOME, ABOUT, and CONTACT pages, and not on the Photos, VIDEOS, and DOWNLOAD pages.

Main Menu Bar

This is shown on all pages. It allows goes to the same page when selected. Once you select the PHOTOS or VIDEOS from this Main Menu, you will be given the Alternate Navigation Menu Bar and Leftside Navigation Bar. Use the Leftside Navigation Bar to continue forward in your browsing for what you are looking for. The Alternate Navigation Bar shows the menu you came from.

Alternate Navigation Menu Bar

This Bar shows you where you came from to get to the current menu selection. You can select any of these menu items to take you back to that page.

Leftside Navigation Bar

This Bar shows you what menus you can select, based in where you are currently at. If there are no menus items displayed here, you have reached the end of this menu.

Photos and Videos

When your using the PHOTOS menu, this area will display a gallery (list of photo icons) of all the photos for this menu selection. When you select one of these icons, the photos will be displayed in this area. You can select the left or right arrow to see the previous or next photo. Click on the “X” to get back to the gallery.

When using the VIDEOS menu, this area displays a video player. If there are no videos associated with the video page you are on, the video player will be displayed with a black screen. If there is at least one video associated with this video page, the pictures associated with this video will be displayed. If there are

more than one video associated with this menu page, a paly list with the video player will appear. You can select any video to play. Controls will be available to make the video full screen, Stop, and Play/Continue.

Main Menus

A typical webpage has static information, meaning when you go to this webpage, it displays the same information every time. This is the case for the Home, About, Contact, and Download pages. However, the Photos and Videos pages will contain different information depending on the what navigation menu was selected.

Single Page Architecture

A typical website has a different page for every menu item selected. This means you can <Copy> the browser address bar and send I to anyone. They would see the same page on their browser. You can bookmark any page, and when you pull up the saved bookmark, it will show you the same page. Mysite website is different.

MySite website only has one Photos Page and one Videos Page. These 2 pages contain 3 different sections that change each time a menu item is selected on these pages. These sections are:

- Content – This sections contains the information need to display the pictures on the Photos page and the video(s) to be played on the Videos page.
- Left Side Navigation Bar – This section has the information needed to display the forward menu items.
- Alternate Navigation Bar – This section has the information needed to display the backward menu items. These are the menu you can from starting at the beginning of the Photos or Videos page from the main menu. To goto the previous menu, don't use the Back-Arrow tab on the browser window. Select any one of the menu item on this Alternate Navigation Bar to get back to that location.

These 3 sections are built into files during the Download process. Each folder name is turned into a menu item (both Left and Alt). Each folder contain the files from the Cloud Service myWebsite and these 3 section files.

When a menu item is selected from the Photos or Videos page, the folder path is sent from the client's browser to the server. The server copies the 3 files from this path to the section of either the Photos or Videos page, and sends the page back to the client's browser.

After the download function is completed, the file system on the server has the same folder and files as is on the Cloud Service myWebsite folder, with the addition of the 3 new section files in every folder.

The biggest advantage of this technique is it allows for a website to be built without having to do any website development. You just upload picture and videos files to your Cloud Service and Download using the Download Function on mySite website.

This technique has other pros & cons

Pros:

- Photos and Videos are hidden from browser search engine cache histories. This is a major security feature. Internet predators can not see your photos and videos via cashed histories. In addition, the Main Photos and Main Videos Pages can be password protected, giving you another level of privacy.

Cons:

- If you <Copy> or bookmark any Photos or Videos page, other than the ones from the Main Menu, and you pull it up, it will not be the one you bookmarked. It will always be the Main Photos or Main Videos Page.

Asynchronous or Synchronous

One of the primary reasons Node.JS was chosen for Release 1 was its asynchronous JavaScript capabilities. Having the capability to do additional processing while waiting on other things to complete seemed like a good thing. Hindsight proved me wrong.

The Download function ended up queuing many of the processing tasks waiting for the firsts ones to complete.

Downloading files from Cloud Services

When issuing a File_Read_Request to download a file from the cloud service, it would take transmission time, which is very close compared to the processing power of the web server. During this transmission time, mysite would get ready to perform the next task (which was to download another file from the cloud service. Since we had to wait for the current File_Read_Request to complete, mysite would call this function reclusively (putting it on the stack). This ended up putting thousands of tasks on the stack. During some Download operations with large numbers of small photos files needing downloaded, the mysite server would reach a stack overflow condition and the server would crash and reboot. All I had to do is issue another "Download Just New Photos" function and the downloaded completed successfully.

This process of putting tasks on the stack was compounded by the cloud services enforcing thresholds as to how often a request could be issued to them.

Threshold Dependencies Imposed from Cloud Services

I found out during the initial testing of Release 1 that the different cloud services imposed thresholds. If a request was make to a cloud service that was too short of time from the last request, the cloud service would return an error and not perform the operation requested. I was under the false impression the CloudRail would handle this condition by waiting some time, then reissuing the request. I was wrong,,, again. My solution to this problem was to implement a Throttling Queue prior to issuing a Folder or File read request. This throttle would slow these requests down, in line with each cloud service thresholds. OneDrive used a very low threshold value (5 ms) while Drive had a much longer threshold (250 ms). This throttling made more tasks being put on the stack, which increased to possibility of system crash do to a stack overflow.

The other problem was that if an error did occur, there was nothing I could do, other than log the error and drop this operation and pull the next thing off the stack. I could have tried to determine if the error was due to a threshold, and delayed the current request, and reissue it, but I didn't. I just kept increasing the throttle value until I stopped getting errors. This just increased the overall download time tremendously for Drive and DropBox.

Again, I was able to provide ‘work arounds’ to solve this issues, but now that I am starting Release 2, I want to provide better solutions.

Recursive Solution

The function AddAllCsFs is one of the recursive functions for the “Download All Photos”. This function is passed a SourceFolderPath which is the file path of the folder to be downloaded from the Cloud Service. The destination of the path contains additional folder names, files, or nothing. If it contains at least one file, then and some folder, all the paths to the new folders will be put on a queue (do NOT call this function reclusively). Wait for the completion of the file(s) in this folder to be downloaded. Then check the queue, and if something there, pull one off and call this AddAllFs function recursively.

When all files for a path has been successfully download, check the queue, and it something there, pull it off and call this AddAllFs function recursively. If nothing on the queue, return to the callback. This will allow the download process to continue, without putting many things on the reclusive stack.

Threshold Solution

Continue to use the same Throttle process as Release 1. However, if an error is returned form the Get_Child or Get_File CloudRail request, try to determine if it is due to a threshold error, an do the Throttle Queue Wait process, to try it again.