
Name: Russel Demos

Subject: IFB 299 Application Development

Assessment: Personal Portfolio

Student Number: 9465987

Release 1

Javascript & Linting

Learning modern javascript industry practices was the first contribution to release 1, and was vital to the entire project. Javascript has many strange properties to the language. For example, at a high level it is a typeless language, however the JIT (Just In Time) Compiler has types. A variable that is set in javascript to 5, will be interpreted to a 31 bit signed integer; assuming static, it's value will be tagged if the variable becomes dynamic.

This is important to remember for instances where one would like to perform a typeof operation. As javascript has no high-level types, instead the function opts to return a string with the type in characters. A number to a "number", a boolean to a "boolean", and null values to "object"; of course. It is edge cases like typeof(null) that will constantly arise when developing in javascript, providing ample amounts of artificial complexity.

Thus, standards have been formed as an attempt to mitigate the peculiarities of the language with options such as JSHint. JSHint will force standards set in a rule file allowing for both assured quality of code between collaborators, as well as a tool for debugging code that can be misinterpreted. The step up from this was to use an industry made style guide from Airbnb, with notable use of 'let' and 'const' for efficiency in interpretation. As mentioned earlier, if a variable changes type, the JIT Compiler will tag the variable, and de-optimize to allow for the dynamic variable. This may be fine in the browser where the client only deals with small data, but in a server environment this can lead to extremely bad scaling performance.

An important video used to research effective javascript development and the peculiarities of the language was made by Rob Ashton:
https://www.youtube.com/watch?v=PV_cFx29Xz0

NodeJS Primer

Jumping into NodeJS was very different from standard web architectures. For example with Java and PHP the server relies on multi threading to handle multiple requests in parallel, compared to NodeJS's single threaded event loop. For Example; An Actor/Worker model is implemented:

In Java using akka, each request job is pushed to a queue, then depending on the server's thread pool, workers are spawned in parallel and execute the given tasks.

In Javascript, the event loop will work on each request partially at a time. Thus a major paradigm is to write asynchronous code that can only be executed concurrently, not in parallel. Thus as requests grow, the work of a java server will remain flat until the thread pool is exhausted. The NodeJS event queue however only adds jobs to do concurrently, thus time to execute increases over time.

It is important to note that NodeJS will be extremely efficient with a callback based system under low load, however as scale grows it cannot compete with Java that runs in 1/N of the time, where N is the number of threads given in its pool.

Another major aspect to NodeJS is the modules created by the community. The Node Package Manager (NPM) is responsible for handling thousands of community created packages, extremely useful for building servers with speed, but detrimental when not handled with care.

The benefits to javascript in this manner comes down to:

- Easy to develop small projects
- Quick to deploy in linux environments
- Large community

The disadvantages however:

- Non-Scalable
- Platform constraints
- A broken package will destroy server

This information was highly important for the coming work.

NodeJS basics were learnt online from ‘thenewboston’:
https://www.youtube.com/watch?v=-u-j7uqU7sI&list=PL6gx4CwI9DGBMdkKFn3HasZnnAqVjzHn_

The HTTP framework Express was learnt online from Derek Banas:
https://www.youtube.com/watch?v=xDCKcNBFsuI

Akka:
http://akka.io/

Project Scoping

Our NodeJS veteran was short for time during the scoping of this project, thus a couple of the team members, myself and two other, attempted with the knowledge learnt about the technology chosen to scope the project out, as well as devise ways we could accomplish the task efficiently with modules and services used in industry. This project scope file can be found as ‘’ in the attached documents.

Framework research was done through:
https://www.devsaran.com/blog/10-best-nodejs-frameworks-developers
http://expressjs.com/

Appropriate Modules were found through NPM’s Best:
https://www.npmjs.com/browse/star

S3 API V1. - Basics

S3 is a cloud based storage service provided by Amazon Web Services. It operates on the basis that folders named ‘buckets’ can be created. These buckets are free of charge to create, however the files stored inside them have charge per MegaByte.

Thus the implementation of this API had to be carefully made in a way that there would be no iteration of file uploading in the edge case that a conditional was improperly set. This was due to the fact one of our team members donated their account for the development of this API. The basic implementation is split into 2 parts; configuration and route.

Note: Anywhere keys or keys.json is referenced in the source is a reference to the API Keys for the project.

In the configuration the module had to both export the S3 object itself, as well as the client. Creating the client was only a matter of setting parameters that were deemed appropriate for the project taken. This configuration file can be found as ‘s3.js’ in the attached documents.

The route was decided upon to allow any form that takes in image requests to upload to the bucket provided by our team member. First we create a stamp based on the time since epoch, this will be used to name the file. Due to the NodeJS event loop structure, this is safe. Next we’ll use multer to take the requested image files and store them locally in a temporary directory. We’ll perform some checks on the files. We’ll then create a client uploader from the S3 API. When the file is done uploading, we’ll get the resulting link from the file in the bucket and delete the file locally. This route file can be found as ‘sendFile.js’ in the attached documents.

S3 Cloud Technology was learnt online from Amazon:
https://www.youtube.com/watch?v=Yyraql9A_Rc

Documentation for the S3 Module was crucial for development:
https://www.npmjs.com/package/s3

Release 2

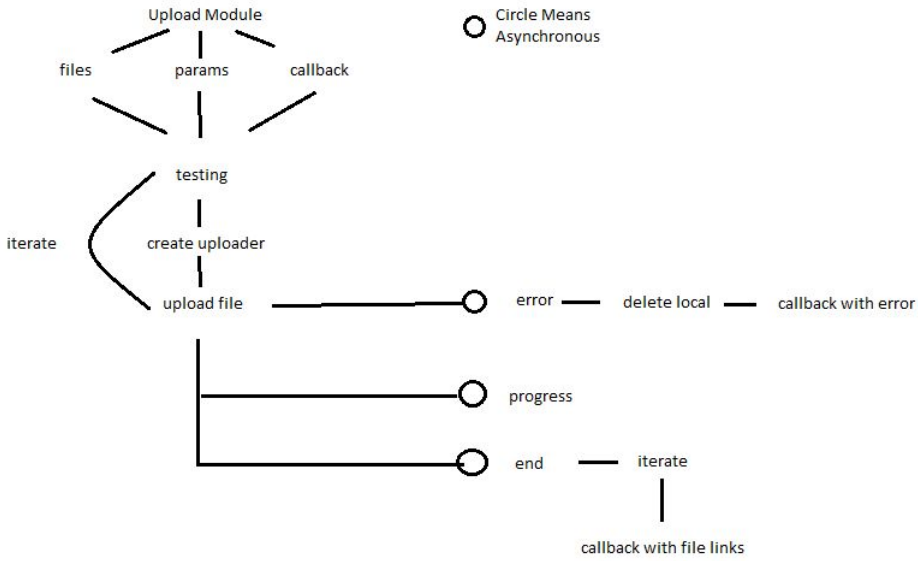
S3 API V2. - Module

Configuration for S3 remained the same in Version 2, however a modularised version of the API was required to allow a file to be uploaded to any bucket, in any route. Thus it was designed to take a callback, similar to a function pointer in other languages. It was also designed to take a json object of parameters, this allowed flexibility in file parameter checking in future as all that would need to be done is to just add the test case at the top of the file. What differs in this version as well is the ability to uploaded as many files as wanted, thus a counter was added to asynchronously deal with different file upload times. Once all the files are uploaded and the links are generated, the module will call the callback with reference to the array of links. This module file can be found as ‘fileUploader.js’ in the attached documents.

Writing modules was learnt about from ‘thenewboston’:
https://youtu.be/9JhvjhZLsEw

Javascript Asynchronous development was aided by ‘The Net Ninja’:
https://www.youtube.com/watch?v=YxWMxJONp7E

Final Diagram of Module



This figure can also be found as ‘s3.png’ in the attached.