**Final Exercise – day 1:**

**Memory File Cache**

* Implement an in memory file cache.

The file cache will be implemented as a module and will expose the following operaiotns :

* + list(path) - Return an array of all the files and directories under the given path in all levels.
  + getFile(path, filename) - Returns the content of the requested file from a memory container, and if it does not exist in memory upload it from disk , store in memory and return it.
  + saveFile(path, filename, content) - saves or update the file content
  + deleteFile(path)
  + reset – clear the memory of the modules

To make the API terse, it has to comply with the following rules:

* + ‘saveFile’ will create the full path if the destination folder doesn’t exists.
  + Only a single copy of each content of different will be kept in memory (same content different files).
  + Delete a leaf folder if it contains no files when ‘deleteFile’ is called.

For simplicity assume:

* All variables are strings.
* No need to validate input types.

Note:

* All operations must be asynchronous – usage of synchronous operations of the fs module is not allowed.
* Use async / await (almost) everywhere in the index.js file – as the main goal of this exercise is to write your entire (or at least most) of the new module that you are writing in async await – like the newer programming paradigms in Node.js.

You have the following repository with tests. All implementations are blank.

<https://github.com/tamarstern/InMemoryFileOPerations>

1. Your first task is to implemented the module, with only asynchronous functions, and make the tests pass.
2. Your second task is to re-write the tests to use generators.
3. To keep the best practices of putting the logic in a different module and wrap the API with promises.

**Echo at Time**

Write a server that will print to the console a message at a given time.

The server should have one model , called message, with 3 fields:

* Time
* Message
* Status

Have a controller that will have only he following methods:

* Post new message
* Get all messages
* Get message by id

The scheduling will be done with a timer of your choice.

To implement the scheduler you can use npm-schedule module, or choose a different scheduler of your choice.

The link to install node-schedule:

<https://www.npmjs.com/package/node-schedule>

Once print is scheduled, change the message status to ‘Done’.

Write all your code with async-await and promises.

Note –

* Cluster support is not required, only schedule of the server printing.