**Unit 8 – Node: Asynchronous approaches**

Callbacks

How promises help us avoid callback hell

Promise.all for parallel promise execution

Introduce async/await syntax.

1. From here on out, we will be performing many async operations in our code.
2. We’ll be communicating with databases and waiting for responses from those databases.
3. Suppose we want to do the following:
   1. Connect to DB
   2. Fetch a markdown file from the file system
   3. Save new article to the database
   4. These tasks are all asynchronous; as engineers, we’ll have to consider how we’ll achieve a sequential completion of these tasks without breaking our app.
4. We can handle this asynchronicity by using callbacks.
   1. A callback is a function we pass into another function.
   2. Async functions can take callback as the last argument.
      1. Using these callbacks, we can catch errors.
      2. We use this to handle success or failure.
      3. The cb in itself is asynchronouse – but it does tell us when the async function has finished executing.
   3. These async functions are passed to libUV which will handle them via their worker threads.
   4. In our callback check, “error” is only passed if an “error” is returned. Otherwise, it will remain undefined.
   5. Fs.readFile will get back some kind of “data” to read, its callback can take “err” and “data.”
   6. This puts us in “callback hell”
   7. When we pass something to a database (mockDb.create())
      1. We will get back what was added to the database.
5. We can also handle this by using promises.
   1. A promise is a placeholder…
   2. It’s an abstraction of non-blocking async execution.
   3. It promises to keep track of the async task with either a success or rejection (error).
   4. Promises are mainly syntactic sugar for adding readability to our async activity.
   5. With promises, we get an immediately returned object within JS, which keeps track of the status of the task.
   6. It’s also responsible for handling callbacks within the success/rejection response.
   7. A promise is an object with 3 states of a promise: pending/rejected/fulfilled.
   8. Chaining promises cleans up our error handling by a lot; we can have a single “catch” at the end of the chain rather than having multiple error-catching callbacks.

function readData(file) {

*return* new Promise(resolve, reject) => {

*//some kind of async function*

*// and have if/else to catch (err, data) within the callback of the async function.*

}

}

* 1. The function passed into .then is what is assigned to “on fulfilled” property of our promise.
  2. .then returns another promise, which is chained to the promise which called it.
  3. .then has two parameters:
     1. onSuccess callback
     2. onReject callback
  4. the .catch method is syntactic sugar for passing in “onReject” parameter only into our “then.”
  5. “Resolve” callback is the function stored in “onfulfilled.”
  6. When fs.readFile is run, the return will be passed automatically into the callback within its parameters.
     1. **Article** is the value…
  7. Remember: stored in the “value” of our promise is the **returned thing.**
     1. Not the evaluated result of running the “resolve” callback.
  8. The value is then stored into the “resolve” callback to pass to the next promise in the chain…
  9. The value returned by the promise created by the handler function is the value that is passed to “then” in the promise chain.
  10. If anything in the chain is rejected, the TOE will skip to the first promise object that has a “onrejected” property, bypassing the other promises.
      1. Updating the status from pending to rejected and
      2. Updating the value to the “err” that was returned
  11. **A PROMISE**
      1. Is a placeholder
      2. Which keeps track of the status of the asynchronous request
      3. Stores the value returned by the request
      4. And stores the functionality needed to process the value returned.
  12. Promise.all
      1. Is a method on promises that takes an array of promises
      2. And this will execute all the promises, in no particular order.
      3. The value will be an array of the returned values from the passed-in promises.
      4. This runs the promises **in parallel**
         1. Takes advantage of the multithreadedness of LibUV.

1. **Async/Await**
   1. “Async” a shortcut for defining a function which returns a promise
   2. And “await” is a keyword for synchronizing our promises.