# Please Await, for I Promise to Rid you of Callback Hell

In which I share a personal journey

from depths of hell to warm sunshine

# Loban Rahman

@LobanRahman

Technology Architect


http://jeeon.co

# Asynchronous Nature of NodeJS

- NodeJS is single-threaded.
- NodeJS avoids “hanging” when waiting for IO by means of asynchronous API calls.
  - Filesystem calls
  - Database calls
  - HTTP calls
  - Lengthy calculations
- Most NodeJS libraries follow a similar convention.

# Callbacks

- Callbacks traditionally have an errorResult as first argument and successResult as second argument.
- Chaining callbacks gets ugly fast.

```
apiUsingCallback1(arg1, function(errResult1, successResult1) {
    if (errResult1) {
        // Handle error
    }
    apiUsingCallback2(successResult1, function(errResult2, successResult2) {
        if (errResult2) {
            // Handle error
        }
        // Continue with successResult2
    }
});
```

# Promises

- A promise is an object that encapsulates a “future” value.
- Promises can either resolve (success) or reject (error).
- Functions attached to either success (using then) or reject (using catch) always return another promise. Hence Promises can be chained.

```
return apiUsingPromise1(arg1)
    .then(successResult1 => {
        return apiUsingPromise2(successResult1);
    })
    .then(successResult2 => {
        // Continue with successResult2
    })
    .catch(errResult => {
        // Handle error
    });
```

# Async / Await

- New keywords in ES7.
- Works with Promises as first-class construct.
- Makes asynchronous code look like synchronous code.

```
try {
    let successResult1 = await apiUsingPromise1(arg1);
    let successResult2 = await apiUsingPromise2(successResult1);
}
catch(errResult) {
    // Handle error
}
```

https://github.com/loban/devcon2017

@LobanRahman


http://jeeon.co

We are Hiring!