Node.js error-handling done right

```
1 + 1 !== 3
```

A guide by Ruben Bridgewater

Why is error-handling hard

1.0. Callbacks

```
'use strict';

const fs = require('fs');

function write() {
  fs.mkdir('./folder');
  fs.writeFile('./folder/foobar.txt', 'Hello World!');
}
// ------
function write(callback) {
  fs.mkdir('./folder', () => {
    fs.writeFile('./folder/foobar.txt', 'Hello World!', callback);
  });
}
// ------
function write(callback) {
  fs.mkdir('./folder', (err, data) => {
    if (data) fs.writeFile('./folder/foobar.txt', 'Hello World!', callback);
    else callback(err);
  });
}
```

1.1. Promises

```
'use strict';

const fs = require('fs');

function write() {
  return fs.mkdir('./folder').then(() => {
    fs.writeFile('./folder/foobar.txt', 'Hello World!');
  }).catch((err) => {
    // Log all potential errors.
    console.error(err);
  });
}
// ------
function doThings() {
  return new Promise((reject, resolve) => {
    callbackApi((err, res) => {
      if (err) reject(err);
      const args = syncAPI(res);
      resolve(args);
    });
  });
}
```

1.2. Express

```
router.get('/:ID', function (req, res, next) {
  database.getData(req.params.userId)
    .then(function (data) {
      if (data.length) {
        // ...
        res.status(200).json(data)
      } else {
        res.status(404).end()
      }
    })
    .catch(() => {
      log.error('db.rest/get : could not get data: ',
        req.params.ID, 'for user:', req.user.userId)
      res.status(500).json({ error: 'Internal server error' })
    })
})
```

1.3. Primitive errors

```
fs.readFile('./file', 'utf8', function (err, general) {
  if (err) {
    reject('Template not found. Error: ' + err)
  } else {
```

```
try {
  throw 'foobar'
} catch (err) {
  console.log(err)
}

try {
  throw new Error('foobar')
} catch (err) {
  console.log(err)
}
```

```
foobar
Error: foobar
    at Object.<anonymous> (/
    at Module._compile (inte
    at Object.Module._extens
    at Module.load (internal
    at tryModuleLoad (intern
    at Function.Module._load
```

Debugging horror


HAPPY PATH

1.4. Nested try / catch

```
async function doThings (input) {
  try {
    validate(input)
    try {
      await db.create(input)
    } catch (error) {
      error.message = `Inner error: ${error.message}`
      if (error instanceof Klass) {
        error.isKlass = true
      }
      throw error
    }
  } catch (error) {
    error.message = `Could not do things: ${error.message}`
    await rollback(input)
    throw error
  }
}
```

1.5. Unhandled rejections

```
3   async function foobar() {
4     throw new Error('foobar')
5   }
6
7   async function baz() {
8     throw new Error('baz')
9   }
10
11  async function doThings() {
12    const a = foobar()
13    const b = baz()
14    try {
15      await a
16      await b
17    } catch (err) {
18      // Ignore all errors
19    }
20  }
21
22  doThings()
```

```
UnhandledPromiseRejectionWarning: Error: baz
/home/ruben/repos/node/node/t.js:8:9)
ngs (/home/ruben/repos/node/node/t.js:13:13)
t.<anonymous> (/home/ruben/repos/node/node/t.j
e._compile (internal/modules/cjs/loader.js:702
t.Module._extensions..js (internal/modules/cjs
e.load (internal/modules/cjs/loader.js:612:32)
duleLoad (internal/modules/cjs/loader.js:551:1
ion.Module._load (internal/modules/cjs/loader.
ion.Module.runMain (internal/modules/cjs/loade
up (internal/bootstrap/node.js:240:19)
UnhandledPromiseRejectionWarning: Unhandled p
hich was not handled with .catch(). (rejection
[DEP0018] DeprecationWarning: Unhandled promi
xit code.
```

Doing it right!

2.0. Error classes

```
'use strict';

class ApplicationError extends Error {
  get name() {
    return this.constructor.name;
  }
}

// Abstraction layer
class DatabaseError extends ApplicationCache {
  get name() {
    return this.constructor.name;
  }
}

// Abstraction layer
class UserFacingError extends ApplicationCache {
  get name() {
    return this.constructor.name;
  }
}
```

2.1. Error classes

```
// Errors used to return errors to the user.
class BadRequestError extends UserFacingError {
  get name() {
    return this.constructor.name;
  }

  get statusCode() {
    return 400;
  }
}

class NotFoundError extends UserFacingError {
  get name() {
    return this.constructor.name;
  }

  get statusCode() {
    return 404;
  }
}
```

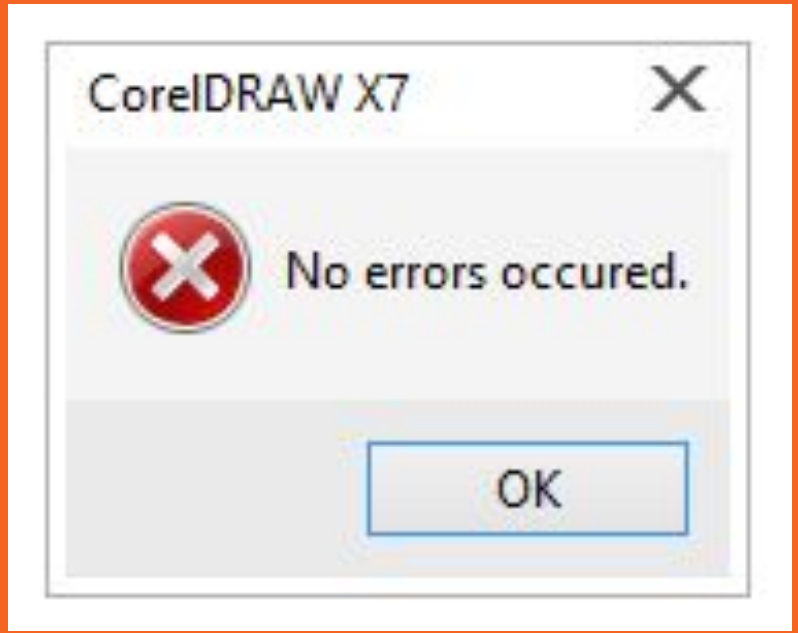
2.2. Error classes

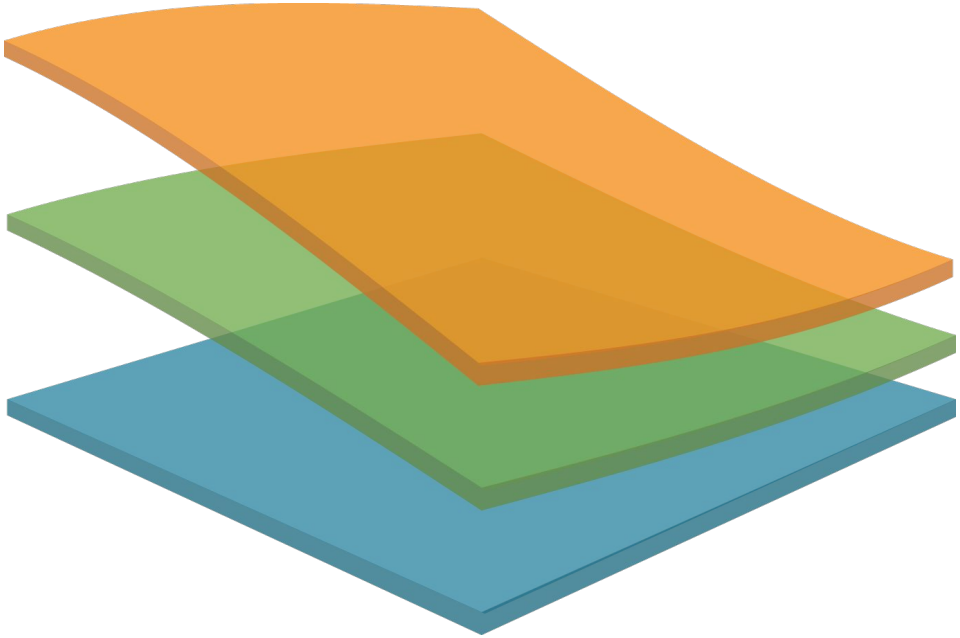
```
class NotFoundError extends UserFacingError {
  constructor(message, options) {
    super(message)
    assert(typeof message === 'string')
    assert(typeof options === 'object')
    assert(options !== null)

    // Attach further information e.g. the username.
    for (const key of Object.keys(options)) {
      this[key] = options.key
    }
  }

  get name() {
    return this.constructor.name;
  }

  get statusCode() {
    return 404;
  }
}
```

2.3. Error classes

- Validate input
- Move in individual module
- Only source of truth
- Contain all information for users and developers.

An abstract error module is easy to use and contains **ALL NECESSARY INFORMATION** in one place.





Error handler layers

Each layer handles a specific part of the application.

Each database should have a layer.

Express / fastify / http should have one.

Outgoing requests ...

3.0. Express

```
// Express
app.get('/:ID', async function (req, res, next) {
  let data
  try {
    data = await database.getData(req.params.userId)
  } catch (err) {
    return next(err)
  }
  if (!data.length) {
    return next(new NotFoundError('Dataset not found'))
  }
  // ...
  res.status(200).json(data)
})

app.use(function (err, req, res, next) {
  if (err instanceof UserFacingError) {
    res.sendStatus(err.statusCode)
    // or
    res.status(err.statusCode).send(err.errorCode)
  } else {
    res.sendStatus(500)
  }

  // Do things...
  logger.error(err, 'Parameters: ', req.params, 'User data: ', req.user)
})
```

3.1. Fastify

```
// Fastify
fastify.get('/:ID', async function (req, reply) {
  const data = await database.getData(req.params.userId)
  if (!data.length) {
    throw new NotFoundError('Dataset not found')
  }
  // ...
  reply.header('Content-Type', 'application/json').code(200)
  return data
})

fastify.setErrorHandler(function (err, req, reply) {
  // Do things...
  logger.error(err, 'Parameters: ', req.params, 'User data: ', req.user)

  if (err instanceof UserFacingError) {
    reply.code(err.statusCode)
    return err.errorCode
  }

  reply.code(500)
  return 'Internal error'
})
```

Databases

- Fallbacks / recoverable errors
- Transparent to the user

3.3. Requests

```
async function send(json, access_token, timesRepeated = 0, delay = 125) {
  validate(json)

  try {
    await request.post({
      uri: facebookConfig.URL.MESSAGES,
      qs: { access_token },
      json
    });
    if (timesRepeated) {
      Logger.warn(`base.facebook.sender: Sending repeated ${timesRepeated} times until success!`);
    }
  } catch (err) {
    const facebookError = err.error || {};

    if (timesRepeated < 10 && (
      err instanceof NetworkError ||
      facebookError.code === 1200 // Temporary send message failure. Please try again later.
    )) {
      await sleep(delay);
      return send(json, access_token, timesRepeated + 1, Math.min(delay + 200, 5000));
    }

    Logger.error('base.facebook.sender:', err);
    if (timesRepeated) {
      Logger.warn(`base.facebook.sender: Sending repeated ${timesRepeated} times!`);
    }
    throw err;
  }
}
```

Debugging utils

- Multiple resolves
- Promise hooks
- Proper logging
- Stack traces

4.0. Multiple resolves

```
process.on('multipleResolves', (type, promise, reason) => {
  console.error(type, promise, reason);
  process.exit(1);
});

async function main() {
  try {
    return await new Promise((resolve, reject) => {
      resolve('First call');
      resolve('Swallowed resolve');
      reject(new Error('Swallowed reject'));
    });
  } catch {
    throw new Error('Failed');
  }
}

main().then(console.log);
// resolve: Promise { 'First call' } 'Swallowed resolve'
// reject: Promise { 'First call' } Error: Swallowed reject
//     at Promise (*)
//     at new Promise (<anonymous>)
//     at main (*)
```

4.1. Stack traces

```
try {
  throw new Error('foo');
} catch (err) {
  console.log(err);
}

const { readFile } = require('fs');

readFile('./inexistent', console.log);
```

```
Error: foo
    at Object.<anonymous> (/home/ruben/repos/node/node/t.js:4:9)
    at Module._compile (internal/modules/cjs/loader.js:707:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:718:10)
    at Module.load (internal/modules/cjs/loader.js:605:32)
    at tryModuleLoad (internal/modules/cjs/loader.js:544:12)
    at Function.Module._load (internal/modules/cjs/loader.js:536:3)
    at Function.Module.runMain (internal/modules/cjs/loader.js:760:12)
    at startup (internal/bootstrap/node.js:303:19)
    at bootstrapNodeJSCore (internal/bootstrap/node.js:872:3)
{ [Error: ENOENT: no such file or directory, open './inexistent']
  errno: -2,
  code: 'ENOENT',
  syscall: 'open',
  path: './inexistent' }
```

5.0. Rules

- ➔ Use error classes specifically set up for the application
- ➔ Implement abstract error handlers
- ➔ Always use async / await
- ➔ Make errors expressive
- ➔ Use promisify if necessary
- ➔ Return proper error statuses and codes
- ➔ Make use of promise hooks

Questions?

Contact

twitter.com/BridgeAR
github.com/BridgeAR
ruben@bridgewater.de