Aw, Snap!

Something went wrong while displaying this webpage.

Learn more

Reload

Error handling: doing it right!

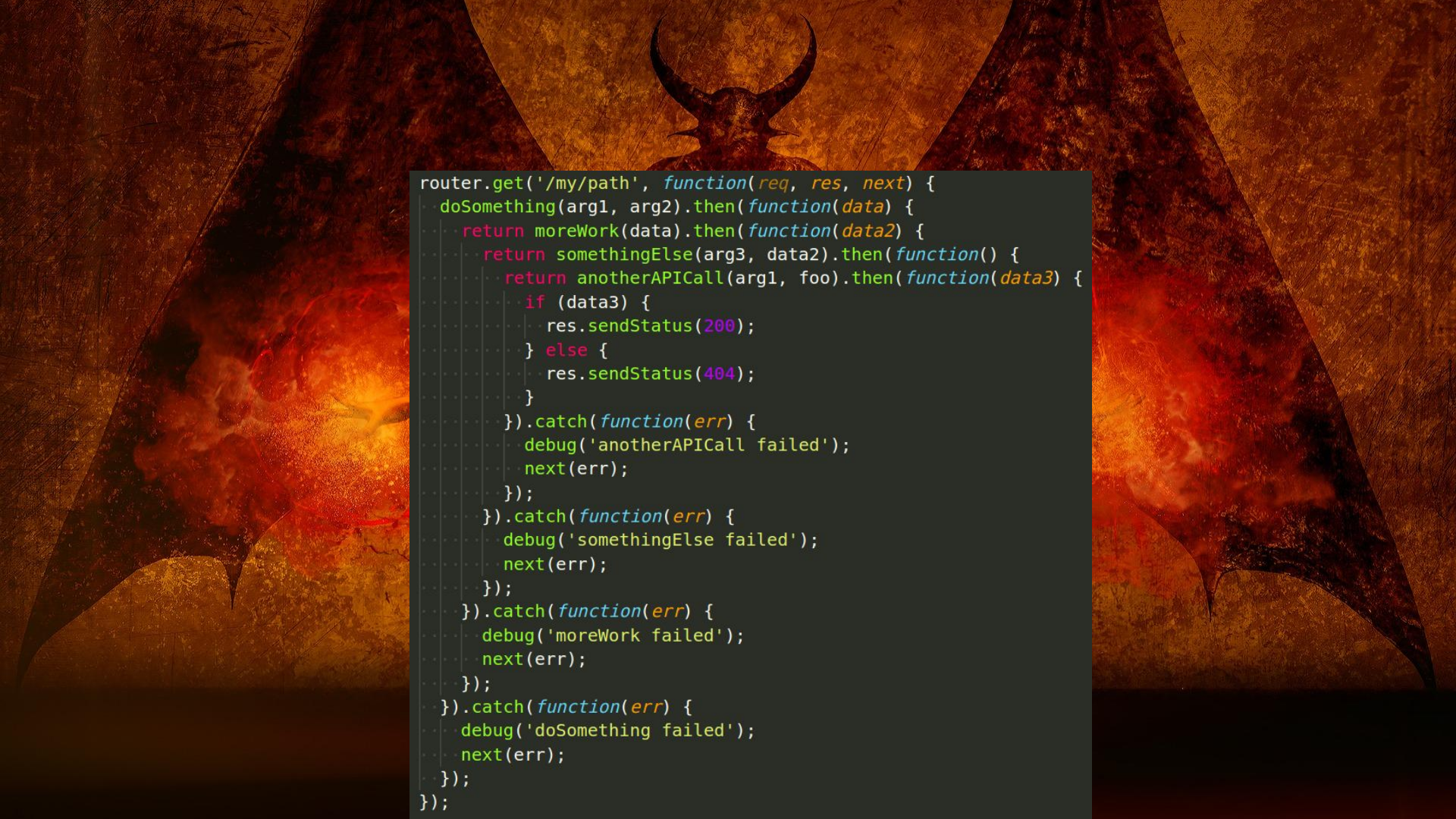
```
1 + 1 !== 3
```

A guide by Ruben Bridgewater

Why is error-handling hard

```
1   'use strict';
2
3   const fs = require('fs');
4
5   function write() {
6     fs.mkdir('./tmp', () => {
7       fs.writeFile('./tmp/foobar.txt', 'ABC');
8     });
9   }
10
11  write();
```

```
'use strict';

const fs = require('fs').promises;

function write() {
  return fs.mkdir('./tmp').then(() => {
    fs.writeFile('./tmp/foobar.txt', 'Hello world!');
  }).catch((err) => {
    // Log all potential errors
    console.error(err);
  });
}

write();
```


router.get('/my/path', function(req, res, next) {
doSomething(arg1, arg2).then(function(data) {
return moreWork(data).then(function(data2) {
return somethingElse(arg3, data2).then(function() {
return anotherAPICall(arg1, foo).then(function(data3) {
if (data3) {
res.sendStatus(200);
} else {
res.sendStatus(404);
}
}).catch(function(err) {
debug('anotherAPICall failed');
next(err);
});
}).catch(function(err) {
debug('somethingElse failed');
next(err);
});
}).catch(function(err) {
debug('moreWork failed');
next(err);
});
}).catch(function(err) {
debug('doSomething failed');
next(err);
});
});

Promisify

```
'use strict';
const assert = require('assert');

function callbackAPI(fn) {
  setTimeout(() => {
    if (Math.round(Math.random())) { // 50%
      fn(new Error('Failed'));
    } else {
      fn(null, 'Data');
    }
  }, 1);
}

function syncAPIWithSideEffects(data) {
  if (Math.round(Math.random())) // 50%
    throw new Error('Failed');
  // Oops
  return data.length;
}
```

```
function promisifyCallback(input) {
  assert(input);
  return new Promise((resolve, reject) => {
    assert(input === true);
    callbackAPI((err, res) => {
      if (err) reject(err);
      const args = syncAPIWithSideEffects(res);
      resolve(args);
    });
  });
}
```

REAL

LIFE

```
'use strict';

function readTemplate() {
  return new Promise((resolve, reject) => {
    databaseGet('query', function (err, data) {
      if (err) {
        reject('Template not found. Error: ' + err);
      } else {
        resolve(data);
      }
    });
  });
}

readTemplate();
```

```
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' })
    })
})
```

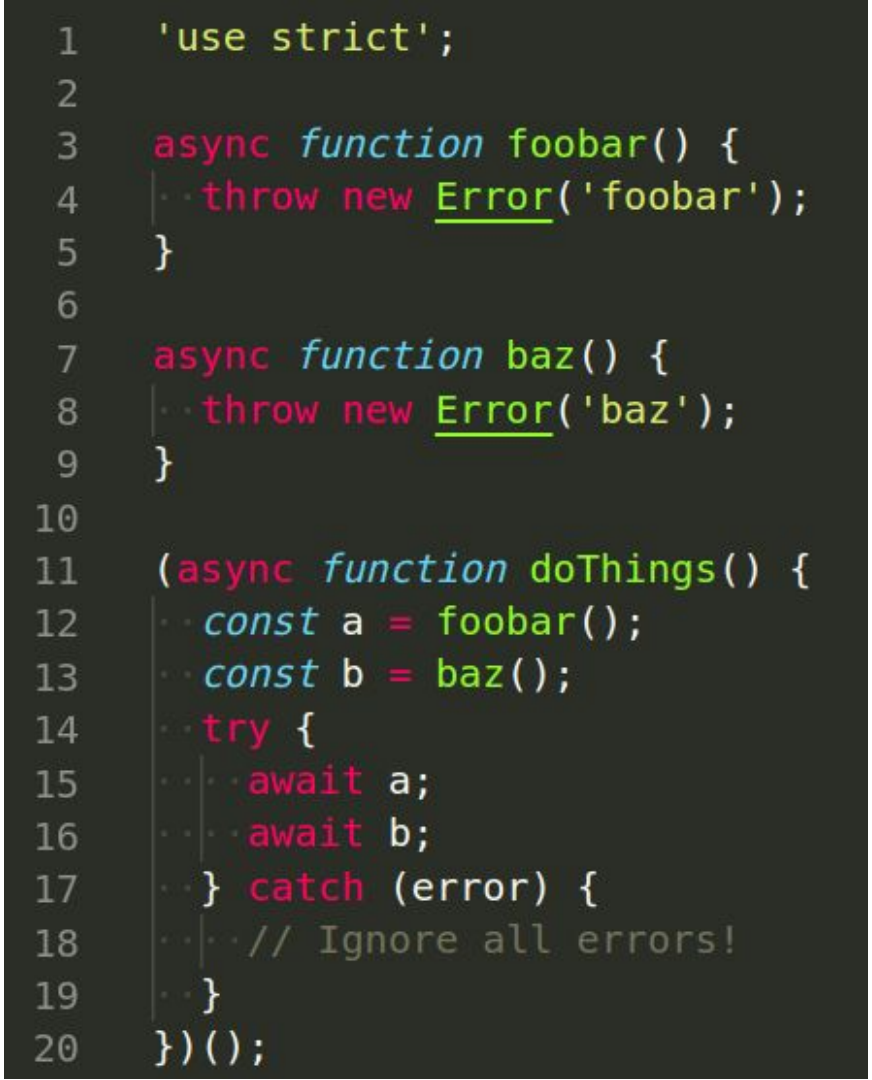
Debugging horror


HAPPY PATH

```
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
  }
}
```

Unhandled rejections

```
(node:10938) UnhandledPromiseRejectionWarning: Error: baz
    at baz (/home/ruben/repos/node/holyjs/unhandled1.js:8:9)
    at doThings (/home/ruben/repos/node/holyjs/unhandled1.js:13:13)
    at Object.<anonymous> (/home/ruben/repos/node/holyjs/unhandled1.js:20:3)
    at Module._compile (internal/modules/cjs/loader.js:774:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:785:10)
    at Module.load (internal/modules/cjs/loader.js:641:32)
    at Function.Module._load (internal/modules/cjs/loader.js:556:12)
    at Function.Module.runMain (internal/modules/cjs/loader.js:837:10)
    at internal/main/run_main_module.js:17:11
```



```
'use strict';

async function foobar() {
  throw new Error('foobar');
}

async function baz() {
  throw new Error('baz');
}

(async function doThings() {
  const a = foobar();
  const b = baz();
  try {
    await a;
    await b;
  } catch (error) {
    // Ignore all errors!
  }
})();
```


'use strict';
async function foobar() {
throw new Error('foobar');
}
async function doThings() {
try {
return foobar();
} catch (error) {
// Ignore all errors!
}
}
doThings();


Doing
it right!

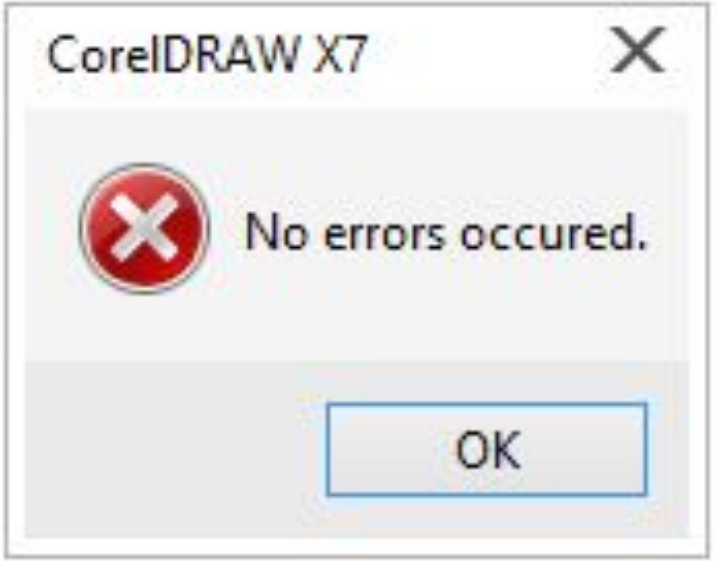
```
'use strict';

// Base error classes to extend from

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

class DatabaseError extends ApplicationError {}

class UserFacingError extends ApplicationError {}

module.exports = {
  ApplicationError,
  DatabaseError,
  UserFacingError
};
```

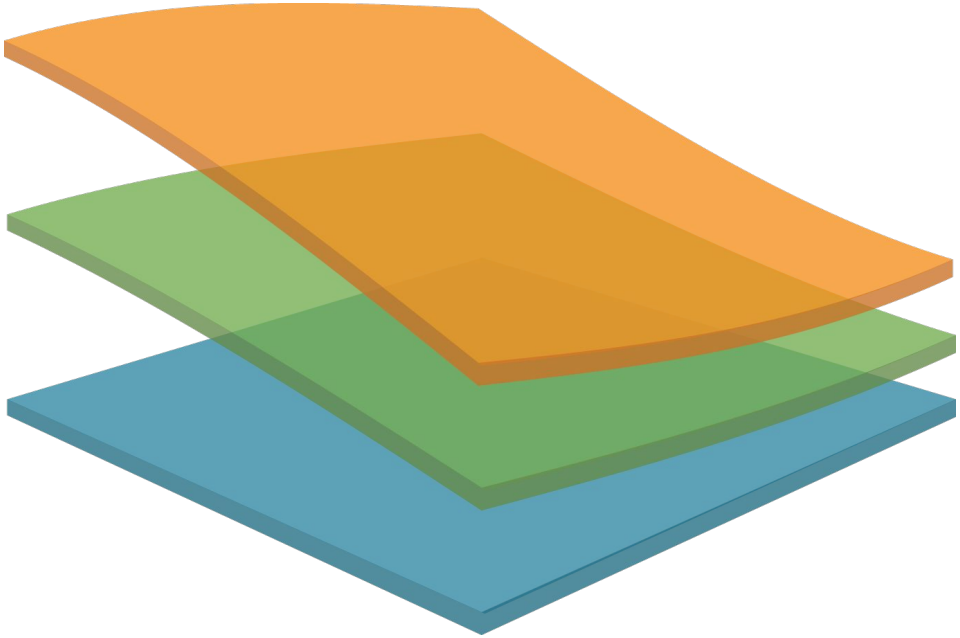
```
const { UserFacingError } = require('./baseClass');

class BadRequestError extends UserFacingError {
  constructor(message, options = {}) {
    super(message);
    // Attach relevant information to the error instance
    // (e.g., the username).
    for (const [key, value] of Object.entries(options)) {
      this[key] = value;
    }
  }
  get statusCode() {
    return 400;
  }
}

class NotFoundError extends UserFacingError {
  // ... constructor ...
  get statusCode() {
    return 404;
  }
}
```

Error classes

- Create application specific base class
- 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 ...

```
// 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)
})
```

```
// 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

```
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

- Proper logging
- Stack traces
- Unhandled rejection flag

```
1  'use strict';
2
3  const database = require('database');
4
5  function promiseGet (query) {
6    return new Promise((resolve, reject) => {
7      // Do the actual database call
8      database.get(query, (err, result) => {
9        if (err) {
10         reject(err);
11       } else {
12         resolve(result);
13       }
14     });
15   })
16 }
17
18 (async function main() {
19   await promiseGet('foobar');
20 })();
```

util.promisify

```
1   'use strict';
2
3   const database = require('database');
4
5   const { promisify } = require('util');
6   const promiseGet = promisify(database.get).bind(database);
7
8   (async function main() {
9     await promiseGet('foobar');
10  })();
```

--unhandled-rejections

```
ruben@BridgeAR-T450s:~/repos/node/holyjs$ node --unhandled-rejections=strict ./unhandled/unhandled1.js

/home/ruben/repos/node/holyjs/unhandled/unhandled1.js:8
  throw new Error('baz');
        ^

Error: baz
    at baz (/home/ruben/repos/node/holyjs/unhandled/unhandled1.js:8:9)
    at doThings (/home/ruben/repos/node/holyjs/unhandled/unhandled1.js:13:13)
    at Object.<anonymous> (/home/ruben/repos/node/holyjs/unhandled/unhandled1.js:20:3)
    at Module._compile (internal/modules/cjs/loader.js:774:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:785:10)
    at Module.load (internal/modules/cjs/loader.js:641:32)
    at Function.Module._load (internal/modules/cjs/loader.js:556:12)
    at Function.Module.runMain (internal/modules/cjs/loader.js:837:10)
    at internal/main/run_main_module.js:17:11
```

```
async function one() {
  await two();
}

async function two() {
  await 'awaitable';
  throw new Error('Woops');
}

(async function test() {
  try {
    await one();
  } catch (error) {
    console.log(error);
  }
})();
```

```
Error: Woops
    at two (/home/ruben/repos/node/holyjs/stackTraces.js:7:9)
    at async one (/home/ruben/repos/node/holyjs/stackTraces.js:2:3)
    at async test (/home/ruben/repos/node/holyjs/stackTraces.js:12:5)
```

Summary / 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


Steve McConnell's, Code Complete:
"There are about 15-50 errors per 1000 lines of code"
¡Muchas Gracias!
Twitter: @BridgeAR
Email: ruben@bridgewater.de