



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**



Callbacks

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



Callbacks

```
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();
```



Callbacks

```
1  'use strict';
2
3  const fs = require('fs');
4
5  function write(callback) {
6    fs.mkdir('./tmp', (err, data) => {
7      if (data) fs.writeFile('./tmp/foobar.txt', 'FOOBAR', callback);
8      else callback(err);
9    });
10 }
11
12 write(console.log);
```



Promises

```
1 'use strict';
2
3 const fs = require('fs').promises;
4
5 function write() {
6   return fs.mkdir('./tmp').then(() => {
7     fs.writeFile('./tmp/foobar.txt', 'Hello world!');
8   }).catch((err) => {
9     // Log all potential errors
10    console.error(err);
11  });
12 }
13
14 write();
```





Promisify

```
20
21  function promisifyCallback(input) {
22    · assert(input);
23    · return new Promise((resolve, reject) => {
24      · · assert(input === true);
25      · · callbackAPI((err, res) => {
26        · · · if (err) reject(err);
27        · · · const args = syncAPIWithSideEffects(res);
28        · · · resolve(args);
29      · · });
30    · });
31  }
```



Promisify

```
1  'use strict';
2  const assert = require('assert');
3
4  function callbackAPI(fn) {
5    ··setTimeout(() => {
6    ··  ··if (Math.round(Math.random())) { // 50%
7    ··  ··  ··fn(new Error('Failed'));
8    ··  ··  } else {
9    ··  ··  ··fn(null, 'Data');
10   ··  }
11   ··}, 1);
12 }
13
14 function syncAPIWithSideEffects(data) {
15   ··if (Math.round(Math.random())) // 50%
16   ··  ··throw new Error('Failed');
17   ··// Oops
18   ··return data.length;
19 }
20
```



Promisify

```
1  'use strict';
2  const assert = require('assert');
3
4  function callbackAPI(fn) {
5    ··setTimeout(() => {
6    ··  ··if (Math.round(Math.random())) { // 50%
7    ··  ··  ··fn(new Error('Failed'));
8    ··  ··  ··} else {
9    ··  ··  ··fn(null, 'Data');
10   ··  ··}
11   ··}, 1);
12 }
13
14 function syncAPIWithSideEffects(data) {
15   ··if (Math.round(Math.random())) // 50%
16   ··  ··throw new Error('Failed');
17   ··// Oops
18   ··return data.length;
19 }
20
```

```
20
21 function promisifyCallback(input) {
22   ··assert(input);
23   ··return new Promise((resolve, reject) => {
24   ··  ··assert(input === true);
25   ··  ··callbackAPI((err, res) => {
26   ··  ··  ··if (err) reject(err);
27   ··  ··  ··const args = syncAPIWithSideEffects(res);
28   ··  ··  ··resolve(args);
29   ··  ··});
30   ··});
31 }
```



REAL

LIFE



Primitive errors

```
1  'use strict';
2
3  function readTemplate() {
4    return new Promise((resolve, reject) => {
5      databaseGet('query', function (err, data) {
6        if (err) {
7          reject('Template not found. Error: ' + err);
8        } else {
9          resolve(data);
10       }
11     });
12   });
13 }
14
15 readTemplate();
```



Primitive errors

```
1  'use strict';
2
3  try {
4    · throw ['foobar', 'baz'];
5  } catch (error) {
6    · console.log(error);
7  }
8
9  try {
10   · throw new Error('foo');
11  } catch (error) {
12   · console.log(error);
13  }
```



Primitive errors

```
[ 'foobar', 'baz' ]  
Error: foo  
  at Object.<anonymous> (/home/ruben/repos/node/  
  at Module._compile (internal/modules/cjs/load  
  at Object.Module._extensions..js (internal/mod  
  at Module.load (internal/modules/cjs/loader.js  
  at Function.Module._load (internal/modules/cjs  
  at Function.Module.runMain (internal/modules/c  
  at internal/main/run main module.js:17:11
```




Express

```
1  router.get('/:ID', function (req, res, next) {
2    database.getData(req.params.userId)
3    .then(function (data) {
4      if (data.length) {
5        // ...
6        res.status(200).json(data)
7      } else {
8        res.status(404).end()
9      }
10   })
11   .catch(() => {
12     log.error('db.rest/get : could not get data: ',
13     req.params.ID, 'for user:', req.user.userId)
14     res.status(500).json({ error: 'Internal server error' })
15   })
16 })
```


Debugging horror





HAPPY PATH



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  
  }  
}
```




Rejections

```
1  'use strict';
2
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 (error) {
18     // Ignore all errors!
19   }
20 })();
```

```
1  'use strict';
2
3  async function foobar() {
4    throw new Error('foobar');
5  }
6
7  async function doThings() {
8    try {
9      return foobar();
10   } catch (error) {
11     // Ignore all errors!
12   }
13 }
14
15 doThings();
```



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
```



Faulty tests

```
1  const assert = require('assert');
2
3  function api(input) {
4    // Do things...
5    if (Math.round(Math.random())) {
6      return;
7    }
8    throw new Error('Wrong input');
9  }
10
11 it ('test case', () => {
12   // Do things...
13   try {
14     api('abc');
15   } catch (err) {
16     assert.strictEqual(err.message, 'Wrong input');
17   }
18 });
```

Doing
it right!





Error classes

```
1  'use strict';
2
3  // Base error classes to extend from
4
5  class ApplicationError extends Error {
6    · get name() {
7    ·   · return this.constructor.name;
8    · }
9  }
10
11 class DatabaseError extends ApplicationError {}
12
13 class UserFacingError extends ApplicationError {}
14
15 module.exports = {
16   · ApplicationError,
17   · DatabaseError,
18   · UserFacingError
19 };
```




Error classes

```
1  const { UserFacingError } = require('./baseClass');
2
3  class BadRequestError extends UserFacingError {
4    get statusCode() {
5      return 400;
6    }
7  }
8
9  class NotFoundError extends UserFacingError {
10   get statusCode() {
11     return 404;
12   }
13 }
14
15 module.exports = {
16   BadRequestError,
17   NotFoundError
18 };
```



Error classes

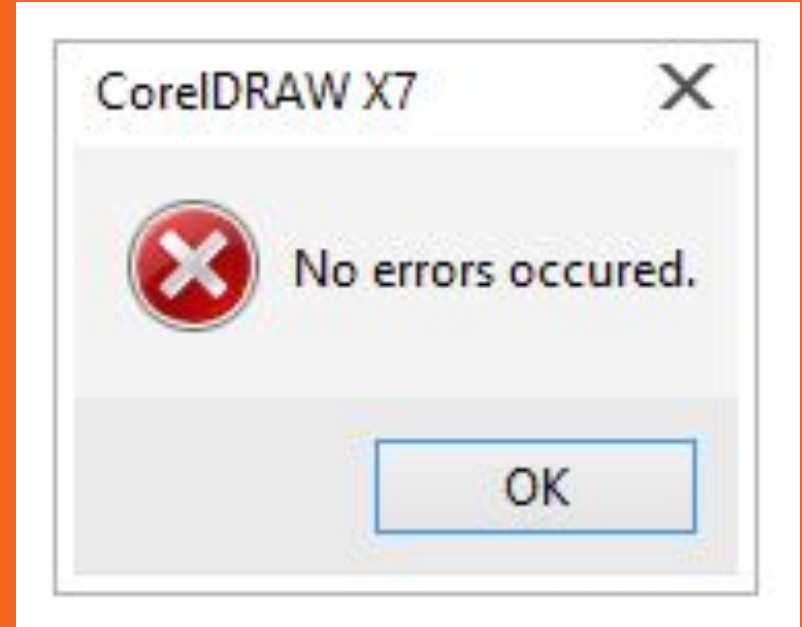
```
1  const assert = require('assert').strict;
2
3  const { UserFacingError } = require('./baseClass');
4
5  class BadRequestError extends UserFacingError {
6    constructor(message, options) {
7      super(message);
8      assert(typeof message === 'string');
9      assert(typeof options === 'object');
10     assert(options !== null);
11
12     // Attach relevant information to the error instance
13     // (e.g., the username).
14     for (const [key, value] of Object.entries(options)) {
15       this[key] = value;
16     }
17   }
18
19   get statusCode() {
20     return 400;
21   }
22 }
```

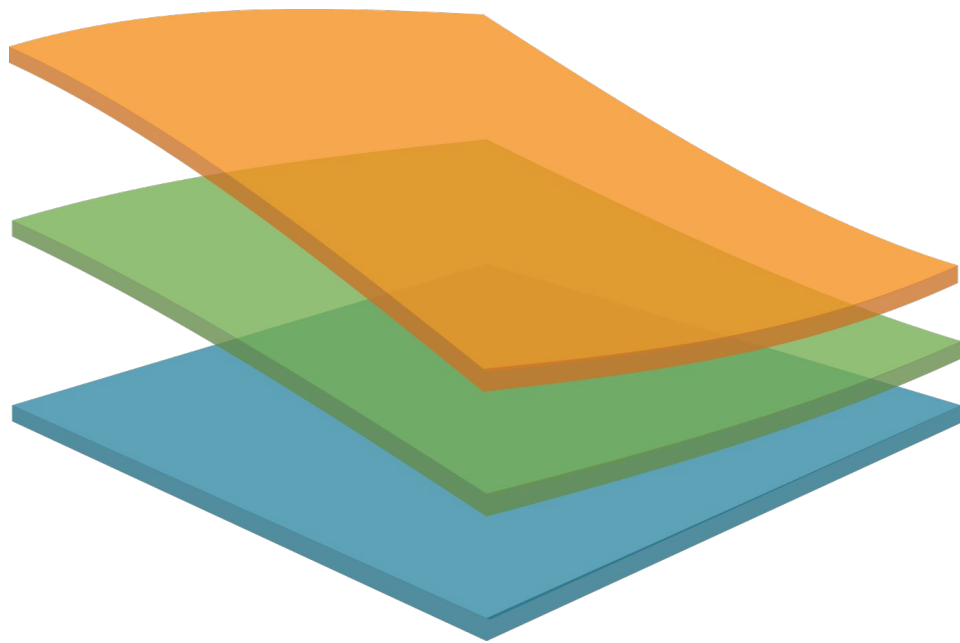


Error classes

- 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

```
// 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
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





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
- Unhandled rejection flag





util.promisify

```
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 })();
```



Multiple resolves

```
1 process.on('multipleResolves', (type, promise, reason) => {
2   · console.warn(type, promise, reason);
3 });
4
5 async function main() {
6   · try {
7     · · return await new Promise((resolve, reject) => {
8     · · · resolve('First call');
9     · · · resolve('Swallowed resolve');
10    · · · reject(new Error('Swallowed reject'));
11    · · });
12   · } catch {
13   · · throw new Error('Failed');
14   · }
15 }
16
17 main().then(console.log);
```



Multiple resolves

```
resolve Promise { 'First call' } Swallowed resolve
reject Promise { 'First call' } Error: Swallowed reject
  at /home/ruben/repos/node/holyjs/multipleResolves1.js:10:14
  at new Promise (<anonymous>)
  at main (/home/ruben/repos/node/holyjs/multipleResolves1.js:7:18)
  at Object.<anonymous> (/home/ruben/repos/node/holyjs/multipleResolves1.js:17:1)
  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
First call
```



--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 Stack traces

```
1  async function one() {
2    await two();
3  }
4
5  async function two() {
6    await 'awaitable';
7    throw new Error('Woops');
8  }
9
10 (async function test() {
11   try {
12     await one();
13   } catch (error) {
14     console.log(error);
15   }
16 })();
```




```
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
- Make use of promise hooks

Steve McConnell's, Code Complete:

“There are about 15-50 errors per 1000 lines of code”

Thank you!



Twitter: @BridgeAR

Email: ruben@bridgewater.de