Handling Errors

Why

errors can happen as a result of bugs in our code

issues in the running environment

our MongoDB server may shut down

remote HTTP service we call may go down.

What to DO?

count for these unexpected errors

log them and return a proper error to the client

Use the Express error middleware to catch any unhandled exceptions in the “request processing pipeline”.

Register the error middleware after all the existing routes:

Express Error Middleware

```
app.use(function(err, req, res, next) {
 // Log the exception
 //and return a friendly error to the client.
       res.status(500).send('Something failed');
});
```

wrap your route handler code in a try/ catch block and call next().

```
try {
 const genres = await Genre.find();
}
catch(ex) {
 next(ex); // call next middleware
});
```

express-async-errors module

Adding a try/catch block to every route handler is repetitive

This module will monkey-patch your route handlers at runtime

To log errors use winston

Winston can log errors in multiple transports.

A transport is where your log is stored.

- Console,
- File and Http.
- MongoDB
- CouchDB
- Redis
- Loggly

Caution

The error middleware in Express only catches exceptions in the request processing pipeline

application startup (eg connecting to MongoDB) will be invisible to Express.

Use **process.on('uncaughtException')** to catch unhandled exceptions

process.on('unhandledRejection') to catch rejected promises

Best Practice

event handlers you pass to process.on(), you should log the exception and exit the process

It’s better to restart the process in a clean state

use a process manager to automatically restart a Node process

Example Code

https://1drv.ms/f/s!AtGKdbMmNBGd01H3gbOBASQmUSLM