L03 - POST requests in Node.js

**Middleware Functions**

Middleware functions are functions that have access to the request object(req), the response object(res), and the next middleware function( next()) in the application's request-response cycle. They are responsible for executing code, modifying the request and response objects, potentially terminating the request-response cycle, and calling the next middle function in the stack. For example, if a middleware function does not terminate the request-response cycle, it must call next() to pass control to the next middleware function otherwise it will be left hanging. In summary, middleware functions are responsible for managing the access to the 'req', 'res', and 'next()' within the express.js framework.

Here are two of how to keep up good practices while working with middleware functions:

1-Follow the principle of Encapsulation. Each middleware function should have a specific and well-defined purpose. Avoid creating a single middleware function that handles multiple tasks. Smaller and focused ones are easier to understand, maintain, and more important, reuse.w

Example:

function deleteBook(req, res, next) {

const id = req.params.id;

const updatedBooks = books.filter(book => book.id !== id)

//filters the book's array returning an array without the book that matches the id chosen.

next(); // Calling the next middleware to end the middleware or router

}

app.use('/api/books/:id', deleteBook);

// Applying the middleware function into the express framework

2- Handle Errors. Middleware functions can also be used for error handling. It takes four arguments(err,req,res,next) to capture and handle errors that occur in another middleware or routes.  
  
Example:  
 function deleteBook(req, res, next) {

const id = req.params.id;

//Checking if id the book id exists

if(!books.some(book=>book.id== id){

const error= new Error(‘book not found’);

error.status=404; //defining the error status

return next(error); // returning the error and ending the res-req-cycle.

}

const updatedBooks = books.filter(book => book.id !== id)

//filters the book's array returning an array without the book that matches the id chosen.  
res.send(‘book deleted succesfuly’);

next(); // Calling the next middleware to end the middleware or router

}  
 app.use('/api/books/:id', deleteBook);

// Applying the middleware function into the express framework

As we apply these two simple good practice technology tips, our code becomes better organized, the errors are easy to find, and the major part of our application can be reused.  
  
Writing middleware functions for use in Express app: <https://expressjs.com/en/guide/writing-middleware.html>  
  
Error handling: https://expressjs.com/en/guide/error-handling.html