Understanding Response Interceptors

In this document, we will try to understand the request-response cycle and through it, the concept and working of response-interception middleware.

Middleware:

In express.js, middleware are those functions that have access to the request (**req**) object, the response (**res**) object and the **next()** function in the application's request-response cycle. The **next()** function belong's to express.js' router. When it is invoked, it executes the middleware succeeding the current middleware. [1]

A given middleware can do the following:

- It can execute any code. [1]
- Modify the **req** and **res** objects. [1]
- End the request-response cycle. [1]
- Call the next middleware in the stack. [1]

If the current middleware wants to continue the request-response cycle, then it must pass the request that it has to the next middleware using the **next()** function. If **next()** isn't called, then the request will hang where it last was. [1]

The request-response cycle is ended when a middleware sends a response back to the client. [3] In other words, if we put a middleware that sends no response (**A**) after a middleware that does send a response (**B**) in the middleware stack, then **A** will never be called, since **B** will terminate the request-response cycle by sending a response. [1] It is important to note that after the request-response cycle is ended, **another response cannot be sent**. [5]

The **app.use()** function is used to load middleware functions. [1]

Note: Middleware functions are executed in the order in which they are loaded. [1]

Let us consider a simple express application that has a simple **GET** handler for a route "/" and sends a response "hello world". We add a middleware called **requestLogger** that adds the current Date to the request object. It is possible to get the date in the **GET** handler. This example shows that the request can be successively modified by each middleware in the middleware stack. [1]

Continuing the example, we remove the **requestLogger** middleware and instead import the **cookie-parser** middleware. This is now the first middleware in the middleware stack. We also create an asynchronous function middleware, **validateCookies** to validate cookies. This middleware returns a promise. This is the 2nd middleware in the middleware stack. Below **validateCookies** is an error handler. This is the 3rd and final middleware in the middleware stack. If **validateCookies** returns a rejected promise, then this will be caught by the error handler. [1]

A Short Note on the Behaviour of next()
If next() is called without any arguments, then the next middleware in the stack gets called. However, if it is called with any argument except the strings **route** and **router**, then it will regard that argument as an error and will skip any remaining non-error handling routing and middleware functions. [1]

Having explained the concept of the request-response cycle, let us introduce the concept of the response interception middleware. **A response interceptor middleware is one that allows a request to pass through it and then intercepts the response, modifies it / logs it and then sends it back to the client without violating the request-response cycle.** [6][7]. We have 2 middlewares that deal with response: One is called getResponseTime() and the other is a response interceptor called responseAnalyzer().

getResponseTime():
This is the 2nd middleware to receive the request. However, it doesn't do anything to the request and the request just passes to the next middleware through the next() function. This middleware uses a listener called **res.on**()**.** This is the response object being attached to the **on** listener and subscribing to the **finish** event. The **finish** event is emitted when the request has been sent from the server. [4] In the line previous to that of the **res.on** listener, the **hrtime** timer is started. When the **finish** event occurs, the following happens:

- The **hrtime** timer is ended.
- The time recorded is converted into milliseconds and formatted for display.
- The **apiName** function is used to get the name of the api from the route.
- The response time and api name are passed to the **createApiWarning** function to determine if the API call takes too long, and in that case, to create a warning.

This middleware records the wrong response time, i.e., the time between the arrival of a request and it being sent from the server. So it will be replaced by the response-time middleware of express.js that measures the correct response time, i.e., the time between the arrival of a request and the writing of the response headers of the client. [8]

responseAnalyzer():
When the request arrives at **getResponseTime()**, then it just passes through to the middleware after it in the middleware stack through the next() function. This is because the request isn't being used over here at all. Had the request been intercepted and the response sent, then the request-response cycle would end and the request would never be able to reach the middlewares that are after it in the middleware stack. Thus, this middleware intercepts only the response body when the response comes back from the server. It does this by overriding the **res.json** method and saving it in a variable called **oldJson** to preserve its behaviour. [2] This is mandatory. If this were not done, then the modified response body being sent would be a new response. This would violate the request-response cycle and cause an error. [5] The **res.json** function is then overridden to intercept the data in the response **(in our case, to get the response body)**. It checks whether the data is empty, an error or non-error data and based on this creates an **info** log entry. After this, in all the 3 cases, the **res.json** function is restored from the **oldJson** variable where it had been saved and it sends the intercepted response to the client.

REFERENCES

[1]https://expressjs.com/en/guide/writing-middleware.html#writing-middleware-for-use-in-express-apps

[2]https://www.geeksforgeeks.org/how-to-intercept-response-send-response-json-in-express-js/

[3]https://medium.com/@navneetskahlon/express-request-response-cycle-and-middleware-with-code-examples-5b01c81b3322

[4] https://nodejs.org/api/http.html#event-finish

[5]https://betterstack.com/community/questions/error-cant-sent-headers-after-they-are-sent-to-client/

[6] https://brianchildress.co/calculating-node-api-response-time/

[7] https://www.sheshbabu.com/posts/measuring-response-times-of-express-route-handlers/

[8] https://expressjs.com/id/resources/middleware/response-time.html