Express-Error Handling Middleware

Error Handling Middleware in Express

In Express, error handling middleware is used to catch and handle errors that occur during the execution of a request. This middleware is typically placed after other routes and middleware to prevent the application from crashing when an error occurs.

Creating Error Handling Middleware

To create an error handling middleware in Express, you can use the following syntax:

```
 app.use((err, req, res, next) => {
  // Handle the error here
});
```

The `err` parameter is an instance of the `Error` class or a custom error object. The `req`, `res`, and `next` parameters are instances of the `Request`, `Response`, and `NextFunction` classes respectively.

Example: Basic Error Handling Middleware

Here's an example of basic error handling middleware that logs errors to the console and returns a generic error message to the client:

```
 app.use((err, req, res, next) => {
  console.error(err);
  res.status(500).send({ message: 'Internal Server Error' });
});
```

In this example, when an error occurs, the middleware logs the error to the console and sends a 500 Internal Server Error response to the client with a generic error message.

Example: Customized Error Handling Middleware

You can also create customized error handling middleware that handles specific types of errors. For instance, you might want to handle authentication-related errors differently:

```
 const { AuthenticationError } = require('jsonwebtoken');

app.use((err, req, res, next) => {
  if (err instanceof AuthenticationError) {
    res.status(401).send({ message: 'Unauthorized' });
  } else {
    console.error(err);
    res.status(500).send({ message: 'Internal Server Error' });
  }
});
```

In this example, the middleware checks if the error is an instance of `AuthenticationError` and returns a 401 Unauthorized response to the client with a customized error message. Otherwise, it logs the error to the console and sends a generic 500 Internal Server Error response.

Best Practices

- Place error handling middleware after other routes and middleware to catch any unhandled errors.
- Log errors to the console for debugging purposes.
- Return a standardized error response with a meaningful status code and message.
- Use customized error handling middleware to handle specific types of errors.

Curated by Brajesh Kumar