Middlewares

HyperionDev

Muhammad Zahir Junejo

Lecture - Housekeeping

- ❏ The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
  - ❏ Please review Code of Conduct (in Student Undertaking Agreement) if unsure
- ❏ No question is daft or silly - **ask them!**
- ❏ Q&A session at the end of the lesson, should you wish to ask any follow-up questions.
- ❏ Should you have any questions after the lecture, please schedule a mentor session.
- ❏ For all non-academic questions, please submit a query: www.hyperiondev.com/support

Lecture Objectives

1. Introduction to Middlewares.
2. Benefits of Middlewares.
3. Anatomy of Middlewares.
4. Execution flow of Middlewares.

Introduction to Middleware

- ❑ Middleware are functions that can be added to the request-response cycle in Express.js.
- ❑ Customize and enhance the functionality of your Express.js application.
- ❑ Think of them as layers of processing for HTTP requests.

Why Use Middleware?

1. Enhance Functionality

❑ Middleware allows you to enhance the functionality of your Express.js application by injecting custom logic into the request-response cycle.

2. Code Organization

❑ Middleware promotes clean and organized code. By separating different concerns into modular middleware functions, your application becomes more maintainable.

3. Reusability

❑ Middleware functions can be reused across different routes and even in multiple Express.js applications. This reusability is a significant advantage, saving development time and effort.

Anatomy of a Middleware Function

- ❏ Middleware is a function with three parameters: (req, res, next).
- ❏ req: Request object containing client request data.
- ❏ res: Response object for sending a response to the client.
- ❏ next: Callback function to pass control to the next middleware.

```
const logMiddleware = (req, res, next) => {
  console.log(`[${new Date().toISOString()}] ${req.method} ${req.url}`);
  next(); // Move on to the next middleware or route handler
};
```

Middleware Execution Flow

- ❑ Order matters: Middlewares are executed in the order they are defined.
- ❑ next(): Move to the next middleware in the chain.
- ❑ Important to call next() to avoid request/response hanging.

```
app.get("/todo", logMiddleware, logMiddleware2, async (req, res) => {
  try {
    let todos = await todoModel.find();
    return res.status(200).json({ data: todos });
  } catch (err) {
    return res.status(502).json({ error: err });
  }
});
```

Express.js Middleware Types

1. Application-Level Middleware:

- ❑ These middleware functions are applied globally to your entire Express.js application using app.use(logMiddleWare).
- ❑ They are executed for every incoming request, regardless of the route.
- ❑ They are registered before defining routes, usually at the top of your code.

2. Route-Specific Middleware:

- ❑ Express.js allows you to apply middleware functions specifically to certain routes or groups of routes.
- ❑ These middlewares are executed in the order they are defined in the code.

Express.js Middleware Order

- ❑ Middleware functions are executed in the order they are defined. This order is crucial because it affects the sequence of processing for each request.
- ❑ When a request comes in, Express.js starts executing application-level middleware first.
- ❑ Once all application-level middleware functions are executed, Express.js proceeds to execute any route-specific middleware registered for the requested route.
- ❑ Finally, the route handler for the specific route is executed.

Hyperiondev

Middleware Use Cases

❑ Logging Middleware: Capture request details.
❑ Authentication Middleware: Secure routes.
❑ Error Handling Middleware: Handle errors gracefully.
❑ CORS Middleware: Manage cross-origin resource sharing.

References

- ❑ https://expressjs.com/en/guide/writing-middleware.html
- ❑ https://expressjs.com/en/guide/using-middleware.html

Questions and Answers

Thank You!