Express.js Middlewares

In Express.js, middleware are functions that have access to the req (request) object, the res (response) object, and the next() function in their signature.

Key Concepts:

- **Purpose:** Middleware functions are designed to perform actions before and after route handlers are executed.
- **Flexibility:** They can be used for a wide range of tasks, including:
  - **Logging:** Log incoming requests and outgoing responses.[1]
  - **Authentication:** Verify user credentials and authorize access to specific routes.[2]
  - **Authorization:** Check if the user has the necessary permissions to access a resource.[3]
  - **Parsing Request Bodies:** Parse incoming request bodies (e.g., JSON, URL-encoded data).[4]
  - **Static File Serving:** Serve static files like HTML, CSS, and JavaScript.[5]
  - **Error Handling:** Handle errors that occur during request processing.[6]
- **next() Function:**
  - The next() function is crucial in middleware.
  - It signals to Express that the middleware has finished its processing and that the request should proceed to the next middleware or the route handler.[7]
  - If next() is not called, the request will be stuck in the middleware.

Example:

JavaScript

```
const express = require('express');
const app = express();
// Define a logging middleware
const logger = (req, res, next) => {
  console.log(`${req.method} ${req.url}`);
  next();
};
// Apply the logger middleware to all routes
app.use(logger);
// Define a route handler
app.get('/', (req, res) => {
```

```
  res.send('Hello from Express.js!');
});
app.listen(3000, () => {
  console.log('Server listening on port 3000');
});
```

In this example:

- The logger middleware logs the request method and URL to the console.
- app.use(logger); applies the logger middleware to all routes in the application.

Built-in Middleware:

Express.js provides several built-in middleware functions:[8]

- express.json(): Parses incoming JSON request bodies.
- express.urlencoded(): Parses incoming URL-encoded request bodies.
- express.static(): Serves static files from a specified directory.

Benefits of Using Middleware:

- **Improved Code Organization:** Separates concerns and makes your code more modular.[9]
- **Enhanced Reusability:** Middleware functions can be reused across different routes and even in different applications.[10]
- **Improved Security:** Middleware can be used to implement authentication, authorization, and input validation.[11]
- **Better Performance:** Middleware can be used to optimize performance by caching data or compressing responses.[12]

By effectively using middleware, you can create more robust, secure, and maintainable Express.js applications.

In **Express.js**, middleware functions are functions that have access to the request (req), response (res), and the next middleware function in the application's request-response cycle. Middlewares can be used for logging, authentication, modifying request/response objects, handling errors, etc.

Types of Middleware in Express.js

1. **Application-level middleware**
2. **Router-level middleware**
3. **Built-in middleware**

4. **Third-party middleware**
5. **Error-handling middleware**

---

1. Application-Level Middleware

Application-level middleware is bound to the entire Express application using app.use() or app.METHOD() (e.g., app.get(), app.post()).

Example: Logging Middleware

```
const express = require('express');
const app = express();

// Middleware function
const logger = (req, res, next) => {
    console.log(`${req.method} ${req.url}`);
    next(); // Move to the next middleware
};
// Use middleware globally
app.use(logger);
app.get('/', (req, res) => {
    res.send('Hello World!');
});
app.listen(3000, () => console.log('Server running on port 3000'));
```

2. Router-Level Middleware

Router-level middleware works the same way as application-level middleware but is applied only to specific routers.

Example: Router Middleware

```
const express = require('express');
const router = express.Router();
// Middleware specific to this router
router.use((req, res, next) => {
    console.log('Router-level middleware executed');
    next();
});
router.get('/', (req, res) => {
    res.send('Router Home Page');
});
const app = express();
app.use('/api', router); // Apply router-level middleware
```

```
app.listen(3000, () => console.log('Server running on port 3000'));
```

---

3. Built-in Middleware

Express comes with some built-in middleware:

Middleware	Description
express.json()	Parses incoming JSON payloads
express.urlencoded({ extended: true })	Parses URL-encoded data
express.static('public')	Serves static files

Example: Using Built-in Middleware

```
app.use(express.json()); // For JSON data
app.use(express.urlencoded({ extended: true })); // For URL-encoded data
app.use(express.static('public')); // Serve static files from 'public' folder
```

---

4. Third-party Middleware

Third-party middleware is installed via npm and used in the app.

Example: cors Middleware

```
const cors = require('cors');
app.use(cors());
```

Other common third-party middlewares:

- morgan (HTTP request logger)
- helmet (security headers)
- cookie-parser (parses cookies)

---

5. Error-Handling Middleware

Error-handling middleware must have **four** parameters: (err, req, res, next)

Example: Global Error Handler

```
app.use((err, req, res, next) => {
```

```
    console.error(err.stack);
    res.status(500).send('Something went wrong!');
});
```

Middleware Execution Order

Middlewares are executed in the order they are defined. If next() is not called, the request will be stuck.

```
app.use((req, res, next) => {
    console.log('Middleware 1');
    next();
});
app.use((req, res, next) => {
    console.log('Middleware 2');
    next();
});
app.get('/', (req, res) => {
    res.send('Response after middleware');
});
// Output in console for / request:
// Middleware 1
// Middleware 2
```

Summary

✅ Use app.use() for application-level middleware.
✅ Use router.use() for router-level middleware.
✅ Use built-in middleware like express.json().
✅ Use third-party middleware like cors, helmet.
✅ Always call next() in middleware to avoid request hanging.

Let me know if you need more details!