**Application-Level Middleware**

Here are more examples of **Application-Level Middleware** in Express.js. These middlewares can be used to handle various tasks like authentication, logging, response modification, and request validation.

**1. Request Logger Middleware**

Logs details about incoming requests.

const express = require('express');

const app = express();

// Logger middleware

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

    console.log(`[${new Date().toISOString()}] ${req.method} ${req.url}`);

    next(); // Proceed to the next middleware or route handler

});

app.get('/', (req, res) => {

    res.send('Home Page');

});

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

// 📌 Output in console for GET / request:

// [2025-01-29T12:00:00.000Z] GET /

**2. Authentication Middleware**

Checks if the request contains an API key.

const authMiddleware = (req, res, next) => {

    const apiKey = req.headers['x-api-key'];

    if (!apiKey || apiKey !== '12345') {

        return res.status(403).json({ message: 'Forbidden: Invalid API Key' });

    }

    next();

};

app.use(authMiddleware); // Apply globally

app.get('/dashboard', (req, res) => {

    res.send('Welcome to the dashboard');

});

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

// 📌 Reques

// GET /dashboard

// x-api-key: 12345  ✅ (Success)

// x-api-key: 00000  ❌ (Forbidden)

**3. Response Time Middleware**

Adds a custom header indicating response time.

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

    const start = Date.now();

    res.on('finish', () => {

        const duration = Date.now() - start;

        console.log(`Request took ${duration}ms`);

    });

    next();

});

app.get('/', (req, res) => {

    res.send('Hello, World!');

});

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

// 📌 Output in console:

// Request took 5ms

**4. Request Data Validator Middleware**

Validates if the request body contains required fields.

app.use(express.json());

const validateUser = (req, res, next) => {

    const { name, email } = req.body;

    if (!name || !email) {

        return res.status(400).json({ message: 'Name and Email are required' });

    }

    next();

};

app.post('/register', validateUser, (req, res) => {

    res.json({ message: 'User registered successfully' });

});

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

// 📌 Request:

// POST /register

// {

//     "name": "John Doe"

// }

// 📌 Response:

// {

//     "message": "Name and Email are required"

// }

**5. Maintenance Mode Middleware**

const maintenanceMode = (req, res, next) => {

    res.status(503).send('Service Unavailable: The site is under maintenance');

};

// Enable maintenance mode

// app.use(maintenanceMode);

app.get('/', (req, res) => {

    res.send('Welcome to the website!');

});

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

// 📌 Uncomment app.use(maintenanceMode) to activate maintenance mode.

// 📌 Response for all requests:

// 503 Service Unavailable

**6. IP Whitelist Middleware**

Allows only specific IPs to access the app.

const allowedIPs = ['127.0.0.1', '192.168.1.100'];

const ipFilter = (req, res, next) => {

    const clientIP = req.ip || req.connection.remoteAddress;

    if (!allowedIPs.includes(clientIP)) {

        return res.status(403).send('Access denied');

    }

    next();

};

app.use(ipFilter);

app.get('/', (req, res) => {

    res.send('Welcome to the secure area');

});

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

**Only requests from 127.0.0.1 or 192.168.1.100 will be allowed.**

**7. Compression Middleware (Using compression)**

Compresses response data to reduce size.

const compression = require('compression');

app.use(compression());

app.get('/', (req, res) => {

    res.send('This response will be compressed.');

});

app.listen(3000, () => console.log('Server running with compression enabled'));

**Response size is reduced using gzip compression.**

**8. Rate Limiting Middleware (Using express-rate-limit)**

Limits requests to prevent abuse.

const rateLimit = require('express-rate-limit');

const limiter = rateLimit({

    windowMs: 15 \* 60 \* 1000, // 15 minutes

    max: 100, // Limit each IP to 100 requests per window

    message: 'Too many requests, please try again later.'

});

app.use(limiter);

app.get('/', (req, res) => {

    res.send('You are within the rate limit.');

});

app.listen(3000, () => console.log('Server running with rate limiting'));

**If an IP exceeds 100 requests in 15 minutes, it will receive a 429 Too Many Requests error.**

**Summary**

| **Middleware** | **Purpose** |
| --- | --- |
| **Logger Middleware** | Logs incoming requests |
| **Authentication Middleware** | Validates API key |
| **Response Time Middleware** | Measures request duration |
| **Request Validator Middleware** | Ensures required fields in requests |
| **Maintenance Mode Middleware** | Blocks requests temporarily |
| **IP Whitelist Middleware** | Restricts access to certain IPs |
| **Compression Middleware** | Reduces response size |
| **Rate Limiting Middleware** | Prevents excessive requests |

Would you like me to add anything else?