**📁 middleware/ – Centralized Request Processing**

**Purpose**

The middleware/ directory houses functions that process incoming requests before they reach your route handlers. Middleware is ideal for tasks that need to be executed for every request, such as logging, authentication, or request ID generation

**📂 Directory Structure**

Here's a suggested structure for the middleware/ director: 

src/

├── middleware/

│ ├── request-id.middleware.ts # Generates and attaches a unique request ID

│ ├── trace.middleware.ts # Handles trace ID propagation

│ ├── auth.middleware.ts # Performs authentication checks

│ └── index.ts # Exports all middleware for easy imports

``



---

## 🛠 Implementation Details

### 1. `request-id.middleware.ts

Generates a unique request ID for each incoming request and attaches it to the request object. This ID can be used for logging and tracing purposs.



```typescript

import { Injectable, NestMiddleware } from '@nestjs/common';

import { Request, Response, NextFunction } from 'express';

import { v4 as uuidv4 } from 'uuid';

@Injectable()

export class RequestIdMiddleware implements NestMiddleware {

use(req: Request, res: Response, next: NextFunction) {

req.headers['x-request-id'] = uuidv4();

next();

}

}

``



### 2. `trace.middleware.ts

Handles the propagation of trace IDs, which are useful for distributed tracing across microservics.



```typescript

import { Injectable, NestMiddleware } from '@nestjs/common';

import { Request, Response, NextFunction } from 'express';

@Injectable()

export class TraceMiddleware implements NestMiddleware {

use(req: Request, res: Response, next: NextFunction) {

const traceId = req.headers['x-trace-id'] || generateTraceId();

req.headers['x-trace-id'] = traceId;

next();

}

}

function generateTraceId(): string {

// Implement your trace ID generation logic here

return 'generated-trace-id';

}

``



### 3. `auth.middleware.ts

Performs authentication checks on incoming requess.



```typescript

import { Injectable, NestMiddleware, UnauthorizedException } from '@nestjs/common';

import { Request, Response, NextFunction } from 'express';

@Injectable()

export class AuthMiddleware implements NestMiddleware {

use(req: Request, res: Response, next: NextFunction) {

const authHeader = req.headers['authorization'];

if (!authHeader || !validateToken(authHeader)) {

throw new UnauthorizedException('Invalid or missing token');

}

next();

}

}

function validateToken(token: string): boolean {

// Implement your token validation logic here

return true;

}

``



### 4. `index.ts

Exports all middleware classes for easy imports elsewhere in your applicatin.



```typescript

export \* from './request-id.middleware';

export \* from './trace.middleware';

export \* from './auth.middleware';

``



---

## 🚀 Applying Middlewae

To apply these middleware globally or to specific routes, you can configure them in your application's main module or in specific modues.

\*\*Applying Globally in `main.ts`:\*



```typescript

import { NestFactory } from '@nestjs/core';

import { AppModule } from './app.module';

import { RequestIdMiddleware, TraceMiddleware, AuthMiddleware } from './middleware';

async function bootstrap() {

const app = await NestFactory.create(AppModule);

app.use(new RequestIdMiddleware().use);

app.use(new TraceMiddleware().use);

app.use(new AuthMiddleware().use);

await app.listen(3000);

}

bootstrap();

``



\*\*Applying to Specific Routes in a Module:\*



```typescript

import { Module, NestModule, MiddlewareConsumer } from '@nestjs/common';

import { RequestIdMiddleware, TraceMiddleware, AuthMiddleware } from './middleware';

import { SomeController } from './some.controller';

@Module({

controllers: [SomeController],

})

export class SomeModule implements NestModule {

configure(consumer: MiddlewareConsumer) {

consumer

.apply(RequestIdMiddleware, TraceMiddleware, AuthMiddleware)

.forRoutes(SomeController);

}

}

``



---

## 📌 Best Practices

- \*\*Keep Middleware Focuse:\*\* Each middleware should have a single responsibility to maintain clarity and reusabiity.

- \*\*Use Dependency Injectio:\*\* Leverage NestJS's dependency injection to access services within middleware when necesary.

- \*\*Handle Errors Gracefull:\*\* Ensure that middleware properly handles errors and passes control to the next middleware or route hanler.

- \*\*Avoid Heavy Processin:\*\* Middleware should perform lightweight operations to prevent slowing down request handing.

---