Middleware Design

Middleware Design Document

> **Project:** {{PROJECT_NAME}} **Version:** {{VERSION}} **Date:** {{DATE}}
> **Author:** {{AUTHOR}} **Status:** Draft | In Review | Approved **Reviewers:** {{REVIEWERS}}

Document History

Version	Date	Author	Changes
0.1	{{DATE}}	{{AUTHOR}}	Initial draft

---

1. Middleware Pipeline Overview

```
sequenceDiagram
    participant Client
    participant CORS as 1. CORS
    participant Security as 2. Security Headers
    participant RequestID as 3. Request ID
    participant RateLimit as 4. Rate Limiter
    participant Logger as 5. Request Logger
    participant Auth as 6. Authentication
    participant Authz as 7. Authorization (RBAC)
    participant Validate as 8. Validation
    participant AuditLog as 9. Audit Logger
    participant Handler as Route Handler

    Client->>CORS: HTTP Request
    CORS->>Security: (CORS headers set)
```

```
    Security->>RequestID: (Security headers set)
    RequestID->>RateLimit: (X-Request-ID injected)
    RateLimit->>Logger: (Rate check passed)
    Logger->>Auth: (Request logged)
    Auth->>Authz: (JWT validated, user attached)
    Authz->>Validate: (Permissions verified)
    Validate->>AuditLog: (Input validated & sanitized)
    AuditLog->>Handler: (Audit record written)
    Handler-->>Client: Response
```

Framework: `{{NestJS / Express / Fastify / Hono}}` **Execution order is strict** — changing order may break security guarantees.

---

2. Request Lifecycle

2.1 CORS Middleware

Library: `{{cors / @fastify/cors}}`

Configuration:

```
// config/cors.config.ts
export const corsConfig = {
  origin: (origin: string, callback: Function) => {
    const allowedOrigins = [
      'https://app.{{domain.com}}',
      'https://admin.{{domain.com}}',
      ...(process.env.NODE_ENV !== 'production'
        ? ['http://localhost:3000', 'http://localhost:3001']
        : []),
    ];

    if (!origin || allowedOrigins.includes(origin)) {
      callback(null, true);
    } else {
      callback(new Error(`Origin ${origin} not allowed by CORS`));
    }
  },
```

```
  methods: ['GET', 'POST', 'PUT', 'PATCH', 'DELETE', 'OPTIONS'],
  allowedHeaders: ['Content-Type', 'Authorization', 'X-Request-ID'],
  exposedHeaders: ['X-Request-ID', 'X-RateLimit-Limit', 'X-RateLimit-Remaining'],
  credentials: true,
  maxAge: 86400, // 24h preflight cache
};
```

Performance impact: < 0.1ms per request (header injection only)

2.2 Security Headers Middleware

Library: `helmet`

```
app.use(helmet({
  // Content Security Policy
  contentSecurityPolicy: {
    directives: {
      defaultSrc: ["'self'"],
      scriptSrc: ["'self'"],
      styleSrc: ["'self'", "'unsafe-inline'"],
      imgSrc: ["'self'", "data:", "https://cdn.{{domain.com}}"],
      connectSrc: ["'self'", "https://api.{{domain.com}}"],
      frameSrc: ["'none'"],
      objectSrc: ["'none'"],
    },
  },
  // HTTP Strict Transport Security
  hsts: {
    maxAge: 31536000,       // 1 year
    includeSubDomains: true,
    preload: true,
  },
  // Other headers
  referrerPolicy: { policy: 'same-origin' },
  frameguard: { action: 'deny' },
  noSniff: true,            // X-Content-Type-Options: nosniff
  xssFilter: true,          // X-XSS-Protection (legacy browsers)
  hidePoweredBy: true,      // Remove X-Powered-By
}));
```

Headers set:

Header	Value	Purpose
`Strict-Transport-Security`	`max-age=31536000; includeSubDomains; preload`	Force HTTPS
`Content-Security-Policy`	(see above)	XSS prevention
`X-Frame-Options`	`DENY`	Clickjacking prevention
`X-Content-Type-Options`	`nosniff`	MIME sniffing prevention
`Referrer-Policy`	`same-origin`	Referrer privacy

Performance impact: < 0.2ms per request

2.3 Request ID Middleware

Purpose: Correlate logs across services for distributed tracing.

```
// middleware/request-id.middleware.ts
export function requestIdMiddleware(req: Request, res: Response, next: NextFunction) {
  const requestId = req.headers['x-request-id'] as string
    || `req_${ulid()}`;

  req.requestId = requestId;
  res.setHeader('X-Request-ID', requestId);

  // Bind to AsyncLocalStorage for log correlation
  requestContext.run({ requestId }, next);
}
```

Format: `req_{ulid}` — e.g., `req_01HX7M2K5N3P4Q5R6S7T8V9W0`

2.4 Authentication Middleware

Strategy: JWT Bearer token validation

```
// guards/jwt.guard.ts
@Injectable()
export class JwtGuard implements CanActivate {
  async canActivate(context: ExecutionContext): Promise<boolean> {
```

```
    const request = context.switchToHttp().getRequest();
    const token = this.extractToken(request);

    if (!token) throw new UnauthorizedException('No token provided');

    try {
      const payload = await this.jwtService.verifyAsync(token, {
        secret: this.configService.get('JWT_SECRET'),
        algorithms: ['HS256'],
        clockTolerance: 10, // 10 second clock skew tolerance
      });

      // Attach to request for downstream use
      request.user = {
        id: payload.sub,
        email: payload.email,
        role: payload.role,
      };

      return true;
    } catch (error) {
      if (error instanceof TokenExpiredError) {
        throw new UnauthorizedException('TOKEN_EXPIRED');
      }
      throw new UnauthorizedException('INVALID_TOKEN');
    }
  }

  private extractToken(request: Request): string | null {
    const [type, token] = request.headers.authorization?.split(' ') ?? [];
    return type === 'Bearer' ? token : null;
  }
}
```

Performance impact: ~2-5ms (crypto operation + optional DB lookup for token revocation)

Token revocation check: `{{Check Redis blocklist on each request | Check on logout only | Use short TTL — no revocation check}}`

2.5 Authorization Middleware (RBAC/ABAC)

Model: `{{RBAC (Role-Based) | ABAC (Attribute-Based) | Hybrid}}`

```
// decorators/roles.decorator.ts
export const Roles = (...roles: Role[]) => SetMetadata('roles', roles);
export const RequirePermission = (permission: string) =>
  SetMetadata('permission', permission);

// guards/roles.guard.ts
@Injectable()
export class RolesGuard implements CanActivate {
  canActivate(context: ExecutionContext): boolean {
    const requiredRoles = this.reflector.get<Role[]>('roles', context.getHandler());
    const requiredPermission = this.reflector.get<string>('permission', context.getHandler());

    if (!requiredRoles && !requiredPermission) return true; // Public route

    const { user } = context.switchToHttp().getRequest();

    if (requiredRoles && !requiredRoles.includes(user.role)) {
      throw new ForbiddenException(`Requires role: ${requiredRoles.join(' or ')}`);
    }

    if (requiredPermission && !this.hasPermission(user, requiredPermission)) {
      throw new ForbiddenException(`Requires permission: ${requiredPermission}`);
    }

    return true;
  }
}

// Usage on controller
@Get('users')
@Roles(Role.ADMIN)
@RequirePermission('users:read')
async listUsers() { ... }
```

Role hierarchy:

```
admin > manager > user > viewer > public
```

Role	Capabilities
`admin`	Full access
`manager`	Read/write own org resources
`user`	Read/write own resources
`viewer`	Read-only

2.6 Validation Middleware

Library: `class-validator + class-transformer` OR `zod`

```
// Global validation pipe (NestJS)
app.useGlobalPipes(new ValidationPipe({
  whitelist: true,              // Strip unknown properties
  forbidNonWhitelisted: true,  // Throw if unknown properties present
  transform: true,              // Auto-transform to DTO types
  transformOptions: {
    enableImplicitConversion: true,
  },
}));
```

Sanitization rules:

- All string inputs: trim whitespace
- HTML content: sanitize with `DOMPurify` / `sanitize-html` (strip dangerous tags)
- SQL parameters: always use parameterized queries (ORM handles this)
- File uploads: validate MIME type by magic bytes (not just extension)

Validation error format:

```
{
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Request validation failed",
    "details": [
      { "field": "email", "message": "must be an email" },
      { "field": "name", "message": "must be longer than 2 characters" }
    ]
  }
}
```

Performance impact: < 1ms for typical DTOs

2.7 Rate Limiting Middleware

Library: `{{@nestjs/throttler | express-rate-limit | rate-limiter-flexible}}` **Storage:** `{{Redis}}` (shared across all replicas)

Algorithms:

Algorithm	Library	Best For
Fixed window	`express-rate-limit`	Simple, low overhead
Sliding window	`rate-limiter-flexible`	Accurate, no burst at window edge
Token bucket	`rate-limiter-flexible`	Bursty traffic patterns

Selected algorithm: `{{Sliding window}}`

Configuration:

```
const rateLimiter = new RateLimiterRedis({
  storeClient: redisClient,
  keyPrefix: 'rl',
  points: 1000,         // Number of points
  duration: 60,         // Per 60 seconds
  blockDuration: 60,    // Block for 60s after exceeded
});

// Per-route overrides
const loginLimiter = new RateLimiterRedis({
  points: 5,
  duration: 900,        // 15 minutes
  blockDuration: 900,
});
```

Key strategy: `{{IP address | User ID (if authenticated) | IP + User ID}}`

Performance impact: ~1-2ms (Redis round-trip)

2.8 Audit Logging Middleware

```
// interceptors/audit.interceptor.ts
@Injectable()
export class AuditInterceptor implements NestInterceptor {
  intercept(context: ExecutionContext, next: CallHandler): Observable<any> {
    const request = context.switchToHttp().getRequest();
    const { method, path, user, requestId } = request;

    // Only audit mutating operations
    if (['POST', 'PUT', 'PATCH', 'DELETE'].includes(method)) {
      this.auditService.log({
        requestId,
        userId: user?.id,
        method,
        path,
        body: this.sanitizeBody(request.body), // Strip PII
        timestamp: new Date().toISOString(),
      });
    }

    return next.handle();
  }

  private sanitizeBody(body: Record<string, unknown>) {
    const REDACTED_FIELDS = ['password', 'token', 'creditCard', 'ssn'];
    return Object.fromEntries(
      Object.entries(body).map(([key, value]) =>
        REDACTED_FIELDS.includes(key) ? [key, '[REDACTED]'] : [key, value]
      )
    );
  }
}
```

What IS logged:

- User ID, request ID, timestamp, method, path
- Response status code, duration
- Mutation summaries (what changed, not full values)

What is NEVER logged:

- Passwords, tokens, API keys

- Payment card data
- Full PII fields (log field names but not values for sensitive fields)

Audit log retention: `{{1 year}}` (compliance requirement: `{{GDPR / SOC2 / internal}}`)

2.9 Error Handling Middleware

```
// filters/global-exception.filter.ts
@Catch()
export class GlobalExceptionFilter implements ExceptionFilter {
  catch(exception: unknown, host: ArgumentsHost) {
    const ctx = host.switchToHttp();
    const response = ctx.getResponse<Response>();
    const request = ctx.getRequest<Request>();

    let status = 500;
    let code = 'INTERNAL_ERROR';
    let message = 'An unexpected error occurred';
    let details: unknown[] = [];

    if (exception instanceof HttpException) {
      status = exception.getStatus();
      const exceptionResponse = exception.getResponse() as any;
      code = exceptionResponse.code ?? 'HTTP_ERROR';
      message = exceptionResponse.message ?? exception.message;
      details = exceptionResponse.details ?? [];
    }

    // Log 5xx errors (not 4xx — those are client errors)
    if (status >= 500) {
      this.logger.error('Unhandled exception', { exception, requestId: request.requestId });
      this.sentryService.captureException(exception);
    }

    response.status(status).json({
      error: {
        code,
        message,
        details,
```

```
        requestId: request.requestId,
        timestamp: new Date().toISOString(),
      },
    });
  }
}
```

---

3. Custom Middleware Development Guide

Template for new middleware:

```
// middleware/{{name}}.middleware.ts
import { Injectable, NestMiddleware } from '@nestjs/common';
import { Request, Response, NextFunction } from 'express';

@Injectable()
export class {{Name}}Middleware implements NestMiddleware {
  use(req: Request, res: Response, next: NextFunction): void {
    // 1. Extract needed data from request
    // 2. Perform validation/enrichment/logging
    // 3. Set data on request object if needed
    // 4. Call next() or throw HttpException

    next();
  }
}
```

Requirements for new middleware:

- ☐ Handles errors without crashing the process
- ☐ Calls `next()` exactly once (or throws)
- ☐ Does not block async operations without async/await
- ☐ Performance impact documented
- ☐ Unit tests covering happy path + error path

---

4. Middleware Ordering & Dependencies

```
Request → [1] → [2] → [3] → [4] → [5] → [6] → [7] → [8] → [9] → Handler

[1] CORS          — No dependencies
[2] Security      — No dependencies
[3] Request ID    — Must be before Logger (Logger reads requestId)
[4] Rate Limiter  — Must be after Request ID (uses requestId for key)
[5] Logger        — Must be after Request ID
[6] Auth          — Must be after Logger (Logger should log auth failures)
[7] Authorization — MUST be after Auth (requires user on request)
[8] Validation    — MUST be after Auth (DTOs may reference user context)
[9] Audit Logger  — MUST be after Auth (logs user ID)
```

NEVER reorder middleware without reviewing this dependency chain.

---

5. Performance Impact Per Middleware

Middleware	Avg Latency Added	P99 Latency Added	Notes
CORS	0.05ms	0.1ms	Header injection only
Security Headers (Helmet)	0.1ms	0.2ms	Header injection only
Request ID	0.1ms	0.2ms	ID generation
Rate Limiter	1.5ms	5ms	Redis round-trip
Request Logger	0.5ms	1ms	Async log write
Authentication (JWT)	3ms	8ms	Crypto + optional Redis
Authorization	0.5ms	1ms	In-memory role check
Validation	0.8ms	2ms	Schema parsing
Audit Logger	0.5ms	1ms	Async DB write
Total	**~7ms**	**~18ms**	Middleware overhead

Target: middleware overhead < 10ms P50, < 25ms P99.

---

6. Testing Strategy for Middleware

```
// Example unit test for Auth middleware
describe('JwtGuard', () => {
  it('should attach user to request on valid token', async () => {
    const token = generateTestToken({ sub: 'usr_123', role: 'user' });
    const mockRequest = { headers: { authorization: `Bearer ${token}` } };
    const result = await guard.canActivate(createMockContext(mockRequest));
    expect(result).toBe(true);
    expect(mockRequest.user).toMatchObject({ id: 'usr_123', role: 'user' });
  });

  it('should throw UnauthorizedException on expired token', async () => {
    const expiredToken = generateExpiredToken();
    const mockRequest = { headers: { authorization: `Bearer ${expiredToken}` } };
    await expect(guard.canActivate(createMockContext(mockRequest)))
      .rejects.toThrow('TOKEN_EXPIRED');
  });
});
```

Test coverage requirements:

- Each middleware: ≥ 90% line coverage
- Security middleware (Auth, AuthZ, Validation): 100% branch coverage

---

7. Configuration Options Per Middleware

Middleware	Environment Variable	Default	Description
CORS	`CORS_ORIGINS`	`localhost:3000`	Comma-separated allowed origins
Rate Limit	`RATE_LIMIT_POINTS`	`1000`	Requests per window
Rate Limit	`RATE_LIMIT_DURATION`	`60`	Window size in seconds
Rate Limit (auth)	`AUTH_RATE_LIMIT_POINTS`	`5`	Login attempts per window
Audit Log	`AUDIT_LOG_RETENTION_DAYS`	`365`	How long to keep audit records

Middleware	Environment Variable	Default	Description
Request Body	`MAX_REQUEST_BODY_SIZE`	`1mb`	Max request body size

Approval

Role	Name	Date	Signature
Author			
Backend Lead			
Security Lead			
Tech Lead			

Revision #6
Created 2026-02-23 12:05:28 UTC by John
Updated 2026-05-25 07:33:32 UTC by John