Middleware Stack

Bilko Middleware Stack

> **Status:** SPECIFICATION (backend not implemented) **Last updated:** 2026-02-20

Purpose

This document specifies the Express middleware stack for Bilko's backend. Middleware order is CRITICAL — security, authentication, validation, and error handling must execute in the correct sequence.

Middleware Execution Order

The order matters. Middleware executes top-to-bottom:

Full Middleware Pipeline

```
flowchart TD
    REQ[Incoming HTTP Request]

    REQ --> H[1. Helmet\nSecurity Headers\nHSTS, CSP, X-Frame-Options,\nX-Content-Type-Options]

    H --> CORS[2. CORS\nAllow: bilko.io, localhost:3000\ncredentials: true\nmaxAge: 86400]

    CORS --> BP[3. Body Parser\nexpress.json limit=10mb\nexpress.urlencoded]

    BP --> RL{4. Rate Limiter\nAuth: 5 req/min\nGeneral: 100 req/min}
```

```
    RL -->|Exceeded| R429[429 Too Many Requests]
    RL -->|OK| LOG[5. Morgan Logger\nWinston transport\nCombined format]

    LOG --> ROUTE[6. Router\n/api/v1/*]

    ROUTE --> AUTH{authGuard\nVerify Bearer JWT}
    AUTH -->|No token| R401A[401 NO_TOKEN]
    AUTH -->|Expired| R401B[401 TOKEN_EXPIRED]
    AUTH -->|Invalid| R401C[401 INVALID_TOKEN]
    AUTH -->|Valid| ATTACH[Attach req.user\n{id, email, role, orgId}]

    ATTACH --> ROLE{roleGuard\nCheck allowed roles}
    ROLE -->|Insufficient| R403[403 INSUFFICIENT_PERMISSIONS]
    ROLE -->|Authorized| VALID{validate\nZod schema}

    VALID -->|Fails| R422[422 VALIDATION_ERROR]
    VALID -->|Passes| SCOPE[organizationScope\nAttach req.organizationId]

    SCOPE --> HANDLER[Route Handler\nBusiness Logic + Prisma]

    HANDLER --> AUDIT[Prisma Middleware\nLoggedAction INSERT]
    AUDIT --> RESP[200/201/204 Response]

    HANDLER -->|Error thrown| ERR[7. Error Handler\nMUST be last middleware]
    ERR --> FMTERR[Format error response\n{error, code, details}]
    FMTERR --> ERRRESP[4xx/500 Response]

    style REQ fill:#00E5A0,color:#000
    style R429 fill:#f87171,color:#fff
    style R401A fill:#f87171,color:#fff
    style R401B fill:#f87171,color:#fff
    style R401C fill:#f87171,color:#fff
    style R403 fill:#f87171,color:#fff
    style R422 fill:#f87171,color:#fff
    style RESP fill:#4ade80,color:#000
```

```
import express from 'express'
import helmet from 'helmet'
import cors from 'cors'
import rateLimit from 'express-rate-limit'
```

```
import { authGuard, roleGuard } from './middleware/auth'
import { validate } from './middleware/validation'
import { errorHandler } from './middleware/error-handler'

const app = express()

// 1. Security headers (helmet)
app.use(helmet())

// 2. CORS configuration
app.use(cors(corsOptions))

// 3. Body parsing
app.use(express.json({ limit: '10mb' }))
app.use(express.urlencoded({ extended: true }))

// 4. Rate limiting
app.use(rateLimiter)

// 5. Request logging (Morgan)
app.use(morgan('combined'))

// 6. Routes (with auth + validation per-route)
app.use('/api/v1', routes)

// 7. Error handler (MUST be last)
app.use(errorHandler)
```

---

1. Helmet (Security Headers)

Purpose: Sets HTTP security headers to prevent common attacks.

Installation:

```
npm install helmet
```

Configuration:

```
import helmet from 'helmet'

app.use(helmet({
  contentSecurityPolicy: {
    directives: {
      defaultSrc: ["'self'"],
      styleSrc: ["'self'", "'unsafe-inline'"],  // Allow inline styles for Next.js
      scriptSrc: ["'self'"],
      imgSrc: ["'self'", "data:", "https://r2.bilko.io"],  // Cloudflare R2
      connectSrc: ["'self'", "https://api.bilko.io"],
      fontSrc: ["'self'"],
      objectSrc: ["'none'"],
      upgradeInsecureRequests: []
    }
  },
  hsts: {
    maxAge: 31536000,            // 1 year
    includeSubDomains: true,
    preload: true
  },
  frameguard: { action: 'deny' },  // Prevent clickjacking
  noSniff: true,                   // Prevent MIME sniffing
  xssFilter: true                  // Enable XSS filter
}))
```

Headers set:

- `Strict-Transport-Security` — Force HTTPS
- `X-Content-Type-Options: nosniff` — Prevent MIME sniffing
- `X-Frame-Options: DENY` — Prevent clickjacking
- `X-XSS-Protection: 1; mode=block` — Enable XSS filter
- `Content-Security-Policy` — Restrict resource loading

---

2. CORS (Cross-Origin Resource Sharing)

Purpose: Allow frontend (Next.js) to call backend API from different origin.

Installation:

```
npm install cors
```

Configuration:

```
import cors from 'cors'

const corsOptions = {
  origin: (origin, callback) => {
    const allowedOrigins = [
      'https://bilko.io',
      'https://www.bilko.io',
      'http://localhost:3000'  // Development only
    ]

    if (!origin || allowedOrigins.includes(origin)) {
      callback(null, true)
    } else {
      callback(new Error('Not allowed by CORS'))
    }
  },
  credentials: true,              // Allow cookies (refresh tokens)
  methods: ['GET', 'POST', 'PUT', 'PATCH', 'DELETE'],
  allowedHeaders: ['Content-Type', 'Authorization'],
  exposedHeaders: ['X-Total-Count', 'X-Page-Count'],  // For pagination
  maxAge: 86400                   // Cache preflight for 24h
}

app.use(cors(corsOptions))
```

Why credentials: true?

- Allows httpOnly cookies (refresh tokens)
- Frontend must set `credentials: 'include'` in fetch()

---

3. Body Parsing

Purpose: Parse JSON request bodies.

Built-in Express middleware:

```
app.use(express.json({
  limit: '10mb',                  // Max request body size
  strict: true,                   // Reject non-arrays/objects
  type: 'application/json'
}))

app.use(express.urlencoded({
  extended: true,
  limit: '10mb'
}))
```

Limits:

- 10MB for file uploads (receipts, CSV imports)
- Reject if Content-Length > 10MB
- Return `413 Payload Too Large`

---

4. Rate Limiting

Purpose: Prevent abuse, brute-force attacks, DDoS.

Installation:

```
npm install express-rate-limit
```

Configuration:

```
import rateLimit from 'express-rate-limit'

// General API rate limiter
const apiLimiter = rateLimit({
  windowMs: 60 * 1000,            // 1 minute
  max: 100,                       // Max 100 requests per minute
  message: {
    error: 'Too many requests',
    code: 'RATE_LIMIT_EXCEEDED',
    retryAfter: 60
  },
```

```
  standardHeaders: true,          // Return RateLimit-* headers
  legacyHeaders: false,
  handler: (req, res) => {
    res.status(429).json({
      error: 'Too many requests',
      code: 'RATE_LIMIT_EXCEEDED',
      retryAfter: 60
    })
  }
})

// Auth rate limiter (stricter)
const authLimiter = rateLimit({
  windowMs: 60 * 1000,            // 1 minute
  max: 5,                         // Max 5 login attempts per minute
  skipSuccessfulRequests: true,  // Don't count successful logins
  keyGenerator: (req) => {
    return req.ip                 // Rate limit by IP
  }
})

// Apply to routes
app.use('/api/v1', apiLimiter)
app.use('/api/v1/auth/login', authLimiter)
app.use('/api/v1/auth/register', authLimiter)
```

Rate limits by endpoint:

Endpoint	Limit	Window	Why
/api/v1/auth/login	5	1 min	Prevent brute-force
/api/v1/auth/register	5	1 min	Prevent spam registration
/api/v1/* (general)	100	1 min	General API protection
Write ops (POST/PUT/PATCH)	50	1 min	Prevent resource exhaustion

---

5. Request Logging

Purpose: Log all HTTP requests for debugging and monitoring.

Installation:

```
npm install morgan
npm install winston
```

Configuration:

```
import morgan from 'morgan'
import winston from 'winston'

// Winston logger
const logger = winston.createLogger({
  level: 'info',
  format: winston.format.json(),
  transports: [
    new winston.transports.File({ filename: 'error.log', level: 'error' }),
    new winston.transports.File({ filename: 'combined.log' })
  ]
})

if (process.env.NODE_ENV !== 'production') {
  logger.add(new winston.transports.Console({
    format: winston.format.simple()
  }))
}

// Morgan HTTP logging
app.use(morgan('combined', {
  stream: {
    write: (message) => logger.info(message.trim())
  }
}))
```

Log format:

```
:remote-addr - :remote-user [:date[clf]] ":method :url HTTP/:http-version" :status
:res[content-length] ":referrer" ":user-agent"
```

Example:

```
192.168.1.100 - user@example.com [20/Feb/2026:10:30:15 +0000] "POST /api/v1/invoices HTTP/1.1"
201 512 "https://bilko.io" "Mozilla/5.0..."
```

Per-Route Middleware Composition

```
graph LR
    subgraph PUBLIC [Public Routes — No Auth]
        P1["POST /auth/register\n[rateLimiter(5/min)] → handler"]
        P2["POST /auth/login\n[rateLimiter(5/min)] → handler"]
        P3["POST /auth/refresh\n[rateLimiter(100/min)] → handler"]
        P4["GET /track/email/:id\n[handler — tracking pixel]"]
    end

    subgraph VIEWER [Viewer Routes — All Roles]
        V1["GET /invoices\n[auth] → [orgScope] → handler"]
        V2["GET /reports/*\n[auth] → [orgScope] → handler"]
        V3["GET /contacts\n[auth] → [orgScope] → handler"]
    end

    subgraph ACCOUNTANT [Accountant+ Routes]
        A1["POST /invoices\n[auth] → [role:owner,admin,accountant]\n→ [validate] → [orgScope]
→ handler"]
        A2["POST /expenses\n[auth] → [role:owner,admin,accountant]\n→ [validate] → handler"]
        A3["POST /transactions\n[auth] → [role:owner,admin,accountant]\n→ [validate] →
handler"]
    end

    subgraph ADMIN [Admin+ Routes]
        AD1["PATCH /expenses/:id/approve\n[auth] → [role:owner,admin] → handler"]
        AD2["POST /users/invite\n[auth] → [role:owner,admin] → handler"]
        AD3["PUT /organization\n[auth] → [role:owner,admin] → handler"]
    end

    subgraph OWNER [Owner-Only Routes]
        O1["DELETE /users/:id\n[auth] → [role:owner] → handler"]
        O2["PUT /users/:id/role\n[auth] → [role:owner] → handler"]
        O3["DELETE /organization\n[auth] → [role:owner] → handler"]
    end
```

```
    style PUBLIC fill:#e2e8f0,color:#000
    style VIEWER fill:#dcfce7,color:#000
    style ACCOUNTANT fill:#fef9c3,color:#000
    style ADMIN fill:#dbeafe,color:#000
    style OWNER fill:#fce7f3,color:#000
```

6. Authentication Middleware

Purpose: Verify JWT access token, attach user to request.

Implementation:

```
import jwt from 'jsonwebtoken'
import { Request, Response, NextFunction } from 'express'

interface AuthRequest extends Request {
  user?: {
    id: string
    email: string
    role: 'owner' | 'admin' | 'accountant' | 'viewer'
    organizationId: string
  }
}

export async function authGuard(req: AuthRequest, res: Response, next: NextFunction) {
  const authHeader = req.headers.authorization

  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    return res.status(401).json({
      error: 'Unauthorized',
      code: 'NO_TOKEN'
    })
  }

  const token = authHeader.substring(7)  // Remove 'Bearer '

  try {
    const payload = jwt.verify(token, process.env.JWT_SECRET!) as {
```

```
      sub: string
      email: string
      role: string
      orgId: string
    }

    // Attach user to request
    req.user = {
      id: payload.sub,
      email: payload.email,
      role: payload.role as any,
      organizationId: payload.orgId
    }

    next()
  } catch (error) {
    if (error.name === 'TokenExpiredError') {
      return res.status(401).json({
        error: 'Token expired',
        code: 'TOKEN_EXPIRED'
      })
    }

    if (error.name === 'JsonWebTokenError') {
      return res.status(401).json({
        error: 'Invalid token',
        code: 'INVALID_TOKEN'
      })
    }

    return res.status(500).json({
      error: 'Authentication error',
      code: 'AUTH_ERROR'
    })
  }
}
```

Usage in routes:

```
app.get('/api/v1/invoices', authGuard, getInvoices)
```

7. Role-Based Access Control (RBAC)

Purpose: Restrict endpoints by user role.

Implementation:

```
type UserRole = 'owner' | 'admin' | 'accountant' | 'viewer'

export function roleGuard(allowedRoles: UserRole[]) {
  return (req: AuthRequest, res: Response, next: NextFunction) => {
    if (!req.user) {
      return res.status(401).json({
        error: 'Unauthorized',
        code: 'NO_AUTH'
      })
    }

    if (!allowedRoles.includes(req.user.role)) {
      return res.status(403).json({
        error: 'Forbidden',
        code: 'INSUFFICIENT_PERMISSIONS',
        details: {
          required: allowedRoles,
          current: req.user.role
        }
      })
    }

    next()
  }
}
```

Usage in routes:

```
// Only owner and admin can delete users
app.delete('/api/v1/users/:id',
  authGuard,
  roleGuard(['owner', 'admin']),
```

```
  deleteUser
)

// Everyone can view invoices
app.get('/api/v1/invoices',
  authGuard,
  getInvoices
)

// Only owner, admin, accountant can create invoices
app.post('/api/v1/invoices',
  authGuard,
  roleGuard(['owner', 'admin', 'accountant']),
  createInvoice
)
```

---

8. Request Validation

Purpose: Validate request body, query, params with Zod schemas.

Installation:

```
npm install zod
```

Implementation:

```
import { z } from 'zod'
import { Request, Response, NextFunction } from 'express'

type ValidateTarget = 'body' | 'query' | 'params'

export function validate(schema: z.ZodSchema, target: ValidateTarget = 'body') {
  return (req: Request, res: Response, next: NextFunction) => {
    try {
      const data = req[target]
      const validated = schema.parse(data)

      // Replace with validated data (coerced types)
      req[target] = validated
```

```
      next()
    } catch (error) {
      if (error instanceof z.ZodError) {
        return res.status(422).json({
          error: 'Validation failed',
          code: 'VALIDATION_ERROR',
          details: error.flatten().fieldErrors
        })
      }

      return res.status(500).json({
        error: 'Validation error',
        code: 'VALIDATION_ERROR'
      })
    }
  }
}
```

Usage in routes:

```
import { z } from 'zod'

const createInvoiceSchema = z.object({
  customerId: z.string().uuid(),
  invoiceDate: z.string().date(),
  dueDate: z.string().date(),
  items: z.array(z.object({
    description: z.string().min(1).max(500),
    quantity: z.number().positive(),
    unitPrice: z.number().positive(),
    taxRate: z.number().min(0).max(100)
  })).min(1)
})

app.post('/api/v1/invoices',
  authGuard,
  roleGuard(['owner', 'admin', 'accountant']),
  validate(createInvoiceSchema, 'body'),
  createInvoice
```

```
)
```

Error response:

```
{
  "error": "Validation failed",
  "code": "VALIDATION_ERROR",
  "details": {
    "customerId": ["Invalid UUID"],
    "items.0.quantity": ["Must be positive"]
  }
}
```

---

9. Organization Scoping

Purpose: Automatically filter all queries by `organizationId` to enforce multi-tenancy.

Implementation:

```
export function organizationScope(req: AuthRequest, res: Response, next: NextFunction) {
  if (!req.user) {
    return res.status(401).json({
      error: 'Unauthorized',
      code: 'NO_AUTH'
    })
  }

  // Attach organizationId to request for easy access
  req.organizationId = req.user.organizationId

  next()
}
```

Usage in Prisma queries:

```
async function getInvoices(req: AuthRequest, res: Response) {
  const invoices = await prisma.invoice.findMany({
    where: {
      organizationId: req.user!.organizationId  // Always filter by org
```

```
    }
  })

  res.json({ data: invoices })
}
```

CRITICAL: NEVER allow cross-organization queries. Always filter by `organizationId`.

---

10. Error Handler

Purpose: Catch all errors, format consistently, log, return to client.

MUST be the last middleware.

```
flowchart TD
    ERR[Error Thrown by any Middleware or Handler]
    ERR --> LOG_ERR[Log to Winston:\nmessage, stack, path, method, userId]
    LOG_ERR --> TYPE{Error Type?}

    TYPE -->|PrismaClientKnownRequestError| PRISMA{Prisma Code?}
    PRISMA -->|P2002 Unique violation| R400A[400 DUPLICATE_RESOURCE\n{field: target}]
    PRISMA -->|P2003 Foreign key| R404A[404 FOREIGN_KEY_ERROR]
    PRISMA -->|P2025 Record not found| R404B[404 NOT_FOUND]
    PRISMA -->|Other| R500[500 INTERNAL_ERROR]

    TYPE -->|ValidationError| R422[422 VALIDATION_ERROR]
    TYPE -->|JsonWebTokenError| R401A[401 INVALID_TOKEN]
    TYPE -->|TokenExpiredError| R401B[401 AUTH_ERROR]
    TYPE -->|Custom AppError| CUSTOM[err.status / err.code]
    TYPE -->|Unknown| R500

    R400A & R404A & R404B & R422 & R401A & R401B & CUSTOM & R500 --> FORMAT[Format
Response:\n{ error, code, details? }]
    FORMAT --> SEND[Send to Client]

    style ERR fill:#f87171,color:#fff
    style R500 fill:#dc2626,color:#fff
    style FORMAT fill:#60a5fa,color:#000
```

Implementation:

```
import { Request, Response, NextFunction } from 'express'
import { Prisma } from '@prisma/client'

export function errorHandler(err: any, req: Request, res: Response, next: NextFunction) {
  // Log error
  logger.error('Error:', {
    message: err.message,
    stack: err.stack,
    path: req.path,
    method: req.method,
    user: req.user?.id
  })

  // Prisma errors
  if (err instanceof Prisma.PrismaClientKnownRequestError) {
    // Unique constraint violation
    if (err.code === 'P2002') {
      return res.status(400).json({
        error: 'Resource already exists',
        code: 'DUPLICATE_RESOURCE',
        details: { field: err.meta?.target }
      })
    }

    // Foreign key constraint violation
    if (err.code === 'P2003') {
      return res.status(404).json({
        error: 'Related resource not found',
        code: 'FOREIGN_KEY_ERROR'
      })
    }

    // Record not found
    if (err.code === 'P2025') {
      return res.status(404).json({
        error: 'Resource not found',
        code: 'NOT_FOUND'
      })
```

```
    }
  }

  // Validation errors (already handled by validate middleware)
  if (err.name === 'ValidationError') {
    return res.status(422).json({
      error: err.message,
      code: 'VALIDATION_ERROR'
    })
  }

  // JWT errors (already handled by authGuard)
  if (err.name === 'JsonWebTokenError' || err.name === 'TokenExpiredError') {
    return res.status(401).json({
      error: 'Authentication failed',
      code: 'AUTH_ERROR'
    })
  }

  // Default error
  res.status(err.status || 500).json({
    error: err.message || 'Internal server error',
    code: err.code || 'INTERNAL_ERROR'
  })
}
```

Error response format:

```
{
  "error": "Human-readable error message",
  "code": "MACHINE_READABLE_CODE",
  "details": {
    "field": "Additional context"
  }
}
```

Complete Middleware Stack Example

```
import express from 'express'
import helmet from 'helmet'
import cors from 'cors'
import morgan from 'morgan'
import rateLimit from 'express-rate-limit'
import { authGuard, roleGuard, organizationScope } from './middleware/auth'
import { validate } from './middleware/validation'
import { errorHandler } from './middleware/error-handler'
import routes from './routes'

const app = express()

// 1. Security headers
app.use(helmet(helmetConfig))

// 2. CORS
app.use(cors(corsOptions))

// 3. Body parsing
app.use(express.json({ limit: '10mb' }))
app.use(express.urlencoded({ extended: true }))

// 4. Rate limiting
app.use('/api/v1', apiLimiter)
app.use('/api/v1/auth/login', authLimiter)
app.use('/api/v1/auth/register', authLimiter)

// 5. Request logging
app.use(morgan('combined', { stream: logger.stream }))

// 6. Routes
app.use('/api/v1', routes)

// 7. Error handler (MUST be last)
app.use(errorHandler)

// Start server
const PORT = process.env.PORT || 4000
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`)
```

```
})
```

Middleware Testing

Unit tests for each middleware:

```
import { describe, it, expect } from 'vitest'
import request from 'supertest'
import app from '../app'

describe('Auth Middleware', () => {
  it('blocks request without token', async () => {
    const res = await request(app).get('/api/v1/invoices')
    expect(res.status).toBe(401)
    expect(res.body.code).toBe('NO_TOKEN')
  })

  it('blocks request with expired token', async () => {
    const expiredToken = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...'
    const res = await request(app)
      .get('/api/v1/invoices')
      .set('Authorization', `Bearer ${expiredToken}`)

    expect(res.status).toBe(401)
    expect(res.body.code).toBe('TOKEN_EXPIRED')
  })

  it('allows request with valid token', async () => {
    const validToken = generateToken({ sub: 'user-id', role: 'owner', orgId: 'org-id' })
    const res = await request(app)
      .get('/api/v1/invoices')
      .set('Authorization', `Bearer ${validToken}`)

    expect(res.status).not.toBe(401)
  })
})

describe('Role Guard', () => {
```

```
  it('blocks accountant from deleting users', async () => {
    const token = generateToken({ sub: 'user-id', role: 'accountant', orgId: 'org-id' })
    const res = await request(app)
      .delete('/api/v1/users/other-user-id')
      .set('Authorization', `Bearer ${token}`)

    expect(res.status).toBe(403)
    expect(res.body.code).toBe('INSUFFICIENT_PERMISSIONS')
  })
})
```

End of Middleware Documentation

Revision #3
Created 2026-02-23 10:47:53 UTC by John
Updated 2026-05-31 20:02:38 UTC by John