Security Architecture

Bilko — Security Architecture

Status: PLANNED (backend not built yet, security measures documented for implementation)

This document defines the security architecture for Bilko, a financial SaaS handling sensitive accounting data.

Security Principles

1. **Defense in Depth** — Multiple layers of security (network, application, database)
2. **Least Privilege** — Users and services get minimum necessary permissions
3. **Zero Trust** — Verify every request, never assume trust
4. **Encryption Everywhere** — Data encrypted in transit and at rest
5. **Immutable Audit Trail** — All actions logged, tamper-proof

---

STRIDE Threat Model

Bilko handles sensitive financial data (tax IDs, IBAN, accounting records) across three jurisdictions. The STRIDE model identifies threats specific to each layer.

Spoofing — Identity Threats

Threat	Attack Vector	Bilko Risk	Mitigation
JWT token theft	XSS attack extracts access token from memory	HIGH — attacker gains full user session	CSP headers block inline scripts; access token not stored in localStorage
Session hijacking	Refresh token cookie stolen via network or MITM	HIGH — 7-day session takeover	httpOnly + Secure + SameSite=Strict cookie; TLS 1.3 only

Threat	Attack Vector	Bilko Risk	Mitigation
Credential stuffing	Automated login with leaked credentials	HIGH — financial platform targeted	Rate limiting (5 req/15min); bcrypt 12 rounds; HIBP breach check
JWT algorithm confusion	Attacker sends `alg: none` or switches HS256/RS256	MEDIUM	`jsonwebtoken` always specifies algorithm explicitly; RS256 enforced
Account enumeration	Timing attack on login endpoint reveals valid emails	MEDIUM	Constant-time response regardless of email existence

Tampering — Data Integrity Threats

Threat	Attack Vector	Bilko Risk	Mitigation
Invoice amount modification	MITM attack modifies invoice amounts in transit	HIGH — financial fraud	TLS 1.3 for all connections; HSTS with preload
Transaction record alteration	Unauthorized user modifies financial records	CRITICAL — accounting integrity	LoggedAction audit trail (append-only); Prisma soft delete only
JWT payload manipulation	Attacker decodes JWT, changes `role: viewer` to `role: owner`	HIGH — privilege escalation	RS256 signature verification; any modification invalidates signature
Database record tampering	Direct DB access bypasses application	HIGH — data integrity loss	Railway access restricted to CTO only; no public DB port
File upload replacement	Upload modified invoice PDF with different amounts	MEDIUM	File stored by hash; original uploaded by authorized user; audit trail

Repudiation — Non-Traceability Threats

Threat	Attack Vector	Bilko Risk	Mitigation
Audit log bypass	Attacker finds code path that skips LoggedAction	HIGH — undetected fraud	Prisma middleware applies audit to ALL model mutations; test coverage
LoggedAction deletion	Admin or attacker deletes audit records	CRITICAL — compliance violation	LoggedAction has no DELETE permission in RBAC; DB-level row security planned
Timestamp manipulation	System clock skewed to invalidate audit timestamps	LOW	Railway NTP; JWT `iat` verified server-side

Threat	Attack Vector	Bilko Risk	Mitigation
User denies action	"I never deleted that invoice"	MEDIUM	Audit log captures: userId, IP, exact timestamp, old values, new values

Information Disclosure — Data Leakage Threats

Threat	Attack Vector	Bilko Risk	Mitigation
Cross-tenant data leak	Missing `organizationId` WHERE clause on Prisma query	CRITICAL — GDPR breach	Org-scoping middleware on all routes; lint rule + automated isolation tests
Financial data in API errors	Stack trace contains query with financial amounts	HIGH	Production error handler returns only generic message + error ID
Tax ID (JMBG/OIB) exposure	DB breach exposes plaintext personal citizen IDs	CRITICAL — identity theft, irrevocable	AES-256-GCM field-level encryption (Tier 1) via prisma-field-encryption (See ADR-014)
Tax ID (PIB/JIB) exposure	DB breach exposes business tax IDs	LOW — publicly available on APR/UIO portals	Disk-level encryption (Railway AES-256) + org-scoping + RBAC (Tier 2, See ADR-014)
IBAN exposure	DB breach or API response over-returning	MEDIUM — routinely shared for payment	Disk-level encryption + IBAN masked in list responses (last 4 digits only) (See ADR-014)
JWT contains PII	Access token readable by any party	MEDIUM	JWT contains only user ID, org ID, role — no email, name, or financial data
Log file leakage	Application logs contain email addresses or amounts	MEDIUM	Logging policy: never log request body for financial endpoints

Denial of Service — Availability Threats

Threat	Attack Vector	Bilko Risk	Mitigation
Authentication flooding	Brute force login with millions of requests	HIGH	Rate limiting: 5 requests/15min on auth endpoints; Cloudflare DDoS protection
Report generation abuse	Repeated complex report requests exhaust DB	MEDIUM	Rate limiting: 10 requests/15min on `/api/v1/reports/*`; caching layer planned

Threat	Attack Vector	Bilko Risk	Mitigation
File upload flooding	Upload large files repeatedly	MEDIUM	10MB limit; multer request counting; Cloudflare rate limiting at edge
Database connection exhaustion	Many concurrent requests exceed pool size	MEDIUM	Prisma connection pool limits; Railway auto-scaling
Webhook replay flooding	Repeat webhook calls to SEF/FINA integration	LOW	Idempotency keys on e-invoice submissions; webhook signature verification

Elevation of Privilege — Access Control Threats

Threat	Attack Vector	Bilko Risk	Mitigation
RBAC bypass via role tampering	Modify JWT role claim to gain admin access	CRITICAL	RS256 signature; role read from verified JWT payload only
Cross-tenant elevation	Org-1 user accesses Org-2 resources by guessing UUID	HIGH — multi-tenant SaaS	UUID v4 unpredictable; org-scoped WHERE mandatory; 404 (not 403) on cross-org requests
Horizontal privilege escalation	Accountant accesses another user's profile in same org	MEDIUM	Per-user data scoped by `userId`; endpoints check `req.user.id === resource.userId`
API endpoint enumeration	Attacker discovers undocumented admin endpoints	LOW	No hidden admin endpoints; all endpoints in API spec; Cloudflare WAF
Dependency hijacking	Malicious package injected via supply chain	MEDIUM	`package-lock.json` committed; Dependabot; npm audit in CI

Authentication

Strategy: JWT (JSON Web Tokens)

Why JWT?

- Stateless (scales horizontally)
- Works with mobile PWA
- Industry standard

Token Types

Access Token

- **Lifetime:** 15 minutes
- **Storage:** `Authorization: Bearer <token>` header
- **Contains:** User ID, organization ID, role
- **Refresh:** Automatic via refresh token

Refresh Token

- **Lifetime:** 7 days
- **Storage:** httpOnly cookie (not accessible to JavaScript)
- **Purpose:** Obtain new access token
- **Rotation:** New refresh token issued on each refresh
- **Revocation:** Stored in database, can be invalidated

JWT Payload Example

```
{
  "sub": "user-uuid",
  "org": "org-uuid",
  "role": "admin",
  "iat": 1640000000,
  "exp": 1640000900,
  "jti": "unique-token-id"
}
```

> `jti` (JWT ID) — unique token identifier used to prevent replay attacks and enable server-side token invalidation.

Token Flow

```
1. User logs in → POST /api/v1/auth/login
   ← Access token (header) + Refresh token (httpOnly cookie)

2. User makes request → GET /api/v1/invoices (Authorization: Bearer <access>)
   ← Protected resource
```

```
3. Access token expires (15 min) → POST /api/v1/auth/refresh (httpOnly cookie)
   ← New access token + New refresh token

4. User logs out → POST /api/v1/auth/logout
   → Delete refresh token from DB
   ← 204 No Content
```

Implementation (Backend)

```
import jwt from 'jsonwebtoken';
import bcrypt from 'bcrypt';

// Generate access token
const accessToken = jwt.sign(
  { sub: user.id, org: user.organizationId, role: user.role },
  process.env.JWT_SECRET!,
  { expiresIn: '15m' }
);

// Generate refresh token
const refreshToken = jwt.sign(
  { sub: user.id },
  process.env.JWT_REFRESH_SECRET!,
  { expiresIn: '7d' }
);

// Store refresh token in DB (for revocation)
await prisma.refreshToken.create({
  data: {
    token: refreshToken,
    userId: user.id,
    expiresAt: new Date(Date.now() + 7 * 24 * 60 * 60 * 1000),
  },
});
```

Token Invalidation Events

Refresh tokens must be revoked server-side on any of these events:

- User logout
- Password change
- Role change by admin
- Account suspension
- Suspicious login from unknown IP/country

Password Security

Hashing: bcrypt

Algorithm: bcrypt with 12 salt rounds

Why bcrypt?

- Designed for passwords (slow by design, resists brute force)
- Auto-salted (each password has unique salt)
- Adaptive (can increase rounds as hardware improves)

Password Requirements

- **Minimum length:** 8 characters
- **Complexity:** At least one uppercase, one lowercase, one number
- **No common passwords:** Check against list of 10K most common passwords
- **No reuse:** Previous 5 passwords stored (hashed) and blocked

Implementation

```
import bcrypt from 'bcrypt';


// Hash password (registration)
const passwordHash = await bcrypt.hash(password, 12);


// Verify password (login)
const isValid = await bcrypt.compare(password, user.passwordHash);
```

Two-Factor Authentication (2FA)

Strategy: TOTP (Time-based One-Time Password)

Compatible with:

- Google Authenticator
- Authy
- 1Password
- Microsoft Authenticator

Setup Flow

```
1. User enables 2FA → POST /api/v1/auth/2fa/setup
   ← QR code + secret (base32)

2. User scans QR code in authenticator app
   → Generates 6-digit code

3. User verifies code → POST /api/v1/auth/2fa/verify { code }
   ← 200 OK (2FA enabled)
```

Login Flow with 2FA

```
1. User logs in → POST /api/v1/auth/login { email, password }
   ← 200 OK + { requires2FA: true, tempToken }

2. User enters code → POST /api/v1/auth/2fa/login { tempToken, code }
   ← Access token + Refresh token
```

Backup Codes

Generate 10 single-use backup codes during 2FA setup:

- Stored hashed (bcrypt)
- Used when authenticator unavailable
- Marked as used after redemption

Authorization (RBAC)

Roles

Role	Permissions
owner	Full access (edit org settings, invite users, delete data)
admin	Manage invoices, expenses, contacts, reports (no org settings)
accountant	Read invoices/expenses, create reports (no edit)
viewer	Read-only access (dashboard, reports)

Permission Matrix

Action	owner	admin	accountant	viewer
Create invoice	□	□	□	□
Edit invoice	□	□	□	□
Delete invoice	□	□	□	□
View invoice	□	□	□	□
Approve expense	□	□	□	□
Generate report	□	□	□	□
Invite user	□	□	□	□
Edit org settings	□	□	□	□

Implementation (Middleware)

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

function requireRole(roles: string[]) {
  return (req: Request, res: Response, next: NextFunction) => {
    if (!roles.includes(req.user.role)) {
      return res.status(403).json({ error: 'Forbidden' });
    }
    next();
  };
}
```

```
// Usage
app.post('/api/v1/invoices', requireRole(['owner', 'admin']), createInvoice);
```

Data Classification

Level	Label	Examples	Controls
L4-A	Restricted (Personal)	JMBG, OIB	AES-256-GCM field-level encryption (prisma-field-encryption) + HMAC-SHA256 hash columns + access log (See ADR-014)
L4-B	Restricted (Business/Financial)	PIB, JIB, IBAN	Disk-level encryption (Railway AES-256) + TLS 1.3 + org-scoping + RBAC + API masking for IBAN (last 4 digits) (See ADR-014)
L3	Confidential	Financial amounts, bank statements, invoices	Org-scoped access, TLS, PostgreSQL AES-256 at rest
L2	Internal	Email, name, address, phone	TLS, authenticated access only
L1	Public	Organization name, public invoice reference	No special controls

L4 Restricted fields use a hybrid encryption approach per ADR-014: personal identifiers (JMBG, OIB) receive AES-256-GCM field-level encryption before persistence because they are irrevocable and high-impact on breach. Business tax IDs (PIB, JIB) and IBAN rely on disk-level encryption plus application-layer controls — field-level encryption for publicly available identifiers would be disproportionate to the risk per GDPR Article 32.

Encryption

In Transit: TLS 1.3

All traffic encrypted via HTTPS:

- Frontend (Vercel): Automatic HTTPS
- Backend (Railway): Automatic HTTPS

- Certificate: Let's Encrypt (auto-renewed)

TLS Configuration:

- Minimum version: TLS 1.3
- Cipher suites: Modern only (no legacy ciphers)
- HSTS enabled (Strict-Transport-Security header)

At Rest: Database Encryption

PostgreSQL (Railway):

- Disk encryption: AES-256 (Railway default)
- Backup encryption: AES-256
- Column-level encryption: **Hybrid approach per ADR-014** — JMBG and OIB fields use AES-256-GCM field-level encryption via `prisma-field-encryption` (Tier 1). PIB, JIB, and IBAN rely on disk-level encryption + application controls (Tier 2). Disk-level encryption alone is insufficient for personal identifiers (JMBG/OIB) due to their irrevocability and high breach impact. (See ADR-014)

Cloudflare R2 (Files):

- Server-side encryption: AES-256 (default)
- No client-side encryption needed (files are receipts/invoices, not PII)

Secrets Management

NEVER commit secrets to git:

- `.env` files in `.gitignore`
- Use platform-provided secrets (Vercel, Railway)
- Rotate JWT secrets quarterly
- Rotate API keys annually

OWASP Top 10 Mitigations

1. Injection (SQL Injection)

Mitigation: Prisma ORM parameterized queries

```
// SAFE — Prisma auto-escapes
await prisma.invoice.findMany({
  where: { customerId: req.params.id }
});

// UNSAFE — Never use raw SQL for user input
await prisma.$queryRaw`SELECT * FROM invoices WHERE customer_id = ${req.params.id}`;
```

---

2. Broken Authentication

Mitigations:

- bcrypt password hashing (12 rounds)
- JWT with short expiry (15 min)
- Refresh token rotation
- 2FA (TOTP)
- Rate limiting on auth endpoints (5 req/min)

---

3. Sensitive Data Exposure

Mitigations:

- TLS 1.3 in transit
- AES-256 at rest
- No PII in JWTs (only user ID)
- No passwords in logs
- No sensitive data in URLs (use POST body)

---

4. XML External Entities (XXE)

Not applicable — Bilko does not parse XML.

---

5. Broken Access Control

Mitigations:

- RBAC enforced on every endpoint
- Organization-scoped queries (middleware)

- No direct object reference (use UUIDs, not auto-increment IDs)

```
// Organization scoping middleware
app.use('/api/v1/*', (req, res, next) => {
  req.prismaWhere = { organizationId: req.user.organizationId };
  next();
});

// Apply to queries
await prisma.invoice.findMany({ where: req.prismaWhere });
```

---

6. Security Misconfiguration

Mitigations:

- Helmet.js security headers
- CORS whitelist (no `*` in production)
- Error messages sanitized (no stack traces in production)
- Disable `X-Powered-By` header

Full Security Headers Configuration

All security headers applied via Helmet.js on the Express API. The Next.js frontend applies equivalent headers via `next.config.js`.

```
import helmet from 'helmet';

// Express API — full security headers
app.use(helmet({
  // Content-Security-Policy — prevent XSS
  contentSecurityPolicy: {
    directives: {
      defaultSrc: ["'self'"],
      scriptSrc: ["'self'"],            // No unsafe-inline needed on API
      styleSrc: ["'self'"],
      imgSrc: ["'self'", "data:"],
      connectSrc: ["'self'"],
      frameSrc: ["'none'"],             // No iframes from this API
      objectSrc: ["'none'"],
      upgradeInsecureRequests: [],
```

```
    },
    useDefaults: false,
  },
  // Strict-Transport-Security — force HTTPS for 1 year, include subdomains
  hsts: {
    maxAge: 31536000,                  // 1 year in seconds
    includeSubDomains: true,
    preload: true,                     // Eligible for browser HSTS preload list
  },
  // X-Frame-Options — prevent clickjacking
  frameguard: {
    action: 'deny',                    // DENY: no framing at all
  },
  // X-Content-Type-Options — prevent MIME sniffing
  noSniff: true,
  // X-XSS-Protection — legacy header for older browsers
  xssFilter: true,
  // Referrer-Policy — don't leak URL in Referer header
  referrerPolicy: {
    policy: 'strict-origin-when-cross-origin',
  },
  // Permissions-Policy — disable browser features not needed by Bilko
  permittedCrossDomainPolicies: { permittedPolicies: 'none' },
  // X-DNS-Prefetch-Control
  dnsPrefetchControl: { allow: false },
  // X-Powered-By removed by default in Helmet
  hidePoweredBy: true,
}));

// Permissions-Policy header (not yet in Helmet — set manually)
app.use((req, res, next) => {
  res.setHeader(
    'Permissions-Policy',
    'camera=(), microphone=(), geolocation=(), payment=(), usb=()'
  );
  next();
});

// CORS — whitelist only known origins
app.use(cors({
```

```
  origin: (origin, callback) => {
    const allowed = [
      'https://bilko.io',
      'https://www.bilko.io',
      'https://app.bilko.io',
      'https://staging.bilko.io',
      'https://bilko.rs',     // Serbia redirect domain
    ];
    if (!origin || allowed.includes(origin)) {
      callback(null, true);
    } else {
      callback(new Error(`CORS: origin ${origin} not allowed`));
    }
  },
  credentials: true,                    // Required for httpOnly cookie (refresh token)
  methods: ['GET', 'POST', 'PUT', 'PATCH', 'DELETE', 'OPTIONS'],
  allowedHeaders: ['Content-Type', 'Authorization'],
}));
```

Next.js Frontend Security Headers (`next.config.js`)

```
// next.config.js
const securityHeaders = [
  {
    key: 'Content-Security-Policy',
    value: [
      "default-src 'self'",
      "script-src 'self' 'unsafe-inline' 'unsafe-eval'",  // Next.js requires these
      "style-src 'self' 'unsafe-inline'",                  // Tailwind requires unsafe-inline
      "img-src 'self' data: https:",
      "connect-src 'self' https://api.bilko.io wss://api.bilko.io",
      "font-src 'self' https://fonts.gstatic.com",
      "frame-ancestors 'none'",
      "upgrade-insecure-requests",
    ].join('; '),
  },
  { key: 'Strict-Transport-Security', value: 'max-age=31536000; includeSubDomains; preload' },
  { key: 'X-Frame-Options', value: 'DENY' },
  { key: 'X-Content-Type-Options', value: 'nosniff' },
  { key: 'Referrer-Policy', value: 'strict-origin-when-cross-origin' },
```

```
  { key: 'Permissions-Policy', value: 'camera=(), microphone=(), geolocation=(), payment=()'
},
  { key: 'X-DNS-Prefetch-Control', value: 'off' },
];

module.exports = {
  headers: async () => [
    { source: '/:path*', headers: securityHeaders },
  ],
};
```

Headers Verification

Use securityheaders.com to verify. Target grade: **A+**.

Header	Expected Value	Purpose
`Content-Security-Policy`	Restrictive directives	Prevent XSS
`Strict-Transport-Security`	`max-age=31536000; includeSubDomains; preload`	Force HTTPS
`X-Frame-Options`	`DENY`	Prevent clickjacking
`X-Content-Type-Options`	`nosniff`	Prevent MIME sniffing
`Referrer-Policy`	`strict-origin-when-cross-origin`	Limit referrer leakage
`Permissions-Policy`	Disable camera/mic/geo/payment	Minimize attack surface

---

7. Cross-Site Scripting (XSS)

Mitigations:

- React auto-escapes output (default safe)
- CSP headers (Content-Security-Policy)
- Sanitize user input (Zod validation)
- No `dangerouslySetInnerHTML` without sanitization

```
// SAFE — React escapes by default
<p>{invoice.description}</p>

// UNSAFE — Only use with sanitized HTML
<div dangerouslySetInnerHTML={{ __html: sanitizedHTML }} />
```

---

8. Insecure Deserialization

Not applicable — Bilko does not deserialize untrusted data.

---

9. Using Components with Known Vulnerabilities

Mitigations:

- Dependabot alerts enabled (GitHub)
- Weekly `npm audit` checks
- Automated security updates (Dependabot PRs)
- Lock file committed (`package-lock.json`)

---

10. Insufficient Logging & Monitoring

Mitigations:

- Audit trail (LoggedAction table)
- Error tracking (Sentry recommended)
- Access logs (Railway built-in)
- Failed login attempts logged
- Anomaly detection (future: alert on 10+ failed logins)

Rate Limiting

Prevent brute force and abuse:

Endpoint	Limit	Window	Rationale
`/api/v1/auth/login`	5 requests	15 minutes	Prevent credential stuffing
`/api/v1/auth/register`	3 requests	60 minutes	Prevent bulk account creation
`/api/v1/auth/refresh`	10 requests	15 minutes	Prevent refresh token flood
`/api/v1/auth/2fa/login`	5 requests	15 minutes	Prevent TOTP brute force
`/api/v1/auth/forgot-password`	3 requests	60 minutes	Prevent email enumeration via flood
`/api/v1/*` (general)	100 requests	15 minutes	General API protection

Endpoint	Limit	Window	Rationale
`/api/v1/reports/*`	10 requests	15 minutes	Prevent expensive query abuse
`/api/v1/*/export`	5 requests	60 minutes	Prevent bulk data export

Implementation

```
import rateLimit from 'express-rate-limit';
import RedisStore from 'rate-limit-redis';

// Auth limiter — strict
const authLimiter = rateLimit({
  windowMs: 15 * 60 * 1000,            // 15 minutes
  max: 5,
  standardHeaders: true,
  legacyHeaders: false,
  message: { error: 'Too many attempts. Try again in 15 minutes.' },
  // Use IP + email combination as key to prevent distributed attacks
  keyGenerator: (req) => `${req.ip}:${req.body?.email ?? 'unknown'}`,
});

// General API limiter
const generalLimiter = rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 100,
  standardHeaders: true,
  legacyHeaders: false,
  skip: (req) => isWebhookRequest(req), // Webhooks bypass general limiter
});

app.post('/api/v1/auth/login', authLimiter, loginHandler);
app.use('/api/v1/', generalLimiter);
```

IP Whitelisting for Webhooks

Webhooks from SEF (Serbian e-invoice portal) and FINA (Croatian HR-FISK) must bypass general rate limiting but are restricted to known IP ranges:

```
// Known webhook source IP ranges
const SEF_WEBHOOK_IPS = [
  '185.54.144.0/24',   // efaktura.mfin.gov.rs — verify with SEF portal docs
];

const FINA_WEBHOOK_IPS = [
  '195.29.61.0/24',    // FINA PKI infrastructure — verify with FINA
];

function isWebhookRequest(req: Request): boolean {
  const clientIp = req.ip ?? req.socket.remoteAddress;
  return [...SEF_WEBHOOK_IPS, ...FINA_WEBHOOK_IPS].some(
    (range) => ipRangeContains(range, clientIp)
  );
}

// Webhook endpoint — IP-restricted, no general rate limit
app.post('/api/v1/webhooks/sef',
  requireWebhookIp(SEF_WEBHOOK_IPS),
  verifyWebhookSignature,
  handleSefWebhook
);

app.post('/api/v1/webhooks/fina',
  requireWebhookIp(FINA_WEBHOOK_IPS),
  verifyWebhookSignature,
  handleFinaWebhook
);

function requireWebhookIp(allowedRanges: string[]) {
  return (req: Request, res: Response, next: NextFunction) => {
    const clientIp = req.ip ?? req.socket.remoteAddress;
    const allowed = allowedRanges.some((range) => ipRangeContains(range, clientIp));
    if (!allowed) {
      return res.status(403).json({ error: 'Webhook source IP not allowed' });
    }
    next();
  };
}
```

Note: Confirm exact SEF and FINA IP ranges from their integration documentation before deployment. Update `SEF_WEBHOOK_IPS` and `FINA_WEBHOOK_IPS` accordingly.

Input Validation

All inputs validated with **Zod** schemas:

Example: Invoice Validation

```
import { z } from 'zod';

const createInvoiceSchema = z.object({
  customerId: z.string().uuid(),
  invoiceDate: z.string().regex(/^\d{4}-\d{2}-\d{2}$/),
  dueDate: z.string().regex(/^\d{4}-\d{2}-\d{2}$/),
  currencyCode: z.enum(['EUR', 'RSD', 'BAM', 'HRK']),
  items: z.array(z.object({
    description: z.string().min(1).max(500),
    quantity: z.number().positive(),
    unitPrice: z.number().nonnegative(),
    taxRate: z.number().min(0).max(100),
  })),
});

// Middleware
function validate(schema: z.ZodSchema) {
  return (req, res, next) => {
    try {
      req.body = schema.parse(req.body);
      next();
    } catch (error) {
      res.status(400).json({ error: error.errors });
    }
  };
}

// Usage
app.post('/api/v1/invoices', validate(createInvoiceSchema), createInvoice);
```

File Upload Security

Allowed File Types

- **Receipts:** JPG, PNG, PDF
- **Max size:** 10MB per file

Validation

```
import multer from 'multer';
import path from 'path';

const upload = multer({
  limits: { fileSize: 10 * 1024 * 1024 }, // 10MB
  fileFilter: (req, file, cb) => {
    const allowedTypes = ['.jpg', '.jpeg', '.png', '.pdf'];
    const ext = path.extname(file.originalname).toLowerCase();
    if (allowedTypes.includes(ext)) {
      cb(null, true);
    } else {
      cb(new Error('Invalid file type'));
    }
  },
});
```

Virus Scanning (Planned)

Phase 2: Integrate ClamAV for virus scanning before upload to R2.

Audit Trail

LoggedAction Table (Immutable)

All mutations logged:

- Table name
- Action (INSERT, UPDATE, DELETE)
- User ID
- Timestamp
- Old values (UPDATE/DELETE)
- New values (INSERT/UPDATE)
- Client IP
- SQL query

Example Audit Log Entry

```
{
  "eventId": 12345,
  "tableName": "invoices",
  "action": "UPDATE",
  "userId": "user-uuid",
  "actionTimestamp": "2026-02-20T10:30:00Z",
  "rowData": { "id": "invoice-uuid", "status": "draft" },
  "changedFields": { "status": { "old": "draft", "new": "sent" } },
  "clientIp": "192.168.1.10"
}
```

Audit Queries

```
// Get user activity
await prisma.loggedAction.findMany({
  where: { userId: 'user-uuid' },
  orderBy: { actionTimestamp: 'desc' },
  take: 100,
});

// Get invoice history
await prisma.loggedAction.findMany({
  where: {
    tableName: 'invoices',
    rowData: { path: ['id'], equals: 'invoice-uuid' },
  },
});
```

Data Retention & Deletion

User Data Deletion (GDPR Right to Erasure)

Process:

1. User requests deletion → POST /api/v1/account/delete
2. Soft delete user record (mark `deletedAt`)
3. Anonymize LoggedAction entries (replace user ID with "deleted-user")
4. Delete PII (email, name)
5. **Keep financial records** (required by law, minimum 5 years)

Soft Delete Implementation:

```
await prisma.user.update({
  where: { id: userId },
  data: {
    email: `deleted-${userId}@example.com`,
    fullName: 'Deleted User',
    passwordHash: '',
    deletedAt: new Date(),
  },
});
```

Security Testing

Static Analysis

- **ESLint:** Security rules enabled (no-eval, no-unsafe-regex)
- **TypeScript:** Strict mode (catches type errors)

Dependency Scanning

- **npm audit:** Weekly checks
- **Dependabot:** Automatic PRs for vulnerabilities

Penetration Testing Plan

Frequency: Annual + after significant architecture changes **Provider:** External certified firm (OSCP or CREST certified) **Environment:** Staging only (`staging.bilko.io`) — **never production without explicit CEO approval**

Scope

Area	Priority	Test Approach
Authentication & session management	P0 — Critical	JWT tampering, refresh token theft, brute force, 2FA bypass
Multi-tenant data isolation	P0 — Critical	Cross-org data access, IDOR on UUIDs, query manipulation
RBAC & privilege escalation	P1 — High	Role tampering, horizontal escalation, missing authorization
API security	P1 — High	All endpoints for injection, auth bypass, mass assignment
Financial data protection	P1 — High	Encrypted field bypass, IBAN/tax ID extraction
File upload security	P2 — Medium	Malicious file upload, path traversal, SSRF
Third-party integrations	P2 — Medium	SEF / FINA webhook manipulation, replay attacks
Business logic	P2 — Medium	Invoice amount manipulation, VAT calculation errors, status bypass

Pre-Engagement Checklist

- ☐ Statement of Work (SoW) signed with pentest firm
- ☐ Staging environment set up with production-equivalent configuration
- ☐ Test data (fake organizations, fake invoices) loaded — **no real customer data**
- ☐ Bilko DBA grants read-only DB access to pentest firm for review (not write)
- ☐ Confirm staging SEF/FINA integrations are in test mode
- ☐ Legal: pentest authorization letter from CEO on file

Acceptance Criteria

- **Zero Critical or High findings** unmitigated before production launch
- **Medium findings**: each assessed, documented, and either fixed or accepted with justification
- **Remediation SLAs:**
  - CRITICAL: 48 hours
  - HIGH: 7 days
  - MEDIUM: 30 days
  - LOW: Next sprint boundary

Pentest Report Requirements

The pentest report must include:

1. Executive summary (risk level, critical findings)
2. Technical findings: CVSS score, proof of concept, affected endpoints
3. Business impact statement for each finding
4. Remediation recommendations
5. Re-test results (confirm fixes for Critical and High)

Incident Response Plan

Detection

- Monitor error rates (Sentry)
- Monitor failed login attempts (>10 in 1 hour = alert)
- Railway metrics (CPU spike, memory leak)

Response

1. **Identify:** What is the breach? (data leak, DDoS, unauthorized access)
2. **Contain:** Block attacker IP, revoke compromised tokens
3. **Eradicate:** Fix vulnerability, patch code
4. **Recover:** Restore from backup if needed
5. **Document:** Write post-mortem, update security docs

Notification

- **Internal:** Slack alert to #security channel
- **External:** Email users if PII compromised (GDPR 72h requirement)

Security Checklist (Pre-Launch)

- ☐ JWT secrets generated (32+ chars)
- ☐ HTTPS enforced (no HTTP allowed)
- ☐ CORS whitelist configured (no `*`)
- ☐ Rate limiting enabled (auth endpoints)

- [ ] Helmet.js security headers configured
- [ ] bcrypt password hashing (12 rounds)
- [ ] Prisma queries parameterized (no raw SQL)
- [ ] Input validation (Zod schemas)
- [ ] File upload restrictions (type, size)
- [ ] Audit trail enabled (LoggedAction)
- [ ] Error messages sanitized (no stack traces)
- [ ] Dependabot alerts enabled
- [ ] Backup strategy tested
- [ ] Incident response plan documented
- [ ] Security review completed

Related Documents

- Compliance: COMPLIANCE.md
- Deployment: ../infrastructure/DEPLOYMENT.md
- Testing: ../testing/TESTING-GUIDE.md

Last Updated: 2026-02-20 **Status:** PLANNED — Backend not built yet, security measures to be implemented **Compliance:** OWASP Top 10, GDPR Article 32 (Security of Processing)

Revision #4
Created 2026-02-24 22:50:52 UTC by John
Updated 2026-05-31 20:03:55 UTC by John