External Services Integration

Bilko External Services

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

Purpose

This document specifies the external service integrations for Bilko's backend. Covers email delivery, file storage, exchange rates, and PDF generation.

Service Integration Architecture

```
graph TD
    subgraph BILKO [Bilko Backend — apps/api]
        API[Express API\nPort 4000]
        PDF[PDF Service\nPuppeteer]
        RATE[Exchange Rate Service\nCron: daily 00:00 UTC]
        AUDIT[Prisma Middleware\nAudit Logger]
    end

    subgraph EXTERNAL [External Services]
        SG[SendGrid\nnoreply@bilko.io\n100 emails/day free]
        R2[Cloudflare R2\nbilko-files bucket\n10GB free]
        ECB[ECB API\nexchangerate.host\nFree, daily rates]
        FIXER[Fixer.io\n100 req/month free\nFallback]
        CLAM[ClamAV\nVirus Scanner\nlocalhost:3310]
    end

    subgraph STORAGE [Database]
```

```
        PG[PostgreSQL 14+\nAll data\nExchangeRate cache]
    end

    API -->|Invoice email + PDF| SG
    API -->|Receipt upload| R2
    PDF -->|Store generated PDF| R2
    RATE -->|Primary rates fetch| ECB
    RATE -->|Fallback if ECB fails| FIXER
    ECB & FIXER -->|Store in DB| PG
    API -->|Scan uploaded files| CLAM
    API -->|All reads/writes| PG
    AUDIT -->|Append-only log| PG

    style SG fill:#00b0f0,color:#fff
    style R2 fill:#f6821f,color:#fff
    style ECB fill:#0070f3,color:#fff
    style FIXER fill:#6366f1,color:#fff
    style PG fill:#336791,color:#fff
    style CLAM fill:#e53e3e,color:#fff
```

Table of Contents

---

1. SendGrid (Email Delivery)

Purpose

Send invoice emails, payment reminders, user invitations, and password resets.

Invoice Email + Tracking Flow

```
sequenceDiagram
    participant API as Bilko API
    participant PDF as PDF Service\n(Puppeteer)
    participant R2 as Cloudflare R2
    participant SG as SendGrid
    participant CUSTOMER as Customer Email

    Note over API,CUSTOMER: Invoice Send Flow
    API->>PDF: generateInvoicePDF(invoiceId)
    PDF->>PDF: Launch headless Chromium\nRender HTML template\nExport A4 PDF
    PDF-->>API: PDF Buffer
    API->>R2: PUT invoices/{orgId}/INV-2026-001.pdf
    R2-->>API: Public URL stored
    API->>API: Update invoice.pdfUrl
    API->>SG: sendEmail({ to, subject, html, attachment:pdf })
    SG-->>API: { messageId }
    API->>API: Update invoice.sentAt, status='sent'
    SG->>CUSTOMER: Deliver email with PDF attachment
    CUSTOMER->>API: GET /track/email/{invoiceId}\n[1x1 pixel load]
    API->>API: UPDATE invoice SET status='viewed'\nviewedAt=now()
```

Setup

Account: SendGrid (free tier: 100 emails/day)

Installation:

```
npm install @sendgrid/mail
```

Environment Variables:

```
SENDGRID_API_KEY=SG.xxxxx
SENDGRID_FROM_EMAIL=noreply@bilko.io
SENDGRID_FROM_NAME=Bilko
```

Configuration

```
import sgMail from '@sendgrid/mail'

sgMail.setApiKey(process.env.SENDGRID_API_KEY!)

interface SendEmailOptions {
  to: string | string[]
  cc?: string[]
  bcc?: string[]
  subject: string
  text: string
  html: string
  attachments?: Array<{
    content: string      // Base64 encoded
    filename: string
    type: string         // MIME type
    disposition: 'attachment' | 'inline'
  }>
}

async function sendEmail(options: SendEmailOptions) {
  const msg = {
    to: options.to,
    cc: options.cc,
    bcc: options.bcc,
    from: {
      email: process.env.SENDGRID_FROM_EMAIL!,
      name: process.env.SENDGRID_FROM_NAME!
    },
    subject: options.subject,
    text: options.text,
    html: options.html,
    attachments: options.attachments
  }

  try {
    const response = await sgMail.send(msg)
    return {
      success: true,
      messageId: response[0].headers['x-message-id']
```

```
    }
  } catch (error) {
    logger.error('SendGrid error:', error)
    throw new Error('Failed to send email')
  }
}
```

Email Templates

1. Invoice Email

Template variables:

- `{{ organizationName }}`
- `{{ invoiceNumber }}`
- `{{ customerName }}`
- `{{ totalAmount }}`
- `{{ currencyCode }}`
- `{{ dueDate }}`
- `{{ viewInvoiceUrl }}`

HTML template:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Invoice {{ invoiceNumber }}</title>
  <style>
    body { font-family: Inter, sans-serif; }
    .container { max-width: 600px; margin: 0 auto; padding: 20px; }
    .header { background: #09090b; color: #fff; padding: 20px; text-align: center; }
    .content { padding: 20px; background: #f9fafb; }
    .footer { padding: 20px; text-align: center; color: #6b7280; }
    .button { display: inline-block; padding: 12px 24px; background: #00E5A0; color: #000;
text-decoration: none; border-radius: 6px; }
  </style>
</head>
<body>
  <div class="container">
    <div class="header">
```

```
      <h1>{{ organizationName }}</h1>
    </div>
    <div class="content">
      <p>Dear {{ customerName }},</p>
      <p>Your invoice <strong>{{ invoiceNumber }}</strong> is ready.</p>
      <table>
        <tr><td>Amount:</td><td><strong>{{ totalAmount }} {{ currencyCode
}}</strong></td></tr>
        <tr><td>Due Date:</td><td>{{ dueDate }}</td></tr>
      </table>
      <p><a href="{{ viewInvoiceUrl }}" class="button">View Invoice</a></p>
      <p>Thank you for your business!</p>
    </div>
    <div class="footer">
      <p>Powered by <a href="https://bilko.io">Bilko</a></p>
    </div>
  </div>
  <!-- Tracking pixel -->
  <img src="{{ trackingPixelUrl }}" width="1" height="1" />
</body>
</html>
```

Attachment: Invoice PDF (generated via PDF service)

2. User Invitation Email

Template variables:

- `{{ organizationName }}`
- `{{ inviterName }}`
- `{{ inviteLink }}`
- `{{ role }}`

Subject: `{{ inviterName }} invited you to {{ organizationName }} on Bilko`

3. Password Reset Email

Template variables:

- `{{ resetLink }}`
- `{{ expiresIn }}` (e.g., "15 minutes")

Subject: `Reset your Bilko password`

Email Tracking

Purpose: Track when customer views invoice email (for `viewedAt` timestamp).

How it works:

1. Embed 1x1 transparent pixel in email HTML
2. Pixel URL: `https://api.bilko.io/track/email/{{ invoiceId }}`
3. When customer opens email, browser loads pixel
4. Backend endpoint logs view:

```
app.get('/track/email/:invoiceId', async (req, res) => {
  const { invoiceId } = req.params

  await prisma.invoice.update({
    where: { id: invoiceId },
    data: {
      status: 'viewed',
      viewedAt: new Date()
    }
  })

  // Return 1x1 transparent GIF
  const pixel = Buffer.from(
    'R0lGODlhAQABAIAAAAAAAP///yH5BAEAAAAALAAAAAABAAEAAAIBRAA7',
    'base64'
  )
  res.writeHead(200, {
    'Content-Type': 'image/gif',
    'Content-Length': pixel.length
  })
  res.end(pixel)
})
```

Rate Limits

SendGrid free tier:

- 100 emails/day
- 40,000 emails first 30 days
- After 30 days: $0.00025/email

Recommendation for MVP: Free tier sufficient for testing. Upgrade to paid plan at launch.

---

2. Cloudflare R2 (File Storage)

Purpose

Store invoice PDFs and expense receipts.

Why R2 over S3:

- S3-compatible API (easy migration)
- Zero egress fees (S3 charges $0.09/GB)
- Cheaper storage: $0.015/GB (vs S3 $0.023/GB)

Setup

Account: Cloudflare (free tier: 10GB storage)

Installation:

```
npm install @aws-sdk/client-s3
npm install @aws-sdk/s3-request-presigner
```

Environment Variables:

```
R2_ACCOUNT_ID=your-account-id
R2_ACCESS_KEY_ID=your-access-key
R2_SECRET_ACCESS_KEY=your-secret-key
R2_BUCKET_NAME=bilko-files
R2_PUBLIC_URL=https://r2.bilko.io
```

Configuration

```
import { S3Client, PutObjectCommand, GetObjectCommand } from '@aws-sdk/client-s3'
import { getSignedUrl } from '@aws-sdk/s3-request-presigner'

const s3 = new S3Client({
  region: 'auto',
```

```
  endpoint: `https://${process.env.R2_ACCOUNT_ID}.r2.cloudflarestorage.com`,
  credentials: {
    accessKeyId: process.env.R2_ACCESS_KEY_ID!,
    secretAccessKey: process.env.R2_SECRET_ACCESS_KEY!
  }
})

interface UploadFileOptions {
  key: string             // File path (e.g., "invoices/INV-2026-001.pdf")
  body: Buffer | Uint8Array
  contentType: string     // MIME type
  metadata?: Record<string, string>
}

async function uploadFile(options: UploadFileOptions): Promise<string> {
  const command = new PutObjectCommand({
    Bucket: process.env.R2_BUCKET_NAME!,
    Key: options.key,
    Body: options.body,
    ContentType: options.contentType,
    Metadata: options.metadata
  })

  await s3.send(command)

  // Return public URL
  return `${process.env.R2_PUBLIC_URL}/${options.key}`
}

async function getSignedDownloadUrl(key: string, expiresIn: number = 3600): Promise<string> {
  const command = new GetObjectCommand({
    Bucket: process.env.R2_BUCKET_NAME!,
    Key: key
  })

  return getSignedUrl(s3, command, { expiresIn })
}
```

File Organization

Bucket structure:

```
bilko-files/
├── invoices/
│   ├── org-uuid-1/
│   │   ├── INV-2026-001.pdf
│   │   └── INV-2026-002.pdf
│   └── org-uuid-2/
│       └── INV-2026-001.pdf
├── receipts/
│   ├── org-uuid-1/
│   │   ├── EXP-2026-001.jpg
│   │   └── EXP-2026-002.pdf
│   └── org-uuid-2/
└── exports/
    └── org-uuid-1/
        └── report-2026-02-20.xlsx
```

Key format: `{category}/{organizationId}/{filename}`

File Upload Workflow

Invoice PDF:

```
async function storeInvoicePDF(invoiceId: string, organizationId: string, pdf: Buffer):
Promise<string> {
  const invoice = await prisma.invoice.findUnique({ where: { id: invoiceId } })
  const filename = `${invoice.invoiceNumber}.pdf`
  const key = `invoices/${organizationId}/${filename}`

  const url = await uploadFile({
    key,
    body: pdf,
    contentType: 'application/pdf',
    metadata: {
      invoiceId,
      organizationId,
      uploadedAt: new Date().toISOString()
    }
  })
```

```
  await prisma.invoice.update({
    where: { id: invoiceId },
    data: { pdfUrl: url }
  })

  return url
}
```

Expense Receipt:

```
async function storeExpenseReceipt(expenseId: string, organizationId: string, file:
Express.Multer.File): Promise<string> {
  const expense = await prisma.expense.findUnique({ where: { id: expenseId } })
  const ext = file.mimetype.split('/')[1]  // 'pdf', 'jpeg', 'png'
  const filename = `${expense.expenseNumber}.${ext}`
  const key = `receipts/${organizationId}/${filename}`

  const url = await uploadFile({
    key,
    body: file.buffer,
    contentType: file.mimetype,
    metadata: {
      expenseId,
      organizationId,
      uploadedAt: new Date().toISOString()
    }
  })

  await prisma.expense.update({
    where: { id: expenseId },
    data: { receiptUrl: url }
  })

  return url
}
```

Security

1. Signed URLs for private files:

- Invoice PDFs: public (anyone with URL can view)
- Expense receipts: private (require signed URL)
- Signed URLs expire in 1 hour

2. Virus scanning:

- Use ClamAV to scan uploaded files
- Reject if virus detected

```
npm install clamscan
```

```
import NodeClam from 'clamscan'

const clam = await new NodeClam().init({
  clamdscan: {
    host: 'localhost',
    port: 3310
  }
})

async function scanFile(filePath: string): Promise<boolean> {
  const { isInfected } = await clam.scanFile(filePath)
  return !isInfected
}
```

---

3. Exchange Rate APIs

Purpose

Fetch daily exchange rates for multi-currency support.

Exchange Rate Fetch & Fallback Flow

```
flowchart TD
    CRON[Cron Job\nDaily at 00:00 UTC]
    CRON --> ECB[Fetch from ECB API\nexchangerate.host/latest?base=EUR]

    ECB --> ECB_OK{Success?}
```

```
    ECB_OK -->|Yes| STORE[Upsert ExchangeRate records\nsource='ECB']
    ECB_OK -->|No| FIXER[Fallback: Fixer.io\napi.fixer.io/latest]

    FIXER --> FIX_OK{Success?}
    FIX_OK -->|Yes| STORE2[Upsert ExchangeRate records\nsource='fixer.io']
    FIX_OK -->|No| PREV[Use yesterday's rates\nWarn in logs]

    STORE & STORE2 & PREV --> AVAIL[Rates available in DB\nfor transaction date locking]

    subgraph LOOKUP [Rate Lookup at Transaction Time]
        L1[getExchangeRate\nbaseCurrency, targetCurrency, date]
        L2{Same currency?}
        L3[Return rate = 1.0]
        L4[Find exact date in DB]
        L5{Found?}
        L6[Return rate]
        L7[Find nearest available date\norderBy effectiveDate DESC]
        L8[Warn in logs\nReturn nearest rate]

        L1 --> L2
        L2 -->|Yes| L3
        L2 -->|No| L4
        L4 --> L5
        L5 -->|Yes| L6
        L5 -->|No| L7
        L7 --> L8
    end

    AVAIL --> LOOKUP
    style PREV fill:#fb923c,color:#000
    style L8 fill:#fb923c,color:#000
```

Primary: European Central Bank (ECB)

Endpoint: `https://api.exchangerate.host/latest`

Free: Yes (unlimited)

Example:

```
async function fetchECBRates(baseCurrency: string = 'EUR'): Promise<Record<string, number>> {
  const url = `https://api.exchangerate.host/latest?base=${baseCurrency}`

  const response = await fetch(url)
  const data = await response.json()

  if (!data.success) {
    throw new Error('ECB API error')
  }

  return data.rates  // { RSD: 117.50, BAM: 1.95, HRK: 7.53, USD: 1.07 }
}
```

Fallback: Fixer.io

Endpoint: `https://api.fixer.io/latest`

Free tier: 100 requests/month

Environment Variables:

```
FIXER_API_KEY=your-api-key
```

Example:

```
async function fetchFixerRates(baseCurrency: string = 'EUR'): Promise<Record<string, number>>
{
  const url =
`https://api.fixer.io/latest?base=${baseCurrency}&access_key=${process.env.FIXER_API_KEY}`

  const response = await fetch(url)
  const data = await response.json()

  if (!data.success) {
    throw new Error('Fixer.io API error')
  }

  return data.rates
}
```

Exchange Rate Service

```
async function updateExchangeRates(): Promise<void> {
  try {
    // Try ECB first
    const rates = await fetchECBRates('EUR')

    // Store in database
    for (const [targetCurrency, rate] of Object.entries(rates)) {
      await prisma.exchangeRate.upsert({
        where: {
          baseCurrency_targetCurrency_effectiveDate: {
            baseCurrency: 'EUR',
            targetCurrency,
            effectiveDate: new Date()
          }
        },
        update: {
          rate,
          source: 'ECB',
          lastUpdated: new Date()
        },
        create: {
          baseCurrency: 'EUR',
          targetCurrency,
          rate,
          effectiveDate: new Date(),
          source: 'ECB'
        }
      })
    }

    logger.info('Exchange rates updated', { source: 'ECB', count: Object.keys(rates).length })
  } catch (error) {
    // Fallback to Fixer.io
    try {
      const rates = await fetchFixerRates('EUR')

      for (const [targetCurrency, rate] of Object.entries(rates)) {
```

```
        await prisma.exchangeRate.upsert({
          where: {
            baseCurrency_targetCurrency_effectiveDate: {
              baseCurrency: 'EUR',
              targetCurrency,
              effectiveDate: new Date()
            }
          },
          update: { rate, source: 'fixer.io', lastUpdated: new Date() },
          create: {
            baseCurrency: 'EUR',
            targetCurrency,
            rate,
            effectiveDate: new Date(),
            source: 'fixer.io'
          }
        })
      }

      logger.info('Exchange rates updated', { source: 'fixer.io', count:
Object.keys(rates).length })
    } catch (fallbackError) {
      logger.error('Failed to update exchange rates', { error: fallbackError })
      // Use yesterday's rates (better than nothing)
    }
  }
}

// Schedule: Daily at 00:00 UTC
cron.schedule('0 0 * * *', updateExchangeRates)
```

Get Exchange Rate

```
async function getExchangeRate(
  baseCurrency: string,
  targetCurrency: string,
  date: Date = new Date()
): Promise<number> {
  // If same currency, rate = 1.0
```

```
  if (baseCurrency === targetCurrency) {
    return 1.0
  }

  // Find rate for exact date
  let rate = await prisma.exchangeRate.findUnique({
    where: {
      baseCurrency_targetCurrency_effectiveDate: {
        baseCurrency,
        targetCurrency,
        effectiveDate: date
      }
    }
  })

  // If not found, use nearest available rate
  if (!rate) {
    rate = await prisma.exchangeRate.findFirst({
      where: { baseCurrency, targetCurrency },
      orderBy: { effectiveDate: 'desc' }
    })
  }

  if (!rate) {
    throw new Error(`No exchange rate found for ${baseCurrency} → ${targetCurrency}`)
  }

  return parseFloat(rate.rate.toString())
}
```

---

4. PDF Generation

Purpose

Generate invoice PDFs with organization branding.

Option 1: Puppeteer (Server-Side Rendering)

Installation:

```
npm install puppeteer
```

Implementation:

```
import puppeteer from 'puppeteer'

async function generateInvoicePDF(invoiceId: string): Promise<Buffer> {
  const invoice = await prisma.invoice.findUnique({
    where: { id: invoiceId },
    include: {
      customer: true,
      organization: true,
      items: true
    }
  })

  // Render HTML template
  const html = renderInvoiceHTML(invoice)

  // Launch headless browser
  const browser = await puppeteer.launch({
    headless: true,
    args: ['--no-sandbox', '--disable-setuid-sandbox']
  })

  const page = await browser.newPage()
  await page.setContent(html, { waitUntil: 'networkidle0' })

  // Generate PDF
  const pdf = await page.pdf({
    format: 'A4',
    printBackground: true,
    margin: { top: '1cm', right: '1cm', bottom: '1cm', left: '1cm' }
  })

  await browser.close()

  return pdf
```

```
}
```

Option 2: @react-pdf/renderer (React Components)

Installation:

```
npm install @react-pdf/renderer
```

Implementation:

```
import { Document, Page, Text, View, StyleSheet, pdf } from '@react-pdf/renderer'

const styles = StyleSheet.create({
  page: { padding: 30 },
  header: { fontSize: 24, marginBottom: 20 },
  table: { display: 'table', width: '100%' },
  row: { flexDirection: 'row', borderBottomWidth: 1, borderColor: '#ddd' },
  cell: { padding: 10 }
})

function InvoicePDF({ invoice }) {
  return (
    <Document>
      <Page style={styles.page}>
        <View style={styles.header}>
          <Text>{invoice.organization.name}</Text>
        </View>
        <Text>Invoice {invoice.invoiceNumber}</Text>
        <View style={styles.table}>
          {invoice.items.map((item) => (
            <View style={styles.row} key={item.id}>
              <Text style={styles.cell}>{item.description}</Text>
              <Text style={styles.cell}>{item.quantity}</Text>
              <Text style={styles.cell}>{item.unitPrice}</Text>
              <Text style={styles.cell}>{item.lineTotal}</Text>
            </View>
          ))}
        </View>
```

```
      </Page>
    </Document>
  )
}

async function generateInvoicePDF(invoiceId: string): Promise<Buffer> {
  const invoice = await fetchInvoiceData(invoiceId)
  const doc = <InvoicePDF invoice={invoice} />
  const pdfBlob = await pdf(doc).toBlob()
  return Buffer.from(await pdfBlob.arrayBuffer())
}
```

Recommendation: Puppeteer for MVP (more flexible HTML/CSS), React PDF for v2 (better TypeScript support).

---

5. Error Handling & Fallbacks

Circuit Breaker & Retry Pattern

```
stateDiagram-v2
    [*] --> closed : Initial state

    closed --> open : failures >= threshold (5)\nService marked as unavailable

    open --> half_open : timeout elapsed (60s)\nAttempt single test call

    half_open --> closed : Test call succeeded\nReset failure count

    half_open --> open : Test call failed\nReset timeout

    note right of closed
        Normal operation
        All calls pass through
        Failures counted
    end note

    note right of open
```

```
        All calls REJECTED immediately
        Error: "Circuit breaker is open"
        No calls to external service
    end note

    note right of half_open
        Single probe call allowed
        Determines if service recovered
    end note
```

Retry Strategy

For transient errors (network timeouts, rate limits):

```
async function withRetry<T>(
  fn: () => Promise<T>,
  maxRetries: number = 3,
  delay: number = 1000
): Promise<T> {
  for (let i = 0; i < maxRetries; i++) {
    try {
      return await fn()
    } catch (error) {
      if (i === maxRetries - 1) throw error

      logger.warn(`Retry ${i + 1}/${maxRetries}`, { error })
      await new Promise((resolve) => setTimeout(resolve, delay * (i + 1)))
    }
  }

  throw new Error('Retry limit exceeded')
}

// Usage
const rates = await withRetry(() => fetchECBRates('EUR'))
```

Circuit Breaker

For external services that frequently fail:

```
class CircuitBreaker {
  private failures = 0
  private threshold = 5
  private timeout = 60000  // 1 minute
  private state: 'closed' | 'open' | 'half-open' = 'closed'
  private nextAttempt = 0

  async execute<T>(fn: () => Promise<T>): Promise<T> {
    if (this.state === 'open') {
      if (Date.now() < this.nextAttempt) {
        throw new Error('Circuit breaker is open')
      }
      this.state = 'half-open'
    }

    try {
      const result = await fn()
      this.onSuccess()
      return result
    } catch (error) {
      this.onFailure()
      throw error
    }
  }

  private onSuccess() {
    this.failures = 0
    this.state = 'closed'
  }

  private onFailure() {
    this.failures++
    if (this.failures >= this.threshold) {
      this.state = 'open'
      this.nextAttempt = Date.now() + this.timeout
      logger.warn('Circuit breaker opened', { failures: this.failures })
    }
  }
}
```

```
const sendGridBreaker = new CircuitBreaker()

async function sendEmailWithBreaker(options: SendEmailOptions) {
  return sendGridBreaker.execute(() => sendEmail(options))
}
```

Graceful Degradation

If external service fails, degrade gracefully:

Example: Invoice email delivery

```
async function sendInvoiceEmail(invoiceId: string) {
  try {
    await sendEmail({ /* ... */ })
    await prisma.invoice.update({
      where: { id: invoiceId },
      data: { sentAt: new Date(), status: 'sent' }
    })
  } catch (error) {
    logger.error('Failed to send invoice email', { invoiceId, error })

    // Fallback: Mark invoice as sent but flag for manual email
    await prisma.invoice.update({
      where: { id: invoiceId },
      data: {
        status: 'draft',  // Keep in draft
        notes: `Email delivery failed: ${error.message}. Please send manually.`
      }
    })

    // Alert admin
    await sendSlackAlert('Invoice email delivery failed', { invoiceId })
  }
}
```

Environment Variables Summary

```
# SendGrid
SENDGRID_API_KEY=SG.xxxxx
SENDGRID_FROM_EMAIL=noreply@bilko.io
SENDGRID_FROM_NAME=Bilko

# Cloudflare R2
R2_ACCOUNT_ID=your-account-id
R2_ACCESS_KEY_ID=your-access-key
R2_SECRET_ACCESS_KEY=your-secret-key
R2_BUCKET_NAME=bilko-files
R2_PUBLIC_URL=https://r2.bilko.io

# Fixer.io (fallback)
FIXER_API_KEY=your-api-key

# Feature Flags
ENABLE_EMAIL_TRACKING=true
ENABLE_VIRUS_SCANNING=false  # For MVP
```

End of Services Documentation

Revision #3
Created 2026-02-23 10:47:53 UTC by John
Updated 2026-05-31 20:02:39 UTC by John