Developer Experience

Bilko developer onboarding, local setup, coding standards, and offboarding

Coding Standards

Coding Standards

> **Project:** Bilko **Version:** 0.1 **Date:** 2026-02-23 **Author:** Ops Architect **Status:** Draft **Reviewers:** Tech Lead, Alem Bašić

Document History

Version	Date	Author	Changes
0.1	2026-02-23	Ops Architect	Initial draft

---

1. Language & Type Safety

TypeScript Strict Mode (Required)

All code must compile with TypeScript strict mode. No exceptions.

```
// tsconfig.json
{
  "compilerOptions": {
    "strict": true,
    "noImplicitAny": true,
    "strictNullChecks": true,
    "strictFunctionTypes": true
  }
}
```

Rules:

- No `any` type without an inline comment explaining why: `// eslint-disable-next-line @typescript-eslint/no-explicit-any`

- Prefer `unknown` over `any` for untyped inputs (Zod validate before using)
- Use interface for object shapes, type for unions/intersections
- All function parameters and return types must be typed

```
// WRONG
function calculateVAT(amount: any, rate: any) {
  return amount * rate / 100;
}

// CORRECT
function calculateVAT(amount: Decimal, rate: number): Decimal {
  return amount.mul(rate).div(100);
}
```

---

2. Financial Logic (Non-Negotiable Rules)

These rules cannot be overridden by any other consideration.

Rule F1: NEVER Use JavaScript `number` for Money

```
// WRONG — IEEE 754 float precision errors
const vatAmount = 123.45 * 0.20;  // 24.690000000000003

// CORRECT — decimal.js exact arithmetic
import { Decimal } from 'decimal.js';
const vatAmount = new Decimal('123.45').mul('0.20');  // 24.6900
```

Rule F2: All Monetary Values Are Strings in API Responses

Prisma returns `NUMERIC` columns as strings. Keep them as strings. Never `.toNumber()` monetary values.

```
// WRONG — loses NUMERIC precision
const invoice = await prisma.invoice.findUnique({ where: { id } });
return { totalAmount: invoice.totalAmount.toNumber() };  // NEVER

// CORRECT — return as string, JSON serializes fine
return { totalAmount: invoice.totalAmount.toString() };
```

Rule F3: Double-Entry Is Always Enforced

Every financial event creates equal debit and credit entries. No exceptions.

```
// CORRECT pattern for creating a transaction
async function createTransaction(data: CreateTransactionInput) {
  const debitTotal = data.entries
    .filter(e => e.type === 'debit')
    .reduce((sum, e) => sum.plus(e.amount), new Decimal(0));
  const creditTotal = data.entries
    .filter(e => e.type === 'credit')
    .reduce((sum, e) => sum.plus(e.amount), new Decimal(0));

  if (!debitTotal.equals(creditTotal)) {
    throw new DoubleEntryError(
      `Imbalance: debit ${debitTotal} ≠ credit ${creditTotal}`
    );
  }

  return prisma.transaction.create({ data: { ...data } });
}
```

Rule F4: VAT Rates Are from Database, Never Hardcoded

```
// WRONG
const vatRate = country === 'RS' ? 20 : 17;

// CORRECT — get from VatRate table, allow configuration
const vatRate = await getVatRate(organizationId, country, itemType, transactionDate);
```

Rule F5: Exchange Rates Are Locked at Transaction Date

```
// WRONG
const rate = await getCurrentExchangeRate(fromCurrency, toCurrency);

// CORRECT
const rate = await getExchangeRateForDate(fromCurrency, toCurrency, transactionDate);
if (!rate) throw new Error(`No exchange rate for ${fromCurrency}/${toCurrency} on
${transactionDate}`);
```

---

3. API Design

RESTful Conventions

Action	Method	Path	Success Code
List	GET	`/api/v1/invoices`	200
Create	POST	`/api/v1/invoices`	201
Read	GET	`/api/v1/invoices/:id`	200
Update	PATCH	`/api/v1/invoices/:id`	200
Delete (soft)	DELETE	`/api/v1/invoices/:id`	200
Bulk action	POST	`/api/v1/invoices/bulk-send`	200

Request Validation with Zod

All request bodies and query parameters must be validated with Zod before use.

```
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(['RSD', 'EUR', 'BAM', 'HRK', 'USD']),
```

```
  items: z.array(z.object({
    description: z.string().min(1).max(500),
    quantity: z.number().positive(),
    unitPrice: z.string().regex(/^\d+(\.\d{1,4})?$/),  // Decimal string
    taxRate: z.number().min(0).max(100),
  })).min(1),
});

// In route handler
const parsed = CreateInvoiceSchema.safeParse(req.body);
if (!parsed.success) {
  return res.status(400).json({ error: 'Validation failed', details: parsed.error });
}
```

Error Response Format

```
{
  "error": "Validation failed",
  "code": "VALIDATION_ERROR",
  "details": [
    { "field": "customerId", "message": "Invalid UUID" }
  ]
}
```

Standard error codes: `VALIDATION_ERROR`, `NOT_FOUND`, `FORBIDDEN`, `UNAUTHORIZED`, `DOUBLE_ENTRY_ERROR`, `INSUFFICIENT_PERMISSIONS`

Pagination

All list endpoints must support pagination:

```
// Query: GET /api/v1/invoices?limit=20&cursor=<uuid>

// Response
{
  "data": [...],
  "pagination": {
    "cursor": "<next-cursor-uuid>",
    "hasMore": true,
    "total": 150
```

```
  }
}
```

---

4. Database (Prisma)

Query Rules

```
// ALWAYS include organizationId filter (multi-tenancy)
const invoices = await prisma.invoice.findMany({
  where: {
    organizationId: req.user.organizationId,  // Required
    deletedAt: null,
  },
});

// NEVER expose deleted records unless explicitly showing "trash"
// NEVER query without organizationId on any user-facing query

// Use select for response DTOs — never return full Prisma models
const invoice = await prisma.invoice.findUnique({
  where: { id, organizationId: req.user.organizationId },
  select: {
    id: true,
    invoiceNumber: true,
    totalAmount: true,
    status: true,
    // ... only what the endpoint needs
  },
});
```

Migration Rules

- NEVER edit existing migrations
- ALWAYS create new migrations for schema changes: `npx prisma migrate dev --name describe_change`
- Migrations must be backward-compatible (old code can still read data) unless explicitly planned otherwise

- NEVER drop columns in a migration — first deprecate, remove code references, then drop in a separate migration
- Always test migrations on staging before production

---

5. Authentication & Authorization

Every Route Needs Auth (No Exceptions)

```
// Middleware applied globally
app.use('/api/v1', authenticateJWT);

// Route-level RBAC
router.post('/invoices', requireRole(['owner', 'admin', 'accountant']), createInvoice);
router.delete('/invoices/:id', requireRole(['owner', 'admin']), deleteInvoice);
router.get('/invoices/:id', requireRole(['owner', 'admin', 'accountant', 'viewer']),
getInvoice);
```

RBAC Roles and Permissions

Role	Can Read	Can Create	Can Approve	Can Delete	Can Manage Users
viewer	Own org only	No	No	No	No
accountant	Own org only	Invoices, Expenses	No	No	No
admin	Own org only	All	Expenses	Soft delete	No
owner	Own org only	All	All	All	Yes

---

6. Code Organization

File Structure (Backend)

```
apps/api/src/
├── routes/
│   ├── auth.routes.ts          # Route definitions only
```

```
│   ├── invoices.routes.ts
│   └── invoices.routes.test.ts  # Integration tests co-located
├── services/
│   ├── invoice.service.ts       # Business logic
│   └── invoice.service.test.ts  # Unit tests co-located
├── utils/
│   ├── vat.ts                   # VAT calculation utilities
│   ├── vat.test.ts
│   ├── double-entry.ts
│   └── currency.ts
├── middleware/
│   ├── auth.middleware.ts
│   └── rbac.middleware.ts
└── lib/
    ├── prisma.ts                # Prisma client singleton
    └── errors.ts                # Custom error classes
```

Naming Conventions

Thing	Convention	Example
Files	kebab-case	`invoice-service.ts`
Functions	camelCase	`calculateInvoiceTotal`
Classes	PascalCase	`InvoiceService`
Constants	UPPER_SNAKE	`MAX_INVOICE_ITEMS = 100`
Types/Interfaces	PascalCase	`CreateInvoiceInput`
Database models	PascalCase (Prisma)	`Invoice`, `InvoiceItem`
API routes	kebab-case	`/api/v1/invoice-items`

---

7. Testing Standards

Write Tests in the Same PR as the Feature

No "I'll add tests later". Tests are part of the feature.

Required Tests for Financial Logic

```
// For EVERY financial utility function, include:
it('calculates correctly', () => { ... });
it('handles zero amounts', () => { ... });
it('handles decimal precision (NUMERIC(19,4))', () => {
  const result = calculateVAT(new Decimal('0.1000'), 20);
  expect(result.toString()).toBe('0.0200');  // Not 0.020000000000000004
});
it('throws on invalid input', () => { ... });
```

Test File Co-location

Tests live next to the file they test:

- `vat.ts` → `vat.test.ts` (same directory)
- `invoices.routes.ts` → `invoices.routes.test.ts` (same directory)

---

8. Linting & Formatting

Configured via ESLint + Prettier. Run automatically on every commit (Husky) and in CI.

```
# Fix all linting issues
pnpm run lint -- --fix

# Check formatting
pnpm run lint
```

Key ESLint rules:

- `no-floating-decimal` — numbers like `.5` must be written as `0.5`
- `@typescript-eslint/no-explicit-any` — warn (add comment to suppress)
- `no-console` — warn in production code (use logger instead)
- Import order enforced

---

9. Security

Never Commit Secrets

```
# .gitignore includes:
.env
.env.*
!.env.example
```

OWASP Top 10 Checklist for New Code

- ☐ SQL injection: using Prisma parameterized queries only (never string concatenation)
- ☐ XSS: user input never inserted into HTML without escaping (React escapes by default)
- ☐ Auth: every route has authentication check
- ☐ IDOR: every DB query filtered by `organizationId`
- ☐ Sensitive data: no passwords/tokens in logs
- ☐ Input validation: Zod on all request bodies

---

10. Version Control

Commit Message Format

```
type(scope): description (#issue-id)

type: feat | fix | refactor | test | docs | chore | perf | security
scope: api | web | db | infra | deps
```

Examples:

```
feat(api): add VAT calculation for Croatia 25% rate (#234)
fix(api): correct decimal precision in invoice total calculation (#456)
test(api): add unit tests for double-entry validation
refactor(web): extract invoice form to separate component
```

Branch Naming

```
feature/<issue-id>-<short-description>
fix/<issue-id>-<short-description>
hotfix/<short-description>
```

PR Requirements

- Title follows commit message format
- Description: What changed, why, any testing notes
- Links to issue: "Closes #XXX" or "Ref #XXX"
- All CI checks green
- No `.only` or `.skip` in test files
- Coverage did not decrease below thresholds

Related Documents

- Test Strategy
- Definition of Done
- Developer Onboarding Guide
- BILKO CLAUDE.md

Approval

Role	Name	Date	Signature
Author	Ops Architect	2026-02-23	
Reviewer	Tech Lead		
Approver	Alem Bašić		

Developer Offboarding

Developer Offboarding Guide

> **Project:** Bilko **Version:** 0.1 **Date:** 2026-02-23 **Author:** Ops Architect **Status:** Draft **Reviewers:** Alem Bašić

Document History

Version	Date	Author	Changes
0.1	2026-02-23	Ops Architect	Initial draft

Overview

This guide covers the process for offboarding a developer from the Bilko project. Complete all items within 5 business days of the developer's last working day. The offboarding manager is responsible for completing the checklist.

Offboarding manager: Alem Bašić (or designated Tech Lead) **Required:** All P0 items must be completed on the developer's last day or before

---

1. Pre-Offboarding (2 Weeks Before Last Day)

Knowledge Transfer

- ☐ Developer documents all work-in-progress in GitHub Issues or Mission Control
- ☐ Developer reviews open PRs — complete or hand off each one

- [ ] Developer documents any undocumented systems or processes they own
- [ ] Knowledge transfer sessions scheduled (30 min per significant area)
- [ ] Access to any personal accounts used for Bilko work shared or transferred

Codebase Handoff

- [ ] All local branches pushed to GitHub (or explicitly discarded)
- [ ] All work-in-progress committed or stashed in a handoff branch: `handoff/<name>-<date>`
- [ ] Any local configuration or environment modifications documented
- [ ] Developer reviews CLAUDE.md files they may have modified — confirm still accurate

---

2. Last Day Checklist

P0 — Complete on Last Day

GitHub Access

- [ ] All open PRs reviewed: merge, close with explanation, or assign to another developer
- [ ] GitHub organization membership removed: GitHub → Organization Settings → Members → Remove
- [ ] Repository-specific access revoked (if different from org membership)

Infrastructure Access

- [ ] Railway access removed: Railway → Project → Settings → Members → Remove
- [ ] Vercel access removed: Vercel → Team Settings → Members → Remove
- [ ] Cloudflare access removed (if granted): Cloudflare → Account → Manage account → Members

Secrets and Credentials

- [ ] Any personal API tokens/PATs used for Bilko rotated immediately
- [ ] Verify developer does not have production secrets stored locally (confirm via discussion)
- [ ] If developer had production Railway access: rotate all production secrets:
  - `JWT_SECRET` (this invalidates all user sessions — notify users)
  - `JWT_REFRESH_SECRET`
  - `SENDGRID_API_KEY` (only if developer had SendGrid access)

- `R2_ACCESS_KEY_ID` / `R2_SECRET_ACCESS_KEY` (only if developer had R2 access)

Communication

- [ ] Developer removed from Slack #bilko-dev and #bilko-deploys
- [ ] GitHub Issues/PRs reassigned from developer to active team members

---

3. Post-Offboarding (Within 5 Business Days)

Audit

- [ ] Review audit log (Railway logs) for any unusual activity in last 30 days
- [ ] Review `LoggedAction` table for developer's user ID (if they had production access)
- [ ] Verify no unauthorized changes to production configuration
- [ ] Review GitHub audit log for developer's last week of activity

Documentation Update

- [ ] Developer removed from team roster / org chart
- [ ] Any documentation with developer's name as contact updated
- [ ] On-call rotation updated if developer was on-call
- [ ] `CLAUDE.md` files updated if developer was listed as a contact

Knowledge Gap Assessment

- [ ] Identify any areas where the developer was the sole owner
- [ ] Create GitHub Issues for knowledge gaps that need documentation
- [ ] Assign ownership of developer's areas to remaining team members

---

4. Offboarding Checklist (Per Developer)

Create a copy of this section for each offboarding:

Developer: _____ Last working day: YYYY-MM-DD **Offboarding manager:** _____

Access Revocation Log

System	Access Removed	Date	By
GitHub	[ ] Yes		
Railway	[ ] Yes		
Vercel	[ ] Yes		
Cloudflare	[ ] N/A or [ ] Yes		
Sentry	[ ] N/A or [ ] Yes		
Slack	[ ] Yes		
BetterStack	[ ] N/A or [ ] Yes		

Secret Rotation (if developer had production access)

Secret	Rotated	Date	Notes
JWT_SECRET	[ ] Yes / [ ] N/A		Users notified: Yes/No
JWT_REFRESH_SECRET	[ ] Yes / [ ] N/A		
Other: _____	[ ] Yes / [ ] N/A		

Open Work Disposition

Item	Type	Disposition	Assigned To
PR #XXX	Pull Request	Merged / Closed / Reassigned	
Issue #XXX	GitHub Issue	Closed / Reassigned	
[Feature X]	WIP	Handoff branch created	

---

5. Data Retention

Per GDPR Article 17 and Bilko data retention policy:

- Developer's commits remain in git history (normal — cannot be removed without rebasing)
- Developer's user account in `bilko_prod` database: mark as inactive (do not delete — audit trail)
- `LoggedAction` audit records: retained indefinitely (regulatory requirement)
- Personal data of the developer stored in Bilko systems: delete per GDPR right to erasure if requested

---

Approval

Role	Name	Date	Signature
Offboarding Manager			
Approver	Alem Bašić		

Developer Onboarding

Developer Onboarding Guide

> **Project:** Bilko **Version:** 0.1 **Date:** 2026-02-23 **Author:** Ops Architect **Status:** Draft **Reviewers:** Tech Lead, Alem Bašić

Document History

Version	Date	Author	Changes
0.1	2026-02-23	Ops Architect	Initial draft

Welcome to Bilko

Bilko is a cloud accounting SaaS for Balkan SMBs (Serbia, Bosnia, Croatia). It handles invoicing, expense tracking, VAT reporting, and financial bookkeeping. This guide gets a new developer from zero to productive in one day.

Critical context: Bilko handles real financial data. Bugs in VAT calculations, double-entry bookkeeping, or NUMERIC precision are not just bugs — they can cause tax compliance failures for users. Read the accounting fundamentals section before writing any financial logic.

Day 1 Checklist

Access & Accounts

Contact Alem Bašić (alem@alai.no) to be granted:

- ☐ GitHub: Added to `alai-holding` organization, `bilko` repository
- ☐ Railway: Added to Bilko project (staging environment access only)

- [ ] Vercel: Added to ALAI team (staging/preview access only)
- [ ] Slack: Added to #bilko-dev channel
- [ ] Sentry: Added to bilko-frontend and bilko-backend projects (view only)

Local Setup

- [ ] Follow local-development-setup.md — estimated 45 minutes
- [ ] Run `pnpm run dev` — both frontend (port 3000) and backend (port 4000) start
- [ ] Run `pnpm run test:unit` — all tests pass
- [ ] Open http://localhost:3000 — dashboard loads (mock data at MVP stage)
- [ ] Open http://localhost:5555 (Prisma Studio) — database is visible

Reading (complete on Day 1)

- [ ] **BILKO CLAUDE.md** (`~/ALAI/products/Bilko/CLAUDE.md`) — project overview, dev rules
- [ ] **Coding Standards** (`docs/templates/DEVELOPER-EXPERIENCE/coding-standards.md`) — rules you must follow
- [ ] **Database Schema** (`packages/database/prisma/schema.prisma`) — the 15 models
- [ ] **Test Strategy** (`docs/templates/TESTING/test-strategy.md`) — how we test

Tech Stack

Frontend (apps/web/)

Technology	Version	Purpose
Next.js	15.0.0	React framework, SSR
React	19.0.0	UI library
TypeScript	5.3.0	Type safety (strict mode)
Tailwind CSS	4.0.0	Styling
shadcn/ui	Latest	Component library (Radix UI + Tailwind)
Zustand	4.5.0	State management
Recharts	2.15.0	Charts (revenue, expenses)

Technology	Version	Purpose
Lucide React	Latest	Icons

Backend (apps/api/)

Technology	Version	Purpose
Express	Latest	API framework
TypeScript	5.3.0	Type safety (strict mode)
Prisma	Latest	ORM + migrations
PostgreSQL	15	Database
Passport.js	Latest	Authentication strategy
Zod	Latest	Request validation
Helmet	Latest	Security headers
bcrypt	Latest	Password hashing (12 rounds)
jsonwebtoken	Latest	JWT tokens
decimal.js	Latest	NUMERIC precision arithmetic

Monorepo

Tool	Purpose
Turborepo	Build orchestration, incremental builds
pnpm workspaces	Package management
packages/database	Prisma schema + client (shared)
packages/ui	Shared UI components (empty scaffold at MVP)

Project Structure

```
Bilko/
├── apps/
│   ├── web/              # Next.js 15 frontend (bilko.io)
│   │   ├── app/          # Next.js App Router pages
│   │   ├── components/   # React components
│   │   ├── lib/          # Utilities, API client
```

```
│   │   └── lib/mock-data.ts  # MOCK DATA — replace with real API calls
│   └── api/              # Express backend (api.bilko.io)
│       ├── src/
│       │   ├── routes/   # Express route handlers
│       │   ├── services/ # Business logic
│       │   ├── utils/    # Utilities (VAT, double-entry, currency)
│       │   ├── middleware/  # Auth, validation, error handling
│       │   └── test/     # Test setup, factories, helpers
│       └── CLAUDE.md     # Backend-specific AI instructions
├── packages/
│   ├── database/         # Prisma schema + generated client
│   │   └── prisma/
│   │       ├── schema.prisma  # 15 models — READ THIS FIRST
│   │       └── migrations/    # All DB migrations
│   └── ui/               # Shared components (scaffold only)
└── docs/                 # Documentation
    ├── infrastructure/   # CI/CD, deployment, environment setup
    ├── testing/          # Testing guide, test inventory
    └── templates/        # Document templates (this file is here)
```

Accounting Fundamentals (Required Reading)

You don't need to be an accountant, but you need to understand these concepts:

Double-Entry Bookkeeping

Every financial transaction has two sides: a debit and a credit. They must always balance (debit = credit). Example:

- Invoice of 60,000 RSD:
  - DEBIT: Accounts Receivable 60,000 RSD
  - CREDIT: Revenue 50,000 RSD + VAT Payable 10,000 RSD

Bilko enforces this in the `Transaction` + `TransactionEntry` models. You CANNOT create a transaction where debit ≠ credit.

NUMERIC(19,4) — Never Use float for Money

```
// WRONG — float arithmetic has precision errors
const vat = 100 * 0.20;  // Could be 20.000000000000004

// CORRECT — use Decimal from decimal.js
import { Decimal } from 'decimal.js';
const vat = new Decimal('100').mul('0.20');  // Exactly 20.0000
```

All monetary columns in PostgreSQL are `NUMERIC(19,4)`. All monetary values in TypeScript must use `Decimal`.

Multi-Currency with Rate Locking

When a transaction is created in EUR but the base currency is RSD, the exchange rate is locked at the transaction date — not today's rate. This is required for accurate historical financial reports.

VAT by Country

- Serbia (RS): Standard 20%, Reduced 10%, Zero 0% (exports)
- Bosnia (BA): Standard 17%, Zero 0% (exports). No reduced rate.
- Croatia (HR): Standard 25%, Reduced 13%, Super-reduced 5%

Development Workflow

Starting Work on a Feature

```
# 1. Pull latest main
git checkout main && git pull

# 2. Create feature branch
git checkout -b feature/INV-123-invoice-pdf-generation

# 3. Make changes
# ... code ...

# 4. Test locally
pnpm run lint
pnpm run type-check
```

```
pnpm run test:unit -- --coverage
pnpm run test:integration

# 5. Commit
git add apps/api/src/services/invoice.service.ts
git add apps/api/src/services/invoice.service.test.ts
git commit -m "feat: add PDF generation for invoices (INV-123)"

# 6. Push and create PR
git push origin feature/INV-123-invoice-pdf-generation
# Create PR on GitHub — CI runs automatically
```

Commit Message Format

```
type: description (issue-id)

Types: feat, fix, refactor, test, docs, chore
Examples:
  feat: add Serbian VAT calculation with 10% reduced rate
  fix: correct decimal precision in VAT calculation (BILKO-456)
  test: add unit tests for double-entry validation
  refactor: extract invoice total calculation to service layer
```

PR Requirements

- Description explains what and why
- Links to GitHub issue or Mission Control task
- All CI checks green
- At least 1 reviewer approval
- Tests written for all new business logic

Running the Application

```
# Install dependencies
pnpm install

# Run database migrations
```

```
cd packages/database && npx prisma migrate dev && npx prisma generate

# Start all services (from root)
pnpm run dev
# Frontend: http://localhost:3000
# Backend: http://localhost:4000
# Prisma Studio: npx prisma studio (port 5555)

# Run all tests
pnpm run test

# Build for production (verify before deploy)
pnpm run build
```

Key Contacts

Person	Role	Contact
Alem Bašić	Founder/CEO (Human)	alem@alai.no, +47 40 47 42 51
John	AI Director (Claude Opus)	Handles coordination, tasks, infra questions

Slack channels:

- `#bilko-dev` — development discussion
- `#bilko-deploys` — deployment notifications and alerts

Getting Help

1. **Code question:** Check `CLAUDE.md` files (root + `apps/api/CLAUDE.md`) first
2. **Architecture question:** Check `docs/` directory
3. **Database schema:** `packages/database/prisma/schema.prisma`
4. **Still stuck:** Ask in #bilko-dev Slack or create a GitHub Discussion

---

First Week Milestones

Day	Goal
1	Dev environment running, tests passing, codebase read
2-3	First small PR merged (bug fix or test)
4-5	First feature PR opened
End of week 1	Comfortable with invoice + expense flow

Related Documents

- Local Development Setup
- Coding Standards
- Test Strategy
- BILKO CLAUDE.md
- ENVIRONMENT.md

Approval

Role	Name	Date	Signature
Author	Ops Architect	2026-02-23	
Reviewer	Alem Bašić		

Local Development Setup

Local Development Setup

> **Project:** Bilko **Version:** 0.1 **Date:** 2026-02-23 **Author:** Ops Architect **Status:** Draft **Reviewers:** Tech Lead, Alem Bašić

Document History

Version	Date	Author	Changes
0.1	2026-02-23	Ops Architect	Initial draft

Overview

This guide gets a Bilko development environment running from scratch. Estimated time: 45–60 minutes on a clean machine.

Full environment documentation: `../infrastructure/ENVIRONMENT.md`

Prerequisites

Software	Version	Check	Install
Node.js	20.x (LTS)	`node --version`	https://nodejs.org
pnpm	8.x+	`pnpm --version`	`npm install -g pnpm`
PostgreSQL	15+	`psql --version`	https://postgresql.org/download or Docker
Git	Latest	`git --version`	https://git-scm.com

Optional (recommended):

- VS Code with extensions: ESLint, Prettier, Prisma, Tailwind CSS IntelliSense
- TablePlus or DBeaver for database browsing
- Postman or Insomnia for API testing

Installation Steps

Step 1: Clone Repository

```
git clone https://github.com/alai-holding/bilko.git
cd bilko
```

Step 2: Install Dependencies

```
# Install all workspace dependencies (from repo root)
pnpm install
```

This installs dependencies for:

- Root workspace (Turborepo)
- `apps/web` (Next.js 15)
- `apps/api` (Express)
- `packages/database` (Prisma)
- `packages/ui` (shared components)

Step 3: Set Up PostgreSQL

Option A: Local PostgreSQL installation

```
# Connect to PostgreSQL as admin
psql -U postgres

# Run in psql:
CREATE DATABASE bilko_dev;
CREATE USER bilko WITH PASSWORD 'bilko';
GRANT ALL PRIVILEGES ON DATABASE bilko_dev TO bilko;
\q
```

Option B: Docker (no PostgreSQL installation required)

```
docker run --name bilko-postgres \
  -e POSTGRES_USER=bilko \
  -e POSTGRES_PASSWORD=bilko \
  -e POSTGRES_DB=bilko_dev \
  -p 5432:5432 \
  --restart unless-stopped \
  -d postgres:15

# Verify it's running
docker ps | grep bilko-postgres
```

Step 4: Configure Environment Variables

Create `apps/api/.env`:

```
cp apps/api/.env.example apps/api/.env
# Edit with your values
```

Content:

```
# Database
DATABASE_URL=postgresql://bilko:bilko@localhost:5432/bilko_dev

# JWT Secrets (generate with: openssl rand -base64 32)
JWT_SECRET=dev-secret-change-in-production
JWT_REFRESH_SECRET=dev-refresh-secret-change-in-production

# Email — leave empty for local dev (invoices won't be emailed but can be created)
SENDGRID_API_KEY=

# Cloudflare R2 — leave empty for local dev (file uploads will fail gracefully)
R2_ACCESS_KEY_ID=
R2_SECRET_ACCESS_KEY=
R2_BUCKET_NAME=bilko-receipts-dev
R2_ENDPOINT=

# App
PORT=4000
NODE_ENV=development
ALLOWED_ORIGINS=http://localhost:3000
```

Create `apps/web/.env.local`:

```
cp apps/web/.env.local.example apps/web/.env.local
```

Content:

```
NEXT_PUBLIC_API_URL=http://localhost:4000
NEXT_PUBLIC_APP_ENV=development
```

Step 5: Run Database Migrations

```
cd packages/database
npx prisma migrate dev
npx prisma generate
cd ../..
```

This:

1. Applies all 15+ table migrations to `bilko_dev`
2. Generates the Prisma Client TypeScript types

Step 6: Seed Database (Optional)

```
cd packages/database
npx prisma db seed
cd ../..
```

Creates:

- Demo organization: `Test Company d.o.o.` (country: RS, currency: RSD)
- Demo user: `demo@bilko.io` / `demo123` (role: owner)
- Sample contacts, one invoice (draft state)

Running the Application

Start All Services

```
# From repo root — starts both frontend and backend via Turborepo
pnpm run dev
```

Services:

- **Frontend:** http://localhost:3000 (Next.js dev server)
- **Backend:** http://localhost:4000 (Express, hot reload via nodemon)

Start Individual Services

```
# Frontend only
cd apps/web && pnpm run dev

# Backend only
cd apps/api && pnpm run dev
```

Prisma Studio (Database GUI)

```
cd packages/database && npx prisma studio
# Opens at http://localhost:5555
```

Features: Browse tables, edit records, view relations, run queries.

Running Tests

All Tests

```
pnpm run test
```

Unit Tests

```
# Run
pnpm run test:unit

# Watch mode (re-run on save)
pnpm run test:unit -- --watch
```

```
# With coverage report
pnpm run test:unit -- --coverage

# Single file
pnpm run test:unit -- vat.test.ts
```

Integration Tests

Requires the test database `bilko_test`:

```
# Create test database
psql -U postgres -c "CREATE DATABASE bilko_test;"
psql -U postgres -c "GRANT ALL PRIVILEGES ON DATABASE bilko_test TO bilko;"

# Run integration tests
pnpm run test:integration

# Single file
pnpm run test:integration -- invoices-api.test.ts
```

E2E Tests

Requires both frontend and backend running:

```
# Terminal 1: Start dev servers
pnpm run dev

# Terminal 2: Run E2E tests
pnpm run test:e2e

# Headed mode (see browser)
pnpm run test:e2e -- --headed

# Debug mode (pause on failure)
pnpm run test:e2e -- --debug
```

Common Tasks

Create a Database Migration

After modifying `packages/database/prisma/schema.prisma`:

```
cd packages/database
npx prisma migrate dev --name describe_your_change
npx prisma generate
```

Reset Database (DEV ONLY — deletes all data)

```
cd packages/database
npx prisma migrate reset
# Type 'y' to confirm
```

Lint & Type Check

```
pnpm run lint          # ESLint + Prettier check
pnpm run lint -- --fix  # Auto-fix formatting
pnpm run type-check    # TypeScript strict mode check
```

Build for Production

```
pnpm run build
# Builds apps/web/.next and apps/api/dist
```

VS Code Setup

Recommended Extensions

Create `.vscode/extensions.json` (commit this file):

```
{
  "recommendations": [
```

```
    "dbaeumer.vscode-eslint",
    "esbenp.prettier-vscode",
    "bradlc.vscode-tailwindcss",
    "prisma.prisma",
    "ms-vscode.vscode-typescript-next"
  ]
}
```

Install all: `Ctrl/Cmd+Shift+P` → "Extensions: Show Recommended Extensions" → Install Workspace Recommendations

Workspace Settings

Create `.vscode/settings.json`:

```
{
  "editor.formatOnSave": true,
  "editor.defaultFormatter": "esbenp.prettier-vscode",
  "editor.codeActionsOnSave": {
    "source.fixAll.eslint": true
  },
  "typescript.tsdk": "node_modules/typescript/lib",
  "tailwindCSS.experimental.classRegex": [
    ["cva\\(([^)]*)\\)", "[\"'`]([^\"'`]*).*?[\"'`]"]
  ]
}
```

Troubleshooting

PostgreSQL Connection Failed

Error: `Can't reach database server at localhost:5432`

1. Check if PostgreSQL is running: `pg_isready -h localhost`
2. If Docker: `docker ps | grep bilko-postgres` — if not running: `docker start bilko-postgres`
3. Verify credentials in `apps/api/.env` match database setup
4. Check port 5432 not in use by another process: `lsof -i :5432`

Port Already in Use

Error: `Port 3000 is already in use`

```
# Kill process on port 3000
lsof -ti:3000 | xargs kill

# Or use a different port
PORT=3001 pnpm run dev
```

Prisma Client Not Generated

Error: `@prisma/client did not initialize`

```
cd packages/database && npx prisma generate
```

TypeScript Errors After Pulling

```
pnpm install
cd packages/database && npx prisma generate
pnpm run type-check
```

Hot Reload Not Working (Backend)

1. Kill and restart dev server
2. Check nodemon is watching the right directory

Hot Reload Not Working (Frontend)

```
# Clear Next.js cache
rm -rf apps/web/.next
pnpm run dev
```

Environment Variables Reference

See environment-configuration.md for full reference.

Quick reference for local dev (`apps/api/.env`):

Variable	Required Locally	Default
`DATABASE_URL`	Yes	`postgresql://bilko:bilko@localhost:5432/bilko_dev`
`JWT_SECRET`	Yes	Any non-empty string
`JWT_REFRESH_SECRET`	Yes	Any non-empty string
`SENDGRID_API_KEY`	No	Empty (email sending won't work locally)
`R2_ACCESS_KEY_ID`	No	Empty (file uploads won't work locally)
`PORT`	No	4000
`NODE_ENV`	No	development
`ALLOWED_ORIGINS`	No	`http://localhost:3000`

Related Documents

- ENVIRONMENT.md — authoritative environment documentation
- Developer Onboarding Guide
- Coding Standards
- BILKO CLAUDE.md

Approval

Role	Name	Date	Signature
Author	Ops Architect	2026-02-23	
Reviewer	Tech Lead		
Approver	Alem Bašić		