

# MyLife VA Configuration and Deployment Handbook

This handbook describes how to configure, deploy, and operate the MyLife VA platform across local, staging, and production environments. It assumes you run the backend (NestJS) and the frontend (Next.js) from the same repository but ship them to separate runtimes.

## 1. Audience and Goals

- **Audience:** engineers and operators responsible for provisioning infrastructure, configuring secrets, and rolling out new releases.
- **Goals:** provide repeatable steps for setting up environments, running migrations, deploying both apps, wiring third party services (Google, Stripe, Mailgun), and handling incidents.

## 2. Repository Top-Level Map

Path	Purpose
backend/	NestJS API, Prisma schema, Stripe webhooks, Google integrations.
frontend/	Next.js 14 App Router marketing site and authenticated app.
docs/	Supporting documentation (including this handbook).
logs/	Local development log output (safe to rotate or delete).

Use Git branches per environment (for example `main` for production, `develop` for staging) and keep environment specific config in each runtime, **never** committed.

## 3. Environment Overview

Environment	Primary Use	Recommended Hosting	Notes
Local	Feature development and QA	Developer workstation	Uses local PostgreSQL, test Google project, Stripe test keys.
Staging	Internal demos, pre-prod validation	Managed Node runtime (Render, Railway, Fly.io) + Vercel preview	Mirrors production config but uses sandbox credentials.
Production	Customer traffic	Managed Node runtime behind HTTPS + Vercel production domain	Requires hardened secrets, logging, monitoring, backups.

## 4. Prerequisites

Install the following on every machine (CI runner, developer laptop, server image):

Tool	Version (minimum)	Notes

Node.js	20.x LTS	Both backend and frontend use Node 20 features.
pnpm	9.x	Package manager for both workspaces.
PostgreSQL	14+	Primary relational database.
Redis (optional)	7+	Only needed if you introduce caching or queues later.
Stripe CLI	Latest	For webhook testing locally.
Google Cloud SDK (optional)	Latest	Helpful for OAuth credential management.
Docker	Latest	Required if you use container based deploys.

Ensure outbound HTTPS access to Google APIs, Stripe, and Mailgun from every runtime.

---

## 5. Secrets and Environment Variables

### 5.1 Backend ( `backend/.env` )

Create a `.env` file per environment based on the following template. Replace placeholder values before deployment.

```
DATABASE_URL=postgresql://USER:PASSWORD@HOST:PORT/dbname?schema=public

PORT=4000
CORS_ALLOWED_ORIGINS=https://app.mylifeva.com,https://api.mylifeva.com
APP_CORS_ALLOW_CREDENTIALS=true

# JWT secrets (set unique values per environment)
JWT_SECRET=replace-with-strong-secret
JWT_EXPIRES_IN=7d
REFRESH_SECRET=replace-with-strong-secret
REFRESH_EXPIRES_IN=30d

# Google OAuth
GOOGLE_CLIENT_ID=...
GOOGLE_CLIENT_SECRET=...
GOOGLE_REDIRECT_URI=https://api.mylifeva.com/api/v1/google/oauth/callback
GOOGLE_SUCCESS_REDIRECT=https://app.mylifeva.com/settings?google=connected
GOOGLE_CALENDAR_TIMEZONE=Europe/Amsterdam

# Mailgun
MAILGUN_API_KEY=...
MAILGUN_DOMAIN=mg.example.com
MAIL_FROM="MyLife VA <no-reply@mg.example.com>"
APP_AUTH_PASSWORD_RESET_URL=https://app.mylifeva.com/reset-password

# Stripe
STRIPE_SECRET_KEY=sk_live_...
STRIPE_PRICE_ID=price_...
STRIPE_WEBHOOK_SECRET=whsec_...
```

```
STRIPE_SUCCESS_URL=https://app.mylifeva.com/billing/success  
STRIPE_CANCEL_URL=https://app.mylifeva.com/billing/cancel
```

## 5.2 Frontend ( `frontend/.env.local` )

```
NEXT_PUBLIC_API_BASE_URL=https://api.mylifeva.com/api/v1  
NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY=pk_live_...  
NEXT_PUBLIC_STRIPE_PRICE_ID=price_... (optional if you fetch from backend)
```

Keep `.env.local` out of version control. Vercel allows you to configure environment variables per environment via the dashboard or the `vercel env` CLI.

---

## 6. Local Development Setup

### 1. Clone and install dependencies

```
git clone <repo>  
cd MyLife VA  
pnpm install --filter ./backend  
pnpm install --filter ./frontend
```

### 2. Configure `.env` files

- Copy the templates above.
- Use Stripe test keys and Google OAuth credentials with `http://localhost` redirect URLs.

### 3. Provision PostgreSQL

```
createdb mylife_va
```

### 4. Run migrations and generate Prisma client

```
cd backend  
pnpm prisma migrate deploy  
pnpm prisma generate
```

### 5. Start backend

```
pnpm run start:dev
```

Alternative: use `backend/start-backend.ps1` to auto free port 4000 and open Swagger.

### 6. Start frontend

```
cd ../frontend  
pnpm dev
```

### 7. Verify

- Swagger UI: <http://localhost:4000/swagger>
  - App UI: <http://localhost:4001>
  - Run manual QA using `TESTING-GUIDE.md`.
- 

## 7. Database Lifecycle

Task	Command	Notes
Create new migration	<code>pnpm prisma migrate dev --name meaningful-name</code>	Generates SQL and updates Prisma client.
Apply migrations (CI/Prod)	<code>pnpm prisma migrate deploy</code>	Use in entrypoint or release script.
Seed data	<code>pnpm prisma db seed</code>	Current seed script is a no-op placeholder.
Inspect schema	<code>pnpm prisma studio</code>	For local debugging only.

**Deployment rule:** run `migrate deploy` before starting the new backend process. Automate this in CI or the container startup sequence.

**Backups:** enable daily point-in-time recovery on your managed PostgreSQL instance with at least 7 days retention (30 days preferred). Schedule a monthly logical dump ( `pg_dump` ) stored in encrypted object storage for disaster recovery drills.

---

## 8. Build and Deployment Flow

### 8.1 Suggested Hosting Topology

- **Frontend:** Vercel (Next.js 14 SSR/ISR support). Alternatives: Netlify, AWS Amplify.
- **Backend:** Render, Railway, Fly.io, AWS Elastic Beanstalk, or a Dockerised service behind an HTTPS reverse proxy (NGINX, Cloudflare Tunnel).
- **Database:** Managed PostgreSQL (Supabase, Neon, Render, RDS) with automated backups.
- **Object storage:** If exporting files, connect AWS S3 compatible bucket (already supported via `@aws-sdk/client-s3` ).

### 8.2 CI/CD Pipeline Outline

1. **Trigger:** Git push to `main` or release tag.

2. **CI Steps:**

- `pnpm install --frozen-lockfile`
- `pnpm --filter backend lint`
- `pnpm --filter backend test`
- `pnpm --filter frontend lint` (once ESLint config is committed)
- `pnpm --filter frontend test` (if you add Playwright or React testing)
- Build artifacts: `pnpm --filter backend build` , `pnpm --filter frontend build`

3. **Artifacts:**

- Backend `dist/`
- Frontend `.next/` (or rely on Vercel build)

4. **Deploy Jobs:**

- **Backend:** upload Docker image or Node bundle, run migrations, restart process manager (PM2, systemd, or platform restart).
- **Frontend:** trigger Vercel deployment or upload static bundle if self-hosted.

### 8.3 Backend Deployment Steps (Generic Node Host)

1. **Provision server** with Node 20, pnpm, reverse proxy.
2. **Fetch code or image** (git pull or container registry).
3. **Install dependencies**

```
pnpm install --prod
```

#### 4. Run migrations

```
pnpm prisma migrate deploy && pnpm prisma db seed
```

#### 5. Build and start

```
pnpm build  
NODE_ENV=production pnpm start:prod
```

Use PM2 or systemd to keep the process alive:

```
pm2 start dist/main.js --name mylife-api  
pm2 save
```

#### 6. Expose HTTPS via NGINX or Cloudflare. Configure reverse proxy to forward /api/v1 and /swagger .

#### 7. Configure Stripe webhook endpoint

- URL: <https://api.mylifeva.com/api/v1/billing/webhook>
- Secret: STRIPE\_WEBHOOK\_SECRET

### 8.4 Frontend Deployment Steps (Vercel)

1. Connect repository to Vercel.
2. Configure environment variables for Production and Preview ( `NEXT_PUBLIC_API_BASE_URL` , `NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY` ).
3. Set build command: `pnpm install && pnpm build` .
4. Set output directory: `.next` .
5. Configure custom domain (e.g. `app.mylifeva.com` ) and route marketing to root.
6. After deployment, smoke test:
  - Home page loads.
  - Auth flows redirect to backend.
  - Enterprise demo page accessible at `/contact` .

If self-hosting, run `pnpm build` followed by `pnpm start` . Place behind reverse proxy with caching disabled for authenticated routes.

## 9. Third Party Service Configuration

## 9.1 Google Cloud OAuth

1. Create a Google Cloud project.
2. Configure OAuth consent screen (internal or external depending on target accounts).
3. Add OAuth 2.0 Client ID (type: Web application).
4. Authorized redirect URIs:
  - o `http://localhost:4000/api/v1/google/oauth/callback` (local)
  - o `https://api-staging.mylifeva.com/api/v1/google/oauth/callback`
  - o `https://api.mylifeva.com/api/v1/google/oauth/callback`
5. Authorized JavaScript origins should include frontend origins.
6. Store Client ID and Secret in the backend environment variables.

## 9.2 Stripe

1. Create a product and recurring price in the Stripe Dashboard (test and live).
2. Capture the Price ID and set it as `STRIPE_PRICE_ID`.
3. Configure success and cancel URLs to point to your frontend billing routes.
4. Create webhook endpoint for events:
  - o `checkout.session.completed`
  - o `invoice.payment_succeeded`
  - o `customer.subscription.updated`
  - o `customer.subscription.deleted`
5. Verify webhooks locally using Stripe CLI:

```
stripe listen --forward-to localhost:4000/api/v1/billing/webhook
```

6. Make webhook handlers idempotent: store `event.id` in the database and skip processing duplicates.  
Stripe retries delivery up to several times; respond with `2xx` only after persisting results so retries stay safe.

## 9.3 Mailgun

1. Verify domain (`mg.example.com`).
2. Create API key with Send access.
3. Set `MAIL_FROM` to a verified address.
4. Update DNS records (SPF, DKIM) before going live.

---

# 10. Observability and Operations

Concern	Implementation	Actions
Logging	Winston console JSON output	Ship to your log aggregator (Datadog, Loki, CloudWatch).
Metrics	prom-client integrated (expose <code>/metrics</code> route if required)	Add middleware to serve metrics behind auth or internal network.
Health checks	<code>healthcheck</code> script and NestJS readiness at <code>/api/v1/health</code> (implement if missing)	Configure load balancer to poll every 30 seconds.
Error handling	Global exception filters + Nest logging	Set up alerting on 5xx spikes.

Rate limiting (todo)	Add Nest rate limiter or upstream (Cloudflare)	Recommended for production safety.
Request tracing	Add middleware that generates an x-Request-ID (or reuse upstream header) and log it	Forward the correlation ID to logs and responses so support can trace incidents quickly.

---

## 11. Data Protection & Compliance

- Host customer data in EU regions (for example: Render EU Central, Vercel FRA, Neon EU-West). Document the region for each managed service.
- Ensure managed PostgreSQL and object storage enable encryption at rest and in transit.
- Limit logging of PII; scrub sensitive fields before writing to structured logs.
- Offer data export and deletion workflows aligned met GDPR (een handmatige procedure is in de opstartfase acceptabel, zorg dat het proces gedocumenteerd is).
- Houd een overzicht bij van sub-processors (Google, Stripe, Mailgun) en verwerk dit in je privacybeleid.

---

## 12. Release Checklist

### 1. Code quality

- `pnpm --filter backend lint`
- `pnpm --filter backend test`
- `pnpm --filter backend test:e2e` (optional when implemented)
- `pnpm --filter frontend lint` (configure ESLint to avoid interactive prompt)

### 2. Database

- Confirm migrations are generated and committed.
- Review Prisma diff for destructive changes.
- Back up production database before applying migrations.

### 3. Secrets

- Ensure environment variables are up to date (Google, Stripe, Mailgun).
- Rotate secrets if moving between staging and production.
- Confirm Stripe webhook endpoints exist for both staging (test mode) and production (live mode), and the corresponding secrets are deployed.

### 4. Infrastructure

- Validate sufficient disk and memory on backend host.
- Confirm SSL certificates are valid.
- Check Stripe webhook endpoint status.

### 5. Smoke tests after deploy

- Sign up new user.
- Connect Google account (sandbox).
- Trigger AI assistant action.
- Create reminder, task, and sample invoice.
- Visit `/contact` to validate marketing flow.

## 13. Troubleshooting

Symptom	Likely Cause	Resolution
502 from backend	Process crashed or not listening on port	Check PM2 logs, confirm PORT matches reverse proxy upstream.
Google OAuth redirect mismatch	Incorrect redirect URI in Google console	Update OAuth client to include exact backend callback URL.
Stripe webhook signature error	STRIPE_WEBHOOK_SECRET mismatch	Recopy secret after recreating webhook endpoint.
Frontend calling wrong API	NEXT_PUBLIC_API_BASE_URL misconfigured	Update environment variable and redeploy frontend.
CORS errors in browser	Missing origin in CORS_ALLOWED_ORIGINS	Add full origin (protocol + host) and restart backend. Include Vercel preview URL ( <a href="https://*.vercel.app">https://*.vercel.app</a> ) if you rely on preview deployments.
Password reset emails not sending	Mailgun API key missing	Configure MAILGUN_API_KEY and verify domain.

## 14. Future Enhancements

- Add IaC templates (Terraform or Pulumi) for reproducible infrastructure.
- Commit ESLint configuration so CI linting is non-interactive.
- Introduce automated Playwright smoke tests for key flows.
- Extend seed script with sensible demo data for staging.
- Implement blue/green or canary deployment once traffic grows.

## 15. Reference Commands Cheat Sheet

```
# Install dependencies
pnpm install

# Backend dev server
pnpm --filter backend start:dev

# Frontend dev server
pnpm --filter frontend dev

# Build for production
pnpm --filter backend build
pnpm --filter frontend build

# Run migrations
pnpm --filter backend prisma migrate deploy
```

```
# Run Stripe webhooks locally
stripe listen --forward-to localhost:4000/api/v1/billing/webhook
```

Keep this handbook updated when architecture or external integrations change. Treat it as the source of truth for onboarding new operators and for disaster recovery rehearsals.