Manual para Desarrolladores — PenguinPath

Versión: 1.3.2
Fecha: 2025-11-13
Propósito: Guía técnica para poner en marcha, desarrollar, probar y desplegar la aplicación PenguinPath (frontend Vue.js, backend Nest.js, PostgreSQL, Prisma, contenedores Docker).

---

1. Resumen y alcance

Este documento está dirigido a desarrolladores que van a trabajar en PenguinPath — tanto en features nuevas como en mantenimiento—. Contiene instrucciones para reproducir el entorno de desarrollo, flujo de trabajo recomendado, convenciones de código, scripts útiles, despliegue y operación básica.

> Basado en el Manual de Usuario (interfaz, flujos de usuario, labs/sandbox) y la estructura del repositorio proporcionados por el equipo de producto.

---

2. Requisitos previos (local)

- Node.js >= 18 (uso de pnpm en repo)
- pnpm >= 7
- Docker >= 24 y Docker Compose V2
- PostgreSQL local (si no se usa contenedor) o acceso a la base de datos de desarrollo
- Git (repositorio remoto)
- Acceso a las credenciales/secretos (ver `.env.example`)

---

3. Estructura del repositorio (alto nivel)

```
/ (raíz)
├─ Backend/
│  ├─ src/               # Código Nest.js (módulos, controladores, servi
cios)
│  ├─ prisma/            # Esquema Prisma, migraciones, seeds
```

```
│   ├─ db/                  # scripts de BD o fixtures (si aplica)
│   ├─ test/                # pruebas unitarias/integración backend
│   ├─ Dockerfile
│   ├─ package.json
│   └─ tsconfig.json
├─ Frontend/
│   ├─ src/                 # Código Vue.js (components, views, store)
│   ├─ public/
│   ├─ Dockerfile
│   ├─ vite.config.ts
│   └─ package.json
├─ docker-compose.yml
├─ docker-compose.dev.yml
├─ docker-compose.prod.yml
├─ Documentacion/
└─ README.md
```

Observaciones: el backend usa `prisma/` (ORM) y el proyecto usa `pnpm` como gestor de paquetes.

---

4. Variables de entorno (principales)

Asegúrate de copiar y completar `.env.example` en tu entorno local (.env). Variables recomendadas: - `NODE_ENV` (development|production) - `PORT` (puerto del servidor backend) - `DATABASE_URL` (cadena de conexión PostgreSQL, p.ej. `postgresql://user:pass@host:5432/dbname?schema=public`) - `JWT_SECRET` (secreto para tokens JWT) - `JWT_EXPIRES_IN` (p. ej. 3600s) - `REDIS_URL` (si se usa cache/colas) - `FRONTEND_URL` (origen permitido para CORS) - `AWS_...` (si hay integración con AWS; credenciales no deben ponerse en repo)

Recomendación: usar Docker secrets / sistemas de secrets del proveedor (AWS Secrets Manager, Parameter Store) en producción.

---

5. Modo desarrollo — pasos rápidos

A continuación hay dos opciones: ejecutar con Docker Compose (recomendado) o ejecutar servicios por separado.

5.1 Opción A — Con Docker Compose (entorno dev)

1. Copia el ejemplo de env: `cp .env.example .env` y complétalo.

2. Levantar contenedores:

```
docker compose -f docker-compose.dev.yml up --build
```

3. Backend estará en `http://localhost:<PORT>` y frontend en `http://localhost:5173` (o el puerto definido).

5.2 Opción B — Ejecutar servicios localmente (sin contenedores)

Backend

```
cd Backend
pnpm install
cp .env.example .env && editar .env
pnpm prisma generate
pnpm prisma migrate dev --name init  # aplica migraciones
pnpm run start:dev  # arranca Nest en modo watch
```

Frontend

```
cd Frontend
pnpm install
cp .env.example .env && editar .env
pnpm run dev  # Vite
```

---

6. Prisma / Base de datos

- Generar cliente Prisma: `pnpm prisma generate`
- Ejecutar migraciones en desarrollo: `pnpm prisma migrate dev --name <desc>`
- En producción: `pnpm prisma migrate deploy`
- Para semillas: `pnpm prisma db seed` (si está configurado)
- Acceder a la consola: `pnpm prisma studio`

Backups y restauración - Backups: `pg_dump` o snapshots del servicio gestionado (RDS/GCP Cloud SQL).
- Restauración: `psql` o herramientas del proveedor.
- Programar backups automáticos en producción (SLA y retención según política).

---

7. Scripts útiles (package.json)

Revisa `Backend/package.json` y `Frontend/package.json`. Algunos scripts esperados: - `start:dev` — arranque en modo desarrollo (Nest+TS-Node) - `build` — build para producción - `start:prod` — arranque desde `dist/` - `test` / `test:watch` — ejecutar pruebas - `lint` / `format` — calidad de código

Ejemplo para backend:

```
pnpm install
pnpm run lint
pnpm run test
pnpm run build
pnpm run start:prod
```

---

8. Tests y calidad

- Backend: Jest (configuración típica en `package.json`), tests en `test/`. Ejecutar con `pnpm run test`.

- Frontend: Vitest o Jest para Vue. Ejecutar con `pnpm run test` en `Frontend/`.
- Linting: ESLint y Prettier — usar `pnpm run lint` antes de los PRs.

Cobertura: generar `pnpm run test:cov` si está configurado.

---

9. Contenedores, Dockerfiles y Docker Compose

- El repo trae `Dockerfile` en Backend y Frontend y varios `docker-compose` (dev/prod).

- Recomendaciones:
  - Dockerfile multietapa para reducir tamaño: compilar TS -> copiar `dist` a imagen de runtime (node:alpine o node:slim).

  - Evitar correr como root dentro del container (crear usuario no-root).

  - Usar variables de entorno inyectadas por `docker-compose` o secretos.

Comandos

```
# build images
docker compose -f docker-compose.prod.yml build
# levantar en prod mínimo
docker compose -f docker-compose.prod.yml up -d
# logs
docker compose -f docker-compose.prod.yml logs -f backend
```

---

10. Despliegue en AWS (recomendaciones)

En el manual de usuario se menciona AWS. Para producción recomendamos: - Usar ECS Fargate o EKS para orquestación.
- Almacenar DB en RDS (Postgres) con backups y Multi-AZ si es requerido.
- Usar ALB/NGINX para TLS (certificados ACM).
- Variables/Secrets en Secrets Manager.
- Pipelines CI/CD (GitHub Actions / AWS CodePipeline) para build, test y despliegue.

---

11. Seguridad y operaciones

- No versionar secrets.
- Configurar CORS en backend con `FRONTEND_URL`.
- Usar Helmet, rate-limiting y validaciones DTO (class-validator) en Nest.js.
- Revisar y limitar permisos en contenedores (no privilegios).
- Rotación periódica de claves JWT y credenciales.

---

12. Gestión de sandboxes/labs

El Manual de Usuario especifica que los labs se ejecutan en contenedores aislados sin privilegios root. Aspectos técnicos a cuidar: - Cada sandbox debe ejecutarse en contenedor efímero con timeouts y límites de recursos (CPU, memoria).
- No exponer puerto a host; acceso vía WebSocket/terminal con proxy que conecte solo a la sesión autorizada.
- Implementar limpieza automática (garbage collection) de contenedores inactivos.

---

13. Registro de logs, monitoreo y alertas

- Logs: estructurados en JSON, centralizados (CloudWatch / ELK / Datadog).
- Métricas: latencia, uso de CPU/mem, contadores de errores y colas (si aplica).
- Alerts: cuando el número de labs fallidos o cola de creación suba, notificar al equipo.

---

14. CI/CD (sugerencia rápida)

- Pipeline: test -> lint -> build -> deploy a staging -> run smoke tests -> deploy a prod.
- Usar artefactos versionados (tags/semver).
- Proteger ramas `main`/`prod` con PRs y revisiones.

---

15. Contribuir: flujo recomendado

1. Crear rama `feature/<descripción>` a partir de `develop` o `main`.
2. Escribir tests unitarios que cubran la funcionalidad.
3. Ejecutar lint y formateo.
4. Abrir PR y enlazar issue.
5. Cuando se apruebe, merge y crear release si aplica.

---

16. Troubleshooting común

- **La terminal del lab no conecta**: revisar logs del servicio de sandboxes, colas y si Docker host agotó recursos.
- **Errores de migración**: chequear `prisma migrate resolve` y backups antes de revertir.
- **Build fallido en CI**: reproducir localmente con `pnpm install` + `pnpm run build` dentro de la misma imagen base usada en CI.

---

17. Documentos de referencia

- Manual de Usuario (interfaz y flujos) — usado como base de requisitos funcionales.
- SAD / Documentacion/SAD.md (arquitectura de aplicación).

---

18. Lista de comprobación antes de desplegar a producción

☐ Tests unitarios e integración pasan.

☐ Migraciones aplicadas en entorno staging y validadas.

☐ Backups programados y verificados.

☐ Secrets gestionados por sistema de secretos.

☐ TLS configurado y probado.

☐ Monitoreo y alertas activas.

---

19. Contactos y soporte interno

- Equipo de Desarrollo (dueño del repo) — responsable de merges y releases.
- Soporte operativo (email en manual usuario) para incidentes: linuxapp@gmail.com

---

Notas finales

Este documento es una base operativa y técnica. Puedo ampliarlo con: - Plantillas de GitHub Actions / GitLab CI.
- Dockerfile optimizados (ejemplos multietapa).
- Scripts de backup y restore para PostgreSQL.
- Checklist de seguridad detallada.