ARCHITECTURE REFERENCE DOCUMENT

mallacademy

Multi-tenant LMS for the Mall & Retail industry.
Monorepo architecture, tech stack, and Google Cloud deployment blueprint.

Version 1.0 — MVP

Stack Next.js 15 · NestJS 11 · PostgreSQL · GCP

Build 8–10 weeks

Contents

1. Project Brief & Vision

mallacademy is a vertically-focused SaaS LMS for the mall & retail industry in Romania. The platform addresses a real market gap: lack of standardized, accessible, certifiable training for operational personnel in shopping centers — center managers, leasing, marketing, operations, security, customer service, facility management, tenant relations.

Business model is **hybrid**: B2C (individual paying learners), B2B (corporate subscriptions for mall chains and developers), digital resources marketplace (templates, guides, case studies). Architecture treats *vertical* as a first-class entity, enabling expansion to HoReCa, retail, and corporate training in Phase 2.

MVP Scope

- **Launch vertical:** Mall & Shopping Centers
- **Core categories:** Center Management, Leasing, Marketing & CX, Security & Compliance, Facility Management
- **No video** — text, images, downloads, quizzes
- **Primary language:** RO, i18n-ready for EN/IT

Non-Functional Requirements

- **Performance:** TTFB < 500ms public pages
- **Security:** GDPR, bcrypt/argon2, HTTPS-only, audit log, PCI-DSS via Stripe
- **Scale:** ~1,000 → 10,000+ concurrent users
- **Backup:** Daily, RPO 24h / RTO 4h

2. Monorepo Folder Structure

Turborepo + pnpm workspaces. Three deployable apps, three shared packages. Add complexity only when growth demands it.

```
mallacademy/
├── apps/
│   ├── web/                      # Next.js — learner + public catalog
│   │   ├── app/
│   │   │   ├── (public)/          # marketing, course catalog (SEO)
│   │   │   ├── (auth)/            # login, register
│   │   │   └── (learn)/           # logged-in learner zone
│   │   ├── features/              # by domain: course, certificate, profile
│   │   ├── components/            # app-level shared components
│   │   ├── lib/                   # api client, auth helpers
│   │   ├── providers/             # context + query providers
│   │   ├── middleware.ts
│   │   └── next.config.ts
│   ├── admin/                    # Next.js — back-office + B2B (MVP)
│   │   ├── app/
│   │   │   ├── (authoring)/       # course/lesson editor
│   │   │   ├── (companies)/       # B2B tenant management
│   │   │   ├── (vouchers)/        # discount codes
│   │   │   └── (reports)/         # completion dashboards
│   │   ├── features/
│   │   ├── components/
│   │   └── lib/
│   └── api/                      # NestJS backend
│       ├── src/
│       │   ├── modules/           # auth, users, tenants, catalog, courses,
│       │   │                      # enrollment, progress, assessment,
│       │   │                      # certification, billing, vouchers, notifications
│       │   ├── common/            # guards, interceptors, tenant middleware
│       │   ├── config/            # env validation
│       │   └── main.ts
│       ├── prisma/                # schema + migrations
│       └── test/
├── packages/
│   ├── shared/                    # types, Zod schemas, constants, utils
│   ├── ui/                        # design system (shadcn-based)
│   └── config/                    # eslint, tsconfig, tailwind, i18n
├── .github/workflows/             # CI for each app
├── .husky/                        # pre-commit hooks
├── docs/                          # brief, ADRs, gdpr.md
├── scripts/                       # seed pilot courses, db helpers
├── docker-compose.yml             # postgres + redis for local dev
├── .env.example
├── turbo.json
├── pnpm-workspace.yaml
└── package.json
```

Why this shape

Two Next.js apps (learner + admin) share one NestJS API. Admin handles back-office, authoring, and B2B for MVP. Split B2B and Marketing into separate apps only when customer demand or content velocity justifies it.

3 & 4. NestJS Module & Next.js Feature Anatomy

Every NestJS module is a **bounded context**. Every Next.js feature is a **domain unit** — organize by domain, not by technical type.

NestJS Module — `modules/courses/`

```
modules/courses/
├── courses.module.ts
├── courses.controller.ts   # HTTP
├── courses.service.ts      # logic
├── courses.repository.ts   # Prisma
├── dto/
│   ├── create-course.dto.ts
│   └── update-course.dto.ts
├── entities/
│   └── course.entity.ts
└── tests/
    └── courses.service.spec.ts
```

Next.js Feature — `features/course/`

```
features/course/
├── components/
│   ├── CourseCard.tsx
│   ├── LessonPlayer.tsx
│   └── ProgressBar.tsx
├── hooks/
│   └── useCourseProgress.ts
├── api.ts          # typed calls
├── schemas.ts      # Zod
└── types.ts        # TS types
```

When to upgrade NestJS layout

When a module passes ~15 files, split it into `domain/`, `application/`, `infrastructure/`, `presentation/` (hexagonal architecture). Not on day one.

5. The Frontend ↔ Backend Bridge

The single most important pattern in the codebase. **One Zod schema in `packages/shared/` is imported by both the NestJS DTO and the Next.js form.** Zero drift between frontend and backend.

```
# Flow of a single type from definition to consumption:

packages/shared/schemas/course.ts      # Zod schema (SOURCE OF TRUTH)
        │
        ├──→ z.infer<typeof schema>
        │
        ├──→ packages/shared/types/course.ts     # Inferred TS type
        │
        ├──→ apps/api/courses.controller.ts      # Validates request body
        │       │
        │       └──→ HTTP (typed contract)
        │               │
        └───────────────┴──→ apps/web/features/course/api.ts
                                    # Typed fetch wrapper
```

> **The rule**
> Change the schema once → types update everywhere → broken usage shows as a TypeScript error in CI before it ever reaches production.

6 & 7. Frontend & Backend Stack

Frontend — `apps/web` + `apps/admin`

Next.js 15 · React 19 · TypeScript 5 · Tailwind v4 · shadcn/ui · TanStack Query v5 · Zustand · react-hook-form · Zod · NextAuth v5 · next-intl · Lucide

TOOL	VERSION	PURPOSE
Next.js	15	React framework, App Router, Server Components
React	19	UI library
TypeScript	5.x	Type safety across stack
Tailwind CSS	v4	Utility-first styling
shadcn/ui	latest	Component primitives (packages/ui/)
TanStack Query	v5	Server state, caching, refetching
Zustand	v5	Lightweight client state (skip Redux)
react-hook-form	v7	Form management
Zod	v3	Validation — shared with backend
NextAuth.js (Auth.js)	v5	Frontend session management
next-intl	latest	i18n — RO base, EN/IT-ready
Lucide React / date-fns	latest	Icons + RO-locale date formatting

Backend — `apps/api`

NestJS 11 · Prisma 6 · Zod · Passport JWT · BullMQ · Throttler · Pino · PDFKit · Stripe SDK · Swagger · helmet

TOOL	VERSION	PURPOSE
NestJS	11	API framework, DI, modules
Prisma	v6	ORM + migrations
Zod	v3	DTO validation (shared with frontend)
Passport (JWT)	latest	Authentication strategy
BullMQ	v5	Background jobs — certs, emails, webhooks
@nestjs/throttler	latest	Rate limiting per tenant + user
@nestjs/swagger	latest	Auto-generated OpenAPI docs
Pino	v9	Structured JSON logging
PDFKit	latest	PDF certificate generation
Stripe SDK / Resend	latest	Payments + transactional email
helmet	latest	Security headers

8, 9, 10. Database, GCP & External Services

Database, Cache & Storage

TOOL	CONFIGURATION	PURPOSE
Cloud SQL Postgres	v16, Enterprise, db-custom-2-7680, europe-west1	Primary database
Row-Level Security	On all tenant tables	Multi-tenancy at DB layer
Cloud SQL Auth Proxy	Sidecar in Cloud Run	Secure DB connection
Upstash Redis	Serverless, pay-per-request	Cache, BullMQ, sessions, rate-limits
Cloudflare R2	S3-compatible, zero egress fees	PDFs, course assets, uploads
pgvector	Postgres extension (Phase 2+)	AI search/recommendations

Google Cloud Infrastructure

SERVICE	PURPOSE
Cloud Run (×3 services)	web, admin, api — autoscaling containers
Artifact Registry	Docker image storage
Secret Manager	All secrets (DATABASE_URL, STRIPE_KEY, JWT_SECRET, etc.)
Cloud Logging + Monitoring	Centralized logs, uptime checks, alerts
IAM + Workload Identity Federation	GitHub Actions auth without service account keys
Cloud Billing Alerts	Budget alerts at $50 / $100 / $200 — set day one

Cloud Run Service Configuration

SERVICE	SUBDOMAIN	MIN	MAX	MEMORY	CPU
web	app.mallacademy.ro	1	10	1 GB	1
admin	admin.mallacademy.ro	0	5	512 MB	1
api	api.mallacademy.ro	1	20	1 GB	2

Region: europe-west1 (Belgium). **Edge:** Cloudflare in front of Cloud Run — free CDN/DNS/WAF/DDoS.

External Services

SERVICE	FREE TIER	PURPOSE
Stripe	No fixed cost — % per transaction	Payments, subscriptions, B2B billing
Resend	3,000 emails/month free	Transactional email
Sentry	5K errors/month free	Error tracking (frontend + backend)
PostHog Cloud	1M events/month free	Product analytics, feature flags
Cloudflare	Free tier	CDN, DNS, WAF, DDoS

11, 12, 13. DevOps, Multi-Tenancy & Deployment

DevOps & Testing

Turborepo | pnpm | Docker | GitHub Actions | Husky | ESLint | Prettier | commitlint | Vitest | Jest | Playwright | Supertest | MSW

TOOL	PURPOSE
Turborepo + pnpm	Monorepo task runner + workspace package manager
Docker	Containerization for Cloud Run
GitHub Actions + WIF	CI/CD with Workload Identity Federation (no JSON keys)
Husky + lint-staged + commitlint	Pre-commit hooks, Conventional Commits
ESLint + Prettier	Shared config in packages/config/
Vitest / Jest / Supertest	Unit + integration tests (Vitest for Next, Jest for Nest)
Playwright / MSW	E2E tests (post-launch) + API mocking

Multi-Tenancy Architecture

The single most important architectural decision. Get this right in Prisma migration #1.

```
# Every business table has:
tenant_id  UUID NOT NULL

# Postgres RLS policy:
CREATE POLICY tenant_isolation ON courses
  USING (current_setting('app.current_tenant_id')::uuid = tenant_id);

# NestJS interceptor (runs on every request):
SET app.current_tenant_id = '${tenantFromJWT}';

# Forgetting a WHERE clause CANNOT leak data across tenants — the DB refuses.
```

Deployment Flow

```
git push origin main
       ↓
GitHub Actions  →  Lint + type-check + test (parallel via Turborepo)
       ↓
   Build Docker image (only changed apps — turbo cache)
       ↓
   Push to Artifact Registry  →  Deploy to Cloud Run
       ↓
   Run Prisma migrations (api only, separate step)  →  Smoke test
       ↓
   ✓ Live in ~2-4 minutes
```

> **Migration safety**
>
> Run Prisma migrations as a separate Cloud Build step, never inside the app's startup script. Two Cloud Run instances racing to migrate the DB will corrupt the schema.

14, 15, 16. Costs, Setup & Gotchas

Monthly Cost Estimate (MVP, low traffic)

ITEM	COST / MONTH
Cloud Run × 3	$0–20
Cloud SQL Postgres (2 vCPU, 7.5GB)	$55–70
Upstash Redis	$0–10
Cloudflare R2 + Artifact Registry + Secret Manager	$1–8
Cloud DNS (or Cloudflare = $0)	$0.40
Resend / Sentry / PostHog / Cloudflare	$0 (free tiers)
Stripe	% per transaction

GCP $300 free credit covers the entire 8–10 week MVP build. Real billing starts at launch.

Day-One Setup Checklist

1. Create GCP projects: `mallacademy-prod`, `mallacademy-staging`
2. Enable APIs: Cloud Run, Cloud SQL, Artifact Registry, Secret Manager
3. Set up Workload Identity Federation for GitHub Actions
4. Provision Cloud SQL (smallest tier — upgrade later)
5. Create Artifact Registry repos: web, admin, api
6. Init Cloudflare zone for `mallacademy.ro`, DNS → Cloud Run
7. Set Cloud Billing alerts: $50 / $100 / $200
8. Scaffold monorepo: `pnpm create turbo`
9. Add `Dockerfile` + `output: 'standalone'` per app
10. Deploy "hello world" to Cloud Run for each app
11. Set up Sentry, Resend, Stripe (test mode), PostHog
12. Write ADR `docs/adrs/0001-multi-tenancy-strategy.md`

Build order: auth → users → tenants → catalog → courses → enrollment → progress → assessment → certification → billing → vouchers → notifications.

Critical Gotchas

1. Cloud Run cold starts

Set `min-instances: 1` on `web` and `api`. The $5/month is worth it — first user shouldn't wait 3s for the homepage.

2. Prisma migrations on Cloud Run

Run as a separate Cloud Build step, never in app startup. Racing instances corrupt the schema.

3. tenant_id enforcement

Build the NestJS interceptor before your second module. Retrofitting multi-tenancy after 5 modules and real data is hours of work, high risk.

4. PORT env var + Next.js standalone

Cloud Run sets `$PORT` (default 8080) — `app.listen(process.env.PORT || 8080)`. Set `output: 'standalone'` in `next.config.ts` for ~40% smaller images and faster cold starts.

5. Never commit secrets

Use Secret Manager — mount as env vars at Cloud Run deploy time. Rotate keys without redeploying.