TECHNICAL BRIEFING

Engineering Standards & Architecture Patterns

Modulifyr · Kathmandu, Nepal · 2026

Engineering Philosophy

Modulifyr operates under a strict engineering-first philosophy. Every technical decision is evaluated against three criteria: **long-term maintainability**, **security posture**, and **operational fit** for the client organization. We reject hype-driven technology choices and favor battle-tested patterns with clear migration paths.

Core Engineering Standards

Code Quality

- All code is reviewed by a senior engineer before merging
- TypeScript enforced across all JavaScript/Node.js projects
- Linting (ESLint/Biome) and formatting (Prettier) on every commit
- Cyclomatic complexity limits enforced via static analysis
- Test coverage targets: 80% unit, 60% integration minimum

Version Control & CI/CD

- Git trunk-based development with feature flags for large changes
- All deployments via CI/CD — no manual production deploys
- GitHub Actions pipelines: lint → test → build → deploy
- Environment parity: dev / staging / production always in sync
- Automated dependency updates via Dependabot

Security Standards

- OWASP Top 10 review on every production system
- Secrets managed via environment variables — never hardcoded
- HTTPS enforced on all endpoints; HSTS enabled
- SQL injection prevention via parameterized queries / ORMs
- Regular dependency vulnerability scans (npm audit, pip audit)

Documentation Standards

- All public APIs documented via OpenAPI 3.0 spec
- Architecture Decision Records (ADRs) for all major decisions
- README files with local dev setup, environment vars, and deploy instructions
- Data flow diagrams for all systems handling PII
- Runbooks for all production incident scenarios

Modular Architecture in Practice

The defining characteristic of Modulifyr's approach is the systematic decomposition of systems into independently deployable, testable, and maintainable modules. This is not simply folder organization — it is enforced at the dependency, data, and API boundary level.

Layer	Responsibility
Presentation Layer	React / Next.js components organized by domain, not by type. Server Components for data-fetching; Cli
Application Layer	Use-case handlers, form validation, and orchestration logic. No direct database access. Pure functions w
Domain Layer	Business entities, rules, and value objects. Zero external dependencies. 100% unit testable.
Infrastructure Layer	Database adapters, API clients, email providers, and storage. Swappable implementations behind interfa

Performance Standards

Metric	Target	Measurement
Core Web Vitals (LCP)	**< 2.5s**	Google Lighthouse
API Response Time (p95)	**< 300ms**	Production monitoring
Time to First Byte (TTFB)	**< 200ms**	Server-side
Database Query Time (p95)	**< 100ms**	Query profiling
Uptime SLA (production)	**99.9%**	30-day rolling
Error Rate	**< 0.1%**	Server error logs

Delivery Process Summary

#	Phase	Duration	Activities
01	**Discovery**	2–4 weeks	Embed with client team, map workflows, define success metrics, write discovery report
02	**Architecture**	3–6 weeks	Design modular structure, data schemas, API contracts, integration specs
03	**Development**	Iterative	Sprint-based builds with weekly demos, code review, documentation
04	**QA**	Parallel	Automated + manual testing, security audit, performance benchmarking

05	**Deployment**	1–2 weeks	Cloud infrastructure setup, CI/CD pipeline, production go-live
06	**Support**	Contractual	SLA monitoring, security patches, feature iterations, retainer options

Request a Technical Deep-Dive

Discuss specific architectural requirements with our lead architect.

modulifyr.vercel.app · contact@modulifyr.com