Enterprise Architecture Guidelines

Spyglass Realty Technology Standards

Version: 1.0

Applies to: Apps and services built using Claude/LLMs, Replit, Render, Vercel, Lovable, and related automation/AI tooling

Owner: Architecture Council

Last Updated: February 2026

Table of Contents

1. Purpose and Outcomes

These guidelines ensure that every application built at Spyglass Realty:

- Is **secure by default**
- Is **observable** (logs/metrics/traces)
- Has **clear boundaries** (modular, maintainable)
- Uses **consistent patterns** across platforms (Render/Vercel/etc.)
- Can scale from prototype to production with minimal rework
- Meets **data governance** and **AI governance** expectations

2. Architecture Principles (Non-Negotiables)

1. **API-first boundaries:** Services communicate via versioned APIs/contracts.
2. **Least privilege everywhere:** Identity, network, data access.
3. **Everything is observable:** Structured logs + metrics + tracing; SLOs.
4. **Automate the boring parts:** CI/CD, infra, backups, drift detection.
5. **Stateless compute by default:** State lives in managed stores (DB/object).
6. **Standardize before you scale:** Reuse reference architectures/paved roads.
7. **Security is a feature:** Threat model for anything with customer data.
8. **Fail safely:** Timeouts, retries, idempotency, circuit breakers.
9. **Cost-aware design:** Budgets, alerts, right-sizing, avoid surprise bills.
10. **AI is a system dependency:** Treat LLMs as probabilistic components.

3. Architecture Domains

Every app/service must have decisions recorded in these domains:

3.1 Business and Product

- Problem statement, users, critical journeys
- Availability needs (e.g., '99.9% during business hours')
- RTO/RPO targets (recovery time and recovery point)

3.2 Application

- Service boundaries (monolith vs modular monolith vs microservices)
- API style (REST/GraphQL/events), versioning strategy
- Dependency inventory (internal/external)

3.3 Data

- Data classification (public/internal/confidential/restricted)
- System of record per entity
- Retention and deletion rules
- Backups + restore drills

3.4 Technology and Infrastructure

- Runtime, hosting (Render/Vercel), regions, network posture
- Secrets management approach
- CI/CD pipeline and environment promotion rules

3.5 Security and Compliance

- Threat model, authn/authz, audit trails
- Vulnerability management and patch policy
- Privacy considerations (PII handling)

3.6 AI/Automation

- Which LLMs/tools are used and why
- Prompting, evaluation, guardrails, data leakage protections
- Human-in-the-loop requirements for high-risk actions

4. Standard Reference Architecture

Use this unless you have a strong reason not to.

4.1 Typical Service Layout

Layer	Technology	Notes
Frontend	Vercel (Next.js/React)	SSR/ISR, edge functions
Backend API	Render (container/web service)	Stateless, health checks
Worker/Jobs	Render background worker	Queue consumer
Database	Managed Postgres	+ Redis for cache/queue
Object Storage	S3-compatible	For files
Auth	OIDC/OAuth2 provider	Or managed auth
Observability	Central logging + metrics	Structured JSON logs
CI/CD	GitHub Actions	Environment promotion

4.2 Environments

Environment	Purpose
dev	Development and testing
staging	Pre-production validation
prod	Production

Each environment must have:

- Separate DBs and secrets
- Separate external integrations (or safe sandboxes)
- Clear promotion gates

4.3 Networking

- Public ingress only where necessary
- Admin endpoints behind auth/VPN/IP allowlist
- Egress controls for sensitive systems if feasible

5. Paved Road Standards

Do this by default.

5.1 APIs

- Consistent resource naming (/users/{id})
- Pagination, filtering, sorting conventions
- Idempotency keys for POST where appropriate
- OpenAPI for REST, Schema registry for events
- Additive changes without version bump; breaking changes require new version + deprecation period

5.2 Reliability

- Timeouts everywhere (client + server)
- Retries with backoff (only for safe/idempotent operations)
- Circuit breakers for flaky dependencies
- Rate limiting per user/API key
- SLOs for key endpoints (start simple)

5.3 Security Baseline

- TLS everywhere
- Secrets never in code or logs
- Principle of least privilege for DB/users/tokens
- Input validation + output encoding
- OWASP Top 10 coverage (at minimum)

5.4 Data Baseline

- Encrypt in transit; encrypt at rest where supported
- Log redaction for PII/secrets
- Backups + restore test cadence (e.g., monthly)
- Migrations are automated and reversible where possible

5.5 Observability Baseline

- Structured logs (JSON), request IDs, correlation IDs
- Key metrics: latency, error rate, throughput, saturation
- Tracing across API - DB - external calls (where feasible)
- Alerts that reflect user impact (not noise)

5.6 DevEx and Maintainability

- One-command local run
- Standard repo layout

- Automated lint/test on PR
- ADRs for significant architectural decisions

6. Repository and Documentation Standards

6.1 Required Files Per Repo

File	Purpose
README.md	How to run, deploy, test
ARCHITECTURE.md	C4-style overview + diagrams
SECURITY.md	Data classification, threat model summary
RUNBOOK.md	How to operate, common incidents, rollbacks
ADR/	Architecture Decision Records directory
openapi.yaml	API contracts
CHANGELOG.md	Version history (optional but recommended)

6.2 Architecture Documentation (Minimum)

Use a lightweight C4 model:

- **Context diagram:** Who uses it, external systems
- **Container diagram:** Frontend/backend/db/queues
- **Component diagram:** Only if needed
- **Deployment diagram:** Render/Vercel/resources by environment

7. Governance: How Decisions Get Made

7.1 Architecture Council (Lightweight)

Meets as needed for:

- New production services
- Major data/security/AI changes
- Significant spend changes

Outputs: Approved architecture summary + ADR links

7.2 Architecture Decision Records (ADR)

Rule: Any decision that changes cost/risk/maintainability needs an ADR.

ADR Template:

- Context
- Decision
- Alternatives considered
- Consequences

- Rollback plan

8. AI + Automation Architecture

LLMs are powerful but probabilistic. Treat them like an external dependency.

8.1 Approved LLM Use-Cases

- Drafting content with human review
- Code generation with tests and review
- Data transformation with validation
- Support copilots (guarded + audited)
- Workflow automation (only with safe permissions)

8.2 High-Risk Actions Require Guardrails

If an LLM can send emails/messages, modify production data, change permissions, or trigger deployments, you MUST implement:

- **Human-in-the-loop approval**
- **Scoped credentials** (least privilege)
- **Audit logs** (who/what/when)
- **Policy checks** (deny lists, allow lists)

8.3 Prompt and Data Protection

- Never send secrets, API keys, or restricted PII to an LLM unless you have an explicit policy and understand retention/training implications
- Use a 'prompt firewall': Input validation, Output validation (schemas, allowlists), Injection defenses

8.4 Evaluation and Quality

- Define success metrics (accuracy, hallucination rate, refusal correctness)
- Use test sets of representative prompts
- Add regression checks whenever prompts/tools change

9. Platform Guidance

9.1 Vercel (Frontend + Edge)

Use for: Frontend hosting, SSR/ISR, Edge functions (lightweight, low-latency)

Avoid: Long-running jobs, Heavy compute or large memory workloads

9.2 Render (APIs, Workers, Managed Services)

Use for: Backend services and background workers, Managed Postgres/Redis

Standards: Separate services for API and worker when job volume matters, Environment variables via platform secret store, Health checks + graceful shutdown

9.3 Replit (Prototyping)

Use for: Rapid prototypes and spikes, Internal demos

Promotion rule: Before production, move to standard repo + CI + proper secrets handling

9.4 Lovable (Rapid UI/App Generation)

Use for: Rapid UI iteration and MVP scaffolding

Before production: Align codebase with repo standards, Add tests, CI, security controls, and observability

10. Security Checklist (Minimum Bar for Production)

- ■ Threat model completed (basic is fine)
- ■ Authn/authz implemented and tested
- ■ Secrets stored in managed secret store (not code)
- ■ Input validation + rate limiting in place
- ■ Logs redact PII/secrets
- ■ Dependency scanning enabled
- ■ Backup/restore plan documented
- ■ Incident runbook created

11. Release Engineering (CI/CD)

11.1 Pipeline Requirements

- PR checks: lint + unit tests + security scan
- Build once, deploy same artifact across environments
- Automated rollback path (previous version)
- DB migrations run via controlled step

11.2 Deployment Strategy

- Start with rolling deploys
- For risky changes, use canary or blue/green if supported

12. Integration Patterns

Prefer:

- **Events** for decoupling when many consumers exist
- **Webhooks** for external triggers
- **Queues** for retries and smoothing bursts
- **Sync APIs** for simple request/response

Rules:

- All external calls must have timeouts + retries + dead-letter behavior where relevant
- Avoid point-to-point spaghetti; document every integration in the context diagram

13. Cost Management

- Tag resources (app, env, owner)

- Budget alerts per environment
- Track unit economics: cost per request/job/user
- Avoid always-on high-tier resources for non-prod

14. Reference Links

Enterprise Architecture

- **TOGAF (The Open Group):** https://www.opengroup.org/togaf
- **ISO/IEC/IEEE 42010:** https://www.iso.org/standard/50508.html

Cloud and Reliability

- **NIST SP 800-53 (Security Controls):** https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final
- **NIST SP 800-190 (Container Security):** https://csrc.nist.gov/publications/detail/sp/800-190/final
- **Google SRE Book:** https://sre.google/sre-book/table-of-contents/

Application Security

- **OWASP Top 10:** https://owasp.org/www-project-top-ten/
- **OWASP ASVS:** https://owasp.org/www-project-application-security-verification-standard/
- **OWASP API Security Top 10:** https://owasp.org/www-project-api-security/

Architecture Documentation

- **C4 Model:** https://c4model.com/
- **arc42 Templates:** https://arc42.org/

Cloud-Native and Operations

- **CNCF Cloud Native Landscape:** https://landscape.cncf.io/
- **OpenTelemetry (Observability):** https://opentelemetry.io/

Modern App Design

- **The Twelve-Factor App:** https://12factor.net/

Repliers API (Real Estate)

- **Geo-spatial Filtering:** https://help.repliers.com/en/article/filtering-listings-geo-spatially-using-the-map-parameter-7sorw0/
- **GeoJSON Polygons:** https://help.repliers.com/en/article/handling-detailed-geojson-polygons-in-repliers-api-1lcc5pm/
- **Aggregates:** https://help.repliers.com/en/article/using-aggregates-to-determine-distinct-values-for-filters-and-parameters-c88csc/
- **Contains/NotContains Filtering:** https://help.repliers.com/en/article/using-contains-and-notcontains-prefixes-in-api-filters-c2nga9/
- **Listings API:** https://help.repliers.com/en/article/listings-api-suwj05/