Claude Code Project

File System Standard

Version 1.0 — Anti-fragile, stack-agnostic, Claude-native

Design Principles

Every directory has a single, unambiguous purpose. If one component fails or is missing, the rest continues to function. Configuration is separated from logic. Secrets never touch source control. Claude's context is explicit, versioned, and machine-readable.

Top-Level Structure

```
project-root/
■
■■■ .claude/          # Claude Code runtime context
■■■ docs/             # Human and AI documentation
■■■ src/              # Application source code
■■■ tests/            # All test types
■■■ scripts/          # Automation and utility scripts
■■■ config/           # Environment-specific configuration
■■■ logs/             # Runtime logs  (gitignored)
■■■ tmp/              # Ephemeral working files (gitignored)
■
■■■ .env.example      # Environment variable template (committed)
■■■ .env              # Actual secrets (gitignored)
■■■ .gitignore
■■■ README.md
■■■ CHANGELOG.md
■■■ LICENSE
```

.claude/ — Claude Code Context Engine

This folder is the brain Claude reads before acting. Keep it lean and accurate. Every file here directly shapes how Claude understands and operates on the project.

```
.claude/
■■■ CLAUDE.md           # PRIMARY: project overview, goals, architecture
■■■ commands/           # Custom slash commands
■   ■■■ review.md       # /project:review
■   ■■■ deploy.md       # /project:deploy
■   ■■■ debug.md        # /project:debug
■■■ context/            # Supplemental context files
■   ■■■ architecture.md # System architecture narrative
■   ■■■ decisions.md    # Architecture Decision Records (ADRs)
■   ■■■ stack.md        # Tech stack + versions + rationale
■   ■■■ gotchas.md      # Known traps and workarounds
■■■ settings.json       # Claude Code project settings
```

CLAUDE.md — What to Include

- Project name and one-paragraph purpose
- Current phase / sprint goal
- Primary entry points (which file Claude should read first)
- Commands to run the project (npm run dev, python main.py, etc.)
- Files and directories Claude must NOT modify

- Key constraints: approved libraries, test rules, style requirements

decisions.md — ADR Format

```
## ADR-001: Use MongoDB over PostgreSQL
**Date:** 2025-01-15  |  **Status:** Accepted
**Reason:** Document schema varies per record; relational overhead unjustified
**Consequences:** No joins; denormalize intentionally
```

---

docs/ — Documentation

```
docs/
■■■ README.md               # Docs index / navigation guide
■■■ architecture/
■   ■■■ overview.md         # System diagram + narrative
■   ■■■ data-flow.md        # How data moves through the system
■   ■■■ diagrams/           # Mermaid .md files or SVGs
■■■ api/
■   ■■■ endpoints.md        # API reference or OpenAPI spec
■   ■■■ auth.md             # Authentication flows
■■■ deployment/
■   ■■■ local-setup.md      # Dev environment bootstrap
■   ■■■ production.md       # Production deployment runbook
■   ■■■ rollback.md         # How to roll back safely
■■■ guides/
    ■■■ contributing.md     # Code style, PR process
    ■■■ troubleshooting.md  # Common failures + fixes
```

---

src/ — Source Code

Structure varies by stack, but the separation principle is constant: no business logic in entry points, no I/O in domain logic.

Node / Express / TypeScript:

```
src/
■■■ app.ts              # App factory (no listen() here)
■■■ server.ts           # Entry point only
■■■ routes/             # Route definitions (thin)
■■■ controllers/        # Request/response handlers
■■■ services/           # Business logic (pure, testable)
■■■ models/             # Data models / schemas
■■■ middleware/          # Express middleware
■■■ utils/              # Stateless helper functions
■■■ types/              # TypeScript types and interfaces
■■■ constants/          # App-wide constants
```

Python:

```
src/
■■■ main.py             # Entry point only
■■■ app/
■   ■■■ api/            # Route handlers
■   ■■■ services/       # Business logic
■   ■■■ models/         # Data models
■   ■■■ utils/          # Helpers
■■■ core/
    ■■■ config.py       # Config loader (reads from env)
    ■■■ logging.py      # Logging setup
```

---

tests/ — Test Suite

```
tests/
```

```
■■■ unit/              # Fast, isolated, no I/O
■■■ integration/       # Tests with real DB / external services
■■■ e2e/               # End-to-end flows
■■■ fixtures/          # Shared test data and mocks
■■■ helpers/           # Test utilities and factories
■■■ conftest.py        # (Python) or jest.setup.ts
```

Rule: each test file mirrors the source file it tests. src/services/emailService.ts → tests/unit/services/emailService.test.ts. Tests must run independently — no shared mutable state between tests.

---

scripts/ — Automation

```
scripts/
■■■ setup.sh           # One-command dev environment bootstrap
■■■ seed.py / seed.js  # Database seeding
■■■ migrate.sh         # DB migration runner
■■■ lint.sh            # Linting shortcut
■■■ deploy/
    ■■■ build.sh
    ■■■ push.sh
```

All scripts: fail loudly on error (set -e), log what they are doing, accept --dry-run where destructive operations are involved.

---

config/ — Configuration

```
config/
■■■ default.json       # Baseline config (no secrets)
■■■ development.json   # Dev overrides
■■■ production.json    # Prod overrides (no secrets — use env vars)
■■■ test.json          # Test overrides
```

config/ holds structure and defaults only. Secrets live in .env, loaded at runtime via dotenv or equivalent.

---

logs/ — Runtime Logs (gitignored)

```
logs/
■■■ app.log            # General application log
■■■ error.log          # Errors only
■■■ debug.log          # Verbose debug (dev only)
```

Use structured JSON logging — one JSON object per line. Rotate logs; never let them grow unbounded. Logs are always gitignored.

Root Files

File	Purpose
`README.md`	Project intro, quickstart, links to docs
`CHANGELOG.md`	Version history in Keep a Changelog format
`.env.example`	All required env vars with placeholder values — committed
`.env`	Actual secrets — NEVER committed
`.gitignore`	Standard + project-specific ignores
`LICENSE`	License file

.gitignore Essentials

```
.env
logs/
tmp/
__pycache__/
*.pyc
node_modules/
dist/
build/
.DS_Store
*.log
```

Anti-Fragility Checklist

Every project should satisfy these before first commit:

- ■ CLAUDE.md exists and is accurate
- ■ .env.example documents every required variable
- ■ scripts/setup.sh bootstraps a clean machine in one command
- ■ No secrets in source control (verify: git grep -r 'password|secret|token' src/)
- ■ Logs write to logs/ and are gitignored
- ■ Tests run independently — no shared mutable state
- ■ Each script exits non-zero on failure and logs the reason
- ■ docs/deployment/rollback.md exists before any production deploy

---

Summary — Quick Reference Tree

```
project-root/
■■■ .claude/
■   ■■■ CLAUDE.md
■   ■■■ commands/
■   ■■■ context/
■   ■■■ settings.json
■■■ docs/
■   ■■■ architecture/
■   ■■■ api/
■   ■■■ deployment/
■   ■■■ guides/
■■■ src/
■■■ tests/
■■■ scripts/
■■■ config/
■■■ logs/              ← gitignored
■■■ tmp/               ← gitignored
■■■ .env               ← gitignored
■■■ .env.example
■■■ .gitignore
■■■ README.md
■■■ CHANGELOG.md
```

This structure is stack-agnostic. Adapt src/ to your runtime. The .claude/, docs/, tests/, scripts/, and config/ layers remain constant across all projects.