Twelve-Factor App Methodology Implementation

This document details how each of the twelve factors is specifically implemented in our Healthcare Information Exchange SOA system.

1. Codebase

One codebase tracked in version control, many deploys

- Each service (Patient, Provider, Appointment, Authentication, Audit, Analytics) has its own dedicated Git repository
- Main branches: `main`, `develop`, `feature/*`, `release/*`, `hotfix/*`
- Consistent deployment to dev, staging, and production environments from the same codebase
- Git hooks to enforce code quality and standards

2. Dependencies

Explicitly declare and isolate dependencies

- Maven/Gradle dependency management with explicit versioning
- No reliance on system-wide packages
- All dependencies declared in `pom.xml` or `build.gradle` files
- Docker containers to isolate runtime environment
- Dependency vulnerability scanning in CI pipeline
- Separate `dependencies` and `devDependencies` clearly

3. Config

Store config in the environment

- Environment-specific configuration stored in environment variables
- Sensitive values (passwords, API keys) stored in HashiCorp Vault
- Configuration files in YAML format with environment variable interpolation
- No hardcoded configuration values in code
- Config validation on application startup
- Centralized configuration server (Spring Cloud Config) with Git backend

4. Backing Services

Treat backing services as attached resources

- All external services (databases, caches, message brokers) accessed via URLs
- Service connection details stored in environment variables
- Ability to swap backing services without code changes
- Health checks for all backing services
- Circuit breakers (with Resilience4j) for graceful handling of backing service failures
- Consistent database access via JPA regardless of database vendor

5. Build, Release, Run

Strictly separate build and run stages

- CI/CD pipeline with distinct stages:
  - Build: Compile code, run tests, create artifacts
  - Release: Combine artifacts with environment config
  - Run: Execute application with immutable release
- Docker images built once, promoted through environments
- Artifact versioning with semantic versioning
- Immutable releases, no changes to deployed code
- Blue/green deployments for zero-downtime releases

6. Processes

Execute the app as one or more stateless processes

- Stateless services that store no local state between requests
- Session data stored in Redis, not in local memory
- Persistent data always written to backing services
- No sticky sessions, enabling easy horizontal scaling
- Graceful handling of process failures
- Designed for concurrent execution of multiple instances

7. Port Binding

Export services via port binding

- Each service exports HTTP API on configured port
- Self-contained services with embedded servers (Spring Boot)
- No reliance on external application servers
- Dynamic port allocation in container environments
- Health and metrics endpoints on dedicated ports
- API gateway for unified external access

8. Concurrency

Scale out via the process model

- Horizontal scaling through multiple service instances
- Workload distributed through load balancing
- Kubernetes Horizontal Pod Autoscaler for automatic scaling
- Share-nothing architecture allowing concurrent instances
- Thread pools and async processing for optimized resource usage
- Event-driven architecture for parallel processing

9. Disposability

Maximize robustness with fast startup and graceful shutdown

- Fast startup times (<10 seconds) for rapid scaling
- Graceful shutdown handling in-flight requests
- Kubernetes readiness and liveness probes
- Connection pooling for efficient resource utilization
- Crash-only software design principles
- Idempotent operations to handle retries safely

10. Dev/Prod Parity

Keep development, staging, and production as similar as possible

- Containerized environments for consistent runtime
- Same backing services in all environments (with different scales)
- Infrastructure as Code for consistent environment provisioning
- Continuous deployment to ensure minimal drift between environments
- Development database populated with anonymized production data
- Consistent monitoring across all environments

11. Logs

Treat logs as event streams

- Logs written to stdout/stderr
- Centralized log collection with ELK stack
- Structured logging in JSON format
- Consistent correlation IDs across service boundaries
- Log levels configurable without code changes
- Separation of application logs from audit logs

12. Admin Processes

Run admin/management tasks as one-off processes

- Database migrations as separate processes (Flyway/Liquibase)
- Scheduled tasks using Kubernetes CronJobs
- Admin functionality exposed through secure API endpoints
- Maintenance scripts checked into version control
- Runbook documentation for operational procedures

- CLI tools for ad-hoc administrative tasks

Implementation Tools and Technologies

Factor	Implementation Technologies
Codebase	GitHub, Git Flow
Dependencies	Maven/Gradle, Docker
Config	Spring Cloud Config, HashiCorp Vault
Backing Services	PostgreSQL, MongoDB, Redis, Kafka
Build, Release, Run	Jenkins/GitHub Actions, JFrog Artifactory
Processes	Spring Boot, Kubernetes
Port Binding	Spring Boot embedded servers
Concurrency	Kubernetes HPA, async processing
Disposability	Graceful shutdown hooks, readiness probes
Dev/Prod Parity	Docker, Terraform, Kubernetes
Logs	ELK Stack, Fluentd
Admin Processes	Flyway, Kubernetes Jobs