Contents

Code Quality Gates Documentation

Gate Overview

The code quality validation system consists of multiple gates that ensure proper implementation of logging, error handling, and reliability patterns.

1. Logs Searchable/Available Gate

Purpose

Ensures that application logs are properly implemented, searchable, and available for debugging and monitoring.

Key Features

- Accepts both structured and standard logging formats

- Validates logging configuration and setup
- Checks for proper log formatting and content
- Ensures logs are searchable and accessible

Validation Patterns

Standard Logging Patterns

```
- log\.(info|debug|error|warning|critical)\s*\(
- logger\.(info|debug|error|warning|critical)\s*\(
- console\.(log|info|debug|error|warn)\s*\(
- System\.(out|err)\.print(ln)?\s*\(
- Console\.(WriteLine|Write)\s*\(
- Debug\.(WriteLine|Write)\s*\(
```

Structured Logging Patterns

```
- JSON\.stringify\(.+\)
- LoggerFactory\.create
- new Logger\(
- createLogger\(
- winston\.createLogger
- log4j\.Logger
- LogManager\.getLogger
```

Framework-specific Patterns

```
- @Slf4j
- logging\.getLogger\(__name__\)
- LoggerFactory\.getLogger\(.*\.class\)
- spring\.logging
- django\.logging
- express\.logger
```

2. Log Application Messages Gate

Purpose

Validates that application events, state changes, and business operations are properly logged.

Key Features

- Ensures comprehensive logging of application state
- Validates business operation logging
- Checks for proper error and exception logging
- Monitors service layer logging

Validation Patterns

Service Layer Patterns

```
- @LogExecutionTime
- @Logged
- logger\.method(Entry|Exit)
- log(Request|Response)
- logServiceCall
```

Business Logic Patterns

```
- log(State|Status|Change)
- businessLog
- auditLog
- log(Success|Failure)
- logTransaction
```

Error Handling Patterns

```
- try\s*{.*}\s*catch.*{.*logger
- log(Error|Exception)
- logStackTrace
- error\.stack
```

3. Error Handling Gate

Purpose

Ensures proper error handling and reporting across the application.

Key Features

- Validates try-catch implementations
- Checks for proper error propagation
- Ensures error logging and reporting
- Validates custom error handling

Validation Patterns

Exception Handling

```
- try\s*{.*}\s*catch
- throw\s+new\s+\w+Error
- throw\s+new\s+Exception
- handleError
- onError
```

Error Reporting

```
- reportError
- sendError
- notifyError
- errorHandler
- ErrorReporter
```

4. API Reliability Gate

Purpose

Ensures APIs are implemented with proper reliability patterns.

Key Features

- Validates retry mechanisms
- Checks timeout implementations
- Ensures proper circuit breaker patterns
- Validates rate limiting

Validation Patterns

Retry Patterns

```
- retry(
- @Retryable
- withRetry
- maxRetries
- backoff
```

Timeout Patterns

```
- setTimeout
- @Timeout
- timeoutAfter
- withTimeout
- deadline
```

Circuit Breaker

```
- @CircuitBreaker
- circuitBreaker
- failureThreshold
- resetTimeout
```

5. Background Jobs Gate

Purpose

Validates proper implementation of background job processing.

Key Features

- Ensures job scheduling
- Validates job error handling
- Checks job logging
- Monitors job status tracking

Validation Patterns

Job Processing

```
- @Scheduled
- @Async
- cron\.schedule
- backgroundJob
- worker\.process
```

Job Monitoring

```
- jobStatus
- taskMonitor
- jobProgress
- workerStatus
- processStatus
```

6. Security Gate

Purpose

Ensures implementation of security best practices.

Key Features

- Validates authentication
- Checks authorization
- Ensures secure communication
- Monitors data protection

Validation Patterns

Authentication

- `@Authenticated`
- `requireAuth`
- `isAuthenticated`
- `authGuard`
- `validateToken`

Authorization

- `@Authorized`
- `hasPermission`
- `checkAccess`
- `roleGuard`
- `validateRole`

Implementation Guidelines

1. Logging Implementation

- Use appropriate log levels
- Include context information
- Add correlation IDs
- Structure log messages properly

2. Error Handling Implementation

- Catch specific exceptions
- Provide meaningful error messages
- Include error context
- Implement proper recovery

3. Reliability Implementation

- Configure proper timeouts
- Implement retry with backoff
- Set up circuit breakers
- Add rate limiting

Scoring Criteria

Each gate is evaluated based on:

1. **Implementation Completeness (40%)**
   - Pattern coverage
   - Feature implementation
   - Code organization
2. **Quality (30%)**
   - Code structure
   - Pattern usage
   - Best practices
3. **Context (30%)**
   - Appropriate usage
   - Error handling
   - Documentation

Best Practices

Logging

- Use consistent formats
- Include timestamps
- Add context data
- Proper log levels

Error Handling

- Specific exceptions
- Proper recovery
- Error reporting
- Context preservation

Reliability

- Proper timeouts
- Smart retries
- Circuit breaking
- Rate limiting