Error Handling

> **Source File:** generated-documents\technical-design\error-handling.md
> **Generated:** 16/07/2025 at 14:00:35
> **Generated by:** Requirements Gathering Agent - PDF Converter

ErrorHandling

Generated by adpa-enterprise-framework-automation v3.2.0
Category: technical-design
Generated: 2025-07-14T21:15:35.827Z
Description:

ADPA Error Handling Guidelines

1. Error Handling Strategy

Objective:
Establish a robust, consistent, and maintainable approach to error handling across the ADPA framework—including CLI, REST API, document generation, AI provider integration, and third-party enterprise integrations.

Core Principles:

- **Fail Fast, Fail Loud:** Detect and surface errors early; do not silently ignore failures.

- **Consistent Error Model:** Use structured error objects (preferably with TypeScript types/interfaces) across all modules.
- **Separation of Concerns:** Distinguish between recoverable (expected) and unrecoverable (unexpected/systemic) errors.
- **Graceful Degradation:** Provide fallback mechanisms wherever possible, especially for multi-provider AI and integrations.
- **User-Centric Messaging:** Show clear, actionable error messages to end users; avoid leaking sensitive technical details.

Patterns:

- Use `try...catch` for asynchronous operations.
- Throw typed/custom errors (e.g., `AIProviderError`, `ValidationError`).
- Handle errors at the appropriate layer (API middleware, CLI entry point, integration adapters).
- Always clean up resources (e.g., file handles, network connections) in error scenarios.

---

2. Error Categories

a. System Errors

- Out-of-memory, disk full, network unavailable, process crashes.

b. Application Errors

- Invalid user input (validation failures)
- Unsupported operations (e.g., unknown document template)
- Configuration errors (missing API keys, misconfigured environment)

c. Integration/Provider Errors

- AI provider API failures (rate limit, quota exceeded, authentication)
- External service outages (SharePoint, Confluence, Adobe APIs)
- Network timeouts, upstream dependency errors

d. Security & Access Errors

- Authentication failures (invalid/missing tokens or API keys)
- Authorization failures (insufficient permissions, forbidden actions)

e. Data Errors

- Corrupt or missing files
- Data parsing/serialization issues
- Database or configuration inconsistencies

f. Business Logic Errors

- Workflow constraint violations (e.g., attempting to generate a document with incomplete data)
- Compliance or standards validation failures

---

3. Error Logging

Objectives:
Enable root cause analysis, support audit/compliance, and facilitate monitoring/alerting.

Standards:

- Use winston as the centralized logging framework.
- Log at appropriate levels: `error`, `warn`, `info`, `debug`.
- Include structured metadata: timestamp, requestId/traceId, userId (if applicable), module, operation, stack trace (for errors).
- Mask sensitive data (API keys, tokens, PII) before logging.

Sample Log Entry (JSON):

```
{
  "timestamp": "2025-07-08T15:23:44.123Z",
  "level": "error",
  "requestId": "a1b2c3d4",
  "userId": "user-123",
  "module": "ai/openaiProvider",
  "operation": "generateDocument",
```

```
  "message": "OpenAI API rate limit exceeded",
  "errorCode": "AI_RATE_LIMIT",
  "details": { "provider": "openai", "quota": "60/min" },
  "stack": "Error: ... at ..."
}
```

Logging Best Practices

- Log errors at the point of occurrence; propagate context upward.
- Use child loggers for different modules.
- Integrate with Express middleware for automatic API request/response logging.
- CLI errors should be logged to both console (stderr) and log files.
- For integrations, log both requests and responses (with redacted secrets).

---

4. Error Reporting

API (REST):

- Use RFC 7807 Problem Details for error responses:

```
{
  "type": "https://adpa.com/errors/validation",
  "title": "Validation Failed",
  "status": 400,
  "detail": "The 'projectName' field is required.",
  "instance": "/api/v1/generate",
  "errors": [
    { "field": "projectName", "message": "Required" }
  ]
}
```

- Always include HTTP status code, error type, and human-readable detail.

CLI:

- Show concise, user-friendly error output.
- Use color coding (e.g., red for errors via chalk).
- For `--verbose`, display stack trace and log file location.

Web/Admin Interface:

- Display alerts or banners with actionable error messages.
- Avoid technical jargon or stack traces for end users.

---

5. Recovery Procedures

Automated Recovery:

- Use retry logic for transient errors (e.g., network, rate limits).
- For AI providers, automatic failover to backup provider on sustained failures.
- When possible, resume interrupted document generation or upload jobs.
- Clean up partial output (temporary files, incomplete database entries) on failure.

Manual Recovery:

- Provide clear instructions in error messages for next steps (e.g., "Please check your API key and try again").
- Enable users to re-run failed jobs via CLI, UI, or API.
- Document recovery steps for common failures in the troubleshooting guide.

---

6. Retry Mechanisms

- Use exponential backoff for retries (e.g., 1s, 2s, 4s, ... up to max 5 attempts).
- Only retry on transient errors (network timeouts, 429/503 from providers).

- Do **not** retry on permanent errors (validation, authentication, permission denied).
- For multi-provider AI, if a provider fails, mark as unhealthy and attempt with the next available provider.
- Log all retry attempts with reasons and outcomes.

Sample Retry Logic:

```
async function withRetry<T>(fn: () => Promise<T>, retries = 5): Promi
  for (let i = 0; i < retries; i++) {
    try { return await fn(); }
    catch (err) {
      if (isTransient(err) && i < retries - 1) {
        await delay(2 ** i * 1000); // exponential backoff
        continue;
      }
      throw err;
    }
  }
}
```

---

7. Circuit Breakers

Purpose:
Prevent cascading failures and improve system resilience when external dependencies are unstable.

- Use a circuit breaker pattern (e.g., opossum) for all critical external integrations (AI providers, SharePoint, Confluence, Adobe).
- Automatically "open" the circuit (block calls) after N consecutive failures.
- After a defined interval, "half-open" to probe if the dependency has recovered.
- Fall back to alternate providers or degrade gracefully if all circuits are open.

- Expose circuit breaker status in health endpoints and monitoring dashboards.

---

8. User Error Messages

Goals:

- Be clear, actionable, and non-technical for end-users.
- For developers/admins, provide detailed diagnostics (with `--verbose` flag or API debug mode).

Guidelines:

- Indicate what went wrong and how the user can resolve it (e.g., "Authentication failed: Please check your API key.")
- Avoid exposing sensitive technical details in UI or API responses.
- Reference documentation or support channels when appropriate.
- For admin/developer errors, display error code and log file reference.

Examples:

- “Unable to generate document: Required field ‘stakeholders’ is missing.”
- “Connection to SharePoint failed: Please verify your network and SharePoint credentials.”
- “AI provider quota exceeded. Retrying with backup provider...”

---

9. Monitoring and Alerts

Objectives:
Enable proactive detection and remediation of errors.

Practices:

- Integrate with observability tools (e.g., Prometheus, Grafana, ELK/ELK stack, Azure Monitor).
- Expose health endpoints (`/api/v1/health` , `/api/v1/health/ready`) with component status, including circuit breaker states.
- Emit metrics for: error rates, provider failures, retry counts, circuit breaker trips, queue backlogs.
- Configure alerting rules for:
    - High error rates (API 5xx, provider errors)
    - Circuit breakers open for extended periods
    - Authentication/authorization failures
    - Resource exhaustion (CPU, memory, disk)
- Provide log aggregation and searchability (e.g., centralized Winston logs, Azure Log Analytics).

---

10. Troubleshooting Guide

For End Users:

- If you see “Authentication failed”, verify API keys and permissions.
- For “Document generation failed”, check input data and template validity.
- For “Provider unavailable”, try again later or contact your administrator.

For Administrators/Developers:

- Review logs in the default log directory or as specified in your environment.
- Use health endpoints to check system/component status.
- Check circuit breaker dashboards for integration health.
- If retry/backoff is exhausted, investigate external provider status (quota, service health).
- For configuration errors, validate your `.env` and integration-specific config files.

- Consult documentation for provider-specific limits, permissions, and setup steps.
- For persistent issues, escalate with logs and error codes to the support team.

Appendix: Error Object Example (TypeScript)

```
interface ADPAError {
  code: string; // e.g., 'AI_PROVIDER_UNAVAILABLE', 'VALIDATION_ERROR'
  message: string;
  type: 'System' | 'Application' | 'Integration' | 'Security' | 'Data'
  details?: any;
  stack?: string;
  provider?: string;
  requestId?: string;
}
```

Summary Table

Category	Example Scenario	Action	User-facing Message
System	Out of memory	Log, escalate, shut down gracefully	"System error. Please try again later."
Application	Validation failure	Return 400, user guidance	"Invalid input: [field] is required."

Category	Example Scenario	Action	User-facing Message
Integration	SharePoint timeout	Retry, circuit breaker, fallback	"SharePoint unavailable. Retrying..."
Security	Invalid API key	401/403, log, no sensitive info revealed	"Authentication failed."
Data	Corrupt template file	Log, abort, user guidance	"Document template is invalid."
Business Logic	Compliance deviation	Return error, explain context	"Document does not meet standards."

References

- RFC 7807 Problem Details
- Winston Logging
- Opossum Circuit Breaker
- TypeScript Error Handling Patterns

Adopt these guidelines to ensure a resilient, maintainable, and user-friendly ADPA platform.
