Integration Design

> **Source File:** generated-documents\technical-design\integration-design.md
> **Generated:** 16/07/2025 at 14:00:37
> **Generated by:** Requirements Gathering Agent - PDF Converter

IntegrationDesign

Generated by adpa-enterprise-framework-automation v3.2.0
Category: technical-design
Generated: 2025-07-14T21:06:38.979Z
Description:

ADPA (Advanced Document Processing & Automation Framework)

Integration Design Document

---

1. Integration Overview

ADPA is a modular, standards-compliant enterprise automation framework for AI-powered document generation, project management, and business analysis. It is architected with a strong API-first philosophy, providing robust CLI and REST API interfaces, and enterprise-grade integrations with platforms such as Atlassian Confluence, Microsoft SharePoint, Adobe Document Services, and multiple AI providers (OpenAI, Google AI, GitHub Copilot, Ollama). The system is designed for scalable, secure, and maintainable automation in large-scale enterprise environments.

Primary Integration Goals:

- Seamless orchestration of document workflows between internal modules and external enterprise platforms.
- Standards-compliant document generation (BABOK v3, PMBOK 7th, DMBOK 2.0).
- Multi-provider AI orchestration with intelligent failover.

- Enterprise security (OAuth2, SAML, JWT, API Key, RBAC).
- Scalable, observable, and robust microservices-based integration.

---

2. Integration Patterns

ADPA applies several proven enterprise integration patterns (EIP):

Pattern	Application in ADPA
API Gateway	Unified REST API endpoint for all automation and integration flows.
Message Routing	Internal event/message bus for workflow orchestration.
Adapter/Facade	Integration modules for Confluence, SharePoint, Adobe, VCS.
Publish-Subscribe	Notification/events for document status and reporting.
Content Enricher	AI providers add intelligence/context to document data.
Retry & Circuit Breaker	AI provider failover and external service resilience.
Request-Reply/Command Query	REST API and CLI command structure.
Batch Processing	Bulk document operations and publishing.
Canonical Data Model	Unified document and template schemas across interfaces.

---

3. System Interfaces

3.1 REST API (Primary Integration Surface)

- **Technology**: Express.js, TypeSpec-generated OpenAPI 3.0 spec, Swagger UI.
- **Endpoints** (examples):

    - `POST /api/v1/generate` – Document generation.
    - `GET /api/v1/templates` – Template listing.
    - `POST /api/v1/confluence/publish` – Publish to Confluence.
    - `POST /api/v1/sharepoint/upload` – Upload to SharePoint.
    - `GET /api/v1/frameworks` – Supported frameworks.

3.2 CLI Interface

- **Technology**: Yargs-based Node.js CLI.
- **Commands** (examples):
    - `adpa generate ...`
    - `adpa confluence publish ...`
    - `adpa sharepoint upload ...`

3.3 Admin Web Portal

- **Technology**: Next.js 14, React 18.
- **API Consumption**: Uses REST endpoints for all operations.

3.4 Integration Adapters

- **Atlassian Confluence**: OAuth2, REST API, document publishing with metadata.
- **Microsoft SharePoint**: Microsoft Graph API, OAuth2, folder management, metadata, versioning.
- **Adobe Document Services**: PDF, InDesign, Illustrator, Photoshop APIs (OAuth2), premium document output.
- **AI Providers**: OpenAI, Google AI, GitHub Copilot, Ollama, Azure OpenAI – via SDKs and REST APIs.

---

4. Data Flow Design

4.1 Document Generation Workflow

```
graph TD
    A[User/API Client] -->|Request| B[REST API/CLI]
    B -->|Validate & Route| C[Workflow Orchestrator]
    C -->|Template Load| D[Template Service]
    C -->|AI Context| E[AI Orchestration Engine]
    E -->|Invoke| F[AI Provider(s)]
    C -->|Compile Data| G[Document Generator]
    G -->|Output| H[Document Storage/Output]
    H -->|Publish/Export| I[Integration Adapters]
    I --> J[Confluence/SharePoint/Adobe/VCS]
```

4.2 Integration Example: SharePoint Upload

- User triggers upload via API or CLI.
- REST API authenticates and validates the request.
- Integration adapter obtains access token via OAuth2.
- Document uploaded to SharePoint via Microsoft Graph API.
- Metadata and versioning applied.
- Status and errors logged and returned to user.

---

5. Error Handling

Principles:

- All integration points implement retry, circuit breaker, and fallback logic where possible.
- Consistent error model (error codes, messages, trace IDs) in API responses.
- Detailed logging for audit and troubleshooting.

Error Scenarios & Handling:

Scenario	Handling Mechanism
External Service Unavailable	Retry with exponential backoff; circuit breaker; return 503 to user.
OAuth2 Authentication Failure	Immediate error, instruct user to re-authenticate; log details.
AI Provider Timeout or Failure	Failover to next configured provider; log incident; return error if all fail.
Invalid User Input	400 Bad Request with validation messages.
Document Template Not Found/Invalid	404 or 422 error, descriptive message.
SharePoint/Confluence API Errors	Return specific error, log request/response, suggest remediation.

Scenario	Handling Mechanism
Bulk Operations (Partial Failure)	Per-item status; return details of failed/successful items.

---

6. Integration Points

System	Direction	Protocol/SDK	Authentication	Key Operations
Atlassian Confluence	Outbound	REST, Node SDK	OAuth2	Publish, update, metadata
MS SharePoint	Outbound	Microsoft Graph API	OAuth2 (Azure AD)	Upload, folder, metadata, version
Adobe Document Cloud	Outbound	REST, Node SDK	OAuth2	PDF/InDesign/Illustrator output
AI Providers (OpenAI, etc.)	Outbound	REST, SDKs	API Key, OAuth2, JWT	Text/image generation, enrichment
CLI	Inbound	Node.js process/io	OS user, env vars	User commands, batch runs
Admin Portal	Inbound	HTTP (REST API)	JWT/OAuth2 session	UI management, reporting
VCS (GitHub, etc.)	Outbound	Git CLI/SDK	SSH/OAuth	Commit, push, history

---

7. Message Formats

7.1 API Requests/Responses

- **Format**: JSON (OpenAPI 3.0 defined schemas)
- **Content-Type**: `application/json`
- **Standard Fields**:
  - `status` (success/error)
  - `data` (payload)
  - `error` (code, message, traceId)
- **Document Payload Example**:

```
{
  "templateId": "ca8d4758-03c5-4110-84a7-2f5bcd318539",
  "inputs": {
    "projectName": "ADPA Integration",
    "stakeholders": [ ... ],
    "requirements": [ ... ]
  },
  "outputFormat": "pdf"
}
```

7.2 Integration with External APIs

- **Confluence/SharePoint**: JSON payloads, multipart for attachments.
- **Adobe Services**: JSON, binary (for images/PDF), OAuth2 bearer tokens.
- **AI Providers**: JSON-based prompts, context, and completions.

---

8. Integration Security

Best Practices Implemented:

- OAuth2 and SAML for enterprise SSO.
- API Key and JWT for REST API authentication.
- Role-Based Access Control (RBAC).
- HTTPS enforced for all endpoints.
- Rate limiting (express-rate-limit) and CORS controls.
- Helmet.js, input validation (joi/zod), and output encoding for API security.
- Secure storage of secrets using environment variables and .env files.

External Integrations:

- Short-lived tokens for Confluence/SharePoint/Adobe.
- No credentials stored in code; all secrets loaded from secure config or environment.

---

9. Performance Considerations

- **Horizontal Scalability**: Microservices ready, stateless API design.
- **Bulk Operations**: Batch processing APIs for high-volume generation/publishing.
- **Caching**: Redis (optional) for metadata, AI completions, and session state.
- **Provider Failover**: Automatic switching between AI providers to ensure uptime.
- **API Rate Limiting**: Protects against abuse and accidental overload.
- **Asynchronous Processing**: For long-running tasks (PDF rendering, large uploads).

---

10. Monitoring Strategy

- **Health Checks**: `/api/v1/health` and `/api/v1/health/ready` endpoints.
- **Structured Logging**: Express-Winston, log correlation with trace IDs.
- **Metrics**: API request/response stats, error rates, latency, provider usage (integrated with Prometheus/Grafana as needed).
- **Audit Trails**: All integration actions logged for compliance.
- **Alerting**: Thresholds for failures, slow responses, provider outages.
- **Admin Interface**: Real-time monitoring dashboards for usage, status, and integration health.

---

11. Dependencies

- **Node.js 18+, TypeScript 5.7+**
- **Express.js, Yargs, Next.js**
- **@azure/msal-node, @microsoft/microsoft-graph-client**
- **@adobe/pdfservices-node-sdk**
- **openai, @google/generative-ai**
- **Joi, Zod, Helmet, dotenv, winston**
- **Jest (testing), Redis (optional), Swagger UI**

Appendix: Integration Flow Examples

Example 1: Generate and Publish a Project Charter to Confluence

1. User calls `POST /api/v1/generate` with templateId and data.
2. API orchestrates AI provider for content, merges with template.
3. Document generated and stored.
4. User calls `POST /api/v1/confluence/publish` with document reference.
5. Adapter authenticates with Confluence, uploads document, applies metadata.
6. Status returned, logs updated.

Example 2: Multi-Provider AI Failover

- AI orchestration engine attempts to use OpenAI.
- If timeout/error, retries or switches to Google AI.
- If all fail, user receives error with traceId and suggestions.

Summary

ADPA’s integration architecture is robust, extensible, and enterprise-ready. It ensures standards compliance, security, and operational resilience across all integration points, supporting both interactive (CLI/Web) and programmatic (API) workflows. The design enables rapid onboarding of additional platforms and providers, and is engineered for scale, observability, and compliance in demanding enterprise environments.
