System Design

> **Source File:** generated-documents\technical-design\system-design.md
> **Generated:** 16/07/2025 at 14:00:46
> **Generated by:** Requirements Gathering Agent - PDF Converter

SystemDesign

Generated by adpa-enterprise-framework-automation v3.2.0
Category: technical-design
Generated: 2025-07-14T21:00:08.666Z
Description:

System Design Specification

Project: ADPA – Advanced Document Processing & Automation Framework
Version: 3.2.0
Date: 2025-07-08

---

1. System Purpose and Scope

Purpose:
ADPA is a modular, standards-compliant automation framework designed to streamline and automate enterprise documentation, project management, and business analysis processes. It leverages multi-provider

AI integrations to generate, analyze, and manage professional, standards-based business documents (e.g., BABOK v3, PMBOK 7th Edition, DMBOK 2.0). ADPA offers both CLI and RESTful API interfaces, supports enterprise integrations (Confluence, SharePoint, Adobe services), and enforces robust security and compliance for large-scale, Fortune 500-ready deployments.

Scope:

- Automated AI-powered document generation and workflows
- Integration with enterprise tools (Atlassian Confluence, MS SharePoint, Adobe Document/Creative APIs)
- Standards compliance (BABOK, PMBOK, DMBOK, regulatory frameworks)
- Multi-modal interaction: CLI, REST API, and web admin interface
- Advanced context management, AI provider failover, and smart template selection
- Secure, scalable, and extensible for future enterprise automation needs

---

2. System Architecture

2.1 High-Level Architecture Overview

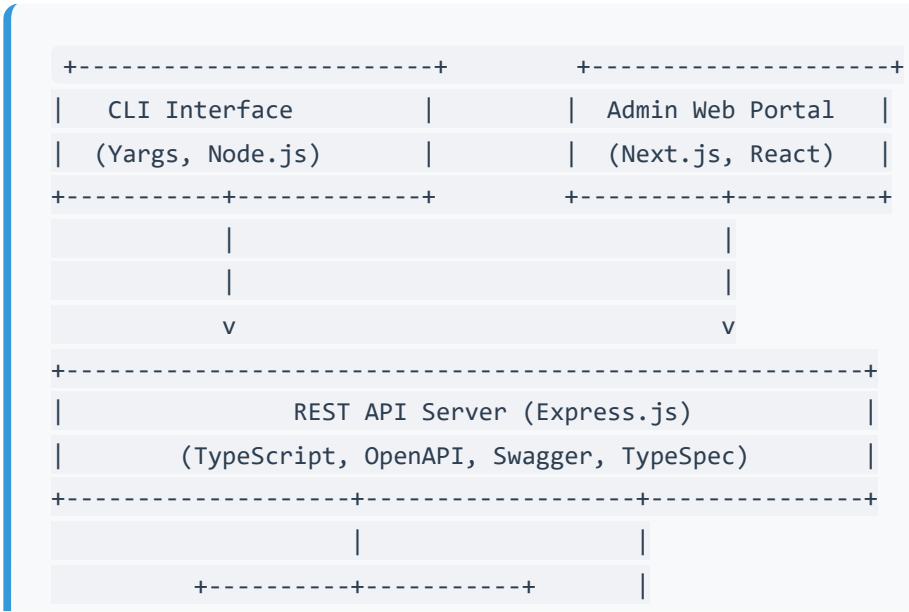

Diagram 1: *High-Level System Architecture* (placeholder)

2.2 Architectural Styles & Patterns

- **API-First Microservices:** OpenAPI/TypeSpec-driven API design
- **Layered Architecture:** Separation between interfaces, orchestration, AI logic, and integrations
- **Plug-in/Provider Pattern:** AI and integration modules are dynamically loaded
- **Security by Design:** Authentication, authorization, and regulatory compliance embedded in all layers

---

3. Module Descriptions

3.1 AI Processing Engine

- **Purpose:** Orchestrates requests to multiple AI providers (OpenAI, Google AI, GitHub Copilot, Ollama)
- **Features:**
  - Multi-provider abstraction and failover
  - Smart context injection and prompt engineering
  - Provider health checks and fallback routing
- **Interfaces:** Internal (to Document Generator), API (for on-demand AI tasks)
- **Key Classes:** `AiProviderManager`, `AiProviderAdapter`, `ContextManager`

3.2 Document Generator

- **Purpose:** Generates professional documents from templates using AI outputs
- **Features:**
  - Template-driven document creation (Markdown, JSON, PDF, etc.)
  - Standards compliance (BABOK, PMBOK, DMBOK)
  - Workflow orchestration (multi-step document pipelines)
- **Interfaces:** API, CLI, Integration Layer
- **Key Classes:** `DocumentGenerator`, `TemplateEngine`, `DocumentJob`

3.3 REST API Server

- **Purpose:** Exposes all system capabilities via RESTful API (Express.js, TypeSpec-generated OpenAPI)
- **Features:**
  - CRUD for documents, templates, jobs, users, integrations
  - Security: API Key/JWT, rate-limiting, CORS, security headers

    - Comprehensive OpenAPI documentation
- **Interfaces:** External (public/enterprise), Auth providers, Analytics
- **Key Files:** `server.ts`, `routes/`, `api-specs/`

3.4 CLI Interface

- **Purpose:** Command-line access to all framework operations
- **Features:**
    - Yargs-powered CLI commands (generate, publish, upload, etc.)
    - Interactive provider selection
    - Batch processing and scripting
- **Key Files:** `cli.ts`, `commands/`

3.5 Integration Layer

- **Purpose:** Adapters for enterprise platforms (Atlassian Confluence, MS SharePoint, Adobe APIs, VCS)
- **Features:**
    - OAuth2/SAML/AD authentication
    - Document publishing, metadata management, versioning
    - Adobe Creative/Document API integration (InDesign, Illustrator, Photoshop, PDF)
- **Key Classes:** `ConfluenceAdapter`, `SharePointAdapter`, `AdobeAdapter`, `VcsAdapter`

3.6 Admin Web Interface

- **Purpose:** Web-based management portal for users, jobs, templates, analytics
- **Features:**
    - Next.js/React 18, Tailwind CSS
    - User/role management, job monitoring, configuration UI
    - Real-time collaboration (future roadmap)
- **Key Files:** `admin-interface/`

3.7 Analytics & Reporting

- **Purpose:** Usage metrics, job status, audit logs, performance dashboards
- **Features:**
  - Real-time and historical analytics
  - Automated compliance and deviation reports
  - Health checks and monitoring endpoints

---

4. Interface Specifications

4.1 REST API Endpoints

- **Authentication:**
  - `POST /api/v1/auth/login` – Obtain JWT
  - `POST /api/v1/auth/refresh` – Refresh token
- **Document Generation:**
  - `POST /api/v1/generate` – Generate a standards-based document
  - `GET /api/v1/templates` – List templates
  - `GET /api/v1/frameworks` – List supported standards
- **Integrations:**
  - `POST /api/v1/confluence/publish`
  - `POST /api/v1/sharepoint/upload`
- **Health & Monitoring:**
  - `GET /api/v1/health`
  - `GET /api/v1/health/ready`
- **Admin:**
  - `GET /api/v1/admin/users`
  - `POST /api/v1/admin/roles`

Diagram 2: *API Endpoint Map* (placeholder)

API Technologies:

- OpenAPI 3.0 (Swagger)
- TypeSpec source for specs (`api-specs/`)

- Express.js middleware: CORS, Helmet, Rate Limiting, JWT, Validation

4.2 CLI Commands

- `adpa generate --key <doc-key> [--format] [--output]`
- `adpa confluence init|publish`
- `adpa sharepoint init|upload`
- `adpa vcs commit|push`
- `adpa provider select` (interactive menu)

4.3 Integration Adapters

Confluence:

- OAuth2 authentication, REST publishing
  SharePoint:
- Azure AD auth, Microsoft Graph API
  Adobe:
- OAuth2, REST API for InDesign, Illustrator, Photoshop

4.4 Data Exchange Formats

- **Input:** JSON, YAML, Markdown, Form-data (uploads)
- **Output:** JSON, Markdown, PDF, DOCX, HTML

---

5. Data Structures

5.1 Document Template

```
interface DocumentTemplate {
  id: string;
  name: string;
  description: string;
  category: string;
  tags: string[];
  templateData: any;
```

```
  variables: Record<string, any>;
  createdAt: Date;
  updatedAt: Date;
}
```

5.2 Document Job

```
interface DocumentJob {
  jobId: string;
  templateId: string;
  status: 'pending' | 'processing' | 'completed' | 'failed';
  requestedBy: string;
  createdAt: Date;
  updatedAt: Date;
  outputLocation?: string;
  errorDetails?: string;
}
```

5.3 AI Provider Configuration

```
interface AiProviderConfig {
  provider: 'openai' | 'google' | 'copilot' | 'ollama' | 'azure-openai
  apiKey: string;
  endpoint?: string;
  model: string;
  fallback?: string[];
}
```

5.4 User & Role (for Collaboration Roadmap)

```
interface User {
  id: string;
  username: string;
  roles: string[];
  permissions: Permission[];
}
```

```
interface Permission {
  resource: string;
  actions: string[];
}
```

---

6. Processing Logic

6.1 Document Generation Workflow

1. **Request Initiation:**
   Via API or CLI, user specifies document type, framework, and options.

2. **Context Analysis:**
   System gathers project context, user input, and previous data.

3. **Template Selection:**
   Appropriate template is chosen (standards, branding, etc.).

4. **AI Engine Invocation:**
   AI Provider Manager routes request to the selected/available AI provider.

5. **AI Output Processing:**
   AI-generated content is injected into the template, variables resolved.

6. **Document Assembly:**
   Output is formatted (Markdown, PDF, etc.), post-processed if needed (Adobe API).

7. **Publishing/Export:**
   Document is stored, optionally published to Confluence/SharePoint/VCS.

8. **Job Tracking:**
   Status is updated, analytics/events logged, audit trails maintained.

Diagram 3: *Document Generation Sequence* (placeholder)

---

7. Error Handling

7.1 General Principles

- All errors are logged with context and stack trace (Winston, Express-Winston).
- API errors use standardized JSON error responses (RFC 7807 Problem Details).
- CLI errors are color-coded, descriptive, and include help tips.

7.2 API Error Responses

```
{
  "error": {
    "code": "INVALID_INPUT",
    "message": "Template ID is required.",
    "details": "Parameter 'templateId' was missing."
  }
}
```

HTTP Status Codes:

- 400: Validation errors
- 401: Unauthorized
- 403: Forbidden
- 404: Not found
- 429: Too many requests
- 500: Internal server error

7.3 AI Provider Failover

- If primary AI provider fails (timeout, quota, error), the system:
    - Logs the error with provider details

    - Switches to next configured provider (if available)
    - Returns user-friendly error if all providers fail

7.4 Integration Failures

- Integration errors (Confluence, SharePoint, Adobe) are caught, logged, and reported with actionable details.
- Partial failures (e.g., PDF generated but upload failed) are tracked and retried when possible.

---

8. Performance Requirements

- **API Response Time:**
    - 95% of document generation requests respond within 5 seconds (excluding external AI/service time)
    - Health checks < 100ms
- **Scalability:**
    - Support for horizontal scaling (stateless API, microservices, Docker/Kubernetes ready)
    - Batch processing: At least 100 concurrent jobs
- **Throughput:**
    - At least 1000 API requests per minute (with Redis caching if enabled)
- **Resource Usage:**
    - Memory and CPU usage monitored; alerts on threshold breaches
- **Availability:**
    - 99.5% uptime for production deployments

---

9. System Constraints

- **Node.js >= 18.0.0, TypeScript >= 5.7.2**
- **Authentication:**

    - All external integrations require secure OAuth2/SAML/AD flows
    - API requires API key or JWT for all write operations
- **Compliance:**
    - Data handling must meet GDPR, SOX, PCI DSS, HIPAA (where applicable)
- **Extensibility:**
    - New AI providers/integrations must implement defined adapter interfaces
- **Persistence:**
    - Default: JSON configuration, but extensible to SQL/NoSQL for jobs, templates, user data
- **No direct file system writes outside `generated-documents/` and `dist/` directories**

---

10. Dependencies

10.1 Runtime Dependencies

- **Node.js, TypeScript**
- **Express.js** (API server)
- **Yargs** (CLI)
- **OpenAI, Google AI, Copilot, Ollama SDKs**
- **@adobe/pdfservices-node-sdk, @azure/msal-node, @azure/openai, @google/generative-ai**
- **@microsoft/microsoft-graph-client** (SharePoint)
- **axios, bcryptjs, cors, dotenv, express-rate-limit, express-validator, form-data, helmet, joi, jsonwebtoken, morgan, multer, node-fetch, swagger-ui-express, uuid, winston, zod**

10.2 Dev Dependencies

- **Jest, ts-jest, @types/*** (testing, typings)
- **@typespec/*** (API specification)
- **Webpack, Prettier, ESLint** (build, formatting, linting)

10.3 External Services & APIs

- **OpenAI, Google AI, Azure OpenAI**
- **Atlassian Confluence REST API**
- **Microsoft SharePoint/Graph API**
- **Adobe Document Services API**
- **GitHub, GitLab, Azure DevOps (VCS)**

[Appendix]

- See ARCHITECTURE.md, API-TESTING-COMPREHENSIVE-SUMMARY.MD, and PHASE-2-IMPLEMENTATION-GUIDE.md for further architectural, API, and integration details.
- Technical diagrams to be provided as part of the final documentation set.

End of System Design Specification