Low Level Design (LLD)

Flogentic Agentflow Platform

Flogentic AI Engineering

December 2025

1. Purpose of This Document

This Low Level Design (LLD) provides a detailed technical description of the internal architecture, components, workflows, data structures, and integration layers of the Flogentic system. It is meant for developers implementing new features, reviewing pull requests, and supporting the EPIC 2 Documentation Sync Agent.

The LLD serves as:

- A reference for all runtime components.
- A machine-updatable document for automatic documentation sync.
- A mapping between source code modules and documentation sections.

2. System Overview

Flogentic is an extensible Agentflow/Flowise-based platform designed to execute AI-driven workflows, integrate DevOps pipelines, and automate documentation generation. The platform consists of:

- **Backend API (apps/api)**: Routes, webhook ingestion, Flowise proxy, document build/publish callbacks.
- **Frontend Web UI (apps/web)**: Flow editor, document viewer, run logs dashboard.
- **Flowise Agentflow Runtime**: Executes custom workflows using LLM, Tool, Custom Function, and Condition nodes.
- **Documentation Engine (packages/api-documentation)**: Template-based doc generation and versioning.
- **CI/CD Workflows**: GitHub Actions for docs-build and publishing.
- **Publishers**: Confluence, GitHub Pages, AWS S3.

3. Repository Structure

The Flogentic repository follows a monorepo structure managed with PNPM workspaces:

```
/apps
    /api
    /web
/packages
    /api-documentation
    /components
    /core
    /nodes
    /utils
.github/workflows
Dockerfile
package.json
```

4. Backend API Design (apps/api)

4.1 Responsibilities

- Provide HTTP routes for Git webhooks.
- Forward webhook events to Flowise Agentflow.
- Store logs and metadata for documentation runs.
- Receive callbacks from CI workflows after docs are built.
- Trigger publishers for documentation output.

4.2 Webhook Flow

- Route: `POST /ci-trigger`
- Verifies GitHub/GitLab signatures.
- Extracts event type: push, pull_request, release.
- Forwards to Flowise with:

```
{
  "input": {
    "webhook_event": "<JSON string>",
    "event_type": "push|pr|tag"
  }
}
```

4.3 Build Callback

- Route: `POST /api/build-callback`
- Receives:
  - build status
  - artifact URL
  - version tag
  - run ID
- Initiates documentation publishing.

5. Flowise Agentflow LLD

5.1 Node Architecture

The agentflow executes sequentially:

1. Start Node (input: webhook payload)
2. Custom Function: parse event
3. Tool Node: fetch PR files (if PR)
4. Custom Analyzer: optional AST-based change detection
5. LLM Node: reason about why code changed
6. Condition Node: route push/PR/release
7. GitHub Tool Node: trigger docs-build workflow
8. Logging Node: record run
9. Wait / Callback for build completion
10. Tool Nodes: publish to Confluence, S3, GitHub Pages
11. Notification Nodes: Slack/Teams/Email

5.2 Input Specifications

- **webhook_event**: JSON string (Git webhook payload)
- **event_type**: push / pull_request / release

5.3 Parsed Event Structure

```
ParsedEvent {
  repo_full_name: string;
  event_type: string;
  branch: string;
  commit: string;
  pr_number: number;
  actor: string;
  changed_files: string[];
  impacted_modules: string[];
  version_tag: string;
  timestamp: string;
}
```

5.4 LLM Output Schema

The LLM generates reasoning for documentation updates:

```
LLMChangeInfo {
  reason: string;
  change_summary: string;
  affected_sections: string[];
  doc_update_suggestion: string;
}
```

6. Documentation Engine (packages/api-documentation)

6.1 Components

- Base LLD: `LLD/base-LLD.md`
- API documentation templates
- HTML/Markdown/PDF exporters
- Version manager

6.2 Module-to-Section Mapping

```
{
  "packages/core/src": "LLD:CoreRuntime",
  "packages/components": "LLD:UIComponents",
  "apps/api": "LLD:BackendAPI",
  "apps/web": "LLD:FrontendUI"
}
```

This enables PR-driven updates to specific LLD sections.

7. CI/CD Documentation Build Workflow

7.1 Triggering

Flowise triggers:

- `workflow_dispatch` via GitHub API
- Inputs passed:

  ```
  version_tag
  changed_files
  affected_sections
  run_id
  ```

7.2 Build Stages

1. Checkout repo
2. Install Sphinx / MkDocs / Typedoc / JSDoc
3. Selective generation for changed modules
4. Upload documentation artifact
5. Notify backend via callback

8. Documentation Publishing

8.1 Publishing Targets

- Confluence
- GitHub Pages
- AWS S3
- Internal portals

8.2 Version Strategy

Artifacts are stored under:

`docs/<version_tag>/`

Each version includes:

- HTML site
- Markdown bundle
- PDF export
- Metadata (commit, author, timestamp)

9. Logging and Auditing

9.1 DB Schema

```
doc_runs {
  id uuid;
  repo text;
  branch text;
  commit text;
  event_type text;
  parsed_event jsonb;
  llm_change jsonb;
  version_tag text;
  status text;
  artifact_url text;
  publish_details jsonb;
  created_at timestamptz;
  updated_at timestamptz;
}
```

9.2 Log Flow

1. Event received → status: received
2. Docs-build triggered → status: started
3. Build finished → status: built
4. Publish done → status: published

10. Notification System

Notifications are sent via:

- Slack
- Microsoft Teams
- Email
- Jira comments

Message format includes:

- Summary of changes
- Reason for change
- Link to documentation
- Version tag

11. Security Considerations

- Webhook signature verification
- Token-based authentication for Confluence and GitHub
- NodeVM sandboxing in Flowise
- Least-privilege IAM policies for S3

12. Testing Strategy

- Unit tests for Custom Function parsing
- Integration tests for Flowise → GitHub Actions → callback
- End-to-end PR merge simulation
- Negative tests for invalid payloads
- Publishing tests in staging environments

13. Mapping to EPIC 2 User Stories

- **US1**: Webhook ingestion, Flowise trigger, version tagging, logging.
- **US2**: Multi-platform publishing (Confluence/S3/GitHub Pages).
- **US3**: Change detection, PR file analysis, LLM reasoning.
- **US4**: Automated documentation generation using MkDocs/Sphinx.
- **US5**: Versioned artifact publishing and navigation sync.
- **US6**: Notifications and communication modules.

14. Conclusion

This LLD documents the functional blocks, runtime design, workflows, CI/CD architecture, and integration points required for the complete Flogentic Documentation Sync Agent. It provides a stable base that can be automatically updated by the agent as new code changes occur.