AutoDoc Agent — Full Project Design

---

1 — Project Summary (one-liner)

AutoDoc Agent is an autonomous multi-agent system that parses a codebase (backend + frontend + DB), builds a knowledge graph of the project, and continuously generates and updates human-readable documentation and diagrams (README, API docs, class/ER/sequence diagrams, changelogs) published into the repo or a documentation site.

---

2 — High-level goals & success criteria

- Automatically produce accurate API docs (endpoints, params, responses).
- Generate up-to-date diagrams (class, sequence, ER, architecture) from code.
- Push docs to /docs folder + optional GitHub Pages site on every commit.
- Achieve ≥90% accuracy on entity extraction for controllers/services/entities in tests.
- Reduce manual doc effort to near-zero for supported patterns (Spring Boot + React).

3 — Core system architecture (components)

Git Repo (code) --> Code Watcher (Agent)

└─> Parser Agent (AST + metadata)

└─> Knowledge Graph Builder

├─> Doc Generator Agent (Markdown/HTML/PDF)

└─> Diagram Generator Agent (Mermaid/PlantUML/Graphviz)

└─> Publisher Agent (commits / site deploy)

Components:

- **Code Watcher Agent** — monitors repo (webhook or cron).
- **Parser Agent** — parses files using AST parsers (JavaParser, Babel), extracts routes, classes, models.
- **Knowledge Graph Builder** — canonical internal model (nodes: modules, classes, endpoints, DB tables; edges: calls, extends, references).
- **Doc Generator Agent** — converts knowledge graph → Markdown/API spec/README/changelog.
- **Diagram Generator Agent** — outputs Mermaid/PlantUML/Graphviz for class/sequence/ER diagrams, plus SVG/PNG.

- **Publisher Agent** — commits to repo /docs, optionally deploys to GitHub Pages or a docs site.
- **Dashboard** (optional) — React app to preview docs/diagrams and approve auto-changes.

4 — Agents & responsibilities (detailed)

Code Watcher Agent

- Trigger: Git push webhook / periodic scan.
- Actions: checkout commit, compute diff, send changed files to Parser Agent, attach commit metadata.

Parser Agent

- Java: JavaParser to extract classes, annotations, method signatures, REST controllers/@RequestMapping, DTOs, Entities.
- JS/TS (React): Babel parser to extract components, props, hooks, route definitions (react-router), API calls.
- SQL/ORM models: parse entity annotations (JPA/Hibernate) or Sequelize/Mongoose schemas.
- Outputs: JSON AST summaries + file-level metadata.

Knowledge Graph Builder

- Ingests parser output, normalizes into nodes & edges:
  - Node types: Module, Package, Class, Interface, Method, Endpoint, Entity/Table, Field, External API.
  - Edge types: calls, returns, persists, extends, imports, maps-to.
- Stores KG in a lightweight graph DB (Neo4j) or as JSON in PostgreSQL.

Doc Generator Agent

- Maps KG → documentation:
  - Project Overview (auto-detect stack & modules).
  - API Reference: endpoints, HTTP method, path, params, request/response shapes (derived from DTOs).
  - Module docs: class responsibilities, service interfaces.
  - Changelog: derive from commit messages + diff summaries (auto-summarized by LLM).
  - Code explanations: per-method short summary (LLM-assisted).

- Outputs: Markdown files, OpenAPI spec (optional), PDF with pandoc.

Diagram Generator Agent

- Generates:
    - Class diagrams (PlantUML or Mermaid class diagrams).
    - Sequence diagrams for common flows (login, create order) by inferring call chains from KG + simple templating.
    - ER diagrams for DB schema.
    - Architecture diagram: services, external APIs, DBs, message buses.
- Exports as Mermaid code + PNG/SVG via CLI renderers.

Publisher Agent

- Runs validation: link checks, image exists, docs build OK.
- Commits to repo /docs branch or PRs changes for review automatically (configurable).
- Optionally deploys GitHub Pages or static docs site (Docusaurus/Sphinx).

5 — Data model (simplified)

Knowledge Graph Node JSON (example)

```
{
"id":"module:orders",
"type":"module",
"name":"orders",
"children":[...]
}
```

Relational DB tables (if using Postgres)

- projects (id, repo_url, default_branch)
- commits (id, project_id, hash, author, message, timestamp)
- nodes (id, project_id, node_type, name, metadata jsonb)
- edges (from_node, to_node, edge_type, metadata jsonb)
- docs (project_id, path, content, generated_at, commit_hash)

---

6 — API Design (service endpoints for dashboard & agent control)
(Expose as REST for dashboard access)

- POST /webhook — GitHub webhook receiver (triggers generation).

- GET /projects — list monitored projects.
- POST /projects — add project (repo url + auth).
- GET /projects/{id}/docs — list generated docs.
- GET /projects/{id}/docs/{path} — fetch doc content.
- POST /projects/{id}/generate — manual trigger (params: commit_hash, force).
- GET /projects/{id}/status — generation status.
  Authentication: JWT for dashboard + repo OAuth app for Git operations.

---

7 — Tech stack

- **Orchestration / Agents**: Python (FastAPI) for agents, or Node if preferred. Python recommended for LLM tooling.
- **AST Parsing**
  - Java: JavaParser (com.github.javaparser)
  - JS/TS: @babel/parser (Node process) or ts-morph
  - SQL/ORM: custom parsers or use SQLFluff tooling
- **Knowledge Graph**
  - Neo4j (recommended) OR Postgres JSONB + graph queries
- **LLM / NLU**
  - OpenAI GPT-4/5 or Llama 3 via API (for summarization & method-level explanations)
  - LangChain or direct LLM SDK to orchestrate prompts
- **Diagrams**
  - Mermaid.js (generate mermaid code, render with mermaid-cli)
  - PlantUML (plantuml.jar) for UML export
  - Graphviz for dependency graphs
- **Storage**
  - Git operations via GitPython or JGit
  - DB: PostgreSQL (metadata), Neo4j (KG)
- **Frontend**
  - React + Vite for dashboard (preview docs + approve)
- **CI/CD**
  - GitHub Actions to run a generation-on-push workflow (optional)
- **Containerization**
  - Docker + Kubernetes (optional) for deployment

GitHub Actions (Agent runs inside GitHub, no external server needed)

How it works

You add a workflow file:

⚙️ **.github/workflows/autodoc.yml**

```
name: AutoDoc Agent


on:
```

```
push:
branches:
- main

jobs:
generate-docs:
runs-on: ubuntu-latest

steps:
- name: Checkout code
uses: actions/checkout@v3

- name: Set up Python
uses: actions/setup-python@v4
with:
python-version: '3.10'

- name: Install AutoDoc dependencies
run: pip install -r autodoc_requirements.txt

- name: Run AutoDoc Agent
run: python run_autodoc.py

- name: Commit updated docs
run: |
git config --local user.name "AutoDoc Bot"
git config --local user.email "bot@example.com"
git add docs/
git commit -m "AutoDoc: updated docs" || echo "No changes to commit"
git push
```

Now GitHub Actions handles everything:

- On each push → it runs AutoDoc Agent inside GitHub → updates /docs.
- **COMPLETE WORKFLOW (Webhook or Actions)**
- Here is the exact logical flow:

Git Push

↓

Trigger (Webhook or GitHub Action)

↓

AutoDoc Agent starts

↓

Pull latest repository code

↓

Parser Agent scans:

- Java Classes

- REST Controllers

- Services

- Interfaces

- DTOs

- React Components

- SQL/ORM models

↓

Knowledge Graph updated

↓

Doc Generator creates/updates:

- README.md

- api_endpoints.md

- architecture.md

- class_diagrams.md

- sequence_flows.md

- ERD.md

↓

Diagram Generator produces:

- Mermaid diagrams

- PlantUML diagrams

- PNG/SVG images

↓

Publisher Agent commits docs back to:

- /docs folder

- docs/auto/ branch

OR

- automatically creates a Pull Request

Example: What you will see after pushing code

You push:

```
git add .
git commit -m "Added new OrderController"
git push origin main
```

After 10–20 seconds, you'll see:

📁 docs/api_endpoints.md

📁 docs/class_diagram.svg

📁 docs/architecture.md

📁 docs/erd.png

A bot commit appears:

AutoDoc Bot — updated docs for commit abcd123