Problem statement :

Manual updating of API documentation for REST, SOAP, and MQ services is time-consuming and often neglected, leading to outdated or inconsistent website content. This causes confusion for API consumers and slows down integration. A solution is needed to automate this process and ensure real-time, accurate documentation delivery.

Objective:

1. To automate API documentation updates using GenAI when changes are committed to Git.
2. To use Retrieval-Augmented Generation (RAG) for context-aware and high-quality documentation.
3. To directly publish the updated documentation to the live website without manual effort.

Need for the System:

1. Reduces human effort and eliminates documentation mismatches.
2. Ensures always-updated website content using real-time Git triggers.
3. Leverages GenAI to generate consistent, readable, and accurate API documentation.

Scope of the Project:

1. Supports REST (OpenAPI), SOAP (WSDL), and MQ (XML/JSON) APIs.
2. Uses GenAI + vector database to retrieve and reuse previous doc styles.
3. Integrates with static/dynamic websites for direct documentation deployment.

Input to the System:

1. API Spec Changes via Git Commits:
   * Changes in API files (e.g. Swagger/OpenAPI, WSDL, MQ schema) pushed to a Git repository.
2. Existing Live Website Content:
   * The current API documentation already hosted on the website (this acts as the “context” for GenAI to understand style, structure, terminology, etc.).
3. Optional Developer Annotations (Inline Comments):
   * Comments added to the API spec files by developers for extra context (optional but helpful for GenAI).

Final Output from the System:

1. Auto-generated Updated API Documentation (via GenAI):
   * Generated using RAG (Retrieval-Augmented Generation), where the retrieval source is the current live website content (extracted and vectorized).
   * Output is in the same style and structure as the existing site content.
2. Direct Website Update (No Manual Edits):
   * The system pushes the new documentation content directly to the website database or CMS through API or content injection.
   * If the site is static, it triggers a rebuild and redeploy via CI/CD (e.g., GitHub Actions, Netlify, or custom script).
   * If the site is dynamic (e.g., built with React, Flask, etc.), the updated content is inserted into the corresponding section or component automatically.

work flow steps

Overview:

Goal: Automatically update live website API documentation when API spec changes are committed to Git using GenAI (RAG-based system).

STEP-BY-STEP IMPLEMENTATION:

Step 1: Git Commit Detection (Change Trigger)

• Description: Monitor a Git repository for changes in API definition files.

• Tools: Git hooks, GitHub Actions, GitLab CI, Jenkins, etc.

• Input:

• Developer commits updated API spec files: swagger.yaml, wsdl.xml, mq\_schema.json, etc.

• Output:

• Trigger signal or event with file path(s) and diff info for changed API files.

• Intermediate Result: List of modified files (with metadata).

Step 2: API Spec Extraction and Parsing

• Description: Extract relevant parts of the API specification from changed files.

• Tools: Swagger Parser (for OpenAPI), WSDL parser, XML/JSON parsers.

• Input:

• API spec files from the Git commit.

• Output:

• Structured data of endpoints, methods, parameters, request/response schema, etc.

• Intermediate Result: Parsed API information in a structured format (e.g., JSON object or Python dict).

Step 3: Retrieve Context from Existing Website Documentation

• Description: Pull the current API documentation from the live website to preserve writing style, formatting, and related descriptions.

• Tools: Web scraper (if static site), CMS API (if dynamic site), or direct DB query.

• Input:

• URL or API endpoint of the documentation section.

• Output:

• Raw text or HTML content of the current documentation.

• Intermediate Result: Raw doc content for each API section (used as context for GenAI).

Step 4: Vector Embedding and Similarity Search (RAG Retrieval Phase)

• Description: Convert existing documentation into vector embeddings and store in a vector database. Perform similarity search for relevant context.

• Tools: Sentence Transformers, OpenAI Embeddings, FAISS, Pinecone, Weaviate.

• Input:

• Website documentation content (text).

• Query: name or description of updated API endpoint.

• Output:

• Top-k most relevant chunks (contextual matches) for the modified API.

• Intermediate Result: Context documents retrieved for input to GenAI.

Step 5: GenAI-Based Documentation Generation (RAG Generation Phase)

• Description: Feed parsed API specs and retrieved context to a GenAI model to generate consistent, accurate, and complete documentation.

• Tools: OpenAI GPT-4, Claude, Gemini, or fine-tuned LLMs.

• Input:

• Parsed API details.

• Retrieved contextual documentation.

• Output:

• Generated documentation including endpoint summary, request/response, authentication info, etc.

• Intermediate Result: Human-readable documentation in natural language (HTML, Markdown, or text).

Step 6: Documentation Injection into the Website

• Description: Automatically push the newly generated documentation to the appropriate section of the website.

• Tools: REST APIs for CMS (Strapi, WordPress, etc.), headless CMS clients, Git push for static websites, or direct DB update.

• Input:

• Generated documentation text.

• Output:

• Updated section on the live website.

• Intermediate Result: Website content replaced or updated in real time or during next deployment.

Step 7: (Optional) Validation and Preview

• Description: Before final deployment, run validation checks or offer a preview window to review changes.

• Tools: Custom UI, diff viewer, approval system.

• Input:

• Newly generated and existing documentation.

• Output:

• Approval signal or diff feedback.

• Intermediate Result: Confirmed, validated doc update.

Step 8: (Optional) CI/CD Deployment

• Description: If the site is static, trigger a redeployment so changes reflect live.

• Tools: Netlify, Vercel, GitHub Pages, Jenkins, etc.

• Input:

• Updated documentation files (HTML/Markdown).

• Output:

• Redeployed and updated website with latest docs.

• Intermediate Result: Live site now shows updated docs.

Example Input/Output Summary:

Step Input Output

1 Git commit with updated spec Trigger with file path

2 API spec files Parsed JSON/dict of endpoints

3 Live website URL/DB Raw documentation text

4 Docs + API name/query Contextual docs (top-k relevant chunks)

5 API data + context Generated new documentation (natural text)

6 Generated docs Updated section on website

7 Old vs New docs (optional) Approval/confirmation

8 Final docs (if static site) Live website redeployed

methodology

The system uses a Retrieval-Augmented Generation (RAG) approach powered by GenAI to automate API documentation updates.

• When an API spec (REST, SOAP, MQ) is modified and committed to Git, a CI/CD pipeline detects the change.

• The updated API definition is parsed and sent as input to the GenAI engine.

• Existing live website documentation is semantically indexed into a vector database to serve as contextual reference.

• The system performs a similarity search on this database to retrieve relevant context related to the changed API.

• The GenAI model uses both the retrieved context and the new API spec to generate updated documentation in a consistent style.

• The generated documentation is then automatically pushed to the corresponding section of the live website using direct API calls or CMS integration.

• This removes the need for manual edits and ensures the website always reflects the latest API changes accurately.

why this methodology

| **Method** | **Context Awareness** | **Accuracy** | **Automation** | **Dev Effort** | **Best For** |
| --- | --- | --- | --- | --- | --- |
| Git Diff to Doc | Medium | Medium | High | Low | Simple use cases |
| Full Schema to Doc (Structured) | High | High | High | Medium | Complex APIs (REST/SOAP) |
| Commit Message + GenAI | Low-Medium | Low-Med | Medium | Low | Fast dev teams with CI |
| PR/IDE LLM Plugin | High | High | Med-High | Medium | Human-in-the-loop workflows |
| GenAI + Vector DB (RAG) | Very High | Very High | High | High | Scalable teams, rich history |
| DSL to OpenAPI+Doc | Medium | High | High | Medium-High | Greenfield projects, new APIs |

**GenAI + Structured Schema Parsing + RAG-based Context Retrieval (Vector DB)**

**Justification**:

* It avoids hallucination by grounding LLM output in actual prior documentation.
* Handles partial changes or additions with better context than raw diff.
* Scalable with growing teams or versioned APIs.
* Prevents duplication or inconsistency in doc tone/format.

**Methodology**:

* Embed all existing API docs in a vector DB (e.g., Pinecone, FAISS)
* On new commits, identify changed endpoints
* GenAI fetches *most similar existing documentation* as context
* Only generate new documentation for changed endpoints using RAG (Retrieval-Augmented Generation)

**Efficiency**: ★★★★★

**Why Efficient?**

* Ensures consistency across old and new docs
* Supports scalable multi-team environments
* Reduces hallucination by grounding with historical docs

**Prompt with Context Retrieval**:

text

CopyEdit

Using this previous endpoint doc as reference:

<similar\_doc>

Generate updated docs for this endpoint:

<new\_schema\_or\_diff>

techstack.

**1. Source Control & CI/CD**

* **Git** – Version control system for tracking API spec changes.
* **GitHub Actions / GitLab CI / Jenkins** – Automates detection of spec changes and triggers the pipeline.

**📄 2. API Specification Parsing**

* **OpenAPI/Swagger Parser** – For REST API specs (swagger.yaml, openapi.json).
* **WSDL Parser / lxml** – For SOAP-based APIs (wsdl.xml).
* **Custom JSON Schema Parser / jsonschema** – For MQ schema files (mq\_schema.json).

**🌐 3. Website Content Extraction**

* **BeautifulSoup / Scrapy** – For scraping static website documentation.
* **CMS APIs (Strapi, WordPress, Contentful)** – For dynamic site content retrieval.

**📦 4. Vector Embeddings & Similarity Search**

* **SentenceTransformers (BERT variants) / OpenAI Embeddings** – To generate semantic vectors from documentation.
* **FAISS / Pinecone / Weaviate / Chroma** – Vector database to perform top-k similarity search for relevant context.

**🤖 5. GenAI (Text Generation)**

* **OpenAI GPT-4 / Claude / Gemini** – Generates natural language documentation from context + API specs.
* **LangChain / LlamaIndex** – Optional orchestration layer for managing RAG flow.

**🌍 6. Documentation Update (Push to Website)**

* **CMS REST APIs / GraphQL APIs** – For pushing updates to systems like Strapi, WordPress, Contentful.
* **Static Site Generator Integration** – (e.g., push Markdown to Docusaurus, Hugo, MkDocs repo).

**🧪 7. Validation & Preview (Optional)**

* **Diff Viewer Libraries (diff2html, jsdiff)** – To visually compare new vs. old documentation.
* **Custom Review UI** – Built using **React.js / Next.js** to allow approvals before final push.

Innovation ideas with respect to this.

✅ **Conversational Interface**

* Add a chatbot inside the docs where users can ask:

“What changed in /users API last week?”

* + Backed by embeddings + vector store (e.g., Pinecone, FAISS)

 ✅ **Smart Review Suggestions**

* GenAI suggests if changes might break backward compatibility
* Suggests warnings or migration notes

 ✅ **Voice or Video Auto-Explainers**

* Generate a quick Loom-style explanation video or audio using TTS + slides from changes