GSoC 2025 PROPOSAL Shah Ayan Swagger / OpenAPI standardization for Jenkins REST API

Google Summer of Code Program 2025 Project Proposal

Name: - **SHAH AYAN ULLAH**
Email: - **SHAHAYAN916@GMAIL.COM**
Github handle: - Link

Project Abstract: -

Jenkins’ REST API documentation is currently fragmented and manually maintained, leading to inconsistencies and difficulties for developers. This project aims to implement OpenAPI 3.0 to extract API specifications from Jenkins Core and plugins dynamically, generate structured documentation, and enable seamless versioning. By integrating Swagger UI/Redoc, we will provide interactive API references, allowing developers to explore APIs efficiently. Automating documentation ensures up-to-date and machine-readable specifications, enhancing usability and third-party integrations

Project Description: -

Jenkins' REST API documentation is currently fragmented, manually written, and inconsistent across core and plugin APIs. Developers face challenges in determining request/response formats, leading to unreliable integrations and inefficient plugin development. Additionally, the lack of machine-readable API specifications prevents leveraging tools like Swagger UI and OpenAPI Codegen to auto-generate client libraries.

This project aims to implement OpenAPI specifications to **automate and standardize Jenkins REST API documentation**. It will provide **real-time, structured, and developer-friendly API documentation**, ensuring extensibility for both **Jenkins Core and plugins**.

Technical Implementation

1. Automated API Metadata Extraction

Goal: Dynamically extract API details from Jenkins Core and plugins.

Sources of MetaData:

- Java annotations: `@Exported`, `@ExportedBean`.
- Stapler routes (Jenkins internal request dispatching mechanism)
- Javadocs and REST endpoints
- Extract REST API details dynamically from **Jenkins Core and plugins**.
- Ensure comprehensive coverage for **request methods, response types, and error handling**.

What to Extract:

- Request methods (`GET`, `POST`, etc.)
- Endpoint paths
- Parameters (query, path, body)
- Response structures and data types
- Authentication mechanisms
- Error codes and messages

Extract Metadata using various methods :-

Extract Metadata from the Jenkins Update Center API

- Jenkins provides an **Update Center API**, which contains metadata for all available plugins.
    - https://updates.jenkins.io/current/update-center.json
    - https://updates.jenkins.io/stable/update-center.json

Extract Metadata from Plugin `.hpi` / `.jpi` Files

- Jenkins plugins are packaged as `.hpi` or `.jpi` files. These are ZIP archives containing metadata

Extract Metadata Using Java or Python Script.

- automated way to extract metadata, I can parse the `update-center.json` or `MANIFEST.MF` files.

2. OpenAPI Specification Generation

Goal: Convert extracted metadata into an OpenAPI 3.0-compliant spec (`.json` or `.yaml`).

Steps to Follow Further :

- **Analyze Metadata**: Identify key information such as endpoints, request methods, parameters, request/response bodies, and authentication details.
- **Define OpenAPI Structure**:
    - Use `openapi` field to specify OpenAPI version.
    - Provide `info` (title, description, version).
    - Create `paths` for each API endpoint with methods (`get`, `post`, etc.).
    - Specify `components` (schemas for request/response bodies).
    - Define `servers` (base URLs).
- **Map Metadata to OpenAPI:**
    - Convert extracted entities into `components.schemas` (data models).
    - Assign API operations to `paths`.
    - Define `responses` based on expected outcomes.

- Implement **versioning support** (e.g., `/rest/api/v1`, `/rest/api/v2`) to manage API changes.
- Host OpenAPI specs via a dedicated endpoint (e.g., `/rest/api/docs`).

3. Plugin API Support and Registration

Goal: Enable plugin developers to contribute and register API documentation.

- Enable **automatic API spec generation for plugins**.
    - **Jenkins Plugin or Script**: Automate extraction via a Jenkins plugin or Python script.

    - **Run Periodically**: Schedule automatic runs to regenerate API specs when plugins are updated.

    - **Expose OpenAPI Docs**: Serve the generated spec using Swagger-UI or a REST API.

- Provide a mechanism for **plugin developers to register APIs** in the central documentation system.
    - **Define a Standard API Metadata Format**
        - Plugin developers should define their APIs in OpenAPI 3.0 format.
        - Provide a standardized metadata file (api-spec.yaml or api-spec.json) inside the plugin repository.
    - **Provide an API Registration Endpoint in Jenkins**
        - Jenkins should expose an API registration endpoint where plugin developers can submit their API specs.
        - New plugins **must** provide an `api-spec.yaml` to be published in the Jenkins plugin repository.

        - Enforce this in the Jenkins **plugin submission process**.
        - Example endpoint : POST /plugin-api/register
        - Payload should include: "apiSpecUrl"
    - **Automate API Spec Collection & Validation**
        - Jenkins (or a dedicated microservice) should periodically scan registered plugins for updated API specs
        - Validate API specs using OpenAPI validators to ensure correctness.

        - Store them in a **central repository** (e.g., Jenkins API Hub).
    - **Expose a Central API Documentation Portal**
        - Use Swagger-UI or Redoc to display all registered APIs in a searchable format.

        - Provide a REST API to query APIs dynamically.
    - **Integrate API Checks in CI/CD**
        - Use a GitHub Action or Jenkins Pipeline to validate OpenAPI specs in plugin repositories.
        - Run a **scheduled Jenkins job** that checks for API updates from registered plugins.

        - Automatically regenerate the **central API documentation**.

Migration strategy for Existing Plugins

- **Auto-Extract APIs from Existing Plugins**
  - Convert extracted data into OpenAPI specs (**Explain in Point 1**)
- **Generate API Spec for Legacy Plugins**
  - If a plugin does not provide api-spec.yaml, generate an OpenAPI spec automatically based on discovered endpoints
- **Notify Plugin Developers**
  - Notify developers to validate and officially register their API specs.

  - Provide a Jenkins Plugin API SDK to simplify API registration

4. Integration with OpenAPI Tooling

Goal: Make documentation easily accessible and developer-friendly.

- **Swagger UI / Redoc Integration** to display interactive API documentation within Jenkins.
- Enable developers to explore APIs directly within the Jenkins UI.
- Use OpenAPI Codegen to generate SDKs for:
  - Java
  - Python
  - Javascript
  - Go
  - ….and more

Benefits to Jenkins

- **Improved Developer Experience**: A structured, interactive API reference reduces onboarding time and errors.
- **Enhanced Integration Capabilities**: OpenAPI specs enable tools to auto-generate SDKs for third-party integrations.
- **Consistency & Maintainability**: Automating API documentation ensures up-to-date and reliable references.
- **Better Plugin Development Workflow**: Plugin developers can seamlessly integrate API documentation alongside their plugins.

Why This Project?

Jenkins thrives on **automation and extensibility**, yet its REST API documentation lacks these qualities. By implementing **OpenAPI-based, real-time API documentation**, we ensure **developer-friendly, standardized, and easily maintainable API references**, benefiting both **Jenkins Core and its plugin ecosystem**. This aligns with Jenkins' long-term goal of becoming more accessible, extensible, and scalable for contributors and integrators alike.

Project Deliverables

Note: The dates in the GSoC Timeline(s) takes precedence over this document

- GSoC Timeline: https://developers.google.com/open-source/gsoc/timeline
- May 8, 2025: Familiarize myself with the jenkins codebase and the areas requiring openAPI documentation. Set up my development environment using intelliJ IDEA, Docker and other required tools. Participate in jenkins mailing lists, forums and online meetings to engage with mentors and the community. Establish a GitHub repository (either an organization or a new account) and share access with mentors and jenkins administrators.
- June 2 (Coding Begins) - Develop an initial Proof of Concept (POC) plugin to demonstrate the feasibility of OpenAPI documentation extraction. Share the POC with mentors for feedback and guidance on improvement areas.

- July 14 - July 18 (Midterm Evaluations) - Expected progress: 30-40% completion of the project. Implement a basic UI (Swagger UI or Redoc) for visualizing the generated API docs. Regular online meetings and email updates with mentors to validate progress.

- Sept 1 - Sept 8 (Final Evaluations) - Expected progress: 90-100% project completion. Ensure the API documentation can be served from a dedicated endpoint (e.g., `/rest/api/docs`).Implement versioning (`/rest/api/v1`, `/rest/api/v2`) for handling API changes. Conduct extensive testing and validation to ensure accuracy and stability.

- Sept 1 - Nov 9 (GSoC Contributors with Extended Timelines Continue Coding) - Focus on performance optimization, bug fixes, and improving documentation accuracy. And conduct further testing to ensure compatibility across jenkins environments.

- Nov 10 (Final Date for all GSoC Contributors to Submit Their Final Work Product) - Finalize and submit the complete OpenAPI Integration along with detailed documentation. Ensure all deliverables are traceable via Git commits and well-documented for future contributors. And provide a step by step usage guide for developers to iterate OpenAPI specs into their jenkins plugins.

- Post GSOC - Continue maintaining and improving jenkins API documentation beyond GSoc. Gain valuable experience in OpenAPI, REST API documentation, and CI/CD automation while making my first significant open-source contribution to Jenkins.

Proposed Schedule

March 24 - April 8, 2025 (Application Period)

Explore interesting GSoC project ideas and find one that aligns with my skills and interests

April 9 - May 7 (Acceptance Waiting Period)

Continue exploring the Jenkins core codebase and REST API structure.
Learn about Jenkins plugin development and how APIs are exposed through plugins.
Identify potential challenges and dependencies in implementing OpenAPI support.
Set up my local Jenkins development environment with necessary tools.

May 8 - June 1 (Community Bonding Period)
Get to know the **mentors and contributors** in the community. Familiarize myself with the codebase and overall project structure. Get a detailed walkthrough of Jenkins API documentation mechanisms.

June 2 - July 13 (Standard Coding Period)
Start implementing a small Proof of Concept (POC) to test API extraction. Generate the first version of an OpenAPI-compliant JSON/YAML file. Ensure extracted API details include request/response formats and error handling. Set up an endpoint like **`/rest/api/docs`** to serve the generated documentation.Commit code regularly and submit for mentor review and feedback. By the end of this phase, a **basic working version** of API documentation should be ready. With basic documentation UI (swagger or Redoc)

July 14 - August 25 (Work Period for Standard Route)
Improve API documentation presentation and searchability in the UI. Implement API versioning support (e.g., `/rest/api/v1`, `/rest/api/v2`).Extend the solution to support Jenkins plugins, making it easy for plugin developers. Add validation tests to ensure documentation accuracy. Continue submitting code daily and reviewing feedback from mentors. **Complete OpenAPI-based documentation system** integrated with Jenkins.

September 1 - November 9 (Extended Coding Period)
Conduct **extensive testing** to fix any remaining bugs. Improve **performance and security** of API documentation generation. Work on **detailed documentation** for future contributors.

Future Improvements

If time allows or after GSoC, I plan to **Automated Documentation Updates**: Implement a mechanism to auto-update API documentation when Jenkins core or plugins change.
Enhanced API Coverage: Expand OpenAPI documentation to cover more Jenkins plugins and core APIs.

Continued Involvement

I plan to stay actively involved in the **Jenkins open-source community** beyond GSoC.
Contributing bug fixes and enhancements to Jenkins core and plugins.
Helping new contributors get started by sharing knowledge and improving documentation.

Conflict of Interests or Commitment(s)

I do not have any major conflicts that would prevent me from fully committing to GSoC. I quit my full-time job five months ago as I am preparing to start my master's degree in September. Previously, I worked while pursuing my graduation, balancing both studies and a job effectively. During GSoC, I will be able to dedicate sufficient time to the project without any external commitments affecting my contributions.
If any unforeseen challenges arise, I will proactively communicate with my mentors to adjust my schedule and ensure steady progress throughout the program.

Major Challenges Foreseen

- Lack of communication would be a major challenge.
- Understanding the Existing Codebase
- Ensuring Compatibility with Jenkins Plugins

References

- Links, books, etc. that are relevant to your project.

Relevant Background Experience

- I have strong experience in **architecting and designing REST APIs, with hands-on expertise in building Web APIs & APIs Documentation** using .NET Core and Angular. My background includes designing scalable backend systems and API documentation using Swagger & Redocly, ensuring efficient data handling, and integrating APIs with frontend applications.

- Additionally, **I have a solid understanding of programming and have taught hundreds of students various programming languages, including Java and Jenkins**. , which will help me adapt quickly to Jenkins' codebase and contribute effectively to this project. My experience in full-stack development and API design aligns well with the goals of this project

Personal

I completed my graduation last year and worked as a full-time Software Developer at WOM Solution Pvt Ltd, specializing in .NET Core and Angular. I got the job while still pursuing my degree, balancing both work and studies. Before that, at the beginning of my second year, I secured an 8-month internship, which gave me hands-on experience in real-world projects and helped strengthen my technical skills.

Five months ago, I left my job to focus on preparing for my master's degree in Computer Science, which hopefully I'll be starting this September.

I first learned about Jenkins while working on CI/CD pipelines for API development and deployment. The automation capabilities and flexibility of Jenkins fascinated me

What makes me special is exploring my field and my ability to adapt and learn quickly. I've worked on real-world projects, managed responsibilities while studying, and developed a deep interest in backend architecture and API design. I also have a strong passion for contributing to open-source projects, and I strongly believe GSoC is a great opportunity to make an impact on my life to open more doors for me.

Availability and commitments

I left my full-time job five months ago to prepare for my master's degree, which begins in September 2025. During GSoC, I will be fully available and committed to completing the project on time. I can dedicate my time effectively, ensuring steady progress with regular contributions and active communication with my mentors.

Experience

Language Skill Set

- C++, Java, c#, Dot Net Core, Angular, Javascript, SQL, JSON, XML, HTML, CSS, Bootstrap

Related Research and Work Experience (if any):

- Software Engineer Internship at Etechni for 8 Months
- Associate Software Engineer at WOM Solution (10 Months)

Reference Links and Web URLs (optional):

- LinkedIn