**1. Project Community Guidelines**

To ensure a positive, inclusive, and productive environment, all community members are expected to follow these guidelines:

**1.1 Be Respectful**

* Treat all members of the community with respect, regardless of their skill level or background.
* Foster a welcoming environment and be mindful of how you communicate. Respect differing opinions, and avoid personal attacks or harassment.

**1.2 Contribute Constructively**

* When contributing, aim to add value to the project. Whether it’s a bug report, code contribution, or feedback, ensure that your input helps move the project forward.
* Provide detailed information when submitting issues or proposing changes, and offer constructive feedback when reviewing others’ work.

**1.3 Follow the Code of Conduct**

* All members must adhere to the project’s **Code of Conduct**, which promotes diversity, inclusion, and professionalism.
* Any violations of the Code of Conduct should be reported to the project maintainers.

**1.4 Stay Engaged**

* Participate in discussions, attend community meetings, and contribute to forums like GitHub Discussions or Slack channels.
* Don’t hesitate to ask questions or offer help. Engaged community members help the project grow and evolve.

**1.5 Share Knowledge**

* If you learn something new while working on the project, share it with the community. Whether it’s in the form of documentation, tutorials, or answering questions, your knowledge can benefit others.

**1.6 Acknowledge Contributions**

* Always give credit where it’s due. Acknowledge the contributions of others, and thank them for their efforts.

**2. Development Guidelines**

These guidelines ensure that development contributions follow best practices, maintain quality, and are consistent across the project.

**2.1 Git Workflow**

* **Fork the Repository:** All contributors should fork the main repository to their own GitHub account and submit pull requests from their fork.
* **Create Feature Branches:** For each new feature or bug fix, create a dedicated branch (feature/your-feature or bugfix/your-bugfix).
* **Pull Request Reviews:** Ensure your pull request (PR) has a clear title and description. Tag relevant issues in the PR. Be prepared for feedback, and revise your PR as needed.

**2.2 Code Standards**

* Follow **PEP 8** for Python code.
* Write clean, readable code with meaningful comments where necessary.
* Use **docstrings** for documenting classes, methods, and functions.
* Avoid hardcoding values; instead, use configuration settings or environment variables.

**2.3 Testing**

* Every new feature or bug fix must be accompanied by relevant **unit tests**.
* Tests should be written using a framework like **pytest**.
* Ensure your code passes all tests before submitting a pull request.
* Code should achieve **at least 80% test coverage** for critical features.

**2.4 Documentation**

* All new features must be documented in the appropriate sections of the project’s documentation.
* Contributions should include both **developer documentation** (API changes, architectural decisions) and **user-facing documentation** (how to use new features).

**2.5 Code Reviews**

* Be open to feedback during code reviews. The goal is to maintain high-quality code, and reviews are an important part of the process.
* When reviewing others' code, provide constructive feedback, focusing on improving the quality and readability of the contribution.

**3. Release Guidelines with QA**

Each release of the project should follow a structured process to ensure stability, performance, and quality.

**3.1 Release Cycle**

* **Alpha Releases:**
  + Early-stage development releases that include incomplete features.
  + Used internally or by selected early testers for initial feedback.
* **Beta Releases:**
  + Feature-complete but potentially unstable.
  + Available for public testing to gather feedback on performance, bugs, and user experience.
* **Release Candidate (RC):**
  + A near-final version with all major bugs fixed.
  + Limited testing for final validation before public release.
* **General Availability (GA):**
  + The official, stable release ready for production use.

**3.2 QA Process**

**3.2.1 Automated Testing**

* **Unit Tests:** All critical components must be covered by unit tests.
* **Integration Tests:** Ensure that new features integrate seamlessly with existing components.
* **Regression Testing:** Run tests to ensure that changes haven’t introduced new bugs in previously stable parts of the codebase.
* **Behavior Driver Development:** Encouraging a BDD approach to deliver features and increments with high-quality by enabling test first development.

**3.2.2 Manual Testing**

* **Functional Testing:** Ensure that all features work as expected from the end user’s perspective.
* **Exploratory Testing:** Conduct exploratory testing to uncover edge cases or unexpected behaviors not covered by automated tests.

**3.2.3 Performance Testing**

* Measure the performance of the platform under various loads.
* **Stress testing:** Evaluate the system’s stability when pushed beyond normal operating conditions.

**3.2.4 Security Testing**

* Perform security audits to check for vulnerabilities.
* Use tools like **OWASP ZAP** or **Bandit** to automate security checks on the codebase.

**3.2.5 User Acceptance Testing (UAT)**

* **Community and Pilot User Testing:** Involve community members or selected teams/enterprises in testing the release candidate.
* Gather feedback on usability, bugs, and overall satisfaction.

**3.3 Release Checklist**

Before releasing any version (Alpha, Beta, RC, or GA):

* Ensure all unit and integration tests pass.
* Complete manual and automated testing.
* Ensure performance benchmarks are met.
* Update documentation (features, API, user guide).
* Tag the release on GitHub (e.g., v1.0.0).
* Create detailed release notes, including:
  + Summary of new features and changes.
  + List of fixed bugs and known issues.
  + Instructions for updating to the new version.
* Ensure the deployment pipelines (CI/CD) are running smoothly.

**3.4 Post-Release**

* **Bug Tracking:** Actively monitor bug reports and feedback following the release.
* **Patch Releases:** If critical bugs are identified after release, issue patch updates as soon as possible (e.g., v1.0.1).
* **Community Feedback:** Gather feedback from users on the new release and incorporate it into future planning.