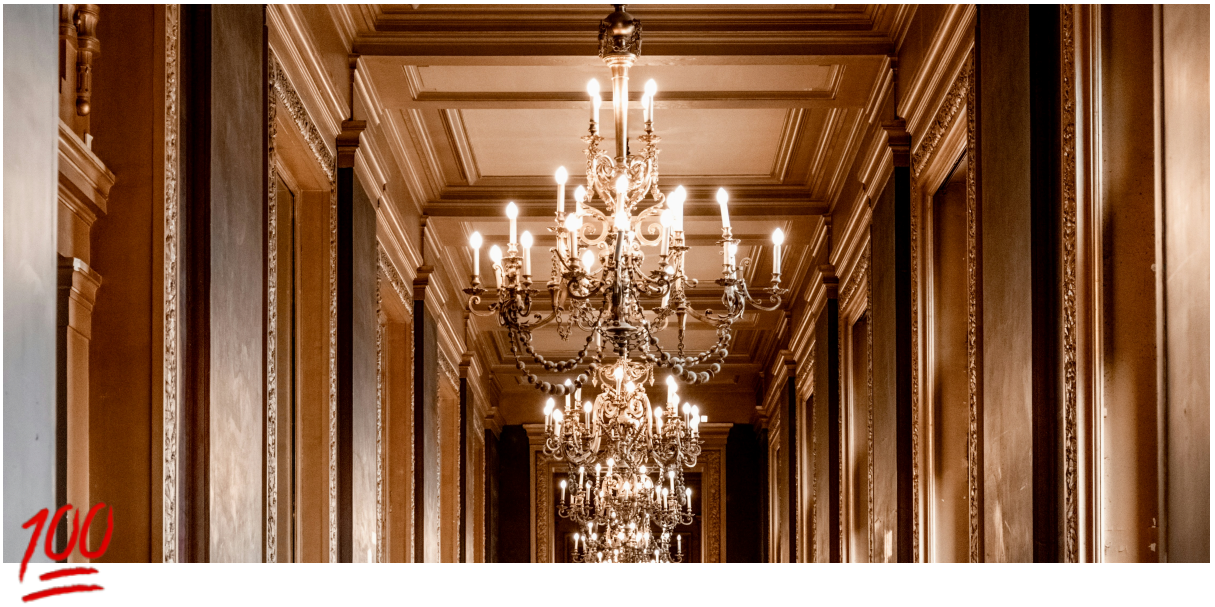

Standards and quality processes

▼ 1. Definition of documentation standards, coding conventions, and activity reporting

▼ Documentation standards

- Use **Markdown (** `README.md` **)** for all instructions.
- Inline documentation in Python (`docstrings`).
- Pull-Request template on GitHub repository.
- Swagger API documentation
- / `docs` directory at project root

▼ Commit Message Format

Specifications for this repository commit messages

Commit structure

```
<type>(<scope>): <short summary>
  |       |           |
  |       |           └⫸ Summary in present tense. Not capitalized. No
period at the end.
  |       |
  |       └⫸ Commit Scope
  |
  └⫸ Commit Type: build|ci|docs|feat|fix|perf|refactor|test|chore
```

The `<type>` and `<summary>` fields are mandatory, the `(<scope>)` field is optional.

Type

Must be one of the following:

- **build** : Changes that affect the build system or external dependencies (example scopes: gulp, broccoli, npm)
- **ci** : Changes to our CI configuration files and scripts (examples: CircleCi, SauceLabs)
- **docs** : Documentation only changes
- **feat** : A new feature
- **fix** : A bug fix
- **perf** : A code change that improves performance
- **refactor** : A code change that neither fixes a bug nor adds a feature
- **test** : Adding missing tests or correcting existing tests
- **chore** : Chore changes (update .gitignore, dependencies, etc)

Scope

The scope should be name of the file, the directory or the feature involved in the commit.

Here are some examples:

- gitignore
- main

- action
- readme

Summary

Use the summary field to provide a succinct description of the change:

- use the imperative, present tense: "change" not "changed" nor "changes"
- don't capitalize the first letter
- no dot (.) at the end

GitHub Branches

If you want to work on a feature you have to create a branch for it. To create a branch you have to create it directly on a GitHub issue. For the name Github will automatically use the following pattern:

```
<issue-number>-<short-description>
```

You can use it as is.

GitHub Issues

If you want to create an issue you have to use the following rules:

- Name the issue with a short description
- Describe the issue with a description
- Add labels to the issue

GitHub Pull Requests

When you do a pull request you have to use the following rules:

- Name the pull request with the issue number and the short description
- Add minimum two reviewers to the pull request
- You have to squash your commits to one commit with the number of the issue and the short description

▼ Coding style

- Code formatting and linting. Look at 🛠️ Tech Stack Overview.
- Respect of coding conventions for each language's framework (**PEP8**).
- "Clean code" : clear naming for variables, methods, classes, etc.
- Weak coupling.
- YAGNI.
- DRY.

▼ Activity reporting

- Weekly progress reports (GitLab/GitHub issues + milestones).
- Teams messages to track team progress and problems.
- Test coverage reports (`coverage.py`).

▼ 2. Consideration of accessibility for people with disabilities

- Proper **color contrast** between text and background for readability by visually impaired users.
- Provide **scalable fonts** and allow zoom without breaking the interface.
- Ensure that interactive elements (buttons, forms, menus) have clear focus indicators and labels.
- All interactive elements must be accessible via **tab navigation** in a logical order.
- Use clear, simple, and consistent language.
- Avoid unnecessary jargon, or explain it when unavoidable.
- Provide error messages that are **explicit and helpful**, guiding the user toward resolution.

▼ 3. Implementation of quality control activities

- For each Pull Request :
  - At least 2 people as reviewers.
  - Use of GitHub Copilot as a bonus reviewer.
  - CI policy with build, test, and linting validation.
  - SonarQube code quality and security gate.
  - CodeScene code health gate.