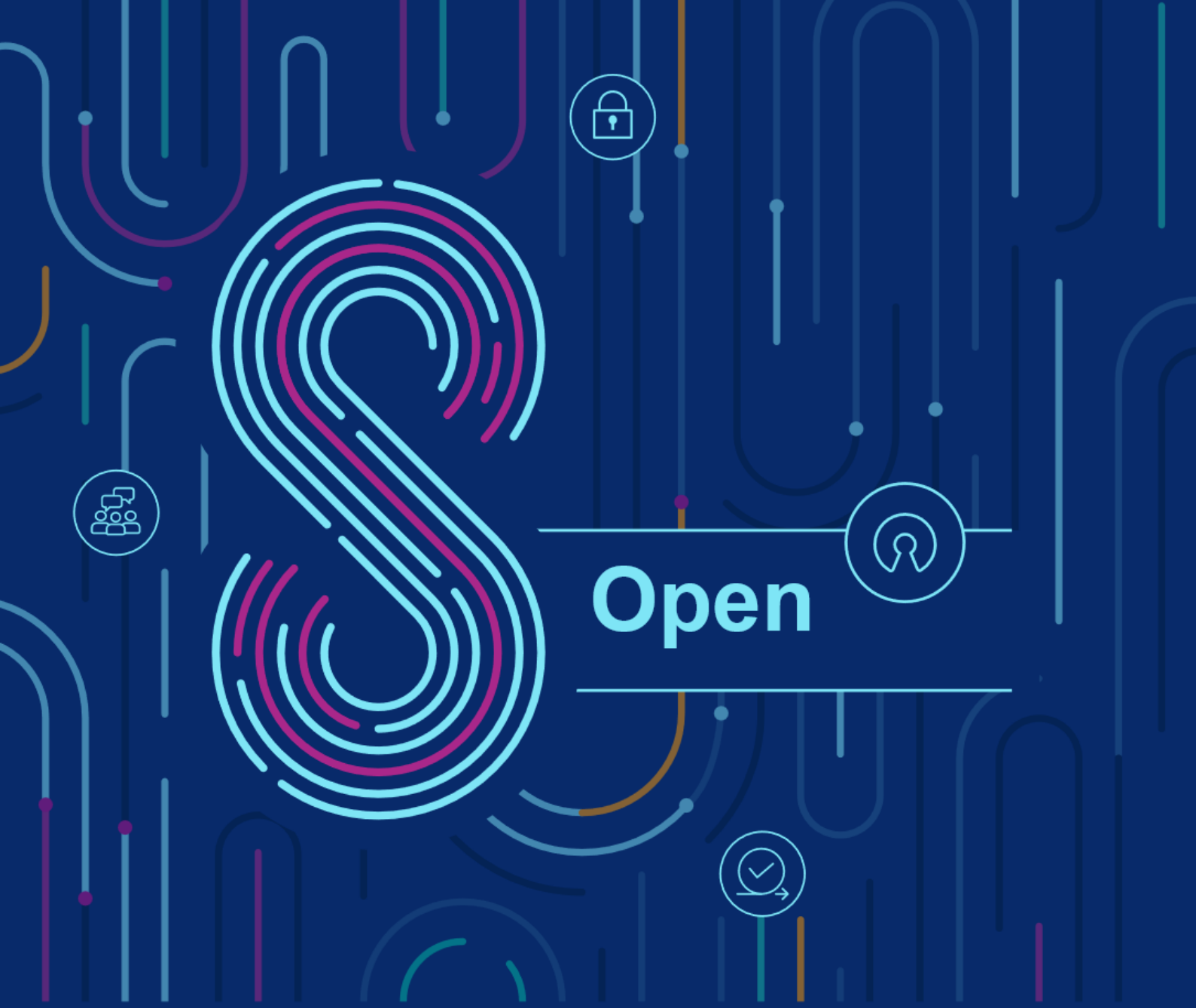

Simpl-Open Contribution

Guidelines

European Commission

Table of Content

Table of figures

1 Introduction

This document provides a framework of compliance criteria and processes for guiding the Simpl-Open community contributions.

The document starts by specifying the compliance criteria for a selection of contributions. These are meant to be used to guide contributors in providing acceptable quality, but also to help evaluators during their approval process.

Then it continues to provide the steps (how to) the community must take to engage with:

- **Comments on website pages:**

  This includes any use of the “comment” or “discussion” functionality on the Simpl programme website.

- **Forum discussions:**

  Any use of the “forum” functionality on the Simpl programme website.

- **Architecture:**

  Any suggested additions or changes to the architecture of Simpl-Open as documented here. This **does not** include the implementation of code necessary to affect the architecture changes. It **does** include changes to Archi models and UML models. The result of accepted architecture changes will be new issues and backlog items for future sprints.

- **Issues:**

  Any defect or discrepancy in code or documentation. If approved and accepted this could lead to changes in documentation, architecture or code.

- **Code contributions:**

  Any changes to the code base at code.europa.eu.

- **Requirements:**

  Any change or addition to the functional or non-functional requirements, provided here.

- **Strategic cooperation:**

  This includes requests like, asking for help with using Simpl-Open to build data spaces, suggestions regarding code-sharing, reciprocal support suggestions, joint data space implementations, etc.

2 Contribution Guidelines and How to's

This chapter answers the question, "How do I contribute to Simpl-Open as an external party?" It contains step-by-step instructions, templates and compliance criteria.

All contributions will be evaluated according to the following criteria. Firstly, the EU Code of conduct on countering illegal hate speech online and the Contributor Covenant. If the contribution is in breach of this Code of Conduct, it will be removed.

Each contribution will also be evaluated against the specific compliance criteria mentioned in the individual sections below.

2.1 Compliance criteria

2.1.1.1 Comments on website pages

Note: Posts must please be in English. This increases the accessibility across our community.

- **Respectful:**
  The comment is polite, constructive, and free from personal attacks or inflammatory language.
- **Clear and Concise:**
  The comment is easy to understand, uses clear language, and avoids unnecessary jargon or ambiguity.
- **Relevant:**
  The comment directly addresses the topic, code, or issue at hand.
- **Actionable (if feedback):**
  Suggestions, critiques, or questions are specific and provide a clear path for improvement or resolution.
- **Fact-based:**
  Opinions are supported by evidence, examples, or references where appropriate.
- **Inclusive:**
  The comment welcomes input from others and avoids exclusionary or dismissive language.
- **Follows Project Guidelines:**
  Adheres to the project's Code of Conduct and contribution guidelines.
- **Well-Formatted:**
  Uses proper formatting (e.g., Markdown, bullet points, code blocks) to enhance readability.
- **Attribution (if applicable):**
  Gives credit to others for their ideas, code, or contributions when referenced.
- **No Sensitive Information:**
  Does not include confidential, proprietary, or personally identifiable information, other than the community member themselves.

2.1.1.2 Forum discussions

Forum discussions have the same evaluation criteria as Comments.

2.1.1.3 Architecture changes

- **Motivation and context are clearly described:**
  Clearly explain why this change is needed, referencing the underlying problem, opportunity, or user need. Provide background information, relevant history, or links to prior discussions to help reviewers understand the significance and urgency of the proposal.
- **Affected components are identified:**
  List all modules, subsystems, APIs, or documentation sections that will be impacted. Specify both direct and indirect dependencies to ensure the scope is well understood and nothing is overlooked.
- **Proposed solution is detailed and justified:**
  Describe the solution in depth, including technical approaches, design decisions, and

rationale. Justify why this approach was chosen over others, referencing requirements, constraints, or architectural principles where relevant.

- **Alternatives are considered and documented:**
  Summarise other solutions or approaches that were evaluated, including their pros and cons. Explain why they were not selected, demonstrating that the chosen solution is the most appropriate for the context.
- **Impact (including benefits and drawbacks) is analysed:**
  Discuss all expected effects of the change, both positive and negative. Consider impacts on all non-functional and functional requirements, e.g. users, maintainers, performance, scalability, security, future development, etc. Be transparent about any trade-offs or limitations.
- **Backward compatibility is addressed:**
  State whether the change preserves compatibility with existing functionality, APIs, or data. If breaking changes are introduced, detail their nature, justification, and any mitigation strategies for affected users.
- **Migration/transition plan is included (if applicable):**
  If the change requires users or maintainers to act, provide a clear, step-by-step migration or transition plan. Include timelines, deprecation notices, and tools or scripts to assist with the process.
- **Stakeholders and reviewers are listed:**
  Identify all individuals, teams, or external partners who should be informed, consulted, or involved in the review and approval process. Include maintainers of affected components, domain experts, and key community members.
- **Relevant attachments (diagrams, files, screenshots) are provided:**
  Attach or link to supporting materials that clarify the proposal, such as architecture diagrams, workflow charts, sample configurations, or screenshots. Ensure all files are up to date and accessible.
- **Acceptance criteria for success are defined (e.g., measurable outcomes, target performance, user impact):**
  List specific, testable conditions that must be met for the contribution to be considered complete. Include functional requirements, performance benchmarks, compliance goals, and user experience outcomes as appropriate.
- **Required documentation updates are identified:**
  Specify all documentation that will need updating because of this change, including user guides, API references, changelogs, and onboarding materials. Outline who will be responsible for making these updates and when they will be completed.
  Note: *The documentation needed may vary and it is left to the approver to make the final decision.*

2.1.1.4 Issues

- **Title and summary are clear and descriptive:**
  Provide a concise, informative title and a summary that accurately captures the essence of the issue. This helps maintainers and contributors quickly understand the nature and scope of the problem immediately.
- **Severity is appropriately assigned and justified:**
  Assign a severity level (e.g., Critical, High, Medium, Low, Trivial) that reflects the issue's impact on users and the project. Include a short explanation for your choice so triagers and developers can prioritise effectively.
- **Steps to reproduce are detailed and reliable:**
  List all the steps needed to consistently reproduce the issue, including any setup or configuration required. Clear reproduction steps ensure others can verify and address the issue without ambiguity.
- **Expected and actual results are clearly described:**
  State what you expected to happen and what occurred. This contrast helps reviewers quickly identify the deviation and understand the nature of the problem.
- **Environment details are fully specified:**
  Include all relevant information about the environment where the issue was observed, such as

operating system, browser version, device, app version, and any special configurations. This context is crucial for reproducing and diagnosing the issue.

- **Relevant logs, screenshots, or attachments are provided:**
  Attach any supporting evidence—such as error logs, stack traces, screenshots, or screen recordings—that can help maintainers and developers investigate and resolve the issue more efficiently.
  Note: *Always ensure that anything supplied does not expose sensitive information like personal identifiable information.*
- **Frequency and impact are explained:**
  Indicate how often the issue occurs be as specific as possible e.g. it occurs 100% of the time. Describe its impact on users or workflows. This information helps prioritising the issue and assessing its urgency.
- **Related issues or dependencies are linked (if applicable):**
  Reference any related issue reports, issues, or dependencies that provide additional context or may be affected by this issue. Linking related items helps maintainers see the bigger picture and avoid duplicated effort.
- **Acceptance criteria for resolution are defined (e.g., issue is fixed, no regressions, tests pass):**
  Clearly state what needs to be true for the issue to be considered resolved, such as the issue no longer occurring, no new regressions, and all relevant tests passing. Well-defined criteria ensure everyone agrees on what "done" means. Provide details of the Definition of Done.
- **Any required documentation updates are noted:**
  Identify any documentation that needs updating because of this issue or its fix, such as automated tests, user guides, FAQs, or API references. Keeping documentation current ensures users and contributors have accurate information.

2.1.1.5 Code contributions

- **Follows Contribution Guidelines:**
  Adheres to the project's coding practices and guidelines, and Code of Conduct.
- **Descriptive Title and Summary:**
  Pull Request (PR) title and description clearly state what the change does and why.
- **Linked Issues:**
  References related issues or tickets (e.g., "Fixes #123").
- **No Duplicate Pull Request:**
  Confirms there are no existing open Pull Request(s) for the same change.
- **Scope is Focused:**
  Addresses a single issue or feature; avoids unrelated changes.
- **Self-Reviewed:**
  Contributor has reviewed their own code for errors and clarity.
- **Code is Clear and Readable:**
  Uses meaningful names, clear logic, and comments where needed.
- **Follows Project Style**:
  Matches formatting, naming, and architectural conventions.
  Note: *If the code change requires **any** architecture change, the contributor must log a separate issue for the architecture change itself too. This architecture change must be approved by the OSGB before the contributor can start the code change.*
- **Dependencies Managed:**
  Adds, updates, or removes dependencies responsibly and documents changes.
- **Meets Requirements:**
  Implements all requirements or acceptance criteria for the issue/feature.
- **Backward Compatible:**
  Does not break existing functionality unless explicitly intended and documented.
- **No Unintended Side Effects:**
  Changes are isolated to intended areas.
- **Tests Included/Updated:**
  Adds or updates unit, integration, or end-to-end tests as appropriate.
- **All Tests Passed:**
  Confirms all tests passed locally and/or in CI.

- **Test Coverage Maintained:**
  Maintains or improves test coverage for affected code.
- **Documents Updated:**
  Updates documentation, comments, or examples as needed.
- **Changelog/Release Notes:**
  Adds or updates changelog entries if required.
- **No Sensitive Data:**
  Does not include credentials, secrets, or personal information.
- **License Compliance:**
  Code and dependencies comply with the project's license.
- **Ready for Review:**
  All checklist items completed, and PR is ready for maintainer review.

2.1.1.6 Requirements

- **Clear Documentation:**
  Requirement is well-documented with detailed descriptions.
- **User story (or stories):**
  A short, informal description of the requirement written from the end user's perspective. It focuses on what the user needs and why, rather than detailing how the system should be built.
- **Acceptance criteria:**
  The set of conditions that the requirement must meet to be considered complete and acceptable by the product owner or stakeholders.
- **Community Input:**
  Discussions initiated, and feedback solicited (link to discussion).
- **Scalability**:
  Requirement can scale with the project's growth.
- **Compatibility**:
  Compatible with existing systems and technologies.
- **Security**:
  Security considerations addressed.
- **Sustainability**:
  Long-term sustainability considered.
- **Openness**:
  Adheres to the open-source principles.
- **Flexibility**:
  Allows for future changes or extensions.
- **Legal Compliance:**
  Complies with relevant legal and licensing standards.
- **Risks/Dependencies:**
  Identifies all major risks and dependencies.

2.1.1.7 Strategic cooperation

- **Clear Purpose and Objectives:**
  The request clearly states the goals, expected outcomes, and strategic value for all parties involved.
- **Alignment with Business and Community Goals:**
  The proposed cooperation aligns with the business objectives and open-source strategy of each party.
- **Stakeholder Identification:**
  All relevant internal and external stakeholders are identified and included in discussions.
- **Scope Defined:**
  The scope, boundaries, and deliverables of the cooperation are well-defined.
- **Roles and Responsibilities:**
  Each party’s roles, responsibilities, and contributions (tangible and intangible) are specified.

- **Governance and Decision-Making:**
  Governance structure, decision-making processes, and conflict resolution mechanisms are outlined.
- **Legal and IP Considerations:**
  Legal, licensing, intellectual property, and compliance issues are addressed.
- **Resource Commitment:**
  Required resources (funding, personnel, infrastructure, etc.) are identified and committed.
- **Performance Metrics and KPIs:**
  Key performance indicators and success metrics are defined for tracking progress and outcomes.
- **Timeline and Milestones:**
  Realistic timeline, phases, and key milestones are included.
- **Communication Plan:**
  Communication channels, reporting cadence, and points of contact are established.
- **Risk Assessment:**
  Potential risks, dependencies, and mitigation strategies are identified.
- **Sustainability and Exit Strategy:**
  Long-term sustainability, renewal, and exit/termination provisions are considered.
- **Openness and Community Principles**:
  The cooperation adheres to open-source principles, transparency, and community engagement.
- **Review and Feedback Mechanism:**
  Mechanisms for regular review, feedback, and adaptation are established.

2.2 How to comment

On the Simpl programme website comments can be done directly on certain pages.

You must be logged in to comment.

To log in, click on the “Log in” link at the top right (see: Figure 1 - Simpl programme login page).

Figure 1 - Simpl programme login page

Follow the steps to **create a new account** or to **log in** if you already have an account.

If the page has the “Comments” functionality enabled, it can be found **at the bottom**.

It will look like Figure 2 (Figure 2 - Add comment to existing conversation) for **pages with comments**. Comments are displayed in chronological order with the newest comment at the bottom.

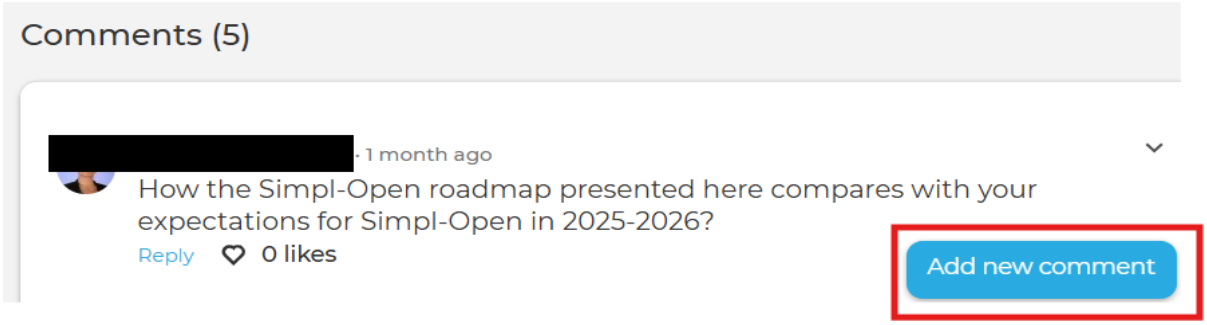

Figure 2 - Add comment to existing conversation

The comment section on website **pages without comments**, will look like Figure 3 (Figure 3 - Add first comment).

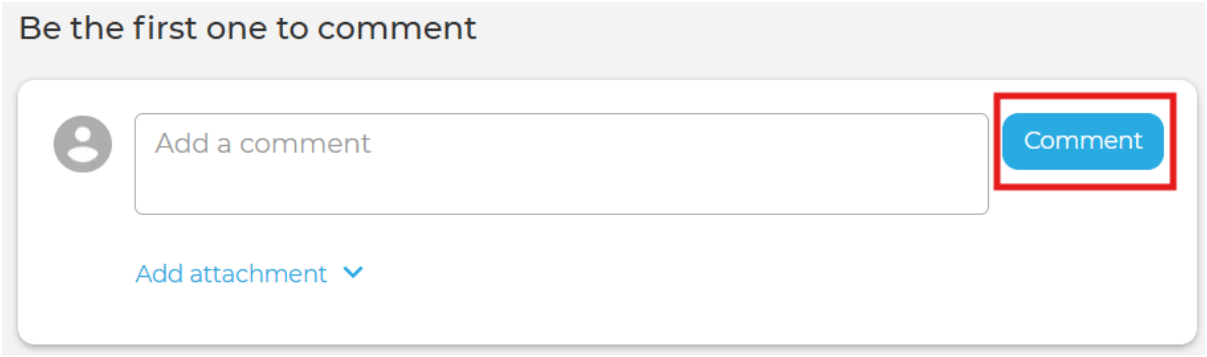

Figure 3 - Add first comment

If you want to add an attachment, click on “Add attachment” and follow the on-screen instructions.

The upload requirements are:

- The maximum size limit for attachments is 200 MB.
- The file must be one of the following types:
  - txt
  - pdf
  - doc
  - docx
  - xls
  - xlsx
  - ppt
  - pptx
  - csv
  - png
  - gif
  - jpg
  - jpeg

2.3 How to discuss forum topics

You don't have to join if you just want to read the public discussions on the forums.

If you want to contribute to on an existing discussion or start your own, you must be logged in (see the steps on how to log in or create an account).

2.3.1.1 How to join the Simpl Forum:

1. Go to the top level forum page.
2. If you are not logged in, click Log in.
3. Click "Join" and complete the process (see Figure 4 – View discussions).

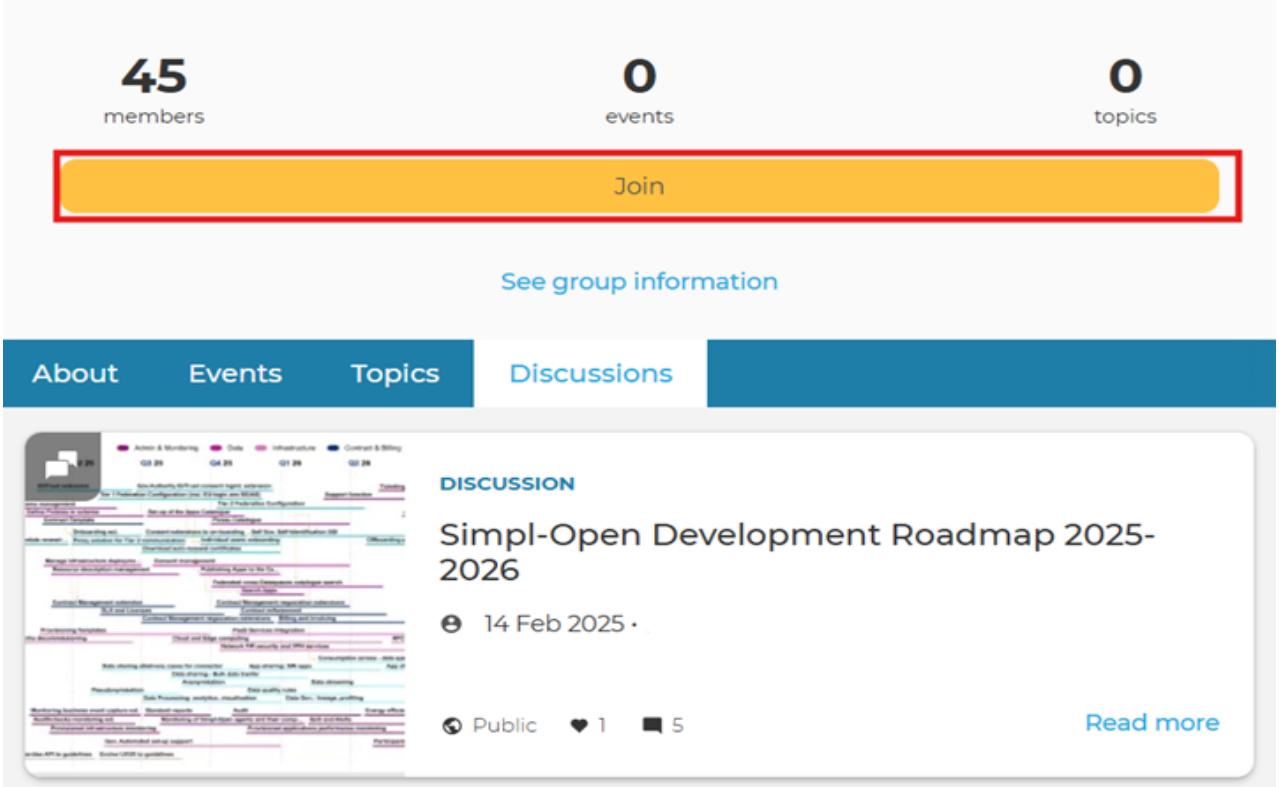

Figure 4 - View discussions

How to contribute to an existing discussion:

1. Find the discussion you are interested in.
2. Click on the title or "Read more" (see Figure 5 – Discussion example).

   For example, the "Modularity and adaptability" discussion below.

Figure 5 - Discussion example

3. Go to this text area at the bottom of the page (see Figure 6 – Comment on existing discussion: option 1).

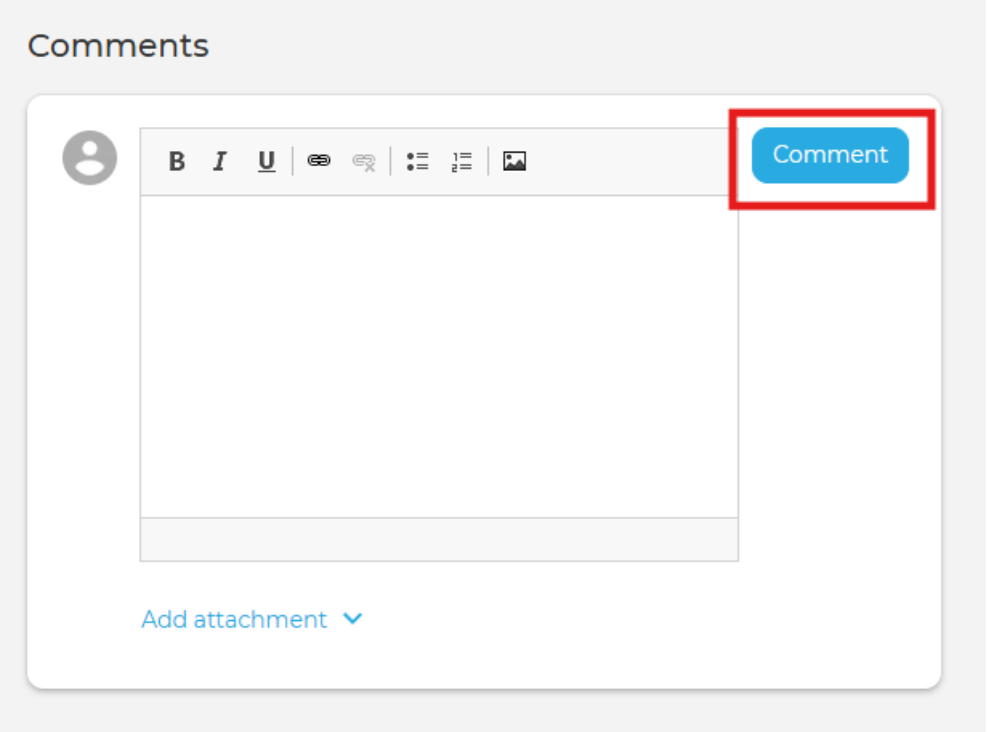

Figure 6 - Comment on existing discussion: option 1

4. Enter your comment.
5. Click on the “Comment” button.

On pages that have more content, you might also be presented with a button labelled “Add new comment”. Refer to Figure 7 - Comment on existing discussion: option 2:

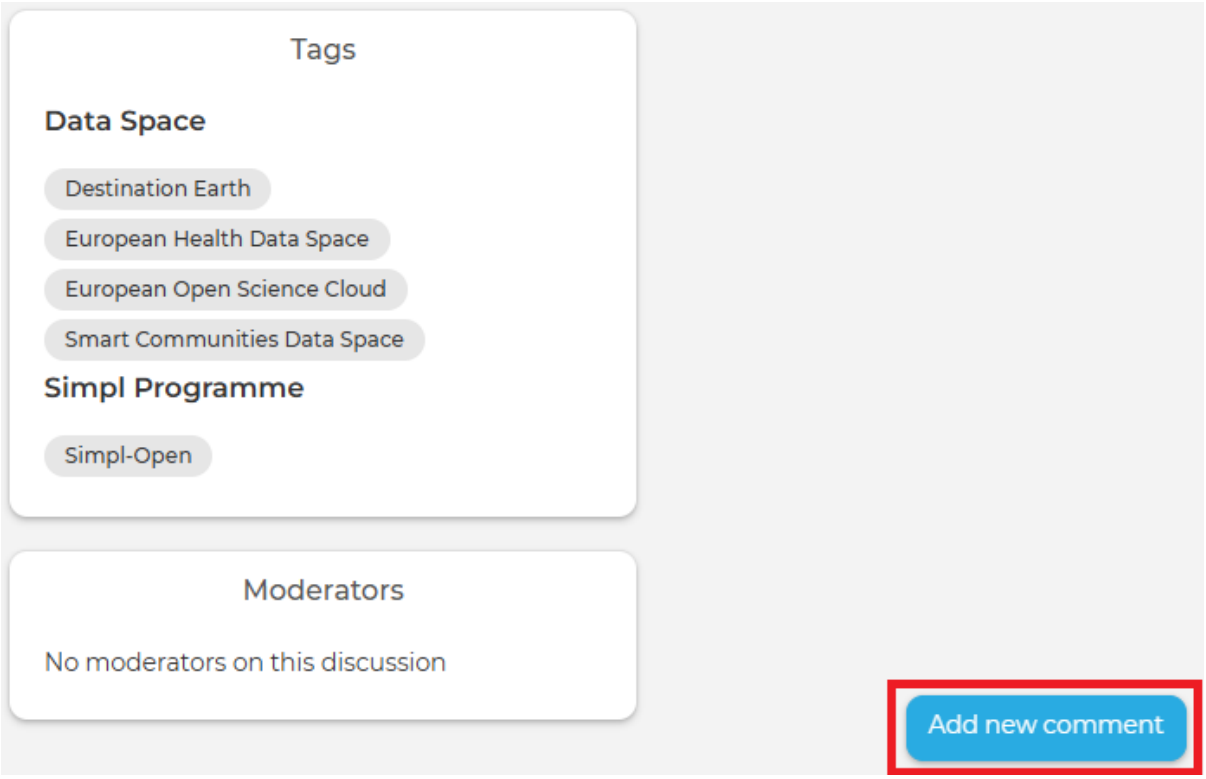

Figure 7 - Comment on existing discussion: option 2

Clicking this button will immediately move you to the “Comments” text area at the bottom of the page. The rest of the functionality is the same as described above.

2.3.1.2 How to start a new discussion:

If you cannot find any suitable discussion for your topic, follow the instructions for Creating a new Forum discussion.

2.4 How to contribute requirements

The Simpl-Open requirements are available on the Simpl programme website. Start the process by going to the top-level requirements page.

You can read the requirements without logging in, but if you want to contribute, you must log in.

Once you log in, you can navigate to the **existing requirement** you want to contribute to.

Scroll to the bottom of the requirement page.

Add your comment in the text area and click “Comment” (see Figure 8 – Comment on requirement).

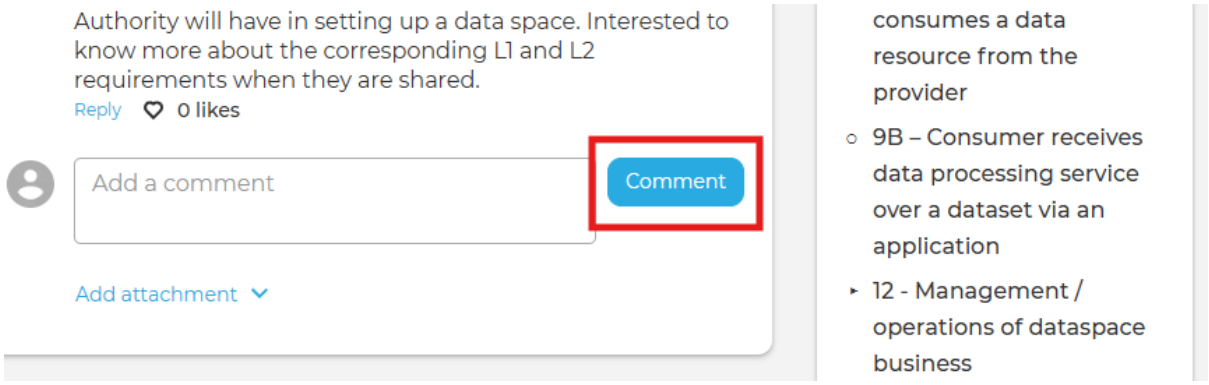

Figure 8 - Comment on requirement

If you want to suggest a **new requirement**, please find the relevant business process here. Then add your comment to the selected business process as described above in Figure 8 - Comment on requirement.

The Simpl-Open team will acknowledge comments within 3 days and endeavour to provide details on the way forward, within 15 working days.

Please note:

This does not mean the requirement will be accepted into the roadmap

You are also welcome to go to the roadmap discussion on the forum and to comment there. Please see the section on forums and discussions for more details.

2.5 How to contribute to the architecture

2.5.1.1 Download the architecture document:

The architecture document is available on GitLab.

If you prefer the ArchiMate models, you can find the instructions for setting up Archi here.

To contribute to the architecture, you can use the GitLab issue tracker.

Log an issue, detailing the change, by completing the form provided (you can find a markdown example here).

The Simpl-Open team will acknowledge comments within 3 days and endeavour to provide details on the way forward, within 15 working days.

Please note:

This does not mean the suggested architecture change will be accepted though.

2.5.1.2 Clone the ArchiMate repository:

The architecture models are also available in a repository.

Follow the instructions for downloading, installing and configuring Archi.

The same page also has the instructions for importing the models into Archi.

2.6 How to report an issue

Simpl-Open uses GitLab issue tracker and service desk. If you find any issues in the architecture, code, tests or documentation use the relevant issue trackers based on the type of issue. The individual issue trackers are discussed below, but in general the steps for logging an issue are:

I. Go to the relevant issue tracker; links and details are provided below.
II. Search that specific issue tracker; someone might have logged your issue already.
III. If you find the issue in the tracker, add a comment and provide more information if relevant.
IV. If you cannot find the issue, create a new one and use the template provided to add information. Give the team as much information as you can.
V. The issue will be assigned to the relevant team, and we will communicate with you via the issue tracker.

The specific issue trackers are:

1. Architecture issue tracker:

   For all issues directly related to the published architecture documentation, use the Architecture Issue Tracker or email address.

   Make sure to give us as much information as possible. This includes screenshots and/or extracts from the relevant documentation.

Please note:
If you decide to use the email address instead of the issue tracker, you will not automatically be provided with an issue template. Copy the Architecture Contribution template and use that to provide the necessary information in your email.

2. Issue tracker:

   If you find any issues in the codebase, use the Code Issue Tracker or email address.

   Provide all relevant log files, stack traces and steps required to replicate the issue. Please include all the data you used too as it is an integral part of replicating the issue.

   Please note:
   This issue tracker is for code in the Simpl-Open repositories only. The code dependencies used in Simpl-Open all have their own repositories and issues trackers.

   If you decide to use the email address instead of the issue tracker, you will not automatically be provided with an issue template. Copy the Issue logging template use that to provide the necessary information in your email.

3. Documentation issues tracker:

   If you encounter any issues in the installation documentation or in the user manual, go to the Documentation Issue Tracker or email address.

   Please make sure to give us as much information as possible. This includes screenshots and/or extracts from the relevant documentation.

4. Security related issues:

   For any security related issues, use the Security Issue Tracker or email address.

   Supply as much information of the as you can. If you can provide refences to any of the public vulnerability trackers like the CVE it will be appreciated.

   When a security related issue is logged via the GitLab security issue tracker, then an email notification is sent to following parties:

   - T1 SPOC,
   - PSO SPOC
   - DG Connect SPOC.

   Once the email notification is sent to these parties, the "**Simpl-Open Security Coordination Meeting**" takes over the responsibility to investigate, prioritise and resolve the security issue.

5. Testing issue tracker:

   If you find any issues related to testing, whether it is automated or manual testing, please use the Testing Issue Tracker or this email address to log it.

2.7 How to contribute to the code

Code contributions can be submitted by following the steps below:

1. Request access to the Gitlab repositories by following the onboarding process for code contributors.
2. Setup Git.
   - Before contributing, ensure your Git configuration and GitLab profile are correctly set up with your real name and email, as this helps accurately track the contributions.
   - If you want to use SSH keys to access GitLab, see the official GitLab documentation for SHH setup.
   - If you prefer personal access tokens for working with GitLab, see the official GitLab documentation for access tokens.
3. Get a local copy of the code.

- Clone the Repository:
    - Clone the repository to your local machine.
- Create a branch:
    - Name the branch based on the ticket number you have received.

4. Make changes
    - Code Guidelines:
        - Follow the style of existing code and best practices. Keep code simple and concise. See the guidelines in this document for more information.
        - Keep merge requests small, clear, and atomic. Limit each merge request to a few commits (no more than 5).
5. Test locally first
    - Run unit tests before and after making changes to ensure nothing breaks.
    - Fix any significant errors and warnings before submitting merge requests.
    - Pay attention to code coverage and linting issues.
6. Commit Messages
    - Commit changes to your branch and remember to pull the latest changes from upstream too.
    - Use a clear format: component name, a brief imperative sentence, and explanatory paragraphs if needed.
7. Submitting Changes
    - Perform a final quality check on your changes.
    - Push changes to an upstream branch named exactly like your local branch.
    - Create a merge request on GitLab against the current development branch.
    - **Note: Ensure the merge request is linked to the Jira ticket you received.**
8. Build pipeline
    - The automatic build process will run unit tests, code-coverage reports, linting checks, dependency checking, static code analysis, performance tests, etc. This is provided to the reviewer.
9. Managing Submissions
    - A reviewer will be assigned to your merge request. Address feedback carefully and resubmit if necessary.
10. Reviewing Merge Requests
    - Reviewers should add comments via GitLab UI or email.
    - Approval from all reviewers is required before committing a merge request.
11. Responding to Reviews
    - Assume positive intent and understand the reviewer's perspective. Avoid being argumentative and work collaboratively with maintainers.
12. Code contribution validation
    - When we receive a contribution, we first check if it complies with the contribution standards and provide feedback to the contributor if it does not.

2.8 How to initiate strategic cooperation

If you want to engage with the Simpl-Open community regarding strategic cooperation such as partnerships, please send an email to CNECT-SIMPL@ec.europa.eu.

After assessing your request, we will provide you direct feedback about the next steps.

Appendix 1: How to create an account

Note: *You will see references to "EU Login" and "Sign in with EU Login" in this section. This is due to the Simpl programme website using the "EU Login" functionality to authenticate users. If you do not have an EU Login yet, you have to create an EU Login account.*

1. Go to the Simpl programme website.
2. Click on "Log in" at the top right of the page (see: Figure 12 - Simpl programme website create account: login).

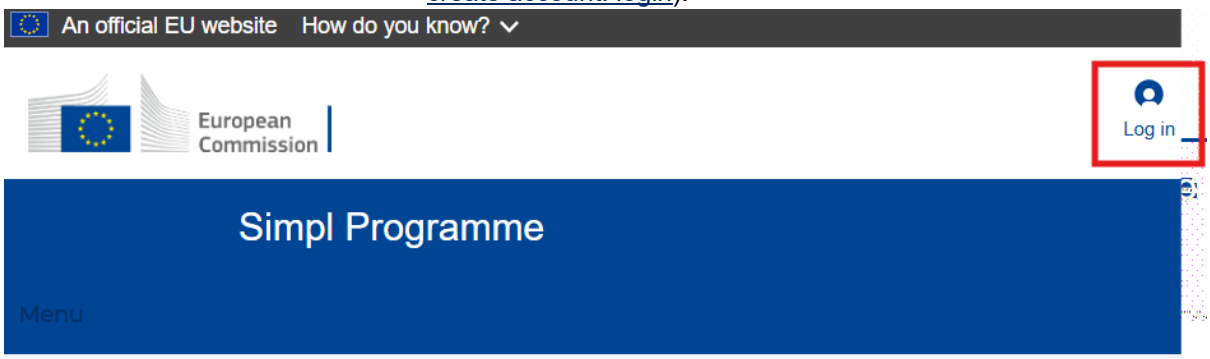

Figure 9 - Simpl programme website create account: login

3. Click on "Sign in with EU Login" (see: Figure 13 - Simpl programme website create account: EU login) and **follow the process** for creating a new account.
   The EU Login site will guide you through this process.

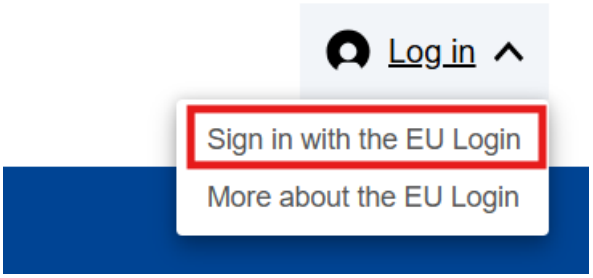

Figure 10 - Simpl programme website create account: EU login

4. When you complete the EU Login process you will be taken to the Simpl programme website.

Appendix 2: How to log in

To log in, click on the “Log in” link at the top right (see: Figure 14 - Simpl programme website login).

Figure 11 - Simpl programme website login

If you do not have an account yet, follow the steps to **create a new account.**

Appendix 3: Onboarding as code contributor

1. **Create EU Login Account**

- Visit EU Login.
- Register with an email address of your choice.
- Download the EU Login application to your phone.
- In 'My Account,' add your mobile device to your EU Login account.

2. **Join code.europa.eu**

- Visit code.europa.eu.
- Log in with your EU Login account and request access.
- Wait for site administrators at code.europa.eu to grant access. You will receive a notification once access has been granted.
  - You may receive an email asking for the reason for your request; if so, mention that you would like to join the Simpl-Open project as a contributor.
  - Queries or support requests can be addressed to: DIGIT-OSPO@ec.europa.eu.

3. **Join Simpl-Open as a contributor**

Prerequisite: Wait for site administrators at code.europa.eu to grant access.

- Visit code.europa.eu and log in with your EU Login account.
- Join the project About code.europa.eu to access platform wikis.
- Request access to the group Simpl Contributions.

Appendix 4: Contribution and request stages

The contribution requests detailed in this document follow these steps:

1. Receive
2. Acknowledge
3. Evaluate
4. Approve / decline / postpone
5. Prioritise
6. Respond
7. Resolve / close
8. Request feedback

1 Receive

A new contribution or request on any of our supported channels. A list of these is curated by the community manager and presented to the attendees of the weekly Open-Source Governance Board meetings.

2 Acknowledge

The Community team acknowledges receipt of an enquiry or when a new member is joining. This happens within 3 working-days of the contact.

3 Evaluate

The relevant stakeholders meet weekly, to discuss the merits of the cooperation requests.

The goals are:

- to decide whether to approve, decline or postpone each request;
- to assign the responsibility to respond to the correct person, by default it is the community manager;
- to assign the parties who must implement (this drives the "prioritise" step below).

4 Approve / decline / postpone

The designated person responds to the collaborator. They inform them of:

- the decision;
- provide reasons where applicable;
- the fact that there will be more communication to follow, if the decision was to 'approve'.

5 Prioritise

The parties identified in step "Evaluate", set the priority and timelines for the request.

6 Respond

The community manager informs the collaborator of the next steps, based on the outcomes of the "Prioritise" step.

7 Resolve / close

The rules for resolution, deliverables, and outcomes depend on the contribution processes detailed in this document.

8 Request feedback

The community manager requests feedback from the collaborator. This will help us improve the processes and support our collaborators better.

Appendix 5: Issue/contribution templates

These templates are meant to be used in issue logging via the GitLab web interface and email issue logging via GitLab service desk.

Note: This is a markdown formatted template.

Architecture contribution template

 # Architecture Contribution

Summary

****Title:****

[Brief and descriptive title of the issue]

****Description:****

Summarise your contribution in a few sentences, outlining what it aims to address or improve.

Details

****Motivation:****

Explain why this change is necessary or beneficial.

****Affected Component(s):****

[List the modules or areas impacted]

****Context:****

[Explain the current state and why this request or issue is relevant.]

****Proposed Solution:****

[Describe the proposed solution or change in detail.]

****Alternatives Considered:****

[List any alternative solutions that were considered and why they were not chosen.]

****Impact:****

[Discuss the potential impact of the proposed change on the project, including benefits and potential drawbacks.]

****Backward Compatibility:****

[Will this change break existing functionality? If so, how?]

****Migration/Transition Plan (if applicable):****

[Describe how the transition from the current to the new architecture will be managed.]

****Stakeholders/Reviewers:****

[List key people who should review or be notified about this issue.]

****Attachments:****

[Attach any relevant files, diagrams, or screenshots.]

Checklist (for maintainers only!)

__Do not fill this part, it is for maintainers only.__

Use this checklist to ensure your architecture contribution is ready for review and implementation:

- [ ] Motivation and context are clearly described
- [ ] Affected components are identified
- [ ] Proposed solution is detailed and justified
- [ ] Alternatives are considered and documented
- [ ] Impact (including benefits and drawbacks) is analysed
- [ ] Backward compatibility is addressed
- [ ] Migration/transition plan is included (if applicable)
- [ ] Stakeholders and reviewers are listed
- [ ] Relevant attachments (diagrams, files, screenshots) are provided
- [ ] Acceptance criteria for success are defined (e.g., measurable outcomes, target performance, user impact)
- [ ] Required documentation updates are identified

Issue logging template

 # Issue Logging Template

Essential Information

- ****Title:****

(Concise, descriptive summary of the issue and affected feature/module)

- **Severity Level:**

- [ ] Critical

- [ ] High

- [ ] Medium

- [ ] Low

- [ ] Trivial

Severity Level	Description	Examples
Critical	Issues that cause system crashes, data loss, or major malfunctions.	System crashes, security vulnerabilities, data corruption.
High	Issues that significantly impact core functionalities but don't cause a complete system failure.	Major feature not working, significant performance issues.
Medium	Issues that affect non-critical functionalities or cause minor errors.	UI glitches, minor feature malfunctions, occasional slowdowns.
Low	Minor issues or cosmetic issues with minimal impact on functionality.	Typos, minor layout issues, non-disruptive visual inconsistencies.
Trivial	Very minor issues that do not affect functionality at all.	Minor spelling errors, slight misalignment of elements.

Detailed Description

- **Summary:**

(One or two sentences summarising the core of the issue)

- **Description:**

(Detailed explanation of the issue, including what is failing, where, and how it was discovered)

- **Steps to Reproduce:**

1. (First step)
2. (Second step)
3. (Continue as needed)

- **Expected Result:**

(Describe the correct/intended behaviour)

- **Actual Result:**
(Describe what happens, including any error messages or unexpected outputs)

Environment

- **Operating System:**
(e.g., Windows 11, macOS 14.2, Ubuntu 22.04, etc.)

- **Browser/Version:**
(If applicable; e.g., Chrome 125, Firefox 120, etc.)

- **Device:**
(e.g., Desktop, MacBook Pro 2021, iPhone 15 Pro, etc.)

- **App/Software Version:**
(e.g., v3.2.1-beta)

- **Screen Resolution:**
(If relevant)

- **Network/Location:**
(If the issue is network-dependent or location-specific)

- **Relevant Software/Plugins:**
(List any other software, plugins, or extensions that may influence the issue)

Supporting Materials

- **Visual Proof:**
(Attach screenshots, screen recordings, or annotated images)

- **Source URL:**
  (If web-based, include the exact URL where the issue occurs)

- **Other Attachments:**
  (Configuration files, crash dumps, relevant logs, error messages, or stack traces)

Additional Context

- **Frequency:**
  (Does it occur always, sometimes, rarely? Under what conditions?)

- **Related Issues:**
  (Link to any similar or related issue reports, issues, or tickets)

- **Workarounds:**
  (Any known temporary fixes or ways to avoid the issue)

- **Impact:**
  (Describe how this issue affects users, business processes, or other systems)

- **Dependencies:**
  (List any dependencies relevant to this issue, such as libraries or services)

Reviewer Checklist (for Maintainers)

- [ ] Reproducibility verified
- [ ] Environment details confirmed
- [ ] Severity and priority assigned
- [ ] Logs and attachments reviewed
- [ ] Assigned to developer
- [ ] Status updated

Appendix 6: Architecture contribution template

This is the markdown architecture contribution template for use in GitLab and via email.

 # Architecture Contribution

Summary

****Title: ****

Brief and descriptive title of the issue

****Description: ****

Summarize your contribution in a few sentences, outlining what it aims to address or improve.

Details

****Motivation: ****

Explain why this change is necessary or beneficial.

****Affected Component(s):****

List the modules or areas impacted.

****Context: ****

Explain the current state and why this request or issue is relevant.

****Proposed Solution: ****

Describe the proposed solution or change in detail.

****Alternatives Considered: ****

List any alternative solutions that were considered and why they were not chosen.

****Impact: ****

Discuss the potential impact of the proposed change on the project, including benefits and potential drawbacks.

****Backward Compatibility: ****

Will this change break existing functionality? If so, how?

****Migration/Transition Plan (if applicable):****

Describe how the transition from the current to the new architecture will be managed.

****Stakeholders/Reviewers: ****

List key people who should review or be notified about this issue.

****Attachments: ****

Attach any relevant files, diagrams, or screenshots.

Checklist (for maintainers only!)

__Do not fill this part, it is for maintainers only. __

_Use this checklist to ensure your architecture contribution is ready for review and implementation: _

- [ ] ****Motivation and context are clearly described****

**Clearly explain why this change is needed, referencing the underlying problem, opportunity, or user need. Provide background information, relevant history, or links to prior discussions to help reviewers understand the significance and urgency of the proposal. **

- [ ] ****Affected components are identified****

**List all modules, subsystems, APIs, or documentation sections that will be impacted. Specify both direct and indirect dependencies to ensure the scope is well understood and nothing is overlooked. **

- [ ] ****Proposed solution is detailed and justified****

**Describe the solution in depth, including technical approaches, design decisions, and rationale. Justify why this approach was chosen over others, referencing requirements, constraints, or architectural principles where relevant. **

- [ ] ****Alternatives are considered and documented****

**Summarize other solutions or approaches that were evaluated, including their pros and cons. Explain why they were not selected, demonstrating that the chosen solution is the most appropriate for the context. **

- [ ] ****Impact (including benefits and drawbacks) is analysed****

**Discuss all expected effects of the change, both positive and negative. Consider impacts on users, maintainers, performance, scalability, security, and future development. Be transparent about any trade-offs or limitations. **

- [ ] ****Backward compatibility is addressed****

**State whether the change preserves compatibility with existing functionality, APIs, or data. If breaking changes are introduced, detail their nature, justification, and any mitigation strategies for affected users. **

- [ ] **Migration/transition plan is included (if applicable) **

*If the change requires users or maintainers to take action, provide a clear, step-by-step migration or transition plan. Include timelines, deprecation notices, and tools or scripts to assist with the process. *

- [ ] **Stakeholders and reviewers are listed**

*Identify all individuals, teams, or external partners who should be informed, consulted, or involved in the review and approval process. Include maintainers of affected components, domain experts, and key community members. *

- [ ] **Relevant attachments (diagrams, files, screenshots) are provided**

*Attach or link to supporting materials that clarify the proposal, such as architecture diagrams, workflow charts, sample configurations, or screenshots. Ensure all files are up to date and accessible. *

- [ ] **Acceptance criteria for success are defined (e.g., measurable outcomes, target performance, user impact) **

*List specific, testable conditions that must be met for the contribution to be considered complete. Include functional requirements, performance benchmarks, compliance goals, and user experience outcomes as appropriate. *

- [ ] **Required documentation updates are identified**

*Specify all documentation that will need updating as a result of this change, including user guides, API references, changelogs, and onboarding materials. Outline who will be responsible for making these updates and when they will be completed. *

Appendix 7: list of acronyms and abbreviations

Here is a list of the acronyms and abbreviations in this document.

ACRONYM	MEANING
API	Application Programming Interface
CD	Continuous Delivery / Continuous Deployment
CI	Continuous Integration
CVE	Common Vulnerabilities and Exposures
DG CONNECT	Directorate-General for Communications Networks, Content and Technology (European Commission)
EC	European Commission
EU	European Union
FAQ	Frequently Asked Questions
KPI	Key Performance Indicator
NWD	Normal Working Day
OSGB	Open-Source Governance Board
OVH	OVHcloud (French cloud computing company)
PR	Pull Request
PSO	Programme Support Office
REST	Representational State Transfer
SAAS	Software as a Service
SSH	Secure Shell
SSO	Single Sign-On
T1	Task 1
T4	Task 4

Thank you

© European Union 2025

Unless otherwise noted the reuse of this presentation is authorised under the CC BY 4.0 license. For any use or reproduction of elements that are not owned by the EU, permission may need to be sought directly from the respective right holders.