D8.4 Report on open-sourcing activities

agricore

Deliverable Number	D8.4
Lead Beneficiary	AAT
Authors	AAT, IDE
Work package	WP8
Delivery Date	M45
Dissemination Level	Public

www.agricore-project.eu

The Agricore project has received funding from the European Union's Horizon 2020 research and innovation programme under the Grant Agreement No. 816078

Document Information

Project title	Agent-based support tool for the development of agriculture policies
Project acronym	AGRICORE
Project call	H2020-RUR-04-2018-2019
Grant number	816078
Project duration	1.09.2019-31.8.2023 (48 months)

Version History

Version	Description	Organisation	Date
0.1	Definition of ToC	AAT	14-jun-2023
0.2	Definition of content	IDE	23-jun-2023
0.3	First draft	AAT	10-jul-2023
0.4	First revision	IDE	13-jul-2023
0.8	Second full draft	AAT	19-jul-2023
0.9	Final revision and formatting	IDE	20-jul-2023
1.0	Final version	AAT/IDE	21-jul-2023

Disclaimer

All the contributors to this deliverable declare that they:

- Are aware that plagiarism and/or literal utilisation (copy) of materials and texts from other Projects, works and deliverables must be avoided and may be subject to disciplinary actions against the related partners and/or the Project consortium by the EU.
- Confirm that all their individual contributions to this deliverable are genuine and their own work or the work of their teams working in the Project, except where is explicitly indicated otherwise.
- Have followed the required conventions in referencing the thoughts, ideas and texts made outside the Project.

Executive Summary

The AGRICORE project, funded under H2020, aims to develop an agent-based support tool for the development of agriculture policies. The project leverages advancements in modelling approaches and ICT to enhance the capacity to model and simulate the impact of agricultural policies. The core component of the project is the AGRICORE tool, a highly modular and customisable suite that simulates the decision-making processes of individual farms within their unique contexts.

Considering the open-source nature of the project, the Report on Open-Sourcing Activities provides an overview of the AGRICORE project's initiatives in promoting transparency, collaboration, and innovation within the agricultural community. By sharing project outcomes, methodologies, and tools with the public, AGRICORE fosters growth in agricultural research and development. The report emphasises adherence to open-source frameworks and appropriate licensing to ensure accessibility and usability for the project users. This report details the development platform selection, licensing typology, and engagement with external contributors. Additionally, it defines procedures for repository inventory, version control, maintenance, and user support to sustain the project's long-term impact and accessibility.

Overall, AGRICORE's open-sourcing initiatives exemplify a commitment to fostering collaboration, innovation, and inclusivity within the agricultural domain. By promoting community engagement and embracing diverse perspectives, the project advances the collective progress of agricultural research and development.

Abbreviations

Abbreviations	Full Name
EU	European Union
GPL	General Public License
MIT	Massachusetts Institute of Technology
OSS	Open-source software
VCS	Version Control System

List of Figures

List of Tables

Table of Contents

1 Introduction

This report aims to provide an overview of the open-sourcing activities conducted as part of the AGRICORE project. Open sourcing plays a crucial role in promoting transparency, collaboration, and innovation within the agricultural community. By sharing project outcomes, methodologies, and tools with the wider public, we contribute to the growth of agricultural research and development. This report summarises the open-sourcing initiatives undertaken, their impact, and recommendations for future engagement in open-source practices.

By adhering to established open-source frameworks and selecting appropriate licensing models, the AGRICORE project ensures that the open-sourced components and tools are accessible, usable, and modifiable according to the needs of the agricultural community. This commitment to open collaboration also aligns with the project's ethical responsibility to contribute back to society, promoting equitable access to advancements and fostering the exchange of ideas that can drive transformative change.

By making project components and tools openly accessible, the project aims to foster an ecosystem of knowledge-sharing, enabling researchers, farmers, industry professionals, and other stakeholders to leverage and build upon the project's findings. Moreover, the AGRICORE project not only aims to disseminate its findings but also to foster a dynamic ecosystem of collaboration and innovation. The project acknowledges the importance of community engagement and the value of diverse perspectives in advancing agricultural research and development.

This report is structured as follows. In the "Development platform" section, the decisions made during the development process, such as choosing the development platform and preparing its structure, are explained. The "Licenses" section presents a typology of existing licenses and highlights the license selected for the AGRICORE project, highlighting the benefits of the selected one. The "External Contributors" section introduces current and potential stakeholders involved in the AGRICORE developments. Finally, there is a section about defined procedures with four sub-sections, one for each defined procedure: "Repository inventory and documentation", "Version control management", "Maintenance and updates", and "Repository access and user support".

2 Development platform

In this section, the AGRICORE development platform and the importance of the structure that contains the project are described, designed with the main objective of facilitating collaboration between stakeholders, the organisation of resources to establish quality standards and the accessibility of project resources with valuable documentation for their proper use and licensing.

AGRICORE's architecture is based on the principle of decomposition into modules, where each module focuses on a specific functionality of the system. Each module is independent and communicates with other modules through well-defined interfaces. This allows developers to work in isolation on each module, which facilitates collaboration and speeds up the development process.

The main objective of this modular architecture is to allow developments in AGRICORE to be programmed in any programming language without compromising interoperability and compatibility. By defining clear interfaces and communication standards, it is ensured that the modules can interact seamlessly, regardless of the language used for its implementation, which is key in the development of the AGRICORE project since a large number of different languages are used in the development ensuring compatibility through a proper definition of the interfaces.

Agricore's modular architecture presents several significant advantages. First, by dividing the system into modules, the development and debugging of each component is simplified, making it easier to identify and fix bugs. In addition, modularity allows for greater code reuse, as modules can be shared and used in different parts of the project.

Another key advantage is the ability for modules to evolve independently. By defining clear interfaces and communication standards, it is ensured that the modules can interact seamlessly, regardless of the language used for its implementation, which is key in the development of the AGRICORE project since a large number of different languages are used in the development ensuring compatibility through a proper definition of the interfaces.

This provides flexibility to introduce changes and adapt the software as new needs or requirements arise. Figure 1 shows some of the different modules that comprise the AGRICORE application, separated into projects on the chosen platform.

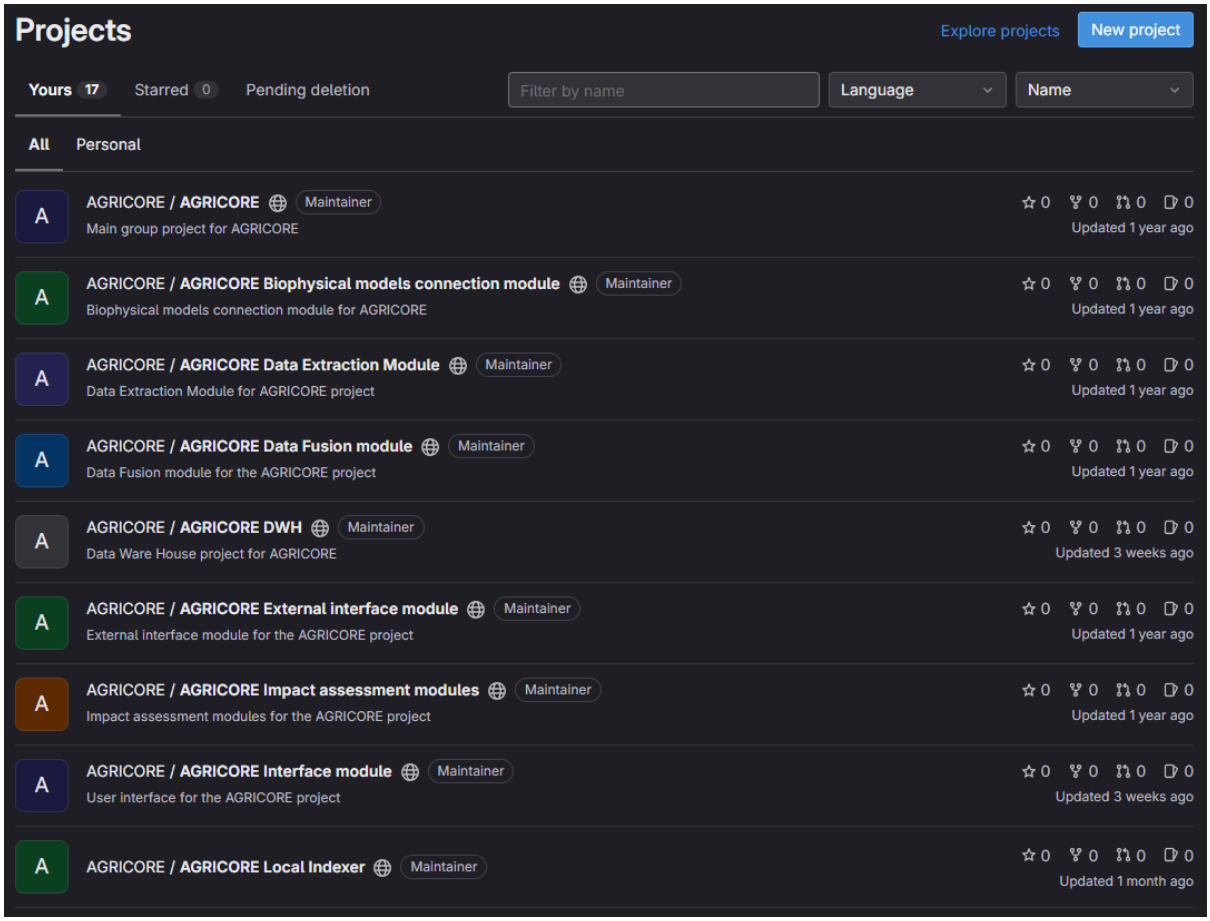

Figure 1. AGRICORE projects.

2.1 Selection of the development platform

Choosing the right platform is crucial to ensure efficient, collaborative development that meets the requirements set by the AGRICORE consortium. To understand the comparison of the different platforms, one must first understand the concept of what a version control system is and its benefits. According to [1]:

"Version control - also known as source control or revision control - is an important software development practice for tracking and managing changes made to code and other files. It is closely related to source code management.

A version control system (VCS) tracks changes to a file or set of files over time. The most common type is a centralized VCS, which uses a server to store all the versions of a file. Developers can check out a file from the server, make changes, and check the file back in. The server then stores the new version of the file.

Version control software facilitates coordination, sharing, and collaboration across the entire software development team. It enables teams to work in distributed and asynchronous environments, manage changes and versions of code and artifacts, and resolve merge conflicts and related anomalies."

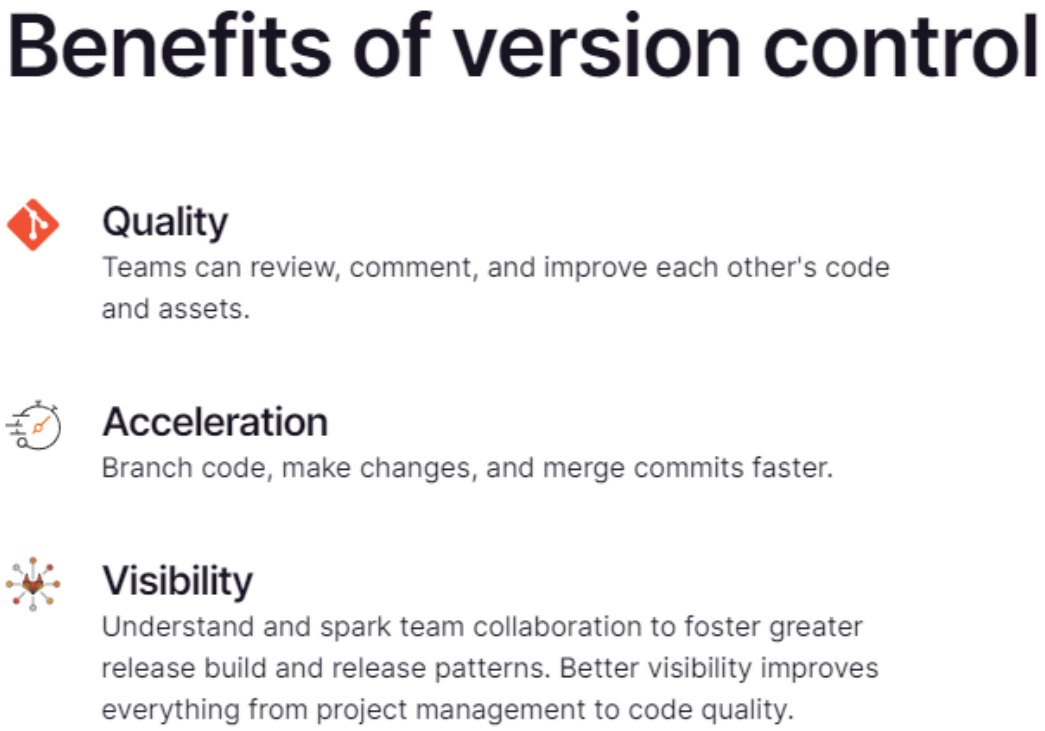

Figure 2. Benefits of version control.

The proposed corporate development platform and version control options for the AGRICORE project were:

- Github: Created in 2008 and acquired by Microsoft in 2018, it is the most widely used collaborative development platform.[1]
- Gitlab: Launched in 2011 under MIT and other open-source licenses.[2]

Both platforms use GIT as VCS[3]. Below is a comparison table with the different features and functionalities offered by the two platforms.

Table 1. Github/Gitlab functionalities and features comparison.

Features	GitHub	GitLab
Number of users	100 million	30 million
Deployment Options	SaaS, Self-Managed	SaaS, Self-Managed
Price	More affordable (Free, $3.67/user/month, $19.25/user/month)	More expensive (Free, $19/user/month, $99/user/month)
Open/Closed Source	Closed	Open

[1] https://github.com/about
[2] https://about.gitlab.com/
[3] https://git-scm.com/

SLA	99.99% uptime guarantee	99.5% uptime guarantee
Continuous Integration	Yes, with GitHub Actions	Yes, with GitLab CI/CD
Auto Pipeline Creation	No	Yes, with Auto DevOps
Integrations	GitHub Apps, OAuth Apps, GitHub Marketplace Apps, Third-Party	Deploy Client Applications, Host GitLab Server, Integrate with Development Cycle, Deep Application Integration
Project Management Tools	Yes, including Issue Tracking, Project Boards, Milestones	Yes, including Issue Tracking, Project Boards, Milestones
Security Features	GitHub offers basic security features.	GitLab offers Static Application Security Testing (SAST), Secret Detection, Code Quality, Dynamic Application Security Testing (DAST), API Security, Fuzz Testing, Dependency Scanning, Container Scanning, and License Compliance.
Code Review Tools	GitHub offers Pull Requests, Diffs, History, Blame, Comments, Review Requests, Reviews, Conflicts Resolution, Permissions & Protected Branches, and Code Review Tools like Codecov, Codacy, Coveralls, and GitColony.	GitLab offers Merge Request, Code Controls, Code Quality Reports, and Analytics.
Web IDE	GitHub offers GitHub Codespaces, a cloud-hosted development environment that lets you customise your project for use with GitHub.	GitLab offers GitLab Web IDE, a development environment that enables users to edit, review, and merge files.
AI Programming Tools	GitHub offers GitHub Copilot, an AI-powered pair programming tool that speeds up the coding process.	GitLab offers GitLab Tabnine, an AI-powered code completion technology that integrates with GitLab repositories.
Personal Use	GitHub is more popular than GitLab for personal use. GitHub offers more storage, more CI/CD minutes per month and lets you work with an unlimited number of contributors.	GitLab offers 5GB of storage, 400 CI/CD minutes/month, and up to 5 users per repository.
Enterprise Use	GitHub Enterprise Cloud offers an enterprise product plan (SaaS) for large businesses and teams. It provides authentication with SAML single sign-on, support for 50,000 minutes of GitHub Actions runtime for CI/CD workflows, and 50GB of storage for shared components and containers.	GitLab offers GitLab Enterprise Edition (EE) and GitLab Ultimate for enterprise use. Both provide advanced security features, DevOps lifecycle management, and project management tools.

Finally, GitLab was chosen as the corporate development platform and VCS. The technical progress of the project is monitored to update and track the architecture of the platform and the documentation associated with the project. In the AGRICORE repository, we can find all the necessary projects for each module that contains the AGRICORE suite. These projects have been created privately to start the development. Once completed and tested, the developments are published in a public repository. There are two key factors that were decisive in the choice of GitLab:

- **Open Source Program:** GitLab launched and still is an open source project. This, together with the work of promoting open-source developments, fits with AGRICORE's vision and

project. In addition, GitLab offers its most complete plan with all its features included for this type of project[4].

- **GitLab Epics:** Epics are a project management feature provided by the GitLab platform. An Epic is a higher-level organisational unit that allows you to group related issues and work items together. It helps in organising and tracking progress on larger, more complex features or initiatives within a project.

Epics provide a way to plan and manage work that spans multiple milestones or iterations. They help teams to break down a large project into smaller, more manageable tasks. Epics are typically used when a project or feature requires collaboration among multiple team members and involves various interconnected issues**.**

- **Define goals and objectives**: Epics allow you to clearly define the overarching goals and objectives of a project or feature. This helps to provide a high-level understanding of the purpose and scope of the work.
- **Group-related issues**: You can associate multiple issues, merge requests, and other work items with an Epic. This grouping helps to provide a unified view of all the work related to a specific feature or initiative.
- **Track progress and milestones**: Epics provide a visual representation of progress, allowing you to track the completion status of individual issues and their associated tasks. You can set milestones and due dates for Epic to ensure timely delivery.
- **Assign ownership and responsibilities**: Epics can be assigned to specific team members or groups, enabling clear ownership and accountability for the overall progress of the work.
- **Monitor discussions and updates**: Epics include a dedicated discussion board where team members can collaborate, ask questions, and provide updates related to Epic. This facilitates effective communication and collaboration among team members.

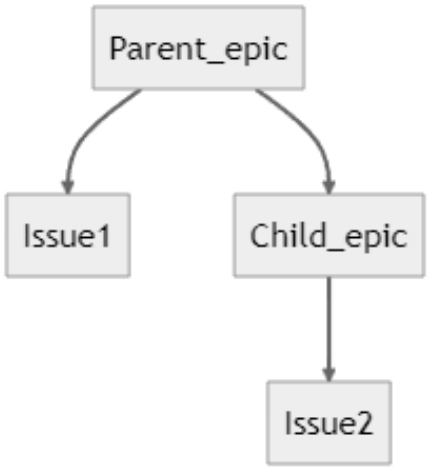

Figure 3. Generic example of how to use Epics to manage and monitor the progress of development together with the use of associated Issues.

More information can be found in the official GitLab link about epics [2].

Figure 4 shows an example of the screen of a parent epic about module 1 ARDIT.

[4] https://about.gitlab.com/solutions/open-source/join/

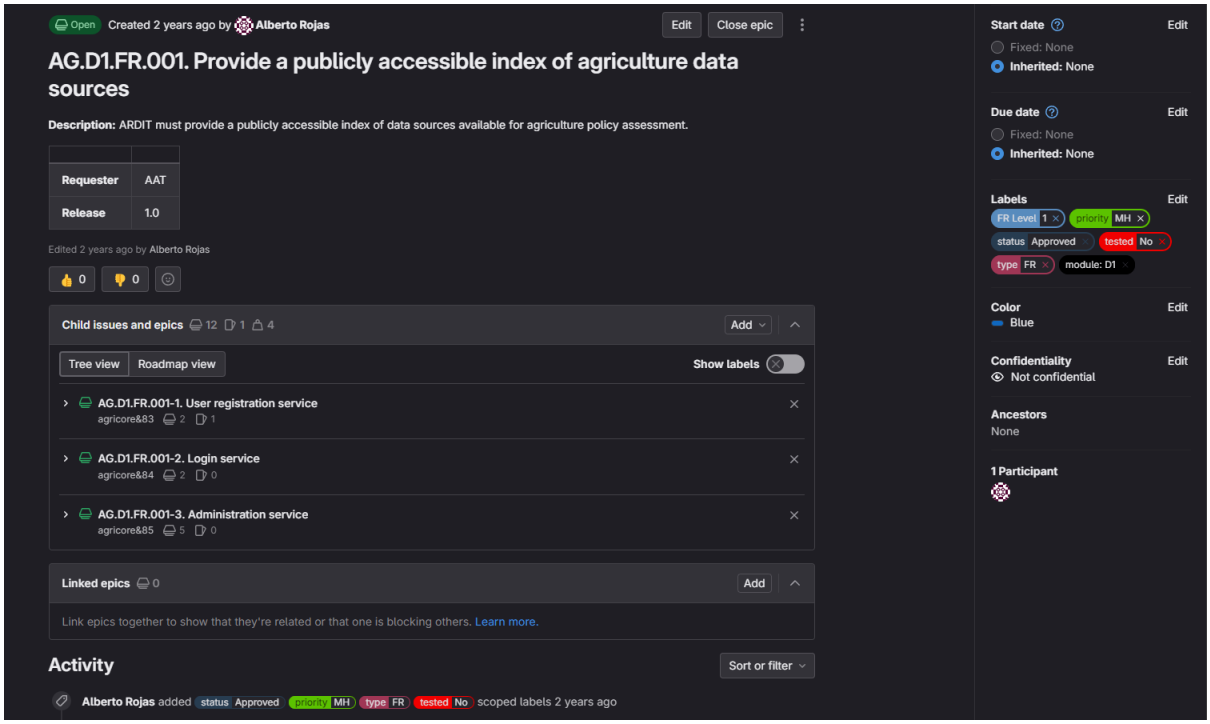

Figure 4. Screen example of a parent epic about module 1 ARDIT.

All the requirements have been created as epics and must contain a number of characteristics that are involved in the analysis, development and evaluation processes, which are defined below:

- **Code**: Unique requirement identifier which describes the module and the type of the requirement. The pattern code for all requirements follows the structure: AG.Dx.<TR>.<ID>.<hierarchy>, for example, AG.D0.FR.001-1-1, where:
    - **AG**: AGRICORE project.
    - **Dx**: Dx module reference, which can be:
        - D0: Global requirement.
        - D1: ARDIT.
        - D2: DWH.
        - D3: Data extraction module.
        - D4: Data fusion module.
        - D5: Synthetic population generator.
        - D6: ABM simulation module.
        - D7: External interface module.
        - D8: Model interaction module.
        - D9: Biophysical models connection module.
        - D10: Impact assessment modules.
        - D11: Policy environment module.
        - D12: AGRICORE interface module.

    - **TR**: Type of requirement.
        - FR: Functional requirement.
        - NFR: Non-functional requirement.
    - **ID**: Incremental id.
    - **Hierarchy**: When a requirement is a child of another, an incremental id is used per layer.
- **Title**: Descriptive title of the requirement.
- **Definition**: Full detailed description of the requirement.
- **Requester**: Partner who has presented the requirement.
- **Type**: Functional or non-functional requirement.
- **Related modules**: Related modules identifiers. Although a requirement is associated with a specific module, a requirement could interact with several modules of the platform at the same time.
- **Release**: The version of the requirement defined. If a requirement changes its scope, the release version number must be modified.
- **Tested**: Test tracking status indicator, which has two possible states:
    - Yes: The requirement has been tested successfully.
    - No: The requirement has not been tested yet.
- **Status**: This characteristic could have one of the following 5 values.
    - Draft.
    - Proposed.
    - Approved.
    - Rejected.
    - Implemented.
- **Priority**: The MoSCoW method is a prioritisation technique used in management, business analysis, project management, and software development to reach a common understanding with stakeholders on the importance they place on the delivery of each requirement; it is also known as MoSCoW prioritisation or MoSCoW analysis. A requirement must have only one of the values that the MoSCoW method defined:
    - Must have (MH): These provide the Minimum Usable SubseT (MUST) of requirements which the project guarantees to deliver.
    - Should have (SH): Important requirement but not vital for delivery in the current delivery timebox. It may be painful to leave out, but the solution is still viable.
    - Could have (CH): Desirable requirement but less important.
    - Won't have (WH): Requirements that have been agreed by stakeholders as the least critical.

All the requirements defined and integrated into the GitLab platform will be included in Annex A.

2.2 Preparation

In this section, the steps taken to prepare GitLab and adapt it to the specific needs of the AGRICORE project are described.

The initial configuration of the project is to create the structure in a suitable environment. In this case, the entire infrastructure is done in the cloud directly on the GitLab page and the tools it offers us. The most appropriate security options for the project, user authentication and access permissions have been configured to ensure adequate protection of the code and data.

The organisational structure of projects and repositories in GitLab is essential to facilitate collaboration and efficient source code management. A clear hierarchical structure has been established, creating projects and sub-projects for each component of the AGRICORE project. A general group called AGRICORE has been created containing all the generated projects and modules. These projects are shown in Figure 1.

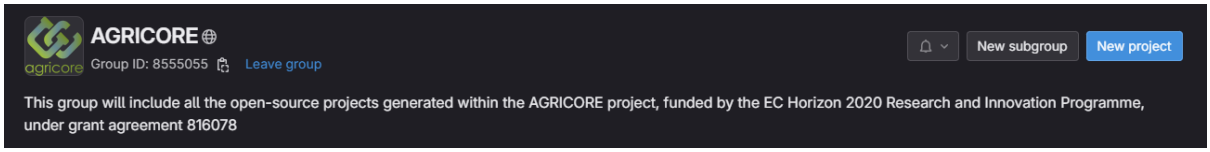

Figure 5. AGRICORE group created in the GitLab platform.

Furthermore, inside the above project, all requirements have been established as Epics. Figure 6 shows a list of Epics defined in the general AGRICORE project according to instructions explained in the previous section.

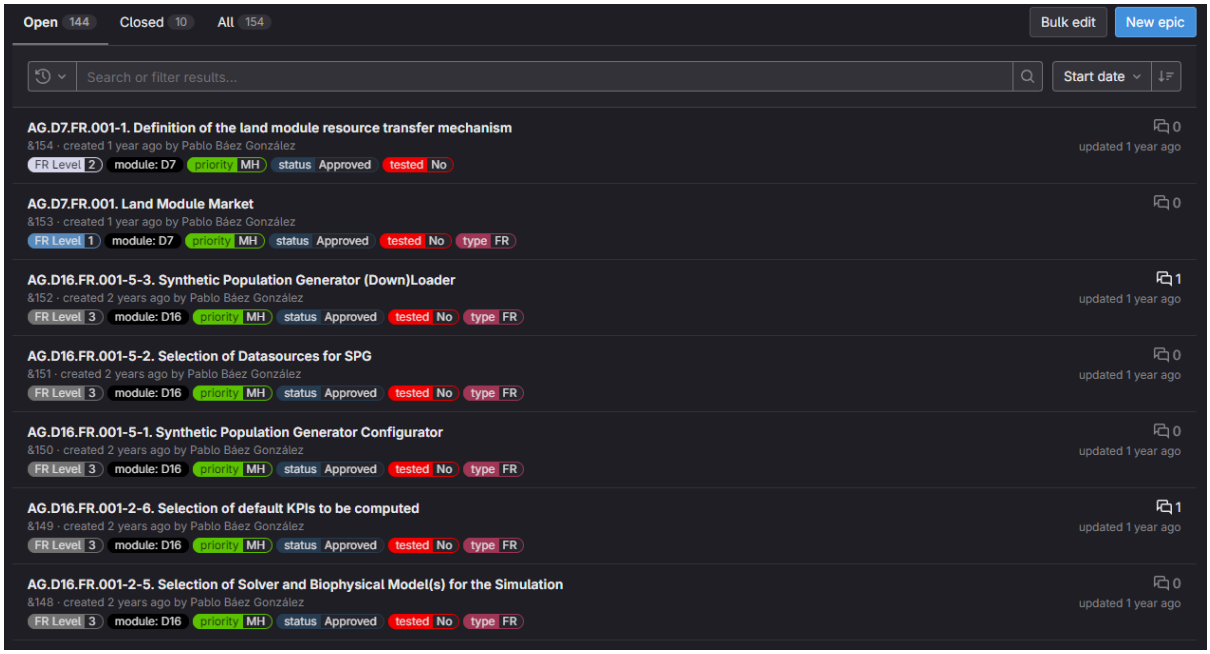

Figure 6. List of epics defined in the general AGRICORE project.

2.3 Management

User and role management in GitLab is critical to ensure that team members have adequate access and control over repositories and functionality. Users have been configured and managed, assigning appropriate roles and permissions according to their responsibilities and requirements. This ensures that each team member has the necessary access to perform their assigned tasks and that information security and confidentiality are maintained.

The following are the roles available in Gitlab:

- Guest (This role applies to private and internal projects only.)
- Reporter
- Developer
- Maintainer
- Owner
- Minimal Access (available for the top-level group only)

A user assigned the Guest role has the least permissions, and the Owner has the most. For more details on user actions, see the section "Repository access and user support". Currently, project leaders and the AGRICORE consortium oversee and coordinate project activities, and clear roles and responsibilities for each team member have been defined, ensuring an equal distribution of tasks and effective collaboration.

Once the project finishes, the AGRICORE project will adopt a future governance approach based on transparency, participation and consensual decision-making. It aims to guarantee long-term sustainability and promote the active collaboration of all team members. The project strategy is regularly reviewed and adapted to emerging changes to stay aligned with project needs and technology trends. The idea is that the main developers will remain as owners of their modules, and new maintainers will be identified through the adoption of the developments. For this reason, we value the contribution of the open-source community and encourage collaboration to improve the project in the long term. Thus, some mechanisms have been established for decision-making and the assignment of responsibilities.

3 Licenses

AGRICORE is strongly committed to open sourcing and open collaboration; therefore, an intensive study has been carried out as part of Task 8.1 by the different partners involved in the development of one or several customised modules. AXIA has been in charge of providing guidance on open-sourcing and licensing in the different IPR workshops, especially the second workshop, providing a concise view of the use and redistribution of OSS and the type of licenses.

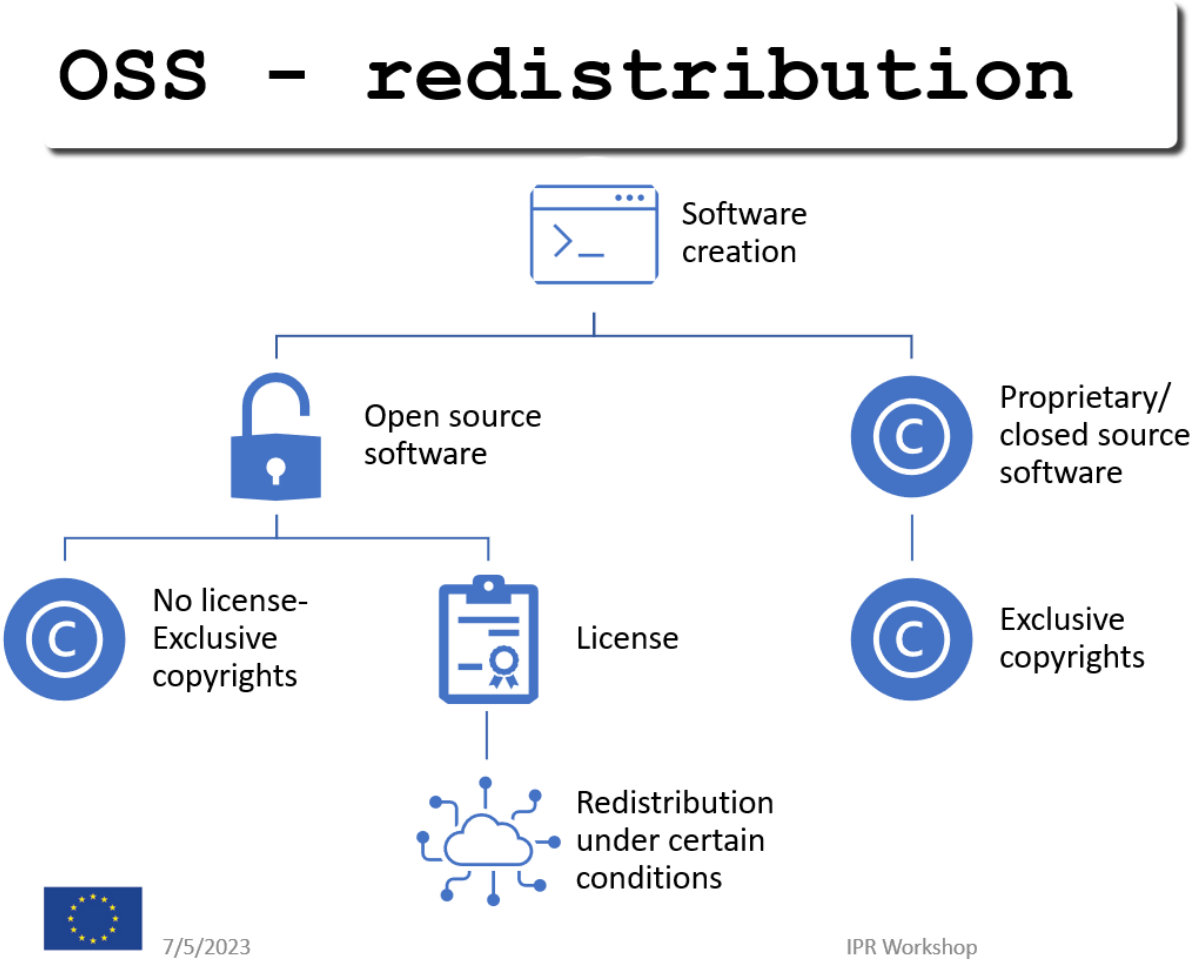

Figure 7. Scheme of the open-source software extracted from the second project's workshop.

Following is an explanation of the categories, criteria and types of licenses to see the different licensing possibilities and, finally, the license chosen for the AGRICORE project. There are two main categories of open-source licenses.

Table 2. Categories of open-source licenses.

Copyleft licenses	Permissive licenses
• Granting the right to use, modify, and share creative works without the permission of the copyright holder. • Any derivative work must maintain the reciprocity of the obligation.	• Granting the right to use, modify, and share creative works without the permission of the copyright holder. • Permitting proprietary derivative works.

Their basic difference is in how they allow/constrain any derivative works to be used, published and distributed. Therefore, the choice of license for open-source software is affected by the licenses of any existing dependencies and will same-wise affect the licenses of any future work based on this open software. Three main examples of open-source licenses used in the software are presented in the table below.

Table 3. Examples of open-source licenses.

Name	Category	Summary of description
MIT License	Permissive	Do whatever you want with this software if you add a copy of the original MIT license and copyright notice to it.
Apache License	Permissive	Freely use, modify, and distribute. Explicitly grants non-exclusive rights to users that can be applied to both copyrights and patents. Must include a copy of the license and add modification notices to all the files that you modify. The unmodified parts of the software must retain the Apache License. The modified parts can be released under different licenses.
GNU General Public License (GPL)	Copyleft	Not allowed to claim patents or copyright on the software. Moreover, you are obligated to display a copyright notice, disclaimer of warranty, intact GPL notices, and a copy of the GPL. Not allowed to change the license or introduce additional terms and conditions. Reciprocity obligation, which means you are obligated to release the source code and all the rights to modify and distribute the entire code.

Below, several popular open-source licenses are presented, highlighting their main aspects for the sake of illustrative examples.

Table 4. GNU General Public Licence.

GNU General Public Licence	GPL V3
Type	Strong copyleft
Main aspects	• It is the most well-known copyleft license. • Contains one of the strongest and most uncompromising copyleft clauses. • Reciprocity obligation: modified or not, any code redistribution will have to maintain this license. • Integration: GPL can not be effectively combined in a single program with any non-GPL software.
Advantages	• Good choice for software applications that are self-standing and that their development is desired to be distributable. • Strong copyleft clause supports the maintenance of the GPL-released code as open source.
Barriers	• Limited applicability in the case of contribution to the work of a third party. • Not recommended for libraries. • Can possibly prohibit the integration of any such work into a commercial proprietary context.

Table 5. GNU Affero General Public Licence.

GNU Affero General Public Licence	AGPL V3
Type	Strong copyleft
Main aspects	• Created by modification of GPL, aiming in particular at applications of software as a service (SaaS) distribution. • Reciprocity obligation: modified or not, any code redistribution will have to maintain this license.

	• Integration: AGPL cannot be effectively combined in a single program with any non-AGPL software.
Advantages	• Good choice for SaaS applications that are self-standing and that their development is desired to be distributable. • Strong copyleft clause supports the maintenance of the AGPL-released software and its code as open source.
Barriers	• Limited applicability in the case of contribution to the work of a third party. • Not suitable/Disputable for libraries. • Can possibly prohibit the integration of any such work into a commercial proprietary context.

Table 6. Apache Licence.

Apache Licence	APLv2
Type	Permissive
Main aspects	• It can grant a license for copyrights but also for patents. • Any unmodified part of the source code needs to remain under the same license. All modified codes can be issued under any license desired (including proprietary). All changes need to be listed. • Integration: Integration of APL-licensed code to a larger software (/code) can be done by any choice of downstream distribution license.
Advantages	• The permissive character of the license supports code integration to third-party works. • Cross-compatibility issues are highly unlikely to arise when pursuing the integration of APL-licensed code with copyleft third-party components.
Barriers	• The patent-related clause may affect the issuer's patenting strategy, in particular patents related to software operation. • Not as easily readable as other permissive licenses.

Table 7. Massachusetts Institute of Technology Licence.

Massachusetts Institute of Technology Licence	MIT Licence
Type	Permissive
Main aspects	• The most well-known permissive license due to its simplicity and permissiveness. • Any unmodified part of the source code needs to remain under the same license. All modified codes can be issued under any license desired (including proprietary). • Integration: Integration of MIT-licensed code to a larger software (/code) can be done by any choice of downstream distribution license.
Advantages	• The permissive character of the license supports code integration to third-party works. • Very short and very simple. • Cross-compatibility issues are highly unlikely to arise when pursuing the integration of MIT-licensed code with copyleft third-party components.
Barriers	• Not including express patent license. Not recommended for software that is patent-dependent.

The agreed license by all the partners of the AGRICORE consortium after the study and dissemination of the options above is **GNU Affero General Public License (GNU AGPLv3).** The benefits and advantages provided by the license are the ones that best adapt to the vision of the OSS project.

- Source code protection: The GNU AGPLv3 license is a copyleft license that ensures that any modifications or improvements made to the source code are kept open and accessible to the community. This protects the project from unauthorised appropriation and ensures that any contributions or improvements benefit the community at large.
- Extended copyleft: The GNU AGPLv3 license extends the copyleft conditions even to applications that interact with the software over a network. This means that if our software is used in a server environment, modifications made in that environment must also be made available as open source.
- Enhanced collaboration: By adopting a strong copyleft license such as GNU AGPLv3, we encourage collaboration and contribution from the developer community. This can result in increased participation, code reviews and project improvements, which can ultimately accelerate development and improve software quality.
- Compliance with the principles of the free software movement: The choice of the GNU AGPLv3 license aligns with the principles and values of the free software movement. This shows our commitment to user freedom, transparency and ethics in software development.
- Legal protection: The GNU AGPLv3 license clearly states the rights and responsibilities of users and contributors. It provides solid legal protection for the project and its contributors, avoiding conflicts and misunderstandings about the use and distribution of the software.

Further information on the license can be found in [3].

In the AGRICORE project, special attention has been paid to ensure compliance with the licenses and the integrity of the software used in the different modules that comprise AGRICORE, guaranteeing compatibility with free software and the philosophy of the project.

4 External contributors

AGRICORE recognises the significance of engaging a diverse range of stakeholders, including researchers, industry experts, technology enthusiasts, and other relevant individuals or organisations. Effective coordination with external contributors fosters collaboration, knowledge exchange, and innovation, all of which are essential to drive the project's success and maximise its impact on the agricultural domain.

Facilitating resource sharing and providing collaboration tools can foster collaboration among external contributors. Granting access to project resources, tools, and data empowers contributors to actively contribute their expertise and accelerate progress.

By actively engaging with external stakeholders, the project can access diverse expertise, resources, and perspectives, thereby enhancing the project's outcomes and impact on the agricultural domain. Coordinating with external contributors promotes collaboration, innovation, and inclusivity and ensures that the project remains relevant and responsive to the needs of the wider agricultural community. In this line, important collaborations have been made by external contributors, highlighting the feedback provided by the External Advisory Board about the ARDIT tool and the AGRICORE tool interface and the feedback provided by policymakers through the meetings and workshops conducted by use case leaders. Indeed, a general workshop on contributing to AGRICORE software developments has to be organised, which all interested parties are welcome to attend. The purpose of this workshop is to provide an in-depth explanation of the AGRICORE architecture and how its platform and modules function. The main objective is to generate interest among developers to contribute to software development and encourage the use of the created tools in various systems and applications, as well as the adoption of the developed software tools.

5 Definition of procedures

The purpose of this section is to provide a comprehensive set of procedures for maintaining the repository after the completion of the AGRICORE project. These procedures aim to ensure the long-term integrity, accessibility, and usability of the repository, thereby safeguarding the valuable agricultural data, code and information accumulated throughout the project. By implementing these maintenance procedures, we can guarantee the continued availability of the repository's resources and support ongoing research and development efforts.

To accomplish that, the following procedures have been established:

- Repository inventory and documentation
- Version control management
- Maintenance and updates
- Repository access and user support

5.1 Repository inventory and documentation

Maintenance procedures have been established to conduct a thorough inventory of the project's repository, encompassing all relevant resources, tools, datasets, documentation, and associated metadata. This inventory is accompanied by comprehensive documentation that outlines the purpose, functionality, dependencies, and usage instructions for each component within the repository. The following table shows all the projects created so far in the AGRICORE group.

Table 8. All the projects within the AGRICORE group.

Project	Description
AGRICORE	Main group project for AGRICORE
AGRICORE Biophysical models connection module	Biophysical models connection module for AGRICORE
AGRICORE Data Fusion module	Data Fusion module for the AGRICORE project
AGRICORE DWH	Data Ware House project for AGRICORE
AGRICORE Impact assessment modules	Impact assessment modules for the AGRICORE project
AGRICORE External interface module	External interface module for the AGRICORE project
AGRICORE Interface module	User interface for the AGRICORE project
AGRICORE Model interaction modules	Model interaction modules for the AGRICORE project
AGRICORE Module Communication	D6.2 development repository
AGRICORE Policy environment module	Policy environment module for the AGRICORE project
AGRICORE Semantic Services Module	Semantic services module
AGRICORE simulation engine	ABM Simulation Engine for the AGRICORE project
AGRICORE Synthetic population generator	Synthetic population generator for the AGRICORE project
ARDIT	Agricultural Research Data Index Tool

The projects have a fixed structure and project files, which depending on the repository, may vary. The structure of the projects includes:

- License file
- README.md with instructions and annotations on how to build the project and access users to the application or properties of the repository itself.
- Code files and folders.

5.2 Version control management

Gitlab is crucial for effective repository maintenance. The installed and well-managed by the AGRICORE organisation allows for tracking and managing changes to the repository's components over time. With its Git-based repository, GitLab enables asset version control, feedback loops, and powerful branching patterns to help developers solve problems and ship value.

GitLab allows us to have complete control over the history of changes and facilitates collaboration between developers. We can also create branches according to the main or secondary needs and navigate between them and the different points where the application is located. Branches are versions of a project's working tree. As the AGRICORE project grows, the team may want to create more branches, preferably by following branch naming patterns.

The best branch naming conventions include the following suggestions:

- **Start the branch name with a Group word.** The group word can be anything to match your workflow: bug, fix, refactor, feature etc. By looking at the branch name, you can understand what this Git branch is about and its purpose.
- **Use Unique ID in branch names.** You can use the issue tracker ID in your branch name, for example, the Jira task number.
- **Use Hyphen or Slash as Separators.** There are two main advantages of using a separator in the branch name: i) it increases the readability and helps to avoid confusion, and ii) it makes it easier to manage, especially if you are dealing with many branches.
- **Avoid using numbers only.** It only means more confusion and risk of mistakes, especially during merging with other git branches.
- **Avoid long descriptive names for long-lived branches.** The essential quality of a branch name is that it should be precise and informative. The name should be short and explain the purpose of the branch.

Each branch represents a set of changes, which allows development work to be done in parallel. Development work in one branch does not affect another branch.

When changes are made to the source code, GitLab is used to record commits and changes in the repository. Each commit includes a descriptive message explaining the purpose of the change made. This allows us to maintain a complete and detailed history of all changes made to the Agricore project, which facilitates review and version management.

In addition, GitLab provides Merge Requests functionality that can be used to request code reviews and to merge secondary branches with the main branch. This allows managers to have a formal process for review and approval of changes made before they are integrated into the main code base. Code review is a fundamental part of version control management to ensure the quality and consistency of the source code.

5.3 Maintenance and updates

Regular maintenance activities are essential to keep the repository functional and up-to-date. Procedures have been defined to assess the need for updates, plan and implement necessary changes, and communicate updates to repository users. Regular maintenance schedules and procedures for addressing critical updates or security patches have been established.

The maintainer role in GitLab plays an essential role in the ongoing management of the AGRICORE project. Maintainer's main function is to monitor and ensure the proper functioning of the

software, as well as to perform maintenance and upgrade tasks as needed. The maintainer is responsible for identifying and fixing problems, responding to user and contributor requests, and ensuring that updates are released properly.

The maintainer should regularly review several common issues to ensure software stability and performance. Some of the common issues that should be checked include critical bugs, security vulnerabilities, incompatibilities, enhancements and functional updates.

The periodicity of maintenance and updates may vary depending on the needs and resources of the project. However, a regular quarterly schedule has been established for performing maintenance tasks and releasing updates. This ensures that the software is kept up to date and that problems are addressed in a timely manner.

A procedure for publishing updates has been established, which involves:

- Verification of the modifications made
- Performing extensive testing
- Documentation of changes
- Communicating with users and contributors

In addition, a process has been established to receive and respond to comments and bug reports after the release of updates, ensuring continuous improvement and adequate attention to users and collaborators.

5.4 Repository access and user support

In order to access the repository, permission should be granted by the project owner or project maintainer via Gitlab. Depending on the permission assigned, the user will be able to perform a set of actions.

A **Guest** user can read content, can create issues, and can comment on tickets but cannot read or write to the repository. The Guest is able to view Wiki pages related to the project as well. When the pipelines for the project are public, the Guest permission role is able to view the list of jobs and review a log of completed jobs. Primarily, a Guest role would be granted to someone who is a non-active member of the project.

The **Reporter** permission role is slightly above the Guest role. Those with Reporter permission are able to work with the issue tracker while also locking issue threads. This role is able to view the error tracking list for the project. A Reporter can create snippets of code, view statistics related to the project, and see the list of merge requests related to the project. The reporter permission role should be assigned to someone if they need to obtain more insights about the project than someone with the Guest role while also needing to track issues.

The **Developer** permission role has significantly more capabilities to perform actions than the Reporter role. Those with Developer permission are able to clone, develop, submit, and push code. Select the Developer role for someone who needs to be able to actively contribute to the project without needing administrative capabilities. Some of the specific actions that the Developer permission role is able to take include:

- Create and approve merge requests.
- Create, edit and delete releases.
- Create new branches.
- Apply code change suggestions.

- Create, edit, and delete project milestones.

The **Maintainer** permission role only ranks below the Owner role in terms of the number of actions available. In the Maintainer role, the member manages the overall project, as well as managing the members of the project. The Maintainer role can take all the actions allowed in the Developer role, as well as actions including:

- Manage GitLab Pages domains and certificates.
- Manage on-call schedules and escalation policies.
- Manage merge approval rules (project settings).
- Delete a package, Manage Error Tracking.
- Add deploy keys.
- Add new team members.
- Rename project.
- Share (invite) projects with groups.

Someone with the **Owner** permission role could be held liable for any content that is part of the project. Even if someone else created the objectionable or dangerous code and content, the member with the Owner permission role is held liable, as an Owner role should know everything about the project's content. Someone with the Owner role is able to set and grant permissions for all other members. Additionally, with the Owner role, the member can delete or migrate projects, delete issues, disable any email notification settings, and change the project's visibility status. These are actions that someone with the Maintainer role would not be able to do. In simple terms, someone with the Owner permission role is able to control any and all actions and aspects of a project, including removing the project. The complete list of permissions available for each role can be found in [4].

All the users, including Guests, are allowed to create issues on Gitlab pages. Thus, future users of the tool could notify the deficiencies in the modules to the maintainers, improve them and solve doubts. There are many ways to create an issue. GitLab provides a detailed guide for any of the preferred ways [5].

6 Conclusions

In conclusion, open-sourcing activities for the AGRICORE project highlight the significant contributions and achievements in promoting open-source principles and collaboration within the agricultural domain. By embracing open-source practices, the project has fostered transparency, innovation, and knowledge sharing among stakeholders, driving impactful advancements in agricultural technologies, data analysis, and decision-making processes.

This showcases the efforts made to open-source various components of the AGRICORE project, including different modules and applications developed, algorithms, datasets, and documentation. Through the release of these resources under open-source licenses, the project has empowered developers, researchers, and industry experts to build upon and improve the project's outcomes, creating an ecosystem of collaboration and collective problem-solving.

7 References

1. GitLab, What is version control? GitLab, 2023. [Online]. Available: https://about.gitlab.com/topics/version-control/

   Referenced at: *1*

2. GitLab, Gitlab Epics. GitLab. [Online]. Available: https://docs.gitlab.com/ee/user/group/epics/

   Referenced at: *1*

3. GNU AFFERO GENERAL PUBLIC LICENSE. [Online]. Available: https://www.gnu.org/licenses/agpl-3.0.en.html

   Referenced at: *1*

4. Permissions and roles. [Online]. Available: https://docs.gitlab.com/ee/user/permissions.html

   Referenced at: *1*

5. Gitlab Create Issues. [Online]. Available: https://docs.gitlab.com/ee/user/project/issues/create_issues.html

   Referenced at: *1*

For preparing this report, the following deliverables have been taken into consideration:

Table 9. Deliverables used to prepare the current report.

Deliverable Number	Deliverable Title	Lead beneficiary	Type	Dissemination Level	Due date
D4.1	AGRICORE requirements and project management platform	AAT	Report	Public	M12
D8.1	Validated design for the AGRICORE interface	AXIA	Report	Public	M18

8 Annex A: Requirements mapped in the GitLab platform

In this annexe, all the requirements registered in GitLab are recovered and grouped by module. Following the methodology explained in Section 2.1, all these requirements have been introduced in the Gitlab Repository of the project as Epics and Issues of the general group "AGRICORE".

Once logged in GitLab, the group AGRICORE has the EPIC button on the left side, as it can be shown in the image below. In this group, all the modules defined are defined for future developments.

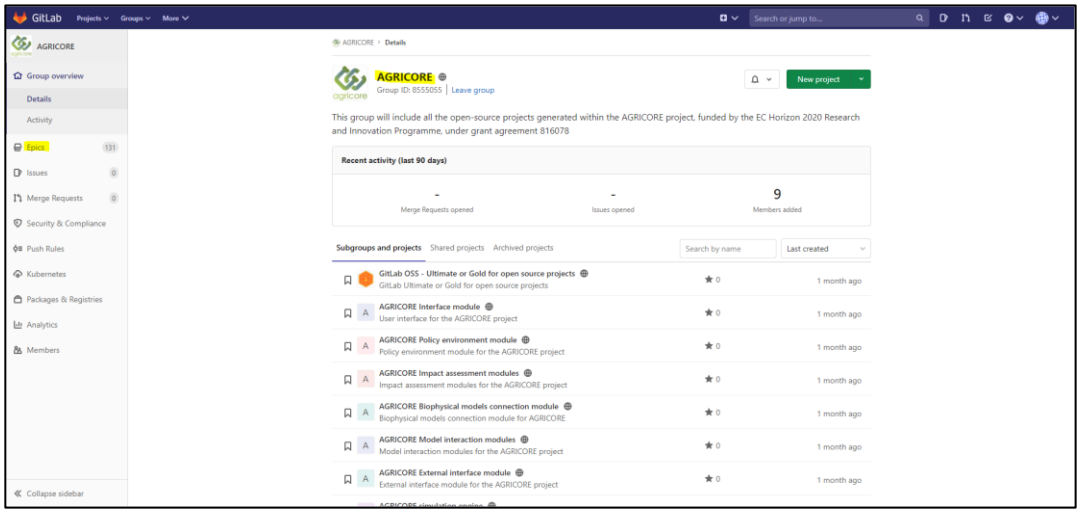

Figure 8. AGRICORE group: EPICS declaration.

After accessing the EPIC section, the next screen is shown, where all requirements can be seen as a list and also can be filtered according to the need of visualisation by clicking on the labels declared and also including further cumulative filters in the bar. The next image shows an example of a filter to show the ARDIT requirements, filtered only by Functional Requirements (FR)

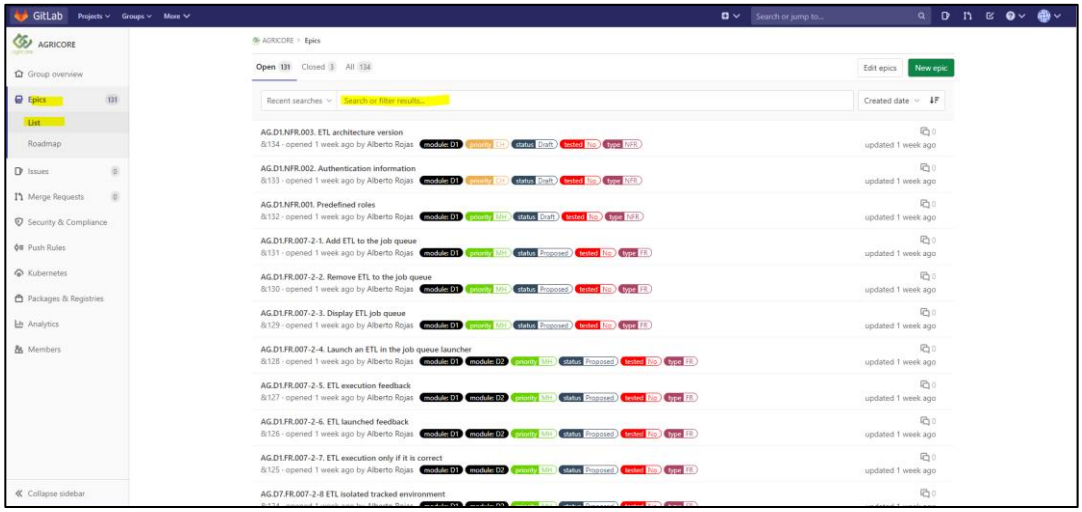

Figure 9. Example of filtering requirements (1/2).

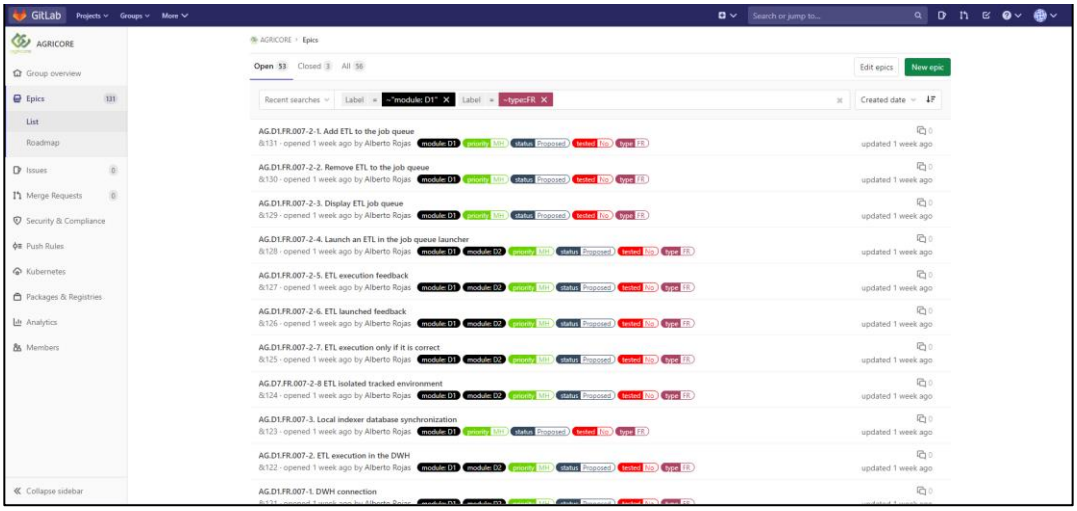

Figure 10 Example of filtering requirements (2/2).

The next images show how all the requirements are seen in the GitLab group.

8.1 D0: General requirements

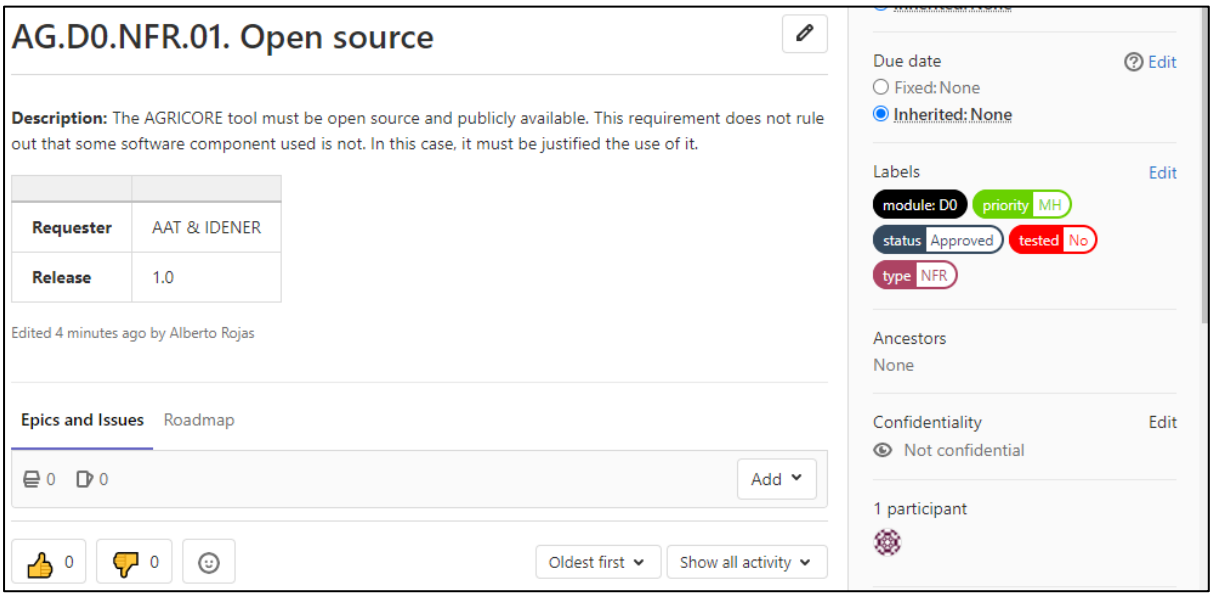

Figure 11. AG.D0.NFR.001. Open source.

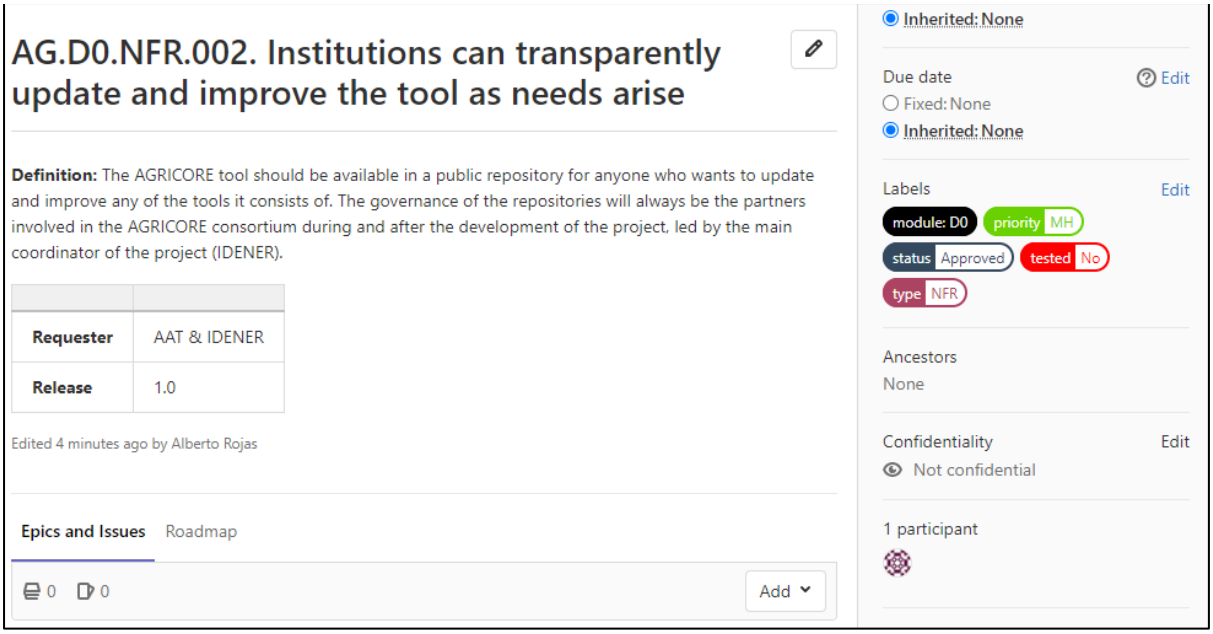

Figure 12. AG.D0.NFR.002. Institutions can transparently update and improve the tool as needs arise.

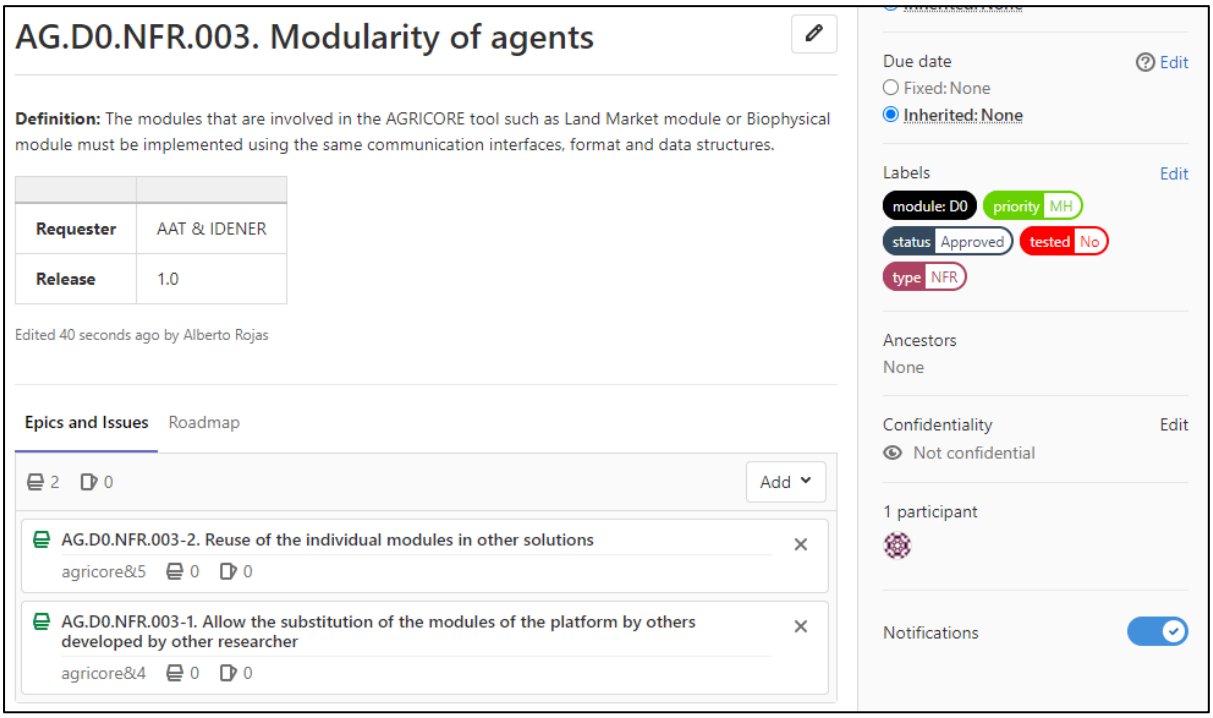

Figure 13. AG.D0.NFR.003. Modularity of agents.

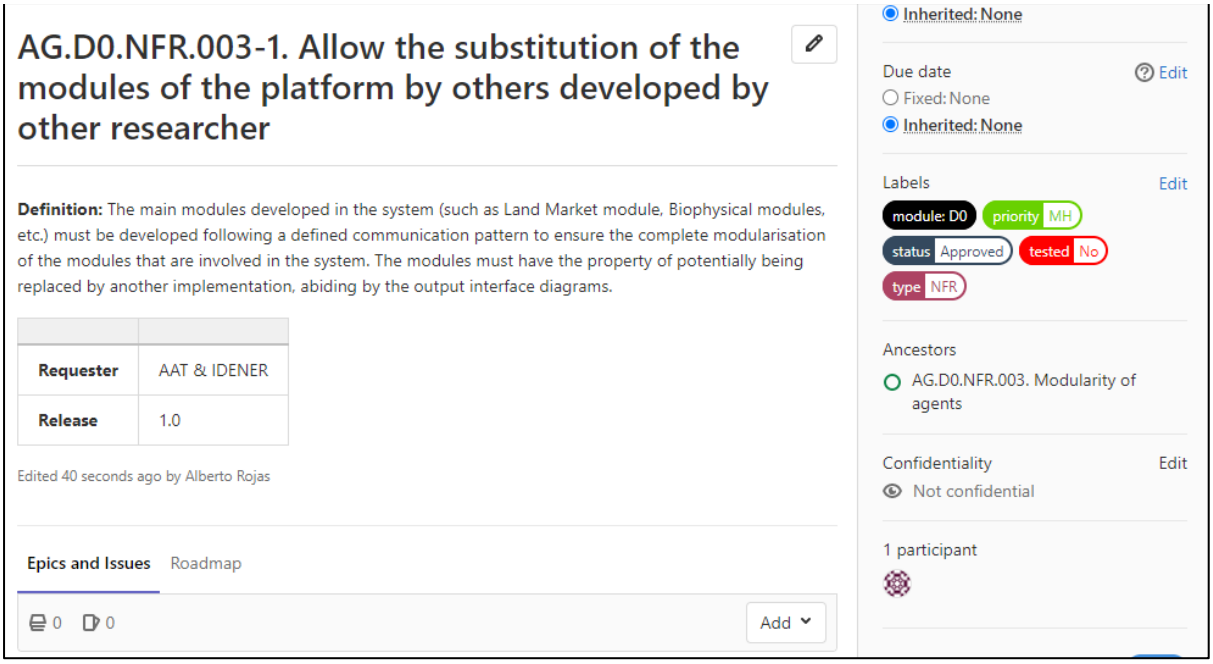

Figure 14. AG.D0.NFR.003-1. Allow the substitution of the modules of the platform by others developed by other researcher.

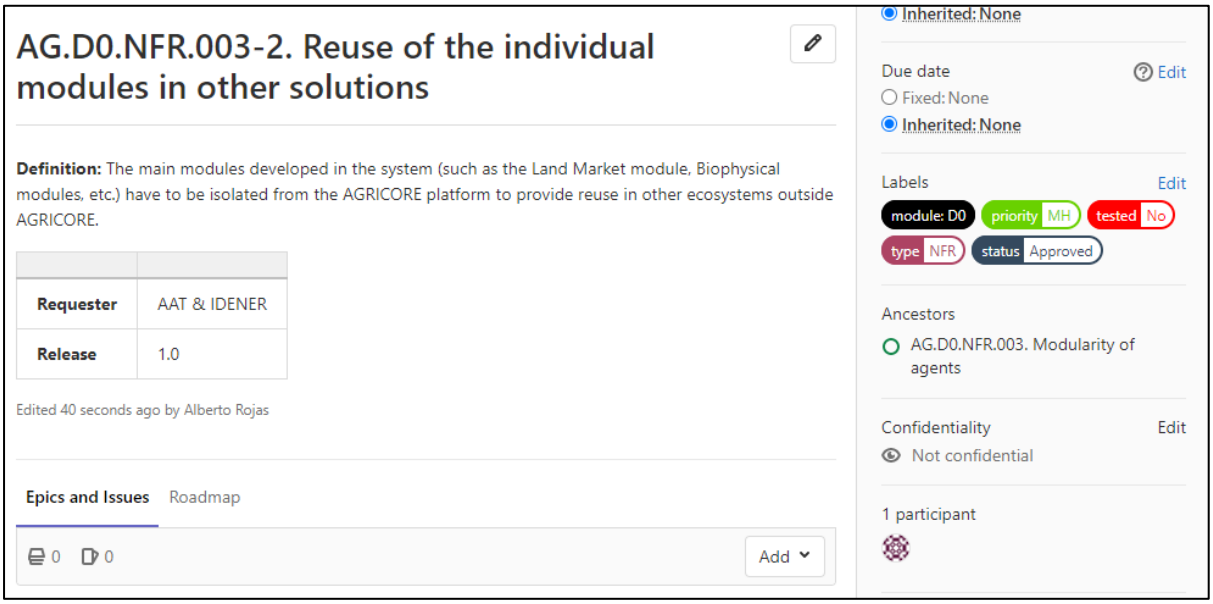

Figure 15. AG.D0.NFR.003-2. Reuse of the individual modules in other solutions.

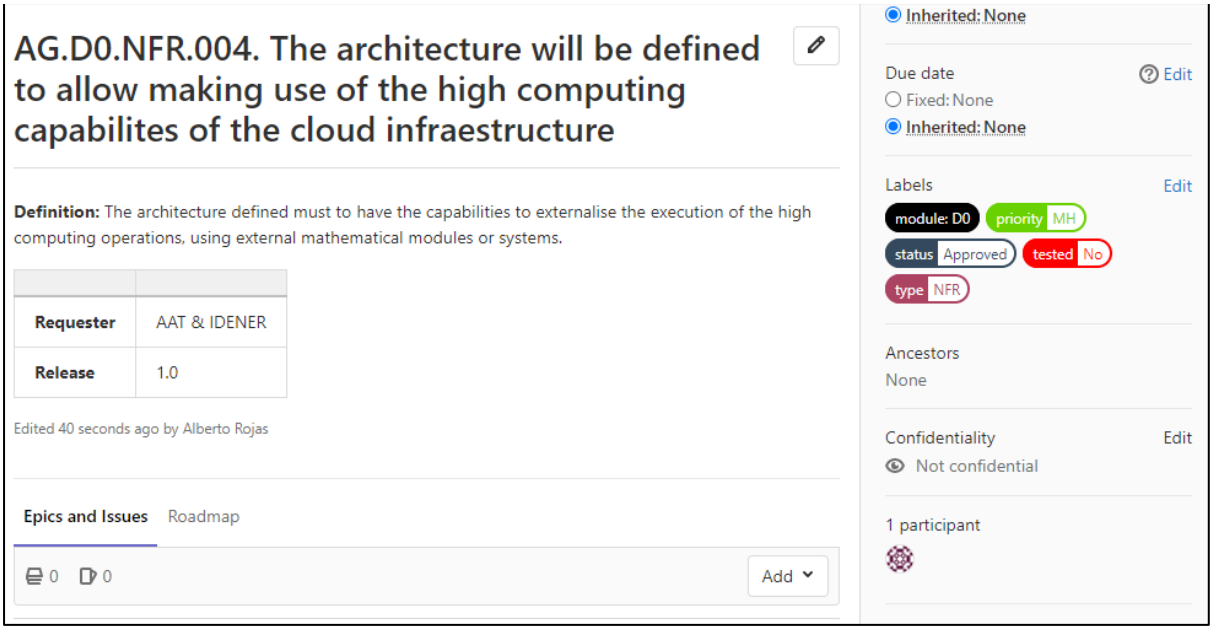

Figure 16. AG.D0.NFR.004. The architecture will be defined to allow making use of the high computing capabilities of the cloud infrastructure.

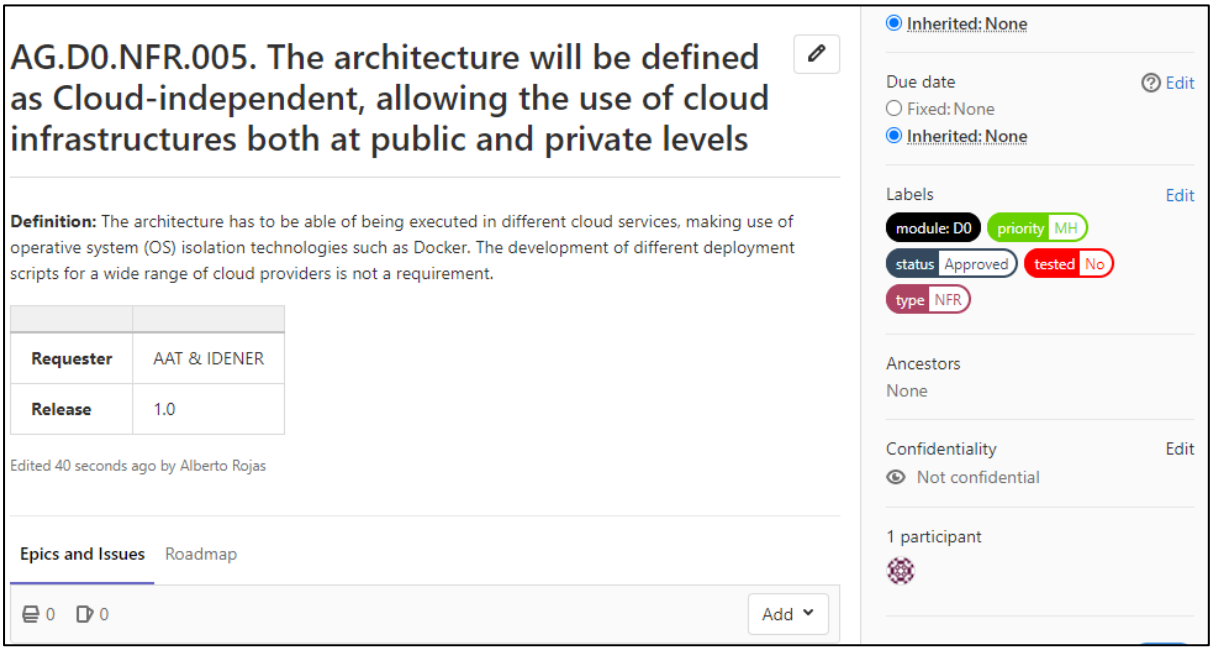

Figure 17. AG.D0.NFR.005. The architecture will be defined as Cloud-independent, allowing the use of cloud infrastructures both at public and private levels.

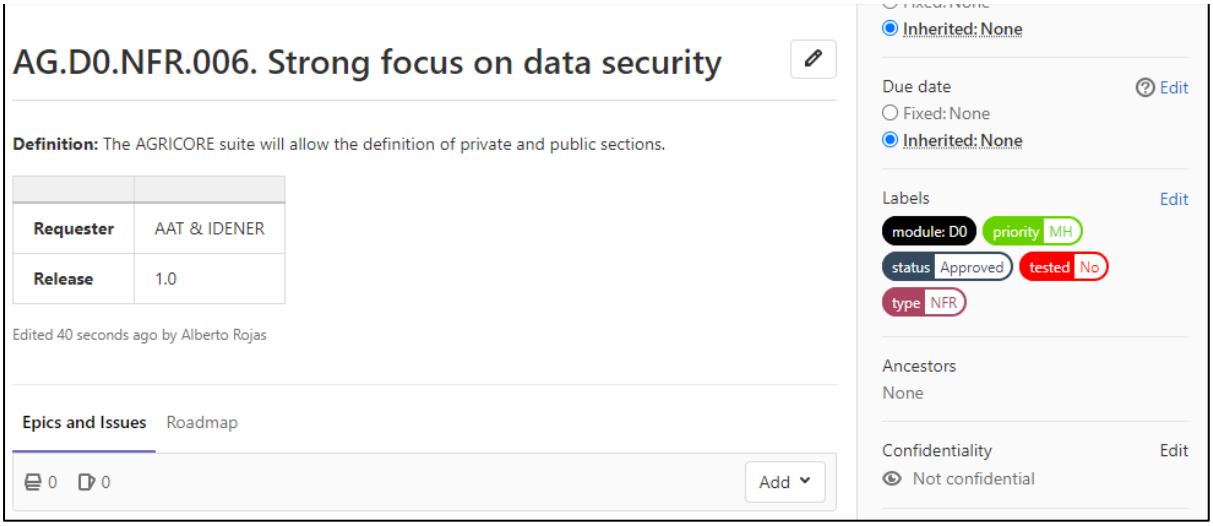

Figure 18. AG.D0.NFR.006. Strong focus on data security.

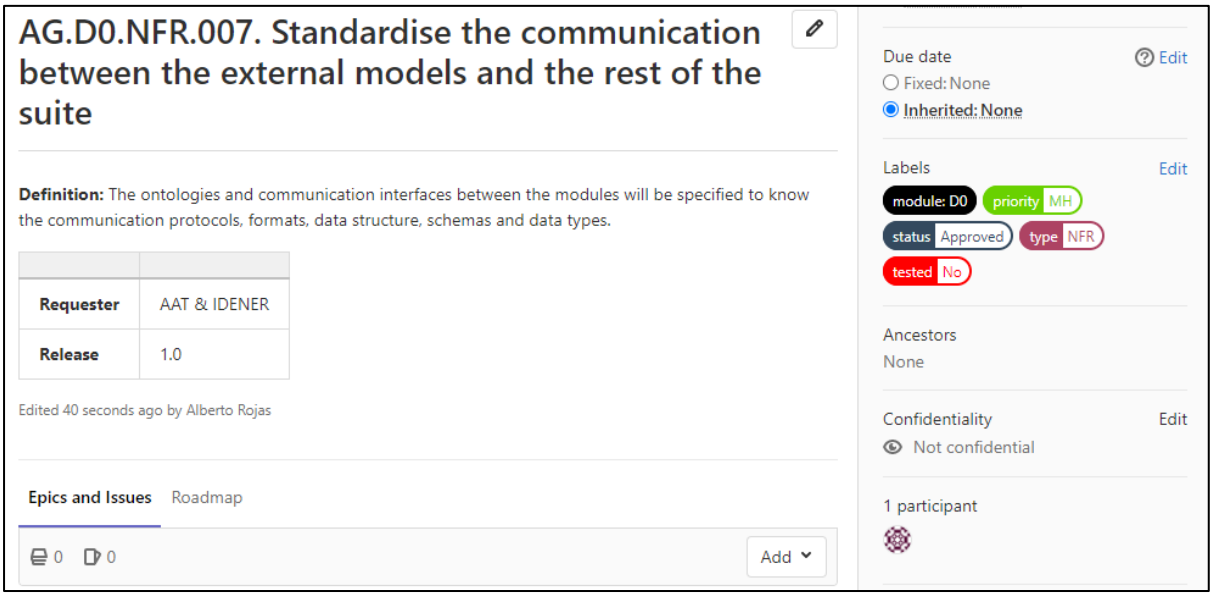

Figure 19. AG.D0.NFR.007. Standardise the communication between the external models and the rest of the suite.

8.2 D1: ARDIT

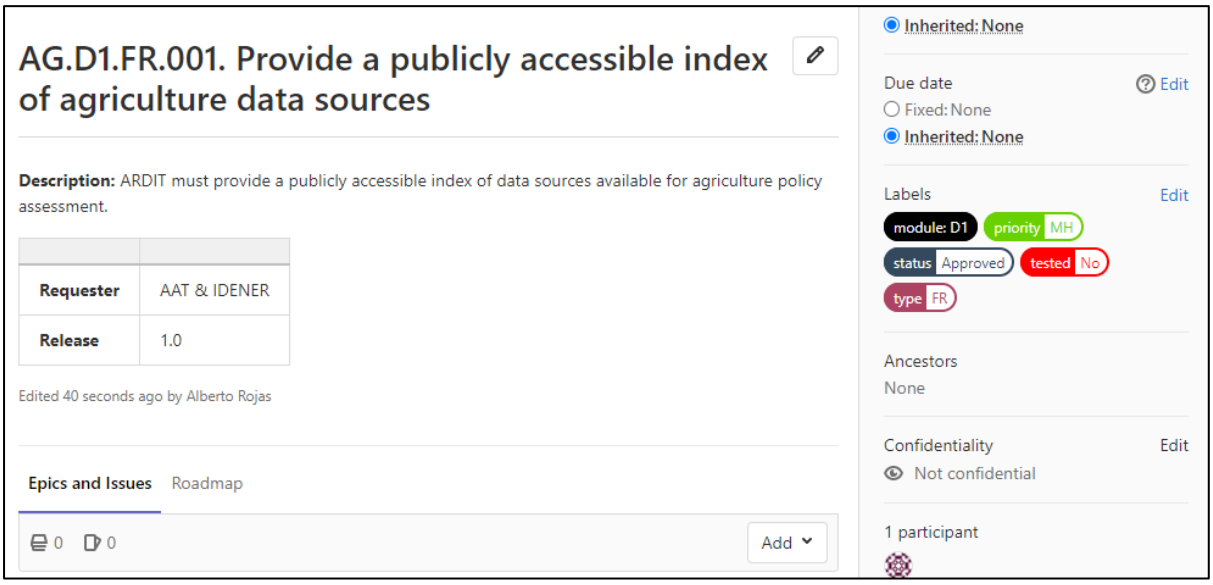

Figure 20. AG.D1.FR.001. Provide a publicly accessible index of agriculture data .sources.

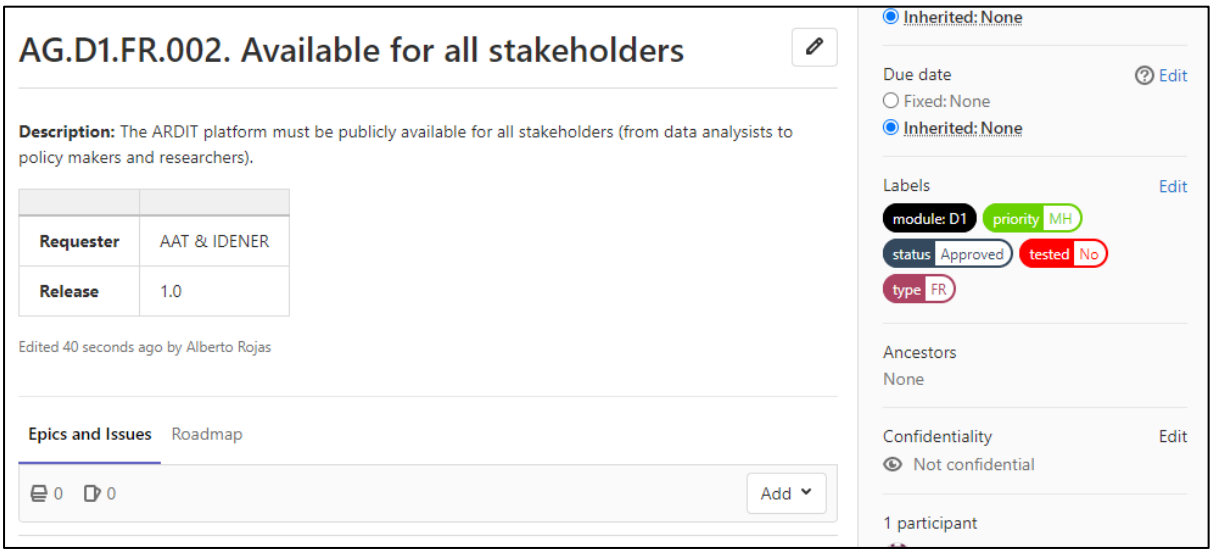

Figure 21. AG.D1.FR.002. Available for all stakeholders.

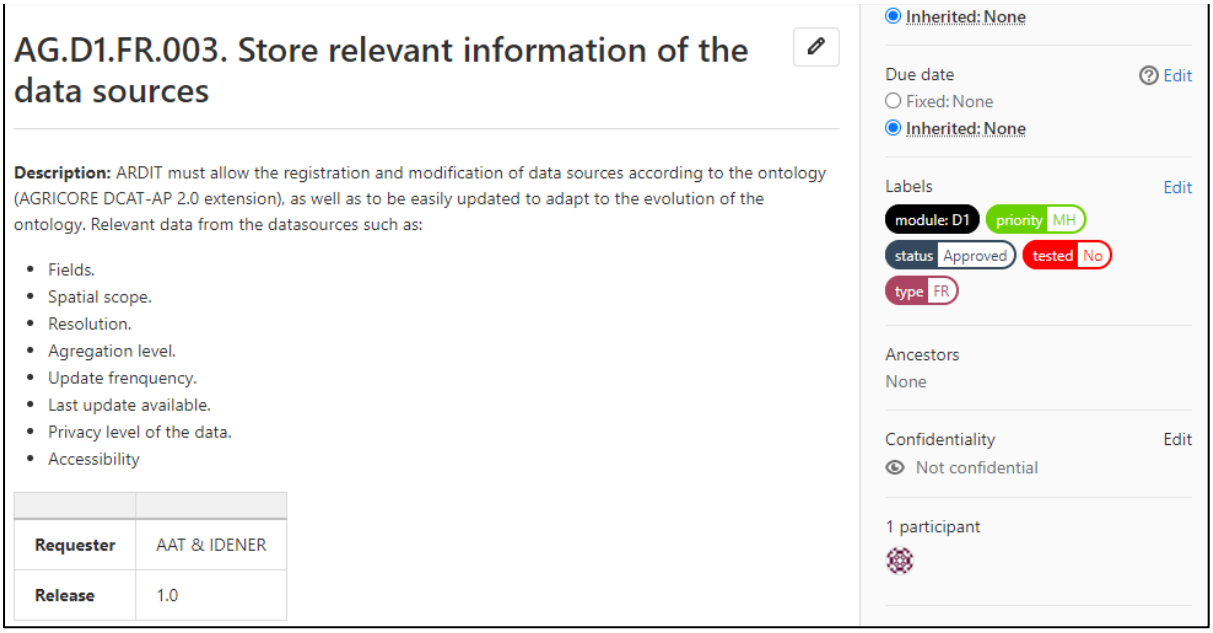

Figure 22. AG.D1.FR.003. Store relevant information of the data sources.

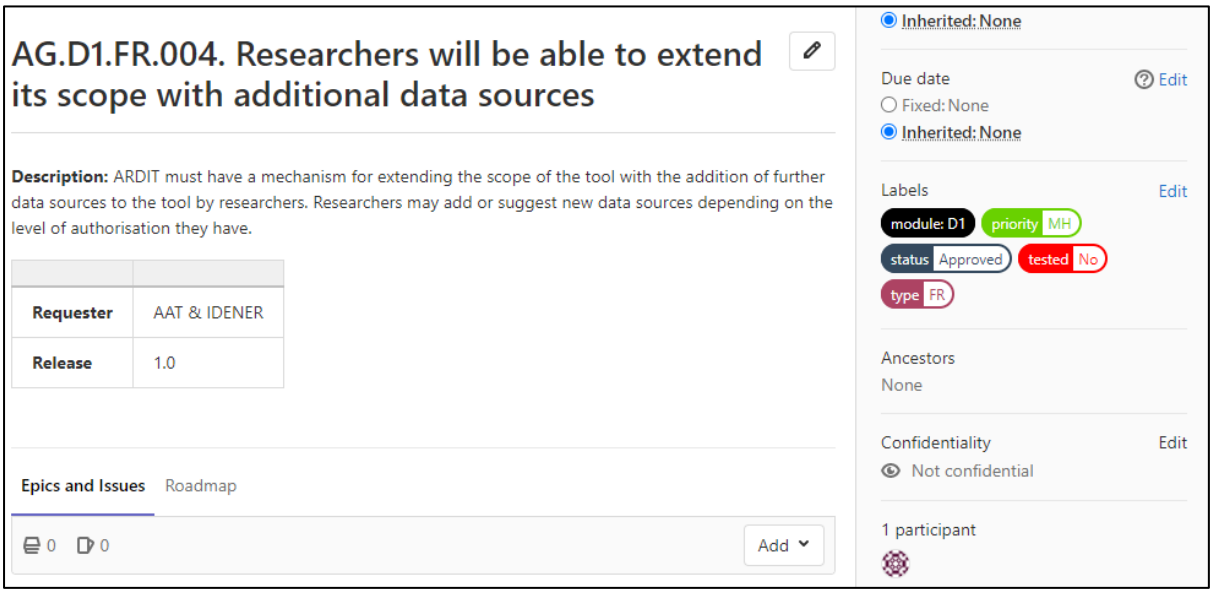

Figure 23. AG.D1.FR.004.Researchers will be able to extend its scope with additional data sources.

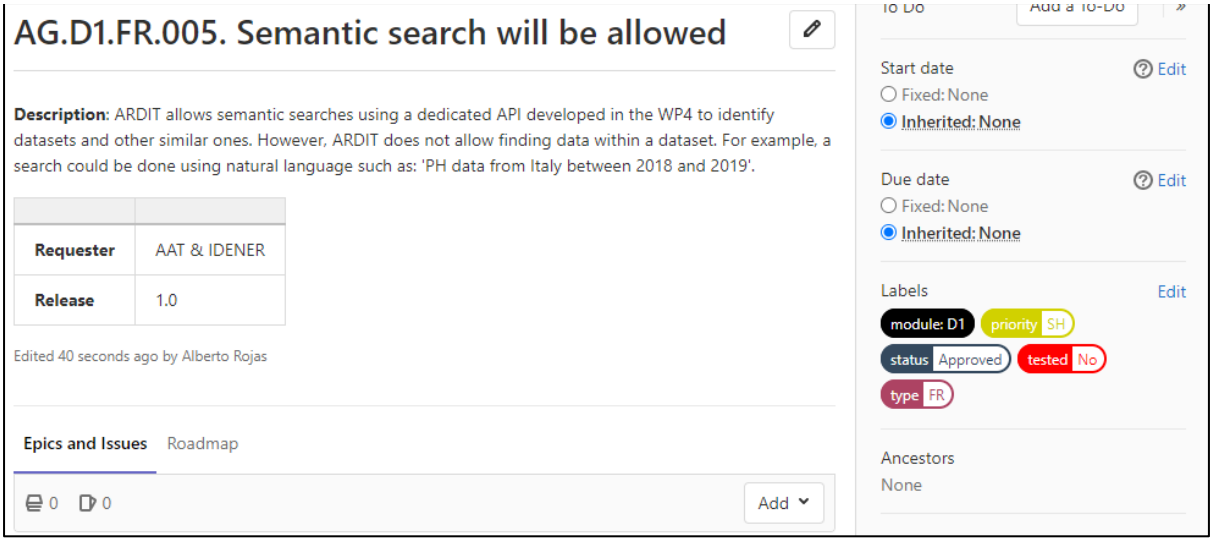

Figure 24. AG.D1.FR.005. Semantic search will be allowed.

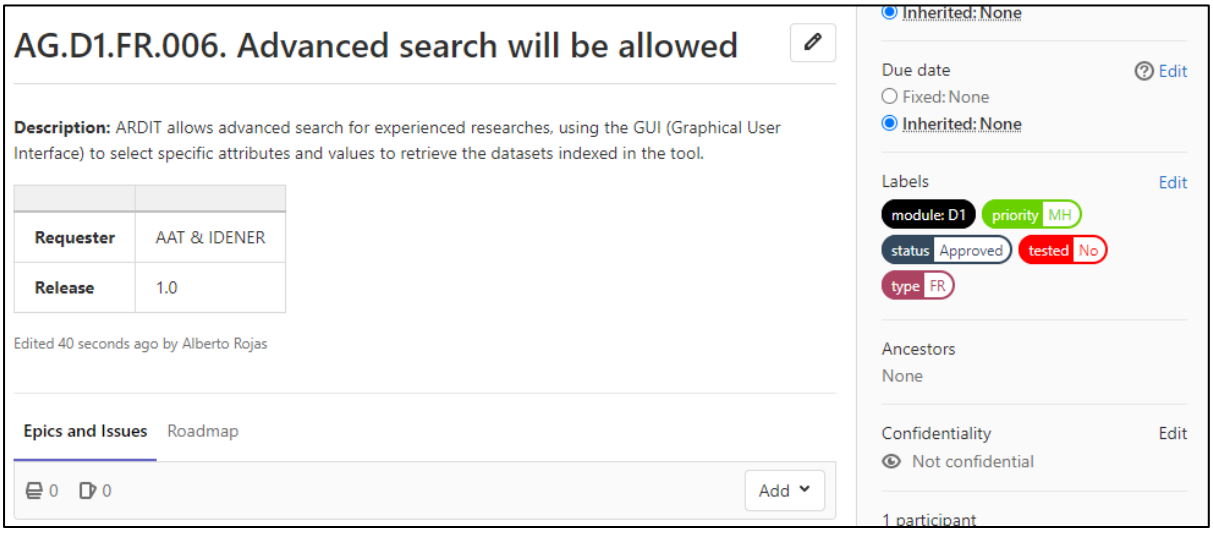

Figure 25. AG.D1.FR.006. Advanced search will be allowed

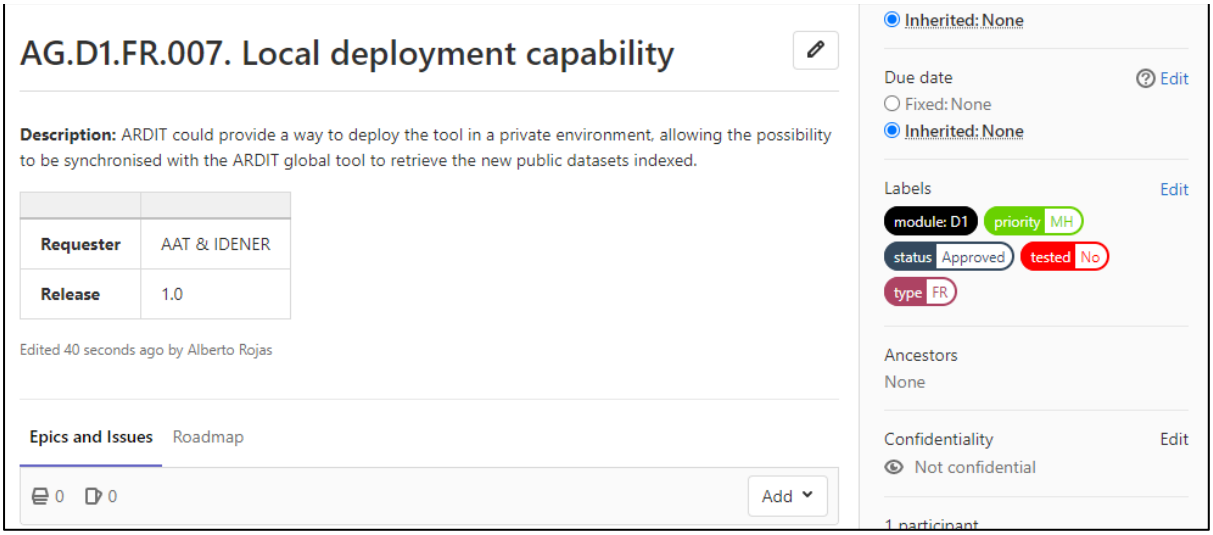

Figure 26. AG.D1.FR.007. Local deployment capability.

8.3 D2: DWH

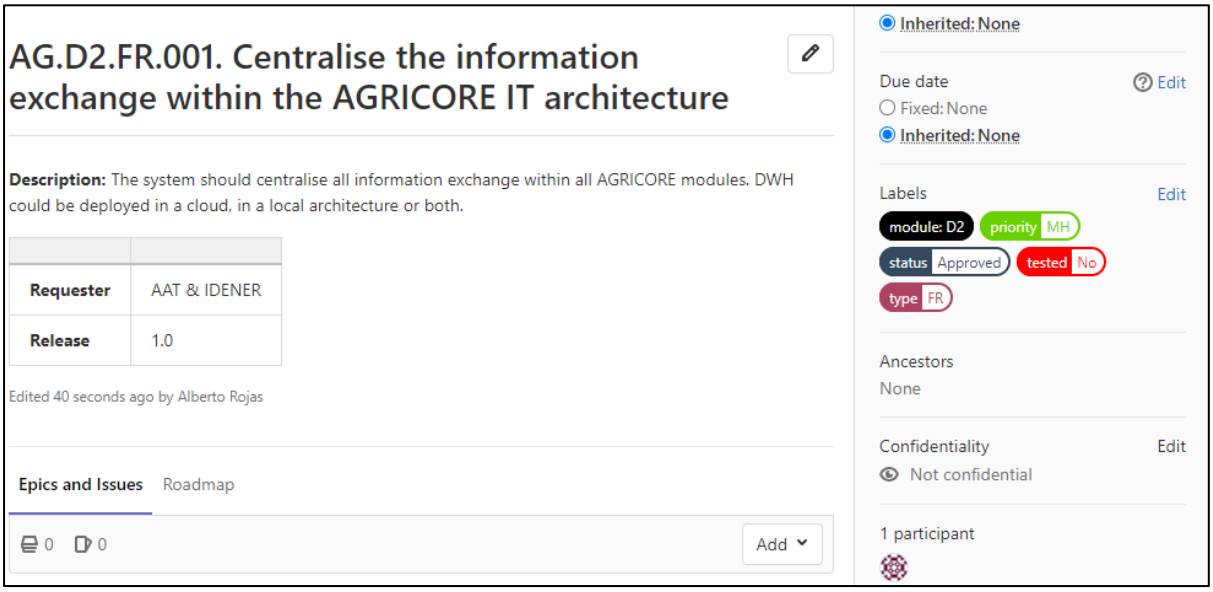

Figure 27. AG.D2.FR.001. Centralise the information exchange within the AGRICORE IT architecture.

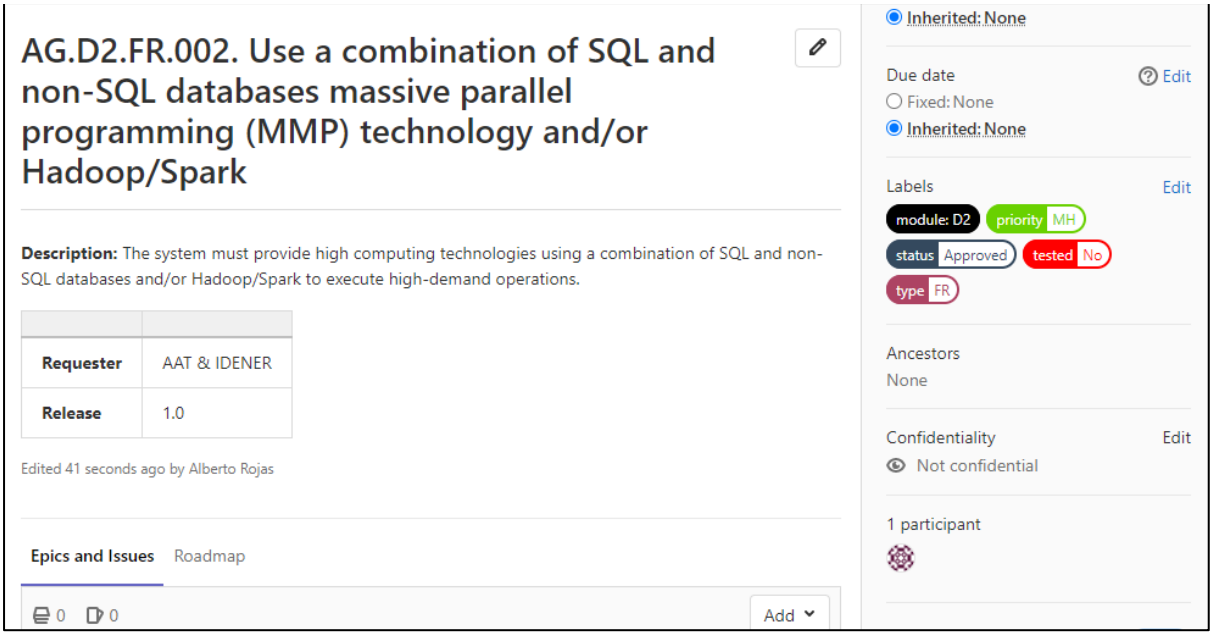

Figure 28. AG.D2.FR.002. Use a combination of SQL and non-SQL databases massive parallel programming (MMP) technology and/or Hadoop/Spark.

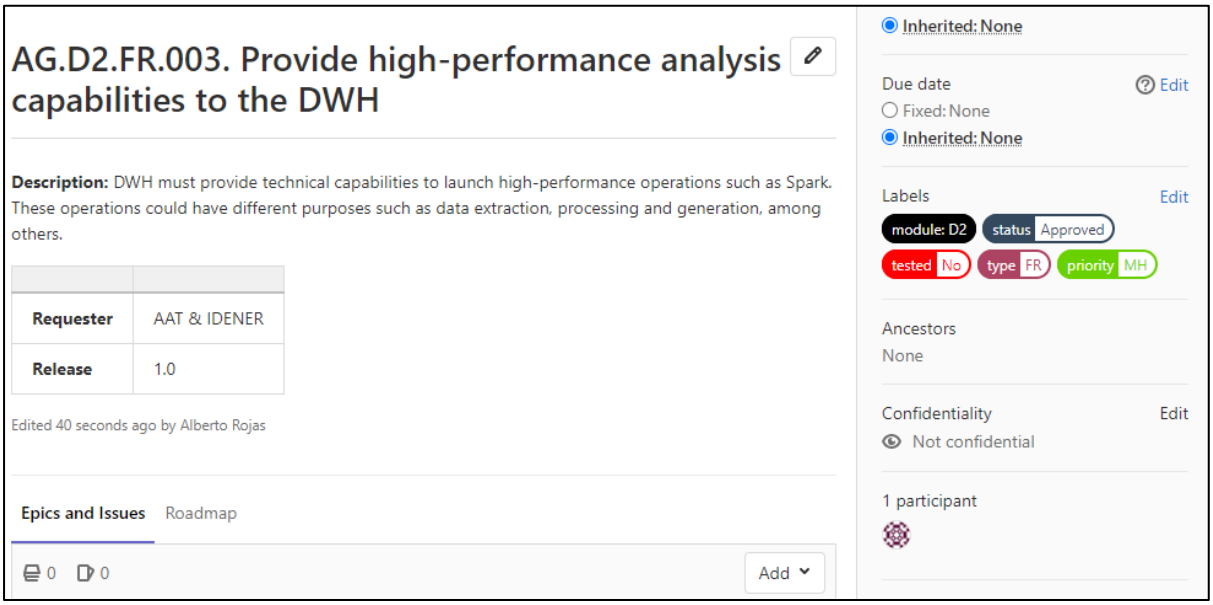

Figure 29. AG.D2.FR.003. Provide high-performance analysis capabilities to the DWH.

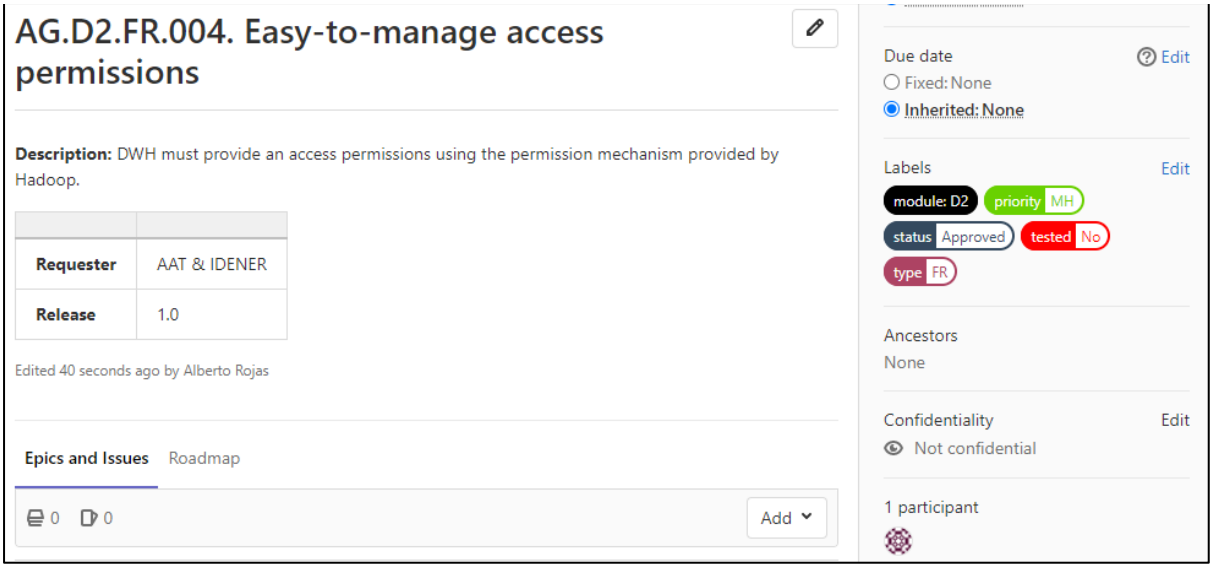

Figure 30. AG.D2.FR.004. Easy-to-manage access permissions.

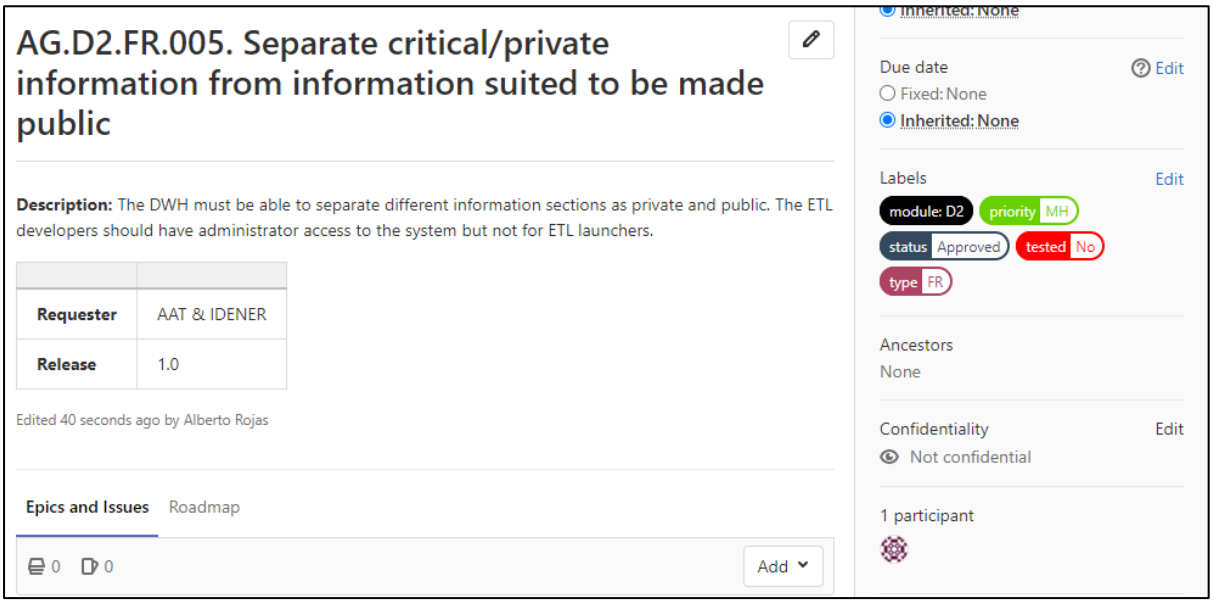

Figure 31. AG.D2.FR.005. Separate critical/private information from information suited to be made public.

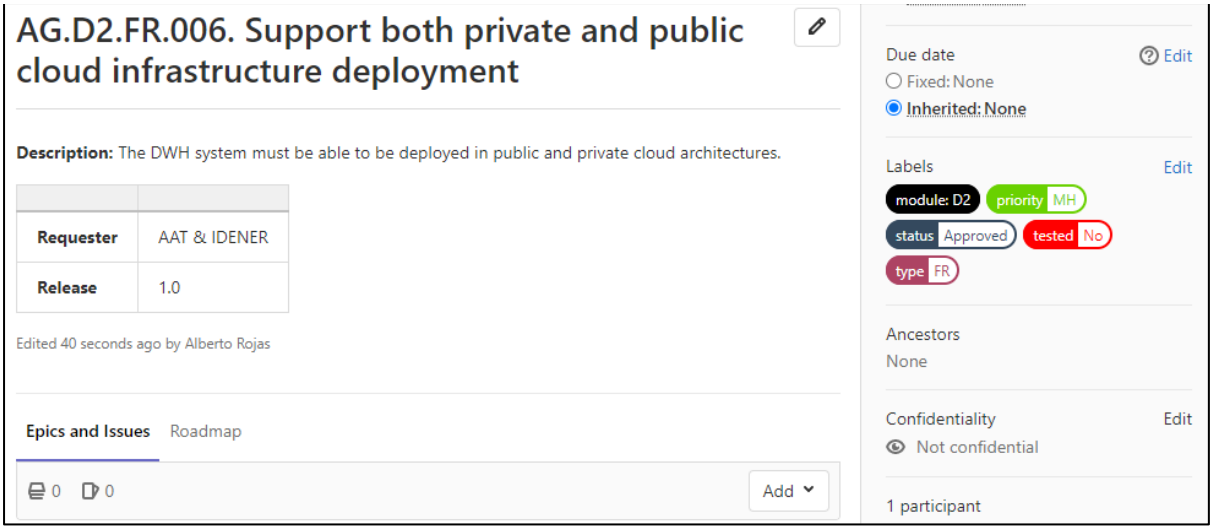

Figure 32. AG.D2.FR.006. Support both private and public cloud infrastructure deployment.

8.4 D3: Data extraction module

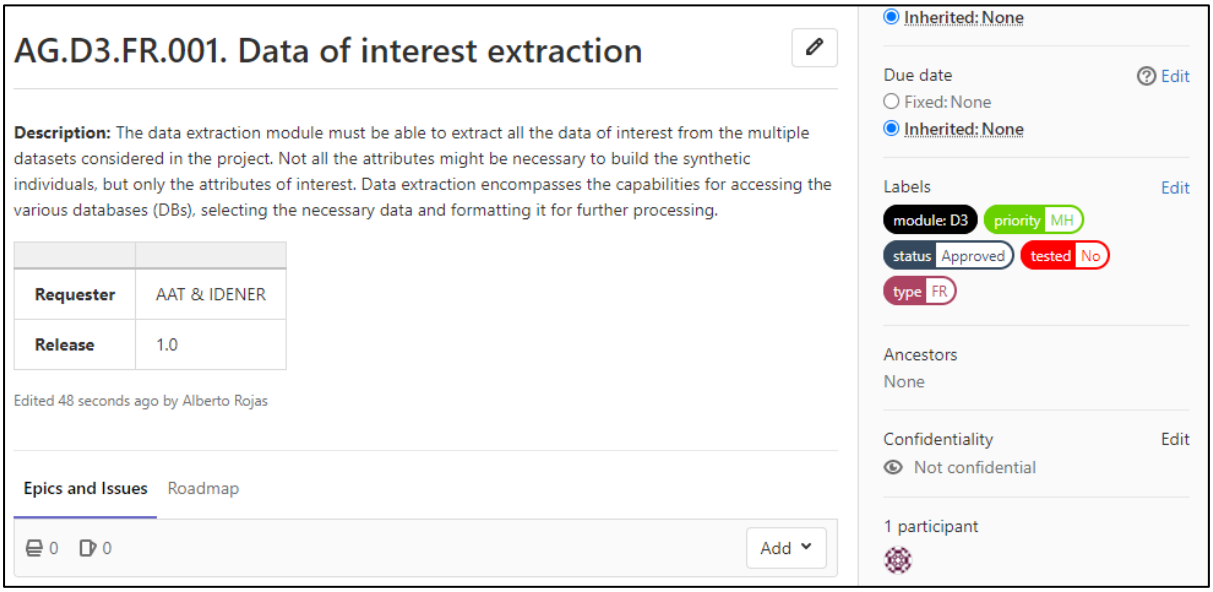

Figure 33. AG.D3.FR.001. Data of interest extraction.

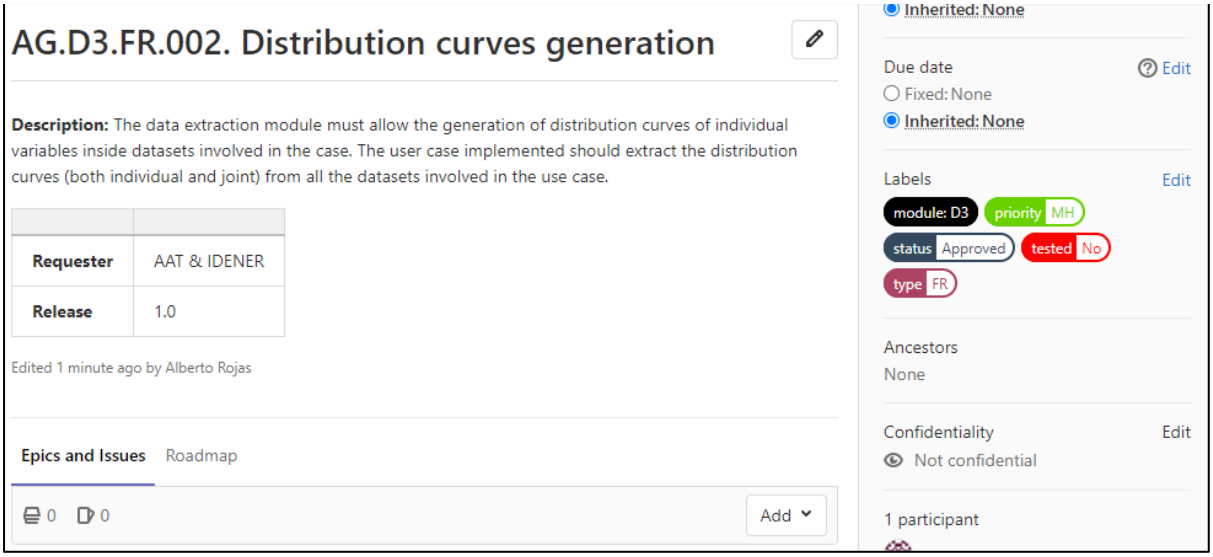

Figure 34. AG.D3.FR.002. Distribution curves generation.

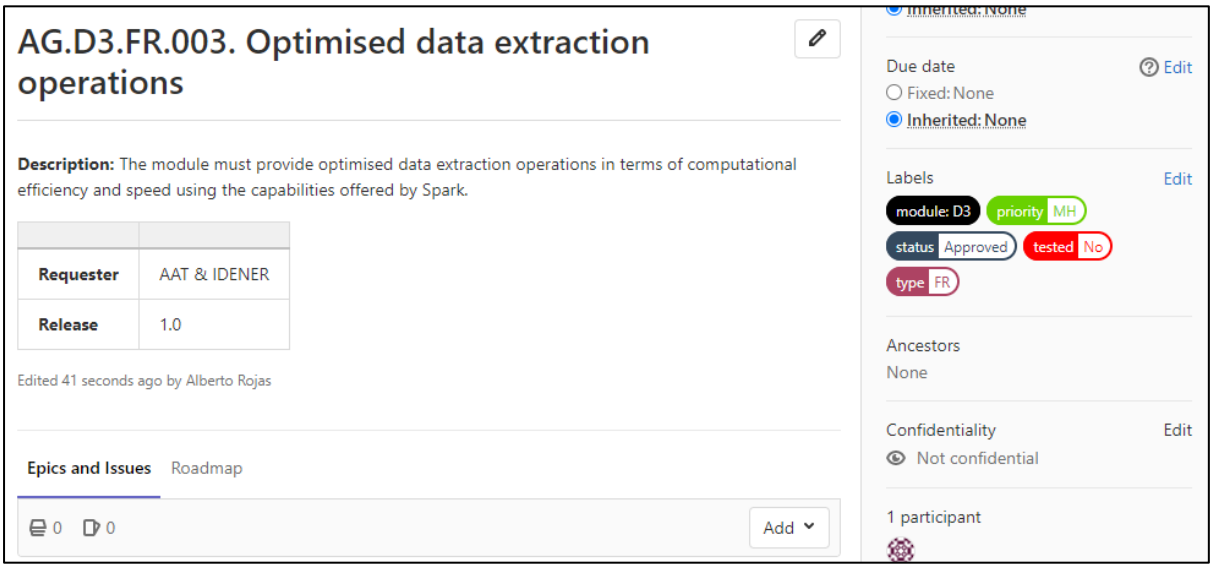

Figure 35. AG.D3.FR.003. Optimised data extraction operations.

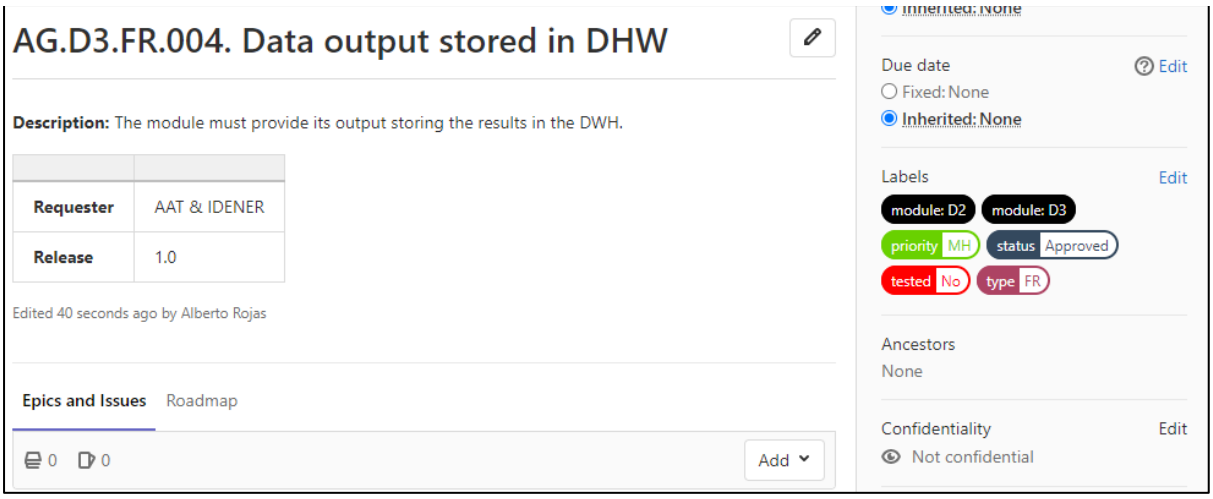

Figure 36. AG.D3.FR.004. Data output stored in DHW.

8.5 D4: Data fusion module

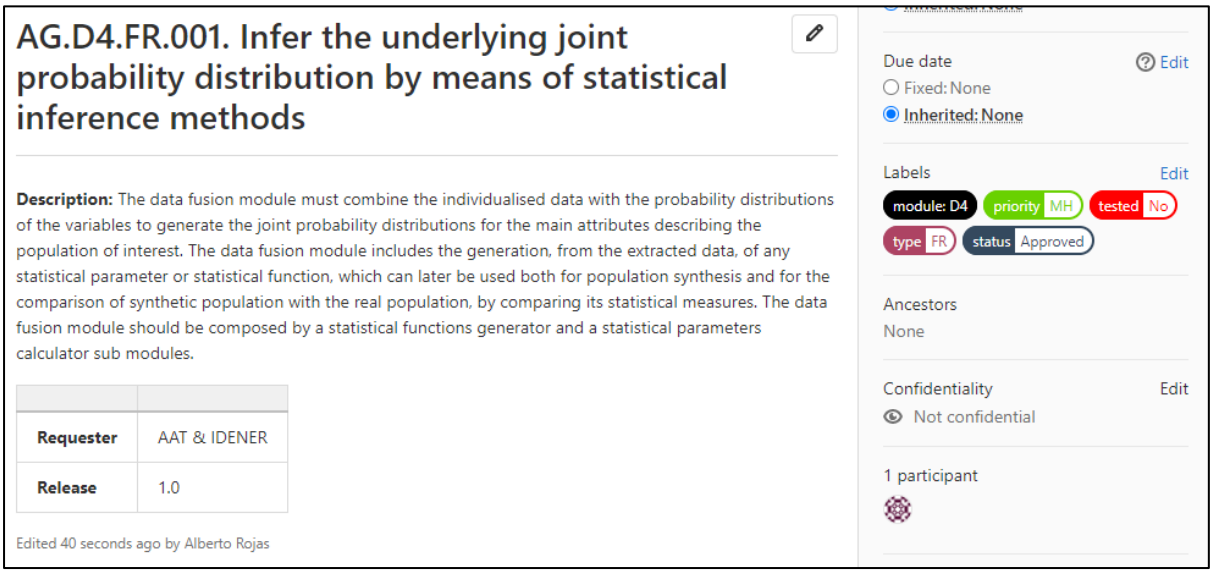

Figure 37. AG.D4.FR.001. Infer the underlying joint probability distribution by means of statistical inference methods.

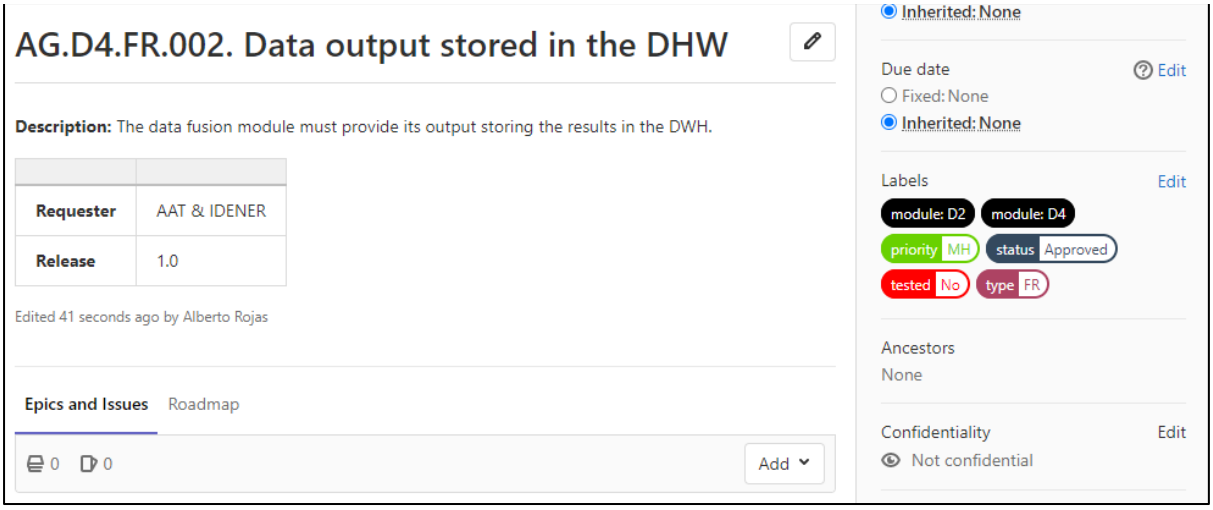

Figure 38. AG.D4.FR.002. Data output stored in the DHW.

8.6 D5: Synthetic populations generator

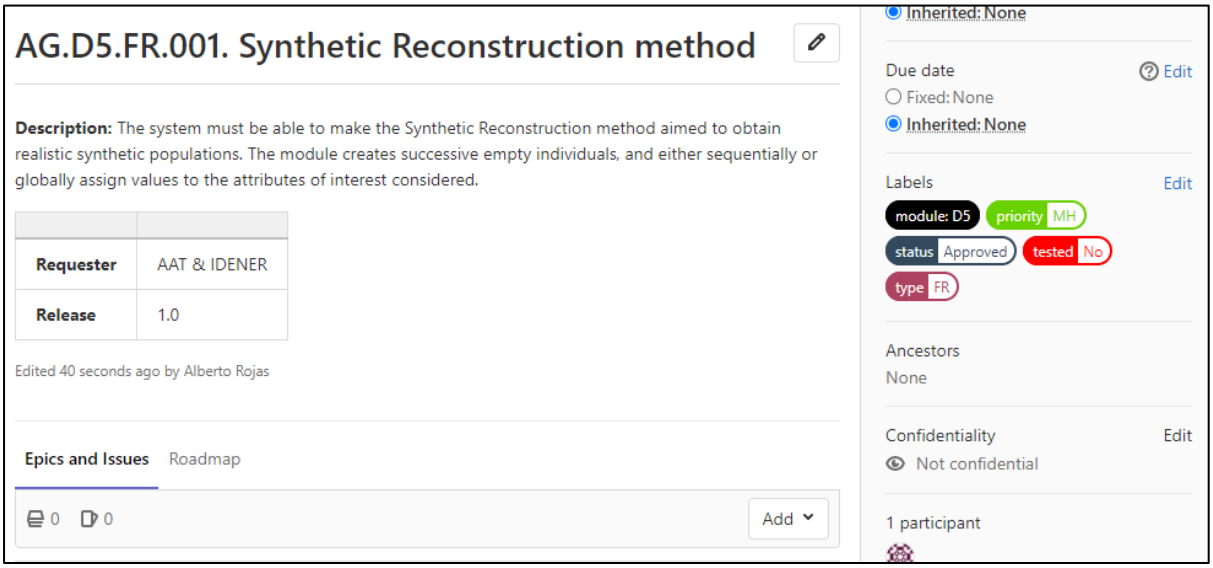

Figure 39. AG.D5.FR.001. Synthetic Reconstruction method.

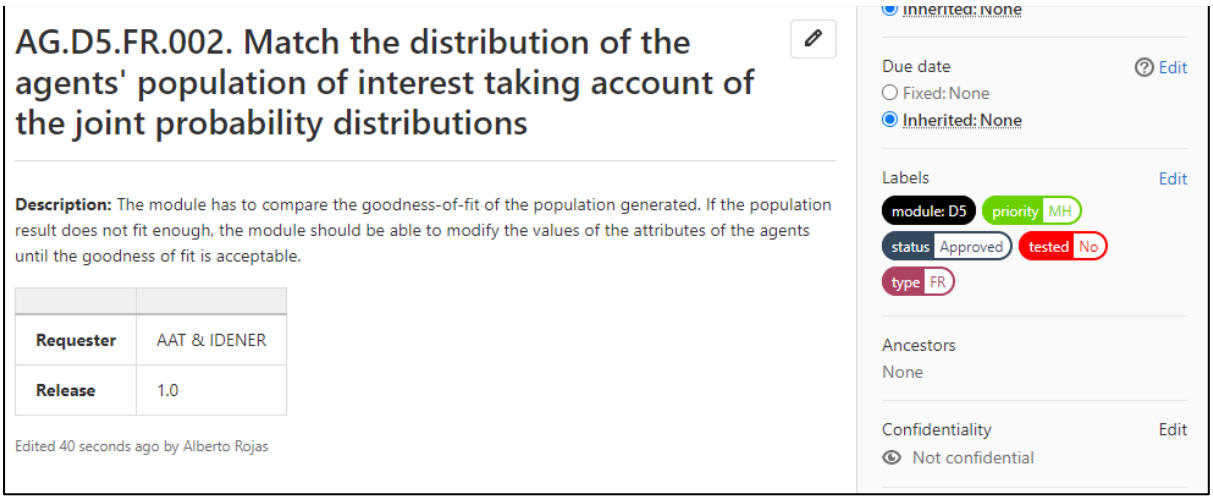

Figure 40. AG.D5.FR.002. Match the distribution of the agents' population of interest taking account of the joint probability distributions.

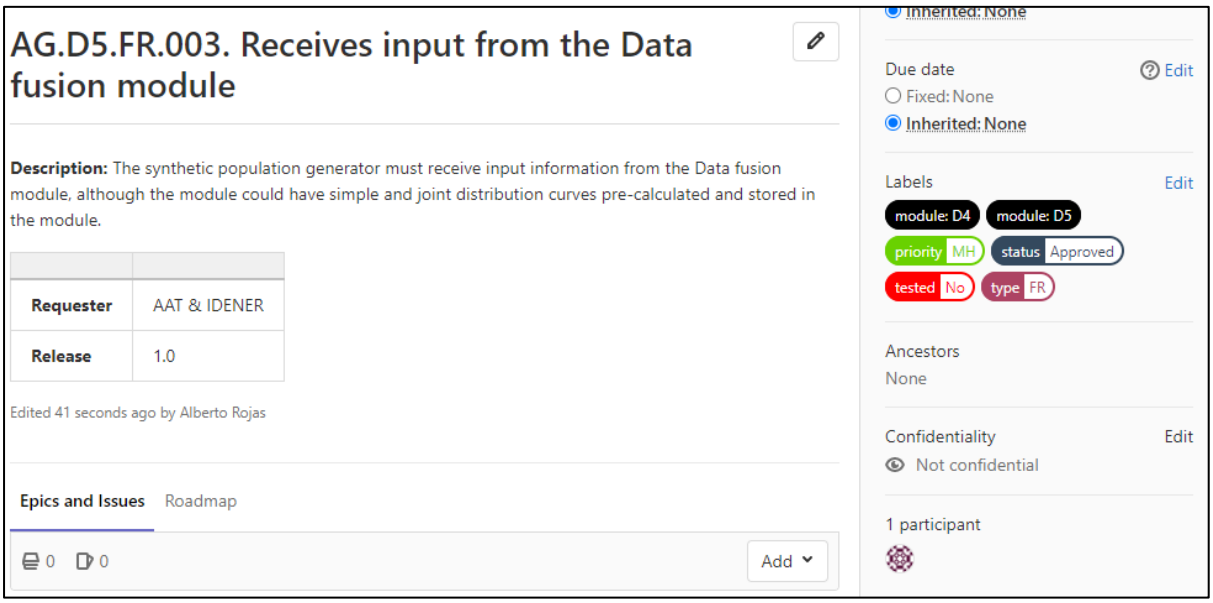

Figure 41. AG.D5.FR.003. Receives input from the Data fusion module.

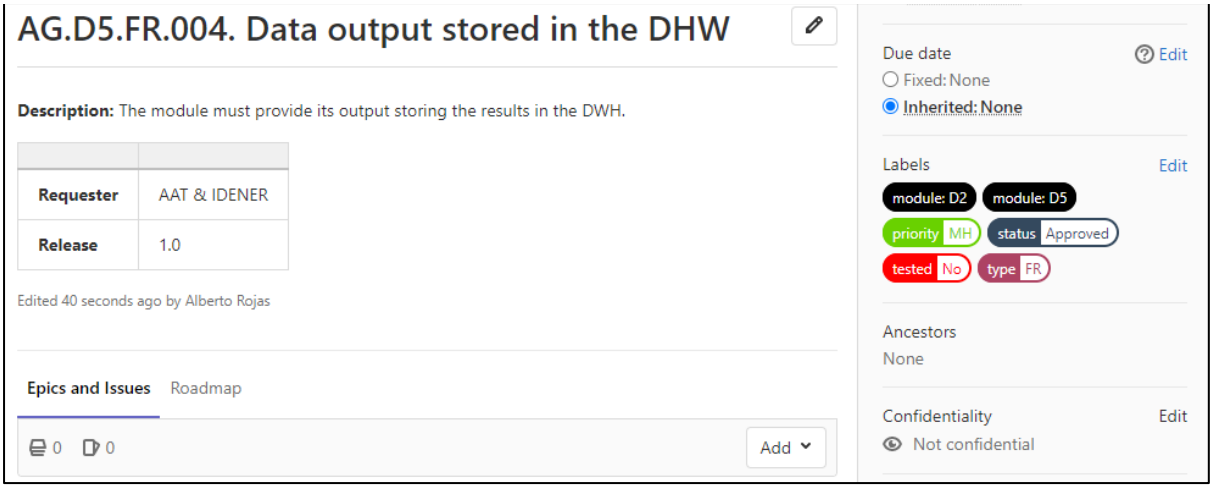

Figure 42. AG.D5.FR.004. Data output stored in the DHW.

8.7 D6: ABM simulation engine

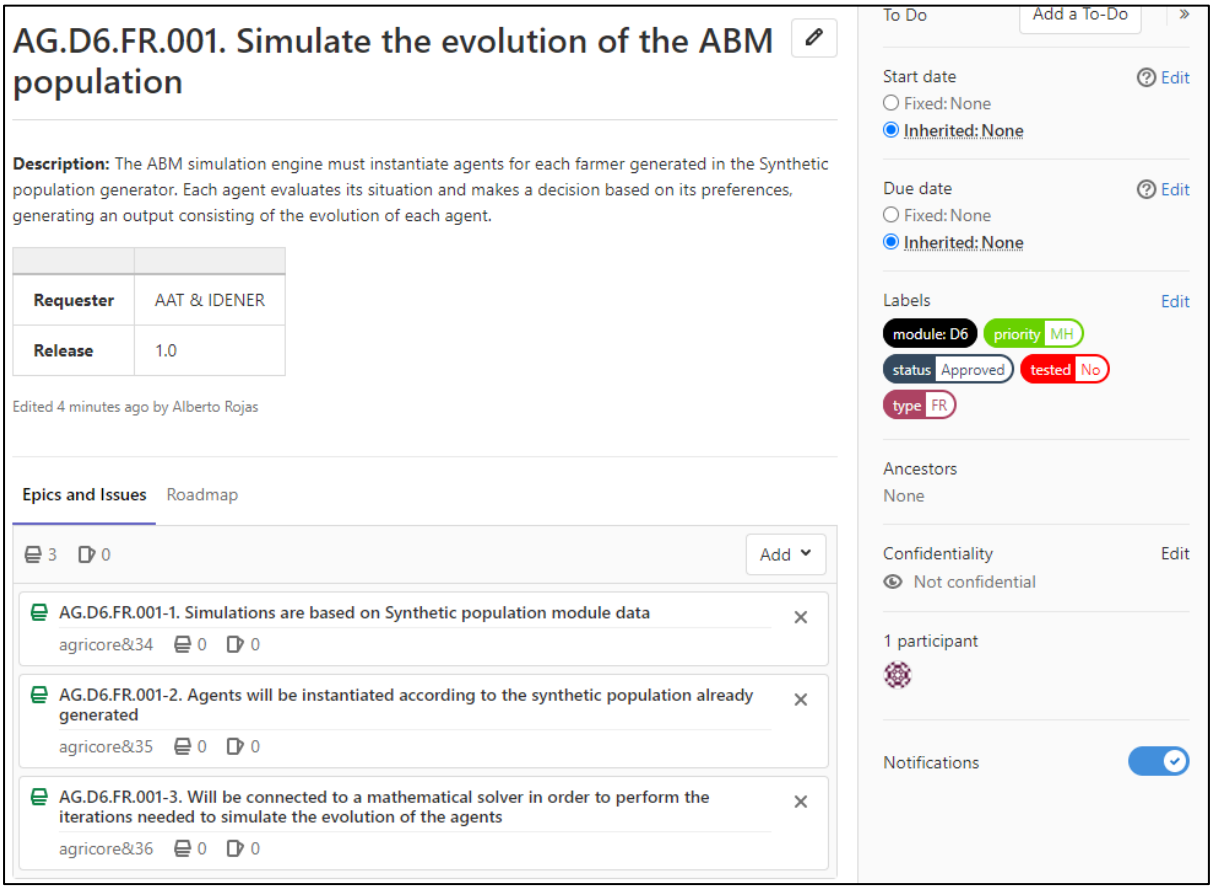

Figure 43. AG.D6.FR.001. Simulate the evolution of the ABM population.

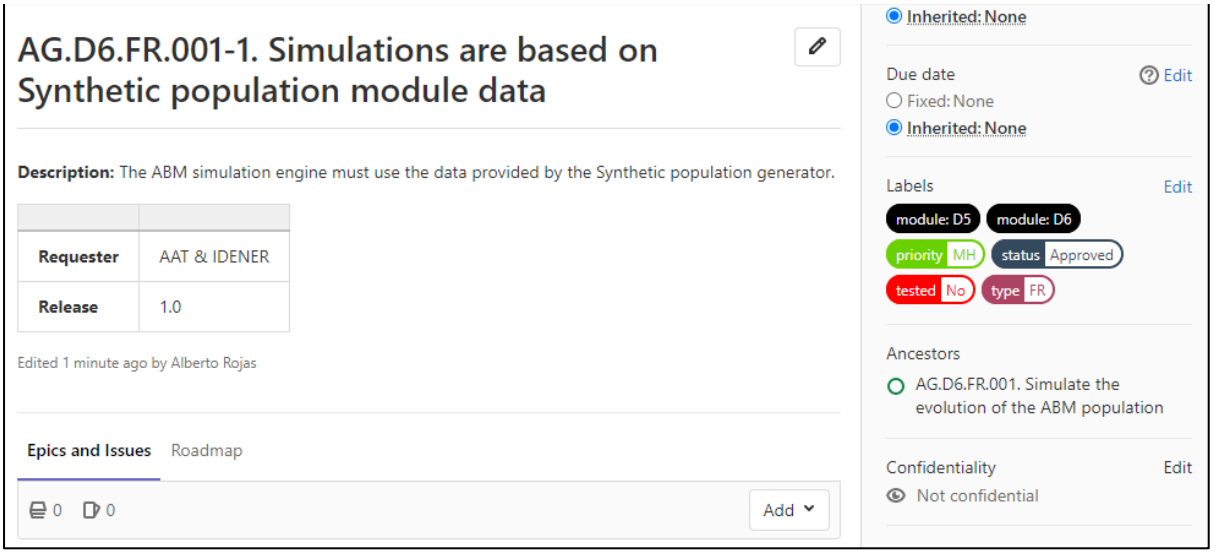

Figure 44. AG.D6.FR.001-1. Simulations are based on Synthetic population module data.

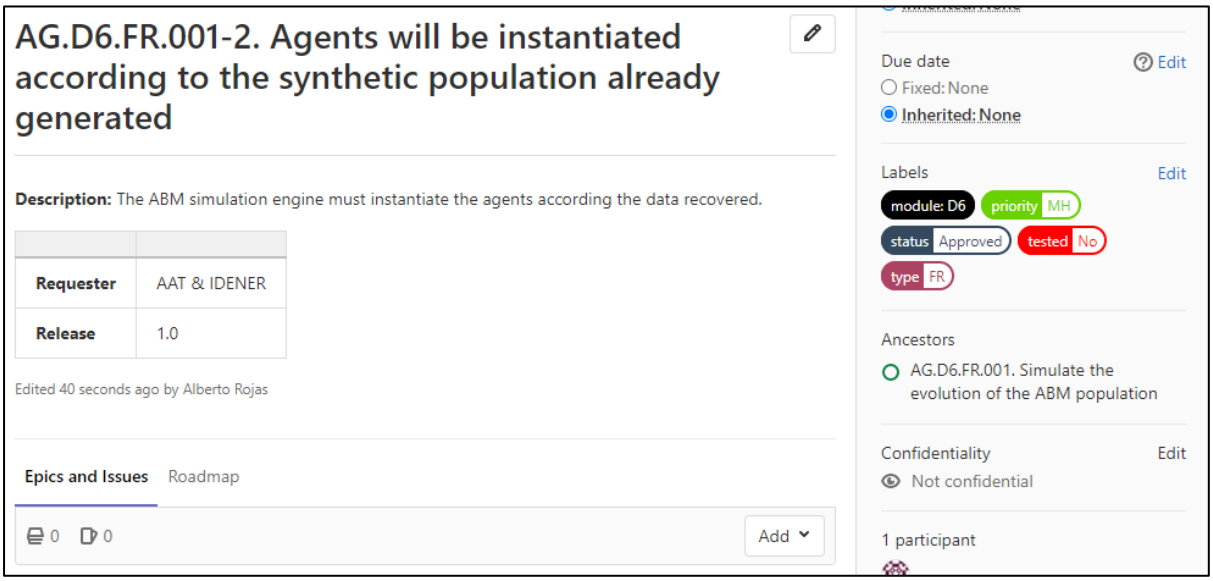

Figure 45. AG.D6.FR.001-2. Agents will be instantiated according to the synthetic population already generated.

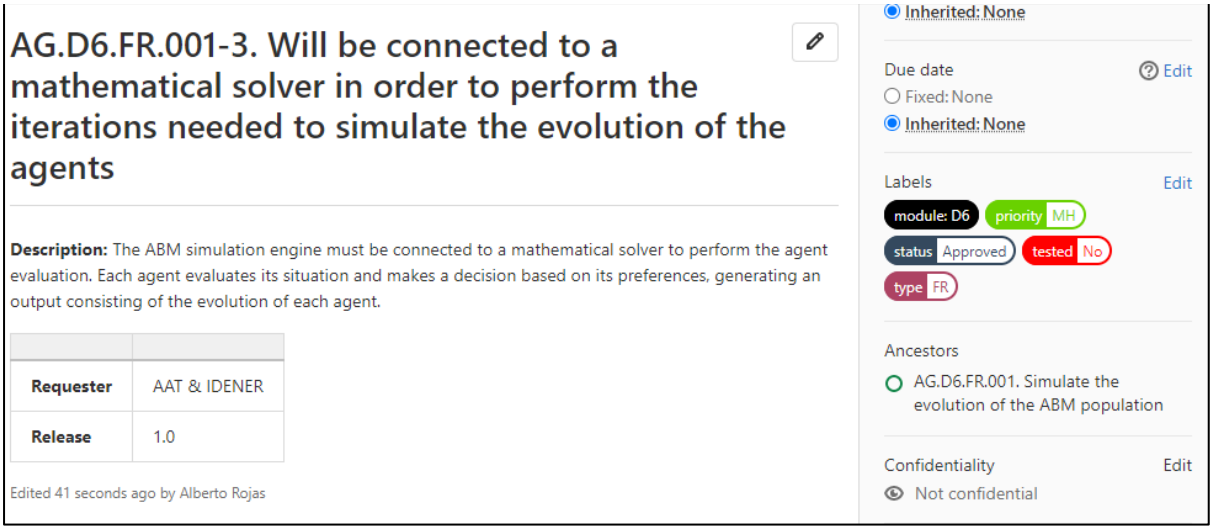

Figure 46. AG.D6.FR.001-3. Will be connected to a mathematical solver in order to perform the iterations needed to simulate the evolution of the agents.

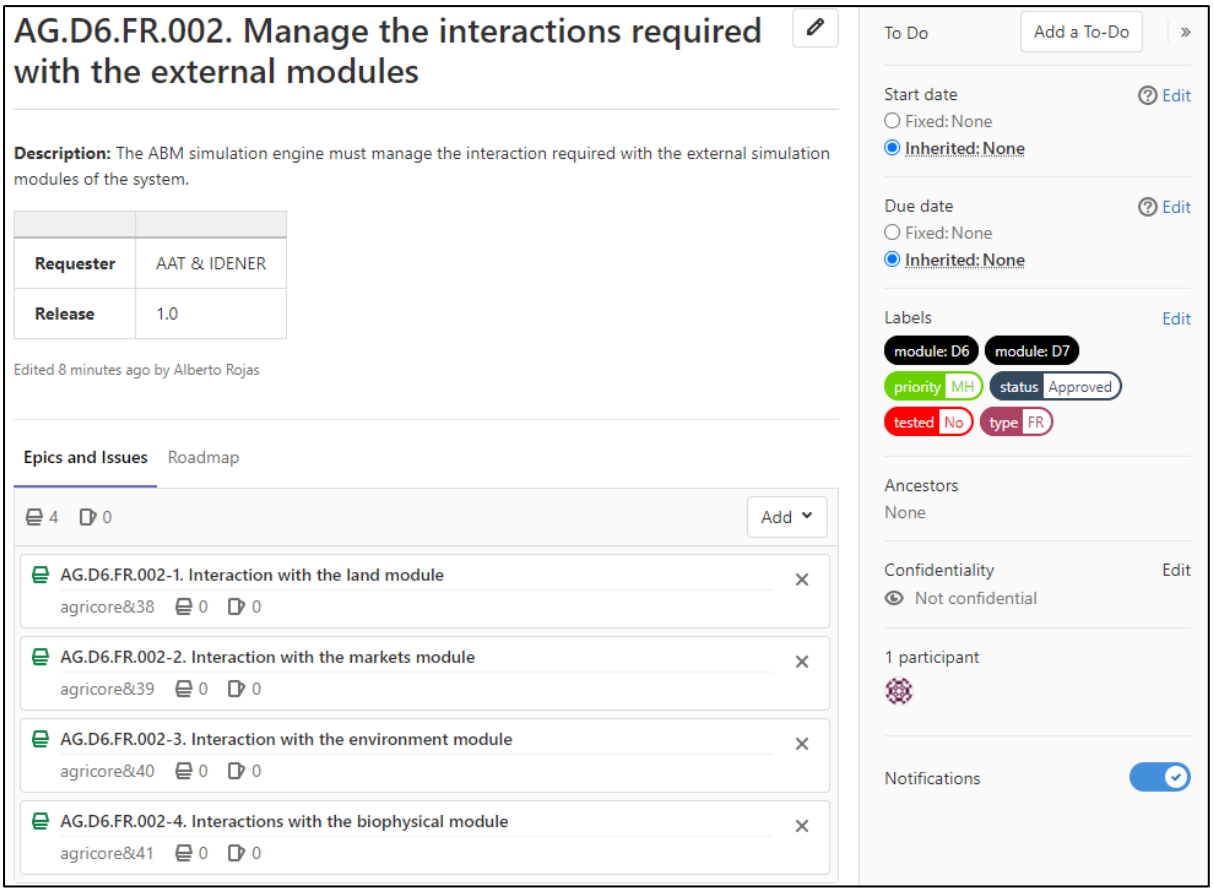

Figure 47. AG.D6.FR.002. Manage the interactions required with the external modules.

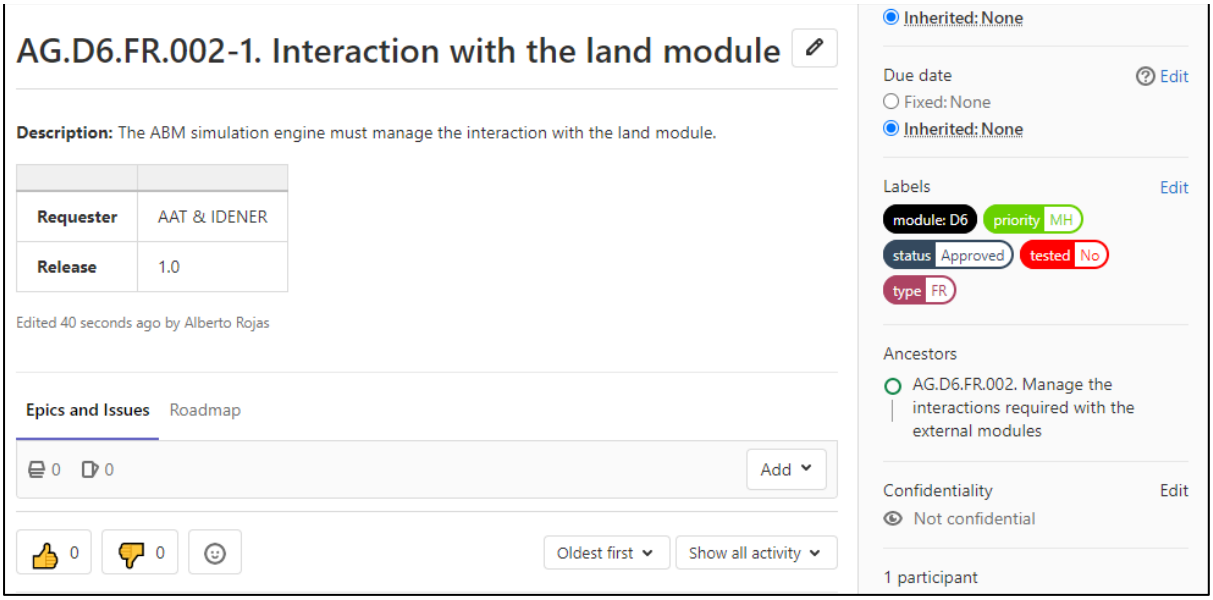

Figure 48. AG.D6.FR.002-1. Interaction with the land module.

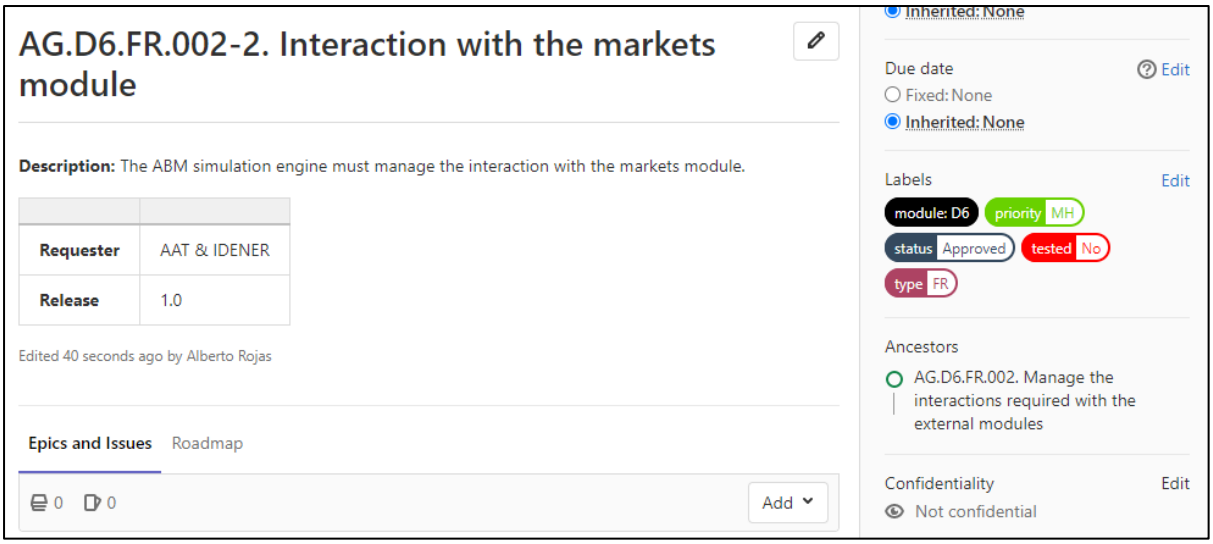

Figure 49. AG.D6.FR.002-2. Interaction with the markets module.

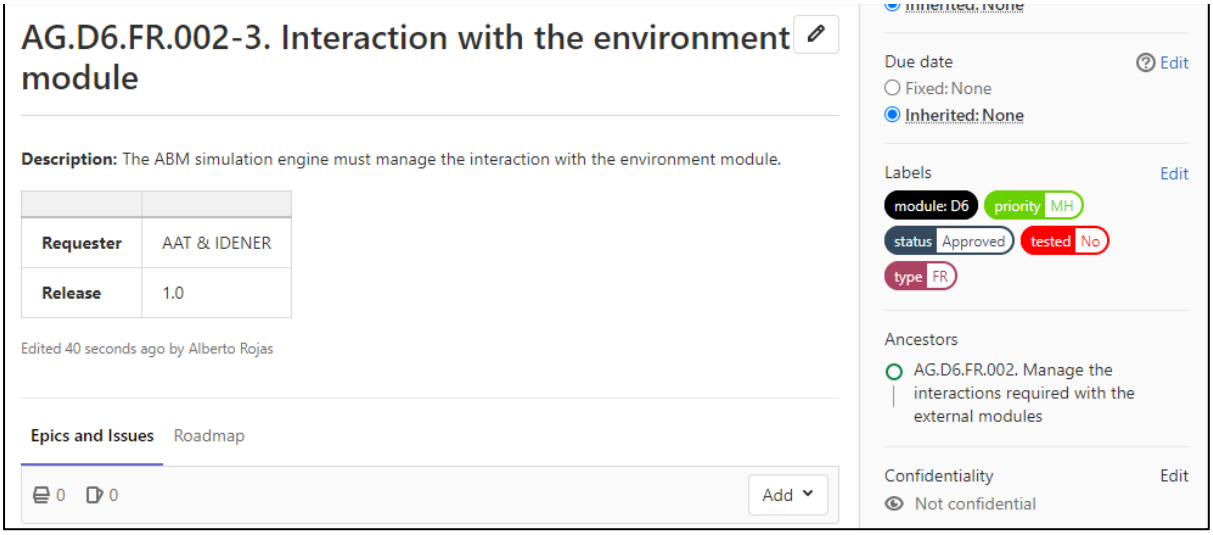

Figure 50. AG.D6.FR.002-3. Interaction with the environment module.

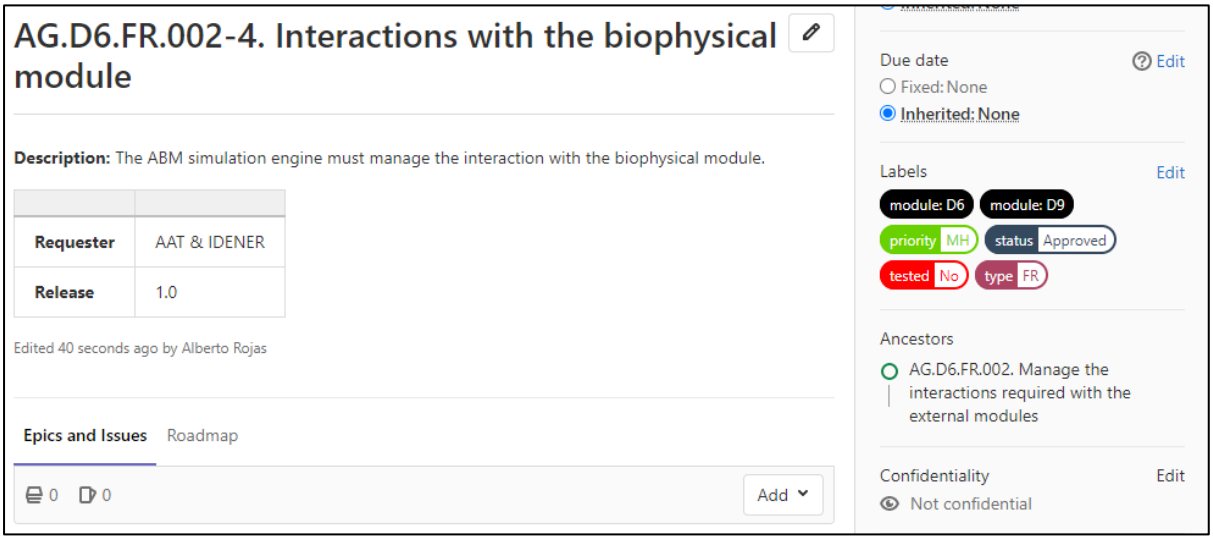

Figure 51. AG.D6.FR.002-4. Interactions with the biophysical module.

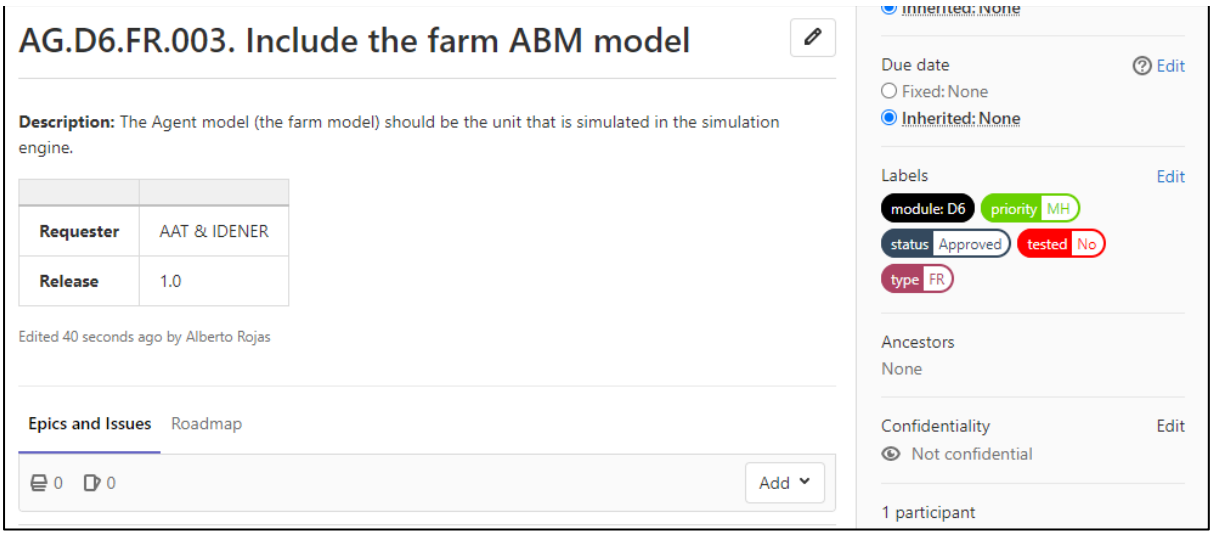

Figure 52. AG.D6.FR.003. Include the farm ABM model.

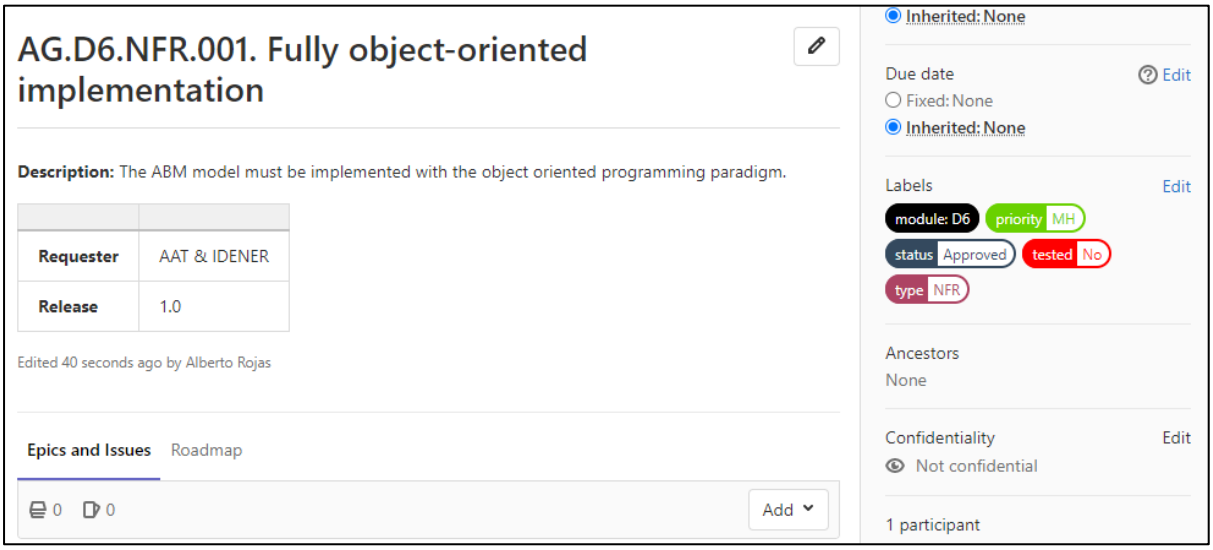

Figure 53. AG.D6.NFR.001. Fully object-oriented implementation.

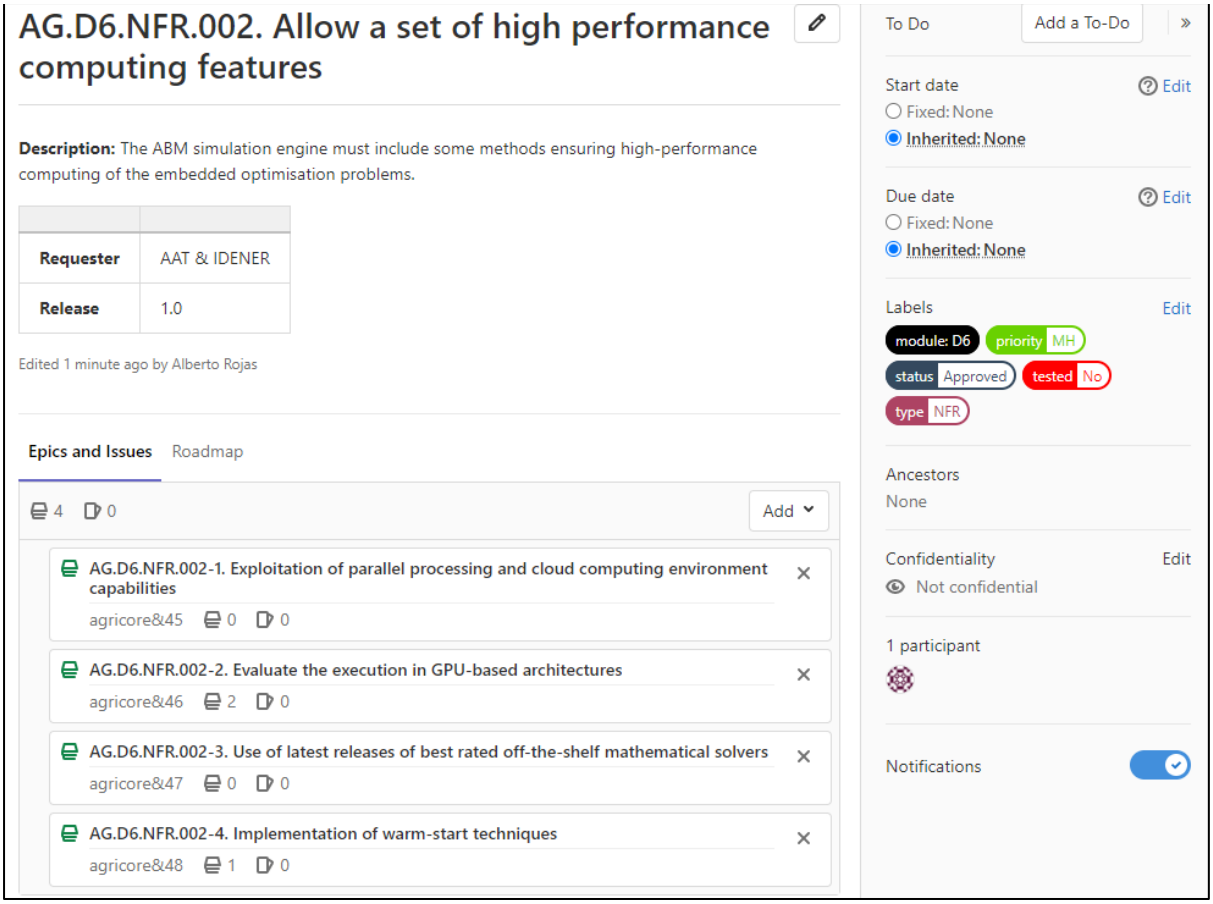

Figure 54. AG.D6.NFR.002. Allow a set of high performance computing features.

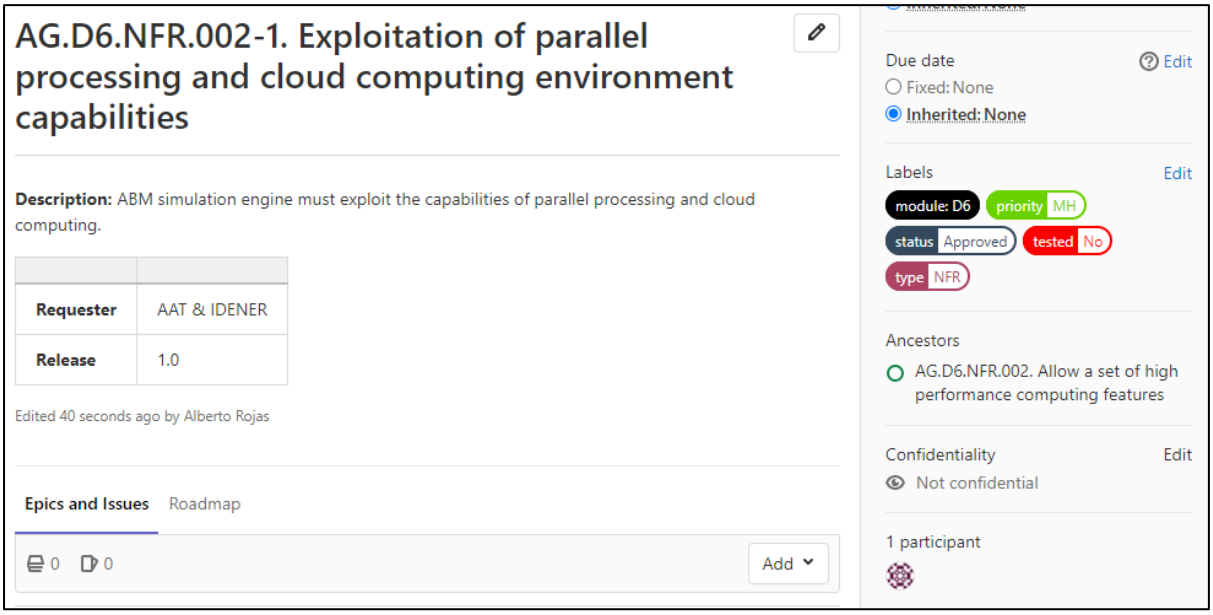

Figure 55. AG.D6.NFR.002-1. Exploitation of parallel processing and cloud computing environment capabilities.

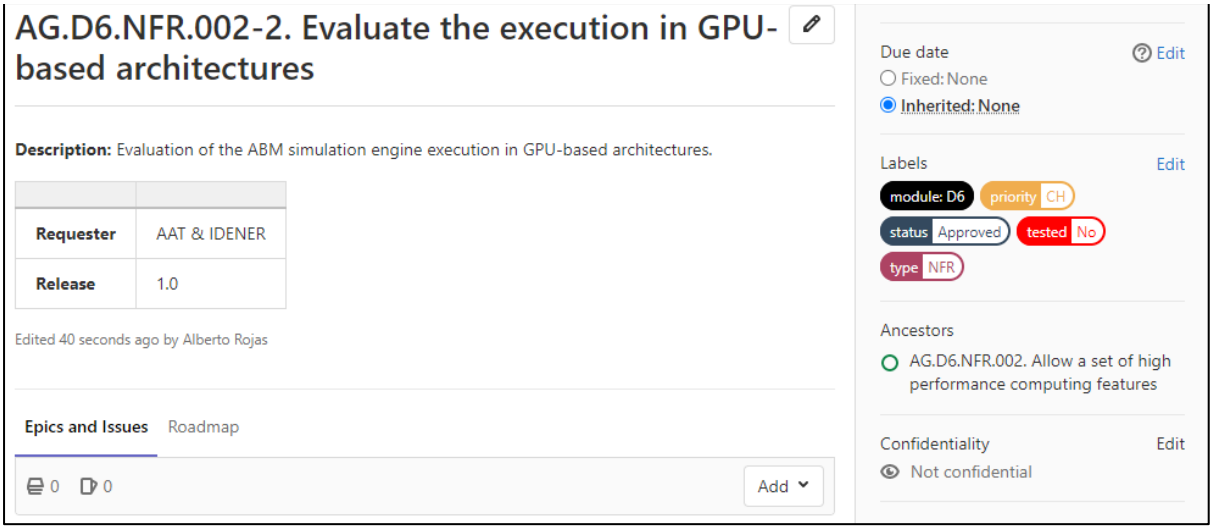

Figure 56. AG.D6.NFR.002-2. Evaluate the execution in GPU-based architectures.

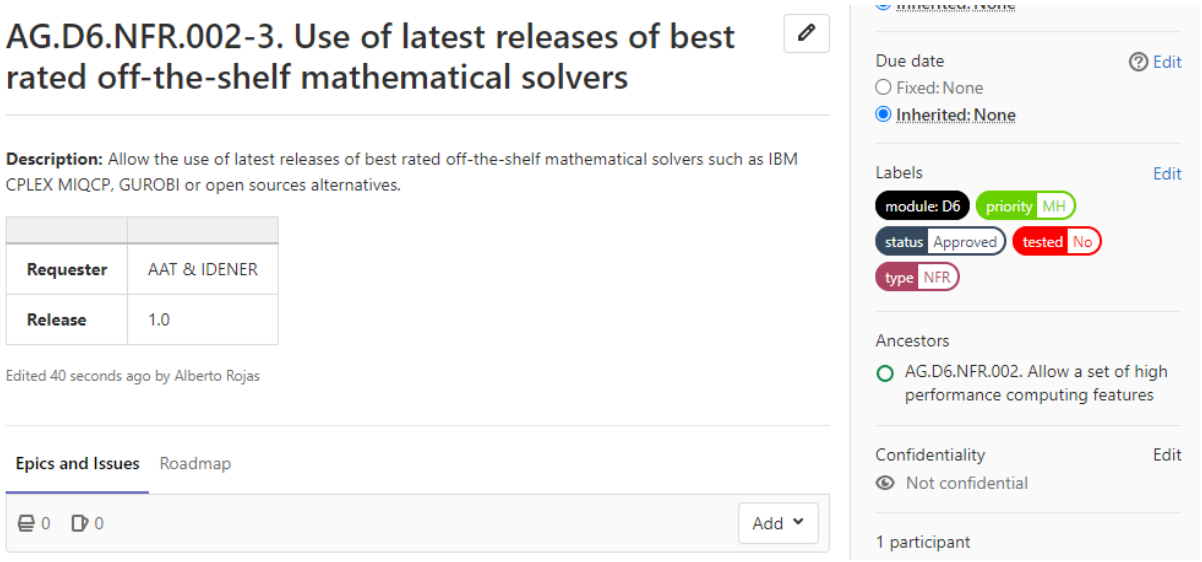

Figure 57. AG.D6.NFR.002-3. Use of latest releases of best rated off-the-shelf mathematical solversnt.

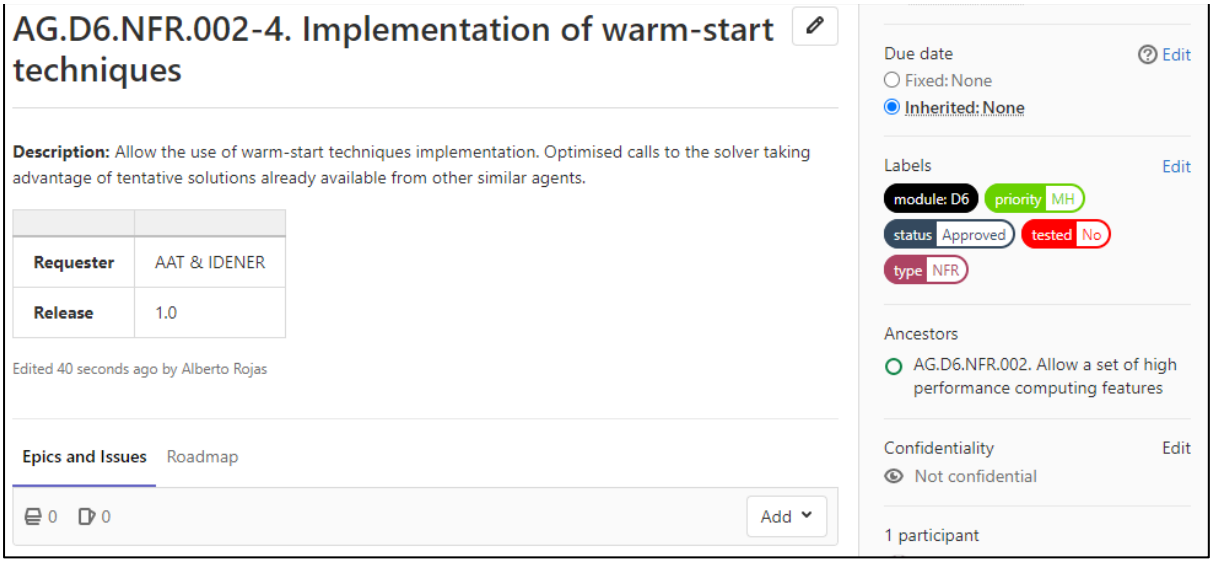

Figure 58. AG.D6.NFR.002-4. Implementation of warm-start techniques.

8.8 D7: External Interface module

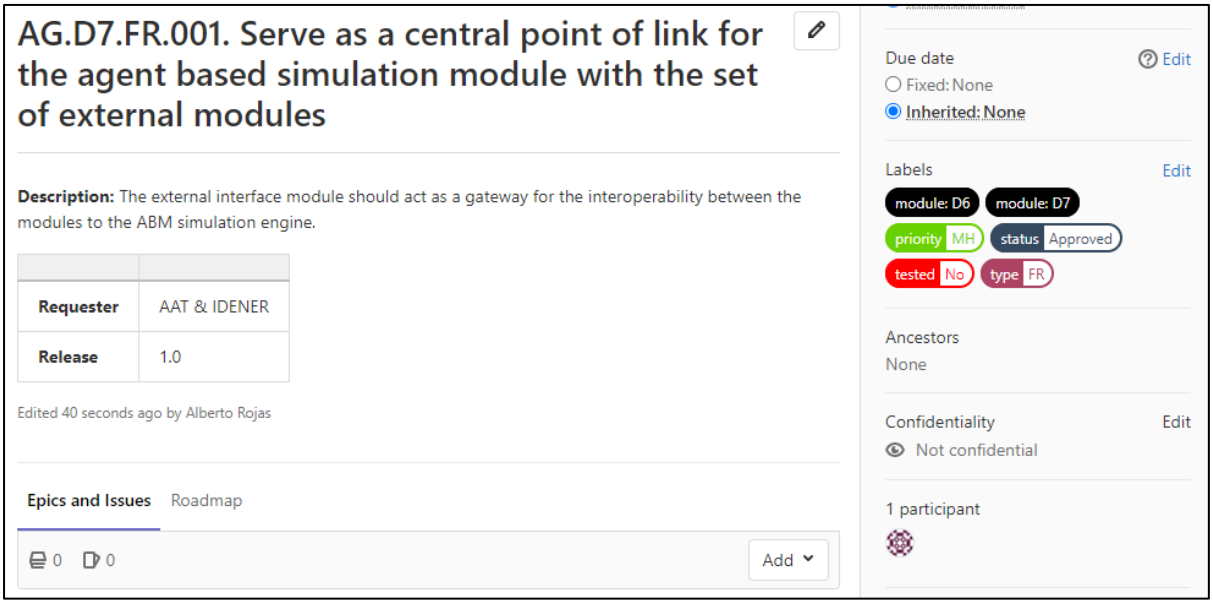

Figure 59. AG.D7.FR.001. Serve as a central point of link for the agent based simulation module with the set of external modules.

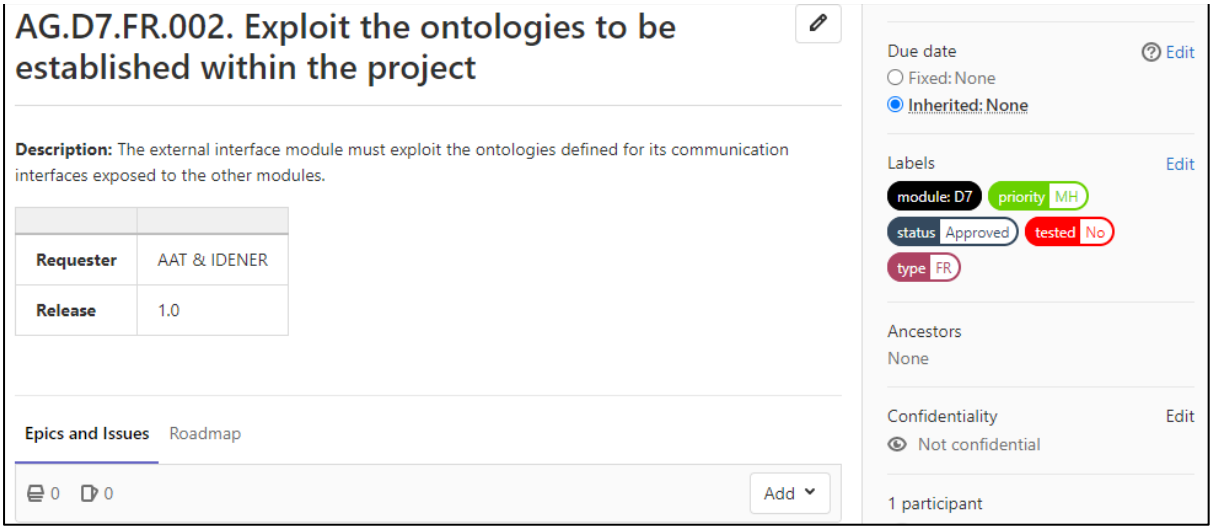

Figure 60. AG.D7.FR.002. Exploit the ontologies to be established within the project.

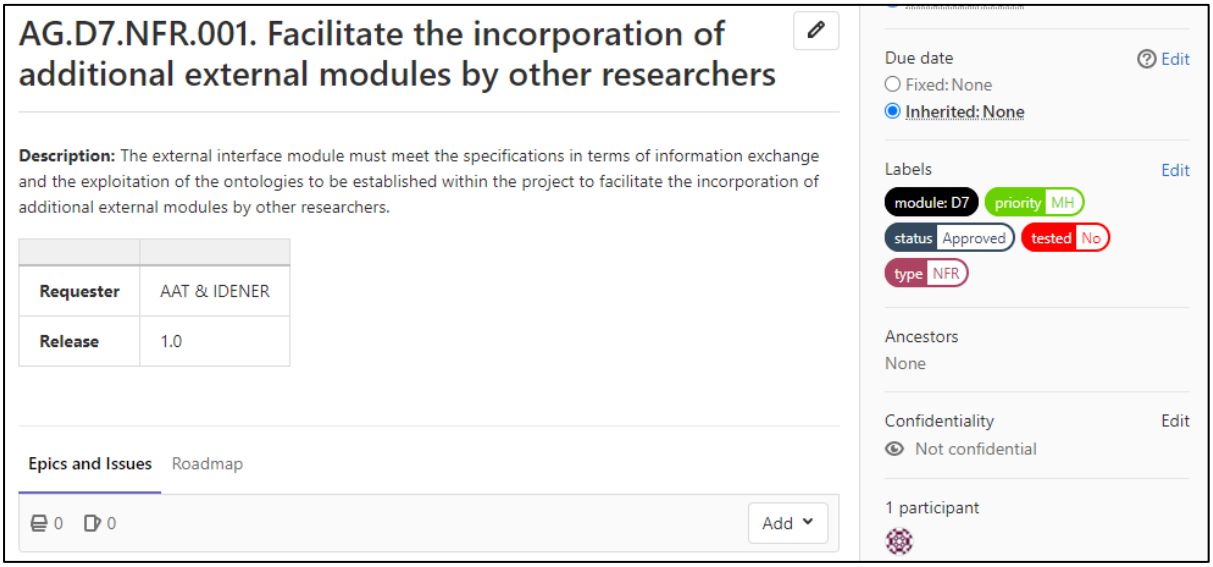

Figure 61. AG.D7.NFR.001. Facilitate the incorporation of additional external modules by other researchers.

8.9 D8: Model interaction modules

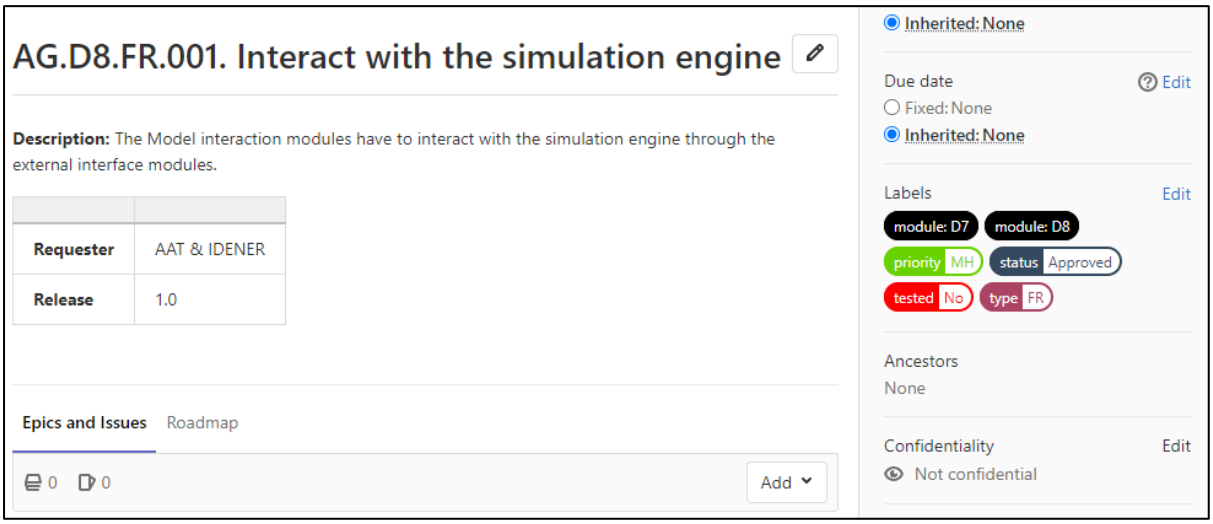

Figure 62. AG.D8.FR.001. Interact with the simulation engine.

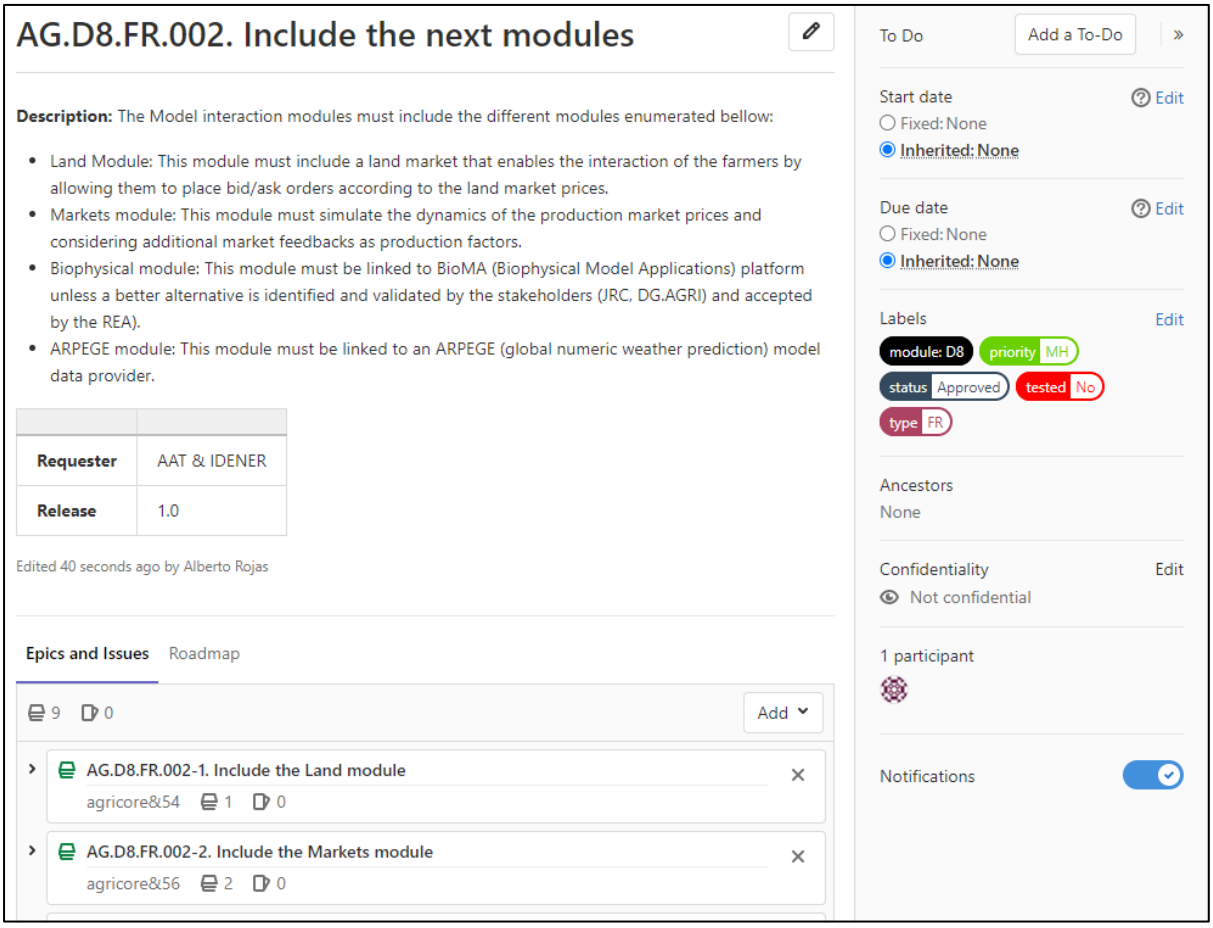

Figure 63. AG.D8.FR.002. Include the next modules.

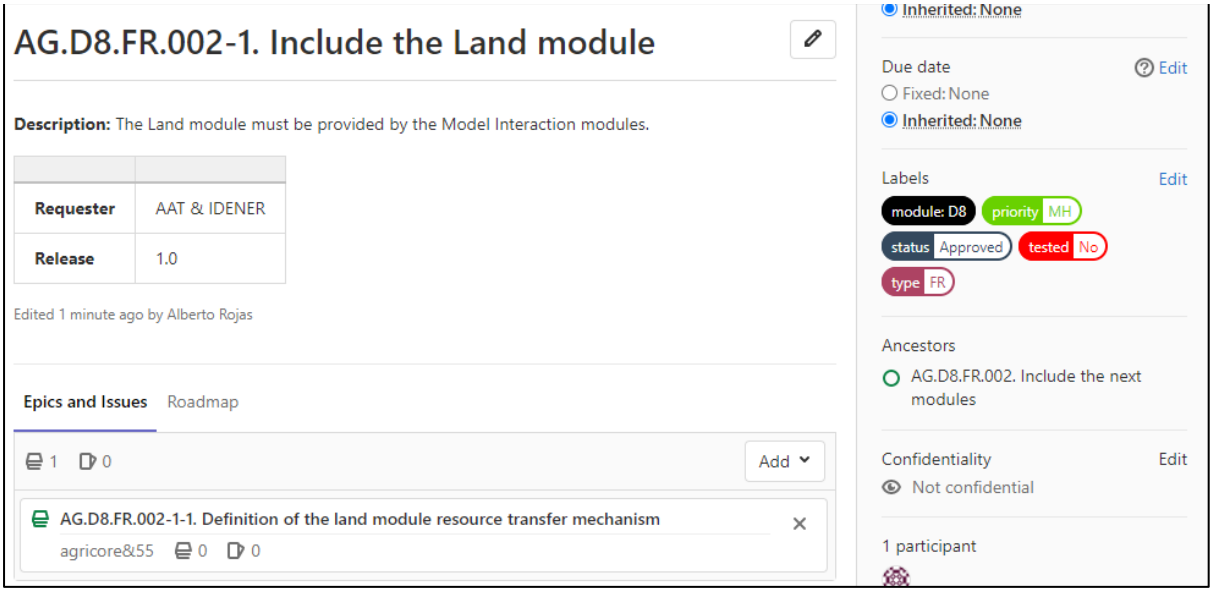

Figure 64. AG.D8.FR.002-1. Include the Land module.

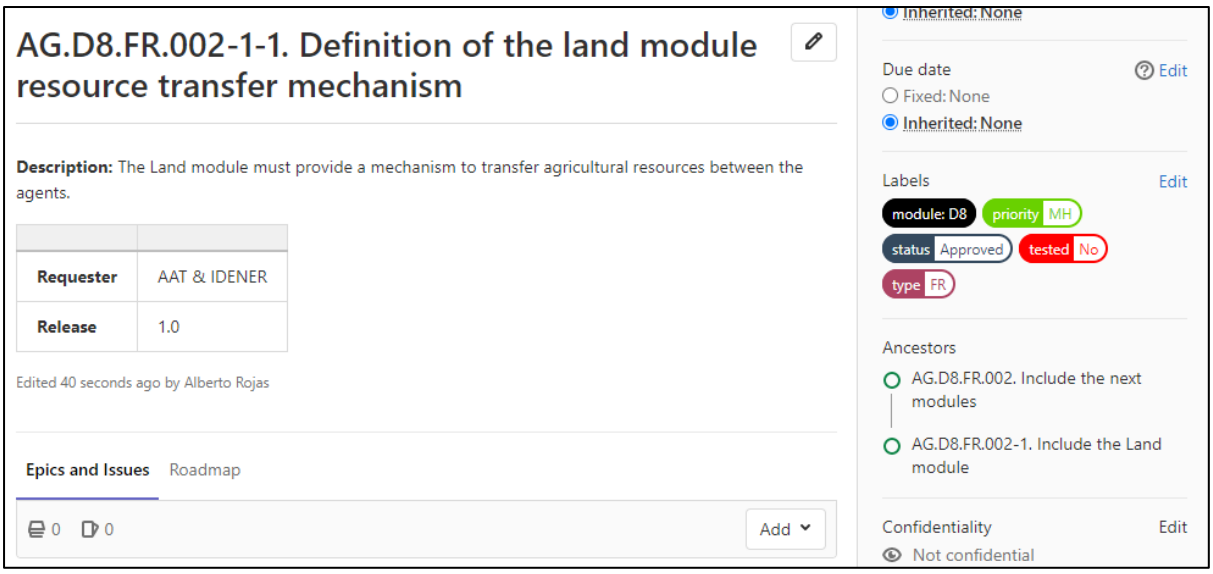

Figure 65. AG.D8.FR.002-1-1. Definition of the land module resource transfer mechanism.

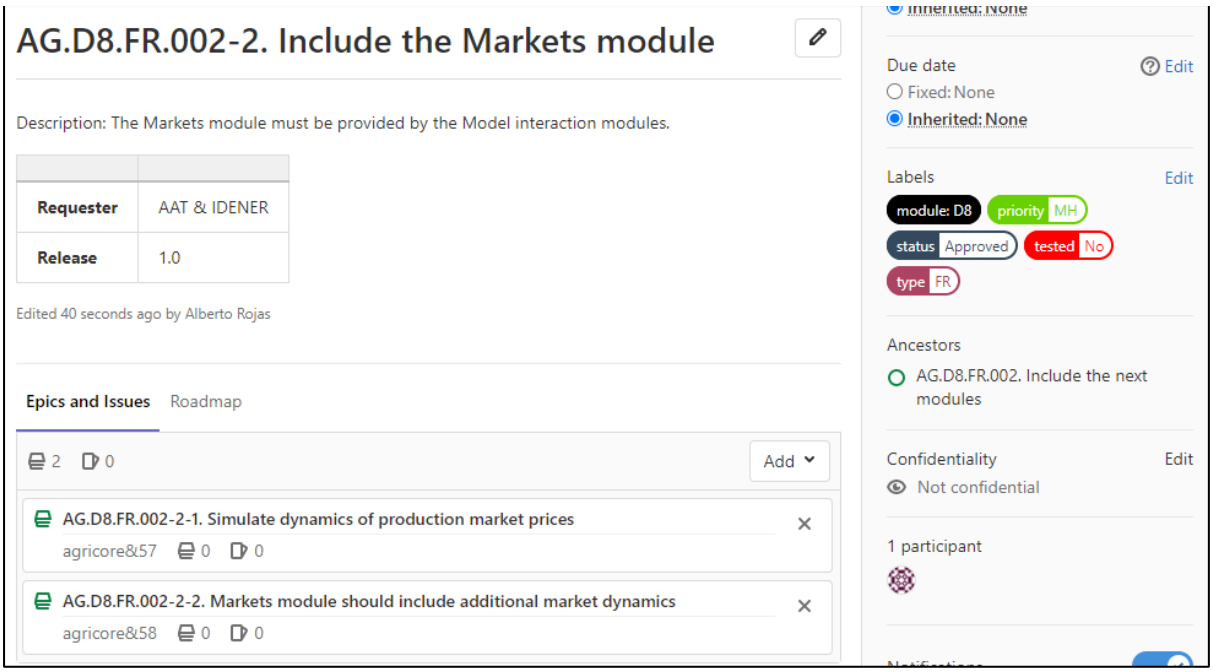

Figure 66. AG.D8.FR.002-2. Include the Markets module.

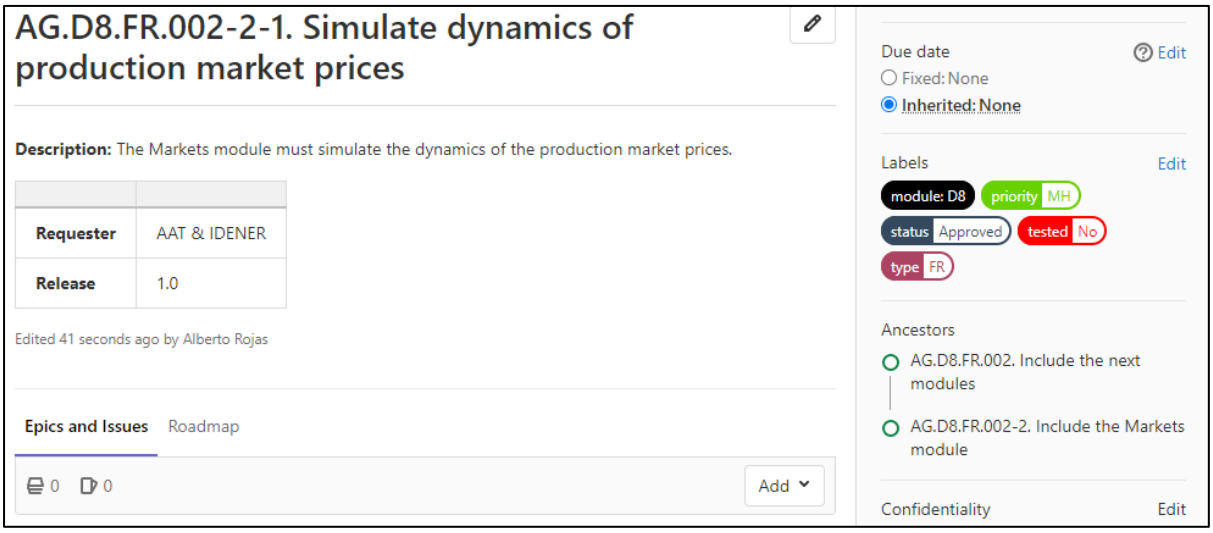

Figure 67. AG.D8.FR.002-2-1. Simulate dynamics of production market prices.

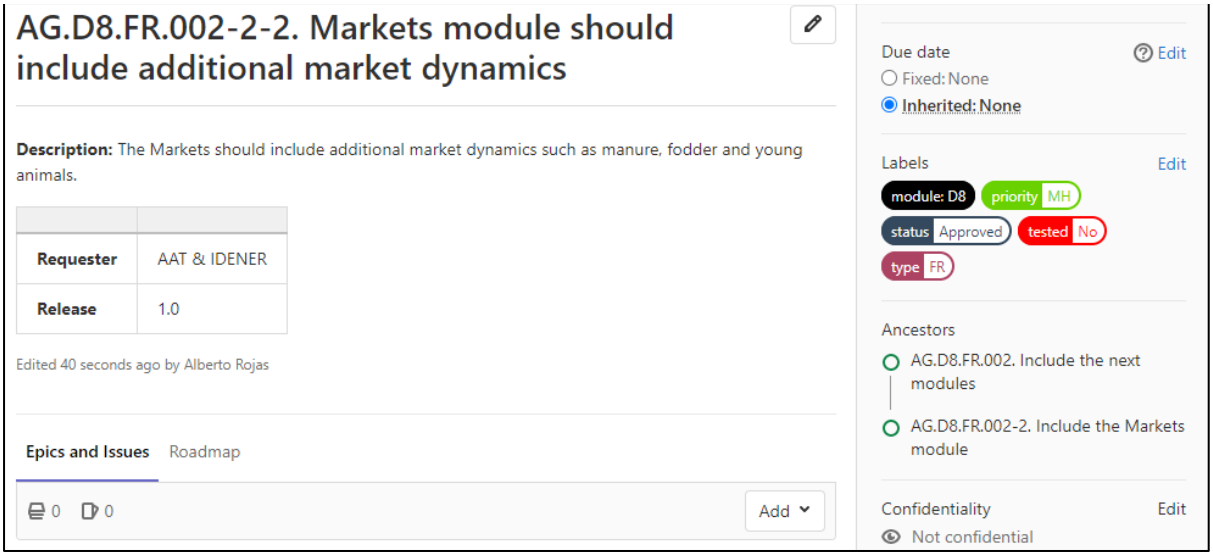

Figure 68. AG.D8.FR.002-2-2. Markets module should include additional market dynamics.

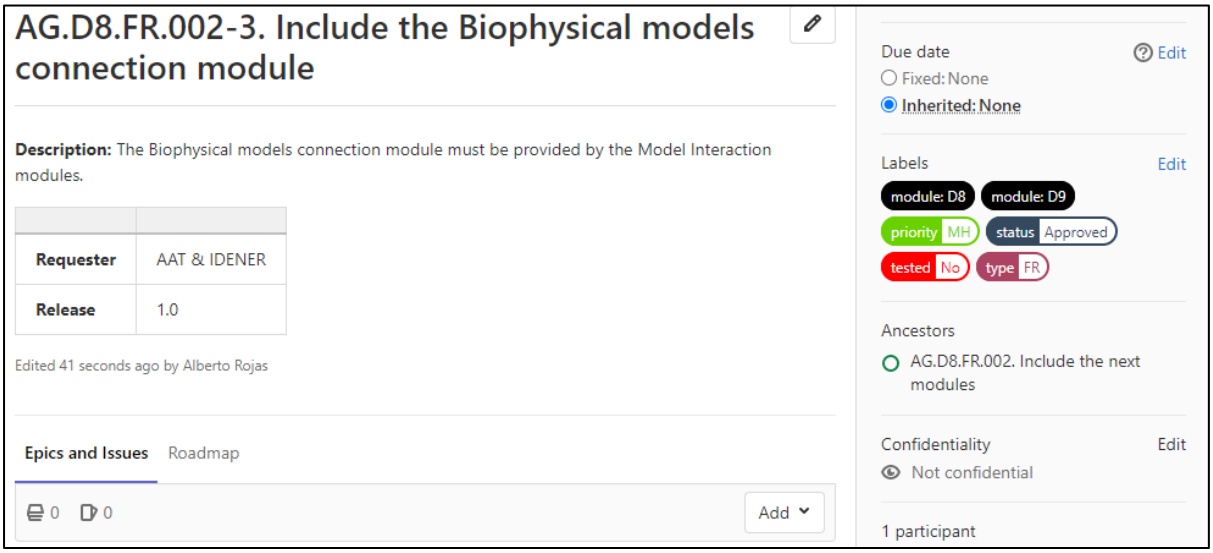

Figure 69. AG.D8.FR.002-3. Include the Biophysical models connection module.

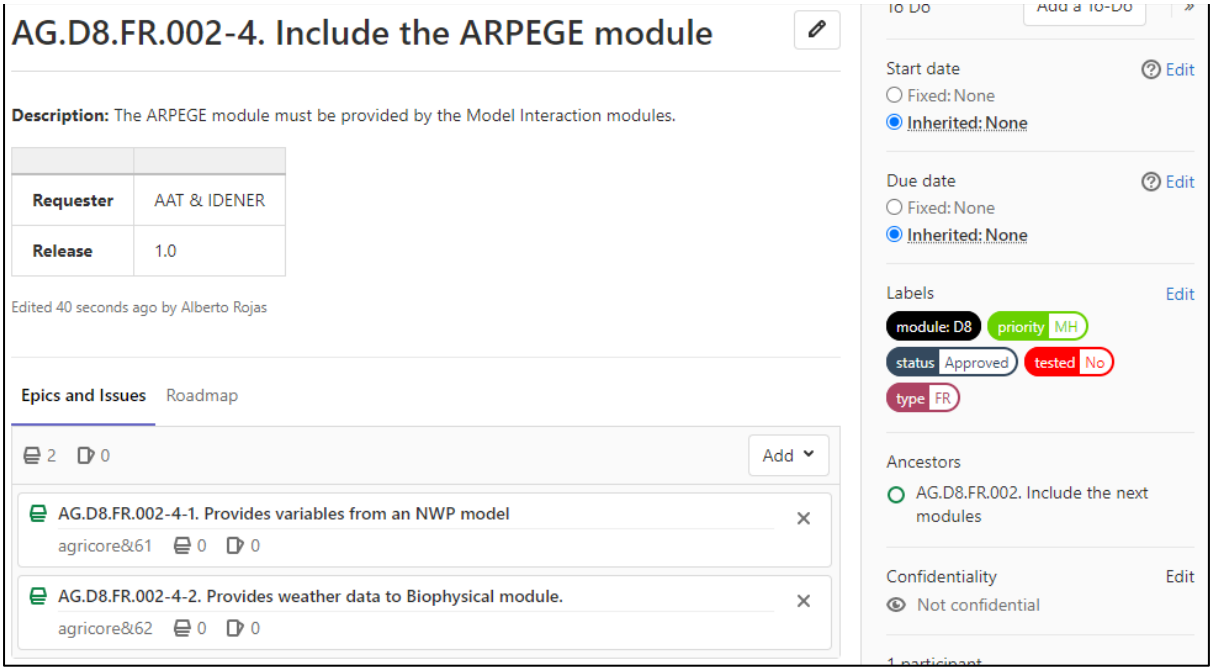

Figure 70. AG.D8.FR.002-4. Include the ARPEGE module.

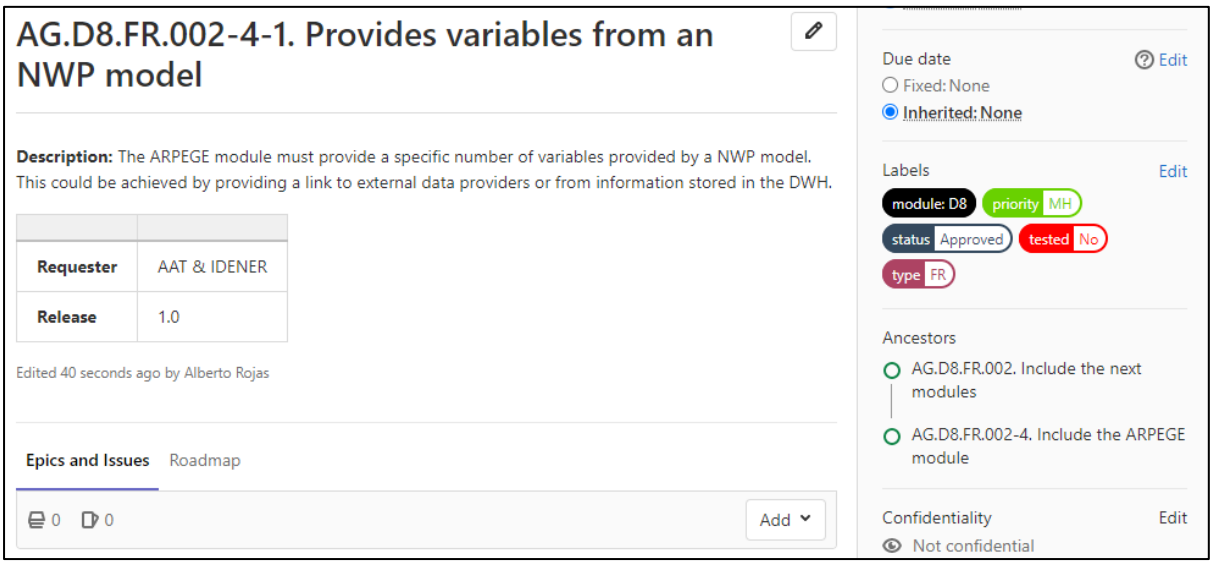

Figure 71. AG.D8.FR.002-4-1. Provides variables from an NWP model.

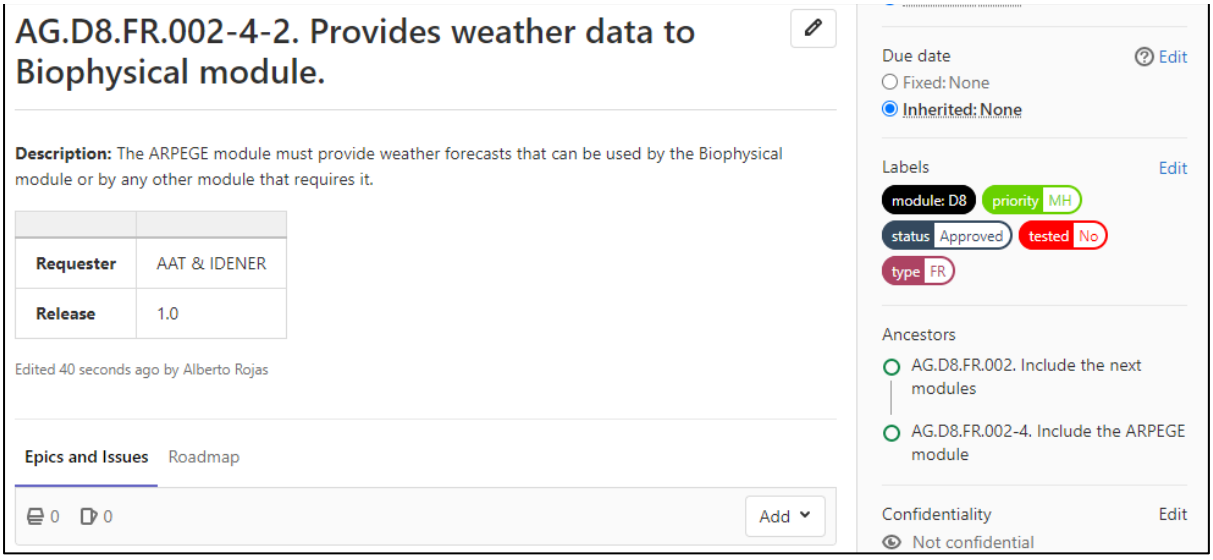

Figure 72. AG.D8.FR.002-4-2. Provides weather data to Biophysical module.

8.10 D9: Biophysical models connection module

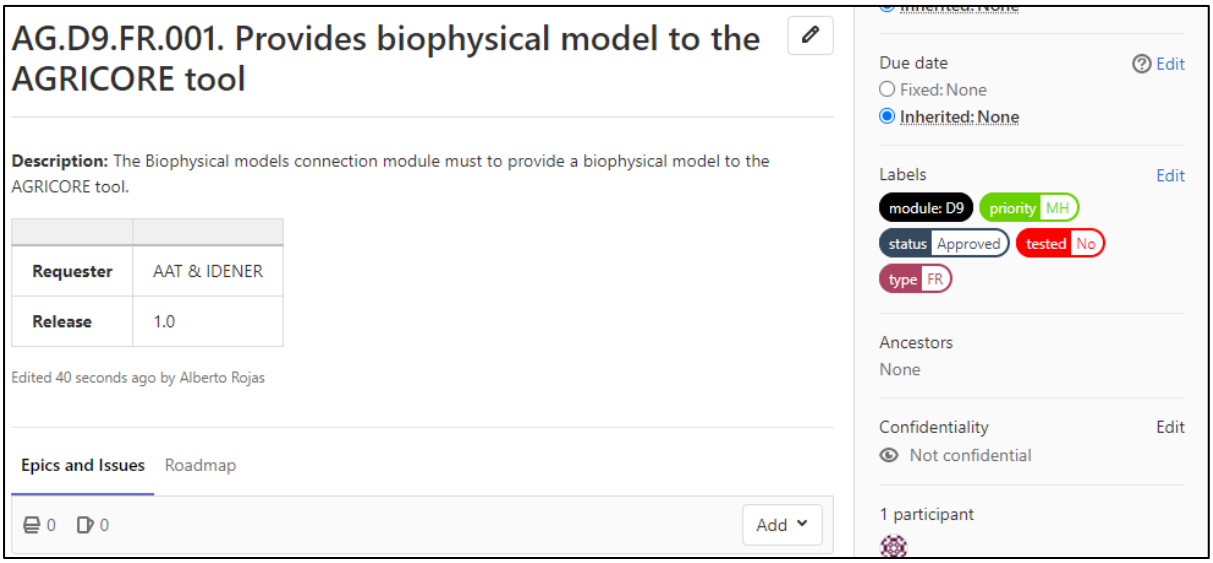

Figure 73. AG.D9.FR.001. Provides biophysical model to the AGRICORE tool.

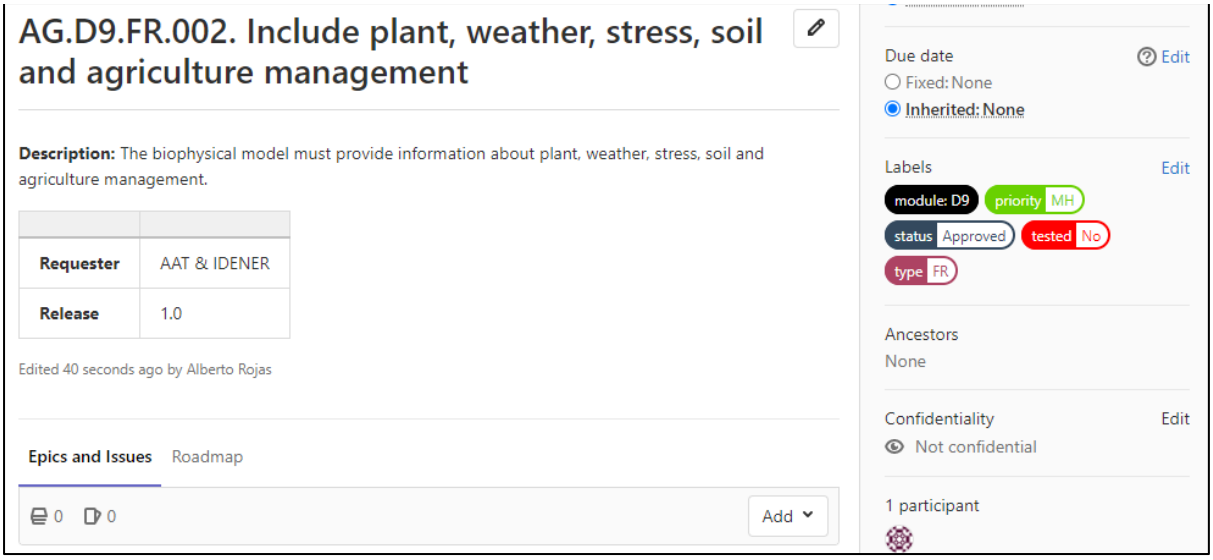

Figure 74. AG.D9.FR.002. Include plant, weather, stress, soil and agriculture management.

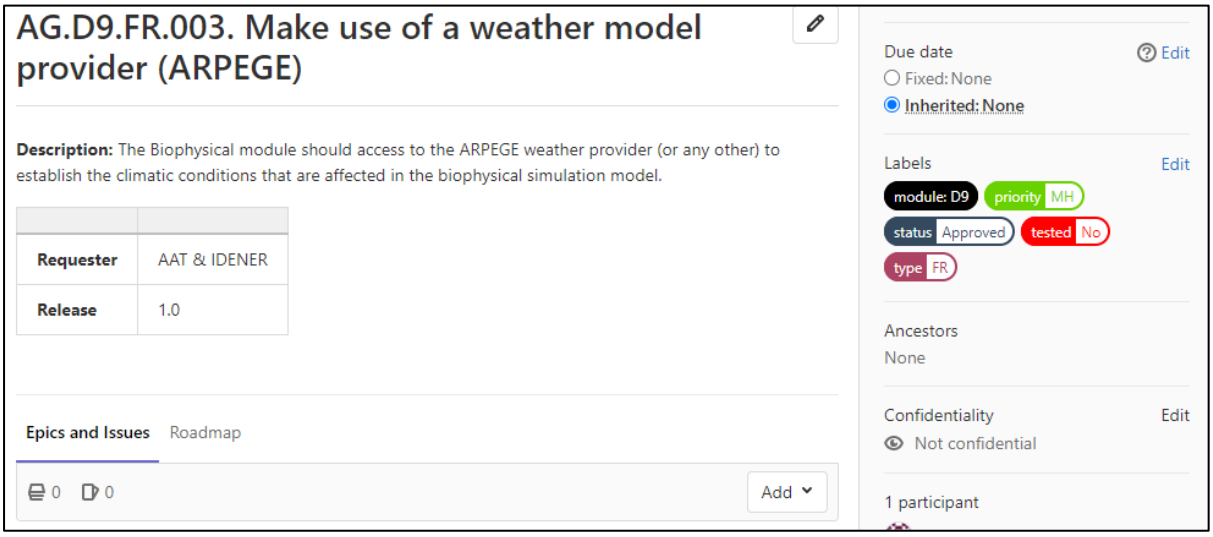

Figure 75. AG.D9.FR.003. Make use of a weather model provider (ARPEGE).

8.11 D10: Impact assessment module

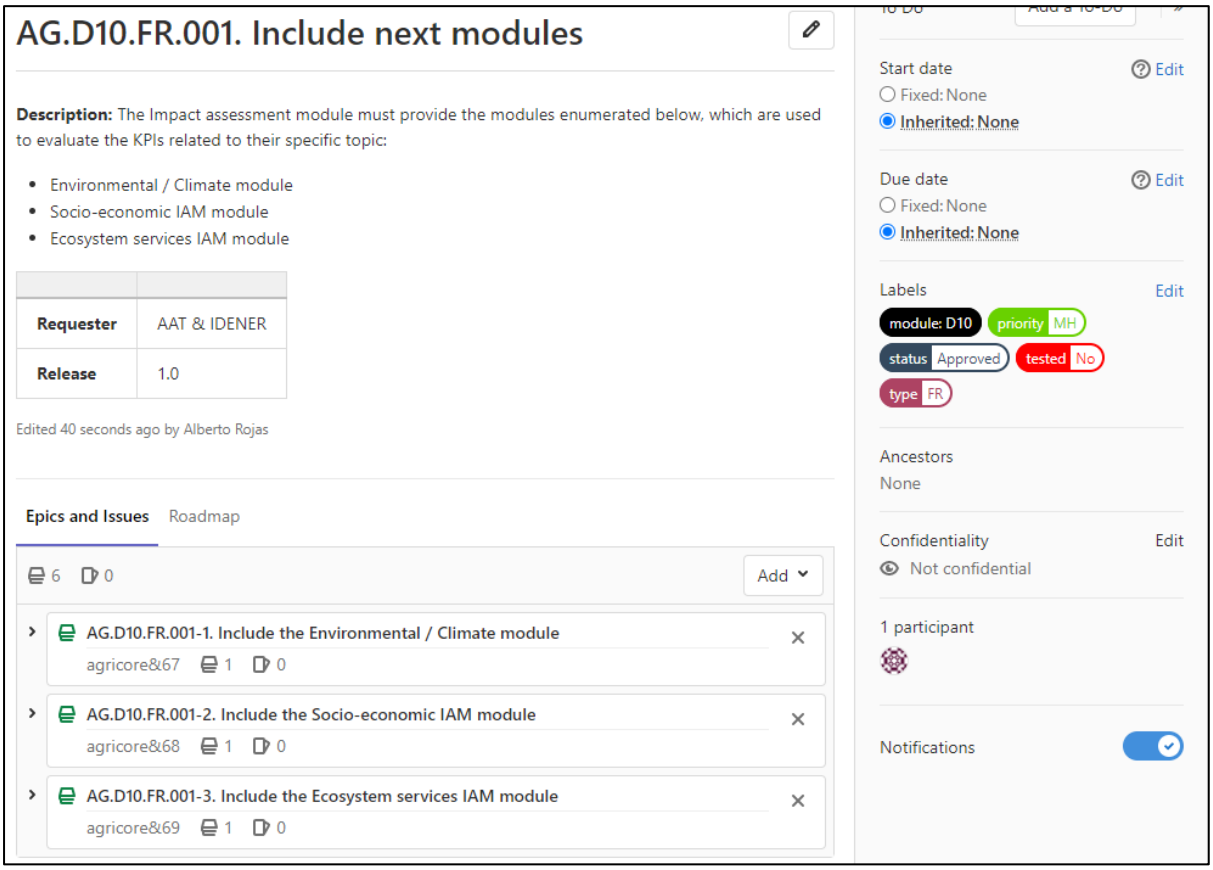

Figure 76. AG.D10.FR.001. Include next modules.

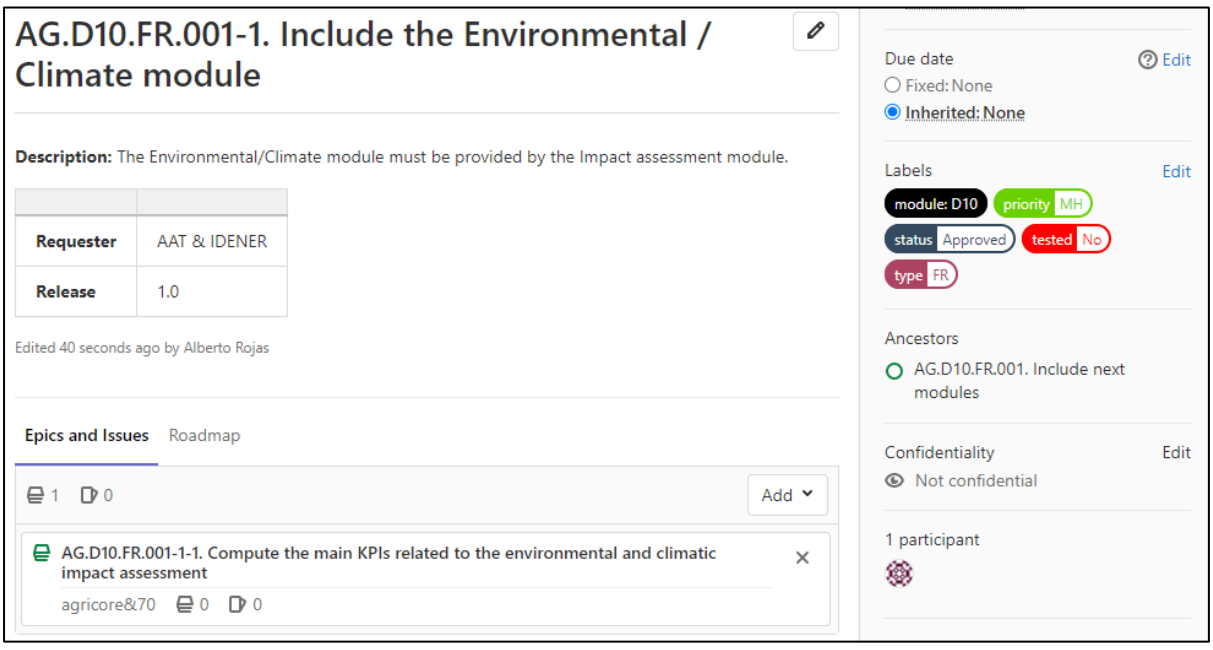

Figure 77. AG.D10.FR.001-1. Include the Environmental / Climate module.

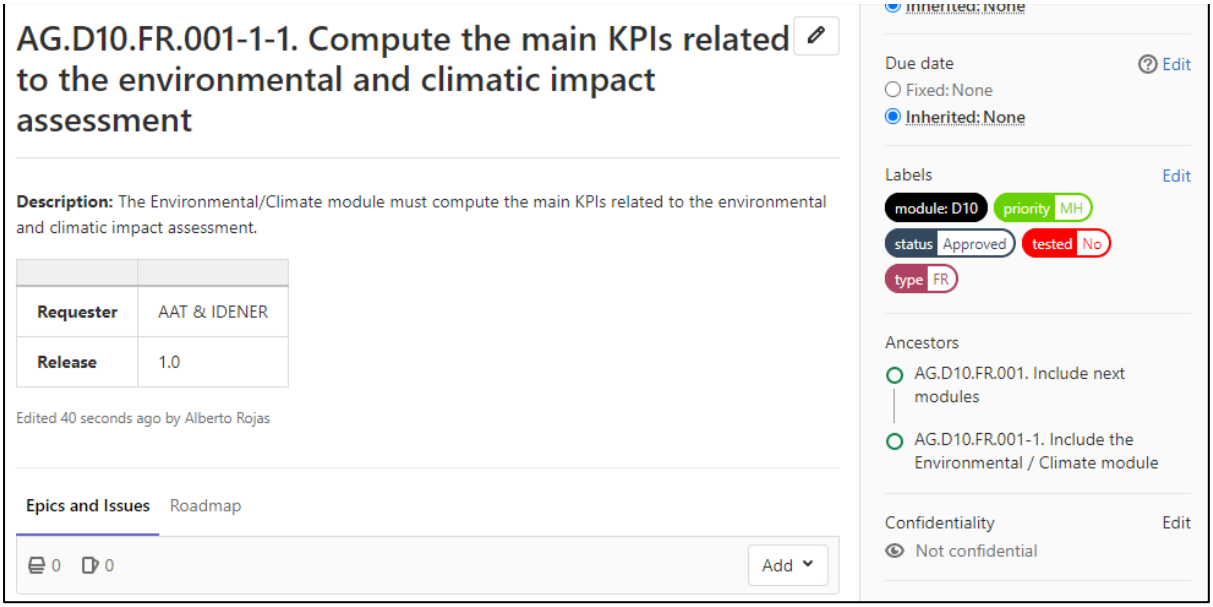

Figure 78. AG.D10.FR.001-1-1. Compute the main KPIs related to the environmental and climatic impact assessment.

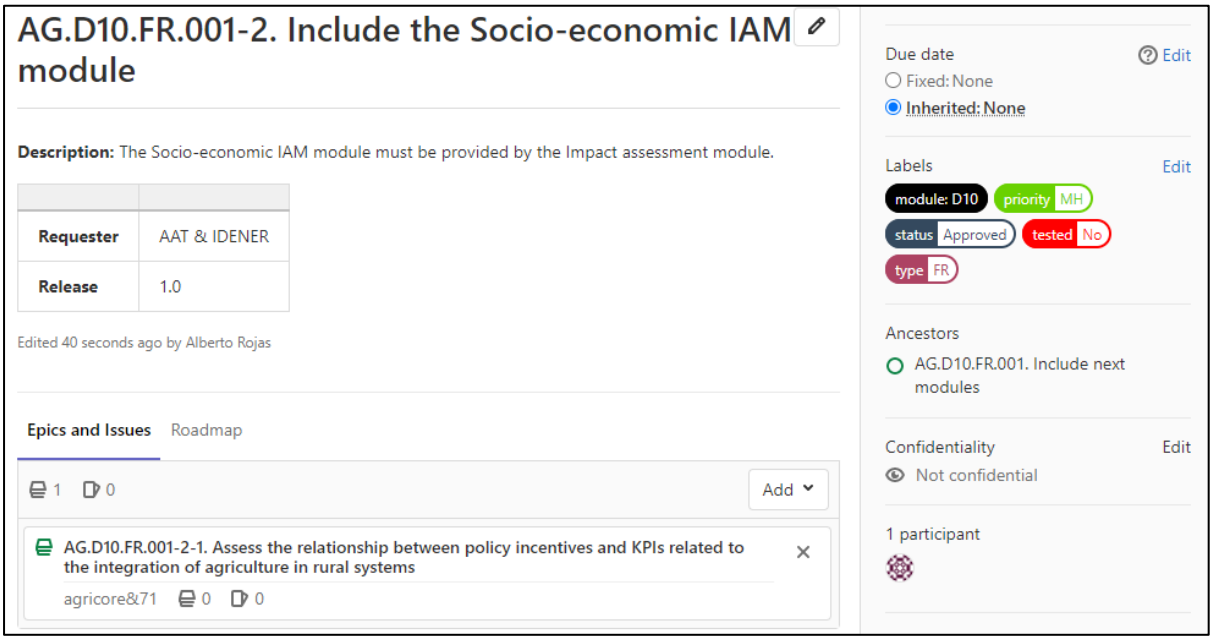

Figure 79. AG.D10.FR.001-2. Include the Socio-economic IAM module.

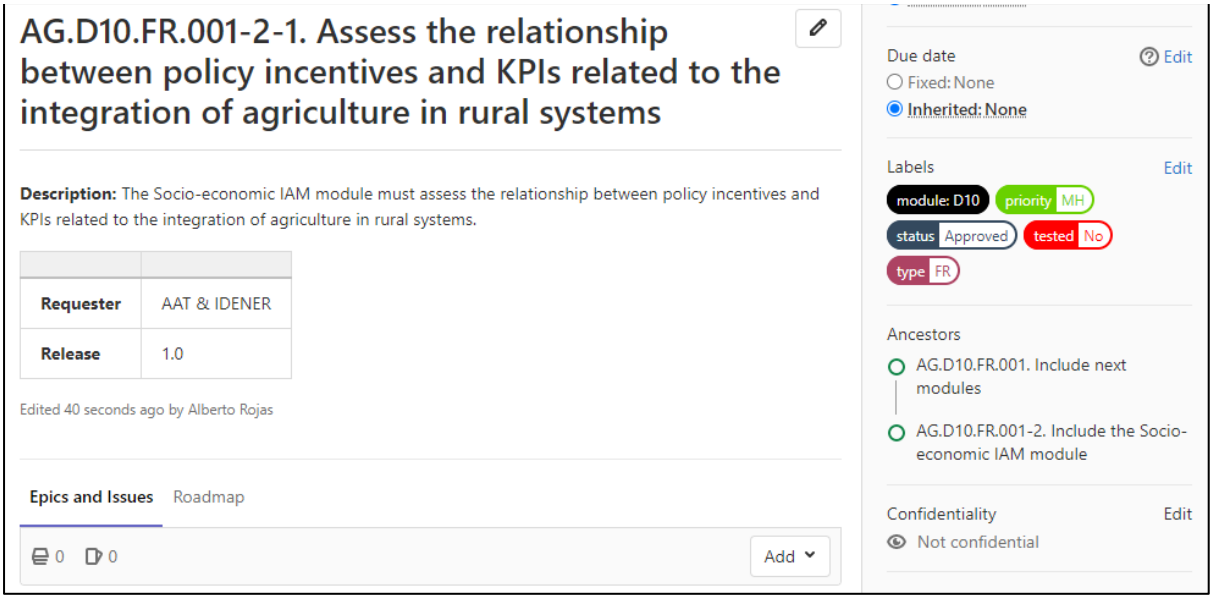

Figure 80. AG.D10.FR.001-2-1. Assess the relationship between policy incentives and KPIs related to the integration of agriculture in rural systems.

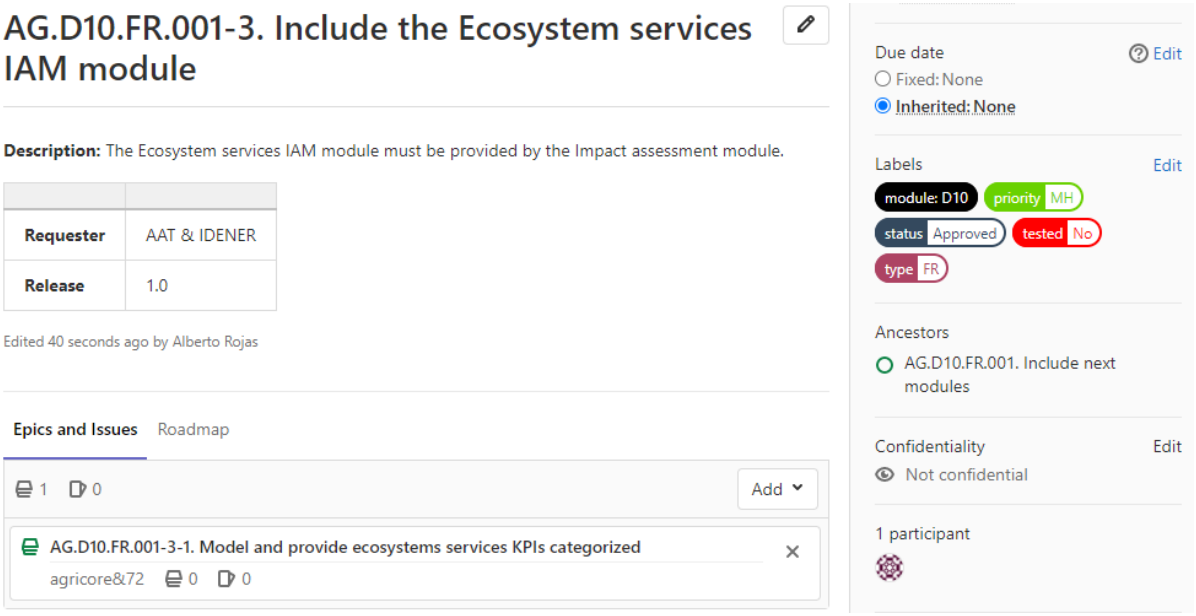

Figure 81. AG.D10.FR.001-3. Include the Ecosystem services IAM module.

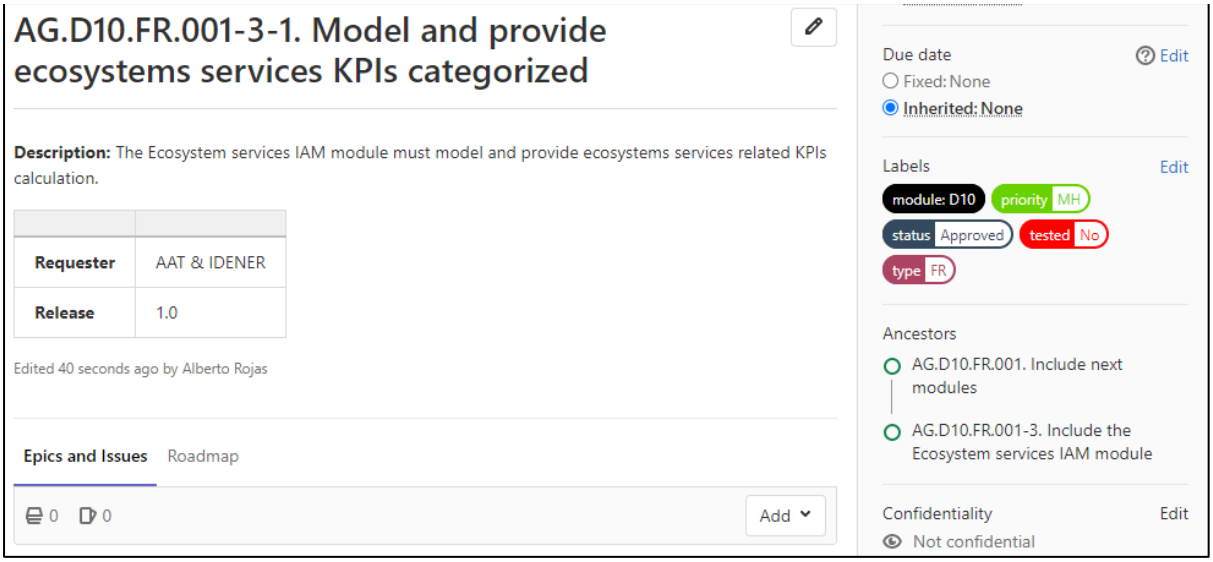

Figure 82. AG.D10.FR.001-3-1. Model and provide ecosystems services KPIs categorised.

8.12 D11: Policy environment module

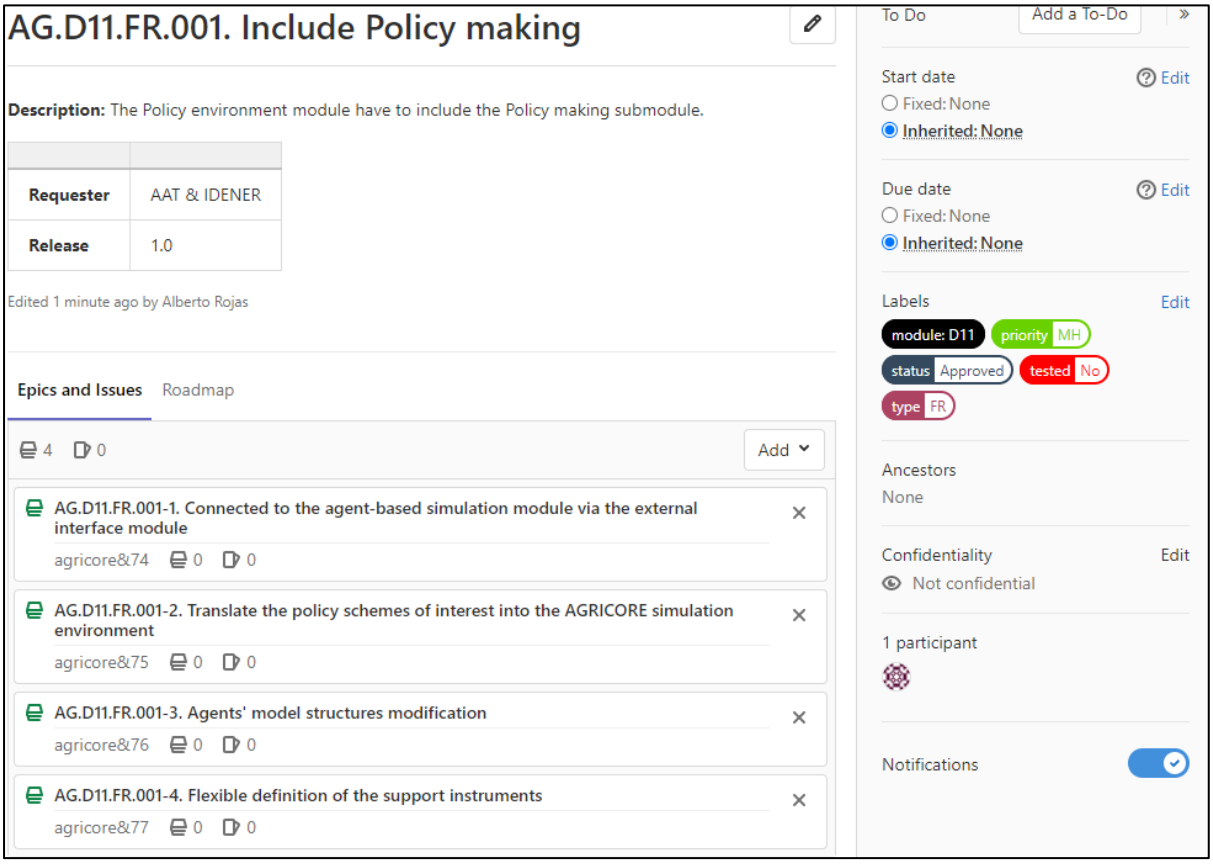

Figure 83. AG.D11.FR.001. Include Policy making.

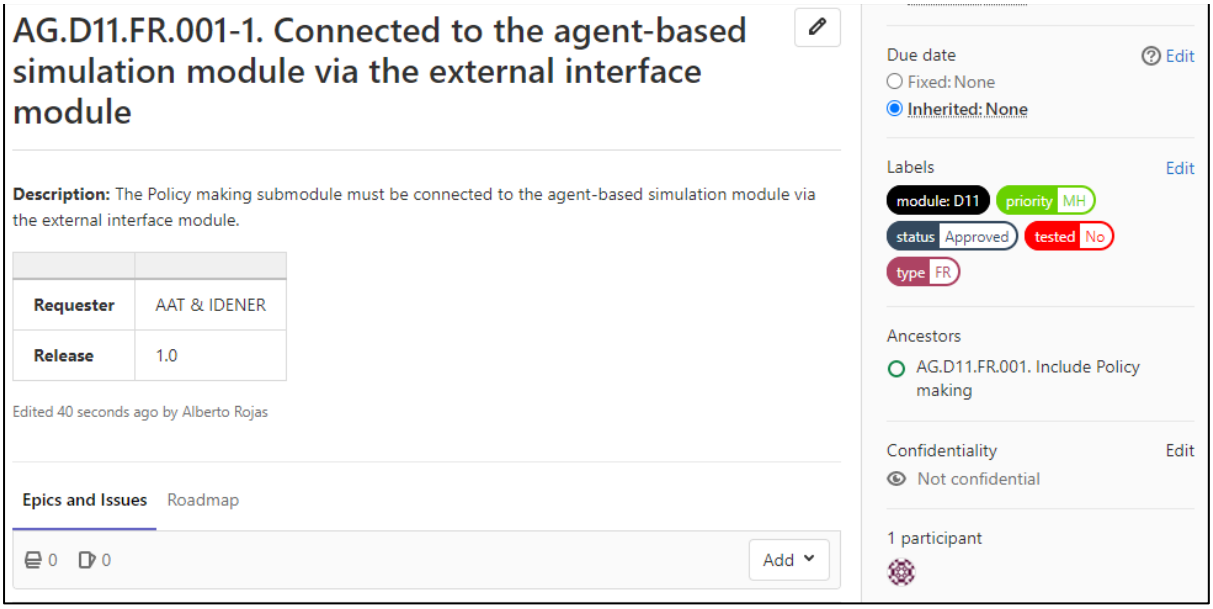

Figure 84. AG.D11.FR.001-1. Connected to the agent-based simulation module via the external interface module.

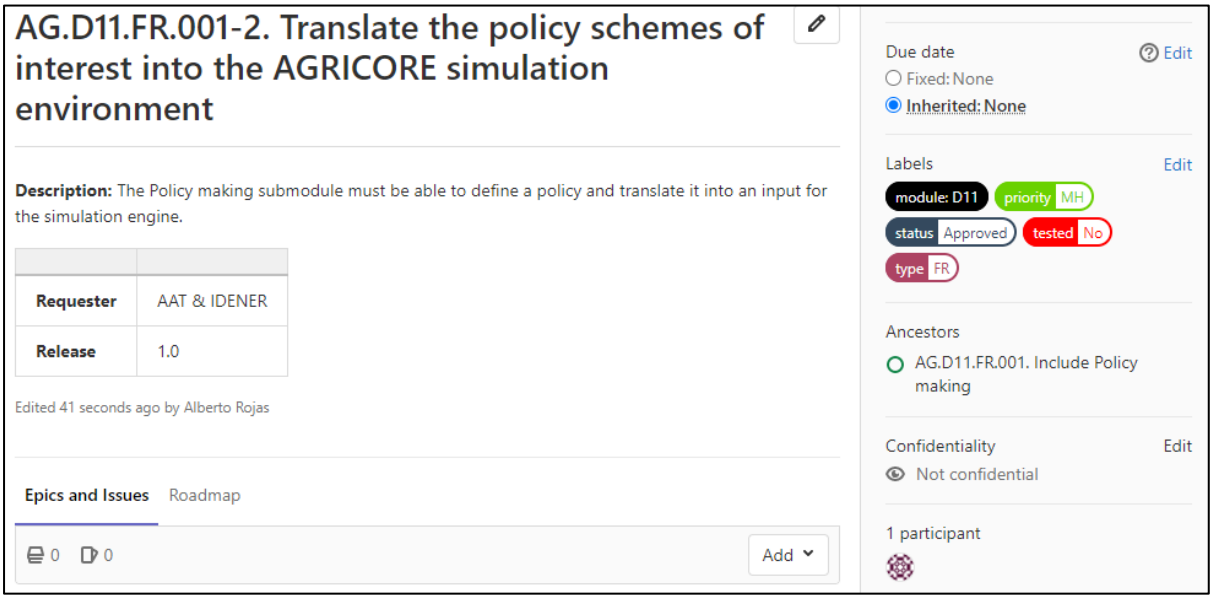

Figure 85. AG.D11.FR.001-2. Translate the policy schemes of interest into the AGRICORE simulation environment.

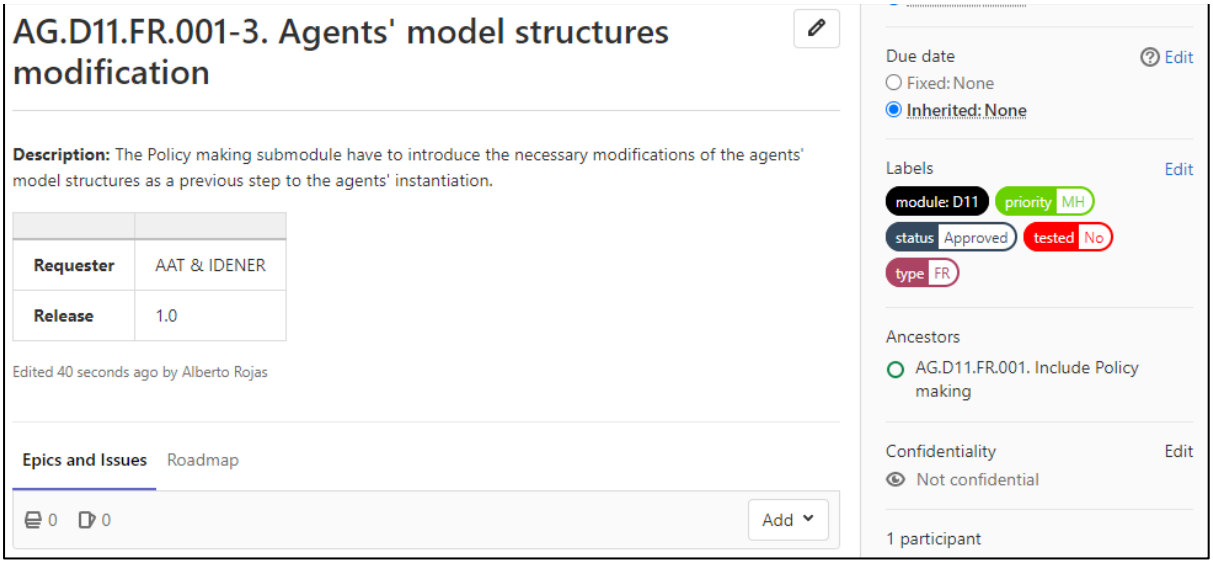

Figure 86 AG.D11.FR.001-3. Agents' model structures modification.

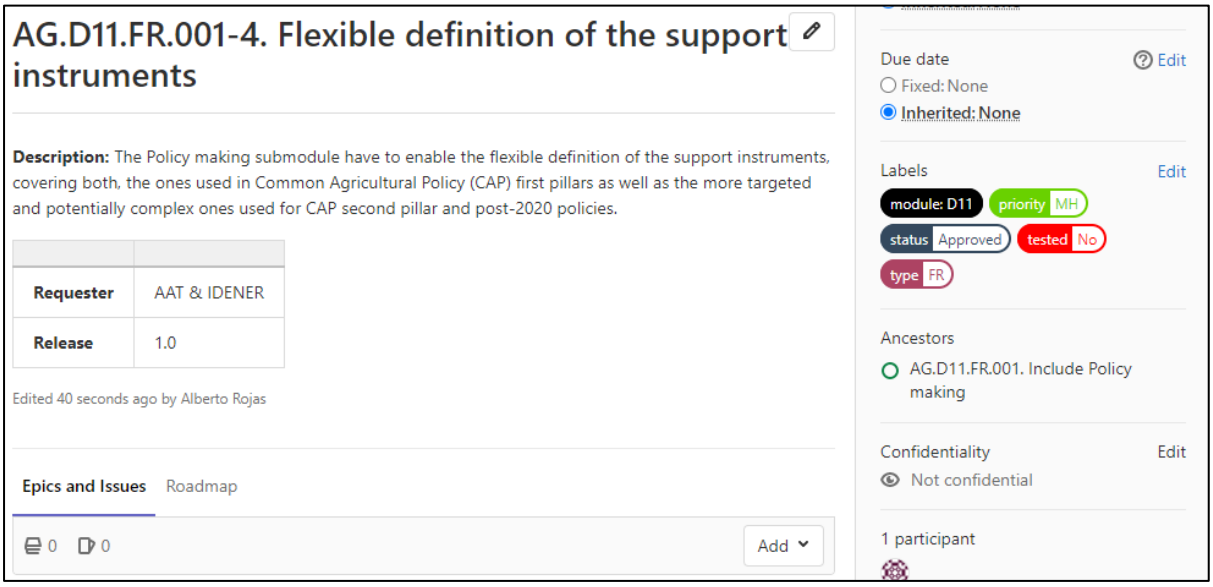

Figure 87. AG.D11.FR.001-4. Flexible definition of the support instruments.

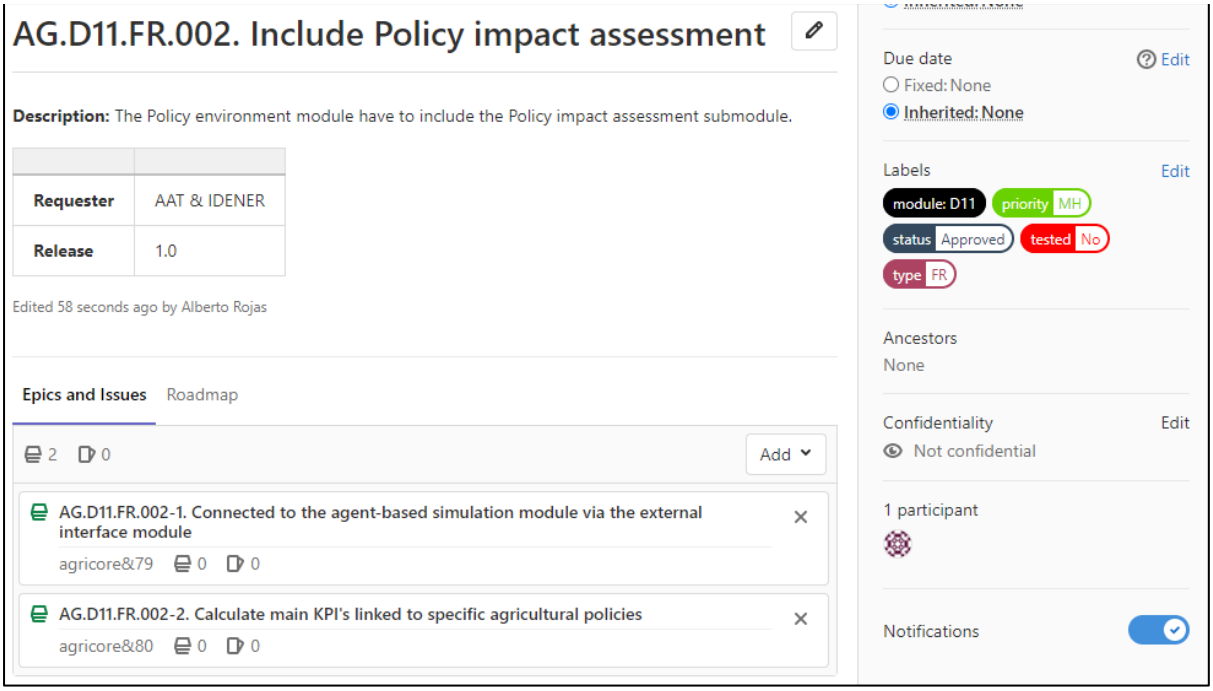

Figure 88. AG.D11.FR.002. Include Policy impact assessment.

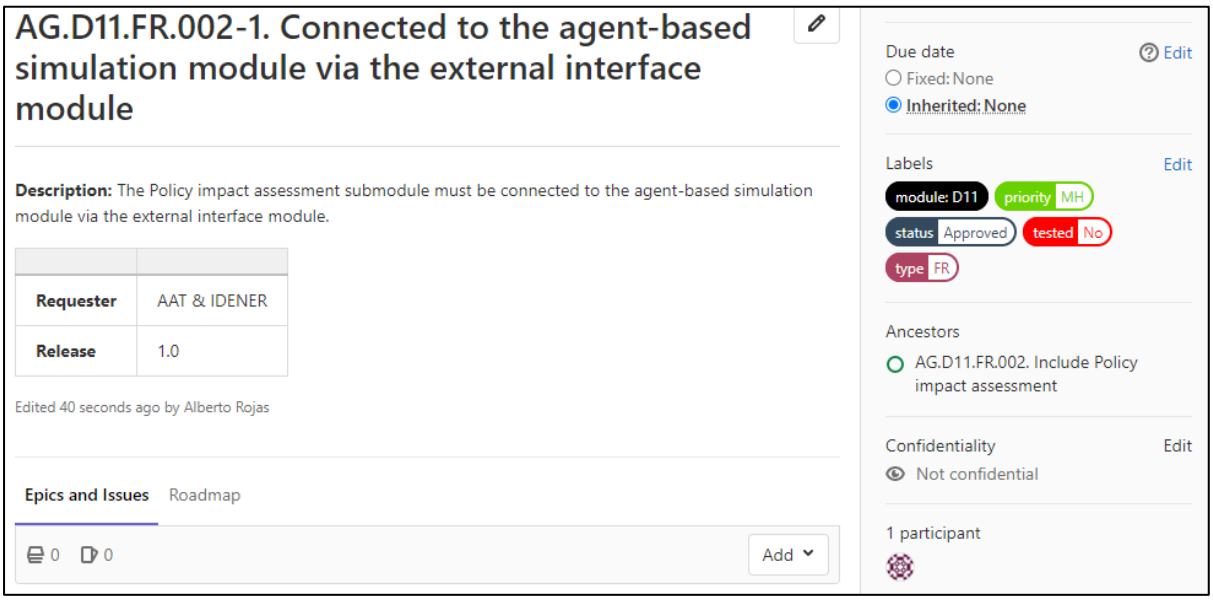

Figure 89. AG.D11.FR.002-1. Connected to the agent-based simulation module via the external interface module.

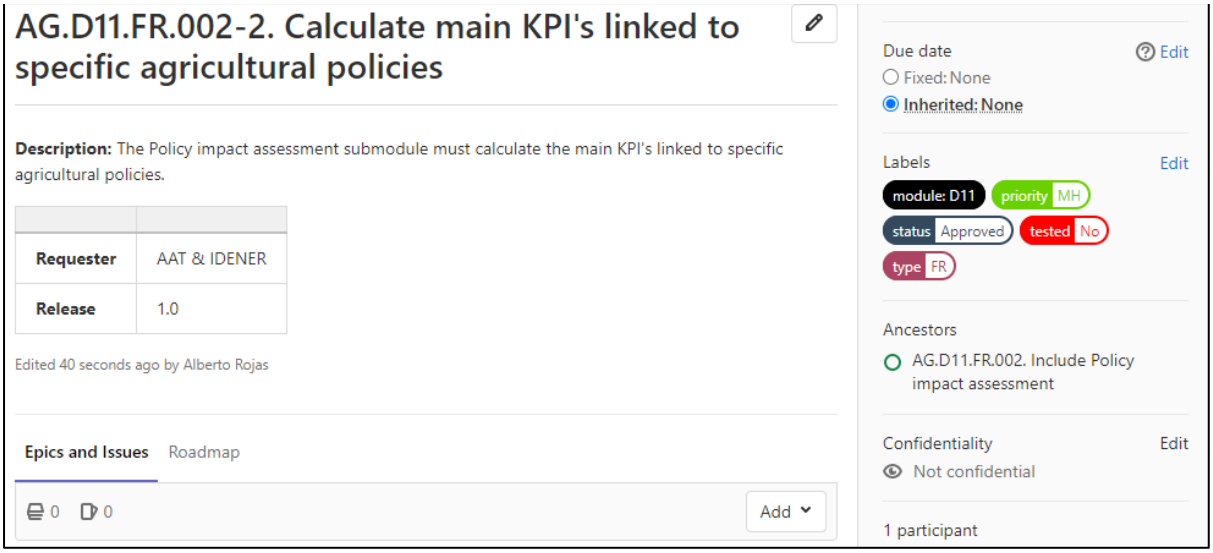

Figure 90. AG.D11.FR.002-2. Calculate main KPI's linked to specific agricultural policies.

8.13 D12: Agricore interface module

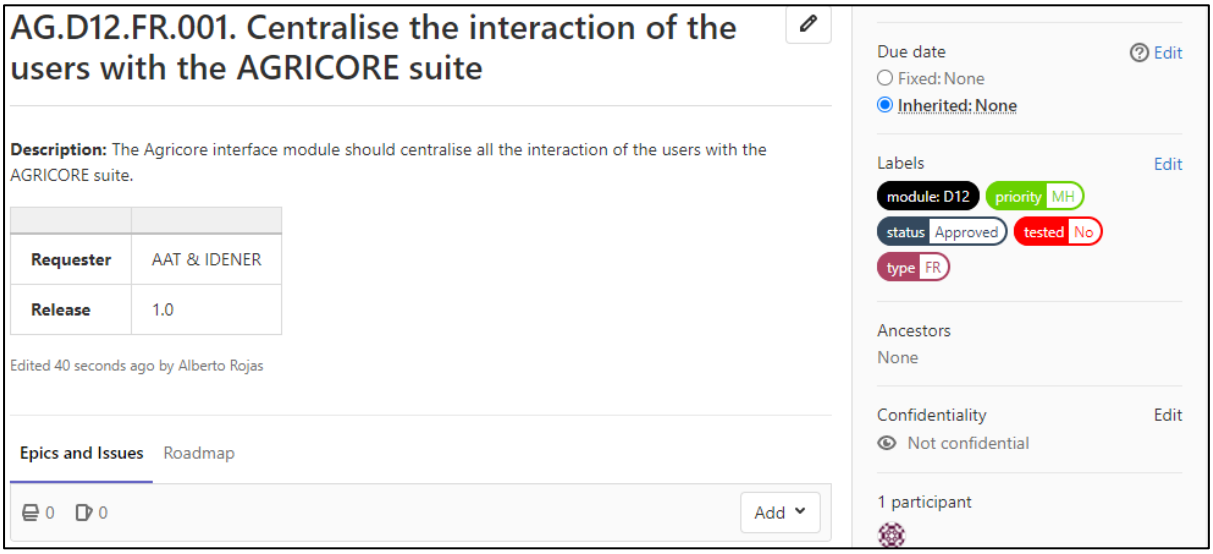

Figure 91. AG.D12.FR.001. Centralise the interaction of the users with the AGRICORE suite.

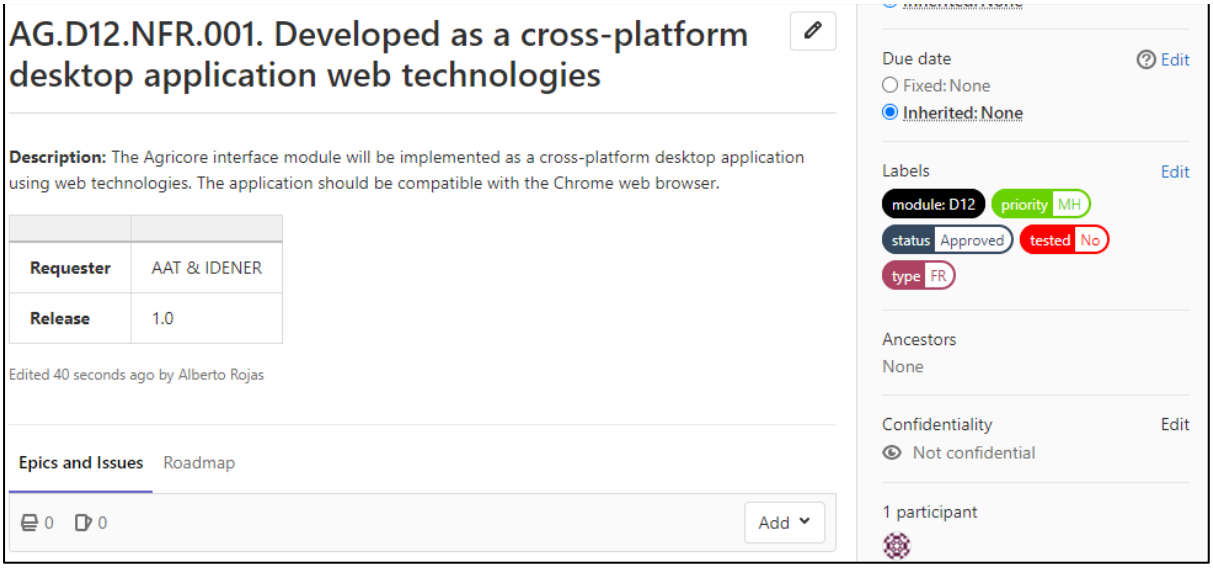

Figure 92. AG.D12.FR.002. Developed as a cross-platform desktop application web technologies.