

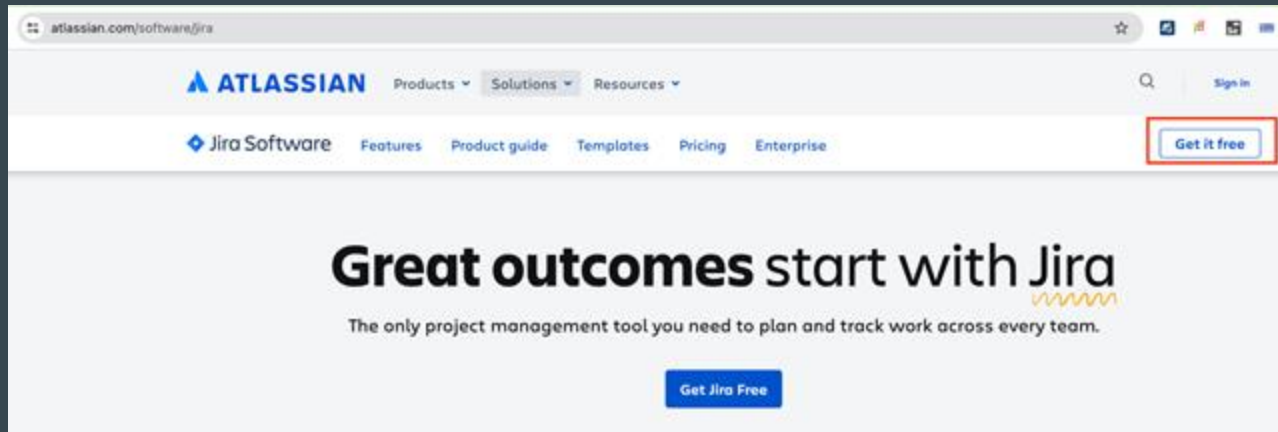
Jira and Xray cloud setup

...

Quick setup instructions of a Jira Software + Xray cloud instance

Jira cloud setup

<https://www.atlassian.com/software/jira>



Jira cloud signup

The image displays three overlapping screenshots of the Jira cloud signup process:

- Leftmost screenshot (Initial Sign-up):** Shows the Jira logo and the heading "Get started with". Below it, text states "It's free for up to 10 users — no credit c". A "Work email" field contains the text "testing.uncovered+batman@gmail.com". Below the field, text says "Find teammates, plus keep work and life separate b email." and "I agree to the [Atlassian Customer Agreement](#), which reference the [AI Product-Specific Terms](#), and ackno [Privacy Policy](#)." A blue "Sign up" button is at the bottom.
- Middle screenshot (Verification Code):** Shows the Jira logo and the heading "Your verification code is:". The code "513 137" is displayed. Below it, text says "If you didn't try signing up, you can safely ignore this em". At the bottom, it says "This message was sent to you by Atlassian Cloud" and the "ATLASSIAN" logo.
- Rightmost screenshot (Final Verification):** Shows the Jira logo and the heading "We've emailed you a code". Below it, text says "To complete your account setup, enter the code we've sent to:" followed by the email "testing.uncovered+batman@gmail.com". A numeric keypad is shown with buttons for 5, 1, 3, 1, 3, and a final button with a vertical line. A blue "Verify" button is at the bottom. At the very bottom, a link says "Didn't receive an email? Resend email".

Jira cloud signup

Take note of your Jira Cloud instance URL. You'll be asked some questions that you may skip.



Jira

Let's name your site

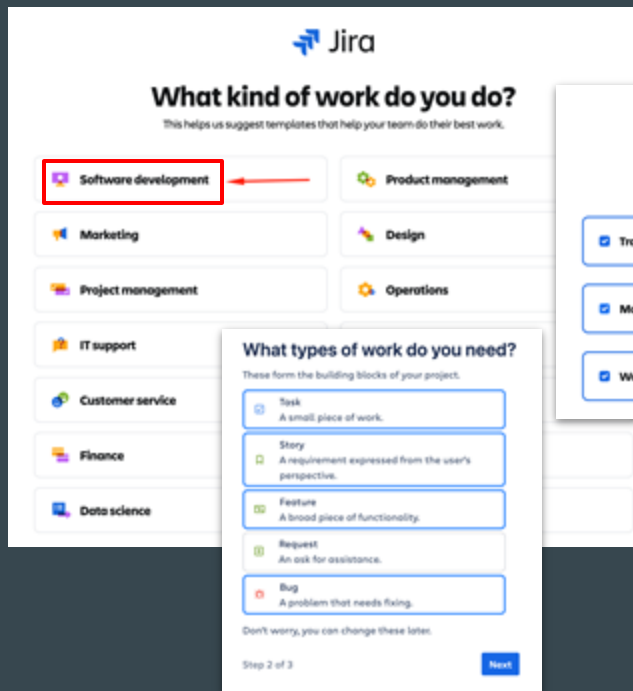
Your site name is part of your Jira URL. Most people use their team or company name.

Your site

testinguncoveredbatman.atlassian.net

This site name is just a suggestion. Feel free to change to something your team will recognize.

Continue



Jira

What kind of work do you do?

This helps us suggest templates that help your team do their best work.

Software development

Product management

Marketing

Design

Project management

Operations

IT support

Customer service

Finance

Data science

What types of work do you need?

These form the building blocks of your project.

☒ Task
A small piece of work.

☒ Story
A requirement expressed from the user's perspective.

☒ Feature
A broad piece of functionality.

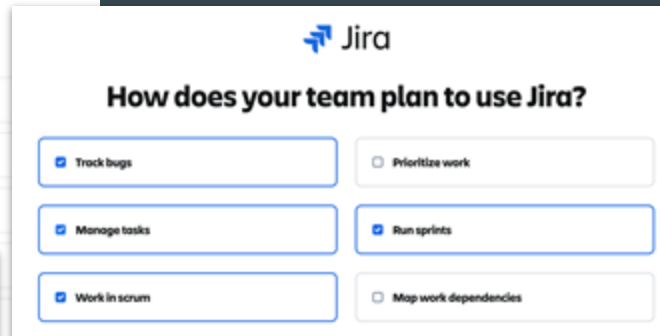
☒ Request
An ask for assistance.

☒ Bug
A problem that needs fixing.

Don't worry, you can change these later.

Step 2 of 3

Next



Jira

How does your team plan to use Jira?

☒ Track bugs

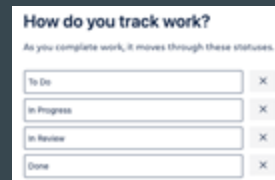
☐ Prioritize work

☒ Manage tasks

☒ Run sprints

☒ Work in scrum

☐ Map work dependencies



How do you track work?

As you complete work, it moves through these statuses.

To Do

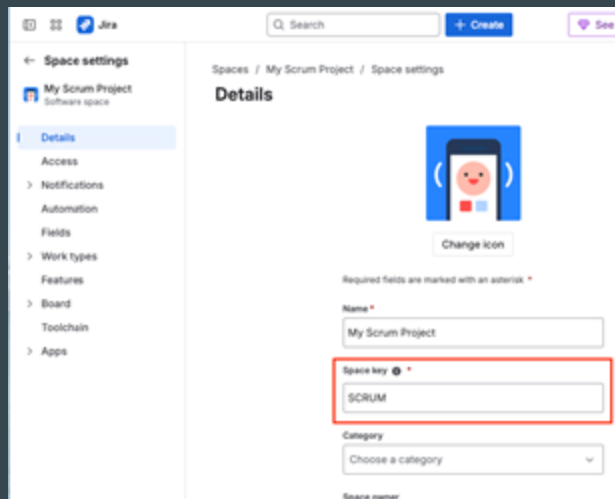
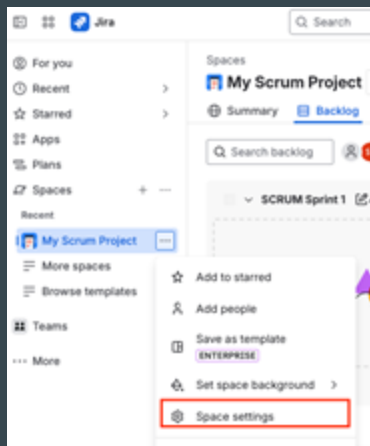
In Progress

In Review

Done

Project creation during signup

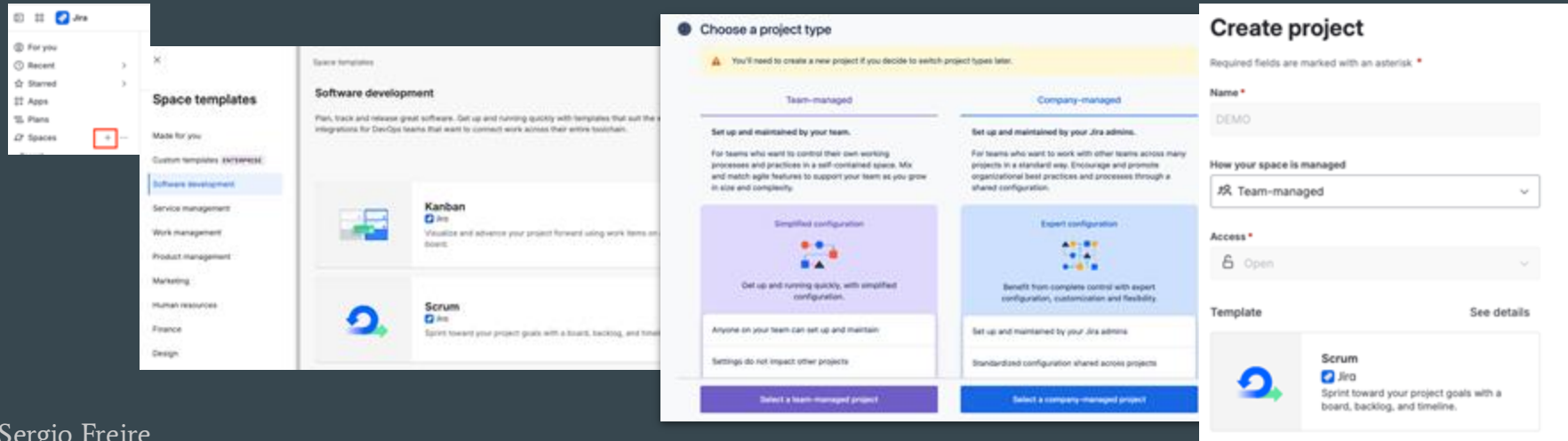
During the Jira cloud instance signup, you'll be asked to create your first project; this will create a *team-managed* project named “My Scrum Project”, having the key “SCRUM”; therefore, all issues on that project will have keys like SCRUM- $\langle \text{int} \rangle$ (e.g., “SCRUM-1”). You can change the project name and key on the project settings panel.



Creation of projects (i.e. “spaces”)

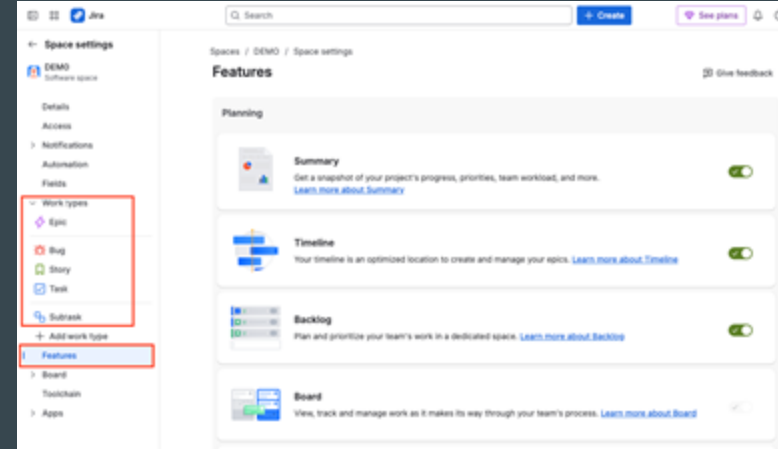
You can also create your Scrum based “space” later on, from the left “Spaces” menu entry. Have in mind though that there are 2 main project types in Jira Cloud:

- **Team-managed** (simpler but not so powerful; good enough for us)
- **Company-managed** (more complex and more powerful)



Final check

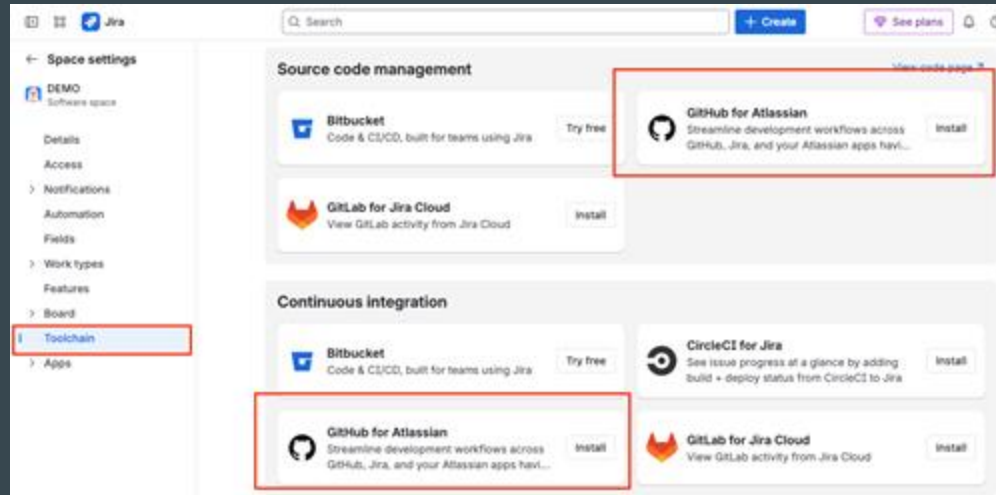
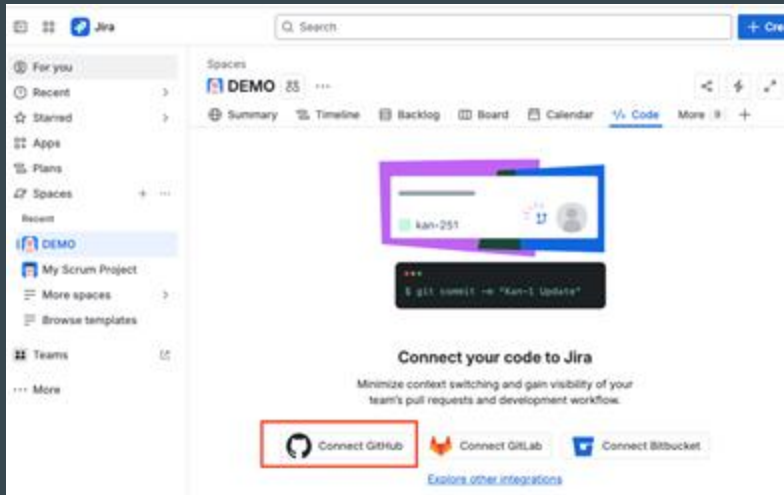
1. Open your space/project settings from the left bar, and go to **Features**. You should have all these, at least, enabled:
 - a. Summary, Backlog, Board, Reports, Sprints, Estimation, Code, Development
 - b. Optionally: Security, Deployments, Project pages
2. Check also that you have relevant “work types” (also known as “issue types”); Epic, Story, Byg and Task should be there.



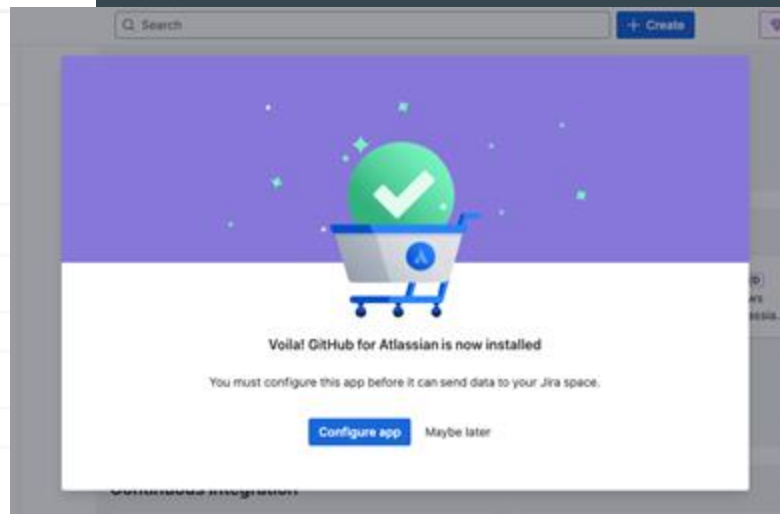
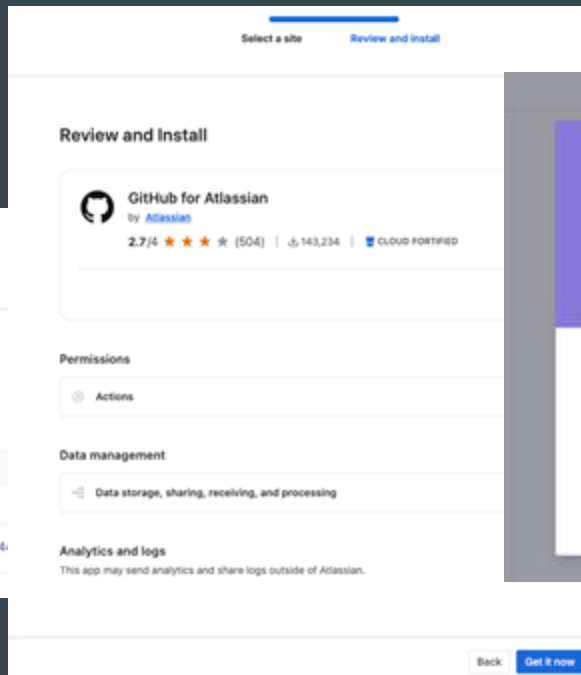
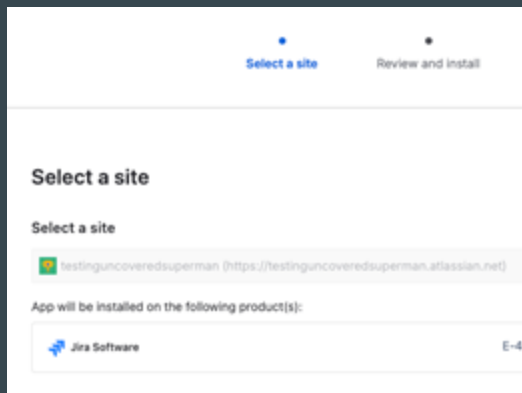
Integration with GitHub

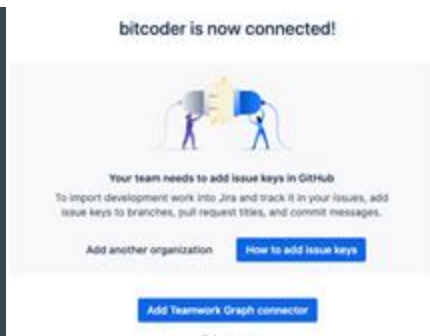
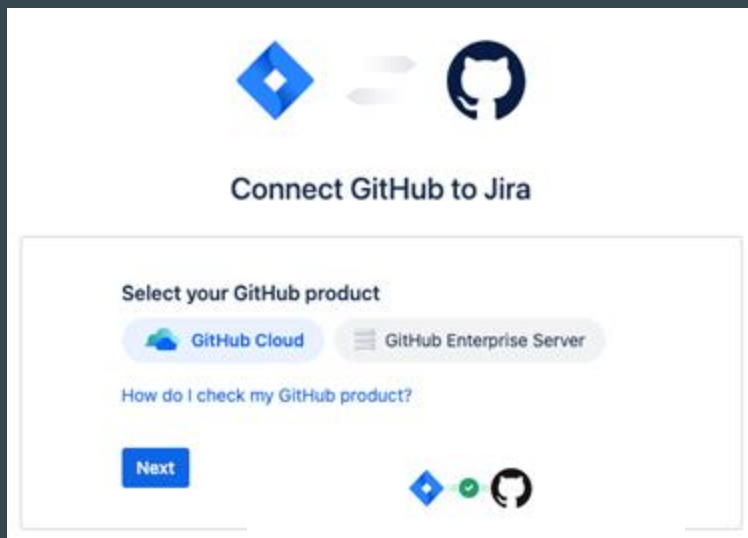
Installing “GitHub for Atlassian” integration

The installation of the integration with GitHub can be initiated in Jira from several places, namely the Code tab. It can also be triggered from the Space settings > Toolchain section.

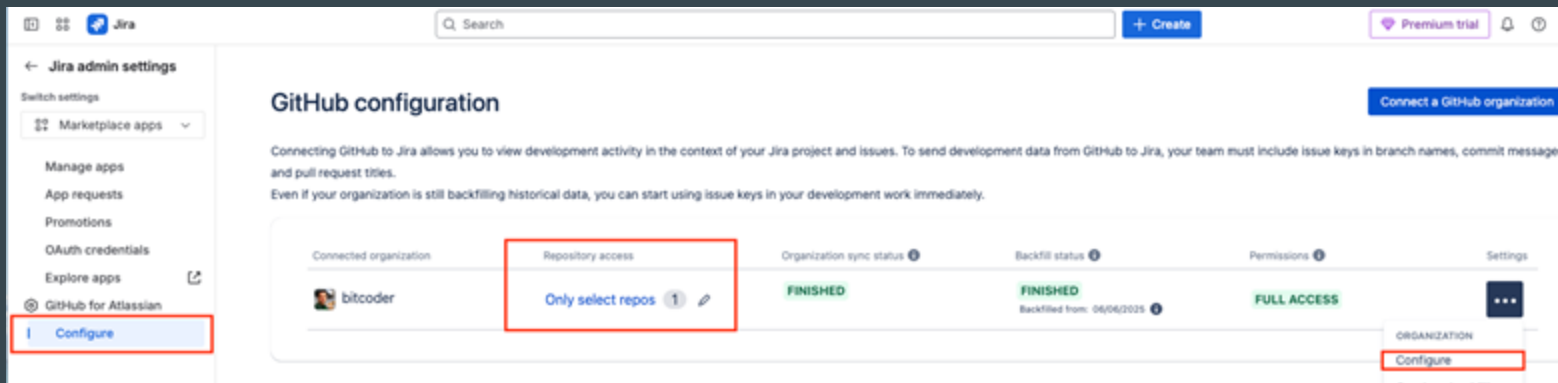
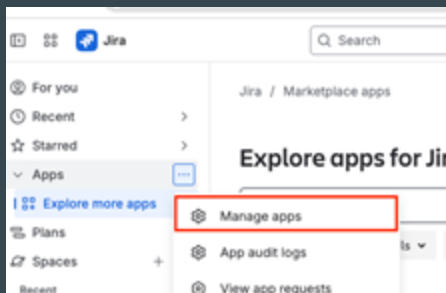


“GitHub for Atlassian”

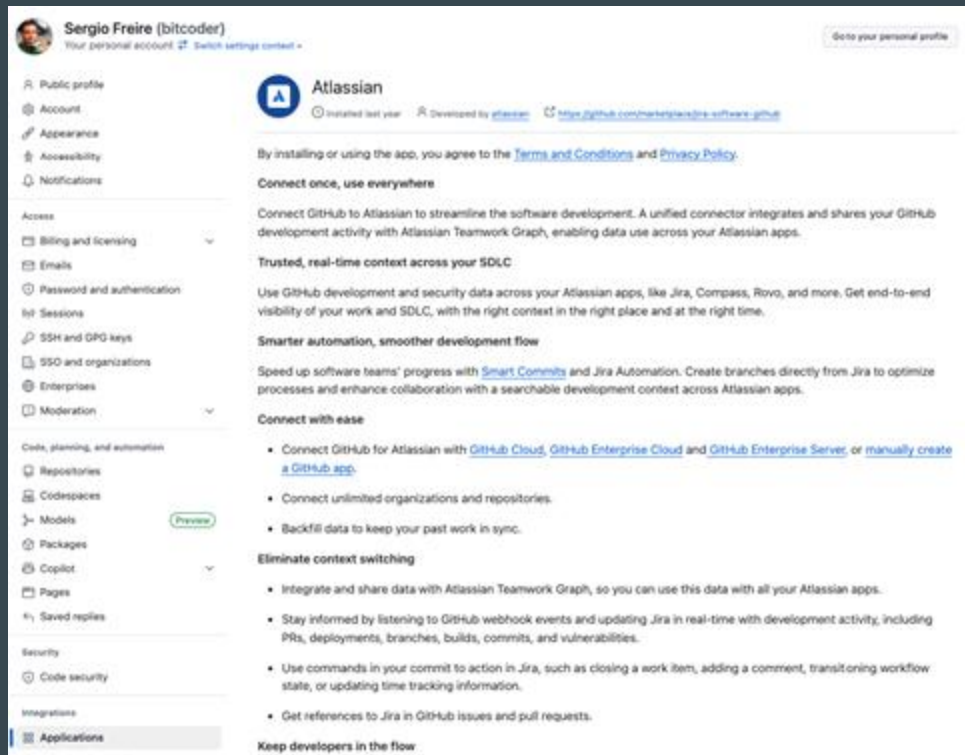




Finishing the configuration to give access to certain repositories



Check configuration on GitHub side...



The screenshot shows the GitHub profile page for Sergio Freire (bitcoder). The left sidebar contains navigation links: Public profile, Account, Appearance, Accessibility, Notifications, Access, Billing and licensing, Emails, Password and authentication, Sessions, SSH and GPG keys, SSO and organizations, Enterprises, Moderation, Code, planning, and automation, Repositories, Codespaces, Models, Packages, Copilot, Pages, Saved replies, Security, Code security, Integrations, and Applications. The main content area displays the Atlassian integration setup. It includes the Atlassian logo, a link to the GitHub Marketplace, and a description of the integration. The integration is described as a unified connector that integrates and shares your GitHub development activity with Atlassian Teamwork Graph, enabling data use across your Atlassian apps. It highlights features like connecting once, using everywhere; trusted, real-time context across your SDLC; smarter automation, smoother development flow; and connecting with ease. The 'Connect with ease' section lists three options: connecting with GitHub Cloud, GitHub Enterprise Cloud, or GitHub Enterprise Server, or manually creating a GitHub app. The 'Eliminate context switching' section lists three benefits: integrating and sharing data with Atlassian Teamwork Graph, staying informed by listening to GitHub webhook events and updating Jira in real-time with development activity, and using commands in your commit to action in Jira. The 'Keep developers in the flow' section is also visible.

Sergio Freire (bitcoder)
Your personal account [Switch settings context](#) [Go to your personal profile](#)

Atlassian
Installed last year · Developed by [atlassian](#) [View on GitHub Marketplace](#)

By installing or using the app, you agree to the [Terms and Conditions](#) and [Privacy Policy](#).

Connect once, use everywhere

Connect GitHub to Atlassian to streamline the software development. A unified connector integrates and shares your GitHub development activity with Atlassian Teamwork Graph, enabling data use across your Atlassian apps.

Trusted, real-time context across your SDLC

Use GitHub development and security data across your Atlassian apps, like Jira, Compass, Rovo, and more. Get end-to-end visibility of your work and SDLC, with the right context in the right place and at the right time.

Smarter automation, smoother development flow

Speed up software teams' progress with [Smart Commits](#) and Jira Automation. Create branches directly from Jira to optimize processes and enhance collaboration with a searchable development context across Atlassian apps.

Connect with ease

- Connect GitHub for Atlassian with [GitHub Cloud](#), [GitHub Enterprise Cloud](#) and [GitHub Enterprise Server](#), or [manually create a GitHub app](#).
- Connect unlimited organizations and repositories.
- Backfill data to keep your past work in sync.

Eliminate context switching

- Integrate and share data with Atlassian Teamwork Graph, so you can use this data with all your Atlassian apps.
- Stay informed by listening to GitHub webhook events and updating Jira in real-time with development activity, including PRs, deployments, branches, builds, commits, and vulnerabilities.
- Use commands in your commit to action in Jira, such as closing a work item, adding a comment, transitioning workflow state, or updating time tracking information.
- Get references to Jira in GitHub issues and pull requests.

Keep developers in the flow



The screenshot shows the 'Permissions' dialog for the Atlassian integration. It has two sections: 'Permissions' and 'Repository access'. The 'Permissions' section has two checked items: 'Read access to Dependabot alerts, actions, administration, metadata, secret scanning alerts, and security events' and 'Read and write access to code, deployments, issues, and pull requests'. The 'Repository access' section has two radio buttons: 'All repositories' (unchecked) and 'Only select repositories' (checked). Below the 'Only select repositories' option, it says 'Select at least one repository. Also includes public repositories (read-only)'. There is a 'Select repositories' button. Below that, it says 'Selected 1 repository:' and shows a list with 'bitcoder/tutorial-spring' and an 'X' icon to remove it. At the bottom, there are 'Save' and 'Cancel' buttons.

Permissions

- ☒ Read access to Dependabot alerts, actions, administration, metadata, secret scanning alerts, and security events
- ☒ Read and write access to code, deployments, issues, and pull requests

Repository access

☐ All repositories
This applies to all current and future repositories owned by the resource owner. Also includes public repositories (read-only).

☒ Only select repositories
Select at least one repository. Also includes public repositories (read-only).

[Select repositories](#)

Selected 1 repository:

- [bitcoder/tutorial-spring](#) X

[Save](#) [Cancel](#)

Reference work items in your development spaces

To reference Jira work items while committing, building, and deploying code with Bitbucket, GitHub, or other supported developer tools:

1. Find the key for the Jira work item you want to link to, for example “JRA-123”. You can find the key in several places in Jira:
 - On the board, work item keys appear at the bottom of a card.
 - On the work item’s details, keys appear in the breadcrumb navigation at the top of the page.

[Find out about work item keys.](#)
2. Check out a new branch in your repo, using the key in the branch name. For example, `git checkout -b JRA-123-<branch-name>`.
3. When committing changes to your branch, use the key in your commit message to link those commits to the development panel in your Jira work item. For example, `git commit -m "JRA-123 <summary of commit>"`.
4. When you create a pull request, use the key in the pull request title.

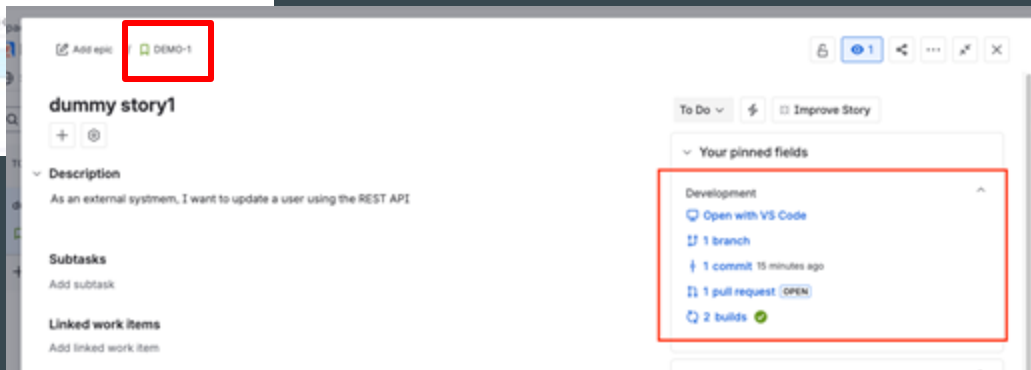
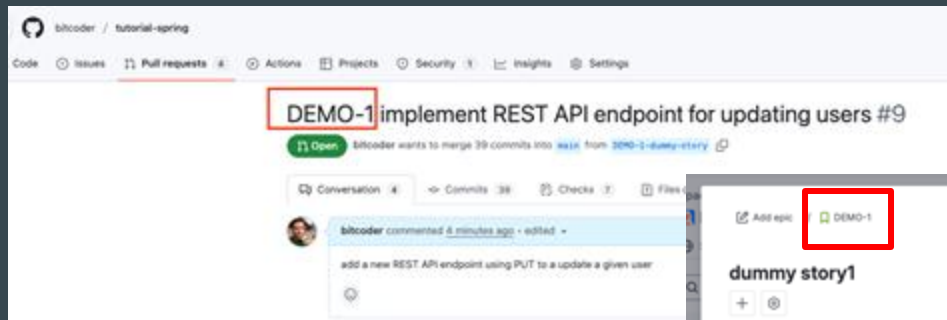
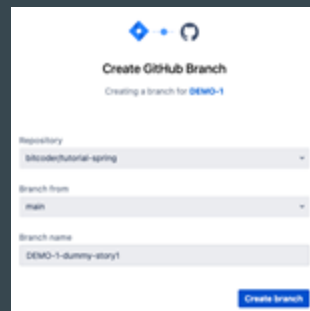
More info at: <https://support.atlassian.com/jira-software-cloud/docs/reference-issues-in-your-development-work/>

Create branch, make a commit and a PR; track on Development pane

`git checkout -b DEMO-1-dummy-story # or using Jira`

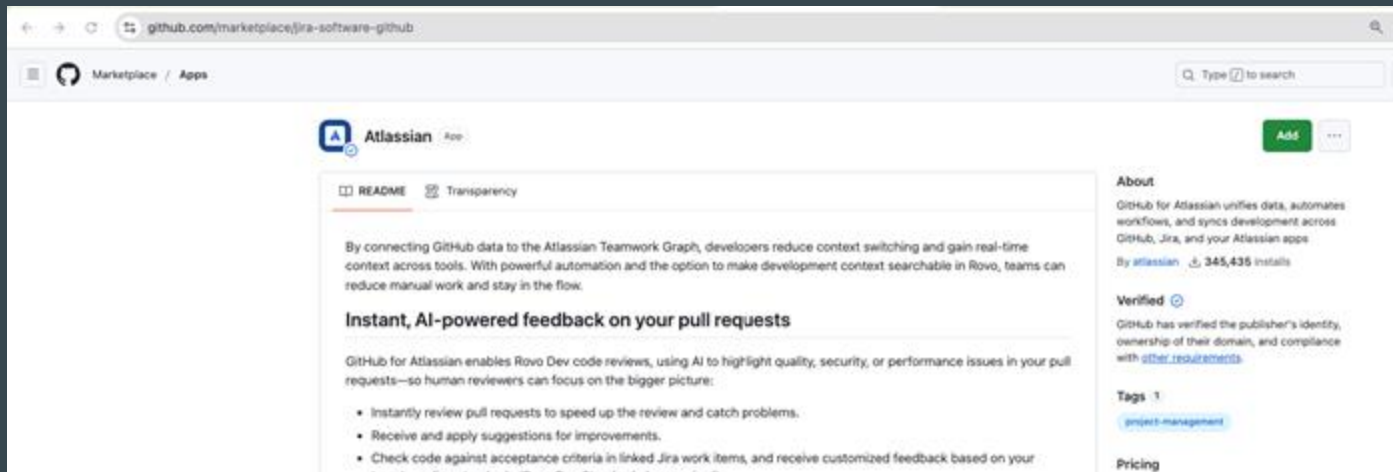
`git add src/main`

`git commit -m "DEMO-1 endpoint for updating user"`



Troubleshooting

- Make sure the “Atlassian” app is installed on GitHub and has right permissions on your project
- Wait up to 5min to see if commits and other info appear on Jira side (on the Development pane)



Xray Installation

Installation of Xray

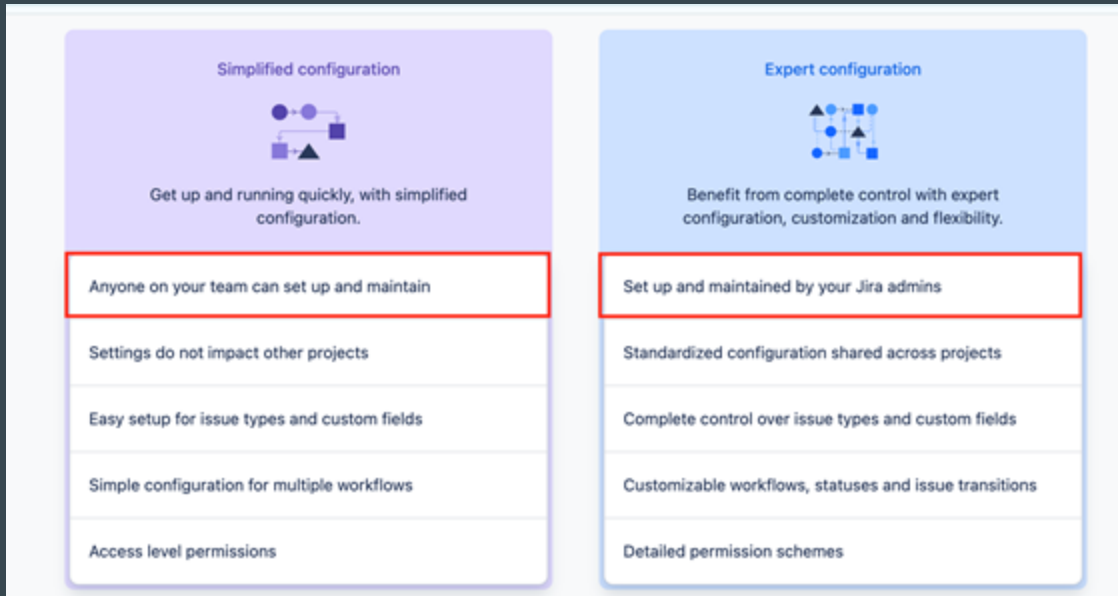
Installation is quite straightforward; go to Apps left side nav > Explore more apps > Xray and then “Try it free”. You need to use the account that created the Jira instance.

The image displays a sequence of four screenshots illustrating the installation process for Xray on Jira:

- Search for Xray:** The first screenshot shows the Jira Marketplace search results for "xray". The "Xray - Test Management for Jira" app by Xblend is highlighted with a red box. The app is rated 3.4/4 stars with 543 reviews and is labeled "CLOUD FORTIFIED".
- Select edition and site:** The second screenshot shows the "Select an edition and site" page. The "Standard" edition is selected, which includes "Onboard your team and scale fast". A site is selected from the dropdown menu.
- Review and install:** The third screenshot shows the "Review and install" page. It displays the app details, including the rating (3.4/4 stars, 543 reviews) and the price (USD 10 / month). A "TRY IT FREE" button is visible, indicating a 30-day free trial.
- Try it free:** The fourth screenshot shows the "Try it free" button, which is highlighted with a red box. The button is labeled "Try it free" and "Admission only to Jira instance".

Enabling Xray on Jira projects

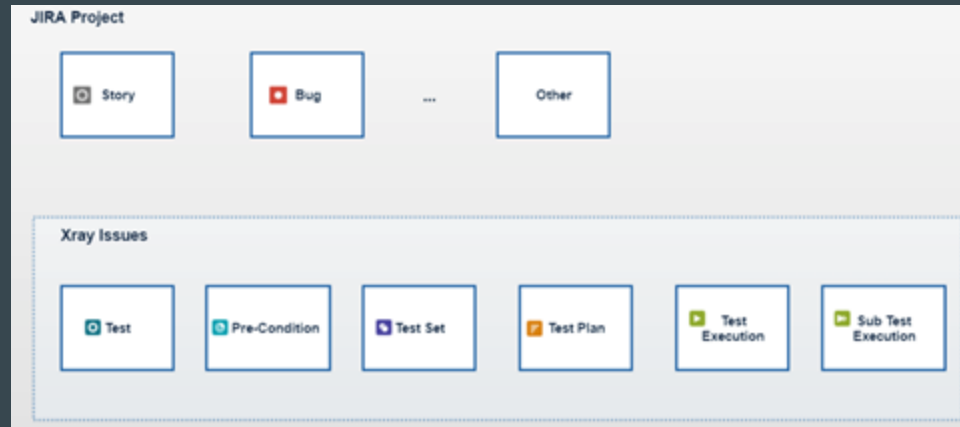
- Note: there are 2 main project types in Jira Cloud:
 - Team-managed
 - Company-managed
- We can enable Xray in any of these project types but the configuration steps are slightly different



Organization strategy: “All-in-One”

A single project to manage your “Requirements” (stories, epics), Defects (bugs), and Test related issues and also have all your Test Executions.

The idea is to use one project to manage everything related to it, including testing.



Enabling Xray on team managed projects

Setting a team-managed project in a nutshell

Create or use an existing team-managed project. You can use the project you created during the Jira signup.

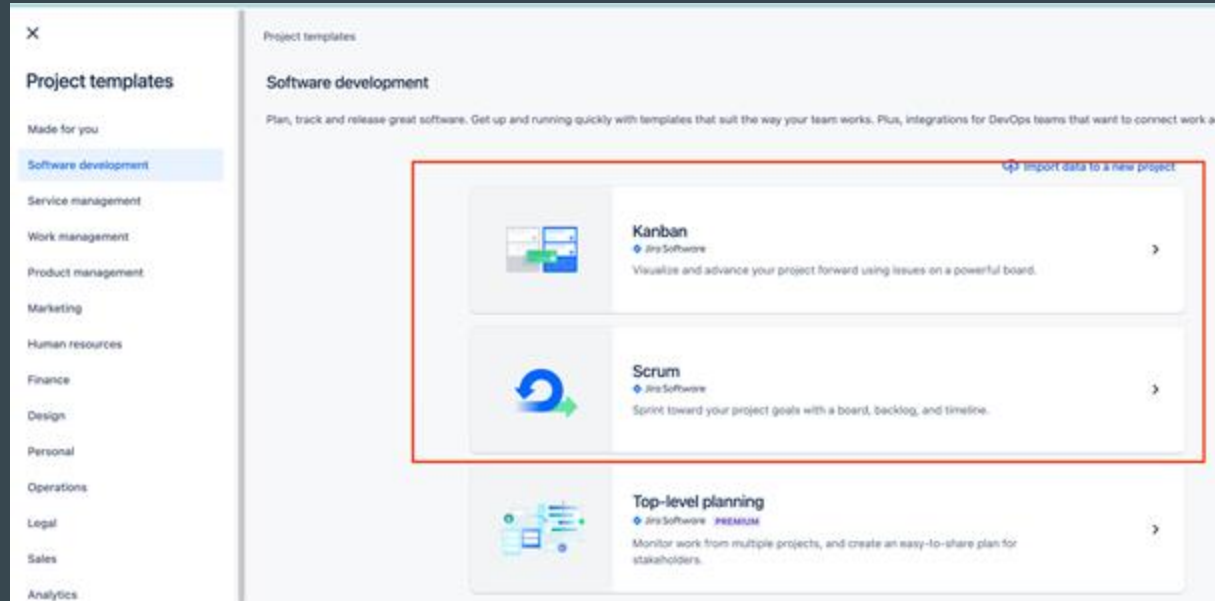
Then:

1. Add (i.e., create) issue types to the project and map them to the Xray concepts
 - a. Precondition, Test, Test Set, Test Execution, Test Plan
2. Define which items you consider to be “requirements” like
3. Define which items you consider to be “defects” like

That's it.

Create project/space (unless you have created one already)

- Create a Jira "space" (i.e., project) using a Kanban or Scrum template



Create project/space (unless you have created one already)



- We'll use the Scrum template


Add project details

Explore what's possible when you collaborate with your team. Edit project details anytime in project settings.


Name *

Access *

 Open 


Key  *

Template




Change template

Scrum

 Jira Software

Sprint toward your project goals with a board, backlog, and timeline.

Type



Change type

Team-managed

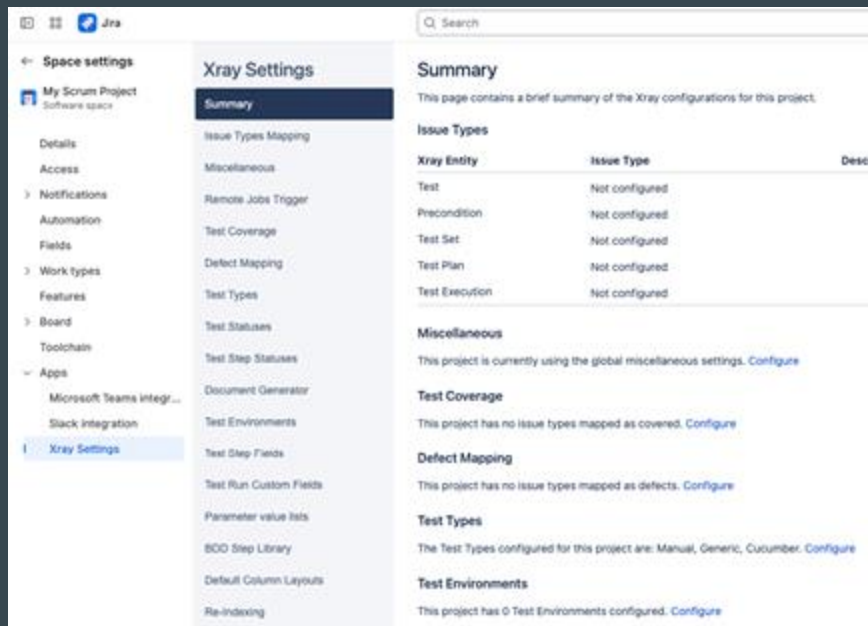
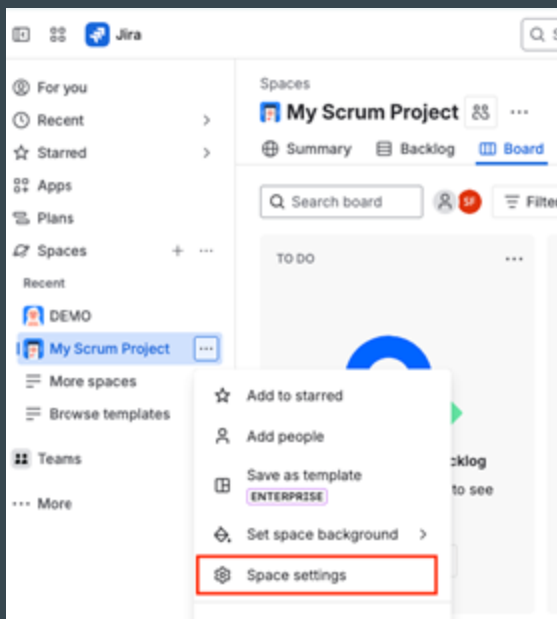
Control your own working processes and practices in a self-contained space.

Cancel

Next

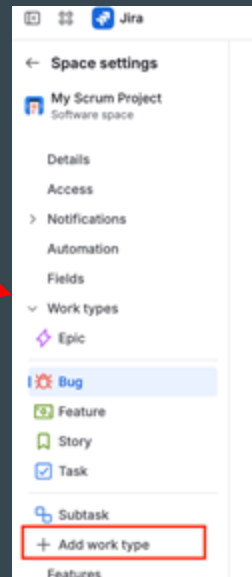
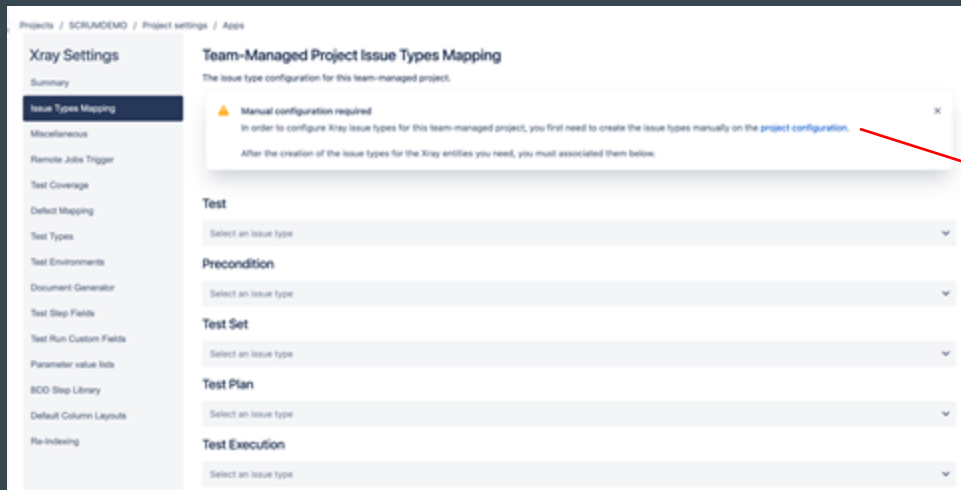
Xray project setup

- Go to Space settings > Apps > Xray Settings
- There are a few configurations to perform



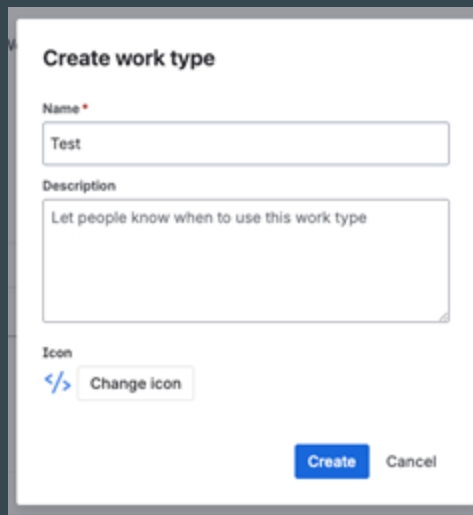
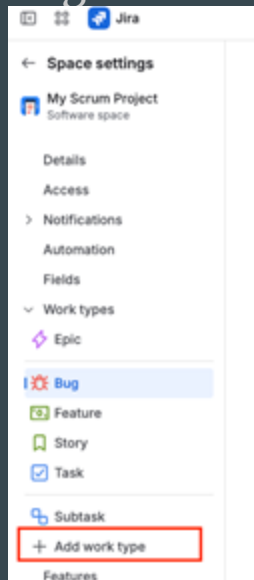
Xray project setup

- Go to **Issue Types Mapping** to configure the work types (aka “issue types”)
- Then go back to **Space settings > Work types** using the “project configuration” link to create the issue/work types used by Xray



Xray project setup: creation of work types

- Create the following work types (previously known as “issue types”) with these names: Precondition, Test, Test Set, Test Plan, and Test Execution
- Optionally: configure the related icons ([download them here](#))

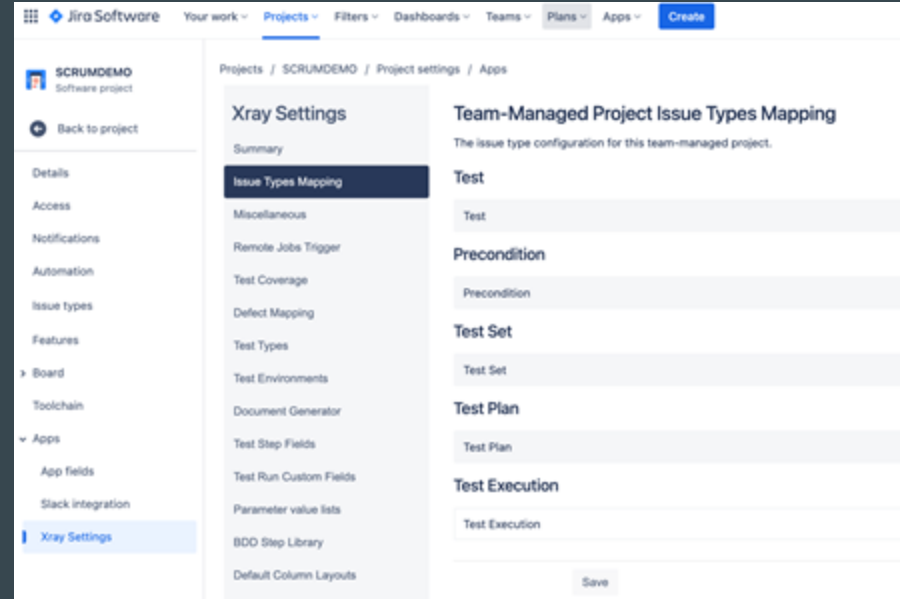
A screenshot of the 'Create work type' form in Jira. The form has fields for 'Name' (containing 'Test'), 'Description' (containing 'Let people know when to use this work type'), and 'Icon' (with a 'Change icon' button). 'Create' and 'Cancel' buttons are at the bottom right.

Xray project setup: associate the issue types

This step is for Xray to be aware of the issue types we want to use for its own entities; we could have created “Test Case”, “Test List” issue types (for example) and associate them with the Test and Test Set concept of Xray.

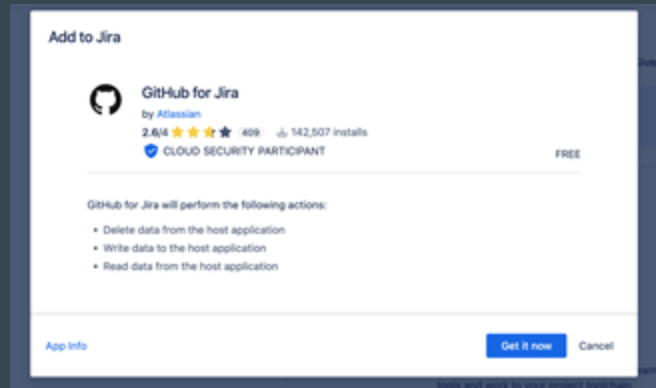
However, in our case, we'll keep it simple, using issue types with the same names of the expected entities:

- Test > Test
- Precondition > Precondition
- Test Set > Test Set
- Test Plan > Test Plan
- Test Execution > Test Execution



Nice-2-Have

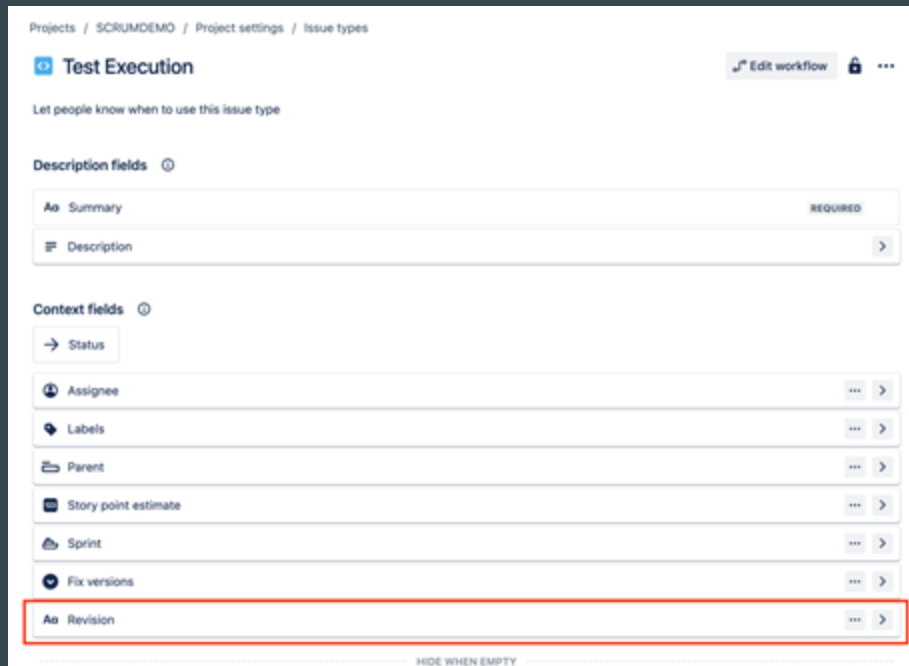
In Space settings > **Features**, enable Code and add the code repository tool (e.g., GitHub). => See detailed instructions provided earlier!



OPTIONAL: Associate some custom fields to Xray issue types

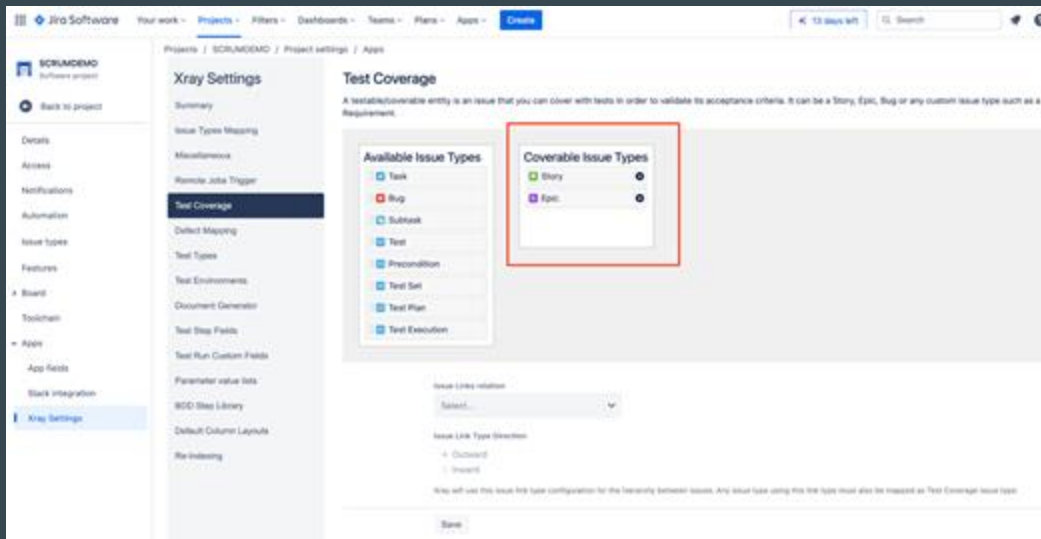
On **Project settings > Issue types**, for the following issue types add the fields

- Test Execution: Revision
 - As short text
- Test Plan: Begin Date and End Date
 - As date



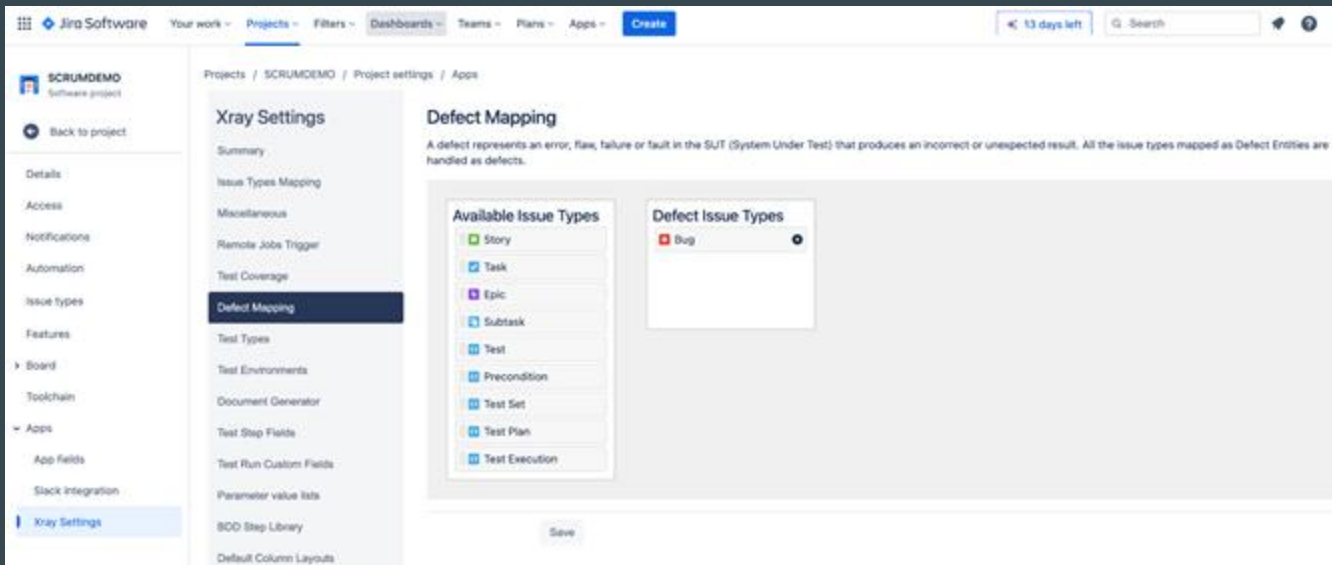
Xray: define “requirements” / issues coverable by tests

- **Coverable Issue Types:** *Which issue types do you aim to cover with tests? (e.g., Story, Epic)*



Xray: define defects

- **Defect Issue Types:** *Which issue types do you want to be handled as defects? (e.g., Bug)*



The screenshot shows the Jira Xray 'Defect Mapping' configuration page for a project named 'SCRUMDEMO'. The page is divided into three main sections:

- Left Sidebar (Xray Settings):** A list of settings categories including Summary, Issue Types Mapping, Miscellaneous, Remote Jobs Trigger, Test Coverage, Defect Mapping (which is currently selected and highlighted in dark blue), Test Types, Test Environments, Document Generator, Test Step Fields, Test Run Custom Fields, Parameter value lists, BDD Step Library, and Default Column Layouts.
- Top Section (Defect Mapping):** Contains a title 'Defect Mapping' and a descriptive text: 'A defect represents an error, flaw, failure or fault in the SUT (System Under Test) that produces an incorrect or unexpected result. All the issue types mapped as Defect Entities are handled as defects.'
- Right Section (Mapping):** Features two side-by-side panels:
 - Available Issue Types:** A list of Jira issue types with checkboxes. The checked items are 'Story', 'Task', 'Epic', 'Subtask', 'Test', 'Precondition', 'Test Set', 'Test Plan', and 'Test Execution'.
 - Defect Issue Types:** A box containing a single entry, 'Bug', which is marked as a defect with a red square icon.

A 'Save' button is located at the bottom center of the mapping area.

Xray: final checkup

Go to Space settings > Apps > Xray Settings > Summary.

The screenshot shows the Jira Software interface for a project named 'SCRUMDEMO'. The left sidebar contains a menu with options like 'Details', 'Access', 'Notifications', 'Automation', 'Issue Types', 'Features', 'Board', 'Toolchain', and 'Apps'. Under 'Apps', 'Xray Settings' is selected. The main content area is titled 'Xray Settings' and has a 'Summary' sub-tab. The 'Summary' section provides a brief overview of the Xray configurations for the project. It includes a table for 'Issue Types' mapping Jira entities to Xray issue types. Below the table, there are sections for 'Miscellaneous', 'Test Coverage', 'Defect Mapping', 'Test Types', and 'Test Environments', each with a brief description and a link to configure the settings.

Xray Settings

Summary

This page contains a brief summary of the Xray configurations for this project.

Issue Types

1 Xray Issue Types in Project

There are 5 of 5 Xray issue types configured for this project. Go to the [Issue Types Mapping](#) to configure the issue types for this project.

Xray Entity	Issue Type	Description
Test	<input checked="" type="checkbox"/> Test	
Precondition	<input checked="" type="checkbox"/> Precondition	
Test Set	<input checked="" type="checkbox"/> Test Set	
Test Plan	<input checked="" type="checkbox"/> Test Plan	
Test Execution	<input checked="" type="checkbox"/> Test Execution	

Miscellaneous

This project is currently using the global miscellaneous settings. [Configure](#)

Test Coverage

This project has the following issue types mapped as covered: ☒ Story ☒ Epic. [Configure](#)

Defect Mapping

This project has the following issue types mapped as defects: ☒ Bug. [Configure](#)

Test Types

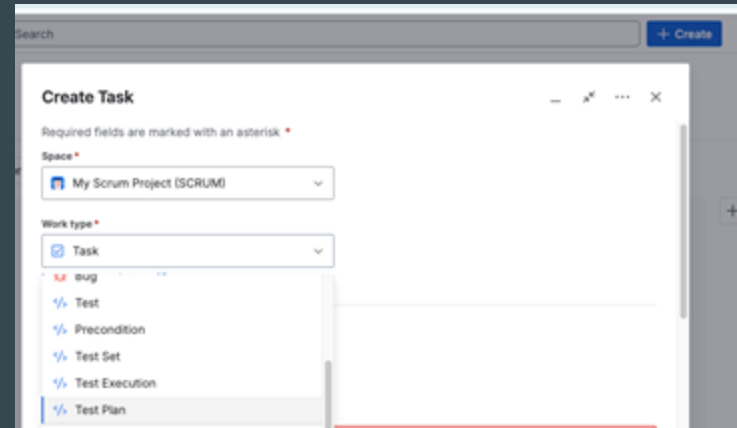
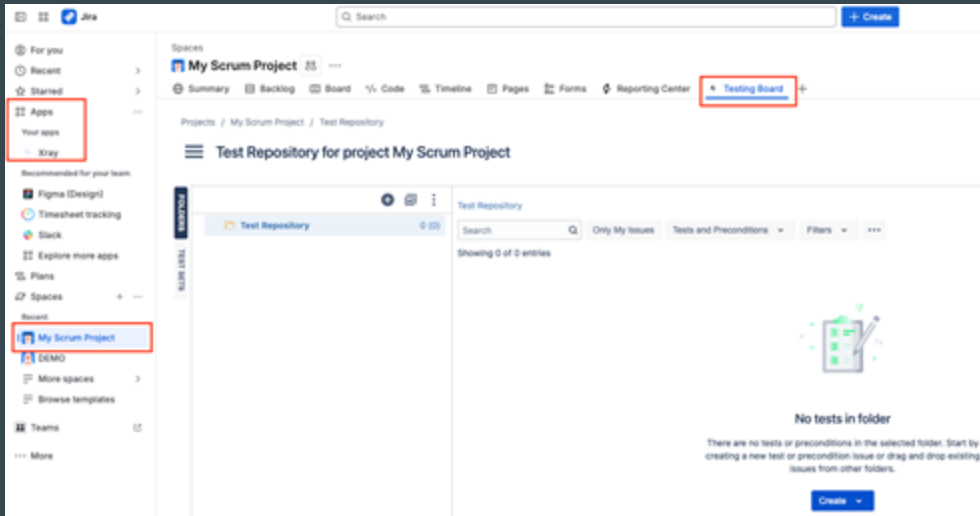
The Test Types configured for this project are: Manual, Generic, Cucumber. [Configure](#)

Test Environments

This project has 0 Test Environments configured. [Configure](#)

If all goes well...

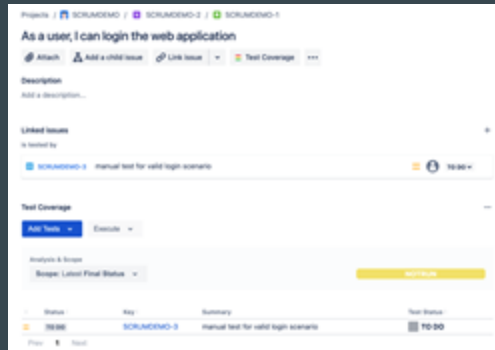
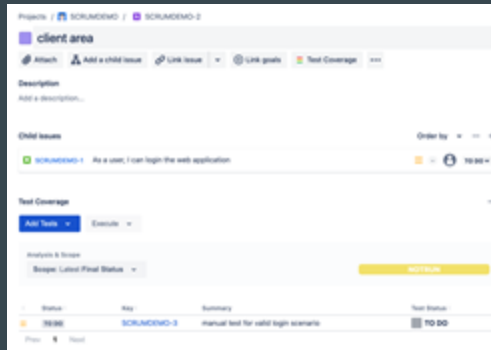
- You should see a Apps > Xray entry on the left side bar, and a “Testing Board” tab on the space/project home page
- Whenever creating issues, we can select the work/issue types that were created earlier



Try it out, just in case :)

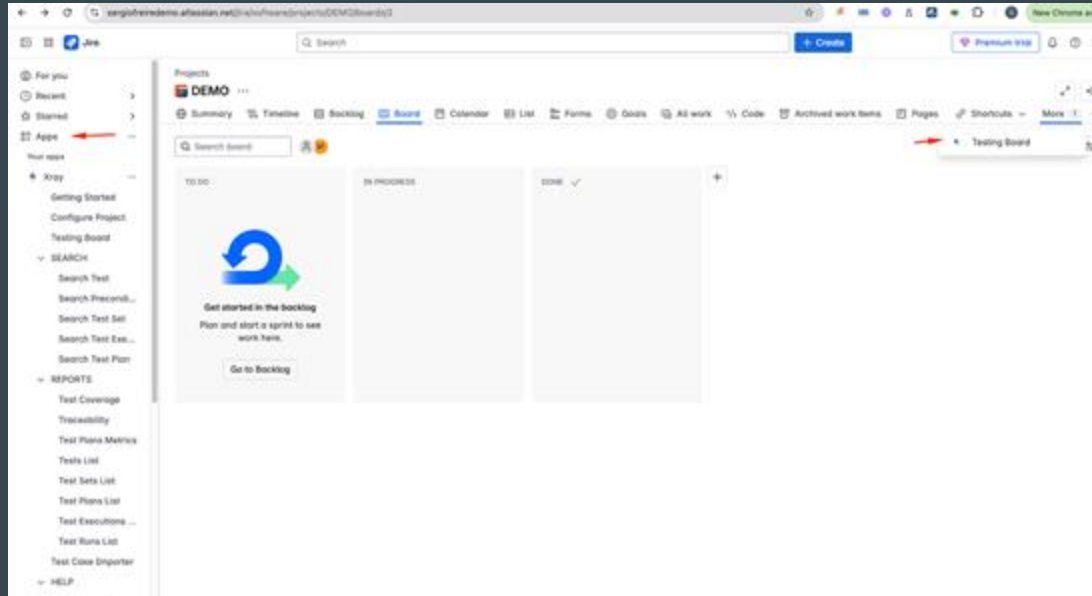
1. Create project with your name in uppercase (e.g., “SCRUMDEMO”); notice the project key (e.g., “SCRUMDEMO”)
2. Create Epic “client area” and a Story “login”; associate Story to the parent Epic
3. Create a Test from the Story screen

You should see coverage information on the Epic and also on the Story issue screen.



NOTE: new Jira UI coming during 2025...

Jira is changing the UI so the “Testing Board” page from Xray will appear on a different place. On the left side menu, Xray related pages are available from within Apps > Xray.



Applying promo code

Your Xray trial ends in 1 month, so you must use the “promo code” given by your teacher. Each group can only use the promo code once! Go to Jira > Administration > Billing > Enter promo code.

The image shows a sequence of three screenshots from the Jira Administration interface, illustrating the steps to enter a promo code.

Screenshot 1: The Jira Administration menu is open. The 'Administration' option, represented by a gear icon, is highlighted with a red box.

Screenshot 2: The 'Billing' tab is selected in the top navigation bar. The 'Billing' tab is highlighted with a red box.

Screenshot 3: The 'Subscription details' page is shown. The 'Enter promo code' button is highlighted with a red box.

Subscription details table:

Subscription details			
Payment details needed We'll need a payment method before May 25, 2025 for an upcoming bill, otherwise your subscription will be billing admin permissions to do this. Add payment method			
Cancel subscription Manage customer contacts			
Xray Test Management for Jira sergiofreiredemo.atlassian.net 1 XRP USD 10.00 USD			
Items	Billing cycle	Next bill date	Next bill estimate (1)
1	Monthly	May 25, 2025	USD 10.00
Billing profile Installed on (2)			
None			
2 subscriptions Manage			

Xray Entities & Concepts

...

Xray entities & concepts (a few)

Xray provides several concepts; we'll focus just on a few ones.

- Test
- Test Run
- Test Execution
- Test Plan
- “Requirements”
- “Defects”

All of the previous entities will be issues in Jira/Xray, except the Test Run which is

Test

An abstraction of a test idea/scenario, automated test, and thus reusable.
It is essentially a way to verify/validate the associated requirement(s).

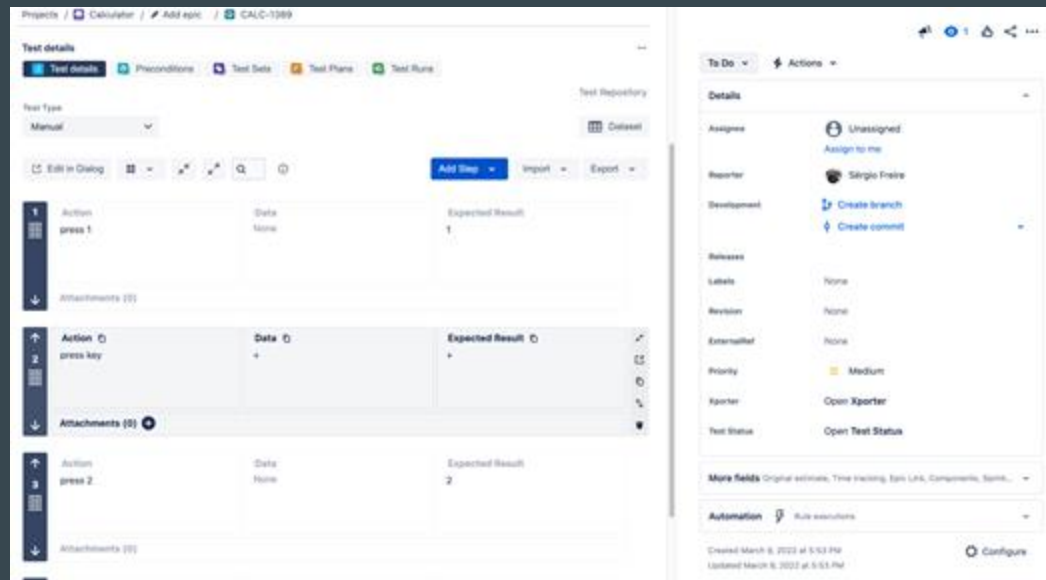


- Can be a scripted (e.g. test case, automated test) or exploratory test
- Can be specified using Gherkin (Scenario)
- Can be executed manually in Jira/Xray or through automation
- May be linked to/cover 1 (or more) requirements, using the “tests” issue link type
- Has one type: “Manual”, “Cucumber”, “Generic”

Test: Manual

A traditional test case composed by a list of steps, thus scripted.

- Each step is mainly defined by:
 - Action/step
 - Expected result



If we don't have test automation yet, or the test scenario is hard/costly to automation, we can create a “manual” test and link it to the related Story, for example. We can then manually record its result in

Test: Cucumber

A Cucumber Scenario/Scenario Outline that provides one or more examples of an acceptance criteria.

- Write tests in a business-readable domain-specific language (Gherkin)
- Specify Cucumber Scenarios and Scenario Outlines with Gherkin syntax highlighting
- Ability to export to .feature files and execute during Continuous Integration

```
Test Type
Cucumber ▼

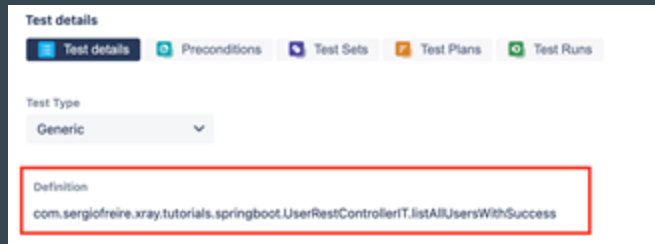
Scenario
1 Given I have entered "<input_1>" into the calculator
2 And I have entered "<input_2>" into the calculator
3 When I press "<button>"
4 Then the result should be "<output>" on the screen
5
6 Examples:
7 | input_1 | input_2 | button | output |
8 | 20      | 30      | add    | 50      |
9 | 2       | 5       | add    | 7       |
10 | 0       | 40      | add    | 40      |
11 | 4       | 50      | add    | 54      |
12 | 5       | 50      | add    | 55      |
```

We can use Xray as the master for the Cucumber/Gherkin scenarios or we can use Git instead. The flows for fully integrating the results from automation are slightly different as shown in [this tutorial](#).

Test: Generic

A way to abstract and have visibility of traditional automated tests or exploratory tests.

- Allows you to track the results of automated tests that are non-Cucumber (e.g. JUnit)
 - Automate tests in any framework and report results back to Xray
 - These tests are usually auto-provisioned whenever importing the results the first time
 - Definition field contains a unique identifier of automated test like `<package_name>.<class_name>.<method_name>`



Note: Generic tests could also be used to have visibility of Cucumber results if upload JUnit XML reports. In that case we lose visibility of the individual Gherkin sentences; we would just have visibility of the whole test result.

Test Execution

A “task” for executing a group of tests on a given version of the system, on a given environment. This task will also contain the results when they’re reported. Sometimes the Test Execution is created alongside with its results (i.e., the Test Runs).



- Contains a list of Tests and their results (i.e. Test Runs)
- May be planned (especially for manual tests) or ad hoc
- May be created manually (i.e., “I want to run these tests...”);
 - Later on someone will “execute” it manually, by going to each test and record the actual results and whether the test passed or failed
- May be created during Continuous Integration
 - in this case it will already contain the results (i.e., a Test Run for each Test that was executed)

Test Run

An instance of a Test in the context of a Test Execution. A run of a Test in some scope.

- Contains the results obtained for a Test, including evidence and linked/reported defects
- For compliance, also contains a copy of the Test specification at the moment of execution
- It's an internal entity; not a Jira issue

Test Plan

A way to define the scope for testing and track its progress; what testing we'll be performing and its latest status.



- Tracks a group of tests and their results independently of the number of executions
- Can be used in a planned way, with its scope for testing (i.e. the Tests) defined beforehand
- Can be used in an Agile way, acting as a testing guidance result aggregator, by allowing you to create/add Tests along with their results at any time
- Test Plans may be assigned to versions, sprints and users, as they're a Jira issue
- A version or a sprint may have multiple Test Plans

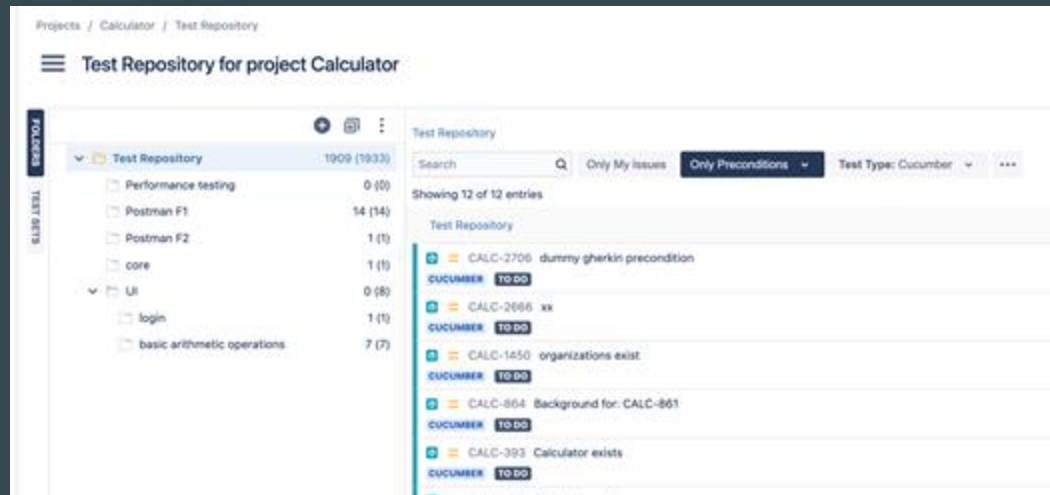
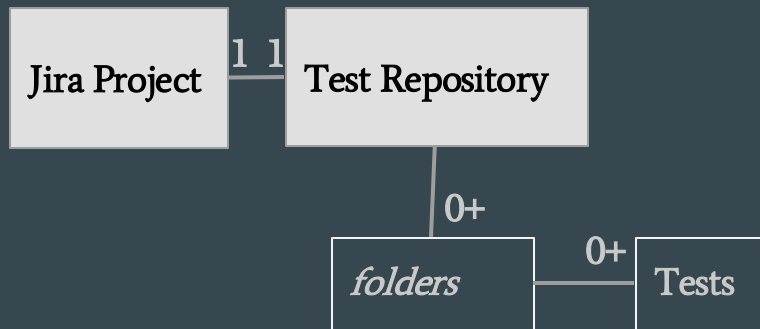
Relation between entities

- “Requirements” are coverable by one or more Tests
- A Test Plan has a list of Tests it tracks and usually several Test Executions related to those Tests
- A Test Execution contains Test Runs, one for each Test that is on the Test Execution

Additional entities... (non issue type based)

Test Repository

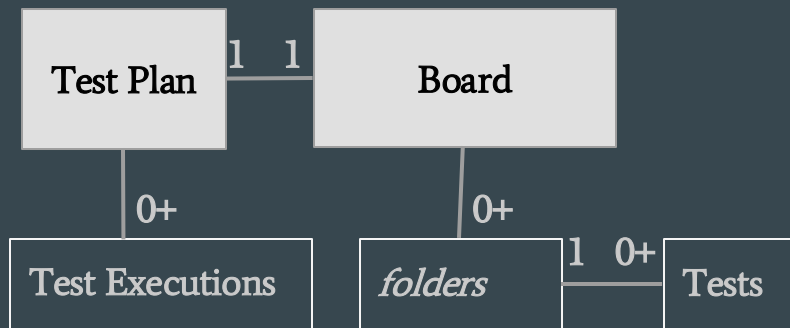
- The place where we can see all Tests in a project, organized in folders
- No information about results
- Mostly useful for manual testing



Additional entities... (non issue type based)

Test Plan' Board

- An implicit structure in each Test Plan to organize the Tests in folders, considering priorities and what makes sense from a execution perspective
- Can be useful to group the results logically, using folders



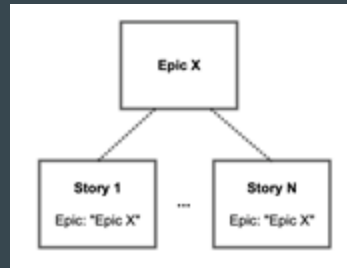
Defects (e.g., “Bug”)

A defect is something that negatively impacts quality (the value perceived to some stakeholder).

- Xray uses the “defect” word in a broader sense, i.e. some deficiency/imperfection related to the product. Usually, these are also known as bugs
- In practice, a “defect” is something we report manually during testing, or after testing whenever analyzing the test results, as a Jira issue (e.g., “Bug”)
- You can define one or more issue types to be treated as “defects”; they can include Bug or any other custom issue types you may want for that purpose

“Requirements” (e.g., Story, Epic)

- Xray uses “requirement” word in broad sense: something that the SUT must meet
- In practice, a “requirement” is any issue in Jira that can be covered (i.e., verified or validated) with Tests
- You can define the issue types to be treated as “requirements”; they can include Story, Epic, or any other custom issue type
- Xray is aware of the hierarchical relation between Epics and Stories; if you create a Test for a Story, you can see it at the related Epic level (i.e., covering the Epic)



Coverage in Xray

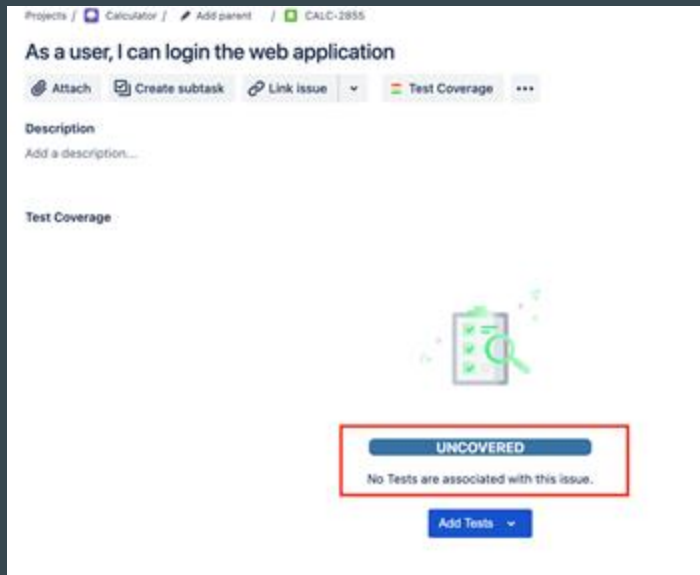
Xray provides an heuristic, called (test) coverage; sometimes people may refer to it as requirement coverage, depending on the perspective.

This is different from code coverage, which is focused on covering code.

Coverage in Xray is at higher level; it's a way to understand if a deliverable/requirement (Story, Epic) has some tests associated and if so, if all those tests have been run and were successful.

Coverage: UNCOVERED

In Xray a “requirement” (Story, Epic) is UNCOVERED if it has no Tests linked/covering it.



Coverage: NOTRUN

In Xray, a requirement is “NOTRUN” if:

- **any** of the linked tests need yet to be run

Remember: it's an heuristic! Tests don't cover everything and not every test (problems found during their execution) has the same importance.

The screenshot displays the Xray web application interface for a requirement. At the top, the breadcrumb navigation shows 'Projects / Calculator / Add parent / CALC-2855'. The requirement title is 'As a user, I can login the web application'. Below the title, there are buttons for 'Attach', 'Create subtask', 'Link issue', and 'Test Coverage'. The 'Description' section is empty. The 'Linked issues' section shows 'is tested by' with a link to 'CALC-2856 test for successful login'. The 'Test Coverage' section has a yellow 'NOTRUN' status. Below this, there is a table with columns 'Status', 'Key', 'Summary', and 'Test Status'. The table contains one row with the status 'TO DO', key 'CALC-2856', summary 'test for successful login', and test status 'TO DO'.

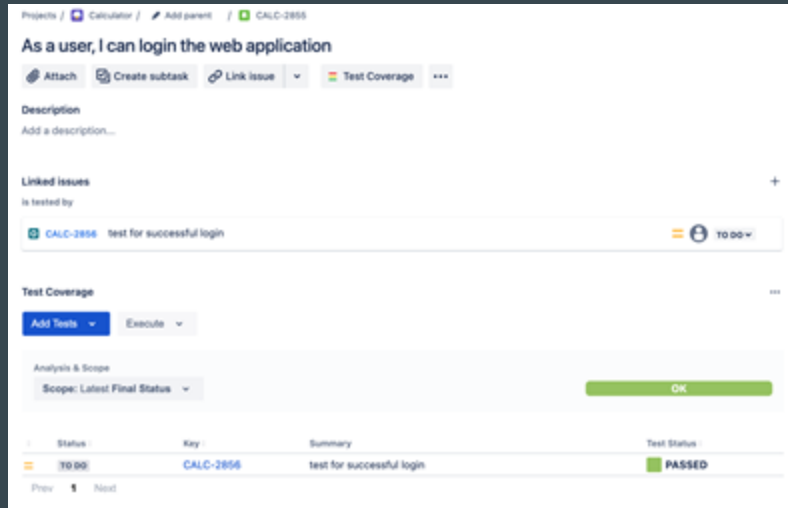
Status	Key	Summary	Test Status
TO DO	CALC-2856	test for successful login	TO DO

Coverage: OK

In Xray, and also in some tools, a requirement is “OK” if ...

- All linked tests passed, according with latest results

Remember: it's an heuristic! Tests passing don't mean we can ensure the requirement doesn't have problems.



The screenshot shows a Jira Xray requirement page for the requirement "As a user, I can login the web application". The page includes a description field, a linked issue "CALC-2856 test for successful login", and a test coverage section. The test coverage section shows a green bar with the status "OK". Below this, a table lists the test results for the requirement.

Status	Key	Summary	Test Status
TO DO	CALC-2856	test for successful login	PASSED

Coverage: NOK

In Xray, a requirement is “NOK” if ...

- *any* of the linked failed, according with latest results

Remember: it's an heuristic! Tests don't cover everything and not every test (problems found during their execution) has the same relevance.

The screenshot shows the Xray web application interface for a requirement titled "As a user, I can login the web application". The requirement is linked to two test cases: "test for successful login" and "test for invalid login", both with key "CALC-2856". The "Test Coverage" section shows a red "NOK" button, indicating that the requirement is not okay because at least one of the linked tests failed. Below this, a table lists the test results:

Status	Key	Summary	Test Status
TO DO	CALC-2856	test for successful login	PASSED
TO DO	CALC-2856	test for invalid login	FAILED

A red arrow points from the "FAILED" status to the "NOK" button, illustrating the heuristic that if any linked test fails, the requirement is marked as NOK.

OTHER, OPTIONAL INFO



Just in case you need it...

Enabling Xray on company managed projects

Setting a company-managed project in a nutshell

Create or use an existing company-managed project and then:

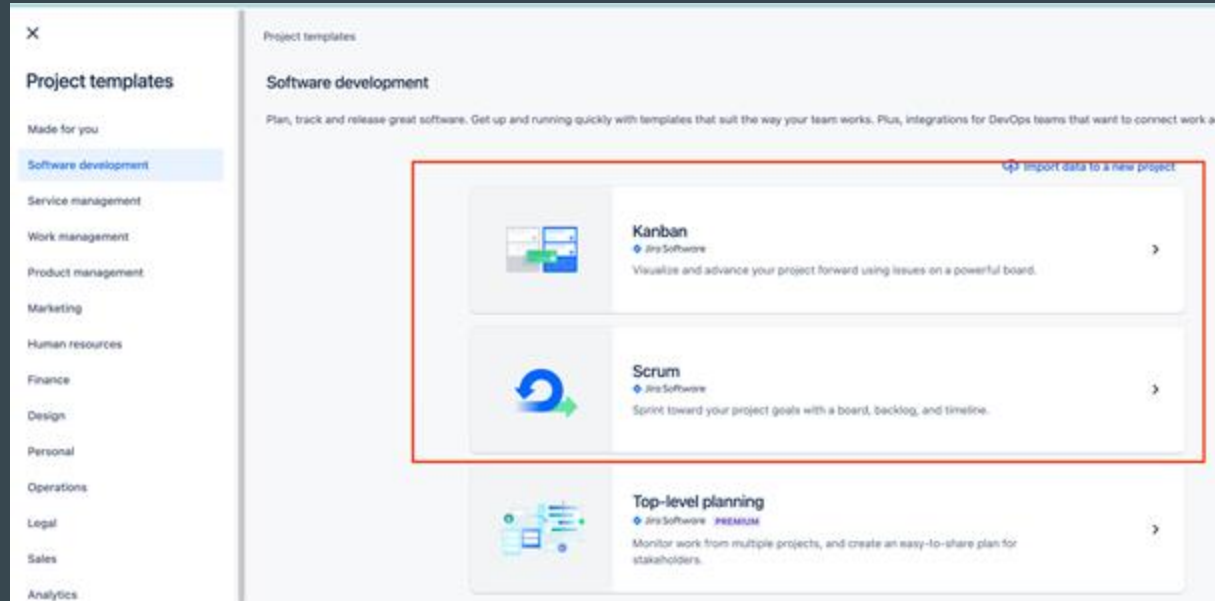
1. Add issue types to the project and map them to the Xray concepts
2. Define which items you consider to be “requirements” like
3. Define which items you consider to be “defects” like

That's it.

There are additional settings that we can fine tune ahead.

Create project

- Create a Jira project using a Kanban or Scrum template



Select type of project

Team-managed	Company-managed
<p>Set up and maintained by your team.</p> <p>For teams who want to control their own working processes and practices in a self-contained space. Mix and match agile features to support your team as you grow in size and complexity.</p> <p>Simplified configuration</p>  <p>Get up and running quickly, with simplified configuration.</p> <ul style="list-style-type: none">Anyone on your team can set up and maintainSettings do not impact other projectsEasy setup for issue types and custom fieldsSimple configuration for multiple workflowsAccess level permissions	<p>Set up and maintained by your Jira admins.</p> <p>For teams who want to work with other teams across many projects in a standard way. Encourage and promote organizational best practices and processes through a shared configuration.</p> <p>Expert configuration</p>  <p>Benefit from complete control with expert configuration, customization and flexibility.</p> <ul style="list-style-type: none">Set up and maintained by your Jira adminsStandardized configuration shared across projectsComplete control over issue types and custom fieldsCustomizable workflows, statuses and issue transitionsDetailed permission schemes
<p>Essential features</p>  <p>A modern Jira experience for teams who don't need advanced features.</p> <ul style="list-style-type: none">Only show your project's issues on your boardEssential agile reporting	<p>Advanced features</p>  <p>All the power and features that Jira Software is known for.</p> <ul style="list-style-type: none">Pull in issues from other projects on your boardComprehensive agile reporting
Select a team-managed project	Select a company-managed project

Add project details

Add project details

Explore what's possible when you collaborate with your team. Edit project details anytime in project settings.

Required fields are marked with an asterisk *

Name *

Key ⓘ *

☐ Share settings with an existing project

Template

Change template



Scrum

◆ Jira Software

Sprint toward your project goals with a board, backlog, and timeline.

Type

Change type



Company-managed

Work with other teams across many projects in a standard way.

Cancel

Next

Xray project setup

- Go to Project settings > Apps > Xray Settings
- There are several configurations to perform

The screenshot shows the Jira Software interface for 'SAMPLEPROJ2'. The left sidebar contains a navigation menu with categories like Summary, People, Permissions, Notifications, Automation, Features, Tools, Workflow, Issues, Types, Layout, Screens, Fields, Collectors, Security, Components, and Apps. The 'Apps' section is expanded, and 'Xray Settings' is selected. The main content area is titled 'Xray Settings' and has a 'Summary' tab selected. The 'Summary' section provides a brief overview of the Xray configurations for the project. Below this, the 'Issue Types' section lists various issue types configured for the project, including Test, Precondition, Test Set, Test Plan, Test Execution, and Sub-Test Execution. Each issue type has a description and a 'Present in Project' status. The 'Miscellaneous' section includes links to configure global miscellaneous settings, test coverage, defect mapping, and test types.

Xray Settings

Summary

This page contains a brief summary of the Xray configurations for this project.

Issue Types

There are 6 Xray issue types configured for this project. In order to enable this project for testing with Xray, you need to configure the Xray issue types. You can do it automatically by pressing "Add Xray Issue Types" or manually by going into the issue type scheme for this project and editing the Xray issue types there.

[Add Xray Issue Types](#) [Configure the issue type scheme manually](#)

Name	Description	Present in Project
Test	This is the Xray Test Issue Type. Used to define test cases of different types that can be executed multiple times using Test Execution Issues.	Yes
Precondition	This is the Xray Precondition issue Type. Used to abstract common actions that must be ensured before the test case execution. A Precondition can be associated with multiple test cases.	Yes
Test Set	This is the Xray Test Set issue Type. Creates a group of test cases. Used to associate all included Tests with other Xray issue types like Test Execution and Test Plan. A Test Set can also be associated with a requirement issue to provide coverage and test status.	Yes
Test Plan	This is the Xray Test Plan issue Type. Used to define the scope of test cases for a given test campaign and to aggregate all executions for those tests displaying the latest result for each test case.	Yes
Test Execution	This is the Xray Test Execution issue Type. Used to execute test cases already defined.	Yes
Sub-Test Execution	This is the Xray Sub-Test Execution issue Type. Used to execute test cases already defined. A Sub-Test Execution can be created for a parent issue like a requirement in order to execute the test cases associated with it.	Yes

Miscellaneous

This project is currently using the global miscellaneous settings. [Configure](#)

Test Coverage

This project has no issue types mapped as covered. [Configure](#)

Defect Mapping

This project has no issue types mapped as defects. [Configure](#)

Test Types

This Test Types configured for this project are: Manual, Generic, Cucumber. [Configure](#)

Xray: define “requirements” / the coverage

- Coverable Issue Types: *Which issue types do you aim to cover with tests?*

(e.g., Story, Epic)

The screenshot shows the 'Xray Settings' interface with the 'Test Coverage' tab selected in the left sidebar. The main content area is titled 'Test Coverage' and includes a description: 'A testable/coverable entity is an issue that you can cover with tests in order to validate its acceptance criteria. It can be a Story, Epic, Bug or any custom issue type such as a Requirement.'

There are two panels: 'Available Issue Types' and 'Coverable Issue Types'. The 'Available Issue Types' panel lists: Task, Sub-task, Bug, Test, Test Set, Test Plan, Test Execution, Precondition, and Sub Test Execution. The 'Coverable Issue Types' panel lists: Story and Epic. A red arrow points from 'Story' in the available list to the 'Coverable Issue Types' panel.

Below these panels, there are configuration options:

- Test Coverage Hierarchy**
 - ☒ **Epic - Issues(Stories) relation**
When this option is enabled, Xray will consider all child issues that are associated with Epics. These child issues must still be mapped as Test Coverage issue types.
 - ☒ **Issue - Sub-tasks relation**
When this option is enabled, Xray will consider all sub-task issues. Either the Issue Type and the Sub-task issue type must be mapped as Test Coverage issue types.
- Issue Links relation**
A dropdown menu with 'Select...' and a downward arrow.
- Issue Link Type Direction**
 - ☒ Outward
 - ☐ Inward

A note at the bottom states: 'Xray will use this issue link type configuration for the hierarchy between issues. Any issue type using this link type must also be mapped as Test Coverage issue type.'

A red box highlights the 'Save' button at the bottom right of the settings area.

Xray: define defects

- Defect Issue Types: *Which issue types do you want to be handled as defects?*

(e.g., Bug)

The screenshot shows the 'Xray Settings' sidebar on the left with 'Defect Mapping' selected. The main panel is titled 'Defect Mapping' and contains a description: 'A defect represents an error, flaw, failure or fault in the SUT (System Under Test) that produces an incorrect or unexpected result. All the issue types mapped as Defect Entities are handled as defects.' Below this, there are two panels: 'Available Issue Types' and 'Defect Issue Types'. The 'Available Issue Types' panel lists: Task, Sub-task, Story, Epic, Test, Test Set, Test Plan, Test Execution, Precondition, and Sub Test Execution. The 'Defect Issue Types' panel shows 'Bug' selected, indicated by a red arrow from the 'Task' item in the 'Available Issue Types' panel. A 'Save' button is at the bottom.

Xray Settings

- Summary
- Miscellaneous
- Remote Jobs Trigger
- Test Coverage
- Defect Mapping**
- Test Types
- Test Environments
- Document Generator
- Test Step Fields
- Test Run Custom Fields
- Parameter value lists
- BDD Step Library
- Default Column Layouts
- Re-Indexing

Defect Mapping

A defect represents an error, flaw, failure or fault in the SUT (System Under Test) that produces an incorrect or unexpected result. All the issue types mapped as Defect Entities are handled as defects.

Available Issue Types

- Task
- Sub-task
- Story
- Epic
- Test
- Test Set
- Test Plan
- Test Execution
- Precondition
- Sub Test Execution

Defect Issue Types

- Bug

Save

Xray: final checkup

Go to Project settings > Apps > Xray Settings > Summary.

The screenshot shows the 'Xray Settings' page for a project named 'SAMPLEPROJ'. The left sidebar contains a navigation menu with categories like 'Summary', 'People', 'Permissions', 'Notifications', 'Automation', 'Features', 'Toolchain', 'Workflows', 'Issues', 'Types', 'Layout', 'Screens', 'Fields', 'Collections', 'Security', 'Components', 'Apps', and 'Development Tools'. The 'Xray Settings' page is divided into three main sections: 'Summary', 'Issue Types', and 'Miscellaneous'.

Summary

This page contains a brief summary of the Xray configurations for this project.

Issue Types

There are 6 Xray issue types configured for this project. Click here to edit the issue types for this project.

Name	Description	Present in Project
Test	This is the Xray Test issue Type. Used to define test cases of different types that can be executed multiple times using Test Execution issues.	✓
Precondition	This is the Xray Precondition issue Type. Used to abstract common actions that must be ensured before the test case execution. A Precondition can be associated with multiple test cases.	✓
Test Set	This is the Xray Test Set issue Type. Creates a group of test cases. Used to associate all included Tests with other Xray issue types like Test Execution and Test Plan. A Test Set can also be associated with a requirement issue to provide coverage and test status.	✓
Test Plan	This is the Xray Test Plan issue Type. Used to define the scope of test cases for a given test campaign and to aggregate all executions for those tests displaying the latest result for each test case.	✓
Test Execution	This is the Xray Test Execution issue Type. Used to execute test cases already defined.	✓
Sub-Test Execution	This is the Xray Sub-Test Execution issue Type. Used to execute test cases already defined. A Sub-Test Execution can be created for a parent issue like a requirement in order to execute the test cases associated with it.	✓

Miscellaneous

This project is currently using the global miscellaneous settings. [Configure](#)

Test Coverage

This project has the following issue types mapped as covered: [Story](#) [Epic](#) [Configure](#)

Defect Mapping

This project has the following issue types mapped as defects: [Bug](#) [Configure](#)

Test Types

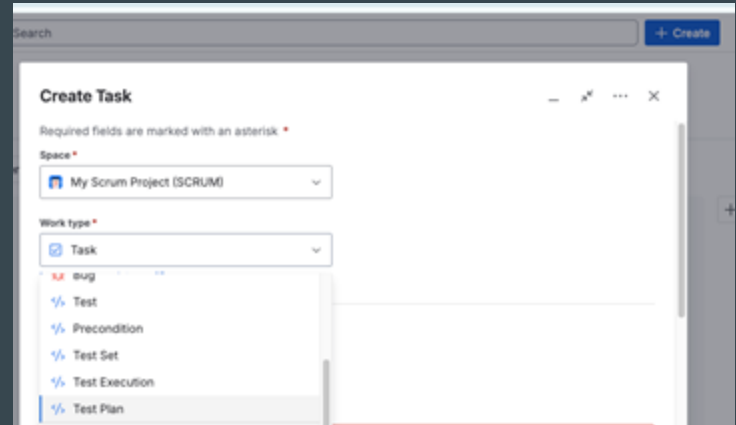
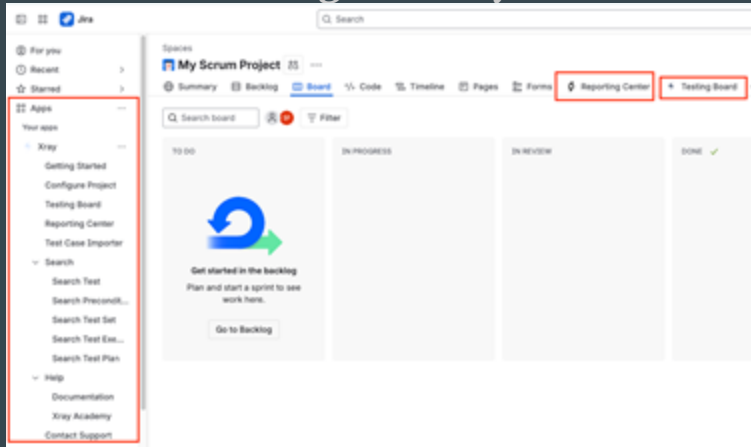
The Test Types configured for this project are: Manual, Generic, Cucumber. [Configure](#)

Test Environments

This project has 0 Test Environments configured. [Configure](#)

If all goes well...

- On the left bar, on **Apps** you should have a Xray dedicated menu On the space/project home accessible from the left bar, you should see a “**Testing Board**” tab (and also a “Reporting Center” btw) on the right side; the Testing Board is the main entry page for Xray but there are other pages
- Whenever creating issues, you should see the issue types that were created earlier



DEMO / Try it out, just in case :)

1. Create Epic and a Story; associate Story to the parent Epic
2. Create a Test from the Story screen

You should see coverage information on the Epic and also on the Story issue screen.

The image displays three overlapping screenshots of a software development tool interface, likely Jira or a similar project management tool, illustrating the workflow for creating and managing tests.

Left Screenshot (Epic View): Shows the 'client area' for 'SAM-1'. The 'Description' field is empty. Under 'Child issues', there is a list item 'SAM-2 As a user, I can login the web application'. The 'Test Coverage' section shows 'Add Tests' and 'Execute' buttons. The 'Analysis & Scope' section shows 'Scope: Latest Final Status' and a 'NOT RUN' button. A table at the bottom lists issues with columns for Status, Key, Summary, and Test Status. The table shows one row with Status 'TO DO', Key 'SAM-3', Summary 'test for valid login scenario', and Test Status 'TO DO'.

Middle Screenshot (Story View): Shows the 'As a user, I can login the web application' story. The 'Description' field is empty. Under 'Linked issues', there is a list item 'SAM-3 test for valid login scenario'. The 'Test Coverage' section shows 'Add Tests' and 'Execute' buttons. The 'Analysis & Scope' section shows 'Scope: Latest Final Status' and a 'NOT RUN' button. A table at the bottom lists issues with columns for Status, Key, Summary, and Test Status. The table shows one row with Status 'TO DO', Key 'SAM-3', Summary 'test for valid login scenario', and Test Status 'TO DO'.

Right Screenshot (Test View): Shows the 'test for valid login scenario' test. The 'Description' field is empty. Under 'Linked issues', there is a list item 'SAM-2 As a user, I can login the web application'. The 'Test details' section shows 'Test details', 'Preconditions', 'Test Data', 'Test Plans', and 'Test Runs' tabs. The 'Test Type' is set to 'Manual'. A message at the bottom states 'There are no steps defined' and provides a link to 'Add Step'.

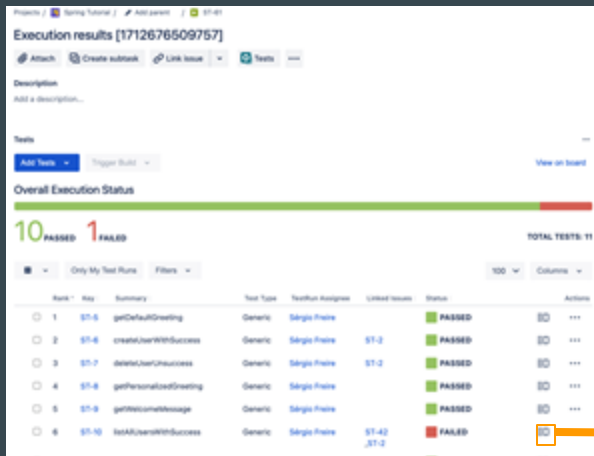
Integrating Xray with Test Automation

Integration with Jira and Xray

Goal 1: Track (i.e., have visibility of) test automation results in Jira, using Xray

Test Execution

(i.e., a batch of test results, composed of multiple Test Runs, one per each Test that was executed)

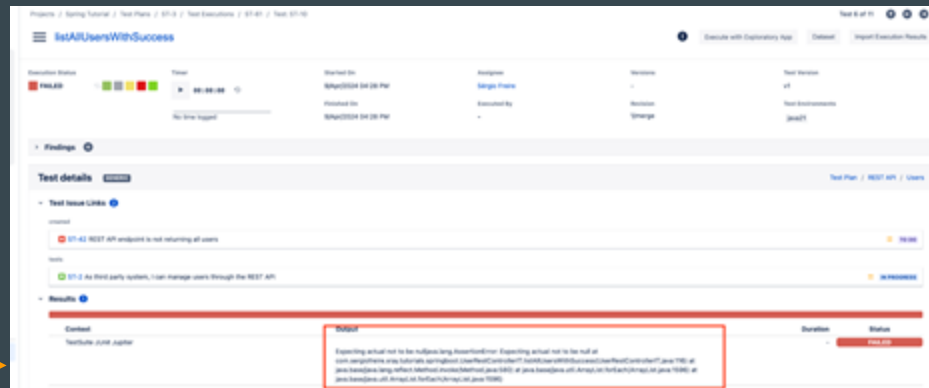


The screenshot shows the 'Execution results' page for a specific test execution. It includes a description field, a 'Tests' section with 'Add Tests' and 'Trigger Build' buttons, and an 'Overall Execution Status' bar showing 10 PASSED and 1 FAILED. Below this is a table of test results.

Rank	Key	Summary	Test Type	TestRun Assignee	Linked Issues	Status	Actions
1	\$7-5	getDefaulGreeting	Generic	Sergio Freire		PASSED	
2	\$7-6	createUserWithSuccess	Generic	Sergio Freire	\$7-2	PASSED	
3	\$7-7	deleteUserUnsuccess	Generic	Sergio Freire	\$7-2	PASSED	
4	\$7-8	getPersonalizedGreeting	Generic	Sergio Freire		PASSED	
5	\$7-9	getWelcomeMessage	Generic	Sergio Freire		PASSED	
6	\$7-10	testUsersWithSuccess	Generic	Sergio Freire	\$7-2, \$7-2	FAILED	

Details of a Test Run

(i.e., result of this Test in the scope of the Test Execution)



The screenshot shows the 'Test details' page for a specific test run. It includes a 'Findings' section with a 'Test issue LINK' and a 'Results' section with a 'FAILED' status. A red box highlights the 'FAILED' status and the 'Test issue LINK'.

Rank	Key	Summary	Test Type	TestRun Assignee	Linked Issues	Status	Actions
1	\$7-5	getDefaulGreeting	Generic	Sergio Freire		PASSED	
2	\$7-6	createUserWithSuccess	Generic	Sergio Freire	\$7-2	PASSED	
3	\$7-7	deleteUserUnsuccess	Generic	Sergio Freire	\$7-2	PASSED	
4	\$7-8	getPersonalizedGreeting	Generic	Sergio Freire		PASSED	
5	\$7-9	getWelcomeMessage	Generic	Sergio Freire		PASSED	
6	\$7-10	testUsersWithSuccess	Generic	Sergio Freire	\$7-2, \$7-2	FAILED	

Integration with Jira and Xray

Goal 2: Track coverage related with test automation

- What “requirements” are covered by automated tests?
- What is the status of those “requirements” given the test automation results?

Test Run details screen

Sergio Freire

Story (or Epic) issue screen

(in this case it is shown as NOK because the latest result failed for one of the Tests covering it)

As third party system, I can manage users through the REST API

Attach Create subtask Link issue Test Coverage

Description

As third-party system, I can manage users through the REST API.

ACs:

- ability to create a user
- ability to delete users
- ability to get all users
- ability to get a specific user

Linked Issues

is tested by

ID	Title	Status	Assignee	Due Date
S7-7	deleteUserUnsuccess	PASSED	[Avatar]	10:00 AM
S7-12	dontCreateUserForInvalidData	PASSED	[Avatar]	10:00 AM
S7-10	setAllUsersWithSuccess	PASSED	[Avatar]	10:00 AM
S7-6	createUserWithSuccess	PASSED	[Avatar]	10:00 AM
S7-11	getUserUnsuccess	PASSED	[Avatar]	10:00 AM
S7-13	deleteUserWithSuccess	PASSED	[Avatar]	10:00 AM
S7-15	getUserWithSuccess	PASSED	[Avatar]	10:00 AM

Test Coverage

Add Tests Execute

Analysis & Scope

Scope: Latest Final Status

Status	Key	Summary	Test Status
PASSED	S7-6	createUserWithSuccess	PASSED
PASSED	S7-7	deleteUserUnsuccess	PASSED
PASSED	S7-10	setAllUsersWithSuccess	FAILED

What test automation frameworks/tools are supported?

You can use any language for developing your automated tests.

Xray processes the reports generated by your test runner, which is usually part of the automation framework.

Your automated scripts can be developed in any language (or tool) if you generate a supported test report ([see here](#)).

Many test runners and tools can generate reports in the "standard" JUnit XML format, which Xray is able to process. Xray also processes a custom, enhanced JUnit XML report containing additional information tailored for it (e.g., issue key of covered Story issue, comments, evidence).

What's the flow?

There are 2 main flows:

1. Common flow (i.e., based on JUnit XML reports)
2. Cucumber specific flows
 - a. To be able to have full integration with Cucumber, it requires some additional step; however, we can follow the “common flow” to keep it simple and if having visibility of overall status of test result is enough; more info ahead

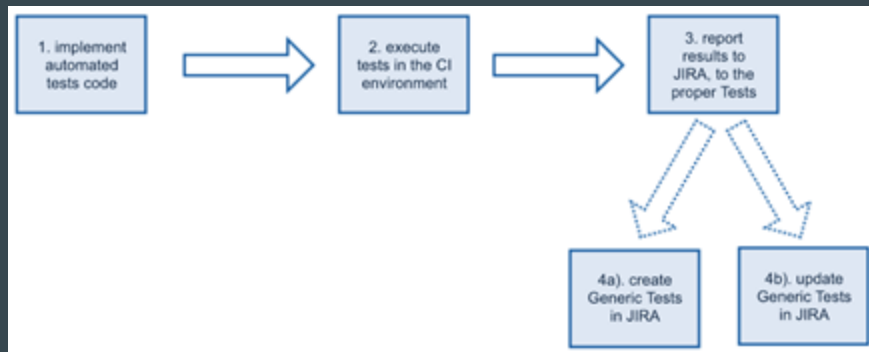
Common flow for having visibility of test automation results

1. Implement test automation code (e.g., in Java + JUnit)
 - a. Use whatever IDE, language and framework; store it in the SCVS (e.g., git)
2. Run the tests, usually in CI/CD
 - a. Using Maven, for example
 - b. Produce a compatible test report; most test runners can output JUnit XML reports as a last resource
3. Push results to Xray
 - a. From the pipeline (or even from your local machine, invoking [Xray's REST API](#))

Common flow for JUnit, TestNG and similar test results

In Xray we don't need to do anything beforehand; we just import the results to it.

- Tests are auto-provisioned the first time results are imported; from them onwards, Tests are reused (just new Test Runs will be created for these Tests)

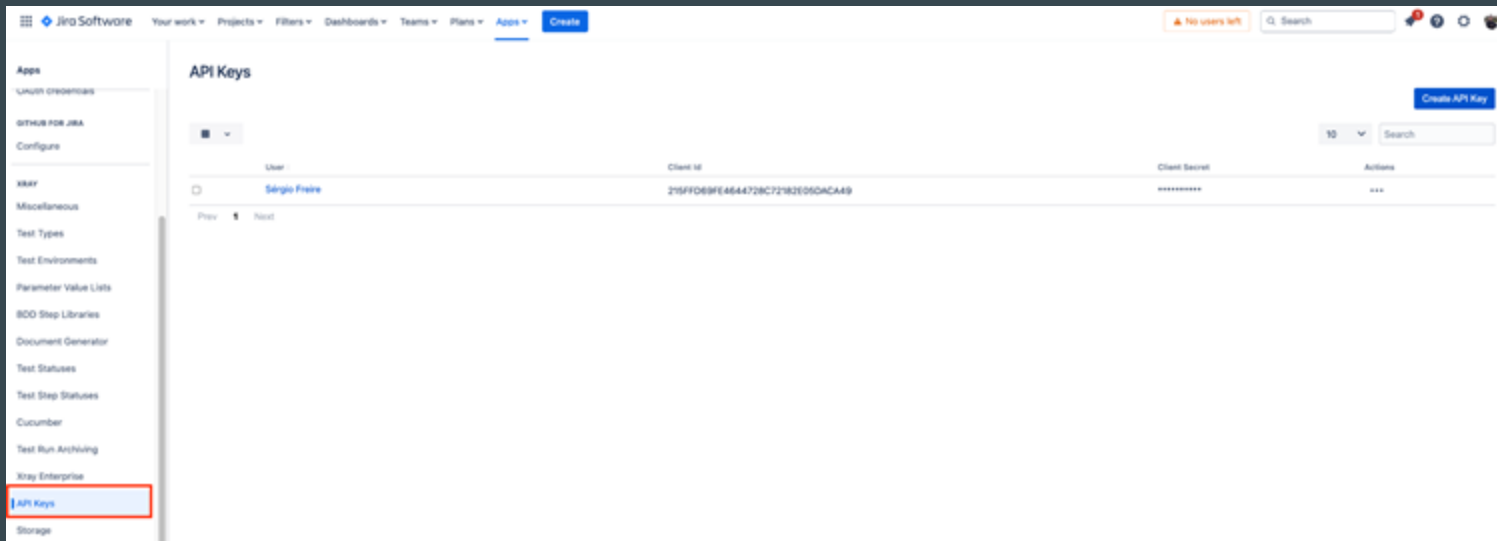
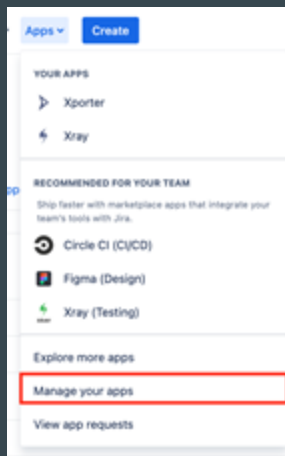


Optionally (depends on framework),

- We can link tests to existing Story issues (i.e., cover them)
- We can report results against existing Tests

Before importing results

Create an “API key” (i.e., a pair of *client id* and *client secret*) to be used on Xray API authentication requests. Go to **Apps > Manage your apps > Xray > Api Keys**.



Submitting test results

To submit test automation results to Xray/Jira we have several options:

- a. Using a plugin for CI/CD tools, like the free [xray-action](#) GitHub action (see a [tutorial](#))
- b. Using a Maven plugin: [xray-maven-plugin](#) ([example of a GH workflow in a sample project](#))
- c. Invoking Xray's [REST API](#) endpoints for importing results directly

Submitting test results through the REST API (using “curl”)

To submit results to Xray using its REST API, we need to obtain a token, based on the client id/secret, and then use it on the request that sends the test results to Xray on Jira.

We need to specify the target Jira project, by its key. Optionally, we can also specify a Test Plan by its issue key; this Test Plan needs to exist beforehand (create it manually in Jira). Test Plan is used to group the results from multiple “builds”/iterations.

```
token=$(curl -H "Content-Type: application/json" -X POST --data '{ "client_id": "<CLIENT_ID>", "client_secret": "<CLIENT_SECRET>" }' https://xray.cloud.getxray.app/api/v2/authenticate)

curl -H "Content-Type: text/xml" -X POST -H "Authorization: Bearer $token" --data @"reports/TEST-junit-jupiter.xml" https://xray.cloud.getxray.app/api/v2/import/execution/junit?projectKey=XPT0&testPlanKey=XPT0-123
```

Note: this way of submitting results may be mostly useful for debugging purposes.

What happens in Jira side?

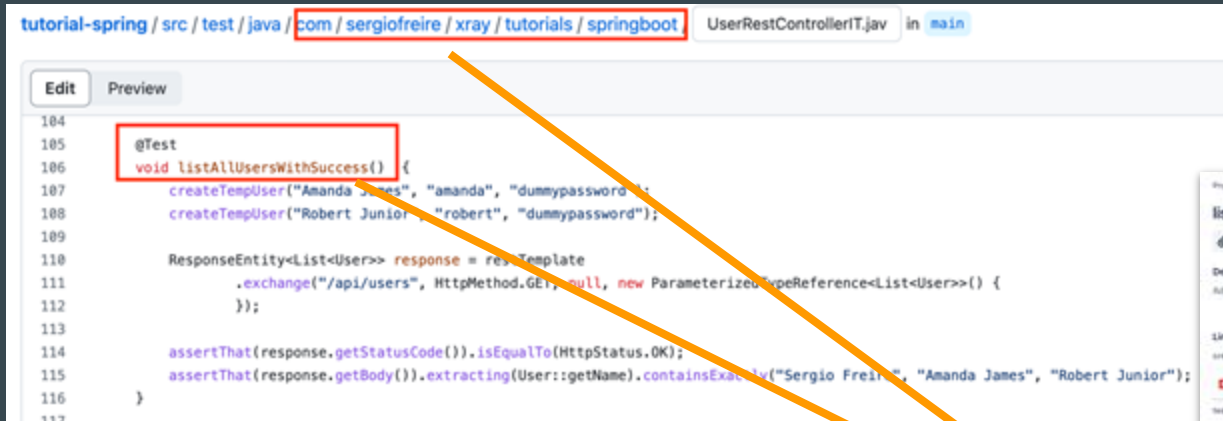
- A Test Execution issue is created on the target Jira project, containing results for the tests contained in the submitted test report file
- Test issues will be created as abstraction of the original test methods; these are only created unless they don't exist yet

The image shows two screenshots from the Jira interface. The left screenshot displays the 'Execution results' page for issue ID 708453141445. It shows a green bar indicating '11 PASSED' tests. Below this, a table lists individual test results, with a red box highlighting the first five tests, all of which are 'PASSED'. The right screenshot shows the 'Test Plan' view for the same issue, with a red box highlighting the 'Test Plan' tab. An orange arrow points from the 'Test Plan' tab in the right screenshot to the 'Test Plan' tab in the left screenshot, indicating the relationship between the two views.

Test ID	Test Name	Status
X7-676	getPersonalizedGreeting	PASSED
X7-677	donCreateUserPartiallyData	PASSED
X7-678	getUsersSuccess	PASSED
X7-679	deleteUserWithSuccess	PASSED
X7-680	createUserWithSuccess	PASSED

If you associated the results to a Test Plan, on it you can see the all the previous results (i.e., Test Executions).

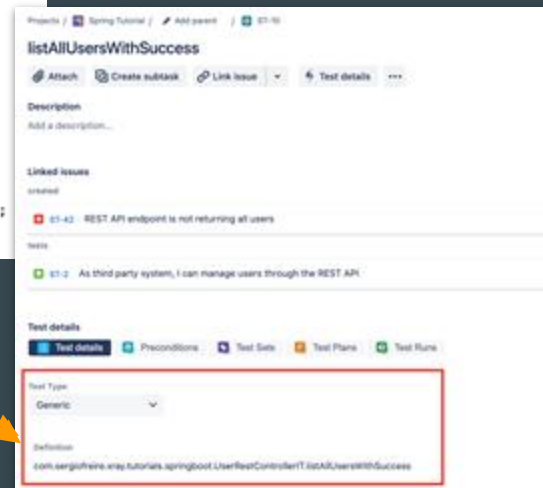
Auto-provisioning of Tests: how does it work?



```
tutorial-spring / src / test / java / com / sergiofreire / xray / tutorials / springboot UserRestControllerIT.java in main

Edit Preview

104
105
106 @Test
107 void listAllUsersWithSuccess() {
108     createTempUser("Amanda James", "amanda", "dummyspassword");
109     createTempUser("Robert Junior", "robert", "dummyspassword");
110
111     ResponseEntity<List<User>> response = restTemplate
112         .exchange("/api/users", HttpMethod.GET, null, new ParameterizedTypeReference<List<User>>() {
113         });
114
115     assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
116     assertThat(response.getBody()).extracting(User::getName).containsExactly("Sergio Freire", "Amanda James", "Robert Junior");
117 }
```



Auto-provisioning of Tests is based on the package name and the test method name.

If you change any of those, you'll have duplicated Tests (you'll need to delete them in Xray).

Linking to user stories

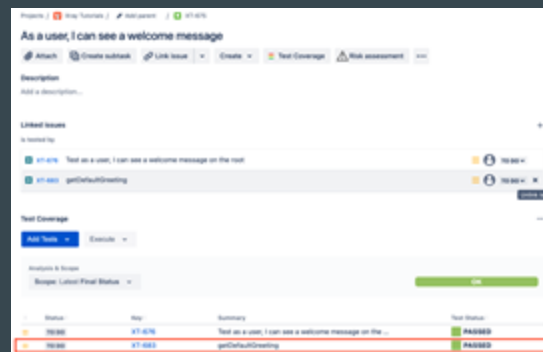
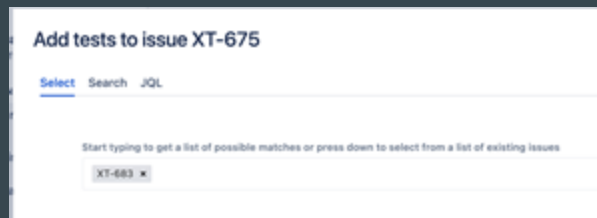
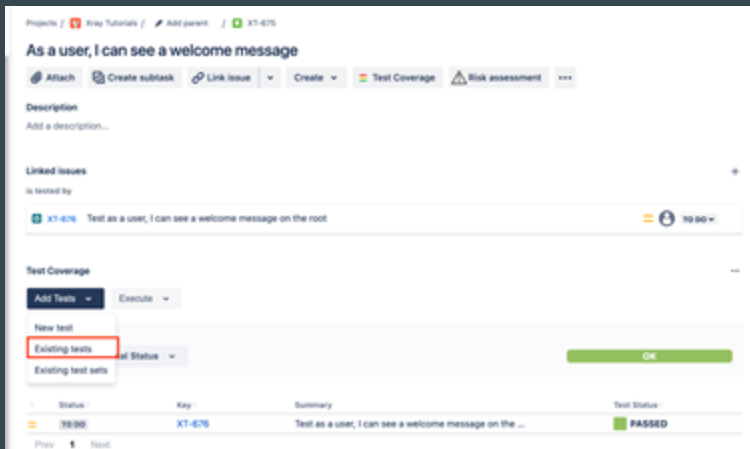
You're writing the test automation code; how do you link it to the user stories, to let Xray know that you're covering that user story with that test?

There are 2 options:

1. Use the standard JUnit XML report produced by surefire or failsafe, upload it as usual, and then link each Test to the Story manually (you'll need to do this for each Test, just once)
2. Use a JUnit 5 extension “xray-junit-extensions”, that provides additionally features

Option 1: manually linking Tests to the user stories

After you imported results at least once, go to the Story issue, and choose **Add Tests > Existing tests**. After this, the Test will appear under the Coverage section.



Note: If we wish to remove/dissociate in terms of coverage, we need to remove the proper issue link in the “Linked Issues”.

Option 2: linking Tests to the user stories right from the test code

Use the [xray-junit-extensions](#) which provides the ability to link a test to an existing “requirement” (e.g., Story, Epic) in Jira, using the **@Requirement** annotation.

```
import app.getxray.xray.junit.customjunitxml.annotations.Requirement;
...

@Test
@Requirement("ST-2") // issue key of related Story issue covered by this test
void listAllUsersWithSuccess() {
    createTempUser("Amanda James", "amanda", "dummyspassword");
    createTempUser("Robert Junior", "robert", "dummyspassword");

    ResponseEntity<List<User>> response = restTemplate
        .exchange("/api/users", HttpMethod.GET, null, new ParameterizedTypeReference<List<User>>() {
        });

    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
    assertThat(response.getBody()).extracting(User::getName).containsExactly("Sergio Freire", "Amanda James", "Robert Junior");
}
```

Ultimately, this Maven plugin will generate an enhanced JUnit XML like report (alternative to the the surefire/failsafe XML reports) with additional information that Xray can take advantage of. It doesn't communicate with Jira or Xray at all!

Option 2: setting up the “xray-junit-extensions”

Add the following dependency to your pom.xml file.

```
<dependencies>
  <dependency>
    <groupId>app.getxray</groupId>
    <artifactId>xray-junit-extensions</artifactId>
    <version>0.8.0</version>
    <scope>test</scope>
  </dependency>
  ...
  <dependency>
    <groupId>app.getxray</groupId>
    <artifactId>xray-maven-plugin</artifactId>
    <version>0.7.5</version>
  </dependency>
</dependencies>
```

Add the optional *xray-maven-plugin* dependency, if you want to push the results to Xray directly from Maven using a specific task.

Option 2: setting up the “xray-junit-extensions”

In order to generate the enhanced, customized JUnit XML report we need to register a specific Test Execution Listener: `EnhancedLegacyXmlReportGeneratingListener`.

- Create a file `src/test/META-INF/services/org.junit.platform.launcher.TestExecutionListener` with the following content:

```
app.getxray.xray.junit.customjunitxml.EnhancedLegacyXmlReportGeneratingListener
```

- By default, the report will be generated under the directory `target/` (`./target/TEST-junit-jupiter.xml`)

More info on the [xray-junit-extensions GitHub project](#).

Tutorials

Check these resources:

- [Tutorial](#) showcasing Spring and integration with Xray
- [Sample project on GitHub](#) showcasing testing Spring apps, fully integrated with Xray (using the xray-junit-extensions to provide specific annotations and to generate an enhanced JUnit XML report with this information that Xray can process)

What about Cucumber?

Cucumber and similar frameworks are a bit different; Cucumber and other Gherkin-based frameworks have specification (based on Gherkin) and the underlying implementation (code).

We have two options:

- I. Treat them as typical automated tests (as mentioned earlier for JUnit), where we just aim to have overall visibility of the test results in Jira
 - A. Generate JUnit XML test report (has less detail, but easier to implement)
 1. Xray will process it as described earlier for standard JUnit tests; in Xray we'll have overall information about the test result but we won't have Gherkin steps based information
 2. This is simpler to implement in CI as we just need to push the JUnit XML report(s)
- II. Handle Cucumber specifics
 - A. Generate a Cucumber JSON test report (has more detail, but more work for setup)
 1. This report contains Gherkin statement level information; however, we cannot simply import the results
 2. We need to have Cucumber' Scenarios as Test issues in Xray; we decide the master for editing the Scenarios and then implement the proper flow as consequence of that (see [tutorial](#))

1. Handling Cucumber similar to regular JUnit tests

We can upload the surefire/failsafe JUnit XML tests, as usual; Xray will create Test entities (of type “generic”), if needed, one per each Cucumber Scenario.

The screenshot shows the Xray interface with the following components:

- Execution results [1714405750095]**: Includes buttons for Attach, Create subtask, Link issue, Create, Tests, and Risk assessment.
- Description**: Add a description...
- Environment**: None
- Tests**: Add Tests, Trigger Build
- Overall Execution Status**: 2 PASSED, TOTAL TESTS: 2
- Table**:

Rank	Key	Summary	Test Type	Dataset	#Defects	TestRun Assignee	Priority	Linked Issues	Status	Actions
1	CALC-355	Successful purchase flight	Generic		0	Sergio Freire			PASSED	
2	CALC-356	Successful purchase flight using declarative style	Generic		0	Sergio Freire			PASSED	

An orange arrow points from the 'Generic' test type in the 'Test details' panel to the 'Generic' test type in the table.

The Test Run (the result report for the Test) just has overall status and elapsed time; there is no Gherkin statement level information; this may be enough though.

The screenshot shows the Xray interface with the following components:

- Successful purchase flight**: Includes buttons for Attach, Create subtask, Link issue, Test details, and Risk assessment.
- Description**: Add a description...
- Environment**: None
- Test details**: Includes buttons for Test issues, Preconditions, Test Data, Test Plans, and Test Runs.
- Test Run**: Includes buttons for Test issues, Preconditions, Test Data, Test Plans, and Test Runs.
- Test Run**: Includes buttons for Test issues, Preconditions, Test Data, Test Plans, and Test Runs.
- Test Run**: Includes buttons for Test issues, Preconditions, Test Data, Test Plans, and Test Runs.

2. Handling Cucumber specifics

Scenarios (and Backgrounds) must exist in Xray, so that you can report results against them; these cannot be created automatically during the import of test results.

Where are you going to make the Gherkin specification of the Cucumber (test) Scenarios?

Specification may be managed/edited with Xray or not; use one of these approaches (not both):

- A. Xray as the master for editing Scenarios/Backgrounds (as Test and Precondition issues, respectively)
- B. VCS (e.g. Git) as the master to store Scenarios/Backgrounds, while the edition is done by some external IDE (e.g. Eclipse, IntelliJ IDEA, Sublime, VSCode, etc)

In sum, we need to decide (A or B) where we will edit the Cucumber related tests; that will affect the flow we need to implement.

Cucumber: generating the JSON report

To be able to show Gherkin statement/step level information in Xray, we need to generate a Cucumber JSON report.

We can specify that we want the Cucumber JSON report right from the maven CLI, or using the `@ConfigurationParameter` annotation on the test class.

```
mvn test -Dtest=pt.ua.sergiofreire.blazedemo.BlazedemoTest -Dcucumber.plugin="json:target/report.json" ...
```

```
import static io.cucumber.junit.platform.engine.Constants.GLUE_PROPERTY_NAME;

import org.junit.platform.suite.api.ConfigurationParameter;
import org.junit.platform.suite.api.IncludeEngines;
import org.junit.platform.suite.api.SelectClasspathResource;
import org.junit.platform.suite.api.Suite;

@Suite
@IncludeEngines("cucumber")
@SelectClasspathResource("pt/ua/sergiofreire/blazedemo")
@ConfigurationParameter(key = GLUE_PROPERTY_NAME, value = "pt.ua.sergiofreire.blazedemo")
@ConfigurationParameter(key = Constants.PLUGIN_PROPERTY_NAME, value = "pretty, json:target/report.json")
public class BlazedemoTest {
}
```

Warning: Cucumber with JUnit 4

If you see code similar to this, then it's for JUnit 4 and not JUnit 5.

```
package webdemo;

import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(plugin = {"pretty"})
public class RunCucumberTest {

}
```

Cucumber related tutorials

Check these:

- [Tutorial](#) detailing the 2 possible flows related with Cucumber and Xray integration
- [GH repo using Cucumber + Selenium](#), supporting the 2 flows

Reporting

The different ways of reporting

Xray provides several ways of reporting:

- Built-in, tailored reports ([more info](#)) => mostly for your awareness
 - Test Coverage and Traceability Reports are 2 of the most important ones, but there's more
- Gadgets to be used on standard Jira dashboards ([more info](#))
 - The way to share information in and between teams, in near real-time

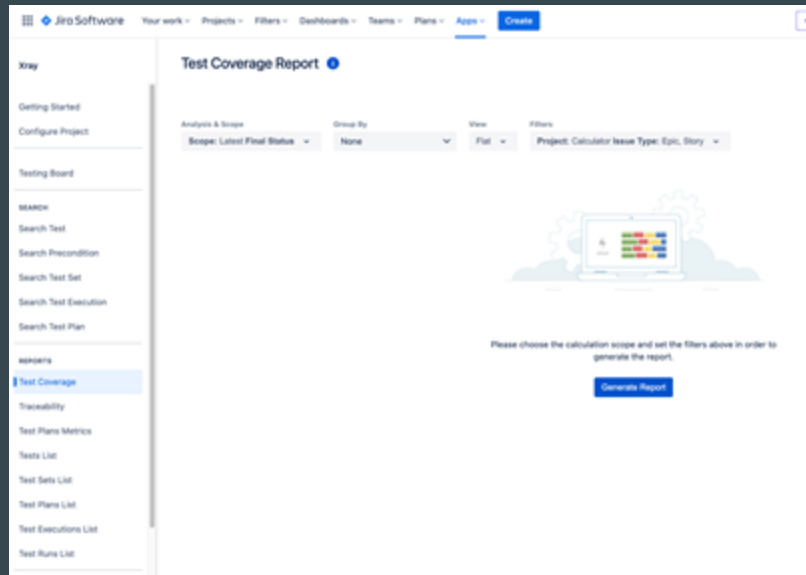
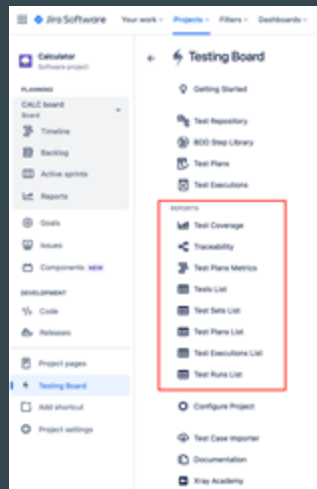
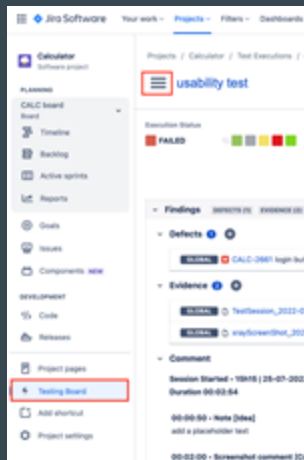
Some built-in reports are also available as gadgets but not all of them. Teams usually use a mix of approaches, depending on their needs.

Built-in Reports (a few examples)

Accessing built-in reports

Built-in reports can be accessed from several places.

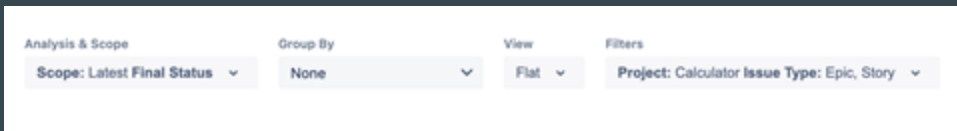
- Testing Board > (hamburger menu) > *report*
- Apps > Xray > Reports > *report*



Typical actions/inputs on built-in reports

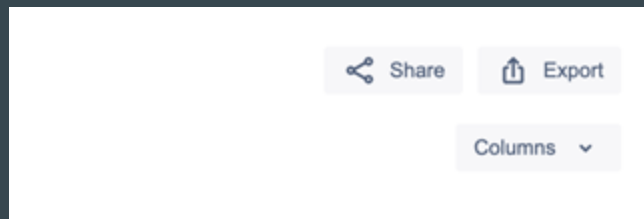
Left side

- Analysis & Scope
 - How to analyze the given data
- Filters
 - What data to use as source data
- Group by
- View



Right side

- Export (to CSV)
- Share
 - Create a unique URL that can be shared
- Columns / Show
 - show/hide some information



Built-in report: Traceability

Provides traceability from “requirements” up to defects.

Traceability works in the forward way:

Parent requirement > sub requirement(s) > Test(s) > Test Run(s) > Defect(s)

Epic > Stories > Tests > Test Runs > Bugs

Built-in report: Traceability

It's dynamic: depends on scope (i.e., the version and environment we want to analyze it on)

Quick filters for requirements based on their coverage status on the selected “scope” (e.g. version and/or environment; or even just using a Test Plan)

- *Show me just the requirements having problems...*
- *Show me just the uncovered issues...*

The screenshot displays a test management interface with a 'Requirements' tab selected. At the top, there are filters for 'Scope: Version: v1.0 Final Status: -' and 'Project: Calculator Issue Type: Epic, Story Labels: MGT'. Below these, a 'Quick Filters' section is highlighted with a purple box, showing 'OK (1)', 'NOK (1)', 'NOT RUN (5)', 'UNKNOWN (5)', and 'UNCOVERED (3)'. The main table lists requirements and their associated tests. Requirements include 'CALC-1120', 'CALC-2657', 'CALC-795', 'CALC-804', 'CALC-1427', 'CALC-1428', 'CALC-1429', 'CALC-1430', 'CALC-1431', 'CALC-1432', 'CALC-1433', 'CALC-1434', 'CALC-1435', 'CALC-1436', 'CALC-1437', 'CALC-1438', 'CALC-1439', 'CALC-1440', 'CALC-1441', 'CALC-1442', 'CALC-1443', 'CALC-1444', 'CALC-1445', 'CALC-1446', 'CALC-1447', 'CALC-1448', 'CALC-1449', 'CALC-1450', 'CALC-1451', 'CALC-1452', 'CALC-1453', 'CALC-1454', 'CALC-1455', 'CALC-1456', 'CALC-1457', 'CALC-1458', 'CALC-1459', 'CALC-1460', 'CALC-1461', 'CALC-1462', 'CALC-1463', 'CALC-1464', 'CALC-1465', 'CALC-1466', 'CALC-1467', 'CALC-1468', 'CALC-1469', 'CALC-1470', 'CALC-1471', 'CALC-1472', 'CALC-1473', 'CALC-1474', 'CALC-1475', 'CALC-1476', 'CALC-1477', 'CALC-1478', 'CALC-1479', 'CALC-1480', 'CALC-1481', 'CALC-1482', 'CALC-1483', 'CALC-1484', 'CALC-1485', 'CALC-1486', 'CALC-1487', 'CALC-1488', 'CALC-1489', 'CALC-1490', 'CALC-1491', 'CALC-1492', 'CALC-1493', 'CALC-1494', 'CALC-1495', 'CALC-1496', 'CALC-1497', 'CALC-1498', 'CALC-1499', 'CALC-1500'. Tests include 'CALC-797', 'CALC-1427', 'CALC-1428', 'CALC-1429', 'CALC-1430', 'CALC-1431', 'CALC-1432', 'CALC-1433', 'CALC-1434', 'CALC-1435', 'CALC-1436', 'CALC-1437', 'CALC-1438', 'CALC-1439', 'CALC-1440', 'CALC-1441', 'CALC-1442', 'CALC-1443', 'CALC-1444', 'CALC-1445', 'CALC-1446', 'CALC-1447', 'CALC-1448', 'CALC-1449', 'CALC-1450', 'CALC-1451', 'CALC-1452', 'CALC-1453', 'CALC-1454', 'CALC-1455', 'CALC-1456', 'CALC-1457', 'CALC-1458', 'CALC-1459', 'CALC-1460', 'CALC-1461', 'CALC-1462', 'CALC-1463', 'CALC-1464', 'CALC-1465', 'CALC-1466', 'CALC-1467', 'CALC-1468', 'CALC-1469', 'CALC-1470', 'CALC-1471', 'CALC-1472', 'CALC-1473', 'CALC-1474', 'CALC-1475', 'CALC-1476', 'CALC-1477', 'CALC-1478', 'CALC-1479', 'CALC-1480', 'CALC-1481', 'CALC-1482', 'CALC-1483', 'CALC-1484', 'CALC-1485', 'CALC-1486', 'CALC-1487', 'CALC-1488', 'CALC-1489', 'CALC-1490', 'CALC-1491', 'CALC-1492', 'CALC-1493', 'CALC-1494', 'CALC-1495', 'CALC-1496', 'CALC-1497', 'CALC-1498', 'CALC-1499', 'CALC-1500'. The 'UNCOVERED' filter is highlighted with a yellow arrow. A pink arrow points from the 'UNCOVERED' filter to the 'CALC-1461' requirement, which is marked as 'FAILED'.

Built-in report: Test Coverage

Provides a birds-eye overview of the (coverage) status of your deliverables (e.g., stories, epics) based on their latest testing results.

It answers this simple question: **What are the impacts of my testing?**

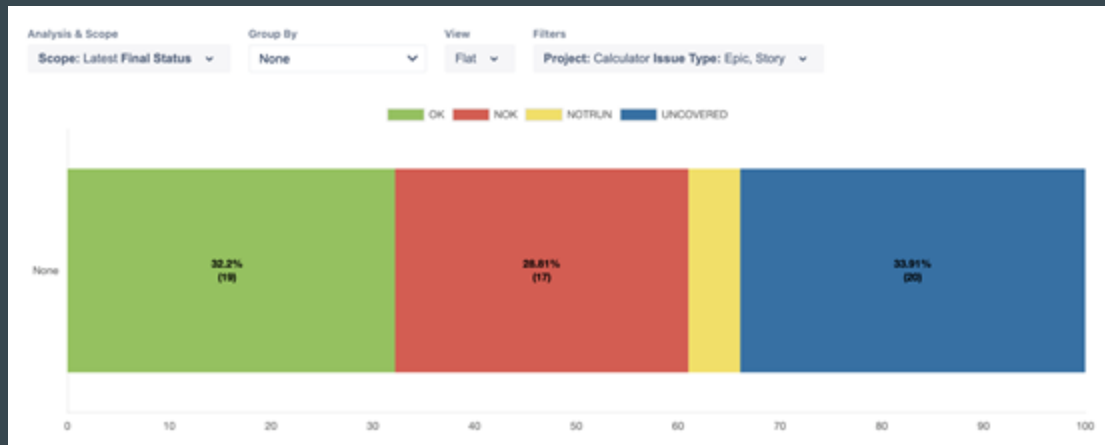
But also,

- *How are my stories on version 2.0 of the SUT?*
- *Are my highest priority stories covered by tests?*
- *How are the requirements related to some component?*
- *Which is the test completeness of certain requirements?*
- *...*

Built-in report: Test Coverage

It's dynamic: depends on scope (i.e., the version and environment we want to analyze it on), considering the latest results obtained on it

- Output will most likely be different on two different versions



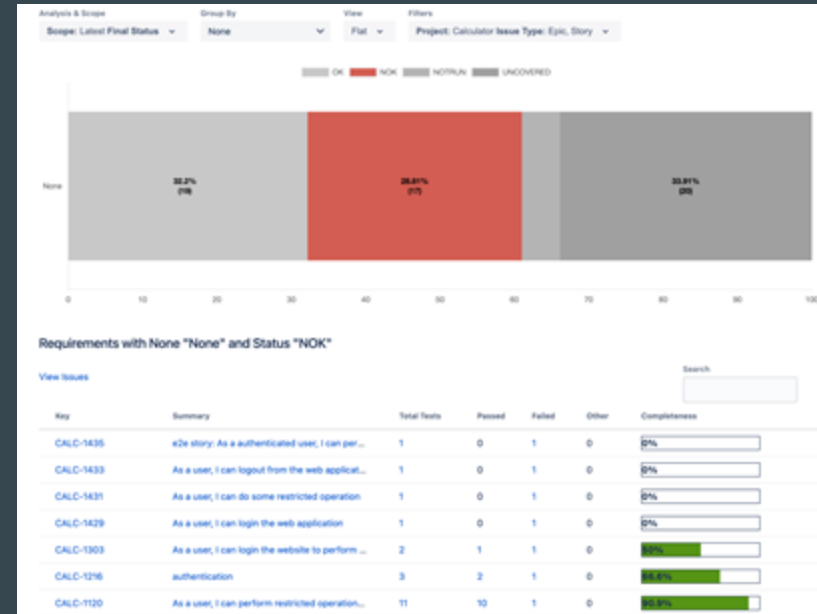
Status meaning:

- OK: covered, and all related tests are passing
- NOK: covered, but one of the related tests is failing
- NOTRUN: covered, but some of the related tests are yet to be run
- UNCOVERED: has not tests

Built-in report: Test Coverage

Drill-down, by clicking on a bar, to see:

- List of related “requirements”
- Test completeness of each requirement
- Total tests that cover each requirement



Built-in report: Test Coverage

Group by component, priority, or other compatible field on the “requirements”.



Jira dashboards with Xray Gadgets

Quality related information: quick considerations

What do we want to answer?

Who is going to look at it?

Is that information understandable and actionable by the team?

Quality related information

Focus on: testing results at high-level

- *What were the testing results, namely from test automation ran in the pipeline?*
- How to achieve it? In different ways depending on what we aim to show...
 - “Test Executions List” gadget, if we want to see each “batch” of test results (i.e., Test Execution issues), usually coming from a specific build on the pipeline
 - Here we see each build result
 - “Test Plans List” gadget, if we want to see the consolidated results from multiple “batch” of test results (i.e., from multiple builds on the pipeline)
 - If you create a Test Plan manually in Jira, and then report the results back to it, the Test Plan will show the consolidated results and on each test you can track the runs you obtained for it
 - Here we see the consolidated results from multiple builds
 - “Overall Test Results” gadget, if you want to show how are your tests, in a pie-chart, grouped by their latest result

Quality related information

Focus on: “Requirements” / epics & user stories

- *How are our deliverables given the latest testing results? Are they ready to be released?*
- How to achieve it?
 - “Overall Test Coverage” gadget, for an overall perspective at high-level
 - “Requirements List” gadget, for a detailed, per user story/epic, information

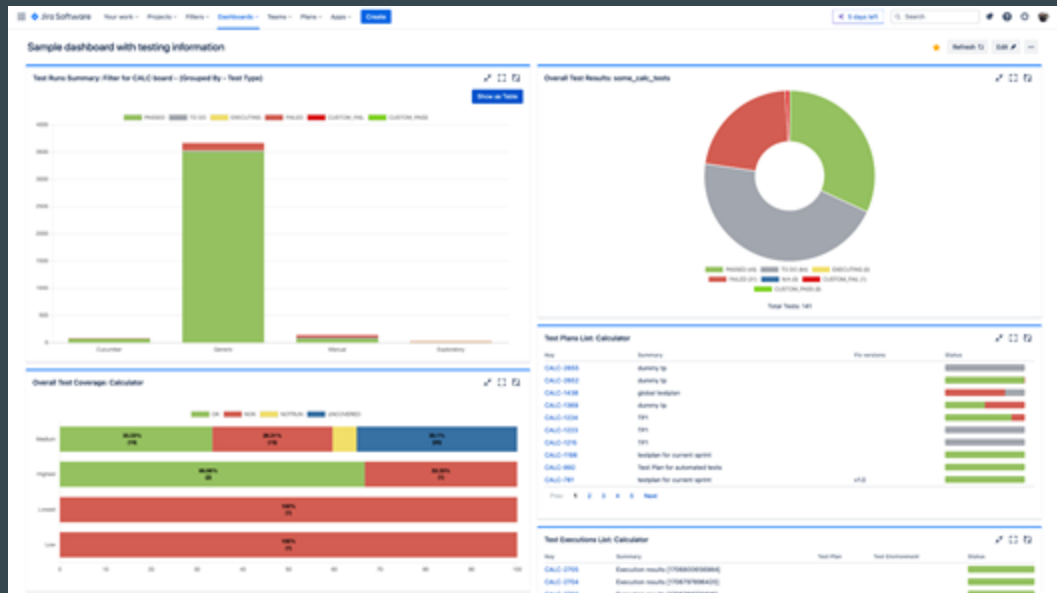
In this case we’re focused on the impacts of our testing. What does it tell about the quality of our deliverables?

Jira dashboards

The typical way to make shareable, near real-time reports, composed of multiple gadgets. Go to **Dashboards** top menu shortcut and create a new dashboard.

In a dashboard, you can use:

- Jira default gadgets
- Xray gadgets
- Gadgets from other apps



Xray gadgets

Listings:

- Requirements List
- Tests List
- Test Sets List
- Test Executions List
- Test Plans List
- Test Runs List

Aggregated or Calculated:

- Overall Test Coverage
- Overall Test Results
- Test Runs Summary
- Tests by Test Type

Xray gadgets: how source data is specified

Depends on gadget, but source data for gadgets can include:

- Saved filter
- Project
- Issue picker

And eventually...

- Test Runs related info (e.g., status, executed by, dates)
- Tests related info (e.g., priority, component)
- ...

Xray gadgets: Requirements List

Show list of requirements, some attributes, and the coverage status for some scope.

- Define the scope for calculating coverage status

Key	Summary	Priority	Components	Status
CALC-2703	As a user, I can login the web application	==		OK
CALC-2667	As a user, I can calculate the sum of 2 numbers	==		OK
CALC-2664	dummy story	==		UNCOVERED
CALC-1435	e2e story: As a authenticated user, I can perform so...	==		NOK
CALC-1433	As a user, I can logout from the web application	==		NOK
CALC-1431	As a user, I can do some restricted operation	==		NOK
CALC-1429	As a user, I can login the web application	==		NOK
CALC-1388	dummy story	==		NOTRUN
CALC-1345	dummy story	==		OK
CALC-1222	As system, the calculator changes to sleep mode wit...	==		UNCOVERED

Prev 1 2 3 4 5 6 Next

Requirements List: Calculator

Project or Saved Filter
Calculator

Number of results
10

Columns to display
Key Summary Priority Components

☒ Flat Test Coverage View

Analysis & Scope
Calculate the Test Coverage for the following scopes.

Latest **Version** Test Plan

Project
Calculator

Version
v1.0

Test Environment
All Environments

☒ Final statuses have precedence over non-final.

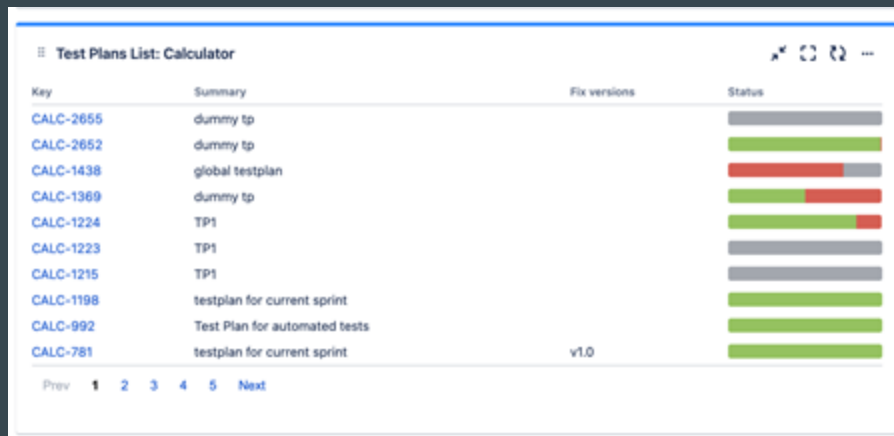
☒ Auto-generate gadget name
If this option is enabled, Xray will generate the gadget name based on its parent's 'Name' action.

Save Cancel

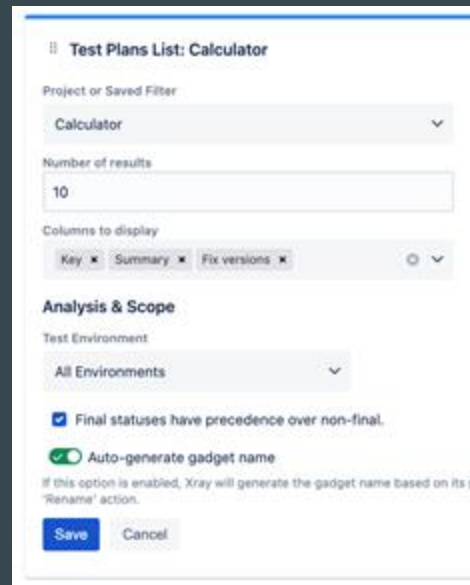
Xray gadgets: Test Plans List

Show a list of Test Executions, some attributes, and overall progress.

- Track consolidated progress considering the results on all Test Environments or just progress on a specific Test Environment
- Similar to the Test Plans List Report



Key	Summary	Fix versions	Status
CALC-2655	dummy tp		
CALC-2652	dummy tp		
CALC-1438	global testplan		
CALC-1369	dummy tp		
CALC-1224	TP1		
CALC-1223	TP1		
CALC-1215	TP1		
CALC-1198	testplan for current sprint		
CALC-992	Test Plan for automated tests		
CALC-781	testplan for current sprint	v1.0	



Test Plans List: Calculator

Project or Saved Filter
Calculator

Number of results
10

Columns to display
Key Summary Fix versions

Analysis & Scope

Test Environment
All Environments

☒ Final statuses have precedence over non-final.

☒ Auto-generate gadget name

If this option is enabled, Xray will generate the gadget name based on its parent's "Rename" action.

Save Cancel

Xray gadgets: Test Executions List

Show a list of Test Executions, some attributes, and overall progress.

- Track in which Test Plan, Test Env., Fix Version, Revision they were performed
- Similar to the Test Executions List Report

Key	Summary	Test Plan	Test Environment	Status
CALC-61	Test Execution of XRAYIntegration of the test WANI...	CALC-12	DEV	<div></div>
CALC-60	Test Execution of XRAYIntegration of the test WANI...	CALC-12	DEV	<div></div>
CALC-58	Exploratory session for CALC-57		CHROME	<div></div>
CALC-56	Ad-hoc execution for CALC-55			<div></div>
CALC-53	Exploratory session for CALC-52		FIREFOX	<div></div>
CALC-49	Exploratory session for CALC-46			<div></div>
CALC-47	Ad-hoc execution for CALC-46		CHROME	<div></div>
CALC-45	Ad-hoc execution for CALC-44			<div></div>
CALC-42	Exploratory session for CALC-41			<div></div>
CALC-39	Test Execution of XRAYIntegration of the test WANI...	CALC-12	DEV	<div></div>

Prev 1 ... 59 60 61 62 63 Next

Xray gadgets: Test Runs List

Show a list of Test Executions, some attributes, and overall progress.

- Track in which Test Environment, Fix Version, Revision they were performed
- Track the Test Run's assignee, executed by, etc

Key	Test Execution Key	Summary	Test Execution Version	Test Environment	Status
CALC-3	CALC-8	Cucumber Test As a user, I can sum two numbers	v3.0		PASSED
CALC-4	CALC-8	Generic Test As a user, I can sum two numbers	v3.0		PASSED
CALC-2	CALC-11	Manual Test As a user, I can sum two numbers	v3.0		PASSED
CALC-14	CALC-13	tete	v3.0		TO DO
CALC-29	CALC-30	Test epic1			CUSTOM_FAIL
CALC-17	CALC-33	WANImpact Local	v3.0	DEV	FAILED
CALC-44	CALC-45	risks around screensize			EXECUTING
CALC-46	CALC-47	assess login usability risks	v3.0	CHROME	FAILED
CALC-46	CALC-49	assess login usability risks	v3.0		EXECUTING
CALC-52	CALC-53	assess usability risks in the login page	v3.0	FIREFOX	FAILED

Test Runs List: Calculator

Project or Saved Filter: Calculator

Test Filter

Priority: Select... Component: Select...

Contains: Select... Status: Select...

Test Run Filter

Assignee: Type to search... Executed By: Type to search...

Status: Select...

☒ Enable date range filtering

Number of results: 10

Columns to display: Key, Test Execution Key, Summary, Test Execution Version, Test Environment

☒ Auto-generate gadget name

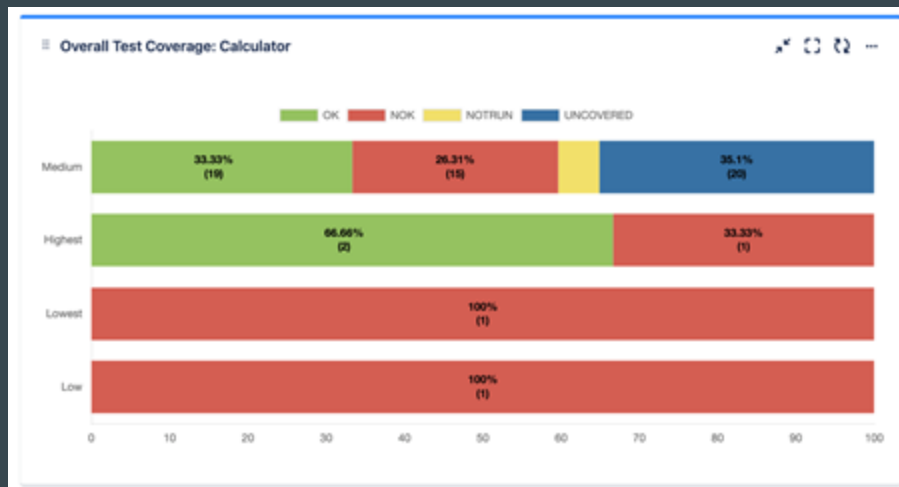
If this option is enabled, Xray will generate the gadget name based on its parameters. Otherwise, you can edit the name manually.

Save Cancel

Xray gadgets: Overall Test Coverage

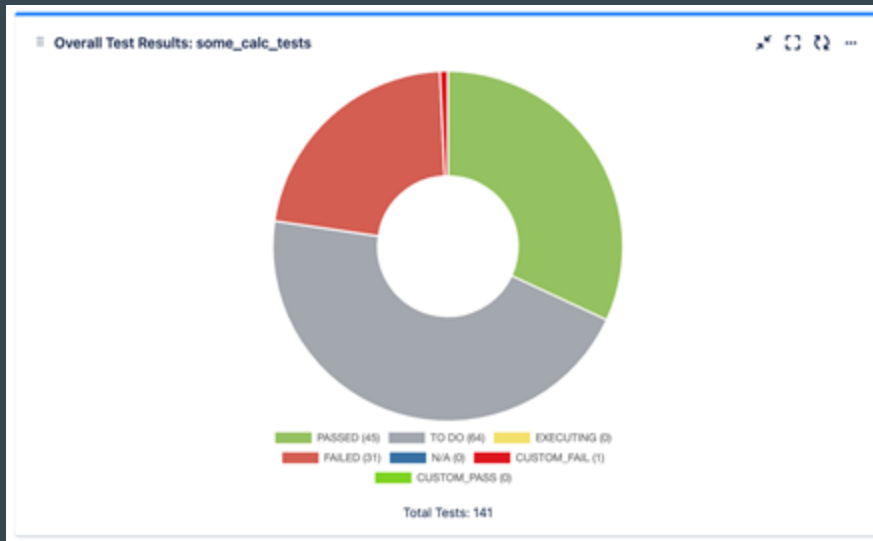
Provides a birds-eye overview of the (coverage) status of your deliverables (e.g., stories, epics) based on their latest testing results.

- Similar to the Test Coverage Report



Xray gadgets: Overall Test Results

Shows how the given Tests are in some “scope” (e.g., version and/or Test Env.).



Overall Test Results: some_calc_tests

Project or Saved Filter
some_calc_tests

Analysis & Scope
Calculate the Test Coverage for the following scopes.

Latest Version Test Plan

Test Environment
All Environments

☒ Final statuses have precedence over non-final.

☒ Auto-generate gadget name

If this option is enabled, Xray will generate the gadget name based on its explicitly define a name for the gadget using the "Rename" action.

Save Cancel

Example of dashboard

