

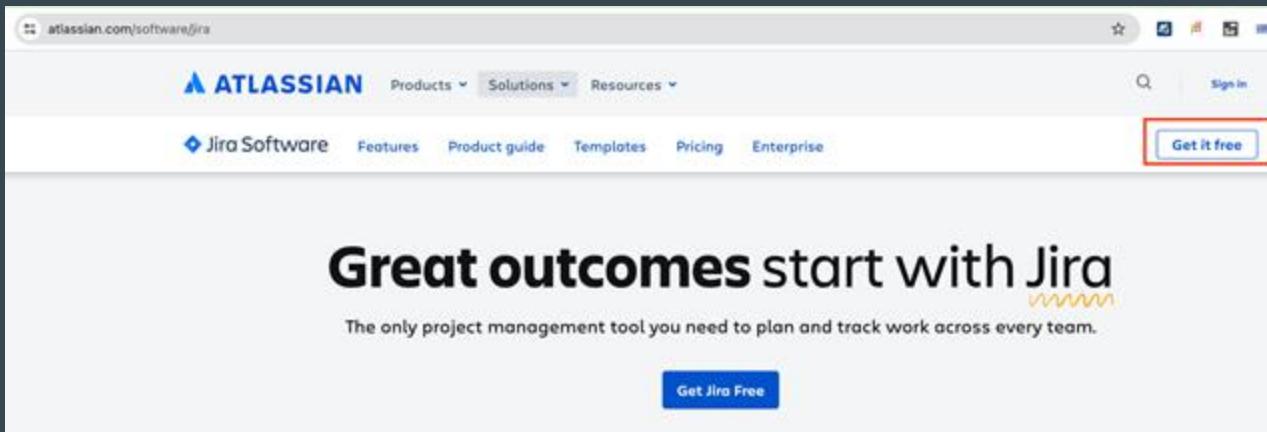
# Jira and Xray cloud setup

•••

Quick setup instructions of a Jira Software + Xray cloud instance

# Jira cloud setup

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



# Jira cloud signup

The image consists of three vertically stacked screenshots from the Jira Cloud sign-up process.

**Screenshot 1: Sign-up Form**

Jira logo

## Get started with Jira

It's free for up to 10 users — no credit card required

Work email

Find teammates, plus keep work and life separate by using your own email.

I agree to the [Atlassian Customer Agreement](#), which references the [Atlassian Product-Specific Terms](#), and acknowledge the [Privacy Policy](#).

**Sign up**

**Screenshot 2: Verification Code Message**

Jira logo

Your verification code is:

513 137

If you didn't try signing up, you can safely ignore this email.

This message was sent to you by Atlassian Cloud

A ATASSIAN

**Screenshot 3: Verification Code Entry**

## We've emailed you a code

To complete your account setup, enter the code we've sent to:

testing.uncovered+batman@gmail.com

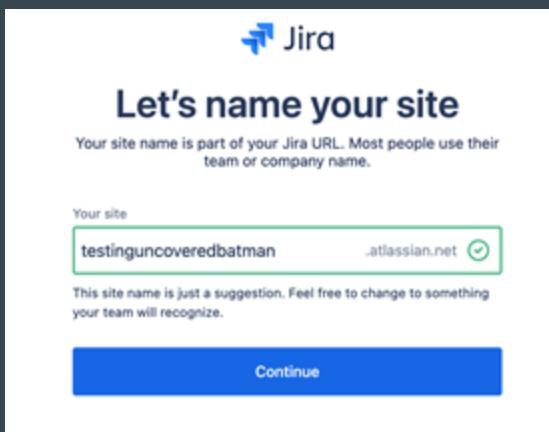
5    1    3    1    3    |

**Verify**

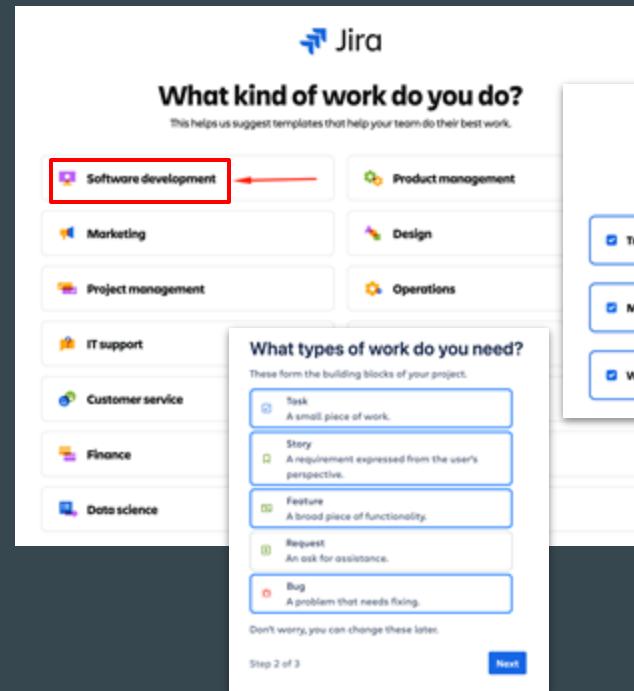
Didn't receive an email? [Resend email](#)

# Jira cloud signup

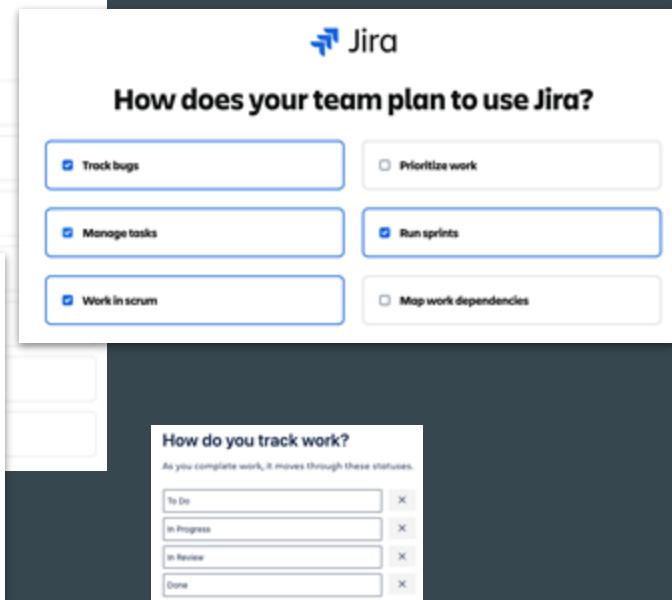
Take note of your Jira Cloud instance URL. You'll be asked some questions that you may skip.



The screenshot shows the first step of the Jira Cloud setup process. It features the Jira logo at the top left. Below it, the heading "Let's name your site" is displayed in a large, bold, dark blue font. A sub-instruction "Your site name is part of your Jira URL. Most people use their team or company name." is present. A text input field contains the placeholder "testinguncoveredbatman.atlassian.net" with a green checkmark icon indicating it's a valid suggestion. Below the input field, a note says "This site name is just a suggestion. Feel free to change to something your team will recognize." At the bottom is a large blue "Continue" button.



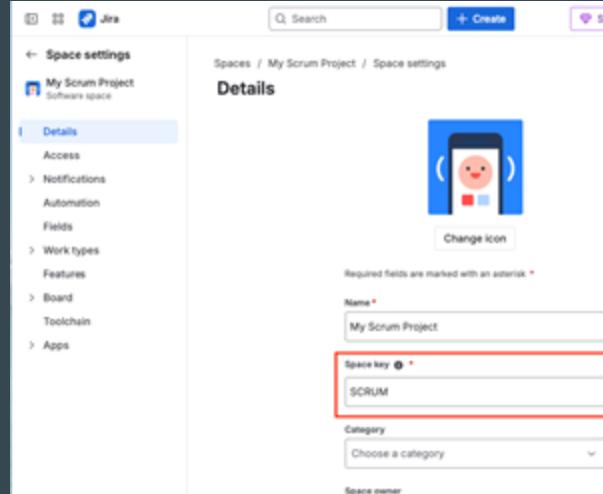
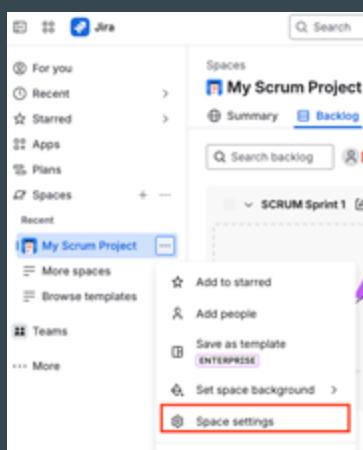
The screenshot shows the second step of the Jira Cloud setup process. The heading "What kind of work do you do?" is centered at the top. Below it, a sub-instruction "This helps us suggest templates that help your team do their best work." is shown. A grid of categories is displayed, with "Software development" highlighted by a red box and a red arrow pointing to it. Other categories include Marketing, Product management, Project management, Design, IT support, Operations, Customer service, Finance, and Data science. To the right of the grid, there's a section titled "What types of work do you need?" which lists "Task", "Story", "Feature", "Request", and "Bug" with their respective descriptions. At the bottom, a note says "Don't worry, you can change these later." and "Step 2 of 3". A "Next" button is located at the bottom right.



The screenshot shows the third step of the Jira Cloud setup process. The heading "How does your team plan to use Jira?" is centered at the top. Below it, a sub-instruction "These form the building blocks of your project." is shown. A grid of checkboxes is displayed, with several checked: "Track bugs", "Manage tasks", "Work in scrum", "Prioritize work", "Run sprints", and "Map work dependencies". To the right, there's a section titled "How do you track work?" which lists "To Do", "In Progress", "In Review", and "Done" with their respective descriptions. At the bottom, a note says "As you complete work, it moves through these statuses." and "Step 3 of 3". A "Next" button is located at the bottom right.

# 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-<int> (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:

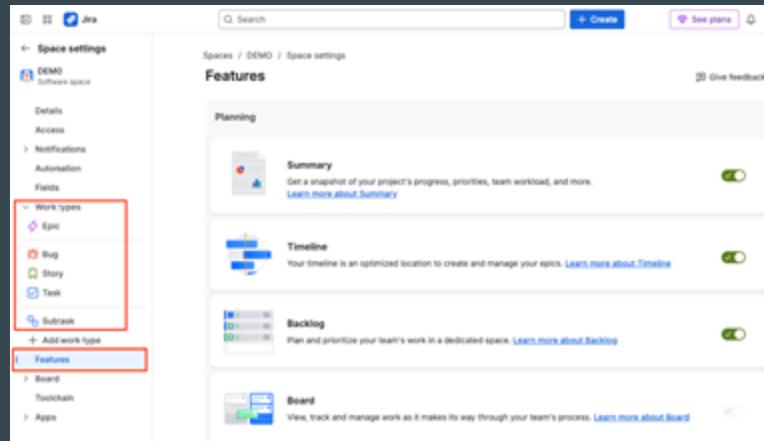
- **Team-managed** (simpler but not so powerful; good enough for us)
- Company-managed (more complex and more powerful)

The image consists of three side-by-side screenshots of the Jira Cloud interface.

- Left Screenshot:** Shows the 'Space templates' section under the 'Plans' category in the sidebar. 'Software development' is selected. Other options include 'Service management', 'Work management', 'Product management', 'Marketing', 'Human resources', 'Finance', and 'Design'. A 'Recent' section is also visible.
- Middle Screenshot:** Shows the 'Choose a project type' screen. It highlights the 'Team-managed' option, which is described as being "Set up and maintained by your team". It includes a 'Simplified configuration' section and a 'Scrum' template card.
- Right Screenshot:** Shows the 'Create project' form. The 'Name' field is filled with 'DEMO'. Under 'How your space is managed', 'Team-managed' is selected. The 'Access' dropdown is set to 'Open'. Under 'Template', 'Scrum' is selected. A 'See details' link is present.

# Final check

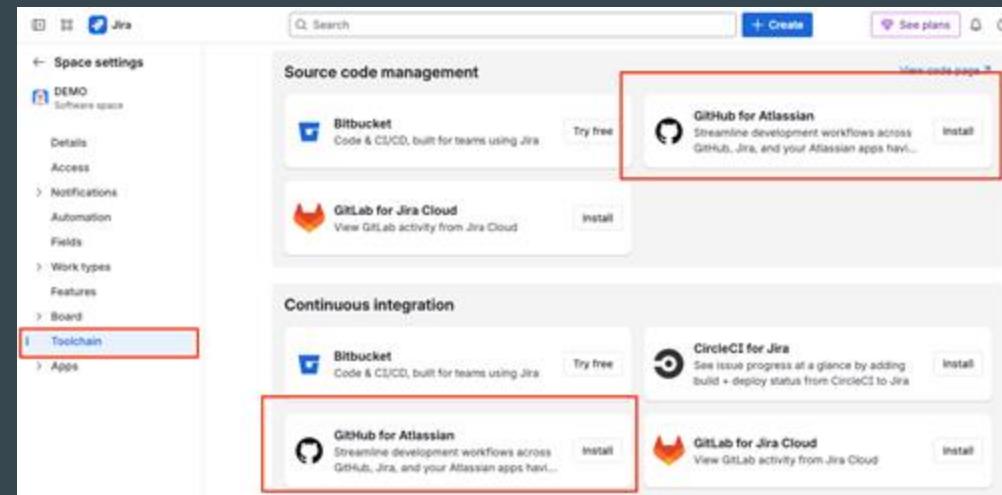
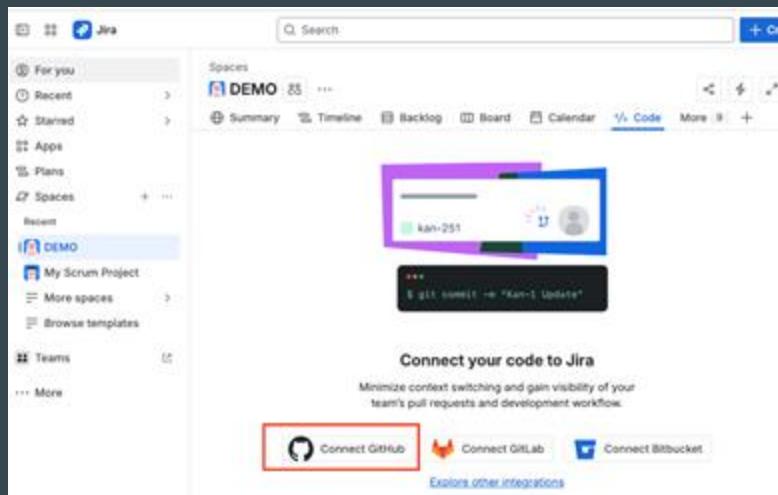
1. Open your space/project settings from the left bar, and go to **Features**. You should have all these, at least, enabled:
  - a. Summary, Backlog, Board, Reports, Sprints, Estimation, Code, Development
  - b. Optionally: Security, Deployments, Project pages
2. Check also that you have relevant “work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

The image is a collage of three screenshots from the Atlassian Marketplace. The top-left screenshot shows the 'Select a site' step, where a single site is selected. The top-right screenshot shows the 'Review and Install' step, displaying the GitHub for Atlassian app details, including a 2.7/4 rating, 504 reviews, and 143,234 installs. The bottom-right screenshot shows the 'Voilà! GitHub for Atlassian is now installed' confirmation screen, featuring a green checkmark icon in a shopping cart, and options to 'Configure app' or 'Maybe later'.

Select a site

Review and Install

Select a site

GitHub for Atlassian by Atlassian

2.7/4 ★ ★ ★ (504) | 143,234 | CLOUD FORTIFIED

Permissions

Actions

Data management

Data storage, sharing, receiving, and processing

Analytics and logs

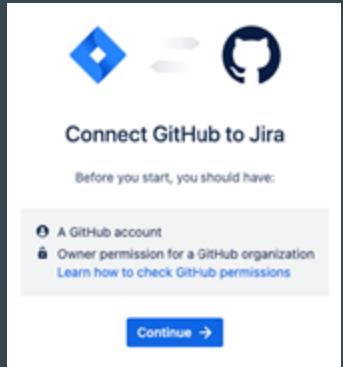
This app may send analytics and share logs outside of Atlassian.

Back Get it now

Voilà! GitHub for Atlassian is now installed

You must configure this app before it can send data to your Jira space.

Configure app Maybe later



## Connect GitHub to Jira

Select your GitHub product

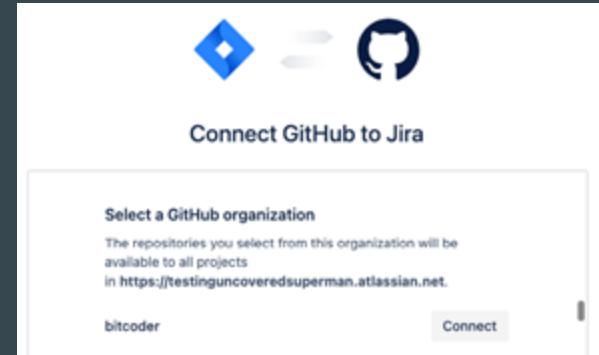
GitHub Cloud    GitHub Enterprise Server

How do I check my GitHub product?

[Next](#)

bitcoder

bitcoder is now connected!



# Finishing the configuration to give access to certain repositories

The image consists of two vertically stacked screenshots of the Jira software interface.

**Screenshot 1: Jira Marketplace Apps**

This screenshot shows the Jira sidebar on the left with various links like 'For you', 'Recent', 'Starred', and 'Explore more apps'. The 'Explore more apps' link is highlighted with a red box. The main content area is titled 'Explore apps for Jira' and lists 'Manage apps', 'App audit logs', and 'View app requests'. The 'Manage apps' link is also highlighted with a red box.

**Screenshot 2: GitHub Configuration**

This screenshot shows the 'Jira admin settings' sidebar on the left with 'Manage apps' selected. Below it, the 'GitHub configuration' page is displayed. The page title is 'GitHub configuration'. It contains instructions about connecting GitHub to Jira for viewing development activity and sending data from GitHub to Jira. A 'Connected organization' section shows 'bitcoder' is connected. A 'Repository access' section is highlighted with a red box and shows the status 'Only select repos 1'. Other sections include 'Organization sync status FINISHED', 'Backfill status FINISHED', 'Permissions FULL ACCESS', and a 'Configure' button at the bottom right.

# Check configuration on GitHub side...

The screenshot shows the GitHub user profile page for Sergio Freire (bitcoder). The sidebar on the left is titled "Atlassian" and lists various integration options. The main content area displays the configuration for the "Atlassian" app, which was installed last year. It highlights features like "Connect once, use everywhere" for streamlining software development across Atlassian apps, "Trusted, real-time context across your SDLC" for visibility of work and SDLC, and "Smarter automation, smoother development flow" for enhancing collaboration. The "Permissions" section shows two checked items: "Read access to Jira 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 is set to "Only select repositories", with "bitcoder/tutorial-spring" selected. A "Save" button is at the bottom.

Sergio Freire (bitcoder)  
Your personal account · Switch settings context · Go to your personal profile

Atlassian · Installed last year · Developed by Atlassian · https://github.com/marketingdrp/Software-GitHub

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

Public profile · Account · Appearance · Accessibility · Notifications · 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 · Applications

The screenshot shows the "Repository access" configuration dialog. It offers two options: "All repositories" (which applies to current and future repositories owned by the resource owner, including public repositories) and "Only select repositories" (which requires selecting at least one repository and includes public repositories). The "Only select repositories" option is selected, and "bitcoder/tutorial-spring" is listed as the chosen repository. A "Save" button is at the bottom.

Permissions

- Read access to Jira 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 · bitcoder/tutorial-spring · X

Selected 1 repository

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.

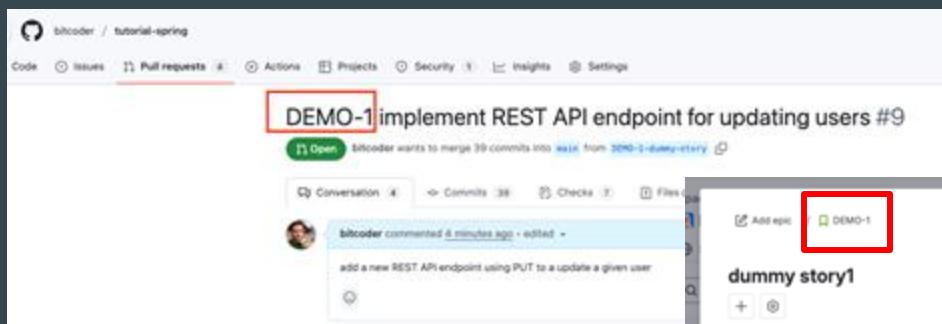
More info at: <https://support.atlassian.com/jira-software-cloud/docs/reference-issues-in-your-development-work/>

# 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"
```

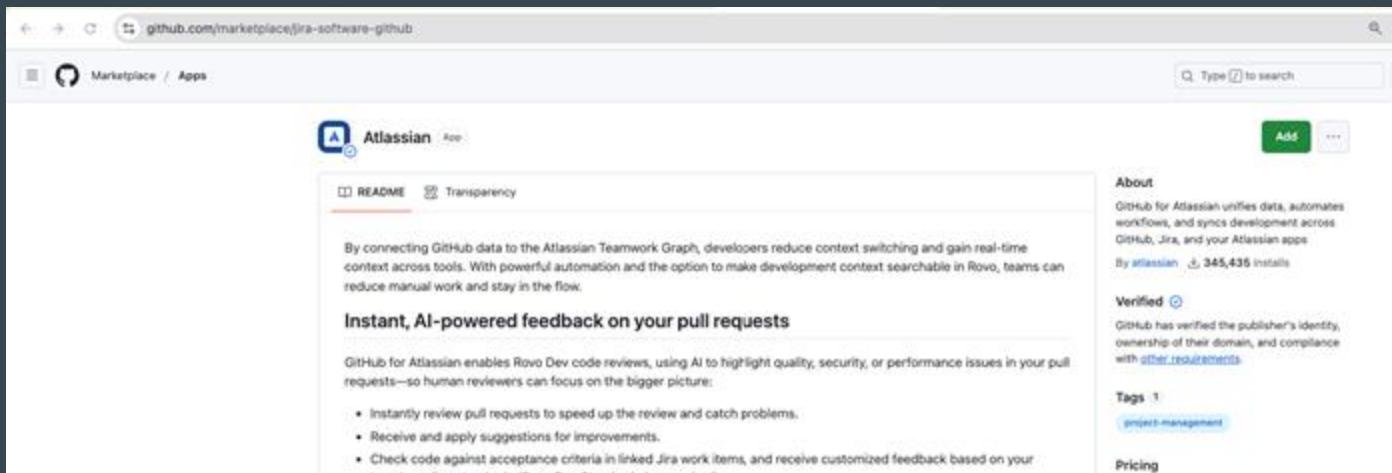


A screenshot of a Jira card titled 'dummy story1'. In the top right corner of the card header, there is a red box highlighting the text 'DEMO-1'. The card contains sections for 'Description' and 'Subtasks', both of which are currently empty. On the right side of the card, there is a 'Development' section with a red box highlighting its contents: 'Open with VS Code', '1 branch', '1 commit 15 minutes ago', '1 pull request OPEN', and '2 builds'.

A screenshot of a 'Create GitHub Branch' dialog box. The 'Branch name' field is highlighted with a red box and contains the text 'DEMO-1-dummy-story1'. Other fields in the dialog include 'Repository' set to 'bitcoder/tutorial-spring', 'Branch from' set to 'main', and a 'Create branch' button at the bottom right.

# 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

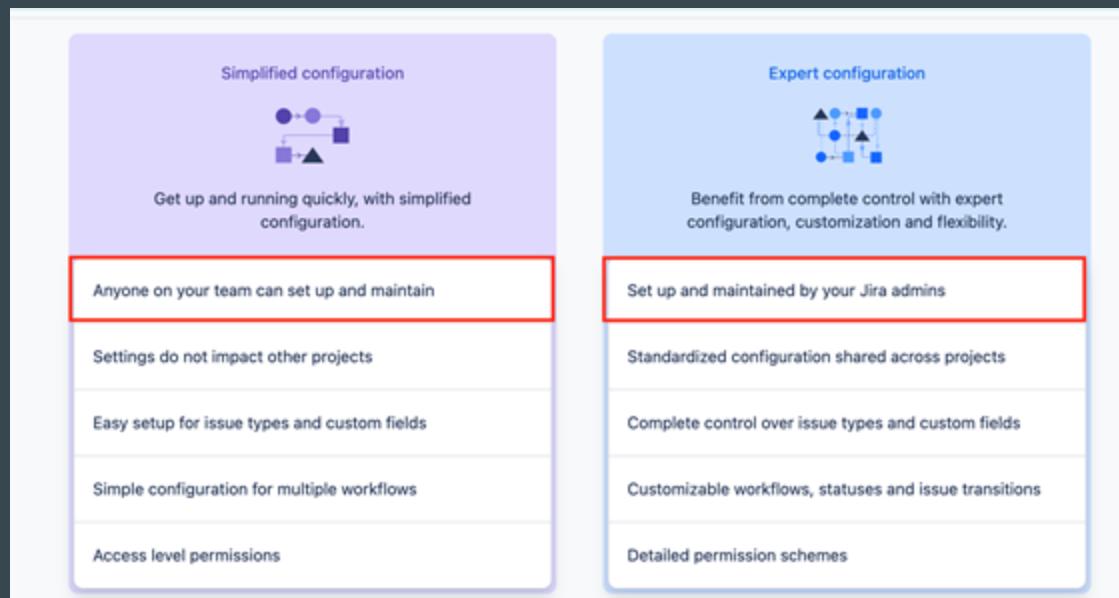
Installation is quite straightforward; go to Apps left side nav > Explore more apps > Xray and then “Try it free”. You need to use the account that created the Jira instance.

The image consists of three screenshots illustrating the Xray app installation process in the Jira Marketplace:

- Screenshot 1: Explore apps for Jira**  
Shows the Jira sidebar with "Explore more apps" selected. The main area displays "Explore apps for Jira" with a search bar for "xray". A red box highlights the "Xray - Test Management for Jira" card, which includes the Xray logo, developer name "Xblend", a brief description, a 3.6/4 rating with 543 reviews, and a "CLOUD FORTIFIED" badge. The card is set against a background showing "Showing over 10,000 apps".
- Screenshot 2: Select an edition and site**  
Shows the "Select an edition and site" step. It offers two options: "Standard" (selected) and "Advanced". Below this, it asks to "Select a site" and lists "https://testinguncoveredsuperman.com" as the chosen site. It also specifies "App will be installed with a trial version on the following product(s): Jira Software" and "E-445-FW/M-VXB-VB4".
- Screenshot 3: Review and Install**  
Shows the "Review and Install" step. It displays the "Xray - Test Management for Jira" card again, along with its details: "3.6/4 ★★★★☆ (543)", "USD 10 / month", and "Estimate after 30-day free trial". It includes sections for "Permissions" (Actions), "Data management" (Data storage, sharing, receiving, and processing), and "Analytics and logs" (Analytics and logs). At the bottom right are "Back" and "Start free trial" buttons.

# Enabling Xray on Jira projects

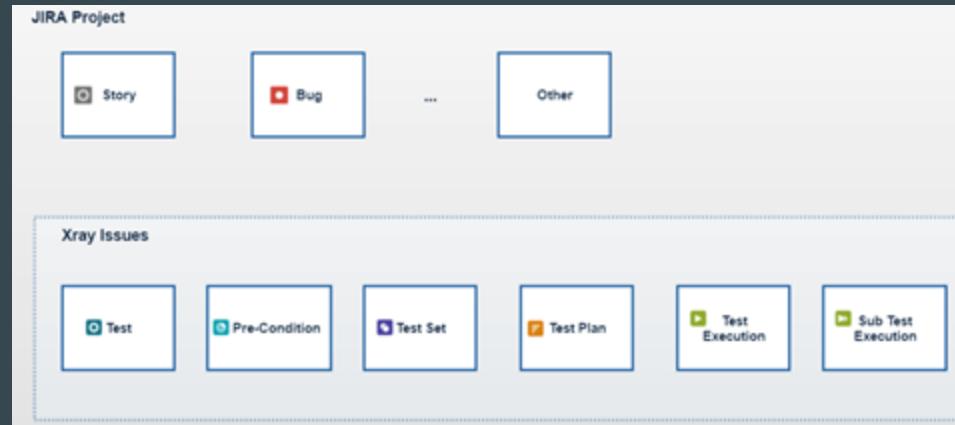
- Note: there are 2 main project types in Jira Cloud:
  - Team-managed
  - Company-managed
- We can enable Xray in any of these project types but the configuration steps are slightly different



# 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.

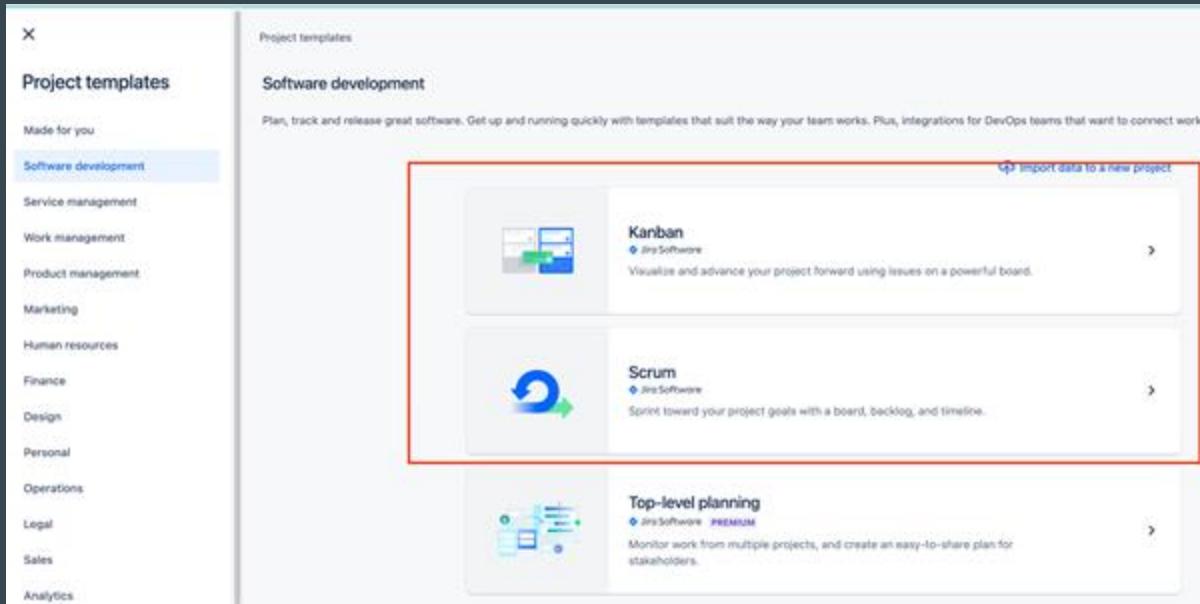
Then:

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

Scrum

Jira Software

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

Change template

Type

Team-managed

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

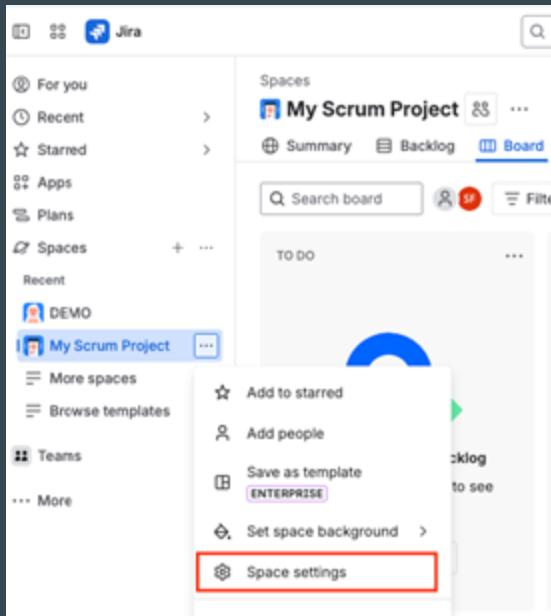
Change type

Cancel

Next

# Xray project setup

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



The screenshot shows the 'Xray Settings' page in Jira. At the top, there is a navigation bar with 'Space settings' and a back arrow. Below it is a search bar. The main content area is titled 'Summary' and contains a brief summary of Xray configurations for the project. It includes sections for 'Issue Types', 'Test Coverage', 'Defect Mapping', 'Test Types', 'Test Environments', and 'Test Step Fields'. Each section provides a brief description and a 'Configure' link. On the left side, there is a sidebar with various configuration tabs like 'Details', 'Access', 'Notifications', etc., and a 'Xray Settings' tab which is also highlighted with a red box.

# 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

Projects / SCRUMDEMO / Project settings / Apps

**Xray Settings**

**Issue Types Mapping**

The issue type configuration for this team-managed project.

Manual configuration required  
In order to configure Xray issue types for this team-managed project, you first need to create the issue types manually on the project configuration.

After the creation of the issue types for the Xray entities you need, you must associate them below:

**Test**  
Select an issue type

**Precondition**  
Select an issue type

**Test Set**  
Select an issue type

**Test Plan**  
Select an issue type

**Test Execution**  
Select an issue type

← Space settings

My Scrum Project  
Software space

Details

Access

Notifications

Automation

Fields

Work types

Epic

Bug

Feature

Story

Task

Subtask

+ Add work type

Features

# 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](#))

The image shows two screenshots from a Jira interface. On the left, a sidebar titled 'Space settings' for 'My Scrum Project' lists several sections: Details, Access, Notifications, Automation, Fields, Work types, Epic, Bug (which is selected), Feature, Story, Task, Subtask, and '+ Add work type'. The '+ Add work type' button is highlighted with a red box. On the right, a modal window titled 'Create work type' is open. It has fields for 'Name\*' (containing 'Test'), 'Description' (with placeholder text 'Let people know when to use this work type'), and an 'Icon' section with a 'Change icon' button. At the bottom are 'Create' and 'Cancel' buttons.

# 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.

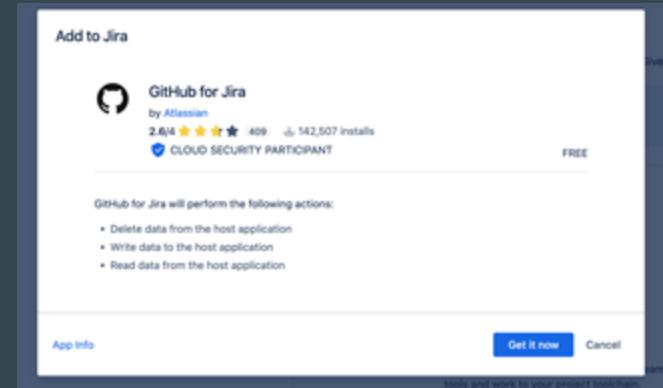
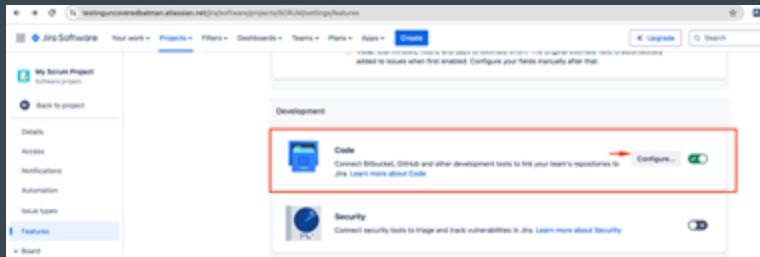
However, in our case, we'll keep it simple, using issue types with the same names of the expected entities:

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

The screenshot shows the Jira Software interface for a project named "SCRUMDEMO". The top navigation bar includes "Jira Software", "Your work", "Projects", "Filters", "Dashboards", "Teams", "Plans", "Apps", and a "Create" button. The "Projects" menu is open, showing "SCRUMDEMO / Project settings / Apps". The "Xray Settings" page is displayed, with the "Issue Types Mapping" tab selected. On the left, there's a sidebar with links for "Details", "Access", "Notifications", "Automation", "Issue types", "Features", "Board", "Toolchain", and "Apps". Under "Apps", there are "App Fields", "Slack integration", and "Xray Settings". The main content area is titled "Team-Managed Project Issue Types Mapping" and describes "The issue type configuration for this team-managed project". It lists several sections: "Test" (with "Test" listed), "Precondition" (with "Precondition" listed), "Test Set" (with "Test Set" listed), "Test Plan" (with "Test Plan" listed), "Test Execution" (with "Test Execution" listed), and a "Summary" table. The "Save" button is located at the bottom right of the main content area.

# Nice-2-Have

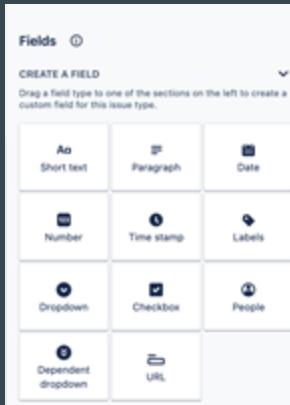
In Space settings > **Features**, enable Code and add the code repository tool (e.g., GitHub). => See detailed instructions provided earlier!



# 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



The screenshot shows the 'Issue types' configuration for 'Test Execution'. It includes sections for 'Description fields' (Summary, Description) and 'Context fields' (Status, Assignee, Labels, Parent, Story point estimate, Sprint, Fix versions, Revision). The 'Revision' field is highlighted with a red border.

# Xray: define “requirements” / issues coverable by tests

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

The screenshot shows the 'Xray Settings' page in Jira Software. The left sidebar has 'Xray Settings' selected. The main content area is titled 'Test Coverage' with the sub-instruction: '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.' On the right, there are two columns: 'Available Issue Types' and 'Coverable Issue Types'. The 'Available Issue Types' column lists: Task, Bug, Subtask, Test, Precondition, Test Set, Test Plan, and Test Execution. The 'Coverable Issue Types' column contains 'Story' and 'Epic', which are highlighted with a red rectangular box. Below these columns are sections for 'Issue Link relation' (selected: 'BelongsTo...') and 'Default Column Layout'. At the bottom is a 'Save' button.

# 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 Software interface for the project 'SCRUMDEMO'. The left sidebar has sections like Details, Access, Notifications, Automation, Issue types, Features, Board, Toolchain, Apps, App Fields, and Slack Integration. Under 'Xray Settings', the 'Defect Mapping' tab is selected. The main area is titled 'Defect Mapping' and contains a sub-section 'Available Issue Types' listing Story, Task, Epic, Subtask, Test, Precondition, Test Set, Test Plan, and Test Execution. To the right, under 'Defect Issue Types', the 'Bug' issue type is selected, indicated by a blue border around it. A note at the top right says '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.' A 'Save' button is at the bottom.

# Xray: final checkup

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

The screenshot shows the Jira Software interface with the following details:

- Header:** Jira Software, Your work, Projects, Filters, Dashboards, Teams, Plans, Apps, Create.
- Project Path:** Projects / SCRUMDEMO / Project settings / Apps.
- Left Sidebar (Xray Settings):**
  - Details
  - Access
  - Notifications
  - Automation
  - Issue Types
  - Features
  - Board
  - Toolchain
  - App Fields
  - Slack Integration
  - Xray Settings** (selected)
- Summary Tab:** This page contains a brief summary of the Xray configurations for this project.
- Issue Types:**
  - Xray Issue Types in Project:** There are 5 Xray issue types configured for this project. Go to the [Issue Types Mapping](#) to configure the issue types for this project.
  - Xray Entity:** Test, Precondition, Test Set, Test Plan, Test Execution.
  - Issue Type:** Test, Precondition, Test Set, Test Plan, Test Execution.
  - Description:** Test, Precondition, Test Set, Test Plan, 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

The screenshot shows the Jira interface for the 'My Scrum Project' space. On the left, the sidebar is open with 'Xray' selected under 'Your apps'. At the top right, there's a 'Testing Board' tab, which is also highlighted with a red box. The main content area displays the 'Test Repository' for this project, showing a message 'No tests in folder'.

This screenshot shows a 'Create Task' dialog box. In the 'Work type' dropdown, 'Task' is selected. Other options listed include 'bug', 'Test', 'Precondition', 'Test Set', 'Test Execution', and 'Test Plan'. A red box highlights the 'Task' option in the dropdown menu.

# 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.

The figure consists of three side-by-side screenshots of the Jira software interface, illustrating the steps to create and manage a test scenario:

- Screenshot 1: Epic Creation**

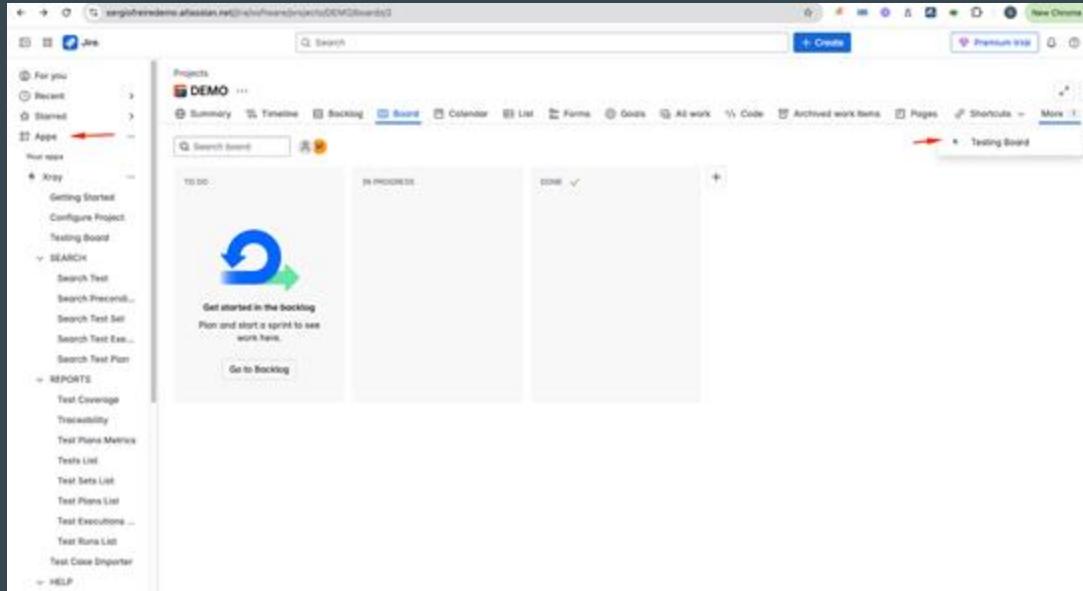
A screenshot of the Jira project navigation bar for "SCRUMDEMO". A new epic named "client area" is being created. The "Test Coverage" tab is selected, showing a single test named "As a user, I can login the web application".
- Screenshot 2: Story Creation**

A screenshot of the Jira project navigation bar for "SCRUMDEMO". A new story named "login" is being created under the "client area" epic. The "Test Coverage" tab is selected, showing the same test as in the first screenshot.
- Screenshot 3: Test Creation**

A screenshot of the Jira project navigation bar for "SCRUMDEMO". A new test named "manual test for valid login scenario" is being created. The "Test Coverage" tab is selected, showing the test associated with the "client area" epic.

# 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 screenshot shows the Jira web interface. On the left, there's a sidebar with links for Home, Jira, Goals, Teams, and Administration, with Administration highlighted by a red box. The main navigation bar at the top has tabs for Admin, sergiofreiredemo, Overview, Directory, Products, Security, **Billing** (which is also highlighted by a red box), and Settings. A modal window is open over the main content area. The modal has a header "See what's happening across your organization with our new overview". Below it, there's a "Quick actions" section with a "Invite users" button and a "For review" section containing a warning about an upcoming trial end. At the bottom of the modal, there's a "Subscription details" section with a yellow warning box about payment details needed. To the right of the modal, a larger "Subscription details" view is shown, featuring a "Enter promo code" input field with a red border. The main content area of the page shows a "Subscription details" table with one row for "Xray Test Management for Jira".

Subscription details	
<b>Xray Test Management for Jira</b> sergiofreiredemo.atlassian.net (J-40F-LWD-HPU-KZK)	<b>Standard</b> FREE 30-DAY TRIAL
Users 1	Billing cycle Monthly
Next bill date May 25, 2025	Next bill estimate USD 10.00
Billing profile Installed on (0)	2 subscriptions   Manage

# Xray Entities & Concepts

...

# Xray entities & concepts (a few)

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

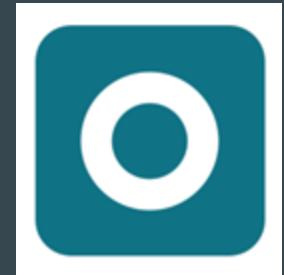
- Test
- Test Run
- Test Execution
- Test Plan
- “Requirements”
- “Defects”

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

The screenshot shows a manual test case in Xray. The test details tab is selected, showing the following steps:

Action	Data	Expected Result
press 1	None	1
press key	4	4
press 2	None	2

The right sidebar displays the following details:

- To Do: Unassigned (Assign to me)
- Reporter: Sérgio Freire
- Development: Create branch, Create commit
- Releases: None
- Labels: None
- Revisions: None
- Externalized: None
- Priority: Medium
- Xporter: Open Xporter
- Test status: Open Test Status
- More fields: Original estimate, Time tracking, Epics Link, Components, Sprint...
- Automation: Rule executions

Log information at the bottom:

- Created March 8, 2023 at 5:53 PM
- Updated March 8, 2023 at 5:53 PM
- Configure

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 **Xray**.

Sérgio Freire

# 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

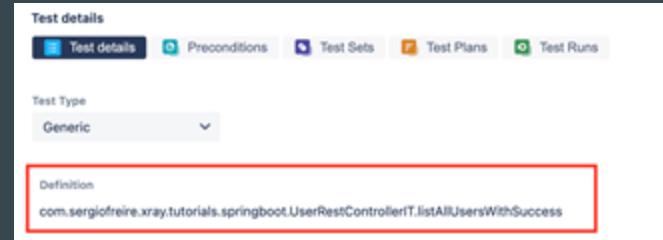
```
1 Given I have entered "<input_1>" into the calculator
2 And I have entered "<input_2>" into the calculator
3 When I press "<button>"
4 Then the result should be "<output>" on the screen
5
6 Examples:
7 | input_1 | input_2 | button | output |
8 | 20     | 30      | add    | 50   |
9 | 2      | 5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



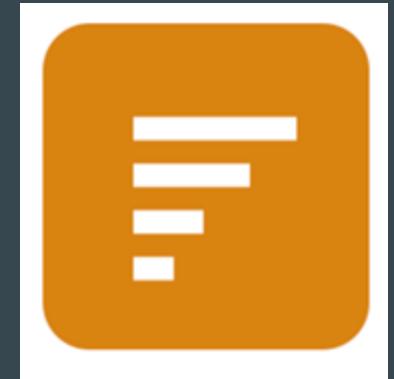
# Test Run

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

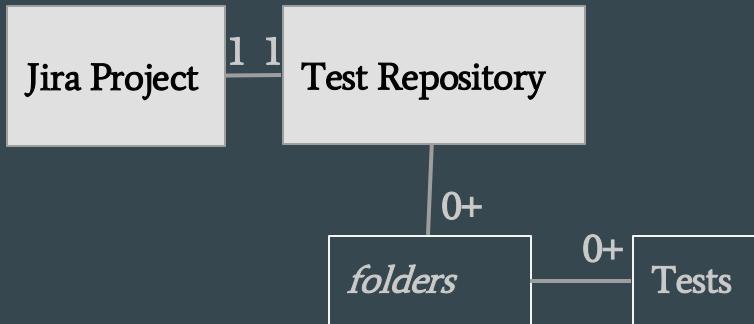
# Relation between entities

- “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



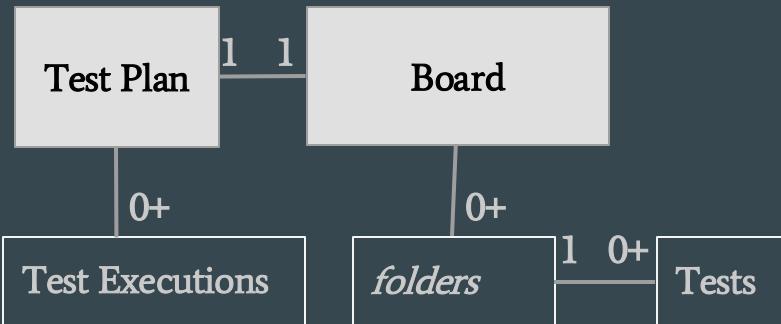
The screenshot shows the 'Test Repository' screen for the 'Calculator' project. The left sidebar lists categories: 'Test Repository' (1909 / 1933), 'Performance testing' (0 / 0), 'Postman F1' (14 / 14), 'Postman F2' (1 / 1), 'core' (1 / 1), 'UI' (0 / 8), 'login' (1 / 1), and 'basic arithmetic operations' (7 / 7). The main area displays 12 entries, filtered by 'Test Type: Cucumber'. Each entry includes a status indicator (e.g., green circle for 'CALC-2706 dummy gherkin precondition', red square for 'CALC-2666 xx').

Test ID	Description	Status
CALC-2706	dummy gherkin precondition	Green
CALC-2666	xx	Red
CALC-1450	organizations exist	Green
CALC-864	Background for: CALC-861	Green
CALC-393	Calculator exists	Green

# 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



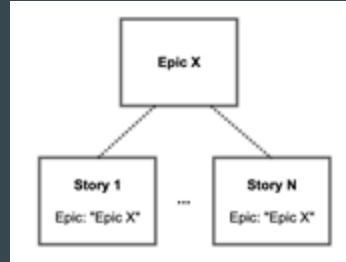
# Defects (e.g., “Bug”)

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

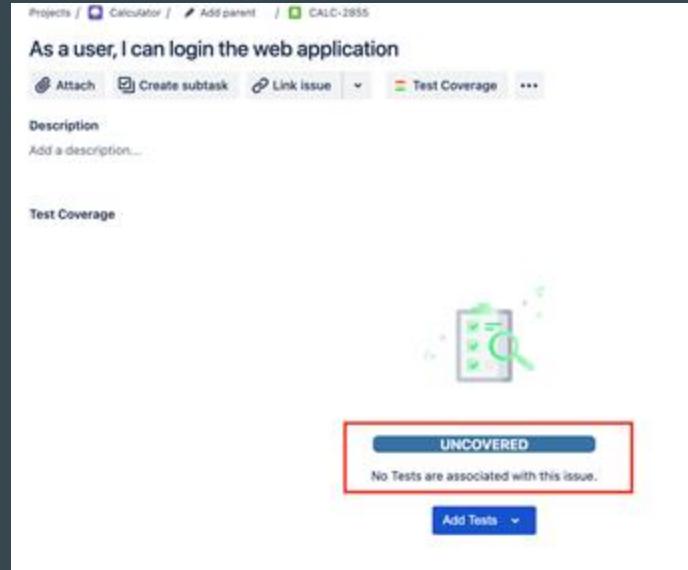
Xray provides an heuristic, called (test) coverage; sometimes people may refer to it as requirement coverage, depending on the perspective.

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 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 shows a requirement in the Xray interface. 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, Test Coverage, and more. A "Description" field is present with placeholder text "Add a description...". Under "Linked issues", it says "is tested by" and lists "CALC-2856 test for successful login" with a status of "TO DO". In the "Test Coverage" section, there is an "Add Tests" button and an "Execute" dropdown. The "Analysis & Scope" section shows "Scope: Latest Final Status". A yellow bar at the bottom right indicates the requirement is in "NOTRUN" status. The table below shows a single test entry:

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

Pagination controls "Prev" and "Next" are at the bottom left of the table.

# 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 requirement in the Xray application. 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", "Test Coverage", and more. A "Description" field is present with the placeholder "Add a description...". Under "Linked Issues", there is a link to "CALC-2856 test for successful login". In the "Test Coverage" section, there is a "TO DO" button. The "Analysis & Scope" section shows "Scope: Latest Final Status". At the bottom, there is a table with one row, showing "Status: TO DO", "Key: CALC-2856", and "Summary: test for successful login". The "Test Status" column shows a green bar with the word "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 a requirement titled "As a user, I can login the web application". Under "Linked issues", there are two entries: "CALC-2856 test for successful login" and "CALC-2858 test for invalid login", both marked as "TO DO". Below this is a "Test Coverage" section. A red box highlights the "NOK" status bar at the bottom right of the coverage table. A red arrow points from the "FAILED" status in the legend to the "NOK" bar. The coverage table lists two rows: one for "test for successful login" (Status: TO DO, Key: CALC-2856) and another for "test for invalid login" (Status: TO DO, Key: CALC-2858). The legend indicates that green means "PASSED" and red means "FAILED".

# 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:

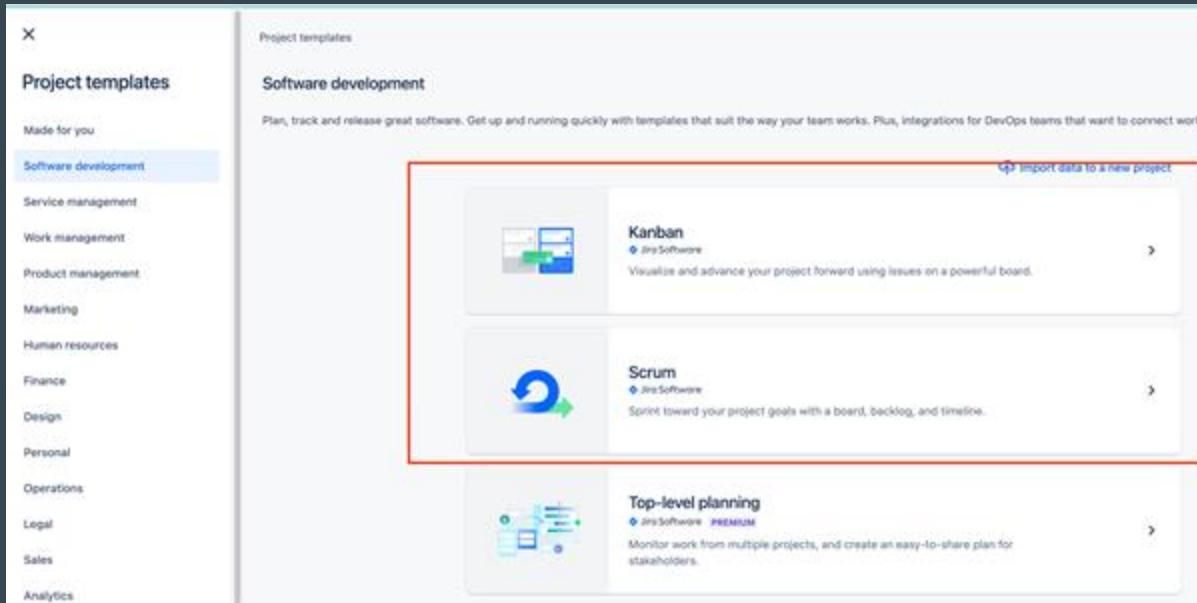
1. Add issue types to the project and map them to the Xray concepts
2. Define which items you consider to be “requirements” like
3. Define which items you consider to be “defects” like

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><b>Simplified configuration</b></p>  <p>Get up and running quickly, with simplified configuration.</p> <ul style="list-style-type: none"><li>Anyone on your team can set up and maintain</li><li>Settings do not impact other projects</li><li>Easy setup for issue types and custom fields</li><li>Simple configuration for multiple workflows</li><li>Access level permissions</li></ul> <p><b>Essential features</b></p>  <p>A modern Jira experience for teams who don't need advanced features.</p> <ul style="list-style-type: none"><li>Only show your project's issues on your board</li><li>Essential agile reporting</li></ul>	<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><b>Expert configuration</b></p>  <p>Benefit from complete control with expert configuration, customization and flexibility.</p> <ul style="list-style-type: none"><li>Set up and maintained by your Jira admins</li><li>Standardized configuration shared across projects</li><li>Complete control over issue types and custom fields</li><li>Customizable workflows, statuses and issue transitions</li><li>Detailed permission schemes</li></ul> <p><b>Advanced features</b></p>  <p>All the power and features that Jira Software is known for.</p> <ul style="list-style-type: none"><li>Pull in issues from other projects on your board</li><li>Comprehensive agile reporting</li></ul>
<a href="#">Select a team-managed project</a>	<a href="#">Select a company-managed project</a>

# 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 with the following details:

- Project:** SAMPLEPROJ (Software project)
- Page:** Xray Settings - Summary
- Left sidebar:** Includes links for Back to project, Summary, People, Permissions, Notifications, Automation, Features, Testcases, Workflows, Issues (selected), Types, Labels, Screens, Fields, Collectors, Security, Components, and Apps (Xray Integration selected).
- Content Area:**
  - Issue Types:** A section titled "Xray Issue Types in Project" shows 0 issue types configured. It includes a red box around the "Add Xray Issue Type" button.
  - Table:** A table listing Xray Issue Types with columns: Name, Description, and Present in 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 current 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:** A section stating "This project is currently using the global miscellaneous settings. Configure".
  - Test Coverage:** A section stating "This project has no issue types mapped as covered. Configure".
  - Defect Mapping:** A section stating "This project has no issue types mapped as defects. Configure".
  - Test Types:** A section stating "The 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. On the left, a sidebar lists various settings like 'Miscellaneous', 'Remote Jobs Trigger', and 'Test Types'. The main area is titled 'Test Coverage' with a sub-section 'Available Issue Types' and 'Coverable Issue Types'. An arrow points from the 'Story' checkbox in the 'Coverable Issue Types' list to the 'Story' checkbox in the 'Available Issue Types' list. At the bottom, there are configuration options for 'Test Coverage Hierarchy' (including 'Epic - Issues(Stories) relation' and 'Issue - Sub-tasks relation'), 'Issue Links relation' (with a dropdown menu), and 'Issue Link Type Direction' (radio buttons for 'Outward' and 'Inward'). A red box highlights the 'Save' button at the bottom.

# 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' interface with the 'Defect Mapping' option selected in the sidebar. The main area is titled 'Defect Mapping' and contains a descriptive text about defects. Below it are two panels: 'Available Issue Types' on the left and 'Defect Issue Types' on the right. An arrow points from the 'Task' checkbox in the 'Available Issue Types' list to the 'Bug' checkbox in the 'Defect Issue Types' list, indicating the mapping process.

**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

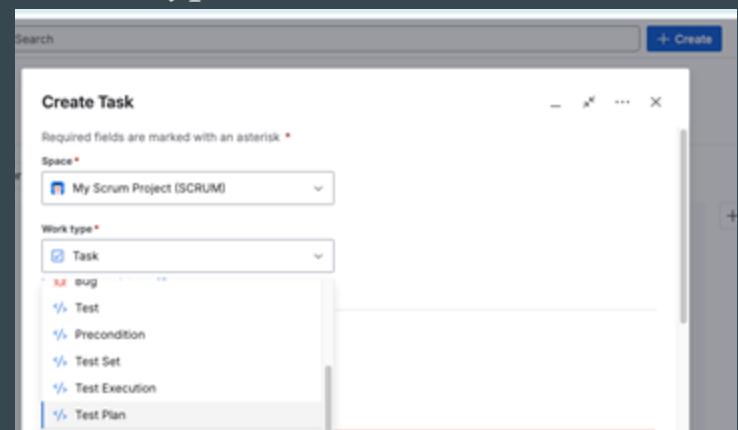
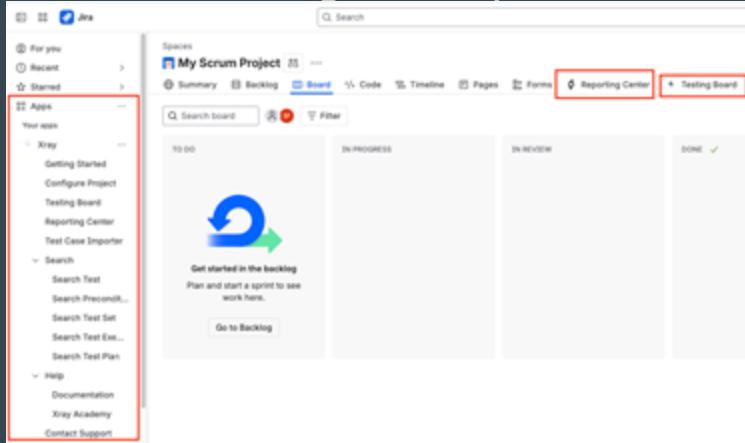
Go to Project settings > Apps > Xray Settings > Summary.

The screenshot shows the 'Xray Settings' page under 'Project settings'. The left sidebar lists various project management options like 'Summary', 'People', 'Permissions', etc. The 'Xray Settings' section is expanded, showing sub-options such as 'Miscellaneous', 'Remote Job Trigger', 'Test Coverage', 'Default Mapping', 'Test Types', 'Test Environments', 'Document Generator', 'Test Step Fields', 'Test Run Custom Fields', 'Parameter value lists', 'BDD Step Library', 'Default Column Layouts', and 'Re-indexing'. The 'Miscellaneous' section indicates the project is currently using global settings. The 'Test Coverage' section shows mapped issue types: Story (green) and Epic (purple). The 'Defect Mapping' section shows mapped issue types: Bug (red). The 'Test Types' section lists 'Manual', 'Generic', and 'Cucumber'. The 'Test Environments' section shows 0 environments configured. A modal window titled 'Xray Issue Types In Project' is open, stating there are 6 Xray issue types configured for the project. It lists the following issue types:

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.	✓

# 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 consists of three side-by-side screenshots of a software application interface, likely Jira, demonstrating the creation of a project structure and associated tests.

**Screenshot 1: Project Overview**  
Shows the 'client area' for 'SAM-1'. It includes sections for 'Description', 'Child issues' (listing 'SAM-2'), and 'Test Coverage'. A modal window is open over the main content, showing a story titled 'As a user, I can login the web application' with a 'NOT RUN' status in the 'Test Coverage' section.

**Screenshot 2: Story Details**  
Shows the details of the story 'As a user, I can login the web application'. It includes a 'Description' field, a 'Linked issues' section (listing 'SAM-2'), and a 'Test Coverage' section. The 'Test Coverage' section shows a single test named 'test for valid login scenario' with a 'NOT RUN' status.

**Screenshot 3: Test Creation**  
Shows the creation of a new test titled 'test for valid login scenario'. The 'Test details' section indicates the test type is 'Manual'. The 'Test Steps' section is currently empty, with a note stating 'There are no steps defined'.

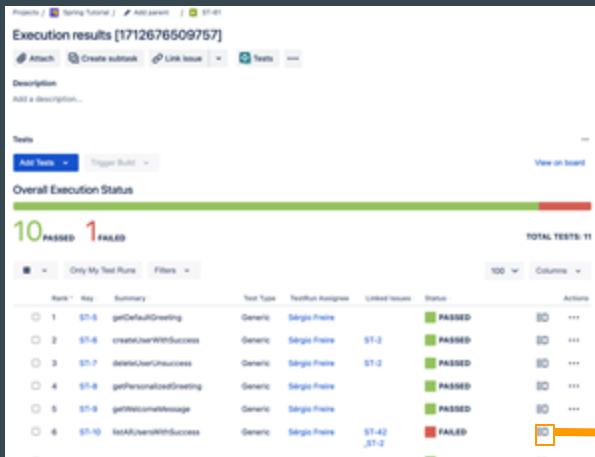
# Integrating Xray with Test Automation

# 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)

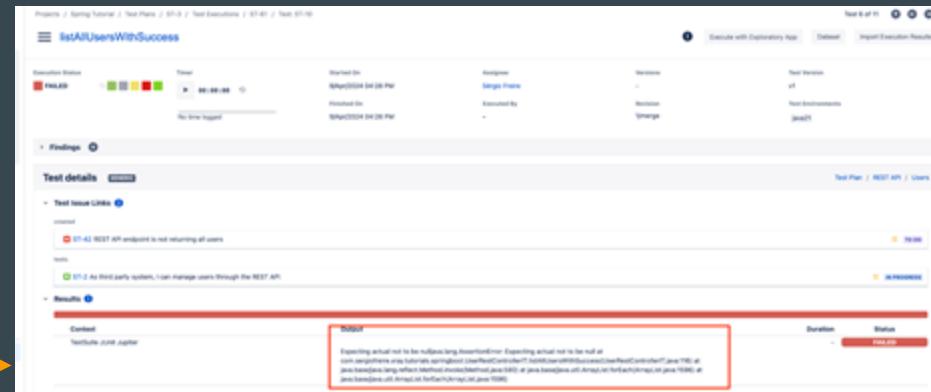


The screenshot shows the Xray Test Execution interface. At the top, it displays 'Execution results [1712678509757]'. Below this is a summary bar with '10 PASSED' and '1 FAILED'. The main area is titled 'Overall Execution Status' and shows a table of test cases. The table includes columns for Rank, Key, Summary, Test Type, Author, LastRun Assigned, LastRun Status, and Actions. The first six rows show successful test cases (Status: PASSED), while the last row, 'ST-10 listAllUsersWithSuccess', shows a failure (Status: FAILED). An orange arrow points from this failed test case to the 'Details of a Test Run' screenshot.

Rank	Key	Summary	Test Type	Author	LastRun Assigned	LastRun Status	Actions
1	ST-5	getDefaultValue	Generic	Sergio Freire		PASSED	...
2	ST-6	createUserWithSuccess	Generic	Sergio Freire	ST-2	PASSED	...
3	ST-7	deleteUserWithSuccess	Generic	Sergio Freire	ST-2	PASSED	...
4	ST-8	getPersonalizedMeeting	Generic	Sergio Freire		PASSED	...
5	ST-9	getWelcomeMessage	Generic	Sergio Freire		PASSED	...
6	ST-10	listAllUsersWithSuccess	Generic	Sergio Freire	ST-2	FAILED	...

## Details of a Test Run

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



The screenshot shows the 'Details of a Test Run' for the failed test case 'ST-10 listAllUsersWithSuccess'. It includes sections for 'Execution Status' (FAILED), 'LastRun' (started at 2024-04-26 09:49:49, ended at 2024-04-26 09:49:50), and 'Findings'. The 'Test details' section shows the test issue link 'ST-42 REST API endpoint is not returning all users' and the test 'As third party system, I can manage users through the REST API'. The 'Output' section contains a red box highlighting an error message: 'Expected actual not to be null or empty. Actual was null or empty. at java.base/java.util.ListIterator.hasNext(ListIterator.java:196) at java.base/java.util.ArrayList.iterator(ArrayList.java:136) at java.base/java.util.ArrayList.listIterator(ArrayList.java:136) at java.base/java.util.ArrayList.iterator(ArrayList.java:136)'.

# 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

The screenshot shows a test run titled "listAllUsersWithSuccess" that has failed. It includes sections for "Findings", "Test details", "Test Issue Links", and "Results". A red box highlights a specific finding: "ST-12 As third party system, I can manage users through the REST API". Another red box highlights the overall test status as "FAILED".

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)

The screenshot shows a Jira issue screen for "ST-10". The "Test Coverage" section indicates a status of "NOK" with a red box. It lists several test cases under "Linked Issues" and their coverage status. The "Test Coverage" table at the bottom shows the status of three tests: "createUserWithSuccess" (PASSED), "deleteUserUnsuccess" (PASSED), and "listAllUsersWithSuccess" (FAILED).

Status	Key	Summary
PASSED	ST-6	createUserWithSuccess
PASSED	ST-7	deleteUserUnsuccess
FAILED	ST-10	listAllUsersWithSuccess

# 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

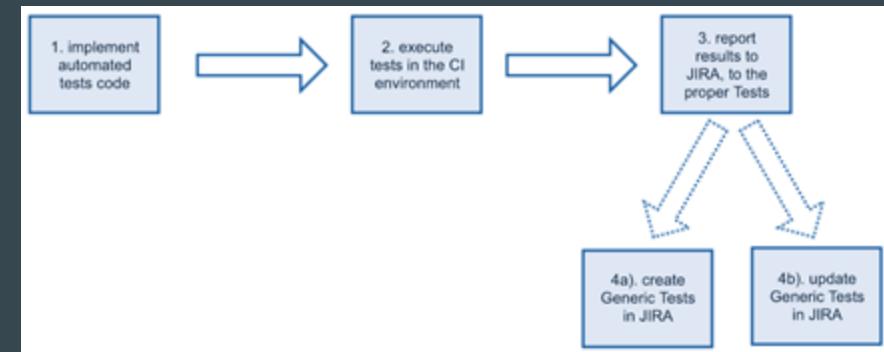
# Common flow for having visibility of test automation results

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.

The screenshot shows the Jira Software interface. On the left, there's a sidebar with 'YOUR APPS' and 'RECOMMENDED FOR YOUR TEAM' sections. A red box highlights the 'Manage your apps' link under 'YOUR APPS'. On the right, the main content area has a 'Jira Software' header with various navigation links like 'Your work', 'Projects', 'Filters', 'Dashboards', 'Teams', 'Plans', and 'Apps'. A 'Create' button is also in the header. Below the header, the 'API Keys' section is selected in the sidebar. The main table displays one API key entry:

User	Client Id	Client Secret	Actions
Sergio Freire	215FFD9FFE464472BC721B2E050CA49	*****	... Actions

At the bottom of the sidebar, another red box highlights the 'API Keys' link.

# 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=XPTO&testPlanKey=XPTO-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 of the Jira interface. The left screenshot displays the 'Execution results' page for issue #708453141445, showing a table of test results. An orange arrow points from the bottom of this page to the right screenshot. The right screenshot shows the 'Details' view of a newly created 'Test Execution' issue (#X-674) in the 'Tutorial-spring' project. The 'Test Plans' field is highlighted with a red border, and an orange arrow points from the 'Test Plan' field in the details to the 'Associated Test Plan' section in the 'Test Environment' table of the 'Test Execution' issue.

Execution results #708453141445

Overall Execution Status

TOTAL TESTS: 11

Rank	Test	Type	Dataset	Objects	TestPlan assigned	Priority	Linked Issues	Status
1	X7- getPersonalizedGreeting	Generic	0	Sergio.Freire	None	PASSSED		
2	X7- dontCreateUserWithSameData	Generic	0	Sergio.Freire	None	PASSSED		
3	X7- getUserUserSuccess	Generic	0	Sergio.Freire	None	PASSSED		
4	X7- deleteUserWithSuccess	Generic	0	Sergio.Freire	None	PASSSED		
5	X7- createUserWithSuccess	Generic	0	Sergio.Freire	None	PASSSED		

Details

Labels: None

Priority: Medium

Assignee: Open Sparker

Test Plans: Open Test Plan

Test Environments: Open Test Environment

More Fields

Original estimate: 0h

Time tracking: No time logged

Epic Link: Replaced by Parent

Components: None

Sprint: None

Fix versions: None

Parent: NEW

Automation: 0 rule executions

Created: February 20, 2014 at 8:19 PM

Last updated: February 20, 2014 at 8:19 PM

Test Environment

Issue	Summary	Assigned	Efforts	Work Hours	Test Environment	Status
X7-674	Test Execution for Test Plan X7-674	Sergio.Freire	0	0	Sergio.Freire	In Progress
X7-671	Execution results #708453141445	Sergio.Freire	0	0	Sergio.Freire	In Progress

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?

The diagram illustrates the auto-provisioning process. On the left, a screenshot of an IDE shows a Java test class (`UserRestControllerIT.java`) with a method annotated with `@Test`. This method contains logic to create users and verify a response. A red box highlights the `@Test` annotation and the method name `listAllUsersWithSuccess`. An orange arrow points from this highlighted area to the right side of the image, which shows the corresponding test entry in the Xray interface. The Xray interface displays the test details, including the test type set to "Generic" and the definition pointing to the Java method `listAllUsersWithSuccess`.

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

Edit Preview

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

Projects / Spring Tutorial / AIM parent / 87-10

listAllUsersWithSuccess

Attach Create subtask Link issue Test details

Description Add a description...

Linked issues

87-12 REST API endpoint is not returning all users

87-12 As third party system, I can manage users through the REST API

Test details

Test Type Generic

Definition com.sergiofreire.xray.tutorials.springboot.UserRestControllerIT.listAllUsersWithSuccess

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.

Status	Key	Summary	Test Status
PASSED	XT-676	Test as a user, I can see a welcome message on the ...	PASSED

Start typing to get a list of possible matches or press down to select from a list of existing issues

XT-683

Status	Key	Summary	Test Status
PASSED	XT-676	Test as a user, I can see a welcome message on the ...	PASSED
PASSED	XT-683	getDefaultValue	PASSED

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", "dummypassword");
    createTempUser("Robert Junior", "robert", "dummypassword");

    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 details:

- Execution results [1714405750095]:** A summary table with 2 PASSED tests.
- Test Type:** Generic (highlighted by a red box).
- Description:** Purchase Right Successful purchase flight (highlighted by a red box).
- Environment:** None.
- Test Details:** A modal window titled "Successful purchase flight" showing the same information as 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 Test Run result report with the following details:

- Overall Status:** Passed (green bar).
- Elapsed Time:** 0s (0ms).
- Test Details:** A table showing the test run details, including:
  - Test ID: CALC-395
  - Test Name: Successful purchase flight
  - Dataset: None
  - Assignee: Sérgio Freire
  - Priority: Low
  - Status: Passed
  - Output: [View Log](#)

## 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*

This screenshot shows the Jira Software interface. On the left, there's a sidebar with various project management sections like Planning, Development, and Reporting. In the main area, a project named 'Calculator' is selected. Under 'Testing Board', a report titled 'usability test' is displayed. This report includes sections for 'Findings', 'Defects', 'Evidence', and 'Comments'. A red box highlights the 'Testing Board' button in the sidebar.

This screenshot shows the Jira Software interface with a different project selected. The sidebar shows the 'Testing Board' section. A red box highlights the 'Reports' button in the sidebar. Below it, a list of report types is shown, with 'Test Coverage' highlighted by a red box.

This screenshot shows the 'Test Coverage Report' interface from the Xray app. The top navigation bar includes 'Jira Software', 'Your work', 'Projects', 'Filters', 'Dashboards', 'Plans', 'Apps', and 'Create'. The main content area is titled 'Test Coverage Report'. It features a sidebar with options like 'Getting Started', 'Configure Project', 'Testing Board', 'SEARCH', and 'REPORTS'. Under 'REPORTS', 'Test Coverage' is selected and highlighted with a blue box. The central area displays a summary with a gear icon and a message: 'Please choose the calculation scope and set the filters above in order to generate the report.' A 'Generate Report' button is at the bottom right.

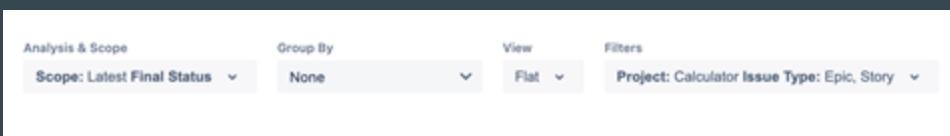
# Typical actions/inputs on built-in reports

## 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 shows a traceability report interface with the following sections:

- Analysis & Scope:** Includes filters for Scope: Version: Version v1.0 Final Status: and Project: Calculator/Issue Type: Epic, Story Label: MOT.
- Quick Filters:** Buttons for OK (1), NOK (1), NOTRUN (0), UNKNOWN (0), and UNCOVERED (3).
- Requirements:** A list of requirements grouped by ID:
  - CALC-1120: 7898:** If I'm anonymous, v1.0.
    - As a user, I can perform restricted operations in my ... **NOK**
    - As a user, I can manage my profile details **UNCOVERED**
  - CALC-795: IN PROGRESS:** If I'm anonymous, v1.0.
    - As a user, I can login the web application **NOK**
- TESTS:** A list of tests grouped by ID:
  - CALC-267: 7898:** Visitor Login using Gherkin style **PASSED**
  - CALC-1449: 80 View Details:** If I'm anonymous, v1.0.
    - Estimated On: 20Apr22 03:16 PM  
Executed By: Sérgio Freire  
Test Environments:   
Test Version: v1
  - CALC-1428: 80 View Details:** If I'm anonymous, v1.0.
    - Estimated On: 23Apr22 03:40 PM  
Executed By: Sérgio Freire  
Test Environments:   
Test Version: v1
  - CALC-1427: 80 View Details:** If I'm anonymous, v1.0.
    - Estimated On: 23Apr22 04:02 PM  
Executed By: Sérgio Freire  
Test Environments:   
Test Version: v1
  - CALC-2645: 80 View Details:** If I'm anonymous, v1.0.
    - Estimated On: 24Apr22 03:19 PM  
Executed By: Sérgio Freire  
Test Environments:   
Test Version: v1
  - CALC-2661: 7898:** login button too small **FAILED**
- DEFECTS:** A list of defects grouped by ID:
  - CALC-2645: 80 View Details:** If I'm anonymous, v1.0.
    - Estimated On: 24Apr22 03:19 PM  
Executed By: Sérgio Freire  
Test Environments:   
Test Version: v1
  - CALC-2661: 7898:** login button too small **FAILED**

Bottom right corner of the screenshot has a pink arrow pointing towards the bottom right.

# 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



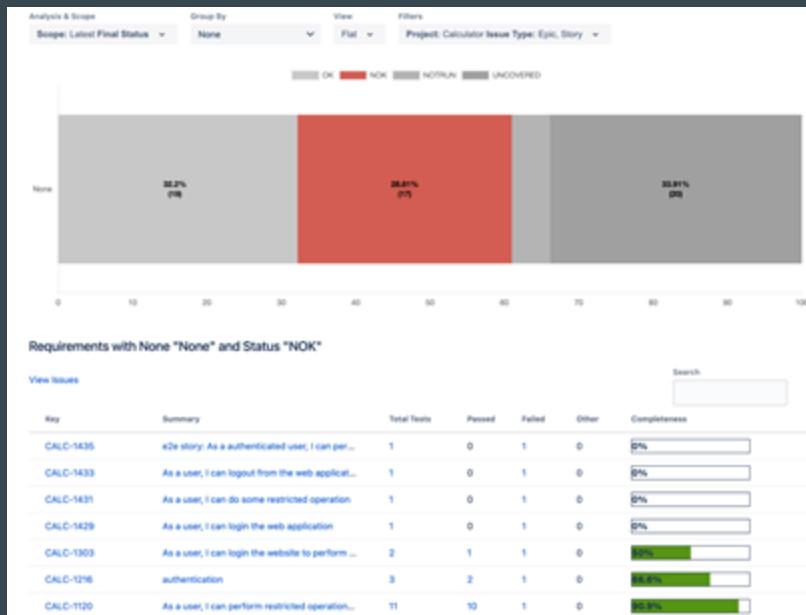
Status meaning:

- 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

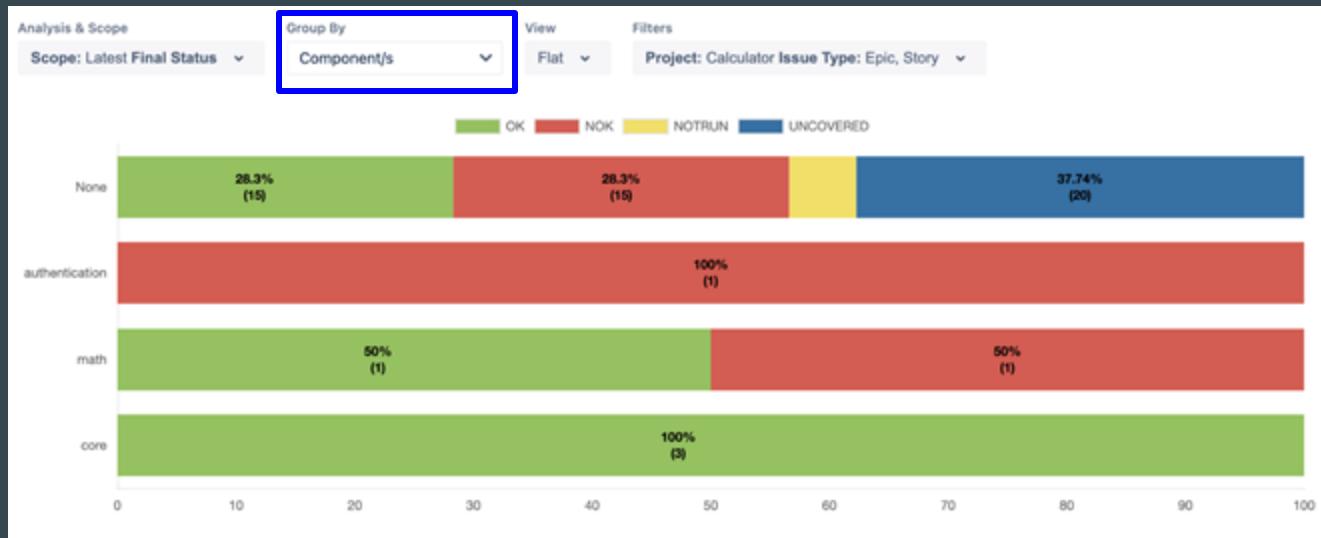
Drill-down, by clicking on a bar, to see:

- 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

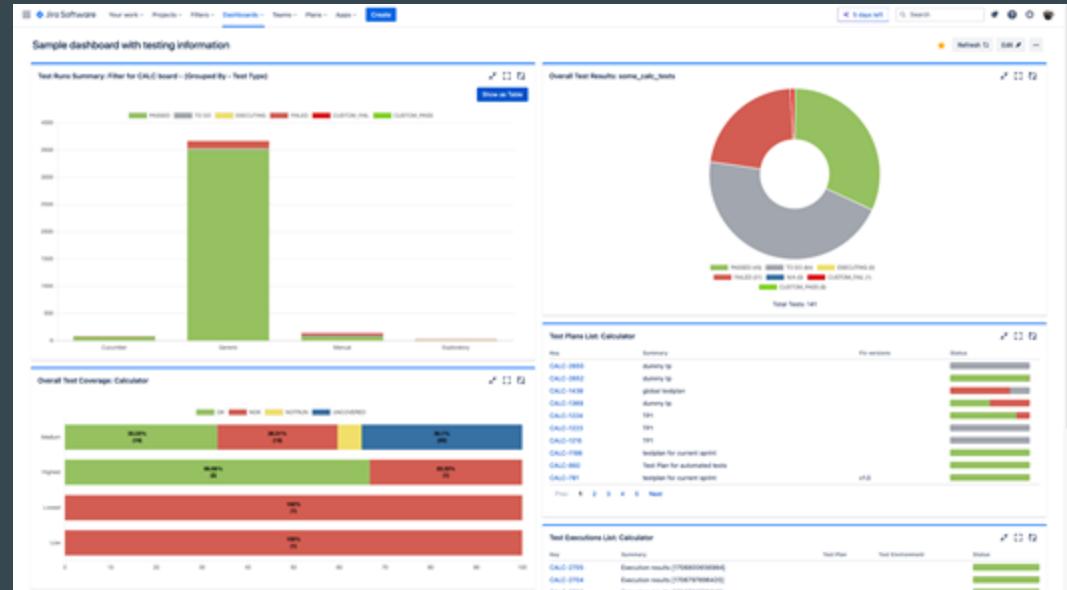
In this case we’re focused on the impacts of our testing. What does it tell about the quality of our deliverables?

# 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...

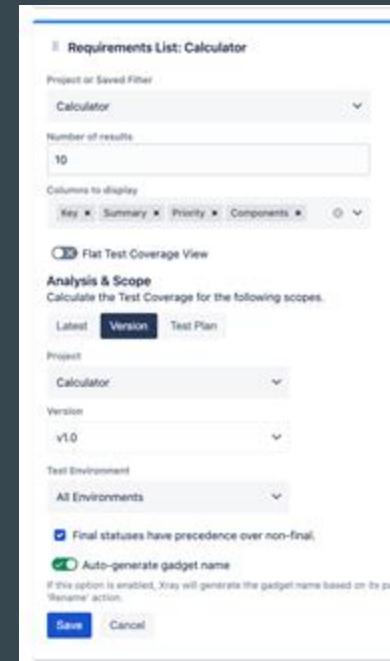
- Test Runs related info (e.g., status, executed by, dates)
- Tests related info (e.g., priority, component)
- ...

# Xray gadgets: Requirements List

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

- Define the scope for calculating coverage status

Requirements List: Calculator				
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



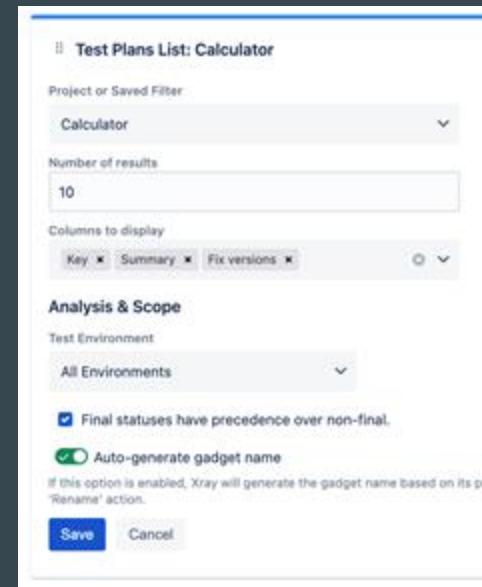
# Xray gadgets: Test Plans List

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

Test Plans List: Calculator			
Key	Summary	Fix versions	Status
CALC-2655	dummy tp		<div style="width: 0%; background-color: #ccc;"></div>
CALC-2652	dummy tp		<div style="width: 100%; background-color: #99ff99;"></div>
CALC-1438	global testplan		<div style="width: 50%; background-color: #cc0000;"></div>
CALC-1369	dummy tp		<div style="width: 30%; background-color: #99ff99;"></div>
CALC-1224	TP1		<div style="width: 80%; background-color: #99ff99;"></div>
CALC-1223	TP1		<div style="width: 0%; background-color: #ccc;"></div>
CALC-1215	TP1		<div style="width: 0%; background-color: #ccc;"></div>
CALC-1198	testplan for current sprint		<div style="width: 100%; background-color: #99ff99;"></div>
CALC-992	Test Plan for automated tests		<div style="width: 100%; background-color: #99ff99;"></div>
CALC-781	testplan for current sprint	v1.0	<div style="width: 100%; background-color: #99ff99;"></div>

Prev 1 2 3 4 5 Next



# 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

Test Executions List: Calculator		
Key	Summary	Test Plan
		Test Environment
CALC-61	Test Execution of XRAYintegration of the test WANI...	CALC-12
CALC-60	Test Execution of XRAYintegration of the test WANI...	CALC-12
CALC-58	Exploratory session for CALC-57	CHROME
CALC-56	Ad-hoc execution for CALC-55	FIREFOX
CALC-53	Exploratory session for CALC-52	CHROME
CALC-49	Exploratory session for CALC-46	FIREFOX
CALC-47	Ad-hoc execution for CALC-46	CHROME
CALC-45	Ad-hoc execution for CALC-44	CHROME
CALC-42	Exploratory session for CALC-41	CHROME
CALC-39	Test Execution of XRAYintegration of the test WANI...	CALC-12
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

Test Runs List: Calculator

Key	Test Execution Key	Summary	Test Execution Version	Test Environment	Status
CALC-3	CALC-B	Cucumber Test As a user, I can sum two numbers	v3.0		PASSED
CALC-4	CALC-B	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-82	CALC-53	assess usability risks in the login page	v3.0	FIREFOX	FAILED

Prev 1 2 3 4 5 ... 391 Next

Test Runs List: Calculator

Project or Saved Filter: Calculator

Test Filter

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

Contains: 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

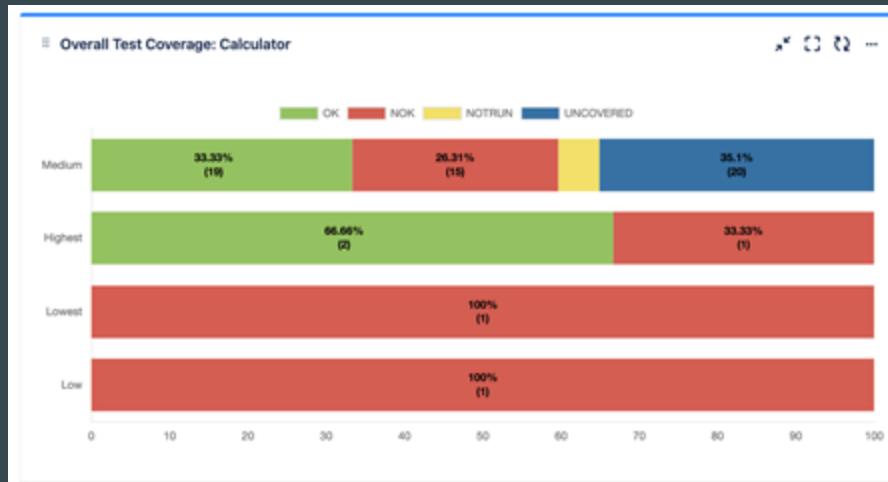
Auto-generate gadget name: If this option is enabled, Xray will generate the gadget name based on its parameters. Otherwise, you can explicitly name it.

Save Cancel

# Xray gadgets: Overall Test Coverage

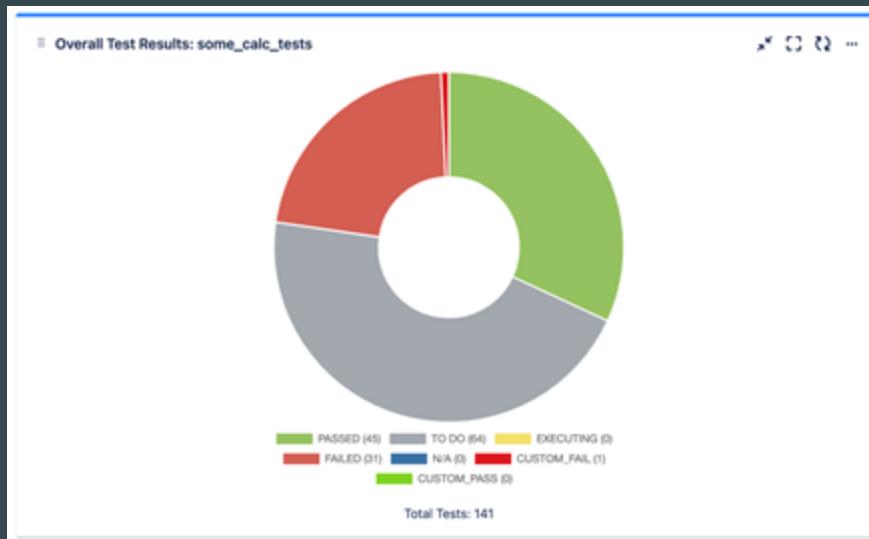
Provides a birds-eye overview of the (coverage) status of your deliverables (e.g., stories, epics) based on their latest testing results.

- 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.).



The configuration dialog for "Overall Test Results: some\_calc\_tests". It includes fields for "Project or Saved Filter" (set to "some\_calc\_tests"), "Analysis & Scope" (with a note to calculate coverage for environments), and "Test Environment" (set to "All Environments"). There are also checkboxes for "Final statuses have precedence over non-final" and "Auto-generate gadget name".

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

JIRA Your work Projects Filters Dashboards Teams Plans Apps Create

No users left Search Refresh Edit

### TQS: Spring Tutorial Dashboard

Overall Test Coverage: Spring Tutorial

Component	Status	Value
backoffice	UNCOVERED	100%
None	NONE	100%
REST	NONE	100%
frontend	OK	100%

Requirements List: Spring Tutorial

Key	Summary	Priority	Components	Status
ST-33	As a admin, I can manage users in a web based back...	Low	backOffice	UNCOVERED
ST-30	User management	Low		NONE
ST-2	As third party system, I can manage users through t...	Low	REST	NONE
ST-1	As a user, I can access a landing page	Low	frontend	OK

Filter Results: ST\_defects

Key	Summary
ST-42	REST API endpoint is not returning all users

Overall Test Results: Spring Tutorial

Category	Count
PASSED	115
TO DO	0
EXECUTING	0
FAILED	0
N/A	0
CUSTOM_FAIL	0
CUSTOM_PASS	0

Total Tests: 115

### Test Executions List: Spring Tutorial

Key	Summary	Test Plan	Test Environment	Revision	Status
ST-63	Execution results [17120676936799]	ST-3	2480007	'Merge	
ST-62	Execution results [17120676936275]	ST-3	2480021	'Merge	
ST-61	Execution results [17120676936979]	ST-3	2480021	'Merge	
ST-60	Execution results [17120676936376]	ST-3	2480007	'Merge	
ST-59	Execution results [17120676947346]	ST-3	2480021	'Merge	
ST-58	Execution results [17120676947198]	ST-3	2480007	'Merge	
ST-57	Execution results [1712067690126408]	ST-3	2480021	'Merge	
ST-56	Execution results [1712067690139672]	ST-3	2480007	'Merge	
ST-55	Execution results [17120684687267]	ST-3	2480021	'Merge	
ST-54	Execution results [17120684687264]	ST-3	2480007	'Merge	

Test Plans List: Spring Tutorial

Key	Summary	Ru versions	Status
ST-3	testing for current sprint		