# JIRA 技术架构概览

On this page:

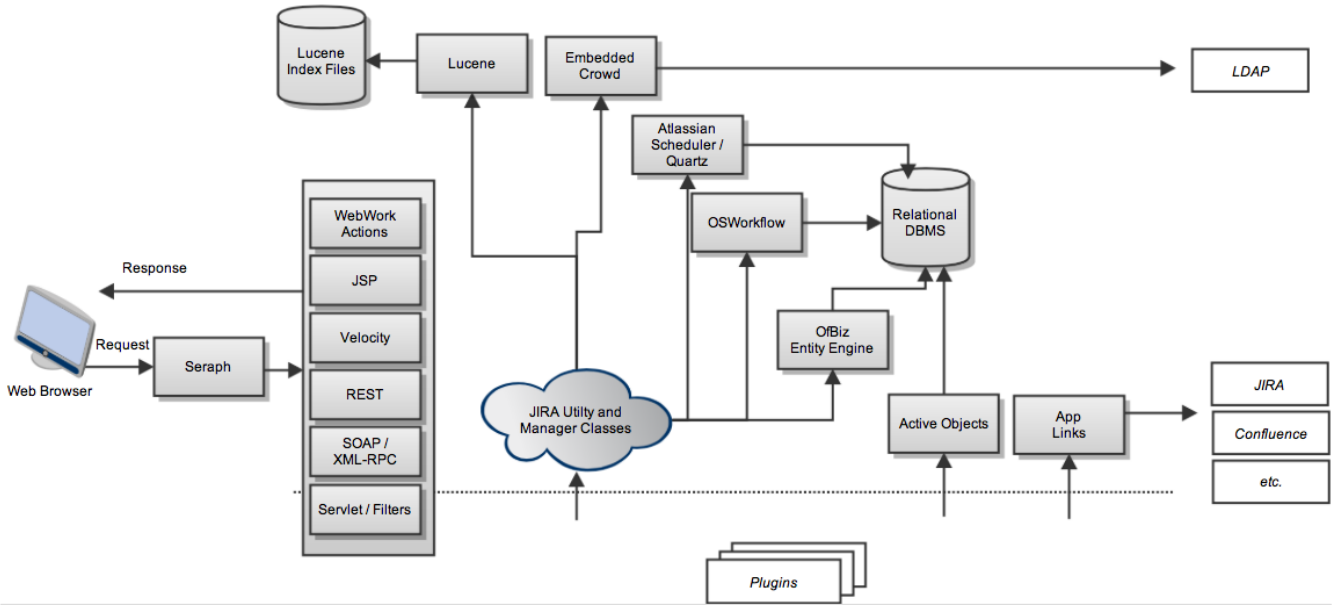


# JIRA Overview

This page provides a very high level overview of JIRA's dependencies and the role each one plays in JIRA. This page makes references to external resources (websites, books) where one can find more information.

## Technical Introduction to JIRA

JIRA is a web application written in Java. It is deployed as a standard Java WAR file into a java Servlet Container such asTomcat.




## WebWork

As JIRA is a web application, users interact with JIRA using a web browser. JIRA uses OpenSymphony's WebWork 1 to process web requests submitted by users. Please note that WebWork 1, not 2, is used. WebWork 1 is a MVC framework similar to Struts. Each request is handled by a WebWork action which usually uses other objects, such as utility and Manager classes to accomplish a task.

JIRA uses JSP for the View layer. So most of HTML that is served to the user as the response to their web request is generated by a JSP. Therefore, to generate a response the WebWork action uses a JSP.

Also see JIRA Webwork Actions.

For more information on WebWork 1 please see its online documentation.

## Seraph

Almost all authentication in JIRA is performed through Seraph, Atlasian's open source web authentication framework. The goal of seraph is to provide a simple, extensible authentication system that we can use on any application server.

Seraph is implemented as a servlet filter. Its sole job is, given a web request, to associate that request with a particular user. It supports several methods of authentication, including HTTP Basic Authentication, form-based authentication (ie. redirect to an internal or external login form), and looking up credentials already stored in the user's session (e.g. a cookie set by a SSO system).

Seraph performs no user management itself. It merely checks the credentials of the incoming request, and delegates any user-management functions (looking up a user, checking a user's password is correct) to JIRA's user-management -JIRA:Embedded Crowd (discussed later in this document).

If you were looking to integrate JIRA with a Single Sign-On (SSO) solution, you would do so by writing a custom Seraph authenticator ( and in fact, many customers have done so). Please note that by default JIRA is not shipped with any SSO integration, customers have to write a custom Authenticator themselves.  You may also want to check out Crowd andintegrating JIRA with Crowd.

Another very important function that Seraph performs in JIRA is to only allow users with Global Admin permission to access WebWork actions that allow the user to perform administration tasks. These WebWork actions are accessed by URLs starting with "/admin". For more information on JIRA's permission please see JIRA's documentation.

For more information on how seraph works internally please see this page.

## Embedded Crowd

Crowd is Atlassian's Identity Management and Single Sign On (SSO) tool.
Both JIRA and Confluence now embed a subset of Crowd's core modules for powerful and consistent user management.

Embedded Crowd provides the following functionality:

1. Stores users and groups in JIRA's database
2. Stores group membership (which users are part of which groups) in JIRA's DB
3. Authenticates users (checks if the users password matches)
4. Provides API that allows to manage (create, delete) users, groups and group memberships (add and remove users from groups).
5. Allows JIRA to connect to external systems to retrieve user /group data (eg Microsoft AD, LDAP or a standalone Crowd server)

6. Keeps a copy of any external data in the local DB for faster retrieval, and synchronises in the background.

As mentioned previously, Seraph delegates to Embedded Crowd to authenticate the user (i.e. check whether the correct password has been entered when a user tries to login).

For information on the DB tables used by Embedded Crowd see "Users and Groups" in the Database Schema page.

JIRA supports the use of JIRA:PropertySet to store user preferences
In JIRA the preferences include things like:

whether the user would like to receive HTML or Text e-mail
number of issues to display in JIRA's Issue Navigator
whether to receive notifications for user's own updates to issues
Locale (Language) of the user

In addition, Embedded Crowd also has its own concept of "User Attributes".
The two concepts, although related, do provide different advantages and disadvantages, so plugin developers should consider their individual requirements in order to choose the more suitable way.

User properties in PropertySets.

PropertySet values are typed.
There are 11 different types allowed including numbers, dates, arrays of bytes, parsed XML DOM's, nested Properties, and arbitrary serialized Objects.

These can be arbitrarily large, as they can be stored as CLOBs or BLOBs in the database.
These large amounts of data are not suitable for searching against.
PropertySets only allow a single value to be stored against a given key.

Crowd Embedded user attributes.

Crowd Attributes store only text data.
In order to facilitate searchability, Crowd limits the values to 255 characters or less.
(Note that MySQL has a limitation that means you can't index columns with > 255 characters )
Crowd allows multiple values to be stored against a single key.
These values must all be unique with a case-insensitive test.
(This follows the standard LDAP behaviour).

## PropertySet

OpenSymphony's PropertySet is a framework that can store a set of properties (key/value pairs) against a particular "entity" with a unique id. An "entity" can be anything one wishes. For example, JIRA's UserPropertyManager uses PropertySet to store user's preferences. Therefore, in this case, the "entity" is a User.

Each property has a key (which is always a java.lang.String) and a value, which can be:

1. java.lang.String

2. java.lang.Long
3. java.util.Date
4. java.lang.Double

Each property is always associated with one entity. As far as PropertySet is concerned an "entity" has an entity name, and a numeric id. As long as the same entity name/id combination is used to store the value and retrieve the value, everything will work.

In JIRA PropertySet uses the following database tables:

1. `propertyentry` - records the entity name and id for a property, its key, and the data type of the property's value. Each record in this table also has a unique id.
2. `propertystring` - records String values
3. `propertydecimal` - records Double values
4. `propertydate` - records Date values
5. `propertynumber` - records Long values

Each of the records in property<type> tables also has an id column. The id is the same as the id of the propertyentry record for this property. As the property's key and value are split across 2 tables, to retrieve a property value, a join needs to be done, between propertyentry table and one of the property<type> tables. Which property<type> table to join with is determined by the value of the `propertytype` column in the propertyentry record.

Here is an example of a full name stored for a user:
(to do)

PropertySet is used in JIRA:

1. By the UserPropertyManager to store users preferences.
2. To store Application Properties, which are configurable settings that a user can change to customise their installation of JIRA. For more information on Application Properties please see JIRA's documentation.
3. To store chosen preferences of gadgets on user's dashboards.

For more information on PropertySet please see its documentation. Also see JIRA Database Schema.

## JIRA Utility and Manager Classes

A lot of business logic in JIRA is implemented in 100s of java classes. The classes can be simple utility classes or Manager Objects.

Manager Objects in JIRA usually have one specific goal (or topic). For example`com.atlassian.jira.project.version.VersionManager` is used to work with project versions, i.e. create, update, delete and retrieve versions.

Manager objects use a lot of external dependencies, most of which are open source, but some are developed by Atlassian and are usually shared between Atlassian products.

Since JIRA 3.7 Manager classes are generally also wrapped by a corresponding service class. The idea is that any validation of business logic necessary is carried out by the service classes whereas manager classes are responsible for actually doing the action. For

instance see the ProjectService's validateCreate method and it's corresponding create method. The ProjectManager then only has a create method which will go off and create a project assuming any validation has already been carried out by the client.  This allows clients to simply call the service class in order to validate and create a project, but still gives the flexibility of circumventing validation if the ProjectManager is used directly.