|  |  |
| --- | --- |
| ***Project Acronym:*** | SUPERSEDE |
| ***Project Title***: | SUpporting evolution and adaptation of PERsonalized Software by Exploiting contextual Data and End-user feedback |
| ***Call identifier:*** | *H2020-ICT-2014-1* |
| ***Topic:*** | ICT-09-2014 Tools and Methods for Software Development |
| ***Type of Action:*** | Research and Innovation action |
| ***Grant agreement no.:*** | 644018 |
| ***Starting date:*** | 1 May 2015 |
| ***Ending date:*** | 30 April 2018 |

**Guidelines for FE-level Integration**

**With Issue Tracking Software**

|  |  |
| --- | --- |
| **WPxxx** | xxxx |
| **Tasks xxxx** | ... |
| **Due Date** | Xxx |
| **Submission Date:** | xxxx |
| **Deliverable Responsible:** | Xxx |
| **Version:** | 0.3 |
| **Status:** | Final |
| **Author(s)** | DELTA: Paolo Busetta. Andrea Gottardi  ATOS: Jesús Gorroñogoitia |
| **Reviewer(s):** | xxxx |
| **Deliverable Type** | R |
| **Dissemination Level** | PU |

**Version History:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Author** | **Date** | **Description** |
| 0.1 | Delta | 11/08/2017 | First draft |
| 0.2 | Delta, Atos | 04/09/2017 | Inserted contribution from ATOS concerning SSO |
| 0.3 | Delta | 22/09/2017 | RePlan integration with JIRA |

***Table of Contents***

[1 Executive Summary 5](#_Toc492295581)

[2 Introduction 6](#_Toc492295582)

[3 General Reference Architecture 7](#_Toc492295583)

[3.1 Main attention points for FE-level integration 8](#_Toc492295584)

[4 Hosting engineering process: methodological considerations 8](#_Toc492295585)

[5 Hosting ITS characteristics 10](#_Toc492295586)

[6 Integration support from the FE 10](#_Toc492295587)

[7 Integration with JIRA 10](#_Toc492295588)

[7.1 Desired Software Engineering Process Features 10](#_Toc492295589)

[7.2 Functional Requirements of the SUPERSEDE JIRA Plug-in (SPI) 12](#_Toc492295590)

[7.2.1 Epics 13](#_Toc492295591)

[7.2.2 User stories for Epic DataAlignment 13](#_Toc492295592)

[7.2.3 User stories for Epic JIRAasMainGUI 13](#_Toc492295593)

[7.2.4 User stories for Epic AlertManagement 14](#_Toc492295594)

[7.2.5 User stories for Epic Prioritization 14](#_Toc492295595)

[7.2.6 User stories for Epic ReleasePlanning 15](#_Toc492295596)

[7.3 SPI user interface design: interaction style and modality, menu, forms, layouts 16](#_Toc492295597)

[7.3.1 Alert management 16](#_Toc492295598)

[7.3.2 Prioritization process 17](#_Toc492295599)

[7.3.3 Release Planning process 18](#_Toc492295600)

[7.4 SPI architecture 18](#_Toc492295601)

[7.4.1 Single-Sign On (SSO) support; automatic login 18](#_Toc492295602)

[7.5 Set up of SPI development environment 19](#_Toc492295603)

[7.6 SPI installation on a JIRA system 20](#_Toc492295604)

[8 References 21](#_Toc492295605)

Acronyms

|  |  |
| --- | --- |
| **Acronym** | **Description** |
| ITS | Issue Tracking System |
| SS | SUPERSEDE |
| FE | SUPERSEDE Front-End |
| IF | SUPERSEDE Integration Framework |
| API | Application Programming Interface |
| DM | Decision-Making (as SUPERSEDE component) |
| DMPP | DM Prioritization Process |
| RP | Release Planning (as SUPERSEDE component RePlan) |
| SPI | SUPERSEDE Plug-In for JIRA |

# Executive Summary

This document summarizes the requirements and general architecture for the integration of SUPERSEDE with a third-party issue tracking software or similar project management tool via the front-end services. It describes the requirements, architectural and UI design for the JIRA plug-in. It provides a manual on how to use the development environment for this plug-in and how to install the plug-in on a JIRA installation.

# Introduction

This document summarizes the requirements and general architecture for the integration of SUPERSEDE with a third-party issue tracking software or similar project management tool via the front-end services. It describes the requirements, architectural and UI design for the JIRA plug-in. It provides a manual on how to use the development environment for this plug-in and how to install the plug-in on a JIRA installation.

This document is structured as follows: Sec. 3 describes the approach to be taken for integration from a technical perspective in the form of a reference architecture. Sec. 4 highlights how this integration would enhance a software engineering process not SUPERSEDE-centred. Sec. 5 defines the expected characteristic of the third-party hosting software to be a good candidate for integration. Sec. 6 highlights the support built into the FE and independent from the third-party tool in use. Sec. 7 details the integration with JIRA.

# General Reference Architecture

The following figures are extracted from [1]. Figure 1 shows the SUPERSEDE Front-end in a standalone installation, with no interaction with any third-party tool in use by the end-user organization. Figure 2 shows the SUPERSEDE Front-end when a third-party issue tracking system (ITS for short) or similar project management tool is in use and SUPERSEDE extends the functionality of the latter.

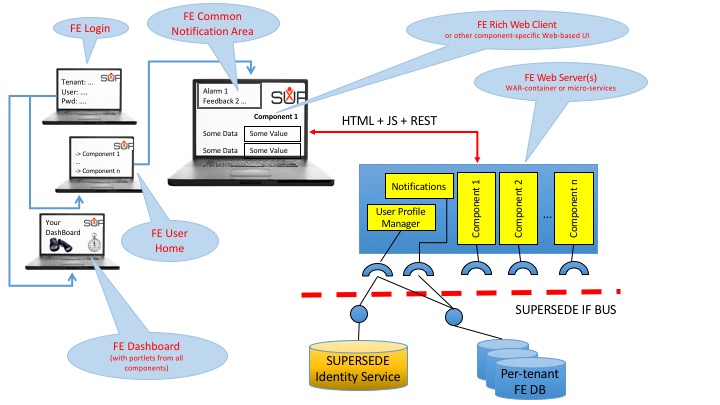


Figure 1: High-level architecture of SUPERSEDE FE, standalone installation

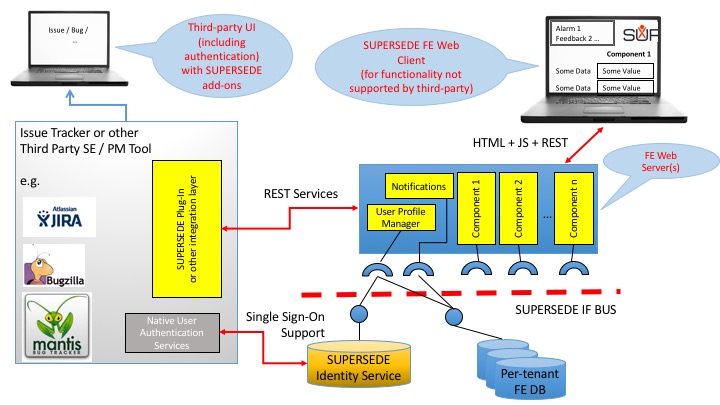


Figure 2: Architecture of the FE as third-party hosting tool extension

In this approach, alternative to (or complementing) back-end integration (e.g. with direct communication among servers or with a common enterprise bus), the normal split between Web client running within the browser and Web server is exploited to (partially or entirely) substitute the native SUPERSEDE GUI (i.e. the Web client) with the one offered by the ITS but extended with a SUPERSEDE plug-in, without touching the SUPERSEDE FE Web server. Data exchanges between SUPERSEDE and ITS are mediated by the SUPERSEDE plug-in, which allows the end-user to select which data to move from one system to the other based on the activities s/he is performing. Thus, SUPERSEDE appears as an extension of the native functionality of the hosting ITS[[1]](#footnote-1), providing additional data and services.

This approach, which we call FE-level Integration, leaves the back-end SUPERSEDE services unaffected by deployment choices concerning the user interfaces, thus reducing integration efforts. However, it opens a few attention points up (see discussion below) and it is not suitable to support automatic, massive data transfers (e.g. as those generated by alerts that need to be elaborated in real-time by installation-specific services) or for occasional small scale imports / exports (e.g. to enterprise-level project monitoring tools such as Microsoft Project, better served with ad-hoc CSV file creations).

## Main attention points for FE-level integration

**Confusion of end users**. If not all user interfaces are provided by the SUPERSEDE plug-in, the user may have to switch between layouts, interaction modalities, processes that are very different, each requiring a learning process that is harder the less a tool is used – thus SUPERSEDE would unavoidably be disadvantaged against an ITS used daily, possibly leading to rejection after some initial experimentation.

**Potential redundancy of databases, disalignment risks**. Data originating from SUPERSEDE may flow into the ITS and viceversa. Redundancy (multiple copies of the same data) is not an issue per se as long as they are all unmodifiable. If they are modifiable but modality and processes are not clear, there are multiple risks, e.g.: (1) the user doesn’t know what is the right interface to use to modify data; (2) modifications from one side are not reflected on the other.

**Multiple authentications.** While providing user credentials once forever or in rare occasions may be acceptable, forcing the user to double authentication (into the ITS and into SS) on a daily basis in order to perform her/his own work may be considered a nuisance, as being forced to use different user names and passwords for different systems is, undoubtably. A Single-Sign On system, possibly exploiting SUPERSEDE’s integrated WS02 Identity Management server and simultaneously opening two user sessions on the browser after authentication (one on the ITS and the other on SS), is highly recommended.

# Hosting engineering process: methodological considerations

This discussion applies to software evolution (as meant in SS), since its requirement identification / prioritization / planning steps are unavoidably part of any software engineering process, in a way or another. Similar considerations may apply to adaptation, where its event identification / decision as reaction / tasking of the related activities steps are part of the daily routine of operating and administering a (large) application infrastructure. In the following however we restrict our focus to software evolution.

In general, SUPERSEDE will be adopted within an existing organization. Existing sources of requirements may vary according to its processes already in place. Adding a data-driven one requires the identification of a repository of requirements and of a workflow where an actor with the appropriate competences and authority analyses inputs from SS (in the form of alerts) and decides if they are evidence for new requirements or they enforce some that were already present in the repository. An organization that already uses an ITS as repository, and thus already has somehow defined a requirement identification workflow and its related actors, can easily extend this workflow to adopt SS if this is integrated with the ITS.

Decision-making concerning what goes in the next release is often a highly manual operation. For instance, in SCRUM the Scrum Master decides the so-called Sprint backlog; the process of Sprint planning is done in a kick-off meeting with the team. Worth noticing: no computer tools are prescribed for the decision process; the Scrum Master has a role of negotiator between stakeholders (represented by the Product Owner) and the development team. No matter whether SCRUM is adopted or not, a similar approach is very common in most organizations. The introduction of a distributed, asynchronous DM tool that can be used for various purposes (helping the Scrum Master to decide what goes in the next release, helping on deciding an order of development, involving additional stakeholders in the process) is undoubtedly facilitated by having the repository mentioned above already in place and integrated with SS. Still, for full acceptance it is required that (1) SS accepts all type of requirements within the repository, not purely those generated from alerts; (2) SS enables the involvement of stakeholders that are not also involved in the requirement repository; (3) the DM step in itself is optional and not rigidly inserted in a workflow, leaving the organization free to rearrange its.

Similar considerations apply to release planning, which should be an optional step as the DM. However, the picture is somewhat more complex because some of the information required by RP, e.g. resource availability, is typically available within the organization but maintained by more specialized tools, i.e. project planning and tracking systems, commonly used but not necessarily fully integrated with the ITS. For instance, JIRA has a number of plug-ins for planning and tracking with different approaches to resource data storage[[2]](#footnote-2).

In conclusion, it is necessary to consider SUPERSEDE as a collection of three macro-services (alert detection, prioritization, release planning) that may be installed and used separately and independently by an organization according to the existing process and tools.

# Hosting ITS characteristics

As mentioned above, the presence of an ITS to support a software engineering process should facilitate the introduction of SS in an organization. Specifically, an ITS should be considered as candidate for SUPERSEDE hosting when it shows the following characteristics:

* It constitutes a repository of requirements (in the generic sense of “things to be addressed” – terminology varies, as discussed below) that are tracked thru their entire lifetime, from identification to conclusion;
* It is meant for daily use by a development team and possibly by others (including operations);
* It supports one or more specific software engineering processes.

As mentioned, the objects maintained by an ITS that correspond to SUPERSEDE objects may have varying denominations and a preliminary analysis and decision on how to map them must be done per-ITS. Typical terms include “issue”, “ticket”, “bug report” which may correspond to SS “requirements”, “features”, “alerts”. This analysis is particularly important both for its impact on the integration efforts and to clarify the unavoidable differences in terminology used by the two tools (ITS and SS) to their users.

# Integration support from the FE

The principal support offered by FE to an ITS plug-in are the REST calls used by the FE’s Web client, listed in the following.



# Integration with JIRA

## Desired Software Engineering Process Features

JIRA, produced by Atlassian, is a widely used, proprietary issue tracking system developed in Java. Its adopters include the Apache Software Foundation to report bugs on many open-source projects (see <https://issues.apache.org)>, as alternative to BugZilla. JIRA has a large eco-system of plug-in tools[[3]](#footnote-3) that extend its functionality well beyond issue tracking and aim at providing a complete project management solution. Note that the extensions to the base platform to support Agile processes and in particular SCRUM, originally a plug-in, have been incorporated in the base itself given their popularity.

Plug-ins are developed in Java for the server side, and can access Javascript libraries on the Web client; a relatively easy-to-use SDK is released by Atlassian at no cost. These technical and licensing characteristics, in addition to the fact that JIRA matches the requirement described in Sec. 5 and other marketing considerations[[4]](#footnote-4), make JIRA an attractive platform for developing a SUPERSEDE plug-in.

In summary, a JIRA instance manages a collection of “issues” divided in “projects”. An issue is the description of something that needs to be addressed somehow by the community of users that work on the issue’s project. An issue, among many other properties, has a “type” that classifies it; the default catalogue of types includes “tasks”, “bug” and “features”, but others can be added on a project- or installation-basis. In other systems, the equivalent of an issue is sometimes named “ticket” or “task”; in SS, an issue largely overlaps with DM “requirements” and RP “features”, with slight differences in meaning. Issues have a state, representing their processing with respect to a workflow that can be defined per project; the default workflow moves issues through three states, “To Do”, “In Progress”, “Done” (incidentally, this terminology reveals that “issue” is really a synonym for “task” in JIRA).

In the following, when referring to “issue” we mean a “JIRA issue”.

Given the flexibility of JIRA, there are potentially many ways to use it to support a software engineering process. We focus here on SCRUM as an example, because it is well known and documented and specific support is built into JIRA[[5]](#footnote-5), but the way a SUPERSEDE plug-in is designed should not prevent different usages. From JIRA’s perspective, SUPERSEDE offers three main macro-services of interest:

1. Automatic alert generation;
2. Prioritization of activities;
3. Planning of activities.

As mentioned above, an “activity” as it is meant in SS (i.e. a requirement or a feature) corresponds to an issue, thus (2) and (3) above could simply the renamed “Prioritization of issues” and “Planning of work on issues”. It is tempting to apply a similar logic to (1): consider SS alerts as issues. However, by definition, alerts capture something notable within a certain amount of user feedback and monitoring data over a period of time; they may be repeated over time, they may not refer to problems or requests to be dealt with by the organization using JIRA, two of them may seem different but refer to the same problem, and so on, thus in general there is no one-to-one correspondence between alerts and issues and human analysis of alerts (possibly assisted by algorithms, e.g. to detect similarities) is necessary to identify related issues.

Figure 3 suggests how SS could enhance a SCRUM process supported by JIRA. Each SS-managed activity deals with JIRA issues. SS-managed activities are started or driven from JIRA (i.e. from SPI).

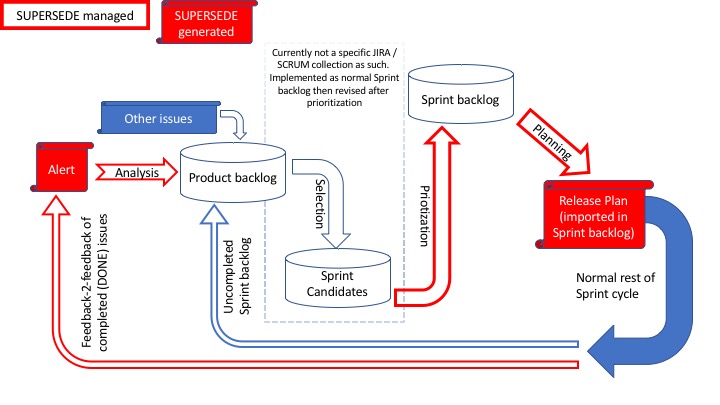


Figure 3: Potential SS-enhanced JIRA Agile Sprint process

## Functional Requirements of the SUPERSEDE JIRA Plug-in (SPI)

Requirements for SPI are described here as Epics and User Stories, using Epics as collections / summaries of correlated User Stories.

In the following, the notation [Y3US: n] is a reference to the User Stories collected in [2], Appendix D.

Actors:

* Support Team (maintaining product installation, interacting with end-users)
  + Subsuming roles such as Helpdesk, Operation team
* Team leader (manager of Development Team)
  + Subsumes SCRUM Master if Agile / SCRUM adopted
  + Subsumes DM Process / DM Game Supervisor
* Development Team (working on software evolution of product)
  + Includes the Team Leader as one of its members
* JIRA Daily User: shorthand for Support Team or Development Team member
* Non-developer Opinion providers (knowing or using the product)
  + Including: Product owner (product manager), Product end-users
* System Administrator (administering the installations of both JIRA and SS)

Assumptions:

* JIRA is used by Support Team to report bugs, take note of requests from users, schedule maintenance activities on a daily basis;
* JIRA is used by Team Leader to define and assign activities to Development Team on a daily basis;
* JIRA is used by Development Team to mark progress on assigned activities on a daily basis;
* Non-developer Opinion providers may not have JIRA access.

No assumption is made on adopted process, even if some terminology is inherited from SCRUM.

### Epics

1. [DataAlignment] As a JIRA Daily User, I want to access SS functionality to operate on my JIRA issues (create / annotate / prioritize / schedule …) without having to duplicate information across systems by hand so that I minimize my efforts and don’t risk misalignments.
2. [JIRAasMainGUI] As a JIRA Daily User, I want to use JIRA not only for all its normal functionality but also for interacting with SS so that I don’t get confused by different interaction models or duplicated functions.
3. [AlertManagement] As a Support Team member, I want to see the alerts generated by SS so that I can create or update JIRA issues concerning bugs, requests or maintenance activities to be performed.
4. [Prioritization] As a Team Leader, I want to prioritize the currently open JIRA issues for my team by means of SS DM so that I can decide what to develop for the next software release (or, equivalently, Sprint).
5. [ReleasePlanning] As a Team Leader, I want to take the JIRA issues representing what to develop for the next release and schedule their development by means of SS RP so that I can plan the activities for my team.

### User stories for Epic DataAlignment

This Epic appears to be, at least partially, a non-functional requirement that affects architecture and all other Epics, the latter concerning at least the phases of creation and removal of SS objects at the end of operations.

1. [DataHandledInJIRA] As JIRA Daily User, I want that all SS user functionality concerning the modification of alerts, requirements for DM and features for RP as well as transformations of one in the other are disabled, so that I can rely on JIRA as my only correct and complete database.
2. [LinkBackToJIRA] As JIRA Daily User, I want that any SS object that mirrors a JIRA-managed object has a link back to the latter, so that I can always access the JIRA source and modify the latter if necessary.

### User stories for Epic JIRAasMainGUI

This Epic, too, appears to be, at least partially, a non-functional requirement that affects architecture and all other Epics, since it requires most SS UIs to be implemented as JIRA extentions.

1. [SingleSignOn] As JIRA Daily User, I want to log into JIRA and automatically be authenticated with a specific user and tenant of SS of my choice so that I don’t have to manage double sessions.
2. [Authorization] As System Administrator, I want to have a way to enable all or a selected set of JIRA users to use SS on a specific tenant, so that I don’t need to duplicate authentication information.

### User stories for Epic AlertManagement

1. [AlertAsDocument] As Support Team member, I want that any alert generated from SS is dealt within JIRA as a document reporting the entire information collected by SS, which I can read with an appropriate application callable from my Web browser and attach to an arbitrary number of JIRA issues, so that I can always analyze the SS source while freely deal with issues the JIRA way.

Indirectly addressing [Y3US: S1, S2, S3, S4, S7, S8, S9], since all of them have to be applied to JIRA issues instead (see also [ClusterAlerts, SuggestIssue] below). [Y3US: S5, S6] will be automatically addressed when the required data will be attached to alerts from the originator (WP2).

1. [SummaryOfAlerts] As Support Team member, I want to see a summary list that reports which alerts have been collected by SS in order of their creation (FIFO) and, for each, if and to which issues they have been already attached, so that I know which problems or requests have been reported recently and if they have been already managed them.
2. [DealingWithAlerts] As Support Team member, I want to be able to attach an alert to a new or existing issue in a JIRA project of my choice directly from the summary list, so that I can manage them efficiently.
3. [CleanAlertsUp] As Support Team member, I want to delete alerts from SS directly from the summary list, so that I can avoid re-examining the same alerts and keep the list of those left to analyse down to a manageable size.

It should be noted that attachments (see [DealingWithAlerts]) are managed by JIRA by internally copying the document, thus the original ones can be changed or removed without affecting any referring issue.

1. [ClusterAlerts] As Support Team member, I want that SS rearranges the summary list on demand by similarity, so that I can easily spot related or duplicated alerts.

Indirectly addressing [Y3US: S9], since multiple alerts concerning the same topic should be grouped together.

1. [SuggestIssue] As Support Team member, I want that to see any SS suggestion of issues to which to attach specific alerts directly from the summary list, so that I minimize the work of identifying alerts that refer to known issues.

Indirectly addressing [Y3US: S9], since multiple alerts concerning the same topic should be automatically directed to the same issue.

### User stories for Epic Prioritization

1. [CreatingDMProcess] As Team Leader, I want to create a DM Prioritization Process from the issues that I select with an arbitrary JIRA query, so that I can have them prioritized.
2. [ExportingIssuesAsReqs] As Team Leader, I want that any JIRA issue selected for a DMPP is copied into a new SS requirement, so that there is a one-to-many correspondence between issues and requirements while multiple DMPP instances deal with their own SS copy.

Indirectly addressing [Y3US: Q1, Q4, Q13, I6], since SS requirements point back to the original JIRA issue (see [LinkBackToJIRA] above) and the latter can be examined and commented. Complementary story is [CleaningSSReqs]; together, they avoid the need for garbage collection of SS requirements. Observe that [DataHandledInJIRA] prevents modifications from SS.

1. [RequirementIssueType] As Development Team member, I want to be able to indicate that a JIRA issue is a requirement for development by means of a specific type in addition to those natively offered by JIRA, so that I can restrict the queries for creating a DMPP.

It should be noted that there are at least two technical solutions (add an issue type to JIRA, add a Boolean application-specific field to issues) whose choice would require further analysis on the expected usage (a “requirement” seems to subsume, rather than replace, the native “task / bug / feature” classification). Further, if JIRA Agile is used, a third and probably better approach would be to create a Sprint backlog before prioritization and use it for the query.

1. [MarkingDependency] As Development Team member, I want to indicate that there is a dependency between two issues using the native JIRA linking mechanism with a new “dependency” link type, so that I can indicate their required order of development.

Dependencies in JIRA are also captured with task / subtask relationships and “blocking” and “relating” dependencies; however, subtle differences in meaning and possible inconsistent usage strongly suggest adding a link type.

1. [ExportingDependenciesToDM] As Team Leader, I want that dependencies among JIRA issues selected for a DMPP are transformed into dependencies among the equivalent SS requirements, so that DMPP can take them in account.

Indirectly addressing [Y3US: Q6].

1. [QueryingDMProcess] As Team Leader, I want to see a summary of the DMPPs that I created, which issues have been included and the DM state, so that I know which processes are active.
2. [ClosingDMProcess] As Team Leader, I want to declare that a running DMPP is completed, so that I prevent further DM activities.
3. [ImportingDMResults] As Team Leader, I want to be able to import the result of a DMPP, if there is exactly one available, by changing the priorities of the affected JIRA issues according to the DM results clustered in the five native JIRA classes (very high, high, medium, low, very low).
4. [CleaningSSReqs] As Team Leader, I want that that all the data of a DMPP that I declared closed from JIRA are removed from SS, so we don’t leave old data around.

This complements [DataAlignment , DataHandledInJIRA]

### User stories for Epic ReleasePlanning

1. [ExportingIssuesAsFeatures] As Team Leader, I want that any JIRA issue selected for a release plan is copied into a corresponding SS RP features with all relevant information, when available (in particular priority and effort), so that I can use RP to assign staff to people and compute deadlines.
2. [UpdatingIssuesFromReleasePlan] As Team Leader, I want that planning information (start and completion dates, assigned people) computed by RP is imported into JIRA, so I can use JIRA to monitor plan execution.

## SPI user interface design: interaction style and modality, menu, forms, layouts

From a UI perspective, a plug-in is formed by a set of pages called from a menu hanging from the main JIRA bar. These pages must follow the layout and graphical style of JIRA, with SUPERSEDE logo placed in a visible but not obtrusive location.

The SPI hanging menu allows to call the main functions (roughly corresponding to the Epics given above) from anywhere in JIRA: alert management, decision-making, release planning, plus configuration. Since these calls are context-independent, the invoked forms must support a dialogue that allows the user to express what s/he wants to perform.

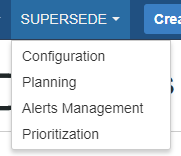


Figure 4: Example of JIRA haning menu for SPI

### Alert management

To support [SummaryOfAlerts] and most other user stories of [AlertManagement], invoking Alerts management from the hanging menu must open a summary like the following.

The example below is extracted from the prototype at the time of writing and most likely will change by the time of project validation.

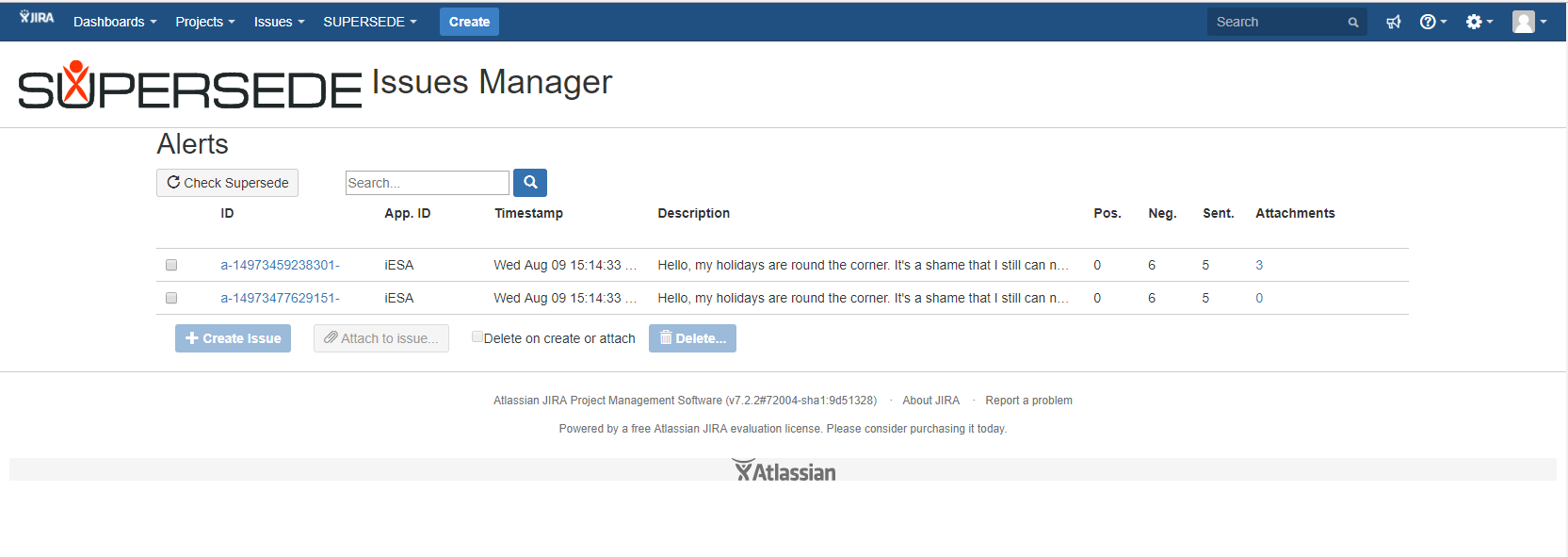


Figure 5: Example of layout for alert summary

### Prioritization process

To support [CreatingDMProcess] and [QueryingDMProcess], a unified form is suggested, that allows the user to select a JIRA query to create new processes and see what is already running and how it was created.

The example below is extracted from the prototype at the time of writing and most likely will change by the time of project validation.

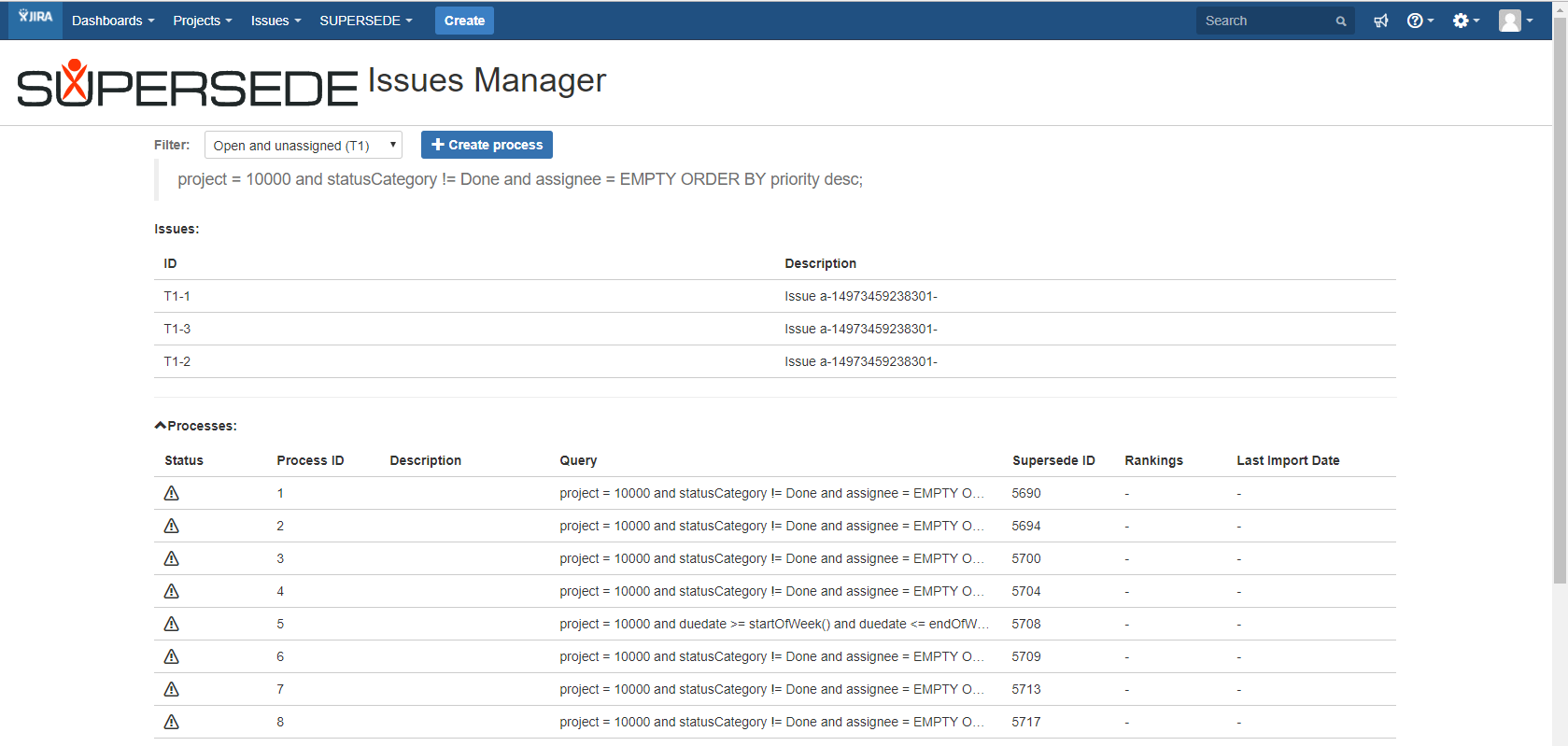


Figure 6: Process-generating JIRA query, summary of running process

When clicking on the Create Process button, a modal creation form must appear, as in the following example:

The example below is extracted from the prototype at the time of writing and most likely will change by the time of project validation.

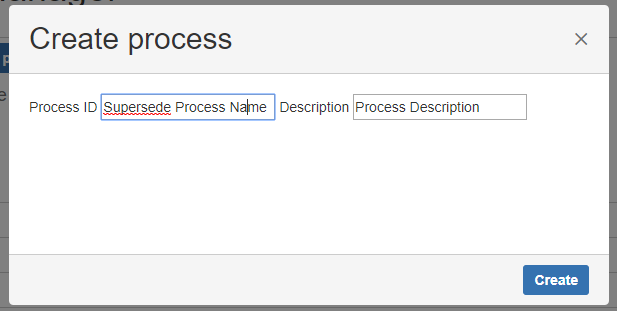


Figure 7: Process creation form

### Release Planning process

A configuration form is required to specify: (1) the end-point for RP; (2) a correspondence table between JIRA users and resources configured in RP:

The example below is extracted from the prototype at the time of writing and most likely will change by the time of project validation.

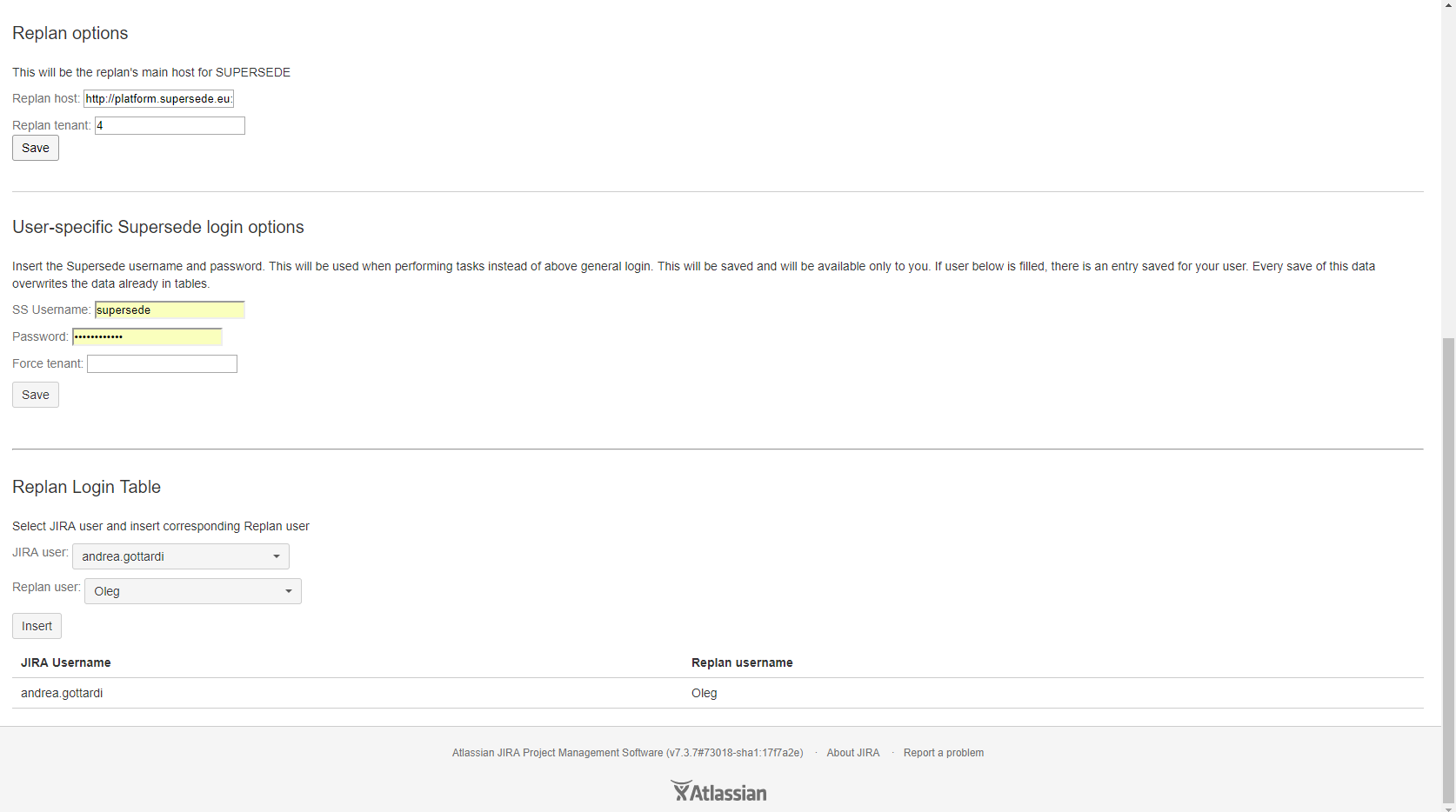


Figure 8: Configuration of RePlan endpoint and resources

To support [ExportingIssuesAsFeatures] and [UpdatingIssuesFromReleasePlan], an import / export facility is available. Similar to the DMPP creation, a JIRA query filter is used to select the issues to be exported as RP features or for which the release planning results have to be imported in JIRA. The figures below show the results of an export to RP operation for a single issue, the first showing that a new feature has been created, the second that a feature corresponding to the selected JIRA issue is already in RP.

The example below is extracted from the prototype at the time of writing and most likely will change by the time of project validation.

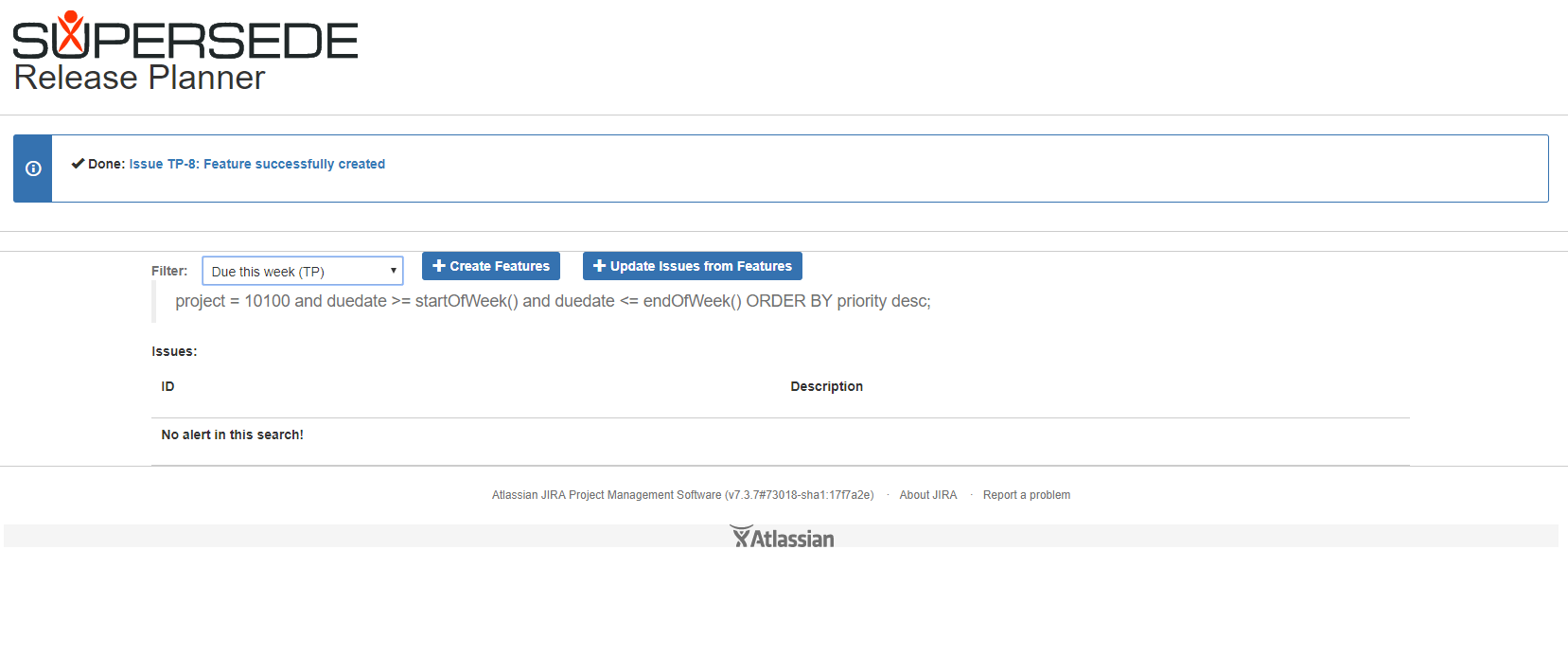


Figure 9: Exporting a JIRA issue as RP feature, success case

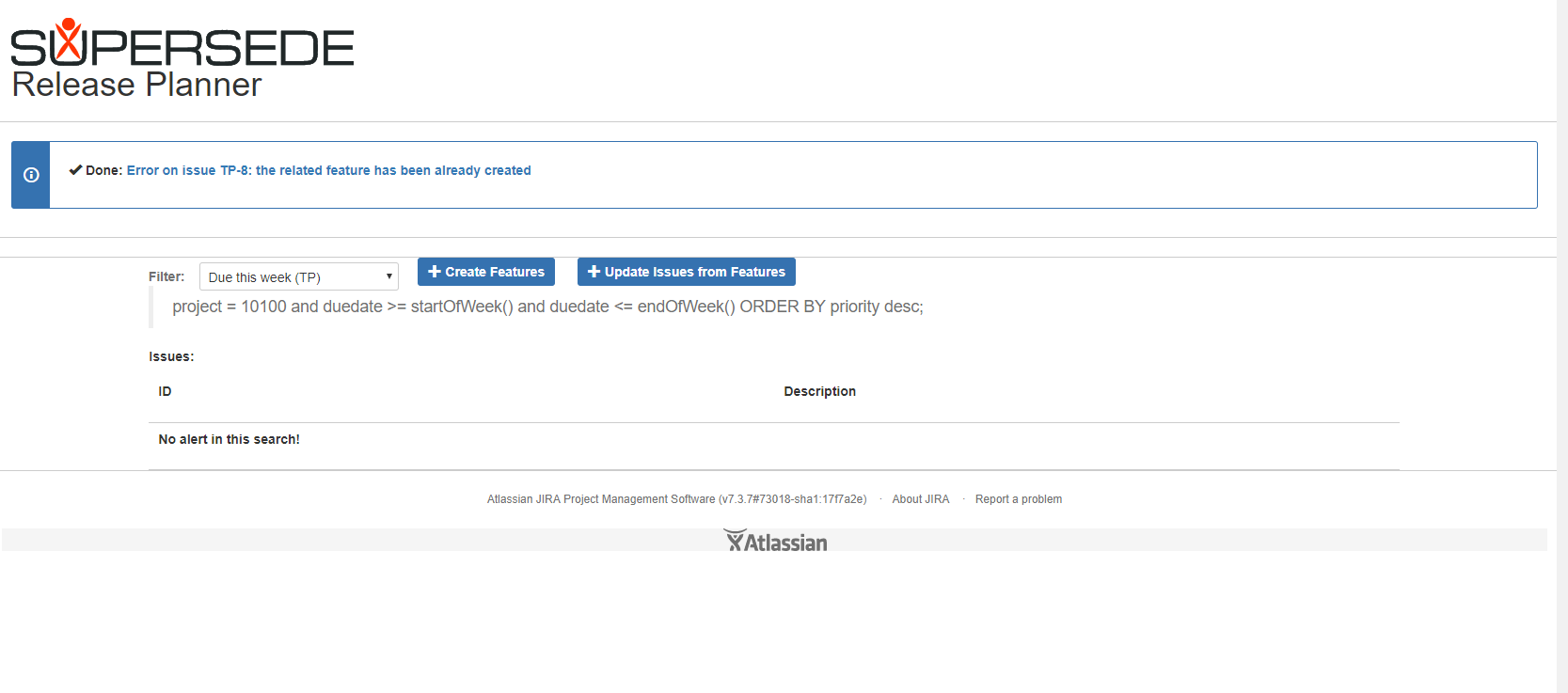


Figure 10: Failed export to RP because of duplication

## SPI architecture

To address one of the attention points described in Sec. 3.1, “Potential redundancy of databases”, the approach taken privileges JIRA as the main repository of information used by the Development Team. Alerts originated in SS are transformed in documents that are saved as attachments to issues, thus incorporated into the JIRA database, after which they can be safely removed from SS even if further information (e.g. user feedback and examples stored in Kafka) should be maintained and properly referred via permalinks or other identifiers stored within the alerts themselves. Conversely, issues that need to be prioritized and addressed by a release plan are transformed into unmodifiable SS objects (at least concerning the fields that are generated from the JIRA repository) that are managed and possibly deleted at the end of the required SS activity.

To support any type of software engineering process, issues should not require special workflows or additional information to receive alerts as attachments and to be exported to DM and RP. There are however two exceptions, dictated by [RequirementIssueType] and [MarkingDependency]. The first is the creation of a special-purpose “requirement” issue type, whose objective is to clarify the intended purpose of an issue and simplify queries from JIRA; its use however is not mandatory. The second is the creation of a “dependency” link that expresses a dependent / dependee relationship between issues that semantically corresponds to its meaning as intended in SS, i.e. that a requirement (the dependent) that cannot be satisfied unless another (the dependee) is not satisfied first. This kind of relationship sounds similar but does not exactly matches various other types of link and relationships in JIRA, so it is preferred to have a new one that, in the worst case, duplicates information already captured in JIRA itself.

The decision whether to reimplement or not all SS UIs concerning DM and RP within JIRA, which would address [JIRAasMainGUI] in the best way, is not constrained by the architecture; rather, it is left to project management considerations concerning, e.g., duplication of efforts and availability of development resources.

### Single-Sign On (SSO) support; automatic login

User authentication is supported by SUPERSEDE by means of the WS02 Identity Server (WS02 IS) [1]. SS’ FE calls, by means of an API provided by SS’ IF, WS02 IS to check the user credential; the IF uses WS02 IS to generate a security token that is used to validate the REST calls among the SS services.

WS02 IS supports single-sign on (SSO)[[6]](#footnote-6), with user credentials checked against third-party user directories. To exploit this capability from SS, some configuration is required on the WS02 IS instance and some work is required on the FE, either to direct the underlying Spring framework to use a standardized authentication protocol (e.g. OAuth2/OpenID or SAML2) or use a WS02 Java library called SSO Agent.

JIRA supports SSO by means of third party extensions for SAML[[7]](#footnote-7) as well as with additional techniques, in addition to Atlassian’s own SSO service[[8]](#footnote-8). Consequently, in principle it is possible to build a SSO facility for JIRA and SS.

Of course, SS’s WS02 IS and the SSO adopted for JIRA must refer to the same authentication domain / mechanism.

Given the high development costs, the customer-specific configuration requirements (which may impose, for instance, using a company’s LDAP repository), and the little value for the short-term project goals (i.e. Y3 validation), it has been decided to not proceed with the full implementation of a SSO system, i.e. not to have a single authentication system and automatic user configuration for JIRA and SS; in other words, the user stories [SingleSignOn] and [Authorization] have not been fully implemented.

That notwithstanding, [SingleSignOn] has been partially addressed by letting SPI deposit an appropriate authentication cookie on the user’s browser after checking the JIRA’s user configuration parameters against SS with the “login” REST service. The result is that the user is not requested to log into SS after performing at least one operation on SPI. To work, this technique requires that JIRA and SS are installed in the same network domain (e.g., **.supersede.eu** in the case of the SS own testing and validation environments).

## Set up of SPI development environment

Prerequisites:

* The Atlassian Plugin SDK (https://developer.atlassian.com/docs/getting-started/set-up-the-atlassian-plugin-sdk-and-build-a-project);
* A SUPERSEDE installation.

The current GIT repository for SPI is

<https://github.com/supersede-project/supersede-issuetracker-integration>

To set up the development environment, do the following:

1. Install Eclipse (e.g. Oxygen or a later version).
2. Download the Atlassian plug-in and install it according to the manual. Let it install maven if this is not already on your system.
3. Pull SPI from GITHub. If you are using Eclipse as GIT interface, let it add the Eclipse project automatically.
4. Run “atlas-mvn eclipse:eclipse” on Eclipse terminal opened from within the project folder. It will automatically set up the environment, making it suitable for eclipse.
5. Run “atlas-run” on a command line to start JIRA with SPI installed.
6. Follow the installation and configuration instructions.

## Installing and configuring SPI on a running JIRA system

The operations described in this chapter must be performed by a JIRA administrator. Instructions refer to JIRA version 7.5.

### Prerequisites

* A JIRA instance, version 7.5 or more recent[[9]](#footnote-9), installed from a distribution kit or as part of the SPI development environment;
* Botronsoft Configuration Manager[[10]](#footnote-10) (BCM for short), installed on the JIRA instance (optional but recommended for clean installations);
* The JAR file with SPI, received with the SUPERSEDE distribution kit or produced by the SPI development environment;
* A ZIP file containing the predefined configuration for JIRA with SPI for use with the BCM, received with the SUPERSEDE distribution kit.

### Installation

In the JIRA Administration menu, select “Add-ons”. In the Add-ons page, select “Manage add-ons” in the left-hand menu bar. Within the Add-on management page, click on “Upload add-on” (top-right). A form appears that allows to import the JAR file of the SPI distribution from your local file system.

### System and user configurations

From the BCM:

Import the predefined JIRA environment (which include a predefined project) from the ZIP configuration file in the SUPERSEDE distribution kit.

In the following, NRIC (Not Required with Imported Configuration) indicates the operations that can be saved with the JIRA configuration and predefined project imported with BCM.

From the JIRA administration pages:

* (NRIC) Enable anybody to read issues (required to a non-JIRA user analysing requirements during a DM Game or while planning):
  + Project-specific setting: from the JIRA Administration menu, select Projects, then the project to be changed, then “Permissions”. Select the “Edit” action. Change the “Browse project” permission to grant it to a “Group” selecting “Anyone”.
  + Multi-project settings: change the Default permission scheme (or one of your choice) to enable “Browse project” to the “anyone” group, as indicated above. Look at the standard JIRA documentation to know about permission schemes and their applicability.
* (NRIC) Enable the “dependency” link (see user story [MarkingDependency]): from the JIRA Administration menu, select Issues, then Issue Linking. On this page,
  + activate issue linking (which should be active by default);
  + add a new link type called “Dependency”, with outward description “has dependent” and inward description “is dependent on”.

From the Project permission page (see above), edit the “Link issue” permission to enable “Application access” -> “Anyone logged in”; as above, this can be set in the Default or other appropriate JIRA permission scheme.

* (NRIC) Create a new user for SUPERSEDE server side operations with appropriate access to the interested projects. On a basic JIRA installation, a simple non-administrative user with the default application access to the core JIRA services is sufficient; see the JIRA manuals about how to create users. The username and password need to be configured in the SUPERSEDE services that push data to JIRA.
* (NRIC) (Optional – to be done only if desired, see user story [RequirementIssueType]) Create a new issue type to distinguish requirements (or other types of issues of interest) from the JIRA default “task” type or the JIRA Agile “user story” and others. Note that no special processing is performed by SPI on the type of issues, however this may help e.g. to create query filters to feed DMPP or RP. The addition of types must be done according to the standard JIRA documentation; in short, from the “Administration” menu, select “Issues”, then “Issue type” and finally “Add issue type” to add a “standard” one; then, select “Issue type scheme” and add it to an existing scheme or create a new one; finally, make sure that your project (or the new ones you will create) adopt the changed / new issue type scheme.

From the SPI configuration page:

As administrator (shared by all JIRA users):

* Input the URL of the SUPERSEDE FE instance (for alerts and DM);
* Input the default SUPERSEDE user, password and tenant necessary for accessing alerts, create DM processes, push issues to RP;
* Input the URL of the RP instance (to access the release plan);
* Associate JIRA usernames with corresponding RP-managed resources, to be used to assign issues when importing the results of RP into JIRA.

As individual user:

* Input the specific SUPERSEDE username and password that will be used for the automatic login operation from the browser where JIRA has been accessed.

### JIRA SS-enhanced project set-up

SPI does not constrain the usage of JIRA in any way. The following is simple advice on how best configure a JIRA project that exploits SPI.

The default JIRA project scheme has only one type of issue, “task”. The default workflow for the latter is a three state “TO DO / IN PROGRESS / DONE” state machine. The Agile / SCRUM project scheme has the typical Agile types: “user stories”, “epics”, “features”, each with the default workflow.

The choice of the project scheme to apply, or the extensions that may be needed, require a case-by-case analysis. In general:

* A project meant for operations, i.e. reacting to bugs or problems arising during an application activity including those discovered by SUPERSEDE and notified with alerts, is best configured with the default schema;
* A project meant for software development, thus using DM and RP no matter whether adopting SCRUM or not as its base process, is best configured with an Agile project scheme. This type of project may benefit from type extensions and from the dependency link (see configuration above) to fine tune the issues and provide important planning information. To this end, an ad-hoc workflow may be appropriate that enriches the base states with intermediate ones such as “being evaluated” (e.g. to identify issues to be prioritized next) or “being deployed” (in a DevOps chain, this would identify the stage where the issue has been tackled in code but is not yet available to the users; in perspective, a feedback-to-feedback signal could be generated when moving from being deployed to DONE, letting the end-user know of the availability of the required fix or feature).

### Querying issues for import / export to SUPERSEDE

As for project set-up, SPI does not constrain the usage of the JIRA query language to create filters that are used for import/export operations from/to SUPERSEDE. We highlight here some interesting points:

* Queries are run every time they are used in SPI, so their results may change if not properly constrained (e.g. by ranges such as issue creation date and issue state);
* Give easily identifiable names to queries that you are going to use with SPI, so that you can distinguish them when creating or querying e.g. DM processes;
* When reusing a previously created filter for a new query (e.g. to change constraints), remember to save it under a different name to prevent confusions with running activities;
* In an Agile project, a query can select issues based on the Sprint name, which is a convenient way to identify backlogs.

# References

1. Gorroñogoitia. J et al. „Requirements and architecture for the integration framework v3“. SUPERSEDE report (2017)
2. Kifetew. F et al., “D3.3 – Decision Making Processes – v2”. SUPERSEDE report (2017)

1. From now on, we’ll use “hosting ITS” as a short hand for “third-party ITS with SUPERSEDE plug-in”. [↑](#footnote-ref-1)
2. A simple search on the JIRA marketplace for “planning” returns various such plug-ins; just as examples, see the Resource Planning Plugin by Altan Senel for a JIRA extension or The Connector - Microsoft Project to Atlassian JIRA Integration by Ecliptic Technologies for an issue import/export based approach. [↑](#footnote-ref-2)
3. https://marketplace.atlassian.com/addons/app/jira [↑](#footnote-ref-3)
4. E.g., popularity, installed base, user groups, public events in Europe and USA. [↑](#footnote-ref-4)
5. https://www.atlassian.com/agile/how-to-do-scrum-with-jira-software [↑](#footnote-ref-5)
6. https://docs.wso2.com/display/IS530/Single+Sign-On [↑](#footnote-ref-6)
7. SAML Single Sign On (SSO) for JIRA: <https://marketplace.atlassian.com/plugins/com.resolution.atlasplugins.samlsso.Jira/server/overview?gclid=EAIaIQobChMIi-PCrbWL1gIVxUAbCh3glgpQEAAYASAAEgJAcfD_BwE> [last accessed on 4th Sep 2017] [↑](#footnote-ref-7)
8. Some information is available at <https://confluence.atlassian.com/kb/single-sign-on-integration-with-the-atlassian-stack-794495126.html> [last accessed on 4th Sep 2017] [↑](#footnote-ref-8)
9. While SPI should have no specific dependencies from JIRA 7.5, which is the version used for development and testing, it is always possible that incompatibilities are introduced by more recent versions. [↑](#footnote-ref-9)
10. https://marketplace.atlassian.com/plugins/com.botronsoft.jira.configurationmanager/server/overview [↑](#footnote-ref-10)