LIÈGE université
Library

https://lib.uliege.be

matheO

https://matheo.uliege.be

Analysis and conception of a solution to synchronise Jira instances

Auteur : Bourdouxhe, Alexandre
Promoteur(s) : Fontaine, Pascal
Faculté : Faculté des Sciences appliquées
Diplôme : Master : ingénieur civil électricien, à finalité spécialisée en "signal processing and intelligent robotics"
Année académique : 2022-2023
URI/URL : http://hdl.handle.net/2268.2/17354

Avertissement à l'attention des usagers :

Tous les documents placés en accès ouvert sur le site le site MatheO sont protégés par le droit d'auteur. Conformément aux principes énoncés par la "Budapest Open Access Initiative"(BOAI, 2002), l'utilisateur du site peut lire, télécharger, copier, transmettre, imprimer, chercher ou faire un lien vers le texte intégral de ces documents, les disséquer pour les indexer, s'en servir de données pour un logiciel, ou s'en servir à toute autre fin légale (ou prévue par la réglementation relative au droit d'auteur). Toute utilisation du document à des fins commerciales est strictement interdite.

Par ailleurs, l'utilisateur s'engage à respecter les droits moraux de l'auteur, principalement le droit à l'intégrité de l'oeuvre et le droit de paternité et ce dans toute utilisation que l'utilisateur entreprend. Ainsi, à titre d'exemple, lorsqu'il reproduira un document par extrait ou dans son intégralité, l'utilisateur citera de manière complète les sources telles que mentionnées ci-dessus. Toute utilisation non explicitement autorisée ci-avant (telle que par exemple, la modification du document ou son résumé) nécessite l'autorisation préalable et expresse des auteurs ou de leurs ayants droit.

Analysis and conception of a solution to synchronise Jira instances

Alexandre Bourdouxhe

alexandre.bourdouxhe@student.uliege.be

Friday 9th June, 2023

A Master Thesis submitted to the Faculty of Engineering
of the University of Liège, for the degree of Master in Electrical Engineering

Supervisor: Prof. Pascal Fontaine

Jury members

Prof. **Pascal Fontaine** (Academic supervisor) - Université de Liège
Prof. **Bertrand Cornélusse** - Université de Liège
Prof. **Christophe Debruyne** - Université de Liège
Thibault Clavier (Industrial supervisor) - DSU Director of ARHS

Acknowledgements

I want to thank Professor Pascal Fontaine for his patience with me, his help with the writing of the report and the mental support he provided me during this difficult period. This Master Thesis could not see the day without him and I cannot thank him enough for everything he did.

I also want to thank my industrial supervisor Thibault Clavier who proposed this project, supervised the progresses and provided the infrastructure (test Jira instances, SCRUM Mastering,...) to facilitate the realisation of the project.

Finally, my gratitude goes to Roman Heinz, who provided me with his technical support and help for testing of the plugin, while fulfilling the role of a Scrum Master during the Thesis.

To everyone, I say: thank you again.

Contents

Introduction 3

1 Goal and architecture of the project 7

1.1 Goal of the project . . . 8
1.2 Architecture discussions . . . 9
1.2.1 Web Browser Plugin . . . 9
1.2.2 Web application . . . 10
1.3 Selected architecture . . . 11
1.4 The coding language . . . 12
1.5 In brief . . . 13

2 Configuration of the Replicator 15

2.1 Analysis of the configuration screen . . . 16
2.1.1 Business requirements . . . 16
2.1.2 Analysis of the requirements . . . 16
2.2 Configuration page architecture . . . 19
2.2.1 General structure of Chrome extensions . . . 19
2.2.2 Libraries and tools . . . 20
2.2.3 Storage of the configuration . . . 21
2.3 Configuration page implementation . . . 21
2.3.1 Settings . . . 21
2.3.2 Base configuration . . . 22
2.3.3 Ticket types mapping . . . 23
2.3.4 Custom fields configuration . . . 24

2.3.5 Action bar 24
2.4 In brief 25

3 Cloning of a single ticket (Replicator 1.0) 27
3.1 Analysis of the single ticket cloning 28
3.1.1 Business requirements 28
3.1.2 Analysis of the requirements 28
3.2 Ticket cloning architecture 31
3.2.1 The service worker 31
3.2.2 The content script 31
3.3 Ticket cloning implementation 32
3.3.1 Conditions for the display 32
3.3.2 The “Clone” and “View” buttons 33
3.3.3 The error message modal 35
3.4 Limitations of single ticket cloning 35
3.5 In brief 36

4 Cloning of tickets in bulk (Replicator 2.0) 39
4.1 Analysis of the tickets bulk cloning 40
4.1.1 Business requirements 40
4.1.2 Analysis of the requirements 40
4.2 Tickets bulk cloning architecture 42
4.2.1 Differentiating between single ticket cloning and bulk cloning 43
4.2.2 Retrieve the JQL query 43
4.2.3 The Jira tickets API 44
4.3 Tickets bulk cloning implementation 45
4.3.1 The bulk clone button 45
4.3.2 The bulk modal 47
4.4 In brief 50

5 Cloud support (Replicator 2.1) 51
5.1 Cloud support architecture 52
5.1.1 Differences between the Jira Server and the Cloud instance 52
5.1.2 Inheritance to solve the compatibility issues 53
5.1.3 How to differentiate a server instance from a cloud one . 54
5.2 Cloud instance integration 54
5.2.1 Single ticket cloning 54
5.2.2 Bulk tickets cloning 55
5.3 In brief 57

Conclusion 59

Bibliography 65

List of Figures 67

List of Tables 69

Introduction

In the beginnings of the IT industry, the majority of the software projects were performed following the Waterfall methodology [16]. The Waterfall methodology was splitting the development of the project between five sequential steps [11]:

- Requirements: the requirements of the project are analysed with the customer and put down in a Product Requirements Document (PRD);
- Design: different solutions are proposed by the software engineers to meet the requirements of the customer;
- Implementation: once the design is decided, the implementation of the project starts. Since all analyses and design questions have already been answered previously, this makes this step relatively short;
- Verification/testing: the implementation of the software is validated by the development team via test scenarios created from the PRD;
- Deployment and maintenance: the product is released to the customer, who will be able to use it directly. Bugs are handled by the development team, while any change of functionality based on users' feedback must follow the same process described above.

Those five steps must necessarily be executed in that order, and each step must be completed before the next step can be started. Because of these limitations, an important number of issues can occur during the realisation of a Waterfall project [7]: the customer does not have any preview of the software before it is released, additional features require a lot of processing before they can be added, customer requirements might not have been correctly understood,... These limitations are the reason why new methodologies giving more emphasis on the interaction with the customer were imagined.

Nowadays, most IT companies follow the Agile methodology or one of its derivative [15]. This methodology emphasises the interactions with the client by performing regular reviews of the developed features. These reviews often lead to modifications requested by the client, so as to better fit their needs and their expectations. With this methodology, the project will follow multiple, small-sized iterations in which: requirements are analysed, solutions are designed, implementation is performed and feature is reviewed by the customer before being corrected if necessary, or released if validated. Furthermore, such iterations can run concurrently: some members of the team can work on the analysis of a given feature, while other members can be testing another feature that was just developed. The project is thus progressively built, following the need and review of the customer, while also being made available early on, as soon as the development is done.

Features are most often requested by the customer through a ticketing system such as JIRA [6], which enables a smooth communication between the customers and the IT companies acting as providers. These tickets allow the customer to assign priorities, describe the feature, discuss a release date with the development team,... Later on in the process, these tickets are further broken down into sub-tasks for the analysis, the development and the testing activities.

The reality comes with challenges: IT companies usually work with fixed price contracts. In this context, a total transparency with the customer is not possible. To cope with this challenge, IT companies generally create two instances of ticketing system: one used by the client to create their tickets, and the other used internally for the detailed tasks and the work assignment. However, having two ticketing systems means that tickets from one side must often be transferred to the other side and vice versa. While this is possible manually, it is often tedious and sometimes complicated to do (for instance, what if the ticket is modified on one side?). The goal of this Master Thesis is thus to develop a tool that is able to automatically perform these information transfers between the two ticketing systems. Many questions can be raised: how to handle the authentication of the user to access the information of the internal ticketing system? What is the best architecture for the tool: is it a plugin? or an app with integrated authentication? Which information should be transferred? How often should it be refreshed? Should any modification on the internal side be copied on the client side? The goal of this thesis will be

to try to address these questions and develop a working tool to perform that task.

Methodology

Since this thesis was realised as a project for the ARHS company, it was decided to follow the SCRUM methodology during the realisation of the project. SCRUM is an Agile methodology that is based on the actors, the Scrum Team, working on small, iterative increments of work called Sprints [14]. Feedback is provided at the end of each Sprint by the stakeholders' representative, the Product Owner, who validates the work done by the development team and discusses corrections that need to be performed. The Scrum process is managed by the Scrum Master, who will ensure the methodology is respected and help the members of the Scrum team to perform their role.

In this context, the Scrum team for this thesis consisted of Thibault Clavier in the role of Product Owner, Roman Heinz in the role of Scrum Master and myself in the role of the Development team, while the stakeholders were the Project Managers of ARHS that would use the replicator.

Limitations of the project

An important limitation of the project will be about the Jira instance owned by the customer. In any situation, whatever the selected architecture, information has to be retrieved from the customer Jira instance. However, the customers sometimes decide to hide their Jira instance behind a firewall, limiting the access to only internal applications that would go through a gateway. In this situation, the project will not work since it would need to redirect its call through the gateway before being able to contact the Jira server. In the rest of the report, we will thus assume the customer Jira instance to be available externally.

Organisation of this Thesis

The organisation of this Thesis Report is as follows :

Chapter 1 will determine the goal that should be fulfilled by the Replicator based on the business requirements. From that goal, different solutions will be compared, the drawbacks and advantages between them will be discussed and an architecture will be decided for the project

Chapter 2 will analyse the Configuration screen, which is the starting point to be able to replicate a ticket from a given Jira instance to another one. This feature is the most critical in that it will be used everywhere in the rest of the project, and thus requires the most analyses to avoid later refactoring of the replicator

Chapter 3 will focus on the main feature of the replicator : the cloning of a ticket from a Jira instance to another. It will provide an analysis of the business requirements and discuss the design chosen but also the limitations that this feature quickly meet.

Chapter 4 will tackle the limitations met with the single cloning during Chapter 3 and provide a solution to the problem.

Chapter 5 will finally discuss the differences between a Jira Server and Cloud instance, and the possible solutions to support Cloud Jira instances without impacting the maintainability of the project.

Chapter 1

Goal and architecture of the project

Contents

1.1 Goal of the project **8**
1.2 Architecture discussions **9**
1.2.1 Web Browser Plugin 9
1.2.2 Web application 10
1.3 Selected architecture **11**
1.4 The coding language **12**
1.5 In brief **13**

In this chapter, the different requirements given by the stakeholders for the Jira replicator will be analysed from a high-level point of view to define the goal of the project. Once the goal is defined, an architecture can be chosen for the project such that the project :

- fulfils the requirements given by the product owner,
- is easily maintainable (on average, a new major version of the Jira software comes out every 3-4 years while a new minor version is released every 1-2 months),
- requires as little resources as possible to be developed (a product owner will always prioritise the cost efficiency of any solution),

This chapter is organised as follows : Section 1.1 presents the goal of the project based on the needs and limitations encountered by the stakeholders. Section 1.2 compares different technologies and designs that could be used for the project, and highlights their advantages/disadvantages. Section 1.3 presents the architecture that was selected and explains the reasons that motivated this choice, while Section 1.4 compares the possible programming languages for the project and explains the choice that was made. The chapter is concluded by a brief summary and presents briefly the perspectives on the topic.

1.1 Goal of the project

For a consultancy company, the majority of its projects have a scope external to the company. This has for consequence that most project managers will have two Jira instances for their projects: one is hosted on the customer side, or is at least transparent to the customer. On that Jira instance, the customer will create tickets that will be handled by the development team. The second instance is a Jira hosted internally on the company side. On this instance, the project manager will provide information related to the management of the project such as how long the team worked on a task, when they worked on it, what was the initial budget,...

While the tasks should be the same on the customer instance than on the internal one, for obvious reasons the information provided on the internal tickets should not be shared with the customer. Furthermore, once a project manager has added information on an internal ticket, it can be assumed that he does not want to synchronise the content of that ticket with the one on the customer side, otherwise any modification to the customer ticket would override the changes he applied on the internal ticket. Cloning should thus be a one-direction (from customer to internal instance), single-time operation (no permanent synchronisation between tickets).

Many project managers work on multiple projects and not only on a single one. Those projects might, or might not have the same customer. It is thus important for the replicator to be able to handle multiple projects that are not necessarily hosted on the same Jira instance at the same time.

When cloning a ticket from a Jira instance to another, information about that ticket needs to be retrieved from the API of the customer Jira instance. Once the information is retrieved, the internal Jira instance can be contacted via its API to create a new ticket with the given information. This means the replicator should be able to retrieve information from an authenticated Jira instance and create new tickets on another authenticated Jira instance. Authentication will thus play an important role here : the two Jira instances are generally not on the same host, and this will highly limit the possibilities of architectures for the project.

1.2 Architecture discussions

Two different architectures could be considered for this project : the first architecture would be the “Web browser plugin”. In such case, the project would be a plugin installed on the browser that would inject content to the existing Jira pages, such as adding a new button for the cloning,...

The second architecture considered is the “Web application”. In that scenario, the project would be a Web application deployed on a server of the company, with its custom set of screens and functionalities depending on the business requirements.

Those two architectures will be analysed in their corresponding section below, and the final decision will be detailed in section 1.3 based on the advantages and drawbacks described.

1.2.1 Web Browser Plugin

Advantages

First of all, plugins are built on the browser functionalities and require less overall setup and development to have a running project.

Furthermore, browser plugins have an easy access to cookies (even cross origin) thanks to the background script running on the browser. This makes authentication automatically performed from the moment the user has logged in to the

corresponding Jira instances during the same session, since the authentication cookies will be provided by default with any request.

Finally, new features can be shortened to the implementation of the code logic itself, which saves the effort of developing a new screen. For instance, the cloning functionality only requires a new button to be injected in the existing ticket screen in terms of UI changes, the majority of the development is thus only the logic performed when clicking that button.

Drawbacks

In browser plugins, functionalities can be quite limited since content is injected in the web page instead of having its own custom-made screens.

Changes in Jira's UI, such as a change of id or class for a container where content is injected, could break the extension. This heavily impacts maintainability since Jira major version upgrades are likely to be accompanied by important UI changes and require adaptations of the code.

Furthermore, a plugin presents a strong dependency on the browser selected. For instance, a Firefox extension will rely on Firefox API and will not be compatible with Google Chrome.

Finally, the plugin would have to run on the users' browser. This means that it would have to be downloaded and installed manually by the user, configuration would be dependent on the user and access for debugging would be heavily restricted since it requires access to the user's computer.

1.2.2 Web application

Advantages

A first advantage of Web applications is that development is not limited : any functionality, any screen, any feature can be developed on a Web application, provided there is enough time for the development.

A Web application would be deployed on a server. This means there is no configuration required for the user to download or install the project. Furthermore, having a shared instance for every user would make it so that debugging

is a lot easier since it would be possible to reproduce the steps of a user, save the logs in a database,...

The last noteworthy advantage for a Web application architecture is its only dependency would be the Jira API. This means maintainability would only be impacted by backward incompatible changes to this API, which are relatively uncommon. Such backward incompatible changes will most often be part of a new major release of the API, which will always be exposed on another URI than previous versions.

Drawbacks

The main drawback of such architecture is that authentication would be extremely complicated because the authentication cookies returned by Jira do not have the same domain origin than the Web application would have. There would thus be the need for a complete authentication protocol in the application that would let the user provide his credentials for the contacted Jira instances to allow retrieving and creating tickets on those instances. Authentication also means increased security is necessary in the app since those provided information would be sensitive

Another drawback is that the development of a complete Web application requires an important work load since every screen, every functionality needs to be created from scratch, even though some frameworks like Angular would slightly ease up the development.

1.3 Selected architecture

For this project, the architecture that was selected was the Web browser plugin.

For the realisation of this project, the development team was limited to a single person. This made it so that the main requirement of the architecture was to have a development as quick as possible, since multiple features were required by the stakeholders.

In addition, the time-frame of the project was limited to a total of 6 months. This further emphasised the need for an architecture that would allow for a quick development of the features.

Finally, the problem of the authentication was considered too risky to solve with a Web application compared to the solution of the Web browser plugin where authentication was "built-in". The goal was to have a running project at the end of the Thesis and this would not necessarily have been possible for a Web application because of the authentication consideration.

The browser choice

Since this project was proposed for the ARHS company, it was decided to build a Chrome Extension, since Google Chrome is the recommended Web browser among the company.

1.4 The coding language

Now that the architecture of the project has been decided, the coding language must be chosen. For a Web browser plugin, the content needs to be written in Javascript. However, there is also the possibility to write the plugin in Typescript. This is due to Typescript being built on top of Javascript and being transcribed into Javascript when building the application. The two languages are analysed and compared on Figure 1.1.

It was decided to use Typescript as the programming language for the plugin. This is mainly due to Typescript making the code considerably more robust than Javascript. Furthermore, every functionality of Javascript is available in Typescript, the only real downside being that debugging is made harder because the browser console generally reports the error from the transcribed Javascript code, which is not easily readable for a human.

Language	Pros	Cons
Javascript	Most popular scripting language A wide variety of libraries available	No compilation validation No notion of typing Less built-in features
Typescript	All the functionalities of Javascript with the added features of Typescript : observables,... More secure development (compilation errors, strong typing,...) Language intended to fix the drawbacks of Javascript	Needs to be converted to Javascript to build the plugin, hence making debugging harder Less libraries than for Javascript (younger language), even though the main ones have been migrated to Javascript

Table 1.1. Advantages and drawbacks of Javascript and Typescript [13]

1.5 In brief

Summary of Chapter 1

- In this chapter, the goal of the project has been defined based on the context of a consultancy company and from the point of view of a project manager.
- Based on the goal of the project, different architectures have been studied. Their advantages have been highlighted, while their disadvantages have been described.
- The final architecture for the project has been selected based on the human resources available for the project, the short time-frame limitation and the risk of the authentication problem in a Web application

- Finally, the question of the language was raised and Typescript was selected due to the help it provides during the development.

Perspectives for Chapter 1

- The architecture that was chosen was that of a Web browser plugin. However, in terms of maintainability, stability, possibilities and user-friendliness, the Web application was actually many times superior. It could have also raised interesting questions, such as how to create an effective authentication system for this two-instances problem, and how to correctly secure those information. It unfortunately required important resources that were not available for this project, but those two questions by themselves might motivate a dedicated future thesis.

Chapter 2

Configuration of the Replicator

Contents

2.1 Analysis of the configuration screen 16
2.1.1 Business requirements 16
2.1.2 Analysis of the requirements 16
2.2 Configuration page architecture 19
2.2.1 General structure of Chrome extensions 19
2.2.2 Libraries and tools 20
2.2.3 Storage of the configuration 21
2.3 Configuration page implementation 21
2.3.1 Settings 21
2.3.2 Base configuration 22
2.3.3 Ticket types mapping 23
2.3.4 Custom fields configuration 24
2.3.5 Action bar 24
2.4 In brief 25

This chapter will introduce the configuration screen of the replicator. This configuration screen is the starting point of the project as the configuration provides the necessary information, such as the target Jira URL or the ticket types mapping, to perform the different functionalities.

The requirements of the configuration screen will be described and analysed in section 2.1. In Section 2.2, the general structure of a Chrome extension will be explained and the architecture of the replicator will be discussed. Section 2.3 details the final implementation of the configuration page and its features. Finally, the chapter will be summarised briefly in Section 2.4.

2.1 Analysis of the configuration screen

2.1.1 Business requirements

A list of requirements has been established by the Product Owner based on the needs of the stakeholders. For the configuration screen, those requirements are the following :

- A screen for setting up configuration
- The possibility to export and import the configuration
- An automated configuration vaidation
- The opportunity to provide multiple source and target URLs, and multiple source and target projects
- A configuration screen for setting up which target Issue Type a source Issue Type should be mapped to

2.1.2 Analysis of the requirements

Once the requirements are provided by the Product Owner, the analysis phase can begin. The goal of this phase is to determine the content of the configuration (i.e. which information is required in the configuration), find a functional solution to each requirement and provide a sketch of the configuration screen.

Content of the configuration

First of all, the user needs to provide the Jira instance she wants to clone from, and the Jira instance she want to clone to. A Jira instance is a Jira Application hosted on a domain chosen by the owner of the instance. Since the same application is being run, except on a different domain, it means that two Jira instances can be differentiated simply based on their domain name. This is the first information required from the user : the source and the target domain name.

To create a Jira ticket in a Jira instance, a user needs to provide some information. First of all, a project needs to be selected to create the ticket in.

That project will be part of the identification key of the ticket. This means that when cloning a ticket from one instance to another, two information are required : the source and the target projects.

The second information necessary for a ticket creation is the issue type. By default, Jira comes already with a set of issue types.

Since the source ticket will also have an issue type, it could be imagined that the issue type could be copied during the cloning operation. Unfortunately, it is possible to create custom issue types in Jira. This means that an issue type on an instance could not be defined on another. Furthermore, some project managers might want to change the issue type during the cloning. For instance, a project manager might want to convert bugs into tasks since it does not really make sense to keep bugs in the internal Jira. The user should thus have the possibility to configure the type mapping. In the case no mapping is provided by the user however, the replicator should attempt to copy the type since in most situations, people are using the default types provided by Jira.

The last mandatory information required by Jira is the summary. For this field, the summary of the source ticket will simply be copied since there is no other place to take that value from. This is not an issue as most of the time, a project manager will keep the name of the original task for more clarity.

There is however a last information that is required from the user, due to a requirement of the cloning task: as part of the cloning task, users want to have a link to the source ticket on the cloned one. For this, a small introduction to custom fields in Jira is required.

Jira custom fields

The Jira administrator of a Jira instance has the possibility to configure additional fields for the tickets of his instance, called "custom fields" [2]. When creating such a field, the user is proposed to select a title for her field, as well as a few other options. Once she creates the field, Jira will generate a key for that field to identify it, based on the following pattern : "customField_XXXXX", where XXXXX is a number generated by Jira. This is this key that must be used to set the value of that field when creating a new ticket via the Jira API.

The requirement of the stakeholders for the ticket creation is to have a link to the original ticket in the cloned ticket (see Section 3.1). Furthermore, as the analysis of this Chapter will prove, this link should be in its own field as it will be used by the replicator to know if a source ticket is already cloned by searching for a ticket in the target Jira pointing towards the source ticket. This link should thus be placed into a custom field created by the Jira administrator. This custom field is mandatory to be able to use the Replicator and it will be called ExternalReference.

Since the key generated by Jira for the ExternalReference field depends on the Jira instance, the user needs to be able to configure that value in the configuration screen. For this reason, a "custom fields configuration" section should be also present with one[1] field to set the value of the ExternalReferences.

Multiple source and target URLs

Most project managers are responsible for more than one project at a time. It can be multiple projects for the same customer, but it can also be managing projects for different customers. For this reason, it was asked to support multiple source and target URLs/projects as part of the requirements.

Unfortunately, the configuration of a project can widely differ from one Jira instance to another, and even from one project to another. It is thus necessary to have a single source/target URLs and projects per configuration.

A solution could thus be to allow the user to create multiple configurations, but this raises the question of the identification of the configuration: how can the replicator know which configuration to use for a Jira instance if there are multiple of them with the same source URL? For these reasons, a same configuration cannot support multiple source and target URLs/projects.

The requirement was thus re-discussed and finally, it was decided to support multiple configurations, with an additional subtlety : an additional check-

[1] In the context of ARHS, an additional custom field was added to the configuration: the EpicName. This is a custom field made mandatory for Epic tickets in the ARHS Jira, there was thus the necessity to add this field in the configuration screen. Ideally, it should have been possible to add/remove new custom fields in the configuration instead of hard-coding this field which only makes sense for ARHS. This was not possible due to time constraints but should be considered a future improvement

box “Active” is present on the configuration. This checkbox determines if the configuration is used by the replicator or not. Furthermore, a validation was added to this checkbox : if there exists already an active configuration with the same source URL, the checkbox cannot be checked for the selected configuration.

2.2 Configuration page architecture

Chapter 1 analysed the different architectures possible for the Jira replicator application and the conclusion was to develop a “Chrome extension”. This section will now introduce the general structure of a Chrome extension and explain how the Configuration page is linked to it. The libraries used for the development will also be discussed and the question of the storage of the configuration will be tackled.

2.2.1 General structure of Chrome extensions

Chrome extensions are mainly comprised of the following elements: the JSON manifest, the options, the service worker and the content script [9].

The JSON manifest

The JSON manifest is a JSON file that defines the configuration of the chrome extension. It is located at the root of the project, and provides meta information to the browser such as the name and description of the extension, the mapping with the different scripts and HTML pages, the permissions of the extension,... This file is mandatory to any chrome extension.

The options

The options page is the part of the extension that provides the user ways to customise the extension. This page can be accessed via the details of the extension, from the chrome Extensions menu.

In the context of the Jira Replicator, the options will be the configuration screen where the users can configure the cloning operations the replicator will perform.

The options are comprised of three files: a Typescript file where the logic of the features will be executed, an HTML file containing the code for the UI and a CSS file for any style customisation that could not be provided by Bootstrap. Furthermore, the options will have some interaction with the service worker for the validation of the configuration (more on this in Chapter 3).

The service worker

The service worker of an extension is a Javascript file that is run in the background of the browser. Since the script is running in the background of the browser, instead of running in the content of the Web Page, it is accessible everywhere by the plugin and can use any cookie of the browser. It is thus often used as an event Manager responsible for listening to the extension's scripts events, executing Web requests,...

The details of the implementation of the service worker will be provided in Chapter 3 - Section 3.2.

The content scripts

The content scripts are scripts run into a Web page (called the host). The content script has access to the DOM (Document Object Model) of the host and is thus able to modify it, for instance by adding or removing elements to the DOM tree. The content script in this context will be the script run in the Jira instances' pages to add the functionalities of the replicator (see Chapter 3 - Section 3.2 for more details).

2.2.2 Libraries and tools

The configuration screen was the only screen to be created entirely. Creating a new screen implies many considerations, such as which colour to use for the buttons, how to structure the sections,... Those questions can require a lot of analysis and work, which would be an issue for completing the rest of

the project. To ease-up the development but still provide an elegant solution, it was decided to use the Bootstrap 5 CSS library for the style of the page. Bootstrap comes with a set of HTML classes to style common elements such as buttons, icons, drop-downs etc [8]. Using Bootstrap allowed to focus on the functionalities themselves instead of spending time on defining the style of the HTML elements.

2.2.3 Storage of the configuration

It is evident that a user should only provide her configuration to the replicator once she first uses it. This means the configuration should be saved somehow and fetched back every time a user opens her browser.

The first investigated solution was to use the local storage of the browser to save the content. However, an important issue quickly appeared: the local storage of a browser is not available in the content scripts. This means that the configuration could not be retrieved for ticket cloning operations.

For this reason, it was decided to use the Chrome storage [10]. This storage is not only available in the content scripts, but also automatically performs serialisation of saved data. This allowed to simplify the save and fetch operations since the serialisation was done by Chrome itself.

2.3 Configuration page implementation

The configuration page was implemented in the form of five different sections allowing the user manage and edit his configuration. They can be seen on Image 2.1. A breakdown of each section and its functionalities is provided below.

2.3.1 Settings

The settings section (see the left column of Figure 2.1) focuses on the management of the different configurations of the user. It contains two elements : the first is a drop-down with the list of the user's configurations, showing in focus the selected configuration. Clicking on the drop-down opens a menu

Figure 2.1. A screenshot of the configuration screen of the Jira Replicator plugin

listing the different configurations, which the user can choose from, as well as an option to create a new one.

The second element is the "Import config" button. After clicking on the button, a file selection window opens and let the user select a JSON file from his computer to import. If the imported configuration has the same name as an existing configuration of the user, the configuration is updated with the imported parameters. Otherwise, a new configuration is created from it.

2.3.2 Base configuration

This section allows the user to provide the following information for his currently selected configuration: the config name, the source URI[2], the target URI, the source project key, the target project key and the active checkbox. The section can be seen on Figure 2.2.

[2]In this case, it is question of URI instead of URL. This is because the value provided not only serves to identify the URL of the Web page, but it is also used to build the URI of the Jira instance API.

Base configuration

Config name	HEINZRO -> AJT
Source uri	http://10.2.6.1:8080
Target uri	https://helpdesk2.arhs-developments.com
Source project key	HEINZRO
Target project key	AJT
Active	✓

Figure 2.2. A screenshot of the base configuration section of the configuration screen

2.3.3 Ticket types mapping

The ticket type mapping section handles the mapping between a source ticket type and a target ticket type. The section is configurable, meaning the user can dynamically add and remove mappings with the help of the side buttons (see Figure 2.3). If no configuration is provided, the replicator will try to keep the ticket type by default.

Ticket types mapping

Source type	Task	Target type	Task
Source type	Bug	Target type	Bug
Source type	Change request	Target type	Change request

Figure 2.3. A screenshot of the ticket types mapping section of the configuration screen

2.3.4 Custom fields configuration

This part of the screen handles the logic related to the custom fields: it allows the user to set the key of the ExternalReference field on the target Jira instance, and of the EpicName on both source and target instances. Note that concerning the EpicName, if no mapping is provided, the value of the source ticket summary is used. A screenshot of that section is provided in Figure 2.4.

Custom fields configuration

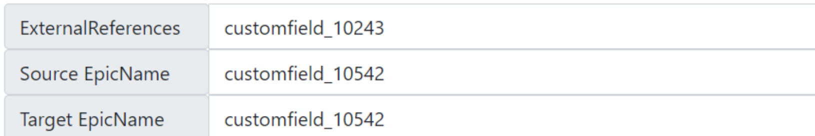

Figure 2.4. A screenshot of the custom fields configuration section of the configuration screen

2.3.5 Action bar

The action bar is an array of buttons located at the bottom of the configuration page (see Figure 2.5). It presents 4 different buttons to the user.

The "Save config" button allows the user to save the current configuration, updating its value in the storage if it was already present, or saving a new element if the configuration did not exist yet.

The "Export config" button exports the current configuration into a JSON file that the user can save on his computer. Together with the "Import config" button, they represent the requirement to export/import configurations from the stakeholders.

The "Test configuration" button allows the user to validate[3] the current configuration by contacting the two Jira instances and validating the URL, the project key and ensure the ExternalReference field correctly exists on the target Jira instance. A modal reprising the details of the validation and the potential errors which occurred is displayed to the user.

Finally, the "Delete config" button allows the user to delete the current configuration.

Figure 2.5. A screenshot of the actions bar of the configuration screen

2.4 In brief

Summary of Chapter 2

- In this chapter, the requirements for the configuration screen were analysed. The content of the configuration was decided based on the Jira conditions for creating a ticket, the needs of the stakeholders and the upcoming needs of the ticket cloning task.
- The requirement of supporting multiple source/target URLs and Projects was re-evaluated due to the technical constraints that would be encountered, and an alternative solution was proposed, with more limitations but easier realisation.
- The general structure of a Chrome extension was explained and the link between that structure and the configuration screen was done. Ques-

[3] Validating the configuration actually requires to contact the Jira API and compare the information with those provided. Since Web requests needs to be performed in the extension service worker, the task was actually performed after the implementation of the ticket cloning feature during which the service worker was implemented (see Chapter 3 - Section 3.2 for more details on the service worker.)

tions such as the storage choice and the use of libraries were debated and solved.

- Finally, the implementation of the Configuration page was shortly described and followed by a complete breakdown of its structure and the features it provides to the user.

Chapter 3

Cloning of a single ticket (Replicator 1.0)

Contents

3.1 Analysis of the single ticket cloning 28

3.1.1 Business requirements 28

3.1.2 Analysis of the requirements 28

3.2 Ticket cloning architecture 31

3.2.1 The service worker 31

3.2.2 The content script 31

3.3 Ticket cloning implementation 32

3.3.1 Conditions for the display 32

3.3.2 The “Clone” and “View” buttons 33

3.3.3 The error message modal 35

3.4 Limitations of single ticket cloning 35

3.5 In brief 36

Once a user has setup an active configuration, she is ready to clone tickets between the corresponding Jira instances. This chapter will look at the ticket cloning feature itself: the requirements of the stakeholders will once again be analysed and cleared-out in Section 3.1. Section 3.2 will discuss the role of the service worker and content script more in details, while the implementation of the feature will be discussed in Section 3.3. After some testing, limitations of the feature will quickly appear and will be described in Section 3.4. Finally, the chapter will be briefly summarised in Section 3.5.

3.1 Analysis of the single ticket cloning

3.1.1 Business requirements

The requirements from the stakeholders for the cloning of a ticket are as follow:

- A "Clone" button should be added on the source Jira ticket details view. The clone button should only appear on pages where an active configuration was found in the plugin.
- A link to the original Jira ticket should be displayed on the cloned ticket.
- Performing a cloning operation should display respectively a success/error message.
- The "Clone" button should become a link to the cloned ticket on target Jira when the ticket is already cloned.

3.1.2 Analysis of the requirements

Once again, the requirements of the stakeholders need to be analysed. The goal of this phase is to: determine the conditions for the activation of the plugin, decide the location of the "Clone" button and discuss the edge-case of one of the requirements of the stakeholders.

Prevent interactions with unconcerned Web pages

Jira instances are Jira applications hosted on a Web domain provided by the owner of the instance. For this reason, the Web domain that the plugin has to interact with is totally unknown and will depend on the Jira instances the user configured.

It was thus necessary to configure the plugin to have access to any Web page, by setting a wildcard in the permissions section of the extension's JSON manifest.

This means that by default, the plugin will attempt to inject content in any Web page the user navigates to. To prevent unwanted interactions of the plugin with such Web pages, it is necessary to ensure the current Web page is a

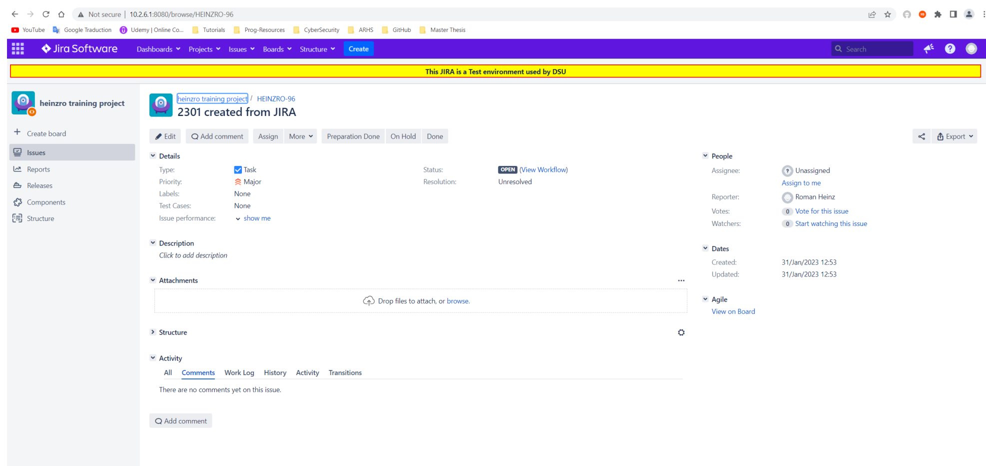

Figure 3.1. A screenshot of the ticket details view in the Jira application

configured Jira instance. For this, the active configurations should be checked and if one is found to match the current domain of the Web page, then the plugin should be allowed to proceed with its logic. Otherwise, nothing should be done.

Location of the “Clone” button

Web browser plugins work by injecting content directly into the DOM (Document Object Model) tree of a Web page. The replicator should thus inject an additional button on the source Jira instance’s ticket details view. Now come the question of the location : where should the button be added?

The Jira ticket view can be seen on Figure 3.1. It can be observed that an array of button is present on the head section of the view. This makes it the ideal place to inject the button since all the other existing buttons are actually used for the edition of the ticket, it thus makes sense to place the “Clone” next to them.

Multiple clones of the same Jira ticket are found

One of the requirements of the stakeholders is that the “Clone” button on a source Jira ticket should become a link to the cloned instance if there is one. However, an edge-case can appear in this situation: what if multiple tickets on destination Jira have a reference to the same source ticket? While technically this situation should not occur via the replicator, it sometimes happens that a user copy a ticket in Jira to speed-up the creation of tickets (for instance, they need a ticket with the same Project Id or the same Epic issue). In this case, the user might forget to remove the ExternalReference field, which would thus result in having more than one ticket linked to the same source.

This case was discussed with the product owner and it was proposed that the “Clone” button becomes a link to the Jira search page, with a search query looking for the ExternalReference field linking to the source ticket. This will allow to user to see the list of tickets with that same ExternalReference and she will easily be able to fix the issue to keep only one cloned ticket. The link button on the Jira source should also display a warning icon.

Display of the error/success messages

It was initially considered to use Windows notifications to show the error messages to the user. However, the idea was later dropped when the question of multiple screens was raised: Windows notifications are sent on the main screen of the user, which is generally the laptop one. However, users often connect additional screens and are most often working on these since they are bigger than the laptop screen. Using Windows notifications would have made it so that the notifications would have appeared on the laptop screen and would have been missed most of the time by the users.

It was thus decided to inject a modal to the DOM tree of the Web page. This modal would then be shown when an error was encountered by the user during the cloning of his ticket. As for success messages, they were finally dropped since they would just slow down the flow for the user since the button already changes when the source ticket is correctly cloned.

3.2 Ticket cloning architecture

Chapter 2 introduced the general structure of a Chrome extension and explained how the configuration screen was configured as the extension's options. In this section, the service worker and the content script will be detailed as they will be responsible for the logic of the ticket cloning feature.

3.2.1 The service worker

As already explained previously, the service worker is a script running in the background of the browser. As such, it has access to all the cookies of the browser, no matter their domain origin. This allows this script to execute Web requests while joining the authentication cookies of the Jira instances the user has logged into.

The service worker will thus serve as a service responsible for performing the Web requests to the Jira instances' API. It will listen to events emitted by the content script: for instance, when the user clicks on the "Clone" button, it will execute the Web requests[1] to retrieve the information of the source ticket from the source Jira instance and will handle the creation of the cloned tickets by emitting POST requests to the target Jira instance.

3.2.2 The content script

The content script is a script running directly in the Web page the user consults. It has directly access to the DOM tree of that page and is able to modify the content of the tree via Javascript commands.

In the context of the replicator, the content script is a Typescript file that will handle the injection of the "Clone" button as part of the ticket cloning feature. It will hold the logic for the display and hiding of the button, the position it should be injected in, it will extract the required information from the user configuration and from the responses of the service worker and finally, it will trigger the logic of the service worker by emitting events listened to.

[1] The Jira API being a REST API, the Web requests will thus be REST calls

It is interesting to note that the content script holds both the UI logic, such as the HTML code for the "Clone" button, and the logic related to the background such as extracting the information from the configuration to build the URI of the Jira API or the request body that is passed along. This will be taken advantage of later on in Chapter 5 where an important refactoring will be required to support the Jira Cloud environment.

3.3 Ticket cloning implementation

The ticket cloning feature is comprised of two main elements : The "Clone" and "View" buttons, which are the buttons respectively used to clone a ticket and redirect towards the clone when one has been found, and the error message modal, which displays error messages to the user.

3.3.1 Conditions for the display

The first condition to inject the ticket cloning feature is that the user has to be consulting a Web page whose domain was configured as a source Jira instance via an active configuration. If no active configuration is found for the page, the plugin will not inject anything.

The second condition is for the current view to be the ticket details view of a Jira ticket. If the user navigates to another page, for instance the tickets filter view of a project, the button will not be injected. This is to prevent situations where the button would not be added at the right place or would even mess with the display of pages it is not intended to be injected into.

Once the user visits a ticket details page, a first button is injected. This button is the "Getting status" button (see Figure 3.2). It actually is a button without any logic performed when clicking on it. This button is shown on the time the replicator is trying to contact the target Jira instance to search for a clone of currently viewed ticket. It is used to inform the user that the plugin is currently retrieving the status of the ticket and he has to wait for the response.

Once the response is correctly received from the target Jira, the button becomes either a "Clone" button is not clone was found on target Jira ; otherwise

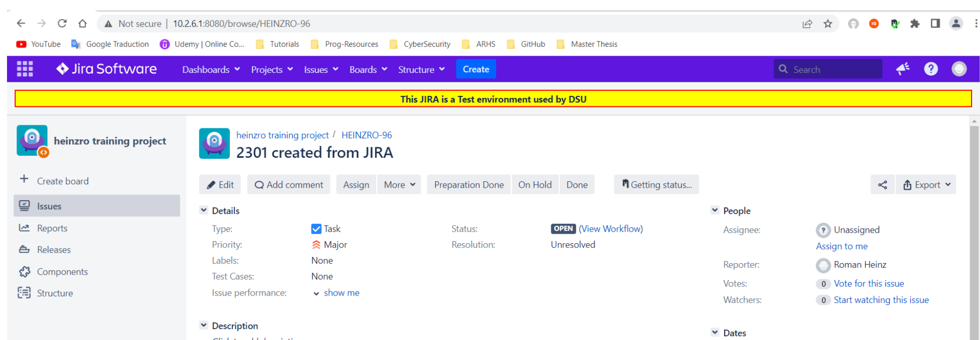

Figure 3.2. A screenshot of the "Getting status" button injected in the ticket details view

it becomes a "View XXXXX" button, where XXXXX is the key of the cloned ticket on the target Jira instance. If an error is received instead, the button remains the "Getting status" button and an error message is displayed to the user with the error which occurred.

3.3.2 The "Clone" and "View" buttons

The "Clone" button

The "Clone" button is injected in the page when no clone of the current ticket was found on target Jira. When the user clicks on the button, the replicator will create a new ticket on the target Jira with the information of the source ticket, except for the ticket type which is mapped according to the configuration (see Chapter 2 for more information about this mapping).

If the cloning operation is a success, the "Clone" button is converted to a "View" button that links towards the newly cloned ticket.

The "View" button

The "View" button is a button injected in the details view when a cloned ticket has been found for the currently consulted ticket. This button can be

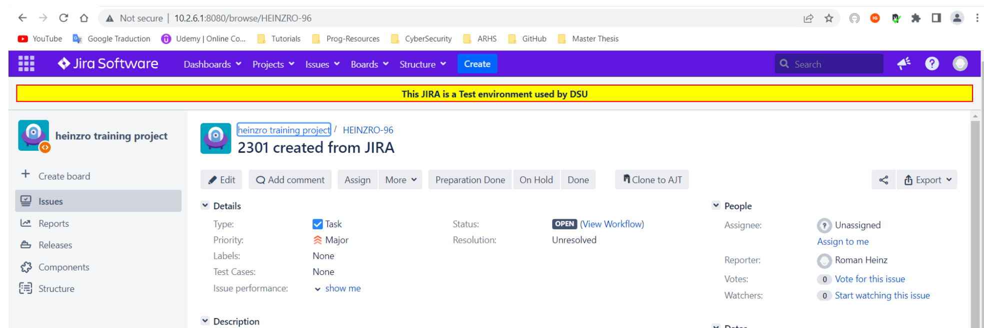

Figure 3.3. A screenshot of the "Clone" button injected in the ticket details view

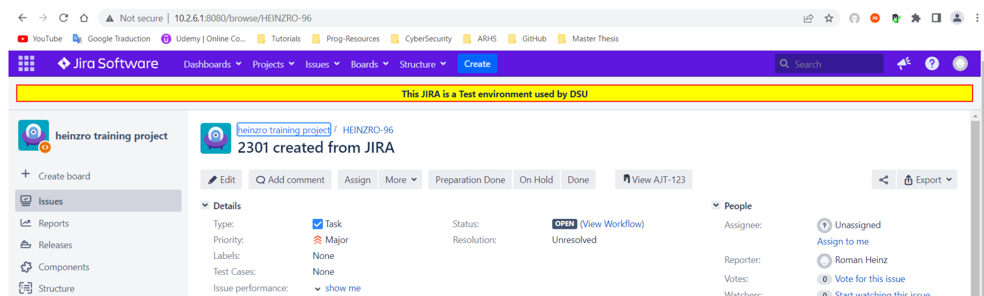

Figure 3.4. A screenshot of the "View" button injected in the ticket details view

seen on Figure 3.4. Clicking on the button redirects the user to the cloned ticket details view in a new browser tab.

The edge-case of multiple cloned tickets

As analysed previously in Section 3.1, in case multiple cloned tickets are found, the "View" button instead becomes a "2+ cloned issues" button which links towards the search page of target Jira, with a search query looking for tickets with an ExternalReference linking to source ticket. This button can be seen on Figure 3.5.

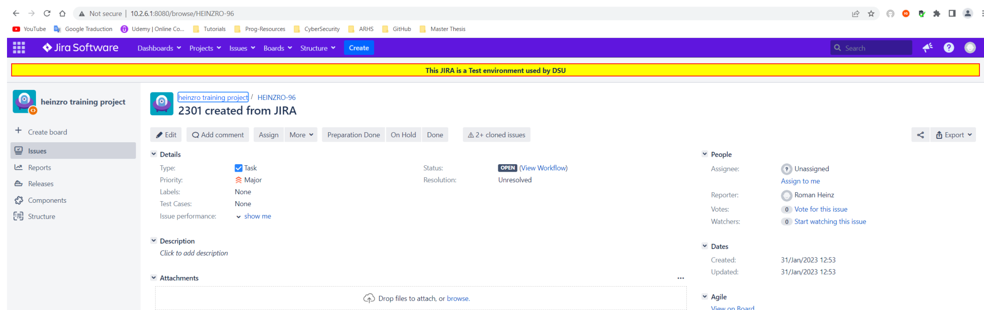

Figure 3.5. A screenshot of the “2+ cloned issues” button injected in the ticket details view

3.3.3 The error message modal

In case an issue occurs with the replicator, the user needs to be informed of the error. For this reason, a modal is injected in the DOM tree of the ticket details page (see Figure 3.6). This modal is hidden by default and is only displayed once one of the Web requests emitted by the service worker returns an error.

In practice, the most common situation is when a user forgot to log in to the target Jira instance. In this situation, the replicator is unable to retrieve the information about a potential cloned ticket and receives an error code from the target Jira instance. In that case, the error modal opens to show the error message received to the user.

3.4 Limitations of single ticket cloning

After the implementation of the ticket cloning feature, a testing period was put in place to let some stakeholders test the plugin and provide their feedback about it. The plugin quickly revealed an important limitation : ticket cloning could only be performed ticket by ticket, while most project managers had hundreds of tickets to duplicate on the internal Jira of the company.

While the cloning operation in itself was relatively quick, the real issue was that users had to switch between the details view of each ticket they wanted to

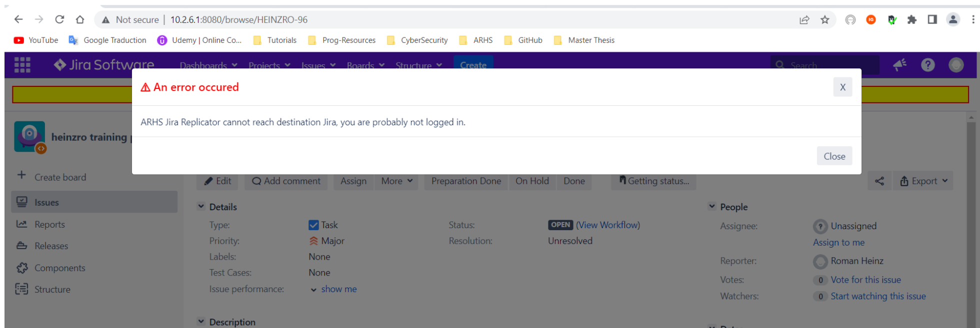

Figure 3.6. A screenshot of the error modal injected in the ticket details view when an error occurred in the ticket cloning flow

clone. This made the plugin tedious to use as the loading time when changing the view was relatively important. The stakeholders thus asked for a way to duplicate tickets in bulk.

3.5 In brief

Summary of Chapter 3

- In this chapter, the requirements for the ticket cloning feature were analysed. The conditions of activation of the plugin were presented, the UI details were cleared out, the edge-cases were resolved and the requirement of the error/success message during cloning was redefined after a discussion with the product owner.
- The architecture of the feature was presented. More specifically, the role of the service worker and of the content script was defined with regard of the ticket cloning feature, and their structure was provided.
- The main elements of the feature implementation were discussed: the "Clone" and "View" buttons and their interactions with each other were described and the error message modal was introduced to provide the user error messages when the cloning of a ticket failed or when information cannot be obtained from the target Jira instance.

- Finally, the plugin was tested by the stakeholders to gather feedback. It quickly appeared that while the plugin was working as intended, it was tedious to use when the user had multiple tickets to clone, especially for bigger projects where some project managers had hundreds of tickets to duplicate. The stakeholders thus asked for the development of a new feature that would allow to duplicate tickets in bulk.

Chapter 4

Cloning of tickets in bulk (Replicator 2.0)

Contents

4.1 **Analysis of the tickets bulk cloning** **40**
4.1.1 Business requirements 40
4.1.2 Analysis of the requirements 40
4.2 **Tickets bulk cloning architecture** **42**
4.2.1 Differentiating between single ticket cloning and bulk cloning 43
4.2.2 Retrieve the JQL query 43
4.2.3 The Jira tickets API 44
4.3 **Tickets bulk cloning implementation** **45**
4.3.1 The bulk clone button 45
4.3.2 The bulk modal 47
4.4 **In brief** **50**

In the previous chapter, the ticket cloning feature was implemented and all the requirements originally requested by the stakeholders were finally done. A testing period by the stakeholders took place and it became clear that the replicator was tedious to use when the user had multiple tickets to clone. For this reason, a new requirement was created: the possibility to perform cloning of tickets in bulk.

This chapter will focus on the realisation of the tickets bulk cloning. The feature will be analysed in Section 4.1 to understand how to integrate the cloning of tickets in bulk with Jira search page. Section 4.2 will discuss the impacts

of the feature on the architecture of the content script file. In Section 4.3, the implementation of the feature will be discussed and the different features provided to the user will be described. Finally, the chapter will be summarised in Section 4.4.

4.1 Analysis of the tickets bulk cloning

4.1.1 Business requirements

Based on the feedback of the stakeholders from their testing of the plugin, and following discussions with the product owner about the feature, the following list of requirements for the cloning of tickets in bulk was established:

- The user should be able to use JQL (Jira Query Language) to filter the list of tickets to clone
- The list of tickets retrieved should appear in a new window with basic information about the ticket (ticket key, summary, version,...).
- The user should be able to clone all the tickets displayed with a single action.
- The user should be able to select a subset of the tickets to only clone the selected ones.
- A status icon should appear after the cloning operation to indicate whether the cloning of the corresponding ticket succeeded or failed.
- In case one or multiple cloning operation failed, the user should be able to print out a report of the errors that occurred.

4.1.2 Analysis of the requirements

Introduction to the Jira Query Language

The Jira Query Language is a language provided by Jira to perform searching, filtering and sorting on Jira tickets [5]. The user can refine her research by creating conditions on ticket fields through comparison (<, >, =) of their

values, by using keywords (AND, OR, IS) to combine conditions on ticket fields and by sorting the search result with the "ORDER BY" keyword.

The Jira search page

Jira instances provides a page, the search page, where the user can define a JQL query to search among projects' ticket [5]s. The result of the research is displayed in the form of multiple pages of tickets which the user can browse and interact with.

The user can provide her query via two different ways : the "basic" and the "advanced" searches (see respectively Figures 4.1 and 4.2).

The "basic" search provides the user with a small list of drop-downs buttons. Each drop-down corresponds to a different ticket field among the most common ones (summary, ticket key,...). Clicking on the drop-down opens a list of possible values for those fields and let the user search for the value she wants or select it from the list directly. The user can also add additional drop-down for any type of field she wants with an additional "More" drop-down. The value selected for all these fields is then combined by Jira to create a JQL query used for the search.

The user can also choose the "advanced" search mode. In that case, the drop-downs are replaced with a single input field. In this input field, the user can write a JQL query entirely, allowing her to exactly specify the conditions of the research. While the user is writing her query, Jira will also suggest a list of values and keywords based on the words currently written by the user, which helps the user to write her query quicker.

Adding the bulk clone to the Jira search page

For the bulk cloning feature, the user needs to provide a JQL query for the plugin to use. While it would be possible to add a simple input field via the plugin in the Web page, Jira provides some interesting features in the search page for the user to write her query, such as the basic search mode and the auto-completion of the JQL query in the advanced mode. Instead of adding a new input, it is thus more interesting to let the user write her query in the search page and extract this exact query for the bulk search.

This decision has for consequences that the bulk clone feature will only be added in the Jira search page, since it is the only place to provide a JQL search tool.

Integrating the bulk search with the configurations' limitations

As was described in Chapter 2, a single configuration can be active at the same time for a given source Jira instance.

Let us assume that a user wants to clone all the tickets that have the status “open”. She will thus write the following search query in the Jira search bar: status = Open.

She will then see the list of all tickets that are open. However, those tickets could all be from different projects since the only condition was for the status to be ”open“. This would conflict with the plugin configuration, which is configured for a single project at a time (a single active configuration is allowed per source Jira instance). The plugin would thus be unable to clone such a list of tickets.

This means an important limitation must be set for the JQL search: a user has to make sure the result of the query returns tickets belonging to a single project, otherwise an error message must be shown to explain she needs to refine her query to display tickets from a single project. The user can achieve this easily by specifying the project in the JQL search. For instance, the previous query would become : project = XXXX AND status = Open.

4.2 Tickets bulk cloning architecture

The architecture will remain the same for the bulk cloning than for the ticket cloning feature: the logic concerning the UI of the bulk clone feature will be handled by the content script, while the HTTP calls with the JQL query to the Jira API will be handled by the service worker. However, it will be important to separate both features because their flows are totally different: the page where the content is injected is not the same (ticket details vs Jira search page), the information retrieved from the Jira source is different and the APIs contacted need different inputs .

This section will thus discuss the method used to distinguish between singlet ticket cloning and bulk ticket cloning. It will also investigate the new APIs contacted and the information they require.

4.2.1 Differentiating between single ticket cloning and bulk cloning

Single ticket cloning consists in adding a “Clone” and a “View” button in a ticket details page. Clicking on the “Clone” button retrieves information from the Jira source about the current ticket and contacts the target Jira to create a new ticket from those information, while taking into account the configuration provided by the user via the Configuration page.

The bulk ticket cloning feature has a completely different flow: the “Bulk clone” button is added to the Jira search page (see Section 4.1), information retrieved from the Jira source depends on the JQL query written by the user and the tickets returned are listed in a new window as per the stakeholders requirements. This means that the logic of the feature is completely different from the single ticket cloning.

For this reason, it was decided to create two different flows in the content script : one for the single ticket cloning and one for the bulk cloning. These flows are separated at the beginning of the content script based on the URL of the web page: the Jira search page has “jql=” as a query parameter. The URL of the page is thus observed at the start of the content script : if it finds the “jql” query parameter, it goes for the bulk search flow, otherwise it selects the single ticket flow.

4.2.2 Retrieve the JQL query

As discussed in Section 4.1, the bulk cloning feature is injected in the Jira search page since it provides the user different ways to write his JQL query. There is however the question of how to retrieve the query in the plugin?

For the advanced search, the user writes her query in a single input field. This input field is part of an HTML container, the script can thus search the content of the page to look for the HTML container, and retrieve the content of its input field.

For the basic search on the other hand, the user creates her query by providing values to a set of drop-downs, there is not a unique place to write the complete JQL query. Furthermore, the user can dynamically add new drop-downs for the fields of his choice to refine her query. This would make it complicated to search the HTML and find each and every drop-down to build the query.

Fortunately, on the basic search, whenever the user selects a value in a drop-down, the "jql" query parameter in the URL is being instantly updated with the resulting query. The plugin will thus search the URL when the user clicks on the "Bulk clone" button to retrieve the query.

Note that it is not possible to use the same behaviour for the advanced search as for the basic search: in the advanced search mode, the "jql" query parameter of the URL is not updated dynamically by Jira when the user writes the query but is instead updated at the moment the user does a search. This would require the user to first do a search before being able to click on the "Bulk clone" button, which would make the usage of the plugin awkward and tedious for the user.

4.2.3 The Jira tickets API

Once the user has provided her JQL query, the plugin needs to retrieve the tickets from the Jira source API via the POST "/tickets" API [3]. To contact the API, a few parameters have to be provided : the JQL query, but also the tickets' fields that need to be retrieved and the maximum number of tickets to retrieve.

The tickets' fields are the ones the stakeholders asked to be displayed in the new window. They are the key, the summary, the status, the issue type and the fix versions.

Concerning the number of tickets to retrieve, the problem was analysed: the greater the number of tickets, the longer the call takes to retrieve them. Except for the first utilisation of the plugin, for which the stakeholders might have hundreds of tickets to clone since they were accumulated over multiple months, it is extremely unlikely for a user to have hundreds and hundreds of tickets to clone when she uses the plugin. For this reason, it was decided to choose 100 tickets for the maximum number of tickets as it is a good compromise in terms of performances and ease of use. As for the first utilisation of the plugin, a

user can refine her search by specifying the creation date of the tickets in the JQL query she writes and display different sets of tickets.

4.3 Tickets bulk cloning implementation

4.3.1 The bulk clone button

As explained above, the bulk clone button is injected by the Jira plugin on the Jira search page. Two cases are treated: the basic search case and the advanced search case.

Basic search mode

In the basic search mode, the JQL query is generated from the different drop-downs the user can fill in (see Figure 4.1). Every time the user enters a value she wants to filter on in a drop-down, the url is updated with the resulting JQL query.

Once her query is ready, the user can click on the "Bulk Clone" button to perform the search and display the bulk clone modal. Note that using the "Bulk clone" button will not update the Jira search page but only display the bulk clone modal. It is however possible to use the "Search" button if the user wants to see what would be the result of the search, since the same query is used by both Jira and the plugin. This can be useful for instance if the search returned more than one project, so that the user can find what was wrong with her query.

Advanced search mode

In the advanced search mode, the drop-downs have disappeared to be replaced by a single input field (see Figure 4.2). In this input field, the user can manually write her JQL query that she wants to use for the search. The input field provides auto-completion of the values and JQL keywords to help the user in writing her query. In this scenario, the JQL query is taken from the input field instead of the URL of the page.

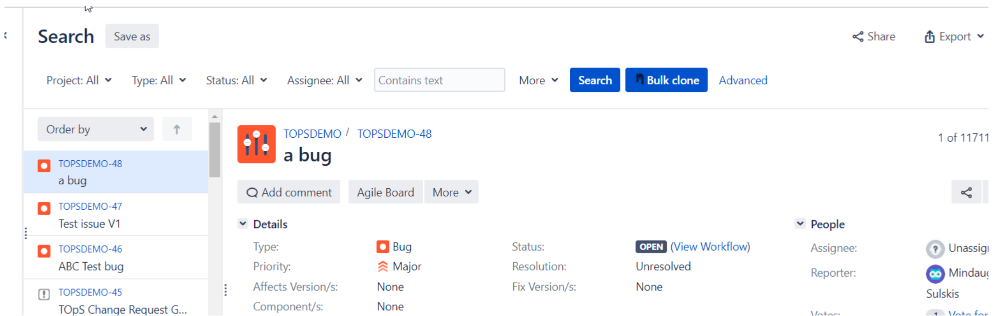

Figure 4.1. A screenshot of the Jira basic search page with the bulk clone button of the plugin

Once the query has been written, the user can click on the "Bulk Clone" button to display the bulk clone modal. The behaviour remains the same as for the basic search in case the query is invalid or if the user wants to see the result of the search in the search page itself.

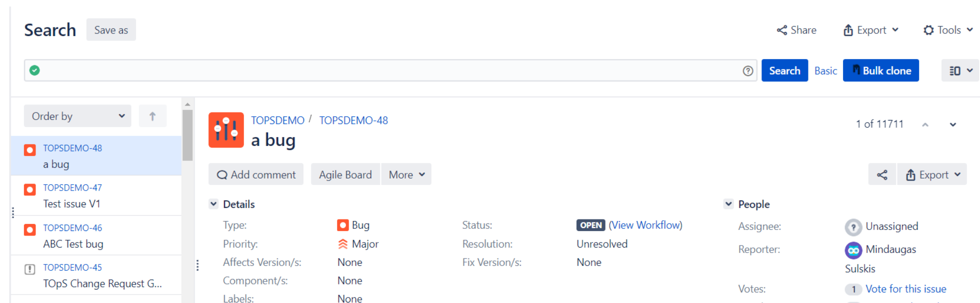

Figure 4.2. A screenshot of the Jira advanced search page with the bulk clone button of the plugin

Invalid search query

As was analysed in Section 4.1, it is not possible to support tickets from multiple projects at the same time in the bulk cloning feature, due to limitations in retrieving the correct configuration. For this reason, once the user clicks on the "Bulk Clone" button, the plugin will retrieve the list of tickets from

the Jira API and then analyse the content: if more than one project is found in those tickets, the bulk modal is not displayed and instead a pop-up will be shown to the user to tell her that the query she used is invalid as it returned more than one project. This pop-up can be see on Figure 4.3.

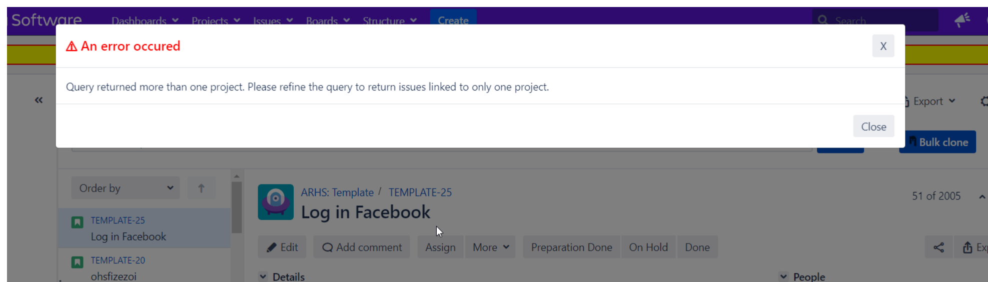

Figure 4.3. A screenshot of the error message displayed to the user when the JQL search resulted in tickets from multiple projects

4.3.2 The bulk modal

Once a user has entered a valid JQL query and clicked on the "Bulk Clone" button, the bulk clone modal is displayed (see Figures 4.4 and 4.5). This modal is comprised of a list of up to 100 tickets (see Section 4.2 for the details on that limit) and an action menu at the bottom.

The tickets table

The first section of the modal is responsible for displaying the list of Jira tickets searched. That list is displayed in the form of a table where different information of the ticket is displayed along them. Note that the number of tickets displayed is shown at the top of the table. If that number has hit the maximal number possible (100), a warning message will be displayed to the user so she is aware she is probably not cloning all the tickets returned by her search.

The first column of the tabular is a checkbox. When the box is checked, the ticket is selected for the bulk cloning operation (see sub-section 4.3.2). The

head of that column also provides a checkbox that allows to select/exclude all tickets in one click. By default, all boxes are checked when the modal is opened.

Columns 2 to 6 are respectively the key, the summary, the status, the issue type and the fix versions of the ticket. This key is the identifier of the ticket in Jira. The summary is a short sentence describing the ticket and displayed at the top of the ticket view, similar to a title. The status indicates the state of the work on the ticket (open, ongoing, to do,...). The issue type relates to the type of task the ticket is related to (bug, story, epic,...). Finally, the fix versions is the expected date for the deployment of the task concerned by the ticket.

Column 7 provides an action button for the ticket: if the ticket is yet to be cloned on the target Jira, the button shown will be "Clone to XXXX", with XXXX being the target project. On the other hand, if the project is already cloned, the button becomes a link to the cloned ticket and will be named "View XXXX', with XXXX the name of the cloned ticket. If more than one cloned ticket has been found by the plugin, it instead displays the warning button "2+ cloned issues" that links towards the search page of the target Jira, with a prebuilt JQL query to search for the cloned tickets.

Finally, the last column shows an icon whenever a clone operation has been attempted on the corresponding ticket. If the operation is still ongoing, the column will show a "Cloning..." message. If the operation succeeded, the icon will become a green check. On the other hand, if the operation failed, the content of the column will be replaced by a red "Failed" message.

The action menu

The action menu is situated at the bottom of the bulk clone modal. It provides three different buttons to the user: the "Export errors", the "Close" and the "Clone selected" buttons.

The "Export errors" button allows the user to generate a text file containing all the errors that occurred during a bulk clone operation. For instance, let us consider a case where the user clones 80 issues and 4 of them failed. When the user clicks on the "Export errors" button. she will download a text file containing 4 different lines, one for each ticket whose cloning operation failed.

Figure 4.4. A screenshot of the bulk clone modal listing in a table the different tickets searched for

Figure 4.5. A screenshot of the action menu of the bulk clone modal

Each line is the error message received by the plugin during the cloning operation. If no error occurred, the file will be empty. Note that only the tickets

of the last cloning operation are concerned. Any new cloning operation will reset the previous errors.

The “Close” button allows the user to close the bulk clone modal.

Finally, the “Clone selected” button allows the user to clone all the tickets whose checkbox has been checked. The cloning is performed sequentially until every selected ticket has been cloned.

4.4 In brief

Summary of Chapter 4

- In this chapter, the requirements for the bulk ticket cloning feature were analysed. The Jira Query Language was presented as the solution to retrieve multiple tickets at once and the page where the bulk flow should be integrated was discussed. The limitations for the bulk search were also analysed based on the configuration design.
- The architecture of the feature was presented: the integration with the existing single ticket cloning feature was determined, the different ways to retrieve the JQL query from the Jira search page were discussed and the content of the REST call to the Jira “tickets” API was discussed.
- Finally, the main elements of the feature were shown: the “Bulk clone” button was presented as well as the condition on the searched tickets’ project field to perform the bulk clone. The bulk modal was introduced, the tickets table was shown and the action menu and its features were discussed.

Chapter 5

Cloud support (Replicator 2.1)

Contents

5.1 **Cloud support architecture** **52**
  5.1.1 Differences between the Jira Server and the Cloud instance 52
  5.1.2 Inheritance to solve the compatibility issues 53
  5.1.3 How to differentiate a server instance from a cloud one 54
5.2 **Cloud instance integration** **54**
  5.2.1 Single ticket cloning 54
  5.2.2 Bulk tickets cloning 55
5.3 **In brief** **57**

The current Jira replicator plugin was designed based on Jira server instances. The reason is that most of the projects the stakeholders are managing were started multiple years ago and Jira servers were either the only Jira software solution at that time, or they were still providing better features to the users than cloud solutions. However, over the recent years, Atlassian (the company owning Jira) started to focus on the cloud instances. Since 2021, they stopped selling server licenses and announced the end of the support for server instances in 2024, while they also provided a service to migrate server instances to cloud ones [1].

Since Jira instances were being totally migrated to the cloud, it was necessary to support cloud instances with the plugin as well. Unfortunately, there were significant differences in the structure of the corresponding HTML pages that were preventing the plugin to work. For instance, the different HTML elements

that were used as placeholders to inject the buttons were not present in the cloud instance, and buttons were thus not injected correctly.

The business requirements being the same between the server instance and the cloud instance, no business analysis was required. On the other hand, the architecture of the plugin had to be completely reworked. This important step is described in Section 5.1. The features being the same, there will be no implementation section. Instead, Section 5.2 will review the integration of the plugin in the UI of the cloud instances. As a conclusion, the chapter will be summarised in Section 5.3.

5.1 Cloud support architecture

5.1.1 Differences between the Jira Server and the Cloud instance

Multiple differences were present between the Jira server and cloud instances. First of all, the Jira plugin injects the different components by locating specific HTML elements of the Web page and injecting additional content to them. For instance, the "Clone" button is injected in the ticket details page by searching for the action bar container and adding an additional button to it.

These operations require to search for the HTML elements based on either their id or their classes. Unfortunately, the Cloud instance of Jira, while keeping a UI similar to the Server instance, has a totally different HTML structure: the classes are different, the ids are different, the HTML elements are different.

This first difference actually introduces a second one as well: the HTML elements that are injected by the plugin (buttons,...) all have CSS classes that are aligned on the design of the page. For instance, the CSS of the button injected in the "ticket details view" is the same as the other buttons of the page to prevent discrepancies in the UI. Since the CSS style of the Cloud Jira instance is different from the server one, this means the elements that are injected by the plugin should also have different CSS classes based on the type of instance that is targeted.

Finally, the last difference observed between the two types of Jira instances is the REST API they use: the REST API of the server instance is a different

API from the one used by the cloud instance [4]. Fortunately, the endpoints used in the context of the Jira replicator exist in both API and require the same types of inputs. The only difference is thus the domain of the API, which will be obtained from the configuration of the user. No change is thus required on this side.

5.1.2 Inheritance to solve the compatibility issues

As can be observed from the previous section, the majority of the differences between the Jira server and cloud instances relate to the UI of the application: the structure of the HTML vastly differs from one type of instance to the other.

A possible solution to this problem could be to code a completely new flow for the Cloud instance. This would mean the different functions used by the plugin would have to be rewritten completely. However such solution would not only require an important work load, but a majority of the code would have to be duplicated (especially the part concerning the REST calls and the logic that is linked to them).

Furthermore, another problem could be raised : what would happen if a new version of the Jira server or cloud instance were to be released, and this version introduced breaking changes in the UI of the application? A solution could be to adapt the corresponding flow to the new UI, but this means that the previous version would not be supported anymore for the users whose Jira instance is not upgraded yet. Another solution could be to write a completely new flow for that version, but this solution is obviously not maintainable in terms of code duplication and work load.

This is why another solution was considered: instead of writing a new flow for the Jira cloud instance, it was decided to extract the common parts in a “UI Service” responsible for the UI logic and a “Background Service” responsible for the REST calls logic. The UI service (as the differences between the server and cloud instances are strictly UI related) is then extended to obtain a “Cloud UI service” and a “Server UI service” that simply override or implement the base service functions where the logic differ between the two instances.

Using this inheritance solution, it is actually extremely easy to support new versions or new types of instances: the only requirement is to create services that extend the base ones and override or implement functions where the logic

is different, and then to use the right implementation based on the flow that is currently considered. This highly reduces the work load required while also providing great maintainability and clean code.

5.1.3 How to differentiate a server instance from a cloud one

Since there will now be multiple extensions of the “UI service”, the plugin needs to be able to identify which one to instantiate when a user opens a Jira instance.

There is unfortunately no endpoint in the API of neither the server nor the cloud instance that is able to provide this information. The only way to confidently differentiate a Jira server instance from a cloud instance is to actually search the url of the page: if the url contains “.atlassian.net” somewhere, this means the instance is a cloud one. If it does not, it is thus a server one. The reason for this is that Cloud instances do not support custom web domains for the time being and are then all hosted on Atlassian servers [12].

5.2 Cloud instance integration

This section will discuss the integration of the plugin with the cloud instance. Both the flows of the single ticket cloning and the bulk tickets cloning will be considered, and their integration respectively with the “ticket details view” and the “search page” of the Jira cloud instance will be presented. However, since the functionalities remain the same as for the server instance, only the HTML integration will be considered. Functionalities are already described in Chapters 3 and 4 respectively for the single ticket cloning and bulk tickets cloning.

5.2.1 Single ticket cloning

Similarly to the Jira server instance, the single ticket cloning operation is injected on the “ticket details page” of the Jira application. The integration can be seen on Figure 5.1: the Jira instance is a cloud one (this can be deducted from the URL of the page whose domain is “.atlassian.net” and from the design

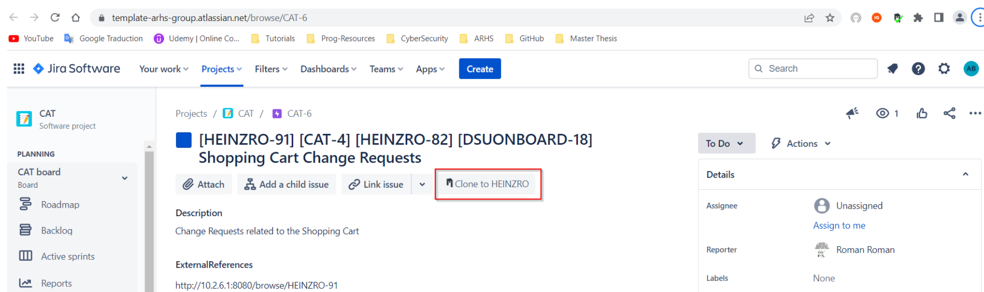

Figure 5.1. A screenshot of the ticket details view in the Jira cloud application with the "Clone" button injected

of the page that differs from the server one) and the clone button is visible under the ticket title.

As can be seen on the screenshot, the button is being injected at a slightly different place than in the server instance (see Section 3.3). Its CSS style is also different, being aligned on the CSS of the cloud instance. However, those are the only differences since both the name of the button and the functionalities remain the same.

5.2.2 Bulk tickets cloning

The bulk tickets cloning feature is injected in the "Advanced issue search" filter option of the Jira cloud instance. This page is the exact equivalent of the search page in the server instance, in that it also provides the user two ways to write JQL queries to search for tickets: the basic and advanced searches.

Screenshots of the "Bulk clone" button in the basic and advanced search have been provided in Figures 5.2 and 5.3. Once again, despite the design being slightly different, the functionalities are exactly the same as described in Chapter 4: clicking the button validates the JQL query provided by the user and opens the bulk modal if the query returns tickets linked to the same project.

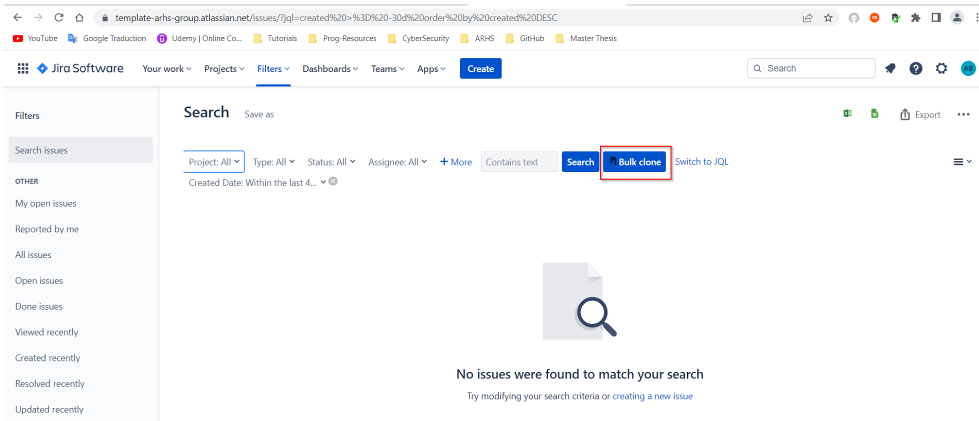

Figure 5.2. A screenshot of the "Advanced issue search" filter page in the Jira cloud application with the "Bulk clone" button injected when in basic search mode

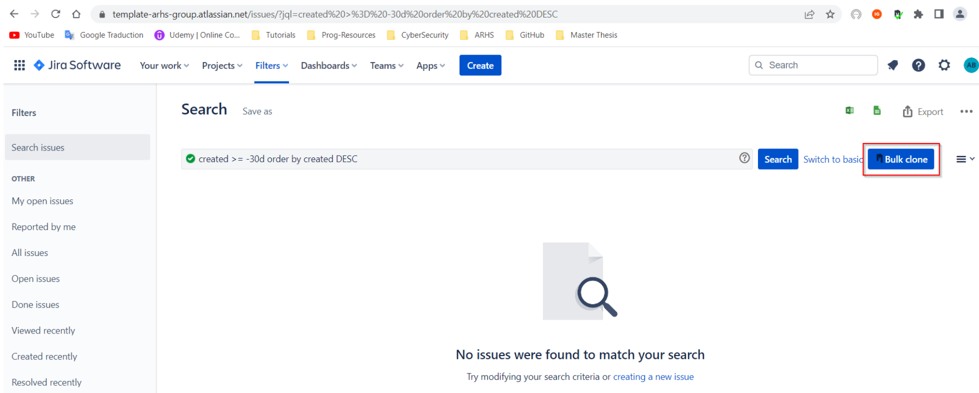

Figure 5.3. A screenshot of the "Advanced issue search" filter page in the Jira cloud application with the "Bulk clone" button injected when in advanced search mode

5.3 In brief

Summary of Chapter 5

- In this chapter, the context of the Jira application was discussed. With the progressive migration towards cloud instances and the upcoming end for the support in server instances, it appears as mandatory for the Jira Replicator plugin to support cloud instances.
- The differences between the server and the cloud instances of Jira were analysed. From them, a new architecture was determined for the plugin that would allow for support of both instances while keeping a robust and easily maintainable application and limiting the work load required.
- Finally, the integration of the plugin in the Jira cloud application was presented. Both the single ticket cloning and the bulk tickets cloning features were integrated in the cloud instance, respectively in the “ticket details view” and the “advanced issue search” (the equivalent of the search page on server instance).

Conclusion

This Master Thesis had for goal to analyse, design and implement an application that would perform cloning of Jira tickets between different instances of Jira applications. It was performed on behalf of the ARHS company which wanted to provide such a tool to its Project Managers (the stakeholders of this project), who have to work with one instance on the customer side, and another one internal to the company.

In Chapter 1, the project and the requirements given by the stakeholders were analysed. This allowed for a clear goal to be determined: an application able to be configured to perform unidirectional, single-time cloning operations of Jira tickets between any couple of Jira instances the user configured. From this goal, the architecture of the plugin was debated between the "Web Application" and the "Web Browser plugin", based on their different advantages and drawbacks. It was finally decided to use a Web Browser plugin for its clear advantage on the authentication process: Web Browser plugins are able to reuse the authentication cookies saved in the browser when the user logs to the different applications to authenticate their requests. Especially, since the recommended browser at ARHS is Chrome, it was decided to select the Chrome plugin for the project. Finally, Typescript was chosen to be the programming language of the plugin, for it has all the advantages of Javascript, being built from it, while providing a more secure development environment thanks to its strong typing, its compilation errors,...

After the general structure of the project was defined, it was time for the development to start. Chapter 2 detailed the first feature to be developed: the configuration screen of the replicator. This feature indeed serves as the basis to the cloning feature since the cloning required some information from the user to be executed. Information such as the domain of the Jira instances involved, the source and the target projects for the cloning or the mapping between ticket types from the customer to the target Jira were mandatory before

the cloning could be started. Before starting on the configuration screen, it was necessary to setup the structure of the plugin. The different files required for a “Chrome plugin”, such as the “JSON manifest”, the “options” file, the “service worker” and the “content scripts”, were implemented to create the plugin. The configuration screen was finally developed as the “options” of the plugin. It was designed to allow the user to create multiple configurations: each configuration will link two Jira instances, the source and the target instances, and hold the information required to perform a cloning between these two instances. The user can add as many configurations she wishes with the condition that there cannot be two configurations with the same source URI active at the same time.

With the information provided by the configurations defined by the user, it was finally possible to develop the cloning feature in Chapter 3. The requirements for the feature were to have a “Clone” button injected in the ticket details view of Jira. Clicking this button would clone it to the target Jira, while converting the button into a link to the clone ticket. Those requirements were then analysed to solve the different edge-cases in preparation to the implementation task. The cloning feature was then implemented partly in the “content script” for all tasks related to the UI of Jira (such as injecting the “Clone” button in the page) and partly in the “service worker” for tasks related to interactions with the Jira API (such as the HTTP requests to perform the cloning of the ticket). The implementation of the feature was finally presented: the interactions of the “Getting status”, “Clone” and “View” buttons was described based on the status of the cloning and the error modal displaying the error messages was described. However, the testing of the feature quickly showed its limitations: for Projects Managers with hundreds of tickets, cloning them one by one was simply too tedious to be useful.

After a discussion with the Product Owner of the project, it was decided that a new feature was necessary. Chapter 4 focused on this new task: the cloning of Jira tickets in bulk. The goal of this new feature was to use JQL (Jira Query Language) to search for multiple tickets, show the tickets found in a modal provided by the plugin and then give the possibility to the user to clone them all at once or select a sub-set to clone. Since Jira applications already provide a page for performing searches via JQL, it was decided to inject the new functionality in this page as well. That way, it was possible to re-use the inputs provided by Jira to encode the JQL query, as those inputs have built-in

functionalities that simplify the creation of the query. After analysing the requirements, the architecture of the feature and its integration in the plugin were determined: it was decided to distinguish single ticket cloning from bulk cloning via the URL of the page, the different ways to retrieve the JQL query were debated and explained, and the conditions on contacting the new Jira API introduced a limitation to the maximum number of tickets that could be displayed. The implementation of the feature was then discussed. The different functionalities related to the bulk cloning feature were presented: the "Bulk Clone" button was described and the condition of all searched tickets to belong to the same project for displaying the modal was explained. Finally, the bulk clone modal was introduced, the tickets table was described and the functionalities such as the check-boxes for selecting tickets to be cloned, the handling of error and the 'Clone selected" button were explained.

Chapter 4 introduced the last feature of the replicator: the possibility to clone tickets in bulk. While the final functionalities had been implemented for the plugin and successfully tested by the Project Managers, a last change had to be performed to finalise the project. Indeed, all the considerations for the Jira replicator plugin were taken on the basis of a Jira server instance since the instances used by the Project Managers of ARHS are all server instances. However, the decision was recently taken by Atlassian (the company owning Jira) to fully migrate Jira server instances to cloud instances and end the support towards server instances. It was thus necessary for the Jira replicator to be compatible with cloud instances. Unfortunately, some UI differences made it impossible for the different elements to be injected in the cloud instances. Chapter 5 focused on making the plugin compatible with cloud instances. For this, the architecture of the plugin had to be completely reworked: it was decided to create a base class for the service handling the UI tasks and have two different services extending this class, one for cloud instances and one for server ones. In these two classes, only the functionalities that differed in logic were being rewritten or implemented. This solution allowed for a robust and easily maintainable code since it would be possible to create new classes to support different versions of the Jira application in case the UI were to present breaking changes. Finally, the integration of the plugin and the UI changes were presented as a conclusion to the chapter.

To summarise, an application in the form of a Chrome plugin was successfully developed to perform cloning of Jira tickets between different Jira instances.

This application provides the user with a configuration screen where she can define various configurations for the different projects she manages. The plugin injects content in Jira instances: in the ticket details view, it injects a "Clone" button to clone the ticket currently considered, while in the Jira search page, it injects a "Bulk clone" button to perform cloning of tickets in bulk based on the result of a JQL search. Finally, the plugin supports both Jira server instances and Jira cloud instances and can be easily extended to support future versions of those applications.

Bibliography

[1] Atlassian. *Atlassian Server end of life*. URL: https://www.atlassian.com/migration/assess/journey-to-cloud. (accessed: 05.05.2023).

[2] Atlassian. *Configure a custom field*. URL: https://support.atlassian.com/jira-cloud-administration/docs/configure-issue-custom-fields. (accessed: 22.04.2023).

[3] Atlassian. *Jira API*. URL: https://developer.atlassian.com/server/jira/platform/jira-rest-api-examples/#request-19. (accessed: 30.04.2023).

[4] Atlassian. *Jira Cloud API*. URL: https://developer.atlassian.com/cloud/jira/platform/rest/v3/intro/#about. (accessed: 30.04.2023).

[5] Atlassian. *Use advanced search with Jira Query Language (JQL)*. URL: https://support.atlassian.com/jira-service-management-cloud/docs/use-advanced-search-with-jira-query-language-jql/. (accessed: 30.04.2023).

[6] Atlassian. *Welcome to Jira software*. URL: https://www.atlassian.com/software/jira/guides/getting-started/introduction#what-is-jira-software. (accessed: 13.04.2023).

[7] Adobe experience cloud blog. *Waterfall methodology: a complete guide*. URL: https://business.adobe.com/blog/basics/waterfall. (accessed: 13.04.2023).

[8] Bootstrap. *Get started with Bootstrap*. URL: https://getbootstrap.com/docs/5.3/getting-started/introduction/. (accessed: 23.04.2023).

[9] Chrome. *Chrome extensions Architecture overview*. URL: https://developer.chrome.com/docs/extensions/mv3/architecture-overview/. (accessed: 22.04.2023).

[10] Chrome. *Chrome storage*. URL: https://developer.chrome.com/docs/extensions/reference/storage/. (accessed: 24.04.2023).

[11] Forbes. *What is Waterfall methodology*. URL: https://www.forbes.com/advisor/business/what-is-waterfall-methodology/. (accessed: 13.04.2023).

[12] Ruben Jackle. *Jira Cloud vs Server: main differences 2022*. URL: https://actonic.de/en/jira-cloud-vs-server/. (accessed: 30.04.2023).

[13] Cameron McKenzie. *Javascript vs Typescript: What's the difference?* URL: https://www.theserverside.com/tip/JavaScript-vs-TypeScript-Whats-the-difference. (accessed: 20.04.2023).

[14] Scrum. *What is Scrum?* URL: https://www.scrum.org/learning-series/what-is-scrum/what-is-scrum. (accessed: 13.04.2023).

[15] Wikipedia. *Agile software development*. URL: https://en.wikipedia.org/wiki/Agile_software_development. (accessed: 13.04.2023).

[16] Wikipedia. *Waterfall model*. URL: https://en.wikipedia.org/wiki/Waterfall_model. (accessed: 13.04.2023).

List of Figures

2.1 A screenshot of the configuration screen of the Jira Replicator plugin . . . 22
2.2 A screenshot of the base configuration section of the configuration screen . . . 23
2.3 A screenshot of the ticket types mapping section of the configuration screen . . . 23
2.4 A screenshot of the custom fields configuration section of the configuration screen . . . 24
2.5 A screenshot of the actions bar of the configuration screen . . . 25

3.1 A screenshot of the ticket details view in the Jira application . . . 29
3.2 A screenshot of the "Getting status" button injected in the ticket details view . . . 33
3.3 A screenshot of the "Clone" button injected in the ticket details view . . . 34
3.4 A screenshot of the "View" button injected in the ticket details view . . . 34
3.5 A screenshot of the "2+ cloned issues" button injected in the ticket details view . . . 35

3.6 A screenshot of the error modal injected in the ticket details view when an error occurred in the ticket cloning flow 36

4.1 A screenshot of the Jira basic search page with the bulk clone button of the plugin 46

4.2 A screenshot of the Jira advanced search page with the bulk clone button of the plugin 46

4.3 A screenshot of the error message displayed to the user when the JQL search resulted in tickets from multiple projects 47

4.4 A screenshot of the bulk clone modal listing in a table the different tickets searched for 49

4.5 A screenshot of the action menu of the bulk clone modal 49

5.1 A screenshot of the ticket details view in the Jira cloud application with the “Clone” button injected 55

5.2 A screenshot of the “Advanced issue search” filter page in the Jira cloud application with the “Bulk clone” button injected when in basic search mode 56

5.3 A screenshot of the “Advanced issue search” filter page in the Jira cloud application with the “Bulk clone” button injected when in advanced search mode 56

List of Tables

1.1 Advantages and drawbacks of Javascript and Typescript [13] . 13