Xporter for JIRA Cloud

Fábio Cristiano Martins Antunes

Thesis to obtain the Master of Science Degree in

**Information Systems and Computer Engineering**

Supervisor: Prof. João Coelho Garcia

**Examination Committee**

Chairperson: Prof. Daniel Jorge Viegas Gonçalves  
Supervisor: Prof. João Coelho Garcia  
Member of the Committee: Prof. Miguel Leitão Bignolas Mira da Silva

October 2016
Dedicated to my family and friends
Acknowledgments

I want to thank my advisor Professor João Garcia, for all the support and attention provided throughout the Thesis.
I want to thank as well to my family and my best friends, all the strength they gave me during the course of the thesis.
Lastly, I would like to thank my co-workers for all the support and all the interesting ideas they shared with me and also to Xpand IT for the opportunity provided and confidence in my work.
Abstract

**XPORTER** for JIRA is a plugin that extends the functionality of JIRA, an Atlassian’s issue and project tracking tool, providing to users an easy way to extract and format data from JIRA using customized templates and producing, as a result, personalized reports. JIRA is currently available in two versions, JIRA Server and JIRA Cloud, Xporter for JIRA was only compatible with JIRA Server. This thesis addressed the challenge of creating a JIRA Cloud compatible version of Xporter for JIRA, an Atlassian Connect add-on that is already available to the public, implemented with the help of a development framework called Atlassian Connect Express, used to create Atlassian connect add-ons in NodeJS. Xporter for JIRA Cloud was implemented mostly in JavaScript using only an external service developed in Java, responsible for the documents generation and that is shared between versions Server and Cloud of the add-on. After finishing the add-on development, all the infrastructure to support it was also created, using for this, the Amazon Web Services and MongoDB Atlas service. This infrastructure has been created in order to ensure that the add-on would be able to operate correctly and efficiently even when subjected to great amounts of work by scaling itself and also to guarantee fault tolerance and high availability, being deployed in different and independent regions.

**Keywords:** Xporter for JIRA Cloud, Issue Tracking Tool, JIRA, Cloud, Connect Framework, Web Application.
Resumo

XPORTER for JIRA é um add-on que estende as funcionalidades do JIRA, uma ferramenta de rastreamento e gestão de problemas, possibilitando aos utilizadores do JIRA uma maneira fácil de exportar e formatar os dados que existem no sistema através de templates personalizados, criando como resultado relatórios personalizados. O JIRA está disponível em duas versões, JIRA Server e JIRA Cloud, sendo que atualmente o Xporter apenas é compatível com a versão Server. Esta dissertação teve como objetivo a criação de uma versão do Xporter for JIRA compatível com a versão Cloud do JIRA, um Atlassian Connect Add-on desenvolvido com a ajuda de uma framework de desenvolvimento chamada Atlassian Connect Express usada para criar Atlassian Connect Add-ons em NodeJS. O Xporter for JIRA Cloud foi desenvolvido maioritariamente em JavaScript usando apenas um serviço externo desenvolvido em Java, utilizado para a geração de documentos e que é partilhado entre as versões Server e Cloud do add-on. Após o desenvolvimento do add-on foi criada também toda a infraestrutura para o suportar, usando para isso os Amazon Web Services e o serviço MongoDB Atlas. Esta infraestrutura foi criada com o objetivo de garantir que o add-on estaria apto a funcionar corretamente e com eficiência mesmo quando sujeito a grandes quantidades de trabalho adaptando-se a estas situações, bem como garantir tolerância a faltas e alta disponibilidade estando a funcionar em diferentes regiões independentes entre si.

Contents

Acknowledgments ............................................................. v
Abstract ................................................................. vii
Resumo ................................................................. ix
List of Figures .......................................................... xiii

1 Introduction 1
  1.1 Motivation .......................................................... 1
  1.2 Objectives .......................................................... 6
  1.3 Thesis Outline ....................................................... 6

2 Background 7
  2.1 Current Competition ................................................ 8
  2.2 Atlassian JIRA ....................................................... 9
    2.2.1 JIRA Server ................................................... 9
    2.2.2 JIRA Cloud ................................................... 10
  2.3 Functional Requirements .......................................... 11
    2.3.1 Functional Requirements Related To System Administrators ................................................... 11
    2.3.2 Functional Requirements Related To System Users ................................................... 13
  2.4 Non Functional Requirements ..................................... 14
  2.5 Technology .......................................................... 16
    2.5.1 Frameworks .................................................... 16
    2.5.2 Database Models .............................................. 17
    2.5.3 Distributed File Systems .................................... 20
  2.6 Tests and Evaluation ............................................... 23
  2.7 Process and Managements ......................................... 24

3 Implementation 25
  3.1 Idea .............................................................. 25
  3.2 Conceptual Implementation ....................................... 25
  3.3 Technical Implementation ....................................... 29
    3.3.1 Project Structure ............................................ 32
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>4 Infrastructure Setup &amp; Deployment</strong></td>
<td></td>
</tr>
<tr>
<td>4.1 Overview</td>
<td>37</td>
</tr>
<tr>
<td>4.2 Components</td>
<td>39</td>
</tr>
<tr>
<td><strong>5 Results</strong></td>
<td></td>
</tr>
<tr>
<td>5.1 Evaluation Questions</td>
<td>43</td>
</tr>
<tr>
<td>5.2 Tests</td>
<td>44</td>
</tr>
<tr>
<td><strong>6 Conclusions</strong></td>
<td></td>
</tr>
<tr>
<td>6.1 Achievements</td>
<td>55</td>
</tr>
<tr>
<td>6.2 Future Work</td>
<td>56</td>
</tr>
<tr>
<td><strong>Bibliography</strong></td>
<td>57</td>
</tr>
</tbody>
</table>
## List of Figures

1.1 JIRA Concepts ......................................................... 4
1.2 Xporter for JIRA - Templates ........................................ 5
1.3 Xporter Templates Input/Output Matrix .......................... 5

2.1 Xporter for JIRA Cloud and Server - UI Compatibility ......... 8
2.2 Xporter for JIRA Cloud and Server - FR Compatibility ........ 8
2.3 JIRA Architecture .................................................... 9
2.4 JIRA Server Architecture .......................................... 10
2.5 JIRA Cloud Architecture ........................................... 11
2.6 Database Categories ............................................... 19

3.1 NodeJS Event Loop .................................................. 26
3.2 Xporter for JIRA Components ...................................... 26
3.3 Connect Add-ons Architecture .................................... 27
3.4 Connect Add-ons Structure ....................................... 30
3.5 Xporter for JIRA Cloud Data Flow ............................... 32
3.6 Xporter for JIRA Cloud - Descriptor ............................ 33
3.7 Xporter for JIRA Cloud - Models ................................ 34
3.8 Xporter for JIRA Cloud - Routes ................................ 35

4.1 Xporter for JIRA Cloud Architecture .............................. 38

5.1 Xporter for JIRA Cloud - Page Loading Times .................. 44
5.2 Xporter for JIRA Cloud - Page Throughput ...................... 45
5.3 Xporter for JIRA Cloud - Overall Page Timings ............... 45
5.4 Xporter for JIRA Cloud - Xporter Cloud Project Workflow .... 46
5.5 Requests Results - Scenario 1 ................................... 47
5.6 Requests Time Over Time - Scenario 1 .......................... 47
5.7 Node EC2 Metrics - Scenario 1 .................................. 48
5.8 Webengine EC2 Metrics - Scenario 1 ............................ 48
5.9 MongoDB Atlas Metrics - Scenario 1 ........................... 49
5.10 Requests Results - Scenario 2 .................................. 49
Chapter 1

Introduction

1.1 Motivation

With the fast development of processing and storage technologies and the success of the Internet, computing resources have become cheaper, more powerful and more available than ever before. This technological trend has enabled the realization of a new computing model called cloud computing, in which resources are provided as general utilities that can be leased and released by users through the Internet in an on-demand fashion. Cloud computing has recently emerged as a new paradigm for hosting and delivering services over the Internet. Cloud computing is attractive to business owners as it eliminates the requirement for users to plan ahead for provisioning, and allows enterprises to start from the small and increase resources only when there is a rise in service demand. Cloud computing is composed of five essential characteristics, three service models, and four deployment models[1, 2, 3].

Essential Characteristics

- **On-demand self-service** - A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.

- **Broad network access** - Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).

- **Resource pooling** - The provider’s computing resources are pooled to serve multiple consumers using a Multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or data-center). Examples of resources include storage, processing, memory, and network bandwidth.

- **Rapid elasticity** - Capabilities can be elastically provisioned and released, in some cases auto-
matically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.

- **Measured service** - Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

**Service Models**

- **Software as a Service (SaaS)** - The capability provided to the consumer is to use the provider’s applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user specific application configuration settings.

- **Platform as a Service (PaaS)** - The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.

- **Infrastructure as a Service (IaaS)** - The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications and possibly limited control of select networking components (e.g., host firewalls).

**Deployment Models**

- **Private cloud** - The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises.

- **Community Cloud** - The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off premises.
• **Public Cloud** - The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider.

• **Hybrid Cloud** - The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load-balancing between clouds).

Cloud Computing is not a single technology. It is better described as a business development, whose realization has been enabled by several disciplines: computer architecture, operating systems, data communications, and network and operations management. Every day that goes by, cloud computing is winning more companies that invest heavily in it, one of that companies is Atlassian.

Atlassian is an enterprise software group of companies founded in 2002 by Mike Cannon-Brookes and Scott Farquhar that has started betting and investing in cloud computing. Atlassian develops products geared towards software developers and project managers. It is best known for its issue tracking application, JIRA. Although commonly used for software issue tracking, due to its advanced customization features the web application is also highly suitable for other types of ticketing systems and project management. According to Atlassian, JIRA is a "workflow management system that lets you track your work in any scenario." [4]

Traditionally, issue tracking systems have been largely viewed as simple data stores where software defects are reported and tracked as “bug reports” within an archival database. Currently the most advanced way of dealing with bugs is to enter them into an Issue tracking system. Issue trackers address the critically important task of tracking and managing issues and bugs that emerge during a project. Issue tracking tools such as JIRA, Trello, Bugzilla, Pivotal, and Mantis BT are a class of project management software that keeps track of various issues for project teams. These tools are also known as project tracking tools. In software projects, these tools are often referred to as bug tracking tools because software defects are the main issues in the context of software development [5]. In other types of projects, issues often mean tasks. Issue tracking systems help organizations manage issue reporting, assignment, tracking, resolution, and archiving.[6, 7]

Functionalities that an Issue Tracker must ensure in Software Projects [8]:

• Share the information across the team

• Have an instant overview of the state of the software

• Expertly decide about releasing

• Set and update the importance of individual fixes and adjustments
• Have a recorded history of changes.
• What should be fixed or created
• What the bug symptoms and appearances are, what actually doesn’t work
• Who reported the request, who confirmed, analyzed, implemented the solution and verified it
• When the request was reported, when it was fixed and when verified
• What led to the decision to choose one way of fixing instead of another
• What changes in code were made
• How long it took to handle the request

To easily understand JIRA we must know what means "Issue", the fine-grained concept inside JIRA. Due to JIRA advanced customization features, different organizations using JIRA to track different kinds of issues so depending on how the organization is using JIRA. An issue could represent a software bug, a project task, a help desk ticket, a leave request form among others. Besides the concept of "Issue" there are other important JIRA concepts: Project, Component and Workflow, as shown in Fig. 1.1.

![JIRA Components](image)
![JIRA Workflow](image)

Figure 1.1: JIRA Concepts

JIRA allows extending its functionality by installing plugins to extend the platform. The JIRA installations come with a set of pre-installed plugins (from Atlassian) but later on other plugins from external companies can be installed through a marketplace provided by Atlassian [9]. Xporter for JIRA is one of that plugins that extends JIRA platform. It was created in 19th December 2011 by XpandIT, a Portuguese global company specialized in strategic planning, consulting, implementation and maintenance of enterprise software, fully adapted to the customers needs. They have services and products in several areas like Business Intelligence, Big Data, BPM and Enterprise Middleware and Collaborative Platforms. XpandIT stands out for its innovative approach fully supported by tools, processes and agile methodologies [10], fully mapped with CMMI.
Xporter for JIRA was created to provide for JIRA users a way to generate custom reports in several formats, filled with data from JIRA issues, as shown in Fig. 1.2. To create this custom reports users must create or adapt Xporter for JIRA templates (Microsoft Word or Excel files), which by using special placeholders are able to access data from issues, as shown in Fig. 1.2. The Fig. 1.3 shows the possible output formats for each template type. With Xporter for JIRA it is also possible to export several issues at the same time using JIRA Bulk operation, or using loops defined in Xporter for JIRA templates.

(a) Xporter for JIRA Template
(b) JIRA Workflow

Figure 1.2: Xporter for JIRA - Templates

Figure 1.3: Xporter Templates Input/Output Matrix
1.2 Objectives

The plugins for JIRA are divided into three types: Plugins Type-1, Plugins Type-2 and Atlassian Connect plugins (JIRA Cloud) [11]. Only Atlassian Connect Plugins can be used/installed in instances of JIRA Cloud. The Atlassian bet and investment in cloud products led to a migration of much of the JIRA Server customers to JIRA Cloud and as such, there is a need to create a new version of Xporter for JIRA (Plugin Type-2) as an Atlassian Connect plugin. That way it can be installed on JIRA Cloud and thereby be sold to new JIRA cloud customers or to customers who have migrated from server version.

1.3 Thesis Outline

The remaining of the document is organized as follows.

Chapter 2 begins by describing Xporter for JIRA progress in Atlassian marketplace and discussed the expected competition of Xporter for JIRA Cloud on JIRA Cloud marketplace. Will be presented the JIRA Cloud and JIRA Server architectures and their advantages and disadvantages according to the customer’s point of view as well as their respective markets. After that are identified the functional and nonfunctional requirements that Xporter for JIRA Cloud will need to meet and the technological challenges derived from them and from its new architecture. Finally, at the end will be discussed and planned the tests and the maintenance strategy for Xporter.

Chapter 3 contains a detailed description of Xporter for JIRA Cloud implementation. It begins by making an analyse of the Xporter for JIRA Server modules and understanding what could be possibly used in cloud version and which changes were needed to do it. All the major components will be described as well as the required communication between them to get all working as expected.

Chapter 4 presents and explains the cloud computing platform architecture where the Xporter for JIRA Cloud will be running. Will be described all the components, their communication, configuration, security rules and strategies to assure high availability, scalability and great performance.

Chapter 5 reports and analyzes the difficulties in each project phase and how they were overcome, what went well and what could have been approached differently and an overview of the performed tests and the process and maintenance strategies that were followed during development.

Finally, chapter 6 concludes this dissertation, by resuming the key results achieved and discussing possible directions for future work.
Chapter 2

Background

JIRA is a highly flexible Issue tracking software that allows extend its functionality by installing plugins developed by external companies, like Xporter for JIRA, which are obtained in the marketplace Atlassian. Xporter for JIRA was first developed in 2011 and was released to the public through the Atlassian marketplace in 19th December 2011. In this first version of the plug-in it was compatible with versions 4.4 - 4.4.5 of JIRA server. Currently Xporter for JIRA has a total of 1112 active customer installations being United States of America, Germany and United Kingdom the countries with Xporter for JIRA biggest clients [12]. Atlassian recently released version 7 of JIRA Server. Due to modifications in the JIRA API for this latest version of the Xporter for JIRA now has to maintain two lines of development, one for versions 6 of JIRA Server (only bug fixes) and one for versions 7 of JIRA server.

In addition to the new versions of its server products, Atlassian has been increasingly focused on their cloud versions. This ambition has to do especially in the ease and speed with which a customer can begin enjoying JIRA, the facilities of integration with other Atlassian cloud products and the fact that customers do not have to worry about buying and maintaining their own infrastructures.

As already mentioned the plugins for JIRA are divided into Plugins Type-1, Plugins Type-2 and Atlassian Connect plugins (JIRA Cloud). Since only Atlassian Connect Plugins can be used-installed in instances of JIRA Cloud an Atlassian Connect plugin version of Xporter for JIRA Cloud needs to be created.

Currently not all Xporter for JIRA Server features could be implemented in Xporter for JIRA Cloud, as shown in Fig. 2.1 and Fig. 2.2.

These limitations, mainly caused due to non-extendable modules, are currently prescribed by Atlassian plugins for the type Atlassian Connect but may in the future become available.
2.1 Current Competition

Currently in JIRA Cloud there is not much competition for JIRA Xporter. It comes down to two other plugins that aim to export data from the instance of JIRA.

- **Exporter – Issues to CSV and Excel** - The focus of this Atlassian Connect Add-on is to allow the exportation of issue comments and issue transitions history to xlsx format (Microsoft Excel) files. It does not allow to generate custom reports, definition of permission schemes or data manipulation as Xporter for JIRA provides. This represent a huge advantage for Xporter because one of the client’s most appreciated feature is the flexibility provided by Xporter templates, allowing them to generate a template that completely fits their purposes. This plugin has 673 active client installations, so it has less than half of the Xporter for JIRA Server client installations [13].
• **All-In-One JIRA Reports** - This Atlassian Connect plugin allows users to create insightful reports for daily use, executive reporting or any other business purpose including invoice creation, status reporting, performance measurement, sprint tracking, epic progress, trend analysis, time tracking among other. This is achieved using drag-and-drop, within a user interface in JIRA, that allows to choose the fields to export and the way they are exported by choosing between a set of predefined template styles. The limitation here is related to the predefined template styles that prevent the clients to have total control over their own templates. Although, the interface provided is a very positive point because it allows great facility in the construction of templates that are not too complex. This plugin has 175 active client installations, so it stills in a initial growing phase [14].

In general we can see that none of the Xporter competitors allows customers to have a freedom and a power of customization of templates as high as the Xporter allows. Moreover, none of them allows data manipulation in the templates, definition of Post Functions to integrate exports with JIRA Workflows or the variety of exportation formats [15]. However, there are positive points in these other plugins, the fact that they provide an interface to build the templates can be a strong argument when choosing the plugin that they will buy to export data from JIRA.

### 2.2 Atlassian JIRA

JIRA is available in two distinct options: **JIRA Server** and **JIRA Cloud** (SaaS — Software as a service) [16]. An overview of general JIRA architecture is shown in Fig. 2.3 [17, 18, 19].

![JIRA Architecture](image)

**Figure 2.3: JIRA Architecture.**

#### 2.2.1 JIRA Server

JIRA Server was the first being released and currently stills the one with more clients. The reason why most of the customers continues to use JIRA Server is related to the following points:
- Source code control.
- Flexibility and customization options.
- More add-ons available from marketplace.
- Upgrades control.
- No admin restrictions.
- Full access to databases.

An overview over JIRA Server plugins architecture is shown in Fig. 2.4.

![JIRA Server Architecture Diagram](image)

**Figure 2.4: JIRA Server Architecture.**

### 2.2.2 JIRA Cloud

JIRA Cloud, formerly known as *OnDemand* is where Atlassian is investing more. The main reason for that is the simplicity in environment setup the little or no concern with infrastructure issues and of course the speed and reliability of cloud-based applications [20]. The most important points of JIRA Cloud are:

- High scalability and availability.
- Own infrastructure not needed.
- No additional hardware requirements and associated costs.
- No additional work on the own IT department.
- No maintenance tasks needed.
- Systems can be used quickly and short term.

An overview over JIRA Cloud architecture is shown in Fig. 2.5.

Once JIRA Cloud marketplace stills very small and more and more JIRA customers are migrating from JIRA Server to JIRA Cloud, Xporter for JIRA has to be released as soon as possible to the market so this way it can ensure their existing customers who have migrated to JIRA Cloud and get new clients that have no available plugins with similar functionality.
2.3 Functional Requirements

Xporter for JIRA Cloud must provide all the features available in Xporter for JIRA Server that are possible to implement in JIRA Cloud version. For that to be accomplished additional logic and components will be needed, for example, the management of a database to store all the add-on data. Below are presented the functional and non-functional requirements related to the new version of Xporter for JIRA.

2.3.1 Functional Requirements Related To System Administrators

- Administration Section
  - Xporter Global Settings
    * I can manage Xporter for JIRA through an Administration Web-Section.
    * I can set the available output formats in Global Settings Page under Administration Web-Section.
    * I can set the default output format in Global Settings Page under Administration Web-Section.
    * I can set the Break Pages format for Bulk Exports in Global Settings Page under Administration Web-Section.
    * I can set the Maximum number of issues being exported for Bulk Exports in Global Settings Page under Administration Web-Section.
    * I can set the Maximum number of simultaneous requests for Bulk Exports in Global Settings Page under Administration Web-Section.
    * I can enable or disable Xporter for JIRA for all projects and users in Global Settings Page under Administration Web-Section.
    * I can enable or disable Xporter for JIRA panel on Single Issue View in Global Settings Page under Administration Web-Section.
    * I can enable or disable servlet anonymous requests in Global Settings Page under Administration Web-Section.
I can enable or disable the inclusion of attachments in exports with pdf as output format in Global Settings Page under Administration Web-Section.

- Manage Templates
  
  * I can add a template by uploading it in Manage Templates Page under Administration Web-Section.
  
  * I can set a template as default in Manage Templates Page under Administration Web-Section.
  
  * I can edit a template name and description in Manage Templates Page under Administration Web-Section.
  
  * I can remove a template in Manage Templates Page under Administration Web-Section.

- Template Store

  * I can install a template in Template Store Page under Administration Web-Section.
  
  * I can re-install a template in Template Store Page under Administration Web-Section.
  
  * I can search a template by name in Template Store Page under Administration Web-Section.
  
  * I can search templates by category in Template Store Page under Administration Web-Section.
  
  * I can sort templates by rating, download or publish date in Template Store Page under Administration Web-Section.

- Permission Schemes

  * I can add a permission scheme in Permissions Schemes Page under Administration Web-Section.
  
  * I can edit a permission scheme in Permissions Schemes Page under Administration Web-Section.
  
  * I can delete a permission scheme in Permissions Schemes Page under Administration Web-Section.
  
  * I can copy a permission scheme in Permissions Schemes Page under Administration Web-Section.
  
  * I can define the projects where a permission scheme is applied in Permissions Schemes Page under Administration Web-Section.
  
  * I can define Permissions for a permission scheme in Permissions Schemes Page under Administration Web-Section.
  
  * I can edit Permissions of a permission scheme in Permissions Schemes Page under Administration Web-Section.
  
  * I can associate issue types to a Scheme's Permission in Permissions Schemes Page under Administration Web-Section.
• I can associate JIRA Roles to a Scheme’s Permission in Permissions Schemes Page under Administration Web-Section.
• I can associate JIRA Groups to a Scheme’s Permission in Permissions Schemes Page under Administration Web-Section.
• I can associate templates to each issue types in a Scheme’s Permission in Permissions Schemes Page under Administration Web-Section.

– Post Function Authentication
  • I can define a username and a password in Xporter Post function Authentication page under Administration Web-Section in order to authenticate requests from post functions.

– License
  • I can add a XpandIT license in Xporter License Management page under Administration Web-Section.
  • I can remove a XpandIT license in Xporter License Management page under Administration Web-Section.

– Documentation
  • I can access Xporter for JIRA documentation in Documentation page under Administration Web-Section.

• Servlet
  – I have a Xporter for JIRA servlet available.

• Post Functions
  – I can define Xporter for JIRA Create Document Post Functions in order to trigger an export after some event occurs.
  – I can define Xporter for JIRA Email Report Post Functions in order to trigger an export and sent by email after some event occurs.

2.3.2 Functional Requirements Related To System Users

• Exportations
  – I can perform a single issue export using the Xporter for JIRA panel on single issue view.
2.4 Non Functional Requirements

- **Versioning/Updates**
  - Xporter for JIRA must support versioning.
  - Xporter for JIRA Global Settings should survive to any update (minor or major version).
  - Xporter for JIRA Permission Schemes should survive to any update (minor or major version).
  - Defined Xporter for JIRA Post Functions should survive to any update (minor or major version).
  - Xporter for JIRA installed/uploaded templates should survive to any update (minor or major version).

- **Permission Schemes**
  - Xporter for JIRA must respect all defined permission schemes and apply them to each exportation performed by a user.

- **Availability**
  - Xporter for JIRA should be always available to users except for scheduled maintenance downtime and disaster scenarios
  - Xporter for JIRA owners must be able to increase Xporter for JIRA availability by increasing the number of servers.
  - No Xporter for JIRA module (service or otherwise) failure should result in system failure.

- **Testing/Maintenance**
  - Xporter for JIRA code should be flexible and easy to maintain.
  - Xporter for JIRA code should rely on independently services that will be used by Xporter for JIRA Server and Xporter for JIRA Cloud.
  - Xporter for JIRA should be tested on clean and independent environments.
  - All Xporter for JIRA tests must pass prior to production release.
  - Each Xporter for JIRA module must be independently testable.

- **Internationalization**
  - Xporter for JIRA should provide internationalization of user interfaces.

- **Methodologies**
  - Xporter for JIRA development will follow XPAGILE methodology (agile software development methodology created by XpandIT).
• **Performance**
  - Xporter for JIRA should support hundreds of concurrent exports.
  - Xporter for JIRA should perform an export on average in $0.05 \times \text{Number of issues being exported}$ seconds.
  - Xporter for JIRA should respond to a user action as fast as possible to user feel that the system is reacting instantaneously.

• **Security**
  - Xporter for JIRA web Services must be secured by authenticating and authorizing consumers.

• **Compatibility**
  - Xporter for JIRA should support all JIRA versions since JIRA 7.

• **System Overview**
  - Develop team should have tools that report detailed information on system usage and capacity.
  - Xporter for JIRA must provide tools for easy administration of clients, schemas and data backups.

• **JIRA Compliant**
  - Xporter for JIRA must support the browsers supported by JIRA.
  - Xporter for JIRA UI should be in compliance with JIRA design guidelines.

• **Documentation**
  - Support documentation should be provided for each Xporter for JIRA release.
  - Xporter for JIRA Security statement must be published.
  - Xporter for JIRA Data Security statement must be published.
2.5 Technology

During Xporter for JIRA Cloud development were found some technological challenges. These challenges are related to several factors: the differences between the Plugins and Connect Frameworks (used to development JIRA Server and JIRA Cloud add-ons respectively), the lack of experience in the new development stack and also with the fact that the new architecture has brought concerns that didn’t exist before. In JIRA Cloud add-ons vendors are responsible for setting up the platform where the add-on will be running, must manage all the add-on data and settings and the JAVA API from Plugins Framework isn’t available anymore.

2.5.1 Frameworks

To help in the development of Atlassian Connect Plugins [21], Atlassian provided a framework that could be used to help develop new add-ons and is available in several languages such as NodeJS, Spring Boot, .NET, Play (Java) [22], Play (Scala) among others. Currently only NodeJS and Spring Boot versions are officially supported by Atlassian, the other ones are community versions.

**Framework Functionalities:**

- Serve descriptor and add-on UI
- Handle add-on installation
- Persistent store
- Handle add-on request
- JWT token handler [23]
- Crypto library
- JSON and HttpClient libs

So if is preferable developing in another stack (e.g. Rails, Django), we need to ensure that functionalities by ourselves. Therefore, it is better to opt for using the framework in the language we feel most confident, achieving that way a faster development and in case of limitations or doubts we can exchange ideas or expose problems in the community that is also using the framework [24, 25].

The most used versions of Atlassian Connect Framework are:

1. **Atlassian Connect Express** [26]
   
   Language/Technology - Node.js and Express

2. **Atlassian Connect Play** [27]
   
   Language/Technology - Play and Java

At the beginning of the development, the Spring Boot version did not exist and the Atlassian Connect Play was officially supported. For that reasons and considering that our development team used to work with java we have chosen Atlassian Connect Play as our connect framework but when that version has ceased to be officially supported we moved into NodeJS version.
2.5.2 Database Models

As stated above, the data management of the add-on and their settings must now be made out of JIRA. This requires the use of an external database which will be used for the add-on and must be managed by the add-on vendor. Therefore, an analysis of the various types of databases was made in order to find the best option in the specific case of Xporter for JIRA Cloud.

1. **Relational databases**, which can also be called relational database management systems (RDBMS) or SQL databases. The most popular of these are Microsoft SQL Server, Oracle Database, MySQL, and IBM DB2. These RDBMS’s are mostly used in large enterprise scenarios, with the exception of MySQL, which is mostly used to store data for web applications, typically as part of the popular LAMP stack (Linux, Apache, MySQL, PHP/ Python/ Perl).

The reasons for the dominance of relational databases are: simplicity, robustness, flexibility, performance, scalability and compatibility in managing generic data. For large databases, especially ones used for web applications, the main concern is scalability. As more and more applications are created in environments that have massive workloads (i.e. Amazon), their scalability requirements can change very quickly and grow very large. Relational databases scale well, but usually only when that scaling happens on a single server (“scale-up”). When the capacity of that single server is reached, you need to “scale-out” and distribute that load across multiple servers, moving into so-called distributed computing [28, 29]. This is when the complexity of relational databases starts to cause problems with their potential to scale. If you try to scale to hundreds or thousands of servers the complexities become overwhelming. The characteristics that make relational databases so appealing are the very same that also drastically reduce their viability as platforms for large distributed systems [30, 31, 32].

Relational-model databases can be tweaked and set up to run large-scale read-only operations through data warehousing, and thus potentially serve a large amount of users but data warehouses are distinct from typical databases in that they are used for more complex analysis of data. This differs from the transactional database, whose main use is to support operational systems and offer day-to-day, small scale reporting [33, 34, 35].
2. **Non-relational databases**, also called NoSQL databases, the most popular being MongoDB [36], DocumentDB, Cassandra, Couchbase, HBase, Redis, and Neo4j. These databases are usually grouped into four categories: Key-value stores, Graph stores, Column stores, and Document stores.

**Non Relational Database Types**

(a) **Key-value stores** – These databases pair keys to values. An analogy is a files system where the path acts as the key and the contents act as the file. There are usually no fields to update, instead, the entire value other than the key, must be updated if changes are to be made. It scales well but it can limit the complexity of the queries and other advanced features. Examples are: Dynamo, MemcacheDB, Redis, Riak, FairCom c-treeACE, Aerospike, OrientDB, MUMPS, HyperDex, Azure Table Storage.

(b) **Graph stores** – These excel at dealing with interconnected data. Graph databases consist of connections, or edges, between nodes. Both nodes and their edges can store additional properties such as key-value pairs. The strength of a graph database is in traversing the connections between the nodes. They generally require all data to fit on one machine, limiting their scalability. Examples include: Allegro, Neo4J, InfiniteGraph, OrientDB, Virtuoso, Stardog, Sesame.

(c) **Column stores** – Relational databases store all the data in a particular table’s rows together on-disk, making retrieval of a particular row fast. Column-family databases generally serialize all the values of a particular column together on-disk, which makes retrieval of a large amount of a specific attribute fast. This approach lends itself well to aggregate queries and analytics scenarios where you might run range queries over a specific field. Examples include: Accumulo, Cassandra, Druid, HBase, Vertica.

(d) **Document stores** – These databases store records as “documents” where a document can generally be thought as a grouping of key-value pairs (it has nothing to do with storing actual documents such as a Word document). Keys are always strings, and values can be stored as strings, numeric, booleans, arrays, and other nested key-value pairs. Values can be nested to arbitrary depths. In a document database, each document carries its own schema — unlike an RDBMS, in which every row in a given table must have the same columns. Examples: Lotus Notes, Clusterpoint, Apache CouchDB, Couchbase, MarkLogic, MongoDB, OrientDB, Qizx, Cloudant, Azure DocumentDB.
NoSQL databases are increasingly used in big data and real-time web applications. They became popular with the introduction of the web, when databases went from a max of a few hundred users on an internal company application to thousands or millions of users on a web application. NoSQL systems are also called “Not only SQL” to emphasize that they may also support SQL-like query languages. Many NoSQL stores compromise consistency in favor of availability and partition tolerance. Some reasons that block adoption of NoSQL stores include the use of low-level query languages, the lack of standardized interfaces, and huge investments in existing SQL. Also, most NoSQL stores lack true ACID transactions or only support transactions in certain circumstances and at certain levels. Finally, RDBMS’s are usually much simpler to use as they have GUI’s where many NoSQL solution use a command-line interface [37].

NoSQL databases Motivations regarding web applications:

- Simplicity of design

- Better “horizontal” scaling to clusters of machines - NoSQL databases automatically spread data across servers without requiring application changes (auto-sharding), meaning that they natively and automatically spread data across an arbitrary number of servers

- Finer control over availability

- Schema-on-read instead of schema-on-write [38]

- Speed

- Cost
RDBMS’s suffer from no horizontal scaling for high transaction loads (millions of read-writes), while NoSQL databases solve high transaction loads but at the cost of data integrity and joins. The bottom line for using a NoSQL solution is if you have an OLTP (On-line Transaction Processing) application that has thousands of users and has a very large database requiring a scale-out solution and/or is using JSON data, in particular if this JSON data has various structures. You also get the benefit of high availability as NoSQL solutions store multiple copies of the data. Just keep in mind that to achieve better performance, you may sacrifice data consistency, as well as the ability to join data, use SQL, and to do quick mass updates [39]. Once Xporter for JIRA Cloud could be view (at a small scale) as a OLTP application that needs to respond to hundreds of clients it must ensure data availability. For example if 500 clients are performing an export, the time for each client to acquire the template file and the scheme configuration must be minimal and relational model is lack of scalability when dealing with OLTP applications, so it is better to choose a Non-relational database. Among the non relational databases we have choose MongoDB.

MongoDB is an open-source database developed by MongoDB, Inc. It stores data in JSON-like documents that can vary in structure. As stated before for non relational databases, MongoDB uses dynamic schemas, meaning that its possible to create records without first defining the structure (scheme less). The choice of MongoDB as the database was made taking into account factors such as: fast developing, ease of scale horizontally (providing new levels of availability and scalability), natural mapping to object-oriented programming languages (like Javascript, that will be used in Xporter for JIRA Cloud development) and the massive community behind it. [40, 41].

2.5.3 Distributed File Systems

As it is known, in JIRA cloud the Atlassian connect plugins are responsible for hosting the users data related to the plugin. So Xporter for JIRA will need to save for each client all the administration data besides the templates uploaded or installed from the Xporter for JIRA template store. Regarding this, Xporter for JIRA Cloud must have/use some storage system to manage all the data and make it available to clients the best way possible.

With cheap storage, fast Internet access and the exponential increase of the amount of data captured, stored and disseminated in electronic, a large group of users and organizations have adopted cloud storage services. It is now possible to store data in cloud at reasonable prices without the limitation of capacity be closely tied to the computing facilities that happen in traditional storage. This kind of services come from several domains, ranging from those deployed in corporate data-centers to applications in research and typically they operate in an environment with local high-end storage services in the form of SAN (Storage Area Network) or NAS (Network Attached Storage) services, which are not easily replaced by public cloud alternatives [42].
As an alternative to cloud storage services exists a different kind of storage technology, a distributed file system with parallel access and fault tolerant features. With such a system, local storage resources as well as resources at remote locations are joined to support concurrent access by multiple clients on copies of the same data. Additionally, it continues to serve data in case of failure of any type of system component assuming that the system has no single point of failure.

The most important high level requirements for distributed file systems that we have identified are:

- Scalable
- Durability
- Generic Interfaces
- High Availability
- Cost Effective

Some examples of existing products and technologies that fulfills the requirements above are [43]:

- **Amazon EFS** - Amazon Elastic File System (Amazon EFS) is a file storage service for Amazon Elastic Compute Cloud (Amazon EC2) instances. It is easy to use and provides a simple interface that allows to quick and easy create and configure file systems. Capacity is elastic, growing and shrinking automatically as files are added or removed. It supports Network File System version 4 (NFSv4) protocol therefore the majority of applications work seamlessly with Amazon EFS. Amazon EFS file system can be used at the same time for several EC2 instances, providing a common data source for workloads and applications running on more than one instance [44].

- **XtreemFS** - XtreemFS is a general purpose storage system and covers most storage needs in a single deployment [45]. It is open-source, requires no special hardware or kernel modules, and can be mounted on Linux, Windows and OS X. XtreemFS is easy to setup and administer, and requires fewer man storage systems. XtreemFS provides several features like File Replication, Read-Only Replication, Elasticity/Scalability, Asynchronous MRC Backups, Asynchronous File System Snapshots, POSIX Compatibility among others. This are achieved by three main components, namely the Directory Service (DIR), the Metadata and Replica Catalog(s) (MRC) and the Object Storage Device(s) (OSD). These components work together to provide the system to network users, and the various components communicate between each other and with the client [46, 47].

- **GlusterFS** - Gluster is a set of open source solutions, it can run on commodity hardware. Gluster uses consistent hashing algorithm and decentralized cluster to enable linear scale-out as much as possible. Gluster’s file system (GlusterFS) is a scalable network file system suitable for data-intensive tasks such as cloud storage and media streaming. It relies on consistent hashing to distribute the data based on a file name. The data is stored on a disk using native formats of a backend storage node. One of the interesting features of GlusterFS is that metadata management is fully distributed, therefore there is no special nodes for metadata management. GlusterFS provides POSIX semantics to its client nodes, is free and open source software and can utilize commonly hardware [48].
- **CephFS** - Ceph is a free software storage platform that stores data on a single distributed computer cluster, designed to be POSIX-compatible, highly distributed without a single point of failure and scalable to the exabyte level [49]. It provides interfaces for object, block and file-level storage (CephFS). CephFS runs over the same object storage system that provides object storage and block device interfaces. The mapping of directories and file names of the file system into objects stored within RADOS clusters is made by Ceph metadata server which can expand or contract. Also, it can rebalance the file system dynamically to distribute data evenly among cluster hosts ensuring high performance and prevents heavy loads [50, 51].

It might be a challenge to decide among these products which one will better fit Xporter for JIRA Cloud needs. However, despite all the benefits that such a system would bring, there are some drawbacks in using this approach, that must be specified. The use of a distributed file system would require: maintaining of one more system, implement the securing, replication and synchronization mechanisms, manage how the system scales, among others. For that reasons, and given that usually the files that have to be stored do not exceed 25MB, we have tried MongoDB GridFS, that is a specification for storing and retrieving files inside MongoDB. Any file being stored with GridFS is chopped into 255KB chunks, those chunks are saved in a bucket called fs and a collection in that bucket, fs.chunks. The files’s metadata is stored in another collection in the same bucket, fs.files. Though you can have more buckets with different names in the same database, using an index makes retrieving the chunks quick. However, chunking with GridFS and the fact that it is done by the driver (NodeJS driver in our case) also means that large operations like replacing an entire file within GridFS are not atomic and there’s no built-in versioning to fall back. For that reasons, as a workaround, it’s necessary to store multiple versions of each file, specify the current version of the file in the metadata and after uploading the new version of one file, in an atomic way, update the metadata field that indicates that it is indeed the latest version, finally remove the older versions of the file [52].

We have tried this approach to store the files and all is working really fast, in fact, we are even surprised by the high performance that this mechanism has revealed. Besides that, now that the files are "living" inside the database we have saved time and avoid concerns in things like replication, synchronization, accesses and scaling because all of that is handle by mongoddb.
2.6 Tests and Evaluation

A very important part of software development is the definition of a set of tests which are applied at different stages of the product lifecycle to guarantee the quality, the consistency and the reliability of the product, to make sure of customer's satisfaction, to reduce maintenance cost and to ensure that the application should not result into any failures because it can be very expensive in later stages of the development [53, 54]. In the development of Xporter for JIRA Cloud the following tests will be implemented:

- **Development Testing** - Development testing is the consistent application of software testing practices that involves synchronized application of defect prevention and detection strategies in order to reduce software development risks, time, and costs. To achieve that, in Xporter for JIRA development will be used tools like CheckStyle, Crucible and SonarQube and besides that all developers will use the same Coding Conventions.

- **Continuous Testing** - Continuous testing is the process of executing automated tests as part of the software delivery cycle to obtain immediate feedback of the global product status. In Xporter for JIRA this is achieved by having a set of expected exports and comparing them with new exports that should produce the same results.

- **Unit Testing** - Unit testing also known as component testing is a software testing method that verify the functionality of individual units of source code, usually at the function level. Xporter for JIRA will have a module of unit tests for assuring the proper functioning of main features.

- **Regression and Non-Regression Testing** - The intent of regression testing is to assure that in the process of fixing a defect, no existing functionality has been broken. Non-regression testing is performed to test that an intentional change has had the desired effect. In Xporter for JIRA, this is achieved in two distinct ways: by having in the issue workflow a testing stage where the QA team will be responsible for verifying the issue and by having a set of expected exports and comparing them with new exports that should produce the same results (also used to assure Continuous Testing).

- **Integration Testing** - The main function or goal of Integration testing is to test the interfaces between the units/modules. Regarding Xporter for JIRA unit tests will be set up to verify the correct operation of all the modules as well as integration with other plugins.
• **Web Testing** - Web testing is the name given to software testing that focuses on web applications and it can help address issues before the system is revealed to the public. In Xporter for JIRA, this kind of testing is achieved by using Selenium tool [55] that simulate events (like a click) and analyse the generated html code with the expected [56].

### 2.7 Process and Managements

The development of Xporter for JIRA Cloud will be planned and managed using JIRA as Issue Tracking tool and ticketing system. All documentation is created/updated at the confluence (Atlassian tool) and will be available to the public. Nowadays it is crucial to develop software over a version control system, in case of Xporter, it will be used bitbucket which is also a tool owned by Atlassian. During the development it will be adopted as a development methodology the XPAGILE a methodology on agile principles [57] and simultaneously mapped with CMMI level 2. Bamboo (Atlassian tool) will be used as continuous integration tool to automate the release management creating a continuous delivery pipeline.
Chapter 3

Implementation

In this chapter will be explained the final implementation and architecture of Xporter for JIRA Cloud as well as all the components and their respective interactions.

3.1 Idea

The idea of the implementation to be made is to develop a web application that will integrate within another service/platform (that could also be another web application) providing that way a set of new features to that service/platform. This web application should try to be as most independent as it can, regarding the platform it is extending, but at the same time, should, as much as possible, try to seem like part of it, giving the feeling to the users of that system that there is only a single service/platform being used. Both, the web application and the service/platform to be extended, should communicate with each other, in a secure way, using for example authenticated REST APIs, and should run in complete isolation from each other. So, the platform being extended should behave like a multi-process architecture that allows partitioning itself into components that will be handled by independent sandboxed apps.

3.2 Conceptual Implementation

As explained before Xporter for JIRA Cloud has been implemented as an Atlassian connect add-on. For that reason we have used Atlassian Connect Express, a framework provided by Atlassian to help in the development process using NodeJS. Node.js, also called Node, is a Java-Script run-time platform. It’s based on Google’s runtime implementation (V8 engine). V8 and Node are mostly implemented in C and C++, focusing on performance and low memory consumption. Whereas V8 supports mainly Java-Script in the browser (most notably, Google Chrome), Node aims to support long-running server processes. Unlike in most other modern environments, a Node process doesn’t rely on multithreading to support concurrent execution of business logic. It’s based on an asynchronous I/O eventing model, shown in Fig. 3.1 [58, 59]. We can think in Node server process as a single-threaded daemon that embeds the Java-Script engine to support customization.
At the beginning we have split Xporter for JIRA Server in modules to understand what could be used in Cloud version or what needed to be adapted before we can use in our connect add-on. The modules we considered were:

- Template Engine (Views)
- Document Engine
- Data Persistence
- JIRA Events Handler
- JIRA Data Access

Given these modules, we quickly conclude that the document engine module could be used in the
cloud version as an external service. For that, major changes were needed in the documents generation module because it was very dependent on other modules, especially in the module responsible for getting data from JIRA (using the java API from Plugins Framework). Considering these needs, a huge refactoring was made so that we could have a fully independent module responsible for generating documents that could be used in both server and cloud versions thus facilitating maintenance and coherence of the add-on on both platforms.

The remaining modules would have to be developed from scratch due to considerable differences that the new architecture and the Connect framework would bring. For example, now all the add-on data needs to be stored and managed outside of JIRA, all add-on screens must be previously rendered and then sent to iframes presented to the end user and all the data must be retrieve by the REST API because there’s no more a java API for that.

Let’s check again the architecture of connect add-ons:

![Connect Add-ons Architecture](image)

**Figure 3.3: Connect Add-ons Architecture**

An Atlassian Connect add-on is essentially a multi-tenanted web application that operates remotely over HTTP in a Sandboxing environment inside Atlassian cloud instances that can be written in any programming language and web framework.

**Ultimately they have three major capabilities:**

1. Add content in defined Atlassian application UI places (inside iframes).
2. Using Atlassian REST APIs to access data.
3. Listen and respond to WebHooks (events) fired by the Atlassian application.
As its verified, without the Java API, all data accesses are made using calls through REST APIs. This means that a mechanism for authentication has to be used. Atlassian Connect uses a technology called JWT (JSON Web Token) to authenticate add-ons [60, 61]. Basically a security context is exchanged when the add-on is installed, and this context is used to create and validate JWT tokens, embedded in API calls. The use of JWT tokens guarantees that:

- Atlassian application can verify it is talking to the add-on, and vice versa (authenticity).
- None of the query parameters of the HTTP request, nor the path (excluding the context path), nor the HTTP method, were altered in transit (integrity).

In that stage, we already know what were the major differences between the plugins type-2 and the connect add-ons, the stack and frameworks that will be used in the development and the newly required components to get it working: the MongoDB database and a mechanism where Xporter templates and generated reports could be saved to. Although, were missing some basic concepts: How can the add-on be installed? Is there any way to running it in a local development environment? if not what are the alternatives? Looking into Atlassian documentation we can find all the answers to that questions and more.

At first, to be installed, an add-on needs to declare itself with a descriptor. That descriptor is a JSON file that tells the Atlassian application about the add-on: where it is hosted, which modules it intends to use, and which scopes it will need. The scopes is a concept that only exists in atlassian cloud instances and the main goal of it is to improve security by forcing the add-ons declaring what type of access they need from the Atlassian application what will define which REST API resources the add-on can use, giving the possibility to atlassian cloud application administrators to accept or decline those accesses. Regarding local development, Atlassian makes it really easy for developers. At beginning it was only possible by using the Atlassian SDK to create a local cloud instance, running the add-on and tunneling it using for example ngrok, which works fine but both add-on and cloud instances were running on the same machine [62]. Recently, Atlassian makes it even easier, with just three steps the development setup is complete [63]:

1. **Getting a development version of JIRA and Confluence** - Atlassian provides free 5 users tier cloud instances for developers.

2. **Enable development mode** - In manage add-ons page, simply check the *Enable Development Mode* option.

3. **Install an add-on** - Once we are using the Atlassian Connect Express this step is accomplish by simply specifying the URL, username and password from the JIRA application created in step 1 in a configuration file of our add-on (credentials.json in newer versions of ACE) and it will try to install itself in that instances.
With this new approach, it’s possible to develop in a way that almost simulates production environment. Summing up, to begin the development of Xporter for JIRA cloud we have to prepare the development environment mentioned above, create the project with its descriptor and from there, develop the business logic.

### 3.3 Technical Implementation

We have started by creating the project skeleton with the help of ACE (Atlassian Connect Express). ACE provides a library of middleware and convenience helpers that make it easier to build Atlassian add-ons, specially:

- An optimized develop loop by handling registration and deregistration on the target Atlassian application at start-up and shutdown.
- A file system watcher that detects changes to atlassian-connect.json (add-on descriptor). When changes are detected, the add-on is re-registered with the host(s).
- Automatic JWT (JSON Web Token) authentication of inbound requests as well as JWT signing for outbound requests back to the host.
- Automatic persistence of host details (client key, host public key, host base URL, etc.)
- Local tunnel'd server (using ngrok) for testing with Cloud instances.

So, first we have to assure that we have installed NodeJS and NPM (Node Package Manager), after that you need to install atlas-connect (ACE client tool), create the project skeleton using it and install the dependencies. That is achieved by running the following commands on console:

```bash
1 npm i -g atlas-connect
2 atlas-connect new <project_name>
3 npm install
```

After that, Xporter for JIRA Cloud skeleton is complete. For testing, it just needs to add the development instance credentials in `credentials.json` file. That is only possible because ACE, behind the scenes, is using ngrok to provide local tunneling to our server.

In Fig. 3.4 is shown the project structure created by atlas-connect for the development of Connect add-ons. The most important components are signed and are responsible for:

1. The `node_modules` folder contains all the server-side dependencies of our project, they’re managed by NPM.
2. The `public` folder is where are placed the project client-side dependencies. To manage this dependencies we will use Bower, a dependency manager tool for front-end components.
3. The **routes** folder is where all the files which define behavior related to the add-on routes should be placed.

4. The **Views** folder is where all the template engine files should be placed. For Xporter for JIRA Cloud we decide to keep the default template engine, handlebars.

5. The **app.js** file is responsible to require some dependencies, create the express application, initialize and configure the components and to start the server were the application will run.

6. The **atlassian-connect.json** file is the add-on descriptor, previously mentioned.

7. The **config.json** file contains the configuration for each run-time environment the plugin runs in.

8. The **credentials.json** file is where are specified the authentication data and URL of target instances so that ACE can automatically register the add-on.

9. The **package.json** file holds several relevant metadata to the project as well as the list of server-side dependencies.

![Figure 3.4: Connect Add-ons Structure](image-url)
After understanding all that concepts, Xporter for JIRA Cloud development has started. In `atlassian-connect.json` were defined all the JIRA UI modules that Xporter needed and defined the routes that will be responsible to handle each one. This routes will receive the Webhooks containing information about the JIRA instance and have the job of rendering the module accordingly. To do that, the routes firstly verify the license and the JWT from the request using `atlassian-connect-express` middleware (JWT must be authenticated and authorized), then if they need additional data they can use the REST APIs to obtain data from JIRA or query the mongodb database to obtain add-on related data. After that, they pass the required context data to a template engine file and finally that page is rendered and sent back as the request response.

All the views must import the Atlassian Connect JavaScript client library that establishes the cross-domain messaging bridge with parent iframes and provides several methods and objects that could be used in views without making a trip back to the add-on server. Besides that kind of requests (from JIRA instance to the add-on) they had to be established new routes to handle requests from the iframe back to the add-on. The behaviour is almost the same, except that in this case the JWT does not exists in the request once it does not come from the JIRA instance. Luckily, ACE generates the JWT as a context variable and place it in our views (client-side) which the add-on can verify later using the middleware (server-side). The requests made back to the add-on are needed when a view needs additional data or for example in case of Xporter for JIRA Cloud, to send requests that will trigger an exportation or an upload/download of a template.

As stated before, the Document Engine module from Xporter for JIRA Server was re-factored to be used now as a service of Xporter for JIRA Cloud. It is running in a tomcat server and the logic is written in JAVA. Besides that, because of time restrictions and business/logistical reasons from Xpand IT, in the first versions, instead of creating our own MongoDB cluster we’re using MongoDB Atlas (mongodb as a service). The final Xporter for JIRA Cloud data flow is summarized in Fig. 3.5.
3.3.1 Project Structure

After declaring all the intended modules to be extended in the add-on descriptor, defined all the routes, implemented all the business logic and solved other required features like: logging, interface internationalization (using 118n module from NPM) and metric reports, the first version of Xporter for JIRA Cloud was completed.

**Descriptor** *(atlassian-connect.json)*

In this file, as stated before, are defined add-on related information, the scopes, the modules to be extended and conditions (for example to decide if some module should or not be displayed). For our add-on were defined the following modules:
- **Web Sections**: Administration Section
- **General Pages**: Bulk Export Details, Bulk Export Results
- **Administration Pages**: Global Settings, Manage Templates, Template Store, Permission Scheme, Manage Servers
- **Web Panels**: Single Issue
- **JIRA Search Request Views**: Bulk Export Details Redirect
- **JIRA Project Admin Tab Panels**: Xporter Manage Project
- **Web Items**: Xporter Header View, Xporter Documentation Link, Xporter Pop-ups
- **JIRA Workflow Post Functions**: Xporter Create Document Post Function
- **JIRA Entity Properties**: Xporter Web Panel Enable Property

Figure 3.6: Xporter for JIRA Cloud - Descriptor
Models

In Xporter for JIRA Cloud models folder are kept the files responsible for retrieving data from MongoDB. These files are mostly used in the routes logic, to obtain data that are required in the UI or to be passed in the Document Webengine rest calls.
Routes

The **routes folder** are kept the files responsible to handle the Webhooks from JIRA instances and the requests made from the iframe back to the add-on. Every single route is secured by using the ACE middleware to handle the JWT verification. The routes logic uses the **routes/services** and the **models** files to respond properly to the requests.

![Figure 3.8: Xporter for JIRA Cloud - Routes](image-url)
Chapter 4

Infrastructure Setup & Deployment

In order to make Xporter for JIRA Cloud available to the public, it needs to be running in a scalable environment that ensures: high availability, fault-tolerance and that can handle a lot of requests maintaining great performance. To ensure high availability and fault tolerance the add-on should be running in several nodes, completely isolated from each other and preferably in different geographic locations. Should also exist a mechanism responsible for checking the nodes status, and in the case of one of them being down or answering incorrectly, immediately start another. That way the architecture could survive to catastrophes that affect only one location and to errors that affect only some nodes. Good performance should be achieved by dynamically allocate resources as they are required and release them after, for example, launching more nodes to answering peak load, and after that remove the additionally created nodes [64].

The first major decision was to choose what would be the best computing cloud platform to implement the Xporter architecture. Despite the difficulty to make a comparison between cloud computing platforms due to the constantly arise of new services and changes in costs and performance, we can reduce our options to the three major existing platforms: Amazon Web Services (AWS), Google Cloud Platform (GCP) and Microsoft’s Azure. Considering price, performance, and support, these three options are very similar. Due to the wide range of services (which may be useful in the future), platform maturity and the interest of our company to have a project running using their services, the choice was AWS.

4.1 Overview

After reading Chapters 2 and 3, the target goal for Xporter for JIRA Cloud was defined, which leads us to the architecture presented in Fig. 4.1. The design decisions of this architecture were made to guarantee good horizontal scalability. In a horizontal scaling model, instead of increasing the capacity of each individual actor (vertical scaling) in the system, we simply add more actors to the system. One of the reasons for that choice relates to the database that will be used in Xporter for JIRA Cloud. As concluded earlier in section 2.5.2, it will be used MongoDB, which provides auto-sharding for horizontal scale out. Also, in MongoDB, native replication and automatic leader election supports high availability across...
racks and data centers, also it makes extensive use of RAM, providing in-memory speed. Xporter for JIRA will receive a lot of concurrent requests that need to be quickly answered and for that is required to distribute them into different cores and even servers. Horizontal scale fit better that scenario than vertical scaling because, in general, adding more nodes in the cluster to achieve certain performance is easier and cheaper than improving the capabilities of one node to achieve that same performance.

Figure 4.1: Xporter for JIRA Cloud Architecture.
4.2 Components

All the components/resources will be attached to a dedicated virtual network in AWS. A virtual private cloud (VPC) is a virtual network dedicate within AWS [65]. It is logically isolated from other virtual networks in the AWS cloud and it could also be viewed like a networking layer for Amazon EC2. The final Xporter for JIRA Cloud Virtual Private Cloud is composed by several elements that together guarantee high availability, resources scaling, access restrictions and great performance.

1. VPC Internet Gateway (1) [66]
2. NAT Gateway (1) [67]
3. Elastic Load Balancer (2) [68]
4. EC2 (Elastic Cloud Compute) (>3) [69]
5. Availability Zone (2) [70]
6. Subnets (6) [71]
7. Auto-Scaling Group (2) [72]

VPC Internet Gateway: An Internet gateway is a horizontally scaled, redundant, and highly available VPC component that allows communication between instances in the VPC with the Internet. It serves two purposes:

- Providing a target in the VPC route tables for Internet-routable traffic.
- Performing network address translation (NAT) for instances that have been assigned public IP addresses.

NAT Gateway: Used to enable instances in a private subnet to connect to the Internet or other AWS services, but prevent the Internet from initiating a connection with those instances.

Elastic Load Balancer: Responsible to automatically distribute incoming application traffic, based on application or network level information, across multiple Amazon EC2 instances. It allows to achieve fault tolerance in applications, seamlessly providing the required amount of load balancing capacity needed to route application traffic. The Load Balancer also monitors the health of its registered instances and ensures that it routes traffic only to healthy instances. When the Load Balancer detects an unhealthy instance, it stops routing traffic to that instance, and then resumes routing traffic to that instance when it detects that the instance is healthy again. As such, this helps to ensure high performance and fault-tolerance.

Elastic Cloud Compute: A Web-based service that allows business subscribers to run application programs with resizable compute capacity in the cloud. The EC2 can serve as a practically unlimited set of virtual machines.
Available Zone: Amazon cloud computing resources are housed in highly available data center facilities. Data center locations are called regions. Each region contains multiple distinct locations called Availability Zones. Availability Zone are engineered to be isolated from failures in other Availability Zones, and to provide inexpensive, low-latency network connectivity to other zones in the same region. By launching instances in separate Availability Zones, applications are protected from failure of a single location, as it was done in Xporter for JIRA Cloud infrastructure.

Subnet: A subnet is a range of IP addresses in VPC. It is possible to launch AWS resources into a specific subnet. Usually a public subnet is used for resources that must be connected to the Internet, and a private subnet for resources that won’t be connected to the Internet.

Auto-Scaling Group: Auto-Scaling Groups helps maintaining application availability and allows scaling EC2 capacity up or down automatically according to conditions you define. They help ensuring that are running the desired number of Amazon EC2 instances as well as automatically increase the number of Amazon EC2 instances during demand spikes to maintain performance and decrease capacity during lulls to reduce costs.

The data flow in the architecture could be detailed in the following steps:

- **Handle Incoming Requests**: In Xporter for JIRA VPC the access to Internet from instances is made using the Internet Gateway. The incoming traffic is handle by an Internet-Facing Classic Elastic Load Balancer (the public one) [73]. The DNS name of an Internet-facing load balancer is publicly resolvable to the public IP addresses of the nodes, therefore, Internet-facing load balancers can route requests from clients over the Internet to the most suitable EC2 instance based on CPU, memory and network informations. The public ELB needs to have Cross-Zone Load Balancing property enabled in order to distribute traffic to the node-private-subnet EC2 instances in any Availability zone. For that, it also needs to belong to each public subnet of each availability zone to properly route out the requests to the Internet Gateway. This public ELB has associated a trusted certificate (to decrypt the requests from Atlassian instances before route the requests to the instances), a security group that defines that will be only accepted TCP incoming traffic in port 443 (SSL) and a Health Check policy to ensure that all the registered instances are available.
• **Requests Processing**: After a request is routed to one of the *node-private-subnet* EC2 instances (Xporter for JIRA Cloud NodeJS node) it is processed by Xporter for JIRA Cloud NodeJS Application. Those instances have a Security Group that restricts the inbound requests to only accept TCP requests to port 3000 and SSH requests on port 22 (to access the instances) and are associated to an Auto Scaling Group that is responsible to start or delete instances based on CPU and memory metrics. In the processing stage, additional requests could be made to the MongoDB database, the Atlassian cloud instance that issued the request or to one of the *webengine-private-subnet* EC2 instances (Xporter for JIRA Cloud Document Engine). For the first two cases, the request response is sent back using the NAT gateway. When a *node-private-subnet* EC2 instance makes a request to the Document Engine the request target is an Internal load balancer [74]. The DNS name of an internal load balancer is publicly resolvable to the private IP addresses of the nodes, therefore, internal load balancers can only route requests from clients with access to the VPC for the load balancer. This internal load balancer will then route the request to the most suitable *webengine-private-subnet* EC2 instance based on CPU, memory and network informations. While processing the request, additional requests could also be made to the MongoDB database, the Atlassian cloud instance that issued the request or to the *node-private-subnet* EC2 instance (as explained before the two first request must use the NAT gateway). *Webengine-private-subnet* EC2 instances have a Security Group that restricts the inbound requests to only accept TCP requests to port 8080 and SSH requests on port 22 (to access the instances) and as *node-private-subnet* EC2 instance, are also associated to an Auto Scaling Group that is responsible to start or delete instances based on CPU and memory metrics.
Chapter 5

Results

In this Chapter will be presented the main business problems/concerns in Xporter for JIRA Cloud development as well as how they were achieved/resolved/proven. For each one of this concerns will be covered the corresponding tests that were performed, the tools that were used to do it and the accuracy metrics used.

5.1 Evaluation Questions

Xporter for JIRA Cloud was designed to create a Xporter for JIRA version that was compatible with the Atlassian JIRA Cloud application, assuring the same available features of its Server version. Given the major differences regarding architecture and development resources between these two versions (Server and Cloud) it was predictable that would arise some problems/questions that would need to be analyzed and thought.

The major problems are related to the fact that the cloud add-ons are not running in the same server as JIRA applications. For that reason, an environment had to be built to run the add-on. That environment needs to be scalable (to handle big amount of concurrent requests), secure (will handle sensitive data) and ensure performance. All add-on data have now to be saved in its own database, therefore, all the concerns related to it, like data backup, data replication and accesses permissions have now to be considered. Since add-ons and JIRA applications are now running on separated machines, all the data must be accessed using REST APIs instead Java API like in server add-ons, this brings security and performance constrains because data must be transferred over network from one server to another, what besides taking more time could also be subjected of forging attempts, that includes the extended UI modules that needs to be rendered fast to keep the user attention. In view of these concerns, emerge questions such as:
1. In average, how long users need to wait in order to get their modules loaded (page load timing)? Is the amount of time acceptable?

2. What happens in case of peak workload conditions caused by simultaneous exportations? Is the performance affected? How much?

3. How can new features be validated? How could be ensured that they don’t break old functionalities?

4. What is the application reaction to continuous requests over an extended period of time?

5. Can Xporter for JIRA features be independently testable?

6. How is the database behaviour under normal and peak workload conditions in terms of memory, network and Process CPU?

5.2 Tests

In order to test the loading time of the modules extended by Xporter for JIRA Cloud was used a software analytics tool called *newrelic* [75]. This tool was used to calculate the time difference between the request being made and the page is fully charged. It has also available a dashboard where the results could be consulted and organized by page load time or by throughput. This allows us to have an easy overview about page loading metrics in Xporter for JIRA Cloud and quickly spot potentially problems.

Figure 5.1: Xporter for JIRA Cloud - Page Loading Times
Figure 5.2: Xporter for JIRA Cloud - Page Throughput

Figure 5.3: Xporter for JIRA Cloud - Overall Page Timings
In Fig. 5.1 are presented the average page load times for each route that handles the JIRA extended modules by Xporter for JIRA Cloud. As it is possible to verify the worst cases are the Xporter for JIRA Cloud web-panel and the Xporter for JIRA Cloud manage permissions page. In web panel, it is probably due to the required conditions checks that needs to be performed and in manage permissions page the problem could be related to complex code (what cloud probably be optimized) while resolving the permissions for that client. With a worst case of 3.520ms to completely render a module can be concluded that Xporter for JIRA Cloud achieved with success the task to present the UI as fast as possible to the end users. In Fig. 5.2 and Fig. 5.3 it is also present the Throughput by route and an overall page loading time regarding Web application, Network, DOM processing and Page rendering times.

To ensure that each bug fix or new features were properly implemented, are working as expected and did not affect/break any old functionality was used testing stages in the Xporter for JIRA project workflow. That way, in order to be resolved, each issue had to go through a stage in the workflow where quality team members verified its functionality and ensured that no related feature has been affected.

Figure 5.4: Xporter for JIRA Cloud - Xporter Cloud Project Workflow
As Fig. 5.4 shows, after opening and start progress in a new issue, there are several possible paths to the closed state but all the paths that have passed by the resolved state had to necessarily passed also by the testing one, ensuring that way that no issues were resolved without passing by the QA team.

To properly test the evaluation questions related to exportations performance was used Jmeter, a performance testing tool developed by Apache [76]. The tests were made from a personal computer outside AWS, so all the requests have a considerable latency. Within Jmeter were performed tests that tried to simulate the following scenarios:

1. Normal/expected scenario of exportations.
2. Peak workloads of simultaneous exportations.
3. Big amount of exportations requests over large periods of time.

For each scenario were collected the time to complete of each request as well as metrics from EC2 instance (CPU and network) and from the database cluster (memory, network, process CPU).

**Scenario 1**: To try simulate an expected/normal amount of work were made 900 exportations with 3 seconds of interval between each request.
Figure 5.7: Node EC2 Metrics - Scenario 1

Figure 5.8: Webengine EC2 Metrics - Scenario 1
In this scenario, in terms of response time to the requests, there was an average of 3.8s with the worst case of time consuming of 18s to be completed. It was verified a slight increase in CPU utilization in EC2 instances and also in the instance of MongoDB, where there was a slight increase in the number of active connections.

**Scenario 2:** In order to simulate peak workloads were made 900 exportations divided in 9 groups of concurrent requests.

<table>
<thead>
<tr>
<th>Label</th>
<th># Samples</th>
<th>Average</th>
<th>Median</th>
<th>90% Line</th>
<th>95% Line</th>
<th>99% Line</th>
<th>Min</th>
<th>Max</th>
<th>Error %</th>
<th>Throughput</th>
<th>KB/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Ex.</td>
<td>900</td>
<td>14454</td>
<td>7842</td>
<td>39753</td>
<td>49788</td>
<td>70275</td>
<td>1329</td>
<td>119743</td>
<td>0.00%</td>
<td>1,7sec</td>
<td>366.3</td>
</tr>
<tr>
<td>TOTAL</td>
<td>900</td>
<td>14454</td>
<td>7842</td>
<td>39753</td>
<td>49788</td>
<td>70275</td>
<td>1329</td>
<td>119743</td>
<td>0.00%</td>
<td>1,7sec</td>
<td>366.3</td>
</tr>
</tbody>
</table>
In Scenario 2, the response time to the requests has had an average of 14.4s with a worst case of 120s to be completed. This values are probably related to some connections timeouts that have happened. It was noticed a considerable increase in CPU usage in Webengine and MongoDB instances, where there also was a big increase in the number of active connections and in network usage.
Scenario 3: In order to simulate continuous work were made 2271 exportations divided in groups of 42 concurrent requests.

<table>
<thead>
<tr>
<th>Label</th>
<th># Samples</th>
<th>Average</th>
<th>Median</th>
<th>50% Line</th>
<th>95% Line</th>
<th>99% Line</th>
<th>Min</th>
<th>Max</th>
<th>Error %</th>
<th>Throughput</th>
<th>KB/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bulk Exp</td>
<td>2271</td>
<td>20717</td>
<td>21321</td>
<td>31047</td>
<td>33572</td>
<td>35567</td>
<td>304</td>
<td>43333</td>
<td>7.49%</td>
<td>2.0sec</td>
<td>1.1</td>
</tr>
<tr>
<td>TOTAL</td>
<td>2271</td>
<td>20717</td>
<td>21321</td>
<td>31047</td>
<td>33572</td>
<td>35567</td>
<td>304</td>
<td>43333</td>
<td>7.49%</td>
<td>2.0sec</td>
<td>1.1</td>
</tr>
</tbody>
</table>

Figure 5.14: MongoDB Atlas Metrics - Scenario 2

Figure 5.15: Requests Results - Scenario 3

Figure 5.16: Requests Time Over Time - Scenario 3
Figure 5.17: Node EC2 Metrics - Scenario 3

Figure 5.18: Webengine EC2 Metrics - Scenario 3
In this scenario in terms of response time to the requests, there was an average of 20s with the worst case time consuming 43s to be completed. It was the scenario that caused the biggest increase in CPU and network usage in both EC2 instances and also in MongoDB instance, where there was a huge increase in the network activity and in the number of active connections.
Chapter 6

Conclusions

As originally proposed, the aim of this thesis was to create a version of Xporter for JIRA add-on that could be used in the Cloud versions of JIRA applications. This objective was achieved with success, having been developed an Atlassian Connect add-on (Xporter for JIRA Cloud) which has available almost all the features that the version for JIRA Server currently offers.

The entire development (stated in Section 2.5) was done using Agile methodologies and best practices, also, the project management was done using JIRA itself as an Issue Tracker and all the documentation is publicly available in Xpand IT Confluence application. One of the important aspects that was taken into account during development was to share as much as possible the existing logic, which was achieved by isolating the document generation module, and is now being used by both versions of the add-on as a shared service.

In addition to the development itself, it had also to be designed in AWS the entire infrastructure that supports the add-on, ensuring that it is prepared to be used by multiple clients and respond effectively to large amounts of requests, allowing scalability, high availability, fault tolerance and of course the customer satisfaction.

Despite the difficulties faced during this work, mainly due to the lack of experience in technologies such as NodeJS, MongoDB, and AWS, all the objectives have been fulfilled, the add-on can effectively manage all the required configuration data for each client, quickly displays front-end content to customers, and ensures high performance in the add-on most common task, the report generation. The project is already commercially available to clients through Atlassian Marketplace since 1st of August of 2016.

6.1 Achievements

The add-on has been available to the public first of August of this year and was for two months, in first place in the Cloud Top Add-ons ranking of Atlassian Marketplace, being now in 2nd place. It currently has 157 customers evaluations and sold 19 monthly subscriptions and one annual subscription. Besides that, it already counts with some pleasant reviews in the customer portal.
6.2 Future Work

In the future, it will be set up a MongoDB cluster that will be running on EC2 machines living within the VPC where will also be running the add-on. That way it's not necessary to rely on the MongoDB service (MongoDB Atlas) that is currently being used, achieving thereby greater control over the add-on configuration and clients data as well as a reduction of costs associated with the use of MongoDB Atlas service. Regarding development, will be fixed eventual bugs detected or reported by customers through the customer portal, and new features can be implemented according to new requirements that may arise.
Bibliography


