Development of a Data Migration Automation Tool From Salesforce To Salesforce

Student: Meir Carlos Mouyal Amselem

Defense Date: Thursday 2nd of July 2020

Bachelor Degree in Informatics Engineering,
Software Engineering

Educational Cooperation Agreement
Modality B: Nuvolar Works

— Bachelor Thesis —

Supervisor	Alicia García Romero	*Team Lead*	*Nuvolar Works*
Co-supervisor	Alberto Otero Pascual	*Senior Software Engineer*	*Nuvolar Works*
Tutor	Cristina Gómez Seoane	*Software Engineering College Professor*	*FIB - UPC*

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

ABSTRACT

Companies that work with Salesforce environments tend to operate with multiple environments for training, development, testing or support purposes. Now, in order to perform tasks, it emerges an essential requirement to populate these environments with data.

However Salesforce environments are completely isolated, meaning that they share no data between them. As a consequence, company employees frequently encounter the need of undergoing through complex data migrations to populate their environments with data.

Nonetheless, data migrations are challenging, manual, error-prone and time-consuming tasks that require ample planning and a really good understanding of the Salesforce environments in hand.

As a result, in an effort to make performing a data migration an easy and effortless process, this project goal is to build a simple and intuitive data migration tool compatible between Salesforce environments, capable of automating all the complex steps a data migration encompasses.

RESUMEN

Las empresas que trabajan con entornos de Salesforce tienden a operar con múltiples entornos para fines de capacitación, desarrollo, pruebas o soporte. Ahora, para realizar tareas, surge un requisito esencial para habitar estos entornos con datos.

Sin embargo, los entornos de Salesforce están completamente aislados, lo que significa que no comparten datos entre ellos. Como consecuencia, los empleados de la empresa con frecuencia se encuentran la necesidad de realizar complejas migraciones de datos con el fin de poblar sus entornos con datos.

No obstante, las migraciones de datos son tareas difíciles, manuales, propensas a errores que requieren de mucho tiempo y de una amplia planificación junto a una muy buena comprensión de los entornos de Salesforce en cuestión.

Como resultado, en un esfuerzo por hacer que realizar una migración de datos sea un proceso fácil y sin esfuerzo, el objetivo de este proyecto es construir una herramienta de migración de datos simple e intuitiva compatible entre entornos de Salesforce, capaz de automatizar todos los pasos complejos que abarca una migración de datos.

RESUM

Les empreses que treballen a entorns de Salesforce tendeixen a operar amb múltiples entorns per a fins de capacitació, desenvolupament, proves o suport. Ara, per a fer tasques, sorgeix un requisit essencial per a habitar aquests entorns amb dades.

No obstant això, els entorns de Salesforce estan completament aïllats, la qual cosa significa que no comparteixen dades entre ells. Com a conseqüència, els empleats de l'empresa amb freqüència es troben la necessitat de realitzar complexes migracions de dades amb la finalitat de poblar els seus entorns amb dades.

Tanmateix, les migracions de dades són tasques difícils, manuals, propenses a errors que requereixen de molt de temps i d'una àmplia planificació al costat d'una molt bona comprensió dels entorns de Salesforce en qüestió.

Com a resultat, en un esforç per fer que realitzar una migració de dades sigui un procés fàcil i sense esforç, l'objectiu d'aquest projecte és construir una eina de migració de dades simple i intuïtiva compatible entre entorns de Salesforce, capaç d'automatitzar tots els passos complexos que abasta una migració de dades.

Notation

K after a number is for Thousands

M after a number is for Millions

Acronyms

API Application Programming Interface

Auth short for authorization, referring to the OAuth protocol

CSV Comma Separated Values

DB Database

FDP Final Degree Project

FR Functional Requirement

HTTP HyperText Transfer Protocol

IT Information Technology

POC Proof Of Concept

OAuth Open Authorization

OSX Macintosh Operating System Version 10

QA Quality Assurance

UI User Interface

Index

1. Introduction

This report is the result of my Final Degree Project specialized on Software Engineering in the Bachelor Degree of Informatics Engineering at the Polytechnic University of Catalunya – Barcelona School of Informatics.

As a Bachelor Thesis, this document encompasses all the phases a project of Software Engineering undergoes during its development process, displaying a real world example of the conducted steps performed to achieve the development goal of this project: a data migration automation tool between Salesforce Orgs.

Beginning with the context, this section will be responsible for explaining the core necessary knowledge in order to understand the project's scope. This will be accomplished by introducing the topics of Salesforce and Data Migrations.

Once the fundamental concepts are understood the problem statement, the goals and the project's stakeholders will be formulated. In this manner, there will be a clear understanding about the project's topic, the problem trying to solve, what can be achieved and who can be interested in it.

With a good and solid context set, an analysis of the state of art of the project will proceed, which will encompass a comparison between the advantages and disadvantages of the 3 most popular analogous applications.

Next, there will be a project management chapter responsible for covering the project's planning detailing its life cycle and describing the corresponding methodology and rigor, risk management and economic management.

Consequently, the project scope will be defined, where the objectives of the project will be outlined followed by a complementary background knowledge section and an example scenario in which the project's tool would be of use.

The next chapters will then be responsible to cover in detail the system's specifications, design, implementation and validation testing. This will be done by describing in depth the system's use cases, the architecture approached, the technologies used, the implementation of particular functionalities and the testing strategies employed.

With all this previous information in mind, the subsequent section will be dedicated to sustainability, incorporating a study of the environmental, economic and social impact of the development of this project.

And to finish, the project technical competences accompanied by the conclusions of this project will be discussed, covering topics as the main challenges faced along the project and the future work of the software developed.

Lastly, a list of the references used for this report will be presented.

1.1. Context

The project covered in this thesis is highly specialized in back-end development, therefore in order to understand the project it is fundamental to have a good comprehension of the concepts and the software involved with it.

As a result of the project's complexity, the following sections will present a basic explanation concerning the topics of data migrations, the core subject of this project, and Salesforce, the software for which this tool is developed.

Further along the sections of this report, when necessary, more specific and detailed knowledge will be presented as a means to provide essential background information, either to comprehend the proposed solution of the problem in hand, or the system behavior.

1.1.1. CRM

Finding customers for a business is not easy, and once they are found, a new challenge appears, establishing and maintaining strong relationships with them.

Customer Relationship Management (CRM) is a technology created in order to improve business relationships by providing a smooth and efficient process that helps companies stay connected with their customers [1].

CRM systems function not only by storing the customer's information, but also by keeping track of the customer's activity. For companies that work with a CRM it is easier to build stronger relationships by understanding and meeting their customer needs.

Besides, over the past years CRM systems have proven to have big advantages in customer experience, customer retention and sales, thus, having a CRM has become a key technology for companies to profit [2].

1.1.2. Salesforce

Salesforce is an American company that provides Software as a Service (SaaS) with the main purpose of offering a cloud-based CRM solution [3]. Industry analysts [4] and leading user review organizations [5] have recognized Salesforce as the market leader for CRM technology, becoming the top CRM platform for growing businesses.

Salesforce is also a software that not only helps companies to manage customer relationships, but also allows integration with other systems, and even to build custom apps (offering Platform as a Service as well).

For this reason, when companies build an application they need to make a really important decision that will determine the development path to follow. This decision is to either develop the product with an app builder like Salesforce, or with development frameworks such as Java, Ruby on Rails or many others.

As a consequence, this project focuses on the companies that develop their product on top of Salesforce, as they have a direct interaction with Salesforce environments and are completely dependent on them.

1.1.2.1. Salesforce Architecture 101

Salesforce uses a multi-tenancy architecture, meaning that a number of customers share IT resources, such as database servers, application servers and the infrastructure necessary to run applications, resulting in an environment that can be standardized and optimized for all efficiently and at a lower cost for each 'tenant' as resources are managed centrally. [6]

With this multi-tenant architecture, Salesforce groups around 10,000 customers in what it calls an Instance. An Instance is a self contained unit that contains all that is required to run Salesforce, including the application server, database server, database itself, search and file system.

Inside Instance's there exist different Organizations, which are mostly referred to as Orgs. An Org essentially is just an identifier that represents a person's version of Salesforce, but more importantly, it is the data and metadata itself within an instance, where its objects, classes, pages, sites, records, apps, etc. are stored.

Salesforce customers interact with Orgs, and when a Salesforce Org is used for a specific purpose in the software's creation, it is called an environment, for example, there can be Orgs used for production, development, testing, integration or training purposes, and these are referred to as production environments, development environments, testing environments, etc.

Through this architecture Salesforce also manages to isolate each Org, meaning that NO information is shared between Orgs, they are completely independent from one another.
This implies that every time a developer creates a new Org, perhaps for reasons like needing a new Org for testing purposes, a new developer joining the team, reproducing client errors or even for having accidentally broken his current Org, then once created, the Org comes with no data, as it should, since it is an entire new isolated independent environment.

Figure 1 illustrates in a clear and graphical way the multi-tenant architecture just explained.

DATA CENTRE

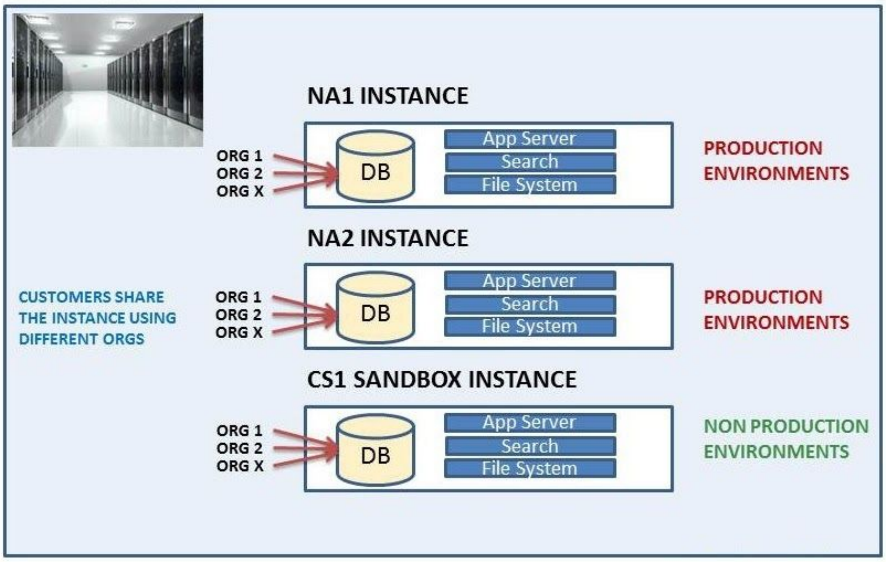


Figure 1 – Salesforce's Multi-tenant Architecture

1.1.2.2. Salesforce Objects

Salesforce Orgs work with relational databases, hence, the data of an Org is represented by its records, which belong to objects. That is to say in Salesforce, objects essentially represent database tables that allow users to store data specific to the Org they belong to. Simply put, in Salesforce terminology, tables are known as objects.

In Salesforce there are 2 main types of objects:

1. Standard Objects – the objects already established within Salesforce, as for example: profiles, users, accounts, contacts, opportunities, products and many others.
2. Custom Objects – the objects created by users to store information specific to their needs.

In addition, the same logic applies to fields, there are standard fields and custom fields. In this way, standard and custom objects can be customized by populating them with new fields.

As usual, fields can come in very different forms, as Salesforce supports a bunch of different data types, such as: text, number, chexboxes, dates, location, formulas, and many more. However over all the Salesforce field supported types, the most relevant one for this project are the relational fields.

Getting in depth about Salesforce relational fields, firstly, it is important to note a key difference in the behavior between the standard relational fields and the custom relational fields, and that is, standard relational fields are allowed to be polymorphic.

As Salesforce defines “a polymorphic field is one where the related object might be one of several different types of objects” [7], put differently, a relational field being polymorphic means that 2 records of the same object can have the same relational field related to different objects, as polymorphic relational fields allow for several different object options.

This feature brings a vital consideration, that is the way to identify to which object a polymorphic relational field belongs to. Salesforce resolves this with object identifiers and record identifiers.

In Salesforce each object is assigned a 3 digit identifier, also known as the object identifier. Correspondingly, each record of an object contains an identifier, which is made of 3 digit prefix, the object identifier, followed by a unique 15 digit identifier. Therefore given a record ID, just by looking at the 3 digit prefix, one can know to which object that record belongs.

Accordingly, relational fields are essentially references, so when a record has a relationship with another record, this implies that the ID of the referred record is stored in the relational field of the record that references it. In this manner, record relationships are only consistent in the Org they live in.

This is true as a result of Salesforce only allowing the storage of reference IDs present in the same Org. In this way, it is impossible for an Org to contain a record with a relationship to a nonexistent record ID of the same Org, even if this record ID exists in another Org, since as explained earlier, Orgs are independent from one another.

1.1.2.3. Objects Restrictions

Complying with Salesforce terminology, metadata is what describes the structure and configurations of objects and their fields, as well as many other sorts of components.

Meanwhile Salesforce does not categorize metadata in groups, in this project I will differentiate between 2 classes of metadata:

1. Internal metadata – is the metadata in the most basic form as it is known in a database schema, like for instance the options of making a field unique or required. Additionally, this class of metadata also includes special behaviors integrated in some of Salesforce standard objects, as for example, the Name field of the Contact object [8].
2. External metadata – is the more complex and diverse metadata, as it is available through very different approaches, such as triggers, workflows, validation rules, processes, duplication rules and lookup filters.

With the use of metadata, Salesforce allows a higher level of control towards the set of rules that govern the database records in an Org, providing its user a variety of different ways of creating restrictions on the Orgs data for the purpose of preserving the Orgs data coherence through personalized sophisticated logic.

1.1.3. Data Migrations

In really simple terms, a data migration is the process of moving data from one location to another. In reality, it is not so simple. For a successful data migration to take place, a great comprehension of the system's behavior, limits and constraints and back-end design is required. [9]

The location from which the data is extracted is usually called the *source* or the *origin*. Analogously, the location to which the data is imported is named the *target* or the *destination*.

Also a data migration is a one-time activity; once all the data is imported, the migration task is complete. So it is very important to avoid any potential mistakes that could cascade into having to do the whole process again or even risk jeopardizing sensitive data.

Data migrations are distinguished into 2 categories:

- ❖ Full data migrations – migrations in which all the data included in the source environment is being migrated into the target environment.
- ❖ Partial data migrations – migrations in which not all the data, but a fraction of the source environment's data is being migrated into the target environment.

Full data migrations are usually recurrent in a long term span, as they are heavy migrations that involve the entire system. On the contrary, partial data migrations are likely to be performed regularly, on a short term span, as they tend to involve small chunks of data.

For instance, a full data migration could be carried out for the purpose of moving from an outdated system to a newer system that is designed for modern data needs, enabling transferring all the data of the outdated system into the new system.

Consequently a partial data migration could be performed in the case of a company acquiring a portion of products from another company, therefore transferring the related data to the portion of products bought from the sellers system into the buyers system.

Now, what determines the complexity of a data migration is not the type of migration, neither the quantity of data to transfer, but the quality and configuration of the data together with the target system rules and restrictions. These are the true factors that determine the complexity of the tasks to be performed during a migration.

Moreover, data migrations can also come in very different forms, with different purposes, different requirements, incorporating even more difficulty to the procedure.

Bearing in mind the risk of significant data repercussions through a vaguely or badly performed data migrations, along with the immense amount of considerations to contemplate during the process, data migrations have the reputation of being complex, challenging, error-prone, time-consuming, manual and tedious tasks that demand ample planning along with a superb understanding of the current system.

Accordingly, as it would result in a waste of time for a company to constantly have to dedicate large amounts of time on performing long manual data migration processes, when in need of undergoing frequent data migrations, it is very reasonable to reach towards some sort of software with the capability of simplifying the process of achieving a successful migration.

At the same time, in order to ensure the success of complex migrations, dedicating a generous amount of time and effort to completing this process manually is completely understandable. However the use of tools to ease the process of certain tasks or the automation of some steps is key for a better experience and use of resources.

1.2. Problem Statement

Companies that work in Salesforce project's tend to operate with different Salesforce environments for development, testing, integration or training purposes. And in order to perform tasks, there is an essential requirement to populate the environment with data.

Conversely, thanks to the previously presented context, we now understand the complexity of this requirement, as we now know that Orgs are isolated independent environments that share no data between each other, and that when created, contain no data.

Another fact we know is that the process of migrating data is a challenging and time-consuming task that demands ample planning, manual work and a really good understanding of the system to get the data to be imported correctly.

Therefore, as stated earlier, dedicating time and effort to a data migration only makes sense when it is not a process that needs to be repeated regularly.

On the other hand, *the user need of working on multiple Salesforce environments along with the difficulty to repeatedly, manually move data between Orgs* creates an opportunity for the automation of this task, the automation of transferring data between Salesforce Orgs.

1.3. Goal & Real Scope

1.3.1. Goal

As a result of the immense amount of topics and possibilities the subject of data migration covers, the goal of this project is to implement a Minimum Viable Product (MVP) of a partial data migration tool that automates the process of transferring data from a Source Salesforce Org to a Target Salesforce Org.

In other words, what is desired to obtain from this project is to have a simple and intuitive tool that allows for a quick and easy way of transferring data between Orgs with the least amount of knowledge necessary.

In such a manner, transferring data between Salesforce Orgs will become an easier, less manual, time-consuming and error-prone task, removing the concerns related to conducting data migrations.

1.3.2. Real Scope

As just stated, the goal of this product is to have an MVP of a data migration tool. Why only an MVP? Because an ETL's* software real scope is so extensive that it would be impossible to develop a complete ETL tool within the few months that this project lasts.

*ETL is short for Extract, Transform and Load, which are the 3 main database operations performed by data migrations tools, hence, ETL software is just another way of referring to data migration tools.

The full scope of a data migration tool could cover numerous functionalities like:

- Several ways of selecting the data to transfer.
- Offer different types of migrations.
- Saving and managing the different data transfers previously done.
- Duplicating entire Orgs, meaning not only migrating the Orgs data but also its entire metadata.
- Support the transfer of data from a Source Org to multiple Target Orgs simultaneously.
- Support the import of external data, being data manually imported from the user.
- offer the update and deletion of records through the user interface.

There are literally an infinite number of use cases that could be carried out for an ETL, as a result, this project involves the core functionalities necessary in order to in future work progressively grow into an extensive ETL.

1.4. Stakeholders

Stakeholders are all the parties that can have an interest in the software being developed in this project. In this report I will distinguish between 2 sorts of stakeholders: internal and external.

Internal stakeholders are the actors who have a direct relationship with the software being developed, and external stakeholders the actors who could be interested in this software, but that do not have any direct relationship with it.

1.4.1. Internal Stakeholders

1.4.1.1. Salesforce Company

Considering that the software developed in this project requires to work with Salesforce software then, without a doubt, Salesforce is an obvious stakeholder because of its direct relationship with the project

For this reason is that Salesforce is a relevant stakeholder, since they can benefit directly from all the advantages this tool provides. In this way they could be interested in buying it or even promoting it by mentioning it through their help forums or other resources so that other clients can benefit from it too.

1.4.1.2. Companies working with Salesforce

Since companies that work with Salesforce use Salesforce, they also have a direct relationship with this project's product, and their interest is clear, since they are the particular group of consumers at which the tool is aimed at.

Additionally, when facing the need of a data migration, they will at least need to consider the use of this tool. Later, if the tool meets their migration requirements, they will work with it.

1.4.1.2.1. Developers

Inside the company, developers are usually the workers assigned to carry out the data migrations, therefore developers are the company employees most interested in this tool. That is because this tool will allow them to do what they do best, to develop, instead of having to do the manual tedious work of comparing, analyzing and planning the transfer of data between Orgs.

1.4.1.2.2. Q/A & Support

Well-structured companies have teams for Q/A and Support so that they can correspondingly test the functionalities of the product and provide support to their clients.

This demands the company with capability of working with different environments, and this means that they have the need of populating the Orgs with data in order to work.

Hence, it is in their interest to have a simple, quick and efficient approach to transfer data to their new or broken Org. In this way they can have a smooth data migration process instead of receiving complex training as a means to capacitate them on performing their own data migrations, or alternatively, delegating the task to a developer.

1.4.2. External Stakeholders

1.4.2.1. Companies not working with Salesforce

Companies looking to build a project while doing some research on the available options or companies looking to switch CRM systems might also be interested in data migration tools that are compatible between Salesforce Orgs, as it would help them in their future development process.

2. State of Art

Because that migrations can be as complex as you may need or want them to be, data migration tools can be planned and developed in many different ways, each with their different functionalities, constraints and limitations.

It is also important to note that there will be no perfect ETL solution that will work for every single project, as different migrations come with different requirements, besides being highly dependent on the Target Orgs internal logic.

The most popular ETL option requested by users is in the form of a cloud-based tool, favoured by its accessibility, simplicity and usability. As this means having a tool easily accessible through any computer, straightforward enough to perform effortless migrations and with a good user experience.

On the contrary, the other forms of solutions come in standalone tools or as Salesforce plugins.

Standalone tools tend to be more complex, sometimes offering Command Line Interface (CLI) solutions allowing their users to have more control and customization over the migration they want to perform.

And lastly, Salesforce plugins or extensions are the least common solution for data migration tools, since while they can offer the system a greater and easier control towards the interaction with an Org, they can be harder to develop, harder to test, and not comfortable to interact with.

In addition, Salesforce ETL tools are usually built for the purpose of operating as:

1. Developer tools

   Advantages – they provide a simpler and more intuitive way of interacting with the Salesforce API. Also offering the users more control and flexibility over the functionalities they can perform.

   Disadvantages – they demand a high understanding of Salesforce and the Orgs structure. They require all the manual control of the import and export of data plus the planning and logic behind it, in other words, they still do not resolve the issue of complex time-consuming plannings and manual migrations.

2. Commercial tools

   Advantages – they offer the bulk import and export of data and they allow for Org backups.

   Disadvantages – they tend to be paid and expensive software in which the cost is pay per use. Additionally, they do not tend to offer support for custom objects.

Having all this in mind, in the following sections I will present, and shortly describe, a few data migration tools that I have specifically filtered and selected for having proved their usefulness and effectiveness over a wide range of use cases to the Salesforce community [10][11].

2.1. Salesforce Data Loader (developer tool)

Salesforce Data Loader is a free client application officially developed by Salesforce for the bulk import and export of data. It is known for its speed in large import or export of records, supports standard and custom objects, has an easy interface, even though not intuitive, and also allows for data updates and deletions.

The negative comments of this tool usually are that it only supports CSV format, needs to be installed locally, which can be annoying for OSX computers, the auto-mapping does not work well, sometimes fields are missing in the UI, and it crashes easily without any feedback of the crash.

2.2. Jitterbit Data Loader (developer tool)

Jitterbit Data Loader is the most used ETL tool for performing Salesforce data migrations and it is available in a free version, even though it has functionality limits. Similar to Salesforce's Data Loader, it allows the bulk of insert, update, upsert, query and delete.

The negative comments of this tool usually are that it is known to be slow and it is not 100% cloud-based, meaning that it requires a local client to be installed.

2.3. Dataloader.io (commercial tool)

Dataloader.io is also a popular integration tool for data migrations with Salesforce, and it is powered by MuleSoft (a Salesforce company). The positive comments about this tool are that it is completely cloud-based, so there are no installations required, that it allows you to save data loading jobs, that you can import and export an unlimited amount of records and that it contains a very user-friendly UI and UX.

The negative comments of this tool usually are that it is a bit slow, it only allows the operations of import, export and delete, and that since its freemium, limits are set by subscription type, in which prices scale really quickly and are measured per month/user.

2.4. Summary

On the following table we can see a comparison of the 3 tools:

Tool	Number of Records you can Import or Export	Platform Compatibility*	Feedback	Speed*	Compatible Formats	Pricing
Salesforce Data Loader	Between 5K and 5M	★★	✗	★★★	*CSV*	Free
Jitterbit Data Loader	*Not specified*	★	✓	★★	*CSV* + connection to databases	Freemium
Dataloader.io	10K/month (free version)	★★★	✓	★★	*CSV* + connection to databases	Varies by plan

*3 stars is the maximum punctuation

These are currently the 3 best reviewed tools by the Salesforce community, however none of these tools preserve the relations between records nor automate the procedure, which is a key feature to ease the process of a migration.

Similarly all these tools are highly dependent on a previous planning done by the user, as the tool is only responsible for performing the operation requested by the User through the Salesforce API and not for going through the entire migration process.

The consequence of this approach is that it relies on the user performing tasks throughout the entire migration process, such as planning and organizing the flow of the migration along with the responsibility of transforming the data and maintaining the Orgs data quality and consistency.

In contrast, what this project tool pursues is the automation of the maximum amount of steps possible so that no technical knowledge, complex planning, heavy thinking or tedious manual work is required to accomplish a successful data migration.

In this way the tool developed in this project will not be providing a solution for the problem of handling specific tasks related to data imports or exports as the previously mentioned tools do, but its goal is putting an end to the unnecessary time and effort required from developers to plan their specific data migration processes.

Additionally, with the purpose of creating a tool with the most benefits, a cloud-based approach was chosen for this project. However in the meantime, as an MVP, the tool is not required to be deployed to the cloud, but to meanwhile work as a standalone tool.

Furthermore, one of the extra advantages of the tool for having been developed with a cloud-based approach, even though not being deployed, is the ability to be accessible through the local network. That is to say, all the users connected to the same local network will be able to access the tool without further installations, operating system inconveniences or compatibility issues.

3. Project Management

This unit is responsible for presenting the initial project planning performed, consisting of the project's life cycle, methodology and rigor, time estimates and gantt diagrams, risk management, resources and economic management.

In addition, the unit will wrap up with a conclusions section with the purpose of describing any experienced deviations on the initial project planning described that took place during the project development.

3.1. Project Life Cycle

Project's Time Span	
Start Date	Estimated End Date
January 20th, 2020	June 15th, 2020

The estimated duration of this project is approximately 5 months, where 100 days will be work days, hence, with 8 hours of work per day, the project will consist of 800 work hours.

Now, in this section I will present the 5 phases in which I have divided this project's life cycle along with a description of the purpose of each phase, its tasks and their corresponding estimated duration.

The following abstract template will be used to describe the tasks involved in each phase:

T{Main Task Number} – Task Topic	**{Task hours}h**	**Task dependencies**
T{Task Number} – Task Name	hours	dependencies
T{Subtask Number} – Subtask Name Have a clear understanding of the context and the problem trying to solve and document them in the report.	hours	dependencies

3.1.1. Learning & Researching

This phase mainly involves the tasks related to acquiring the core knowledge associated with the project. In this way awareness of the project's full scope will be appreciated along with a good comprehension of the project's technologies implicated, its requirements and challenges.

Tasks involved:

T1 – Hello World Salesforce	**74h**	**None**

T1.1 – Introduction to Salesforce Introduction to Salesforce: understand what is Salesforce, what is a CRM and get to know its terminology and functionalities.	4h	None
T1.2 – Salesforce Training Learn how Salesforce works, completing Trailhead tutorials and playing around with the software to get familiar with it.	30h	T1.1
T1.3 – Knowing Salesforce's Back-end Architecture Reading Salesforce official documentation and more Trailhead tutorials in relation to how is data and metadata managed, data modeling and its restriction towards what it allows and forbids.	20h	T1.1, T1.2
T1.4 – Salesforce API's Research about the different ways data and metadata can be extracted from Salesforce Orgs, identify the differences between the APIs, their constraints and compatibility and play around with them.	20h	T1.1, T1.2, T1.3

T2 – Data Migration	**26h**	**None**
T2.1 – Full Scope Read and understand about the full scope of a data migration process.	2h	None
T2.2 – Best Practices Investigate the standard procedure when facing data migrations, the recommended best practices, the possible risks and how to prevent them and how it applies to Salesforce.	5h	T1, T2.1
T2.3 – Main Challenges Identify the main challenges I will need to face in the development of the project when dealing with data migrations and have a good understanding of what is achievable.	4h	T2.2
T2.4 – Analysis of Approaches Read about how other developers have dealt with this problem before, what are the different options available, and think about possible alternative approaches. Also analyze the best approach taking into consideration the goal of the project and its basic requirements, the different ways of interacting with the API and the time and complexity involved.	15h	T2.3

T3 – Proof of Concept	**20h**	**T2**
Specify step by step a proposal solution methodology for the process of data migration in order to demonstrate and verify the feasibility of the project goal with the time available and test this proposal manually.		

T4 – Previous FDP's	**8h**	**None**

Review the index, content, structure and format of previous FDP's to have a better idea of what is expected of this project.

3.1.2. Project Vision: Conception & Initiation Phase

The purpose of this phase is to have a clear conception of what would be desired for the project to cover. It is a phase in which first approaches towards the project definition, its initial scope, the fundamental functionalities it should incorporate and the potential risks are brainstormed.

Tasks involved:

T5 – Project Management	**32h**	**T1 – T4**
T5.1 – Initial Scope	24h	None
T5.1.1 – Context & Problem Statement Have a clear understanding of the context and the problem trying to solve and document them in the report.	2h	None
T5.1.2 – Project Definition Identify the goals and objectives along with the core functionalities the tool of this project should satisfy and document them.	4h	T5.1.1
T5.1.3 – Project Justification Research about the similar solutions available and existing software, and analyze the reason for the existence of the development of this project and its benefits.	5h	T5.1.2
T5.1.4 – Identify Possible Requirements Attend meetings to brainstorm about functional and non-functional requirements that this project's tool should satisfy, differentiate between the must have features and the enhancements that would be nice to have.	7h	T5.1.2
T5.1.5 – Risk management plan Identify, analyse and evaluate the possible risks that could appear during this project, how to prevent them and how to manage them in case they occur. Document it.	4h	T5.1.4
T5.1.6 – Identify stakeholders Identify and document all the parties that could be interested in this project's tool, which parties it affects and who is going to use it. Document it.	2h	T5.1.2
T5.2 – Methodology & Rigor Compare the different work methodologies that could apply to the development of this project, select the best and document the work methods in detail. Also think and document about monitoring and validation options that will be used.	3h	T5.1.1 - T5.1.4
T5.3 – Basic Tools Identify and set up the basic tools that will need to be used during the development, communication, project management and all the topics involved with this	2h	T5.2

project.		
T5.4 – Technology & Architecture Assess the different types of technologies and architectures that would best fit this project goal and requirements.	6h	T5.3

3.1.3. Project Strategy: Planning & Analysis Phase

Having elapsed through the previous phases the goal of the project strategy is to finally get into depth into the project's planning. This phase includes analysing, organizing, specifying and narrowing down all the previous tasks done until this phase so that a realistic project scope can be defined.

The project strategy is also responsible for specifying a list of tasks with their corresponding estimated time of work and creating an initial planning of the expected project's evolution leaving everything ready to begin the execution phase.

Tasks involved:

T6 – Refine Project Management	**8h**	**T1 - T5**
Organize new meetings to discuss the considerations of the previous phase and organize, review and refine the previous tasks documenting the changes.		

T7 – Tasks	**12h**	**T1 - T6**
Identify and organize the tasks done in previous phases so that they can be documented during this phase and plan and identify the tasks for future phases.		

T8 – Product Backlog	**8h**	**T1 - T5**
Create a product backlog that contains all the user stories and use cases and estimate their amount of work hours.		

T9 – Project Planning	**12h**	**T1 - T8**
Create an initial schedule of the expected project progress organized by phase with outlined milestones and dependencies. Also create Gantt Diagrams to have a more attractive graphical and easy to consult representation of the project's plan.		

3.1.4. Project Execution: Design & Implementation Phase

This phase is in charge of the project development & management and its sprint organization will contain the following procedures:

1. Sprint Kick-off – first meeting of the sprint in which the stories that are going to be developed during the sprint are selected and the sprint's

expectations are set. It is a moment in which user stories can be refined and discussed.

2. Daily stand-up – short meetings to report about what was done the previous day, notify possible blocked situations or inconveniences, inform about what will be done today and express the confidence on successfully finishing the sprint.
3. Sprint Review (Demo) – meeting towards the end of the sprint in which the user stories developed in the sprint are demoed, and therefore, it is discussed if the stories are accepted, rejected or what it can be improved.
4. Sprint Retrospective – last meeting of the sprint in which the sprint's progress and performance is analyzed, and good things and things to improve are discussed for future improvement of the process.

This phase will consist of 5 sprints and 3 releases, being the 2 first releases of 2 sprints each and the last release of 1 sprint only. The sprint duration is the same for each sprint, 13 days, 104h.

Tasks involved:

T10 – Sprint Kick-off	3h	T1 - T9
Meeting to analyse and determine the user stories that will be done in this sprint.		

T11 – Implementation	55h	T1 - T10
Time to implement and develop user stories.		

T12 – Testing	20h	T1 - T11
Time to test the user stories implemented in this sprint.		

T13 – Documentation	20h	T1 - T11
Time to write and document relevant information from the sprint or other tasks such as sequence diagrams or time to improve the definition of user stories or their acceptance criteria.		

T14 – Sprint Review (Demo)	2h	T1 - T12
Meeting to present a Demo of the progress achieved during this sprint in order to get feedback about the functionality or the expected behavior.		

T15 – Sprint Retrospective	2h	T1 - T12, T14
Meeting to analyze the performance of the sprint, analyze what went well and what did not and provide ideas towards how things can be improved for future sprints.		

3.1.5. Project Closure

The project closure being the last phase of the project is the phase responsible to do some last updates on the sections that are left to finish or to improve, make sure that all the documentation and implementation is up to date and working well so that it is ready to be delivered.

Tasks involved:

T16 – Monitoring	**15h**	**T1 – T15**
Observe and check the progress or quality of the project's tool.		

T17 – Documentation Wrap Up	**40h**	**T1 – T15**
Update and finish the project's documentation, make sure all the content is understandable, it follows a good structure and format, there are no grammar mistakes, and everything that involves a successful final version.		

T18 – Formal Presentation	**20h**	**T1 – T15**
Prepare the slideshow that will be used for the defense of the project to the judges and practice presenting it.		

3.2. Methodology and Rigor

Development methodologies are usually thought for teams, and even though I will be working alone in the development of this project, these methodologies are still the best fit to the type of work that will be done.

The methodologies of a software engineering project are divided in 2 categories: Traditional and Agile.

The Traditional methodology is meant to follow a fixed linear process, which according to figure 2, would be: analyze, design, code and test. This type of methodology is really firm about the order of tasks, and not only demands from the start a really good comprehension of the project, but are also built for large-scale projects with strict time and budget constraints.

On the other hand, the Agile methodology is iterative, flexible and responds to modifications quickly. This is mainly due to that as figure 2 shows, this type of methodology is based on shorter sprints, where all tasks: analyze, design, code and test are applied in every sprint. This procedure is usually used in small or medium-scale projects, is meant for products that demand a high involvement of clients, and it highly relies on teamwork and collaboration.

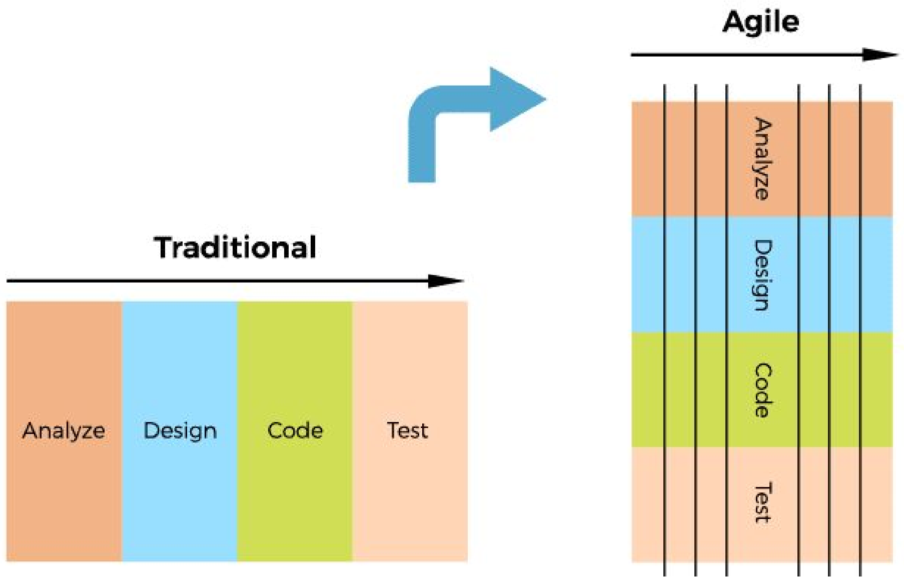


Figure 2 – Traditional VS Agile Methodologies

Having this in mind, based on the complexity of the project, its short development time frame available and that I will be working alone, I decided to go through an Agile methodology.

Due to the complexity of the project many uncertainties are to be discovered along the project development process, hence, modifications are expected to happen, and taking into account the project's deadline it is important to count with the possibility of adapting quickly to these changes.

Also, decisions will need to be made through the evolution of the product, and it would be naive to try and define all these decisions beforehand. Therefore it is also a project requisite to have the opportunity of actively receiving feedback from the product owner, Nuvolar Works.

Applying this procedure will make it easier for me to design and develop this project with confidence, together with the benefits of flexibility to quickly adapt to changes, and receiving regular feedback.

Particularly, I will be using a SCRUM methodology [12] because of its ability to manage complex software, sprints of two weeks long and the set of roles, responsabilites and meetings it follows: sprint planning, daily stand-up, sprint demo and sprint retrospective.

In addition, the validation methods for the project monitoring, besides consisting of meetings with the project director, co-director and tutor, it will involve the validation methods covered by the SCRUM methodology: a daily catch-up, the sprint's testing phase, and the product backlog.

3.3. Time Estimates & Gantt Diagram

Time associated to each phase of the project's life cycle:

Phase	Time (hours)
Learning & Researching	128h
Project Vision: Conception & Initiation	32h
Project Strategy: Planning & Analysis	40h
Project Execution: Design & Implementation	520h
Project Closure	80h
Total Hours	**800h**

As this planning covers the hours up to the Project Execution Phase, as a prevention technique to the risk of any overlooks or a bad estimation of tasks, the sum of the estimated time assigned to all the tasks of the Project Execution and Project Closure phase are less than the estimated time of the overall phase.

Followingly figure 3 will show a Gantt Diagram of the project's life cycle:

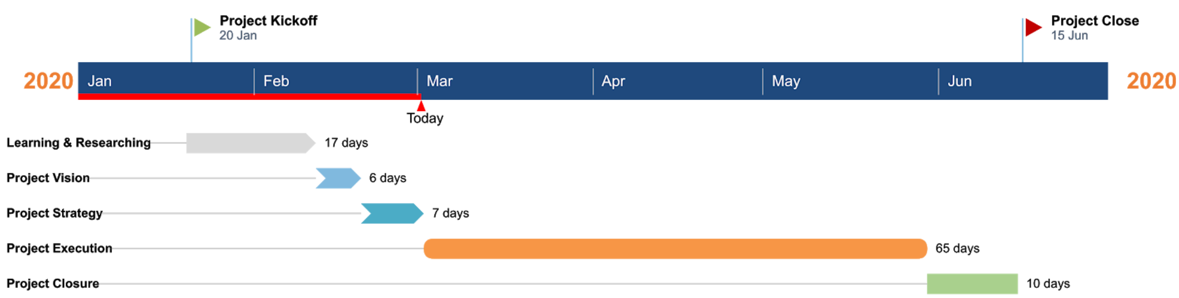


Figure 3 – Project Gantt Diagram

To provide further detail about the Project Execution phase, figure 4 nicely illustrates the previously described procedure of the product releases and project sprints organization through a Gantt Diagram:

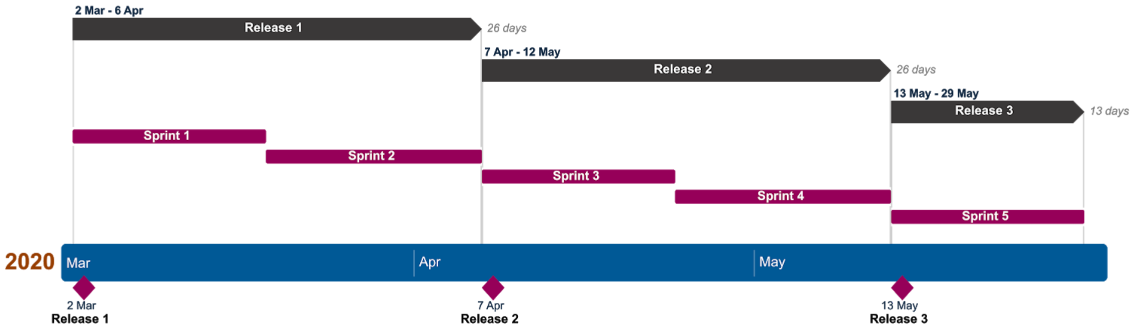


Figure 4 – Project Execution Gantt Diagram

3.4. Risk Management

In this section I will introduce the possible risks that were initially considered as a hazard for the expected development progress of the project. In addition, an evaluation of how to prevent each risk will be provided together with a proposed solution towards how to mitigate the risk given the situation in which it occurs.

In this way, anticipating any of these possible risks would allow me a better control towards the repercussions the risk can have on the development of the project.

Risk	How To Prevent	How To Manage
Encountering any problems or overlooks that would prolong the duration of the planning performed and therefore preventing me from reaching the delivery deadline with the project finished.	❖ Have a robust but flexible planning and organization of the project's life cycle phases. ❖ Have good communication with members of the company to have multiple points of view and therefore reduce the probability of this risk. ❖ Have a clear definition of the project goal and objectives and a realistic estimation of time for the tasks to be done. ❖ Create an MVP of an MVP, and later on increase the amount of functionalities and enhancements if more time is available.	There are 2 viable options: 1. Review the project's plan and restructure accordingly. 2. Delay the presentation of the thesis to the next turn. 3. Cutback the project's scope. 4. Become a POC instead of an MVP.
Bad planning, organization, estimation of tasks or time management	Take advantage of the opportunity of doing the FDP in a company so that I can receive feedback in relation to my workflow.	Create a quick and abstract plan that would enable the delivery of the project on time.

Any relevant Salesforce errors or modifications that could affect the development of this tool either in the use of APIs, in my Salesforce environments, new constraints or any other service.	Impossible to prevent, even though it should be an improbable situation because of the fact that most of their customers depend on it.	Impossible to manage, given that this tool depends on external services like the ones provided by Salesforce. I will need to trust in its reliability and support.
Any personal or public health issues, as the COVID-19 pandemic we are currently living.	It cannot really be prevented, it is not under my control. The only thing I can do is be careful, aware of the situation, take care of my health and well-being and have a preventive attitude.	Follow the corresponding procedures of the company and adapt accordingly with any planning adjustments if needed.

3.5. Resources and Economic Management

Once finished establishing the project's planning, we can proceed to identify the elements that will take part in the costs of the project and estimate them so that we can have a proper knowledge about the budget of the project and its viability.

3.5.1. Direct Costs

In this section the costs that are related to a task of the project will be described.

3.5.1.1. Human Resources

Considering that real world projects would not be developed by a single person, as in this case by me, having 4 different roles, I have decided to include in this section the cost estimated for this Final Degree Project (FDP) and also the costs that would have been associated with a Real World Project (RWP).

The following table presents the costs per role calculated per work hours.

<table>
<tr><th rowspan="2">Role</th><th colspan="2">€/Hour</th></tr>
<tr><th>FDP</th><th>RWP</th></tr>
<tr><td>Team leader</td><td rowspan="4">9</td><td>25</td></tr>
<tr><td>Analyst</td><td>20</td></tr>
<tr><td>Developer</td><td>16</td></tr>
<tr><td>Tester</td><td>16</td></tr>
<tr><td>Director</td><td>25</td><td>-</td></tr>
<tr><td>Co-director</td><td>20</td><td>-</td></tr>
</table>

Rapporteur	20	-

Important note: in my project I'm considering no designer or design phase since it is an MVP and what should fundamentally work is the back-end, therefore the designer requirements are really basic and it is required from the developer to also act as the designer.

Once we have the costs per work hour we can then calculate the amount of work hours required from each role in each phase, and therefore conclude the costs associated with each role, as shown in the following table.

Phase	TL	A	Dev	T	Dir	Co	R
L&R	13	64	51	-	20	10	-
PV	16	24	-	-	6	3	2
PS	19	29	-	-	8	4	5
PE	102	-	256	154	40	20	20
PC	58	-	7	7	10	5	10
Total Hours	**208**	**117**	**314**	**161**	**84**	**42**	**37**

Then finally the next step is to multiply the cost per hour associated to each role from the first table with the amount of hours excepted from the second table and we get the following table, which shows the total cost estimate per role.

<table>
<tr><th rowspan="2">Role</th><th colspan="2">Cost (€)</th></tr>
<tr><th>FDP</th><th>RWP</th></tr>
<tr><td>Team leader</td><td rowspan="4">7,200</td><td>5,200</td></tr>
<tr><td>Analyst</td><td>2,340</td></tr>
<tr><td>Developer</td><td>5,024</td></tr>
<tr><td>Tester</td><td>2,576</td></tr>
<tr><td>Director</td><td>2,100</td><td>-</td></tr>
<tr><td>Co-director</td><td>840</td><td>-</td></tr>
<tr><td>Rapporteur</td><td>740</td><td>-</td></tr>
</table>

Total Cost	10,880	15,140

For the rest of the project I will use the total cost value of the FDP as a reference to the direct costs.

3.5.1.2. Software

The following table represents the cost for the software used in this project. The software that is indented from Google Chrome is to specify that the web version is being used.

Software	Cost (€)	Amortization (€)
Google Chrome ➜ Clickup ➜ Google Drive + Gmail + Calendar + Meets ➜ Office Timeline Online ➜ BitBucket ➜ Trailhead ➜ Salesforce	0 / Free version	0
Slack		
Visual Studio Code		
Postman		
Fork		
Robo 3T		
Total		**0**

3.5.1.3. Hardware

The following table shows the cost associated with the hardware used for this project.

Product	Cost (€)	Units	Life Expectancy	Amortization (€, in 5 months)
Apple Macbook Pro	1,499	1	8	79

Retina 13-inch, Early 2015				
Dell monitor U2415	239	1	6	17
Apple Magic Mouse	85	1	3	12
Apple Magic Keyboard	99	1	3	14
Total Costs				**122**

3.5.2. Indirect Costs

These are the costs non-related to the project's activity.

Service	Cost (€/month)	Period (months)	Amortization (€)
Electricity	7	5	35
Water	3	5	15
Internet	30	5	150
Transport	27	5	135
Local	350	5	1,750
Total Cost			**2,085**

3.5.3. Contingencies

Every project budget needs to have a contingency plan in order to be prepared for any unexpected costs because of oversights or simply bad estimations. For this thesis caution a 10% of the budget of the direct and indirect costs will be considered as potential contingencies.

Cost Type	Estimated Cost (€)	Percentage (%)	Contingency (€)
Direct Cost	11,002	10	1,101
Indirect Cost	2,085	10	209
Total Cost			**1,310**

3.5.4. Incidentals

In order to be covered of any potential risks associated with additional costs, the cost attributed on the basis of probability of its occurrence will be considered as an estimate of incidentals costs.

	Incidental	Management (€)	Risk (%)	Cost Associated (€)
Malfunction	Apple Macbook Pro Retina 13-inch, Early 2015	150	15	23
	Dell monitor U2415	239	10	24
	Apple Magic Mouse	85	30	26
	Apple Magic Keyboard	99	30	30
	Total Cost			**103**

3.5.5. Total Budget

In the following table a summary of the costs of each cost type can be seen with the corresponding total budget estimated for the fulfilment of this bachelor thesis.

Cost Type	DC	IC	Contingency	Incidental	**Total Cost**
Cost (€)	11,002	2,085	1,310	103	**14,500**

In addition, as no contingencies or incidents took place during the project's development, the following table describes the exact project total costs:

Cost Type	DC	IC	**Total Cost**
Cost (€)	11,002	2,085	**13,087**

3.5.6. Cost Control

The objective of this section is to establish a mechanism that allows the control of costs during the project and therefore compare and assess any variance between the expected budget and the actual costs at different stages of the project.

As for my case I have all the costs calculated in a spreadsheet where I can add an extra column for any additional non-expected costs, therefore as the project progresses, if a new non-considered cost appears, I would have an easy way of controlling and being aware of the actual costs of the project.

Anyways, there does not seem to be any additional reason for which additional costs should appear but in case they do then hopefully the contingency cost estimated will be enough to cover it.

3.6. Conclusions

The work plan just described was already developed having in mind that several drawbacks will arise through the project development, therefore forcing modifications in the product behavior.

As a result, the plan was followed successfully without having the need to apply any modifications to the initial planning thanks to the anticipation of risks.

However some slight variations took place in the project sprints, particularly related to the amount of hours dedicated to each task. For instance in some sprints documentation hours were reduced in order to have more implementation hours.

Nonetheless this was resolved thanks to the extra time included in the Project Execution phase that was not assigned to its tasks as a prevention measure for this kind of scenarios.

Also, even though the current pandemic drastically changed my working environment from the company offices to working from home, thankfully there were no repercussions in terms of additional costs or required development resources.

As a result, the following table provides an approximation of the total amount of hours dedicated to each task:

Task	Time (hours)
Learning & Researching	130h
Design & Development	60h
Implementation	300h
Debugging	120h
Testing	80h
Documentation	100h
Total Hours	**800h**

4. Scope

Having gone through the project context, analyzed its state of art and planning, the purpose of this unit is to have a better understanding of the scope of the project, have a good comprehension of the project's terminology and be fully aware about the context in which this project tool belongs.

This will be achieved by outlining the project objectives, providing complementary background knowledge about data migrations, and lastly describing an example scenario in which this project's product would be of use.

4.1. General Objectives

The tool developed in this project is at least expected to satisfy the following objectives:

1. Have access to both a Salesforce Source and Target Org simultaneously.
2. Be compatible with most standard and custom Salesforce objects and fields.
3. Preserve the proper relations between the migrated data.
4. Determine the compatibility of the specified data migration.
5. Successfully automate the process of a specific data migration taking into account the order of insertion of records.
6. Give feedback from any errors that might occur.

On another note, as the tool is not yet expected to include any user or session management, for the time being, it is not an objective for the tool to be deployed, but is expected to remain working locally.

4.2. Complementary Background Knowledge

In this section I will describe some more context and must know considerations about the topic of data migrations that is involved in this project so that the reader is able to fully understand and correctly follow this project's requirements.

So with all the previous concepts about Salesforce in mind, I will begin by stating that from the 2 types of restrictions, internal and external metadata, it is only viable for this tool (and I would say any tool), to consider the objects internal metadata, in this way delegating the task to the control of the objects external metadata to the Target Org itself.

As a reminder, this decision is mostly because of the fact that the objects external metadata was not built to be controlled externally from a system that is not a Salesforce Org.

Accordingly, it is important to have in mind that the tool is not expected to work for every single case even if the migration is considered feasible, as the tool will only recognize the objects internal metadata.

Therefore there are scenarios where the data migration will fail because of an external metadata restriction. For this reason the tool intends to provide feedback to the user through descriptive and informational error messages, so the user can manually relax the restriction that caused the data migration to fail and therefore successfully achieve the data migration.

In addition, this tool will only incorporate one type of partial data migration, concretely, what I call a *parent related objects migration*. This type of migration was chosen for the fact that it is the fundamental migration required in order to tackle relationships between objects, and most importantly the parent objects, which are the most significant as a means to a feasible migration.

Explaining this concept more in depth, a parent related objects migration is a migration in which given one or multiple records, all the given records will be migrated, including its relations, plus all the relations related records, and all the relations of relations related records, and so on recursively.

For a better comprehension of the parent related objects migration I will now provide a more detailed explanation supported by an example.

In a nutshell, the record relations are fundamentally the record dependencies of a migration. Correspondingly, these dependencies can either be required or optional. Therefore given a record B with a required relationship to a record A, in migration terms, this is understood as that record A is a required dependency for record B. In other words, in order for record B to exist in the Org, then record A needs to exist first.

Let's continue with a graphic illustration.

Figure 5 below, nicely illustrates the data structure of a Tree, where for example purposes, let's say that each node represents the different objects of an Org, and that each arrow represents the required dependencies (or relationships) that exist between objects. In such representation, the Tree below, is essentially portraying the dependencies of the Orgs database schema.

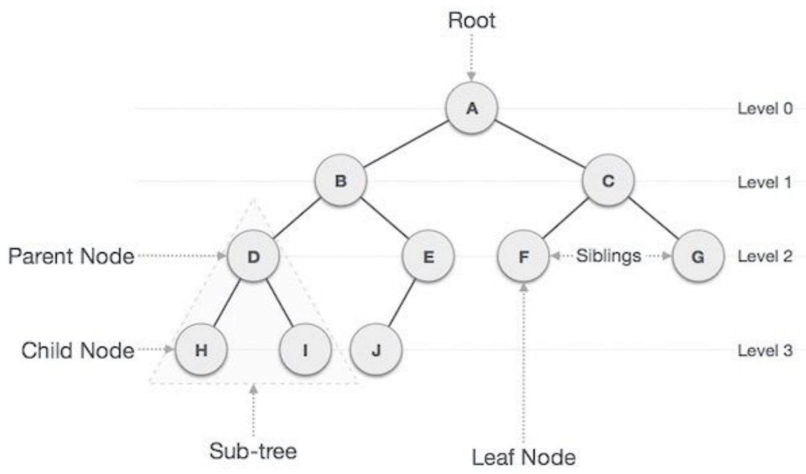


Figure 5 – Terminology of the Tree Data Structure

Continuing with the example, as a Tree that Figure 5 represents, the terminology used to interpret its content is as done with a family tree. To begin with, it is read from top to bottom, hence, object (node) A, is the parent object (node) of objects B and C, or put differently, objects B and C have a parent required relationship with object A. As stated earlier, this indicates that in order for a record of the objects B or C to exist in an Org, there first needs to exist a record of object A so that the records of objects B or C can be related with.

Followingly, in terms of a parent related objects migration, given records of object D, then the migration will involve all the object related records of the given records of object D in a parent level, ergo the related record belonging to objects D, B and A. Moreover, wording it as Graph Theory nomenclature, the objects involved in the migration will be all the object's ancestors.

Also as you may imagine, this is just a simplistic example with the purpose of understanding the concepts encompassed between object relationships that make up a parent related objects migration.

In real Orgs though, object dependencies can be far more sophisticated than in the example provided, as circular required relationships can be created, resulting in impossible migrations, also required relationships between siblings can occur or even objects that belong to a different subtree, as it would be in Figure 5 in the case object F had a relationship with object J.

As a result the representation of the Orgs database schema dependencies cannot be done through the data structure of a Tree, bu the tool actually makes use of DAGs, Directed Acyclic Graphs

• • •

With these concepts clear in mind, now we should be prepared to brainstorm and understand the steps this tool should undergo with the intention of conducting a successful data migration. The steps are the following:

1. Access to the Source and Target Orgs (API Authorization)
2. Identification of Relevant Objects
3. Orgs Mapping (Object & Field)
4. Data Extraction
5. Data Import

The purpose of the first step is straightforward, to give the tool access to the Source and Target Orgs so that the system can communicate with the Orgs through the Salesforce APIs.

Secondarily, the following step is to identify the objects that could be involved in the migration, what I prefer to call the relevant objects or the objects of interest of the migration. And in a parent related objects migration, from the previous explanation we know that given a selected object the relevant objects would be its ancestors, which can be found by recursively iterating through the objects relationships on a parent level.

Upon having identified the objects of interest of the migration, the next step is to map the Source Org objects and fields with their equivalent Target Org objects and fields, what I refer to as the Orgs Mapping.

This step is necessary as a result of the possibility of different Orgs having different objects internal metadata, hence, the system needs a way of identifying which objects and fields correspond with each other so it is able to manage the data correctly. Additionally, another core purpose of this step is also to verify the compatibility between the Source and Target Orgs Salesforce schema.

On the other hand, the system does not have a proper methodology to automatically perform this mapping with certainty. Therefore it requires the manual assistance of the user.

Fourthly, given one or many Source record IDs of the main selected object of the parent related object migration, each of the records data and the parent related records data can be extracted from the Source Org recursively.

And lastly the final step is associated with the import of the extracted Source Org data into the Target Org. Accordingly, this step actually consists of many tasks where the migration is prepared, organized and planned so that the Org restrictions are correctly handled.

Likewise, the system also needs to verify other considerations such as ensuring that all the necessary data is obtained and ready for import, determining an appropriate order of insertion.

Then the system can begin the insertion process, in which between insertions has to perform some tasks such as mapping the new Target Org IDs with the Source Org IDs, as a means to preserve the relationships between records.

Later, once the system has performed all the imports a last step might be required in order to manage circular dependencies between optional fields as these could not be imported at the moment of creation. In this manner, the system may update some records data with its missing optional relationships.

Otherwise, if anything fails during the import or update of data the system will roll back all any changes done in the Orgs data by the imports or updates performed.

• • •

With all these steps in place, it should be acknowledgeable how performing a parent related objects migration has the potential of being as easy as clicking some buttons in the right places, and letting a system automate all these steps.

However the inner complexity of the migration is still there, meaning that the system is not expected to succeed in any possible parent related objects migration, but expected to fail in cases of faulty Orgs Mappings, circular required dependencies between the object records relationships or with incompatible objects external metadata rules.

And for all these reasons it is so important for the system to provide feedback to the user towards any errors that take place during the migration.

4.3. Example Scenario

Let's imagine the case of a company that is developing a custom app on top of Salesforce that manages the contents of the most popular streaming platforms.

Following this scenario, imagine that a particular developer is working on a specific functionality related to users commenting spoilers about tv show episodes, and therefore in order to test this functionality, the developer first needs to populate its Org with quality data similar to the data end users would visualize.

The manual approach to do this would be to think and manually insert into his Org a fake set of data that replicates real data, nonetheless, carrying out this task can be more sophisticated than what it seems.

That is because the developer would not only need to think about the fake data related to the user comments, but also about the related data stored about these user comments. For instance, a comment is written by a user, about a certain episode, which belongs to a tv show, of a particular streaming platform, and so on.

As a consequence, in a Salesforce Org, a comment record can depend on the existence of many other records like the ones mentioned, hence, creating this fake data is not a simple task.

Despite this, considering the developer already counts with the fake data in a particular Org, the developer would then need to carry out a data migration to transfer the fake data to its Org.

For the purpose of successfully achieving the data migration, the developer should have a really good understanding of the Org and therefore begin by planning the procedure he should follow to correctly transfer the fake data into his Org.

However as explained in the previous section, as a result of the immense amount of considerations the developer would need to contemplate, this is a really delicate and error-prone task.

Not to mention all the errors and problems the developer would face when executing the planned procedure due to silly mistakes, overlooks or a vague planning that would cost the developer additional time to fix the errors and refine the planning.

Anyhow once finished, the developers goal on performing a data migration would be achieved.

However, this is just a simple case, as now also imagine this same scenario but with multiple developers working on similar or related use cases, that is to say, they would all have to independently go through this data migration procedure. And this definitely does not seem a good approach, it is really counterproductive.

On the other hand, with the use of the product developed in this project, the developer, or developers, will be able to quickly and effortlessly replicate the data of another Org into their Orgs.

5. Requirement Analysis

The purpose of this unit is to define the conditions that determine the software requirements the system should meet.

The course in which these requirements were collected is through a combination of the product owner's* requirements, and a thorough analysis of an accomplishable scope bearing in mind this project's lifetime.
*as this FDP is carried with a company, Nuvolar Works, is the product owner.

These conditions will be available in 2 forms: functional and non-functional requirements.

5.1. Functional Requirements

Functional requirements are responsible for describing any specific functionality which defines what function a software must perform, in other words, they are in charge of determining the intended behavior of a system.

Consequently the systems functional requirements will be presented along with a description of their corresponding use cases.

5.1.FR1. Org Auth Management

- *Sign Up Org* – The system must allow the user to authenticate into an existing Salesforce Org independently of its instance (meaning that the Org can be of any type) and authorize its API access to the system through the OAuth 2.0 protocol.
- *Auth Life Validity* – The system must be capable of automatically maintaining the API authentication life validity of each registered Org without the need of additional user interaction.

5.1.FR2. Metadata Management

- *Basic Metadata Extraction* – The system must be capable of retrieving all Salesforce Org API accessible objects metadata and its corresponding fields metadata whenever sufficient API calls are available.
- *Parent Related Objects Identification* – The system must be capable of identifying all the Org's optional and mandatory objects related to a given object on a parent level.

5.1.FR3. Orgs Mapping Management

- *Object Mapping* – The system must allow the user to manually map the Source and Target Orgs objects related to a migration.
- *Field Mapping* – The system must allow the user to manually map the Source and Target Orgs object fields related to a migration.

- *Compatible Mapping Selection* – The system must only allow the user to map compatible Org object fields.
- *Smart Relevant Objects Selection* – The system must allow the user to select which objects and fields are desired to be migrated, maintaining at all moments the selection of the required objects. (in order to help the user to satisfy the basic requirements for a migration to occur)
- *Object Auto-mapping* – The system must be capable of auto-mapping the Source and Target Orgs objects by name independently of having a prefix.
- *Field Auto-mapping* – The system must be capable of auto-mapping the Source and Target Orgs fields by name and type independently of having a prefix.

5.1.FR4. Migration Records Input

- *Source Main Object* Salesforce IDs – The system must allow the user to input multiple Salesforce ID's of the Source main object record.

5.1.FR5. Migration Automation

- *Parent Related Data Extraction* – The system must be capable of extracting data from a registered Salesforce Org whenever sufficient API calls are available.
- *Verify Migration's viability* – Given the orgs mapping and the extracted data, the system must be capable of determining if a migration is viable.
- *Determine Order of Insertion* – Given a viable migration, the system must be capable of determining a viable order of insertion.
- *Import Data* – The system must be capable of importing data to a registered Salesforce Org whenever sufficient API calls are available.

5.1.FR5.1. Considerations

In the automation of a migration there can appear certain scenarios in which decisions have to be taken according to the behavior that the system must follow.

The following list specifies the criteria the system should adhere to for the given scenarios:

- Unique fields in the Target Org – extracted Source Org records with unique fields in the Target Org should be checked if they already exist. If it exists, then the system should not migrate this particular record nor proceed with its parent related records, however, it should preserve the original records relations with the matching Target Org record.
- Users migration – every record has at least a field related to a User by default, however the system should not attempt to migrate Source Org Users into the Target Org. Instead the system should replace all the Source Org User references by the Target Org User that is performing the migration, which is provided through its credentials authorization.

- RecordType object – since the RecordType is a peculiar object with an uncommon behavior, RecordType records should be treated as unique fields. This implies that RecordType records should not be migrated, but queried from the Target Org, and if an equivalent RecordType record exists, then original records relations must be preserved with the matching Target Org record.

5.2. Non-Functional Requirements

Non-functional requirements define the quality attributes of a software that enhances its functionality. They are essential to judge the system and determine if the user needs are met.

The following template will be used to describe them:

Name	**Code #**
Description	Description text.
Justification/Motivation	Justification text.
Acceptance Criteria	Acceptance Criteria text.

Efficiency	**# 1**
Description	This tool should consume the least amount of resources possible for tasks related to the Salesforce Target Org.
Justification/Motivation	Salesforce Orgs have many restrictions that can easily escalate into costs or use limits, therefore the system should always try to interact with the Target Org in the most efficient way.
Acceptance Criteria	Requests to the Target Org should always be optimized in a way in which the amount of total requests without being optimized is bigger than the amount of total requests with the optimization.

Robustness	**# 2**
Description	This tool should have the ability to cope with errors and provide useful feedback on errors.
Justification/Motivation	As it is expected for the system to fail because of Salesforce Orgs restrictions, the system should provide the user with enough feedback when an

	error occurs so that it can adjust any configuration, when possible, with the intention of making the migration feasible.
Acceptance Criteria	Whenever an error occurs the user should be presented with a descriptive error that points to where or why the error has occurred.

Usability	**# 3**
Description	This tool should be intuitive and easy to use with a click not code approach.
Justification/Motivation	The user should not feel confident and comfortable when using this tool, hence, the system should provide the user a smooth way to interact with the tool that does not require a lot of thinking so that tasks can be easily achieved.
Acceptance Criteria	Given several tasks, users from different backgrounds that are familiar with Salesforce Orgs should be capable of successfully performing the provided tasks intuitively.

6. Specification

With a clear scope and requirement analysis that have displayed the demands the software should fulfill to please the end user needs, I can now proceed to depict a more technical description of the tasks that need to be carried out by the system to accomplish the previously specified use cases.

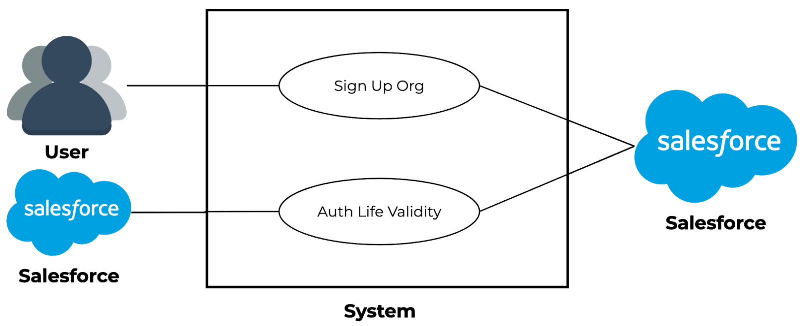
Accordingly, the following sections of this unit will integrate the specification of each of the functional requirements use cases.

One of the artifacts that will be used in these specifications are use case diagrams, which depict through a graphical approach the series of actions the system allows external actors to perform in order to carry out the tasks that activity involves.

In addition a table with the following template will be employed to provide a detailed description of the use case in hand:

Use Case	#Code: Name	**Primary Actor**	Actor Name
Preconditions	- Precondition 1 - ... - Precondition N		
Trigger	Trigger description		
Primary Use Case			
1. Step 1 2. ... 3. Step N			
Extension: Extension Name			
1. Step 1 2. ... 3. Step N			

6.FR1. Org Auth Management



Use Case	**#1: Sign Up Org**	**Primary Actor**	User
Preconditions	- The Org has not been registered yet		
Trigger	The user clicks on the “Sing Up Org” button		

Primary Use Case

1. The user is on the home page.
2. The user clicks the “Sign Up Org" button.
3. The user selects the Org Type of the new Org it wants to register.
4. The system asks Salesforce for an OAuth 2.0 logIn URL and redirects the user to that URL.
5. The user enters and submits their Salesforce credentials.
6. The user allows access to the system’s Connected App.
7. Salesforce communicates to the system a code to request the access token, redirecting the user to a loading page while the system registers the new Org.
8. The system requests the access token and requests Salesforce other complementary information to register the new Org.
9. The system redirects the user to the home page.

Extension: Failed Authentication

1. The user is on the home page.
2. The user clicks the “Sign Up Org" button.
3. The user selects the Org Type of the new Org it wants to register.
4. The system asks Salesforce for an OAuth 2.0 logIn URL and redirects the user to that URL.
5. The user enters and submits wrong Salesforce credentials or enters the correct credentials but denies access to the system’s Connected App.
6. Salesforce returns the system an error.
7. The system redirects the user to the home page with an error message.

Use Case	**#2: Auth Life Validity**	**Primary Actor**	Salesforce
Preconditions	- The Org is registered in the system		
Trigger	The system needs to perform a functionality that requires making a request to Salesforce with the specified Org’s credentials		

Primary Use Case

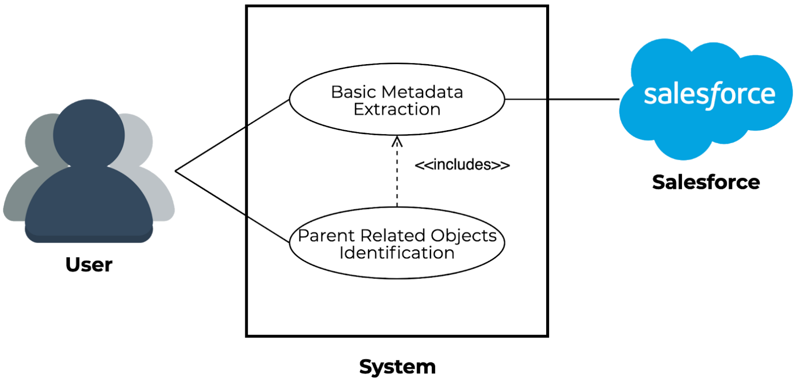
1. The system makes a request to Salesforce.
2. Salesforce returns an error regarding the access token being expired.
3. The system requests Salesforce a new access token.
4. Salesforce returns a new access token.
5. The system stores this new access token and retries the query.
6. Salesforce returns the expected response.
7. The system continues with the task.

Extension: Failed Refresh Token

1. The system makes a request to Salesforce.

2. Salesforce returns an error regarding the access token being expired. 3. The system requests Salesforce a new access token. 4. Salesforce returns an error. 5. The system returns an error to the user.

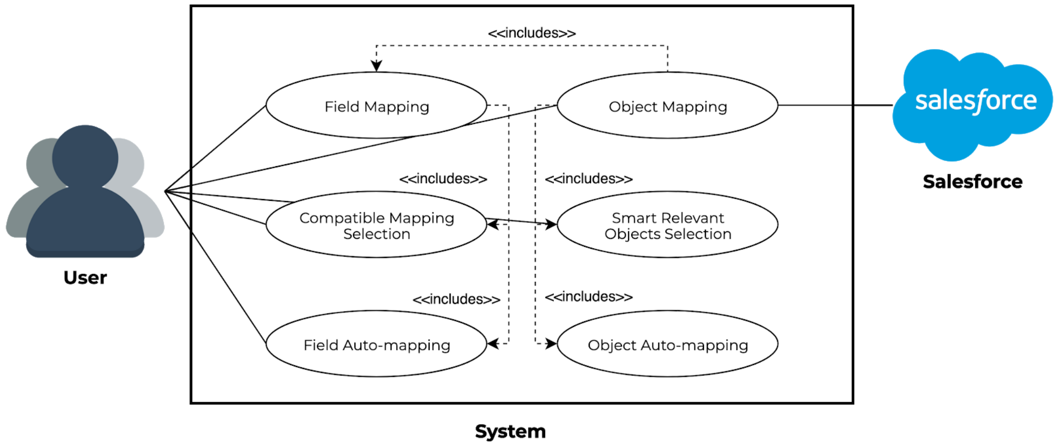
6.FR2. Metadata Management



Use Case	**#3: Basic Metadata Extraction**	**Primary Actor**	User
Preconditions	- There needs to be at least 2 Orgs registered in the system		
Trigger	The user starts a migration or the user clicks on the refresh metadata button		
Primary Use Case			
1. The user is on the parent related objects migration page. 2. The user clicks on any of the select components or clicks on the refresh metadata button. 3. The system makes a request to Salesforce. 4. Salesforce returns all the Orgs objects basic metadata. 5. The system stores the necessary data.			

Use Case	**#4: Parent Related Objects Identification**	**Primary Actor**	User
Preconditions	- There needs to be at least 2 Orgs registered in the system		
Trigger	The user has selected to perform a parent related objects migration		
Primary Use Case			
1. The user is redirected to the parent related objects migration page. 2. The user selects the main object for the Source Org and its corresponding Target Org equivalent object. 3. The user clicks the “Continue” button. 4. The system identifies the parent related objects for the selected Source main object and the Target main object.			

6.FR3. Orgs Mapping Management



Use Case	**#5: Object Mapping**	**Primary Actor**	User
Preconditions	- There needs to be at least 2 Orgs registered in the system - The system has identified the parent related objects of the Source and Target Orgs main objects		
Trigger	The user has clicked the “Continue” button of the main objects selection step of the parent related objects migration page		
Primary Use Case			
1. The user is on the Orgs Mapping step of the parent related objects migration page. 2. The user is looking at the Object Mapping nested page. 3. The user correctly manually maps the objects and its corresponding fields. 4. The user clicks on the “Continue” button.			

Use Case	**#6: Field Mapping**	**Primary Actor**	User
Preconditions	- There needs to be at least 2 Orgs registered in the system - The system has identified the parent related objects of the Source and Target Orgs main objects - The user has performed the object mapping of the object to field-map		
Trigger	The user clicks on the “Field-Map” button of an object		
Primary Use Case			
1. The user is on the Orgs Mapping step of the parent related objects migration page. 2. The user is looking at the Field Mapping nested page. 3. The user correctly manually maps the fields and returns to the object mapping page. 4. The user clicks on the back to object mapping button.			

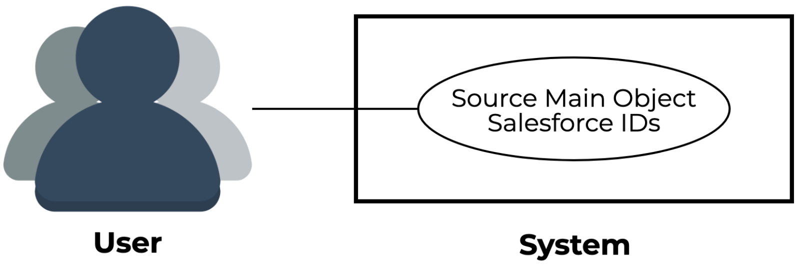
Use Case	#7: Compatible Mapping Selection	Primary Actor	User
Preconditions	- There needs to be at least 2 Orgs registered in the system - The system has identified the parent related objects of the Source and Target Orgs main objects - The user has performed the object mapping of the object to field-map		
Trigger	The user clicks on the “Field-Map” button of an object		
Primary Use Case			
1. The user is on the Orgs Mapping step of the parent related objects migration page. 2. The user is looking at the Field Mapping nested page. 3. The user clicks on the select component. 4. The user can only see and select the compatible fields that have no’t already been selected.			

Use Case	#8: Smart Relevant Object Selection	Primary Actor	User
Preconditions	- There needs to be at least 2 Orgs registered in the system - The system has identified the parent related objects of the Source and Target Orgs main objects		
Trigger	The user clicks on the relevant checkbox of an object		
Primary Use Case			
1. The user is on the Orgs Mapping step of the parent related objects migration page. 2. The user is looking at the Object Mapping nested page. 3. The user clicks on the relevant checkbox of an object. 4. The required relevant objects are re-evaluated.			

Use Case	#9: Object Auto-mapping	Primary Actor	User
Preconditions	- There needs to be at least 2 Orgs registered in the system - The system has identified the parent related objects of the Source and Target Orgs main objects		
Trigger	The user has clicked the “Continue” button of the main objects selection step of the parent related objects migration page		
Primary Use Case			
1. The user is on the Orgs Mapping step of the parent related objects migration page. 2. The user is looking at the Object Mapping nested page. 3. All the Source objects with a match have been automatically mapped.			

Use Case	#10: Field Auto-mapping	Primary Actor	User
Preconditions	- There needs to be at least 2 Orgs registered in the system - The system has identified the parent related objects of the Source and Target Orgs main objects - The user has performed the object mapping of the object to field-map		
Trigger	The user clicks on the “Field-map” button of an object		
Primary Use Case			
1. The user is on the Orgs Mapping step of the parent related objects migration page. 2. The user is looking at the Field Mapping nested page. 3. All the Source objects fields with a match have been automatically mapped.			
Extension: Auto-Map all fields			
1. The user is on the Orgs Mapping step of the parent related objects migration page. 2. The user is looking at the Object Mapping nested page. 3. The user clicks the “Auto-map all” button. 4. All fields of the relevant mapped objects with a match have been automatically mapped.			

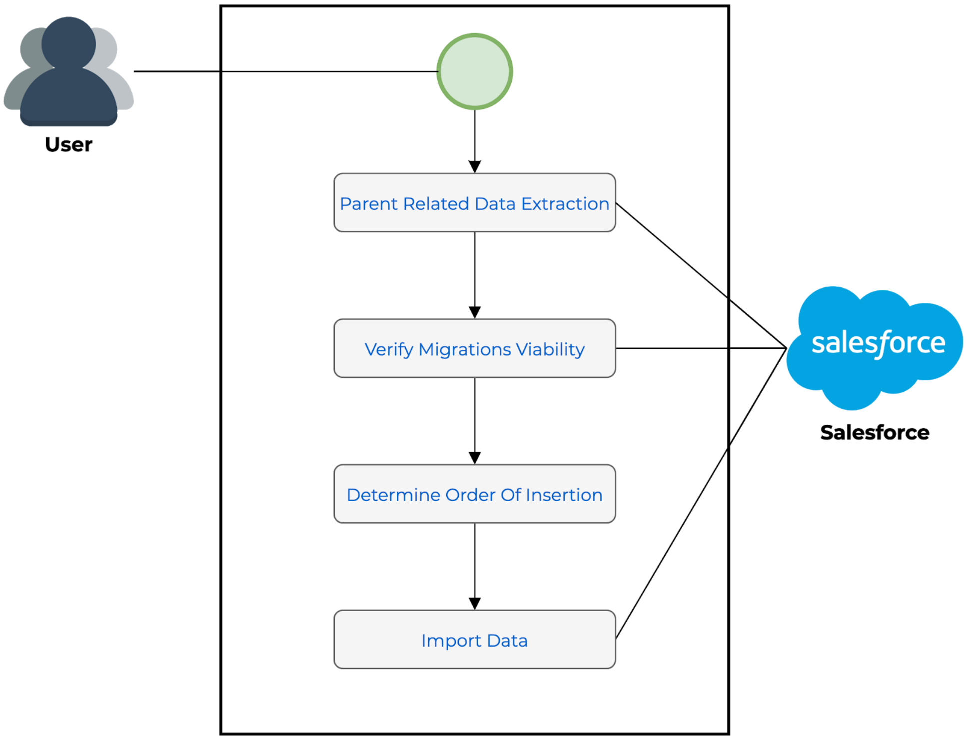
6.FR4. Migration Records Input



Use Case	#11: Source Main Object Salesforce IDs	Primary Actor	User
Preconditions	- There needs to be at least 2 Orgs registered in the system - The system has identified the parent related objects of the Source and Target Orgs main objects - The user has performed the orgs mapping		
Trigger	The user clicks on the “Continue” button of the Orgs Mapping step.		
Primary Use Case			
1. The user is on the Salesforce IDs step of the parent related objects migration page. 2. The user enters the Source main object’s Salesforce ID’s.			

6.FR5. Migration Automation

This specific functional requirement is distinct to the previous use cases. There is only one use case, but this use case is made of a process flow. Considering this, the use case diagram of this functional requirement will encapsulate a BPM following the BPMN standard.



Use Case	#12: Migration Automation	Primary Actor	User
Preconditions	- There needs to be at least 2 Orgs registered in the system - The system has identified the parent related objects of the Source and Target Orgs main objects - The user has performed the object mapping of the object to field-map - The user has entered at least one valid Salesforce ID of the Source main object		
Trigger	The user clicks on the “Begin migration” button of the Salesforce IDs step		
Primary Use Case			

1. The system creates a new migration and stores all the data related to the steps previously done of the migration (the orgs mapping and the salesforce IDs).
2. The process flow begins:
    a. Parent Related Data Extraction: The system extracts from the Salesforce Source Org the data of each of the given salesforce IDs and all their parent related records of the objects specified as relevant in the Orgs Mapping and stores them in the database.
    b. Verify Migrations Viability: The system verifies that all extracted records

can be migrated, handles any non-supported objects and confirms if there are any circular dependencies between the records relationships.

c. Determine Order of Insertion: The system analyzes the required dependencies between the records fields with relationships and determines a valid order of insertion by levels*.

*A level represents all the objects that are at the same level of dependencies, meaning that they are dependent from the objects with levels above them, and the objects with levels below them are dependent on them

d. Import Data: The system first imports to the Salesforce Target Org the required and non-related fields of each record by levels of the order of insertion and once all records have been created it updates all their optional related fields.

Extension: Import Data Failed

1. The system performs a roll back of all the inserted records. If for some reason it also fails to conduct roll back then return the inserted records IDs.

6.2. Conceptual Model

A conceptual model is a representation of a system's substantial concepts of a system. Its objective is to help people understand the domain represented through the use of entities, relations between the entities and the integrity rules of the model as a whole.

Figure 6 illustrates the schema of the conceptual model of this project, where the following entities appear:

- User – represents a user present in a Salesforce Org.
- Org – represents a Salesforce Org.
- Object – represents an object of a Salesforce Org.
- Field – represents the fields of an object.
- Object Mapping – represents the object mapping performed between 2 objects of 2 different Orgs.
- Field Mapping – represents the field mapping performed between 2 fields of 2 different objects of 2 different Orgs.
- Orgs Mapping – represents a collection of object mappings of 2 different Orgs.
- Record – represents a record extracted or imported, from or to a Salesforce Org object.
- Migration – represents a migration between 2 Orgs.

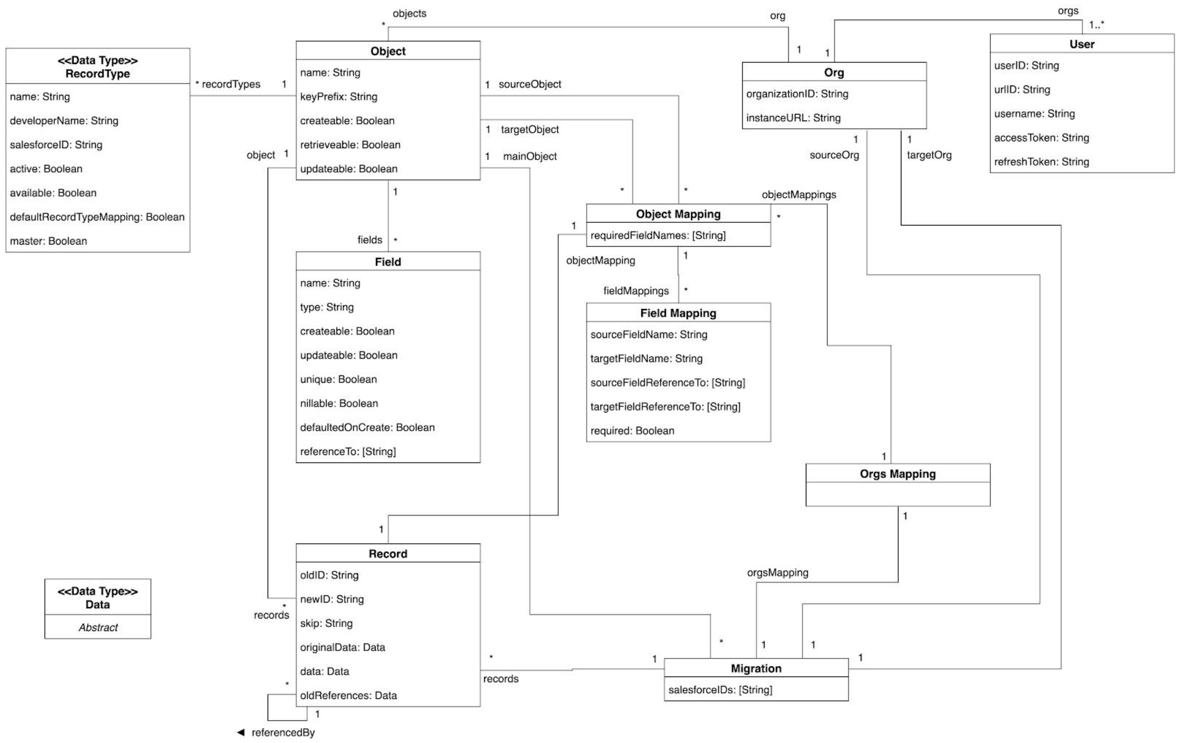


Figure 6 – Conceptual Schema

Additionally there are 2 Data Types:

- ❖ RecordType – represents the particular Salesforce standard object Record Type.
  It is necessary to explicitly have this Salesforce object as an entity because of its special behavior in an Org.
- ❖ Data – represents the data of different Salesforce object records.
  It is *abstract* because its attributes depend on the fields of the given Salesforce object, therefore it is impossible to determine a particular structure.

6.2.1. Integrity Rules

Lastly the integrity rules are the conditions the conceptual model must satisfy for the purpose of guaranteeing the consistency of the system.

The following list covers the most relevant integrity rules of the conceptual model:

1. Identifiers:
   a. User(userID)
   b. Org(organizationID)
   c. Object(name, org.organizationID)
   d. Record(oldID, newID)
2. In an Object Mapping, the sourceObject and targetObject cannot be the same object.
3. In an Orgs Mapping, all the Object Mappings should belong to the same 2 Orgs, being the Source Target Orgs.

4. An Object Mapping needs to consist of the Field Mappings that correspondingly represent the sourceObject and targetObject fields.
5. A Record’s object must match with the corresponding objectMapping sourceObject.
6. A Migration consists of the same 2 Source and Target Orgs involved in the Orgs Mapping.
7. A Migration’s mainObject must belong to an Object of the sourceOrg.
8. A Migration’s records must belong to Objects of the sourceOrg.

7. Design

At this point of the report it has been presented all the information necessary to start defining the design of the software in question.

Therefore, the goal of this unit is to overview the architecture by which the system is built and describe in depth each of the components that compose it.

7.1. Architecture

This project is complex enough just through its subject, and the procedure to accomplish a data migration is highly coupled. That is because the sequential process of tasks demands the procedure steps to await and highly rely on the correct execution of the previous tasks.

As a consequence, it is in the project interest to be developed through independent components, as an approach to loose the internal coupling of the system.

In contrast to a monolithic approach, modular design is more about the organization and internal implementation of the project, and not so much about the system's behavior. Modular code offers many benefits such as the ability to change the full implementation of one area without modifying its functionality nor leading to unintended consequences in other areas of the code.

Through smaller well-defined chunks of code, this technique makes the system more readable and easier to comprehend. Also thanks to the possibility of working with independent modules, the system's maintainability is also improved, since if mistakes are made throughout maintenance, they will be limited to the specific modules changed.

Bugs and errors can now be fixed by working on a small number of modules instead of having to touch all the code, so issues are easier to track down.

Mainly, what applying a microservice architecture ment in terms of the system design is separating the client side with the server side. This division from the front-end and the back-end side is what changes the core structure of the project.

This technique contributed to the project assisting on the benefits of reusability, responsiveness, scalability and versioning.

Improved reusability for a separate API. This means the opportunity of sharing the application logic, allowing other systems to consume this API as a service, for free or for a price.

Better responsiveness for the advantage of using the API to provide the users with responsive feedback. Once the server returns the response, the client side can effortlessly update the view contents.

Greater scalability because modularity means breaking the system into separate modules with a single responsibility, resulting much easier for developers to work independently on separate parts of the project without getting in each other's way. Plus, developers can work more comfortably in a module without the concern of not having a full comprehension of the system behavior.

And enhanced version control for the fact that with separate API and UI projects, it is easier to upgrade, deploy or even revert errors without having to sacrifice the improvements made in the other part of the project.

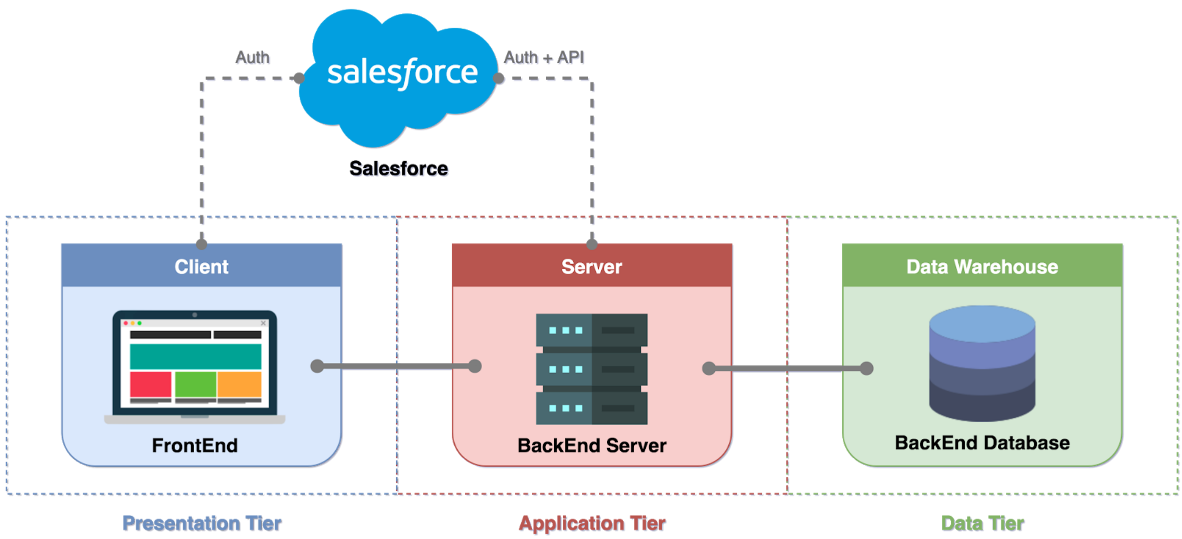


Figure 7 – Logical And Physical Architecture

The system will be accessessible for the client through a web browser, which will be responsible for interacting with the user and communicating with the server.

Then the server responsibility will be to handle all the complex logic of the system by storing and organizing data to ensure everything is working and providing the client side with an API for sending and receiving information.

In accordance, both the client and server will interact with Salesforce. The client just for authentication purposes without handling any tokens and the server for authentication and other API services for interacting with an Org.

Lastly, the server will continuously communicate with a database as a means to persist data and rely on it for future tasks. Correspondingly the database will serve as the Source of Truth regarding the Orgs information and any previous tasks accomplished by the system.

As the figure 7 also shows the system has been accordingly divided in three tiers. This is known in software engineering and web application development as a 3

Tier Architecture, in which this technique is applied to separate the 3 main pieces of the software into isolated environments.

The Presentation Tier is responsible for interacting with a browser and the user, handling the representation of the UI by gathering the information necessary to be displayed and managing the state of the views and user interaction.

The Application Tier is in charge of processing all the business logic for the application, also acting as an intermediary between the presentation and data tier.

And the Data Tier is in simple terms the database management system that persists provides access to the applications data.

Some of the benefits of applying this 3 tier architecture pattern are [13]:

1. It gives you the project the ability to adapt and update the technology stack of one tier, without impacting other areas of the application, hence, it is ready for the future.
2. As a result of the complexity of each tier, usually developers tend to specialize in one area, like for instance, working on the front-end of the application instead of working on the full stack. And this architecture allows for different development teams to each work on their own areas of expertise.
3. Having independence between services provides an ease of maintenance of the code base, since managing presentation code and business logic separately, means that a change to the business logic, for example, does not suppose an impact on the presentation tier.

Followingly, the project has been organized in such a way to avoid duplication of code and any bad patterns, meanwhile it is focused on improving stability and scalability, which will be described in the following sections.

7.2. Front-end

The front-end is the part of a website with which users interact with, meaning that it is the portion of the tool that makes possible the communication between the user and the system.

Then here I will explain what the user will see to communicate with the system, how it will navigate through the views, and finally how this side of the application is designed internally.

7.2.1. User Interface

In this particular section I will present the different pages and views the application incorporates and also shortly outline how the user is expected to interact with it.

7.2.1.1. Home Page

This is the main page of the application, and its goal is to let the user either sign up new Orgs into the system, or to indicate the basic information of the migration it wants to perform. Figure 8 is a screenshot of the page.

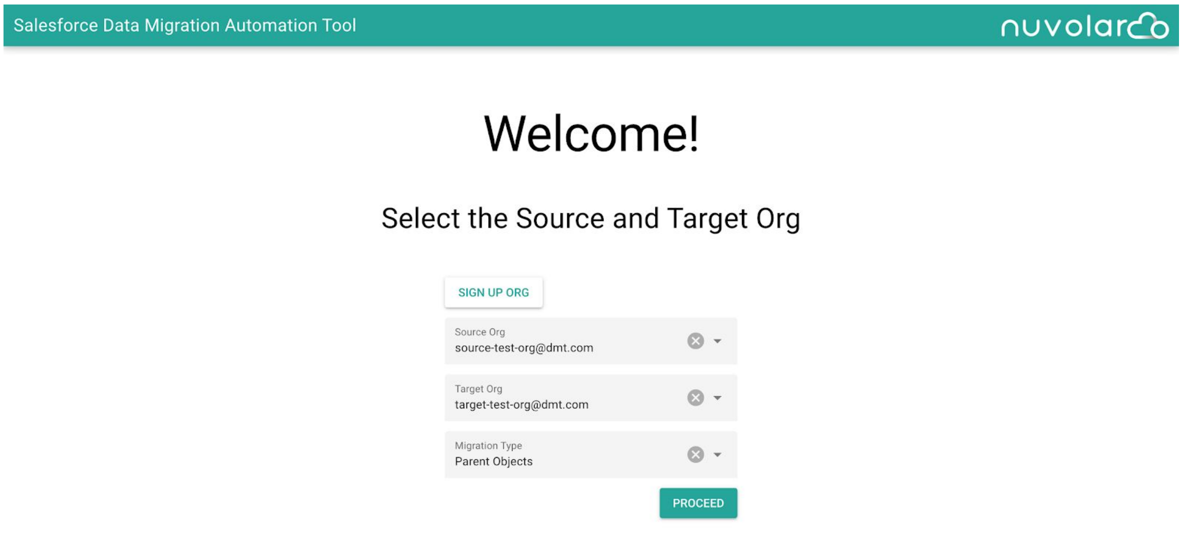


Figure 8 – Home Page

Given the case the user wants to sign up a new Org the dialog that can be seen on figure 9 will appear whenever the user clicks the "Sign Up" button so the user can select if it wants to register an Org that belongs to a production or a sandbox environment.

Then once selected the user will be redirected to the corresponding Salesforce logIn page to authenticate with its credentials to the desired Org and lastly allow the application with the permissions to externally access the authenticated Org.

On the other hand if the user regrets clicking the "Sign Up" button and decides he does not want to sign up any Org then to dismiss the dialog he would only need to click anywhere outside the dialog and it will disappear accordingly.

Alternatively if the user wants to perform a migration, then in order to proceed it should select the origin Org from which it wants to extract data, the corresponding destination Org to which it wants to import the data and the type of migration it wants to perform (to this day there is only one type).

Only once the user has selected all the previous information it will be allowed to click the "Proceed" button and navigate to the next page to continue with the steps necessary to carry out the migration.

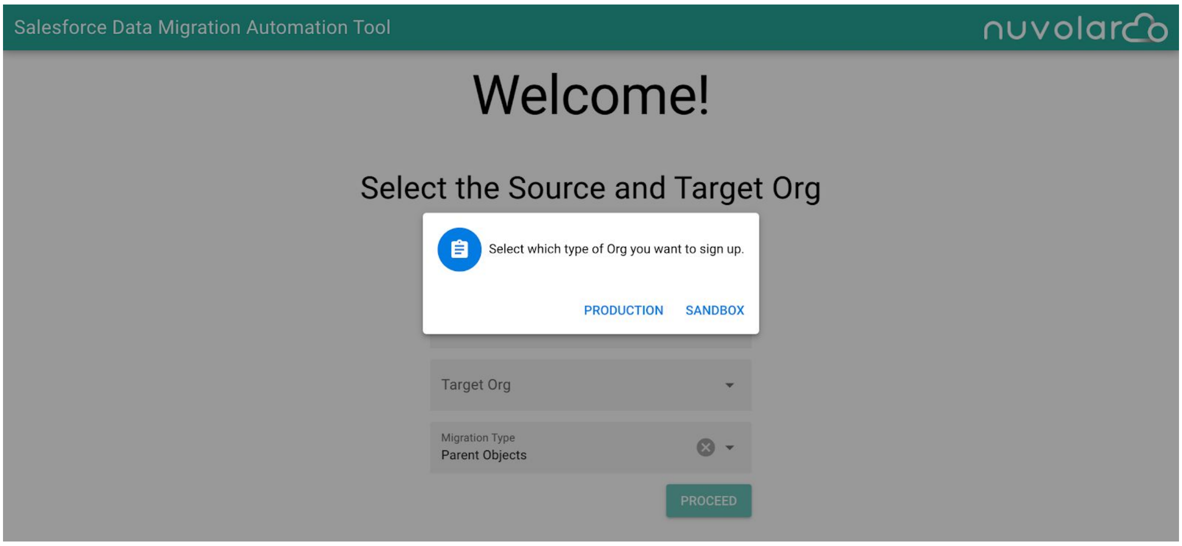


Figure 9 – Home Page Sign Up Dialog

7.2.1.2. Authentication Callback Page

This page is a loading page, and it should be automatically called by Salesforce with a code for requesting the Orgs access token once the user has finished authenticating in the Salesforce page.

Therefore its sole purpose is to send a request to the back-end of the application with the code provided and once the back-end finishes registering the new Org redirecting the user to the Home page indicating if the registration was successful or otherwise.

The view of this page is what figure 10 displays, a loader and a message.

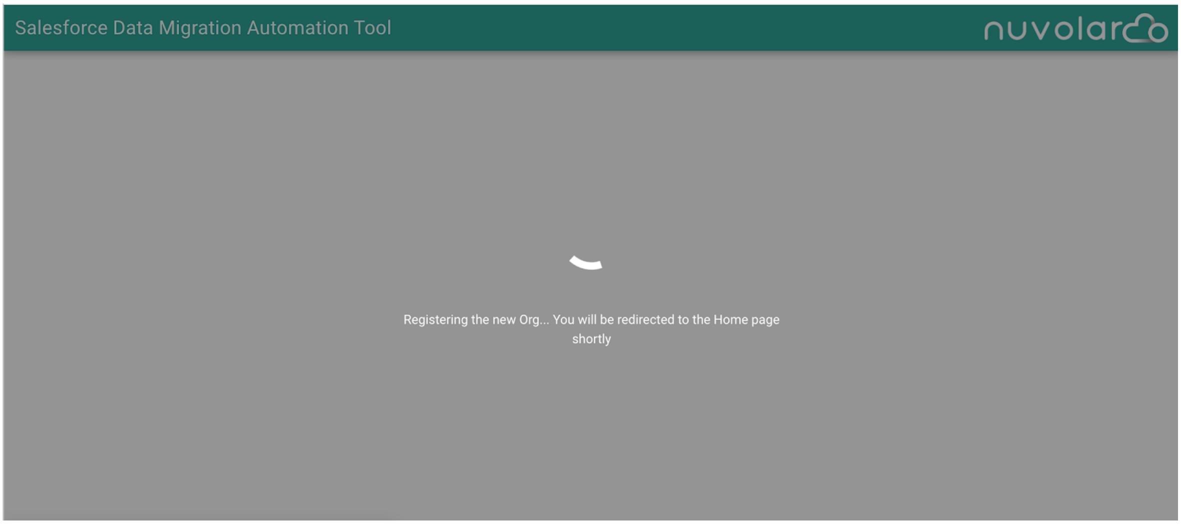


Figure 10 – Parent Objects Migration Page: Main Object Selection step

7.2.1.3. Parent Objects Migration Page

This page is the primary page where the user will perform all the steps necessary to conduct a parent related objects migration. That is where a series of mandatory manual steps need to be carried out by the user in order to provide the system with the information necessary for the automation of the migration.

Therefore this page is represented by a stepper by which the user will need to progress by completing the prompted tasks of each step.

In this way this page is designed by nested views for each step, which are actually represented by components.

7.2.1.3.1. Main Object Selection Step

The Main Object Selection is the first step of the Parent Objects Migration page, hence, as figure 11 illustrated, this is the step with which the user first visualizes the Parent Objects Migration page.

The task to be performed here is really simple, the user only needs to select which is the main object of the Source Org from which it wants to extract the records and the equivalent or matching object of the Target Org to which it wants to import the records.

If the object does not appear then the user can click on the reload button to the right in order to update the corresponding Orgs objects metadata in the system's database.

Once the user has provided the previous information then it will be allowed to continue to the next step.

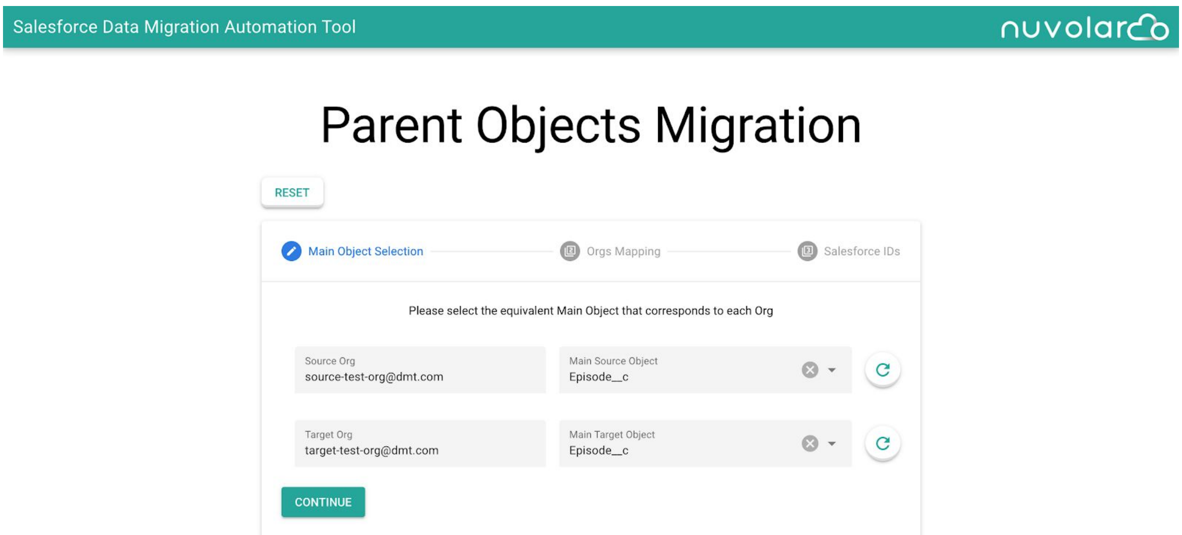


Figure 11 – Parent Objects Migration Page: Main Object Selection step

7.2.1.3.2. Orgs Mapping Step

The next step is the Orgs Mapping, where the user will need to perform the object mapping and the field mapping of all the objects it wants to have involved in the migration.

Figure 12 shows the object mapping, in which the user can map the Source objects with their equivalent Target objects, select which objects it wants to be part of the migration, see the relationships dependency graph between objects and perform the field mapping.

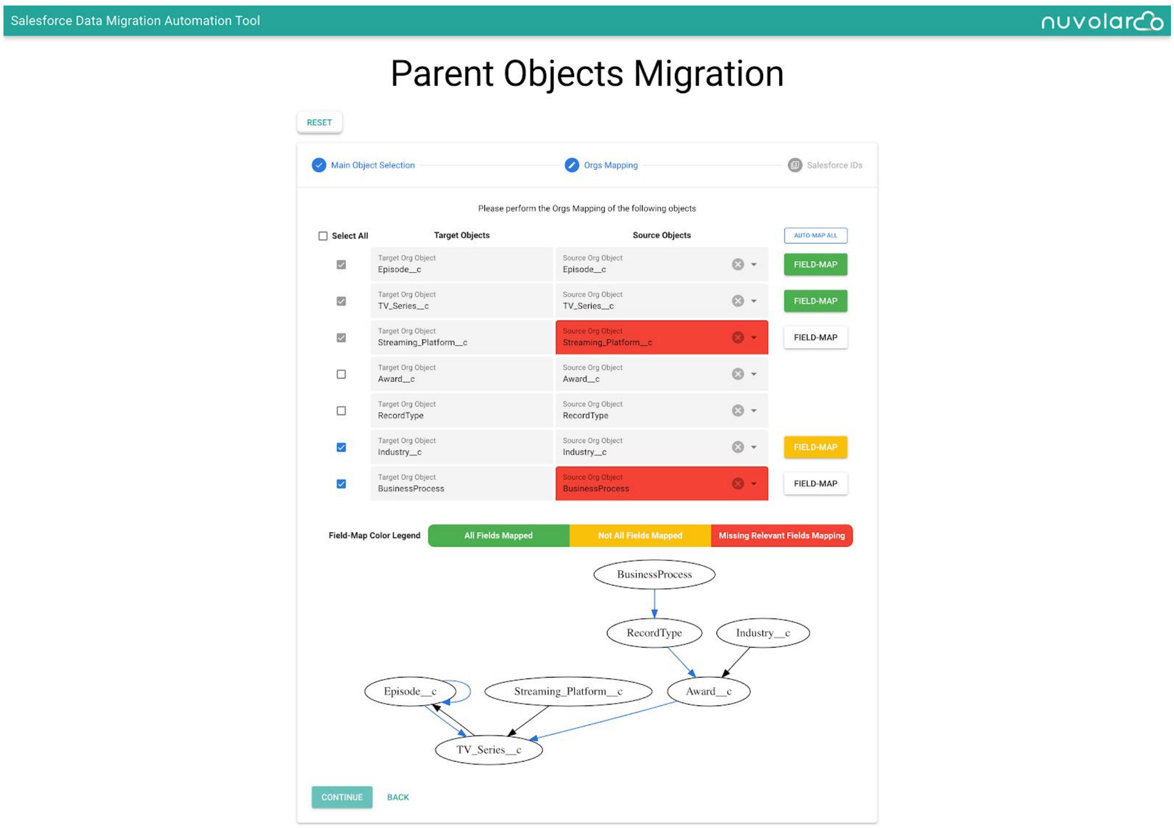


Figure 12 – Parent Objects Migration Page: Orgs Mapping step: Object Mapping

In the objects mapping nested view the user can actually perform the objects field mapping in 2 different ways. The first approach is by clicking the “Auto-Map All” button, which will automatically perform the field mapping of all the selected and mapped objects. However since this auto-mapping is not guaranteed to be accurate the user can always edit it manually.

That is the second approach, in which by clicking an object's “Field-Map” button the field mapping nested view will replace the object mapping nested view, as Figure 13 shows.

Here, similar to the object mapping, the user will be able to manually map theSource object fields with their equivalent Target object fields and select which are the fields it wants to take part in the migration.

Whenever the user desires to return to the object mapping, he only needs to click the “< Object Mapping” button.

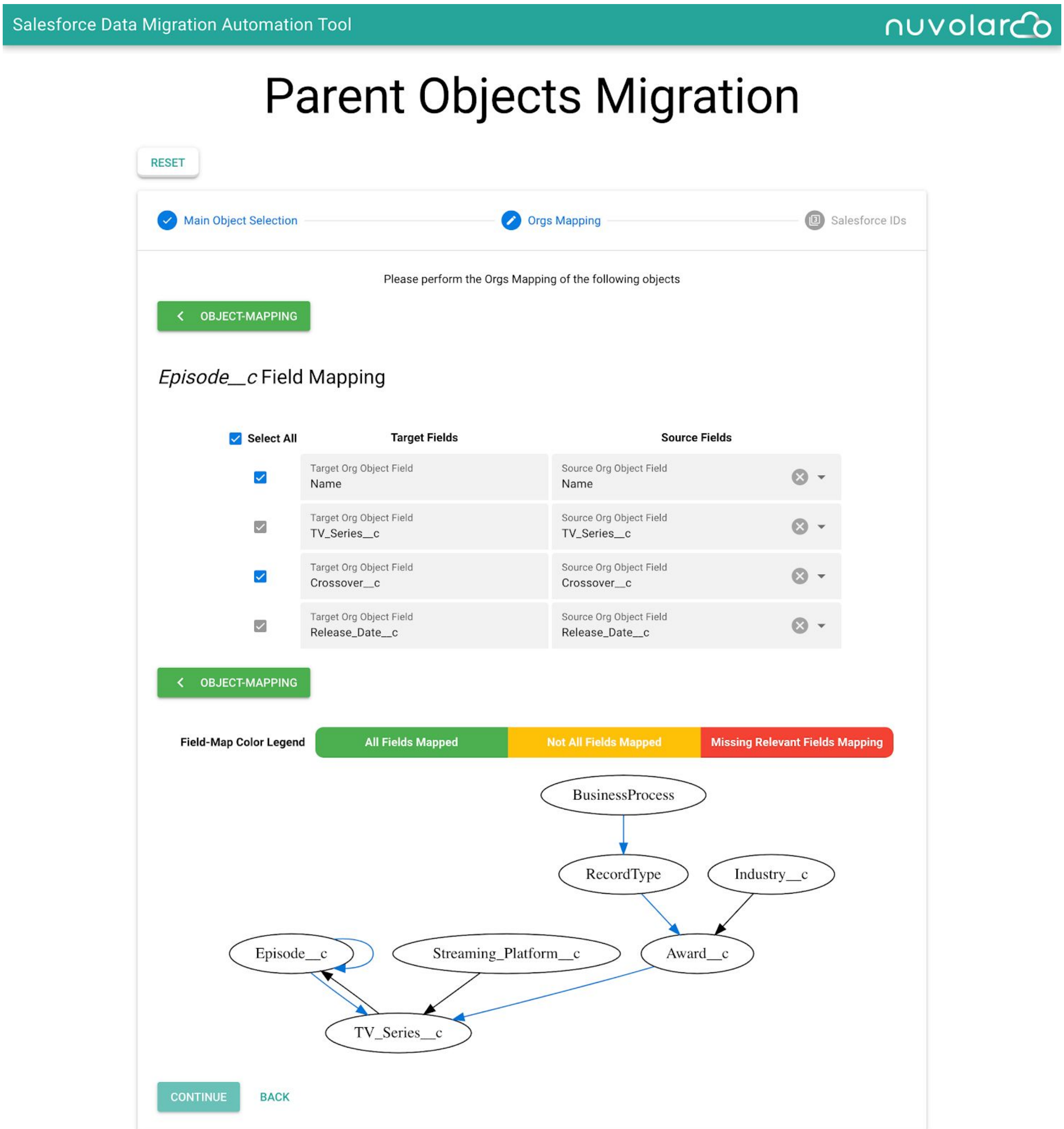


Figure 13 – Parent Objects Migration Page: Orgs Mapping step: Field Mapping

If at any moment the user wants to navigate to the previous step it will only need to click the “Back” button.

Alternatively, once the user has completed the Orgs Mapping it will then be allowed to click the "Continue" button and proceed to the final step.

7.2.1.3.1. Salesforce IDs Step

This last step is where the user is prompted to input all the Source Org record Salesforce IDs intended to be migrated, of course, together with all its parent related object records.

In figure 14 a representation of this nested view can be seen.

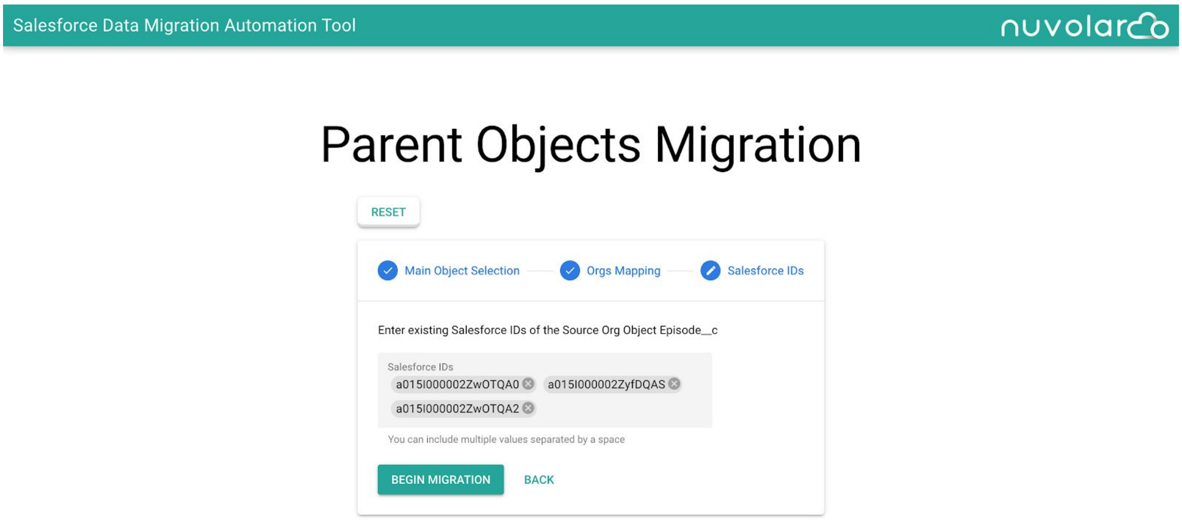


Figure 14 – Parent Objects Migration Page: Salesforce IDs step

Accordingly, once the user has entered at least one record Salesforce ID it will be allowed to click the "Begin Migration" button and as a result the automation of the migration will start.

As a means to provide the user with responsive feedback the dialog illustrated in Figure 15 will appear indicating the automated tasks performed by the system.

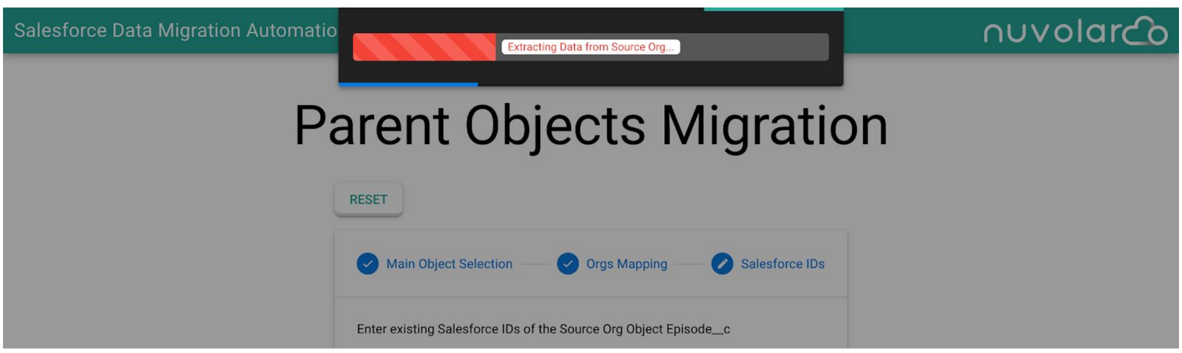


Figure 15 – Migration Automation Dialog with Responsive Feedback

7.2.2. Site Map Navigation

Having explained the role of each view, here, the complete site map navigation of the site will be displayed.

This can be seen in the Figure 16 below, where all the views appear as elements, the arrows indicate the navigation between views and the dotted rectangles represent nested views.

Besides, a mix of the BPMN standard notation was used to indicate which is the first view presented when a particular nested view is shown.

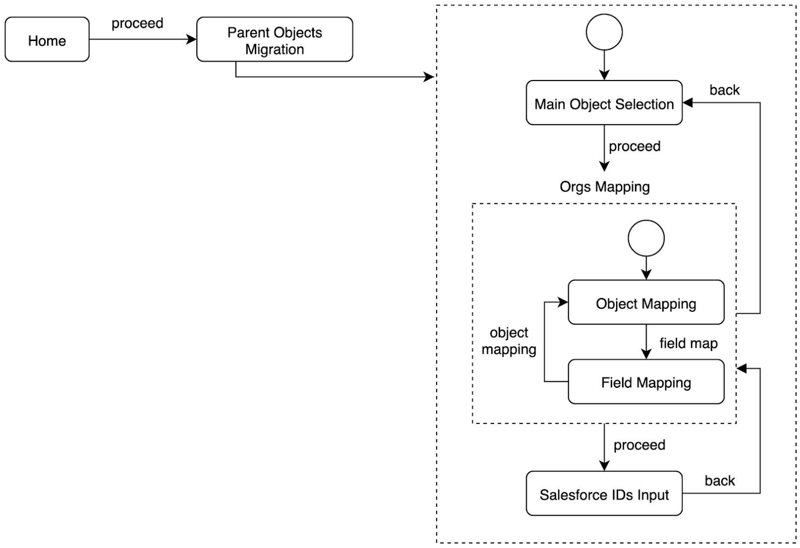


Figure 16 – Migration Automation Dialog with Responsive Feedback

7.2.3. Internal Design

From what I have presented in the previous sections, we can note that the front-end is composed of 3 routes, however these routes involve many nested views, therefore demanding a high level of state management.

In this section I will present the internal design of the front-end previously described and thus have a better understanding of the structure in place to facilitate the state management.

The most important technique for a good state management system is breaking down the front-end into small components. In this way, each component is responsible for managing its own data, state and implementation.

Besides another advantage separating the front-end in components is that in this manner components can be reusable in many parts of the site without dealing with repeated code.

Figure 17 shows the different modules used in the client side. It consists of all the components that make up the front-end and the utils, which assist the components to complete certain tasks.

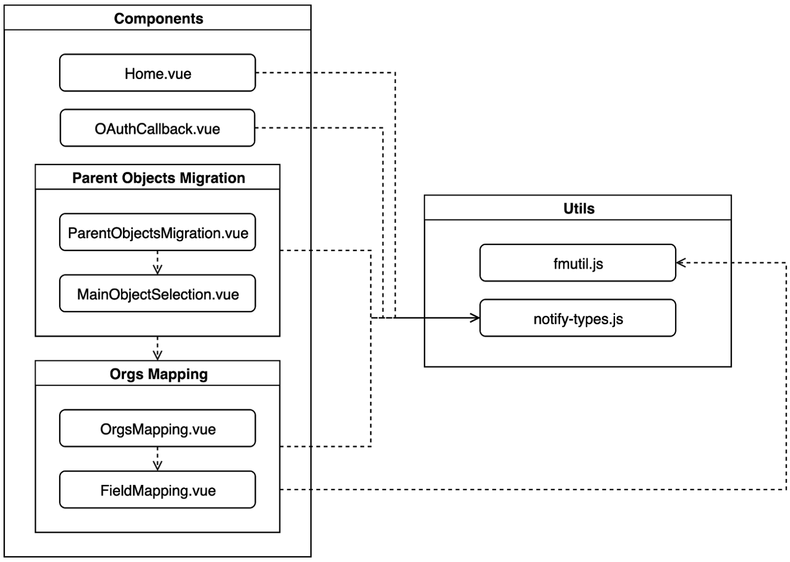


Figure 17 – Front-End Modules

Through this approach the application is organized into nested components, where each component is accountable for its behavior and it can communicate with other components, ergo, delegating tasks and making it easier to maintain the state.

Besides the use of the store pattern [14] is used to to easily manage difficult changes in state between components.

For a better comprehension of this issue the reader can refer to the sequence diagram of figure 18 where an example of how this technique is applied.

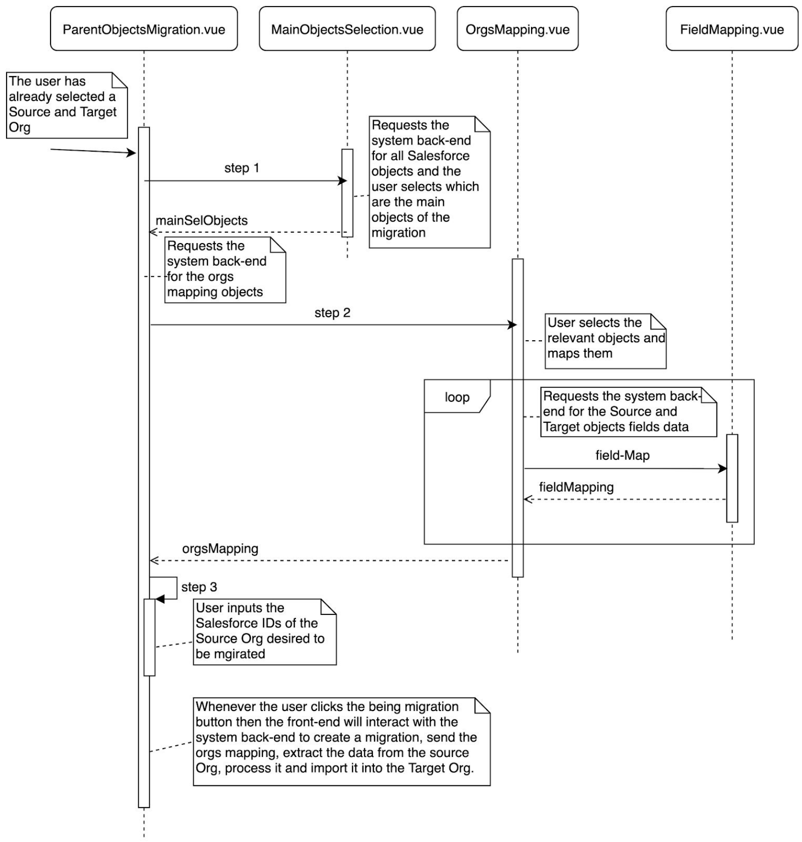


Figure 18 – Front-End Modules

As it can be seen, the work performed by the ParentObjectsMigration.vue component is distributed through the use of other components, enabling the delegation of responsibilities.

7.3. Back-end

Opposite to the front-end, the back-end is the side of the application responsible for storing, managing and providing data required by the front-end to satisfy the users needs.

In the case of the software involved, the back-end is composed of a server and a database. The server acts as the communication channel with the front-end side and the database as a service in which to persist any necessary data.

Consequently in the following sections I will introduce the REST API developed in the server along with its internal design, and then continue with the database conceptual schema.

7.3.1. REST API

Here I will present the different endpoints the server has active for interaction with the client side of the application. Accordingly their method and a description of their role will be available.

Besides, the endpoints are organized into 4 groups:

7.3.1.1. Authentication Endpoints

GET	/salesforce/oAuth2
Request Salesforce for an OAuth logIn URL and returns it.	

GET	/salesforce/token
Request Salesforce an access token, then requests some additional information related to the Org and stores it in the database.	

7.3.1.2. Org Management Endpoints

GET	/orgs
Returns the basic information about all the Orgs present in the database.	

GET	/orgs/{orgID}/objects
If the query reload is true or the objects are not present in the database then it requests Salesforce for all the Orgs {orgID}. It returns all the retrievable objects present in the Org {orgID}.	

7.3.1.3. Orgs Mapping Endpoints

GET	/orgsMapping/parentRelatedObjects/{sourceObjectID}/{targetObjectID}
Responds with all the object mappings information necessary in order to begin the Orgs Mapping*. For the sourceObjectID object returns all the parent related objects that are retrievable from the Org and for the targetObjectID object returns all the parent related objects that satisfy the restrictions necessary to be correctly migrateable. Additionally, it returns the dependency graph of the target parent related objects. *If the object's fields are already stored in the database then it operates with them, otherwise it requests them to Salesforce.	

GET	/orgsMapping/fieldMapping/{sourceObjectID}/{targetObjectID}

Responds with all the objects field mapping information necessary in order to continue with the Orgs Mapping. For the sourceObjectID object returns all the fields that are retrievable from the Org and for the targetObjectID object returns all the fields that satisfy the restrictions necessary to be correctly migrateable.

7.3.1.4. Data Migration Endpoints

POST	/migration
Creates a migration with the requested type and returns the created migration's ID.	

POST	/migration/{migrationID}/orgs-mapping
Stores the Orgs Mapping provided in the migrationID migration.	

GET	/migration/{migrationID}/exportData
Requests Salesforce all the necessary data from the Source Org of the migrationID migration and it stores it in the database.	

GET	/migration/{migrationID}/importData
The system makes any transformations necessary to prepare the data for the import and before beginning the import procedure, performs a verification check with the Target Orgs internal metadata to determine the compatibility of the migration. This includes determining if a compatible order of insertion exists. And lastly, it begins the process of importing data to the Target Org, which if fails it performs a rollback of the inserted data.	

7.3.2. Internal Design

In the server is where all the complex logic of the application resides, since it is responsible for the main purpose of the tool, the automation of the migration.

For this reason the project structure was carried out through a modular approach, as a means to maintain the code organized so that it is easier to manage and therefore to debug, test and build new functionalities.

In this way, the system will also respect the single responsibility principle, by which every module will only have a single responsibility over a particular functionality of the system.

As a result, the back-ends project structure is separated into 8 different entities, as it figure 19 illustrates:

- ❖ Routes – they are in charge of managing the application available routes and assigning them the proper behavior.
- ❖ Controllers – they control the request and response information related to a certain route and manage the logic and error control necessary to ensure the application's expected behavior.
- ❖ Services – they implement all the complex code and therefore do the work related to specific tasks of a topic.
- ❖ API Requests – they are services but exclusively responsible for handling the interaction with Salesforce API.
- ❖ Models – they handle and represent the information of the database as objects by providing getters, queries, setters and instance methods.
- ❖ Helpers – they help the services to smoothly manage complex inner logic.
- ❖ Utils – similarly to helpers, they assist other entities with necessary or useful resources for achieving the intended result.

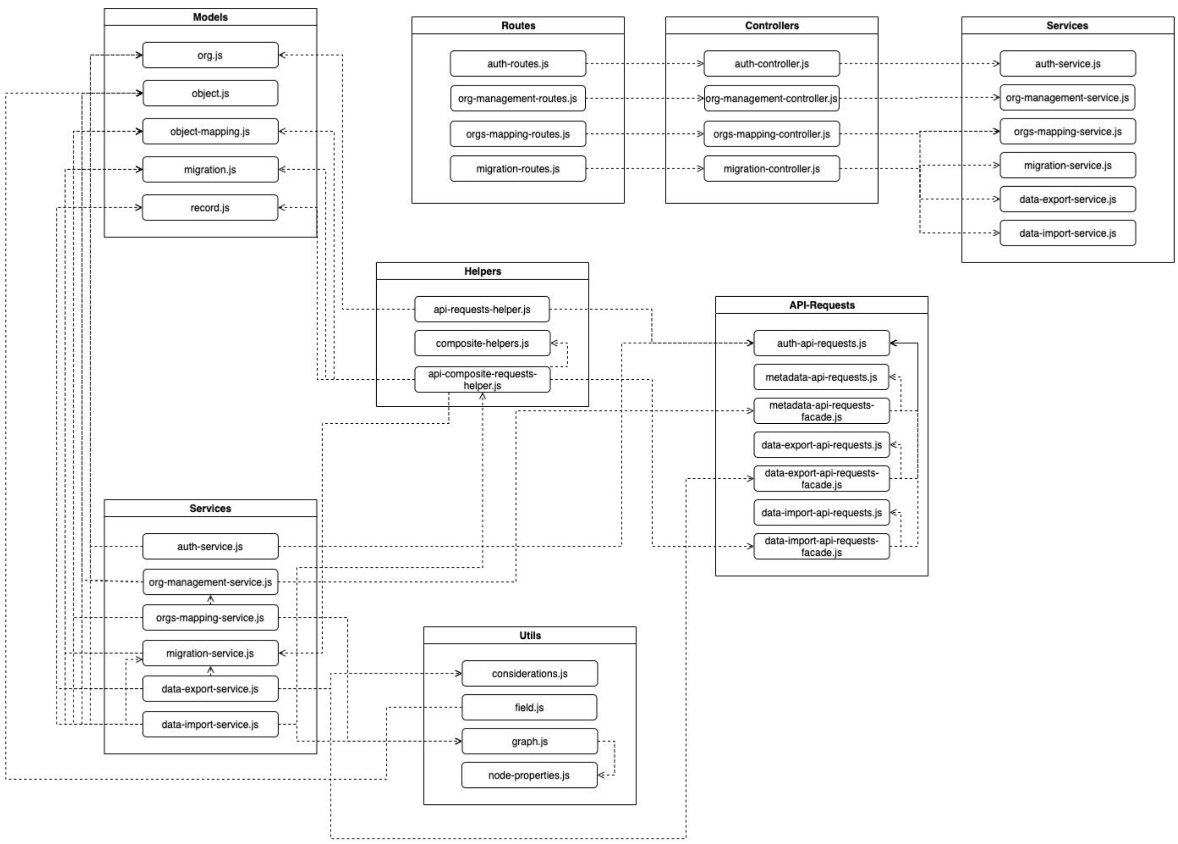


Figure 19 – Back-End Modules

This project organization was also inspired by a lot of research of development best practices together with advice from experienced developers to find a solid project's architecture that has proven to be useful and follow good patterns for avoiding duplication of code, improve stability and potentially, help scale the system services [15].

Accordingly, the manner in which these entities are expected to behave is represented by the following sequence diagram of figure 20.

This is just a simple example, since in more complex procedures, Services actually communicate with other Services, with more than one Model and even get to use Helpers and Utils.

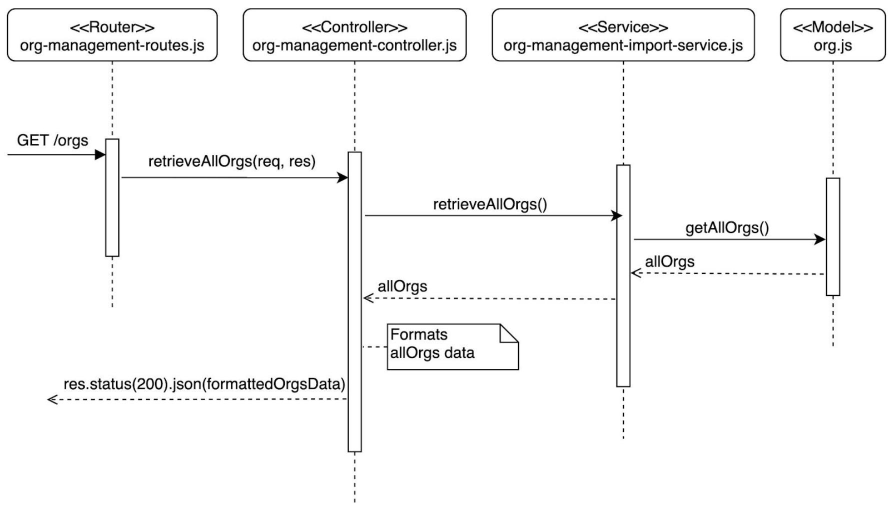


Figure 20 – Sequence Diagram of the Interaction Between Back-End Entities

In addition, the design patterns used in this architecture are:

- Singleton – pattern in which only a single instance of that class can exist. It is used for example in the considerations.js Util module.
- Facade – pattern responsible for masking complex code of other modules, hence, wrapping these modules and offering an easier usage. It is used particularly for the modules of the API Requests entity.
- Dependency Injection – pattern in which an object receives as parameters the objects that it depends on. It is used in all the modules of the Services entity.

7.3.3. Database Physical Schema

In terms of the type of database with which to work in this project, a non-relational database was chosen for many reasons, but mostly for simplicity purposes, as it is easy to use, understand and to set up.

Additionally it is appropriate for agile development, quick sprints and frequent changes, and also has superior performance than relational databases hence, hence, it also brings scalability benefits [16].

Besides these reasons, the best advantage that it provides to this project is its ability for creating flexible data models, allowing me to easily store data in the way my application needs it.

Which additionally offers me the advantage of working with flexible document schemas, ergo, removing me the need to dynamically create tables that replicate the Salesforce Orgs schemas as it would need to be done in a relational database.

7.3.3.1. Collection Schemas

I will represent the collection schemas in a sort of JavaScript object notation where the values will represent the data type stored for each key.

Color Map of the data types:

- Green ⇒ MongoDB objects
- Red ⇒ JavaScript primitive data types
  - [ ] ⇒ represent arrays
  - { } ⇒ represent nested objects
- Blue ⇒ DB models

Orgs Collection

```
Orgs.Schema({
        _id: ObjectId,
        organizationID: String,
        instanceURL: String,
        userID: String,
        username: String,
        accessToken: String,
        refreshToken: String,
        urlID: String,
        objects: [Objects],
        createdAt: ISODate,
        updatedAt: ISODate
})
```

Objects Collection

```
Objects.Schema({
        _id: ObjectId,
        org: Orgs,
        properties: {
                createable: Boolean,
                keyPrefix: String,
                name: String,
                retrieveable: Boolean,
                updateable: Boolean
        },
        fields: [{
```

```
            createable: Boolean,
            name: String,
            nillable: Boolean,
            defaultOnCreate: Boolean,
            referenceTo: [String],
            type: String,
            updateable: Boolean,
            unique: Boolean
      }],
      recordTypes: [{
            active: Boolean,
            available: Boolean,
            defaultRecordTypeMapping: Boolean,
            devloperName: String,
            master: Boolean,
            name: String,
            salesforceID: String
      }],
      createdAt: ISODate,
      updatedAt: ISODate
})
```

ObjectMappings Collection

```
ObjectMappings.Schema({
      _id: ObjectId,
      migration: Migrations,
      sourceObject: Objects,
      targetObject: Objects,
      fieldMapping: [{
            sourceFieldName: String,
            targetFieldName: String,
            sourceFieldReferenceTo: [String],
            targetFieldReferenceTo: [String],
            required: Boolean
      }],
      requiredFieldNames: [String],
      createdAt: ISODate,
      updatedAt: ISODate
})
```

Migrations Collection

```
Migrations.Schema({
	_id: ObjectId,
	orgs: {
		sourceOrg: Orgs,
		targetOrg: Orgs
	},
	mainObject: {
		object: Objects,
		salesforceIDs: [String],
	},
	orgsMapping: [ObjectMappings],
	records: [Records],
	createdAt: ISODate,
	updatedAt: ISODate
})
```

Records Collection

```
Records.Schema({
	_id: ObjectId,
	migration: Migrations,
	object: Objects,
	objectMapping: ObjectMappings,
	IDs: {
		oldID: String,
		newID: String
	},
	skip: Boolean,
	data: {},
	originalData: {},
	oldReferences: {},
	referencedBy: [Records],
	createdAt: ISODate,
	updatedAt: ISODate
})
```

As explained in the conceptual model section of the specification unit, the fields *data*, *originalData* and *oldReferences* contain abstract structures therefore no key-value pairs have been assigned.

7.3.3.1. Identifiers

The identifiers of a database collection are the fields that can be employed to query for a unique record, hence, in a database there should never be 2 or more records that share the same identifier values.

The invariants for the presented db schema are the following:

- Orgs(_id || userID || urlID)
- Objects(_id || (properties.name, org))
- ObjectMappings(_id || (migration, sourceObject || targetObject))
- Migrations(_id)
- Records(_id, (migration, IDs.oldID || IDs.newID))

7.3.3.2. Integrity Rules

In this section I will mention the database integrity rules, which are the constraints embodied in a database for the purpose of maintaining the data integrity.

The following list specifies the required fields for each model that should be populated on creation*:
*the *_id* field is autogenerated

- Orgs(organizationID, instanceURL, userID, username, accessToken, refreshToken, urlID)
- Objects(org, properties)
- ObjectMappings(migration, sourceObject, targetObject, fieldMapping)
- Migrations(orgs)
- Records(migration, object, IDs)

Additionally, as a means to ensure the uniqueness of the identifiers previously listed, indexes were incorporated to the database.

The creation of these unique indexes were created for both single fields like for instance the Orgs collection “userID”, and for compound fields, as for the pair of the Objects collection (“properties.name”, “org”).

The use of indexes offers a better efficiency on the execution of queries. Without indexes, the query must scan every document in a collection for the purpose of returning the documents that match the query statement. On the other hand, with a suitable index in place, the collections indexes are used to limit the number of documents to inspect.

In this fashion, the quality and coherence of the database data is ensured to be maintained.

8. Implementation

As mentioned in the methodology section of this report the development of this project follows an Agile methodology, therefore the implementation was carried out incrementally, per sprints.

This was a great advantage because as the systems behavior criteria was being refined through the sprints, this did not imply significant modifications on the future planification, but only slight modifications on a particular detailed level.

In this unit I will mention the stack of technologies used, together with the libraries employed, afterwards I will reveal in more detail the implementation of the primary behaviors the system is based on and lastly a brief description of the system limitations.

8.1. Technologies

Beginning with the client side, I chose to work with VueJS as the core front-end framework. I made this decision mostly because it is a modern lightweight quickly growing technology that combines the best parts of the two most popular front-end frameworks, AngularJS and ReactJS.

In addition to that, VueJS counts with an extensive and well-described documentation, which together with a large and growing community, smoothes the learning curve and development time of a project.

Moving to the back-end, on the server side I decided to build a web server with NodeJS for reasons such as its great performance and easy scalability, its large and supporting developer community, its popular package management system (NPM), counting with an efficient concurrency handler and the great benefits of easily making use of JSON for the exchange of data with the client.

Also, an additional advantage of working with this stack is that it gives me the benefit of a faster development process by making coding in JavaScript for both the client and server side possible, which is great.

And lastly for the database of all the non-relational databases available I chose to use MongoDB because it is the most popular database for modern apps, which is justified considering it is easy to use, efficient, productive and prepared for general purpose projects. Plus together with mongoose, it is the perfect NoSQL database for this project's requirements.

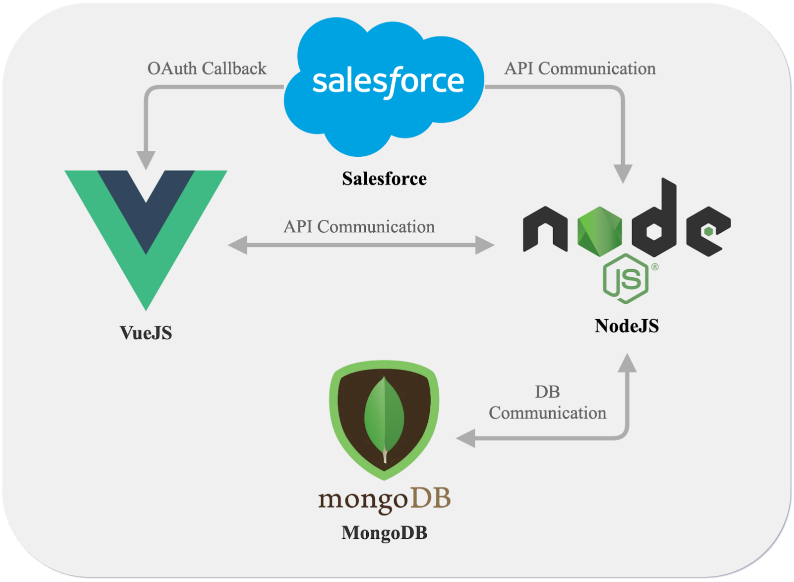


Figure 21 – System Technology Stack

Nextly the libraries used in this project will be presented.

8.1.1. Front-End Libraries

For a cloud-based approach the chosen technologies were quite straightforward:

Quasar – being a VueJS based framework, Quasar was perfect for this project since it allowed me to quickly create this tool's sites by using their built in components and plugins, giving me the opportunity of almost effortlessly providing the user with a comfortable look & feel of a desktop application on the web.

Axios – is a popular promise based HTTP client for the browser and node.js, just perfect for this project allowing to use the same library for the front-end and back-end. Additionally, it is easy to use and comes with amazing features such as automatic transformation to JSON data, enhanced error handling, interceptors and the opportunity of monitoring requests progress.

Viz.js – a library that allows me to effortlessly incorporate Graphviz diagrams on the browser.

Vue-router – is the official router for VueJS, giving me the opportunity of efficiently managing the front-end routes and navigation.

8.1.2. Back-End Libraries

Express – the most known web application framework for making REST APIs with NodeJS.

Axios – HTTP client selected for the same reasons explained in the front-end technologies section.

Bottlejs – as the project incorporates dependency injection, this library is a tiny and powerful dependency injection container that helps handle and organize correctly the dependencies between the project's modules.

Cors – middleware library for Express that eases the process of enabling Cross-Origin Resource Sharing, CORS.

Dotenv – library that assists with locally controlling the configuration of an application that is separate from the code, as for instance, environment variables.

Mongoose – is an Object Data Modeling, ODM, library for MongoDB and Node.js. Its main purpose is to ease the work with pure data from NoSQL documents by making it look like you are working with objects. In addition it makes managing relationships between data much easier and provides schema validation.

Mocha – most popular NodeJS test framework compatible with asynchronous testing. Most commonly used with Chai.

Chai – most popular NodeJS assertion library for testing.

Chai-as-promised – plugin for Chai with a fluent language for asserting JavaScript Promises.

8.2. Primary Implementation Procedures

The purpose of this section is to describe in more detail the techniques employed in the implementation for some of the tool's main functionalities.

8.2.1. The Graph module

As stated in previous units, the software developed in this project is using the DAG data structure to deal with the problem of cycles formed by required dependencies between records.

As a result, the Graph module was created together with the NodeProperties module it utilizes for handling each of the nodes dependencies information.

The algorithms implemented in Graph.js to solve this problem are inspired on the Topological Sort algorithm [17], which consists of linearly ordering the DAGs vertices (or nodes) such that for every edge from a vertex "A" to a vertex "B" (meaning that "A" is referenced by "B", or in other words, "B" is dependent on "A"), then the vertex "A" comes before the vertex "B" in the ordering.

In spite of that, it can happen that a particular graph can have multiple different topological sort orderings. Therefore as a means to meeting the project's non-functional requirement of efficiency, a topological sort algorithm that best fits the order of insertion of records for optimizing Salesforce requests was implemented.

This resulting topological sort algorithm is classified by dependency levels. That is, every level is composed of all the nodes that depend on at least one node of the previous levels and none of the following levels. Figure 22 nicely illustrates this concept.

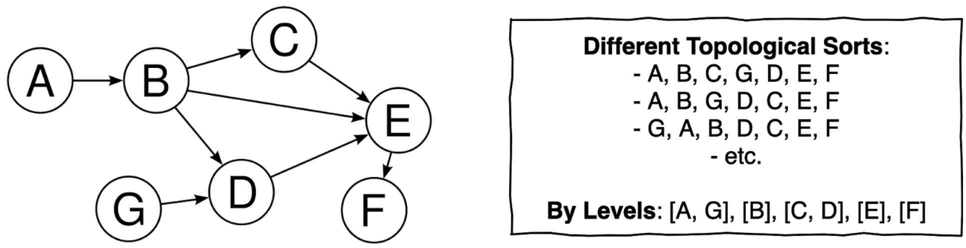


Figure 22 – Topological Sort Solutions

With this in place, the Graph module has a vital dutty on identifying if a migration is feasible, which is carried out by looking at the relationships between records. In addition the graph module contains other algorithms that demand working with parent related nodes or anything related to graph theory terminology.

As a consequence, the Graph module is not only used for determining the order of insertion, but for instance, is also present in other sections of the code, as in the extraction of data from the Source Org, since it easies the management of working with parent records.

8.2.2. Optimization of Salesforce Requests

As a means of satisfying the non-functional requirement of efficiency in which a productive use of resources is requested the implementation of multiple tasks were carried out in particular manners.

8.2.2.1. Salesforce Requests for Metadata

All requests related to an Orgs metadata are only performed when necessary, that is to say, if the metadata information does not exist in the database then the system will request the metadata and save it in the database.

However if the metadata already exists in the database the default behavior of the system is using the stored metadata unless it is explicitly prompted to make a request to Salesforce.

8.2.2.2. Import, Update and Deletion of Migration Records

For the purpose of successfully populating the Target Org with data, or conversely performing a rollback, in the most efficient way possible, not only the migration is analyzed to verify its feasibility, but also all the Salesforce requests involved are carried out through a particular approach with the goal of minimizing the number of API Requests.

That is because for instance, when inserting multiple records different approaches can be carried out. The most basic approach would be to perform a request per each record. However this is not a good approach if the Salesforce API provides better ways of handling this kind of situation [18].

As a consequence, with the objective of implementing the most efficient approach possible for the import of data to the Target Org, the system structures and organizes requests to make use of Salesforce Composite Resources [19], with which the system can combine multiple Salesforce API resources into a single request.

The technique employed is wrapping SObject Collection [20] requests into Composite requests [21].

SObject Collections allows you to execute actions on multiple records in one request. In the best case scenario this implies being able to accomplish 200 operations in 1 request. On the contrary, as a result of restrictions in the request, in the worst case scenario 10 operations can be executed with the cost of 1 request.

On the other hand, Composite requests allow you to execute a maximum of 25 requests in a single request, by which the entire request will count as a single call towards the API limits, which is exactly what I'm looking for.

This implies that combining both resources and making a composite request made up of SObject Collection requests, considering the mean of the best and worst case scenarios of each resource, (200+10)/2 x (25 + 1)/2) = 1365 operations can be performed as a single request. Which is 1365 times better than 1 operation for a single request.

For this reason this approach was selected for the system's implementation, regardless of the additional complexity it adds to the management and creation of the requests. Accordingly to smooth the handling of these types of requests, the composite helper modules were created.

8.3. System Limitations

In this project there was not a requirement related to the minimum amount of records a migration should support, however the implementation was carried out to uphold the maximum amount of records possible.

As a result the system limitations were not tested, however because of the system's implementation, the theoretical limitations of the amount of records involved in a migration is equivalent to the Target Org storage capacity and API requests limits.

9. Validation Testing

In this unit I will outline the tests conducted during the development of this project with the intention of evaluating and finding out whether the developed software meets the specified requirements described in the scope unit or not.

But before getting into that I would like to mention that testing the software was in fact one of the most difficult tasks of the project, particularly in terms of automated testing. The reason for this is that all the system functionalities highly rely either on the data stored in the database or on the data to be requested from Salesforce.

Accordingly, one might think that the easy solution is to use stub objects that can simulate the behavior for both the database and Salesforce requests. Nevertheless this is not an easy approach either, as the challenge of the problem is more closely related to the creation of good samples of fake data for the sake of taking into consideration a diversity of test cases and simulating scenarios that truly replicates real life data migration samples.

The difficulty of this task is given not only by the planning and organization necessary to generate false useful and consistent data, but also maintaining the data quality in the database, as there are many relationships between records and it is critical that they are well preserved so that the system can function as intended.

With the intention of making this point clear, I will now provide an example of why it would be highly inefficient and tremendously time-consuming to attempt to simulate a test scenario.

Given the case I wanted to create an integration test for verifying if a migration is compatible, for starters I should begin by manually creating false records that represent the Source and Target Org in the Org model. Then I would need to think about all their corresponding objects and fields for both the Source and Target Org, along with their metadata.

Consequently I would need to have an already manually performed Orgs Mapping of the Source and Target Orgs given a main object. And lastly create a whole set of false records matching the established Source and Target Orgs objects and fields metadata so that the migration can take place.

This approach demands a lot of work and effort for something that could easily be replaced by system tests through the creation of a testing environment. Now the only inconvenience is that no automation of these test cases would exist. Nonetheless it definitely sounds a much better time efficient and more realistic approach, so it is what I decided to go with.

Ironically, the problem this software embodies for creating automated tests is somehow similar to the problem it resolves.

9.1. Testing Environment

In order to create a proper testing environment I had 2 main tasks, setting up 2 Salesforce Orgs, for the purpose of using one as a Source Org and the other as a Target Org, and then a database for testing only with which to execute the system tests and the integration tests.

The first step was to plan which tests I would like to execute so that I could set up the Orgs with compatible metadata and data to create the system tests. As a consequence I created a list of system tests to perform through a white box testing strategy, in which I take into account real scenarios of the end user along with the systems internal behavior and its potentially conflictive steps as a means to ensure that the desired behavior is achieved.

Once this was done I began building the Salesforce testing environment by creating the 2 Orgs and producing similar objects schemas for each Org. And then populate the Source Org with the records to be used during the tests along with some records in the Target Org just for replicating behavior of unique fields and real scenarios.

Counting with this setup I was then ready to conduct the system tests, which will be presented in the following sections.

9.2. Functional Requirements Tests

Having in mind the previous section in which it was explained why testing this tool is a sophisticated task, when possible unit and integration tests were built, however, most of the tests are carried out as system tests.

9.2.1. Automated Tests

All the tests of this section belong to the back-end, as the front-end tests are performed manually as system tests through a black box testing approach, that is not taking into account any inner code implementation logic, and in consequence investigating the system's behavior by attempting edge cases, or alternatively, just simulating how the end user would interact with the system.

The reason for this testing strategy being applied to the front-end is because front-end tests are mostly graphical, due to that not only the state should be guaranteed, but it is most crucial that the reactivity system that keeps the views updated on data modifications is working correctly.

Also as in the back-end is where all the complex application code resides, automated tests were only performed for this end of the tool.

9.2.1.1. Unit Tests

Unit testing is a strategy by which individual units of the application code are tested with the purpose of validating each unit performed as expected.

The following table shows the number of unit test that performed per functional requirement:

Functional Requirement	Number of Unit Tests
Org Auth Management	3
Metadata Management	5
Org Mapping Management	12
Migrations Records Input	4
Migration Automation	33

9.2.1.2. Integration Tests

Integration tests are a technique by which individual units are combined and tested as a group with the intention of revealing errors in the interaction between the integrated units.

And as a consequence of the difficulties previously explained, integration tests were only carried for the database models, as a way to ensure that they handle database operations satisfactorily.

The following table shows the number of integration tests performed per Model:

Model	Number of Integration Tests
Org	15
Object	2
Object Mapping	4
Migration	8
Record	6

9.2.2. Manual System Tests

In this project the manual tests are the most relevant type of tests that can be conducted. This is for the previously explained reason by which integration tests are not easy to create and sometimes not even possible.

For this purpose, all the manual tests are carried out as system tests, in which the software is tested as a whole. In this way, system tests will serve the purpose of manually testing the integration between the system, the database and Salesforce.

The following set of tables describe the system tests performed per Use Case:

Use Case #1: Sign Up Org		
Scenario	**Success**	**Observations**
The user registers a new Org	✓	The system registers the Org correctly
The user registers an existing Org	✓	The system updates the existing Org record information

Use Case #2: Auth Life Validity		
Scenario	**Success**	**Observations**
The access token expires so the system automatically requests a new one	✓	The system registers the new token information correctly

Use Case #3: Basic Metadata Extraction		
Scenario	**Success**	**Observations**
The user is going to select the main object of the migration and the Orgs objects have not been loaded yet	✓	The system requests Salesforce the Orgs objects and stores them correctly
The user is going to select the main object of the migration and the Orgs objects have already been loaded	✓	The system just returns the Orgs objects

The user reloads the Org objects to update their metadata	✓	The system requests Salesforce the Orgs objects and replaces them the existing ones

Use Case #4: Parent Related Objects Identification		
Scenario	**Success**	**Observations**
The user proceeds to the Orgs Mapping step having selected simple main objects	✓	The system correctly retrieves the parent related objects of the Source and Target Org
The user proceeds to the Orgs Mapping step having selected a target Org main object that does not satisfy the conditions to be migrateable	✓	The system returns an error message

Use Case #5: Object Mapping		
Scenario	**Success**	**Observations**
The user correctly performs an object mapping	✓	The system correctly stores the object mapping and when selecting the user is only presented with the remaining objects
The user is performing an object mapping but it is missing or has wrongly performed some mappings	✓	The system correctly maintains the state and is updated in the view

Use Case #6: Field Mapping		
Scenario	**Success**	**Observations**
The user correctly performs an objects field mapping	✓	The system correctly stores the objects field mapping and when selecting the user is only

		presented with the compatible and remaining object fields
The user is performing an objects field mapping but it is missing or has wrongly performed some mappings	✓	The system correctly maintains the state and is updated in the view

Use Case #7: Compatible Mapping Selection		
Scenario	Success	Observations
The user is performing an objects field mapping	✓	When selecting to map an objects field, the user is only presented with the remaining and compatible objects fields

Use Case #8: Smart Relevant Objects Selection		
Scenario	Success	Observations
The user is performing the object mapping and selects an object as relevant	✓	The system correctly updates the relevant objects
The user is performing the object mapping and deselects an object as relevant	✓	The system correctly updates the relevant objects
The user is performing the object mapping and selects all objects as relevant	✓	The system correctly updates the relevant objects
The user is performing the object mapping and deselects all the possible objects as relevant	✓	The system correctly updates the relevant objects

Use Case #9: Object Auto-Mapping		
Scenario	Success	Observations

The user proceeds to the Orgs Mapping step of main objects that contain different prefixes in their names	✓	The system correctly auto maps the objects by names independently of their prefixes

Use Case #10: Field Auto-Mapping		
Scenario	**Success**	**Observations**
The user clicks for the first time on the field mapping button of an object that contains fields with different prefixes in its names	✓	The system correctly auto maps the fields by names independently of their prefixes

Use Case #11: Source Main Object Salesforce IDs		
Scenario	**Success**	**Observations**
The user fails to insert any string as Salesforce ID that is not composed of 18 digits	✓	The system correctly maintains this behavior
The user can only insert Salesforce IDs with the same prefix as the first inserted ID	✓	The system correctly maintains this behavior
The user inserts multiple Salesforce IDs separated by spaces	✓	The system correctly stores all the entered Salesforce IDs
The user attempts to insert a repeated Salesforce ID	✓	The system does not insert the repeated Salesforce ID

Use Case #12.1: Parent Related Data Extraction		
Scenario	**Success**	**Observations**
The user attempts to migrate records with required relationships dependency of more	✓	The system correctly extracts all the parent related data

than one level		
The user attempts to migrate records without relationships	✓	The system correctly extracts all the data
The user attempts to migrate records that reference unique fields of the Target Org	✓	The system correctly extracts all the parent related data and handles correctly the unique field

Use Case #12.2: Verify Migrations viability		
Scenario	Success	Observations
The user attempts to migrate records with cycles between the required relationships	✓	The system returns an error message
The user attempts to migrate records with cycles between the optional relationships	✓	The system correctly performs the migration
The user attempts to migrate a record with an optional Source Org object field that is not populated with data but it is required in the Target Org object field	✓	The system returns an error message

Use Case #12.3: Determine Order of Insertion		
Scenario	Success	Observations
The user attempts to migrate records with simple relationships	✓	The system correctly determines the order of insertion
The user attempts to migrate records with required relationships of multiple levels	✓	The system correctly determines the order of insertion
The user attempts to migrate a record with cycles between the	✓	The system returns an error message

required relationships

Use Case #12.4: Import Data		
Scenario	**Success**	**Observations**
The user attempts a simple migration	✓	The system correctly imports the data
The user attempts a migration with multiple levels of insertion	✓	The system correctly imports the data
The user attempts a migration with records that violate active Target Org restrictions	✓	The system rollbacks any imports and returns an error message

9.3. Non-Functional Requirements Tests

Correspondingly, as a means to evaluate if the system non-functional requirements have been satisfied, the following tables show the tests performed per non-functional requirement based on their acceptance criteria:

Non-Functional Requirement #1: Efficiency		
Acceptance Criteria	**Success**	**Observations**
The migration should not migrate Users as they can suppose costs.	✓	The system correctly imports the data
Additionally requests to the Target Org should always be optimized in a way in which the amount of total requests without being optimized is bigger than the amount of total requests with the optimization	✓	The system correctly imports the data

Non-Functional Requirement #2: Robustness		
Scenario	**Success**	**Observations**

Whenever an error occurs the user should be presented with a descriptive error that points to where or why the error has occurred	✓	The system correctly auto maps the fields by names independently of their prefixes

Non-Functional Requirement #3: Usability		
Scenario	Success	Observations
5 different users will be assigned a set of tasks which they will need to perform thinking out loud so that any confusion can be recorded.	✓	All 5 users completed the set of tasks intuitively. Some users mentioned the tooltips, the state represented by colors and the buttons being only clickable when possible as helpful.

10. Sustainability

The 3 dimensions of sustainability are the economic, the environmental and the social impact of a project. Therefore the sustainability matrix can be seen below with the marks considered for each cell, which will be justified in detail in the following sections.

	Environmental	Economic	Social
PPP	9	9	10
Exploitation	8	7	9

*Each cell is over 10

In the following sections each cell of the matrix will be covered in detail.

10.1. Self-Assessment of the Domain of Sustainability

Just as people say that the dog is the man's best friend, people also think that machines are the computer engineer's best friend, and this is a concept so internalized in society's mindset that we often tend to forget the differences between a computer engineer and a machine.

The main role for which a computer engineer is wanted is for its ability to get a software or hardware product to perform the function we expect from it. And just as we expect and trust that a software or hardware product behaves according to the purpose for which it was created, we expect the same from a computer engineer.

When in need of a computer engineer, a person that questions the project's repercussions is not desired, but a person capable of performing the associated work without questioning its morality, in other words, a machine.

Luckily, or rather, as a result of previous failed experiences, we are beginning to realize the power affiliated to a computer engineer, the power to create and associate a specific behavior desired to a piece of software or hardware. And we well know that with great power comes great responsibility.

Step by step, computer engineer's are being educated to become more aware of their responsibility as professionals, and the responsibility associated with the repercussions of their work, the environmental, economic and social sustainability.

The sustainability problem is real and it is up to us, computer engineers, to solve it. We often blame large multinational companies for not having environmental commitments, but if we are the ones who ignore these same commitments when working for them then we are no better than them.

The problem is not easy at all, it is really complex, since as humans, we do not only work only for pleasure, but because it puts a roof over our heads, therefore before developing a project I believe that we, computer engineers, should ask ourselves questions like the following: How is my roof going to affect other people's roofs? And not only today's people's roofs, but also the future's people roofs.

The 100% sustainable project does not exist, we have to be realistic, it is essential to create sustainable solutions in the 3 dimensions, and in cases where it is simply not possible, we should at least try to reduce the associated repercussions.

For these reasons I find that computer engineers are much more than machines, and that we have the moral responsibility of analyzing the potential repercussions of carrying out a project.

10.2. Environmental Dimension

10.2.1. Regarding PPP

The only way in which the preparation of this project can have negative repercussions to the environment is through energy consumption, since no waste is generated. Therefore, the amount of energy this project benefits from in order to work is the measure that will be taken into consideration to determine the environmental impact of the bachelor's thesis.

Now, I'm not going to make an estimation of the ecological footprint of the project realization since I believe that making this calculation would not be realistic as a result of the amount of speculation it demands.
In order to determine the amount of energy consumed during the PPP it would require for me to know not only the computational energy used during all the implementation and testing process, which highly depends on the amount of data I use during each procedure, but also knowing the source from which the energy used is produced.

In contrary, I can still think about ways in which to minimize this amount of computational energy used by applying the following criteria:

- Always performs the first tests with small amounts of data to ensure that the current approach passes the test, and therefore avoid wasting non-productive energy with large amounts of data.
- Perform the least amounts of tests with large amounts of data possible, and when doing them, only perform in the cases in which I feel confident that the tool will succeed, or in the cases in which there is a need to analyze the behavior of the tool with large amounts of data.

10.2.2. Regarding Useful Life

The worst way in which the project ecological footprint of this project throughout its useful life could be increased is by giving it a bad use, in other words, by constantly attempting to perform incompatible data migrations, which means doing un-useful work and consuming un-productive energy.

Having this in mind what I consider the core of the resources consumed are based on the amount of mistakes made during the process of a data migration, meaning that if a person is trying to perform incompatible data migrations or the tool that performs the data migration does not have a good implemented behavior for error control then the amount of resources will definitely have a negative impact.

I cannot specify how currently other tools manage this problem, or even if they try managing it since it would demand really specific knowledge about its internal implementation.

What I can try is to build a better tool with the most favourable environmental behaviors towards encountered errors in data migrations. For example, a key concept in a data migration is the speed, as a consequence, this promotes the parallelization of tasks, hence at the same time it would mean doing a lot of unnecessary work when an error occurs during one of the parallelized processes.

Still one could argue that the consumption of energy that this tool might use is quite small in comparison to other procedures that take large amounts of energy, therefore, worrying too much about its consumption and making the tool slower is not a good approach since it will probably just encourage the use of other tools.

Wrapping up, the main way in which this tool can help the environment is through reducing the computational power and energy consumed during long, error-prone and manual data migrations, and this is what really matters.

10.3. Economic Dimension

10.3.1. Regarding PPP

For this phase the relevant consumption to analyse are the resources, either human or material, used during the entire project and their corresponding costs.

Having in mind that the total cost estimation for the completion of project is of 14,500, it is relevant to take into account that a large learning and research curve had to be covered in the realization of the project on account of the lack of knowledge in the technologies required. Also being a junior developer and having little experience in project planning and management probably increased the costs of the project.

After all I still would not criticize this aspect a lot since considering the development it would have taken less time to an experienced team, the costs associated would have been higher given that their pay per hour would also be higher.

10.3.2. Regarding Useful Life

Currently, in similar software that addresses the same problem as this project, the economic issues of costs are solved by providing both a free-tire and pay per use option. In this way they can not only cover the costs involved but also win profit from the tool, therefore it has a true economic potential if desired.

Now, in my case, for my project's viability plan and for my solution as an MVP, the approach previously explained is not possible yet since it was not its purpose, hence, in the case in which its development progresses, it perfectly has the potential of being monetized.

Alternatively, the way in which I see the economical benefit and viability plan of this tool is by reducing the amount of time spent by workers on doing the manual tedious work of a data migration. With the use of this tool, workers would be able to spend their work hours in more useful and productive work which they are only capable of doing, and not machines, rather than wasting their time and effort to something that could be done faster and easier through automation.

In conclusion, a better use of worker resources will be available with the incorporation of this tool, which results in more productive costs for the company.

10.4. Social Dimension

10.4.1. Regarding PPP

The personal impact that carrying out this project has had on me is mostly in terms of personal growth, meaning that not only I will achieve to build an MVP data migration tool, but I will also learn new technologies used in real life projects, go through the complete life cycle of a project creation and delivery, and in addition, performing this thesis in a company, also benefits me by kicking-off my work experience and exploit many advantages related to real world projects.

Working on this project did much more to me personally, since it completely changed my daily routine from a student perspective to an intern with a full-time job with the advantage of a first salary. With this opportunity I'm also getting to understand many concepts practically through the approach of learning by doing, providing me with more knowledge, plus tips and best practices from the company's employees.

Another impact this project had on me is by updating my vision of life in relation to the job of my career, computer engineering. Now I have been able to see and feel how real life teams work, the steps they go through in the development of a project and understand, be aware that mistakes are something really common in software development, when I actually had a vision much more strict of this aspect which pressured me into thinking that I was not good enough.

Now I know that in reality things are really different and that before developing a project it is not necessary to previously have all the knowledge necessary to start implementing it, but that part of the development process is to learn and fail.

10.4.2. Regarding Useful Life

The impact of this project creation is mostly directed to the company's employees. By automating the task of data migrations the workers time can be better spent, also their corresponding stress level resulting from deadlines and accumulated workload is reduced, therefore contributing to their personal well-being by avoiding unnecessary headaches or stress and eradicating their annoyance feelings of having to perform difficult and tedious manual work.

Managers are also positively affected by paying productive salaries, as a result of their employees workload being reduced. Additionally, they can also use this tool to populate a sample environment for demos to potential clients.
Also the end-clients are also benefited since this tool may allow to quickly extract the necessary data from a user's support case and therefore reproduce the reported error and be able to receive quicker feedback.

This is actually one of the main purposes for this project's development, to increase the positive social impact and improve the quality of work life of the workers, meaning, there is a real social need for this project's creation apart from its economic advantages.

Now, regarding the negative social implications there is not much to say, but the only potential group target I can think about (I say potential referring to the future once it is not only an MVP), is about professionals strictly dedicated to perform data migrations which salaries depend on it. This would affect them negatively since in the worst case i could mean their job loss, or at the least a lower pay.

Thinking realistically though I would not even consider this a truly real negative impact due that really complex and manual data migrations should not be performed regularly and the goal is to eventually be able to automate all kinds of data migration, then, this might be a warning to the type of persons that their only job is to perform data migrations so that they start looking for alternative job opportunities. Plus if they have the knowledge to handle such complex processes that demand strong knowledge and really good abilities they should be able to find a new job easily.

10.5. Conclusions

This project's intentions are completely honest, and there clearly is a problem waiting to be addressed and a solution waiting to be performed.

The environmental, economic and social harm is almost nonexistent and from the problem associated, the benefits it generates are more than enough to consider this project viable and sustainable.

Just as a last statement I would like to add that as an MVP this project does not cover implications relating to privacy of user data, but future improvements a data scrambler could be incorporated for the cases in which sensible data is being migrated.

11. Technical Competencies

In this unit the technical competencies associated with the project during the Bachelor's thesis enrollment will be presented.

11.CES1

CES1: To develop, maintain and evaluate software services and systems which satisfy all user requirements, which behave reliably and efficiently, with a reasonable development and maintenance and which satisfy the rules for quality applying the theories, principles, methods and practices of Software Engineering.

11.CES1.1

CES1.1: Develop, maintain and evaluate complex and/or critical systems and services. [In depth]

Through this report I have explained the topics of Salesforce Orgs and data migrations, I have described the problem in hand and justified the reasons why is complex undergoing through the process of a data migration together with the numerous considerations that are required in order to accomplish a successful data migration.

With all that being said, it is important to note how not only the topics that involve the project are complex, but also the project itself for being responsible for developing, maintaining and evaluating all these previously mentioned factors.

Besides before achieving the tool built without any previous knowledge about Salesforce, an advanced and elaborated research, planning and organization had to be conducted to understand and determine the viability of the project's scope and the feasibility of its functionalities, with the goal of reaching the approached solution implemented.

In addition, developing this tool meant constantly dealing with challenges and making critical decisions towards the system's behavior, as different criterias reach to different development approaches, and at all times, the procedures selected were carefully considered for achieving the best results.

11.CES1.3

CES1.3: To identify, evaluate and manage potential risks related to software building which could arise. [A little bit]

Many potential risks were identified and evaluated during the project's development, to the point in which the most important potential risks were any risks that could have been overlooked that could suppose very strict limitations towards the software's development.

Also, delivery related risks were critical for the project, as the tool consists of a continuous flow of functionalities in order to work. Thus being the system so dependent on each functionality for the system to work as a whole and meeting its purpose, required a great amount of time and effort to complete all the tasks involved on time for the project's deadline.

Additionally operational risks such as lack of sufficient training were taken into account, since overlooks where easily a possibility in this project as a result of the lack of documentation available relating to the topic in hand.

For these reasons an Agile methodology was selected for the project, so that at any moment in time I could manage and quickly adapt to any potential risks without risking jeopardizing the entire project's planning.

11.CES1.5

CES1.5: To specify, design, implement and evaluate databases. [Enough]

Every piece of this project is core for the tool to function, as the front-end steps require manual assistance from the user, and the back-end contains all the sophisticated logic to communicate with Salesforce Orgs and handle the automation of steps necessary to perform the migration.

However what is critical for the system to function is the database, as this one acts as the Source of Truth for the server. Furthermore, in order for the server to work efficiently and effectively it highly depends on the design of the non-relational database used in the project.

Therefore as the sections 6.3.3. about the Database Conceptual Schema and 7 explain, many considerations took place related to the specification, design, implementation and evaluation of the database the system built in this project encompasses.

11.CES2

CES2: To value the client needs and specify the software requirements to satisfy these needs, reconciling conflictive objectives through searching acceptable compromises, taking into account the limitations related to the cost, time, already developed systems and organizations.

11.CES2.1

CES2.1: To define and manage the requirements of a software system. [Enough]

Defining the projects requirements was a complex task itself, as not only a really good comprehension of the topic in hand was required at the time, but a key factor towards the requirements criteria was the amount of time available to complete and deliver the project on time.

That is many requirements were not a possibility given the projects deadline, hence, an in depth brainstorming of the most basic and relevant requirements was made to define this projects requirements.

This task was specifically difficult since it required brainstorming the functionalities of the ideal desired tool so that they could be broken down into smaller essential functionalities for a basic tool.

This meant prioritizing the core functionalities of the project, which actually resulted in being really useful by providing a lot of knowledge towards the feasibility of future enhancements.

In this way, this project requirements describe the results of this reflection about the core functional and non-functional requirements to build a data migration automation tool between Salesforce Orgs.

11.CES3

CES3: To identify and analyse problems; design, develop, implement, verify and document software solutions having an adequate knowledge about the current theories, models and techniques.

11.CES3.1

CES3.1: Develop services and multimedia applications. [A little bit]

This technical competence was selected “A little bit” for the purpose of not dealing with multimedia content. On the other hand it includes a really refined analysis for the problem related to data migrations, plus it also designs, develops and implements this tool as a service to deal with the problem data migrations arise.

Hence, as demonstrated in a variety of sections throughout this report, a lot of research towards the current techniques and best practices for conducting data migrations and building this tool were investigated to have a solid understanding about the current techniques and approaches on the subject of data migrations.

12. Project Retrospective

12.1. Closing Remarks

This project came to happen as a result of the trouble generated to companies developing software on top of Salesforce by the recurrent need of having to perform complex data migrations between Salesforce Orgs.

The consequences of frequently undergoing through data migrations can cost companies a considerable amount of development time and even lead to the employment of bad practices such as postponing necessary data migrations or vaguely performing them inducing to future development repercussions.

The trouble of data migrations is somewhat related to the phrase John Wooden once said “If you don’t have time to do it right, when will you have time to do it over?”.

As a result, this project aims to remove the concerns resulting from dealing with arduous data migrations with the goal of building a simple and intuitive tool capable of automating the procedure of a data migration between Salesforce Orgs.

Counting with this tool would imply considerably reducing the workload associated with data migrations, hence, offering many benefits to the company’s development efficiency.

In approximately a timespan of 5 months, I was expected to accomplish this goal with no Salesforce and data migrations prior knowledge. This required me to study, learn, research, estimate, plan, analyze, specify, design, implement, test and document all the project’s work described in this report.

Additionally, with the help of my teammates at Nuvolar Works, I also got to benefit from their years of experience in software engineering by incorporating their working methodologies into the project, receiving useful feedback and brainstorming sophisticated proposals together.

Yet, at this moment, having completed the project, I can gladly and proudly affirm that the project goals, objectives and requirements have been successfully achieved.

Furthermore, despite having met the product owner needs, working on this project has also brought me a lot of personal growth from the professional perspective, as a Junior Software Engineer, and from the personal perspective, as a person.

Developing this project I got to learn new technologies, methodologies and best practices, which at the same time has provided me with a lot of knowledge and experience as well as great relationships with my company colleagues, who have supported me, given me advice and inspired me during this project's development.

For being my first time working in a company I feel highly satisfied about the results obtained, as not only I got to start from zero on a project with a huge potential, but working with the motivation of there existing a real necessity for this tool has been incredibly rewarding and motivational.

If I were to do this project again, I would work more on the front-end design, as having prioritized the back-end I only performed wireframes on paper and skipped making proper mockups for ensuring the best user experience possible. Additionally, I would improve the implementation of certain modules and apply additional design patterns that allow for a better scalability.

Having said this I always find important to note that developing software requires great skills, and I am not referring to technical skills, but the emotional skills that are not usually discussed, as for instance, the constant persistence and encouragement to face new challenges, to feel comfortable and not overwhelmed with uncertainties and everything yet to learn and to reinvent yourself by keeping up with new technologies and methodologies.

I believe it is relevant to remind about this for the reason that the known image of software development is pressing the keyboard keys and clicking the mouse in the right spots and magically a software is built, however in reality, a lot of hours are dedicated to planning, thinking, refactoring, debugging and figuring out why the implemented code is not working as expected.

And what I love about this concept is the fact that when debugging it sometimes can be particularly difficult and frustrating to discover the reason or the piece of code which is making the system not to work as expected, hence, at this moment it is easy to blame an external service for malfunctioning, however in most cases it is always the developers fault. And I find this beautiful because it reminds me about honesty and how we should always blame ourselves before blaming others.

4 years ago I decided to study Informatics Engineering because it simply amazed me how computers, computers programs and websites worked.

Today, I am a Junior Software Engineer and I aspire to build great tools that fill a necessity and therefore are useful and helpful for end users and therefore contribute in making the world a better place.

12.2. Main Challenges

This section outlines a list of the main challenges that have been encountered through the project's development:

- Salesforce Research & Documentation – Salesforce counts with a great deal of documentation, however this documentation is not organized for the purpose of this project, therefore finding a list considerations the tool should have in mind was more of a journey by which I had to play around with Salesforce Orgs to learn and gain knowledge about its behavior and try to discover any edge cases present not easily found through the Salesforce documentation.
  At the same time it also happened sometimes that due to the immense amount of documentation Salesforce needs to maintain, some information is vaguely explained or even outdated, leading to unnecessary difficulties in the implementation of certain methods.

- Adapting to Special Salesforce Behaviors – as a consequence of the previous challenge, this made the project deal with a degree of uncertainty towards Salesforce special behaviors that would have to be discovered through the project's development.
  This is a challenge, because as a result of the tool completely relying on Salesforce, it implies that the tool would have to adapt and include the behavior necessary to deal with these special mentioned behaviors, which luckily only took place to a small degree.

- Preserving Record Relations – as a concept explained in depth through this report, preserving record relations was the core challenge of the project implementation, as it is the main requirement of the automation of data migrations and it incorporated a large amount of sophistication to the system.
  It specifically required a lot of organization, detailed planning and great management in the system's behavior, as many responsibilities are involved to ensure the sustainability of this task.

- Decision Making – many difficult decisions had to be made during the project development, from the definition of project requirements to the design of the system and its implementation. The way in which these assignments were carried could have a great impact on future work, hence really meticulous decisions had to be made.

- Testing – testing should be quick, easy and a smooth process, however this is not always possible. As mentioned in the validation testing unit, the main challenge of testing this system relies on the automation of tests, and because of this, not only a numerous amount of testing approaches were attempted, but in order to ensure the correct execution of the system a numerous list of manual system steps has to be performed.

12.3. Integration of Knowledge

In this section I will present a list of the subjects coursed during the Bachelor Degree that have provided me with knowledge used in the development of this project.

PRO1 & PRO2 – Programming 1 & Programming 2
These two subjects were my introduction to programming, providing me the building blocks of coding through organized, reusable, clean and efficient coding practices.

M1 – Mathematics 1
M1 taught me the basics of Graph theory, which has a fundamental part in the tool developed in this project, concretely is used to handle the dependencies between Salesforce objects required relationships.

EDA – Data Structures and Algorithms
I was introduced to the concepts of efficiency and cost of an algorithm along with data structures such as maps, sets and lists.
In the development of this project these topics were highly considered and compared with its alternatives in order to produce the highest efficiency possible.

BD – Databases
The NoSQL database of this tool is essential in order for the tool to function correctly, and this subject taught me the core topics of relational and NoSQL databases along with the SQL language.

SO – Operating Systems
The influence SO has on this project is that it taught me the proper way of using a terminal, to get comfortable using it and reading the documentation necessary to exploit its functionalities, being a fundamental tool in the creation, set up and version control of the project.

IDI – Interaction and Interface Design
IDI introduced me to the essentials of interface design and usability, specially relevant in the system's front-end, where all the design principles play an important role.

IES – Introduction to Software Engineering
This particular subject provided me with the knowledge of the specification and design sections of my thesis.

XC – Computer Networks
In XC I had the opportunity to learn and understand local networks becoming one of the justifications of the approached solution of this project.

PROP – Programming Projects
Influential for learning about classical design patterns and putting into practice

concepts of Object Oriented programming, implementing a three-tier architecture, handling errors and exceptions, debugging and performing unit tests.

PAR – Parallelism
From PAR I learnt to differentiate parallelism from concurrency and their corresponding strengths and weaknesses, helping in the implementation of concurrent processes in the system.

AS – Software Architecture
AS taught me knowledge about design patterns and the concept of a tier architecture and how the system should communicate between tiers.

ASW – Web Applications and Services
ASW was a key subject in order to improve my comprehension towards web services and how to design, implement or work with service APIs, the deployment process of projects, and getting to know many different modern technologies.

DBD – Database Design
The relevance of DBD in this project is mostly shown in the design chosen for the current database, the structure of my NoSQL documents, and the implementation of queries and their corresponding cost I make use of in each case.

ER – Requirements Engineering
Completely influential for identification and definition of stakeholders, the system requirement analysis and its corresponding goals and objectives.

GPS – Software Project Management
From GPS I gained experience on working with a project's documentation, estimating a project's costs and was introduced to useful tools like Trello.

PES – Software Engineering Project
Without a doubt PES was one of the most significant and influential subjects in which along with my teammates we planned and built a complete project practicing the Agile methodology and going through the different roles of Agile Teams. I also got to work with new technologies and methodologies, and adapt to team meetings documentation as a part of the development of a project.

CAP – Advanced Programming Concepts
Fundamental in order to have a more in depth understanding of prototype based languages like JavaScript as well as new programming concepts such as closures.

C – Cryptography
By understanding how RSA encryption works and having previously generated RSA keys it allowed me to rapidly generate the RSA keys necessary in order to use the Https protocol.

12.4. Future Work

The last point I want to make is stating some enhancements that could be applied to this project if more time would have been provided:

- Session Management – it would be nice to implement session management in the tool so different users can only access their registered Orgs, and therefore deploy the tool to the cloud meeting security and privacy requirements.
- Handling Org and API Limits – right now the technique applied is to use the least amount of API resources possible, however it could be enhanced by making the system aware of the remaining Org API requests or the Orgs data storage space. In this way the system can not only determine if a migration is feasible, but if the Org has the sufficient resources available to be carried out.
- Summary of Migrated Records – right now once the migration finishes a message is displayed and that is it. It would be nice to have a summary about the records involved in the migration with their corresponding Source and Target Org IDs. Also it could be useful being able to export this data to a CSV for logging purposes, that is to say, to keep track of the inserted records.
- Saving and Reusing an Orgs Mapping – currently the Orgs Mapping is performed in every data migration, and even though some steps are automated it can happen that for certain data migrations a specific Orgs Mapping is desired, hence, storing a particular Orgs Mapping would be quicker for later uses. The only inconvenience about this is dealing with updated metadata, as all the previous saved Orgs Mapping would need to be updated.
- Automatic Update of Metadata – currently the user has to manually refresh the Orgs metadata as the system does not verify for any modifications, it would be nice for the system to handle this automatically so the user does not have to remember if any modifications were made to the Orgs metadata.

13. References

[1] “CRM 101: What Is CRM?” *Salesforce.com*, www.salesforce.com/crm/what-is-crm/

[2] “The 6 Biggest Benefits of CRM.” *Salesforce.com*, www.salesforce.com/hub/crm/benefits-of-crm/

[3] “Salesforce” *Wikipedia*, Wikimedia Foundation, en.wikipedia.org/wiki/Salesforce.

“Salesforce Tutorial: What Is, Introduction, Architecture, Apex.” *Guru99*, www.guru99.com/salesforce-tutorial.html

[4] “Read the Latest Studies and Reports from the World's Best Known Business Analysts.” *Salesforce.com*, www.salesforce.com/company/recognition/analyst-reports/

[5] “Best CRM Software 2020: Compare Reviews on +300 CRMs”, www.g2.com/categories/crm

[6] “Salesforce Instances vs Orgs vs Environments”, FocusOnForce, https://focusonforce.com/platform/salesforce-instances-vs-orgs-vs-environments/

[7] “Understanding Relationship Fields and Polymorphic Fields”, Salesforce Developers, https://developer.salesforce.com/docs/atlas.en-us.soql_sosl.meta/soql_sosl/sforce_api_calls_soql_relationships_and_polymorph_keys.htm

[8] “Contact Fields”, Salesforce Help, https://help.salesforce.com/apex/HTViewHelpDoc?id=contacts_fields.htm

[9] Garret Alley. “What is Database Migration?”, Alooma, 1 March 2019, https://www.alooma.com/blog/what-is-database-migration

[10] “Top 5 Salesforce Integration Tools For Data Migration And The Best Practices To Accomplish It”, DotSquares, https://www.dotsquares.com/news_events/top-5-salesforce-integration-tools-for-data-migration-and-the-best-practices-to-accomplish-it/

[11] “Choosing a Method for Importing Data”, Salesforce Help, https://help.salesforce.com/articleView?id=import_which_data_import_tool.htm&type=5

[12] “What's the Difference? Agile vs Scrum vs Waterfall vs Kanban.” Smartsheet, www.smartsheet.com/agile-vs-scrum-vs-waterfall-vs-kanban

[13] “5 Benefits of a 3-Tier Architecture”, Izenda, 14 March 2017, https://www.izenda.com/5-benefits-3-tier-architecture/

[14] “Store Management”, Vue, https://vuejs.org/v2/guide/state-management.html

[15] Sam Quinn, “Bulletproof NodeJS project Architecture”, 28 July 2019, https://dev.to/santypk4/bulletproof-node-js-project-architecture-4epf

[16] “When To Use A NoSQL Database”, MongoDB, https://www.mongodb.com/scale/when-to-use-nosql-database

[17] “Topological Sorting”, Wikipedia, 19 May 2020, https://en.wikipedia.org/wiki/Topological_sorting

[18] “Which API Do I Use?”, Salesforce Help, https://help.salesforce.com/articleView?id=integrate_what_is_api.htm&type=5

[19] “REST API Developer Guide – Composite Resources”, Salesforce Developers https://developer.salesforce.com/docs/atlas.en-us.api_rest.meta/api_rest/resources_composite.htm

[20] “REST API Developer Guide – SObject Collections”, Salesforce Developers, https://developer.salesforce.com/docs/atlas.en-us.api_rest.meta/api_rest/resources_composite_sobjects_collections.htm

[21] “REST API Developer Guide – Composite”, Salesforce Developers, https://developer.salesforce.com/docs/atlas.en-us.api_rest.meta/api_rest/resources_composite_composite.htm