POLITECNICO DI TORINO

Master's Degree thesis in Computer Science

Functional analysis and implementation of B2B e-commerce on Salesforce B2B Commerce platform

Relator

Prof. Giorgio Bruno

Candidate

Sofia Munari

A.Y. 2020/2021

INDEX

ABSTRACT 3

ACKNOWLEDGEMENTS 4

INTRODUCTION 5

1. WHAT IS B2B? 7

2. MAIN CHARACTERISTICS OF B2B and HOW IT DIFFERS FROM B2C 8

3. B2B MARKET AND TRENDS 10

4. SALESFORCE B2B COMMERCE 13

4.1 SALESFORCE B2B LIGHTNING 15

4.2 SALESFORCE B2B COMMERCE – Technical Characteristics 16

5. A PRACTICAL EXAMPLE OF A B2B PROJECT ON SALESFORCE B2B LIGHTING COMMERCE 35

5.1 PROBLEM EXPLANATION 36

5.2 REQUIREMENTS 37

5.3 USER REGISTRATION AND MANAGEMENT 37

5.4 PRODUCTS, CATALOG AND PRICES MANAGEMENT 48

5.5 CART AND CHECKOUT MANAGEMENT 56

6. GENERAL PROJECT DETAILS 71

7. CONCLUSION 74

8. APPENDIX 78

Code 1: Contact Point Address 78

Code 2: Reset Password page 82

Code3: Login Form 84

Code 4: Self Registration 85

Code 5: User Profile 89

Code 6: Check Inventory 90

Code 7: Calculate Shipping Cost 92

Code 8: Calculate Taxes 94

Flow 1: Payment Flow 94

Flow 2: User Import Flow 95

Flow 3: Order Export 95

Flow 4: Order Summary 96

Flow 5: Update Fulfilled Orders 96

BIBLIOGRAPHY 97

ABSTRACT

The main purpose of this thesis is to present the core aspects of Salesforce B2B Commerce technology, how it is designed, how it works and how to exploit all its functionalities. Moreover, the thesis will deeply dive into the presentation of an e-commerce website implementation for a real company. This project has been one of the very first use of the newest Salesforce B2B Commerce on Lightning Experience in Italy and in Europe too. The technology is forecast to grow exponentially in the next few years since it is already striking the market and it has already imposed upon the American market. The script is divided in three sections. The first part will simply be an introduction of B2B, it is not the purpose of the work to present all its peculiarities, but mostly to give a clear explanation of what is it and why it is important to distinguish it from B2C. The second section will explain the design and the functionalities of Salesforce B2B Commerce platform. At last it will be presented an accurate analysis of a real world scenario of a company in need of a new e-commerce website. The solution is built through a previous analysis exploiting software engineering design and then through the main characteristics, configurations and pieces of code used to produce the expected result.

ACKNOWLEDGEMENTS

I would like to thank my relator, Professor Giorgio Bruno for supporting and assisting me throughout this project and for his insightful comments and suggestions on how to approach the problem and present the solution.

I wish to also thank my work team and Deloitte Digital for having taught me a lot in these few months and having given me the possibility to take part in this experience.

Moreover, I am deeply grateful to my parents, who have supported my studies in all these years. Thanks for having taught me the importance of hard work and for having believed in me at my lowest and at my highest. They have inspired me to follow my passions and they have always been present in any important decision of my life. I would also like to thank my brother, who has been my constant landmark and my biggest supporter.

A final thought goes to my closest friends who have shared these last 5 years with me, thanks for having made my University experience the best time of my life.

INTRODUCTION

“What we need to do is always lean into the future; when the world changes around you and when it changes against you – what used to be a tail wind is now a head wind – you have to lean into that and figure out what to do because complaining isn’t a strategy.” (Bezos, 2013)

It has become clear that we are surrounded by a continuously evolving world and we are facing an exponential growth of technology, therefore companies can either embrace the changes and dominate the market or slowly disappear. Indeed consumers have changed their habits and companies who want to survive must adapt to it. Companies must change their way of doing marketing, their way of reaching out to clients and mostly they must offer a new purchase experience that satisfies the modern customers’ expectations. This radical mutation of the buying experience has been mostly significant in the B2C market, which represents all the transactions between a vendor and a final user. In fact, many companies has decided to move part of their business online or even some companies are directly born online (we can think of some banks born natively as an online service). Online sales means no need to rent a store, no need to pay for employers to run the store, no need to pay electricity and everything that comes from having a physical shop to maintain. This has obviously led many B2C businesses to switch partially or totally their traditional way of selling to an online one. But what about Business to Business market? B2B stands for all those transactions that happen between two companies, so final clients are still people; however, they buy for the company and not for themselves. We have a totally different type of sails experience, because we are talking about few high price transactions, smaller amount of clients and long term relationships with the vendor. B2B is indeed a completely different problem from B2C and we must face it according to its own rules. What remains absolutely true in B2B commerce too, is the fact that also companies who buy from other companies expect a totally new experience. They expect to be able to buy what they need without having to call the vendor, they expect to receive specific targeted marketing and discounts: the possibility to purchase online is not an optional anymore, it is considered a commodity. In the following pages we will see exactly what we mean for B2B, how it differs from B2C and which are its key points. Then we will shortly analyze how B2B is evolving in relation to youngest business owner and in a post Covid era that has undoubtedly speeded up the evolution towards online services.

We will see many McKenzie’s surveys that state how much is growing the desire and the utilization of websites to make purchases also on B2B and we will see how this might impact the general business. Buyers are paying much more attention to their purchase experience and they might even choose a product over another one based on that. B2B e-commerce has become a basic instrument for companies to reach out to possible clients, offer unique experiences and create long term relationships. These platforms help companies gather much more information about their clients, analyze it and improve their services, but they also allow them to integrate all the services, from the CRM to the OMS, so that they all share data about clients and they can build an overall complete user experience. This first part doesn’t want to be a deep explanation of B2B or its marketing rules, but it aims to give a clear representation of what is the problem that B2B companies are facing and why they should quickly adapt to the changing trend of their market.

Then we will discuss one of the possible B2B commerce solutions that is striking the market nowadays, which is salesforce B2B commerce. It is a new software product that is natively built and designed to support the implementation of commerce for business to business companies. We will see how the platform is designed, what types of advantages it offers and how it can be useful to companies. The goal of the discussion is to deeply understand the product and the logic behind it, which are the standard solutions offered and which parts can be customized by the company. It is impossible to explain all the platform functionalities and features, therefore the focus will not be on giving too much technical details, but mostly to show how the platform is built to answer B2B problems. We will see how the platform deals with each of the B2B key points, in order to really understand the structure behind an e-commerce website. We will also deeply analyze the database structure through class diagrams, the different processes and features that the platform offers and the main characteristics of the Checkout process, which obviously represents the heart of a commerce experience.

In the second part of the discussion we will see practical implementation of a B2B e-commerce on Salesforce B2B commerce platform. We will analyze the project, and explain step by step the requirements and user stories of each part, the data model used, the processes structure and activity flow and also the actual implementation of the most interesting functions. This part wants to be much more technical than the previous one and will exactly show how a typical B2B problem can be tackled and which solutions we decided to implement in this specific situation. This will also give the possibility to analyze how much of standard parts are actually used compare to how much we needed to customize. Moreover, it will offer the possibility to discuss the qualities of the platform, but also its weakness and the features that should be improved. It is important to keep in mind that this is a very new software and compare to its specular B2C solution is still extremely behind. Having said this, it is now time to start the discussion and dive into this incredibly new and exciting world of evolving technologies and ever-changing business, how one is influenced by the other, how important it is to embrace the first to support the second. There is no way to survive in the business if you do not adapt to the customer expectations and there is no way to lean into the future without welcoming the newest technologies.

1. WHAT IS B2B?

Business-to-Business (B2B) refers to a form of transaction conducted by two companies such as a manufacturer and a wholesaler, or a wholesaler and a retailer, so it mainly concerns companies that provide products or services to other companies. A typical example is the old fashion manufacturing industry (like steel industry or concrete industry), but also the newest software houses and consulting services industry. Nevertheless, B2B marketing also concerns companies whose clients are both small businesses and single individuals. It is called Business-to-Business to underline the contrast with Business-to-Consumer (B2C) where the target of the business is, indeed, an individual consumer rather than a company. Contrary to some common assumptions, these two businesses, even if they have a common ground, actually diverge deeply. It is well said that in nowadays world, where customers, both companies and individuals, are used to receive high quality services and targeted advertising, it is not possible anymore to consider B2C and B2B as the same type of problem.

2. MAIN CHARACTERISTICS OF B2B and HOW IT DIFFERS FROM B2C

In this first section I will provide the main characteristics of B2B, how it diverges from B2C and why it is so important to have specific technologies to help companies managing this type of business, which, by the way, is evolving with such an incredible speed. I will show the importance of understanding the different mentalities that sit behind a purchase made by a person for himself and one made by a company, in order to build a service that the customer can easily use and personalize even without any type of informatics knowledge. After all, understanding the intricacies of a marketing is essential for any business looking to increase sales and profitability.

First of all, exactly like B2C marketing, also B2B marketing includes all the strategies that allow a company to promote its products or services to potential buyers, which in the case of B2B are companies. Unlike many people believe, marketing a target audience of business versus marketing an audience of individuals may not be a one-size-fits-all problem. Individuals buying something for themselves will have a greatly different emotional experience than someone buying for their company. The main goal of B2B is to convert prospects into customer, the process is much longer and much more involved than a B2C, which mostly works with short campaigns. A B2C tool will have the ultimate goal of hooking customers, which might never visit the website or purchase something from it ever again. A B2B relationship, on the other hand, is based on long "courtships" and long-term relationships with the business they are trying to attract. Therefore, a B2B tool will be characterize by ways of making your customer feel special to you, rather that flat give-away campaigns. Indeed B2B clients tend to make way less impulsive purchases and rather have specific needs and make purchases of many units of product, sometimes even for third parties.

Some of the main differences between B2C and B2B are the following:

Limited number of Customers

Indeed the potential number of possible buyers for each product or service in B2B is usually much lower than it is in B2C. The advertising, therefore, must be extremely targeted and suited on the single client. An example of how this is put into practice by B2B commerce software tools is to give the possibility to assign buyers (client accounts) to different Buyer Groups and assign to each group different Price books, Discounts and Promotions. (This will be well discussed later). It is important to underline that in this business even one single customer not happy, might mean loosing a discrete amount of revenues.

Larger Orders

On the other hand, B2B sales are usually characterized by high number of units purchased at once. Orders are, indeed, made in bulk. As a consequence, this will bring to higher revenues on fewer sales in respect to B2C commerce. This also explains why every single customer satisfaction is important.

Long-Term commercial relationships

When an individual decides to buy something from a website, it usually represents the last stage of a marketing and sales funnel. On the other hand, when the buyer is a company, it usually is the first stage of a long lasting relation. It is, therefore, fundamental to make sure that the client will be able to access the finest post-sales and maintenance services. Moreover, B2B marketing continues even after the sale, for example using newsletters to update the client on new products or invite them to webinar and other similar ways to maintain a strong relationship with him. Sometimes, companies have specific people, Account Managers, who take care of the relationships with the customers in order to make them feel unique. In these cases, becomes fundamental to have a software tool that allows to easily manage client accounts, this is another piece at the basis of any well-built B2B commerce tool.

B2B Buyers are usually more rational.

B2B buyers are usually experts of the sector, therefore they look for a specific product or service and tend to avoid impulsive purchases. Moreover, they are spending company money, so they also are accountable to the stakeholders for the purchases that they make. B2B buyers are not hooked to buy just for a personal whim, they might want to make many questions before buying something. This means give the clients an easy way to ask questions and receive well-formed and exhaustive answers in a fast, easy way in order to generate an adequate user experience, but mostly an efficient system.

Having said so, it is important to remember that not all B2B are the same. Different businesses might differ from many point of views, which are an important part to take into account when trying to build your commerce experience. (what-is-b2b-marketing, s.d.)

Firstly small/medium businesses obviously can rely on fewer resources than larger ones, therefore there will be a few “key accounts” which will bring the largest portion of profits over the other ones with lower purchasing power. This is much different from B2C Commerce where the target audience has more or less the same economical status.

Secondly, because of the higher costs and dimension of the orders, the decision-making process of B2B takes longer time and usually involves sector specialists to contribute in the purchase and stakeholders who will need to approve the transaction. Therefore, it will past more time from the initial contact with the potential client and the receiving of the payment and each customer will negotiate individual discounts and promotions as incentives.

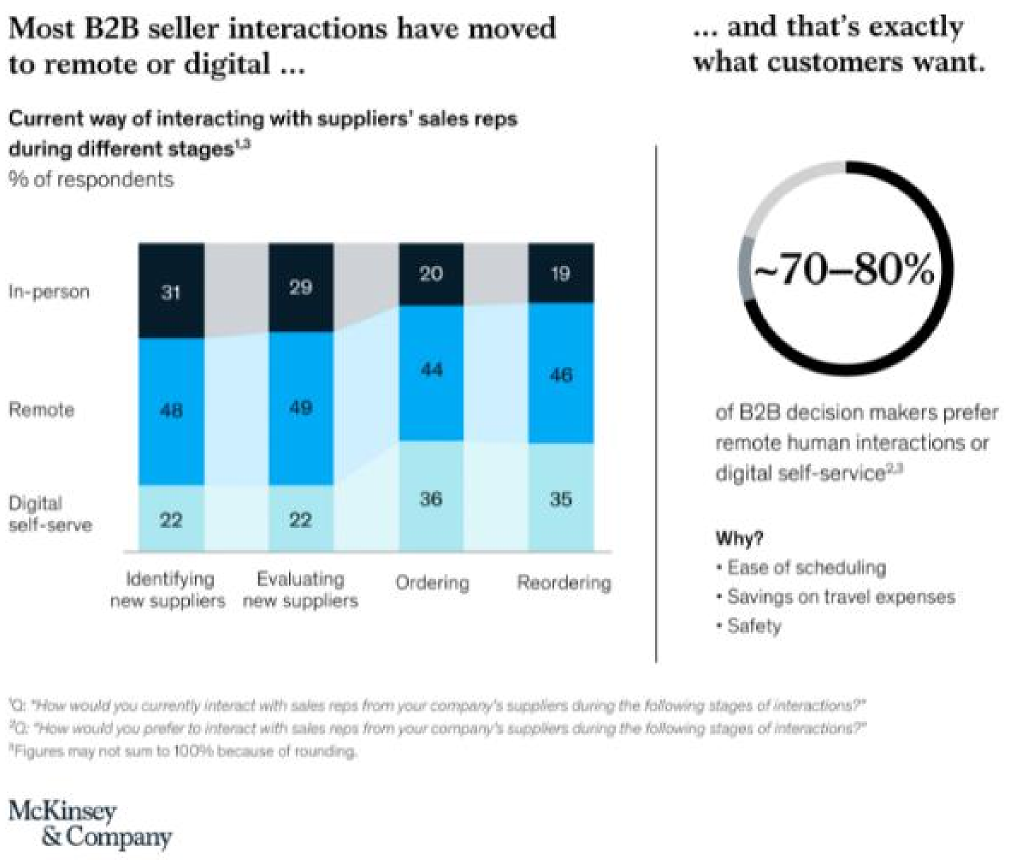
3. B2B MARKET AND TRENDS

Having understood what B2B means and which are its key points, it is now time to understand how big is the B2B market and how much is forecast to grow.

We are now seeing an exponential growth in the market of B2B e-commerce systems. This is directly connected to the need of many companies to be able to reach their clients throughout the world with the simplicity and practicality of an automatic system. Until now the methods available for B2B companies were quite unsophisticated, with superficial personalization and contents to attract a rather broad audience (much more similar to the one of B2C), since companies weren’t able to characterize well enough their potential customers. But with the evolving of technologies we can say that, nowadays, there are advances that allow a much greater degree of tailoring and personalization, including the use of AI algorithms on data to improve the customer experience and relationship management, in order to build a so called customer-centric approach. Another functionality that has been recently included in modern B2B software tools is the Mobile-First approach. As it is well known, buyers are moving a lot from PC purchases to buying directly on their cellphone. Even if this is mostly true for B2C, also a great deal of B2B buyers (usually small businesses) tent to use a lot tablets and any small screen device, making fundamental to being able to optimize the website for a mobile use. The idea is to make the user experience on mobile devices as pleasant as on the laptop. Indeed B2B buyers tent to use the mobile device mostly to have an easy access to first-hand information rather that to make the actual order, but also the purchase will need to be taken into considerations anyways as a possible scenario. Nowadays B2B companies can count on sophisticated e-commerce platforms to allow their customer to conduct the transactions online, avoiding the burden of having to make purchase through a phone call. Let's see some interesting statistics to better understand why it is important to have a solid basis technology to conduct a successful B2B (Arnau, Liz, Dennis, & Jennifer, 2020).

Base on a B2BecNews article, B2B e-commerce had a $900 billion market in 2017 in the USA alone and according to Forbes the marked will reach $1.8 trillion in 2023 (John, Susan, Charlie, & Rachel, 2019). In general, B2B companies are increasingly looking for digital tools to boost their sales and mostly to adapt to changing buyers. Indeed, in 2015 a Google search found that 50 % of the B2B Buyers where millennials, while in 2012 it was around 25%, nowadays the percentage has grown to 73%. The buyers change means the business' sales process should change too (Kelsey & Pashmeena, 2015). Millennials prefer personalized experience and digital channels rather than a sales representative. According to a McKinsey survey in October 2020 more than 75 % of buyers and sellers prefer digital self-serve and remote human interactions over face-to-face engagements. This sentiment has much grown during the lockdowns and keeps intensifying even after it. According to Google only 20 % of the B2B purchases are based on the actual price and offering, while the remaining 80% are directly or indirectly linked to the user experience.

The following graph from McKinsey shows how the B2B interactions have moved to remote and customer are pleased by it.



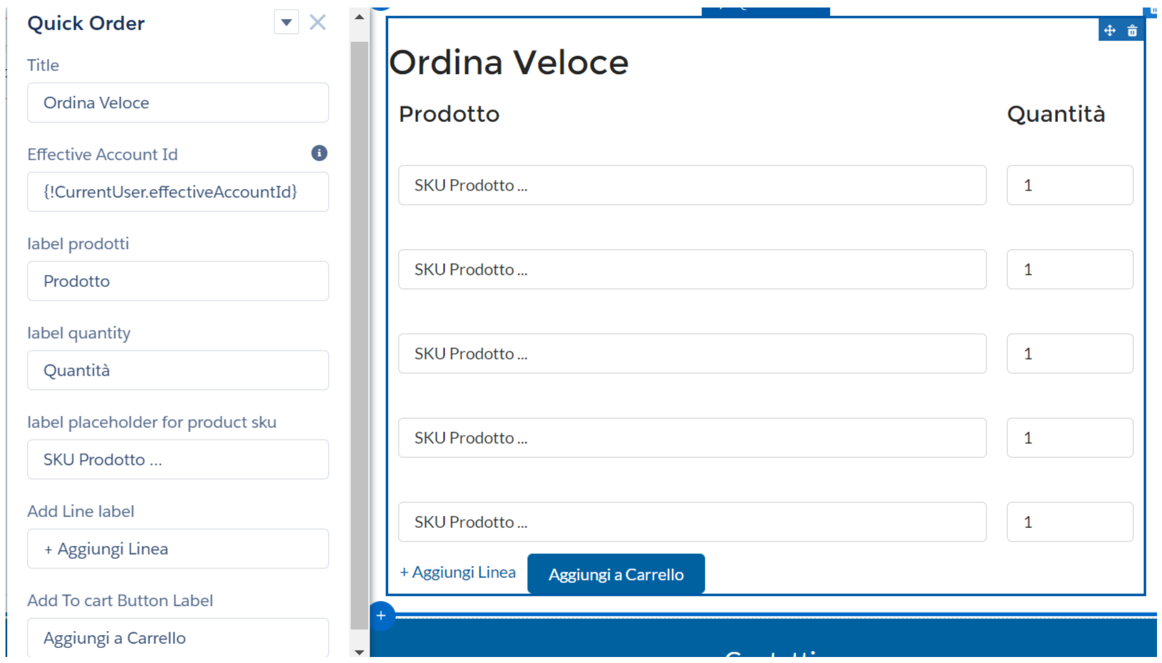
Even more interesting of the McKinsey survey is the fact that also the amount of money B2B buyers are willing to spend on online e-commerce purchases has increased exponentially. 70% of B2B buyers states that they would spend up to $ 50,000 in an online purchase and 27 % would go over $500,000.

This has brought the majority of B2B companies to move from a traditional go-to-market to a digital one, with heavily reliance on video and online chats. And this incredible change of marketing that was boosted by the Pandemic, always according to McKinsey surveys, will most likely become permanent after the COVID era. Indeed almost 90% of decision makers say that new e-commerce sales practices will stick even beyond this year.

We can, therefore, say that B2B technologies' impact on today's way of making B2B transactions is and will definitely increase in the next years.

4. SALESFORCE B2B COMMERCE

It is now time to discuss one specific software technology that is now striking through the market of Business to Business: Salesforce B2B Commerce. In particular, I will refer to the newest come out platform called Lightning Salesforce B2B Commerce, that improved the previous Classic one. We can say that the two solutions are two sides of the same coin, but there are some differences that make the newest solution more flexible and easier to be used, even by a not programming expert. One typical example is the fact that Classic Salesforce is built natively on Visualforce, which is a powerful framework that can be used to create front-end interfaces and works well with Apex language on the back-end. So what is the problem about it? Well, the main issue is that you do not give the possibility to the B2B company to autonomously create their own front-end pages unless they have programming language knowledge. The incredible breakthrough of the newest Lighting Experience is that it uses Aura Components or even the latest LWC (Lighting Web Components), which can be configured simply using some clicks. Custom components will obviously need to be programmed, but you can create these components so that then the B2B company will be able to change part of them from the "Experience Builder", through a nice, user friendly interface, that does not involve any type of programming knowledge. The following picture, shows how a single LWC component might look like on the Experience Builder. In particular, it is a custom LWC programmed by me, that can be placed on the homepage of a website and allows the final user (if logged in) to create quick orders just by searching the products through their SKU. As you will see the B2B Commerce User will easily be able to add customizations to the component without having to write a single line of code. On the left, there is the customization wizard, while on the right there is how the component will show.


Quick Order

Title: Ordina Veloce

Effective Account Id: {!CurrentUser.effectiveAccountId}

label prodotti: Prodotto

label quantity: Quantità

label placeholder for product sku: SKU Prodotto ...

Add Line label: + Aggiungi Linea

Add To cart Button Label: Aggiungi a Carrello

Ordina Veloce

Prodotto	Quantità
SKU Prodotto ...	1
SKU Prodotto ...	1
SKU Prodotto ...	1
SKU Prodotto ...	1
SKU Prodotto ...	1

+ Aggiungi Linea | Aggiungi a Carrello

This is just an example to show how powerful could be the Lighting Experience.

Another important difference between Classic and Lightning Experience is that Classic is built as a managed package on the core Salesforce platform which limits its ability to be a completely integrated OOTB cross-cloud solution. This means, basically, that it has a set of components and applications that must be installed in the Salesforce Organization before being used. On the other hand, the B2B Lightning commerce is designed on the core Salesforce platform, which allows also on the data model point of view a complete control over the customer journey. For example, the Product object used in B2B Lightning is the same of the one used in CPQ, Sales Cloud and Service Cloud. This allows to build a connected OOTB solution for the entire customer journey and helps learn about which products the user is interested in, which ones he actually chose to buy (added to cart), the products the user bought in the past (past Orders) and which ones he has some concerns about (products and orders ready but not concluded). To be precise also Classic Salesforce provides an around the circle customer journey data model, but it is not seamlessly connected with other clouds and is custom and complex in nature.

4.1 SALESFORCE B2B LIGHTNING

Salesforce B2B commerce focuses on enabling companies to create e-commerce storefronts designed specifically for making large volume purchases from other businesses online. This particular technology includes all the basis elements to operate in a B2B transactions, such as authenticated websites to make users create their account and being uniquely threated. This means having the possibility to create custom storefronts for a unique look for each account group and being able to offer different product catalogs by account selecting only a subsets of the products for each one. Shopping carts must be able to accommodate hundreds of items per order and the users must be able to reorder in a easy and quick way, with just few clicks. More than that it allows to create products and variant products and negotiate specific contracts and prices by single account. Finally it offers solutions to integrate complex shipping functionalities, many different payment methods, such as credit card and purchase orders, and many different storefront and order templates. Since B2B Commerce is built natively on the Salesforce Lighting Platform, it integrates directly with Salesforce CRM data from Service Cloud, Sales Cloud and Experience Cloud. This means that important data will be available and accessible easily by both buyers and sellers. In this type of services it is always important to remember that there are two sides of the system. There is the sales organization (the B2B company that sets up the store in order to sell its products/services) and the buyer organization (the company that uses the platform, the actual clients). The first one needs to access all the information about the accounts of the customers in order to offer them the best user experience as possible and also needs to access the purchases information in order to carry on the purchase orders through an integrated Order Management System (OMS). On the other hand the second ones need to be able to access their CRM data directly on the website, to see the special promotions and prices that were reserved to them but also to easily create orders, access their order history and account information. What is really impressive about Commerce Lightning Experience is that provides a quite high number of personalization without any kind of need of programming. In particular, it offers administrators a tool to easily build their storefront called Lightning Experience Builder, which allows to simply drag and drop components on the page and create the webpages. It also provides a well structured BackOffice and class diagram implementation, ready to use, in order to store all user information and that can be easily integrated with custom implementations. So, even if the software offers a quite high number of out of the box features it also allows to create custom objects, storefront pages, profiles, permission sets and basically any part of the platform can be customized accordingly to the user need. This is what makes this platform extremely flexible; the ability to both have a ready to use website that can be very much easily configured, and, at the same time, it gives the user the possibility to implement almost any type of personalization. (Salesforce Official Website, s.d.)

4.2 SALESFORCE B2B COMMERCE – Technical Characteristics

The platform can be divided into two main parts: the storefront and the back-office.

The Storefront represents the front-end site of the e-commerce; basically what the clients of the B2B company will see and access while visiting the website.

The back-office is the part of the platform that will be accessed by the company itself in order to make the configurations, create the store and the objects, access all the accounts information, eventually create promotions and access the order management system. As said before B2B commerce can easily integrate with other systems to allow the import and export of data from any CRM system to the e-commerce database. It can for example be integrated with SAP systems trough a middleware (for example Boomi) in order to export the e-commerce orders to a SAP management system that will manage the OMS. A Salesforce Database is integrated by default, it already presents a standard structure, that is applicable to B2B systems. Anyways the Database can be customized with other classes as well as the objects can be customized with more fields, other than the standard ones.

When an organization is created, it can assume different structures. It can be a single site with one storefront, a single site with multiple storefronts or even multiple sites with multiple storefronts. The idea is quite comparable to a traditional brand with many physical stores situation; a company can sell its products in a single city (single site) or in many different cities (multiple sites). At the same time, the stores in one city can have the same display window or different ones. These would be your storefronts. The term “storefront” is usually used by B2C Commerce websites but it also describes any online ecommerce experience, it represents any place a shopper goes to buy something. When you build your website you need to think about how many actual websites you want and for each how many storefronts you want to offer to the users. So, each site will have its own domain URL to be accessed and then each storefront will share the domain but will be associated with different store names.

Once the store is set up, it is time to populate it with your products. Notice that I will use products to refer to actual physical objects but also software, services or anything that a B2B commerce could sell. To associate products, firstly, you need to create a Storefront Catalog. As the name suggests each storefront can only have one single catalog (that will be the storefront catalog), so if the organization has more catalogs you will have to create multiple storefronts. The storefront catalog needs to be structured using products categories. These categories will create the storefront default navigation. (Even if we will see that this navigation is actually quite customizable too). The categories are important because they group similar products together, and separate different ones. They are the spine of the website, so it is important to think them through before writing any type of custom code. Categories can be structured in top-level and child categories. All categories are children of the root category that does not have a name, while all other categories must have one. The category tree can be as deep as you want it to be and you can choose not to show a specific category or an entire branch of categories on the default navigation menu, without any need of customization on the default navigation menu. Once the categories are set up, it is time to add the products.

Products are characterized by their attributes (as per any other object), but in the case of products the attributes are more important since they will contain the information that the shopper will see on the storefront and use to decide whether to buy or not something. For example, for a company selling glasses the attributes could include size, color, material, lens type and brand. When you define the attributes, these will be applied to all your store products. So each product is owned by a catalog, but you can include a product in as many catalogs and categories of your store as you wish. To be displayed on the storefront a product must at least be assigned to a storefront category, searchable, must be assigned to an entitlement policy and a price book. You can then decide which attributes of your product can be used by the user to search it on the storefront.

Sometimes you need to have the possibility to create variations of the same products. For example, if you sell sunglasses you might want to sell products with the same exact characteristics but with different sizes and colors. To do so, Salesforce has created variations, variation groups and master products. The idea is that you can create a Master Product that is basically the sunglass that comes in different colors and sizes. Then you can create a variation group, that groups together all variations of the master that share one variation parameter, such as all sunglasses that comes in blue. Finally, you specify the single variations, that means you create the sunglass with size small and color blue, another one with color yellow and size medium and so on. Normally variation attributes are either picklists or checkboxes. It is important to underline a limit that Variations have, that is the fact that product images displayed can only be associated to the parent object, this means that you cannot add directly an image for each variation of the master product. This means that is some cases you will need to use a naming convention to get the specific variation image on a custom component placed on the storefront.

It can also happen that some products can be sold as a set of products, for example a makeup kit that comes with a mirror, some make up and a hairbrush; in this case you can exploit the Set object that will display all these elements together on the storefront. Notice that with Set objects, the products can also be sold and displayed separately.

On the other hand, if you need to bound some products together, so that they can only be displayed and sold together you can exploit Bundles, which will group the products and sell them only as a group.

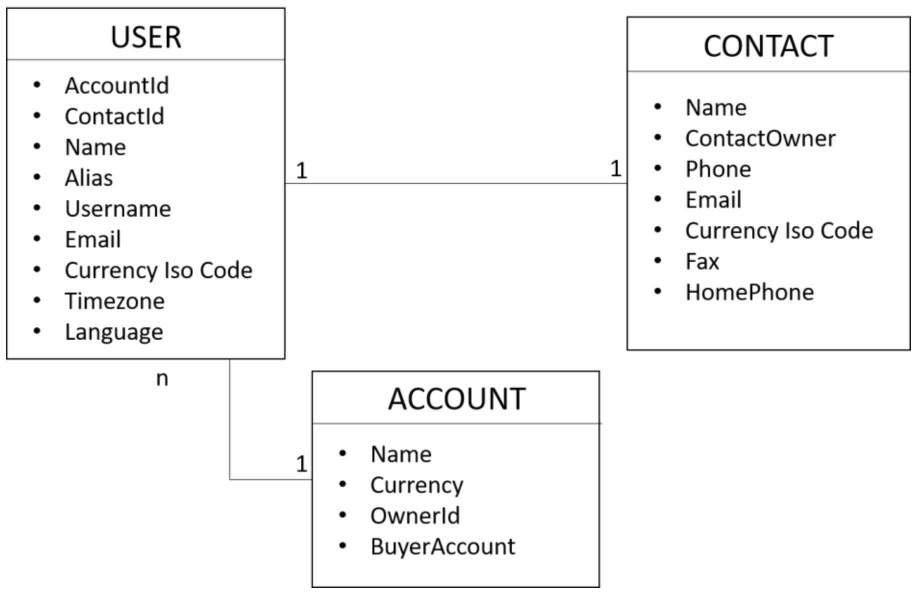
Finally, there are Options, which are accessories of a product, that cannot be sold by themselves, do not come with an image and are strictly related to a product. A typical example are warranties for different time periods. These types of products are nor searchable nor orderable separately from the product they refer to.

In general, products can and should be associated with images. These can be links or can be images directly loaded on the Commerce back-office, and managed by the Content Management System (CMS). Each product will have a primary image, shown on default category details page, and other images that will be shown on the default product detail page in a carousel. Note that I always add the “default” term, because as usual all these pages are customizable and therefore you can show whatever image of your products you want if you write the code and the storefront component to do so.

What is also quite important in an e-commerce is the Inventory. Indeed the user must be informed about when a product will be delivered, if the product is in stock or not and eventually when the product will be available. This comes in a built-in solution provided by B2B Commerce. In particular, you can use a list of product IDs to map to inventory details, such as amounts, allocations, preorders and in-stock dates; this list represents the online inventory. You can assign an inventory list to a specific site, but also to multiple sites to share the product availability across your organization. At the same time it is important to underline that inventory can also be managed using integrations with other systems. In this case, all the inventory checks will have to be done by custom code and HTTPS calls to external services. But, other than that, the process is pretty straight forward. We will see an example of how to implement this in the practical section.

As I mentioned before products to be shown on the storefront must be associated with a Price Book. These objects are used to define your products prices and also contain information about the currency used. Consider that an organization might have multiple currencies, so it will have different price books for each currency. More than that price books can also be assigned to buyer groups, that means basically that you can assign different prices to the same products depending on the user who is navigating the website. So you can have more active price books at the same time on the same storefront. This is a powerful tool, because it allows the Commerce Organization to establish specific negotiated prices to specific user. This is really important in the B2B Marketing to make sure that the customer receives a treatment that makes him feel special and unique. As we said at the beginning, one of the key concepts of B2B Commerce Marketing is to create special offers targeted on the single (or on few) account, in order to create a long term relation with each client. You can associate a price book to one or more sites of your organization, but it must be associated with at least one to be seen on the storefront. Moreover you can also define different prices for varying quantities of a specific product using price tables. So for example you can decide to create a price table that sells one sunglass for 10$ and two to three sunglasses for 8 $ each. This is helpful to manage what so called Volume discount, that is one of the most used discounts in B2B Commerce together with Coupons. Volume discounts are nothing more than discounts based on quantity. In case of B2B Commerce it is not unusual that a user buys a large volume of products all at once, so here comes the Volume discount.

Another important subset of objects are the ones used to characterized the users of the Commerce, precisely I'm referring to Contact, Account and User. As we already said we always have to consider the two sides of the Commerce problem: the B2B Company that needs to know who their customers are and the clients side, who wants to access their information and eventually update them. In order to do so B2B salesforce Commerce offers some standard Objects to track this information and make it available as needed. This object represent the Client Company, so for each customer that enroll in the platform will be created an Account, basically every client will be described by an Account. However, sometimes, it happens that a customer needs to have different "profiles" for its employees, so that the Account is shared among them, but each company user can access the platform separately and make its orders. To do so an Account can be associated with many Contacts and each Contact refers to a User. The following image better describes the relationship between these objects.



This schema does not contain all the default attributes of these objects, but I selected a representative subset of them.

The User keeps the information about the physical person who is logging in the portal. So, it has a username and a secret password, other than the information about the default language that the user wants to use, the currency type and his time zone.

The Contact specifies more details of the User, in particular the ones that concern how to contact him, such as: Phone, Fax, Email, etc…

Contact and User have a one to one relationship.

Finally, the Account represent the company. It might be associated with many users and indirectly with many contacts. It also stores the information about Buyer Account. So, what is a Buyer Account?

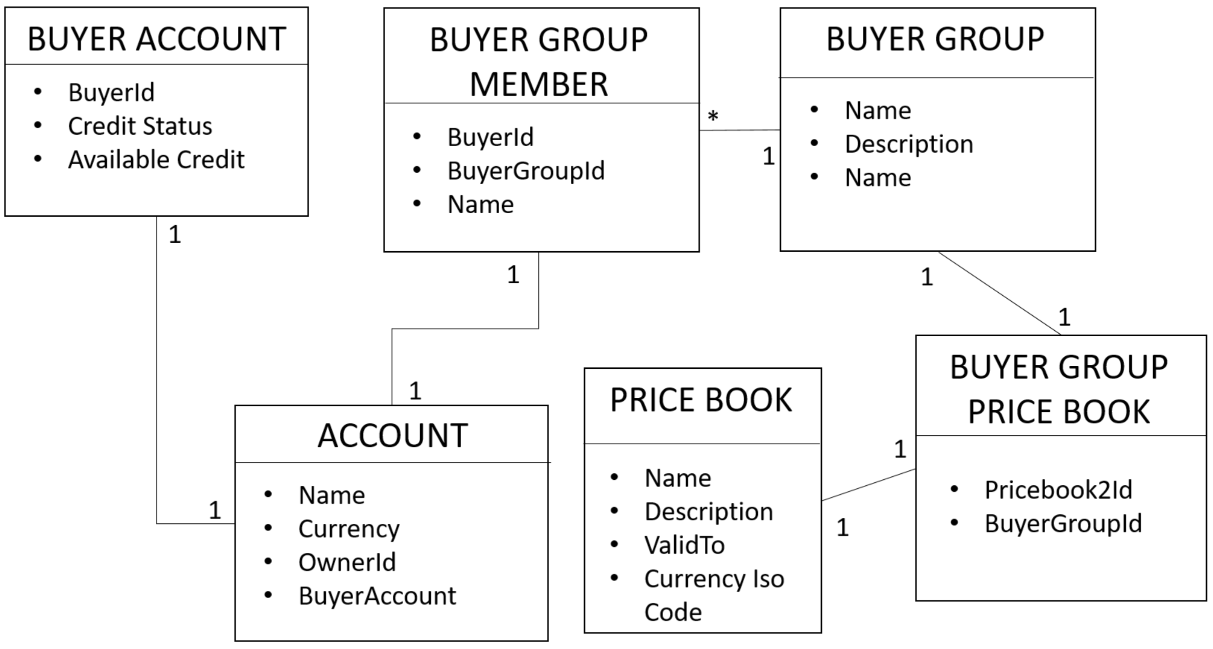
Let’s remember that also the employees of the company owning the B2B Commerce might want to access the platform. They for sure will not buy anything from the e-commerce, but they might for example need to access the back-office in order to make changes or gather clients’ information. Or even, sometimes, companies do not build their own e-commerce platform, but they have their e-commerce programmed by somebody else. Also in this case there will be some Users (for example the developers) who do not need to access the web store, but they need to access the back office functionalities and code in order to do their job. So the question now would be how salesforce distinguishes accounts and users who can access the storefront and the ones who can access the back-office? The answer is Buyer Account. When an account is enabled as a buyer, than the contact can be enabled as a Buyer as well and so the user associated with that contact will be able to access the e-commerce. This double enabling, both for the Account and the Contact, might seem useless, but is, actually, not. Image the scenario in which an Account is linked to many Contacts, but not all of them should access the portal, then the Account will be enables as a buyer, and only the Contacts that need to be make purchase will be enabled too.

Now it is time to deal with the problem of how Accounts are linked to Price Books and Products. As we already mentioned many times, we want to be able to control the prices to which each Account is associated, and eventually offer special deals for some of them.

First things first, we want to be able to decide which product a user can see. Maybe we have special products, like limited products that we want to show only to a subset of our clients. To do so there is the Entitlement Policy object, that associates products with Buyers. So each Product to be seen but at least somebody, must be associated with an entitlement policy, and on the other hand also a Buyer to see some products on the store must be associated with at least one entitlement policy. One possibility to manage this, would be to associate each buyer to an entitlement policy that groups all the products we want to show him. Obviously, this solution is not quite scalable.

Secondly, there is the problem of price-books. As we said earlier, each product might be associated with different price book entries coming from different price books. So each Buyer should be associated with a specific price book as well. Again this is not very handy.

So what is actually used are Buyer Groups. A Buyer Group is used to organize similar Buyer Account together. This means grouping all Buyers who share store, products, prices, and entitlement policies. So each time you have a new user that falls in these characteristics you can just assign him to this buyer group and he is ready to go. The following class diagram gives a visual representation of the relationships between these entities.



One last important thing to mention with Accounts is Contact Point Address. This class stores the information related to the addresses of an Account, both for Billing and for Shipping. In the standard solution each Account can have many Shipping Addresses and Billing ones, so on the checkout phase the user will be asked to choose one address for each type in order to proceed with the order. We will see in the last section a practical example that has many more constraints on how many addresses a User can have and also on the possible countries a user can choose. As usual any customization is possible if you are willing to write some code. So for the standard solution the relationship between accounts and Contact Point Addresses will be as follows. To be fully complete, I have also included the Contact Point Phone object, which is quite uncommon, since the phone number is already a field of the contact itself, but it still exists and in some cases it might come in handy to have a more complex structure to store more information related to the phone number (other than the number itself).

Another fundamental part of the platform deals with managing the object access. In order to create a secure website it is important to apply the principles of least privilege and need to know, which mean a user should be able to perform only the necessary actions for him to use the platform (least privilege) and should be able to see only the data that concern him (need to know). So, in this mind set, Salesforce provides two main objects that can be used to manage the accesses: Profile and PermissionSet.

Profiles grant the minimum permissions on objects and actions that all users of a particular type need. Each User must be associated with one and only one Profile, which gives him the possibility to access objects or perform certain actions on them, like create, delete, edit, clone, etc… On the other hand, permission sets grant permissions and access settings to a specific user.

So you can use profiles to grant the minimum permissions and settings that all users of a particular type need, then you can use permission sets to grant more permissions as needed. The combination of profiles and permission sets gives you a great deal of flexibility in specifying object-level access.

In particular, a profile is a collection of permissions and object settings. Object settings determine which objects the user can access and also which attributes of each object is accessible, while permissions determine what kind of actions a user can do with those objects (create, delete, etc…). Each profile can be associated with many Users, but each User can have only one profile. There are some standard Profiles, such as: Standard User, Marketing User, Contract Manager, System Administrator and Minimum Access – Salesforce. Each one includes some specific permissions, for example a standard user can create and edit objects, while Minimum Access-Salesforce Users can only view records. The system administration profile has the widest permissions and access to data, so it can configure a great deal of the

Salesforce Organization, therefore, it is important to assign this profile only to specific users. On standard profiles it is not possible to edit permissions on Standard Object, while you can set the permissions on the custom objects that you create on the platform.

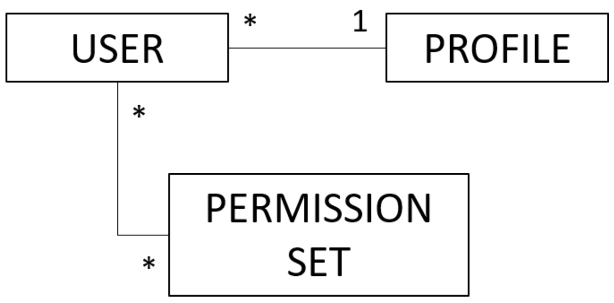
Sometimes you need to create profiles outside the standard ones, to do so, the easiest way is to clone an existing profile and modify the permissions and access to objects. Another way would be to create a new profile from scratch, but that is quite complex and mostly it is easy to make mistakes or forget something.

One particular profile is the guest one. With the term guest I refer to any user, who is not logged in the platform, but still should be able to navigate the store. This profile must be carefully configured, since it will be assigned to anyone who visits your website. Obviously not all problems are the same, so there is no standard configuration for it, although there is a default one, which can be modified as needed. Usually a guest user is set to be able to access products and categories of the Store, but not prices and cannot add products to cart or proceed to checkout.

A permission set is a collection of settings and permissions that give users access to various tools and functions. The settings and permissions in permission sets are also found in profiles, but permission sets extend users' functional access without changing their profiles.

Users can only have one profile, but they can have multiple permission sets.

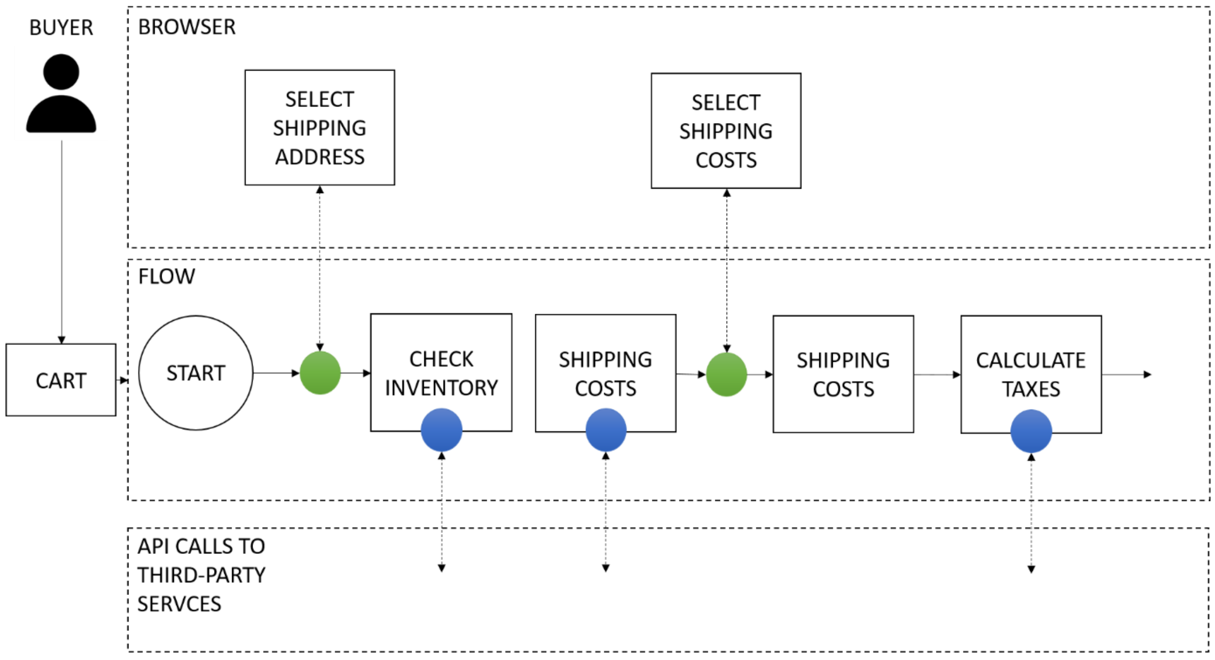
Permission sets are usually used for two main general purposes: either grant additional access to custom apps and objects or to grand additional access to custom fields.

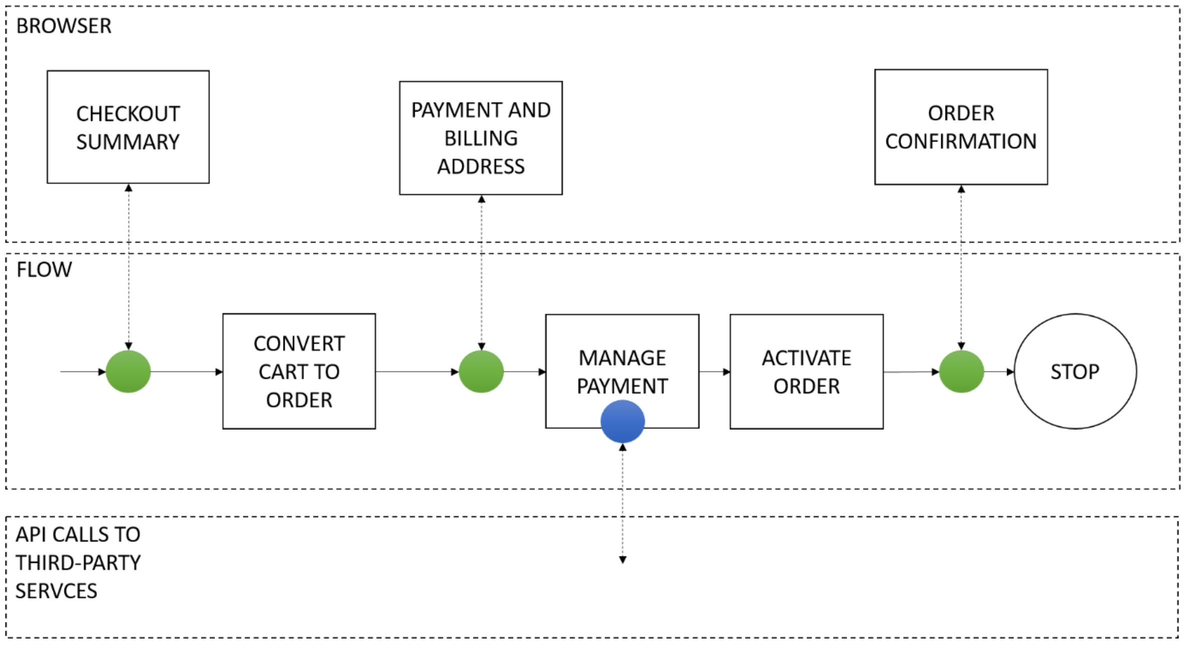


It is now time to discuss the data model of the most important objects in an e-commerce: the Cart, the Order and the Order Summary. Obviously, the core aspect of an e-commerce website is that it offers the users the possibility to order their favorite products just in few clicks. Therefore, the Checkout process and all the related objects are a fundamental part of the platform.

The checkout flow has a default design offered by the platform, but it is completely customizable as needed. Here we will discuss the standard design, which is the basis of any checkout flow, while in the last section we will deeply discuss an example of a real implementation of it.

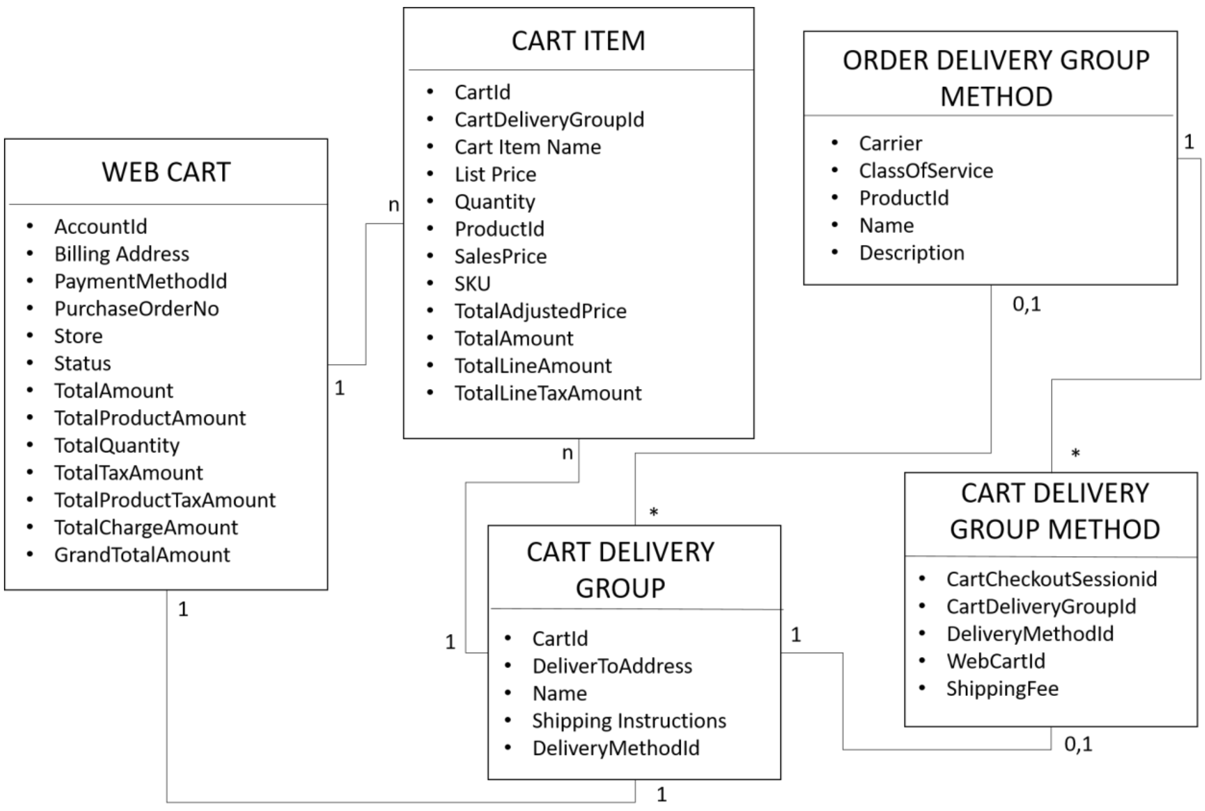
The Checkout flow design takes into considerations three main views: the browser side, the flow side and the third-party services side, which we will map as the pools of our process model. The flow starts when the buyer creates a cart and clicks on the checkout button. Immediately after the user is asked to insert the desired delivery date, to do so the flows stops and uses a Screen Flow object to pass information from backend to the user interface. This integration is basically managed by the Salesforce platform, but the developer can decide to write its own screen flow or use a standard one. A screen flow can be any Aura Component that implements the Screen Flow Interface. The difference between how a screen flow page and a web page are visualized on the user interface is only the fact that web pages are characterized by a unique url, while all screen flow pages of the checkout share a common url. After the user has selected the desired shipping address, the flow will manage the check inventory and the shipping costs calculations. Has you can see from the diagram, both these actions might require the connection to an external system service. A typical example is the check inventory, which might need to connect to a SAP system to gather the information about the available quantity for each product of the cart. The connection to external systems is manage through REST API calls, fully integrated in Salesforce. Shipping costs are instead a little different, they might need to connect to an external system to me calculated, but then they pass through another screen flow on the user interface so that the user can pick which type of delivery he wants (with this I mean like: fast delivery, standard delivery, etc…). Once the flow receives the data from the user interface it proceeds updating the shipping services costs on the cart and, then, calculates the taxes. Also this action might require an external service connection. Once all the costs information has been collected and saved the user will see on the screen a summary page of its cart, with the total amount he will need to pay in the next step. Before the payment, though, there is a flow action, that transform the cart to and order. At this point, the cart object is abandoned and instead it is created a new Order object with all the related information from the cart. After this, the user will choose a payment method and again the flow will need to connect to an external system that manages the payment (for example Stripe for a card payment). Then the flow will change the Order status to Active. This action will automatically create a new object: Order Summary, which will keep all the final information about the user order. Once the Order Summary is created, the user will be redirected to an actual new page (so the URL will change) that shows the Order Summary details. Order Summary objects can be exported to external systems if needed. The following process diagram gives you a visual representation of what I just explained. Consider the green dot as an intermediate action that shows the User Interface, while the blue one represents a connection to an external system.





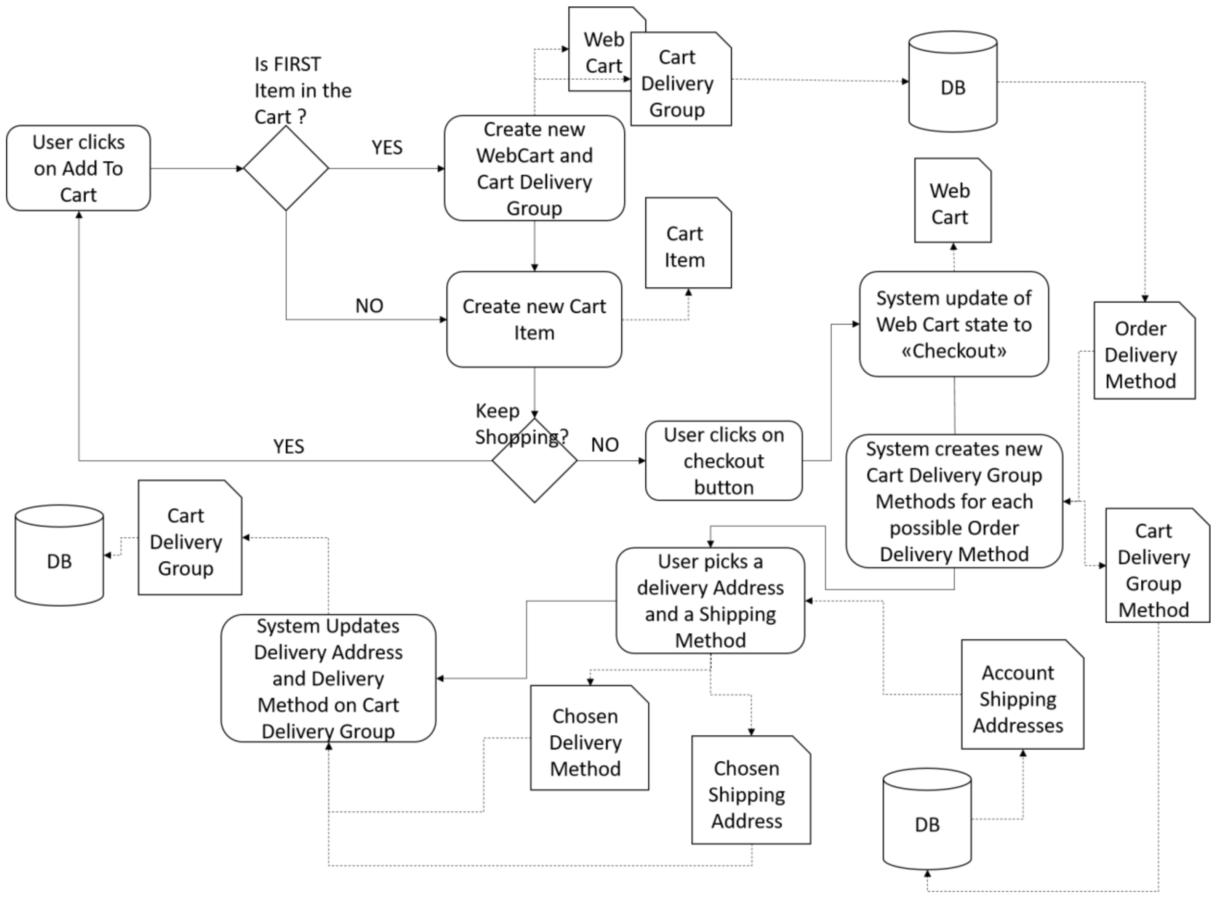
This previous diagram can quite well explain how the workflow of the process, but what about the data? As we said before there are many objects involved in the process so I will try to dive deeper into this aspect.

Firstly, it might be interesting to understand the relationships between these entities. Let's start from the ones concerning the Cart (called Web Cart in the Salesforce DB). As I did for the previous class diagrams, I will not include all attributes of the objects but just the ones that I consider more interesting and useful to understand the meaning and the relationship of them. Moreover I want to remind that this entities, even if they are standard ones, they can be customized with other attributes as needed, so this will remain a high level model to understand the structure and architecture.



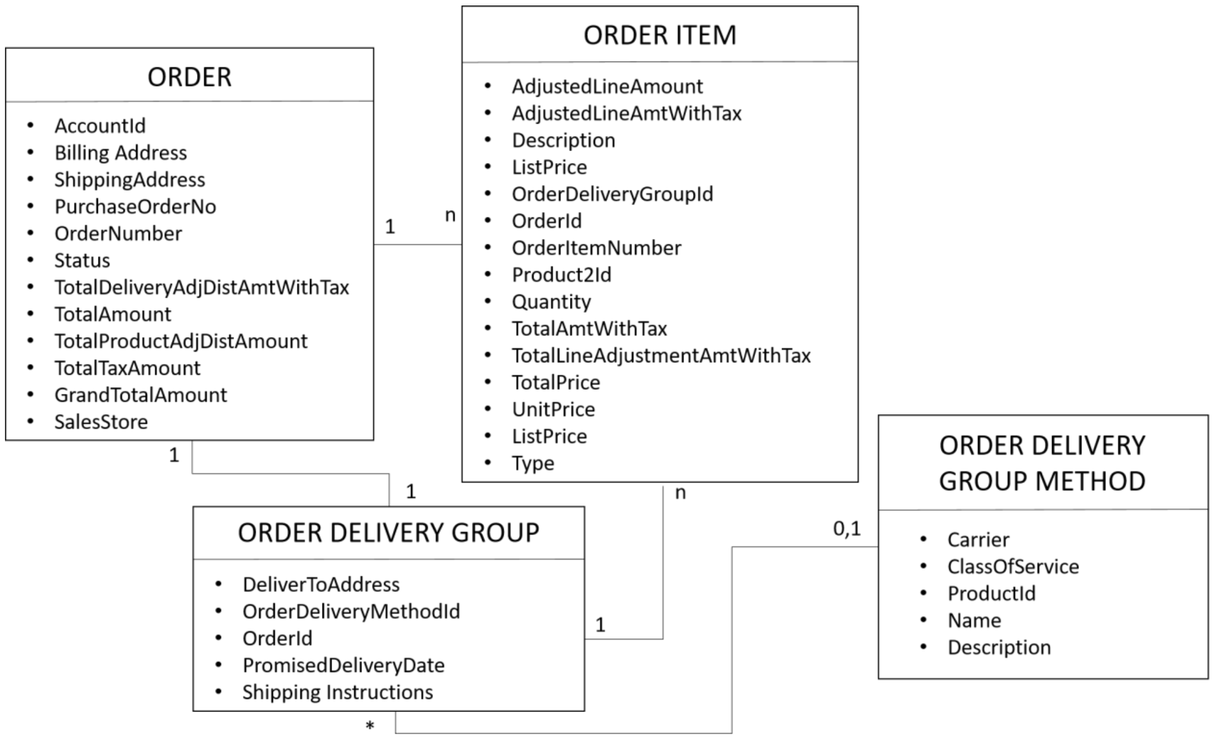
The Web Cart stores all the information related to the User's purchase cart, that includes the account related to the purchase, its billing address and payment method and all the prices data. The prices are actually divided in the price of all the products with and without taxes, the total tax amount and the total discount amount (if any), and obviously the total amount considered all products and charges and also the shipping costs. A Web Cart is created as soon as the user click on a product add to cart button. For this reason the relationship between a Web Cart and a Cart Item is 1 to n. The Web Cart will have at least one product when it is created an potentially more than one, while the Cart Item will be immediately associated with a Web Cart when it is created. A Cart Item represents the single product added to the Web Cart, so it keeps a link to the Product, but also the quantity of that purchased product, its SKU (that is a product identifier commonly used in B2B transactions) and all the prices of the single element. These include the unit cost, the cost with taxes and discounts and also the total amount of the line, quantity considered. Moreover as soon as a Web Cart is created also a Cart Delivery Group is inserted in the database. This object keeps all the information about the delivery, that means the shipping address and shipping information. It as a 1 to 1 relationship with the web cart, since they are created together, but it also has a one to many relationship with each cart item. Finally ,the cart delivery group stores the information about the chosen delivery method, this link is created after the user actually picks a delivery method during the checkout phase, but the same order delivery method can be linked to many cart delivery group, therefore the relationship is many to zero or one.

The order delivery group is the object used to store the possible delivery method offered by the B2B company, therefore, it usually happens that the instances of this object are created a priori during the data import. Each Order Delivery Group is related to a special Product that represent the specific delivery method, never the less the actual price of that delivery method might change based on, for example, the shipping address chosen by the user during the checkout phase. To manage this variability of the price Salesforce offers another object, that is the Cart delivery Group Method. When all the information needed to calculate the total delivery method cost is gathered, then a new Cart Delivery Group Method instance is inserted in the database. This instance will be linked to the related Order Delivery Method with a many to one relationship (so an Order Delivery Method can be associated with many Cart Delivery Group Method, but this last one refers to only one possible Order Delivery Group), but it will also be linked to the Cart Delivery Group instance. Notice that when the Cart Delivery Group instance is created it has no related Cart Delivery Group Method, therefore the relationship is zero or 1, while for the vice versa the relationship is 1 because the Cart Delivery Group Method will immediately be associated with the related Cart Delivery Group. In the following picture you can see the Data Flow related to the previously considered objects. This flow does not consider all the actual operations the system and the user perform during the checkout, but it stores the main actions that concern the instances in which we are interested right now. So you can better understand when and how these object instances are created and updated during the first part of the checkout process. As shown before the checkout process after the update of the delivery method will convert all the Cart related objects into Order ones, so the data following data flow stops right before the Cart to Order action. We will after see how the data flows continues with the new objects.


Web Cart
Cart Delivery Group
DB
Is FIRST Item in the Cart ?
User clicks on Add To Cart
YES
Create new WebCart and Cart Delivery Group
Web Cart
Cart Item
NO
Create new Cart Item
System update of Web Cart state to «Checkout»
Order Delivery Method
Keep Shopping?
YES
NO
User clicks on checkout button
System creates new Cart Delivery Group Methods for each possible Order Delivery Method
Cart Delivery Group
DB
User picks a delivery Address and a Shipping Method
Cart Delivery Group Method
System Updates Delivery Address and Delivery Method on Cart Delivery Group
Account Shipping Addresses
Chosen Delivery Method
Chosen Shipping Address
DB

So when the user clicks on add to cart button of an object, the system will firstly understand if the item added to a cart by that user is the first one or not. If it is the first element, it means that there is still no web cart created, so it will create a Web Cart instance and a related Cart Delivery Group Instance and put them in the Db, then it will create the new Cart Item instance and add it to the Db too. Once the user decides to proceed to the checkout the system will present him with a screen flow to pick a shipping address and a delivery method. To present the user with this information the system queries the DB to gather the related data and create for each possible order delivery group a corresponding cart delivery group method, that stores the information about that specific shipping cost. When the user picks the two options and proceed, the system saves the ids of the chosen items in local variables to pass them to the next action. The following activity is the update of the cart delivery group with the shipping and delivery information.

Once this is completed the Order objects comes in play. So we will again first discuss the relationships between the Order related entities and then we will see how the data flow continues.



As you can see the order entities reflect the cart ones. The Order object keeps track of the Account and Billing information, other than the Payment information, like purchase order number and the prices. Every time there is the word “Adj” or “Adjustment” in the prices, it stands for the discounts applied. It could be coupons or volume discounts. As usual, the Order is related to one or many Order Items, which represent the single lines of the order and they basically include the same information of the Cart Items objects. The Order Delivery Group remains the object to keep track of the shipping information and is also stores the information about the chosen Order Delivery Method. This last entities is exactly the same of the schema related to the Cart, indeed, this entity is not recreated when moving from the cart to the order, but it is always the same. As you may notice we have lost the information of the Cart Delivery Group Method, because we do not have a

Order Delivery Group Method. So how can we store the information about Shipping Costs? Well when a Cart is transformed into an Order the Shipping Cost will be saves saved as an Order Item of type Charge, to distinguish it from the other Order Items, which will have the type Product. It follows the official prospect of how the Cart is transformed in an Order during the checkout phase.

ORDER OBJECT

Order Field	Required for Order Summaries	Required for B2B Orders	Field Type	Value
`Id`	Yes	Yes	EntityId	*Generated at runtime
`AccountId`	No	Yes	EntityId	*WebCart.AccountId*
`EffectiveDate`	Yes	Yes	Date	*CreatedDate*
`BillingCity`	No	No	Address	*WebCart.BillingCity*
`BillingCountry`	No	No	Address	*WebCart.BillingCountry*
`BillingEmailAddress`	No	No	Email	*WebCart.GuestEmailAddress*
`BillingLatitude`	No	No	Address	*WebCart.BillingLatitude*
`BillingLongitude`	No	No	Address	*WebCart.BillingLongitude*
`BillingPhoneNumber`	No	No	Phone	*WebCart.GuestPhoneNumber*
`BillingPostalCode`	No	No	Address	*WebCart.BillingPostalCode*
`BillingStreet`	No	No	Address	*WebCart.BillingStreet*
`BillingState`	No	No	Address	*WebCart.BillingState*
`CurrencyIsoCode`	Yes	Yes	CurrencyCode	*WebCart.CurrencyIsoCode*

`OrderedDate`	No	No	DateTime	*CreatedDate*
`OwnerId`	No	Yes	Reference	*WebCart.OwnerId*
`PoNumber`	No	No	String	Collected at checkout.
`SalesStore`	No	Yes	EntityId	*WebCart.WebStoreId*
`Status`	Yes	Yes	DynamicEnum	*Draft*

ORDER DELIVERY GROUP OBJECT

OrderDeliveryGroup Field	**Required for Order Summaries**	**Required for B2B Orders**	**Field Type**	**Value**
`Id`	Yes	Yes	EntityId	*Generated at runtime
`DeliverToStreet`	No	No	Address	*CartDeliveryGroup.DeliverToStreet*
`DeliverToCity`	No	No	Address	*CartDeliveryGroup.DeliverToCity*
`DeliverToState`	No	No	Address	*CartDeliveryGroup.DeliverToState*
`DeliverToPostalCode`	No	No	Address	*CartDeliveryGroup.DeliverToPostalCode*
`DeliverToCountry`	No	No	Address	*CartDeliveryGroup.DeliverToCountry*
`DeliverToLatitude`	No	No	Address	*CartDeliveryGroup.DeliverToLatitude*
`DeliverToLongitude`	No	No	Address	*CartDeliveryGroup.DeliverToLongitude*
`DeliverInstructions`	No	No	TextArea	*CartDeliveryGroup.ShippingInstructions*
`DesiredDeliveryDate`	No	No	Date	*CartDeliveryGroup.DesiredDeliveryDate*
`OrderDeliveryMethodId`	Yes	Yes	EntityId	*CartDeliveryGroup.OrderDeliveryMethodId*

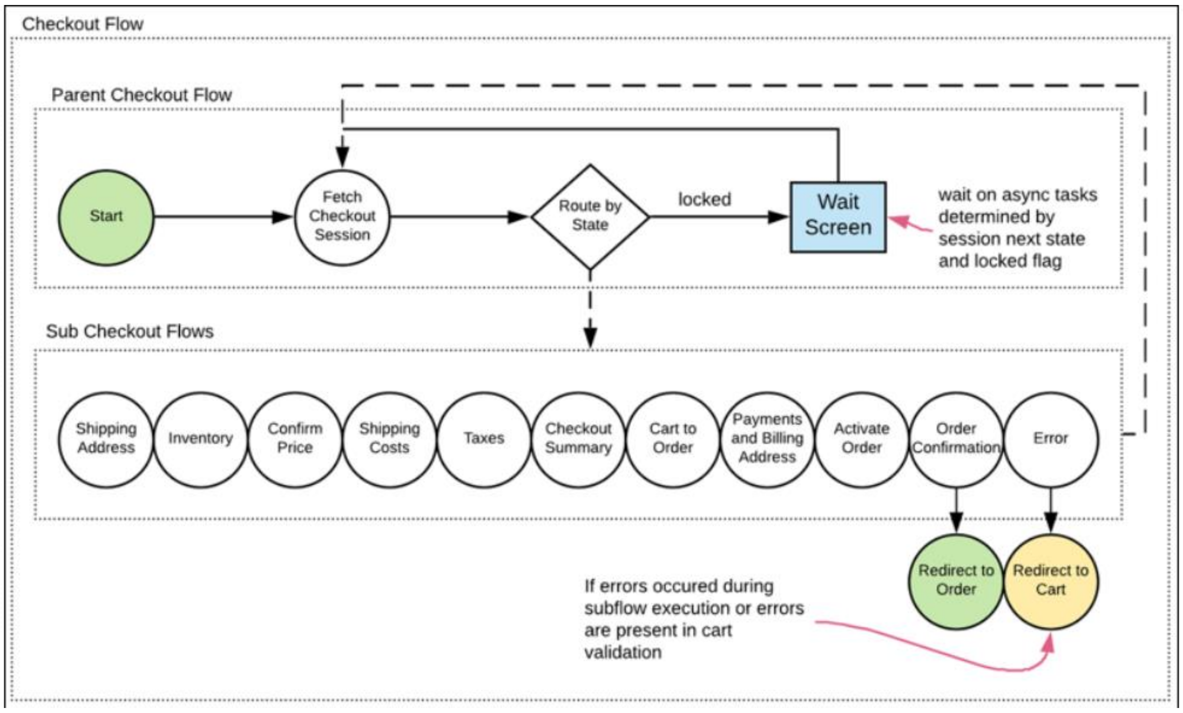
`OrderId`	Yes	Yes	EntityId	*Order.Id*
`DeliverToName`	Yes	Yes	Text	*CartDeliveryGroup.DeliverToName* **Note**

ORDER ITEM OBJECT

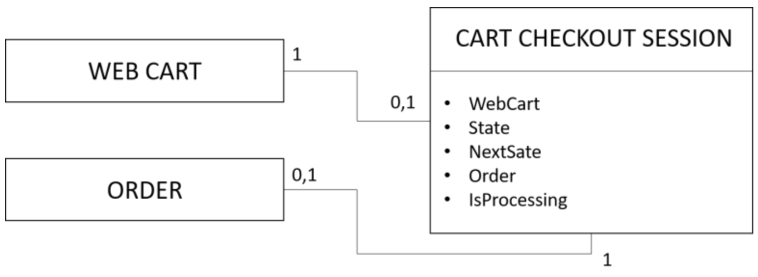
OrderItem Field	**Required for Order Summaries**	**Required for B2B Orders**	**Field Type**	**Value**
`Id`	Yes	Yes	EntityId	*Generated at runtime
`OrderId`	Yes	Yes	EntityId	*Order.Id*
`OrderDeliveryGroupId`	Yes	Yes	EntityId	ID of `OrderDeliveryGroup`
`Product2Id`	Yes	Yes	EntityId	*CartItem.Product2Id*
`Quantity`	Yes	Yes	Double	*CartItem.Quantity*
`TotalLineAmount`	No	Yes	Currency	*CartItem.TotalLineAmount*
`Type`	No	Yes	Picklist	*CartItem.Type*
`UnitPrice`	Yes	Yes	Currency	*CartItem.SalesPrice* or *CartItem.ListPrice* if `SalesPrice` is empty.
`ListPrice`	No	No	Currency	*CartItem.ListPrice* or *CartItem.SalesPrice* if `ListPrice` is empty.

Once the order is created the payment is proposed to the user. The payment can be managed in many different ways and can be performed using different methods. The easiest way (from the managing point of view) is using Purchase Order. This means the User will simply write a Purchase Order Number that is saved by the system directly on the Order (and later on the Order Summary) object. This solution does not need any type of integration with external systems. Sometimes you do not trust your buyers enough to allow a purchase order payment, so you can for example use a credit card payment method. To do so it is needed an integration with an external system. We will see the example of an integration with Stripe system in the last section. In this case, what happens is that during the payment the system creates an instance of the Payment Authorization Object, which stores, among all, the information about: the Amount, the Balance, the Currency Iso Code, the Payment Date and Status (pending, succeeded, failed) and also the Gateway Reference Number that is needed to trace the payment on the external Platform. After the payment is completed, the Order is Activated, this operation locks the Order instance so it won't be modifiable anymore, basically the order is completed and the User can not go back and change something. In particular, one last family of objects is created, the Order Summary ones. These objects are very much similar to the Order, so I won't show it again in a class diagram. The main difference is that now we also have a Order Payment Summary instance that is linked to a Payment Authorization and together they store all the necessary information to trace the payment from the Order Summary.

From a more technical point of view, the checkout flow will be structures as follows. The picture is taken directly from Salesforce documentation.



What makes the flow move forward are the State of the Checkout Session Object, which also contains the pointers to the Web Cart Id and Order Id. States are totally configurable, but the standard ones are: Start, Shipping Address, Inventory, Confirm Price, shipping Cost, Taxes, Checkout Summary, Cart to Order, Payment, Activate Order, Order Confirmation and Error. When the flows starts, the state is “Start”, so the decision node will redirect and execute the subflow, whose activation condition matches the “Start” state. In the standard case, as you can see from the picture, the first subflow will be Shipping Address, which will make the user choose a Delivery Address. This subflow is activated if the state is Start or Shipping Address, for all other subflows, the activation condition is just their state. In each subflow, if no error occurs the state is updated with the next one, otherwise, if an error occurs, it will be updated with the “Error” state.



5. A PRACTICAL EXAMPLE OF A B2B PROJECT ON SALESFORCE B2B LIGHTING COMMERCE

In this second part of the script, I would like to present a practical example of a real business to business company that has chosen to switch from a traditional way of selling to a more modern and much more efficient e-commerce website. The company’s core business is selling professional refrigerators to other companies and businesses (of different sizes). An example of a client could be a supermarket or a restaurant. The clients can, indeed, differ from small businesses to very large worldwide known companies. This makes extremely important being able to identify the clients and to divide the accounts in representative buyer groups, in order to provide targeted treatments. It goes without saying that one big client might be worth basically what hundreds of other small businesses are. This doesn’t mean that small clients are to be less considered, but on the contrary, it’s important to be able to recognize the clients and segment them appropriately.

The discussion will include the explanation of the project, the functional analysis to better understand the context and the most interesting pieces of code used for the actual implementation.

5.1 PROBLEM EXPLANATION

The project was divided into two main releases, called waves. The idea was to being able to go live fast with almost all the functionalities, leaving out only some more time-consuming and not core functionalities to the second release. The company used to sell its product through direct phone call from the clients to their referent seller, this meant that prices and promotions were discussed in a one to one relationship with each client. It goes without saying that this is extremely inefficient both for the client and mostly for the company. Just imagine if the company decides to make some discounts for specific clients (for examples clients that had purchased more than a certain amount of products in the last year), then it would have to contact all the sails representatives of those clients and inform them about the sale. Then each sales representative would have to call all its client who meet the requirements to access the discount and inform them about the new offerings. Or let's say that the company decides to create some discounts for new clients, then again it would have to pass through all its sails representatives to inform them about the new policy. But this is not all. Let's say the company wants to have some information about its clients to conduct a survey and understand which is its market segment, or how to improve its marketing campaigns. The company does have access to the clients basic information, like company Name and contacts, but doesn't know other fundamental information, like the history of all purchases. Another inefficiency could be also the fact that new clients are obtained only if the sails representatives find them, while exploiting the power of a website we can manage the possible client to find the company. Nowadays anybody that needs something, simply goes on the internet and search for it. Specially small business who need a new refrigeration system they tent to go on the internet and search for offerings, so why not exploiting this totally free sale possibility? We could actually go on with a thousand of other reasons why that way of selling was inefficient but they all converge in one point: having the possibility to access all clients at once, means having one single system that both the company and its client can access an exploit to buy or sell. That is exactly why the company decided to use Salesforce B2B Commerce platform, to implement its brand new ecommerce website, have the possibility to access all possible information on all its clients and make any type of change (on products, prices, marketing campaigns, etc..) available to its customers at once.

5.2 REQUIREMENTS

The sales countries of the organizations are Italy and France, therefore the website is completely translated in one of the two languages based on the browser setup. On the other hand, since it’s always good practice to write code that can be eventually expanded, the project also included the possibility to extent the website to all possible countries, by adding the related translations.

The basic functionalities included are:

- User Registration and Management
- Products, Catalog and Prices Management
- Cart and Checkout Management

Obviously, this short list is a very high level view, so now I will break down each functionality in its user stories. This will help understand what we want to achieve and therefore what we will need to implement. As we will see user stories are written in a non-technical way, they do not include anything about how the implementation will be done, but they simply specify the company expectations of how the website will work. User stories are, indeed, just an informal way of describing the features to implement using the natural language, in order to have a common understanding between the developers and the business level.

5.3 USER REGISTRATION AND MANAGEMENT

Firstly, let’s talk about the user registration. This functionality must be divided in two main requirements:

- User Self Registration
- User Import

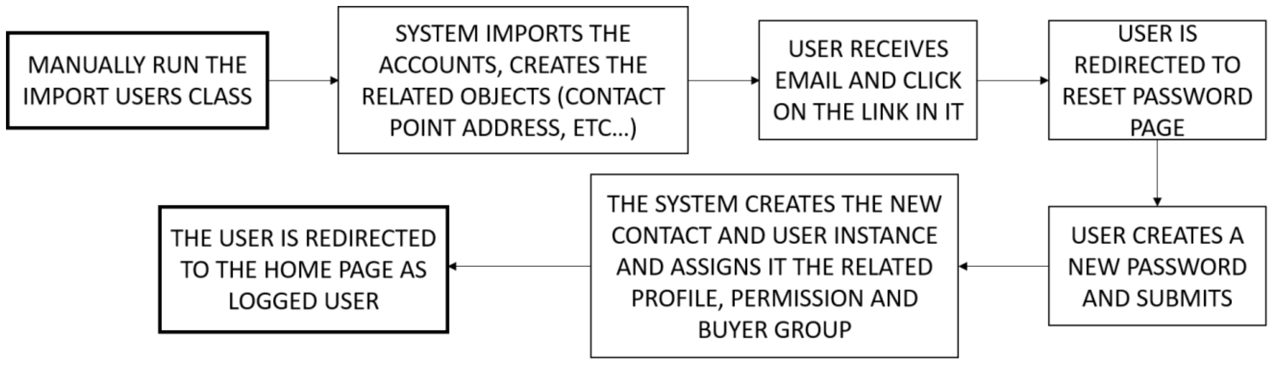
New users must be able to create a personal account just by visiting the website and completing the registration form. On the other hand, the company already has many clients and we want to import the information of this clients directly from the company database systems, without the need to make the customers fill up the registrations form. Therefore the website will have a complete registration form and process for new user and also a special registration (containing only the creation of a password) for all the previous customers of the company, for which the information will be uploaded automatically (through a process) from the company database.

USER STORY	STEP	DESCRIPTION	EXPECTED RESULT	POSSIBLE MISTAKE CASES
User Import	1	The company exports all clients information to the website. Data included is the same of the one inserted in registration form, plus the following: • Billing Address • Shipping Address	- Important information are used to create the Accounts - Creation of a new User and Contact instances for each. - Creation of two Contact Point Addresses related to the billing and shipping addresses. - Creation of a Buyer Account for each Account and associate it to a Buyer Group.	If the imported Account has the 'Account For Ecommerce' flag set to false than the record must be skipped and no account created
	2	The client receives an email with a link to complete the registration. The Client clicks on the link and arrives on a Change Password page	- Email created with user information and sent	
	3	The client inserts the new password and click on the registration button. The client is redirected directly to the home page as authenticated user	- The new User is created and activated in the organization. Then it is associated with that Contact and with the Customer Community Plus Login User profile and Customer 2 Permission Set. - The Account become active.	
New User wants to submit its registration to buy products	1	- Clients click on Login button and then on Create new Account link.	- Clients gets on the registration page	
	2	Clients insert the following information: • *Country (France/Italy) • *Contact Name • *Contact Last Name • *Company Name • *PIVA (TVA for France) • *Email and confirm Email • *Phone Number • *SDI (only for Italy) • *PEC (only for Italy) • *Fiscal Code (only for Italy) • SIREN (only for France) • SIRET (only for France) • *Accept terms and conditions checkbox • Accept Marketing checkbox • *Accept Privacy Policy checkbox Rules: fields must have specific checks on the content inserted	If the specific PIVA is not already present in the DB: - A new Account is created with all the related information - The account is associated with a Buyer Account and a new contact	If the PIVA is already present on the DB an error message must be shown to the user. Same if any check on the inserted field fails.

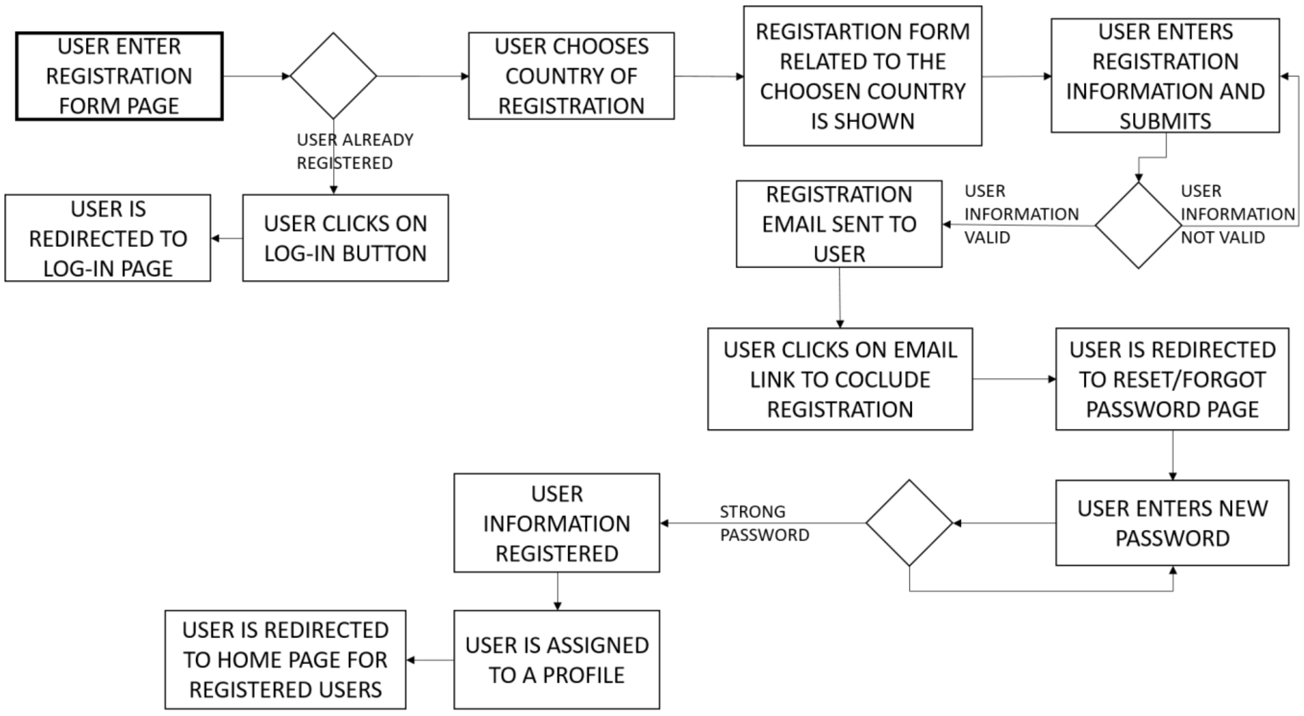
		and fields with * are mandatory. At least one between PEC and SDI must be evaluated.		
	3	Client receives a confirmation email with a link to complete the registration. (the email used is the client email inserted in the form)	A new User instance is created associated with the Contact. The assigned Profile is Customer Community Plus and the assigned Permission Set is Customer. Account becomes active.	

From the user stories we can understand in a high level way the final expected results of the account creation.

In the User Import case there is not a complex process to follow, indeed the new accounts are created by simply running a snippet of code once (on the launch of the new website) in order to import all the previous company clients from their systems to Salesforce platform and inform them of the brand new e-commerce website to make the orders. The users will then receive an email asking to complete the registration by clicking on the link contained. The users will be redirected to the reset Password page, were they can create their new password and complete the registration. From the system point of view, the Accounts and related Contact Point Addresses are created during the import form the company DB, then the Users and Contacts will be created when the client actually completes the registration by creating a new password. The process look as follows:


MANUALLY RUN THE IMPORT USERS CLASS
SYSTEM IMPORTS THE ACCOUNTS, CREATES THE RELATED OBJECTS (CONTACT POINT ADDRESS, ETC…)
USER RECEIVES EMAIL AND CLICK ON THE LINK IN IT
USER IS REDIRECTED TO RESET PASSWORD PAGE
USER CREATES A NEW PASSWORD AND SUBMITS
THE SYSTEM CREATES THE NEW CONTACT AND USER INSTANCE AND ASSIGNS IT THE RELATED PROFILE, PERMISSION AND BUYER GROUP
THE USER IS REDIRECTED TO THE HOME PAGE AS LOGGED USER

For the self registration case, the process is a little bit more complex. Once the User gets on the self register page must firstly pick the country for the registration, then the related form will be shown to him (as specified in the User stories there are some different attributes used between the French and the Italian clients). Once the user enters the information, the system performs all the necessary checks on the field values and if everything is correct, it sends the user a confirmation email. The user will then be redirected, exactly like before, to the Reset Password page to create a new password and complete the registration. Finally the user will be redirected to the home page for logged users. You can see the process in the following activity diagram:


USER ENTER REGISTRATION FORM PAGE
USER ALREADY REGISTERED
USER CHOOSES COUNTRY OF REGISTRATION
REGISTARTION FORM RELATED TO THE CHOOSEN COUNTRY IS SHOWN
USER ENTERS REGISTRATION INFORMATION AND SUBMITS
USER IS REDIRECTED TO LOG-IN PAGE
USER CLICKS ON LOG-IN BUTTON
REGISTRATION EMAIL SENT TO USER
USER INFORMATION VALID
USER INFORMATION NOT VALID
USER CLICKS ON EMAIL LINK TO COCLUDE REGISTRATION
USER IS REDIRECTED TO RESET/FORGOT PASSWORD PAGE
USER INFORMATION REGISTERED
STRONG PASSWORD
USER ENTERS NEW PASSWORD
USER IS REDIRECTED TO HOME PAGE FOR REGISTERED USERS
USER IS ASSIGNED TO A PROFILE

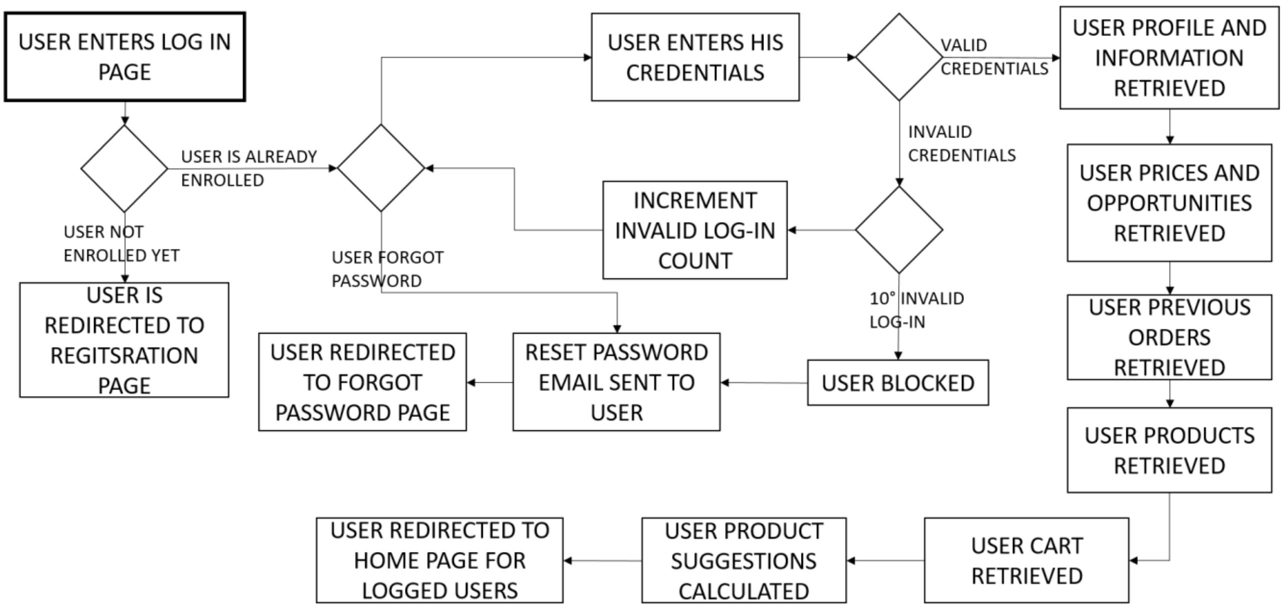
Now that we have understood the processes for the User Registration we can move forward to analyze the user login process. The situation is a quite standard one, the user has created username and password and wants to login to its account. The user stories from the client are the following:

USER STORY	**STEP**	**DESCRIPTION**	**EXPECTED RESULT**	**POSSIBLE MISTAKE CASES**
Client wants to log in the website	1	Client gets on the store and clicks on login button	Login page is shown	
	2	Clients inserts username and password	System verifies the credentials inserted	If credentials are wrong the user must be informed with an error message “User not found”
	3	Client access the store. The website is in the language associated to the Country related to that Account		

Since the user login process intersects the forgot password one we add here also the user stories related to it. Notice that the forgot password process is used also during the registration phase when the user completes the registration and is redirected to the Reset Password page.

USER STORY	STEP	DESCRIPTION	EXPECTED RESULT	POSSIBLE MISTAKE CASES
Client has forgotten his password	1	Client gets on the store and clicks on login button and then on forgot password link	Reset Password page is shown	
	2	Clients inserts username	System verifies the username and sends an email to the user with a link to reset the password	If the username doesn't exists, the user receives an error message
	3	Client receives the email and resets the password	System verifies that the password matches some standard security criteria on length and characters type	If the password isn't strong the client is asked to create a new one
	4	Client receives an email that the password was reset and can now log in the website		

Since the Reset Password page is actually shared between users who wants to create their first password and clients who want to reset their forgotten one, we need a way to distinguish the two cases to present the user two slightly different pages (for example with two different titles). In order to do so the Account object has the attribute "HasResetPassword__c" set to false for new users creating a password for the first time, set to true for users who forgot their password. We will see the attribute later in the Data Model. For what concerns the login process, what happens is that the user gets on the log in page and can either click on the forgot password link or insert his credentials to be authenticated. In the first case he will be redirected to the Reset Password page and the process will continue exactly like explained for the registration completion. In the second case the system will check the credentials, if the user gets the wrong credentials ten times in a row the account is blocked and the user will be asked to reset his password. Otherwise, the user is logged into the website, the system firstly retrieves all his information and then redirects the user to the home page for logged in users. The process is the following:


USER ENTERS LOG IN PAGE
USER IS ALREADY ENROLLED
USER NOT ENROLLED YET
USER IS REDIRECTED TO REGITSRATION PAGE
USER FORGOT PASSWORD
USER ENTERS HIS CREDENTIALS
VALID CREDENTIALS
INVALID CREDENTIALS
USER PROFILE AND INFORMATION RETRIEVED
INCREMENT INVALID LOG-IN COUNT
10° INVALID LOG-IN
USER BLOCKED
RESET PASSWORD EMAIL SENT TO USER
USER REDIRECTED TO FORGOT PASSWORD PAGE
USER PRICES AND OPPORTUNITIES RETRIEVED
USER PREVIOUS ORDERS RETRIEVED
USER PRODUCTS RETRIEVED
USER CART RETRIEVED
USER PRODUCT SUGGESTIONS CALCULATED
USER REDIRECTED TO HOME PAGE FOR LOGGED USERS

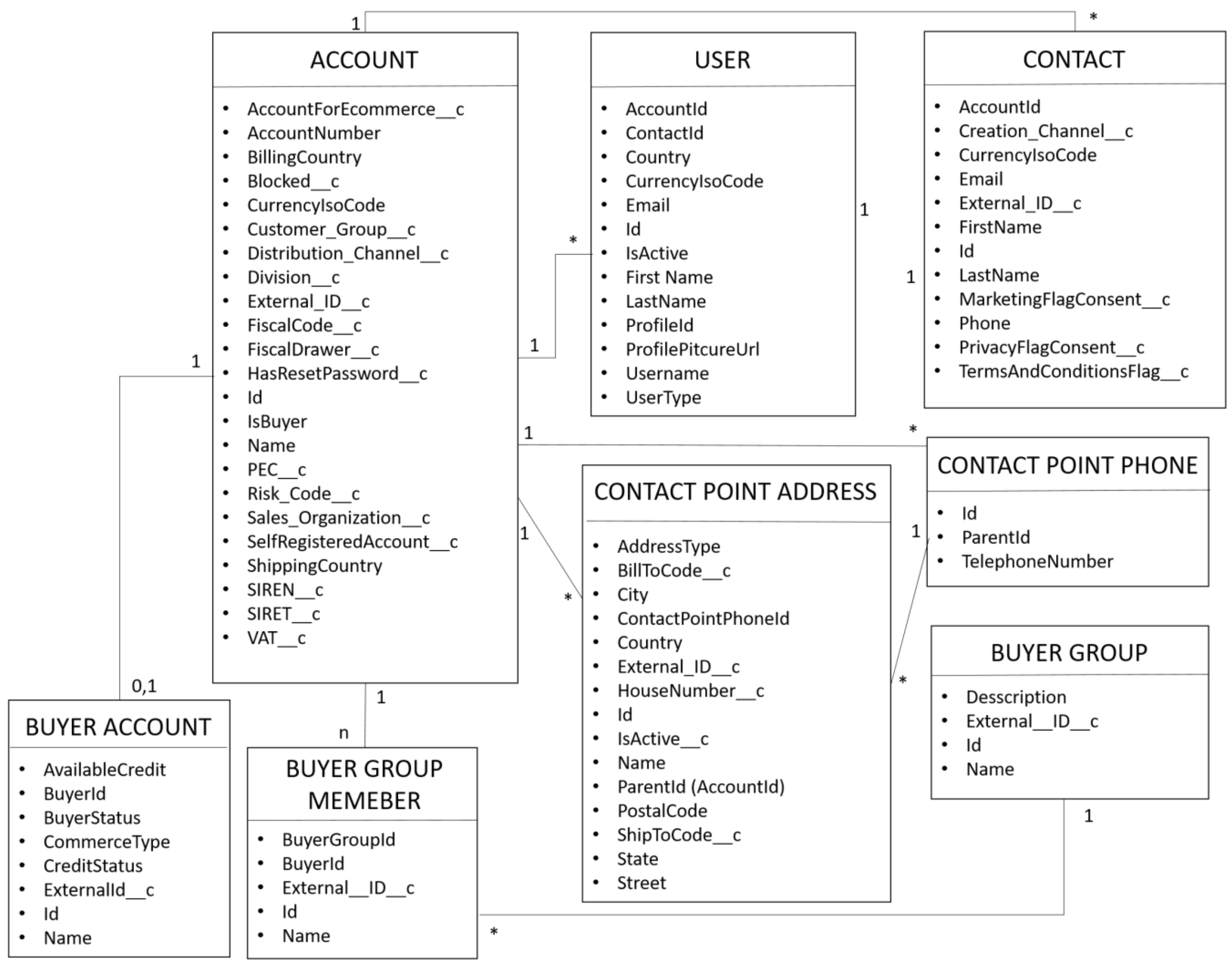
Look at the final 'APPENDIX' section to see:

- the Reset Password page implementation: code 2
- the Login Form page implementation: code 3
- the Self Registration page implementation: code 4
- the User Import Flow implementation: flow 2

Data Model

Each B2B Customer must belong to a single Account. An Account may have multiple contacts and each contact is associated with a User and has its own credentials (username and password). Basically, the company is represented by the Account object and for each company we might have many employers (Contacts) which are allowed to log in and make purchases. In order to see products, each Account is then associated with a Buyer Group through a Buyer Group Member object. We will see in the next session how Buyer Groups are then associated with product prices.

The following class diagram better explains the relationship among the entities with their attributes. (Notice that there are more attributes than the ones shown, since Salesforce uses many standard attributes like "CreationDate" or "LastModifyDate", which however are not interesting for our discussion and will therefore be left out). The entity shown are: User, Contact, Account, Buyer Group Member, Buyer Group, Contact Point Address and Contact Point Phone. For each object you can see the relationships with the others and its attributes. The attributes with a name that ends with "__c" are the custom attributes (the same notation is used for custom Objects, but in this case only standard Object are needed).



Starting from the Account, we can see all the attributes needed to store the registration form information. In particular, notice that in the user stories related to importing Users, we want to import only accounts with the AccountForEcommerce__c flag set to true. The Blocked__c attribute is another interesting one, it is used to verify if a specific account has been disabled by the company. Indeed the company can decide from the backoffice to block a client Account and therefore all contacts related to that account will still be able to access the website and log in but will not be able to make any purchase. The SalesOrganization__c attributed refers to the Area of purchase of the client, in particular there are three areas, two for France customers and one for Italian ones. This attribute is used in the Check Inventory Requests, since the same Product might have different availabilities respect to the Area it is sold to, so the HTTP request to the external service must include this information. It is also important to underline that the information about Marketing flag Consent, Privacy Policy Flag Consent and Terms and Conditions Flag Consent, which were all asked at registration time, are saved on the Contact object and not on the Account one, since the consent is related to the single user and not to the company. Each Account can have many Contacts and many Users referring to it. While the relationship between Contacts and Users is one to one. In order to be allowed to make purchases an Account must be associated with a Buyer Account object, so the relationship here is again one to one.

The Buyer Account object stores important information related to the Account credit and credit status, which the company can exploit to decide the financial solidity of its clients. In particular for this project we used the attribute Risk_Code__c on the Account object to distinguish risky clients from the others. When created each Account self registered was assigned to Risk_Code__c = 'Z6', that means risky client, while for Imported client the risk code value was retrieved from the external system. The company can decide to change the code at any time. This code is used at Checkout time since only trustworthy clients (therefore with Risk_Code__c != Z6) have the possibility to pay with Purchase Order, instead of only with Credit Card like the other ones.

Each Account is also associated with one or more Buyer Group through the Buyer Group Member object. Buyer Groups are really important because they allow to assign different prices to the same object and make them available to the Accounts enrolled in that Buyer Group. Since each Account could be associated with many Buyer Groups the default behavior is to show always the lowest price available for that user. We will see later how price books and buyer groups are associated, for now it is important to understand that each Account can see different prices for the same object thanks to its association with at least one Buyer Group. In practice we implemented 7 different buyer groups (and therefore 7 different price books, since the relation between the two objects is one to one), which are the following: Standard, IT Gold, FR Gold, IT Silver, FR Silver, IT Bronze, FR Bronze. The meaning of them will be explained later when we will talk about products and prices.

For what concerns the Contact Point Address Object, this entity is used to store the information about the addresses that will be presented to the user at Checkout time to make its purchase. The logic chosen for the management of this object is quite complex, therefore to support it, it was coded a totally custom form to create and edit it. The high level logic consists in the following rules:

- Each Account in order to make a purchase must have at least one Billing Address and one Shipping Address

- In particular only one Billing Address is allowed for each Account, while multiple Shipping Addresses are permitted
- If the Account is imported, then it will have exactly one Shipping Address and one Billing Address from the creation (the needed information is imported from an external system) and will be able to add other Shipping Addresses accessing the website “myAccount” page
- If the Account is self registered, then it is created without any Address associated, therefore it might happen that at checkout time it does not have any address available for billing and/or shipping. In this case the checkout page will give the user the possibility to create the needed address.
- Shipping Addresses can always be created either from the checkout page or from the “myAccount” page.
- Billing addresses once created can not be modified, while shipping addresses once created can be set to default or activated/deactivated. Only one and at least one shipping address is the default one, so if you set one shipping address to default the previous default address will be unset from the default one.

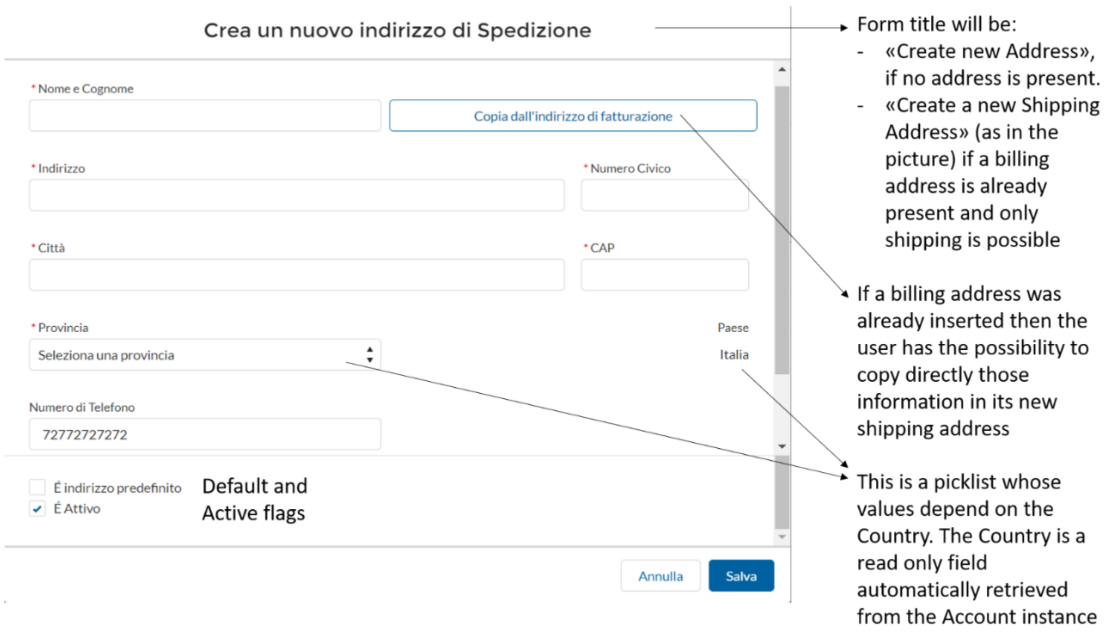
We can better analyze these specifics by splitting them in pre-conditions, user action and post-conditions. Notice that, in order to reduce the number of cases, when you see ‘User completes the form’, we imply that all the inserted information is correct, therefore it respects the fields requirements and no mandatory field is missing.

PRE-CONDITION	USER ACTION	POST-CONDITION
User doesn’t have any address yet	User completes the form selecting Address Type: Billing(Shipping)	A new billing(shipping) address is created and automatically set to default one and active.
User already has a billing address and no shipping addresses	The form presents only the possibility to create a shipping address.	A new shipping address is created and automatically set to default one and active.
User already has both a Billing and at least one shipping address	The form presents only the possibility to create a shipping address.	A new shipping address is created. If ‘Is default’ flag in the form is true, the systems unsets the old default shipping address

		and sets this one as the new shipping default one.
User edits a shipping address. (The form presents only the possibility to activate/deactivate the address or to set/unset the default option)	User deactivated a default address	The system returns an error message asking the user to first set another shipping address as default one and then deactivate this address.
User edits a shipping address	User sets to default a non active address	The system returns an error message asking the user to activate the address to make it the default one
User edits a shipping address	User deactivate a non default address or user sets to default an active address	The address is deactivated or the address is set to default and the old default address is updated as not default.

You can see the related code implementation in the Appendix section under Code1.

To better visualize the result you can see a screenshot of the actual form implemented.



Form title will be:
- «Create new Address», if no address is present.
- «Create a new Shipping Address» (as in the picture) if a billing address is already present and only shipping is possible

If a billing address was already inserted then the user has the possibility to copy directly those information in its new shipping address

This is a picklist whose values depend on the Country. The Country is a read only field automatically retrieved from the Account instance

Notice that the Phone Number is filled by default with the Contact phone number, but it can be modified and in this case a new Contact Point Phone is created related to that Account and to that Contact Point Address.

Finally, you can see in the form that no Address Type dropdown menu is present, since the form refers to the case in with the user can only create a shipping address. If, instead, there were the possibility to create either a billing or a shipping address then there would have been a dropdown menu to select the address type to be created (this would be the case in which the title of the form would be just "Create a new address").

As we said before also at checkout time there is the possibility to create a billing address (if missing) and a hipping address (in any case). For this situation, we will see that the page offers to the user two different buttons to create either a billing address (mandatory if not present) or a new shipping address (mandatory only if none is present), therefore we will have to possible forms: "create a new billing address" and "create a new shipping address", which will not present the address type dropdown, since they are specific and there is no choice.

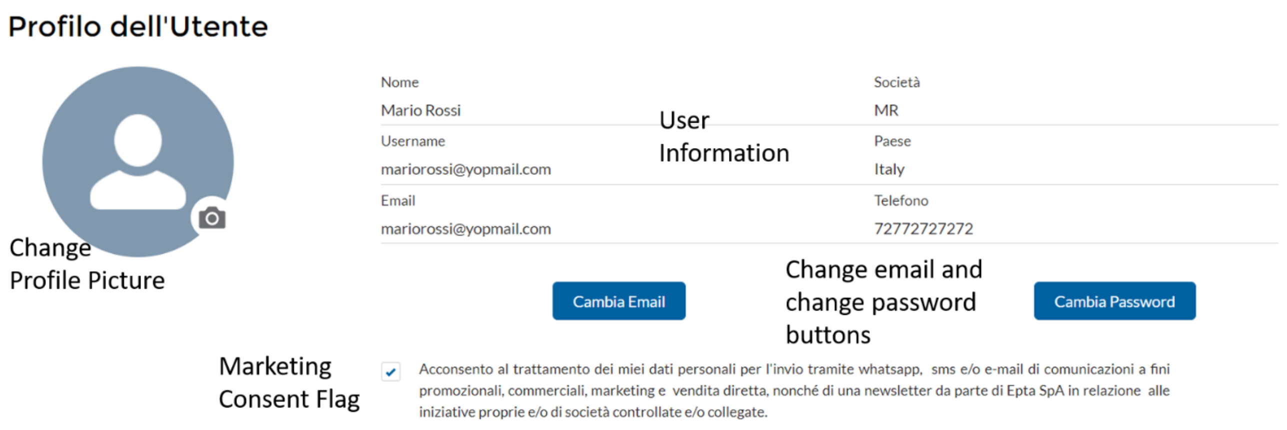
Another interesting part of the User management is related to the user Profile page, which allows the user to see his information, change his email and/or password, update his profile picture and change his Marketing Consent settings.

The change password process is very simple. The user needs to fill a form with a new password and submits. The systems performs some checks on the strength of the password and if everything is fine sets the new password and returns a success message, otherwise returns an error message. If the password was actually changed then the user is requested to log in the system again.

To change the Marketing settings the user can click on the marketing checkbox (which is selected or not depending on the previous value of the marketing consent flag for that user). Then the user will see a modal asking to confirm the decision of changing his marketing consents settings and if the user accepts the system updates the attribute on the client and changes the checkbox value at front end.

The profile picture update is simply a button that allows the user to import a png or svg picture from a local folder and upload it as his image. This image will substitute the standard user icon on the top right corner of the website header (where there is the user menu, viewable only to logged in users).

For what concerns the email update the process is slightly more complex. The user inserts the new email, then the system sends him two emails. One to the old email address of the client to inform him that the email was changed and the other one to the new email address to allow the user to confirm his new email address through a link. The client must click on the link and the system will automatically complete the update operation. If the user does not click on the link the system will not perform the update and the email will not be changed. Once the address is updated the user will see the new email address on his profile page. The profile page will look as follows:


Profilo dell'Utente

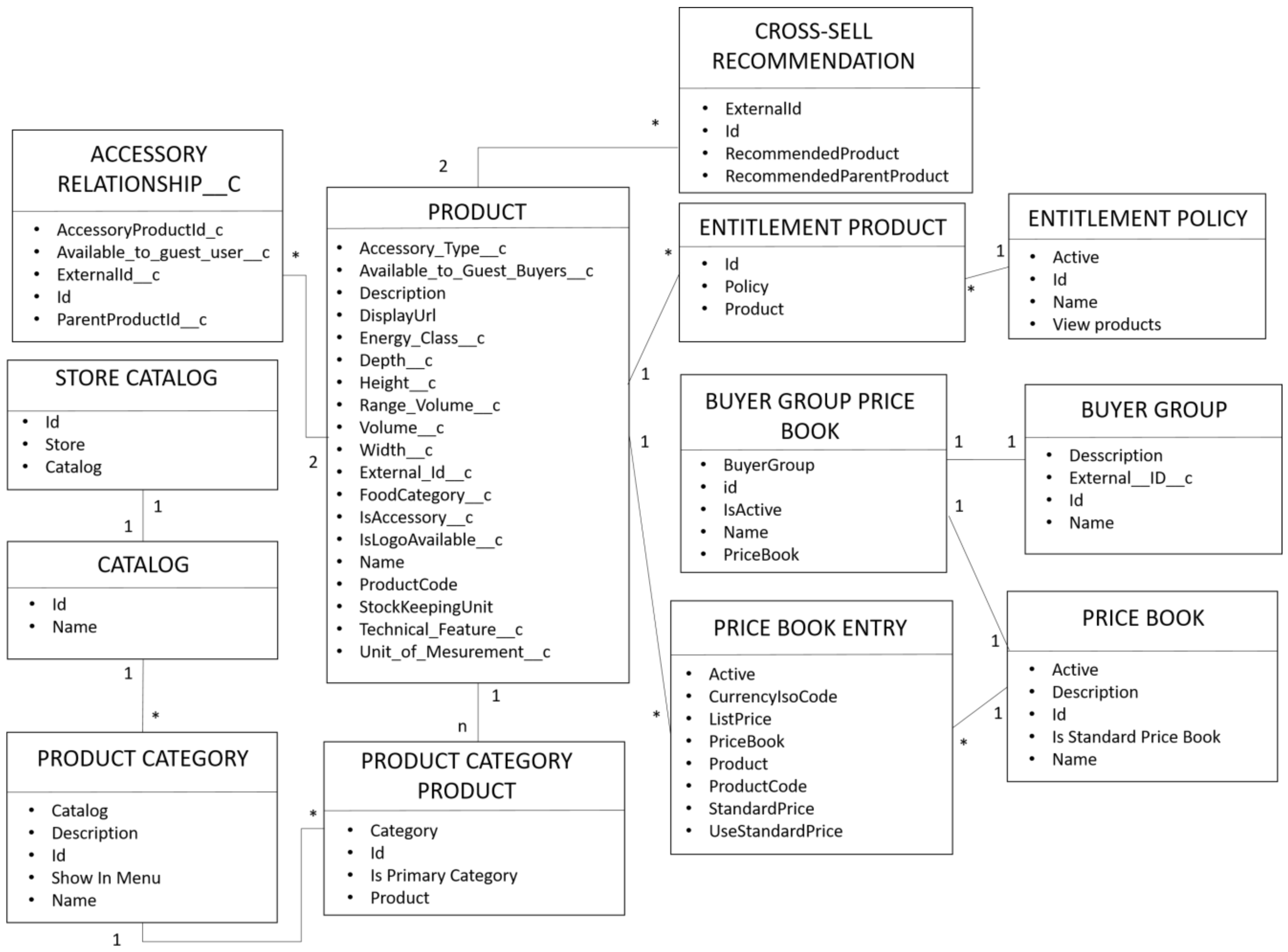
Nome	Società
Mario Rossi	MR
Username	Paese
mariorossi@yopmail.com	Italy
Email	Telefono
mariorossi@yopmail.com	72772727272

✓ Acconsento al trattamento dei miei dati personali per l'invio tramite whatsapp, sms e/o e-mail di comunicazioni a fini promozionali, commerciali, marketing e vendita diretta, nonché di una newsletter da parte di Epta SpA in relazione alle iniziative proprie e/o di società controllate e/o collegate.

The code implementation is at Code 5 in the Appendix section.

5.4 PRODUCTS, CATALOG AND PRICES MANAGEMENT

Salesforce B2B Commerce allows to configure a product catalog for each web store. In this case we have one single web store, therefore one single product catalog. All products that the company wants to sell must be associated with that product catalog in order to be available to clients. To do so we need to firstly create some Product Categories, then assign each product to one or more product category through the Product Category Product object. Finally you can assign each category to the previously defined catalog. So, the relation between products and categories is many to many, while each category car refer to only one catalog, but each catalog can have many categories. Here follows the Data Model of the objects of interest for this section:



The Store Catalog links the Catalog to the Store, and the categories are represented with the Product Category object, which has a many to one relationship with the Catalog object.

Categories and Products are linked through the Product Category Product class which uses the “Is Primary Category” attribute to define for each product which is its primary category. We can have categories without any product, while each product when is created is automatically linked to at least one category, therefore the relationship is “one to n” (at least one) with the product category product. Categories can have complex hierarchical structures composed of parent and sub categories. For the purpose of this project, the category structure was the following:

- Ice Cream and Frozen Food
    - Horizontal
    - Impulse
    - Scooping
    - Vertical
- Drink & Dairy
    - Counter

    - Horizontal
    - Impulse
    - Vertical

- Food Storage Equipment
    - Counter
    - Horizontal
    - Impulse
    - Vertical
- Vending
- Additional contents
    - Shelves
    - Basket
    - Priceholder
    - Bin
    - Sticker
    - Other

As you can see we have five parent categories and four of them have some sub category. Notice that categories are by default shown in the navigation bar of the website is the attribute “Show In Menu” of the specific category instance and of all its parent category instances is set to true. However the client wanted to have five tabs in the navigation bar, the first four tabs were related to the first four categories and their subcategories (if present), while the fifth one was just Stickers, which is a subcategory of Additional Content and therefore it would be viewable only as a subcategory of Additional Content tab in the standard implementation. To allow this, the header navigation bar component was completely rewritten according to the specific given by the client.

It is possible, with Salesforce Commerce, to decide which products each Account can see, by exploiting the use of many Entitlement policies, but for the purpose of this project, this was not needed, since all products are available to all users. This means firstly that only one entitlement policy was created and all Buyer Groups are associated to it, secondly that also all Products are associated with that entitlement policy. The products are put in relation with the Entitlement policy through the Entitlement Product object, which has a many to one relation with both the Product and the Entitlement Policy. Nevertheless, as we saw earlier, we created many Buyer Groups, because we want to be able to offer the clients different prices. Therefore, we have seven Buyer Groups associated one to one with seven Price Books. Note that a price book does not keep any type of information about the actual price of each product, but it behaves like a container of all the price book entries which actually store the prices. To sum up, for each price book we have many price book entries, which link a price book to a product. Each product can have many price book entries and so can be associated with many price books, but each price book can keep track of only one price book entry for a specific product. Price books and Buyer Groups are associated through the Price Book Buyer Group object. The price books defined for the project have the same name of the corresponding buyer groups: Standard, IT gold, FR gold, IT silver, FR silver, IT bronze, FR bronze. To access the product from the store, it must be associated with at least the Standard price book, this means each product at least has one price book entry with the Standard price book. This is the highest price assigned to that product. Guest users who access the website will see the Standard prices for the products. When a user logs in, he will then see the prices related to the price book of its buyer group. In particularly for imported user the buyer group is given by the external system, while for self registered users the system automatically assign the bronze buyer group (and as a consequence the bronze price book). The 'IT' or 'FR' are related to the Account country of the client. The company can decide to change a user buyer group from the back-office at any time. When browsing as a logged in user, the client is able to see for each product its standard price and the special price assigned to him. (Gold accounts have access to the best prices while Bronze ones have access to worst ones). Object prices might lower based on the quantity selected, indeed some products have special discounts, called Volume discounts if the quantity bought is higher than some breakpoints. This problem was handled exploiting a CVS file loaded on the platform and keeping track of the percentage of discount for each volume rage of each product that has a volume discount. We will see later how the product detail page looks like and how the Volume discount is presented to the client, if available.

Products are of two possible types: Simple products or products with accessories. The first ones are standalone products, while the second ones are sellable both as standalone and with customized accessories included. On the product detail page the user can specify the desired attributes to add to a product. The accessories (also represented through the Product object) are not sellable as standalone. Products with accessories can present this cluster of accessories:

1. Sticker: it refers to the sticker which can be applied to the product. For some stickers, there will be the possibility for the customer to insert their personal logo (eg. the name of their business) which will be applied on the sticker.
2. Shelves: it refers to the kit of shelves compatible and purchasable together with in the products (e.g. wooden shelves, molded shelves).

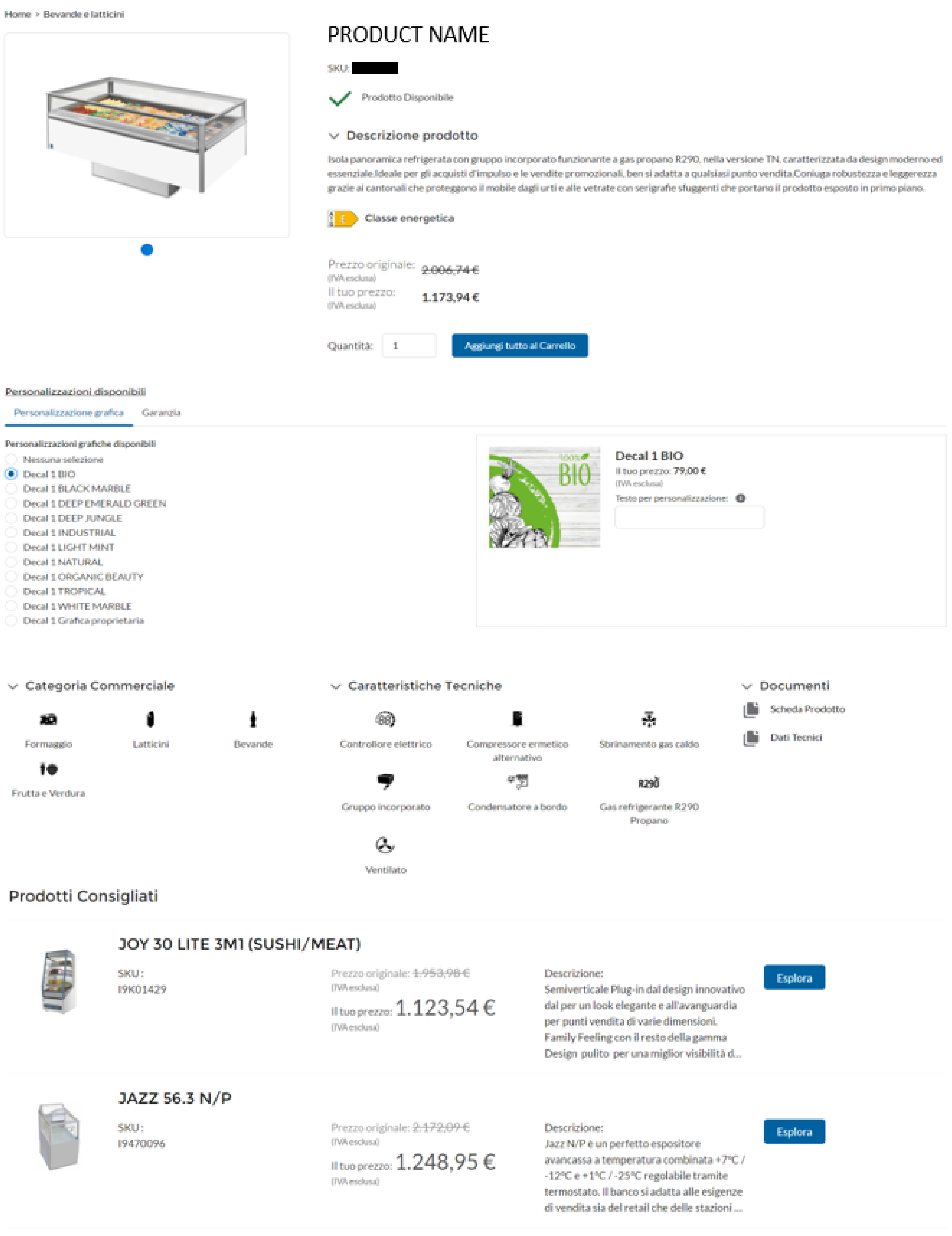
3. Priceholder: it refers to the kit of price holders compatible and purchasable together with the product
4. Basket: it refers to the kit of baskets compatible and purchasable together with the product
5. Bin: it refers to the kit of bins compatible and purchasable together with the product.
6. Other: it refers to the kit of other type of accessory compatible and purchasable together with the product (e.g. foot stainless)

Notice that the accessories correspond to the subclasses of the parent class “Additional Content”.

On top of the previous accessories, products might also have some additional services associated:

1. Warranty extension service: it is the warranty extension service associated to the product (it may be extra 1-year or extra 2 years warranty)
2. Eco participation tax: it is a tax that only French customers have to pay whenever they purchase a refrigerator.
3. Assembly: it is the cost of the assembly service of certain refrigerators for French customers. Cost varies depending on the refrigerator it is applied to.

In order to keep track of all the accessories of each parent product it was used the “Accessory_Relationship__c” object that links one parent product to an accessory one. Each accessory relationship has exactly two products linked, while each product might have zero or more accessory relationships. The “IsAccessory__c” attribute on the product is used to distinguish refrigerators from accessories, while the “Accessory_Type__c” attribute indicates the accessory name (among the ones listed above or is null for parent products). The other attributes, like dimensions, material, description, etc… are shown to the user on the product detail page, which is available only for parent objects. The implementation of the product detail page was almost completely custom, since the requirements were very specific for this company case. Indeed as you can see the Accessory Relationship object is a totally custom one and the whole management of parent and accessories products is totally custom. The final product detail page looks like this:


Home > Bevande e latticini

PRODUCT NAME

SKU: [redacted]

✓ Prodotto Disponibile

Descrizione prodotto

Isola panoramica refrigerata con gruppo incorporato funzionante a gas propano R290, nella versione TN, caratterizzata da design moderno ed essenziale.Ideale per gli acquisti d'impulso e le vendite promozionali, ben si adatta a qualsiasi punto vendita.Coniuga robustezza e leggerezza grazie ai cantonali che proteggono il mobile dagli urti e alle vetrate con serigrafie sfuggenti che portano il prodotto esposto in primo piano.

Classe energetica

Prezzo originale: ~~2.006,74 €~~
(IVA esclusa)
Il tuo prezzo: 1.173,94 €
(IVA esclusa)

Quantità: 1 Aggiungi tutto al Carrello

Personalizzazioni disponibili

Personalizzazione grafica | Garanzia

Personalizzazioni grafiche disponibili
- Nessuna selezione
- ● Decal 1 BIO
- Decal 1 BLACK MARBLE
- Decal 1 DEEP EMERALD GREEN
- Decal 1 DEEP JUNGLE
- Decal 1 INDUSTRIAL
- Decal 1 LIGHT MINT
- Decal 1 NATURAL
- Decal 1 ORGANIC BEAUTY
- Decal 1 TROPICAL
- Decal 1 WHITE MARBLE
- Decal 1 Grafica proprietaria

Decal 1 BIO
Il tuo prezzo: 79,00 €
(IVA esclusa)
Testo per personalizzazione:

Categoria Commerciale
Formaggio, Latticini, Bevande, Frutta e Verdura

Caratteristiche Tecniche
Controllore elettrico, Compressore ermetico alternativo, Sbrinamento gas caldo, Gruppo incorporato, Condensatore a bordo, Gas refrigerante R290 Propano, Ventilato

Documenti
Scheda Prodotto
Dati Tecnici

Prodotti Consigliati

JOY 30 LITE 3M1 (SUSHI/MEAT)
SKU: I9K01429
Prezzo originale: ~~1.953,98 €~~
(IVA esclusa)
Il tuo prezzo: 1.123,54 €
(IVA esclusa)
Descrizione:
Semiverticale Plug-in dal design innovativo dal per un look elegante e all'avanguardia per punti vendita di varie dimensioni. Family Feeling con il resto della gamma Design pulito per una miglior visibilità d...
Esplora

JAZZ 56.3 N/P
SKU: I9470096
Prezzo originale: ~~2.172,09 €~~
(IVA esclusa)
Il tuo prezzo: 1.248,95 €
(IVA esclusa)
Descrizione:
Jazz N/P è un perfetto espositore avancassa a temperatura combinata +7°C / -12°C e +1°C / -25°C regolabile tramite termostato. Il banco si adatta alle esigenze di vendita sia del retail che delle stazioni ...
Esplora

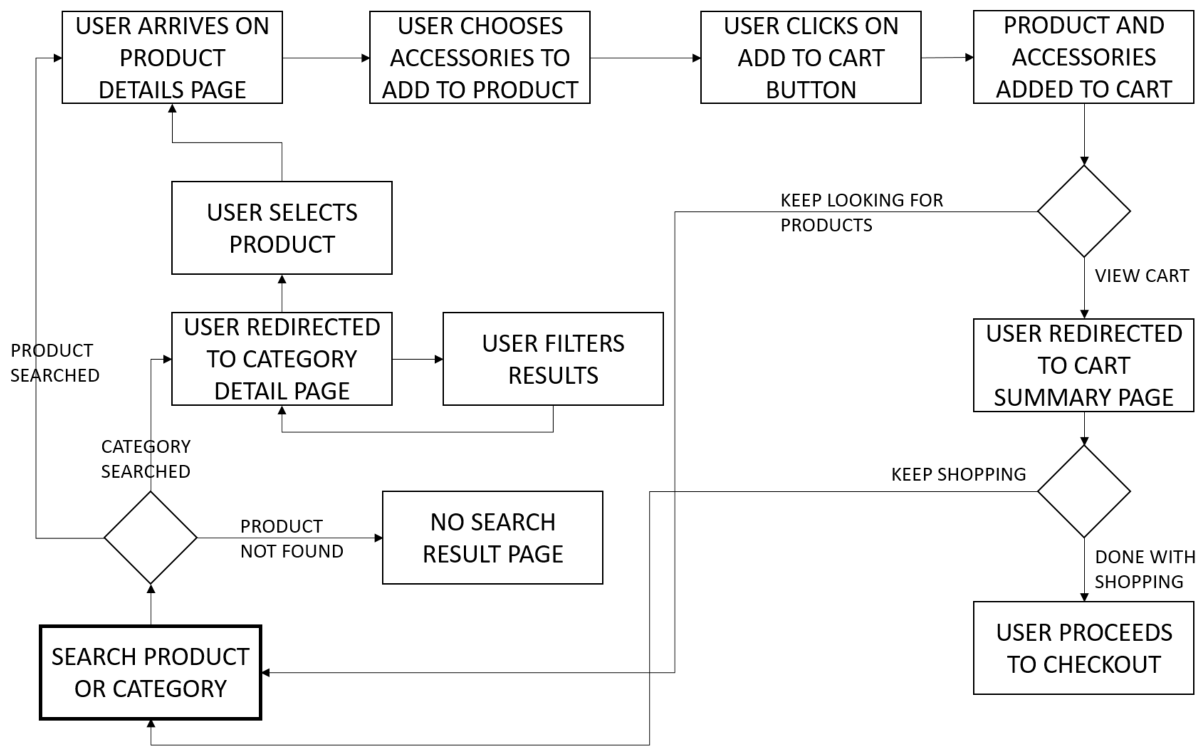
For each product we have its images, the main information, the standard price crossed and the user price. You can also notice the green tick that shows the client that the product is in stock. This is obtained through a check inventory process that exploits an external service call in order to retrieve the availability information of the product for the specified quantity. The solution implemented makes a new call each time the client changes the product quantity, this might not be scalable and definitely slows down the process, but these were the requirements given by the client. The check inventory service is called also at checkout time to verify again the availability of all the products in the cart right before the purchase, therefore we will better discuss this process later. Under this part there is a dynamic tab menu showing all the accessories that can be added for the product. The number and types of tabs shown depends on the accessories available for the specific product. For each added accessories the system opens a window on the right, showing the price and the information of that accessory. In the image you can see that for the chosen sticker the user can also insert a logo name to be added to the sticker. Below the accessory section you can see all the data related to the product (which are contained in the custom attributes of the Product object). To improve the user experience each commercial category and technical feature is accompanied by an illustrative icon. In the bottom part of the page the system presents the client some recommended products. This feature exploits the "Cross-Sell Recommendation__c" custom object, which represents the relationship between a parent product and a linked recommended one. Each product might be linked to many recommended ones and each product can be the recommended for many parent products. Each cross-sell recommendation is related to exactly two products. Notice that Salesforce B2B Commerce platform offers more sophisticated systems of product recommendations based on Artificial Intelligence routines, but these functionalities were not requested by the client. Indeed the solution used was totally custom and simply based on the objects stored in the Database, there was no dynamic logic applied. This means that every client visiting the product detail page of a certain object will see the same recommended product on the bottom.

The user stories related to the product navigation are the following:

USER STORY	**STEP**	**DESCRIPTION**	**EXPECTED RESULT**	**POSSIBLE MISTAKE CASES**
Client navigates the products	1	Guest user or logged in user search a product through the search bar or by clicking on one of the navigation bar tabs	All products related to the research are shown to the client. Logged user can see the prices related to its buyer group, while guest user sees standard prices. The page gives the possibility to filter the results based on Commercial category, energetic class, family, category or subcategory of the products. If the user uses the search bar the products are searched based	If there are no products that match the searched words, the user will be redirected to the No search Result Page

			on the best match with name and description.	
	2	User access the product detail page by clicking on a specific product	All product information is shown, together with the inventorial availability. The user can see accessory tabs and cross-selling recommendations	If the check inventory returns out of stock the add to cart button is disabled for that product
	3	User can add products and accessories to cart	User clicks on add to cart button and adds all the products selected with their accessories to his cart.	

The following activity diagram explains the process built from the given user stories:


USER ARRIVES ON PRODUCT DETAILS PAGE
USER CHOOSES ACCESSORIES TO ADD TO PRODUCT
USER CLICKS ON ADD TO CART BUTTON
PRODUCT AND ACCESSORIES ADDED TO CART
USER SELECTS PRODUCT
KEEP LOOKING FOR PRODUCTS
VIEW CART
PRODUCT SEARCHED
USER REDIRECTED TO CATEGORY DETAIL PAGE
USER FILTERS RESULTS
USER REDIRECTED TO CART SUMMARY PAGE
CATEGORY SEARCHED
KEEP SHOPPING
PRODUCT NOT FOUND
NO SEARCH RESULT PAGE
DONE WITH SHOPPING
SEARCH PRODUCT OR CATEGORY
USER PROCEEDS TO CHECKOUT

The User can search the product through the search bar on the header or can exploit the navigation bar available also in the header on the webpage. In case of products not found the No Search Result page is shown to the user, otherwise the client is redirected either to the product searched page (if the research bar was used) or to the category detail page (if the navigation bar was used). In the second case the page offers the possibility to the user to filter the results accordingly to the attributes specified in the user stories at step 1. Once the user has found the product he was looking for he can click on it and will be redirected to the product detail page, that we have already discussed plenty. If the clients wants to buy the product with the combination of attributes chosen he clicks on the add to cart button. If there are no mistakes the user will see on the top right corner of its page the cart icon with the notification of the new objects added and will also receive a success alert banner. If for some reason the operation fails (for example for a network temporary interruption) the user will see an error banner appearing on the top of the page. The client can then decide to keep shopping or can go to the cart summary page by clicking on the cart icon.

5.5 CART AND CHECKOUT MANAGEMENT

For this last section we will discuss the most important part of an e-commerce, that is the process that manages the web cart and the order purchase. At a high level point of view, the logged in client needs to be able to add a certain amount of products to his web cart. Each product can have different sets of associated accessories.

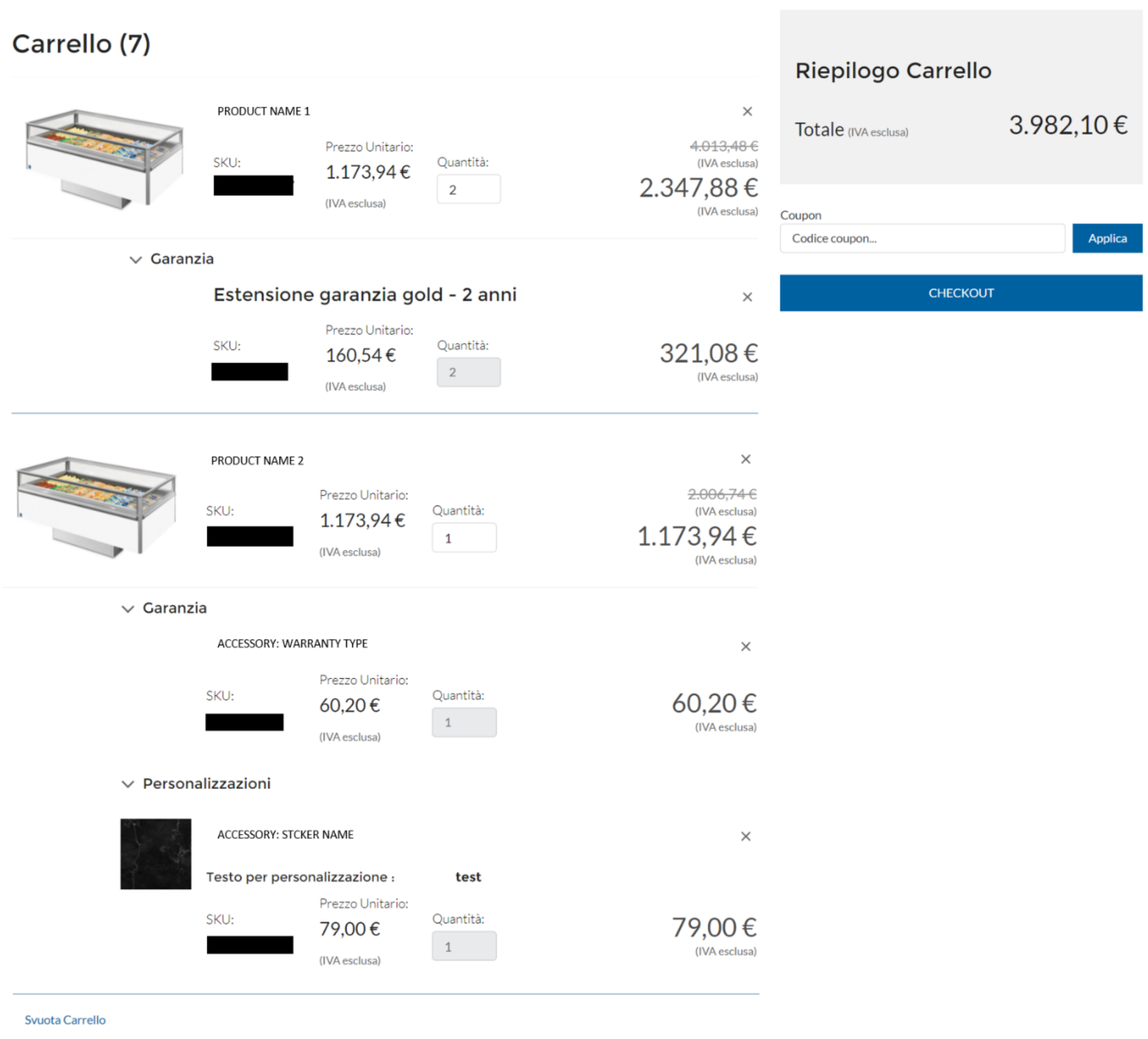
In case of French accounts, the user can also decide to add the Assembly service as an “accessory” of a product, that means that specific refrigerator will be assembled by the customer service. Always for French accounts, the system needs to automatically add for each refrigerator a tax called “Écopartecipation”.

The user can add to his web cart either from the product details page or from the quick order form on the home page.

- When adding from the product details page, the user can select a certain quantity for the specific configuration of refrigerator and accessories selected. The check inventory process will make sure that the picked products are available and, in the positive case, will activate the “add to cart” button.
- When adding from the quick order form, the user needs to insert the SKU of the chosen refrigerator and the desired quantity. The functionality doesn’t allow the user to select accessories too. As soon as the user inserts the SKU and the quantity of a product, the system will perform the check inventory to verify the availability. In case of positive response, the “add to cart” button will be enabled and the user will be able to add the items to his web cart.

When the user adds a product to the cart, the system firstly checks if there is already a web cart active for the account. If not it creates a new web cart instance and a related Cart Delivery Group instance (which we will see, is the object keeping all shipping data), otherwise it retrieves the information related to the existing cart and cart delivery group. Then it creates one cart item for each added element (refrigerators and accessories). Notice that also the écopartecipation tax and the assembly service are considered as accessories and therefore will have a corresponding cart item. In order to keep track of the accessories of each parent product, the cart item object stores the parent information in a custom attribute called “ParentAccessoryCartItem__c”.

The user can check his web cart on the cart summary page, that looks as follows.



Carrello (7)

PRODUCT NAME 1

SKU: [redacted]

Prezzo Unitario: 1.173,94 € (IVA esclusa)

Quantità: 2

~~4.013,48 €~~ (IVA esclusa)

2.347,88 € (IVA esclusa)

Garanzia

Estensione garanzia gold - 2 anni

SKU: [redacted]

Prezzo Unitario: 160,54 € (IVA esclusa)

Quantità: 2

321,08 € (IVA esclusa)

PRODUCT NAME 2

SKU: [redacted]

Prezzo Unitario: 1.173,94 € (IVA esclusa)

Quantità: 1

~~2.006,74 €~~ (IVA esclusa)

1.173,94 € (IVA esclusa)

Garanzia

ACCESSORY: WARRANTY TYPE

SKU: [redacted]

Prezzo Unitario: 60,20 € (IVA esclusa)

Quantità: 1

60,20 € (IVA esclusa)

Personalizzazioni

ACCESSORY: STCKER NAME

Testo per personalizzazione : test

SKU: [redacted]

Prezzo Unitario: 79,00 € (IVA esclusa)

Quantità: 1

79,00 € (IVA esclusa)

Svuota Carrello

Riepilogo Carrello

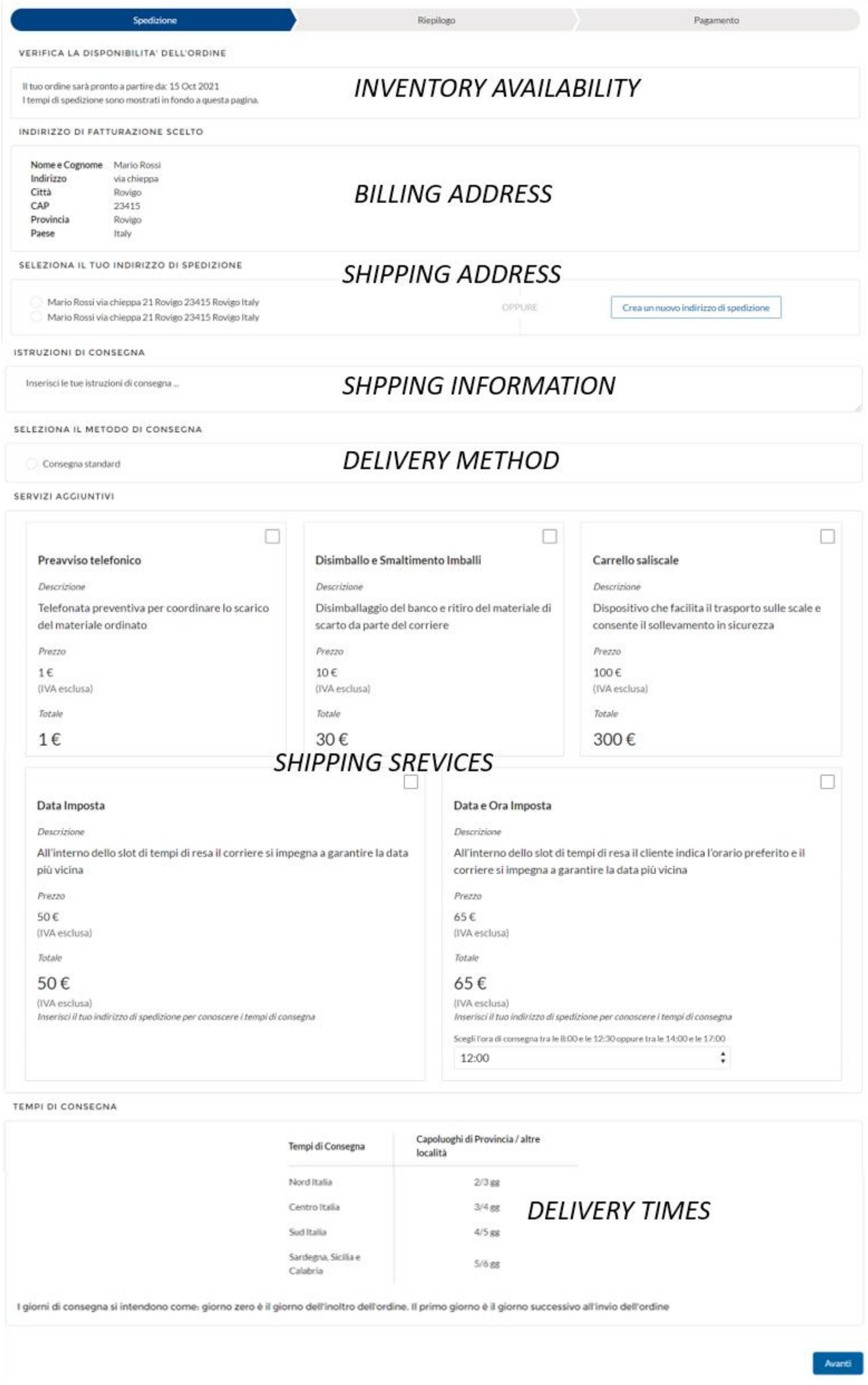
Totale (IVA esclusa) 3.982,10 €

Coupon

As you can notice the web cart shows two level of products: the parent refrigerator (first level) and its accessories (second level). The quantity of the accessories is directly calculated by the system from the chosen quantity of the parent object. The quantity field of the parent refrigerators is still editable in the cart summary page. The user can also decide to remove lines (either refrigerators or accessories), by clicking on the 'X' symbol on the right, or he can clear the whole web cart with the "Clear Cart" button. Notice that even if the parent refrigerator in the shown web cart is the same, they are not considered as a unique block, since the accessories configuration is not the same, therefore they will be treated as separated lines in the final order. The cart summary page offers the user also the possibility to add a coupon code. In this case the system will firstly check the availability of the coupon and apply the related discount to the web cart total price shown on the top right panel. Once the user is satisfied with his web cart he can proceed to the checkout by clicking on the "Checkout" button.

From the user point of view, the checkout process is divided in three main steps:

- Order data selection: Firstly, the system checks the inventory availability on the whole web cart and shows the user the delivery date. Then the system sets the billing address (if not available the system asks the user to create a new one), then the user picks a shipping address among all his active addresses (if no shipping address is available , the system asks the user to create a new shipping address). The possibility to insert a new shipping address is always available at checkout time (even if the user already has a certain number of addresses for the delivery), while the creation of a new billing address is possible only if the account doesn't have any billing address associated. Then the client can add some shipping information in a dedicated text area and select the desired delivery method. The possible delivery costs depends on the shipping address selected, therefore the price of the delivery method might change if the client picks a different shipping address. The information related to the cost of the delivery in relation to the shipping zip-code, is kept in a CVS file stored in the assets file of the platform. This allows the company to easily access the file and edit it, the system will be automatically updated and the clients will immediately see the new prices. Finally the user can add to his web cart one or more delivery services. This services depends on the client's account country, therefore, French account will see different services from Italian ones. Each service has a price and, in the case of "chosen delivery hour" service the user has to insert the desired delivery time too. The mandatory information is: billing address, shipping address and delivery method, therefore if the user clicks the next button without any of this data inserted the system will show the same form again asking to insert that. The bottom part of the form show a static table of the delivery times. Obviously, the schedule depends again on the user account country code, that is also the country of the billing and shipping addresses of the user. The described form for an Italian account looks as follows:



VERIFICA LA DISPONIBILITA' DELL'ORDINE

Il tuo ordine sarà pronto a partire da: 15 Oct 2021
I tempi di spedizione sono mostrati in fondo a questa pagina.

INDIRIZZO DI FATTURAZIONE SCELTO

Nome e Cognome	Mario Rossi
Indirizzo	via chieppa
Città	Rovigo
CAP	23415
Provincia	Rovigo
Paese	Italy

SELEZIONA IL TUO INDIRIZZO DI SPEDIZIONE

- Mario Rossi via chieppa 21 Rovigo 23415 Rovigo Italy
- Mario Rossi via chieppa 21 Rovigo 23415 Rovigo Italy

ISTRUZIONI DI CONSEGNA

Inserisci le tue istruzioni di consegna ...

SELEZIONA IL METODO DI CONSEGNA

Consegna standard

SERVIZI AGGIUNTIVI

Preavviso telefonico

Descrizione
Telefonata preventiva per coordinare lo scarico del materiale ordinato
Prezzo
1 €
(IVA esclusa)
Totale
1 €

Disimballo e Smaltimento Imballi

Descrizione
Disimballaggio del banco e ritiro del materiale di scarto da parte del corriere
Prezzo
10 €
(IVA esclusa)
Totale
30 €

Carrello saliscale

Descrizione
Dispositivo che facilita il trasporto sulle scale e consente il sollevamento in sicurezza
Prezzo
100 €
(IVA esclusa)
Totale
300 €

Data Imposta

Descrizione
All'interno dello slot di tempi di resa il corriere si impegna a garantire la data più vicina
Prezzo
50 €
(IVA esclusa)
Totale
50 €
(IVA esclusa)
Inserisci il tuo indirizzo di spedizione per conoscere i tempi di consegna

Data e Ora Imposta

Descrizione
All'interno dello slot di tempi di resa il cliente indica l'orario preferito e il corriere si impegna a garantire la data più vicina
Prezzo
65 €
(IVA esclusa)
Totale
65 €
(IVA esclusa)
Inserisci il tuo indirizzo di spedizione per conoscere i tempi di consegna
Scegli l'ora di consegna tra le 8:00 e le 12:30 oppure tra le 14:00 e le 17:00
12:00

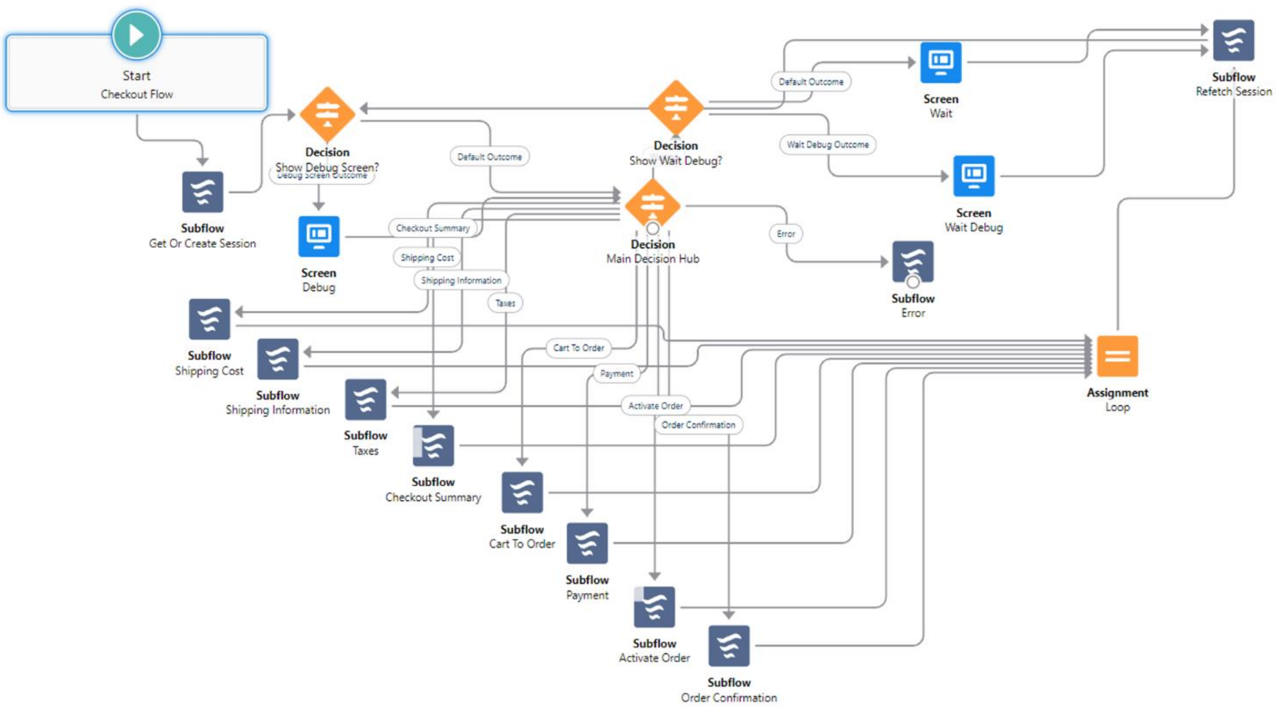
TEMPI DI CONSEGNA

Tempi di Consegna	Capoluoghi di Provincia / altre località
Nord Italia	2/3 gg
Centro Italia	3/4 gg
Sud Italia	4/5 gg
Sardegna, Sicilia e Calabria	5/6 gg

I giorni di consegna si intendono come: giorno zero è il giorno dell'inoltro dell'ordine. Il primo giorno è il giorno successivo all'invio dell'ordine

- Checkout summary: once the user clicks on the next button, the system will show a final screen summarizing the purchase information, which include: all the products in the web cart (including the delivery services chosen), the total prices (including the taxes) and the shipping information and addresses chosen. This way the user can verify his purchase before proceeding to the next and final section.
- Payment: for this part the system, firstly verifies the user associated risk code and according to its value shows different payment methods. Trustworthy client can pay both with credit card and purchase order, therefore both payment methods will be shown and the user can pick the preferred one. Risky clients will only be able to pay through a credit card. Once the user has inserted the payment information the system will elaborate the purchase and if everything proceeds without mistakes the client will be redirected to the Order Confirmation payment page. (Flow 1 Appendix)

From a back end point of view the checkout process has many more phases which represents the actual flows that were used to implement the process. As explained earlier Salesforce allows to create process using simple and intuitive diagrams, called flaws. Flows can be created through many different standard components (like object update, create or delete) and also some special components like the ones linked to an apex action (for example the flow that calculates the taxes uses an apex action component to call an apex class that produces the call to the external system) or other special components called screen flow, which can contain lightning web components or aura components and these are the ones through which the user can interact. If we see the checkout flow in terms of flows, we have a hierarchical structure in which, the parent flow is the Checkout_Flow, which looks as follows:

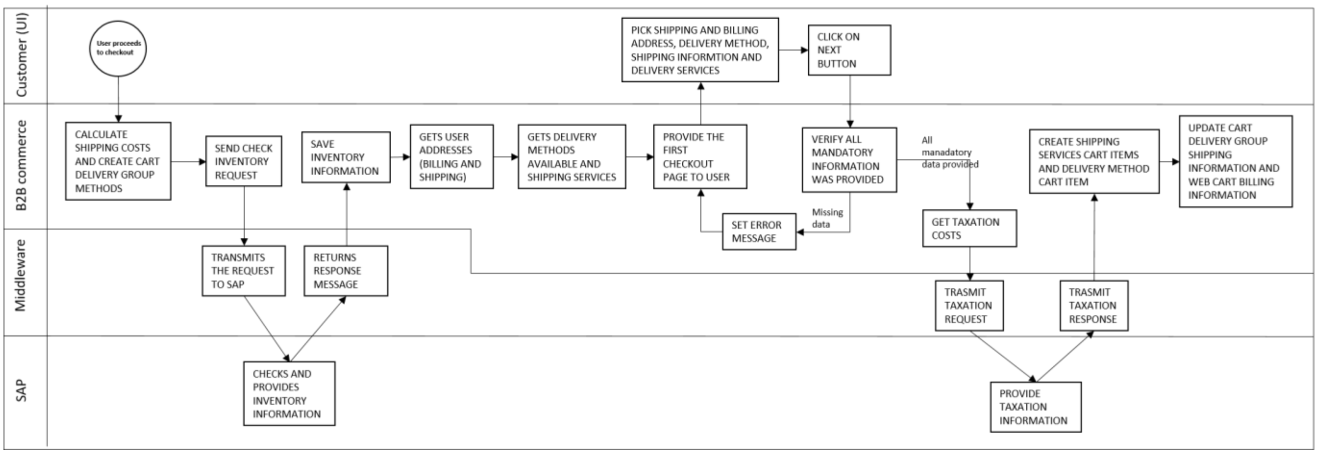


The start component of this flow is standard and configured to begin when the user clicks on the checkout button on the cart summary page. Then the process passes through the "Main Decision Hub" component, which is like a "switch" command in any programming language. The system compares the checkout status, which is kept in a variable throughout the whole process, with the possible values (the ones in the circles) and chooses the corresponding path to take. Each path leads to the execution of one sub flow. As you can see the process is characterized by many sub flows for each user phase. In particular we have:

- Phase 1: Shipping Cost, Shipping Information (which contains the screen flow that the user sees on the first phase and Taxes
- Phase 2: Checkout Summary and Cart To Order
- Phase 3: Payment, Activate Order and Order Confirmation

Now we will take a deeper look into each flow to better understand what actually was implemented to manage these process. In order to better understand also the relationships between the involved parts we use the following BPMN model.

Phase 1:


Customer (UI)
B2B commerce
Middleware
SAP
User proceeds to checkout
CALCULATE SHIPPING COSTS AND CREATE CART DELIVERY GROUP METHODS
SEND CHECK INVENTORY REQUEST
TRANSMITS THE REQUEST TO SAP
CHECKS AND PROVIDES INVENTORY INFORMATION
RETURNS RESPONSE MESSAGE
SAVE INVENTORY INFORMATION
GETS USER ADDRESSES (BILLING AND SHIPPING)
GETS DELIVERY METHODS AVAILABLE AND SHIPPING SERVICES
PROVIDE THE FIRST CHECKOUT PAGE TO USER
PICK SHIPPING AND BILLING ADDRESS, DELIVERY METHOD, SHIPPING INFORMTION AND DELIVERY SERVICES
CLICK ON NEXT BUTTON
VERIFY ALL MANDATORY INFORMATION WAS PROVIDED
Missing data
SET ERROR MESSAGE
All manadatory data provided
GET TAXATION COSTS
TRASMIT TAXATION REQUEST
PROVIDE TAXATION INFORMATION
TRASMIT TAXATION RESPONSE
CREATE SHIPPING SERVICES CART ITEMS AND DELIVERY METHOD CART ITEM
UPDATE CART DELIVERY GROUP SHIPPING INFORMATION AND WEB CART BILLING INFORMATION

The first operation performed by the B2B commerce is to perform the Shipping Cost sub flow, which retrieves from the database all the Order Delivery Method active instances for the country of the account performing the checkout, which represent all the possible delivery methods (in that country). Then the system creates one Cart Delivery Group Method instance for each order delivery method, which stores the information about the price of that service. Notice that this is the standard price, because the final price might change depending on the actual shipping address chosen on the next step by the user. An interesting part of this flow implementation is that the Cart Delivery Group Method object references both the Cart Delivery Group and the Order Delivery Method, and the combination of this two must be unique, that means we cannot have more Cart Delivery Group Methods associated with the same Order Delivery Method and Cart Delivery Group; therefore the flow firstly checks and, if necessary, deletes the previous instances, before inserting the new ones. Indeed it can happen that the user decides to interrupt the checkout before the end (on step 1 or 2) and so the process will start again from the beginning, but the Cart Delivery Group Methods were already inserted. In this way we won't have any issue when inserting the new instances in the database. The Check Inventory implementation code is at Code 6, the Shipping Cost Calculation at Code 7 and the Tax Implementation at Code 8 in the Appendix.

Then the process goes back to the main checkout flow, which will update the status and pass to the execution of the Shipping Cost flow. This is one of the more complex flow of the project, it executes a great deal of operations because it has to manage the front end screen and selections of the user and all the backend database interactions needed to support it. The first operation is an apex function that performs an HTTP request to an external system (SAP) to retrieve the check inventory information. In order to integrate Salesforce systems with SAP systems it was used a middleware called Boomi, which provides an integration platform as a service enabling the connection of applications and data sources. The HTTP request includes the list of products (only refrigerators) to be checked, some related information and the sales organization of the account. The response includes again all products with the related availability date and one global availability date for the whole order. The request fields and structure is the following:

Field name	Required	Type	Example	Description
salesOrg	True	String	BN10	4 letter of the sales organization
isCheckOutStage	True	Boolean	True	If True => Checkout If False => Product Detail Page
sku	True	Array		List of Refrigerator SKU
>sku	True	String		Refrigerator SKU to be checked
>quantity	True	Number		Quantity to be checked
>hasAccessory	True	Boolean		Specifies if Refrigerator has accessories. It will be filled only at Checkout stage.

The “isCheckOutStage” field is used to specify if the check inventory is related to the checkout stage or is performed on the product detail page or by the quick order form. For each “sku” the system provides the quantity of the refrigerators orders and a flag “hasAccessory” set to true if the refrigerator is bought together with some accessories. The request is transferred to SAP through by the middleware and a response comes back as follows:

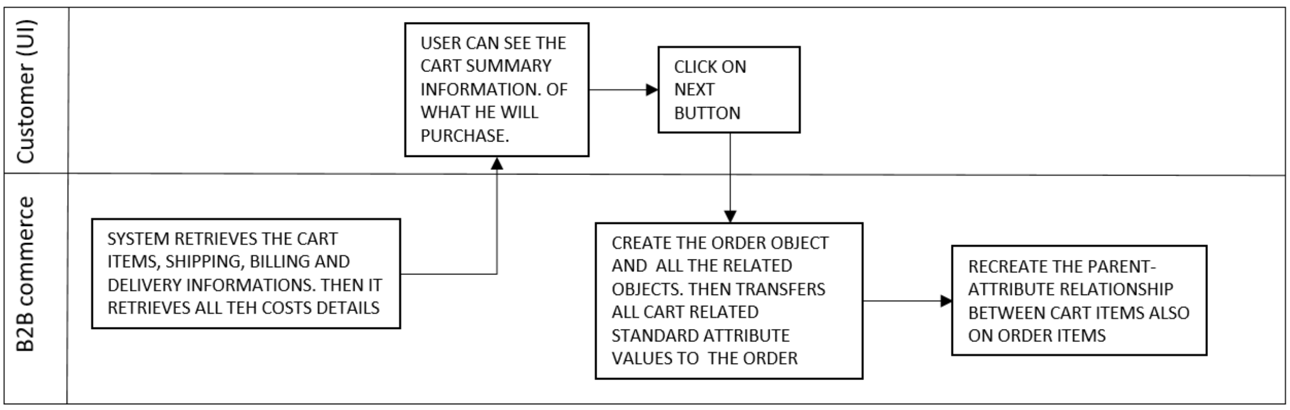
Field name	Type	Example	Description
sku	List	List	
>sku	String	12345	Refrigirator Sku to be checked
>available	String	True, False, Block	If Block, on PDP product will be shown as out of stock and then Add to cart is not possible.
>skuAvailableDate	String	01-06-2021	This is the available data show on PDP for the product selected by customer. It will be ignored on Checkout
OrderAvailableDate	String	01-06-2021	It will give the availability at order level for Checkout check inventory call. It will ignored in PDP

For each product sku the external system provides an “available” flag set to: “True” if the product is available, “False” if the product is not available but in production, “Block” if the product is not in production anymore. This information is used by the check inventory of a single product on the product detail page or by the check inventory of the products added with the quick order form.

For what concerns the shipping services and the delivery methods, they are all treated as product objects, with accessory type attribute “Shipping service” or “Delivery charge” and is accessory flag set to True. Once the user selects a delivery method and eventually some shipping services the system will create the related cart items. In particular, for the delivery method, the system creates a cart item of type “Charge”, while the shipping services are items of type “Product” like all other cart items. Once the user has clicked on the next button and the mandatory fields checks are passed, the system performs a new HTTP request toward the external system, SAP to retrieve the country taxation data. The request involves only the sales organization of the account and the chosen shipping address, while the response is the percentage of the taxation to be applied. Then the system calculates the tax amount for each cart item and creates the related Cart Tax instances. This will automatically update the Web Cart object attributes, which depend on the taxation applied to each cart item. Finally the shipping information and shipping address is saved on the cart delivery group instance and the billing address is stored on the web cart instance. For what concerns the objects involved in this first part, they are all the standard objects, so you can refer to the class diagram of the Web Cart and the related classes that was explained earlier. The only important custom attributed added was the “ParentProductCartItemId__c” on the Cart Item object. This attribute is used to take track of the relationship between parent objects and their accessories, because we will need it at the end of the checkout process for the order export to the external SAP system.

Phase2:

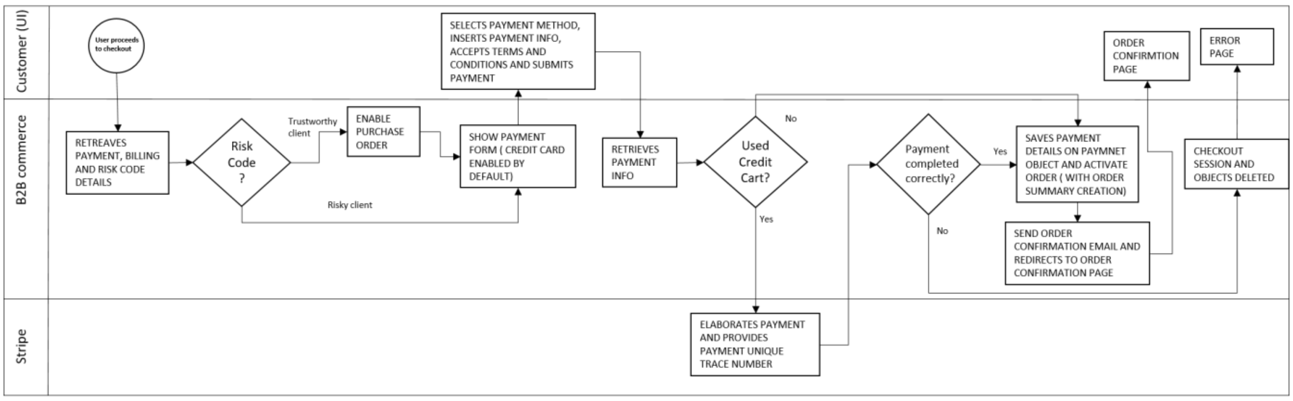
For this phase we do not have any type of interaction with external systems, but only with the user, through again a screen flow component.


Customer (UI)
B2B commerce
USER CAN SEE THE CART SUMMARY INFORMATION. OF WHAT HE WILL PURCHASE.
CLICK ON NEXT BUTTON
SYSTEM RETRIEVES THE CART ITEMS, SHIPPING, BILLING AND DELIVERY INFORMATIONS. THEN IT RETRIEVES ALL TEH COSTS DETAILS
CREATE THE ORDER OBJECT AND ALL THE RELATED OBJECTS. THEN TRANSFERS ALL CART RELATED STANDARD ATTRIBUTE VALUES TO THE ORDER
RECREATE THE PARENT-ATTRIBUTE RELATIONSHIP BETWEEN CART ITEMS ALSO ON ORDER ITEMS

The Checkout Summary flow will simply retrieve all the information about the cart items, the shipping and billing addresses and the delivery method details. Then it collects the total costs (which include cost of all products, cost of delivery method, coupon discounts (if present), total taxes amount and the total purchase cost. This way the user can take a final check to the information inserted before ordering. When the user clicks on the “next” button the system (user transparent) will create the new object Order with all its related ones. The Order is put in a “Pending” status and all the standard attributes for the web cart and related (or the custom ones that appear with the same name on the two objects) are copied on the new instances of the Order object and related ones. The standard operation was used here, so the copied fields are exactly the ones shown during the Salesforce explanation. Obviously for the Parent Product Cart Item attribute, which is a custom one and needs some behind logic, it was used a specific apex class to recreate the parent accessory relationship from the cart items to the order items. (this because as said before this relationship must be brought until the order export, therefore until the end of the flow).

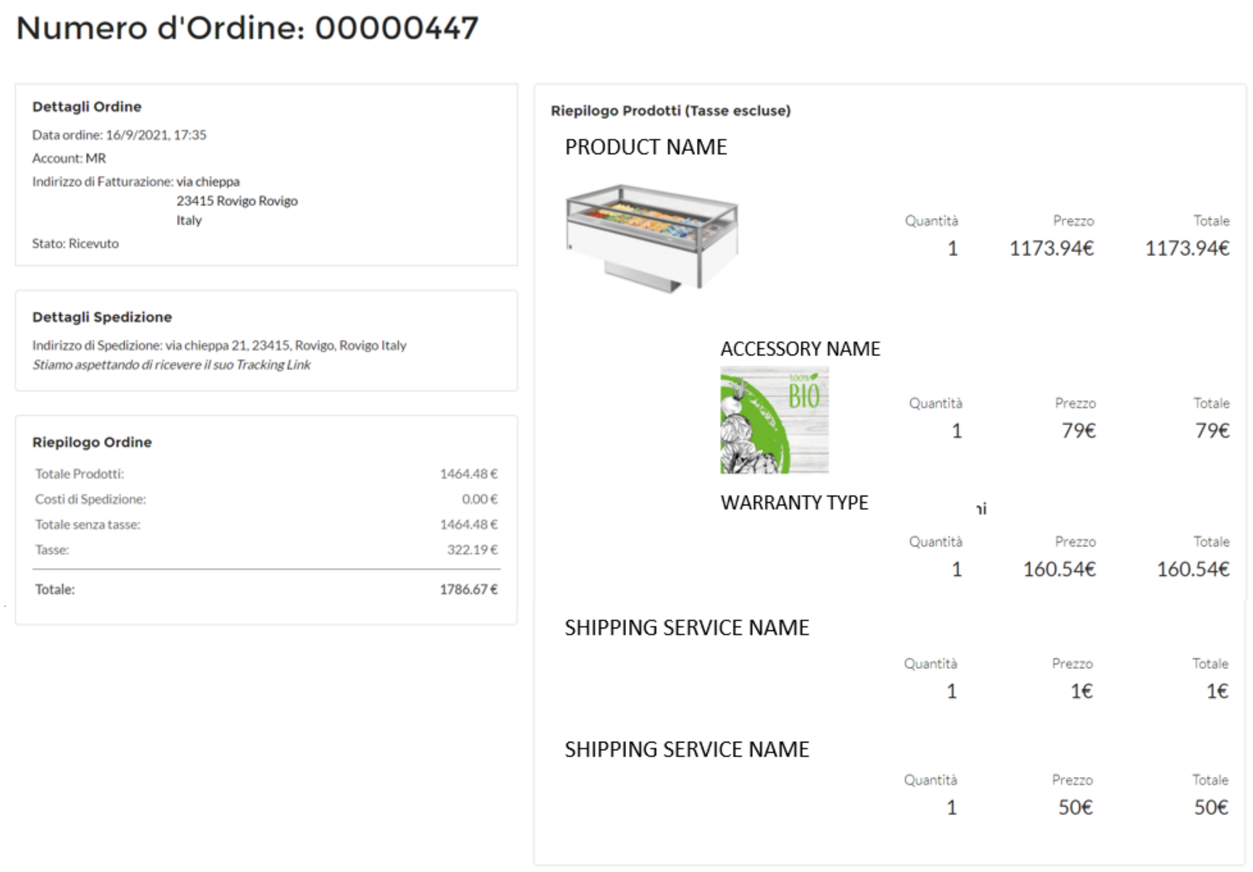
Phase 3:

For the final phase we firstly have the payment. The system can manage two types of payments: credit card payments or purchase orders. However not all the users can pay through purchase order, so the system firstly checks the RiskCode__c attribute on the Account object and only for trustworthy clients the system gives also the possibility of using the purchase order payment option. The Credit card option is instead available to any user. The credit card payment system is managed by an external service, which is widely used in e-commerce nowadays, called Stripe. While the purchase order (which is not an actual payment) is manage internally by the standard solution.


Customer (UI)
B2B commerce
Stripe
User proceeds to checkout
RETREAVES PAYMENT, BILLING AND RISK CODE DETAILS
Risk Code ?
Trustworthy client
Risky client
ENABLE PURCHASE ORDER
SHOW PAYMENT FORM (CREDIT CARD ENABLED BY DEFAULT)
SELECTS PAYMENT METHOD, INSERTS PAYMENT INFO, ACCEPTS TERMS AND CONDITIONS AND SUBMITS PAYMENT
RETRIEVES PAYMENT INFO
Used Credit Cart?
No
Yes
ELABORATES PAYMENT AND PROVIDES PAYMENT UNIQUE TRACE NUMBER
Payment completed correctly?
Yes
No
SAVES PAYMENT DETAILS ON PAYMNET OBJECT AND ACTIVATE ORDER (WITH ORDER SUMMARY CREATION)
SEND ORDER CONFIRMATION EMAIL AND REDIRECTS TO ORDER CONFIRMATION PAGE
ORDER CONFIRMTION PAGE
ERROR PAGE
CHECKOUT SESSION AND OBJECTS DELETED

Once the system has enabled or not the PO (purchase order) form, it presents the final payment form to the user. The client picks the payment method and provides the details needed, then he must accepts the terms and conditions in order to enable the submit payment button. If the payment was through credit card the system connects to the external Stripe service that elaborates the credit card information given and produces a trace number for the payment. Otherwise, if the payment was a PO, there are no controls to make. Then if the payment has been completed correctly the system saves the related information in the payment object and the related ones. Then the system performs the Order Activation flow, which changes the Order status to Activated and creates the Order Summary object and its related ones. This is again a standard process, exactly like from the Web Cart object to the Order one. Again the relation between Order Items Summary is recreated from the one among the Order Items, with a total custom code. Once the order is active and the Order Summary object is completed the system executes the Order Confirmation flow, which completes the checkout session and sends an Order Confirmation email to the user (Flow 4 Appendix), which shows all the data of the order purchased. Then the client is redirected to the Order Confirmation page (which is outside the checkout flow). In case of errors in the payment the system deletes the checkout session and all the related information and objects created. In general the error management is really simple, in case of failures the system clears all the information and the checkout restarts from the beginning. It might seems a drastic solution, but form the user point of view, the checkout is completed in three simple steps, so it is quite quick, therefore it is not a big deal to perform again the steps. On the other hand this solution allows to avoid many errors that could be created when trying to restore from a system failure. The same solution has been adopted if the user interrupts the payment and logs out the system. This means if the client doesn’t complete the checkout steps all at ones the system will clear the checkout session and start from the beginning again. In this case the problem was mostly related to the product availability problem. Indeed if the user decides to interrupt the checkout in the middle and logs out the product availability of his cart could change. So if we allow the user to reload the same checkout started earlier, eventually the products are not available anymore (or at least the delivery date has changed) and the order will not reflects the user expectations. For this reason an incomplete checkout is deleted and restarted, so that the check inventory operation (performed on the first phase) is always executed right before the payment and the order submission.

The Order Summary page offers the client the possibility to access the order details, the UI looks as follows:


Numero d'Ordine: 00000447

Dettagli Ordine
Data ordine: 16/9/2021, 17:35
Account: MR
Indirizzo di Fatturazione: via chieppa
23415 Rovigo Rovigo
Italy
Stato: Ricevuto

Dettagli Spedizione
Indirizzo di Spedizione: via chieppa 21, 23415, Rovigo, Rovigo Italy
Stiamo aspettando di ricevere il suo Tracking Link

Riepilogo Ordine

Totale Prodotti:	1464.48 €
Costi di Spedizione:	0.00 €
Totale senza tasse:	1464.48 €
Tasse:	322.19 €
Totale:	1786.67 €

Riepilogo Prodotti (Tasse escluse)

	Quantità	Prezzo	Totale
PRODUCT NAME	1	1173.94€	1173.94€
ACCESSORY NAME	1	79€	79€
WARRANTY TYPE	1	160.54€	160.54€
SHIPPING SERVICE NAME	1	1€	1€
SHIPPING SERVICE NAME	1	50€	50€

The most interesting part of this page is related to the order tracking link, placed under the shipping details section box on the left. In particular when a new order is created the order is exported to an external system (we will see later the process). Then the external system manages the logistic of the shipping, which in case of Italian orders is managed by an external delivery service that provides a tracking link for his customers to retrieve the information about the delivery (like position of the refrigerator and courier contacts). Therefore the tracking link is actually inserted by the middleware directly in Salesforce database on the related Order Summary object when it is ready. Once the "orderTrackingURL__c" field is updated on the Order Summary instance the systems automatically triggers a process that sends an email to the user that his tracking URL for a specific order is now available on the order detail page (Flow 5 Appendix). In order to perform this operation Salesforce offers two possible solutions. The first one is to create a trigger on the Order Summary object, which is an apex class executed each time a specific action is taken on the chosen object. In this case the action would be after the update of the "orderTrackingURL__c". The second possibility, that is also the one used in the project, is to exploit the process builder to create a process that executes a flow which sends the email. Processes can be triggered in different ways, one way is to update an object, like in this case. There was probably no best choice in this case, because both solution were equally valid, however in general the using process builder allows you much more flexibility since you can attach a flow from which you can basically do anything, while a trigger is just an apex class, so its functionalities are wide, but more restricted. Another thing that might make flows and process preferable to apex classes is that apex classes require at least a 80% test coverage overall. This means that every time you write an apex class you have to keep in mind to write also the Junit test class for it, this is why if you can perform the same operation using the standard components of a flow it is usually the chosen solution.

The last important part of the checkout phase is the Order Export. This operation is actually performed in the Order Export flow in a totally asynchronous way. Indeed the system completes the checkout as shown before and at the same time it runs the operations to create the JSON file to export to the middleware system, which will be finally be exported to SAP systems. The information exported is basically all the details that the company needs in order to manage the client order, delivery and billing. The JSON format of the exported details is the following:

Field name	Required	Type	Description
orderDate	True	String	Timestamp of the order date in UTC format
accountNumber	True	String	The customer number for identification
IsNewCustomerFirstOrder	True	boolean	Defines if this order is the very first order of a self-registered Customer
accountGroup	True	String	It will always be: Z019
deliveryMethod	True	String	The delivery mode for the goods. Both For Italy and France it will be 01
orderNumber	True	String	Salesforce order Number
shippingNetCosts	True	Number	Shipping costs without VAT
shippingNotes	False	String	Shipping notes by the customer
couponCode	False	String	It is the coupon code used by the customer. Only one coupon may be used for each purchase.
couponDiscount	False	Number	It is the amount value to be discounted by the total cart value
paymentId	True	String	Transaction number (if payment is prepaid) or purchase order number (if payment type == Invoice)
isPrepayment	True	boolean	Payment type: if true it is Prepaid (in case of card payments), if false it is PO (in case of Purchase Order)
orderTotalAmount	True	Number	Order Final Amount paid by customer with tax and including the discount(coupon) and shipping
countryCurrency	True	String	Currency
shipping	True	Object	Contains shipping information about customer name, address and contact information
>shipToCode	True	String	It is the code which identifies the shipment customer. Range starting from 30000000

> name	True	String	Name for shipping recipient. This field will be splitted by Boomi in 2 line if the length is too large.
> address	True	String	Address for shipping recipient.
>houseNumber	True	String	House Number
> zip	True	String	Zip Code for shipping recipient
> city	True	String	City for shipping recipient
> province	True	String	Province for shipping recipient. This field will be transcoded by Boomi (only for France by removing the first two letter es from FR001 -> 001)
> country	True	String	Country name
> phone	True	String	Phone number for shipping recipient
billing	True	Object	Contains billing information about customer name, address and contact informatio2
>vat	True	String	Vat Code
>salesOrganization	True	String	Sales organization of the customer Account
>distributionChannel	True	String	Distribution channel of the customer Account
>division	True	String	Fixed value
>pec	False	String	Optional, only in case of self-registered account for Italian Customer
>sdi	False	String	Optional, only in case of self-registered account for Italian Customer
>cassettoFiscale	False	boolean	True only if neither pec nor sdi are present and specifies that customer doesn't have neither pec nor sdi.
>fiscalCode	False	String	Optional, only in case of self-registered account for Italian Customer
>siren	False	String	Optional, only in case of self-registered account for French Customer
>siret	False	String	Optional, only in case of self-registered account for French Customer
> name	True	String	Name for billing. It will be split in 2 by Boomi
> address	True	String	Address for billing.
>houseNumber	True	String	House Number
> zip	True	String	Zip Code for billing
> city	True	String	City for billing
> province	True	String	Province for shipping recipient. This field will be transcoded by Boomi (only for France by removing the first two letter es from FR001 -> 001)
> country	True	String	Country code
> phone	True	String	Phone number for billing. It is the contact phone number Salesforce side.
> email	True	String	Email for billing It is the contact email address Salesforce side.
orderLines	True	Array	Contains objects with information about the individual items on the order such as item id, quantity and sales price.

> lineId	True	String	Salesforce Order line ID
>productCode	True		It is the product Code (SKU). May be a Refrigerator or Accessory (only Sticker, Eco participation and Warranty)
>uom	True		It is the Unit of measure of the product
>productName	True		It is the product Name. For example Glee Glass if it is a refrigerator or Kit Stainless if it is an Accessory
>parentProductLineItem	False	String	It is the reference to the parent Product (Refrigirator) for Accessory items
>isAccessory	True	Boolean	Defines if the product is an Accessory or not.
>accessoryType	False	String	May be: Sticker, Bin,Basket, Shelves, Other, Warranty,Ecopartecipation,Shipping Services, Deliver. It is an optional field, since it will not be filled in case the productCode is not an accessory
>shopName	False	String	It will be filled with the ShopName in case accessoryType is Sticker. It is an optional field.
> netLineAmount	True	Number	In case Product Code = Refrigirator it will be the Net line amount with adjustments (discounts on volume, TBD IF to consider also the distributed Coupon Discount) without taxes which compounds the price of accessories related to the Refrigirator (except for Warranty and Ecopartecipation Price) Otherwise, in case Product Code = Warranty, this netLineAmount will the Net line amount without taxes and adjustments of the warranty Otherwise, in case Product Code = Ecopartecipation, this netLineAmount will the Net line amount without taxes and adjustments of the ecopartecipation Otherwise, in case Product Code = Sticker, this netLineAmount will the Net line amount without taxes and adjustments of the sticker In all the other cases (e.g. Accessory like Bin,Basket, Shelves, Other), this field will not be filled.
> quantity	True	Integer	Order line quantity
>hourOfShipping	False	String	It's the hour selected by the customer in case he chooses the Additional Shipping Service:"Data e Ora Imposta"

As you can see the files structure is divided in the main information related to the order in general(like account number, order date, all payment information and some more) and then there are three subsections:

- Shipping: stores the shipping address and other related details
- Billing: stores the billing details and other related data
- Delivery Items: is the list of products ordered by the user. Each product can be an accessory or not. All accessories must have the line number of the parent product under the ParentProductLineItem field. Each product then has all the important details, like quantity,

price, product code. The shopName field is used to save the customized name to be added to a sticker accessory, if it was inserted by the client.

The request is sent as usual with a HTTP request and it expects a response in maximum 60 seconds. If the system doesn't receive one in that time it sends again the request for maximum of other two times. If also the third tentative fails the Order Summary status remains 'Sent', otherwise the status is updated in 'Exported'. Notice that it can happen that the order is received fine by the middleware, but the answer comes in more time than 60 seconds, so the order results as failed in Salesforce, but the order was actually exported fine. This is not a problem, since when the middleware receives correctly an Order it updates the corresponding Order Summary object status on Salesforce system with 'Received'. The client can keep track of the Order status changes on the order summary page, under the 'Order Details' section box on the left. This concludes the more interesting parts of the Cart and Checkout management. (Flow 3 Appendix)

6. GENERAL PROJECT DETAILS

In this final section we will discuss just some general details of the project, that we haven't yet seen.

The first interesting characteristic is the management of the languages and labels. The website must, as obvious, be able to adapt to the browser language in order to offer the best user experience. Salesforce, by default, offers the possibility to create a what so called Multilingual organization, which, if enabled, it creates several Language object instances, one for each enabled language. This will also create many "versions" of the experience builder, so that when you insert the labels directly from the experience builder you can put the labels in the different languages. In addition, Salesforce offers for standard objects and attributes default translations, which can be customized. For what concerns the labels introduced by apex classes or directly on Lightning components, the Label global variable is used. In particular, you can create any type of label (with a unique name) and associate it a translation for any language active in the organization. These labels will automatically assume the correct translation according to the browser language. If the browser is in a language not active in your organization, the website will assume the default language.

Another interesting feature is the country and state picklist setting. This feature allows to customize the possible Country and related States values that can be inserted in the Address attribute present on many Salesforce Objects (like the Account or the Contact Point Address). Basically, the Address object accepts by default any value as Country and State, but you can configure State And Country Picklist setting so that the country and state fields of the Address become picklists. This is really important in order to make sure that the values inserted are correct and to avoid having the same country name or state name written in many different ways. The problem of this feature is that Salesforce offers many countries but only for some of them it also offers the related States. This means that you should have to manually configure them one by one. Since this operation would be extremely inefficient, we actually built a code that was able to perform the same HTTP request that the manual configuration would do to insert the custom states, in this way we were able to insert in a bulk way all the states for each country of interest at once. Since we activated France and Italy countries and the relative states (Italian provinces and French regions), in the Contact Point Address form we were able to dynamically show the user the French or the Italian picklist for the state field (remember that the country was a read only field on the form, because was directly read from the Account instance).

A final interesting feature of Salesforce is the Content Management System (CMS), that is the part of the platform used to store images, videos and files for the website. The CMS system can be structured in many folders and libraries. The File Asset library is the main one used to access images also from outside the organization (if the content is set to be accessible from outside). The problem with the current Salesforce Asset library is that it doesn't allow a bulk immigration of content, that means all images must be uploaded one by one on the Asset library of the organization. The good news, is that once you have uploaded them on one org you can easily transfer them on another one (this for example was not possible with the state and country configuration). In general, we used the CMS system to store all product images, some CVS files (used for example to store the shipping service SKUs or the delivery cost based on the postal code) and other images used in the website (like the logo, the footer and header in the emails, the home page banner and other banners around the website). Nevertheless, we encountered a problem on the product images, because the images given by the company were all links but the ones related to the sticker accessories; for which we had their png versions. This wouldn't be a problem if the images were just used on the website, because all type of CMS images (links or png) can be accessed from the website pages. However it became a problem in the order summary email (which contains also the product images bought by the client), because the client that receives the email is not allowed to see the image, since they have a link inside the Salesforce organization and the png images uploaded automatically earn a link inside the Salesforce organization. To solve this problem, thanks to the fact that the images were just the ones related to stickers, so not too many, we decided to manually upload the product images for sticker accessories on the Asset Library. In this way the images were available also outside the organization, therefore also in the emails sent. Obviously, this solution implied a manual insertion of all those images on the file asset library. For all other products, the images were imported using a custom code, since they were just links, so there was no problem with them. Unfortunately, the CMS system of Salesforce is not very sophisticated, for now, so for projects that are more complex it is recommended to use an external Digital Asset Management (DAM) system, like Cloudinary or Adobe, which are specifically designed to maintain your digital content and expose an external link to them that can be used on your Salesforce platform.

7. CONCLUSION

In nowadays world, it has become impossible to ignore the exponential growth of technology. Companies that do not keep up the pace with the new trends and the consumer desires are destined to fail. Anybody, from the youngest user to the oldest companies, are inevitably changing their way of approaching the buying experience. For sure, the change has had a different speed race in the different fields, but it is indeed changing and if you do not evolve with it you won’t be able to stay in the market. Competition on digital marketing is getting stronger and consumers expect more and more from their experience; companies who cannot give that won’t last long. We need to “lean into the future” and embrace the evolution, by giving the clients always the best services, the top advertising and the most targeted marketing that will hook them to your products and services. It is clear that the quality of what you sell is always fundamental, but the presentation and the accessibility of your products is gaining a great deal of importance too. People expect to be able to purchase quickly and efficiently from their home, without the need of interactions or phone calls, but just using few clicks.

The evolution of the digital commerce in the B2B world has undoubtedly been slower that the one in B2C commerce, probably due to the age of the target consumers, but it is changing and companies need to acknowledge it to survive. B2B e-commerce is becoming the normality for many companies and clients expect that, so it not an optional anymore, it is a commodity. Not having a well working and efficient e-commerce simply means cutting out half of your possible clients and the percentage is increasing. B2B must be approached as a totally different and unique problem respect to B2C. We are talking about different type of clients, different types of purchase quantities and prices, different types of expectations and services required. The problem is just very unlike B2C and cannot be considered as just a branch of it. This was well understood by software house who started building specific platforms to approach the problem. Salesforce B2B Commerce is an example of a software build specifically to create a B2B ecommerce and integrate it with the classic CRM, Sales and OMS systems. Indeed the commerce is the very heart of all other systems, is the starting point of everything else and it must integrate with all other services. The client will interact with the CRM system through the commerce website and the CRM system will improve and offer the best services thanks to the great deal of data that it retrieves from the commerce platform. Then the client, hopefully, will purchase your products on the commerce platform and the order will then be exported to the OMS systems, but again the starting point is the commerce. The Sales team will too interact with the e-commerce, to manage prices, offers and discounts. Everything is at the end wrapped around the commerce platform. Before the e-commerce all these services where separated and sometimes didn’t even share information and details about clients and offers. But now they have the possibility to work together, to interact and share information, all thanks to the e-commerce. Salesforce B2B Commerce allows this full circled user experience.

Salesforce B2B Lightning Experience is the newest version of the B2B e-commerce platform, it is revolutionary thanks to its ability to work without the need of using any type of custom code, but at the same time it offers the possibility to create any type of custom functionality and integrate it in the system. It is efficient and intuitive and it offers the developer all the basic codes and objects that any B2B commerce website would need, but then it allows the company to create its totally unique experience.

Last but not least, we studied a real problem; a company that creates its new commerce website. We have decomposed the problem and we have seen how to approach each part with standard and, sometimes, custom configurations and code on Salesforce.

The project has been for me extremely interesting and challenging. We had an initial phase of discovery, that implied talking to the client and understand their needs, but mostly their brand values and messages that they wanted to give to their customers. It is, indeed, very important to deeply understand the product of the company in order to build and experience that is completed wrapped around it. The clients must be able to access the product, seeing it, gather information, be involved in the product making and in the company way of working to truly be hooked to that supplier. The second analysis that was made, was related to the target consumers. We had to understand which was our market segment, who are the clients, what they expect, what’s their age, what’s their purchase capability, how much they are willing to spend and how much they want to know about the products. The problem with B2B is that you could sell the same exact product (in different quantities) to very different type of clients. We had to sell professional refrigerators to the small ice cream shop, that makes one purchase every two years and the same refrigerator, but in a much higher quantity, to the big supermarket chain, which makes orders of thousands of euros many times a year. We had to build specific prices, discounts, marketing advertising and experiences, unique for each type of client. This is the key point that makes the difference between a fine commerce experience and a great one. If you want to make the difference of the market, you need to understand the client segments and build your website according to it. On this first part of the project the objective was to understand what the client wanted. It might seem to be quite simple and straight forward, they ask for something and you give them that, but it is definitely not. Companies are composed of many people and each one has ideas and expectations and usually they don’t talk much across teams. So what happened was that it took us around two weeks to draft the first version of the users stories. Obviously we adopted a totally agile methodology, so the client could change a little bit the user stories during the project. Once we had a basic understanding of what the client wanted, we started the technical analysis and we build the BPMN high level models for the checkout flow, the activity flow diagrams for other processes and obviously we build the data model. Then we also used some designers to create the UI with the client and then reproduce them on the website pages. The developments started after few weeks from the beginning of the analysis. Since the project wasn’t very big, we didn’t have much time to build a very solid and deep analysis of the project before starting the developments, this obviously led to some problems along the way, because the client changed mind on many things and we had to recode many features. Probably a deeper initial analysis, would have helped reducing the time spent in rewriting code, but we also had to deal with the problem of fitting the whole project in just 4 months of work. On the other hand this project really gave me the possibility to get in contact with what it really means working with customers and be able to understand their needs and transform it in code. Sometimes it is really important to be able to advise the client too, because we have seen many of these projects and might have more experience, so it is always useful to reach a great result to advice and discuss with the client. They set the specifics and the high level descriptions of what they wanted and we gave them the solutions to realize them. We had to make changes along the way, since the platform was new for us too, so we had learnt how to properly use it along the way. Moreover, we didn’t have much reference to consult, since this has been literally one of the very first projects in Italy and in Europe that has used this technology. We had to write a lot of custom code to realize a unique experience. A very interesting part of this project is also the fact that you always have to keep in mind that there are two clients. The company that wants to build the commerce platform and will then use the back office to access customers data and orders information and the final clients, who will use the website to make their purchases. So it is important to create what the company wants, but is also important to keep in mind who are the final clients and discuss which are the best solutions to win them. Moreover, you need to make sure the website you build is simple, intuitive and efficient, but also the back office must be organized and eventually you might need to add custom code and configuration to make sure the company can easily access it and perform the needed operations. For example, the company wanted to be able to create a discounted pricelists to make special offers to some clients or in some periods of the year. To create a new pricelist on Salesforce you can simply use the standard creation button, but then you must remember to create all the related price book entries for each product and the buyer group to associate the price list with. This solution is fine for a developer or somebody who knows the platform and its configurations, but it is not adapted to a customer. So we implemented a custom button directly on the price book list page on backend, that allows the user to create a new price list based on a previous one adding a percentage discount and some possible configurations. In this way the company user just needs to click on a button and then will be guided trough the whole configuration without the need of knowing anything about the platform. Basically there are two goals: building a unique website experience for the final customer and giving the company the possibility to make configurations without knowing anything about coding and very little (explained with some power point presentations to the client) about the platform.

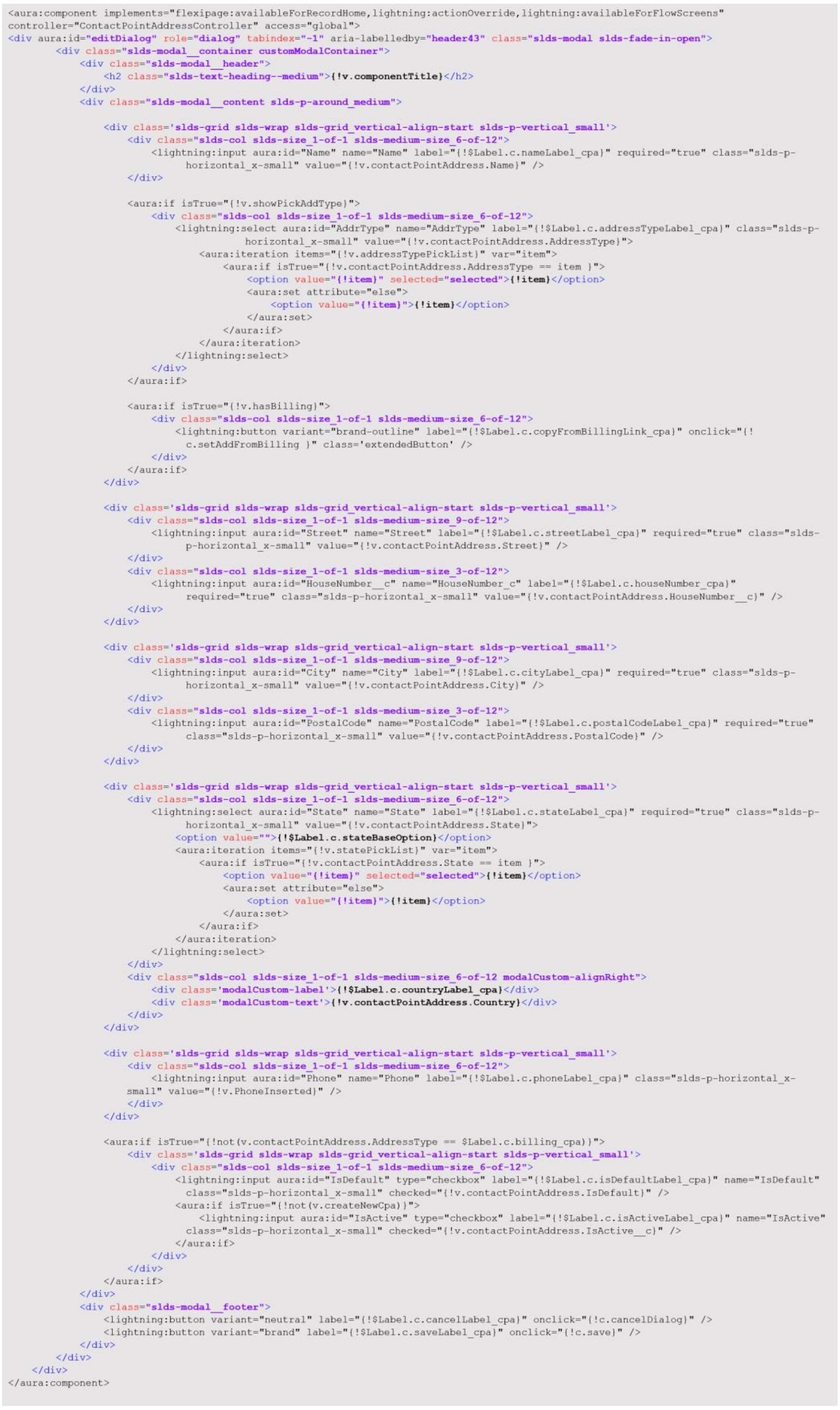
To conclude, the main purpose of this thesis was to, firstly, show the importance, in nowadays world, of being supported by a structured and efficient B2B e-commerce websites. Then to present a new technology that was developed to help reaching that goal by avoiding the developer the burden of tons of configurations and lines of code that should be just a commodity and allowing hit to concentrate on the custom features, which are the heart and soul of a project. They are what makes the experience unique.

The ultimate goal of an e-commerce website is to be able to intercept the customer needs, present the perfect solution, allow an easy and straight forward purchase process and, last but not least, hook the client to your unique products and unforgettable experience.

8. APPENDIX

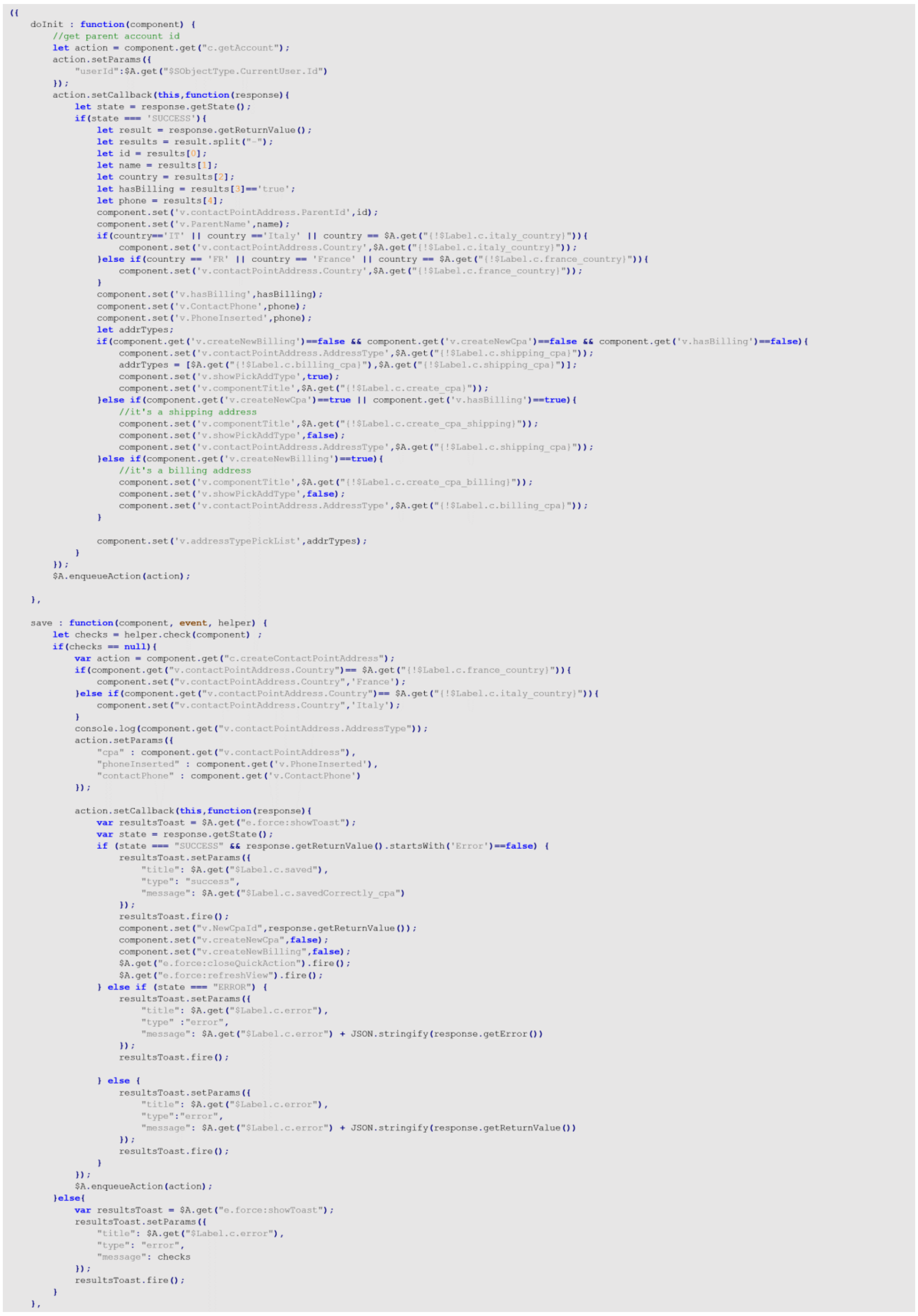
Code 1: Contact Point Address

FRONT END (Aura Component, HTML)



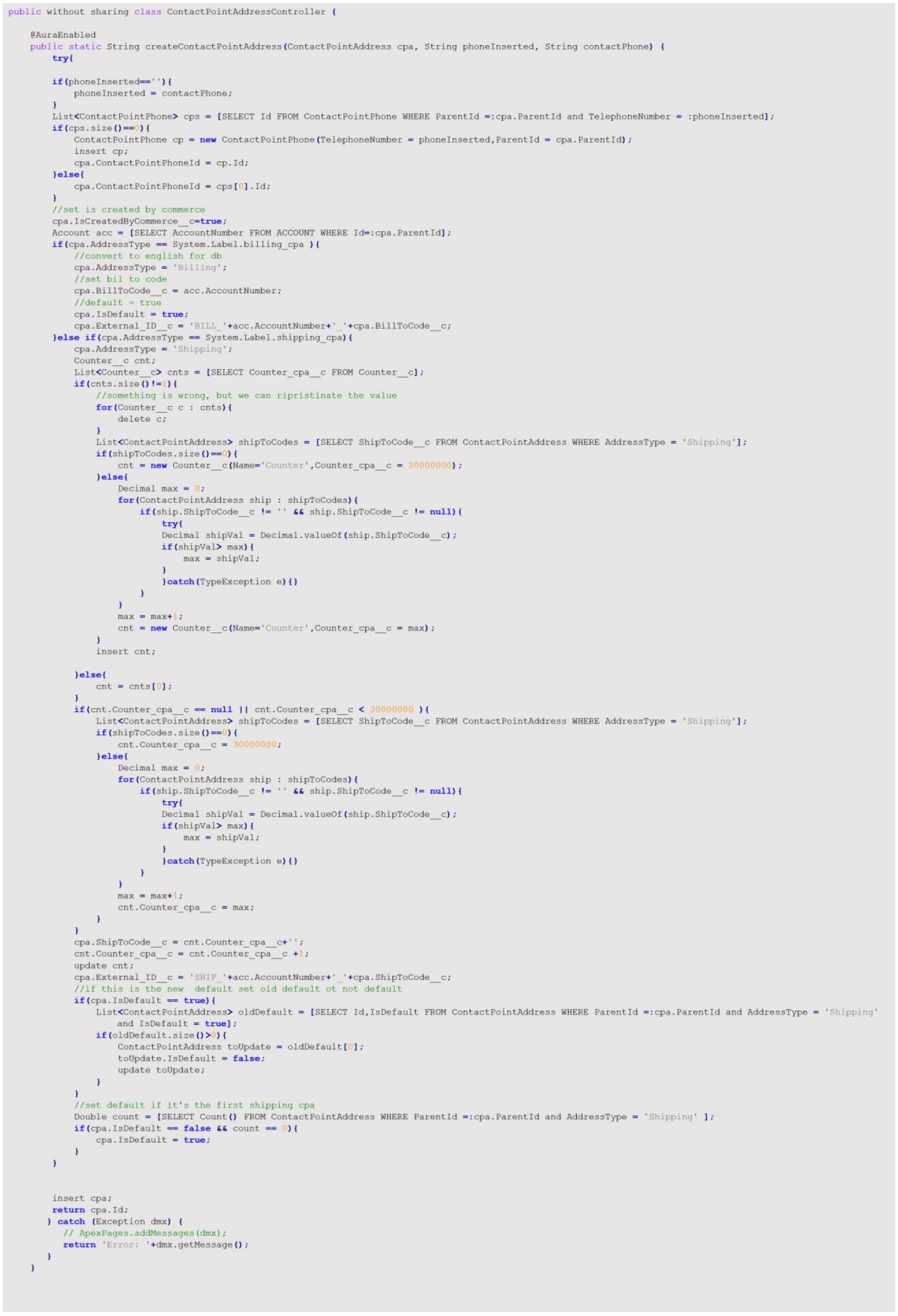
```
<aura:component implements="flexipage:availableForRecordHome,lightning:actionOverride,lightning:availableForFlowScreens"
controller="ContactPointAddressController" access="global">
<div aura:id="editDialog" role="dialog" tabindex="-1" aria-labelledby="header43" class="slds-modal slds-fade-in-open">
        <div class="slds-modal__container customModalContainer">
            <div class="slds-modal__header">
                <h2 class="slds-text-heading--medium">{!v.componentTitle}</h2>
            </div>
            <div class="slds-modal__content slds-p-around_medium">

                <div class='slds-grid slds-wrap slds-grid_vertical-align-start slds-p-vertical_small'>
                    <div class="slds-col slds-size_1-of-1 slds-medium-size_6-of-12">
                        <lightning:input aura:id="Name" name="Name" label="{!$Label.c.nameLabel_cpa}" required="true" class="slds-p-
                              horizontal_x-small" value="{!v.contactPointAddress.Name}" />
                    </div>

                    <aura:if isTrue="{!v.showPickAddType}">
                        <div class="slds-col slds-size_1-of-1 slds-medium-size_6-of-12">
                            <lightning:select aura:id="AddrType" name="AddrType" label="{!$Label.c.addressTypeLabel_cpa}" class="slds-p-
                                        horizontal_x-small" value="{!v.contactPointAddress.AddressType}">
                                <aura:iteration items="{!v.addressTypePickList}" var="item">
                                    <aura:if isTrue="{!v.contactPointAddress.AddressType == item }">
                                        <option value="{!item}" selected="selected">{!item}</option>
                                        <aura:set attribute="else">
                                            <option value="{!item}">{!item}</option>
                                        </aura:set>
                                    </aura:if>
                                </aura:iteration>
                            </lightning:select>
                        </div>
                    </aura:if>

                    <aura:if isTrue="{!v.hasBilling}">
                        <div class="slds-col slds-size_1-of-1 slds-medium-size_6-of-12">
                            <lightning:button variant="brand-outline" label="{!$Label.c.copyFromBillingLink_cpa}" onclick="{!
                              c.setAddFromBilling }" class='extendedButton' />
                        </div>
                    </aura:if>
                </div>

                <div class='slds-grid slds-wrap slds-grid_vertical-align-start slds-p-vertical_small'>
                    <div class="slds-col slds-size_1-of-1 slds-medium-size_9-of-12">
                        <lightning:input aura:id="Street" name="Street" label="{!$Label.c.streetLabel_cpa}" required="true" class="slds-
                              p-horizontal_x-small" value="{!v.contactPointAddress.Street}" />
                    </div>
                    <div class="slds-col slds-size_1-of-1 slds-medium-size_3-of-12">
                        <lightning:input aura:id="HouseNumber__c" name="HouseNumber_c" label="{!$Label.c.houseNumber_cpa}"
                              required="true" class="slds-p-horizontal_x-small" value="{!v.contactPointAddress.HouseNumber__c}" />
                    </div>
                </div>

                <div class='slds-grid slds-wrap slds-grid_vertical-align-start slds-p-vertical_small'>
                    <div class="slds-col slds-size_1-of-1 slds-medium-size_9-of-12">
                        <lightning:input aura:id="City" name="City" label="{!$Label.c.cityLabel_cpa}" required="true" class="slds-p-
                              horizontal_x-small" value="{!v.contactPointAddress.City}" />
                    </div>
                    <div class="slds-col slds-size_1-of-1 slds-medium-size_3-of-12">
                        <lightning:input aura:id="PostalCode" name="PostalCode" label="{!$Label.c.postalCodeLabel_cpa}" required="true"
                              class="slds-p-horizontal_x-small" value="{!v.contactPointAddress.PostalCode}" />
                    </div>
                </div>

                <div class='slds-grid slds-wrap slds-grid_vertical-align-start slds-p-vertical_small'>
                    <div class="slds-col slds-size_1-of-1 slds-medium-size_6-of-12">
                        <lightning:select aura:id="State" name="State" label="{!$Label.c.stateLabel_cpa}" required="true" class="slds-p-
                              horizontal_x-small" value="{!v.contactPointAddress.State}">
                            <option value="">{!$Label.c.stateBaseOption}</option>
                            <aura:iteration items="{!v.statePickList}" var="item">
                                <aura:if isTrue="{!v.contactPointAddress.State == item }">
                                    <option value="{!item}" selected="selected">{!item}</option>
                                    <aura:set attribute="else">
                                        <option value="{!item}">{!item}</option>
                                    </aura:set>
                                </aura:if>
                            </aura:iteration>
                        </lightning:select>
                    </div>
                    <div class="slds-col slds-size_1-of-1 slds-medium-size_6-of-12 modalCustom-alignRight">
                        <div class='modalCustom-label'>{!$Label.c.countryLabel_cpa}</div>
                        <div class='modalCustom-text'>{!v.contactPointAddress.Country}</div>
                    </div>
                </div>

                <div class='slds-grid slds-wrap slds-grid_vertical-align-start slds-p-vertical_small'>
                    <div class="slds-col slds-size_1-of-1 slds-medium-size_6-of-12">
                        <lightning:input aura:id="Phone" name="Phone" label="{!$Label.c.phoneLabel_cpa}" class="slds-p-horizontal_x-
                    small" value="{!v.PhoneInserted}" />
                    </div>
                </div>

                <aura:if isTrue="{!not(v.contactPointAddress.AddressType == $Label.c.billing_cpa)}">
                    <div class='slds-grid slds-wrap slds-grid_vertical-align-start slds-p-vertical_small'>
                        <div class="slds-col slds-size_1-of-1 slds-medium-size_6-of-12">
                            <lightning:input aura:id="IsDefault" type="checkbox" label="{!$Label.c.isDefaultLabel_cpa}" name="IsDefault"
                              class="slds-p-horizontal_x-small" checked="{!v.contactPointAddress.IsDefault}" />
                            <aura:if isTrue="{!not(v.createNewCpa)}">
                                <lightning:input aura:id="IsActive" type="checkbox" label="{!$Label.c.isActiveLabel_cpa}" name="IsActive"
                              class="slds-p-horizontal_x-small" checked="{!v.contactPointAddress.IsActive__c}" />
                            </aura:if>
                        </div>
                    </div>
                </aura:if>
            </div>
            <div class="slds-modal__footer">
                <lightning:button variant="neutral" label="{!$Label.c.cancelLabel_cpa}" onclick="{!c.cancelDialog}" />
                <lightning:button variant="brand" label="{!$Label.c.saveLabel_cpa}" onclick="{!c.save}" />
            </div>
        </div>
    </div>
</aura:component>
```

CONTROLLER FRONT END (JAVASCRIPT):



```
({
    doInit : function(component) {
        //get parent account id
        let action = component.get("c.getAccount");
        action.setParams({
            "userId":$A.get("$SObjectType.CurrentUser.Id")
        });
        action.setCallback(this,function(response){
            let state = response.getState();
            if(state === 'SUCCESS'){
                let result = response.getReturnValue();
                let results = result.split("-");
                let id = results[0];
                let name = results[1];
                let country = results[2];
                let hasBilling = results[3]=='true';
                let phone = results[4];
                component.set('v.contactPointAddress.ParentId',id);
                component.set('v.ParentName',name);
                if(country=='IT' || country =='Italy' || country == $A.get("{!$Label.c.italy_country}")){
                    component.set('v.contactPointAddress.Country',$A.get("{!$Label.c.italy_country}"));
                }else if(country == 'FR' || country == 'France' || country == $A.get("{!$Label.c.france_country}")){
                    component.set('v.contactPointAddress.Country',$A.get("{!$Label.c.france_country}"));
                }
                component.set('v.hasBilling',hasBilling);
                component.set('v.ContactPhone',phone);
                component.set('v.PhoneInserted',phone);
                let addrTypes;
                if(component.get('v.createNewBilling')==false && component.get('v.createNewCpa')==false && component.get('v.hasBilling')==false){
                    component.set('v.contactPointAddress.AddressType',$A.get("{!$Label.c.shipping_cpa}"));
                    addrTypes = [$A.get("{!$Label.c.billing_cpa}"),$A.get("{!$Label.c.shipping_cpa}")];
                    component.set('v.showPickAddType',true);
                    component.set('v.componentTitle',$A.get("{!$Label.c.create_cpa}"));
                }else if(component.get('v.createNewCpa')==true || component.get('v.hasBilling')==true){
                    //it's a shipping address
                    component.set('v.componentTitle',$A.get("{!$Label.c.create_cpa_shipping}"));
                    component.set('v.showPickAddType',false);
                    component.set('v.contactPointAddress.AddressType',$A.get("{!$Label.c.shipping_cpa}"));
                }else if(component.get('v.createNewBilling')==true){
                    //it's a billing address
                    component.set('v.componentTitle',$A.get("{!$Label.c.create_cpa_billing}"));
                    component.set('v.showPickAddType',false);
                    component.set('v.contactPointAddress.AddressType',$A.get("{!$Label.c.billing_cpa}"));
                }

                component.set('v.addressTypePickList',addrTypes);
            }
        });
        $A.enqueueAction(action);

    },

    save : function(component, event, helper) {
        let checks = helper.check(component) ;
        if(checks == null){
            var action = component.get("c.createContactPointAddress");
            if(component.get("v.contactPointAddress.Country")== $A.get("{!$Label.c.france_country}")){
                component.set("v.contactPointAddress.Country",'France');
            }else if(component.get("v.contactPointAddress.Country")== $A.get("{!$Label.c.italy_country}")){
                component.set("v.contactPointAddress.Country",'Italy');
            }
            console.log(component.get("v.contactPointAddress.AddressType"));
            action.setParams({
                "cpa" : component.get("v.contactPointAddress"),
                "phoneInserted" : component.get('v.PhoneInserted'),
                "contactPhone" : component.get('v.ContactPhone')
            });

            action.setCallback(this,function(response){
                var resultsToast = $A.get("e.force:showToast");
                var state = response.getState();
                if (state === "SUCCESS" && response.getReturnValue().startsWith('Error')==false) {
                    resultsToast.setParams({
                        "title": $A.get("$Label.c.saved"),
                        "type": "success",
                        "message": $A.get("$Label.c.savedCorrectly_cpa")
                    });
                    resultsToast.fire();
                    component.set("v.NewCpaId",response.getReturnValue());
                    component.set("v.createNewCpa",false);
                    component.set("v.createNewBilling",false);
                    $A.get("e.force:closeQuickAction").fire();
                    $A.get("e.force:refreshView").fire();
                } else if (state === "ERROR") {
                    resultsToast.setParams({
                        "title": $A.get("$Label.c.error"),
                        "type" :"error",
                        "message": $A.get("$Label.c.error") + JSON.stringify(response.getError())
                    });
                    resultsToast.fire();

                } else {
                    resultsToast.setParams({
                        "title": $A.get("$Label.c.error"),
                        "type":"error",
                        "message": $A.get("$Label.c.error") + JSON.stringify(response.getReturnValue())
                    });
                    resultsToast.fire();
                }
            });
            $A.enqueueAction(action);
        }else{
            var resultsToast = $A.get("e.force:showToast");
            resultsToast.setParams({
                "title": $A.get("$Label.c.error"),
                "type": "error",
                "message": checks
            });
            resultsToast.fire();
        }
    },
```

SERVER SIDE (APEX):



```
public without sharing class ContactPointAddressController {

    @AuraEnabled
    public static String createContactPointAddress(ContactPointAddress cpa, String phoneInserted, String contactPhone) {
        try{

        if(phoneInserted==''){
            phoneInserted = contactPhone;
        }
        List<ContactPointPhone> cps = [SELECT Id FROM ContactPointPhone WHERE ParentId =:cpa.ParentId and TelephoneNumber = :phoneInserted];
        if(cps.size()==0){
            ContactPointPhone cp = new ContactPointPhone(TelephoneNumber = phoneInserted,ParentId = cpa.ParentId);
            insert cp;
            cpa.ContactPointPhoneId = cp.Id;
        }else{
            cpa.ContactPointPhoneId = cps[0].Id;
        }
        //set is created by commerce
        cpa.IsCreatedByCommerce__c=true;
        Account acc = [SELECT AccountNumber FROM ACCOUNT WHERE Id=:cpa.ParentId];
        if(cpa.AddressType == System.Label.billing_cpa ){
            //convert to english for db
            cpa.AddressType = 'Billing';
            //set bil to code
            cpa.BillToCode__c = acc.AccountNumber;
            //default = true
            cpa.IsDefault = true;
            cpa.External_ID__c = 'BILL_'+acc.AccountNumber+'_'+cpa.BillToCode__c;
        }else if(cpa.AddressType == System.Label.shipping_cpa){
            cpa.AddressType = 'Shipping';
            Counter__c cnt;
            List<Counter__c> cnts = [SELECT Counter_cpa__c FROM Counter__c];
            if(cnts.size()!=1){
                //something is wrong, but we can ripristinate the value
                for(Counter__c c : cnts){
                    delete c;
                }
                List<ContactPointAddress> shipToCodes = [SELECT ShipToCode__c FROM ContactPointAddress WHERE AddressType = 'Shipping'];
                if(shipToCodes.size()==0){
                    cnt = new Counter__c(Name='Counter',Counter_cpa__c = 30000000);
                }else{
                    Decimal max = 0;
                    for(ContactPointAddress ship : shipToCodes){
                        if(ship.ShipToCode__c != '' && ship.ShipToCode__c != null){
                            try{
                            Decimal shipVal = Decimal.valueOf(ship.ShipToCode__c);
                            if(shipVal> max){
                                max = shipVal;
                            }
                            }catch(TypeException e){}
                        }
                    }
                    max = max+1;
                    cnt = new Counter__c(Name='Counter',Counter_cpa__c = max);
                }
                insert cnt;

            }else{
                cnt = cnts[0];
            }
            if(cnt.Counter_cpa__c == null || cnt.Counter_cpa__c < 30000000 ){
                List<ContactPointAddress> shipToCodes = [SELECT ShipToCode__c FROM ContactPointAddress WHERE AddressType = 'Shipping'];
                if(shipToCodes.size()==0){
                    cnt.Counter_cpa__c = 30000000;
                }else{
                    Decimal max = 0;
                    for(ContactPointAddress ship : shipToCodes){
                        if(ship.ShipToCode__c != '' && ship.ShipToCode__c != null){
                            try{
                            Decimal shipVal = Decimal.valueOf(ship.ShipToCode__c);
                            if(shipVal> max){
                                max = shipVal;
                            }
                            }catch(TypeException e){}
                        }
                    }
                    max = max+1;
                    cnt.Counter_cpa__c = max;
                }
            }
            cpa.ShipToCode__c = cnt.Counter_cpa__c+'';
            cnt.Counter_cpa__c = cnt.Counter_cpa__c +1;
            update cnt;
            cpa.External_ID__c = 'SHIP_'+acc.AccountNumber+'_'+cpa.ShipToCode__c;
            //if this is the new  default set old default ot not default
            if(cpa.IsDefault == true){
                List<ContactPointAddress> oldDefault = [SELECT Id,IsDefault FROM ContactPointAddress WHERE ParentId =:cpa.ParentId and AddressType = 'Shipping'
                    and IsDefault = true];
                if(oldDefault.size()>0){
                    ContactPointAddress toUpdate = oldDefault[0];
                    toUpdate.IsDefault = false;
                    update toUpdate;
                }
            }
            //set default if it's the first shipping cpa
            Double count = [SELECT Count() FROM ContactPointAddress WHERE ParentId =:cpa.ParentId and AddressType = 'Shipping' ];
            if(cpa.IsDefault == false && count == 0){
                cpa.IsDefault = true;
            }
        }


        insert cpa;
        return cpa.Id;
       } catch (Exception dmx) {
         // ApexPages.addMessages(dmx);
         return 'Error: '+dmx.getMessage();
       }
    }
```

```
@AuraEnabled
    public static String editContactPointAddress(ContactPointAddress cpa){
        try {
            if(cpa.AddressType == System.Label.billing_cpa ){
                //convert to english for db
                cpa.AddressType = 'Billing';
            }else if(cpa.AddressType == System.Label.shipping_cpa){
                cpa.AddressType = 'Shipping';
            }
            //verify that default is always only one
            List<ContactPointAddress> oldDefault = [SELECT Id,IsDefault FROM ContactPointAddress WHERE ParentId =:cpa.ParentId and AddressType =
                    :System.Label.shipping_cpa and IsDefault = true];
            if(oldDefault.size()>0){
                ContactPointAddress cpaDef = oldDefault[0];
                if(cpaDef.Id != cpa.Id){
                    cpaDef.IsDefault = false;
                    List<ContactPointAddress> result = new List<ContactPointAddress>();
                    result.add(cpa);
                    result.add(cpaDef);
                    update result;
                }else{
                    update cpa;
                }
            }else{
                update cpa;
            }
            return null;
        }  catch (Exception dmx) {
            return dmx.getMessage();
        }
    }

    @AuraEnabled
    public static string getAccount(String userId){
        String accountId = null;
        String country = null;
        String name = null;
        String phone = null;
        Boolean hasBilling = false;
        String contactId = null;
        List<User> u = [SELECT ContactId,AccountId FROM USER WHERE Id =: userId];
        if(u.size()>0){
            if(u[0].ContactId != null && u[0].AccountId!= null ){
                //customer user
                contactId=u[0].ContactId;
                accountId=u[0].AccountId;
                List<Contact> c = [SELECT Phone FROM Contact WHERE Id=:contactId ];
                List<Account> a = [SELECT BillingCountry,Name FROM Account WHERE Id=:accountId];
                if(c.size()>0 && a.size()>0){
                    phone = c[0].Phone;
                    country = a[0].BillingCountry;
                    name = a[0].Name;
                }
                Integer cpa = [SELECT COUNT() FROM ContactPointAddress WHERE ParentId =:accountId and AddressType = 'Billing'];
                if(cpa>0){
                    hasBilling = true;
                }

        }
        return accountId+'-'+name+'-'+country+'-'+hasBilling+'-'+phone;
    }

    @AuraEnabled
    public static string getBillingAddress(String accountId){
        String state = null;
        String city = null;
        String street = null;
        String postalCode = null;
        String houseNumber = null;
        String name = null;
        List<ContactPointAddress> cpa = [SELECT Name,State,Street,City,PostalCode,HouseNumber__c FROM ContactPointAddress WHERE ParentId = :accountId and
AddressType = 'Billing' ];
        if(cpa.size()>0){
            name = cpa[0].Name;
            state = cpa[0].State;
            city = cpa[0].City;
            street = cpa[0].Street;
            postalCode = cpa[0].PostalCode;
            houseNumber = cpa[0].HouseNumber__c;
        }
        return name+'?'+state+'?'+city+'?'+street+'?'+postalCode+'?'+houseNumber;
    }

    @AuraEnabled
    public static string getRecord(String recordId){
        List<ContactPointAddress> cpa = [SELECT Name, ParentId, ContactPointPhoneId, AddressType, IsDefault, IsActive__c,
IsRelatedToAnotherCompany__c,City,Country,State,Street,HouseNumber__c,PostalCode FROM ContactPointAddress WHERE Id = :recordId];

        if(cpa.size()>0){
            if(cpa[0].AddressType == 'Shipping'){
                cpa[0].AddressType = System.Label.shipping_cpa;
            }else if(cpa[0].AddressType == 'Billing'){
                cpa[0].AddressType = System.Label.billing_cpa;
            }
            List<Account> acc = [SELECT Name FROM Account WHERE Id=: cpa[0].ParentId];
            List<ContactPointPhone> cpp = [SELECT TelephoneNumber FROM ContactPointPhone WHERE Id= :cpa[0].ContactPointPhoneId];
            if(acc.size()>0 && cpp.size()>0){
                return
cpa[0].Name+'?'+acc[0].Name+'?'+cpp[0].TelephoneNumber+'?'+cpa[0].AddressType+'?'+cpa[0].IsDefault+'?'+cpa[0].IsActive__c+'?'+cpa[0].IsRelatedToAnotherCompany__c
+'?'+cpa[0].City+'?'+cpa[0].Country+'?'+cpa[0].State+'?'+cpa[0].Street+'?'+cpa[0].HouseNumber__c+'?'+cpa[0].PostalCode+'?'+cpa[0].ParentId;
            }
        }
        return null;

    }
}
```

Code 2: Reset Password page

FRONT END (Visualforce, HTML)

```
<apex:page id="ResetPassword" showHeader="false" controller="ChangePasswordController" title="{!$Label.ResetPassword}" docType="html-5.0">

  <div class="resetPasswordPage">
    <apex:image styleClass='logoImage' width='183px' height="87px" value="{!LEFT($Api.Partner_Server_URL_510, FIND('/services', $Api.Partner_Server_URL_
510))}file-asset-public/iarploginlogo?oid={!$Organization.Id}"
    />

    <apex:outputPanel rendered="{!ifFromSAP}" styleClass="welcomeBox">
      <apex:outputText styleClass="titlePage" value="{!$Label.welcome_epta}" />
      <br/>
      <apex:outputText style="font-size:14px; " value="{!$Label.set_account1}" />
      <br/>
      <apex:outputText style="font-size:14px; " value="{!$Label.set_account2}" />
      <br/>
    </apex:outputPanel>

    <apex:outputPanel rendered="{!ifFromSAP==false}" styleClass="welcomeBox">
      <apex:outputText styleClass="resetPageHeader" value="{!$Label.reset_your_password}" />
    </apex:outputPanel>

    <apex:outputPanel>
      <div class="errorMessage">
        {!errorMessage}
      </div>
    </apex:outputPanel>

    <apex:panelGrid cellpadding="" cellspacing="" bgcolor="white" columns="1" styleClass="passwordTable">
      <apex:outputLabel value="{!$Label.passwordInfo}" />
      <apex:outputLabel value="{!$Label.passwordInfo2}" />
      <apex:outputLabel value="{!$Label.passwordInfo3}" />
      <apex:outputLabel value="{!$Label.passwordInfo4}" />
    </apex:panelGrid>

    <apex:form id="theForm" styleClass="formContainer">
      <div class="labelSpaceBottom">
        <apex:outputLabel style="color:rgb(240, 52, 52, 1)" value="*" />
        <apex:outputLabel style="color:rgba(0, 0, 0, 0.81); font-size:14px;" value="{!$Label.site.new_password}" for="psw" />
      </div>
      <apex:inputSecret id="psw" value="{!newPassword}" styleClass="inputPassword" />

      <div class="labelSpaceBottom">
        <apex:outputLabel style="color:rgb(240, 52, 52, 1)" value="*" />
        <apex:outputLabel style="color:rgba(0, 0, 0, 0.81); font-size:14px;" value="{!$Label.site.verify_new_password}" for="vpsw"
        />
      </div>
      <apex:inputSecret id="vpsw" value="{!verifyNewPassword}" styleClass="inputPassword" />
      <apex:outputPanel rendered="{!ifFromSAP}">
        <table class="termCondition">
          <tr>
            <td style="vertical-align: top;">
              <apex:outputLabel style="color:rgb(240, 52, 52, 1)" value="*" />
              <apex:inputCheckbox id="privacyflag" value="{!privacyflag}" label="Privacy Flag" selected="false" style="display: inline;
              font-size:10px;
              position:relative;
              color: rgb(0, 0, 0);
              background-color:#1797c0;" />
            </td>
            <td style="font-size: 12px; font-family: sans-serif; color: rgba(5, 5, 5, 0.87);">
              <apex:outputLabel value="{!$Label.cond1}" />
              <apex:outputPanel rendered="{!ifIsFra}">&nbsp;</apex:outputPanel>
              <!--<apex:outputLink target="_blank" value="../../../s/privacy-policy" id="theLink">{!$Label.PrivacyPolicy}</apex:outputLink>-->
              <apex:commandButton value="{!$Label.PrivacyPolicy}" action="{!openPolicy}" rerender="plpopup" style="background: none;
              border: none;
              padding: 0;
              margin: 0;
              font-size: 12px;
              color: #0061a1;
              font-family: sans-serif;"/>
              <apex:outputLabel value="{!$Label.cond2}" />
              <br/>
              <br/>
            </td>
          </tr>
        <tr>
          <td style="vertical-align: top;">
            <apex:outputLabel style="color:rgb(240, 52, 52, 1)" value="*" />
            <apex:inputCheckbox id="termsflag" value="{!termsflag}" label="Terms Flag" selected="false" style="display: inline;
                font-size:10px;
                position:relative;
                color: rgb(0, 0, 0);
                background-color:#1797c0;" />
          </td>
          <td style="font-size: 12px; font-family: sans-serif; color: rgba(5, 5, 5, 0.87);padding-top: 4px;">
            <apex:outputLabel value="{!$Label.termsCond1}" />&nbsp;
            <apex:commandButton value="{!$Label.termsCond2}" action="{!openTermsAndCond}" rerender="tcpopup" style="background: none;
            border: none;
            padding: 0;
            margin: 0;
            font-size: 12px;
            color: #0061a1;
            font-family: sans-serif;"/>
            <!--<apex:outputLink target="_blank" value="../../../s/terms-and-conditions" id="theLink3">{!$Label.termsCond2}</apex:outputLink>-->
            <br/>
            <br/>
          </td>
        </tr>
        <tr>
          <td style="vertical-align: top; padding-left: 6px; font:lato; font-size:12px;">
            <apex:inputCheckbox id="marketingflag" value="{!marketingflag}" label="Marketing Flag" selected="false" style="display: inline;
                font-size:10px;
                position:relative;
                color: rgb(0, 0, 0);
                background-color:#1797c0;" />
          </td>
          <td style="font-size: 12px; font-family: sans-serif; color: rgba(5, 5, 5, 0.87); ">
            <apex:outputLabel value="{!$Label.marketing_label1}" />
            <apex:outputLabel value="{!$Label.marketing_label2}" />
            <apex:outputLabel value="{!$Label.marketing_label3}" />
            <apex:outputLabel value="{!$Label.marketing_label4}" />
          </td>
        </tr>
       </table>
     </apex:outputPanel>
     <apex:outputPanel rendered="{!isFirstLogin}">
       <apex:commandButton id="cpwbtn" action="{!changePassword}" value="{!$Label.genPassButton}" styleClass="submitButton" />
     </apex:outputPanel>
     <apex:outputPanel rendered="{!IF (isFirstLogin==false,true,false)}">
       <apex:commandButton id="cpwbtn1" action="{!changePassword}" value="{!$Label.ResetPassword}" styleClass="submitButton" />
     </apex:outputPanel>
     <br/>


     <apex:outputPanel id="tcpopup">
        <apex:outputPanel styleClass="popupBackground" layout="block" rendered="{!openTermsAndCond}"/>
        <apex:outputPanel styleClass="custPopup" layout="block" rendered="{!openTermsAndCond}">
           <c:termsAndCondComponent isFra="{!ifIsFra}"></c:termsAndCondComponent>
           <br/><br/>
           <apex:commandButton value="{!$Label.closeLabel}" action="{!closeTermsAndCond}" rerender="tcpopup" styleClass="myButtonClose"/>
        </apex:outputPanel>
     </apex:outputPanel>

     <apex:outputPanel id="plpopup">
        <apex:outputPanel styleClass="popupBackground" layout="block" rendered="{!openPolicy}"/>
        <apex:outputPanel styleClass="custPopup" layout="block" rendered="{!openPolicy}">
           <c:policyComponent isFra="{!ifIsFra}"></c:policyComponent><br/><br/>
           <apex:commandButton value="{!$Label.closeLabel}" action="{!closePolicy}" rerender="plpopup" styleClass="myButtonClose"/>
        </apex:outputPanel>
     </apex:outputPanel>

    </apex:form>
   </div>
   <br/>
  </apex:define>
  <apex:define name="footer"></apex:define>
</apex:page>
```

Code3: Login Form

FRONT END (Aura Component, HTML)

```
<aura:component controller="LightningLoginFormController" implements="forceCommunity:availableForAllPageTypes, force:hasRecordId">
  <aura:attribute name="showError" type="Boolean" required="true" description="" default="false" access="private" />
  <aura:attribute name="errorMessage" type="String" required="false" description="" access="private" />
  <aura:attribute name="startUrl" type="String" required="false" default='/' description="The url you go to after a successful login" />
  <aura:attribute name="usernameLabel" type="String" required="false" default="Username" />
  <aura:attribute name="passwordLabel" type="String" required="false" default="Password" />
  <aura:attribute name="loginButtonLabel" type="String" required="false" default="Log in" />
  <aura:attribute name="RememberMeLabel" type="Boolean" required="false" default="Remember Me" />
  <aura:attribute name="expid" type="String" required="false" description="The branding experience ID" />

  <aura:attribute name="forgotPasswordLabel" type="String" required="false" default="Forgot your password?" />
  <aura:attribute name="selfRegisterLabel" type="String" required="false" default="Not a member?" />
  <aura:attribute name="forgotPasswordUrl" type="String" required="false" default="/ForgotPassword" />
  <aura:attribute name="selfRegisterUrl" type="String" required="false" default="/SelfRegister" />

  <aura:attribute name="isUsernamePasswordEnabled" type="Boolean" access="private" />
  <aura:attribute name="isSelfRegistrationEnabled" type="Boolean" access="private" />
  <aura:attribute name="communityForgotPasswordUrl" type="String" access="private" />
  <aura:attribute name="communitySelfRegisterUrl" type="String" access="private" />

  <aura:registerevent name="sitePropagatedStartUrl" type="c:setStartUrl" />
  <aura:handler name="init" value="{!this}" action="{!c.initialize}" />
  <aura:dependency resource="c:setStartUrl" type="EVENT" />
   <aura:handler event="c:setStartUrl" action="{!c.setStartUrl}" />
  <aura:handler event="c:setExpId" action="{!c.setExpId}" />
  <aura:dependency resource="c:setExpId" type="EVENT" />

  <div>
    <aura:renderIf isTrue="{!v.isUsernamePasswordEnabled}">
      <span>
        <aura:renderIf isTrue="{!v.showError}">
          <div id="error">
            <ui:outputRichText value="{!v.errorMessage}" />
          </div>
        </aura:renderIf>
      </span>
      <div id="sfdc_username_container" class="sfdc">
        <span id="sfdc_user" class="login-icon" data-icon="a"></span>
        <ui:inputText value="" aura:id="username" placeholder="{!v.usernameLabel}" keyup="{!c.onKeyUp}" class="input sfdc_usernameinput sfdc" label="{!v
.usernameLabel}" labelClass="assistiveText" />
      </div>

      <div id="sfdc_password_container" class="sfdc">
        <span id="sfdc_lock" class="login-icon sfdc" data-icon="c"></span>
        <ui:inputSecret value="" aura:id="password" placeholder="{!v.passwordLabel}" keyup="{!c.onKeyUp}" class="input sfdc_passwordinput sfdc" label="{!
v.passwordLabel}" labelClass="assistiveText" />
      </div>

      <div class="sfdc">
        <ui:button aura:id="submitButton" label="{!v.loginButtonLabel}" press="{!c.handleLogin}" class="sfdc_button" />
      </div>
      <div>
        <lightning:input type="checkbox" label="{!$Label.c.Remember_Me}" aura:id="RememberMe" placeholder="{!v.RememberMeLabel}" />
      </div>


      <div id="sfdc_forgot" class="sfdc">
        <span><a href="{!if(v.communityForgotPasswordUrl == null, v.forgotPasswordUrl, v.communityForgotPasswordUrl)}">{!v.forgotPasswordLabel}</a></s
pan>
        <aura:renderIf isTrue="{!v.isSelfRegistrationEnabled}">
          <span style="float:right"><a href="{!if(v.communitySelfRegisterUrl == null, v.selfRegisterUrl, v.communitySelfRegisterUrl)}">{!v.selfRegisterLabel}</a
></span>
        </aura:renderIf>
      </div>
    </aura:renderIf>
  </div>

</aura:component>
```

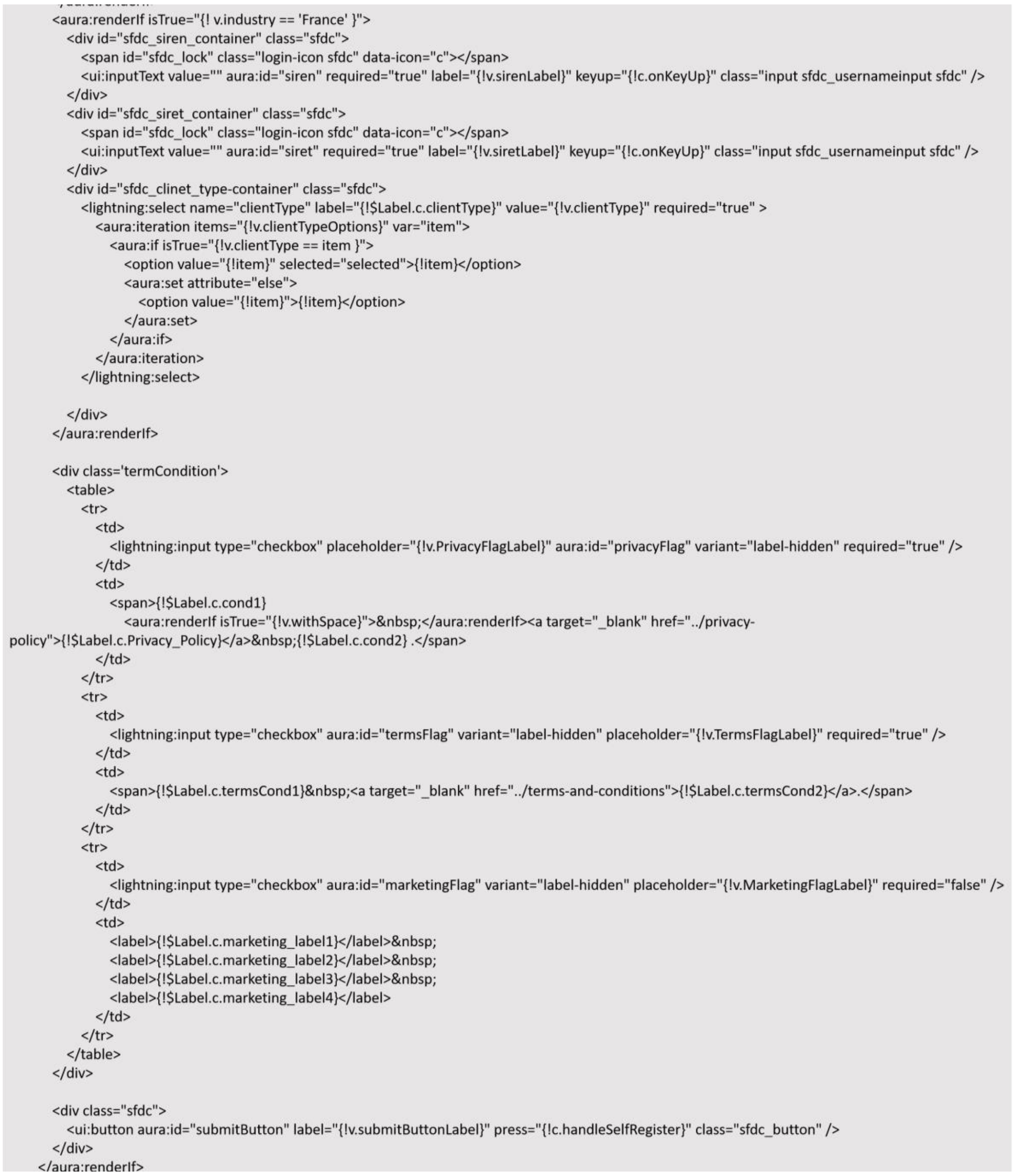
BACK END (APEX)

```
@AuraEnabled
  public static String login(String username, String password, String startUrl) {

    if(username == null || String.isEmpty(username)) {
      return Label.email_is_required;
    }

    if(!Pattern.matches('^[a-zA-Z0-9+_.-]+@[a-zA-Z0-9.-]+$', username.toLowerCase())) {
      return Label.email_invalid;
    }

    if(password == null || String.isEmpty(password)) {
      return Label.password_required;
    }
    Savepoint sp = null;
    try {
      ApexPages.PageReference lgn = Site.login(username, password, startUrl);
      if(lgn == null ){
        //return null;
         ApexPages.addmessage(new ApexPages.message(ApexPages.severity.Info, '* You\'ve entered an incorrect username and/or password. Please re-
enter your log-in information.'));
      }
      System.debug('Login is successful: ' + username);

      sp = Database.setSavepoint();

      Id accountId = [SELECT AccountId FROM User WHERE Username = :username LIMIT 1].AccountId;
      System.debug('Username: ' + username + ', AcconutId: '+ accountId);
      Account accountToUpdate = [SELECT Id, Status__c FROM Account WHERE Id = :accountId LIMIT 1];
      accountToUpdate.Status__c = 'Active';
      update accountToUpdate;
      System.debug('Update completed - Username: ' + username + ', AcconutId: '+ accountId);

      System.debug('Id: ' + accountId + ', BuyerId: '+ accountId);
      buyerAccount buyerAccountoUpdate = [SELECT Id, BuyerStatus FROM buyerAccount WHERE BuyerId = :accountId LIMIT 1];
      buyerAccountoUpdate.BuyerStatus = 'Attivo';
      update buyerAccountoUpdate;
      System.debug('Update completed - Id: ' + accountId + ', BuyerId '+ accountId);


      aura.redirect(lgn);
      return null;
    }
    catch(System.DmlException e) {
      Database.rollback(sp);
      System.debug('DmlException caught: ' + e.getMessage());
      return 'Please contact system administrator';
    }
    catch (Exception ex) {
      System.debug('Error login - Username: ' + username);
      return Label.failedLogin;
    }
  }
```

Code 4: Self Registration

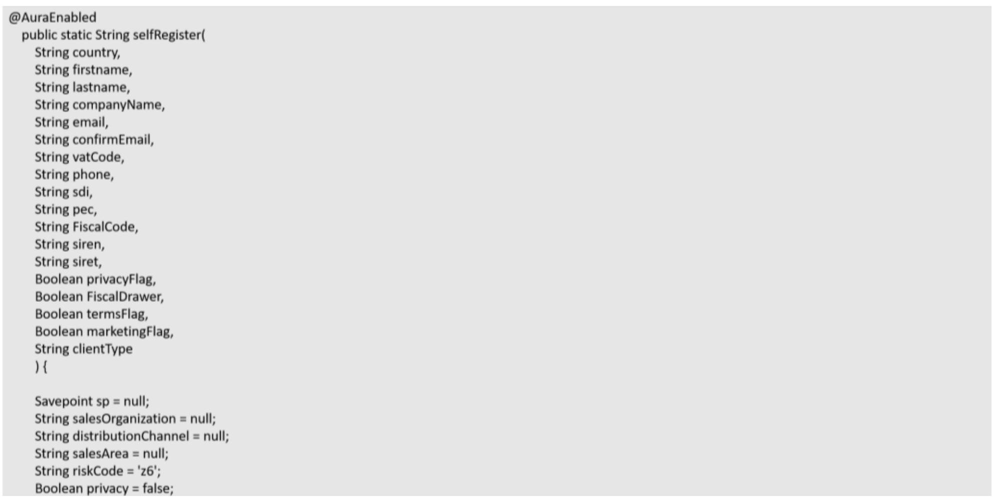
FRONT END (Aura Component, HTML)

```
<!-- add implements="forceCommunity:availableForAllPageTypes" to surface the component in community builder -->
<aura:component implements="flexipage:availableForAllPageTypes,forceCommunity:availableForAllPageTypes" access="global"
controller="LightningSelfRegisterController">

  <!-- Please uncomment
  <aura:dependency resource="siteforce:registerQueryEventMap" type="EVENT"/>
  -->
  <aura:handler event="c:setStartUrl" action="{!c.setStartUrl}" />
  <aura:handler event="c:setExpId" action="{!c.setExpId}" />
  <aura:dependency resource="c:setExpId" type="EVENT" />

  <div id="sfdc_register_form" class="sfdc">
    <aura:renderIf isTrue="{!v.showError}">
      <div id="error">
        <ui:outputRichText value="{!v.errorMessage}" />
      </div>
    </aura:renderIf>
    <!-- <div class="alreadyCustomerCheckbox">
      <ui:inputCheckbox aura:id="checkbox"  change="{!c.onCheck}"/>
      <p style="padding-left:0.5rem"> {!v.alreadyCustomer}</p>
    </div> -->

    <table >
      <tr>
        <td style="padding: 0 15px;">
          <ui:inputCheckbox  aura:id="checkbox"  change="{!c.onCheck}"/>
        </td>
        <td>
          <p> {!v.alreadyCustomer}</p>
        </td>
      </tr>
    </table>
    <aura:renderIf isTrue="{!v.myBool}">
      <div id="sfdc_VATCode_container" class="sfdc" required="true">

        <!-- <div id="sfdc_VATCode_container" class="sfdc">
        <span id="sfdc_vatcode" class="login-icon" data-icon="k"></span>
        <ui:inputText value="" aura:id="vatcode" required="true" label="{!v.VATCodeLabel}" keyup="{!c.onKeyUp}" class="input sfdc_usernameinput sfdc"
/>
      </div> -->
        <span id="sfdc_vatcode" class="login-icon sfdc" data-icon="k"></span>
        <lightning:input id="vatcode" aura:id="vatcode2" value="{!v.Vat}" required="true" label="{!v.VATCodeLabel}" onkeyup="{!c.onKeyUp}" class="input
sfdc_usernameinput sfdc customInput" fieldLevelHelp="{!v.infoPIVA}" />
      </div>
      <aura:renderIf isTrue="{!v.includeEmailField}">
        <div id="sfdc_email_container" class="sfdc" required="true">
          <span id="sfdc_email" class="login-icon sfdc" data-icon="k"></span>
          <ui:inputText value="{!v.Email}" aura:id="email2" required="true" label="{!v.emailLabel}" class="input sfdc_usernameinput sfdc" />
        </div>
      </aura:renderIf>

      <div class="sfdc">
        <ui:button aura:id="submitButton2" label="{!v.activateButtonLabel}" press="{!c.handleRegister}" class="sfdc_button" />
      </div>

    </aura:renderIf>
    <aura:renderIf isTrue="{!!v.myBool}">

      <div id="sfdc_country_container" class="sfdc">
        <ui:inputSelect class="single" aura:id="country" value="" label="{!v.industryLabel}" change="{!c.handleChange}" required="true">
          <ui:inputSelectOption label="None" value="" text="" />
          <ui:inputSelectOption label="{!$Label.c.italy_country}" value="Italy" text="Italy" />
          <ui:inputSelectOption label="{!$Label.c.france_country}" value="France" text="France" />
        </ui:inputSelect>
      </div>
      <div id="sfdc_username_container" class="sfdc">
        <span id="sfdc_username" class="login-icon" data-icon="s"></span>
        <ui:inputText required="true" value="" aura:id="firstname" label="{!v.firstnameLabel}" keyup="{!c.onKeyUp}" class="input sfdc_usernameinput
sfdc" />
      </div>
      <div id="sfdc_nickname_container" class="sfdc">
        <span id="sfdc_nickname" class="login-icon" data-icon="s"></span>
        <ui:inputText value="" aura:id="lastname" required="true" label="{!v.lastnameLabel}" keyup="{!c.onKeyUp}" class="input sfdc_usernameinput
sfdc" />
      </div>
      <div id="sfdc_CompanyName_container" class="sfdc">
        <span id="sfdc_companyname" class="login-icon" data-icon="k"></span>
        <ui:inputText value="" aura:id="companyname" required="true" label="{!v.CompanyNameLabel}" keyup="{!c.onKeyUp}" class="input
sfdc_usernameinput sfdc" />
      </div>
      <aura:renderIf isTrue="{!v.includeEmailField}">
        <div id="sfdc_email_container" class="sfdc" required="true">
          <span id="sfdc_email" class="login-icon sfdc" data-icon="k"></span>
          <ui:inputText value="{!v.Email}" aura:id="email" required="true" label="{!v.emailLabel}" class="input sfdc_usernameinput sfdc" />
        </div>
      </aura:renderIf>
      <div id="sfdc_VATCode_container" class="sfdc">

        <!-- <div id="sfdc_VATCode_container" class="sfdc">
        <span id="sfdc_vatcode" class="login-icon" data-icon="k"></span>
        <ui:inputText value="" aura:id="vatcode" required="true" label="{!v.VATCodeLabel}" keyup="{!c.onKeyUp}" class="input sfdc_usernameinput sfdc"
/>
      </div> -->
        <span id="sfdc_vatcode" class="login-icon" data-icon="k"></span>
        <lightning:input id="vatcode" aura:id="vatcode" value="" required="true" label="{!v.VATCodeLabel}" onkeyup="{!c.onKeyUp}" class="input
sfdc_usernameinput sfdc customInput" fieldLevelHelp="{!v.infoPIVA}" />
      </div>

      <div id="sfdc_phone_container" class="sfdc">
        <span id="sfdc_phone" class="login-icon" data-icon="k"></span>
        <ui:inputPhone value="" aura:id="phone" required="true" label="{!v.phoneLabel}" keyup="{!c.onKeyUp}" class="input sfdc_usernameinput sfdc" />
      </div>
      <aura:renderIf isTrue="{! v.industry == 'Italy' }">
        <div id="sfdc_sdi_container" class="sfdc">
          <span id="sfdc_lock" class="login-icon sfdc" data-icon="c"></span>
          <ui:inputText value="" aura:id="sdi" label="{!v.SDILabel}" keyup="{!c.onKeyUp}" class="input sfdc_usernameinput sfdc" />
        </div>
        <div id="sfdc_pec_container" class="sfdc">
          <span id="sfdc_lock" class="login-icon sfdc" data-icon="c"></span>
          <ui:inputEmail aura:id="pec" label="{!v.PECLabel}" class="input sfdc_usernameinput sfdc" />
        </div>
        <div>
          <lightning:input type="checkbox" label="{!$Label.c.FiscalDrawer}" aura:id="FiscalDrawer" placeholder="{!v.FiscalDrawerLabel}" />
        </div>
        <div id="sfdc_Fiscalode_container" class="sfdc">
          <span id="sfdc_lock" class="login-icon sfdc" data-icon="c"></span>
          <ui:inputEmail value="" aura:id="FiscalCode" required="true" label="{!v.FiscalCodeLabel}" keyup="{!c.onKeyUp}" class="input
sfdc_usernameinput sfdc" />
        </div>
      </aura:renderIf>
```



```
      <aura:renderIf isTrue="{! v.industry == 'France' }">
        <div id="sfdc_siren_container" class="sfdc">
          <span id="sfdc_lock" class="login-icon sfdc" data-icon="c"></span>
          <ui:inputText value="" aura:id="siren" required="true" label="{!v.sirenLabel}" keyup="{!c.onKeyUp}" class="input sfdc_usernameinput sfdc" />
        </div>
        <div id="sfdc_siret_container" class="sfdc">
          <span id="sfdc_lock" class="login-icon sfdc" data-icon="c"></span>
          <ui:inputText value="" aura:id="siret" required="true" label="{!v.siretLabel}" keyup="{!c.onKeyUp}" class="input sfdc_usernameinput sfdc" />
        </div>
        <div id="sfdc_clinet_type-container" class="sfdc">
          <lightning:select name="clientType" label="{!$Label.c.clientType}" value="{!v.clientType}" required="true" >
            <aura:iteration items="{!v.clientTypeOptions}" var="item">
              <aura:if isTrue="{!v.clientType == item }">
                <option value="{!item}" selected="selected">{!item}</option>
                <aura:set attribute="else">
                  <option value="{!item}">{!item}</option>
                </aura:set>
              </aura:if>
            </aura:iteration>
          </lightning:select>

        </div>
      </aura:renderIf>

      <div class='termCondition'>
        <table>
          <tr>
            <td>
              <lightning:input type="checkbox" placeholder="{!v.PrivacyFlagLabel}" aura:id="privacyFlag" variant="label-hidden" required="true" />
            </td>
            <td>
              <span>{!$Label.c.cond1}
                <aura:renderIf isTrue="{!v.withSpace}">&nbsp;</aura:renderIf><a target="_blank" href="../privacy-
policy">{!$Label.c.Privacy_Policy}</a>&nbsp;{!$Label.c.cond2} .</span>
            </td>
          </tr>
          <tr>
            <td>
              <lightning:input type="checkbox" aura:id="termsFlag" variant="label-hidden" placeholder="{!v.TermsFlagLabel}" required="true" />
            </td>
            <td>
              <span>{!$Label.c.termsCond1}&nbsp;<a target="_blank" href="../terms-and-conditions">{!$Label.c.termsCond2}</a>.</span>
            </td>
          </tr>
          <tr>
            <td>
              <lightning:input type="checkbox" aura:id="marketingFlag" variant="label-hidden" placeholder="{!v.MarketingFlagLabel}" required="false" />
            </td>
            <td>
              <label>{!$Label.c.marketing_label1}</label>&nbsp;
              <label>{!$Label.c.marketing_label2}</label>&nbsp;
              <label>{!$Label.c.marketing_label3}</label>&nbsp;
              <label>{!$Label.c.marketing_label4}</label>
            </td>
          </tr>
        </table>
      </div>

      <div class="sfdc">
        <ui:button aura:id="submitButton" label="{!v.submitButtonLabel}" press="{!c.handleSelfRegister}" class="sfdc_button" />
      </div>
    </aura:renderIf>
```

BACK END (APEX)



```
@AuraEnabled
  public static String selfRegister(
    String country,
    String firstname,
    String lastname,
    String companyName,
    String email,
    String confirmEmail,
    String vatCode,
    String phone,
    String sdi,
    String pec,
    String FiscalCode,
    String siren,
    String siret,
    Boolean privacyFlag,
    Boolean FiscalDrawer,
    Boolean termsFlag,
    Boolean marketingFlag,
    String clientType
    ) {

    Savepoint sp = null;
    String salesOrganization = null;
    String distributionChannel = null;
    String salesArea = null;
    String riskCode = 'z6';
    Boolean privacy = false;
```

```
        if(!checkFieldsMandatory(fieldsToCheck)) {
            return Label.fieldsMandatory;
        }

        if(!Pattern.matches('^[a-zA-Z0-9+_.-]+@[a-zA-Z0-9.-]+$', email.toLowerCase())) {
            return Label.email_invalid;
        }


        if(!Pattern.matches('^[0-9a-zA-ZàáâäãåąčćęèéêëėįìíîïłńòóôöõøùúûüųūÿýżźñçčšžÀÁÂÄÃÅĄĆČĖĘÈÉÊËÌÍÎÏĮŁŃÒÓÔÖÕØÙÚÛÜŲŪŸÝŻŹÑßÇŒÆČŠŽ∂ð ,.\'-]+$',
firstname)) {
            return Label.firstname_invalid;
        }

        if(!Pattern.matches('^[0-9a-zA-ZàáâäãåąčćęèéêëėįìíîïłńòóôöõøùúûüųūÿýżźñçčšžÀÁÂÄÃÅĄĆČĖĘÈÉÊËÌÍÎÏĮŁŃÒÓÔÖÕØÙÚÛÜŲŪŸÝŻŹÑßÇŒÆČŠŽ∂ð ,.\'-]+$',
lastname)) {
            return Label.lastname_invalid;
        }



        if(country == 'France') {
            if(!Pattern.matches('^(FR)[0-9A-Z]{2}[0-9]{9}$', vatCode)) {
                return Label.vatcode_invalid;
            }
        } else if(country == 'Italy') {
            if(!Pattern.matches('^(IT)[0-9]{11}$', vatCode)) {
                return Label.vatcode_invalid;
            }
        }
        phone = phone.replaceAll('(\\s+)', '');
        if(!Pattern.matches('^([0-9\\+]{0,20})$', phone)) {
            return Label.phone_invalid;
        }


        if(country == 'Italy') {
            if ((sdi == null || String.isEmpty(sdi)) && (pec == null || String.isEmpty(pec)) && (!FiscalDrawer)) {
                return Label.SDI_PEC_FiscalDrawer_is_required;

            }

            if(FiscalDrawer){
                FDrawer = true;
            }

            if((sdi != null && !String.isEmpty(sdi)) && !Pattern.matches('^[0-9A-Z]{7}$', sdi)) {
                return Label.sdi_invalid;
            }

            if((pec != null && !String.isEmpty(pec)) && !Pattern.matches('^[a-zA-Z0-9+_.-]+@[a-zA-Z0-9.-]+$', pec)) {
                return Label.pec_invalid;
            }


            if (FiscalCode == null || String.isEmpty(FiscalCode)) {
                return Label.FiscalCode_required;
            }

            if(!Pattern.matches('^([A-Za-z]{6}[0-9lmnpqrstuvLMNPQRSTUV]{2}[abcdehlmprstABCDEHLMPRST]{1}[0-9lmnpqrstuvLMNPQRSTUV]{2}[A-Za-z]{1}[0-
9lmnpqrstuvLMNPQRSTUV]{3}[A-Za-z]{1})$|([0-9]{11})$', FiscalCode)) {
                return Label.FiscalCode_invalid;
            }


        }
        else if(country == 'France') {
            if (siren == null || String.isEmpty(siren) ) {
                return Label.siren_is_required;
            }

            if (siret == null || String.isEmpty(siret) ) {
                return Label.siret_is_required;
            }

        }

        if(!termsFlag) {
            return Label.terms_flag_is_required;
        } else {
            termsFlag = true;
        }

        if(!privacyFlag) {
            return Label.privacy_flag_is_required;
        } else {
            privacy = true;
        }

        if(!marketingFlag){
            marketingConsentDate = null;
        }else {
            marketingConsentDate = Date.today();
        }
        //calculate account number
```

Code 5: User Profile

FRONT END (JS controller for Aura Component)

```
doInit : function(component) {
        let action = component.get("c.getUserInfo");
        action.setParams({
                "userId":$A.get("$SObjectType.CurrentUser.Id")        });
        action.setCallback(this,function(response){
         let state = response.getState();
        let result = response.getReturnValue();
        if(state === 'SUCCESS' && result!= null && result!=''){
                let res = JSON.parse(result);
                let u = {};
                u.Name = res.Name;
                u.Username = res.Username;
                u.Email = res.Email;
                u.Phone = res.Phone;
                u.FullPhotoUrl = res.FullPhotoUrl;
                u.CompanyName = res.Company;
                u.Id = res.Id;
                u.Country = res.Country;
                u.Marketing = res.Marketing=='true'?true:false;
                component.set('v.User',u);
                component.set('v.newEmail',u.Email);
                if(u.FullPhotoUrl){
                        component.set('v.hasPicture',true);
                }
        }else if(state === 'ERROR'){
                component.set('v.Error',response.getError()[0].message);
        }else{
                component.set('v.Error','There was an error');
        }
})
$A.enqueueAction(action);
},


 handleSaveMarketing : function (component){
        let action = component.get("c.updateMarketing");
        action.setParams({
                "userId":$A.get("$SObjectType.CurrentUser.Id"),
                "marketingFlag":component.get('v.User.Marketing')
        });
        action.setCallback(this,function(response){
                 let state = response.getState();
                let result = response.getReturnValue();
                let resultsToast = $A.get("e.force:showToast");
                if(state === 'SUCCESS' && result == true){
                         resultsToast.setParams({
                                "title": $A.get("$Label.c.saved"),
                                "type": "success",
                                "message": component.get('v.marketingUpdated')
                                 });
                        resultsToast.fire();
                }else{
                        resultsToast.setParams({
                                "title": $A.get("$Label.c.error"),
                                "type": "error",
                                 "message": component.get('v.marketingError')
                        });
                        resultsToast.fire();
                }
                component.set('v.changeMarketing',false);
        })
        $A.enqueueAction(action);
},
```

BACK END (Apex)

```
@AuraEnabled
public static String getUserInfo(String userId) {
        if(userId==null){
                return null;
        }
        Map<String,String> result=new Map<String,String>();
         List<User> u = [SELECT Name,Username,Email,AccountId,ContactId,FullPhotoUrl FROM User WHERE Id=:userId];
        if(u.size()>0 && u[0].ContactId != null && u[0].AccountId!= null){
                Contact c = [SELECT Marketing_Flag__c,phone,Email FROM Contact WHERE Id =:u[0].ContactId];
                Account a = [SELECT Name,BillingCountry FROM Account WHERE Id =:u[0].AccountId];          result.put('Name',u[0].Name);
                result.put('Username',u[0].Username);
                result.put('Id',u[0].Id);
                result.put('Email',u[0].Email);
                result.put('FullPhotoUrl',u[0].FullPhotoUrl);
                result.put('Phone',c.Phone);
                result.put('Company',a.Name);
                result.put('Country',a.BillingCountry);
        result.put('Marketing',c.Marketing_Flag__c+'');
                if(c.Email != u[0].Email){
                        c.Email = u[0].Email;
                        update c;
                }
                return JSON.serialize(result);
        }
         return JSON.serialize(result);
}

@AuraEnabled
public static Boolean updateMarketing(String userId, Boolean marketingFlag){
        try {
                String contactToUpdate = [SELECT ContactId FROM User WHERE Id=:userId].ContactId;
                Contact contact = [SELECT Id,Marketing_Flag__c,MarketingConsentDate__c FROM Contact WHERE Id=:contactToUpdate];
contact.Marketing_Flag__c = marketingFlag;
                contact.MarketingConsentDate__c = Date.today();
                update contact;
                return true;
        } catch (Exception e) {
                return false;
         }
}

@AuraEnabled    public static String changeUserEmail(String userId, String newEmail){
        if(userId==null){
                 return 'Error: Invalid UserId';
        }
        if(!Pattern.matches('^[_a-z0-9-]+(.[a-z0-9-]+)@[a-z0-9-]+(.[a-z0-9-]+)*(.[a-z]{2,4})$', newEmail.toLowerCase())) {
                return Label.email_invalid;
        }
         List<User> u = [SELECT Email FROM User WHERE Id=:userId];
        if(u.size()==0){
                return 'Error:User not Exists';
        }
        User myUser = u[0];
        Decimal checkRepetitions = [SELECT Count() FROM User WHERE Email =:newEmail];
        if(checkRepetitions> 0) {
                return Label.email_already_used;
        }
        myUser.Email = newEmail;
        update myUser;
        return 'ok';
}
```

Code 6: Check Inventory

APEX FUNCTION

```
// This must implement the sfdc_checkout.CartInventoryValidation interface
// in order to be processed by the checkout flow and used for your Check Inventory integration.
public with sharing class EPTAB2BCheckInventorySample   {

   public static Map<String,String> startCartProcessAsync(List<RefrigeratorToCheck> quantitiesFromSalesforce, String country, Boolean
isCheckOutStage) {
      Map<String,String> returnValue = new Map<String,String>();
      returnValue.put('Error',null);
      try {

         //prepare final http request body;
         HttpRequestObject requestBody = new HttpRequestObject(country,isCheckOutStage,quantitiesFromSalesforce);

         // Get all availabilities.
         Result quantitiesFromExternalService = getAvailabilityFromExternalService(requestBody);


         for(ResultSku result : quantitiesFromExternalService.sku ){
            //get all products answers
            if ( (String) result.available == null){
               String errorMessage = 'The product with sku ' + result.sku + ' could not be found in the external system';
               returnValue.put('Error',errorMessage);
               return returnValue;
            }else if (((String) result.available).toLowerCase() == 'block'){
               //String errorMessage = 'Insufficient quantity for the product with sku ' + result.sku + '. The product must be Blocked';
               returnValue.put(result.sku,'Block');
               //return answer;
            }else if(((String) result.available).toLowerCase() == 'false'){
               //String errorMessage = 'Insufficient quantity for the product with sku ' + result.sku ;
               //return answer;
               String ans = 'False_'+result.skuAvailableDate;
               returnValue.put(result.sku,ans);
            }else if (((String) result.available).toLowerCase() == 'true'){
               // If the product exists and the available quantity is enough, set status as SUCCESS
               String ans = 'True_'+result.skuAvailableDate;
               returnValue.put(result.sku,ans);
            }

         }
         returnValue.put('OrderAvailableDate',(String)quantitiesFromExternalService.OrderAvailableDate);

      } catch(Exception e) {
         // For testing purposes, this example treats exceptions as user errors, which means they are displayed to the buyer user.
         // In production you probably want this to be an admin-type error. In that case, throw the exception here
         // and make sure that a notification system is in place to let the admin know that the error occurred.
         // See the readme section about error handling for details about how to create that notification.
         returnValue.put('Error','There was an exception '+e);
         return returnValue;
      }
      return returnValue;
   }

   private static Result getAvailabilityFromExternalService (HttpRequestObject requestBody) {
      Http http = new Http();
      HttpRequest request = new HttpRequest();
      Integer SuccessfulHttpRequest = 200;




      //this is the mock
      HttpResponse response = new HttpResponse();
      Result r;
      if(Test.isRunningTest()){
         if(requestBody.sku[0].sku == 'A1111'){
            r=mockResult(requestBody);
         }else if(requestBody.sku[0].sku == 'A2222'){
            r=mockResultFalse(requestBody);
         }else if(requestBody.sku[0].sku == 'A3333'){
            r=mockResultBlock(requestBody);
         }else if(requestBody.sku[0].sku == 'A4444'){
            r=mockResultError(requestBody);
         }
         response.setBody( JSON.serialize(r));
         response.setStatusCode(SuccessfulHttpRequest);
      }else{
         //r = mockResult(requestBody);

         //this is the real request;
         //Get MiddleWare Settings
         MiddleWareSettings__c theMiddleWare = [SELECT Inventory_Endpoint__c, Inventory_Key__c FROM MiddleWareSettings__c];
```

```
public static Result mockResult(HttpRequestObject requestBody){
   List<ResultSku> resList = new List<ResultSku>();
   for(RefrigeratorToCheck ref : requestBody.sku){
      ResultSku rss = new ResultSku();
      rss.sku = ref.sku;
      if(ref.sku != null){
         rss.available = 'True';
         rss.skuAvailableDate = '27/12/2021';
      }
      resList.add(rss);
   }
   Result res = new Result();
   res.sku = resList;
   res.OrderAvailableDate = '27/06/2021';
   return res;
}

public static Result mockResultFalse(HttpRequestObject requestBody){
   List<ResultSku> resList = new List<ResultSku>();
   for(RefrigeratorToCheck ref : requestBody.sku){
      ResultSku rss = new ResultSku();
      rss.sku = ref.sku;
      if(ref.sku != null){
         rss.available = 'False';
         rss.skuAvailableDate = '27/12/2021';
      }
      resList.add(rss);
   }
   Result res = new Result();
   res.sku = resList;
   res.OrderAvailableDate = '27/06/2021';
   return res;
}

public static Result mockResultBlock(HttpRequestObject requestBody){
   List<ResultSku> resList = new List<ResultSku>();
   for(RefrigeratorToCheck ref : requestBody.sku){
      ResultSku rss = new ResultSku();
      rss.sku = ref.sku;
      if(ref.sku != null){
         rss.available = 'Block';
         rss.skuAvailableDate = '';
      }
      resList.add(rss);
   }
   Result res = new Result();
   res.sku = resList;
   res.OrderAvailableDate = '';
   return res;
}

public static Result mockResultError(HttpRequestObject requestBody){
   List<ResultSku> resList = new List<ResultSku>();
   for(RefrigeratorToCheck ref : requestBody.sku){
      ResultSku rss = new ResultSku();
      rss.sku = ref.sku;
      if(ref.sku != null){
         rss.available = null;
         rss.skuAvailableDate = '';
      }
      resList.add(rss);
   }
   Result res = new Result();
   res.sku = resList;
   res.OrderAvailableDate = '';
   return res;
}

public class HttpRequestObject{
   public String salesOrg;
   public Boolean isCheckOutStage;
   public List<RefrigeratorToCheck> sku;

   public HttpRequestObject(String country, Boolean isCheckOutStage, List<RefrigeratorToCheck> sku){
      this.salesOrg = country;
      this.isCheckOutStage=isCheckOutStage;
      this.sku = sku;
```

Code 7: Calculate Shipping Cost

APEX FUNCTION

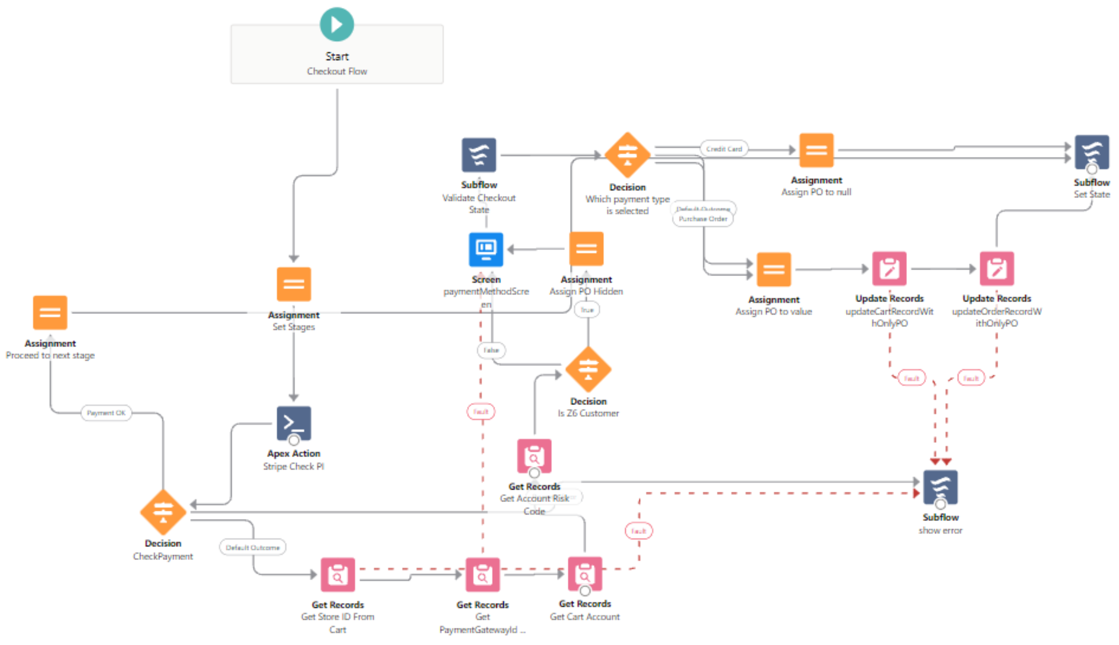
```
public with sharing class CreateDeliveryServicesCartItems {
   @InvocableMethod(label='Create delivery services cart items' description ='Create delivery services cart items')
   public static void createDeliveryServicesCartItems(List< DeliveryServicesInput> input) {
      List<String> productsId = input[0].services;
      if(productsId.size()>0){
         Boolean hasTime = false;
         Boolean hasDate = false;
         //String communityId = Network.getNetworkId();
         List<ShippingAddressScreenController.Service> allServices =
ShippingAddressScreenController.getDeliveryServices(input[0].accountId,input[0].cartId);
         List<CartItem> cartItemToCreate = new List<CartItem>();
         List<String> possibleServices = new List<String> ();
         for (ShippingAddressScreenController.Service ser : allServices){
            //add product ids of services
            possibleServices.add(ser.Id);
         }
         if(Test.isRunningTest()){
            //add a fake service just for testing purproses
            String id = CreateDeliveryServicesCartItemsTest.fakeDel.Id;
            ShippingAddressScreenController.Service myfakeService = new ShippingAddressScreenController.Service (id,'fake service','this is a
fake service');
            myfakeService.UnitPrice = 10;
            myfakeService.TotalPrice = 20;
            myfakeService.Npieces = 2;
            allServices.add(myfakeService);
         }

         //delete all previous delivery services
         List<CartItem> oldService = [SELECT Id FROM CartItem WHERE CartId=:input[0].cartId AND Product2Id IN :possibleServices];
         delete oldService;
         List<String> skus = new List<String>();
         String skusAsString = [SELECT ShippingServices__c FROM b2bconfiguration__c].ShippingServices__c;
         skus = skusAsString.split(',');
         List<Product2> productsNewList = [SELECT Name FROM Product2 WHERE Accessory_Type__c = 'Shipping service' and ProductCode IN
:skus];
         for(ShippingAddressScreenController.Service service : allServices){
            if(productsId.contains(service.Id)){
               Product2 prod;
               for(Product2 p : productsNewList){
                  if(p.Id == service.Id){
                     prod = p;
                     break;
                  }
               }
                //= [SELECT Name FROM Product2 WHERE Id=:service.Id];
               CartItem cti = new cartItem(Name = prod.Name, cartDeliveryGroupId = input[0].cartDeliveryGroupId, AdjustmentAmount =
0,CartId = input[0].cartId,Product2Id = service.Id, Quantity = service.Npieces,SalesPrice = service.UnitPrice, TotalLineAmount =
service.TotalPrice,TotalPrice= service.TotalPrice,Type='Product');
               cartItemToCreate.add(cti);
               if(service.hasTime == true){
                  hasTime = true;
               }
               if(service.hasDate == true){
                  hasDate = true;
               }
            }

         }
         insert cartItemToCreate;
         if(hasDate == true){
            CartDeliveryGroup cdg = [SELECT DesiredDeliveryDate,HasDeliveryTime__c,HasDeliveryDate__c FROM CartDeliveryGroup WHERE
Id=: input[0].cartDeliveryGroupId ];
            cdg.HasDeliveryDate__c = true;
            if(hasTime == true){
               cdg.HasDeliveryTime__c = true;
               String theTime = input[0].deliveryTime;
               Date dtToday = Date.today();
               Integer h = Integer.valueOf(theTime.split(':')[0]);
               Integer m = Integer.valueOf(theTime.split(':')[1]);
               Time t = Time.newInstance(h,m,0,0);
               Datetime myDT = DateTime.newInstance(dtToday,t);
               TimeZone tz = UserInfo.getTimeZone();
               Integer offset = tz.getOffset(myDT)/3600000;
               myDT = myDT.addHours(offset);

               cdg.DesiredDeliveryDate = myDT;
            }
            update cdg;
         }
      }

   }
```

Code 8: Calculate Taxes

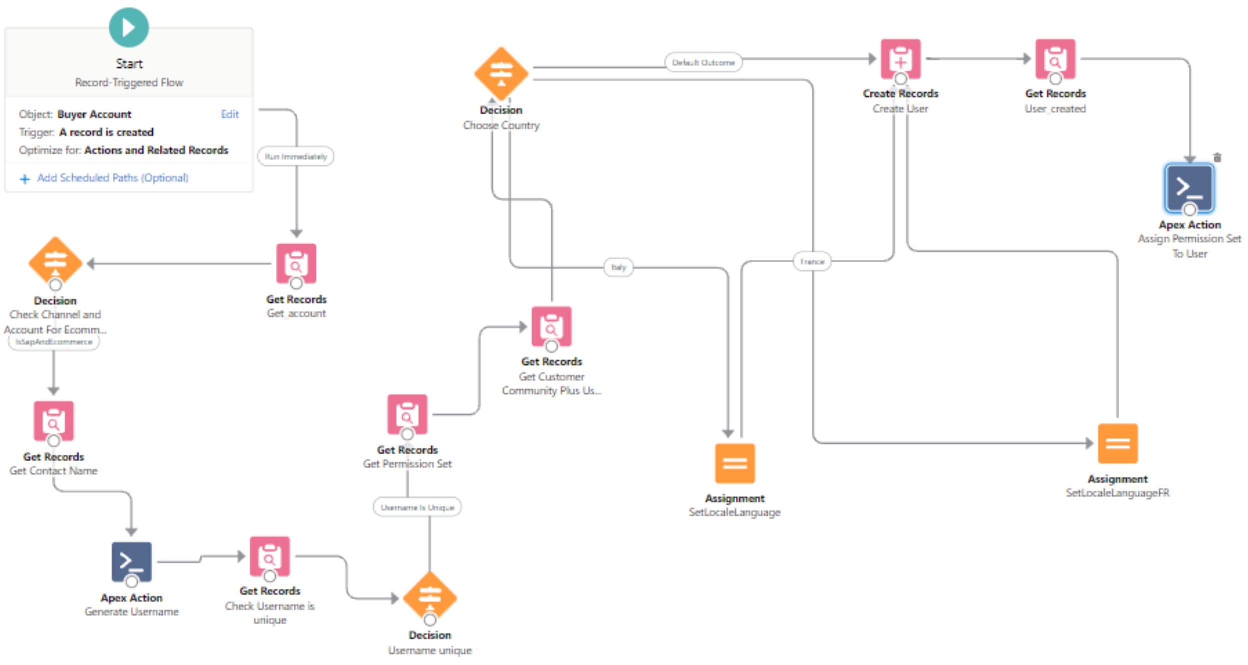
APEX FUNCTION

```
public with sharing class EPTAB2BTaxSample {
	@InvocableMethod(label='EPTA Calculate taxes' description='Calculate and create taxes within cart')
	public static List<String> createCartTaxes(List<CalculateCartTaxesInput> input) {
		List<CartTax> taxdelete = [SELECT Id FROM CartTax WHERE CartId = :input[0].cartId];
		delete taxdelete;
		//Get Org defualt taxations
		List<CartTax> cartTaxestoInsert = new List<CartTax>();
		Taxation__c orgTax = Taxation__c.getValues(input[0].country);
		Decimal taxRate;
		Decimal taxAmount;
		//Define tax rate
		 taxRate = orgTax.Tax__c/100;
		// Get all SKUs, the cart item IDs and product information, and the total prices from the cart items.
		for (CartItem cartItem : [SELECT Id,CouponDiscount__c,Name,Sku,TotalPrice,Type FROM CartItem WHERE CartId =
		:input[0].cartId WITH SECURITY_ENFORCED]) {
			String cartItemSKU = '';
			if (cartItem.Type == 'Product') {
				cartItemSKU = cartItem.Sku;
			}else if (cartItem.Type == 'Charge') {
				cartItemSKU = 'ChargeSKU';
			}
			//Calculates taxes amount
			taxAmount = (cartItem.TotalPrice - cartItem.CouponDiscount__c) * taxRate;
			//Define CartTax values
			CartTax tax = new CartTax( Amount = taxAmount, CartItemId = cartItem.Id, Name = 'Tax ' + cartItem.Name,
		TaxCalculationDate = Date.today(), TaxRate = taxRate, TaxType = 'Actual' );
			cartTaxestoInsert.add(tax);
			System.debug(cartTaxestoInsert);
		}
		try {
			insert(cartTaxestoInsert);
		} catch(Exception e){
			System.debug(e.getMessage());
		}
		return new List<String>{'Taxes correctly created'};
		}
```

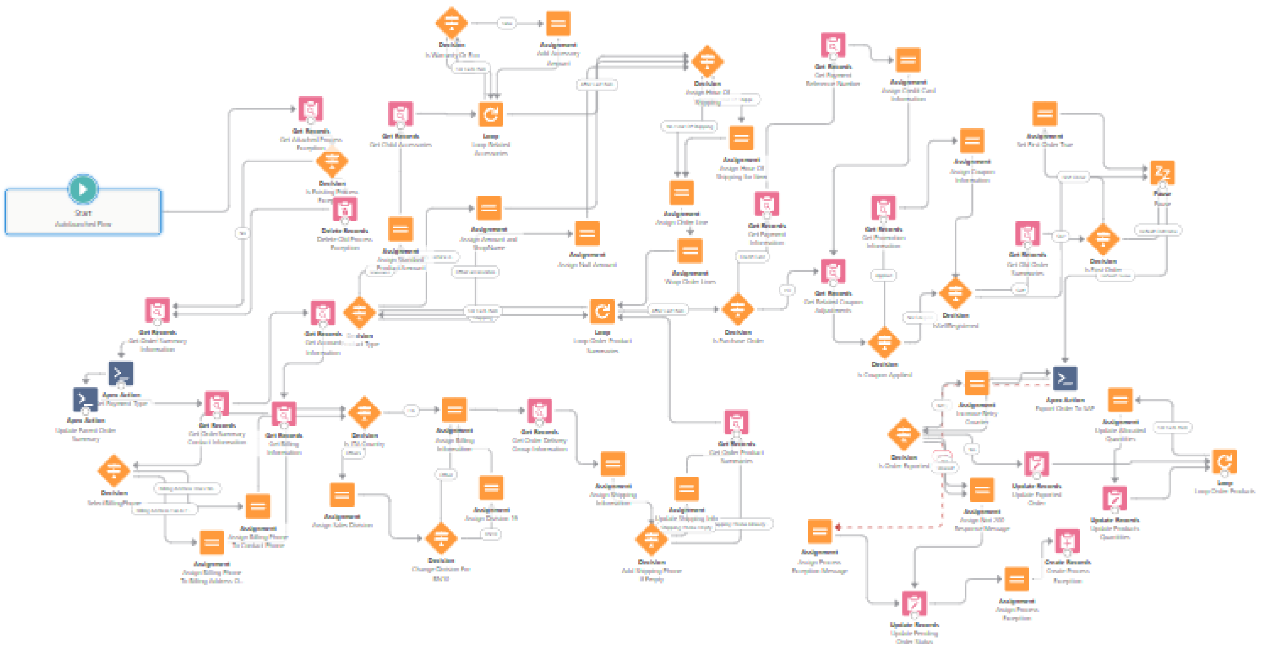
Flow 1: Payment Flow



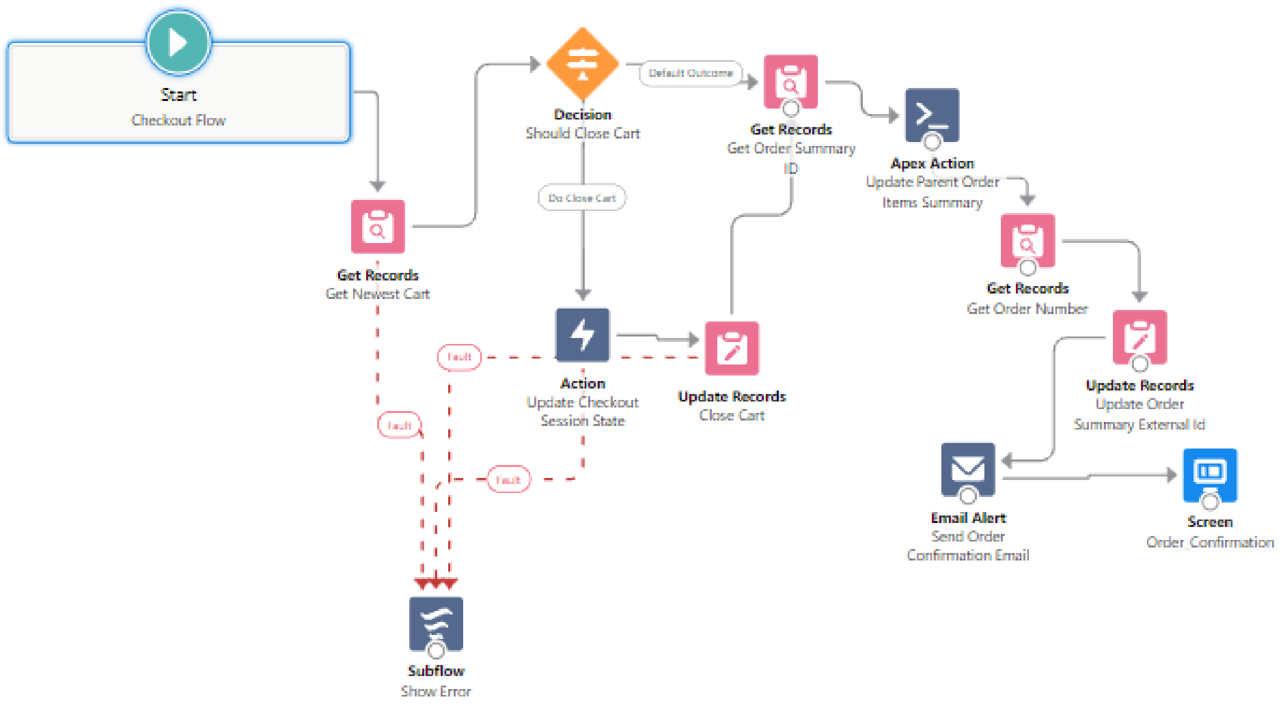
Flow 2: User Import Flow



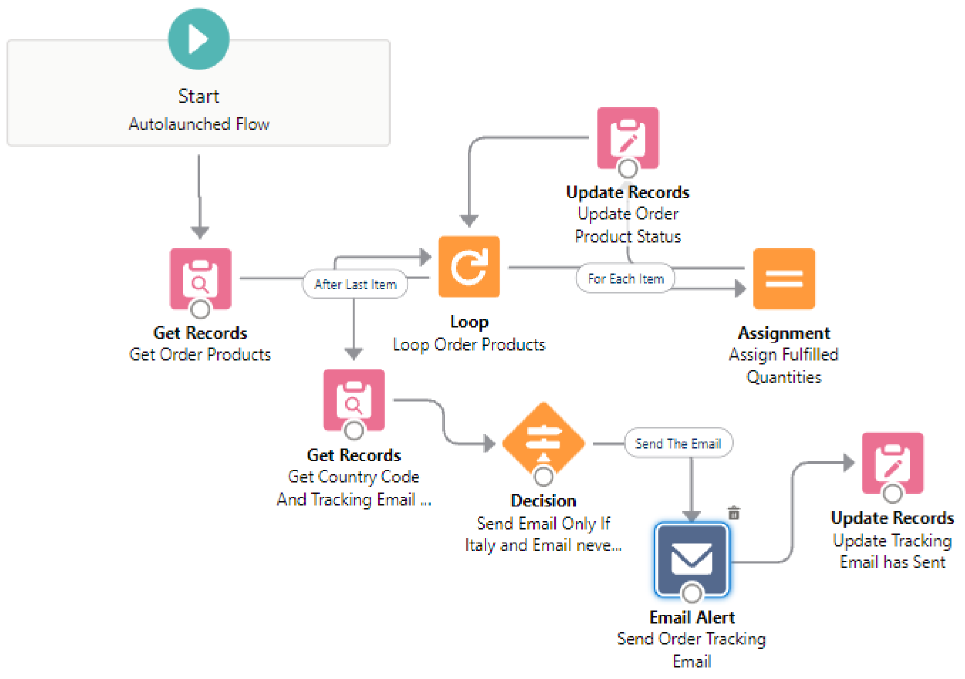
Flow 3: Order Export



Flow 4: Order Summary



Flow 5: Update Fulfilled Orders


Start
Autolaunched Flow
Get Records
Get Order Products
After Last Item
Loop
Loop Order Products
Update Records
Update Order Product Status
For Each Item
Assignment
Assign Fulfilled Quantities
Get Records
Get Country Code And Tracking Email ...
Decision
Send Email Only If Italy and Email neve...
Send The Email
Email Alert
Send Order Tracking Email
Update Records
Update Tracking Email has Sent

BIBLIOGRAPHY

Retrieved from Salesforce Official Website: https://www.salesforce.com

Arnau, B.-A., Liz, H., Dennis, S., & Jennifer, S. (2020). *New analysis makes it clear: For B2B sales, digital is the wave of the future.*

Bezos, J. (2013, 09 25). Jeff Bezos: A Down-to-Earth CEO Reaching for the Stars. (J. STERN, Interviewer)

John, B., Susan, W., Charlie, R., & Rachel, B. (2019). *US B2B eCommerce Will Hit $1.8 Trillion By 2023.* FORRESTER.

Kelsey, S., & Pashmeena, H. (2015). *The Changing Face of B2B Marketing.* Google.

what-is-b2b-marketing. Retrieved from Salesforce: https://www.salesforce.com/it/learning-centre/marketing/what-is-b2b-marketing/