POLITECNICO DI MILANO

School of Industrial and Information Engineering
Master of Science in Computer Science and Engineering
Dipartimento di Elettronica, Informazione e Bioingegneria

Re-platforming operation of the eCommerce web portal for a retail global enterprise

Supporting the rollout of new countries in the company's Omni-channel program by developing new features and exporting existing ones from SiteGenesis onto the StoreFront Reference Architecture

Relatore
Prof. Pierluigi Plebani

Laureando
Fabrizio Schembari
matr. 912892

Fabrizio Schembari

Sommario

Il seguente lavoro di Tesi ha lo scopo di descrivere gli aspetti generali ed effettuare analisi in ambito tecnico riguardo l'esperienza lavorativa acquisita durante un tirocinio nell'ambito della consulenza aziendale.
Il progetto in questione — seguito come back-end developer del team di Deloitte Digital — era parte del programma di Omni-channel per una multinazionale di vendite retail.
Il suo focus era diretto su due topic in particolare: l'operazione di re-platforming dell'infrastruttura esistente dall'architettura SiteGenesis alla (più recente e performante) StoreFront Reference Architecture, e l'estensione di tale rinnovato eCommerce su due nuove country dell'Asia meridionale, costituenti le prime localizzazioni di rollout nell'ambito dell'intero progetto.
Questo documento descrive l'argomento in ogni sua sfaccettatura, includendo la pipeline relativa alla gestione, le relazioni col cliente o i provider dei vari servizi esterni e alcuni esempi pratici delle attività comprese nel software engineering.
Nel fare ciò, il paper fornirà al lettore una vista a tutto tondo sulla realtà di team working in tale ambito sia da un punto di vista tecnico che strategico, e assumerà sia il punto di vista del team sia quello del singolo developer nello spiegare come si sia arrivati al raggiungimento degli obiettivi prefissati.

Fabrizio Schembari

Abstract

The following document has the purposes of describing the general aspects and analyzing the technical facets connected to the work experience gathered during an internship in business consulting.
The project of matter — followed as a back-end developer hired by Deloitte Digital — was part of the Omni-channel program for a B2C worldwide retail enterprise.
It focused mainly on two primary topics: the re-platforming of the already present infrastructure from SiteGenesis architecture onto the (most recent and better performing) StoreFront Reference Architecture, and the extension of such all-new eCommerce to two south-Asian countries, representing the first locales performing a rollout in the scope of the entire project.
This paper will describe such topic in all its aspects, including the pipeline through which it was managed, the interface with the business client or the external services' providers and a few examples of the actual activities involved in the software engineering.
In doing so, the document will provide the reader with an all-round insight on the reality of team working in such business from a technical and executive point of view, and will assume both the team perspective and the point of view of a single developer to better explain how the forecast goals were reached.

Fabrizio Schembari

Contents

1. Introduction

1.1. The project in a nutshell

The following Thesis work is centered on the business consulting work experience gathered during an internship in Deloitte Digital.
During these months, the team followed a sub-part of an already launched and long-lasting project: the global transformation of a B2C retail global enterprise, through which the firm wanted to provide an Omni-channel experience for the whole group.

To achieve the awaited results, the client company started the rollout of a new Commerce within some central Europe countries, such as Italy; however the adopted solution soon showed a few weak points concerning the flexibility and scalability of its backbone, SiteGenesis infrastructure, factor that lowered the performance of the platform and the expected rollout speed of the entire project.

From this need comes the engagement with a Partner such as Deloitte which offered, together with a proactive consultancy approach, to develop the integration with the state-of-the-art *StoreFront Reference Architecture* in place of SiteGenesis, plus a few other measures (such as the *adoption of a PIM system* — more on that later) that would imply a relevant speedup factor in the overall Omni-channel program.
And in this context happens to be my involvement in the team, located in a phase of the overall project in which the focus was mainly to work on the rollout of the new Commerce on two south-asian countries — India and Malaysia to be more specific.

To perform this task, the goal was to develop the SFRA integration of the currently existing architecture — in other words, to perform a re-platforming of the existing infrastructure from SiteGenesis to StoreFront Reference Architecture —, and applying the necessary customizations that each of these two countries required to launch their all-new storefronts and use such countries as a starting point for the overall rollout plan.

All in all, the Indian country — that was in a past-halfway state of development at my arrival — ended up being a very customized version of the SFRA storefront, instead the requirements on the Malaysian locale allowed us to construct it as a canvas for the default case, thus maintaining the aim of building everything with scalability in mind, and ease the upcoming transition of more and more countries onto such new architecture.

1.2. The structure of the document

To perform the descriptive task introduced, the following paper is organized in a few sections:

- The **technology background** chapter will describe all the systems involved in a project of such entity, explaining their roles, a few of their interactions, and the tools that have been used in the whole project;
- The **starting point** chapter will provide an insight on the previous situation, with an overview of the SiteGenesis architecture and the negativities that brought the client firm to embark in this re-platforming operation;
- The **goal** chapter will then describe the SFRA architecture, compare it more in detail to SiteGenesis and illustrate the improvements that such adoption brought, together with the introduction of a PIM system;

- The **project** chapter will finally provide:
  - A few information on the pipeline through which the project was carried, from a team point of view;
  - An insight on the roles covered by the generic team member, together with a few examples of activities portraying a few of my contributions in reaching the goal;
- Finally, a **conclusion** will highlight my personal point of view, what I learnt during these months of internship and the skillset that this experience allowed me to develop.

2. The technology background

2.1. Preliminaries: retail market and trends

It's no secret that the trend in eCommerce sales has seen them growing steadily over the last years; moreover it is expected to continue growing over a few of the next ones — though not at a rate as fast as before (*Figure 1*).
And retail industry makes no exception, in a context of growing availability of technology and home-delivery services (and most recent strikes of worldwide pandemics imposing home quarantining).

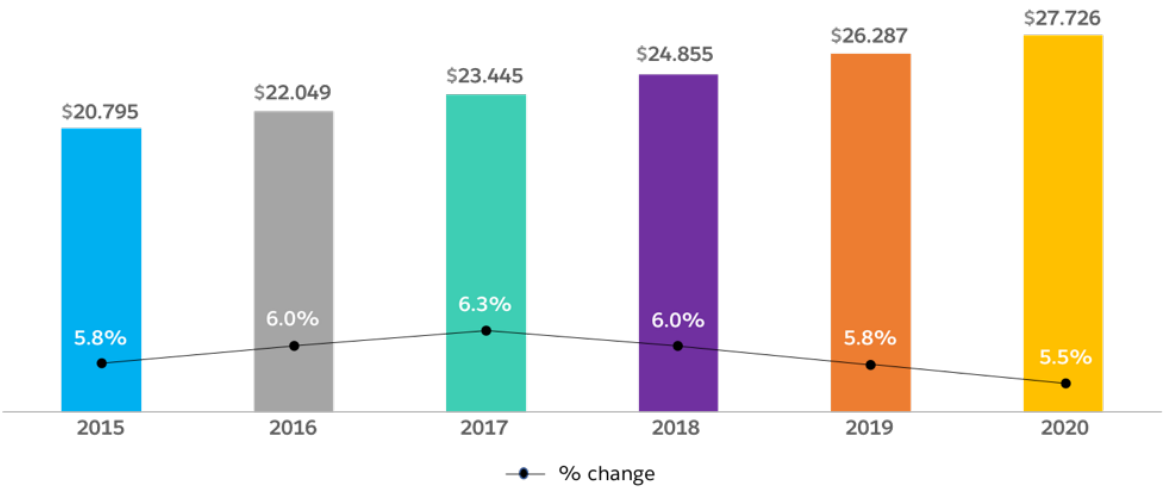

Figure 1: Total retail sales worldwide — travel and event tickets excluded —, 2015-2020 (trillions)
Source: eMarketer, Aug 2016

In this scenario, there are some major industry trends:

1. ***The connected customer***: such figure represents how the ideal buyer is changing and is substantially transforming the fruition of retail services. The connected customer is social, connected and smart, but still interested in having a spotless in-store shopping experience.
   In terms of proportions, nowadays it has been estimated that more than 62% of eCommerce traffic comes from mobile devices, with a share of orders growing 41% year after year. And more than 50% of retail orders still belongs to physical stores, but more than half of such commerce is heavily influenced by the digital environment around, populated of different channels (social media, blogs, online retail) over different types of devices (mobile, tablet, desktop).
   This situation produces in the customer an expectation so high that only a few brands can keep up with it, leaving most of commerce teams struggling with:
   - Diverse and duplicative systems: each touchpoint with the customer is often managed through a specific technology that's not always integrated with any of the others present;
   - Customer data scattered across these many systems: this prevents business groups such as marketing, customer service, and merchandising from having access to the data they need, or simply makes it difficult to know where to find it;
   - Multiple sources of truth: retailers have no single view of the customers, products, prices, promotions, and so on;
   - Complex back-end integration: when retailers are able to integrate data across their multitude of systems, such integrations tend to be inflexible, frequently breaking down whenever there is a software update or an upgrade across any of the endpoints.

2. **Omni-channel and Unified Commerce**: The already used term "*Omni-channel*" generally refers to the practice of creating a consistent buying experience across all channels available to a company — mobile, in-store, social; "*unified commerce*" takes things one step further, since it focuses on the customer's entire commerce experience, from marketing to shopping, custom product

recommendations and customer service with the associated marketing and retargeting.
It's basically the idea that brands must think beyond individual retail channels and beyond shopping to meet the customer's expectations for personalized attention, relevant product recommendations, responsive customer service, and more.

These factors are the ultimate motivation that makes retailers such as the client firm want to upgrade, acknowledge the need of technology and provide a unified commerce experience that can support them for a long-term benefit, increased margins, revenues, and brand value.

2.2. The involved entities

Of course, a project of a substantial size such as the one depicted by this document requires many entities to cooperate. These different systems are entrusted with managing each one the build-up of a subset of the entire platform, handling a few of the large amount of the overall data, exchanging the necessary ones with each other and having different responsibilities.
What all of them have in common is the same objective: making such environment work in harmony and as flawlessly as possible.

In this scenario, the main systems we distinguish are:

- The **business client** — the retail enterprise;
- The **service provider Partner**, providing services regarding:
    - the Commerce implementation,
    - the middleware implementation,
    - the application maintenance,
    - the InStoreApp implementation;

- The **Customer Relationship Management** system;
- The **Enterprise Resource Planner**;
- The **Order Management System**;
- The **Product Information System**;
- The **Payment Service Provider**;
- The **Point of Sale** system;
- The **Warehouse Management System**;
- The **Email Service Provider**;

In the following section, an overview of each of these systems will be given.

2.2.1. Service provider Partner: **Deloitte.**

As already said, the consulting firm that hosted me for the whole internship experience and that was in charge of managing the whole project is Deloitte.

In the relationships with the client, Deloitte has a Partner role, meaning that we are not just talking of a commissioned work of system integration; instead, the firm is meant to keep a consulting approach — according to which keep performance in mind and always suggest the best solution — and it's entrusted with the whole Omni-channel operation, covering the extension to new countries, to all platforms (such as the InStoreApp together with the eCommerce platform) and within all the steps of its existence (including the maintenance of the applications on the already running countries).

But what is Deloitte?

Deloitte Touche Tohmatsu, commonly known as Deloitte, is a multinational professional services network founded by William Welch Deloitte in London, England, in 1845 and today having headquarters in London and bases in over 150 countries around the world.

Deloitte is one of the Big Four accounting organizations (together with KPMG, PwC and Ernst & Young) and it's actually the global leader over its competitors according to revenues (US$ 47.6 billion in aggregate revenues in fiscal year 2020, setting a record, and constantly growing each FY) and number of professionals (estimated around 312,000 professionals around the globe in fiscal year 2020).

Deloitte operates providing services in different areas: Audit, Tax, Financial Advisory and *Consulting*.
The branch of interest for the purpose of this document is actually the last one, in which the firm operates providing services around:

- Strategy and management
- IT consulting
- Supply chain management
- Systems integration
- Outsourcing
- Human Resource

And it is in this environment that the **Deloitte Digital** and **MuleSoft** teams have their own spot.

2.2.1.1. Commerce team (Deloitte Digital)

The branch of Deloitte that actually hosted my internship is Deloitte Digital, Deloitte's Creative Digital Consultancy. More specifically, in the scope of the project I belonged to the Commerce team.

This division is entrusted with the task that represent the main topic of this work of Thesis: the buildup of the Commerce onto the SalesForce StoreFront Reference

Architecture, mission that includes the re-platforming of the existing services and the customizations of the new platform to bring new countries to the rollout of their state-of-the-art eCommerce in the Omni-channel plan.

2.2.1.2. Enterprise Service Bus (MuleSoft Integration Services Architecture)

As the title suggests, MuleSoft is an integration platform belonging to SalesForce to which the **middleware** was delegated in the context of the project.
In a scenario where different entities send the data they're requested in different formats — not necessarily compatible with what their receiver expects — some sort of decoupling is necessary: for this purpose, such software was managed by a specialized unit belonging to Deloitte that was in charge of implementing the bridge node between all these flows, connecting the external systems with each other (*****) or with the storefront.
This means that such team was entrusted with building the integration layer among the various systems, decoupling their data and controlling the information flow, making sure that each of the involved entities receives the data in the format they expect them, even though the sender sent them under another shape.

(*****) With the update of the Commerce, a few of the flows in which two systems communicate directly or aggregate data with each other before transmitting them to the commerce are:

- During the **update on warehouse and store inventory** the data periodically propagated from the ERP gets aggregated by the OMS and then forwarded to the Middleware that sends it to SFCC;
- Also the **import of the product Master** starts from the ERP, then data gets elaborated by the PIM before getting to the commerce, where it's managed

2.2.1.3. InStoreApp team (Deloitte Digital)

The omni-channel plan expects from Deloitte, among the other functions, to build and take care of the InStoreApp together with the eCommerce.
And such function has a dedicated team, another fragment of Deloitte Digital which is entrusted with the management of everything that concerns the InStoreApp: this means build it where it's not present, keeping it up-to-date, monitoring that everything that's already built works as expected and extending its functionalities as a limb of a broader omni-channel system with a worldwide commerce scope.

2.2.1.4. Application Maintenance (Deloitte Digital)

Finally, to cover all the duties held by Deloitte, even though this is not actually one of the "systems" involved in the project, it's important to name the AM: another team internal to the Deloitte Digital division, whose role is basically monitoring that everything on the storefronts still on SiteGenesis (or also on SFRA for the locales that go live with it) runs smoothly and satisfy eventual late needs of the client on such platforms.

2.2.2. The external systems

2.2.2.1. CRM - Customer Relationship Management

The Customer Relationship Management (or CRM) is a tool that's used by the client firm, as well as many other companies, to keep track of consumers that are already scheduled and potential new ones.
The software has the aim of improving the efficiency and effectiveness of the customer relationship management; however this is not the only goal that it can be achieved, since CRM can help deepening relationships also with colleagues, suppliers and partners.

This system is basically a cloud-based loyalty rewards platform, a database that keeps track of the records about customers' contact information (e-mail, telephone, website social media profile and more) and loyalty data within the Commerce. However, it that can also pull in data on recent news about the company's activity and details such as the clients' personal preferences on communications. Moreover, it is provided with AI technologies which, together with tips from the customers themselves, is used to build complex analytics about them.

So the main focus of this type of system is to centralize the data the business has access to, but also to digitize the processes and automatize the tasks of providing the customers with the targeted experience they would expect, hence granting an improved fidelity towards the firm.

If we want to categorize the beneficial impacts of a CRM, we find that they're related to:

- *Sales*: sales managers can monitor the progress of the individual team members in achieving their sales targets and how well individual sales teams, products and campaigns are performing;

- *Marketing*: since data from customers' social media activity — their likes, dislikes and sentiment about specific brands and businesses — is available, forecasting is made simpler and more accurate;
- *Customer service*: customers' interactions don't have a common platform (an issue can be raised on a channel, say a social media, but then switch to another platform as phone, chats or e-mails), so the centralization of all the contact data the consumers provide ensures to avoid — or at least limit — missing interactions or losing communications in the flood of information (which, from the business point of view, can bring to unsatisfactory response to potentially valued customers);
- *Supply chain, procurement and partner management*: the relationships with other entities involved in the business can benefit from a CRM since it's able to track meetings or requests made and add to them useful notes that can help comparing the efficiency of suppliers, and more generally managing the entire supply chain;
- *HR*: this function is improved by speeding up the on-boarding process, mostly through the automation of the process of managing candidates, analyzing needs, identifying skill gaps and supporting the pursuit of staff retention targets.

2.2.2.2. ERP - Enterprise Resource Planner

The Enterprise Resource Planner, or ERP, is another among the most important tools used by many companies in the B2B and B2C businesses.
Its functions, in the specific case of the client firm, are related to products and sales, and they have the ultimate task of collecting and aggregating information to provide the company with statistical insight and let it draw conclusions for the decisions to take.

Concerning the first point, the ERP is basically in charge of constructing a database with the products when they get created. Their creation takes place with their "raw", a profile that includes the very least amount of data, such as an ID, the UPC code — the barcode — and a few information such as its English name; with the updates of the client firm within the project this database is meant to be then enriched through the intervention of a PIM.
Moreover, the data aggregation the ERP performs is also related to purchase and selling prices, allowing the company to perform inference on the revenues related to each and every product.

Then, talking about sales, it works strictly in contact with OMS when an order is created to record the actual sales and, yet, performing statistic inference for decision making.

2.2.2.3. OMS - Order Management System

In an eCommerce system, even in the most complex ones, where the user expects absolute ease of use and transparency in the whole order placement experience nevertheless, there's for sure the need of advanced order management capabilities.

In this scenario, the Order Management System — or OMS — is the system involved in performing this task: in a retail business such as the one assessed by this document, it is given the objective of:

- Managing all the phases of an order while it gets created (including the interface with payment handlers, credit cards and billing), canceled or returned (with the subsequent refund);
- Handling the whole processing of such orders after they are placed (so all the phases such as selection, invoices' printing, picking, packing or shipping). This point shows how the OMS is involved in the sourcing (once a product is ordered, finds the closest logic node of the business network to the client) and fulfillment logics (how the shipment actually takes place, from the store/central warehouse to the interface with the carriers);
- Keeping track of the existing orders for all the customers;
- Synchronizing orders that tackle different channels separately (aggregating store orders to eCommerce ones) or together (as for orders placed online but set to pick up in stores — feature called *Click & Collect*),
- Managing the inventory (keeping track of the stocks belonging to every store as well as to the central warehouse as an ensemble),

So of course considering the retail enterprise of matter, it has a thorough integration with all the systems, since it works strictly in contact with:

- the ERP (to register the sales);
- the WMS (for the availability of products according to the stock levels);
- the PSP (for the payments);
- the POS (to produce the invoices);
- the CRM (for what concerns the customers' info and loyalty data);

- and the PIM in the upgraded scenario (for the data related to products, catalogs and pricing).

Finally, the operation it performs require multiple steps, since they are included in a workflow that involves validation, fraud check, payment authorization and the whole backorder management, together with the associated customer communications.

2.2.2.4. PIM - Product Information Management

The Product Information Management system — or PIM — is an entity needed by the biggest B2C and B2B firms that sell products through a big variety of sales channels (including an eCommerce platform) in order to effectively manage all the information required to market and sell through such distribution channels.
If the ERP is in charge of creating the raw products, with very few data on them, the PIM constitutes a central hub where to keep all the additional information about such products, together with categories, catalogs and assets (meaning media, brands and so on).
So in many cases, as for the project concerned, PIM functionalities are delivered by platforms designed to work also as Content Management Systems (or CMS), which takes care of all the media and content through which all items in the eCommerce are enriched.

It has already been stated that, in the context of the business of matter, the PIM system was not present before the engagement with Deloitte, and it was indeed one of the potential improvements detected, together with the adoption of SFRA as underlying architecture, at the beginning of the business relationships; more detail on this will be given in the **goal** chapter later on in this document.

2.2.2.5. PSP - Payment Service Provider

The Payment Service Provider — or PSP — is the software in charge of the transactions of the customers, a central hub used in many systems to integrate several or all the payment networks or bank circuits allowed by their Commerce and manage the sales through them.

In practical terms, in the context of the eCommerce buying experience, it's the environment in which the customer is redirected once they place an order, that is the very moment in which they confirm their commitment into proceeding to buy. This means that it's devoted mainly to assist the customer along the payment, making sure that it makes it from point A to B safely and securely.

However, it is also generally capable of performing functions related to:

- *Sales reporting*:
    - Generate detailed sales reports, based for example on products, employees, total cost of items sold and their retail amount, net profits and percentages;
    - Provide quick snapshots and charts on stores' sales performances;
- *Customer management*:
    - Attach a sale/transaction to a customer;
    - Keep track of the customers' purchase history;
    - Capture customers' information (name, age, birthday, phone number and email address);
    - Use email marketing to keep in touch with customers.

A few examples of PSP systems for the purpose of this document — so by focusing mainly on south-Asian countries — are *Adyen* and *CCAvenue*.

2.2.2.6. POS - Point Of Sale

In simple terms, the Point of Sale system is the software present within all the cashes in the various retail shops.
This tool, apart being accessed by the stores' salesmen to perform payments and complete retail sales, can be used for the interest of the eCommerce storefront, as it allows to *prepare the invoices* for the customers — that are basically cash register printouts — and *handle coupons and gift cards' creation and validation*, since dedicated offers for specific customers are allocated starting from cashes themselves.

2.2.2.7. WMS - Warehouse Management System

The Warehouse Management System, as the name suggests, is a software application designed to support and optimize warehouse functionality and distribution center management.
So its duties make it work hand in hand with systems such as the ERP or the OMS, since it's devoted to:

- handle the monitoring of stock levels and changes to the data, controlling the utilization of available resources in all the nodes of the business network (let's recall that within the order placement the allocation of the bought products may interest one of the stores — the closest possible to the customers' location — or one of the central warehouses);
- In general, supporting the warehouse staff in the processes required to handle all of the major and many minor warehouse tasks: I'm talking of receiving, inspection and acceptance, put-away, internal replenishment to picking positions, value added services, order assembly on the shipping dock, documentation, and shipping (loading onto carrier vehicles).

2.2.2.8. EMS - Email Service Provider

Finally, another important utility tool used by companies such as the client firm is the Email Service Provider — or EMS — that is, indeed, in charge of handling the marketing e-mails.
It allows, on one side, to access features such as creating and sending campaigns to lists of subscribers, automatizing marketing and emails that can also contain dynamic content and performing segmentation based on what the subscribers are interested in; on the other side, it can help unifying the management of the templates the business sends, to make sure that the communications received by the clients are always the same, valid, coming from the same society and including legitimate and up-to-date content, media and links.

2.3. The development tools

2.3.1. Development setting

When developing storefront applications, the developer persona is easily going to keep three important tools open:

- The **Integrated Development Environment** (**IDE**), where they write and test the application;
- The **Business Manager**, which is the SalesForce B2C Commerce online tool for configuring and managing B2C Commerce storefronts.
  It's a key element of the storefront applications development because it's the command center for merchandising tools (to perform functions such as setting data on products, images, campaigns, promotions, and search settings), administration (to set preferences, settings on the different sites or import/export sites' data) and site development capabilities (mainly for troubleshooting through debugging or configure development-specific settings).
  Of course, depending on the permissions associated to the role of a developer, the BM can be a hub for accessing one or a set of the available storefronts, such as different geographies of the same eCommerce application;
- The **Storefront application**, which is the actual application being written, kept open to see the real-time results of any changes.

2.3.2. Paradigms used

- **MVC architecture**: The storefront application under analysis follows the Model-View-Controller architecture, which divides it into 3 parts (as *Figure 2*):
  - The model, which is the business logic, data and rules underlying the application; it basically contains the data used to populate the view.
    In B2C Commerce is represented by APIs;
  - The view, which is essentially the front-end view, i.e. what's seen by the customer on the storefront;
  - The controller, which is the set of classes and objects that convert the customer's inputs on forms or button clicks into actions taken by the model or the view.

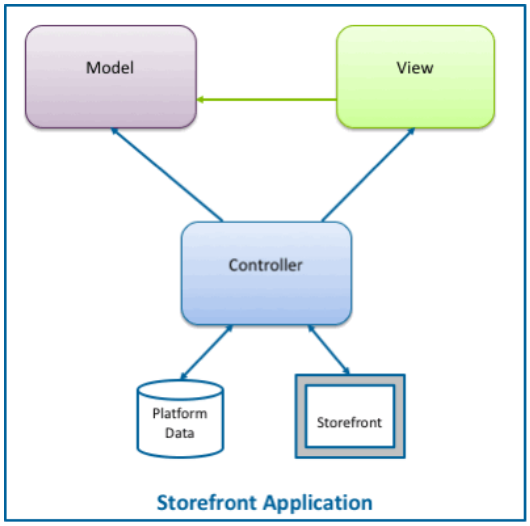

Figure 2: the MVC flow chart

While this architecture provides a high-level map of how the elements work together, the code still needs a deployment mechanism. That's where the cartridges' system comes in.

- **B2C Commerce Cartridges**: *Cartridges* are containers used by B2C Commerce to package and deploy program code and data.
  They are used to extend business functionalities or integrate with external systems, moreover they are structured in a directory-based pattern for maximum efficiency and customizability; a storefront needs at least one of them, but typically it has multiple.

  A cartridge can carry generic or application-specific functionalities: the former include standard processes that can be deployed in many sites, while the latter contain merchant-specific code and data.

And such code and data is retrieved in a hierarchical way: B2C Commerce loads individual cartridges in a specific order, loading the ones containing the most basic functionalities first (the Base, to be kept edit-free by the developers for a safe common fallback), followed by increasingly customized ones (the external Plugin, third party functionalities from LINK partners, up to the Custom level). This bottom up representation of the increasing levels of customization is called the **cartridge stack**.

Then, cartridges can contain different elements, such as controllers, form definitions, scripts, static contents (text, images, CSS files, and client-side JS files) and templates.
So together with the definition of the cartridge stack, we must introduce the concept of **cartridge path**: this determines the order according to which B2C Commerce identifies the correct element to run (cartridges earlier on the path — to the leftmost part of the directory — override the functionality of the ones later on the same path).

In practice, the organization of the cartridges according to these two paradigms helps maximizing their reusability together with a correct differentiation of the storefronts, thanks to the assignment of values to variables according to the most specific localization in which they are defined, or eventually falling back to more generic — homonymous — modules (up to the 'default' ones).
This allows the eCommerce application to have properly consistent and coherent information in all its geographies, yet, again, maximizing the code reutilization.

2.3.3. Javascript

Javascript is one of the most used programming languages in browser applications, since it conforms to the ECMAScript specification (commonly used for client-side scripting on the World Wide Web and meant to ensure the interoperability of web pages across different web browsers).
Here are some of its main features:

- *High-level*: it has a very strong abstraction, setting it closer to the human readability than to direct machine interpretability (it may even contain elements of natural language)
- *Object-oriented*: it's based on the paradigm of objects, structures that contain attributes (data fields keeping the information on them) and methods (procedures representing the actions that can be called on the instances' of such object). Differently from *procedure-based* programming, which structures the overall flow into sub-programs performing a defined function and that can be called from throughout the code, it's based on the dynamic interaction among objects, sending messages with each other, but maintaining their status and data. This helps in identifying actual "software components", factor that can grant re-usability, modifiability and possibility of maintenance of the code.
- *Event-driven*: as Javascript instructions do not follow fixed paths in their execution, the program dynamics are based on user actions/interactions triggering the next flow of events. This means that the code is structured on loops that continuously listen for new events and event handlers that are triggered by them.
- *(mainly) Just-In-Time compiled*: most of times Javascript code is compiled at runtime, rather than before execution. This means that the compiler typically analyzes the code being executed continuously and identifies the parts where the speedup gained from compilation/recompilation would outweigh the overhead costs of compiling such code, thus allowing adaptive optimization.

2.3.4. HTML / CSS

The UI in all the pages of the storefront is rendered on templates coded in SalesForce's proprietary extension called Internet Store Markup Language (**ISML**), which is based on HTML and uses CSS for formatting industry standard stuff.
HTML is the standard markup language for the documents designed to be displayed in a web browser, meaning that it's used to give a layout to files such as web pages by adding opening and closing hypertext tags (such as '<tagName>text</tagName>') to the text that's going to be displayed on them.
The language is meant to be assisted by Javascript to provide the content populating such web pages, and by CSS for what concerns the styling.
CSS is a language meant to enrich and diversify the aesthetic representation on HTML pages, and constituting together with such language a standard whose rules come from a set of recommendations for web pages provided by the World Wide Web Consortium.
It works through the assignment of classes to the different HTML elements of the page, and allows to manipulate the various attributes that each of these classes carry, such as colors, fonts, positions, visibility and many more, everything to provide a more complex customizability to the HTML document.

2.3.5. Node.js

Node.js is a back-end JavaScript runtime environment, a software providing the necessary services to execute JavaScript instructions outside a web browser.
It is open source, cross-platform and event-driven, and it's mainly used to build back-end services, also called APIs. But what does it mean?
Prior to Node.js, JavaScript was only used on client side code and then evaluated by an engine within the browser. Then, the introduction of such tool made possible for the code to perform computations of JS instructions also on the server side.

This results in the implementation of the **JavaScript everywhere** paradigm, meant to unify the developments of web applications around the same programming language.

Its Package Manager, ***npm***, is the world's biggest libraries' ecosystem, and it's composed of a command line client (called also 'npm') and a database of private and public packages, named 'npm registry'.
This last tool was also used in the scope of the project, mainly to launch instructions on the command prompt, like re-building the FE scripts or the SCSS after modifications on the UX code.

2.3.6. JQuery

JQuery is one of the most used and important Javascript libraries employed in the project constituting the scope of this document.
It's used to simplify the manipulation of HTML documents, as well as event-handling, animations and Ajax calls through which data can be sent from the front-end to the back-end side of the application and received back to populate FE variables for the UX.

3. The starting point

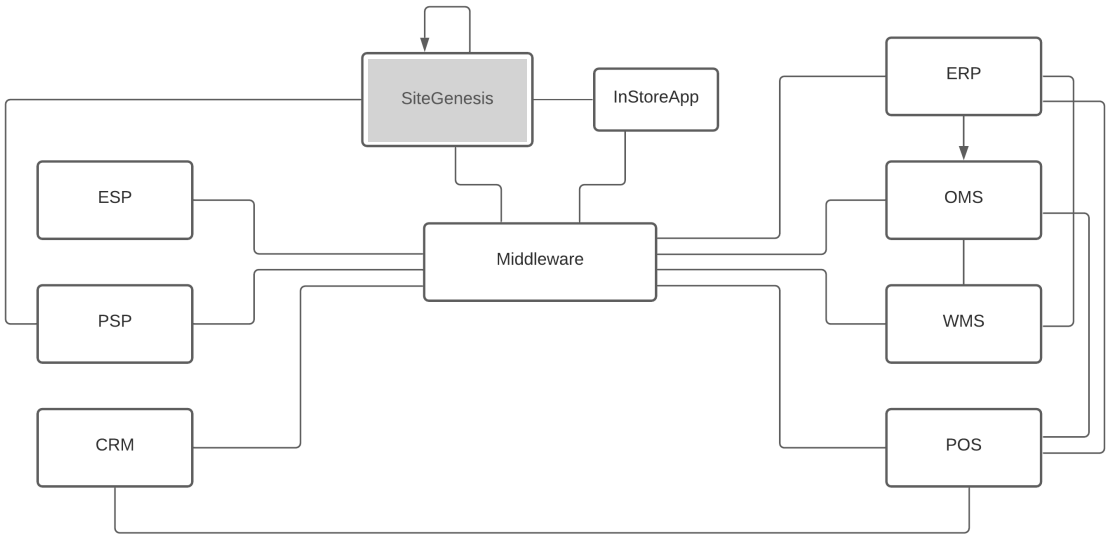

Figure 3: Flow diagram of the communication among the systems **before** the switch to SFRA and the introduction of a PIM system

3.1. Overview

The previous section explored all the entities and the technology tools that are involved in the project, representing all the gears that interleave with each other and let the overall machine perform its duty.
The following section is meant to portray the structure underlying the client firm's Commerce at the beginning of its Omni-channel program, when — in early 2019 — the first European countries had their rollout (as portrayed by *Figure 3*).
The focus in particular will be given to the description of SiteGenesis, the starting architecture.

3.2. SiteGenesis by Demandware

The first EU countries having their rollout in the overall Omni-channel program had SiteGenesis as their common backbone.
SiteGenesis JavaScript Controllers (SGJC) is an architecture launched in the year 2009 by the software technology company **Demandware** and based on their eCommerce platform.
The company was a revolutionary upstart launching the Software as a Service (SaaS) model for e-commerce long before others did, though it did not natively support mobile web experiences as it was built around a proprietary 'Pipeline' architecture.
Later on, in 2016, Demandware was acquired by Salesforce, that then renamed it SalesForce Commerce Cloud.
SiteGenesis has been one of the leading technologies for the storefronts, and still nowadays it's currently used in more than 2700 Commerce Cloud sites, including some of the world's leading brands.
However, from the beginning it showed a few deficiencies that brought SalesForce to develop a state-of-the-art heir for the next generation storefronts. Such successor is the StoreFront Reference Architecture.

A full comparison of the two systems has been made on the next chapter, to better identify the differences that make SFRA stand out against its predecessor.

4. The goal

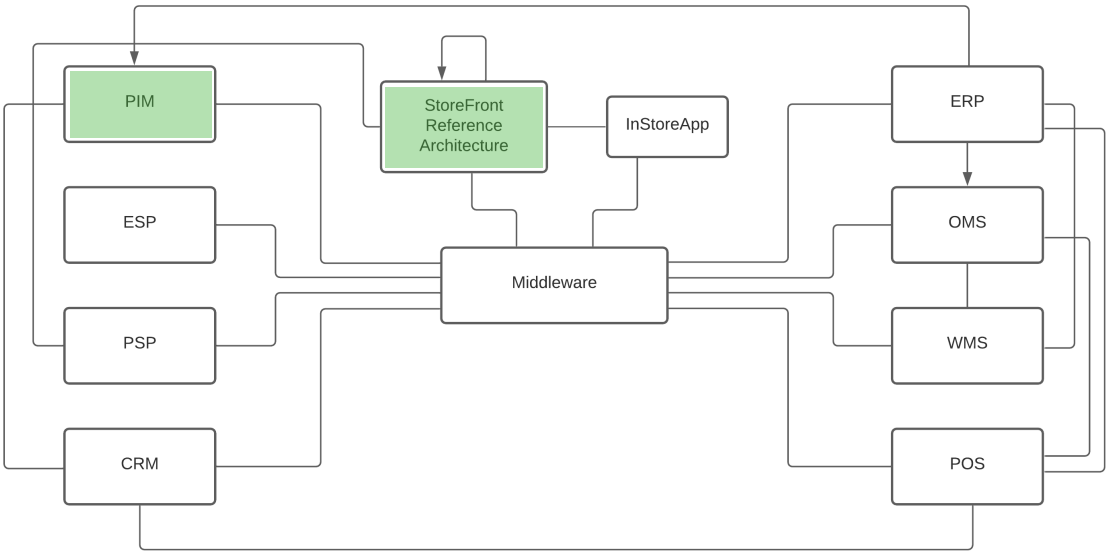

Figure 4: Flow diagram of the communication among the systems **after** the switch to SFRA and the introduction of a PIM system

4.1. Overview

If the previous section was devoted to describe the situation of the infrastructure at the beginning of the project, the following one aims at outlining the features of the reaching point (represented by the flow chart in *Figure 4*), by defining what the StoreFront Reference Architecture is and the benefits that it brought, together with the addition of a PIM system, to the Commerce.

4.2. SalesForce SFRA

4.2.1. Overview

As already stated, from the beginning of year 2018 the retail enterprise started a global transformation for the whole group, with the aim of bringing a unified Omni-channel experience.
The designated Commerce chosen for this global transformation was SalesForce Commerce Cloud, since such cloud-based platform holds the position of global leader within the companies in the sector.
In the following section, a description will be given to better explain, among the different components of the entire system, what SalesForce Commerce Cloud is.

4.2.2. SalesForce Commerce Cloud (SFCC)

Commerce Cloud is, as the name suggests, a cloud service offered by SalesForce that can perform CRM, OMS and In-store functionalities; in the last few years it has become one of the global standards when building an eCommerce platform.
Its core capabilities are related to B2B, B2B2C and B2C customers, granting a high level of specific features and customization of the storefronts.
Moreover, this service allows online sellers, such as B2C retailers, to launch sites for multiple brands or geographies (if they sell in many countries) and then manage them all in a single place.
But consumes don’t just visit web and mobile sites, they also interact with brands through emails, social media, and in stores: that’s where SFCC integration of

eCommerce sites with marketing and service solutions comes in handy, making every email and social media interaction more personalized and relevant to shoppers.
Another feature that SFCC provides is a direct embedding with **Einstein** features, a tool that uses AI to mine data related to Product Recommendations, Predictive Sort, Commerce Insights, Search Dictionaries, and Search Recommendations.

Both the architectures under the spotlight of this Thesis work belong to SFCC — we recall that SFCC is just another name for Demandware, the platform foundation of SiteGenesis —, one is just the newer generation of the other (they came out 9 years apart from each other).

4.2.3. StoreFront Reference Architecture (SFRA)

Talking about B2C Commerce, we can distinguish a few categories that compose it: *merchandising and marketing*, *multi-site management and localization*, *B2C Commerce API extensions* and, mostly relevant for the scope of this document, *commerce storefront*: this last category represents essentially an eCommerce site, but enriched with a few features: brands can indeed also differentiate, manage, and customize the customers' whole shopping experience.
Among the newest capabilities offered by SFCC, there is the StoreFront Reference Architecture (SFRA): a robust reference architecture that helps brands build and launch sites quickly and easily using mobile-first best practices.
While each site has its own storefront rich of own features, retailers can use the reference architecture to create it.
It sits — yet fully customizable — outside the platform API layer, lets the business create storefronts that support multiple languages and currencies, provides a modern shopping cart and checkout that include interesting features such as *Save for Later* (to save an item to a wishlist for a logged in user), optimized user flows,

Apple Pay, and other wallet options to make the checkout process as fast and seamless as possible.
SFRA capabilities also include prebuilt integrations to extend site functionality, and open APIs that allow sites to connect to any third-party data or applications like user reviews.

4.3. SFRA vs SiteGenesis

The comparison of SFRA with its predecessor shows that a few of the improvements come from:

- Its Mobile-First fully responsive design, key factor for a business such as the client firm (forecasting mainly smartphone traffic on its storefronts), making it more scalable and bringing reduced Total Cost of Ownership in the long term;
- Improved modularity and abstraction together with a full compatibility with the MVC paradigm, allowing to extend Models and Controllers without replicating the templates. This affects extensibility and inheritance, since it encourages a modular approach in the customizations, and producing a better maintainability factor;
- General performance improvements (affecting also the user experience) due to:
    - The compatibility with a Page Designer, a tool available to the Business on the BM that allows them to configure their web pages with ease, by building them visually (as if it was block programming);
    - An overall faster system of templates, that takes advantage of Javascript controllers and frameworks as Bootstrap 4 or JQuery;
    - Many out-of-the-box integrations for the cartridges, such as with ApplePay or In-store pickup;
    - A better testability of the former, that comes with unit tests for many of the base functionalities.

4.4. Benefits of introducing a PIM

In the previous infrastructure of the Commerce, the management of the assets was assigned to jobs within SFCC itself.
This solution produced centralized activities, however — in addition to the operations being really slow — it resulted in numerous lacks from the point of view of maintainability, scalability and human interaction.
Hence, the proposal of the adoption of a PIM system. The main reasons were related to:

- *Data quality*: a PIM is more effective in detecting gaps in the products and solving issues such as incomplete attribution, inconsistent attribute values, missing images;
- *Centralization*: a PIM helps in aligning the content across channels by creating a single repository of product data for multichannel publishing;
- *Performance*: a PIM provides improvements in by reducing redundancy, rework and repetition and the Business doesn't need to rely anymore on error-prone, complex spreadsheets;
- *Information*: Finally a PIM helps accelerating business value since it allows to connect products to the transactional, interactional and analytics data it produces.

5. The project

5.1. Overview

Now that the reader is provided with all the basic information on all the preliminary entities, decisional drivers and tools involved, it is possible to enter into the details of the actual project and all the processes related.

The first section will assume the perspective of the team, and it aims to explain all the steps that are necessary to perform the job, portraying how the client and the consulting firm get to an agreement on the project guidelines and how it is carried out by the latter.

Finally, the section devoted to the development will enter the single member's point of view, and it will reflect some of the actual activities that marked my contribution within the team.
The purpose of this section is to illustrate the actual tasks that the technical team is involved into, and providing also a few concrete examples of issues faced, solution approaches and results achieved.

5.2. The development pipeline

5.2.1. Design

The earliest phase of this software engineering case study is the definition of what the delivered product has to be, which features must include and how it needs to look.

During this preliminary step, these factors are established between the client and the consulting firm with the highest details possible, they try to cover all the possible use cases and scenarios that may show up and include the agreed responses of the Commerce in a document gathering all the possible **user stories**.

This document, from one side, states how the client requests the storefront to be, the UX features that the platform has to present and how it needs to respond to all possible inputs from the users.

On the other side, such document implicitly defines the responsibilities of the different parties, it provides a bound to the client demands and by implication a definition to which requests are duty of the Commerce team and which are considered extra, hence requiring an additional effort to be satisfied (and additional costs for the client).

Once the user stories are given a first agreed definition, the team has a concrete milestone to achieve, so the first developments can take place.
However, defining the design step as preliminary doesn't mean that it reaches an early termination, as such phase is spread along the whole project, or at least until the relationship with the client gets to an end: that's of course because design features may vary due to forgotten scenarios, changes in the client's policy or requests by the same — thus implying changes in the user stories.

5.2.2. Tests

During the whole development of the platform, the results achieved are constantly tested to make sure that the product being delivered fits the requests of the client and that everything is working seamlessly.
Depending on the phase of the development, different types of tests can take place, and they might involve or be conducted by different entities.

5.2.2.1. Point-To-Point tests (P2P)

Point 2 point tests can be considered among the earliest forms of testing, and they are performed internally in the Commerce team.
They tackle the task of verifying that the integration with each of the external services — considered singularly, as the embedding is developed — works as expected.

5.2.2.2. System Integration Tests (SIT)

Further in the development of the software, once the integration among all the communicating external systems has been elaborated, System Integration Tests (or SIT) take place.
Their aim is to validate all possible procedures of sequential calls to the present services, and they consist in live sessions where exponents (that could be technical managers or the developers themselves for major features) of each of these systems meet and — of course keeping a testbook as a starting point is always advised — check that most of the flows in the program work as expected.
This is done in order to check whether the communication among all the entities is correctly functioning, and take on-line solutions as much as possible if that's not the case.

5.2.2.3. Quality Assurance and User Acceptance tests (QA & UAT)

The final step in the testing pipeline before the go-live date of the application consists in the quality assurance (QA) and user acceptance tests (UAT).

In the context of the project, it was actually an external enterprise — specialized in testing — being in charge of performing the checks related to this first phase: these are done through extensive runs of all the flows in the platform, to make sure that everything works as defined by the user stories, considered one by one at first, then by looking at entire end-to-end procedures; then a dedicated ticketing service is used to notify the issues still present.
After the QA, the UAT phase of testing consists in the same validation, this time performed by the Business.
Of course, internal checks for bugs are also kept active within the Commerce team during this and all the other steps (looking for blocking issues beforehand), however the problems reported externally certainly have a higher priority.

5.2.2.4. No regression tests

When code coming from bug fixes or developments is released in production environment for some country, it is main interest of the developer to make sure that the other countries are not influenced in unwanted ways.
No regression is a testing phase that takes place at the turn of a software release in production with this aim.
This step is composed by a set of checks that are performed on the already existing storefronts on the processes that run on the code that's been affected by the modifications or developments released, to ensure that they are not impacted by unexpected changes or malfunctions.

5.2.2.5. Smoke tests

Finally smoke testing, also known as confidence testing or build acceptance testing, is another validation phase that aims at pointing out general failures within a release in production. These tests are performed with the focus of covering the most important functionalities of the system, to validate whether the main functions of the platform appear to work as expected.

5.2.3. My contribution: the development

Both the design step and the different types of testing are essentially meant to shape the product to be delivered.
This, in other words, means that they interleave with the developments to provide them with a guideline, but also to validate that the expected results are reached by the storefront being built.

So at this point, we can finally describe the main technical activities present within the project, the actual developments.

This, again, is the context in which my actual contribution, as well as for the other technical members within the team, takes place.

5.2.3.1. Types of technical activities

In a re-platforming operation such as the one portrayed in this document we can distinguish three types of activities that a developer might be involved into: bug-fixing, change requests and new developments. All of them will be thoroughly described in the following section, which will also provide a few concrete examples of real tasks that have been tackled by me during my internship.

5.2.3.2. Bug-fixing

The activity of bug fixing is the prevalent occupation performed during an already started team development, especially on a re-platforming operation: as the name suggests, the task represents the resolution of malfunctions or the fix of features showing up differently from the user stories, found (most of times) directly on the storefront.

Once the bug is detected — primarily during dedicated testing sessions performed by the functional team —, it gets reported through the usual ticketing services and assigned to the proper technical team member to get rid of it.

When approaching a bug, the first step is of course the replication of the issue (it's indeed fundamental to provide as many details as possible when reporting it), and then consulting the tools considered appropriate — such as the ones provided by a debugger or the logs associated to the environment where such issue is found — to make hypotheses on what could be the cause of it.

Generally, the issues found can be related to:

- wrong configurations on the Business Manager, which require an update on the faulty preference(s) or a notification to the Business communicating what needs to be changed (that's because after the rollout of a site the management of its settings on the BM is up to them);
- problems in the information flow among some services — or in this case prevalently with Mulesoft, since it stands in the communication as a data hub between the Commerce and the other external systems. These issues can figure in the storefront as missing or wrong data in the view, and need to be redirected to the failing system to deal with it, possibly with all the additional insights gained during the analysis;
- actual bugs present in the code and generated by a rough integration of new developments producing regressions or mismanaged cases.

  These last issues are the ones that actually require intervention on the code.

Actual example: Order placed without e-mail and phone on Production environment — India:

An interesting case of bug-fixing, not for the fix itself but for the analysis behind, was related to a real case happened on the third day the eCommerce platform went officially live in the Indian country (first of the two countries having a rollout):

A (real) customer was able to place an order without filling two mandatory contact fields, the e-mail and the phone number.

Giving a quick analysis to the export of the orders by SFCC, they showed two, and only two, orders coming from the same customer and placed around 8 minutes one from the other.
The first one, though, resulted in an ABORTED status, while the second (the newer — and faulty — one, on which the ticket was based) was SUCCESSFUL, yet with email and phone data missing.
So it made sense to look for a possible correlation between the two orders.

After a bunch of further analyses on the logs related to the Production environment, and consulting the SFCC documentation, I came to the realization that the default behavior of an SFCC controller might be involved in the issue, but to better explain the problem some preliminary notions are necessary: in some eCommerce systems, older for the most part, when the customer places an order, the flow starts with the build-up of a **basket**, containing the SKUs about to be ordered and gradually filled with the mandatory information related to the contact data (email and phone), type of shipment, shipping/billing addresses and payment coordinates during the checkout.
Then, once the shopper confirms by clicking on "place order", they get redirected to the PSP — the payment handler's page —, where they are asked to authorize an amount to pay, which is established by the basket itself.
Finally, once the payment is authorized, the user is redirected to the storefront, where the basket has been wiped clean and an **order** has been created from the data contained on it.

But this means that it's possible for a customer with a malicious intent to game the system and get free products: this can easily be done by manipulating the basket at the right moment.
This happens if the user, once redirected to the payment handler's site — that requests a payment for an amount defined on the current basket — opens another tab and adds a few new products to the basket. Then, by performing the payment on the payment handler's site, the order will be created out of the new basket, which includes also the new products, but paying for the old basket.

However, this is not the behavior carried by SalesForce.
Differently from the scenario described above, on SFRA storefronts once the shopper is redirected to the payment handler's site the **basket** is deleted and the **order** is created straight away, then if the payment flow is not successful such order results in a FAILED or ABORTED status, the user is redirected to the storefront at the last step of the checkout and the basket is restored if no new one has been created (due to constraints on SFCC).
This means that the user tried to perform this exploit, the payment was refused (resulting in an ABORTED order) and so they got redirected to the confirmation step in the checkout; however, in doing so, the old basket was not restored because in the meantime some new products were added and a new one was created.
This all in all means that the shopper was redirected to the checkout on the final step, asking a confirmation for a basket in which all the mandatory data (except for the addresses — pre-filled through radio buttons —) were not compiled during the previous steps.

In this circumstance, to cover this edge case the fix was pretty basic, it was only necessary to add a safety check for the contact details to be present on the basket once the confirmation button is pressed; however, this analyses taught clearly how the behavior of the customer is not always the one planned by developers and, moreover, it's not always naive or unintentional.

5.2.3.3. Change Requests

As it's already been said, once the project starts the first step that the firm needs to take care of is the design of the desired platform. This is done by discussing logic and UI features that the client needs, consider their feasibility (and associated costs) and producing a set of user stories from the results.
This, though, doesn't mean that the first design is going to be the mirror of the final product. Indeed, changes can often happen: it could be for the lack of use cases among the ones considered or for an explicit request from the client, due to changes to the internal policies or just new features to be implemented w.r.t. the former system.
Once this happens, an associated ticket is raised towards the firm: these don't represent bugs that prevent the platform from having the expected behavior, but it's a demand from the client to change what was agreed during the design phase (which means that it consists in a source of costs, since an extra effort is being expected).
At this point the ticket is treated just as any other issue, requiring to analyze the current state of the storefront, with the code associated, and perform the modifications, additions or tweaks (potentially involving also external services' providers) to make sure that the request is fulfilled and that all the other locales are not impacted by it if not requested.

Actual example: tracking link extension — Malaysia:

An interesting example of change request faced during the participation in the project was associated to the tracking link in the Malaysian country.
Beforehand, once opening the Order Details page on the storefront, the SFRA code logic (already developed and working on the Indian storefront) expected a unique tracking link (*Figure 5*), a static string coming from a custom preference on the Business Manager and populated with the URL of an agglomerator provided with two placeholders to allow it to retrieve the shipment information according to:
{1}: the carrier;
{0}: the associated tracking link.

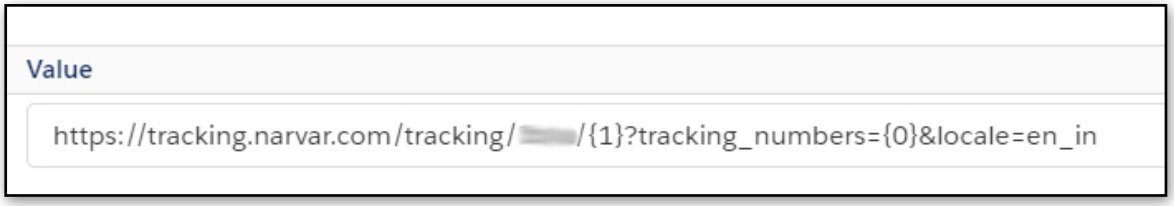

Figure 5: Tracking link custom preference - before

However, with the addition of the Malaysian locale, the retail firm asked to extend this logic in order to allow them not only to use a single tracking link from an agglomerator, but to introduce the possibility of adding many different tracking link templates coming straight from the carriers' websites, making sure that for the older countries everything worked fine as before.

The solution approached for this issue was to dismiss such custom preference (which was limited to a single string) and deploy a new one, shaped as a JSON map, in which the key is represented by the carrier (whose ID is already received from Mulesoft) and the value is its associated tracking link.
Through this preference, and the related adaptations to the code, the correct tracking link is fetched among all the ones available for a particular locale and the associated button on the storefront is populated with the correct URL, in which the placeholders are substituted.

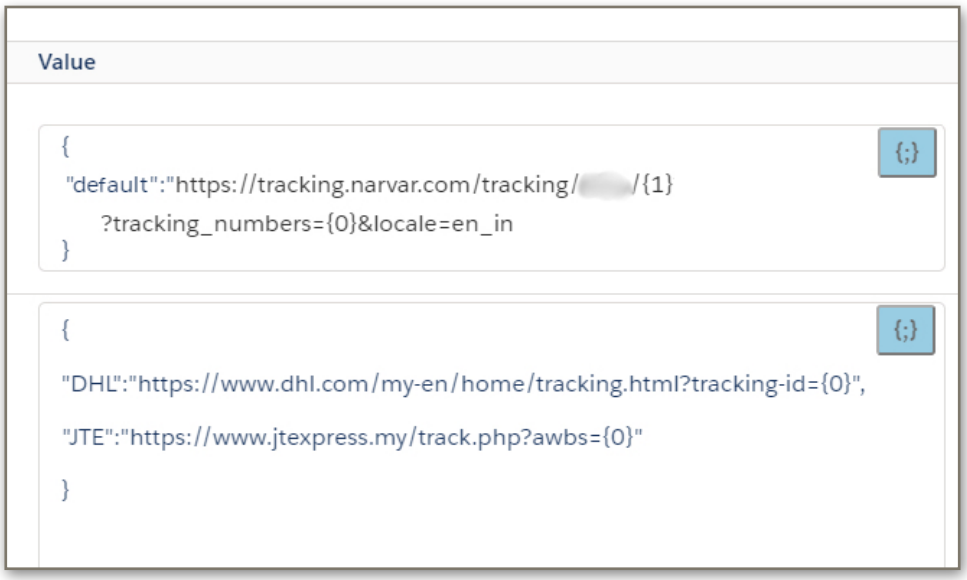

Figure 6: The body of the new site preference introduced in the two new countries.

How is the correct behavior for the old countries preserved?

That's thanks to a fallback logic according to which, if the carrier received is not found in the map, the tracking link used to populate the button from the SFRA code comes from the "default" key (which is going to be the only one populated in case of unique link from an agglomerator), or if even this situation resolves as a failure (for example, if the Business forgot to update the custom preference on the BM) then still rely on the older preference.

Actual developments

The last form of activities that a developer faces during a team work such as the one presented involves the development of features.
The generic name of "developments" refers to generic code modifications not associated to bugs or CR requests, or design traits that are either new or inherited and adapted during the re-platforming.

The new developments are related to features that are unprecedented w.r.t. the older countries already present in the infrastructure, so they can often be associated to change requests willing to satisfy peculiar requests of the business according to the new country.

Then, there are the re-platforming developments: these tasks take place mostly in the early phases of the build-up of the new storefront for a certain country, in which the developer needs to analyze the SFRA code deployed for other geographies or the SiteGenesis version of the same locale (if present); modifications are then made or new locale-specific pages are detached with the customizations that allow the new storefront to properly work with the integration of the forecast services' providers and business requirements.

Finally, there's one other type of developments: such are those peripheral developments that aim for a general improvement.

Actual example: performance improvement — India:

A relevant example of development belonging to the last category is related to a task addressed when the Indian site was about to go live on Production environment:

Improve the performances on the overall eCommerce.

Approaching this topic, all pages of the storefront have been considered, however the evaluation of the improvements was done according to their immediate feasibility and to which page they were going to be applied to.

This means that the pages were weighted on their relevance, focusing primarily on:

- homepage;
- primary category landing pages;
- product listing page (PLP);
- product details page (PDP);
- checkout.

The software used to perform the analyses was Google Lighthouse, a tool that nowadays has become a standard more and more used. It analyzes webpages by dividing the indices into:

- *Performances*: everything that's related to the speed of the website and how long each page takes to completely load from scratch;
- *Accessibility*: all the details that make the browsing on the page user-friendly, from the color contrast to the 'alt' tag definition on the images and so on;
- *Best Practices*: all the activities representing how good the subject is at following the basic webpages rules, such as the correct functioning of the SSL protocol, the presence of HTTP/2 or valid APIs only and more;

- *SEO*: this category ensures that a webpage is optimized for search engine results ranking, resembling how friendly or optimized the webpage is for the user experience;
- *Progressive Web App (PWA)*: this parameter is related to all those resources loading as webpages but behaving as mobile apps, thus it has not been monitored.

The gathered indices were then tabulated into Microsoft Excel to report the starting situation (*Figure 6*).

PAGE NAME	Before:	Performance	Accessibility	Best Practices	SEO
Homepage		35	81	87	91
Order history		45	85	87	77
Login		50	87	87	77
Wishlist		48	84	87	77
ViewBag		47	79	87	77
Stores		27	81	80	77
LoyaltyPage		53	86	87	77
MyAccountPage		54	85	87	77
Women		47	86	87	85
Men		47	86	87	85
PLP		25	75	87	69
TrackOrder		52	87	87	77
WhoWeAre		55	85	87	85
PDP		36	76	87	77

Figure 7: Google Lighthouse parameters - before

Before assessing the concrete development, a few remarks are needed:

- the eCommerce portal has a *mobile-first* design, meaning that the user experience is forecast to be coming mainly from smartphone traffic; thus, the optimization has to keep that in consideration;
- the subject of the analysis is an already established eCommerce portal; as such, several indications for the optimization provided by Lighthouse — mostly related to JS/CSS — were basically unfeasible due to the great quantity of modifications that they would require and the huge impact that they could have on a website that was almost ready to be deployed in Production environment;
- The numbers computed by Lighthouse have to be considered as approximative: they were computed on the common Development code branch, the *before* and

after state were a few days apart from each other, and during those days the overall code was affected by modifications and additions of features by the other team members solving bugs or managing change requests.
Moreover Lighthouse computations were done on the default standardized test case and regularly showed fluctuations in the values, though limited in value.

As it can be immediately be deducted from the picture, mere performance was the main point to address.
And by looking at the detailed reports provided by Lighthouse what was immediately noticeable is that most of the potential improvements, aside from the (minor or unfeasible) JS/CSS code optimizations already nominated before, were related to images' management.
In particular, the two interventions to bring the highest reward were:

- applying ***lazy loading*** to the images, which a good measure that involves loading the images in a smart way, deferring all those that are not immediately on screen or they are not even about to be;

- applying a more precise ***resizing*** operation to the images: previously, the images were all sized according to a few fixed breakpoints (*mobile*, *tablet*, *desktop*) representing the *width* of the *entire screen*.
  However, many images in the storefront are placed in containers that are oftentimes smaller than the entire view, and they mostly come from static components that are added to the page through the Page Designer on the BM and always behave in the same way.
  Thus, I tackled the task by modifying the resizing algorithm present in the storefront to make sure that the operation was not done anymore on the *screen* size, but on the size of the *container* in which such component was inserted in; this change involved the definition of a few substitutive breakpoints representing the width *of such container in each* possible *device* breakpoint (as before, *mobile*, *tablet* and *desktop*), and then using this parameter to give the image a proper size.

So at the end of this step in the development the images got their size no more on the *full width* of the screen at a certain breakpoint, but on the *width of their associated container* at that specific breakpoint.

On the other hand, there was more that could be done according to this last point: and for the reader, the rationale behind these additional considerations was “favor an overall performance improvement by penalizing a negligible amount of users”.

To be more specific, the previous breakpoints were chosen safely to make sure that all the devices of a certain kind got the images loaded as crisp as they could get.

This resulted in a set of width breakpoints (in pixels) equal to:

- mobile: 767
- tablet: 1024
- desktop: 1440
  (from 1440px on the size stops growing, instead what grows is the two white sides of the page).

However, not all users use tablets or desktops as much as smartphones, and most of smartphones have a screen width that barely exceeds half that safety breakpoint value.
So a few statistic considerations could be made to further optimize.
The website gs.statcounter.com provided the following data:

- Percentages of users for each type of device worldwide (from January 2020) - *Figure 7*;

Date	Mobile	Tablet	Desktop
2020-01	52,02	45,29	2,7
2020-02	51,69	45,66	2,65
2020-03	52,03	45,32	2,65
2020-04	53,81	2,92	43,27
2020-05	50,48	3	46,51
2020-06	50,13	2,81	47,06
2020-07	50,88	2,74	46,39
2020-08	51,33	2,78	45,9
2020-09	50,21	2,62	47,17
2020-10	48,62	2,5	48,88
2020-11	52,95	2,83	44,22
2020-12	55,73	2,81	41,46
2021-01	55,68	2,87	41,45
2021-02	54,46	2,91	42,63
2021-03	54,22	2,85	42,93
2021-04	54,57	2,76	42,66

Figure 8: Percentages of users for each device type
Source: gs.statcounter.com

- Highest market shares for each screen size according to each device type worldwide (from April 2020) - *Figure 8*.

Mobile screen resolutions:

		(04/2020-2021)
Vert.	**Horiz.**	**Market Share (%)**
1080	1920	0,87
1080	2340	1,02
320	568	1,44
360	640	14,66
360	720	3,97
360	740	2,62
360	760	5,13
360	780	5,81
360	800	2,77
375	667	6,83
375	812	4,93
393	851	2,71
412	732	1,48
412	846	3,05
412	869	2,95
412	892	3,32
412	915	1,93
414	736	3,45
414	896	7,24
720	1280	1,02
Other + totals:		**22,83**

Tablet screen resolutions:

		(04/2020-2021)
Vert.	**Horiz.**	**Market Share (%)**
1024	1366	2,01
600	1024	1,42
768	1024	2,11
712	1138	0,6
720	1280	1,01
800	1280	7,26
534	854	0,98
600	1024	2,62
600	960	1,14
601	962	4,83
712	1138	0,57
768	1024	48,19
800	1280	5,58
810	1080	2,41
834	1112	2,49
834	1194	1,15
534	854	0,58
540	960	0,41
600	960	1,54
601	962	3,45
Other + totals:		**9,63**

Desktop screen resolutions:

		(04/2020-2021)
Width	Height	**Market Share (%)**
1366	1024	0,52
1024	768	2,41
1280	1024	2,53
1280	720	5,06
1280	800	2,73
1360	768	1,25
1366	768	22,14
1440	900	6,87
1536	864	9,04
1600	900	3,9
1680	1050	2,03
1920	1080	21,04
1920	1200	0,86
2048	1152	0,49
2560	1440	2,1
640	360	0,45
1024	768	2,52
800	600	0,98
1080	810	0,67
1112	834	0,57
Other + totals:		**11,82**

Figure 9: Highest market shares for each screen size according to each device type worldwide
Note:
- The colors represent the associated view of the storefront for each particular device (oriented vertically or horizontally).
- The "other" devices are considered as homogeneously distributed among the other distributions.

These data have then been tabulated onto Microsoft Excel, the weighted distribution of users for each view type has been computed, and by automatically minimizing the 3 breakpoint parameters I looked for the best trade-off overall, getting to this solution (*Figure 9*):

MOBILE screen width considered:	414
TABLET screen width considered:	1024
DESKTOP screen width considered:	1440
"happy" MOBILE users coverage (%):	**98,11528**
"happy" TABLET users coverage (%):	**82,05866**
"happy" DESKTOP users coverage (%):	**98,08303**
Total "happy" users coverage (%):	**96,37122**

Figure 10: Results of the minimization process.
Note: A user is defined "happy" if the breakpoint used for the scaling of the images is equal or higher w.r.t. the width of the device they're using.

This, compared to the starting breakpoints — respectively **767**, 1024 and 1440 —, produces an overall performance improvement, by still visually *fully* satisfying more than 96.3% of the users; the increase in performance mostly influences smartphone consumers and "sacrifices" a few tablet users — proven to be the less used kind of device —, meaning that a few of them, depending on their screen size, might get slightly less crisp images on a particular view.

PAGE NAME	After:	Performance	Accessibility	Best Practices	SEO		Improvement:			
Homepage		42	82	73	91		20%	1%	-16,1%	0%
Order history		59	86	87	77		31%	1%	0%	0%
Login		58	88	87	77		16%	1%	0%	0%
Wishlist		60	86	80	77		25%	2%	-8,0%	0%
ViewBag		33	81	87	76		-29,8%	3%	0%	-1,3%
Stores		28	82	80	77		4%	1%	0%	0%
LoyaltyPage		55	88	87	77		4%	2%	0%	0%
MyAccountPage		54	86	87	77		0%	1%	0%	0%
Women		53	88	73	85		13%	2%	-16,1%	0%
Men		50	88	73	85		6%	2%	-16,1%	0%
PLP		28	76	87	69		12%	1%	0%	0%
TrackOrder		47	88	87	77		-9,6%	1%	0%	0%
WhoWeAre		65	86	87	85		18%	1%	0%	0%
PDP		32	78	80	76		-11,1%	3%	-8,0%	-1,3%

Figure 11: Google Lighthouse parameters - after

All in all, considering the lazy loading and the resizing operation of the images, the increase in performance was thorough, with a few oscillations for certain values and some negative values (due to previous resizing parameters that were erroneous or the update of their value was left behind during the re-platforming operation from their SiteGenesis counterpart).

6. Conclusions

Even though the correlation with what I learnt during my academic years was not immediate in the first place, the months of involvement in the project were insightful on a broader level.

Indeed, everything up to the programming languages used in the day-by-day activities were new to me; but nevertheless I had the chance of putting into practice the general ability to interpret a foreign code with all its flows, and of course this was the most immediate match with my University path at Politecnico di Milano.

However, the most tangible benefits that I got from this experience count the opportunity of being able to put into practice what *engineering* means, tackling the concept of *problem solving* in a professional context, where there's an issue and it's required to learn the most effective way to solve it, yet being able to rely on my teammates for brainstorming or asking for any advices.

Then, the skills that I had the chance to improve are related to aspects that are fundamental in nowadays worklife: the possibility of facing the professional approach of the team and be part of it in managing the delivery of the project, having relationships with the Business client and learning about the latest state-of-the-art technologies and actual global standards employed in the digital environment.

But all in all, I found the biggest source of personal growth in the opportunity of getting out of the student comfort zone and finally having my first real work experience: thanks to that, I could access the business reality, see the effectiveness of different approaches when facing a problem and perform actions that, for the first time, allowed me to reach concrete results.

7. Appendix

7.1. Acronyms:

AM = *Application Maintenance*
API = *Application Programming Interface*

BE = *Back end*
BM = *Business Manager*

COD = *Cash on delivery*
CR = *Change Request*
CRM = *Customer Relationship Management*

DD = *Deloitte Digital*

ERP = *Enterprise Resource Planner*
ESB = *Enterprise Service Bus* (*example: MuleSoft*)
ESP = *E-mail Service Provider*

FE = *Front end*

IDE = *Integrated Development Environment*
ISML = *Internet Store Markup Language*

JS = *JavaScript*

MVC = *Model-View-Controller*

OMS = *Order Management System*

P2P = *Point-to-point* (*tests*)
PDP = *Product Details Page*
PIM = *Product Information Management*
PLP = *Product Listing Page*
POS = *Point of Sale*
PSP = *Payment Service Provider*

QA = *Quality Assurance* (*tests*)

SEO = *Search Engine Optimization*
SFCC = *SalesForce Commerce Cloud*
SFRA = *StoreFront Reference Architecture*
SIT = *System Integration Tests*
SKU = *Stock Keeping Unit*

UAT = *User Acceptance Tests*
UI = *User Interface*
UPC = *Universal Product Code* (*12-cipher code associated to a barcode mainly used in North America*)
US = *User Story*
UX = *User eXperience*

WMS = *Warehouse Management System*

7.2. Terms definition:

7.3. References

https://www.wikipedia.org

https://www.salesforce.com

https://trailhead.salesforce.com

https://www.softwareadvice.com/resources

gs.statcounter.com

https://warehouse-management.com/What-is-a-WMS-92163.htmlhttps://

www.tadigital.com/insights/perspectives/storefront-reference-architecture-sfra-vs-sitegenesis-understanding-key