République Tunisienne
*****
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
*****
Université de Carthage

ENICARTHAGE
المدرسة الوطنية للمهندسين بقرطاج
Ecole Nationale d'Ingénieurs de Carthage

الجمهورية التونسية
*****
وزارة التعليم العالي و البحث العلمي
*****
جامعة قرطاج

Final Year Internship Project

Presented to obtain a

Computer Science Engineer Diploma

Specialization : Information System

Subject :

TITRE

Made by

KAIS NEFFATI

Host company : SOFRECOM TUNISIE

Sofrecom
The Know-How Network

Entreprise supervisors : Mr Walid HAMOUDA ET Mr Mehdi KALAI

Univercity supervisor : Mrs Ilhem Abdelmouleh

JUIN 2017

College year : 2016/2017

Abstract

Here goes the abstract

Dedication and acknowledgements

Here goes the dedication.

AUTHOR'S DECLARATION

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: ..... DATE: .....

TABLE OF CONTENTS

List of Tables

Table **Page**

List of Figures

Figure **Page**

CHAPTER 1

PROJECT SCOPE

In this chapter, we introduce the host company "Sofrecom Tunisie". Then we explain the project by putting it in context , exposing the problematic and proposing a solution. Finally, we describe the development methodologies and the analysis/Design one.

1.1 Host company

1.1.1 Sofrecom

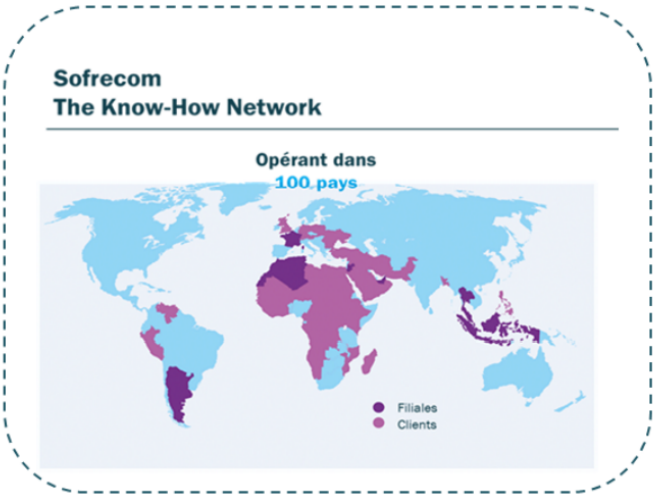

FIGURE 1.1. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

Sofrecom, an Orange subsidiary, has developed over 50 years unique know-how about oper-

ators businesses, making it a world leading specialist in telecommunications consultancy and engineering.Its experience of mature and emerging markets, combined with its deep understanding of the structuring changes affecting the telecoms market, make it a valued partner for operators, governments and international investors. In recent years, 200 major players in over 100 countries have entrusted strategic and operational projects to Sofrecom. Sofrecom is above all a network of men and women, a powerful network of know-how and expertise that ties its personnel to customers, Orange experts and industrial and local partners.Sofrecom's Know-How Network is also the guarantee of effective transfer of knowhow and skills for sustainable transformation based on internationally certified methodologies.

1.1.2 Sofrecom Tunisie

FIGURE 1.2. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

Sofrecom Tunisie, a young company in the local market, inaugurated in October 2012, is the youngest and largest subsidiary of the Sofrecom group in the Africa and Middle East zone. In 5 years it has positioned itself as a major player in consulting and engineering in telecommunications. More than 400 Experts, Sofrecom Tunisia has acquired its strength and credibility through hundreds of diversified international projects allowing us to be one step ahead of our competitors. As part of the Orange Group, with an international reputation, working for two major Orange customers: DSI France and OLS

1.2 Problematic

When we talk about selling products and increasing revenues, we totally refer to making offers and publishing promotions. So many companies are using a variety of ways to promote their service. One of them is the Orange group. Orange Group is spending a lot of money on marketing campaigns. Orange is using SMS, MMS and TV emission to be attached to it's clients which is not always the best way.As many marketing experts said, the key success of a marketing strategy is to know where, when and to whom a product and/or a service should be exposed.

A team of engineers in Sofrecom Tunisia, and in an innovation scope, proposes to make a new dimension in Orange marketing strategy. The idea is to provide a mobile platfrom that use proximity to allow users to see promotions ,services ,Events ,Sensitization ... depending on their intersts and their locations. This application will have a free, location-based orange ibeacons , use device's GPS capability to locate Orange ibeacons. Subscribers can capture gifts and promotions when nearby ibeacon using BLE devices through application. The Backend engine of this application gets customers relevant information (rate plan, balance, services,...) in order to propose a bespoke award. This idea brings some added values ; promote Orange low-rated services usage ,increase customers loyalty ,trigger based promotions and gifts as per user preferences and targeting subscribers based on usages and consumption habits (Voices, SMS, Data, TopUp...).

Bringing this application to life require implementing a system that allows us to recommend a service and track user behavior. This system should be apt to deal with a huge amount of data (Orange Clients), a Big data should be analysed. We were influenced the most by YouTube, which, in first use, exposes videos in an arbitrary way. As users begin to interact with the dashboard YouTube recommend the right list of videos. YouTube takes into consideration recent and old user data. Once the user changes his behavior YouTube will try to rebalance the dashboard view.Our system finally should be able to analyse big data, make recommendation based on interests and proximity and support the logic of application in the backend.

1.3 Solution

1.3.1 Intorduction

Our application is divided mainly in two parts, a part responsible for data analytics and anther one for application logic. Many patterns and type of architecture are required.

1.3.2 Big Data Architecture

1.3.2.1 Lambda Architecture

is a data-processing architecture designed to handle massive quantities of data by taking advantage of both batch- and stream-processing methods. This approach to architecture attempts to

balance latency, throughput, and fault-tolerance by using batch processing to provide comprehensive and accurate views of batch data, while simultaneously using real-time stream processing to provide views of online data. The two view outputs may be joined before presentation. The rise of lambda architecture is correlated with the growth of big data, real-time analytics, and the drive to mitigate the latencies of map-reduce. Lambda architecture depends on a data model with an append-only, immutable data source that serves as a system of record.It is intended for ingesting and processing timestamped events that are appended to existing events rather than overwriting them. State is determined from the natural time-based ordering of the data.

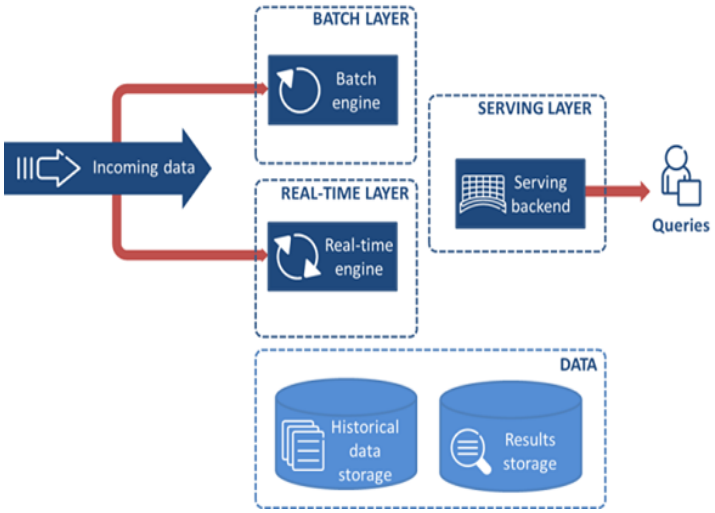

FIGURE 1.3. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

Batch Layer Precomputes results using a distributed processing system that can handle very large quantities of data. The batch layer aims at perfect accuracy by being able to process all available data when generating views. This means it can fix any errors by recomputing based on the complete data set, then updating existing views. Output is typically stored in a read-only database, with updates completely replacing existing precomputed views.

Streaming Layer The speed layer processes data streams in real time and without the requirements of fix-ups or completeness. This layer sacrifices throughput as it aims to minimize latency by providing real-time views into the most recent data. Essentially, the speed layer is responsible for filling the "gap" caused by the batch layer's lag in providing views based on the most recent data. This layer's views may not be as accurate or complete as the ones eventually produced by the batch layer, but they are available almost immediately after data is received, and can be replaced when the batch layer's views for the same data become available.

Serving Layer Output from the batch and speed layers are stored in the serving layer, which responds to ad-hoc queries by returning precomputed views or building views from the processed data.

1.3.2.2 Kappa Architecture

is a simplification of Lambda Architecture. A Kappa Architecture system is like a Lambda Architecture system with the batch processing system removed. To replace batch processing, data is simply fed through the streaming system quickly.

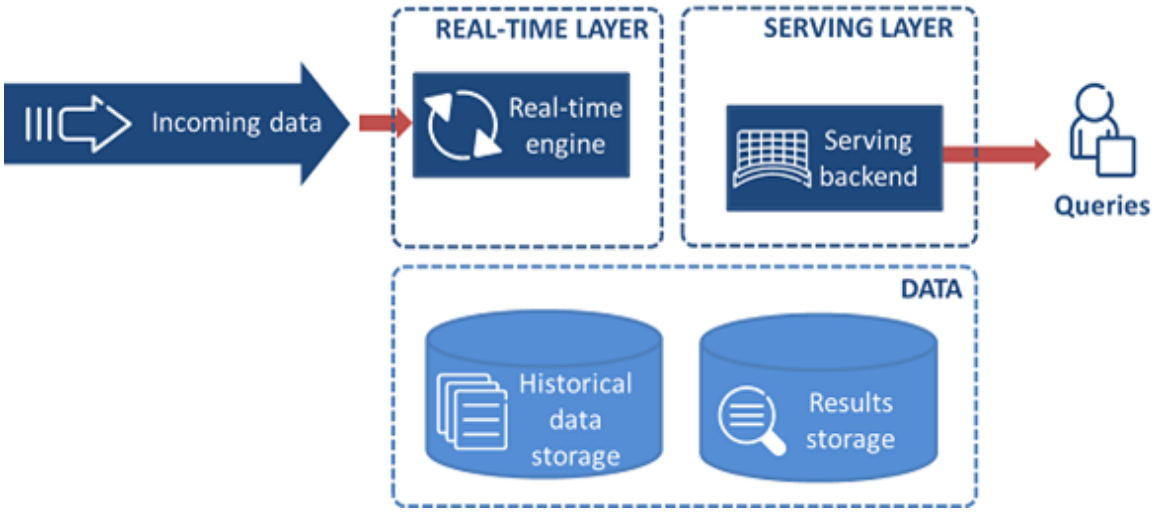

FIGURE 1.4. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

1.3.2.3 Synthesis

The question now is which one fit the best for us? In fact, choosing whether to work with lambda or kappa architecture depends on the use case, so if we treat the same way streaming and existing data the kappa architecture would be a great solution. And if we treat them differently, we should implement the lambda one. In our case, promotions will be recommended upon historical and streaming data. So the best architecture for our case is the lambda architecture. Still, one thing, Which tools shall we use!

1.3.3 Application Architecture

we are developing a server-side enterprise application that handle the logic of application. It must support a variety of different clients including desktop browsers, mobile browsers and native mobile applications. This application might also expose some APIs for 3rd parties to consume. It

might also integrate with other applications via either web services or a message broker. The application handles requests (HTTP requests and messages) by executing business logic; accessing a database; exchanging messages with other systems; and returning a HTML/JSON/XML response. There are logical components corresponding to different functional areas of the application.

1.3.3.1 Monolithic application

One of the solutions is to build an application with a monolithic architecture.The application consists of several components and deployed as a single monolithic application. For example, a Java web application consists of a single WAR file that runs on a web container such as Tomcat. A Rails application consists of a single directory hierarchy deployed using either, for example, Phusion Passenger on Apache/Nginx or JRuby on Tomcat. we can run multiple instances of the application behind a load balancer in order to scale and improve availability.

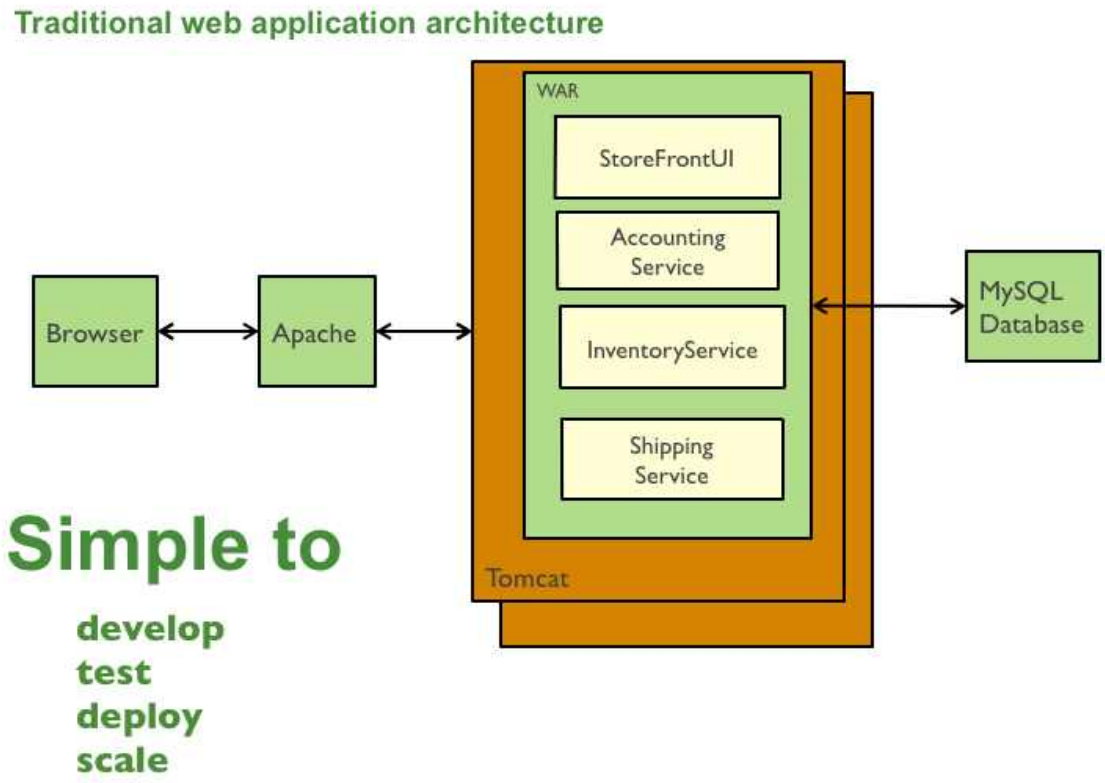

FIGURE 1.5. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

This solution has a number of benefits:

- Simple to develop - the goal of current development tools and IDEs is to support the development of monolithic applications

- Simple to deploy - you simply need to deploy the WAR file (or directory hierarchy) on the appropriate runtime

- Simple to scale - you can scale the application by running multiple copies of the application behind a load balancer

However, once the application becomes large and the team grows in size, this approach has a number of drawbacks that become increasingly significant:

- The large monolithic code base intimidates developers, especially ones who are new to the team. The application can be difficult to understand and modify. As a result, development typically slows down. Also, because there are not hard module boundaries, modularity breaks down over time. Moreover, because it can be difficult to understand how to correctly implement a change the quality of the code declines over time. It's a downwards spiral.

- Overloaded IDE - the larger the code base the slower the IDE and the less productive developers are.

- Overloaded web container - the larger the application the longer it takes to start up. This had have a huge impact on developer productivity because of time wasted waiting for the container to start. It also impacts deployment too.

- Continuous deployment is difficult - a large monolithic application is also an obstacle to frequent deployments. In order to update one component you have to redeploy the entire application. This will interrupt background tasks (e.g. Quartz jobs in a Java application), regardless of whether they are impacted by the change, and possibly cause problems. There is also the chance that components that haven't been updated will fail to start correctly. As a result, the risk associated with redeployment increases, which discourages frequent updates. This is especially a problem for user interface developers, since they usually need to iterative rapidly and redeploy frequently.

- Scaling the application can be difficult - a monolithic architecture is that it can only scale in one dimension. On the one hand, it can scale with an increasing transaction volume by running more copies of the application. Some clouds can even adjust the number of instances dynamically based on load. But on the other hand, this architecture can't scale with an increasing data volume. Each copy of application instance will access all of the data, which makes caching less effective and increases memory consumption and I/O traffic. Also, different application components have different resource requirements - one might be CPU intensive while another might memory intensive. With a monolithic architecture we cannot scale each component independently

- Obstacle to scaling development - A monolithic application is also an obstacle to scaling development. Once the application gets to a certain size its useful to divide up the engineering organization into teams that focus on specific functional areas. For example, we might want to have the UI team, accounting team, inventory team, etc. The trouble with a monolithic application is that it prevents the teams from working independently. The teams must coordinate their development efforts and redeployments. It is much more difficult for a team to make a change and update production.

- Requires a long-term commitment to a technology stack - a monolithic architecture forces you to be married to the technology stack (and in some cases, to a particular version of that technology) you chose at the start of development . With a monolithic application, can be difficult to incrementally adopt a newer technology. For example, let's imagine that you chose the JVM. You have some language choices since as well as Java you can use other JVM languages that inter-operate nicely with Java such as Groovy and Scala. But components written in non-JVM languages do not have a place within your monolithic architecture. Also, if your application uses a platform framework that subsequently becomes obsolete then it can be challenging to incrementally migrate the application to a newer and better framework. It's possible that in order to adopt a newer platform framework you have to rewrite the entire application, which is a risky undertaking.

1.3.3.2 Microservices Architecture

another solution is to define an architecture that structures the application as a set of loosely coupled, collaborating services. This approach corresponds to the Y-axis of the Scale Cube. Each service implements a set of narrowly, related functions. For example, an application might consist of services such as the order management service, the customer management service etc.Services communicate using either synchronous protocols such as HTTP/REST or asynchronous protocols such as AMQP. Services can be developed and deployed independently of one another. Each service has its own database in order to be decoupled from other services. Data consistency between services is maintained using an event-driven architecture .

This solution has a number of benefits:

- Each microservice is relatively small (Easier for a developer to understand, The IDE is faster making developers more productive, The application starts faster, which makes developers more productive, and speeds up deployments)

- Each service can be deployed independently of other services - easier to deploy new versions of services frequently

- Easier to scale development. It enables you to organize the development effort around multiple teams. Each (two pizza) team is owns and is responsible for one or more single

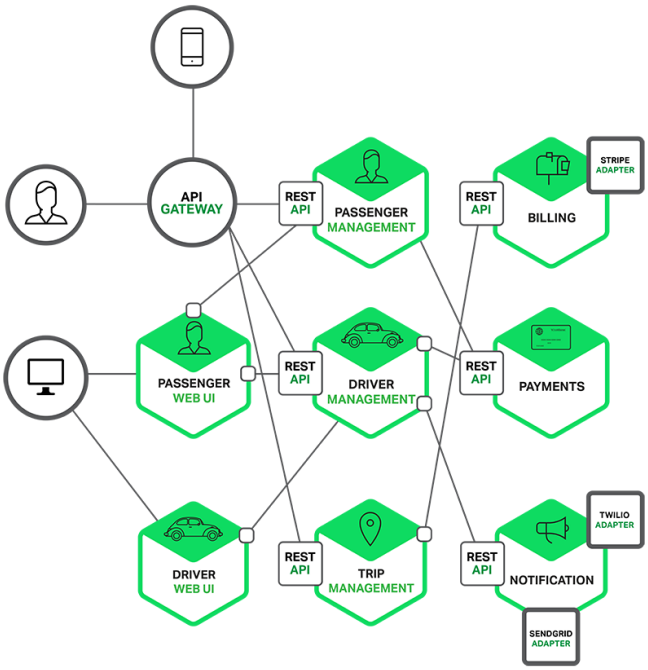

FIGURE 1.6. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

service. Each team can develop, deploy and scale their services independently of all of the other teams.

- Improved fault isolation. For example, if there is a memory leak in one service then only that service will be affected. The other services will continue to handle requests. In comparison, one misbehaving component of a monolithic architecture can bring down the entire system.

- Each service can be developed and deployed independently

- Eliminates any long-term commitment to a technology stack. When developing a new service you can pick a new technology stack. Similarly, when making major changes to an existing service you can rewrite it using a new technology stack.

This solution has a number of drawbacks:

- Developers must deal with the additional complexity of creating a distributed system (Developer tools/IDEs are oriented on building monolithic applications and don't provide explicit support for developing distributed applications. Testing is more difficult , Developers must implement the inter-service communication mechanism , Implementing use cases that span multiple services without using distributed transactions is difficult , Implementing use cases that span multiple services requires careful coordination between the teams)

- Deployment complexity. In production, there is also the operational complexity of deploying and managing a system comprised of many different service types.
- Increased memory consumption. The microservice architecture replaces N monolithic application instances with NxM services instances. If each service runs in its own JVM (or equivalent), which is usually necessary to isolate the instances, then there is the overhead of M times as many JVM runtimes. Moreover, if each service runs on its own VM (e.g. EC2 instance), as is the case at Netflix, the overhead is even higher.

1.3.4 Synthesis

Choosing whether to work with microservices or monolithic application is depending on the project maturity. So when developing the first version of an application, we often do not have the problems that the microservice approach solves. Moreover, using an elaborate, distributed architecture will slow down development. But looking to the benefits that offer the microservices architecture is really impressive.And as this project is dedicated to a big company with a big customer range. This application should be apt to scale up quickly and ready to add so much features. So implementing the microservices architecture is a great choice. The question now is which tools or patterns shall we use ?

1.4 Development methodology

1.4.1 Introduction

2TUP is a unified process (i.e. a software development process) built on the UML modeling language. Every process answers the following main characteristics:

- It is an incremental process, allowing a better technical and functional risk management and thus constituting the deadlines and the costs control.
- It is an iterative process. The degrees of abstraction are increasingly precise at each iteration.
- It is component oriented, offering flexibility to the model and supporting the re-use.
- It is user oriented because built from their expectations.

The 2TUP process answers to the constraints of change of the information systems subjected themselves to two types of constraints: functional constraints and technical constraints as shows it the following diagram: In concrete terms, the process is modeled by two branches (tracks):

- A functional track (capitalization of knowledge trade)
- A technical track (re-use of a technical knowhow).

FIGURE 1.7. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

Then these two tracks amalgamate for the realization of the system. This is why this process is still called Yshaped process. With this development process, a model is essential in order to anticipate the results. A model can be used with each step of the development with an increasing detailed manner. The industrial standard of object modeling, UML, was selected as the development tool. It appeared very difficult to consider the 2TUP process without using UML and, more particularly UML 2.0 which support the oriented design component.

1.4.2 Implementation

1.4.2.1 Introduction

The first phase of the implementation starts with the preliminary study which introduces the project. The specifications, initial document of the functional and technical needs, are partly the result of this work. It is supplemented by the modeling of the total system context. Not everything is to be described at this stage but simply to identify the external entities interacting (actors), to list the interactions (messages) and to represent this unit on a model (context).

1.4.2.2 Feature track

The functional branch makes an inventory of the functional needs and analyzes it. This phase formalizes and specifies the elements of the preliminary study. The applied use case technique translates the whole interactions between the system and the actors. The obtained use cases are then organized (treated on a hierarchical basis, generalized, specialized...). They make it possible to identify the classes and they permit the oriented object modeling generated in the analysis part.

1.4.2.3 Technical track

The technical branch lists the technical needs and proposes a generic design validated by a prototype. The pre-necessary techniques revealed in the preliminary study, showing the operational needs and the strategic choices of development, lead to the development of the construction process. To do this, several stages are necessary:

- The inventory of technical specifications related to the hardware.

- The inventory of the software specifications.

1.4.2.4 Middle track

The medium branch supports the preliminary design, the detailed design, coding, the tests and the validation. The preliminary design is one of the most sensitive steps of the 2TUP process. It represents the fusion of the functional and technical tracks. It finishes when the deployment model (working stations, architectures), the operating model (components, applications), the logical model (classes diagrams, classes), interfaces (users and components) and the software configuration model are defined.

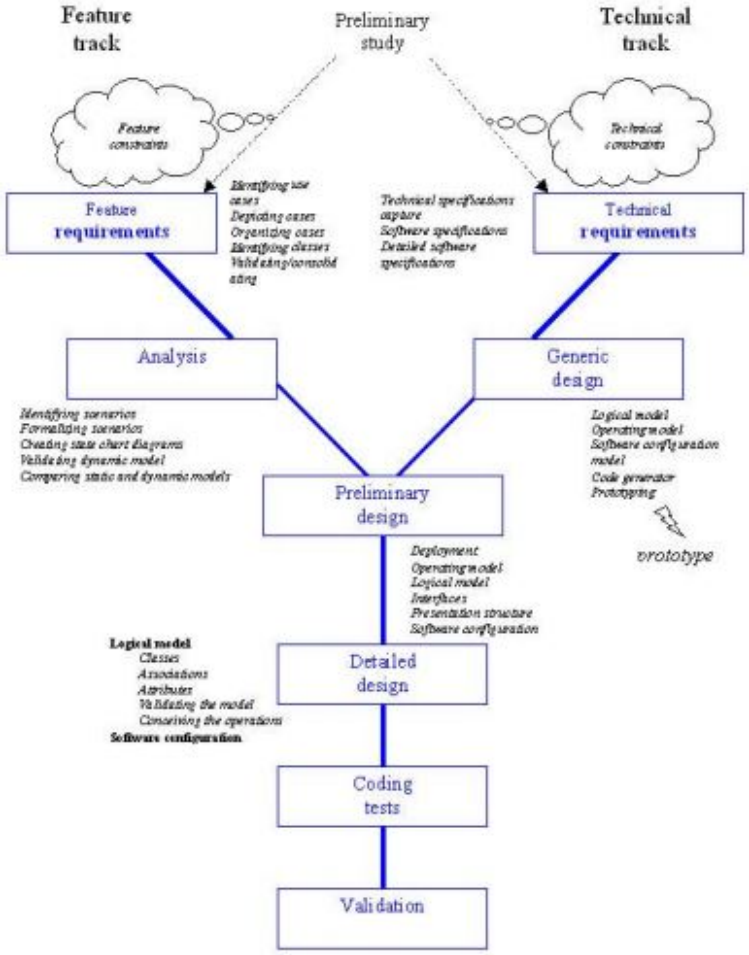

FIGURE 1.8. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

The detailed design conceives and documents very exactly the code that will be generated. It is largely founded on UML representations and implements, in an iterative manner, a system construction process to obtain a "model ready to code".

The various components carried out in the detailed design are coded. The code units obtained are tested as they are written and produced. The validation, called ‚Äúrecipe‚Äù step, consists of the homologation of the developed system functions.

1.5 Analysis and design methodology

1.5.1 Introduction

Large enterprise applications - the ones that execute core business applications, and keep a company going - must be more than just a bunch of code modules. They must be structured in a way that enables scalability, security, and robust execution under stressful conditions, and their structure - frequently referred to as their architecture - must be defined clearly enough that maintenance programmers can (quickly!) find and fix a bug that shows up long after the original authors have moved on to other projects. That is, these programs must be designed to work perfectly in many areas, and business functionality is not the only one (although it certainly is the essential core). Of course a well-designed architecture benefits any program, and not just the largest ones as we've singled out here. We mentioned large applications first because structure is a way of dealing with complexity, so the benefits of structure (and of modeling and design, as we'll demonstrate) compound as application size grows large. Another benefit of structure is that it enables code reuse: Design time is the easiest time to structure an application as a collection of self-contained modules or components. Eventually, enterprises build up a library of models of components, each one representing an implementation stored in a library of code modules. When another application needs the same functionality, the designer can quickly import its module from the library. At coding time, the developer can just as quickly import the code module into the application.

Modeling is the designing of software applications before coding. Modeling is an Essential Part of large software projects, and helpful to medium and even small projects as well. A model plays the analogous role in software development that blueprints and other plans (site maps, elevations, physical models) play in the building of a skyscraper. Using a model, those responsible for a software development project's success can assure themselves that business functionality is complete and correct, end-user needs are met, and program design supports requirements for scalability, robustness, security, extendibility, and other characteristics, before implementation in code renders changes difficult and expensive to make. Surveys show that large software projects have a huge probability of failure - in fact, it's more likely that a large software application will fail to meet all of its requirements on time and on budget than that it will succeed.

1.5.2 Abstraction

Models help us by letting us work at a higher level of abstraction. A model may do this by hiding or masking details, bringing out the big picture, or by focusing on different aspects of the

prototype. In UML 2.0, we can zoom out from a detailed view of an application to the environment where it executes, visualizing connections to other applications or, zoomed even further, to other sites. Alternatively, we can focus on different aspects of the application, such as the business process that it automates, or a business rules view. The new ability to nest model elements, added in UML 2.0, supports this concept directly.

The OMG's Unified Modeling Language (UML) helps specify, visualize, and document models of software systems, including their structure and design, in a way that meets all of these requirements. (We can use UML for business modeling and modeling of other non-software systems too.) Using any one of the large number of UML-based tools on the market, we can analyze our future application's requirements and design a solution that meets them, representing the results using UML 2.0's thirteen standard diagram types.

We can model just about any type of application, running on any type and combination of hardware, operating system, programming language, and network, in UML. Its flexibility let us model distributed applications that use just about any middleware on the market. Built upon fundamental OO concepts including class and operation, it's a natural fit for object-oriented languages and environments such as C++, Java, but we can use it to model non-OO applications as well in, for example, Fortran, VB, or COBOL. UML Profiles (that is, subsets of UML tailored for specific purposes) help us model Transactional, Real-time, and Fault-Tolerant systems in a natural way.

1.5.3 Models VS. Methodologies

The process of gathering and analyzing an application's requirements, and incorporating them into a program design, is a complex one and the industry currently supports many methodologies that define formal procedures specifying how to go about it. One characteristic of UML - in fact, the one that enables the widespread industry support that the language enjoys - is that it is methodology-independent. Regardless of the methodology that we use to perform our analysis and design, we can use UML to express the results. And, using XMI (XML Metadata Interchange, another OMG standard), we can transfer our UML model from one tool into a repository, or into another tool for refinement or the next step in our chosen development process. These are the benefits of standardization! UML 2.0 defines thirteen types of diagrams, divided into three categories: Six diagram types represent static application structure; three represent general types of behavior; and four represent different aspects of interactions:

Structure Diagrams include the Class Diagram, Object Diagram, Component Diagram, Composite Structure Diagram, Package Diagram, and Deployment Diagram. Behavior Diagrams include the Use Case Diagram (used by some methodologies during requirements gathering); Activity Diagram, and State Machine Diagram. Interaction Diagrams, all derived from the more general Behavior Diagram, include the Sequence Diagram, Communication Diagram, Timing Diagram, and Interaction Overview Diagram.

1.5.4 UML 2.0

OMG have already integrated the new features into this writeup, and here's a summary:

Nested Classifiers: This is an extremely powerful concept. In UML, almost every model building block we work with (classes, objects, components, behaviors such as activities and state machines, and more) is a classifier. In UML 2.0, we can nest a set of classes inside the component that manages them, or embed a behavior (such as a state machine) inside the class or component that implements it. This capability also let us build up complex behaviors from simpler ones, the capability that defines the Interaction Overview Diagram. We can layer different levels of abstraction in multiple ways: For example, we can build a model of our Enterprise, and zoom in to embedded site views, and then to departmental views within the site, and then to applications within a department. Alternatively, we can nest computational models within a business process model. OMG's Business Enterprise Integration Domain Task Force (BEI DTF) is currently working on several interesting new standards in business process and business rules. Improved Behavioral Modeling: In UML 1.X, the different behavioral models were independent, but in UML 2.0, they all derive from a fundamental definition of a behavior (except for the Use Case, which is subtly different but still participates in the new organization). Improved relationship between Structural and Behavioral Models: As we pointed out under Nested Classifiers, UML 2.0 let us designate that a behavior represented by (for example) a State Machine or Sequence Diagram is the behavior of a class or a component. That is, the new language goes well beyond the Classes and Objects well-modeled by UML 1.X to add the capability to represent not only behavioral models, but also architectural models, business process and rules, and other models used in many different parts of computing and even non-computing disciplines.

1.6 Conclusion

Two main parts are waiting to be implemented, the first one is the lambda architecture and the data analytics strategy. the second one is the microservices approach and the application logic.

CHAPTER 2

REQUIREMENTS ANALYSIS AND SPECIFICATION

After defining the project scope and making a preliminary study. In this chapter, we will deep dive in two parts, one is the functional branchy part which presents the application logic and the other one is the technical branch which present the infrastructure support.

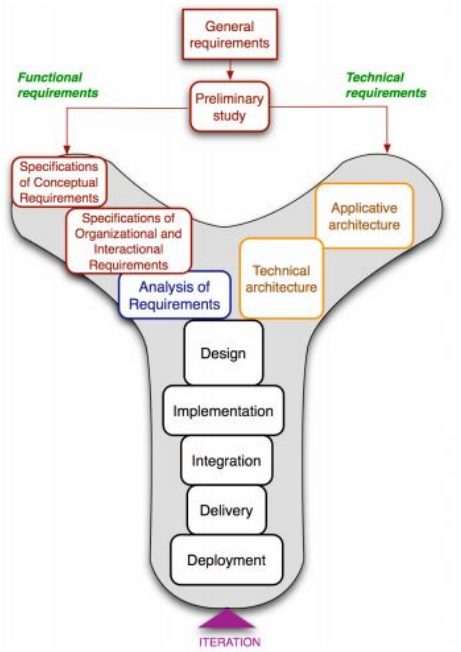

FIGURE 2.1. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

2.1 Functional branch

The functional (left) branch corresponds to the traditional task of domain and user requirements modeling, independently from technical aspects. Considering the design of a Mixed Reality system. It includes interaction scenarios, task analysis, interaction modality choices and mock-ups. This branch ends by structuring the domains with Interactional and Business objects required to implement the Mixed Reality system.

2.1.1 Requirements analysis

2.1.1.1 Functional requirements

Our application should be able to provide the following requirements

ID	Nom	Description
1	Track users behavior	As administrator i would like to be able to track users depending on thier geolocations and interests
1	Account managment	As administrator i would like to be able to track users depending on thier geolocations and interests
1	Promotion managment	As administrator i would like to be able to track users depending on thier geolocations and interests
2	Recommend promotions and services	As administrator i would like to be able to automatically recommend services, promotions depending on users locations and interests.
3	Analyse users data flow	As administrator i would like to be able to analyse and extract informations from users data flow
4	Promote a service	As third party i would like to be able to promote my services and target a users segment
5	Benifit from services and promotions	As a client application user i would like to be able to see near promotions and services depending on my interests

2.1.1.2 Non functional requirements

Scalability Scalability is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged to accommodate that growth. For example, a system is considered scalable if it is capable of increasing its total output under an increased load when resources (typically hardware) are added. An analogous meaning is implied when the word is used in an economic context, where a company's scalability implies that the underlying business model offers the potential for economic growth within the company.

Operability Operability is the ability to keep an equipment, a system or a whole industrial binstallation in a safe and reliable functioning condition, according to pre-defined operational requirements.

Security Security is the degree of resistance to, or protection from, harm. It applies to any vulnerable and/or valuable asset, such as a person, dwelling, community, item, nation, or organization.

Fault tolerance Fault tolerance is the property that enables a system to continue operating properly in the event of the failure of (or one or more faults within) some of its components. If its operating quality decreases at all, the decrease is proportional to the severity of the failure, as compared to a naively designed system in which even a small failure can cause total breakdown. Fault tolerance is particularly sought after in high-availability or life-critical systems. The ability of maintaining functionality when portions of a system break down is referred to as graceful degradation.

Extensibility Extensibility is a software engineering and systems design principle where the implementation takes future growth into consideration. The term extensibility can also be seen as a systemic measure of the ability to extend a system and the level of effort required to implement the extension. Extensions can be through the addition of new functionality or through modification of existing functionality.

2.1.2 Requirements specification

In order to properly establish the planned objectives during our development process, we will specify in more detail the previous requirements using the UML design methodology.

2.1.2.1 Actors identification

Starting from the last requirements collection , we identify three main actors that interact continuously with our application.

- Administrator - the administrator is able to see users data analytics , track them spatially and manage users , promotions and services .

- Third party - the third party is able to submit a promotion via an application portail.
- User - the user is able to see promotions and services depending on his interests.

2.1.2.2 Use case modelization

This phase aims to describe the expected behavior of the application. For this purpose, we use the use case diagram as an essential element of object-oriented modeling. It makes it possible to model the functionalities of the application .

Data analytics use case diagram :

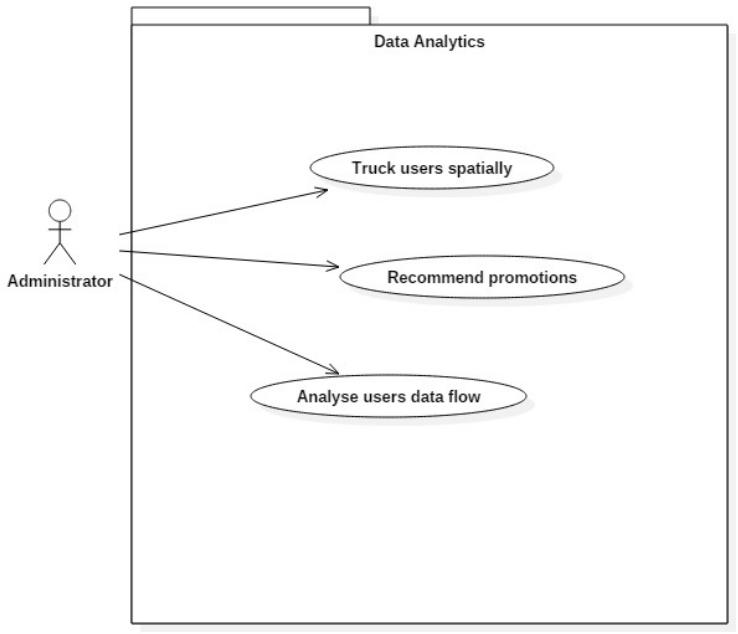

FIGURE 2.2. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

Use case identification :

Title : Data analytics

Summary : Our application administrator is able to truck users spatially and analyse their data flow , relaying on the big data engine , to recommand the most apporiate promotion depending on thier intersts and locations.

Actors : administrator

Senarios description :

Precondition : This functionality user require the administrator privilege.

Post-condition : Users could receive administrator logic promotions recommendation.

Nominal Scenario :

1. Administrator login to dashboard
2. Administrator choose between Dashboard and World Map
3. If Administrator choose dashboard
    a) administrator is able to see data flow analytics in deferent type of charts
4. If Administrator choose world map
    a) administrator is able to see users Heatmap

Alternative Scenario :
There is no alternative senario.

Error Scenario :
If the dashboard could not get informations form the microservice then the dashboard show intiale informations.

Use case diagram of account managment :

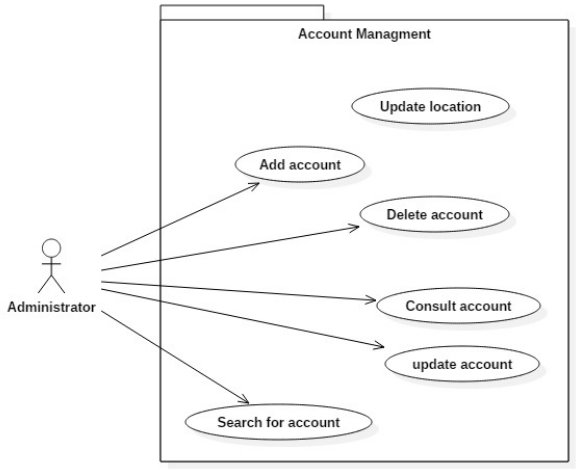

FIGURE 2.3. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

Use case identification :
Title : Account managment
Summary : Our application administrator is able to add, remove, update , search and consult users. Our system provide an update position functionality that allwo real time user position track. Mobile client application send periodically the device position to the account microservice.

Actors : administrator

Senarios description :
Precondition : This functionality user require the administrator privilege.
Post-condition : Administrator could mange users account and track their positions.

Nominal Scenario :

1. Administrator login to dashboard
2. Administrator choose account managment
3. Administrator could search, add, update, delete and consult an account.

Alternative Scenario : there is no alternative senario.

Error Scenario : If the administrator was not able to do the opration then he will be notified.

Promotions managment :

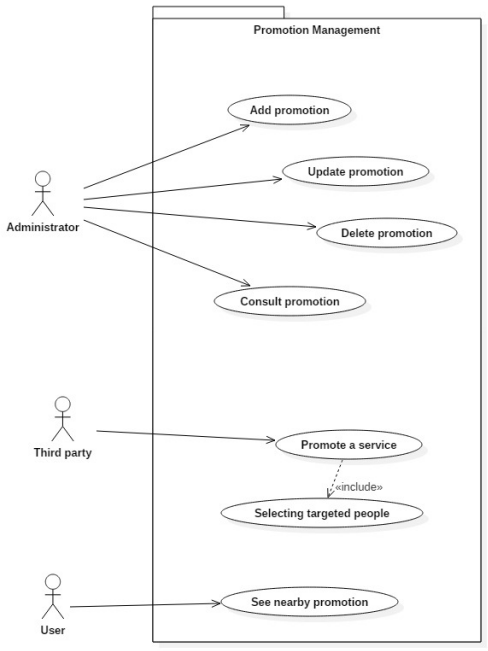

FIGURE 2.4. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

Use case identification :

Title : Promotion managment

Summary : Our application administrator is able to search, add, delete, update and consult promotions, Third party representative is able to make to promote a serive and select wich kind of people his intersted about and user can see promotion and take advantage of promotions services.

Actors : administrator , Third party and user

Senarios description :

Precondition : This functionality user require a varity of privilege.

Post-condition : Promotions are well exposed to app users.

Nominal Scenario :

1. Administrator login to dashboard
2. Administrator choose Promotion managment
3. administrator is able to search, add, update, delete, consult promotion

1. Third party representative login to dashboard
2. Choose Promotion managment
3. He is able to search, add, update, delete and consult his promotions

1. User login to the mobile application
2. User is able to see promotions

Alternative Scenario : There is no alternative senario.

Error Scenario : If the any application user (administrator, third party representative, mobile application user) was not able to do the opration then he will be notified.

2.1.2.3 Sequence diagrams

With sequence diagrams, the UML provides a graphical way to represent interactions between objects over time. These diagrams typically show an actor, the objects and the actors with whom he interacts during the execution of the use case. In this section, we present some sequence diagrams to describe the different interactions between the user and the application

Sequence Diagram for the Authentication Scenario :

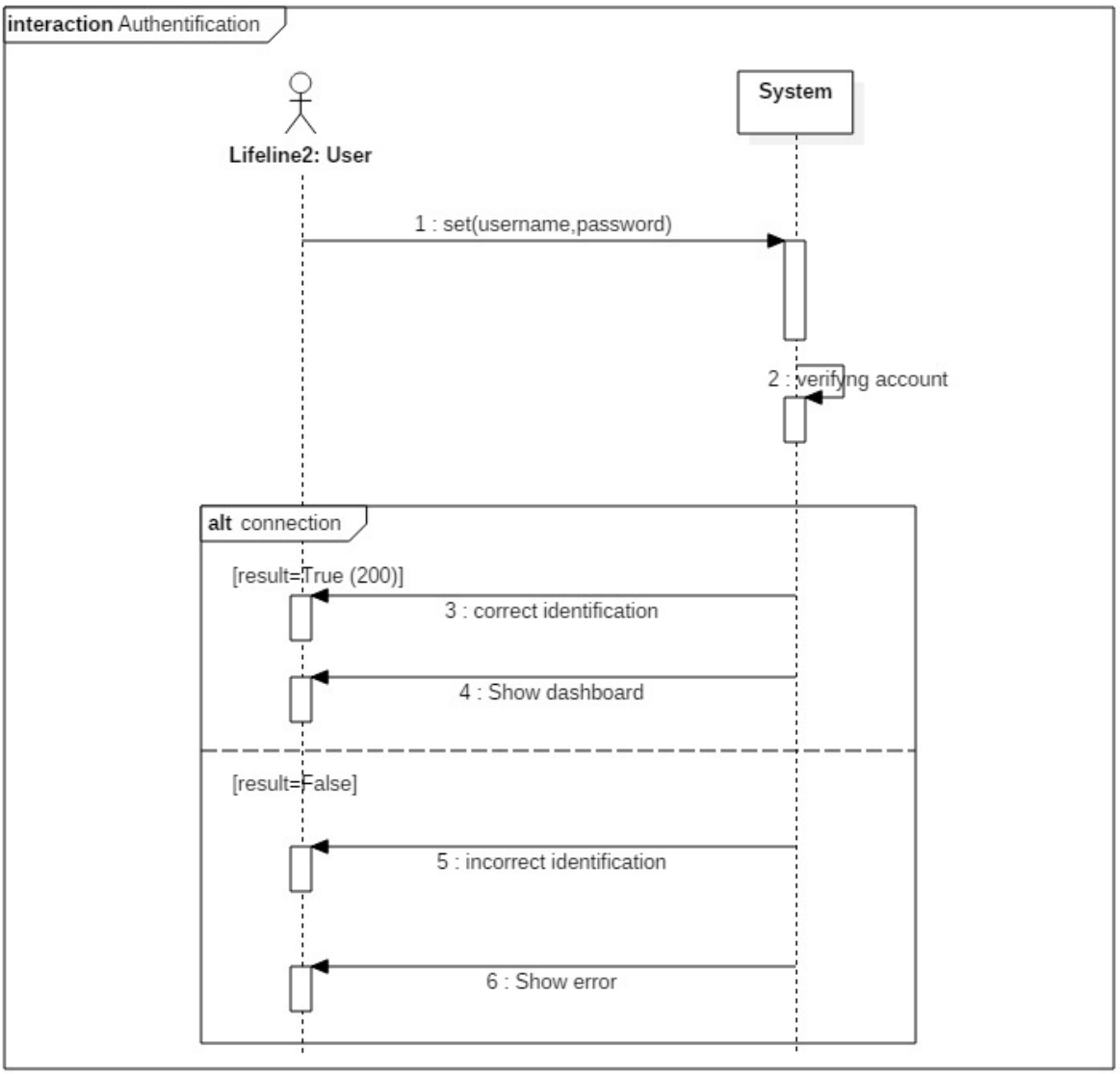

FIGURE 2.5. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

Sequence Diagram for account managment :

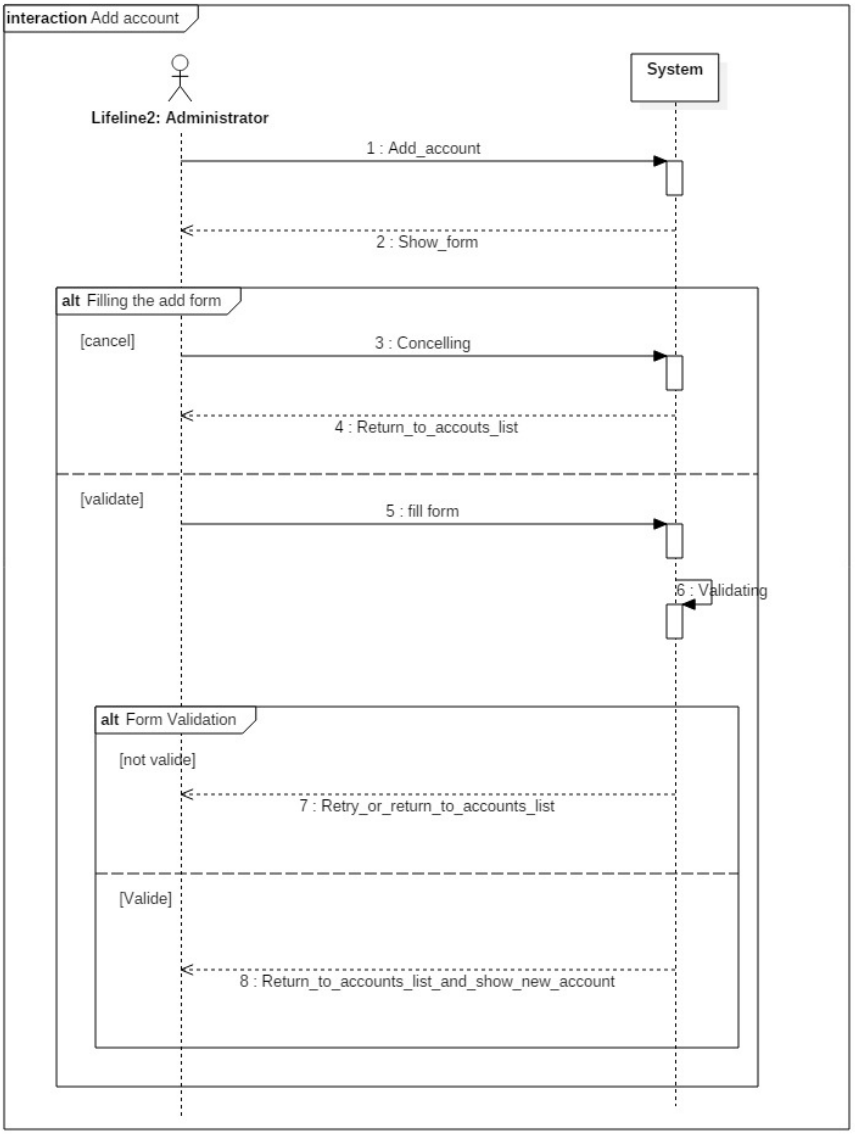

FIGURE 2.6. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

Conclusion this phase allows us to initiate to the following chapter which deals with our application design. In the next section we will deep dive in technical specification.

2.2 Technical branch

Since the functional requirements have been defined, it's time to choose which tools shall we use and especially on which software architecture we will be reposing.

2.2.1 Big Data architecture

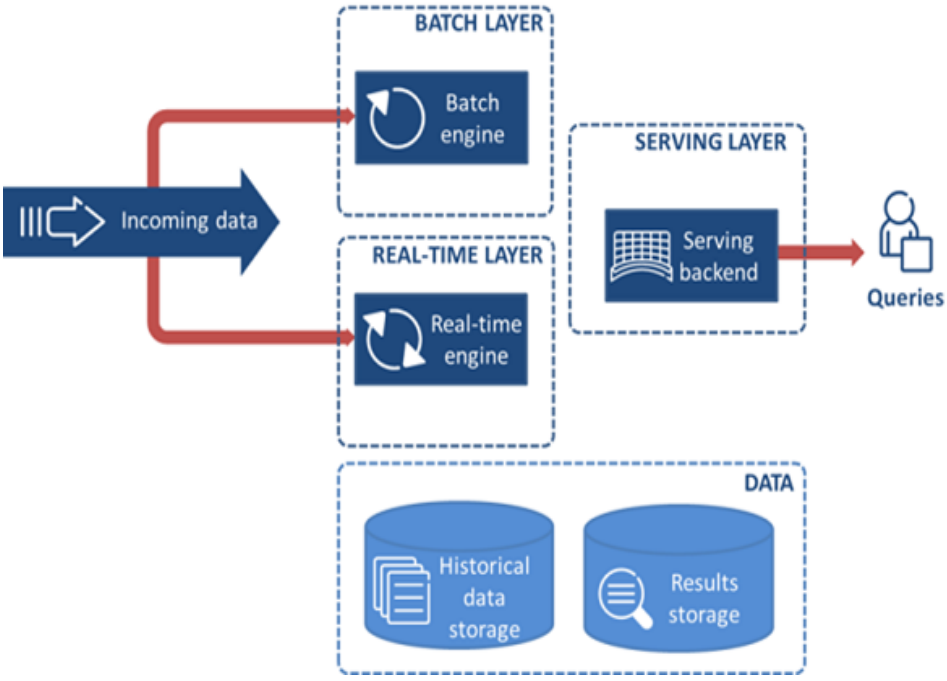

FIGURE 2.7. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

In the last chapter, we choosed to go with the lambda approach. the question now is how to choose the right tools. So in the next sections we will be making a comparative study in each field (yellow tags on the previous figure) to pick the most appropriate tools.

2.2.1.1 OSS Message brokers layer

The incoming data could be from different sources . Data should be gathered in one data flow pipeline. Otherwise we will find ourselves dealing with a complexe grid , providing all apis compatibility and resolving extras problems.

A great solution for this problem could be a messaging layer that gives a various guarantees of message persistence and delivery.In this section , we will introduce some of the best message brokers that can handle this activity.So our Competitors are SQS AWS, Apache Kafka, Mongodb message queue, HornetQ, RabbitMQ, Apache ActiveMQ

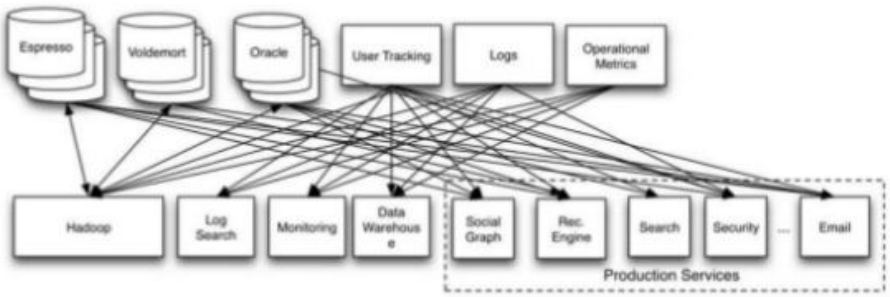

FIGURE 2.8. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

SQS AWS SQS, Simple Message Queue, is a message-queue-as-a-service offering from Amazon Web Services. It supports only a handful of messaging operations but thanks to the easy to understand interfaces, and the as-a-service nature. SQS guarantees that if a send completes, the message is replicated to multiple nodes. It also provides at-least-once delivery guarantees. We don't really know how SQS is implemented, A single thread on single node achieves 430 msgs/s sent and the same number of msgs received. These results are not impressive, but SQS scales nicely both when increasing the number of threads, and the number of nodes . On a single node, with 50 threads, we can send up to 14 500 msgs/s, and receive up to 4 200 msgs/s. On an 8-node cluster, these numbers go up to 63 500 msgs/s sent, and 34 800 msgs/s received.

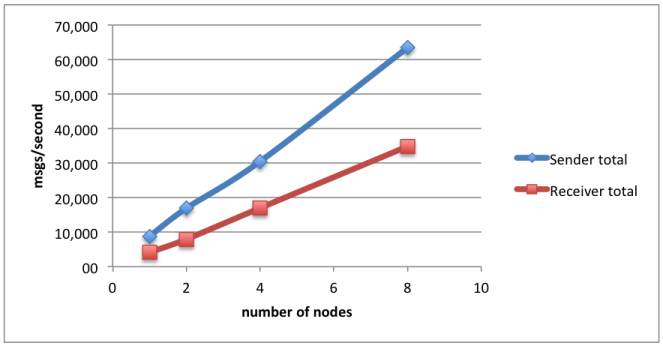

FIGURE 2.9. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

Apache Kafka Apache Kafka is an open-source stream processing platform developed by the Apache Software Foundation written in Scala and Java. The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds. Its storage layer is essentially a massively scalable pub/sub message queue architected making it highly valuable for enterprise infrastructures to process streaming data. Kafka's performance is great. A single-node single-thread achieves about 2 550 msgs/s, and the best result was 33 500msgs/s with 25 sending and receiving threads and 4 nodes.

THREADS	NODES	SEND MSGS/S	RECEIVE MSGS/S
1	1	2 558,81	2 561,10
5	1	11 804,39	11 354,90
25	1	29 691,33	27 093,58
1	2	5 879,21	5 847,03
5	2	14 660,90	13 552,35
25	2	29 373,50	27 822,50
5	4	22 271,50	20 757,50
25	4	**33 587,00**	**31 891,00**

FIGURE 2.10. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

Mongodb message queue Mongo has two main features which make it possible to easily implement a durable, replicated message queue on top of it. A single-thread, single-node setup achieves 7 900 msgs/s sent and 1 900 msgs/s received. The maximum send throughput with multiple thread/nodes is about 10 500 msgs/s, while the maximum receive rate is 3 200 msgs/s, when using the ‚Äúsafe‚Äù write concern.

THREADS	NODES	SEND MSGS/S	RECEIVE MSGS/S
1	1	7 968,60	1 914,05
5	1	9 903,47	3 149,00
25	1	**10 903,00**	**3 266,83**
1	2	9 569,99	2 779,87
5	2	10 078,65	3 112,55
25	2	7 930,50	3 014,00

FIGURE 2.11. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

HornetQ HornetQ is an open source asynchronous messaging project from JBoss. It is an example of Message-oriented middleware. HornetQ is an open source project to build a multi-protocol, embeddable, very high performance, clustered, asynchronous messaging system. Speaking of performance, HornetQ is very good! A single-node, single-thread setup achieves 1 100 msgs/s. With 25 threads, we are up to 12 800 msgs/s! And finally, with 25 threads and 4 nodes, we can achieve 17 000 msgs/s.

THREADS	NODES	SEND MSGS/S	RECEIVE MSGS/S
1	1	1 108,38	1 106,68
5	1	4 333,13	4 318,25
25	1	12 791,83	12 802,42
1	2	2 095,15	2 029,99
5	2	7 855,75	7 759,40
25	2	14 200,25	13 761,75
1	4	3 768,28	3 627,02
5	4	11 572,10	10 708,70
25	4	**17 402,50**	**16 160,50**

FIGURE 2.12. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

RabbitMQ RabbitMQ is one of the leading open-source messaging systems. It is written in Erlang, implements AMQP and is a very popular choice when messaging is involved. It supports both message persistence and replication, with well documented behaviour in case of e.g. partitions. Such strong guarantees are probably one of the reasons for poor performance. A single-thread, single-node gives us 310 msgs/s sent and received. This scales nicely as we add nodes, up to 1 600 msgs/s

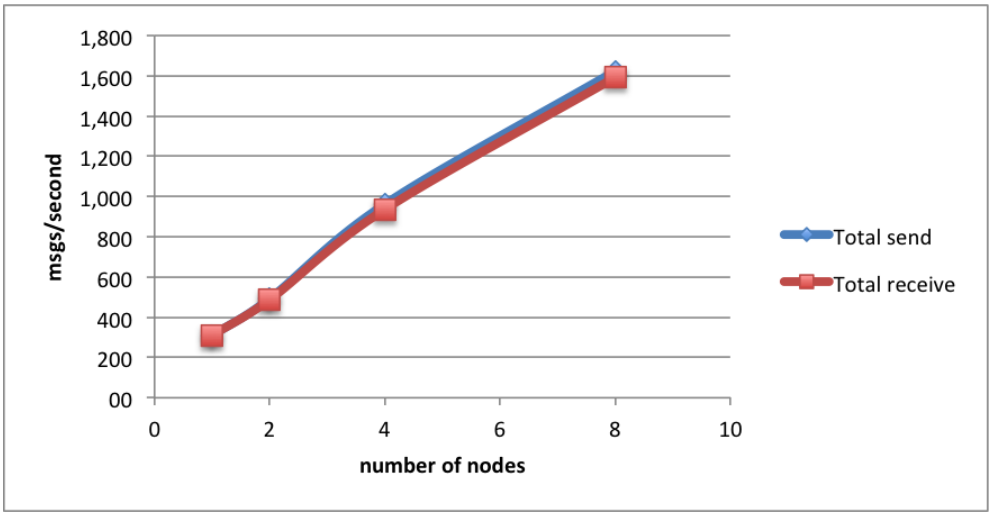

FIGURE 2.13. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

Apache ActiveMQ Apache ActiveMQ is an open source message broker written in Java together with a full Java Message Service (JMS) client. It provides Enterprise Features which in

this case means fostering the communication from more than one client or server. Supported clients include Java via JMS 1.1 as well as several other "cross language" clients. The communication is managed with features such as computer clustering and ability to use any database as a JMS persistence provider besides virtual memory, cache, and journal persistency. The next benchmark show the perfomance of ActiveMQ :

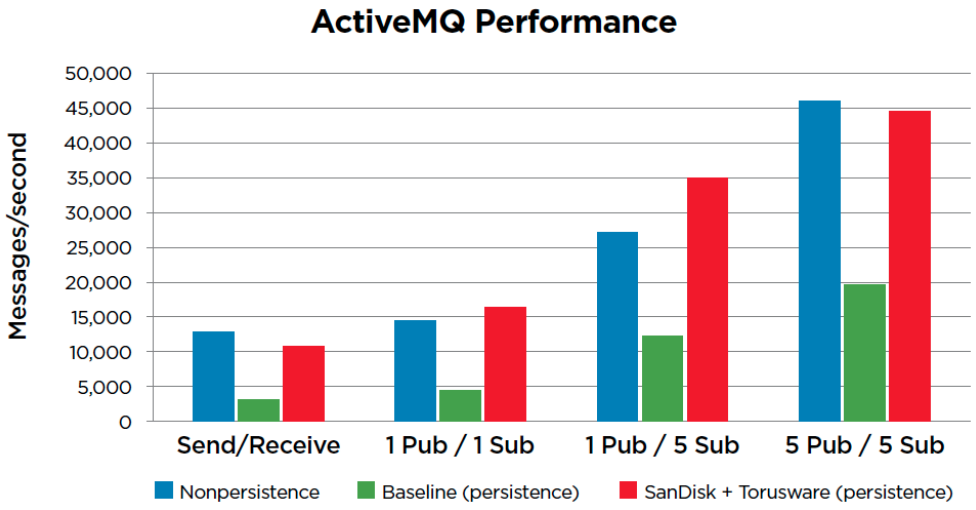

FIGURE 2.14. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

Summary Now the question is which message queue we should choose . All of the above solutions have some good sides:

- SQS is a service, so especially if you are using the AWS cloud, it's an easy choice: good performance and no setup required
- if you are using Mongo, it is easy to build a replicated message queue on top of it, without the need to create and maintain a separate messaging cluster
- if you want to have high persistence guarantees, RabbitMQ ensures replication across the cluster and on disk on message send.
- HornetQ has great performance with a very rich messaging interface and routing options
- Kafka offers the best performance and scalability
- ActiveMQ employs several modes for high availability,A robust horizontal scaling mechanism and a flexibility in configuration

When looking only at the throughput, Kafka is a clear winner (unless we include SQS with multiple nodes, but as mentioned, that would be unfair)

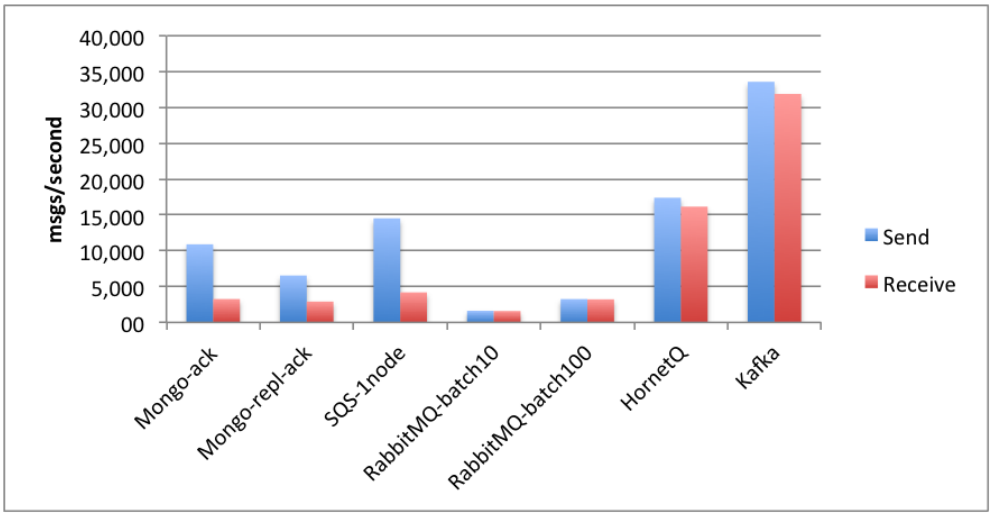

FIGURE 2.15. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

It's also interesting to see how sending more messages in a batch improves the throughput, when increasing the batch size from 10 to 100, Rabbit gets a 2x speedup, HornetQ a 1.2x speedup, and Kafka a 2.5x speedup, achieving about 89 000 msgs/s!

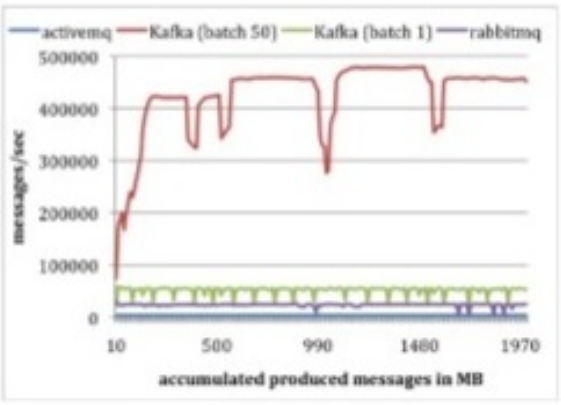

Figure 4. Producer Performance

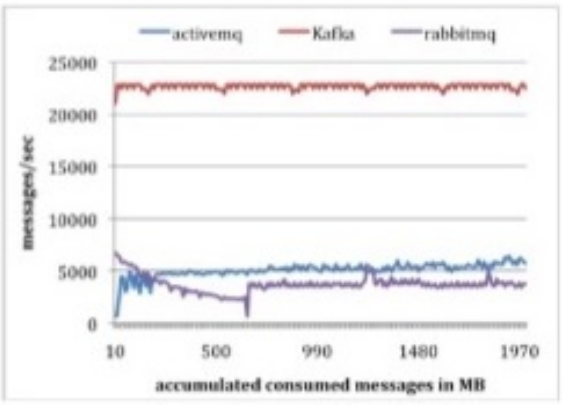

Figure 5. Consumer Performance

FIGURE 2.16. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

There are of course many other aspects besides performance, which should be taken into account when choosing a message queue, such as administration overhead, partition tolerance, feature set regarding routing, etc. And as we are interested in first with performance , we will choose Kafka for our mission .

2.2.1.2 NoSQL Databases

After performing the Data analytics logic , results are persisted in a serving layer. This layer is mostly a databes and more precisely a noSql database. In this section , we will introduce some of the best noSql databases that can handle this activity.So our Competitors are:

- **Document oriented :** Mongodb , CouchDB
- **Key Value oriented :** Redis
- **Graph oriented :** Noe4j
- **Column oriented :** Cassandra , Riak , HBase

Document oriented :
A document-oriented database, or document store, is a computer program designed for storing, retrieving and managing document-oriented information, also known as semi-structured data. Document-oriented databases are one of the main categories of NoSQL databases, and the popularity of the term document-oriented database has grown with the use of the term NoSQL itself.

Mongodb Mongodb is one of the most popular document based NoSQL database as it stores data in JSON like documents. It is non-relational database with dynamic schema. It has been developed by the founders of DoubleClick, written in C++ and is currently being used by some big companies like The New York Times, Craigslist, MTV Networks. The following are some of MongoDB benefits and strengths :

- Speed: For simple queries, it gives good performance, as all the related data are in single document which eliminates the join operations.
- Scalability: It is horizontally scalable i.e. you can reduce the workload by increasing the number of servers in your resource pool instead of relying on a stand alone resource.
- Manageable: It is easy to use for both developers and administrators. This also gives the ability to shard database
- Dynamic Schema: Its gives you the flexibility to evolve your data schema without modifying the existing data

CouchDB CouchDB is also a document based NoSQL database. It stores data in form of JSON documents. The following are some of CouchDB benefits and strengths:

- Schema-less: As a member of NoSQL family, it also have dynamic schema which makes it more flexible, having a form of JSON documents for storing data.

- HTTP query: You can access your database documents using your web browser.
- Conflict Resolution: It has automatic conflict detection which is useful while in a distributed database.
- Easy Replication: Implementing replication is fairly straight forward

Key Value oriented :
A key-value store, or key-value database, is a data storage paradigm designed for storing, retrieving, and managing associative arrays, a data structure more commonly known today as a dictionary or hash. Dictionaries contain a collection of objects, or records, which in turn have many different fields within them, each containing data. These records are stored and retrieved using a key that uniquely identifies the record, and is used to quickly find the data within the database.

Redis Redis is an Open Source NoSQL database which is mainly used because of its lightening speed. It is written in ANSI C language. The following are some of Redis benefits and strengths:

- Data structures: Redis provides efficient data structures to an extend that it is sometimes called as data structure server. The keys stored in database can be hashes, lists, strings, sorted or unsorted sets.
- Redis as Cache: You can use Redis as a cache by implementing keys with limited time to live to improve the performance.
- Very fast: It is consider as one of the fastest NoSQL server as it works with the in-memory dataset.

Graph oriented :
The graph based DBMS models represent the data in a completely different way than the previous three models. They use tree-like structureswith nodes and edges connecting each other through relations.

Noe4j Neo4j is a graph database management system developed by Neo Technology,Described by its developers as an ACID-compliant transactional database with native graph storage and processing, Neo4j is the most popular graph database. Neo4j is implemented in Java and accessible from software written in other languages using the Cypher Query Language through a transactional HTTP endpoint, or through the binary bolt protocol.

Column oriented :
A column-oriented DBMS (or columnar database management system) is a database management system (DBMS) that stores data tables by column rather than by row. Practical use of a column store versus a row store differs little in the relational DBMS world. Both columnar and row databases can use traditional database query languages like SQL to load data and perform queries.

Cassandra Apache Cassandra is the leading NoSQL, distributed database management system driving many of today's modern business applications by offering continuous availability, high scalability and performance, strong security, and operational simplicity while lowering overall cost of ownership. Cassandra has decentralized architecture. Any node can perform any operation. It provides AP(Availability,Partition-Tolerance). Cassandra has excellent single-row read performance as long as eventual consistency semantics are sufficient for the use-case.

Riak Riak is a distributed NoSQL key-value data store that offers high availability, fault tolerance, operational simplicity, and scalability.In addition to the open-source version, it comes in a supported enterprise version and a cloud storage version. Riak implements the principles from Amazon's Dynamo paper with heavy influence from the CAP Theorem. Written in Erlang, Riak has fault tolerance data replication and automatic data distribution across the cluster for performance and resilience.

HBase HBase is an open source, non-relational, distributed database modeled after Google's Bigtable and is written in Java. It is developed as part of Apache Software Foundation's Apache Hadoop project and runs on top of HDFS (Hadoop Distributed File System), providing Bigtable-like capabilities for Hadoop. That is, it provides a fault-tolerant way of storing large quantities of sparse data HBase is a column-oriented key-value data store and has been idolized widely because of its lineage with Hadoop and HDFS. HBase runs on top of HDFS and is well-suited for faster read and write operations on large datasets with high throughput and low input/output latency.

Summary :

It's too hard to pick the right database for our serving layer . In this paragraph we try to choose the most appropriate one depending on some benshMarks well known. This benchmarks elect

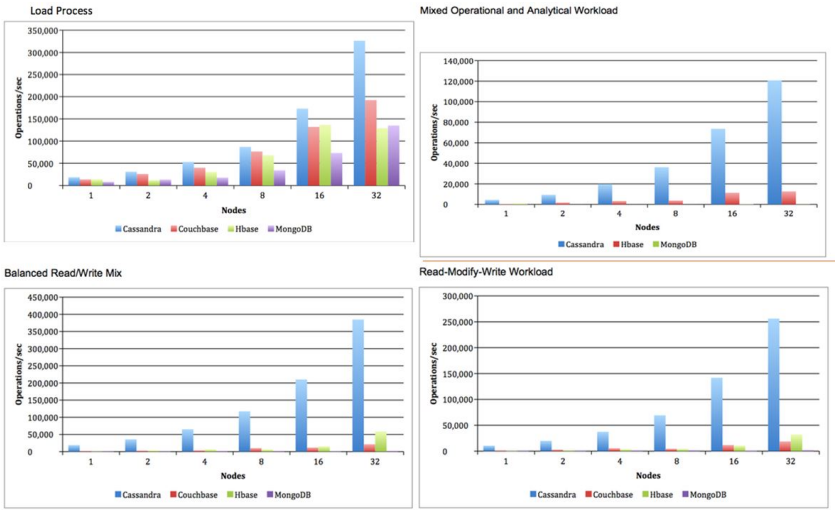

FIGURE 2.17. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

Cassandra as the best database in performance point of view. But infact , there is no best database , choosing the best one depend on the use case . It's the Polyglot persistence which we'll explain in the next section.

Polyglot persistence :

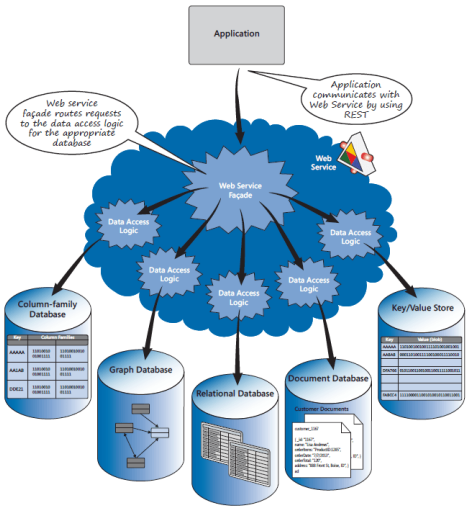

FIGURE 2.18. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

Polyglot Persistence is a fancy term to mean that when storing data, it is best to use multiple data storage technologies, chosen based upon the way data is being used by individual applications or components of a single application. Different kinds of data are best dealt with different data stores. In short, it means picking the right tool for the right use case. It's the same idea behind Polyglot Programming, which is the idea that applications should be written in a mix of languages to take advantage of the fact that different languages are suitable for tackling different problems.

FIGURE 2.19. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

And as we want to make data analytics. We choose as a noSql database for our system , the oriented column one "Cassandra".

2.2.1.3 Big Data Processing Frameworks

Only-Batch processing frameworks

Apache Hadoop (MapReduce) Modern versions of Hadoop are composed of several components or layers, that work together to process batch data:

- HDFS: HDFS is the distributed filesystem layer that coordinates storage and replication across the cluster nodes. HDFS ensures that data remains available in spite of inevitable host failures. It is used as the source of data, to store intermediate processing results, and to persist the final calculated results.
- YARN: YARN, which stands for Yet Another Resource Negotiator, is the cluster coordinating component of the Hadoop stack. It is responsible for coordinating and managing the underlying resources and scheduling jobs to be run. YARN makes it possible to run much more diverse workloads on a Hadoop cluster than was possible in earlier iterations by acting as an interface to the cluster resources.

- MapReduce: MapReduce is Hadoop's native batch processing engine.
- Hadoop commons : librerie that supports others component.

MapReduce's processing technique follows the map, shuffle, reduce algorithm using key-value pairs.
Apache Hadoop and its MapReduce processing engine offer a well-tested batch processing model that is best suited for handling very large data sets where time is not a significant factor.

Only-Streaming processing frameworks

Apache storm Apache Storm is a stream processing framework that focuses on extremely low latency and is perhaps the best option for workloads that require near real-time processing. It can handle very large quantities of data with and deliver results with less latency than other solutions. Storm stream processing works by orchestrating DAGs (Directed Acyclic Graphs) in a framework it calls topologies. The topologies are composed of:

- Streams: Conventional data streams. This is unbounded data that is continuously arriving at the system.
- Spouts: Sources of data streams at the edge of the topology. These can be APIs, queues, etc. that produce data to be operated on.
- Bolts: Bolts represent a processing step that consumes streams, applies an operation to them, and outputs the result as a stream. Bolts are connected to each of the spouts, and then connect to each other to arrange all of the necessary processing. At the end of the topology, final bolt output may be used as an input for a connected system.

Storm is probably the best solution currently available for near real-time processing. It is able to handle data with extremely low latency for workloads that must be processed with minimal delay. Storm is often a good choice when processing time directly affects user experience, for example when feedback from the processing is fed directly back to a visitor's page on a website.
Storm with Trident gives you the option to use micro-batches instead of pure stream processing. While this gives users greater flexibility to shape the tool to an intended use, it also tends to negate some of the software's biggest advantages over other solutions. That being said, having a choice for the stream processing style is still helpful. For pure stream processing workloads with very strict latency requirements, Storm is probably the best mature option. It can guarantee message processing and can be used with a large number of programming languages. Because Storm does not do batch processing, you will have to use additional software if you require those capabilities. If you have a strong need for exactly-once processing guarantees, Trident can provide that. However, other stream processing frameworks might also be a better fit at that point.

Samza Apache Samza is a good choice for streaming workloads where Hadoop and Kafka are either already available or sensible to implement. Samza itself is a good fit for organizations with multiple teams using (but not necessarily tightly coordinating around) data streams at various stages of processing. Samza greatly simplifies many parts of stream processing and offers low latency performance. It might not be a good fit if the deployment requirements aren't compatible with your current system, if you need extremely low latency processing, or if you have strong needs for exactly-once semantics.

Hybrid processing frameworks

Apache Spark Spark is a great option for those with diverse processing workloads. Spark batch processing offers incredible speed advantages, trading off high memory usage. Spark Streaming is a good stream processing solution for workloads that value throughput over latency.

Apache Flink Flink offers both low latency stream processing with support for traditional batch tasks. Flink is probably best suited for organizations that have heavy stream processing requirements and some batch-oriented tasks. Its compatibility with native Storm and Hadoop programs, and its ability to run on a YARN-managed cluster can make it easy to evaluate. Its rapid development makes it worth keeping an eye on.

Summary :

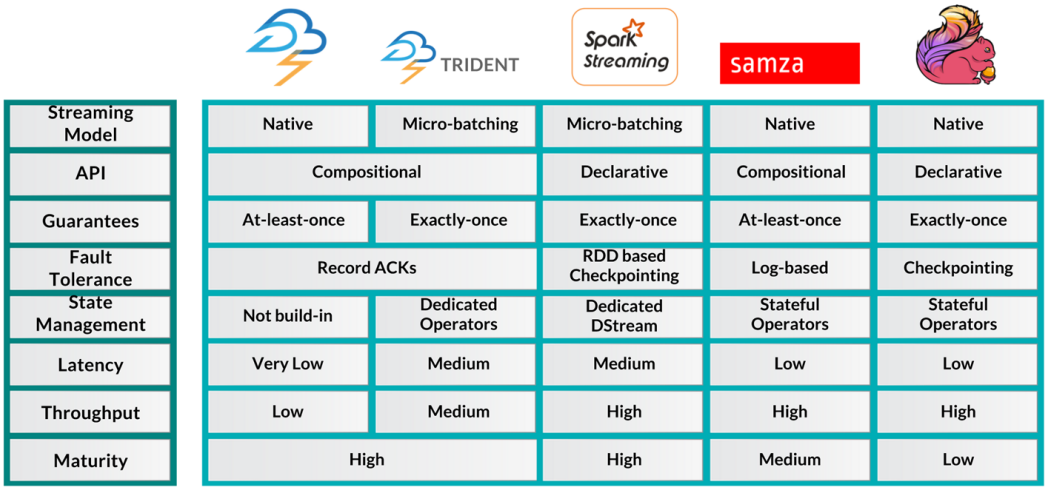

FIGURE 2.20. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

Infact, as before there is no best processing framework . Each one of them is good for a specific usage. So three cases occur:

- Only Batch processing : Hadoop and MapReduce would be a great solution.
- Only Streaming processing : Apache Strom is now undefeatable.
- Hybrid processing : Spark would be a very good solution..

And as we would like to implement the lambda architecture and we definitely want to go with the hybrid processing option. We choose as a data processing engine for our batch and streaming layers "Apache Spark".

2.2.1.4 Conclusion

Finally, we are now able to gather all pieces into one. This comparative study supports our choices in each layer. In the next section we will talk about our microservices architecture.

2.2.2 Architecture microservice

Our application is divided in two parts, the first one is the big data part explained in the last section and the second one is the microservices part. The microservices architecture presents so many patterns. In this section, we will introduce some microservices patterns, then we choose for our application the most appropriate one. After fixing all required patterns we finally move to define our microservices architecture.

2.2.2.1 Decomposition

The microservice architecture structures an application as a set of loosely coupled services.The application should be decomposed in a way so that most new and changed requirements only affect a single service. So the question now is how to decompose an application into services? the next two paragraphs show some useful decomposition strategy .

Decompose by business capability Define services corresponding to business capabilities. A business capability is a concept from business architecture modeling. It is something that a business does in order to generate value. A business capability often corresponds to a business object.

Decompose by subdomain Define services corresponding to Domain-Driven Design (DDD) subdomains. DDD refers to the application's problem space - the business - as the domain. A domain is consists of multiple subdomains. Each subdomain corresponds to a different part of the business.

Summary Dividing services upon business capabilities is more suitable for us. The decomposition by business capability present so much benefits , some of them are :

- Stable architecture since the business capabilities are relatively stable
- Development teams are cross-functional, autonomous, and organized around delivering business value rather than technical features
- Services are cohesive and loosely coupled

2.2.2.2 Cross cutting concerns

The cross-cutting concern include:

- Externalized configuration - includes credentials, and network locations of external services such as databases and message brokers
- Logging - configuring of a logging framework such as log4j or logback
- Health checks - a url that a monitoring service can ‚Äúping‚Äù to determine the health of the application
- Metrics - measurements that provide insight into what the application is doing and how it is performing
- Distributed tracing - instrument services with code that assigns each external request an unique identifier that is passed between services.

There are so much services. we will frequently create new services, each of them will only take days or weeks to develop. we cannot afford to spend a few days configuring the mechanisms to handle cross-cutting concerns. What is even worse is that in a microservice architecture there are additional cross-cutting concerns that we have to deal with including service registration and discovery. So the solution is to build our microservices using a microservice chassis framework, which handles cross-cutting concerns.

Microservice chassis The major benefit of a microservice chassis is that we can quickly and easy get started with developing a microservice. we need a microservice chassis for each programming language/framework that we want to use. This can be an obstacle to adopting a new programming language or framework. There are so much microservice chassis :

- Spring Boot and Spring Cloud (java)
- Dropwizard (java)
- Gizmo (go)

- Micro (go)
- Go kit (go)

Summary We didn't make a comparative study on chassis frameworks, Spring boot and Spring Cloud have a big community and present so much quick start features. So we choose to go with Spring boot and Spring cloud as a chassis framework.

2.2.2.3 External API

Data in the Microservice architecture pattern is spread over multiple service. Each client needs to fetch information from all of these services. The question now is how do the clients of a Microservices-based application access the individual services? Exposing an Api gateway could be a great solution :

API gateway :
Implement an API gateway that is the single entry point for all clients. The API gateway handles

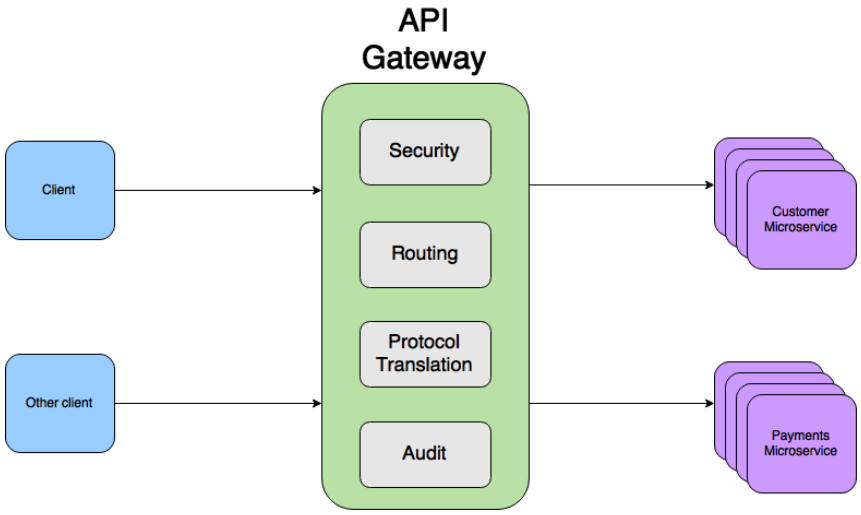

FIGURE 2.21. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

requests in one of two ways. Some requests are simply proxied/routed to the appropriate service. It handles other requests by fanning out to multiple services. the API gateway can expose a different API for each client. The API gateway might also implement security, e.g. verify that the client is authorized to perform the request

Summary Exposing an Api gateway is a bright idea. To implement this solution there is some tools that can handle this :

- Kong API Api Gatway

- Zuul and Spring Cloud

As we choosed Spring boot and Spring Cloud for our application chassis framework, we definitely go with the "Zuul and Spring cloud" api gatway option.

2.2.2.4 Data management

Most services need to persist data in some kind of database. The question is what's the database architecture in a microservices application ?

Database per Service Keep each microservice's persistent data private to that service and accessible only via its API. The service's database is effectively part of the implementation of that service. It cannot be accessed directly by other services. There are a few different ways to keep a service's persistent data private.

- Private-tables-per-service
- Schema-per-service
- Database-server-per-service

Shared database Use a (single) database that is shared by multiple services. Each service freely accesses data owned by other services using local ACID transactions.

Summary choosing a the option of database per service could be a great solution. This pattern present some benefits :

- Helps ensure that the services are loosely coupled. Changes to one service's database does not impact any other services.
- Each service can use the type of database that is best suited to its needs

so we choose to go with the the database per service choice.

2.2.2.5 Conclusion

In this section we have fixed all patterns and choosed which technologies we would like to implement , in the next section we move to talk about our deployment strategy.

2.2.3 Deployment architecture

Our system now is a set of services. Each service is deployed as a set of service instances for throughput and availability. The question now is how are services packaged and deployed?

Multiple service instances per host :
Run multiple instances of different services on a host (Physical or Virtual machine). There are various ways of deploying a service instance on a shared host including:

- Deploy each service instance as a JVM process. For example, a Tomcat or Jetty instances per service instance.
- Deploy multiple service instances in the same JVM. For example, as web applications.

Service instance per host :
Deploy each single service instance on its own host

Service instance per VM :
Package the service as a virtual machine image and deploy each service instance as a separate VM

Service instance per Container :
Package the service as a (Docker) container image and deploy each service instance as a container

Summary :
Deploying our services in containers is the most appropriate option for us. Due resource and budget limits, our application is deployed under a VM that conatin many containers carrying services (service per container). This approach has some benefits :

- It is straightforward to scale up and down a service by changing the number of container instances.
- The container encapsulates the details of the technology used to build the service. All services are, for example, started and stopped in exactly the same way.
- Each service instance is isolated.

- Containers are extremely fast to build and start. For example, it‚Äôs 100x faster to package an application as a Docker container than it is to package it as an AMI. Docker containers also start much faster than a VM since only the application process starts rather than an entire OS.
- A container imposes limits on the CPU and memory consumed by a service instance.

2.3 Conclusion

Two main parts are waiting to be implemented, the first one is the lambda architecture. The second one is the microservices approach and the application logic. In our scoope , due to time limits we will not be able to implement all patterns , we will just make the most importants one. In the next chapter we move to our application Design.
Notice :
In our scoope , due to time limits we will not be able to implement all functionalities and patterns , we will just make the most importants one.

CHAPTER 3

DESIGN

This chapter will be devoted to present the design stage of our system. Designing is a creative process that describes the manner in which our system works. We will start by presenting our architecture design, then we move to the general conception and finally the detailed design.

3.1 General design

Our project is essentially based on two main parts as described before. In the next section, we present the architecture of the system.

3.1.1 Architectural design

Architectural design refers to the high level structures of a software system, the discipline of creating such structures, and the documentation of these structures. These structures are needed to reason about the software system. Each structure comprises software elements, relations among them, and properties of both elements and relations. It's about making fundamental structural choices which are costly to change once implemented. Architecture design choices include specific structural options from possibilities in the design of software.

3.1.1.1 Big Data architectural design

In the last chapter we fixed what we will be using in term of patterns and tools.Now,we are able to just start implementing our lambda architecture.

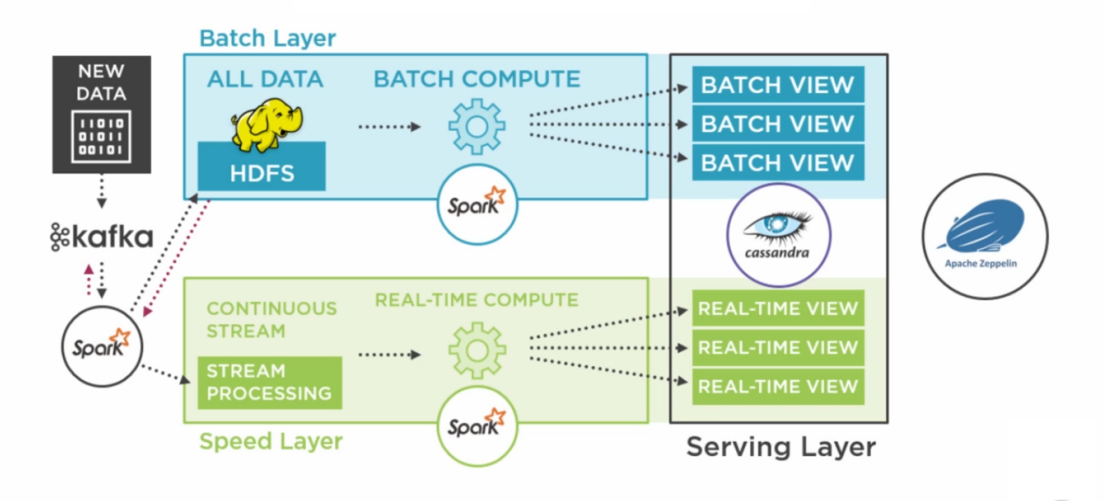

FIGURE 3.1. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

3.1.1.2 Microservices architectural design

After fixing all patterns and choosing which technologies we would like to implement , it's time to make the last step. The next figure show our microservice application architecture in it's first versions:

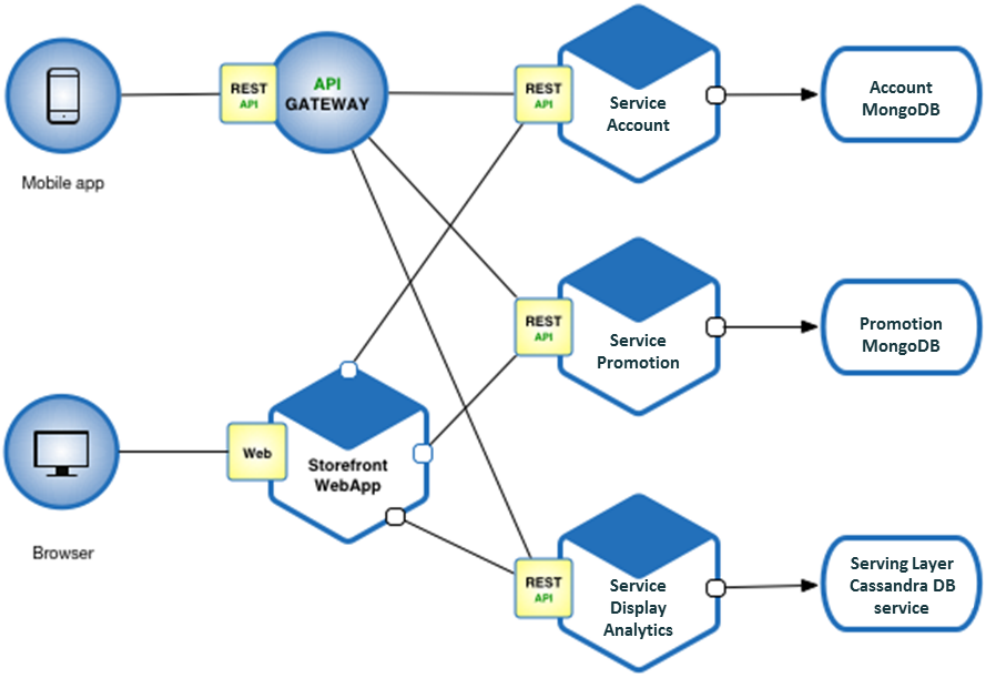

FIGURE 3.2. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

3.1.1.3 Deployment architectural design

We have designed our deployment architecture showing all containers communication and linking pipelines. The next two figures show the big data and the microservices deployment architecture.

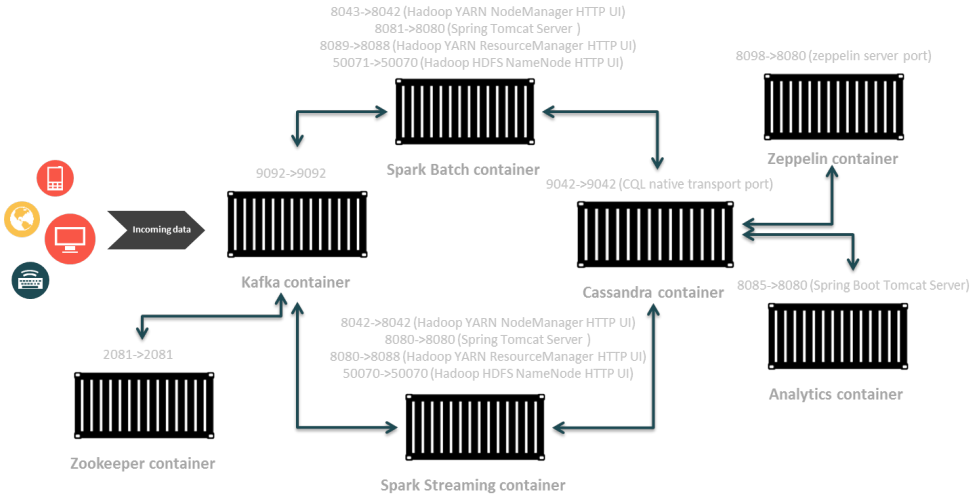

FIGURE 3.3. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

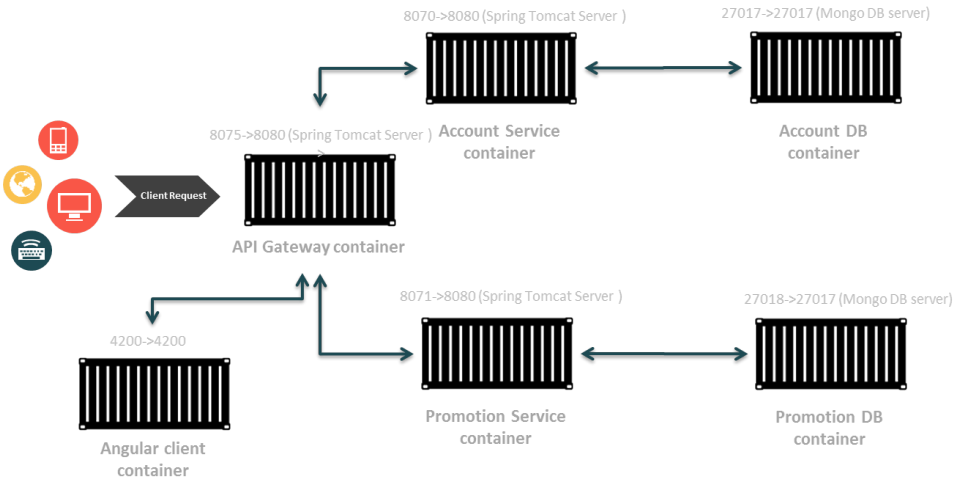

FIGURE 3.4. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

3.1.1.4 Summary

Finally, the next figure resume all patterns and architecture discribed before.

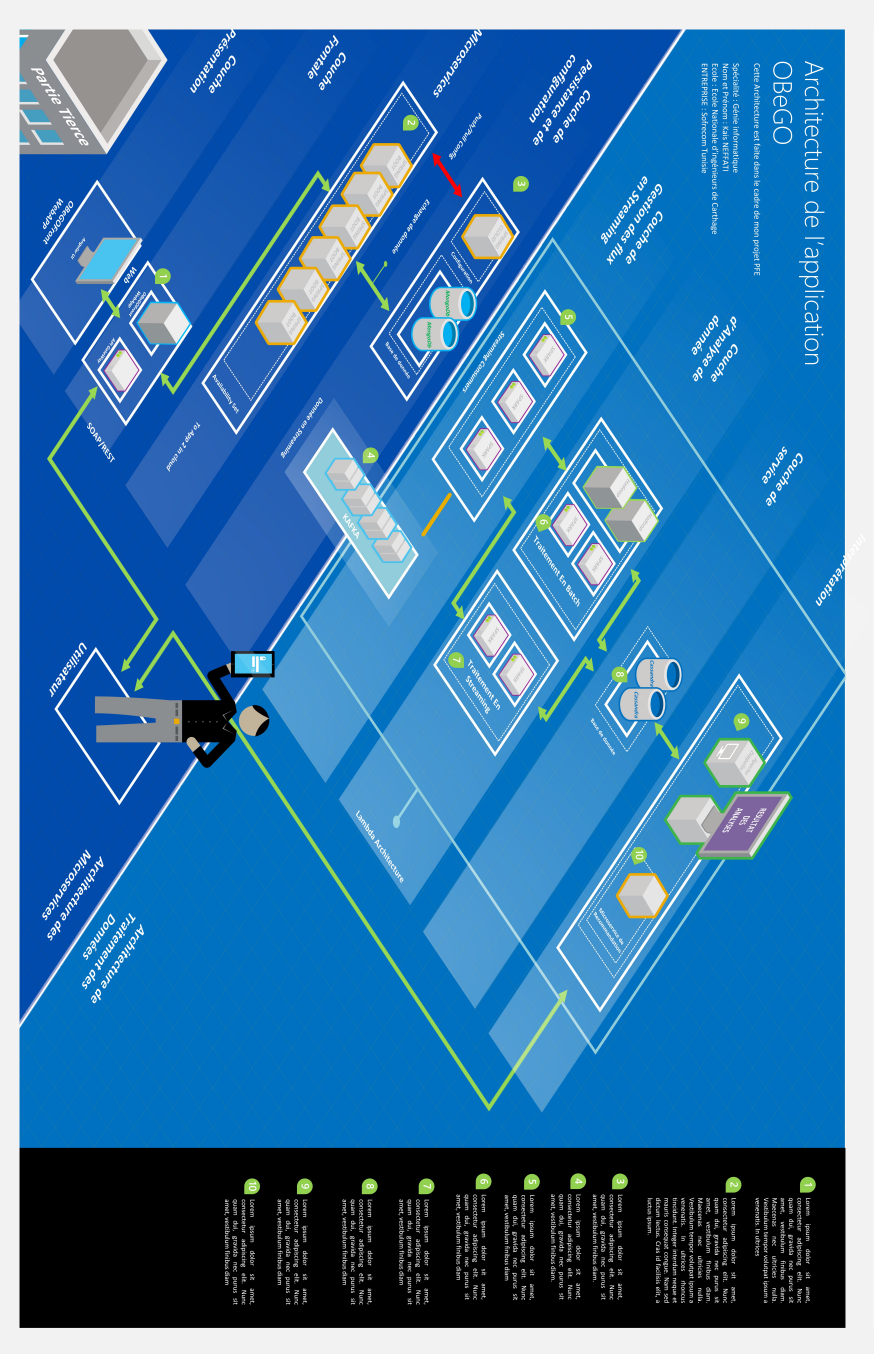

FIGURE 3.5. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

3.2 Detailed design

3.2.1 Package Diagram

3.2.2 Class Diagram

3.2.3 Object Sequence Diagrams

3.3 Conclusion

Two main parts are waiting to be implemented, the first one is the lambda architecture and the data analytics strategy. the second one is the microservices approach and the application logic.

CHAPTER 4

IMPLEMENTATION

4.1 Work environnement

This part presents the Hardware and Software environment available for this project.

4.1.1 Hardware environnement

In our project we developed our application using a computer with the following characteristics :

- Process Intel Core i5 4310M .
- Memory RAM: 16 GB .
- Space Discs: 500 GB.

4.2 Software environnement

We've used for our project the following Software :

- VMware Workstation : Our application is deployed under VMware Workstation virtualization platform as a set of decoupled service. Each service per container.
- Centos 7 : Our application operating system is Centos 7 carring Docker as demon.
- Docker : is our main virtualization tool and our containers manager.

- Intellij idea : is our application Back End IDE , we used it for Scala and java (spring) developement.
- Visual Studio Code : is our application Front End IDE , we used it for angular 2 and webpack (TypeScript developement).

4.3 Realized work

4.3.1 Environnement preparation

Our application require some containers preparation. The following list contain all needed containers.

- Kafka container : used for collecting data and stream them through a message queue. This container does not existe in dockerhub repository so we should Dockerize it(Dockerize kafka means create a container that contain kafka)
- Zookeeper container : used to save brokers address and messages offsets . this container exists in docker hub.
- Scala container : used to build the Hadoop container.This container does not existe in dockerhub repository so we should Dockerize it
- Hadoop container : used to build the spark container and provide HDFS persistance and YARN (ressource manager) for the spark jobs. This container does not existe with the Hadoop 2.7 and centos 7 version, so we should Dockerize it
- Spark container : used to to consume the data from kafka , process it then feed it to the serving layer (persisting results in cassandra).
- Cassandra container : used as a serving layer . this container exists in docker hub , we will be using the 2.2 cassandra version.
- Zeppelin container : used as a notebook to interpret Cassandra and spark then display the results in a dashboard.
- Spring boot containers : we need some containers that carry each one a microservice. This containers just need to have a JVM.
- Mongodb containers : each service is linked to a mongodb container. this containers exists in dockerhub.
- FrontEnd container : container for the angular dashboard.

4.3.1.1 Dockernize Tools

In this section we present the process of dockernizing our containers .All installation setps are written in Dockerfiles and each container has an OS kernel. We will be using alpine and centos 7.

Dockernize Kafka To dockernize this container, we will start by putting a base OS version wich is alpine linux . The next step is the Jdk installation. After fixing all paths we move to install kafka.

FIGURE 4.1. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

Dockernize Scala To dockernize this container, we will start by putting a base OS version wich is centos 7 . The next step is the Jdk installation. After fixing all paths we move to install Scala.

FIGURE 4.2. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

Dockernize Hadoop To dockernize this container, we will start from the previous Scala container. The next step is the Hadoop installation which takes too much time due to the big configuration amount. After dockernizing Hadoop 2.7 we are able now to see some Doshboard. Hadoop YARN ResourceManager HTTP UI : Hadoop HDFS NameNode HTTP UI :

FIGURE 4.3. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

Dockernize Spark This container is the main container. We built the last two ones for this step. This container is based on the haddop one . Spark uses HDFS to persist data and uses YARN as a ResourceManager.This container require much more then the last one in terms of configuration. We chose to built spark on yarn to benefit the real sparks power.
So spark present in totaly 3 deployment mode :
Standalone deployment: With the standalone deployment one we can statically allocate resources on all or a subset of machines in a Hadoop cluster and run Spark side by side with Hadoop MR. The user can then run arbitrary Spark jobs on his HDFS data. Its simplicity makes this the deployment of choice for many Hadoop 1.x users.
Hadoop Yarn deployment: Hadoop users who have already deployed or are planning to deploy Hadoop Yarn can simply run Spark on YARN without any pre- installation or administrative access required. This allows users to easily integrate Spark in their Hadoop stack and take advantage of the full power of Spark, as well as of other components running on top of Spark.
Spark In MapReduce (SIMR): For the Hadoop users that are not running YARN yet, another option, in addition to the standalone deployment, is to use SIMR to launch Spark jobs inside MapReduce. With SIMR, users can start experimenting with Spark and use its shell within a couple of minutes after downloading it! This tremendously lowers the barrier of deployment, and lets virtually everyone play with Spark.
Notice: Spark is an in-Memory data processing framwork so when trying to configure it make sure you put the right RM consumption. In our case we ve used 1 core o

Dockernize Zeppelin

FIGURE 4.4. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

FIGURE 4.5. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

FIGURE 4.6. Sofrecom's subsidiaries and customers. Figure taken from [**?**].

4.3.2 Log Producer

4.3.3 Spark Jobs

4.3.4 Show Resualts on Apache Zeppelin

4.3.5 Account Managment

4.3.6 Promotion Managment

4.3.7 Dashbord Microservice

4.3.8 Display Analytics

4.3.9 Heat Map users

4.3.10 Summary

4.3.11 General conclusion

Appendix A

Begins an appendix