Implementació d'una aplicació web utilitzant una arquitectura de comunicació MuleSoft

Miquel Barberà Tor

Resum— En aquest treball es presenta el desenvolupament d'un frontal web per a la monitorització de vendes d'una empresa. Per tal de gestionar la comunicació entre tots els actors implicats (BBDD, SAP, Front-end...) utilitzarem una arquitectura MuleSoft que ens permetrà crear APIs que gestionin tota la comunicació interna. Aquest treball serveix per fer una prova de concepte i començar a implementar aquesta nova tecnologia en tota l'empresa que ha de millorar el rendiment respecte el Enterprise Service Bus que hi ha actualment el OSB12 (Oracle).

Paraules clau— Front-end, Angular, Javascript, Oracle, MuleSoft, ESB, XML, API, Anypoint Studio, OSB12, Cloud

Abstract— This project introduces the development of a web front for sales monitoring of a company. In order to manage the communicacion between all the actors involved (DB, SAP, Front -end...) we will use a MuleSoft architecture that will allow us to create APIs that handle all the internal communication. This work is a proof of concept (POC) that will be used to initiate the implementation of this new Technology across the company that should improve the performance over the Enterprise Service Bus that is currently running on OSB12 (Oracle)

Index Terms— Front-end, Angular, Javascript, Oracle, MuleSoft, ESB, XML, API, Anypoint Studio, OSB12, Cloud

1 Introducció

Punto Fa S.L. més coneguda com a MANGO és una empresa multinacional dedicada a la indústria de la moda. En aquest sector es dedica al disseny, fabricació, distribució i comercialització de peces de vestir i complements. Mango compta amb més de 2700 punts de venda distribuïts en més de 105 països.

Aquest Treball de Fi de Grau (TFG) es realitza en aquesta empresa, en el departament d'informàtica (IT), més concretament en l'equip de MuleSoft [1]. En aquest departament ens encarreguem de realitzar la transformació de la comunicació entre aplicacions, on passarem d'utilitzar el bus de dades d'Oracle (OSB12) [2] a la plataforma de MuleSoft.

Mango disposa d'una xarxa d'aplicacions molt gran ,més d'un centenar d'aplicacions, tant per ús intern com per ús extern. Aquestes aplicacions es comuniquen amb els ERP (SAP) [3], bases de dades (Consumer Repositories...) i altres aplicacions diverses mitjançant un Enterprise Service Bus, el bus de dades d'Oracle.

Un Enterprise Service Bus [4], antigament anomenats Enterprise Application Integration (EAI), és un middleware que serveix per gestionar la comunicació d'aplicacions dins d'una mateixa organització. L'arquitectura del ESB proporciona una capa d'abstracció que permet la comunicació entre les aplicacions normalment construïda sobre estàndards XML o JMS i seguint protocols SOAP o REST [5].

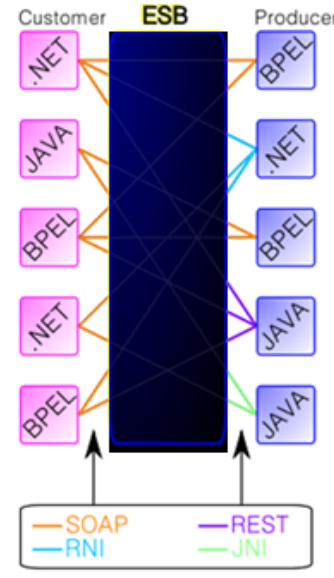

Fig. 1 Exemple d'ESB

- *E-mail de contacte: miquel.barbera@e-campus.uab.cat*
- *Menció realitzada: Tecnologies de la Informació.*
- *Treball tutoritzat per: Sergi Robles (DEIC)*
- *Curs 2019/20*

En el context actual de l'empresa es va decidir començar a deixar d'utilitzar el OSB12 i apostar per una nova arquitectura d'ESB anomenada MuleSoft. Aquesta ens proveeix de diverses eines per desenvolupar fàcilment la comunicació entre aplicacions com ara l'Anypoint Studio, l'Anypoint Platform etc, que més endavant explicarem amb més detall. El motiu de deixar de banda el OSB12 i apostar per MuleSoft és sobretot degut a la voluntat d'innovació tecnològica en l'empresa, buscant les millors eines per millorar el rendiment.

2 Objectius

En aquest apartat es presentaran els objectius i subobjectius del projecte en ordre d'importància. L'objectiu principal és programar un frontal que permeti visualitzar les vendes realitzades en els punts de venda de MANGO a temps real, i que la comunicació entre tots els actors sigui a través de la plataforma de MuleSoft. Per assolir-ho s'han plantejat els següents subobjectius:

- Programar el Front-end en Angular que permeti a un treballador iniciar sessió si té permisos per visualitzar la informació mostrada en el frontal
- Llistar el top 100 de peces de roba venudes en els punts de venda de l'empresa i mostrar la informació dels productes (talles, punts de venda...)
- Organitzar l'arquitectura d'APIs que utilitzarà el frontal per tal de gestionar tota la comunicació que es necessitarà
- Definir les APIs en RAML [6] desenvolupar-les i publicar-les en el cloud de MuleSoft per a l'ús del frontal.
- Construir les mètriques personalitzades en el Anypoint Platform per tal de realitzar un seguiment i monitorització de les APIs que tenim publicades
- Comparar les mètriques obtingudes en MuleSoft i amb les del bus de dades d'Oracle (OSB12)

3 Metodología

En aquest treball utilitzarem dues metodologies, una primera per desenvolupar el frontal web i una segona pel projecte Mule.

En la primera part del projecte utilitzarem la metodologia Kanban [7], on tindrem un Product Owner (PO) que ens anirà definint tasques en un backlog. Aquestes tasques tenen tres estats, To do, Doing i Done. Aquesta metodologia ens permet anar adaptant la feina al temps disponible, tenir una visió clara i rapida de la feina pendent i la realitzada com també un control de les tasques que tenim obertes.

Per tal de tenir una millor visualització utilitzarem JIRA [8] tant per la definició com pel control i resolució de tasques, que amb la seva vinculació amb bitbucket podrem tenir un seguiment de tot el treball realitzat i un bon control de versions.

Com podem veure en la següent imatge (Figura 2) tindrem dos entorns de desenvolupament, develop i master. Totes les noves funcionalitats s'aniran desenvolupant en la branca develop. Un cop es decideixi, es puja tot el codi revisat a la branca master, la branca definitiva. Si sorgeixen alguns bugs es poden realitzar bugfix o inclús recuperar alguna versió anterior i tornar a crear una tasca a develop per solucionar-ho.

En la segona part del projecte la figura del PO desapareix, per tant la metodologia Kanban ja no té molt sentit. Per aquesta segona part utilitzarem la metodologia de prototip incremental. Aquesta metodologia consisteix en crear un primer prototip d'aplicació, i a mesura d'iteracions incrementals anar millorant aquesta aplicació fins a tenir un prototip final funcional. Aquesta metodologia ens ajuda a tenir una visió clara dels objectius a seguir i ens permet tenir un prototip bàsic força ràpid i a partir d'allà identificar les mancances que pot tenir i resoldre-les.

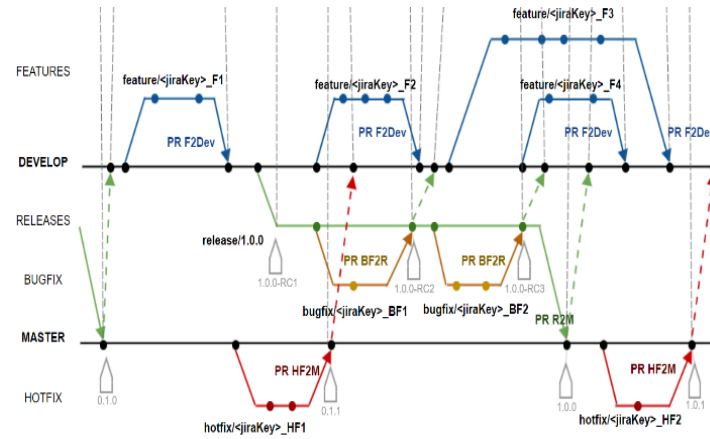

Fig.2 Estructura de les branques de GitFlow

4 Estat de l'art

Com hem explicat en l'apartat d'introducció, la motivació d'aquest treball és la transformació de la connectivitat entre aplicacions, dispositius i dades. En aquest apartat explicarem quina és la situació global en què es troben els conceptes que utilitzarem com MuleSoft, Anypoint Platform, OSB12... També descriurem breument les tecnologies emprades i en quin estat de desenvolupament es troben en l'actualitat.

El front-end el programarem amb Angular 7.0, un framework de JavaScript open-source creat per Google. En aquest framework es programa en Typescript (Javascript tipat) juntament amb HTML i SCSS [9] per la maquetació de la web. En el moment de la redacció d'aquesta memòria (Gener 2020) Angular se situa en el top 3 de framworks més utilitzats en tot el món juntament amb React i Vue [10]. En aquest treball s'ha decidit utilitzar Angular per estar harmonitzat amb la resta de frontals d'aplicacions de Mango, ja que tots els frontals del Backoffice estan programats en Angular i així també aprofitar el "know how".

Tota la comunicació la gestionarem amb l'arquitectura de

MuleSoft. MuleSoft és una companyia dedicada al desenvolupament de Software creada a San Francisco, Califòrnia l'any 2006. El principal producte (que és el que utilitzem en aquest treball) és Anypoint Platform. Aquesta plataforma ens permet dissenyar, construir i fer un seguiment de les APIs [11] que garanteixen una connexió segura entre tota la xarxa d'aplicacions que es troben en l'empresa. Primer es dissenya l'API en el Design Center on es creen totes les especificacions de l'API en RAML/OAS/SOAP (en el nostre cas construirem totes les APIs en RAML-RESTful API Modeling Language). Un cop construïda, es publiquen en l'Exchange, on seran consumides des de les diferents aplicacions. Per tal de desplegar i gestionar els diferents projectes, utilitzarem l'Anypoint Studio on programarem l'aplicació mule.
En aquests moments (Gener 2020) podem trobar molts exemples d'empreses punteres en el sector tecnològic que han començat a implementar aquesta arquitectura de comunicacions. Exemples en podem posar molts com per exemple Netflix, Mastercard, Airbnb, HSBC, Airbus... Sobretot ho podem trobar en empreses amb un gran volum de xarxes d'aplicacions.
El bus de dades d'Oracle (OSB12) proveeix a l'empresa d'un servei de comunicació entre aplicacions (ESB). Oracle és una companyia que desenvolupa programari per gestionar empreses i gestionar bases de dades. En aquests moments (Gener 2020) està en la posició numero 6 de ESB més utilitzats en el món empresarial (MuleSoft es troba en la segona posició). El més utilitzat és el webMethods Integration Server però bàsicament perquè és el que té versió gratuïta que no encaixa en el nostre projecte.

5 Arquitectura MuleSoft

En aquest apartat descriurem l'arquitectura de MuleSoft i com funciona. Mulesoft és una plataforma que permet la comunicació entre aplicacions sigui quina sigui la tecnologia utilitzada. Això permet que grans empreses amb centenars d'aplicacions puguin tenir una eina que faciliti la feina a l'hora d'utilitzar la xarxa d'aplicacions.
Per tal d'aconseguir aquesta comunicació entre aplicacions crearem una sèrie d'APIs que es publicaran en el cloud de MuleSoft i que seran consumides per les pròpies aplicacions a través d'endpoints per tal de comunicar-se entre elles. La capa de middleware del mig seran aquestes APIs que garantiran la comunicació.
Una API és un conjunt de definicions i protocols que s'utilitza per desenvolupar i integrar el software de les aplicacions. Les API permeten que els seus productes i serveis es comuniquin amb altres sense necessitat de saber com han estat implementats. Això simplifica el desenvolupament d'aplicacions i permet estalviar temps i diners. Les API aporten flexibilitat, simplifiquen el disseny, l'administració i l'ús de les aplicacions.
També es poden veure les API com a contractes, documentació que representa un acord entre les parts. Si una de les parts envia una sol·licitud remota amb certa estructura en particular aquesta mateixa estructura determinarà com respondrà el software de l'altra part.
Les API permeten la col·laboració entre l'equip comercial/negoci i el d'IT. Les necessitats comercials solen canviar ràpidament, sobretot en el sector de retail que és en el que ens trobem, i poder generar canvis d'una manera ràpida pot generar un avantatge competitiu molt important. El desenvolupament d'APIs natives en un cloud és una forma d'augmentar la velocitat de desenvolupament
A l'hora de crear aquestes APIS utilitzarem la plataforma de disseny anomenada Design Center, una plataforma on crearem aplicacions Mule, APIs i fragments d'APIs. Els fragments d'APIs són parts d'APIs que s'utilitzen normalment quan una part d'una API serà reutilitzada freqüentment. D'aquesta manera aconseguim més facilitat a l'hora de programar i desenvolupar APIs.
A l'hora de crear les APIs les situarem a diferents nivells depenent de la funció que realitzen, d'aquesta manera podem tenir una certa organització. Les separarem en 3 "layers", les APIs experience, process i System.

- System APIs: En general són les que accedeixen als sistemes 'core' i serveixen per aïllar a l'usuari final de la complexitat del sistema o si hi ha qualsevol canvi en els sistemes subjacents. Un cop creats els usuaris poden accedir a les dades sense conèixer la lògica que hi ha darrere i poden ser reutilitzades múltiples vegades.
- Process APIs: Aquestes APIs interactuen directament amb les dades i són les que realitzen les transformacions principals. Tenen directa dependència amb els sistemes d'on provenen les dades com també dels sistemes que han de rebre aquestes dades.
- Experience APIs: Aquestes APIs són les que interactuen amb el client final i estan dissenyades sobretot perquè l'experiència de l'usuari sigui el més fàcil i àgil possible. Estan pensades perquè sigui una API per cada punt de dades comú i no pas una per cada usuari que en necessiti una. Per exemple tindrem una API Experience per accedir a la base de dades que ens proporcioni les dades de les vendes, cosa que per darrere es necessitin més a l'hora de crear la transformació de dades.

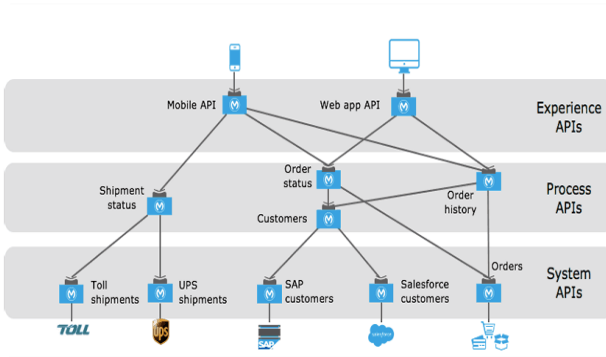

Fig. 3 Exemple d'Arquitectura i Layers de MuleSoft

A l'hora de desenvolupar aquestes APIs és molt important que tinguem en compte la seva possible reutilització, ja que per exemple la nostra API que permet la comunicació entre el frontal i els Consumer Repositories (BBDD on estan emmagatzemades les dades dels consumidors) poden ser utilitzades per una altra multitud d'aplicacions que necessiti accedir a les dades de la BBDD.
Un cop tenim les APIs desenvolupades, aquestes seran publicades en el 'marketplace' dels connectors, APIs, templates, exemples... Anomenat Anypoint Exchange. En el nostre cas allà publicarem de forma privada les nostres APIs per tal de tenir una visualització de tot el que es va desenvolupant en l'empresa.
Per acabar aquestes APIs correran en el Cloudhub, el servidor cloud que proveeix MuleSoft.
Des d'allà seran consumides per les aplicacions per tal de comunicar-se entre elles. En aquest servidor disposarem tant dels entorns de Development (per realitzar les proves) com el de Producció.

5.1 CloudHub

Aquí descriurem que és el CloudHub [12] i perquè és un factor clau en el desenvolupament d'aquest projecte.
CloudHub és una plataforma d'integració as a service (iPaaS) on es despleguen les APIs en el cloud, integra aplicacions on-premise amb serveis cloud...
CloudHub és un cloud elàstic, això vol dir que escala segons la demanda que hi hagi sense haver de parar cap procés que es trobi executant en el cloud. Si una aplicació necessita més potència el cloudhub directament posa més nodes a treballar.
Les aplicacions en el CloudHub corren en instàncies de Mule anomenades workers. Tenen quatre característiques principals:

- Capacitat: Cada worker té una capacitat específica de processar dades que pots seleccionar al configurar l'aplicació.
- Aïllament: Cada worker s'executa en un contenidor diferent per evitar propagació d'errors o col·lapse.
- Localització: Cada worker corre en un cloud específic, Estats Units, Asia-Pacific... Els workers que nosaltres utilitzem estan situats a Irlanda.
- Gestió: Cada worker es gestiona de manera independent de la resta.

En la següent imatge (Figura 4) podem veure la arquitectura fàcilment resumida. Les aplicacions d'integració serien les APIs que creen i despleguem. El Runtime Manager és la plataforma per interactuar amb el cloud a través de la seva consola. La plataforma de serveis serveix per gestionar, monitoritzar, controlar... Totes les APIs en el cloud.

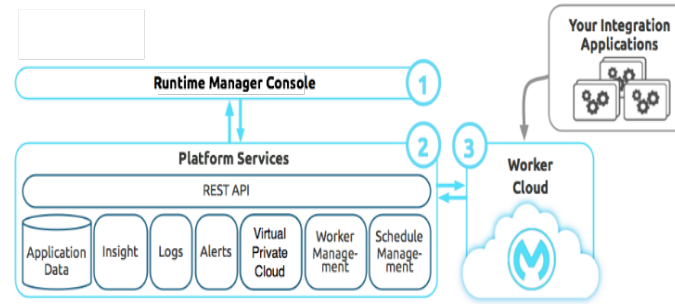

Fig. 4 Arquitectura del CloudHub

El Global Worker Cloud és el cloud elàstic d'instàncies de Mule on corren directament les APIs. Quan ajuntem tots aquests components tenim l'arquitectura del CloudHub.
Cloudhub també proveeix de seguretat per totes les comunicacions que es fan a través del cloud. Això ho aconsegueix donant a cada aplicació la seva pròpia màquina virtual (VM). Tota la comunicació entre la plataforma CloudHub i els clients està garantida utilitzant SSL amb certificat d'autentificació. Això fa que ningú sense autorització pugui llegir les dades que viatgen pel CloudHub.

6 Desenvolupament

En aquesta secció es pretén presentar el desenvolupament del projecte. Primerament començarem explicant el desenvolupament del frontal web. Després passarem a explicar com s'ha muntat l'arquitectura MuleSoft, la implementació de la comunicació entre APIs i finalment revisarem els resultats obtinguts.

6.1 Desenvolupament del frontal

Com hem dit abans el frontal consistirà en una web on primerament trobarem un inici de sessió on l'usuari introduirà el seu codi d'empleat i la seva contrasenya. Si l'usuari té permisos, podrà entrar a visualitzar el top 100 (podrà decidir si 100, 50 o 20) peces de roba venudes en les ultimes 24 hores. Tindrem l'opció de filtrar per països, per tipus de peça de roba, per categoria (Mango Man, Woman, Kids...). La pàgina es refresca cada 10 segons sense necessitat d'actualitzar la pàgina per veure les dades actuals de manera que es pot fer un seguiment actua-

litzat i detallat. A l'hora de seleccionar un producte podrem veure tota la informació disponible, el nom del producte, les unitats venudes, i un parell de gràfics per veure quines són les talles més venudes i les tendes on s'han venut.
Aquesta informació ha de servir al departament de Negoci a l'hora de prendre les millor decisions per a l'empresa.
Per tal de construir el frontal ho dividirem en quatre grans mòduls:

- Inici de Sessió: Aquest serà el mòdul més fàcil de desenvolupar a nivell de JavaScript, ja que aprofitarem el mòdul d'inici de sessió que s'utilitza en la majoria d'aplicacions de l'empresa. Aquest mòdul per tal que sigui compartit s'ha posat en una llibreria anomenada 'mango-lib'. L'únic que haurem d'implementar serà la transferència de dades cap al microservei d'autentificació.

- Llistat de Producte: Aquest mòdul serà la part central de l'aplicació. Serà on llistarem els productes més venuts. L'estructura que haurem de mostrar serà primer la imatge en gran perquè els usuaris puguin veure ràpidament quins són els productes més venuts, després el número d'unitats venudes... Com que des del back ens arriben en fitxers separats la informació del producte i la imatge del producte ho haurem d'ensamblar en el mateix frontal, primer identificant el producte venut i després buscant la imatge que se li correspon. Com que a l'hora de carregar la informació pot tardar, per tal de no mostrar la pantalla en blanc programarem un skeleton amb la forma que hauria de tenir la llista però sense mostrar les dades. La principal dificultat d'aquest mòdul ha sigut que com que no està implementada la comunicació amb les bases de dades vam estar treballant tot el desenvolupament del frontal amb mock data que vam crear especialment per l'ocasió.

- Visualització de Producte: Aquest serà el mòdul dedicat a mostrar tota la informació de vendes que tenim d'un producte. Quan l'usuari seleccioni el producte s'obrirà una finestra on sortirà la imatge del producte en gran, tota la informació extra que no hem pogut posar en el llistat i dues gràfiques. Aquestes gràfiques mostraran el numero d'unitats venudes per talla i les 5 tendes on s'han venut més peces d'aquest producte. Per construir aquestes gràfiques s'han utilitzat unes llibreries de Google anomenades Google Charts.

- Filtrat de Producte: Aquest serà el mòdul de més dificultat. Cada producte té assignat certs paràmetres que ens arriba des de la base de dades en el fitxer JSON de la informació. Construirem un objecte producte amb els atributs pels quals filtrarem com per exemple país, talla, data de venda i categoria (MANGO woman, man ,kids o violeta). Quan l'usuari seleccioni els filtres indicats, els productes que tinguin els atributs igual als filtres seleccionats seran els mostrats. Quan l'usuari seleccioni un filtre sortirà un petit Tag amb una creu del filtre marcat. Si l'usuari selecciona la creu, aquest filtre deixarà d'estar seleccionat. També inclourem el filtratge del numero de productes que volem llistar, si volem el top100, top50 o top20 de productes més venuts. Això agafarem l'atribut d'unitats venudes de la classe Producte, els ordenarem de major a menor i seleccionarem els 20/50/100 productes.

Per últim ens dedicarem a la maquetació del frontal amb SCSS, un superset de CSS amb una sintaxi més fàcil. L'aplicació ha de ser user-friendly, ja que els usuaris que utilitzaran aquesta aplicació son persones de l'equip de negoci amb poc coneixement tècnic. Per tal d'aconseguir-ho comptarem amb l'ajuda de l'equip de UX (User Experience) que ens ajudarà a decidir la maquetació de la web i on ha d'anar cada component en el HTML. També seguirem un codi d'estil propi de l'empresa utilitzant certes fonts i patrons. L'últim retoc serà posar-li a l'aplicació un dark-mode, que enfosqueixi tota l'aplicació. Això és degut a que l'aplicació s'estarà ensenyant per uns monitors de mida elevada i si la deixàvem tal com la teníem podia arribar a ser una mica molest així que a l'apartat de filtre hi ha una opció per poder-ho canviar.

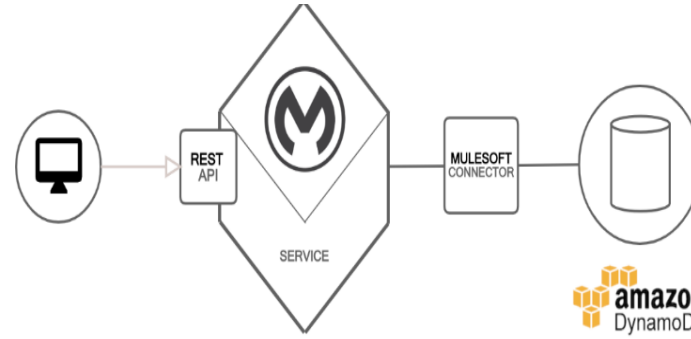

Fig. 5 Consum de les dades per part del frontal

6.1.1 Tests del frontal

Per realitzar els tests del frontal utilitzarem el framework de test Jasmine [13]. Jasmine és un framework per generar test unitaris de Javascript. En aquests tipus de test el que es testeja és el comportament de la unitat mínima de codi, que en el cas de Javascript seria la funció. El que utilitzem és una funció describe() on descriurem el comportament que hauria de fer la funció. Si per exemple tenim una funció que printa "Hello World", el que farem en el test és invocar la funció i comprovar que l'output de la funció és el string indicat. Si hi ha funcions que depenen d'altres

(que hagin de passar-li informació...) el que farem serà crear mocks de les dades que hauria de rebre aquesta funció. D'aquesta manera si alguna funció no té el comportament indicat, ho podem veure fàcilment abans de pujar el codi a producció i ens evitarà pujar codi que falla on ho puguin veure els clients.
Com que el projecte ha sigut creat pel CLI d'Angular ja ve preparat per ser testejat directament. Només executar la comanda "ng test" l'app recull tots els fitxers .spec.ts i els executa.

6.2 Backend amb microserveis

En aquest apartat explicarem els microserveis que consumeix el nostre frontal web.
Una característica pròpia de la cultura DevOps [14] és la descentralització dels elements. Precisament un dels motius per els quals fem aquest projecte és per descentralitzar les crides a les APIs. Per això no seria molt coherent tenir un backend únic per al projecte generant dependències, per tant s'ha decidit apostar per una estructura de microserveis.
Els microserveis consisteixen en dividir les funcionalitats del backend de manera que en comptes de tenir un backend unificat, tens petits backs distribuïts en servidors. D'aquesta manera quan fem una consulta al back apuntem només al que necessitem.
El fet de tenir els serveis descentralitzats genera un cost una mica superior però té molts beneficis com per exemple que si un servidor cau, només cau una part de la funcionalitat de l'aplicació i permet que se segueixi utilitzant. També permet que diversos fronts apuntin a un mateix microservei, per tant només fa falta que es desenvolupi una primera vegada, no fa falta que s'hagin de desenvolupar vàries vegades una mateixa funcionalitat.
En el cas del nostre frontal utilitzem els microserveis per a l'autentificació dels usuaris de la web i per obtenir els fitxers de traducció de la web. En el primer cas enviem l'usuari i contrasenya xifrat al microservei que serà en encarregat de dir si se li ha de permetre a l'usuari accedir a la web o no. En el cas del microservei de traducció, segons l'idioma indicat per l'usuari (o el que està predeterminat en el navegador web que utilitza) el microservei ens retornarà un fitxer JSON amb les traduccions a l'idioma indicat.
Aquests microsereveis ja es troben programats i l'únic que hem de fer és que en els mètodes que cridem en el front apuntin correctament als micros.

6.3 Load Balancers

En aquest apartat passarem a explicar què són els Load Balancers de l'arquitectura MuleSoft i com els hem implementat en aquest projecte. Els Load Balancers de MuleSoft són un component opcional de la plataforma de desenvolupament Anypoint Platform que permeten dirigir tràfic extern HTTP i HTTPS a diferents aplicacions Mule desplegades als nodes workers del CloudHub per tal de distribuir la càrrega de manera igualitària entre els nodes on es troben desplegats les nostres APIs.
Les funcionalitats dels Load Balancers són les següents:

- Balancejar les càrregues que suporten els diferents nodes workers del CloudHub.
- Configurar SSL per tal de definir certificats per a l'autentificació de clients.
- Configurar normes Proxy per tal de mapejar les aplicacions Mule a dominis determinats.

Per tal de crear i configurar els Load Balancers ho fem a través de les APIs de Cloudhub a través dels següents endpoints:

- anypoint.mulesoft.com/cloudhub/api/organizations/{orgid}/loadbalancers
- anypoint.mulesoft.com/cloudhub/api/organizations/{orgid}/vpcs

on orgid serà la id assignada a la nostra empresa (que per raons de seguretat no es posarà en aquest informe).
Des d'aquests endpoints es podrà configurar els Load Balancers de les nostres aplicacions Mule. De moment a nivell d'empresa es configurarà un node Manager cada dos nodes Workers perquè distribueixi el tràfic de peticions a les nostres aplicacions Mule desplegades al CloudHub.
Pel volum de peticions i d'aplicacions que gestionem en aquest projecte potser no seria necessari preparar i configurar aquests balancejadors de càrrega però com que aquest projecte és només una petita part de tota la transformació tecnològica de l'empresa es va decidir d'implementar-los per poder-los reaprofitar en un futur.

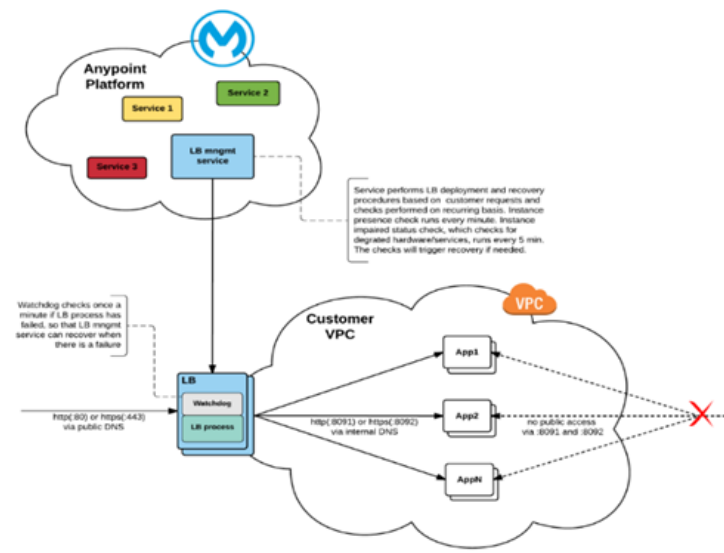

Fig. 6 Estructura i Funcionament dels Load Balancers

Per tal de testejar el correcte funcionament executarem uns scripts que ens proporciona directament MuleSoft. Aquests scripts generen tràfic creant missatges mock i podem observar si el balanceig segueix els estàndards que ha de seguir o no.

6.4 Desenvolupament de les APIs

Un cop construïts els balancejadors de càrrega ja podem passar a construir les nostres APIs que gestionin la comunicació entre les diferents aplicacions del nostre projecte.
El primer que hem de decidir és quines APIs s'han de programar per tal de garantir la comunicació del frontal amb la resta de xarxa. Per al nostre frontal necessitarem APIs que ens permetin la comunicació en tres canals, el d'autentificació de l'usuari, el d'accés a la base de dades on està tota la informació de les vendes i el de traduccions.
Un cop tenim totes les APIs programades les publicarem en el cloud (CloudHub) amb els endpoints corresponents per tal que el frontal pugui fer les crides i obtenir les respostes correctes.

6.4.1 Seguretat

En aquest apartat passarem a explicar com hem desenvolupat les APIs per gestionar la seguretat del frontal. En el nostre cas per tal d'autentificar l'usuari els passos a seguir són els següents.
L'usuari introdueix el seu codi d'empleat i la seva contrasenya al formulari d'inici de sessió, aquest s'envia xifrat al servidor de seguretat que al comprovar que les credencials són correctes envia un token d'autentificació que l'usuari es guarda. Això permetrà a l'usuari accedir al nostre frontal en les següents dues hores, un cop passades les dues hores el token caduca i s'haurà de tornar a sol·licitar un de nou.
Així doncs amb una sola API n'hi haurà prou per a gestionar la comunicació que consistirà amb un GET del frontal al servidor per autenticar i el retorn d'informació per part del servidor. La part més complexa del desenvolupament d'aquesta API serà el fet que la informació del usuari ha de viatjar xifrada. Per aconseguir això utilitzarem OKTA [15], una plataforma de seguretat que disposa l'empresa per tal de xifrar les dades i que no viatgin de manera descoberta per la xarxa.

6.4.2 Consumer Repositories

Un cop tenim l'inici de sessió confirmat, passarem a la part més complexa que és que el frontal mostri tota la informació de les vendes. Per tal d'aconseguir tota la informació de les vendes hem de fer consultes en les bases de dades indicades. Aquestes consultes poden arribar a ser força complexes, per tant les separarem en diferents APIs per no sobrecarregar el sistema de missatgeria.
Separarem les consultes en tres grans grups, la consulta per aconseguir el llistat de productes, la consulta per aconseguir la imatge assignada a cada producte i la consulta per aconseguir la informació específica quan se selecciona el producte.
Un cop establerts els contractes (quin tipus/estructura de dades envia la BBDD i quin tipus/estructura espera el frontal) s'han de programar els muleFlows. Aquests flows són el camí que segueixen aquestes dades per tal de realitzar els canvis pertinents per tal que no sorgeixi cap problema i incompatibilitat. Aprofitarem per afegir els logs corresponents per tal de monitoritzar tot el procés més endavant.

6.4.3 Traduccions

Aquesta és la API més senzilla de realitzar. El frontal indicarà l'idioma del qual es necessiten les traduccions fent una petició GET amb un paràmetre on s'indicarà de quin idioma es necessiten les traduccions, per exemple si es necessiten les traduccions en anglès, el paràmetre serà 'EN'. Això s'envia al servidor de traduccions que respondrà amb un fitxer JSON on hi haurà totes les claus que s'utilitzen amb el seu equivalent en l'idioma sol·licitat. No s'haurà de canviar cap paràmetre en el fitxer JSON de manera que el muleFlow serà només amb logs de que tot el procés ha anat correctament. Aquest nou fitxer JSON serà el utilitzat per el frontal per tal de traduir tot el HTML.

6.4.3 Tests de les APIs

A l'hora d'editar els muleFlows també aprofitarem i crearem els tests de comportament (tests unitaris) per tal de comprovació el correcte funcionament de la API. Com hem explicat abans en l'apartat de test del frontal els tests unitaris consisteixen en validar cada unitat de codi que funcioni correctament. En el context de les APIs la unitat més petita testejable que podem trobar son els sub-flows. Normalment aquests sub-flows estan en contacte directe amb altres flows i per tal d'aïllar aquestes parts a l'hora de realitzar el testeig haurem de crear mock data per observar el comportament del codi.
També ens ajudarem de Munit [16], un framework que ens permetrà automatitzar fàcilment el testeig d'aplicacions Mule. MUnit ens permet programar en test en codi Mule o en Java, crear missatges mock, verificar les transformacions de les dades dels missatges...

7 Resultats

En aquest apartat explicarem els resultats obtinguts en aquest treball. Primer explicarem com fem la monitorització i creem les mètriques de MuleSoft, després com fem una auditoria al bus OSB12. Finalment amb els resultats obtinguts en les dues plataformes farem una anàlisi dels resultats obtinguts.

7.1 Mètriques i Monitorització de les APIs de MuleSoft

Aquí passarem a explicar com hem creat les mètriques i com hem gestionat la monitorització de les APIs que tenim desplegades en el projecte. A l'hora de crear les mètriques i monitoritzar les nostres APIs utilitzarem les plataformes de MuleSoft de Visualizer i Monitoring.
El Visualizer serveix per crear un mapa visual de la xarxa de APIs que es troben desplegades en el cloudhub, detec-

tant les dependències i les interaccions que es creen entre elles.
Es troben separades en els diferents layers que hem explicat abans, process, utility, System... Això ens serveix per veure rapidament si hi ha alguna API sobrecarregada (masses dependencies)... És una eina molt important sobretot en empreses on la xarxa d'aplicacions és molt gran i necessitem una visió global de totes les APIs que tenim en funcionament (producció).
Per tal de monitoritzar tot el seguiment de les dades utilitzarem MuleSoft Monitoring. Aquesta eina ens dóna un gran ventall de possibilitats per tal de monitoritzar les nostres APIs. Nosaltres aprofitarem els Reports que genera cada 24 hores on es mostraran la Performance, Failures, CPU Utilization i la Memory Utilization, que ens serviran per fer un checkout, normalment pel matí, de que tot funciona correctament. També crearem alertes que ens avisaran si hi ha algun comportament que no és l'adequat. Establim certs paràmetres i si alguna API ho supera el propi sistema ens avisa.

7.2 Auditoria de l'OSB12

En aquest apartat passarem a explicar l'auditoria que es va realitzar el 13/11/2019 al bus de dades d'Oracle OSB12. El motiu principal de l'auditoria és per revisar el funcionament i rendiment del OSB12 dues setmanes abans del BlackFriday, una data clau per l'empresa degut al gran volum de dades de comunicació que circulen pel bus durant aquesta jornada.
En el nostre cas aprofitarem aquesta auditoria per extreure les dades de rendiment i així poder-les comparar amb les obtingudes per el nostre projecte de MuleSoft. D'aquesta manera podem realitzar una comparació que farem més endavant en l'apartat d'anàlisi de resultats.
L'objectiu d'aquesta auditoria es observar el volum de registres del bus durant períodes de càrrega massiva. D'aquesta manera es vol intentar reduir aquest volum de registres i/o permetre millorar la informació "útil" que es registra en la BBDD de l'auditoria a través del bus de dades d'Oracle. Primer definirem un estàndard en els nivells de càrrega del bus com podem observar en la taula següent.

Estat	Descripció
CRITICAL	Representa una situació crítica del sistema. Suposa un risc imminent per el sistema.
STRESS	Representa una situació delicada, de molta activitat que podria arribar a desestabilitzar el sistema.
CALM	Representa la situació de normalitat de càrrega del sistema i el estat configurat per defecte. Estat adequat.
OFF	Aquest estat s'utilitza per parar tot l'enviament de registres d'auditoria a les cues.

Un cop establerts els nivells que pot assolir el bus posarem en marxa l'auditoria, que consistirà en executar uns scripts que generin un tràfic elevat amb variacions per tal d'observar el seu comportament. Per tal de recol·lectar la informació seguint l'esquema mostrat a continuació. Primer sol·licitem el servei del bus, un cop ens permet l'accés podem fer la consulta a la cache de l'auditoria. Aquest procediment és el responsable de recuperar l'estat actual de l'auditoria del servei. La cache conté un objecte XML amb els nivells que hem establert en la taula anterior, que obté d'una consulta de la base de dades de l'auditoria.

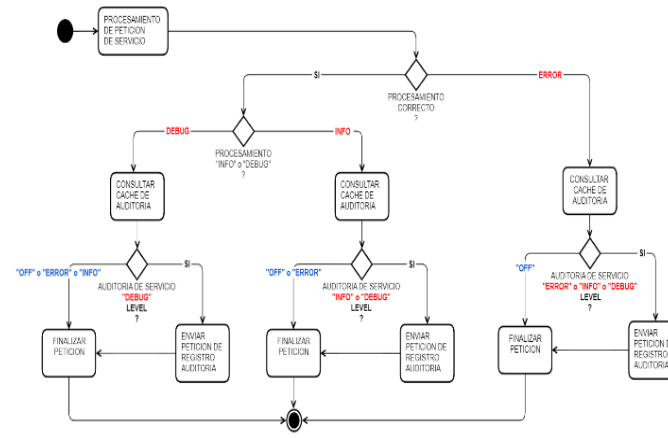

Fig. 8 Procediment de l'auditoria del OSB12

7.3 Anàlisi de Resultats

En aquest apartat passarem a analitzar els resultats obtinguts en l'anàlisi de rendiment de l'auditoria i MuleSoft. Per realitzar aquests anàlisi de resultats observarem en profunditat els resultats de la auditoria i els resultats que hem anat observant en els diaris de salut tant de MuleSoft com del bus. Els diaris de Salut són el checkout diari que es realitzen en les dues plataformes on es recullen els paràmetres basics per tal de detectar si ha sorgit algun problema.
Els paràmetres que analitzarem principalment seran els de perfomance (missatges enviats vs missatges rebuts) la utilització de memòria i la utilització de CPU. En les gràfiques podem veure la mitjana dels paràmetres obtinguts en les APIs de Mulesoft per els dies que vam fer la auditoria al bus.

Performance

Application Name	Request Volume	Response Time
Prod-e-OAuth	26	507 ms
Prod-e-Trad	7	161 ms
Prod-e-Sell	767	1081 ms
Prod-u-Datasell	545	764 ms
Prod-u-Imagesell	545	1309 ms
Prod-u-Productsell	52	489 ms
Prod-s-DBAcess	321	431 ms

Failures

Application Name	Request Volume	Failed Request	Successful Request
Prod-e-OAuth	26	2	24
Prod-e-Trad	7	0	7
Prod-e-Sell	767	0	767
Prod-u-Datasell	545	0	545
Prod-u-Imagesell	545	3	545
Prod-u-Productsell	52	0	52
Prod-s-DBAcess	321	0	321

CPU Utilitzation

Application Name	CPU Utilitzation
Prod-e-OAuth	1.03 %
Prod-e-Trad	0.67 %
Prod-e-Sell	0.52 %
Prod-u-Datasell	4.37 %
Prod-u-Imagesell	7.46 %
Prod-u-Productsell	3.25 %
Prod-s-DBAcess	6.98 %

Memory Utilitzation

Application Name	Memory Utilitzation	Total Memory	Memory Pressure
Prod-e-OAuth	177.1 MB	456 MB	37 %
Prod-e-Trad	151.9 MB	456 MB	31 %
Prod-e-Sell	206.3 MB	456 MB	43 %
Prod-u-Datasell	276.4 MB	456 MB	57 %
Prod-u-Imagesell	320.8 MB	456 MB	67 %
Prod-u-Productsell	228 MB	456 MB	47 %
Prod-s-DBAcess	311.5	456 MB	68 %

Com podem observar en el gràfic següent podem notar una lleugera millora en la performance però on sobretot podem dir que l'avantatge és important es en utilització de memòria i CPU.

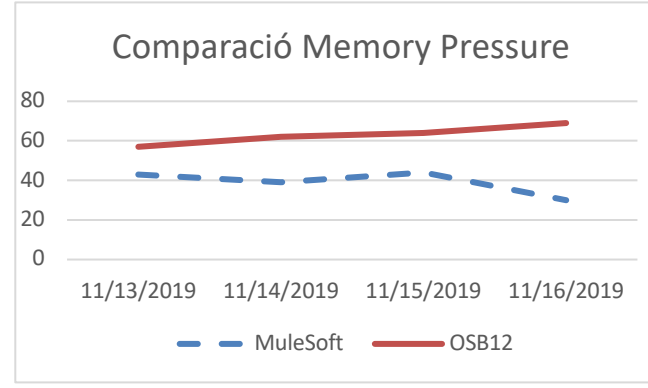

Fig. 9 Gràfic de comparació de Memory Pressure entre MuleSoft i OSB12

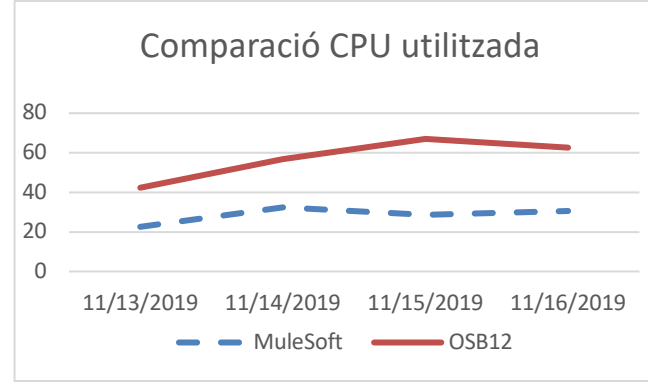

Fig. 10 Gràfic de comparació de CPU Utilitzada

8 Conclusions

En aquest apartat es presentaran les conclusions obtingudes al llarg d'aquest projecte. Les conclusions les dividirem en dues parts. La primera de les conclusions explicarem les conclusions del funcionament del frontal amb tota la nova arquitectura muntada i després les conclusions de la comparació entre l'antiga arquitectura d'Oracle que hi havia muntada i la nova arquitectura de comunicació basada en MuleSoft.

Com a primera conclusió podem afirmar que hem construït amb èxit un frontal web que es comunica amb la resta d'aplicacions de la xarxa empresarial mitjançant l'arquitectura MuleSoft. El rendiment de la pagina web és l'esperat i vist el resultat els usuaris finals d'aquest frontal estan satisfets amb l'ús que en fan.

Aquesta eina serà molt utilitzada per l'equip de negoci a l'hora de prendre decisions ja que la informació que obtenim és fiable i de gran utilitat.

També podem afirmat vist l'anàlisi de resultats que ha sigut una bona decisió el canviar l'arquitectura de gestió de comunicació entre aplicacions. Com hem explicat anteriorment en l'anàlisi de resultats la millora tècnica és im-

portant però també hauríem de tenir en compte la gran facilitat que ofereix MuleSoft per als programadors. El fet de que les APIs siguin reutilitzables agilitzen molt el seu desenvolupament com també que es disposi de un entorn que fa la feina molt més fàcil (Anypoint Platform).
En el nostre cas s'ha reduït un 30% el temps en el desenvolupament d'aquestes noves APIs si ho comparem amb el temps que es tardava en fer-ho amb el bus de dades d'Oracle, i inclús podria ser millor ja que no hem reutilitzat cap API ni hem utilitzat cap API del Exchange. També el fet que existeixi un "mercat" d'APIs, el Anypoint Exchange també facilita la feina en el desenvolupament d'aquestes APIs.

8.1 Treball futur

En aquesta part exposarem les possibles millores o camins que pot seguir en un futur. Aquestes millores estan pensades sobretot per millorar la xarxa de MuleSoft que hem començat a crear amb aquest projecte. No s'han arribat a dur a terme en aquest projecte degut a la magnitud de treball i hores que comporten però ja s'estan portant a terme els primers passos perquè sigui una realitat. Les millores proposades són:

- Implementar el sistema de SalesForce [17] dintre la xarxa d'aplicacions de MuleSoft. Salesforce és un CRM [18] (Costumer Relationship Managment) per gestionar tota la xarxa de clients de l'empresa.
- Transformar tot el sistema d'E-business (botiga online) perquè la comunicació sigui a través de les APIs de MuleSoft (des de que es crea la comanda fins a l'enviament amb tots els processos del mig inclosos).
- Buscar una nova arquitectura diferent de les estudiades en les que s'obtinguin millors resultats que en els obtinguts en aquest treball.

Agraïments

M'agradaria agrair el paper que ha pres l'empresa en el desenvolupament del projecte, sobretot agrair a Alberto Gutierrez Maceiras, el meu tutor a l'empresa per l'ajuda i el suport que m'han donat. També m'agradaria donar les gràcies als meus pares, companys de classe i companys de feina per la paciència i l'ajuda que m'han facilitat al llarg de tot aquest projecte.

Bibliografia

[1] LOLIĆ, Teodora, et al. INTEGRATION OF APPLICATIONS USING ORACLE SOA AND MULESOFT, 2014.

[2] SCHMUTZ, Guido; BIEMOND, Edwin. *Oracle Service Bus 11g Development Cookbook*. Packt Publishing Ltd, 2012.

[3] BURGOS, Roberto Núñez. *Software ERP: análisis y consultoría de software empresarial*. IT Campus Academy, 2016.

[4] CHEN, Liqiang. Integrating cloud computing services using Enterprise Service Bus (ESB). *Business and management research*, 2012, 1.1: 26-31.

[5] ZUR MUEHLEN, Michael; NICKERSON, Jeffrey V.; SWENSON, Keith D. Developing web services choreography standards—the case of REST vs. SOAP. *Decision support systems*, 2005, 40.1: 9-29.

[6] SURWASE, Vijay. REST API modeling languages-a developer's perspective. *Int. J. Sci. Technol. Eng*, 2016, 2.10: 634-637.

[7] KNIBERG, Henrik, et al. Kanban y Scrum–obteniendo lo mejor de ambos. *Prólogo de Mary Poppendieck & David Anderson. C4Media Inc*, 2010.

[8] FISHER, John, et al. Utilizing Atlassian JIRA for large-scale software development management. In: *14th International Conference on Accelerator & Large Experimental Physics Control Systems (ICALEPCS)*. 2013.

[9] CRYER, James. Using Grunt with HTML and CSS. In: *Pro Grunt. js*. Apress, Berkeley, CA, 2015. p. 31-58.

[10] WOHLGETHAN, Eric. *SupportingWeb Development Decisions by Comparing Three Major JavaScript Frameworks: Angular, React and Vue. js*. 2018. PhD Thesis. Hochschule für Angewandte Wissenschaften Hamburg.

[11] MASSE, Mark. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. " O'Reilly Media, Inc.", 2011.

[12] ARMBRUST, Michael, et al. A view of cloud computing. *Communications of the ACM*, 2010, 53.4: 50-58.

[13] RAGONHA, Paulo. *Jasmine JavaScript Testing*. Packt Publishing Ltd, 2013.

[14] BASS, Len; WEBER, Ingo; ZHU, Liming. *DevOps: A software architect's perspective*. Addison-Wesley Professional, 2015.

[15] LEIBA, Barry. Oauth web authorization protocol. *IEEE Internet Computing*, 2012, 16.1: 74-77.

[16] DAVIS, Andrew. Quality and Testing. In: *Mastering Salesforce DevOps*. Apress, Berkeley, CA, 2019. p. 269-336.

[17] MANCHAR, Anuradha; CHOUHAN, Ankit. Salesforce CRM: A new way of managing customer relationship in cloud environment. In: *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*. IEEE, 2017. p. 1-4.

[18] CHORAFAS, Dimitris N. *Integrating ERP, CRM, supply chain management, and smart materials*. Auerbach Publications, 2001.