UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI INGEGNERIA INFORMATICA, MODELLISTICA, ELETTRONICA E SISTEMISTICA

TESI DI LAUREA IN INGEGNERIA INFORMATICA

L'utilizzo della piattaforma Mulesoft per lo sviluppo di API: il progetto Contentful

---

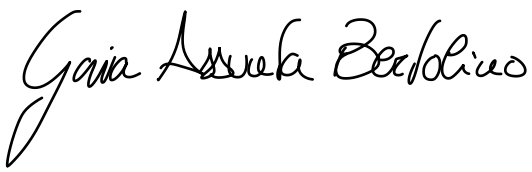
Relatore

Chiar.mo Prof. Giancarlo Fortino

Candidata

Gaia Assunta Bertolino

Matricola 209507



Anno accademico 2021/2022

"People who are crazy enough to think they can change the world, are the ones who do"

Steve Jobs

Alla mia cara nonnina,
un giorno ci abbracceremo di nuovo

Indice

Sommario

Elenco delle figure

Abstract

L'integrazione fra sistemi dotati di tecnologie differenti, sia dal punto di vista temporale che strutturale, richiede un approccio capace di velocizzare, semplificare e democratizzare l'uso di tools di sviluppo in grado di gestire a trecentosessanta gradi le funzionalità delle interazioni.

Nell'ambito di questa tesi, gli aspetti dell'integrazione fra sistemi che operano in modo eterogeneo, tra cui ad esempio cloud, SaaS e microservices, sono trattati attraverso lo sviluppo di API (abbreviazione per Application Programming Interface), il cui fine è di gestire le modalità di interazione con un singolo servizio.

Lo scopo di questo lavoro è di descrivere il funzionamento dell'ambiente fornito dall'azienda Mulesoft, il quale è in grado di offrire una serie di strumenti che accompagnano il programmatore dalla scrittura della specifica alla pubblicazione di una API, illustrando il suo utilizzo attraverso un progetto concreto realizzato per l'azienda Contentful e le possibili applicazioni dell'ambito dell'Internet delle Cose.

Introduzione

Secondo una ricerca di Juniper Research risalente al 2020, nel 2024 il numero di connessioni IoT (abbreviazione per *Internet of Things*) raggiungerà gli 83 miliardi, con una crescita del 130% rispetto al 2020. Un tale incremento, legato alle nuove ed emergenti necessità e alle tecnologie risolutive proposte, solleva delle questioni inerenti diverse tematiche, tra le quali la sicurezza, la velocità di sviluppo, l'interscambiabilità delle informazioni fra tecnologie profondamente differenti e la gestione dei dati raccolti dagli stessi smart objects.

Da sempre lo scopo delle grandi aziende è rimanere al passo delle esigenze del mercato, il quale si aspetta una continua consegna di prodotti nuovi e maggiormente performanti rispetto ai precedenti. Dunque, il processo di evoluzione delle tecnologie di produzione vira con sempre più forza verso la creazione di supporti in grado di velocizzare i meccanismi produttivi senza che ciò intacchi la qualità del prodotto stesso. In ambito informatico, ciò si realizza tramite l'utilizzo di piattaforme ad hoc che offrono tools in grado di semplificare gli step coinvolti nella creazione e nel testing di software nonché framework e librerie che forniscono soluzioni immediate e spesso ottimizzate per le problematiche degli sviluppatori [1].

In questa tesi è illustrato un approccio strutturato per la realizzazione delle interfacce di interazione con servizi e prodotti attraverso l'uso della piattaforma Mulesoft, in grado di fornire tools utili alla semplificazione nonché automatizzazione della programmazione e della gestione delle interfacce stesse. Nello specifico, si vuole affrontare il suo possibile utilizzo nell'ambito dello sviluppo API per l'Internet delle Cose riportando, a tale scopo, un progetto concreto per mostrare le funzionalità dell'*environment* nonché la tipologia di approccio fornito allo sviluppatore.

Nel primo capitolo, in particolare, si descrivono le innovative funzionalità messe a disposizione dall'azienda Mulesoft, tra cui una piattaforma di development basato su codice a blocchi e un sistema interno di gestione del deployment.

Nel secondo capitolo, si approfondisce un progetto concreto realizzato tramite tale environment per la crescente azienda Contentful nell'ambito di un Erasmus Traineeship sviluppato presso l'azienda Cap4Lab a Lussemburgo.

Nel terzo capitolo, infine, vengono trattate le possibili applicazioni nell'ambito dell'Internet delle Cose e di come Mulesoft sia in grado di fornire uno strumento innovativo e aderente alle esigenze per la realizzazione di interfacce di scambio di dati fra sistemi di smart objects.

Capitolo 1: L'*environment* di Mulesoft

Mulesoft, acquisita nel 2018 da Salesforce, è una piattaforma di integrazione software [2][3] utilizzata oggigiorno da molte aziende tech leader nel settore ICT come Accenture ed emergenti come Cap4Lab. Essa mette a disposizione dello sviluppatore di API una serie di strumenti che accompagnano la produzione a partire dalla programmazione fino al testing.

1.1 ACCENNI SULLE API

Nella sua accezione più generale, con il termine API (acronimo di *application programming interface*) si indica un insieme di pratiche atte allo scopo di porre in comunicazione diversi sistemi, spesso dotati di tecnologie differenti e quindi non in grado di scambiare messaggi senza un ausilio terzo.

Per estensione, nell'uso comune il termine fa riferimento ad anche altre tre categorie:

- API *specification* - la specifica delle funzionalità di una interfaccia
- API *proxy* - l'interfaccia che regola l'accesso a delle risorse
- *Web service* - l'implementazione dell'interfaccia come regolatore della comunicazione fra un client e un server

Dal punto di vista dell'API intesa come interfaccia di gestione di un web service, la tecnologia utilizzata può fare riferimento a due tipi diversi di standard:

- Un RESTFUL Web Service, chiamato semplicemente REST, non prevede un vero e proprio protocollo ma un approccio flessibile in quanto si compone di linee guida e dei cosiddetti principi del REST *(Representational State Transfer)* la cui implementazione pratica è lasciata allo sviluppatore. La rappresentazione dell'informazione avviene sottoforma di ipermedia in grado di contenere simultaneamente i dati e la parte di controllo e un oggetto può essere restituito sotto diversi formati come ad esempio JSON, XML o testo. In particolare, JSON è un formato di interscambio orientato ai dati mentre XML è un linguaggio di markup orientato ai documenti, privo di una semantica intrinseca, e quindi ideale per la personalizzazione attraverso un linguaggio che sia comprensibile sia per l'uomo che per i computer.
- Un SOAP Web Service (acronimo di s*imple object access protocol)* è un vero e proprio protocollo che segue il paradigma della programmazione orientata agli oggetti e opera su differenti protocolli di rete, prediligendo solitamente l'HTTP.

Esso si basa su XML e la sua struttura segue la configurazione head-body, analogamente ad HTML.

Lo scopo generico di una API è fornire un servizio ad un client sotto le sembianze di una scatola nera: permette, infatti, di poter accedere ai dati di un sistema senza doverne necessariamente conoscere la tecnologia, agendo perciò come intermediario. Dunque, accetta le richieste dall'esterno e le elabora per passarle al sistema sottostante da cui trae una risposta e/o dei dati da restituire. Tale principio è ripreso nel concetto di *API-led connectivity*, il quale prevede che le interazioni fra sistemi differenti siano realizzate attraverso una gerarchia di interfacce che incapsulino singoli servizi e li rendano fruibili ai livelli superiori della struttura [4][5].

Le API sviluppate secondo l'approccio appena descritto vengono catalogate in tre categorie in base alla loro funzione:

- System APIs – Svolgono il compito di accedere ai dati e alle risorse di base come ad esempio databases, web services e applicazioni SaaS. Ciò permette di fornire una modalità di accesso indiretto in grado di nascondere al client la complessità nell'interrogare le strutture di back-end.
- Process APIs – Le API di tale livello hanno lo scopo di aggregare i dati prelevati dal livello inferiore per renderli indipendenti dalla sorgente (secondo il concetto del *breaking down data silos*) e processandoli secondo specifiche richieste per renderli maggiormente fruibili al livello superiore.
- Experience APIs – Sono atte a definire la formattazione finale dei dati che sarà fornita al client per mezzo di piattaforme diverse. Le *experience* APIs sono, dunque, sviluppate secondo l'uso che il client specifico dovrà farne.

Figura 1: Struttura a tre livelli secondo l'API-led connectivity

Fonte: https://blogs.mulesoft.com/bloghome/

Rispetto ad una tecnologia *point-to-point* che risente della sua specificità e tendenza ai guasti, l'approccio API-led per la creazione delle connessioni fra applicazioni è caratterizzato da proprietà focali che lo rendono altamente funzionale nel mercato odierno [6]; dal punto di vista dell'ingegneria del software, disciplina che si occupa dell'applicazione di un approccio sistematico, disciplinato e quantificabile nello sviluppo, funzionamento e manutenzione del software, esse sono riassumibili sotto quelle che vengono definite qualità del software. In particolare:

- L'incapsulamento delle operazioni si realizza scomponendo una richiesta in parti più semplici gestibili da singole API. Ciascuna di esse ha il compito di rendere il più possibile generica la propria operazione, implementando così la riusabilità, altra importante proprietà.
- L'applicazione del concetto del *dividi et impera* comporta la realizzazione di più API per una singola operazione; tuttavia, esse presentano un'alta riusabilità in quanto vengono progettate per essere orientate il meno possibile al singolo progetto ma più genericamente ad un qualsiasi loro uso. Infatti, è poi l'API di livello superiore ad estrapolare dall'inferiore il sottoinsieme di informazioni ricercate, il quale è tuttavia in grado di offrire un servizio completo e riutilizzabile anche in altri contesti. Tale meccanismo di comunicazione e selezione dei dati desiderati viene perpetuato fra i vari livelli, diminuendo il grado di generalizzazione per avvicinarsi alla richiesta specifica del client.
- Un sistema così *layerizzato* e incapsulato è altamente *fault tolerant* e robusto [7]. Nell'ingegneria dell'affidabilità, la tolleranza ai guasti (o *fault-tolerance*) è la capacità di un sistema di non subire avarie, cioè interruzioni di servizio, anche in presenza di guasti e, insieme ad altri aspetti, concorre a caratterizzare l'affidabilità di un sistema, definita come la probabilità che esso rispetti nel tempo il comportamento atteso. Dunque, la mancanza di uno dei servizi della catena non provoca necessariamente la caduta dell'intero sistema ma può essere opportunamente gestita in attesa di una riparazione. Col termine robustezza si intende, invece, la capacità di un software di comportarsi in maniera ragionevole in caso di situazioni non previste dalle specifiche come, ad esempio, input non corretti o malfunzionamenti hardware. Nei problemi di robustezza rientrano i requisiti progettuali che non fanno parte della specifica.
- La suddivisione secondo diversi livelli di filtraggio delle informazioni, unita alla realizzazione di specifiche experience APIs per i diversi sbocchi front-end, influenza ulteriori qualità esterne quali la comprensibilità, implementata attraverso la personalizzazione delle interfacce, e l'usabilità, intesa come la facilità di utilizzo della tecnologia da parte dell'utente, la quale migliora di pari passo con la specificità della realizzazione.

- La realizzazione indipendente di ciascuna funzione tramite una API genera un'alta portabilità del sistema in quanto, dato un nuovo ambiente di esecuzione, vi è la necessità di redigere nuovamente solo la parte inerente alla experience API piuttosto che l'intera struttura.

Inoltre, le proprietà di incapsulamento e modularità influenzano positivamente altre qualità quali:

- La manutenibilità, intesa come la misura nella quale il sistema si presta ad interventi successivi al rilascio. Essa è favorita dalla modularità in quanto è possibile modificare un singolo modulo indipendentemente dagli altri senza che l'intero sistema ne risenta.
- L'evolvibilità, definita come la capacità del software di cambiare in maniera ordinata e controllata sfruttando accortamente la propria duttilità intesa come la possibilità di apportare dei cambiamenti senza sforzo al prodotto finito. È importante sottolineare come, in generale, la modularità garantisca una maggiore evolvibilità ma modifiche successive nel tempo possono introdurre dipendenze fra i moduli per cui tale qualità tende a diminuire con rilasci successivi.
- La riparabilità, la quale esprime la misura nella quale è possibile correggere dei difetti con un costo ragionevole in termini economici e di risorse umane.
- Le prestazioni, le quali riguardano aspetti come l'efficienza del sistema, l'utilizzo di risorse e la scalabilità definita come la capacità del sistema di funzionare anche con input di dimensioni maggiori e senza incremento esponenziale dei costi.

In particolare, la modularità esprime il suo potenziale nel momento in cui esiste un'alta coesione interna a ciascun modulo, dove per coesione si intende la dipendenza fra le risorse, e una bassa coesione esterna fra moduli, qui intesi come API. Ne consegue che le qualità sopra elencate possono risentire sia positivamente che negativamente del principio appena esposto.

Oggigiorno, l'approccio di tipo API-led risulta essere chiave di volta per la necessità di sviluppare servizi multipiattaforma in grado di funzionare su dispositivi con software e hardware differenti grazie alla presenza intrinseca di riutilizzo del codice.

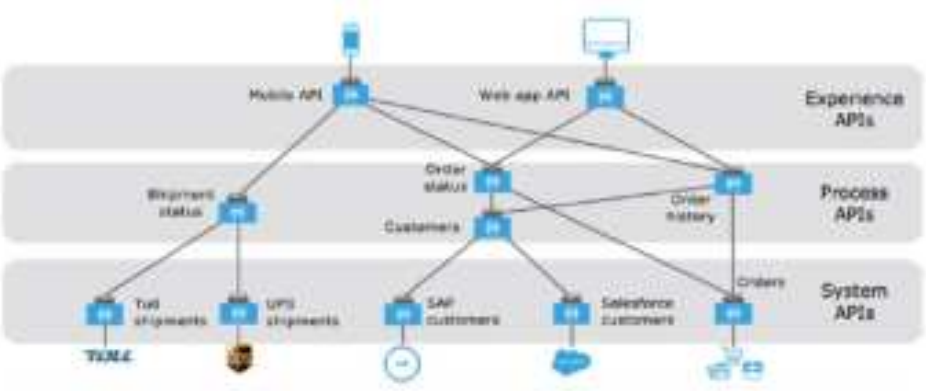


Figura 2: Esempio di struttura multilayer sviluppata secondo il principio dell'API-led connectivity

Fonte: https://unogeeks.com/mulesoft-architecture-and-api-led-connectivity/

Inoltre, secondo il Connectivity benchmark report del 2022, nell'ambito dell'IT (acronimo per *Information Technology*), è spesso necessario aumentare il carico di lavoro, nonostante le risorse disponibili rimangano immutate, a causa dell'aspettative di molte aziende che il budget rimanga lo stesso o incrementi di poco a fronte di maggiori commissioni. Ciò causa il cosiddetto *IT delivery gap* fra capacità e richieste, il quale crea la necessità di ricorrere a delle tecnologie in grado di diminuire tale distanza.

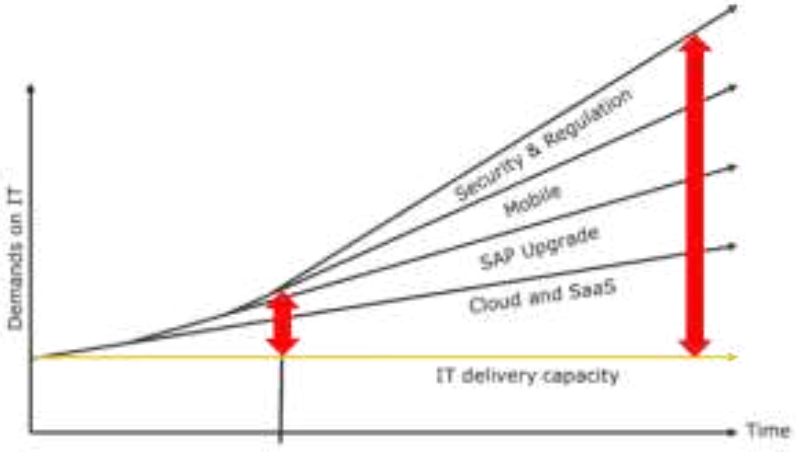


Figura 3: IT delivery Gap

Fonte: https://blogs.mulesoft.com/digital-transformation/business/how-to-increase-it-productivity/

Tramite l'approccio fornito dall'API-led connectivity, si ottiene una velocità di *project delivery* tra le tre e le cinque volte più veloce nonché un aumento della team productivity stimata da Mulesoft in circa il 300%.

In aggiunta a ciò, l'utilizzo di una gerarchia di API permette di realizzare un'operazione di filtraggio delle informazioni al quale l'utente finale può avere accesso oltre che a realizzare diversi possibili livelli di autorizzazione in base alle credenziali inserite o, se posta come porta di un sistema basato su tecnologia meno recente o su standard di comunicazione poco diffuso, può fornire un metodo di accesso completamente moderno e universale [8].

1.2 MULESOFT TOOLS

La piattaforma Mulesoft fornisce una serie di strumenti in grado di gestire l'intero processo di sviluppo e testing di una API.

L'Anypoint Design Center è un web-based tool che permette la realizzazione della specifica attraverso linguaggi come RAML e OpenAPI, la quale è utilizzata per definire quali sono i comportamenti e le modalità di integrazione dell'API con altri sistemi. La specifica ottenuta è poi pubblicabile sul market virtuale di Anypoint Exchange dove è anche possibile scoprire connettori API che le aziende o altri sviluppatori forniscono per accedere ai propri dati e servizi. Tra i vari servizi, rende possibile pubblicare nuove versioni di un connettore personale e fornisce una sezione di *mocking* dove poter realizzare una prima fase di testing, dove per mock si intende un servizio in grado di simulare il comportamento di oggetti reali in modo controllato.

La sicurezza è, invece, gestita attraverso il pannello Anypoint Security mentre il Management Center è l'hub operazionale per gestire e analizzare le performance e le funzionalità di una API. Esso comprende tre parti:

- Runtime Manager - responsabile del deployment e del monitoraggio delle applicazioni, compresa la loro performance.
- API Manager - gestore delle policies di sicurezza e dei gateaway.
- Analytics - gestore del traffico e di metriche di performance.

La manipolazione dei dati viene operata attraverso il linguaggio Dataweave, basato su Java. Per quanto riguarda l'implementazione dell'interfaccia, essa può avvenire sia attraverso l'ambiente online del Flow Design sia sulla piattaforma offline Anypoint Studio, un ambiente di sviluppo integrato (IDE) basato su Eclipse.

1.3 IL LINGUAGGIO RAML

Il RAML (acronimo di *RESTFul API Modeling Language*) è un linguaggio open source di facile comprensione per l'uomo utilizzato per realizzare la specifica di una API [9].

Esso imita la formattazione YAML e prevede la possibilità di specificare una serie di parametri, tra i quali:

- Lo *Uniform Resource Identifier* (in acronimo URI) di base rappresenta il punto di accesso univoco ed universale al servizio. A partire da esso è possibile definire una gerarchia di livelli, identificati da una parola chiave anteceduta da un backslash, ciascuno dei quali permette l'interazione con una diversa sezione. Inoltre, per ciascun layer è possibile specificare gli URI parameters, i quali sono indicati fra parentesi graffe e vengono utilizzati per accedere ad una singola risorsa caratterizzata da un valore identificativo unico. I parametri di tipo URI si distinguono dai query parameters in quanto quest'ultimi sono utilizzati per eseguire un filtraggio all'interno di un insieme di risorse attraverso l'uso di variabili.

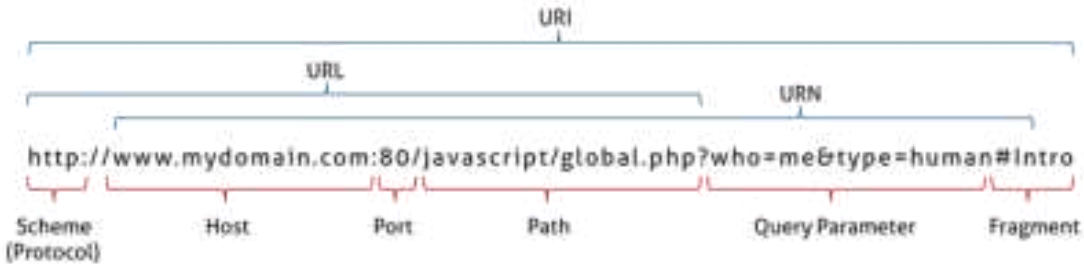


Figura 4: Esempio di composizione di un URI

Fonte: https://hanseul-lee.github.io/2020/12/24/20-12-24-URL/

Per ciascun nodo è possibile impostare quali sono i metodi HTTP di accesso disponibili. Riassunti sotto l'acronimo CRUD, i principali sono:

- GET (o Read): prelievo di una informazione.
- POST (o Create): alterazioni di dati (inserimento, modifica o eliminazione).
- PATCH (o Update): sostituzione parziale dei dati di un record. Essa si differenzia dalla PUT che ne prevede l'intera sovrascrittura.
- DELETE: eliminazione di un record.

Per ciascun metodo viene definita la formattazione richiesta per i dati associati da inviare in input nonché le modalità di risposta, la quale può comprendere

un oggetto e/o un messaggio specifico per ogni casistica; inoltre, è possibile realizzare una differenziazione nell'output da restituire nei casi di riscontro positivo e negativo.

- I tipi di dati utilizzati all'interno dell'API. Per chiarezza e modularità, essi vengono definiti in file separati e successivamente importati all'interno del principale. La formattazione di ciascun tipo si comporta in maniera imperativa in quanto è in grado di generare automaticamente una eccezione se i dati rilevati non rispettano le informazioni in essa contenute. Allo stesso modo è possibile definire anche degli esempi dei tipi di dati, utili sia per mantenere chiarezza all'interno del team di sviluppatori sia per la visualizzazione nell'ambito dell'Exchange Market.
- Parametri vari che definiscono, tra gli altri, la sicurezza, le librerie esterne utilizzate e il riutilizzo di blocchi di codice. Quest'ultimo viene operato tramite il comando *traits*.

Il RAML è stato sviluppato affinché non funga da sola documentazione ma sia anche in grado di modellare una API. Infatti, tramite un'opportuna importazione della specifica all'interno del progetto, l'IDE Anypoint Studio è in grado di generare automaticamente lo scheletro dell'API, successivamente popolabile con l'implementazione, fornendo autonomamente tutte le verifiche sulla formattazione dei dati.

Segue un esempio di specifica RAML completo della risposta nel caso di errore.

```
#%RAML 1.0
title: Baeldung Foo REST Services API
version: v1
baseUri: http://rest-api.baeldung.com/api/{version}
mediaType: application/json
types:
  Foo:   !include types/Foo.raml
  Error: !include types/Error.raml

  newtype:
    type: any
/foos:
  get:
    description: List all Foos matching query criteria, if provided;
                 otherwise list all Foos
    queryParameters:
      name?: string
      ownerName?: string
    responses:
      200:
        body:
```

```
        application/json:
          type: Foo[]
          example: !include examples/Foos.json

  /{id}:
    get:
      description: Get a Foo by id
      responses:
        200:
          body:
            application/json:
              type: Foo

              example: !include examples/Foo.json
        404:
          body:
            application/json:
              type: Error
              example: !include examples/Error.json
    patch:
      description: Update a Foo by id
      body:
        application/json:
          type: Foo
          example: !include examples/Foo.json
      responses:
        200:
          body:
            application/json:
              type: Foo
              example: !include examples/Foo.json
        404:
          body:
            application/json:
              type: Error
              example: !include examples/Error.json
```

Fonte: https://www.baeldung.com/raml-restful-api-modeling-language-tutorial

1.4 I CONNETTORI

L' innovazione da parte dell'*environment* Mulesoft è l'uso dei connettori, chiamati in gergo *scopes* o *wrappers*, vale a dire blocchi in grado di racchiudere sia una singola operazione che una intera API.

Un'applicazione viene realizzata attraverso i flow (cioè "flussi") che contengono al loro interno una o più catene di scopes ma anche eventuali connessioni di richiamo ad altri flow definiti flow references; l'individuazione e l'inserimento dei connettori avviene tramite il panello della Mule Palette.

L'*environment* permette di visualizzare la trascrizione automatica in XML dell'intero file; essa rende possibile operare delle modifiche attraverso la scrittura di codice che si ripercuotono direttamente sui flow fisici con esito analogo all'uso del *drag and drop* utilizzato per posizionare gli scopes all'interno del *canva*.

Di seguito è riportato un esempio al fine di trattare quali sono alcuni dei connettori fondamentali e il loro utilizzo:

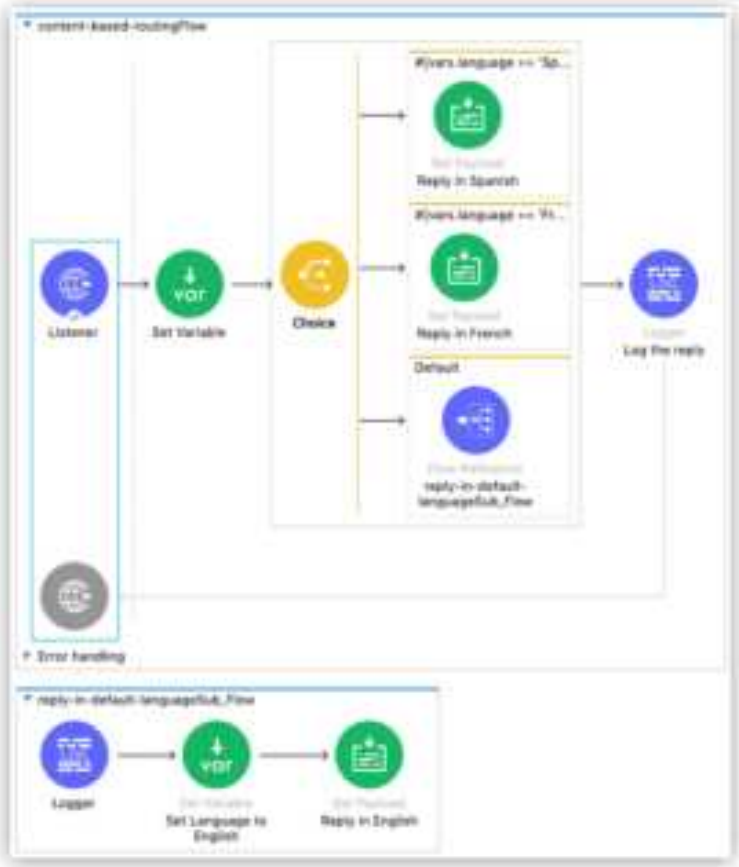


Figura 5: Esempio di un flow completo

Fonte: https://docs.mulesoft.com/general/

- Il Listener è un connettore che permette di definire le impostazioni del server HTTP e, nel momento in cui arriva una richiesta, provoca l'esecuzione del programma. Nelle sue impostazioni, è possibile indicare quali sono i metodi che accetta e il path da utilizzare per interagirvi. Quando il flow termina, viene restituito un HTTP response per indicare se l'esecuzione è avvenuta con successo o meno.
  Inoltre, è possibile implementare una maggiore sicurezza nella trasmissione dei dati utilizzando il protocollo HTTPS, il quale si basa su una comunicazione HTTP effettuata all'interno di una connessione criptata tramite crittografia

asimmetrica. A tal proposito, è possibile configurare opportunamente lo scope inizializzando gli appositi campi presenti nelle sue impostazioni.

Figura 6: Pannello di configurazione dello scope Listener

- Lo scope Set Variable viene utilizzato per inizializzare una o più variabili o per modificarne il contenuto. Esso è utilizzato nel caso di assegnamenti semplici mentre per operazioni più complesse si ricorre al Transform Message, un connettore che utilizza il linguaggio Dataweave (basato su Java) per mappare i campi di un corpo di dati o per convertirlo in diversi formati come, ad esempio, JSON o CSV.

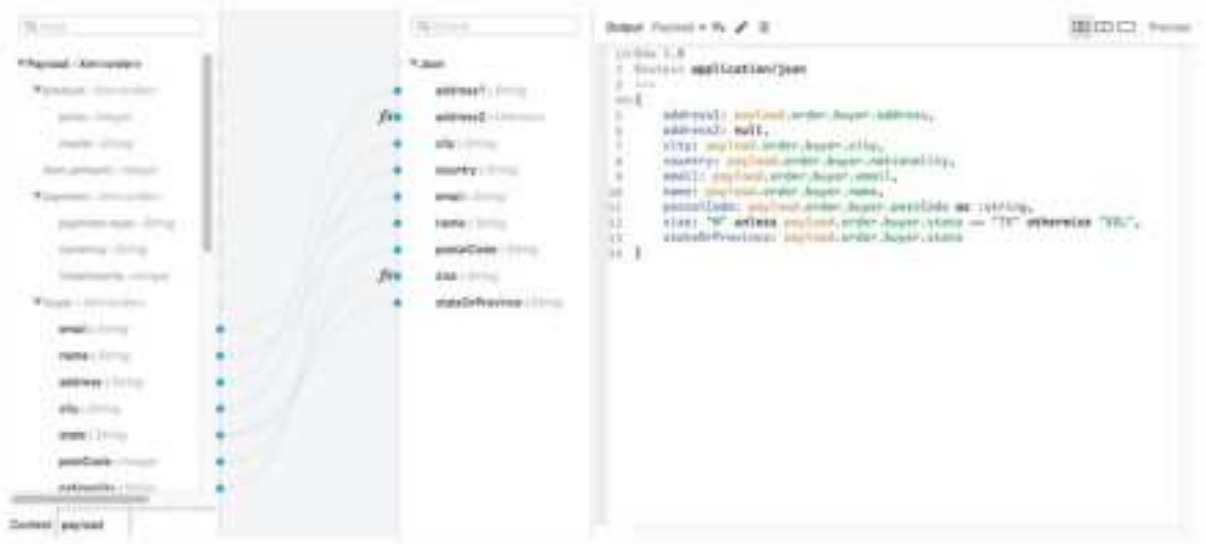


Figura 7: Pannello di configurazione dello scope Transform Message

Figura 8: Esempio di flow che utilizza lo scope Transform Message

- Lo scope Choice è l'equivalente del blocco *if-else* dei comuni linguaggi di programmazione. Si basa su un meccanismo di *routing* che verifica quale dei branch eseguire in base alla condizione di ingresso e, nel caso in cui nessuna di esse venga verificata, prosegue lungo il ramo di default.

- Il Set Payload ha un funzionamento simile al Set Variable precedentemente mostrato ma con lo scopo specificatamente di modificare il contenuto del payload. Quest'ultimo presenta comportamenti assimilabili a quelli di una variabile generica ma prevede un uso privilegiato in quanto utilizzato come contenitore automatico dei dati da parte di diverse operazioni come, ad esempio, per la risposta ricevuta a seguito di una connessione HTTP.

- Il Flow Reference implementa la riusabilità del codice nonché la modularità, permettendo sia di richiamare dei flow precedentemente sviluppati sia di ripartire le fasi di una operazione in sezioni diverse, garantendo un maggiore incapsulamento.

- Il Logger ha funzionalità meramente di supporto allo sviluppatore che potrà così verificare la corretta esecuzione di parti di codice attraverso la visualizzazione di messaggi su terminale.

- La libreria dei Database rende possibile inserire connettori in grado di accedere a diverse tipologie di basi di dati. Le operazioni permesse vanno dalle più basilari come, ad esempio, il prelievo o l'inserimento dei dati, alle più complesse come l'iterazione automatica di una eliminazione basata su parametri di binding.

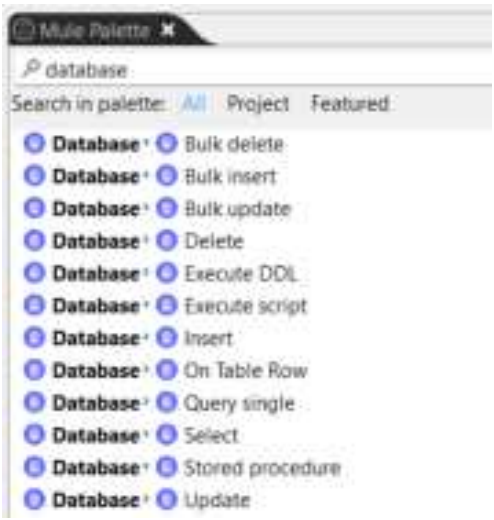


Figura 9: Connettori disponibili dalla ricerca con la parola chiave "database"

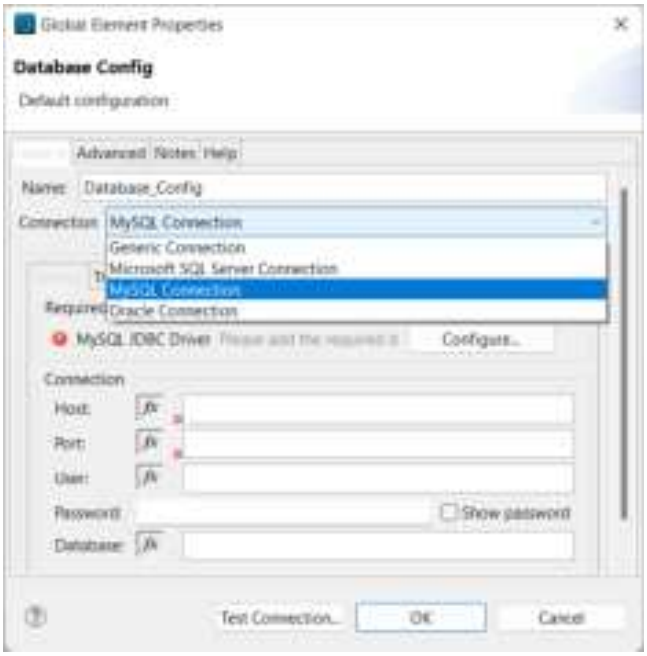


Figura 10: Pannello di configurazione della connessione ad un database

All'interno delle specifiche degli scope di tipo Database è presente un campo SQL Query Text dove è possibile inserire la query che verrà profilata al server specificato nella configurazione della connessione.

1.5 PROCESSO DI SVILUPPO E PUBBLICAZIONE

Il ciclo di sviluppo e delivery di una API segue un approccio flessibile ma guidato da alcune best practises.

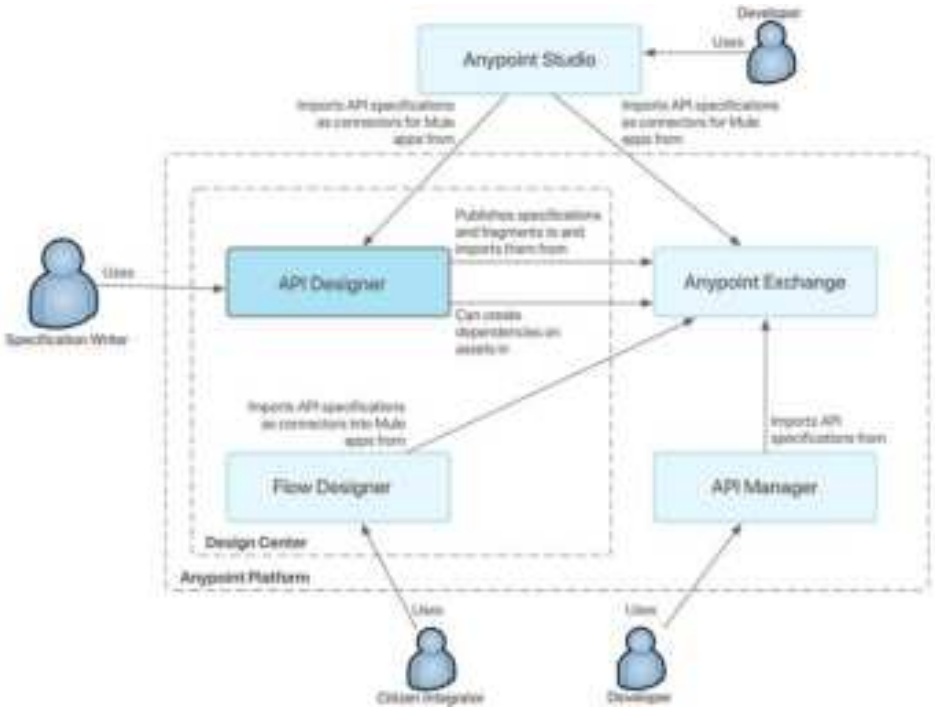


Figura 11: Schema di ciclo di sviluppo e delivery di una API

Fonte: https://docs.mulesoft.com/design-center/design-create-publish-api-specs

La realizzazione di una API inizia dal Design Center dove viene descritta la specifica. Una volta ultimata, essa viene pubblicata sull'Exchange Market, dove può essere resa disponibile a tutti gli utenti o ristretta ad un solo uso interno; ciò permette di operare una fase di analisi e testing in cui può essere coinvolto il cliente affinché si verifichi che tutte le richieste siano state previste correttamente prima di realizzare una implementazione completa. Dopo la prima fase di feedback, la specifica può essere importata all'interno di un nuovo progetto, il quale può essere creato attraverso l'IDE Anypoint Studio in modalità offline o nell'ambiente Flow Design accessibile online tramite browser.

Una volta concluso lo sviluppo dell'applicazione, è possibile eseguire il codice in locale, il che permetterà l'interazione con l'API fino a quando rimarrà in esecuzione sulla macchina. Nel caso della presenza di più layer di API, è necessario che siano tutte attive contemporaneamente ma ciascuna su una porta differente; in questo caso, l'URL

base di comunicazione sarà http://localhost seguito dal numero di porta sulla quale gira l'API più esposta e dall'eventuale path definito per l'accesso alle risorse.

Alternativamente, è possibile pubblicare le proprie applicazioni all'interno di CloudHub, una *integration platform as a service*, abbreviata in iPaas, la quale rappresenta un ambiente di esecuzione per l'interfaccia [10]. La sua peculiarità è di fornire un'architettura scalabile in quanto permette di aumentare le risorse dedicate alla gestione del servizio in base alla necessità attraverso i cosiddetti Workers Cloud.

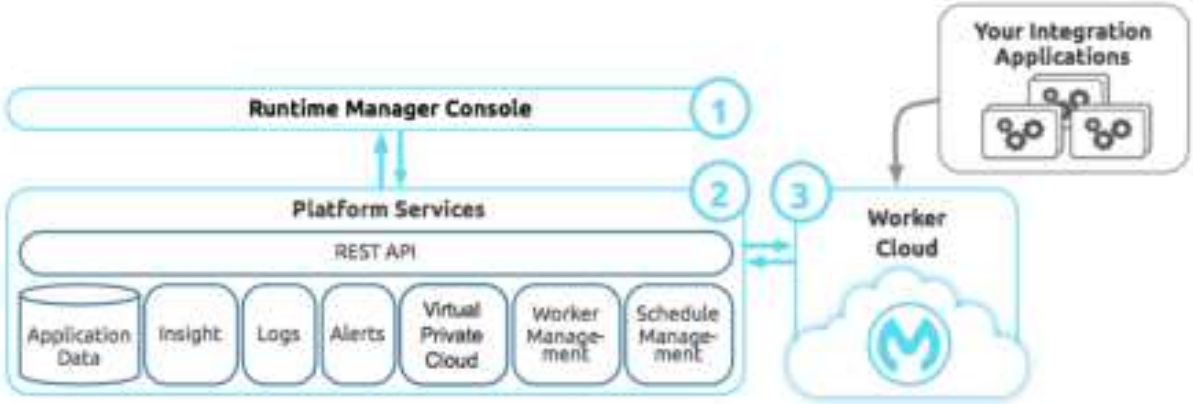


Figura 12: Schema di relazione fra i tools di deployment

Fonte: https://dzone.com/articles/introduction-to-cloudhub-with-mulesoft

I Workers Cloud sono istanze dedicate del motore di runtime che eseguono le applicazioni di integrazione. Ogni lavoratore ha una specifica quantità di capacità, selezionabile in fase di configurazione, che utilizza per elaborare i dati ricevuti. Inoltre, ogni worker viene isolato e monitorato in maniera separata nonché eseguito in un ambito cloud di lavoro geograficamente specifico come, ad esempio, negli Stati Uniti, nell'UE o nell'area Asia-Pacifico. Ne consegue che l'applicazione può essere scalata orizzontalmente aggiungendo più workers e gestendo code persistenti per distribuire il carico di lavoro, le quali sono in grado anche di suddividere un file di grandi dimensioni per processarlo in parallelo in modo da garantire un'alta disponibilità del sistema.

I Platform Services coordinano la distribuzione delle applicazioni, monitorano le integrazioni, forniscono dati di analisi, archiviano i dati delle applicazioni ed eseguono lavori pianificati.

La Runtime Manager Console, invece, riporta informazioni utili di monitoraggio e funge da dashboard completa per la gestione sia a livello di applicazione sia a livello di account. Inoltre, attraverso questa stessa console è possibile eseguire il deployment su CloudHub.

Figura 13: Esempio di Runtime Manager Console

Fonte: https://www.mulesoft.com/platform/saas/cloudhub-ipaas-cloud-based-integration

Capitolo 2: Un progetto concreto

Nell'ambito di questo capitolo, viene illustrato un progetto concreto realizzato tramite l'ambiente Mulesoft allo scopo di mostrare le potenzialità dei tools precedentemente descritti. L'opportunità di partecipare al suo sviluppo in qualità di developer si è presentata nell'ambito di un Erasmus Traineeship svoltosi presso l'azienda Cap4Lab con sede a Lussemburgo. L'esperienza ha coperto un arco temporale di poco più di due mesi, tra cui tre settimane circa di formazione, una settimana di introduzione al progetto e le successive sei settimane dedicate allo sviluppo del codice.

2.1 INTRODUZIONE AL PROGETTO

Il progetto è stato commissionato dall'azienda Contentful, la quale offre un servizio basato su API per la gestione, creazione e distribuzione di contenuti digitali che si adattano in modo flessibile a qualsiasi piattaforma (web, mobile, smart watches, TV digitale, Virtual Reality ed Internet of Things).

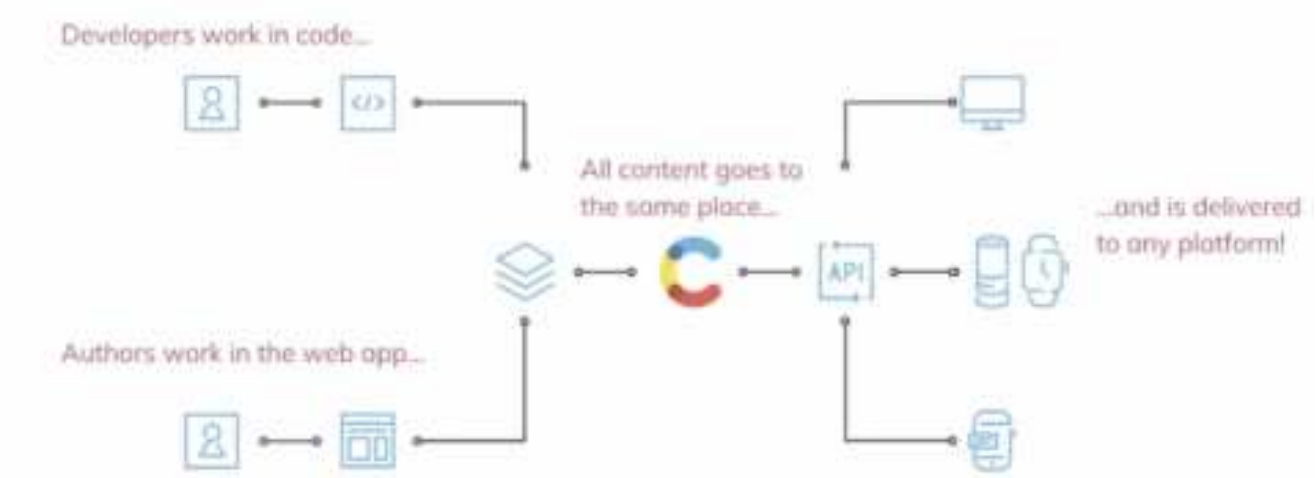


Figura 14: Schema di funzionamento della piattaforma Contentful

Fonte: https://www.contentful.com/developers/videos/quick-start-tutorials/

In particolare, il prodotto richiesto doveva essere in grado di interagire con gli account dei clienti (dove sono contenute le informazioni di fatturazione) per creare, modificare o eliminare i dati registrati e le informazioni di pagamento. Il suo funzionamento, per come previsto nella fase iniziale, non ha necessitato di alcuna API di livello experience.

Dal punto di vista organizzativo, sono state adoperate delle scelte che potessero permettere una maggiore sicurezza nonché chiarezza nello sviluppo, tra le quali:

- La definizione di una gerarchia di autorizzazioni per l'accesso alle risorse attraverso l'uso di credenziali. In particolare, alcuni comportamenti sono stati gestiti e rilasciati solo da parte del committente in modo che ne fosse completamente amministratore.
- È stata regolarmente tenuta una serie di incontri in modo da indirizzare le scelte progettuali rispettando il feedback del client. In aggiunta a ciò, esigenze impellenti e/o nuove questioni sorte sono state discusse in meeting organizzati ad hoc.
- Per questioni di sicurezza e portabilità dello sviluppo, è stata utilizzata una Virtual Machine su Amazon Web Service (AWS) generata e gestita dal team tecnico di Contentful. Ciò ha permesso all'azienda di visionare le risorse sviluppate nel tempo e di avere una maggiore sicurezza dettata dal salvataggio del lavoro in Cloud senza la necessità di realizzare un backup locale. Inoltre, l'utilizzo di un ambiente virtuale offre, in generale, una maggiore flessibilità lavorativa in quanto il progetto può essere visionato e modificato su diversi dispositivi senza dover importare l'intero codice.
- Il processo di realizzazione della API è stato suddiviso in tre fasi:
    - Development - fase di sviluppo del codice e di testing interno che, nelle sue fasi finali, è stato ampliato ai responsabili del progetto lato committente.
    - Staging - fase di prima vera esternalizzazione con testing aperto solo ad una cerchia ristretta di utenti.
    - Production - fase di resa pubblica dell'applicazione con estensione del testing.

  Per i primi due stadi, si è ricorso all'uso di un sandbox, espressione usata per indicare un'area di testing dove gli sviluppatori provano nuovi programmi ancora in fase di sviluppo prima del loro lancio definitivo. Ciò ha permesso di interagire con un ambiente di backend in grado di funzionare con gli stessi meccanismi del vero database ma contenete degli account di prova allo scopo di non intaccare la banca dati reale.

- Secondo la convenzione generalmente accettata, sono indicati con SAPI le interfacce di livello System e con PAPI di livello Process.

Nel rispetto dell'approccio API-led, all'interno del progetto è prevista una system API che si occupa dello scambio di dati con il lato back-end e di una process API che fornisce l'accesso al client. Il servizio di raccolta dei dati scelto dal committente è offerto da Zuora, azienda che fornisce sistemi di gestione degli abbonamenti, da cui la denominazione Zuora SAPI per l'interfaccia di tipo system.

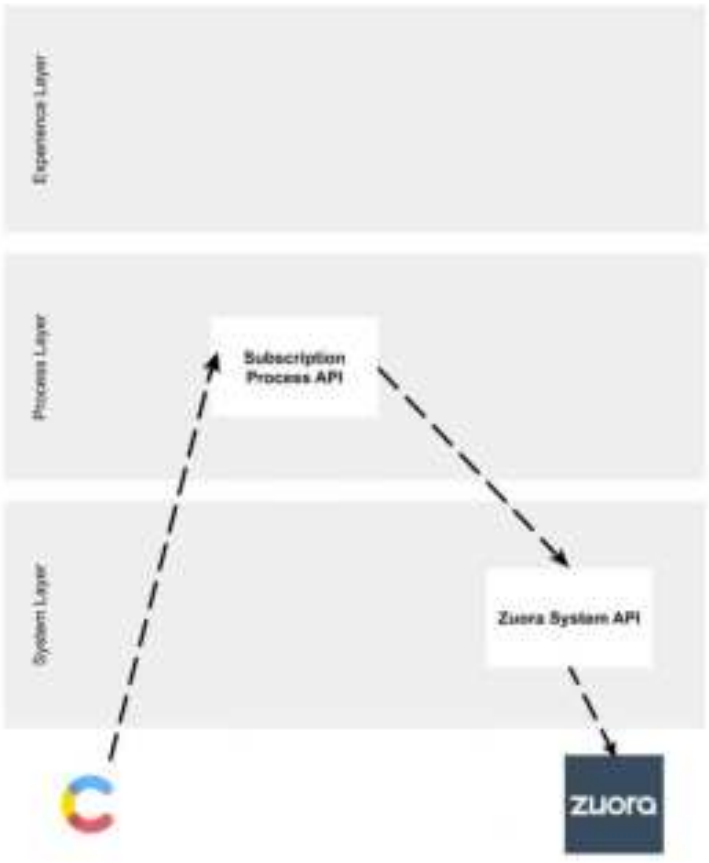


Figura 15: Schema descrittivo dell'interazione fra la system API e la process API

2.2 SYSTEM API

Nel rispetto dei principi di modularità, l'implementazione di una API prevede che venga realizzata una gerarchia di file diversi che incapsulino opportunamente le diverse funzionalità dell'interfaccia. Secondo la convenzione scelta nell'ambito del progetto, tra questi si hanno:

- Un primo file che gestisce la ricezione della richiesta e solleva eventuali errori causati da eventi come l'uso di metodi non permessi o l'invio di un payload non conforme alla specifica RAML.

- Un secondo file che contiene i flow implementativi delle operazioni previste per ciascuna richiesta.
- Un terzo file che contiene i comportamenti che l'interfaccia eseguirà nel caso di situazioni di errore.
- Un quarto file che raccoglie le definizioni delle configurazioni di connessione dell'interfaccia.

In particolare, il file che contiene l'interfaccia di ricezione della richiesta richiamerà il file di implementazione delle operazioni attraverso l'uso dello scope Flow Reference visto nel capitolo 1.4.

Ad ognuna delle tre operazioni previste di POST, PATCH e GET è associato un flow che viene eseguito alla ricezione di una richiesta riportante lo specifico metodo.

Figura 16: Flows contenenti il codice a blocchi che descrive l'interfaccia della system API

La comunicazione col sistema di backend all'interno del file di implementazione avviene in maniera diretta attraverso una serie di connettori sviluppati dall'azienda Zuora [11] e resi accessibili attarverso la pubblicazione all'interno dell'Exchange Market [12].

Figura 17: Flows contenenti il codice a blocchi che descrive l'implementazione delle operazioni della system API

Figura 18: Pannello di configurazione della connessione al sistema di backend Zuora

Come visto nel capitolo 1.4, uno dei ruoli fondamentali è svolto dal Transform Message, il quale è l'elemento che regola in che modo i dati vengono restituiti o inviati. Di conseguenza, eventuali decisioni future sulla formattazione o sui campi da fornire si traducono solamente in modifiche del codice Dataweave al suo interno.

Nel caso dell'operazione di GET, il flow si occupa di richiedere le informazioni su uno specifico account il cui ID è contenuto all'interno dell'URL. Il Transform Message che si occupa di gestire la formattazione dei dati, una volta ottenuti dal database, presenta un parametro *skipNullOn* inizializzato su "everywhere", il quale prevede che venga saltato (e dunque non restituito) qualsiasi campo della mappatura che risulti essere nullo. Il codice della trasformazione appena descritta è di seguito riportato.

```
%dw 2.0
//null fields will be skipped
output application/json skipNullOn = "everywhere"
---
{
	paymentMethodId: payload.hpmCreditCardPaymentMethodId,
	name: payload.basicInfo.name,
	sys: {
		"type": "BillingAccount",
		id: payload.contentful_account_id__c
	},
	billingAddress: {
		address1: payload.billToContact.address1,
		address2: payload.billToContact.address2,
		city: payload.billToContact.city,
		state: payload.billToContact.state,
		country: payload.billToContact.country,
		zipCode: payload.billToContact.zipCode
	},
	billToContact: {
		firstName: payload.billToContact.firstName,
		lastName: payload.billToContact.lastName,
		workEmail: payload.billToContact.workEmail
	},
	vat: payload.taxInfo.VATId
}
```

La specifica RAML contiene anche file di esempio, i quali risultano essere utili per gli sviluppatori in quanto mostrano in maniera immediata e visiva la formattazione degli oggetti all'interno dei tools Mulesoft come l'Exchange Market. Di seguito è riportato l'esempio di un account afferente all'utente Mario Rossi.

```
{
  "name": "Mario Rossi",
  "sys": {
    "type": "BillingAccount",
    "id": "2c92c0fa72a83f630172b85e1a326cc9"
  },
  "billingAddress": {
    "address1": "Via Alessandro Manzoni, 5",
    "address2": "Via Giuseppe Verdi, 7",
    "city": "Milano",
    "state": "Italia",
    "country": "Italia",
    "zipCode": "20019"
  },
  "billToContact": {
    "firstName": "Mario",
    "lastName": "Rossi",
    "workEmail": "mariorossi@gmail.com"
  },
  "paymentMethodId": "6OASoW4gPgF5OuD1y5MPze",
  "vat": " 999999999"
}
```


L'operazione di POST (indicata all'interno del progetto anche come *create*) prevede che avvenga necessariamente una trasformazione del payload prima del suo invio in quanto il suo contenuto deve rispettare le specifiche definitive dal connettore di creazione account fornito dal servizio di backend Zuora. La sua formattazione è, dunque, suscettibile alle modifiche apportate al funzionamento dello scope stesso e richiede la verifica costante di nuovi aggiornamenti, i quali possono introdurre variazioni riguardo le variabili da inviare in ciascuna richiesta o le modalità di comunicazione. Viene di seguito riportata la mappatura dei campi effettuata.

```
%dw 2.0
output application/json

//calculates the company entity (inc or gmbh)
fun setBillingEntity(aString) = aString match {
	case "Germany" -> "gmbh"
	case "United States" -> "inc"
	else -> ""
}
---
{
	autoPay: true,
	name: payload.name,
	notes: "Self-service",
	billCycleDay: null,
	currency: "USD",
	vat_id__c: payload.vat,
	billing_entity__c: setBillingEntity(payload.billingAddress.country),
	invoiceDeliveryPrefsPrint: false,
```

```
        bill_run_frequency_in_months__c: "1",
        organization_key__c: payload.attributes.headers.'X-Contentful-Organization',
        contentful_account_id__c: payload.sys.id,
        hpmCreditCardPaymentMethodId: payload.paymentMethodId,
        billToContact: {
                country: payload.billingAddress.country,
                lastName: payload.billToContact.lastName,
                zipCode: payload.billingAddress.zipCode,
                address2: payload.billingAddress.address2,
                city: payload.billingAddress.city,
                workEmail: payload.billToContact.workEmail,
                address1: payload.billingAddress.address1,
                county: payload.billingAddress.country,
                firstName: payload.billToContact.firstName,
                state: payload.billingAddress.state
        },
        invoiceDeliveryPrefsEmail: true,
        batch: if (isEmpty(payload.vat)) ("Batch1") else ("Batch7"),
        paymentTerm: "Due Upon Receipt",
        (if ( setBillingEntity(payload.billingAddress.country) == "gmbh" ) {
                invoiceTemplateId: p('zuora.ids.gmbhInvoiceTemplateId'),
                Subsidiary__NS: "Contentful global Inc. : Parent : Contentful GmbH",
                taxInfo: {
                        companyCode: "CONTENTFULGMBH",
                        VATId: payload.vat,
                        exemptCertificateId: payload.vat
                },
                creditMemoTemplateId: p('zuora.ids.gmbhCreditTemplateId'),
                debitMemoTemplateId: p('zuora.ids.gmbhDebitTemplateId'),
                paymentGateway: "Stripe Gateway 3DSv2"
        }

        else if ( setBillingEntity(payload.billingAddress.country) == "inc" ) {
                invoiceTemplateId: p('zuora.ids.incInvoiceTemplateId'),
                Subsidiary__NS: "Contentful global Inc. : Parent : Contentful Inc.",
                taxInfo: {
                        companyCode: "CONTENTFULINC",
                        VATId: payload.vat,
                        exemptCertificateId: payload.vat
                },
                creditMemoTemplateId: p('zuora.ids.incCreditTemplateId'),
                debitMemoTemplateId: p('zuora.ids.incDebitTemplateId'),
                paymentGateway: "Stripe Gateway Inc 3DSv2"
        }

        else
        {
        })
}
```

In assenza di problematiche che sollevino eccezioni, l'identificativo dell'account generato dal sistema viene inserito all'interno del payload affinché le API di livello superiore possano facilmente prelevarlo ed eventualmente restituirlo; in caso opposto, alle stesse è demandata la generazione di eventuali messaggi di errore.

```
%dw 2.0
output application/json
---
{
	id: payload.accountId,
}
```

L'operazione di *updating* di un account (operata tramite il metodo PATCH) segue il funzionamento della creazione appena vista, con la differenza che la specifica del connettore utilizzato prevede che non tutti i campi associati debbano essere rispediti o aggiornati; perciò, viene utilizzato il campo *skipNullOn*, visto in precedenza, inizializzato su "everywhere" affinché avvenga la sola sovrascrittura dei campi che presentano dei valori e non lo svuotamento di tutte le variabili dell'account lasciate nulle all'interno del payload inviato.

L'*environment* permette, inoltre, la definizione dei comportamenti che l'interfaccia deve adottare in caso di errore. A tale scopo, esistono due tipologie di blocco con il compito di generare esiti differenti:

- On Error Continue, il quale termina l'esecuzione e non trasferisce la situazione di errore.
- On Error Propagate, il quale interrompe l'esecuzione e propaga l'errore fino alla restituzione dell'output. Si tratta di una situazione caratterizzata da una gravità maggiore che necessita dell'interruzione del programma.

In entrambe le tipologie appena descritte, per implementare la risposta da eseguire è possibile inserire i connettori generici visti nel capitolo 1.4, tra i quali la definizione delle variabili e il richiamo di flow.

All'interno dell'unico blocco di gestione degli errori, sono presenti vari flow, ciascuno dei quali viene richiamato automaticamente nel caso in cui si sollevi l'errore per cui è stato configurato. Per limitare la copia del codice, ciascun flow è responsabile di riempire in maniera specifica alcuni campi globali che saranno utilizzati nel flow finale per customizzare l'output. In particolare, ogni blocco è realizzato tramite un flow di tipo On Error Continue in quanto permette di gestire opportunamente l'applicazione restituendo un messaggio generato ad hoc ma senza interromperne bruscamente l'esecuzione.

Figura 20: Flows contenenti il codice a blocchi che descrive il comportamento in caso di errore

Figura 19: Flow contente il codice a blocchi che definisce il messaggio da restituire in caso di errore

Di seguito è riportato il codice utilizzato nel Transform Message per la definizione del messaggio di errore.

```
%dw 2.0
output application/json
---
error: {
       error_code: vars.errorCode,
       error_datetime: now() as String { format: "yyyy-MM-dd'T'HH:mm:ssZ" },
       (error_message: vars.errorMessage) if(vars.errorMessage != null),
       (error_description: vars.errorDescription) if(vars.errorDescription != null)
}
```

2.3 PROCESS API

La process API è responsabile della standardizzazione, secondo le richieste specifiche del cliente, dei dati ottenuti dal sistema di back-end. L'esistenza di un layer intermedio permette di assecondare i piani futuri del committente, intenzionato a sostituire i propri servizi di billing forniti da Zuora, senza dover procedere alla realizzazione di un nuovo sistema completo. In quanto gestore degli account di fatturazione, l'interfaccia è chiamata Billing Process API.

In particolare, la struttura segue i principi dell'API led-connectivity, come visto nel capitolo primo e nel paragrafo precedente con la system API, per cui sono previsti, tra gli altri, i seguenti file:

- Un file che gestisce ricezione delle richieste.
- Un file che implementa le operazioni relative ai metodi permessi.
- Un file che ingloba i comportamenti in caso di errore.
- Un file che raccoglie le configurazioni di connessione.

Per il client è essenziale la conoscenza della specifica RAML della process API per poter utilizzare gli end point di accesso e rispettare la formattazione dei dati e delle richieste; la conoscenza della specifica del system API è, invece, omessa all'utente finale secondo il principio della black box ma fondamentale da parte dello sviluppatore, il quale dovrà ricorrervi per porre in comunicazione le due interfacce.

La chiamata alla system API viene gestita attraverso lo scope di richiesta http nonostante sia possibile generare e pubblicare un connettore nell'Exchange Market della system API, come citato nel primo capitolo. Infatti, diviene complesso dover verificare e aggiornare manualmente la versione utilizzata e, soprattutto nell'ambito di un updating lento nel tempo del codice, si rischierebbe di avere un prodotto potenzialmente non funzionante o poco sicuro. Dunque, dal punto di vista dello sviluppo di interfacce multi-layer, è una best practise ricorrere all'uso di connettori di richiesta in quanto permettono di fare sempre uso di una API nella sua versione più recente tramite l'utilizzo dell'URL di accesso.

Inoltre, prima di ciascuna elaborazione, sono presenti degli scope di logging i quali hanno lo scopo di fornire allo sviluppatore informazioni riguardo la corretta esecuzione del codice.

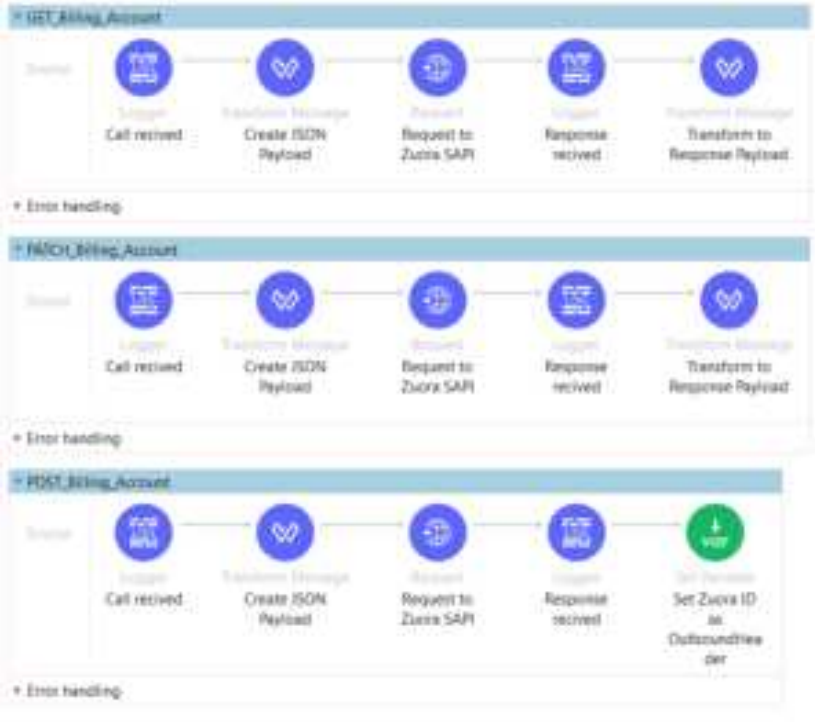


Figura 21: Flows contenenti il codice a blocchi che descrive l'implementazione della process API

Segue, a titolo di esempio, la specifica inerente all'operazione di POST, la quale regola i campi e i loro formati che, se violati, generano un errore.

```
/customer_accounts:
  /{customer-account-id}:
    uriParameters:
      customer-account-id:
        type: string
        description: Unique Contentful ID for a customer account
        example: "6OASoW4gPgF5OuD1y5MPzg"

    /billing_account:
      post:
        description: Creates a new billing account in Zuora
        is: [errors.codes, responses.201]
        headers:
          X-Contentful-Organization:
            displayName: Contentful Organization Key
            description: Contentful Organization ID for the created Account
            type: string
            required: false
            example: 6OASoW4gPgF5OuD1y5MPze
        body:
          application/json:
            type: !include datatypes/billing-account-creation.raml
            example: !include examples/billing-account-creation.raml
        responses:
          201:
            headers:
```

```
    X-Zuora-Account:
      displayName: Zuora Account
      description: Zuora Account ID
      type: string
      example: 2c92c0fa72a83f630172b85e1a326cc9
  body:
    application/json:
      example:
        message: Creation successful
```

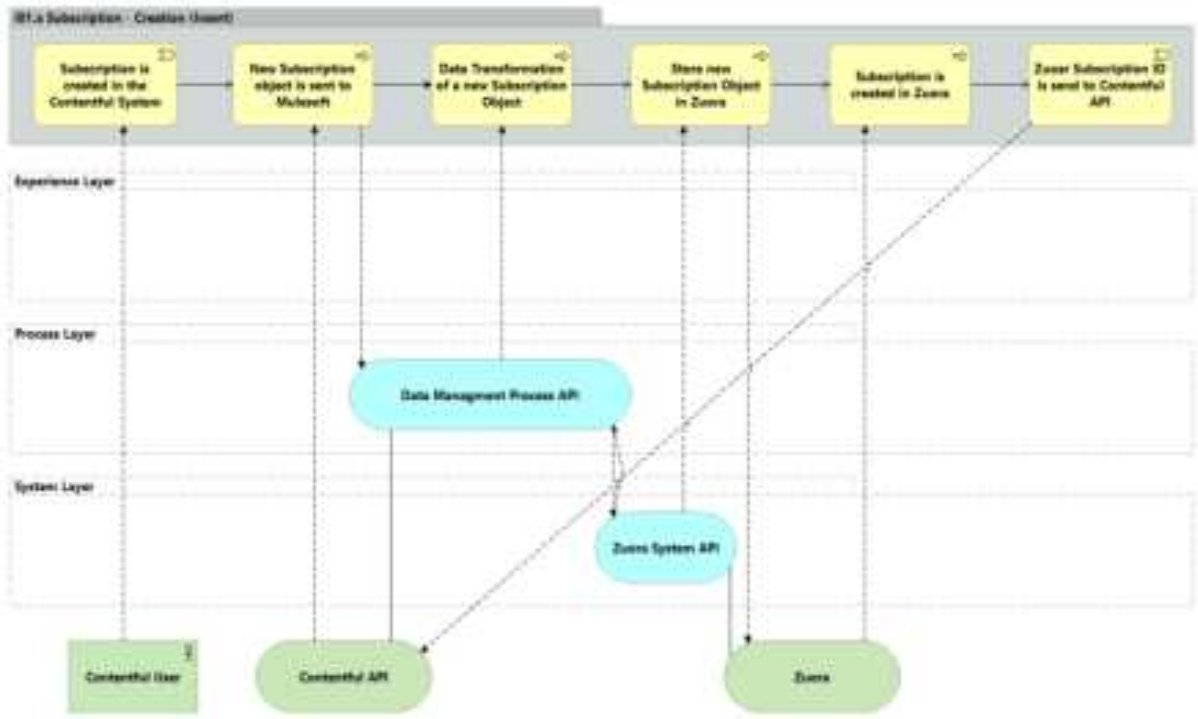


Figura 22: Diagramma che illustra le interazioni durante la creazione di un account

Nel caso dell'operazione di POST e di PATCH, l'utilizzo di un Transform Message regola la formattazione del payload inviato nella richiesta affinché vi sia corrispondenza con i campi previsti, in maniera del tutto similare a quanto descritto per la system API. Tuttavia, a differenza di quanto visto nel capitolo precedente, grazie all'incapsulamento e alla modularizzazione in due layer differenti, una eventuale modifica del tipo di database utilizzato non comporta modifiche all'interno delle trasformazioni della process API (a meno che non siano richiesti campi che attualmente non sono previsti) bensì solamente variazioni all'interno della system API che dovrà conformarsi al nuovo sistema utilizzato.

La formattazione dei dati da inserire nel payload segue la seguente specifica.

```
#%RAML 1.0 DataType
type: object
properties:
```

```
name: string
sys?:
  type: object
  properties:
    type: string
    id: string
billingAddress?:
  type: object
  properties:
    address1: string
    address2?: string
    city: string
    state: string
    country: string
    zipCode: string
billToContact?:
  type: object
  properties:
    firstName: string
    lastName: string
    workEmail?: string
paymentMethodId?: string
vat?: string
```

Nel caso dell'operazione di creazione, uno scopo di Set Variable è utilizzato per restituire un identificativo dell'account sottoforma di header.

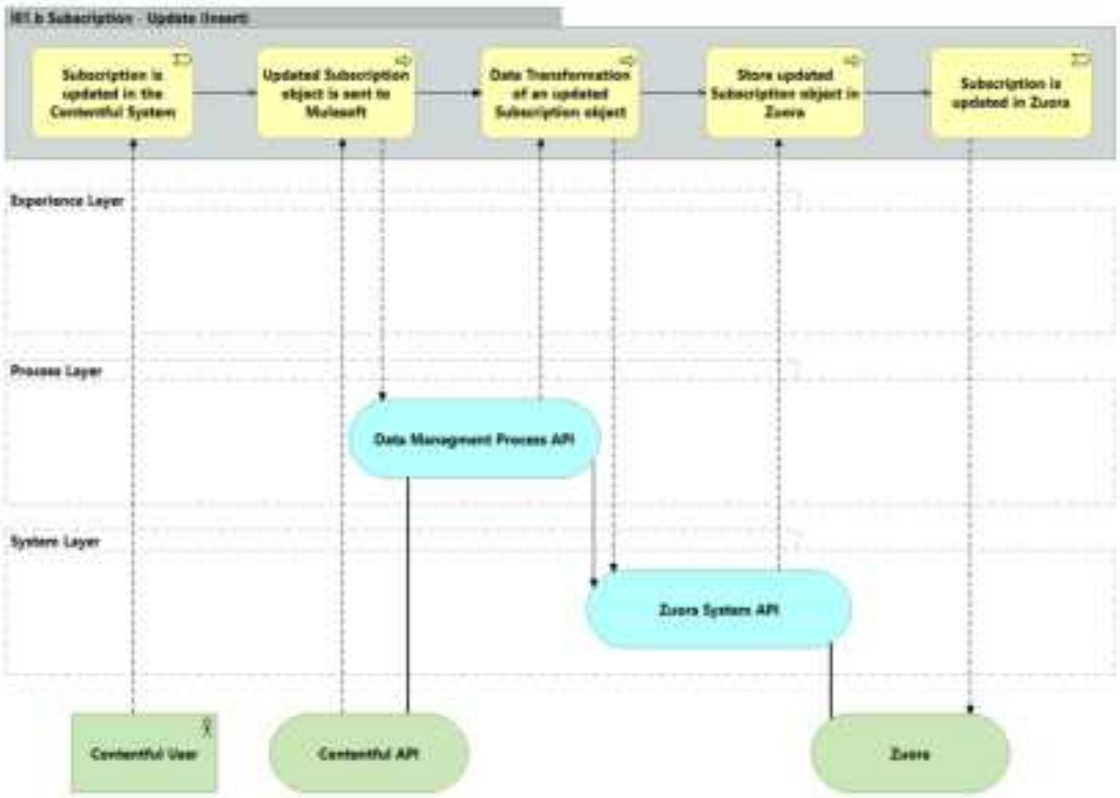


Figura 23: Diagramma che illustra le interazioni durante la modifica di un account

L'operazione di GET prevede che i codici identificativi, utilizzati per catalogare gli account, vengano estratti dagli header e raccolti all'interno del payload in modo sia da renderne più agevole l'accesso sia per effettuarne un salvataggio prima che la richiesta http al sistema di backend vada a sovrascriverne i valori.

```
%dw 2.0
output application/json
---
{
	organization_key__c: payload.attributes.headers.'X-Contentful-Organization'
default "",
	zuora_account_id__c: payload.attributes.headers.'X-Zuora-Account',
	contentful_account_id__c: vars.customerID,
}
```

Dopo la richiesta, i dati estrapolati vengono formattati seguendo la specifica RAML.

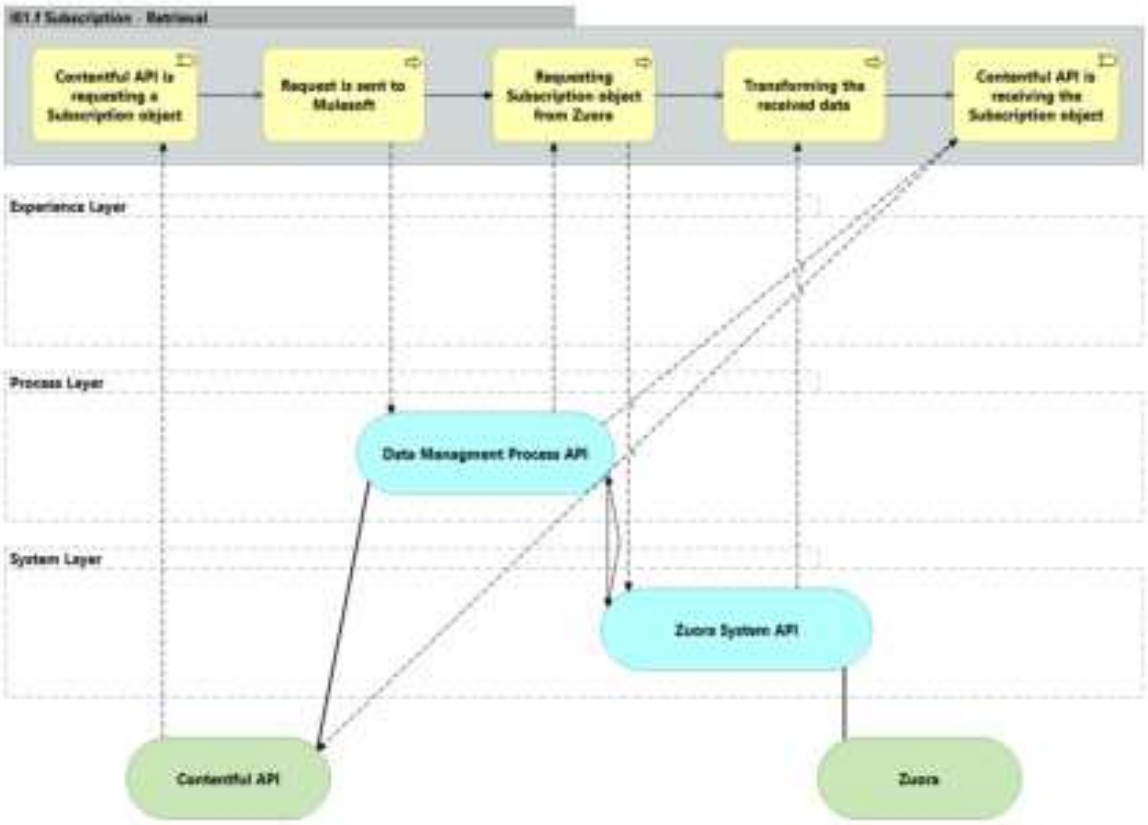


Figura 24: Diagramma che illustra le interazioni durante la richiesta di un account

2.4 API IN ESECUZIONE

La verifica del funzionamento previsto viene eseguita tramite tools ad hoc denominati *REST client.* Nell'ambito di questa tesi, viene utilizzato Postman, il quale permette di eseguire delle chiamate all'API specificando i parametri necessari a ciascuna operazione (come headers e payload) e visualizza, infine, l'esito della richiesta.

La fase di testing di seguito mostrata simula il comportamento di un cliente tipo, il quale avrà accesso solo alla PAPI (il livello più esposto nel caso in esame) e non alla SAPI, con hosting in locale dell'esecuzione delle due API.

L'intero progetto è eseguito sulla porta 8092 per cui l'indirizzo di connessione sarà https://localhost:8092. All'interno del path di ciascuna operazione è inserito un campo che contiene un codice univoco assegnato dall'azienda ad ogni nuovo account; similmente, come header è inserito un ulteriore campo con una funzionalità di identificazione simile.

Di seguito è riportata una richiesta di tipo GET per l'account con ID `6OASoW4gPgF5OuD1y5MPzg` (utilizzato internamente da parte dell'azienda Contentful) ed ID Zuora `2c92c0fa72a83f630172b85e1a326cc9` (identificativo appartenente al sistema backend).

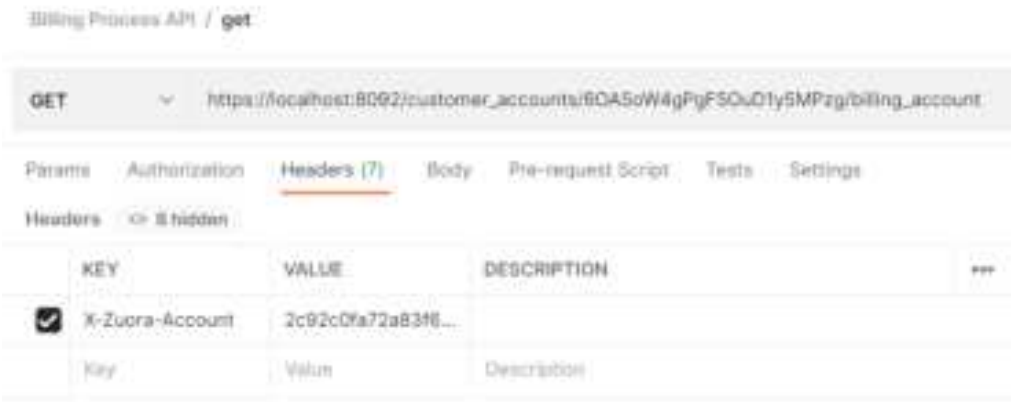


Figura 25: Esempio di richiesta di tipo GET eseguita tramite Postman

Rispettando la specifica, la chiamata di POST prevede l'inserimento dell'identificativo interno alla compagnia (denominato X-Contentful-Organization) sottoforma di header, oltre ai campi precedentemente discussi, e richiede che il corpo dell'account che si vuole creare sia inviato all'interno del payload. Nel caso in cui i campi o il loro formato non si confaccia alla specifica, verrà sollevato un errore.

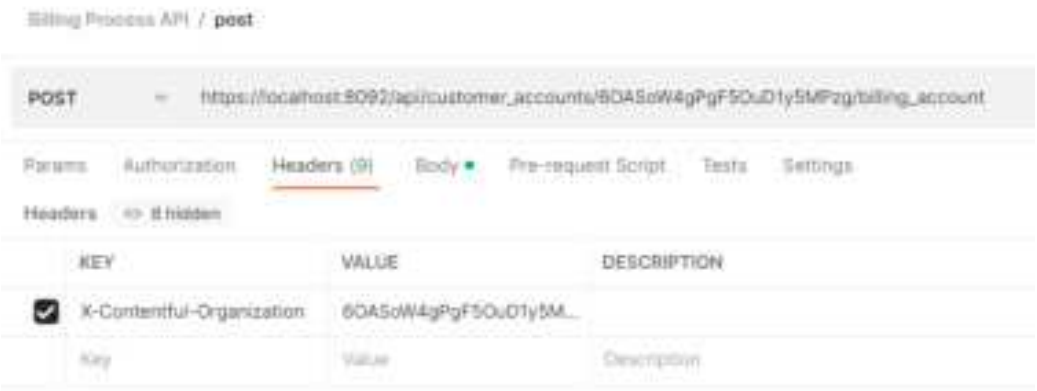


Figura 26: Esempio di richiesta di tipo POST eseguita tramite Postman

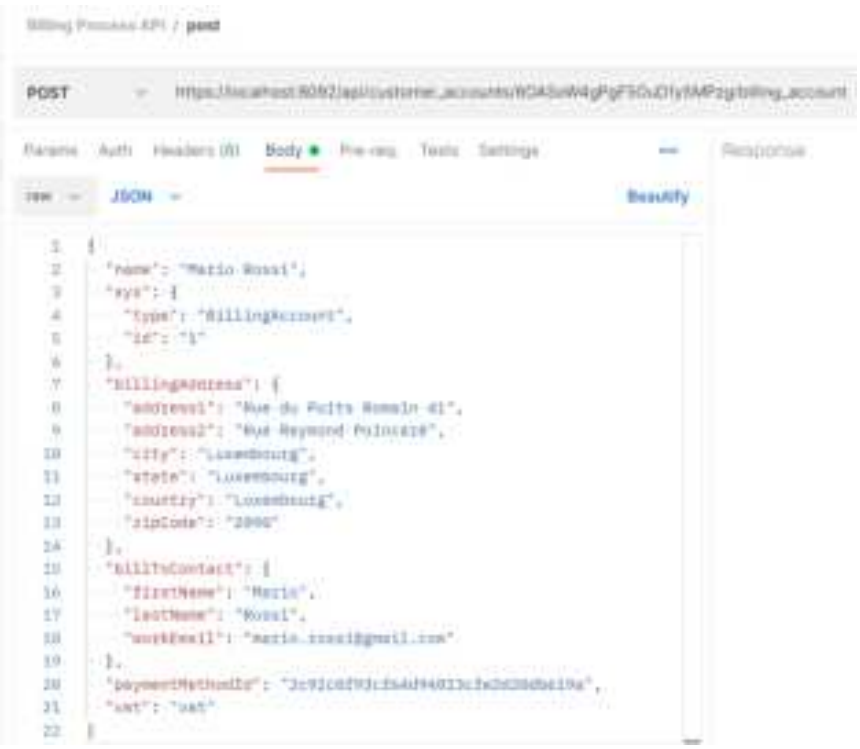


Figura 27: Esempio di payload inviato in una richiesta di tipo POST eseguita tramite Postman

L'operazione di PATCH prevede, in aggiunta, l'inserimento dell'identificativo del sistema backend Zuora sottoforma di header

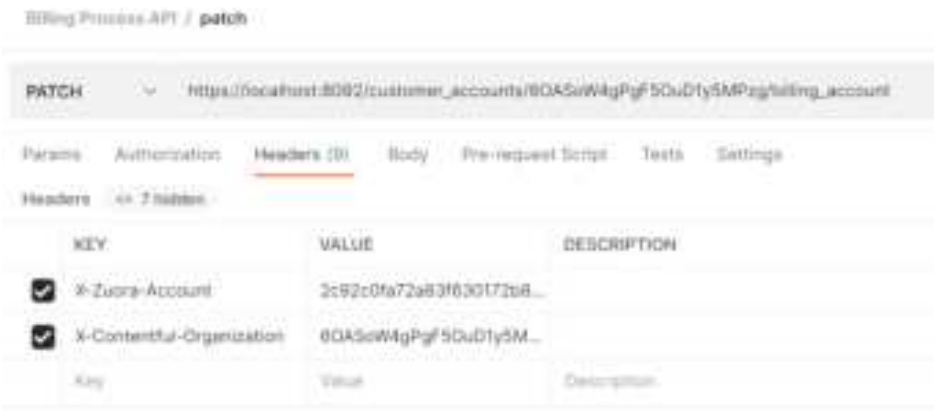


Figura 28: Esempio di richiesta di tipo PATCH eseguita tramite Postman

Ogni tipologia di risposta ottenuta è caratterizzata da un codice di esito denominato *HTTP Status Code* (ad es. 200, 405, 500); nel caso di esito positivo, verrà restituito un messaggio specifico o il corpo dell'oggetto richiesto (nel caso di una operazione GET).

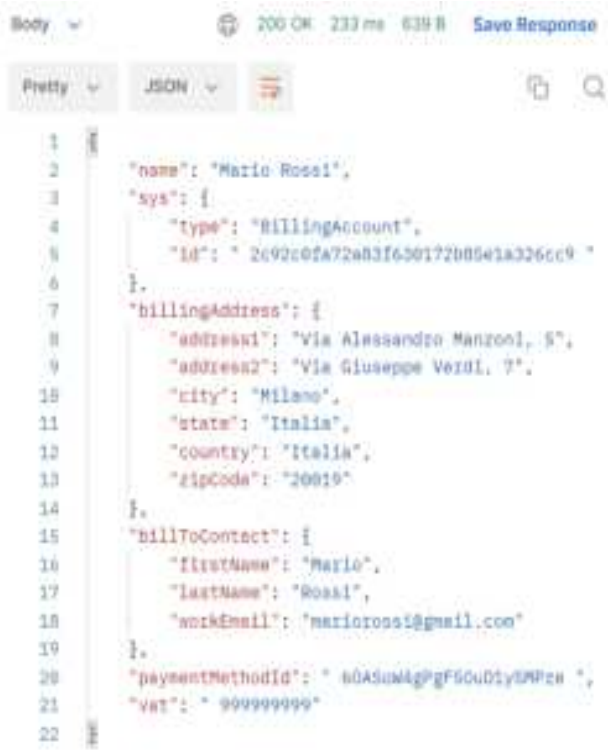


Figura 29: Payload ottenuto in risposta ad una richiesta di tipo GET eseguita tramite Postman

Figura 30: Messaggio ottenuto in risposta ad una richiesta di tipo POST eseguita tramite Postman

Figura 31: Messaggio ottenuto in risposta ad una richiesta di tipo PATCH eseguita tramite Postman

Nel caso che venga generato un errore, l'output conterrà il nome della problematica (ad es. "METHOD_NOT_ALLOWED" è sollevato nel momento in cui una chiamata usa un operatore non permesso), corredata da un messaggio specifico per l'eccezione.

Figura 33: Messaggio ottenuto a causa dell'uso di un metodo CRUD non previsto nella specifica

Figura 32: Messaggio ottenuto da un errore generico che non rientra nelle specificazioni effettuate

Capitolo 3: Le prospettive in ambito IoT

In questo capitolo sono affrontate le prospettive future di diffusione delle tecnologie legate all'Internet delle Cose e la necessità di approcci innovativi in grado di accompagnare il loro sviluppo.

3.1 IL CONCETTO DI INTERNET DELLE COSE

Con il termine Internet delle Cose (in inglese *Internet of Things*, da cui l'acronimo IoT), si fa riferimento all'estensione di internet al mondo degli oggetti e dei luoghi concreti, che acquisiscono una propria identità digitale in modo da poter comunicare con altri oggetti nella rete e poter fornire servizi agli utenti [13]. Tale concetto comincia a delinearsi formalmente negli ultimi decenni del XX secolo ma è nel corso degli anni '10 che avviene una enorme diffusione a livello pubblico, soprattutto grazie alla domotica (*Smart Home*), ai sistemi di intrattenimento nelle automobili e alla videosorveglianza, ma anche alla e-health e ad altri settori.

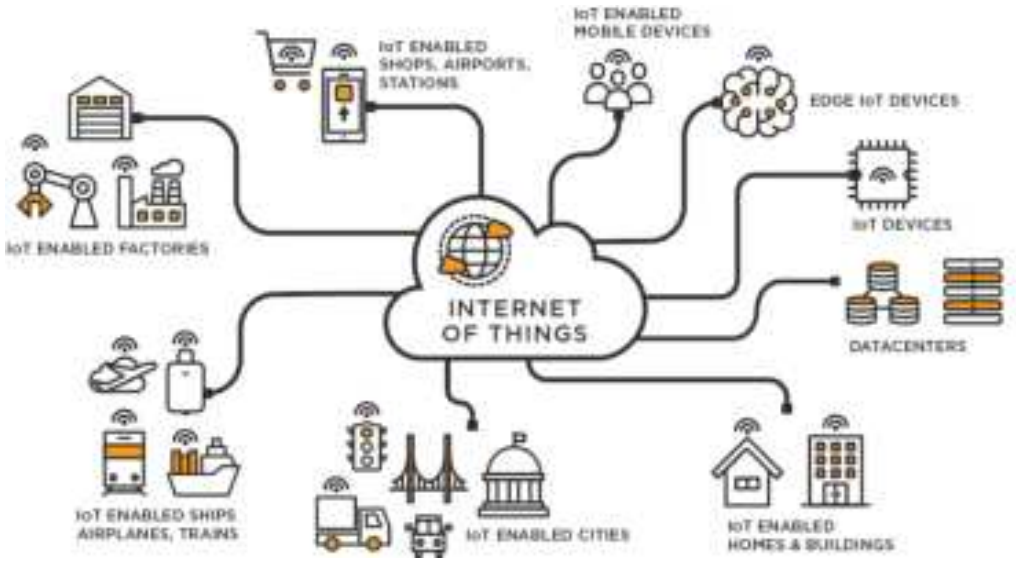


Figura 34: Diagramma dei devices rientranti nella definizione di Internet delle Cose

Gli oggetti connessi che sono alla base dell'Internet delle Cose si definiscono più propriamente *smart objects* e, tramite l'uso della rete, sono in grado di comunicare fra loro e di raccogliere dati. Quest'ultimi spesso rientrano, per quantità e complessità, nella definizione di Big Data con la quale si indica una mole di dati così estesa in termini di volume, velocità e varietà da richiedere tecnologie e metodi analitici specifici per l'estrazione di valore o conoscenza [14].

Gli oggetti intelligenti vengono spesso utilizzati per descrivere l'ambiente circostante, grazie alla possibilità di interagire con il mondo esterno. A tal proposito si parla di:

- sensing, nel caso misurino variabili di stato come la temperatura, la pressione o il livello di inquinamento.
- metering, nel caso misurino variabili di flusso come i consumi di energia elettrica, acqua o gas.

Inoltre, un oggetto può avere la capacità di elaborare dati in locale per operare semplici valutazioni o per selezionare quali informazioni trasmettere tra quelle raccolte e può interagire attivamente con l'*environment* che lo circonda compiendo azioni, comandate sia in autonomia che da remoto, che possono essere adottate a seguito di particolari valori soglia misurati in precedenza. In aggiunta, un oggetto smart è in grado di fornire informazioni importanti per garantire l'originalità e l'integralità dei prodotti, come ad esempio il suo stato di funzionamento o una richiesta di manutenzione così come la sua localizzazione e la tracciabilità della sua produzione.

In tempi recenti, il concetto di IoT è stato associato alla tecnologia della Blockchain [15] [16] , la quale è una struttura dati condivisa e "immutabile" definita come un registro digitale le cui voci sono raggruppate in "blocchi", concatenati in ordine cronologico, e la cui integrità è garantita dall'uso della crittografia; il suo contenuto, una volta scritto tramite un processo normato, non è più né modificabile né eliminabile a meno di non invalidare l'intero processo. La Blockchain può fungere da garante dell'identità dei diversi nodi della rete, ad esempio attraverso l'impiego di certificati o chiavi digitali, e da certificatore della provenienza e dell'integrità dei dati raccolti dagli oggetti connessi grazie all'apposizione di un timbro digitale e alla registrazione dell'istanza temporale, fornendo così una possibile risoluzione ad alcune delle problematiche legate alla raccolta dei dati in ambito IoT.

Tuttavia, senza una opportuna manipolazione dei dati non sarebbe possibile estrapolare informazioni rilevanti, soprattutto nel caso dei Big Data dove la quantità non permette una facile e immediata elaborazione [17][18]. In questo caso, l'Intelligenza Artificiale gioca un ruolo importante in quanto fornisce algoritmi in grado di valutare opportunamente i dati con lo scopo di integrare, estrapolare pattern e trarre possibili soluzioni alle problematiche presenti [19]. L'AI, insieme al Machine Learning ed altre tecniche di apprendimento automatico, permette anche di fornire un nuovo approccio all'integrazione fra l'uomo e l'ambiente circostante: si consideri, ad esempio, la gestione dei dispositivi smart presenti in casa dove un algoritmo, a partire dalle abitudini e dai gusti del consumatore, decide il momento in cui sia più opportuno attivare la lavatrice precaricata dall'utente o quando accendere l'impianto di riscaldamento [20]. Da ciò ne consegue che la commistione fra buona integrazione e

l'uso di algoritmi di intelligenza artificiale e machine learning può essere la chiave per estrapolare schemi complessi e correlazioni che, per bias o difficoltà, un essere umano non potrebbe individuare [21].

Figura 35: Esempio di sistema IoT e delle tecnologie coinvolte

Fonte: https://www.scnsoft.com/blog/iot-architecture-in-a-nutshell-and-how-it-works

3.2 L'USO DELLE API IN AMBITO IOT

Nonostante i tentativi di unificazione e standardizzazione della comunicazione, sono ancora molte le tecnologie profondamente diverse tra di loro che caratterizzano gli smart objects e i sistemi in cui interagiscono. Inoltre, la democratizzazione della programmazione necessita di meccanismi di sviluppo software intuitivi e che non richiedano necessariamente ai developer la conoscenza degli schemi di comunicazione sottostanti [22].

La realizzazione di API secondo il principio dell'API-led connectivity, come visto nel primo capitolo, permette di creare un livello di astrazione che incapsula comportamenti e tecnologie di back-end in modo da rendere maggiormente fruibili e intuitivi sia dati che servizi. In particolare, l'introduzione delle API può essere

utilizzata per gestire i flussi di dati che da un oggetto (spesso munito di sensori) vengono incanalati su di un server. Si può così operare una prima fase di filtraggio, elaborazione e/o formattazione, utile nel caso in cui si debbano conformare dati da diverse sorgenti per utilizzarli all'interno di un unico sistema. In maniera complementare, esiste il concetto di *fog computing* inteso come una tecnologia in cui i dati generati non vengono caricati direttamente in Cloud ma si delega parte del carico computazionale a dei nodi di frontiera (definiti *edge devices*) che fungono da intermediari tra il Cloud e i vari dispositivi IoT [23][24][25].

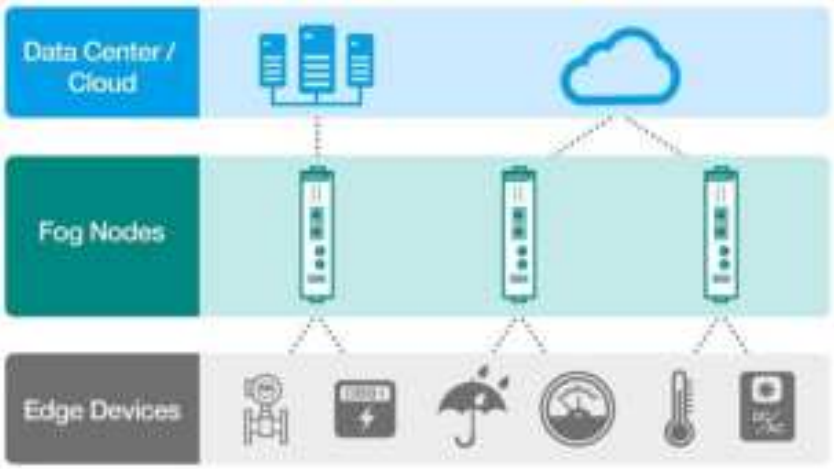


Figura 36: Rappresentazione della suddivisione in layer dell'elaborazione dei dati nel fog computing

Inoltre, l'Intelligenza Artificiale, utilizzata in varie circostante dell'IoT come visto nel paragrafo precedente, necessita di strumenti in grado di agevolare l'interazione con i database al fine di poter applicare uno stesso algoritmo a dati sempre nuovi e aggiornati; si prenda, ad esempio, il riconoscimento dei comandi negli assistenti vocali, i quali necessitano di accedere a sistemi di back-end tramite opportune interfacce per ottenere informazioni sui prodotti o per creare un ordine [26][27].

A quanto detto in precedenza, è possibile aggiungere che le aziende possono beneficiare dell'uso di API tramite la monetizzazione dei servizi costruiti sulla base dei propri sistemi IoT forniti ad aziende terze nonché ottimizzare l'intera logistica attraverso l'uso di interfacce in grado di fornire dati relativamente all'andamento delle proprie dislocazioni [28]. A tal proposito, si prenda ad esempio un sistema di distribuzione delle forniture di un locale, il quale prevede che venga fatto dell'inventariato settimanale prima di effettuare un ordine; tramite della sensoristica, è possibile ridurre i tempi verificando le mancanze in modo da effettuare ordini in maniera automatica, ottimizzando così i costi. Tale approccio può essere applicato anche all'economia domestica [29] di una casa dotata di tecnologie in grado di fornire

una lista completa e dettagliata del cibo mancante in frigo così come dei consumi, dei guasti e dei malfunzionamenti, permettendo una ottimizzazione delle prestazioni economiche ma anche ambientali.

Le API, dunque, si affermano come le protagoniste della comunicazione nei dispositivi IoT, i quali beneficiano della loro modularità e riusabilità senza che si vada ad intaccare il loro meccanismo di funzionamento interno e le tecnologie alla base.

3.3 L'UTILIZZO DEI TOOLS MULESOFT

Secondo il Connectivity Benchmark Report della Mulesoft redatto nel 2022, circa il 70% delle aziende ha difficoltà a fornire un'esperienza multipiattaforma completa ai propri clienti, azione complicata dalla presenza di sistemi operativi che richiedono linguaggi e tecnologie di programmazione eterogenee.

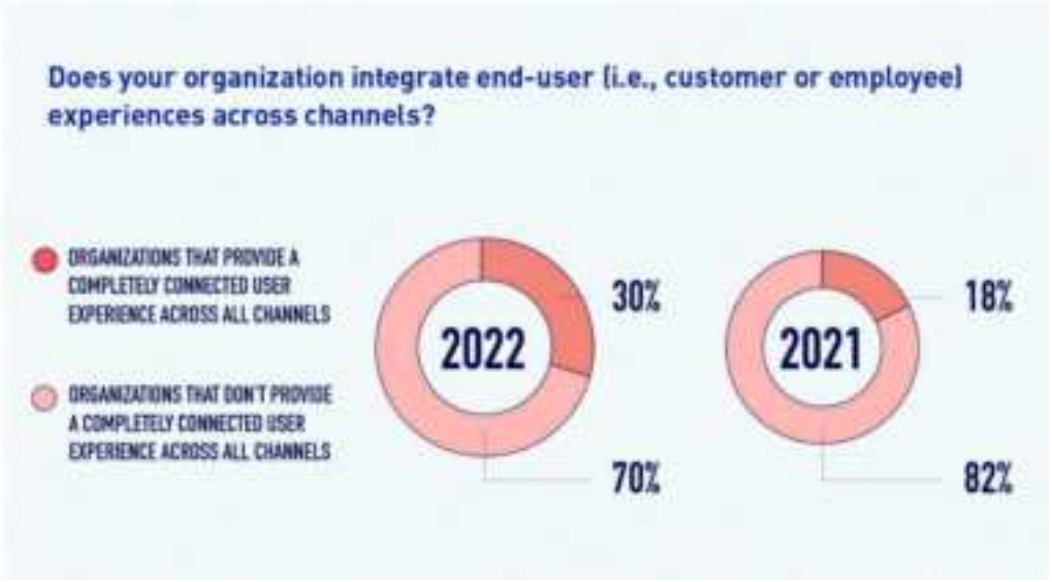


Figura 37: Aziende in difficoltà a fornire un'esperienza multipiattaforma

Fonte: https://www.mulesoft.com/press-center/feb-2022-connectivity-benchmark-report

Dalla stessa survey, è risultato come la percentuale di asset riutilizzabili internamente ad una azienda rimanga sotto la soglia del 50% e, seppur sia in aumento rispetto al 2021, raffigura come ancora più di metà del codice venga prodotto, consumato e mai riutilizzato.

Figura 38: Asset riutilizzabili all'interno di una azienda

Fonte: https://www.mulesoft.com/press-center/feb-2022-connectivity-benchmark-report

Tali evidenze possono essere il punto di partenza per l'individuazione di una strategia che permetta alle aziende di evolversi velocemente mantenendo contenuti i costi e ottimizzando l'intero processo di sviluppo. In particolar modo, in ambito IoT, grazie alla crescita esponenziale degli ultimi anni, è sempre più importante adottare un approccio metodologico improntato alla ottimizzazione dell'integrazione fra sistemi, il quale riesca a rispettare richiesta, sicurezza e fast delivery senza che ciò vada ad intaccare la qualità del prodotto o del servizio realizzato.

Come dimostrato nel progetto Contentful riportato nel capitolo precedente, l'utilizzo dei tools Mulesoft permette facilitazioni e migliorie nello sviluppo del codice grazie ad alcune caratteristiche fondamentali:

- L'ambiente è caratterizzato da una tecnologia di facile apprendimento grazie ad un codice prevalentemente a blocchi, dunque visivo ed intuitivo, dove è possibile realizzare una interfaccia pienamente funzionante già a partire dai primi approcci al sistema. Ciò rende possibile aprire la posizione di programmatore anche a personalità con un background non legato allo sviluppo software, addestrandole attraverso un breve periodo di formazione.
- La documentazione reperibile sul sito ufficiale dell'azienda guida lo sviluppatore lungo il suo lavoro a partire dalla spiegazione sul funzionamento

dei singoli scope fino alle best practises riguardo il deployment su Cloud e alla distribuzione su larga scala.

- La presenza di grande eterogeneità di scopes, i quali forniscono la risposta alle problematiche più disparate e riescono a adattarsi ai diversi contesti attraverso la definizione delle configurazioni da parte dello sviluppatore.
- La possibilità di definire nuove API e generarvi un connettore pubblico all'interno dell'Exchange Market che possa sia essere utilizzato per scopi personali che reso fruibile all'intera platea Mulesoft.
- La disponibilità di scopes sviluppati da altri utenti in grado di fornire soluzioni riutilizzabili e pronte all'uso. Inoltre, da diverse ricerche è evidente come il riuso del codice, accompagnato da una visione d'azienda improntata a tale scopo [30], permetta di ottenere un miglioramento nelle prestazioni [31] in quanto già testato, funzionante e conosciuto dai team di sviluppo.
- L'utilizzo di un approccio lineare che permette di eseguire l'intero processo nello stesso ambiente, a partire dalla progettazione fino all'implementazione passando attarverso alcuni automatismi in grado di velocizzare la programmazione.
- La presenza di meccanismi in grado di implementare sicurezza, controllo e analisi del traffico dati.

Dalle considerazioni precedenti, ne consegue che l'*environment* Mulesoft per lo sviluppo API sia uno strumento agevole ed intuitivo grazie al quale poter realizzare l'integrazione fra sistemi eterogenei che, normalmente, richiederebbero sforzi e personale maggiore [32].

La flessibilità negli usi degli scopes permette di poter realizzare delle system API in grado di interagire con sistemi di back-end differenti nonché di integrarli con sistemi datati o non più supportati al fine di renderli nuovamente fruibili. Inoltre, è possibile superare i limiti imposti dalle tecnologie proprietarie dei marchi che richiedono gli utilizzi di singole interfacce, sviluppando un approccio *open-source* alla gestione degli ambienti quali possono essere le mura domestiche o intere città al fine di accedere tramite un unico punto ai vari smart-objects.

Conclusioni

Come affrontato nel primo capitolo, esiste un gap fra le risorse investite dalle aziende e la richiesta di nuove tecnologie per cui tale prospettiva richiede mezzi in grado di modificare nonché meglio adattare come il codice viene ideato, prodotto e distribuito. In risposta a ciò, una soluzione è fornita dalle API che si identificano come uno strumento in grado di integrare sistemi differenti, più o meno datati, e di fornire l'accesso a dati e servizi attraverso il principio di incapsulamento del black-box, il quale permette di implementare sicurezza e versatilità nell'interfaccia. In qualità di strumento per sviluppo API, la piattaforma Mulesoft presenta un ricco insieme di tools sviluppati per coprire le necessità più disparate grazie alla customizzazione e al riutilizzo di codice già collaudato, fornendo modularità e riusabilità.

Nel secondo capitolo, è stata mostrata l'immediatezza e la semplicità nella realizzazione di una interfaccia tramite i tools Mulesoft attarverso la discussione di un progetto concreto sviluppato con lo scopo di sopravvivere all'evoluzione temporale e alla morte dei servizi grazie ad una profonda adattabilità e facilità di aggiornamento intrinseca alla tecnologia utilizzata, fornendo uno strumento capace di isolare ogni operazione e, in particolare, il sistema di back-end.

Nel terzo capitolo, si è affrontato il concetto di Internet delle Cose così come la sua evoluzione che è strettamente determinata dalle modalità di interazione esistenti negli ambienti dotati di tecnologie intelligenti dove, dunque, l'utilizzo di API si innesta come una valida alternativa all'isolamento tecnologico e alla necessità di fondere grandi bacini di dati per le applicazioni di algoritmi di Intelligenza Artificiale e Machine Learning, in grado di estrapolare informazioni intangibili all'occhio umano.

In conclusione, dalle esigenze sollevate è evidente come l'ambiente fornito dalla Mulesoft sia uno strumento intuitivo nonché un mezzo proiettato nel futuro in grado di portare le aziende che lo adottano ad una forte espansione organica e lavorativa grazie alla velocità di sviluppo delle API e alla rapidità di apprendimento data dall'intrinseca intuitività del codice. In particolare, è l'Internet delle Cose a risentirne positivamente dell'applicazione della tecnologia Mulesoft per via della rapida democratizzazione degli smart devices e del crescente uso della tecnologia nella vita quotidiana che richiedono la presenza di velocità, riusabilità e sicurezza per implementare le interfacce di accesso a dati e servizi e l'integrazione fra sistemi eterogenei.

Fonti bibliografiche e sitografiche

[1] «Integration of Digital Technology by Enterprises in the Digital Economy and Society Index | Shaping Europe's digital future». https://digital-strategy.ec.europa.eu/en/policies/desi-integration-technology-enterprises

[2] «What Is MuleSoft?», Salesforce Blog. https://www.salesforce.com/ap/blog/2020/05/what-is-mulesoft.html

[3] Infomentum, «What is MuleSoft? Anypoint Platform in simple terms». https://www.infomentum.com/mulesoft-plain-language

[4] M. R. S. Chouhan e S. Mishra, «API-led Connectivity Using Mulesoft For Healthcare», vol. 23, fasc. 6, p. 10, 2021.

[5] «Mulesoft Architecture and API led connectivity», UnoGeeks, 22 febbraio 2021. https://unogeeks.com/mulesoft-architecture-and-api-led-connectivity/

[6] «The benefits of an application network», MuleSoft. https://www.mulesoft.com/lp/whitepaper/api/application-network-benefits

[7] H. Garg e M. Dave, «Securing IoT Devices and Securely Connecting the Dots Using REST API and Middleware», in 2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU), apr. 2019, pp. 1–6. doi: 10.1109/IoT-SIU.2019.8777334.

[8] «Cos'è un'API (Application Program Interface) (Interfaccia di programmazione delle applicazioni)? | TIBCO Software». https://www.tibco.com/it/reference-center/what-is-an-api-application-program-interface

[9] «Welcome», RAML. https://raml.org/

[10] «CloudHub Architecture | MuleSoft Documentation». https://docs.mulesoft.com/runtime-manager/cloudhub-architecture#runtime-management-console

[11] «API Reference», Zuora. https://www.zuora.com/developer/api-reference/

[12] Infomentum, «What is MuleSoft? Anypoint Platform in simple terms». https://www.infomentum.com/mulesoft-plain-language

[13] «Internet delle cose», Wikipedia. 23 luglio 2022. https://it.wikipedia.org/w/index.php?title=Internet_delle_cose&oldid=128502748

[14] «Internet of Things: gli oggetti intelligenti prima di ogni "cosa"». https://blog.osservatori.net/it_it/internet-of-things-oggetti-intelligenti-prima-di-ogni-cosa?hsLang=it-it

[15] «Integrating Blockchain and Edge Computing in Internet of Things: Brief Review and Open Issues». https://ieeexplore.ieee.org/document/9736164/

[16] G. Fortino, «Keynote Speech 1: Blockchain-enabled Trust in Edge-based Internet of Things Architectures: State of the art and Research Challenges», in 2021 Third International Conference on Blockchain Computing and Applications (BCCA), nov. 2021, pp. 1–1. doi: 10.1109/BCCA53669.2021.9656983

[17] «The internet of things - Expanding connectivity | MuleSoft». https://www.mulesoft.com/resources/api/internet-of-things-expanding-connectivity

[18] C. Savaglio, P. Gerace, G. Di Fatta, e G. Fortino, «Data Mining at the IoT Edge», in 2019 28th International Conference on Computer Communication and Networks (ICCCN), lug. 2019, pp. 1–6. doi: 10.1109/ICCCN.2019.8846941.

[19] «What Do Integration and AI Have in Common?», MuleSoft Blog, 7 novembre 2022. https://blogs.mulesoft.com/learn-apis/api-led-connectivity/what-do-integration-and-ai-have-in-common/

[20] «IoT and Big Data: Challenges and Applications». https://www.scnsoft.com/blog/iot-big-data

[21] M. Pruthvi, S. Karthika, e N. Bhalaji, «"SMART COLLEGE"- Study of Social Network and IoT Convergence», in 2018 2nd International Conference on 2018 2nd International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), ago. 2018, pp. 100–103. doi: 10.1109/I-SMAC.2018.8653787.

[22] S. Asano, T. Yashiro, e K. Sakamura, «A proxy framework for API interoperability in the Internet of Things», in 2016 IEEE 5th Global Conference on Consumer Electronics, ott. 2016, pp. 1–2. doi: 10.1109/GCCE.2016.7800450.

[23] «OpenFog», OPC Foundation. https://opcfoundation.org/markets-collaboration/openfog/

[24] HU, Yun Chao, et al. Mobile edge computing—A key technology towards 5G. ETSI white paper, 2015, vol. 11, no 11, p. 1-16.

[25] «Fog Computing: sistema decentralizzato per IoT e Cloud», IONOS Digital Guide. https://www.ionos.it/digitalguide/server/know-how/fog-computing/

[26] «API strategy essentials | MuleSoft». https://www.mulesoft.com/lp/whitepaper/api/api-strategy-essentials

[27] «5 Ways AI, IoT, and security are shaping the API economy | MuleSoft», 5 Ways AI, IoT, and security are shaping the API economy | MuleSoft. https://www.mulesoft.com/lp/ebook/api/artificial-intelligence-chatbot

[28] «How IoT can enhance customer experiences», MuleSoft Blog, 30 novembre 2018. https://blogs.mulesoft.com/digital-transformation/iot-enhance-customer-experiences/

[29] S. Bandara, T. Yashiro, N. Koshizuka, e K. Sakamura, «Towards a standard API design for open services in smart buildings», in 2016 TRON Symposium (TRONSHOW), dic. 2016, pp. 1–7. doi: 10.1109/TRONSHOW.2016.7842883.

[30] M. Morisio, M. Ezran, e C. Tully, «Success and failure factors in software reuse», IEEE Transactions on Software Engineering, vol. 28, fasc. 4, pp. 340–357, apr. 2002, doi: 10.1109/TSE.2002.995420.

[31] «Top 3 Ways to Make Your IT Team More Productive», MuleSoft Blog, 26 luglio 2017. https://blogs.mulesoft.com/digital-transformation/business/how-to-increase-it-productivity/

[32] M. Iansiti e J. West, «Technology Integration: Turning Great Research into Great Products», Harvard Business Review, 1 maggio 1997. Disponibile su: https://hbr.org/1997/05/technology-integration-turning-great-research-into-great-products

Ringraziamenti

Mi è doveroso dedicare questo spazio della mia tesi alle persone che mi hanno supportata attraverso la loro presenza, il loro amore, la loro costanza.

In primis, un ringraziamento speciale al mio relatore Fortino Giancarlo per la sua pazienza, per il supporto e la disponibilità mostrati durante la scrittura di questo elaborato.

Ringrazio immensamente i miei genitori, i quali hanno sempre supportato le mie passioni e curato le mie necessità. Insieme possiamo superare ogni avversità.

Ringrazio la mia sorellina. Siamo così diverse ma mi insegni ogni giorno qualcosa di nuovo.

Un ringraziamento va alla mia cara nonna, scomparsa poco prima della discussione di questo elaborato. Spero che tu, ovunque ti trovi, stia sorridendo nel vedermi giunta a questo traguardo.

Un grazie va alle mie amiche ed amici che in questi anni mi sono stati vicini e con cui ho condiviso le gioie e i dolori dei vent'anni. Parlo di Myriam, Simona, Enola, Marika, Giorgio.

Un ringraziamento va a tutti i colleghi con cui ho trascorso il mio tempo tra una lezione e l'altra.

Ringrazio i ragazzi del Gruppo Fai Giovani di Catanzaro. Con voi ho condiviso il piacere di scoprire angoli della mia terra e insieme ci siamo sempre impegnati a rendere migliore la nostra città.

Ringrazio anche i ragazzi delle realtà associative e politiche dove ho potuto portare il mio impegno in questi tre anni. Mi avete insegnato cosa vuol dire perseveranza, amore e dedizione ma anche coerenza.

Un ringraziamento va a tutte le persone che in questi anni ho avuto modo di aiutare nel loro percorso di studi. Ogni vostra parola di riconoscenza è stata per me la ricompensa più bella per ogni sacrificio. Spero di potervi rappresentare al meglio in questi anni a venire.

Ringrazio fortemente l'azienda Cap4Lab dove ho vissuto una delle esperienze più formative ed emozionanti della mia vita. Mi avete mostrato cosa vuol dire amare il proprio lavoro. E ringrazio di cuore i miei colleghi in Lussemburgo che hanno reso la mia permanenza un momento indimenticabile, in particolare la mia amica Yulia.

Ringrazio di cuore il mio fidanzato Antonio, la mia fonte di ispirazione quotidiana, che mi ha sempre supportata nei momenti di difficoltà. Ci aspettano mille avventure ancora, insieme.

Un pensiero va a tutte le persone che ho conosciuto e da cui mi sono separata. Mi avete reso la persona che sono: tenace, forte, indistruttibile.

Infine, ringrazio me per tutte le volte in cui ho superato gli ostacoli della vita.

"Siate molto ambiziosi, farete grandi cose".