Manuale sviluppatore

Gruppo *N.O.S* - Progetto *Emporioλambda*

nos.unipd@gmail.com

Informazioni sul documento

Versione	2.0
Approvatori	*Panighel Cristiano*
Redattori	*Terrani Giulia* *Panighel Cristiano*
Verificatori	*Varotto Davide*
Uso	Esterno
Distribuzione	*RedBabel* *Prof. Vardanega Tullio* *Prof. Cardin Riccardo* Gruppo *N.O.S*

Descrizione

Il documento presenta le tecnologie e l'architettura della piattaforma agli sviluppatori interessati al prodotto *Emporioλambda*.

Registro modifiche documento

Versione	Data	Nominativo	Ruolo	Verificatore	Descrizione
2.0	2021_05_18	*Panighel Cristiano*	Responsabile	-	Approvazione del documento.
1.2	2021_05_16	*Panighel Cristiano*	Programmatore	*Varotto Davide*	Aggiornata §4.2.
1.1	2021_05_15	*Terrani Giulia*	Programmatore	*Varotto Davide*	Redatta §5.1 e §5.3, corretto UML §5.3.1.
1.0	2021_04_29	*Varotto Davide*	Responsabile	-	Approvazione del documento.
0.7	2021_04_24	*Tredese Leonardo*	Programmatore	*Terrani Giulia*	Redazione §6.
0.6	2021_04_23	*Tredese Leonardo*	Programmatore	*Terrani Giulia*	Redazione §5.
0.5	2021_04_19	*Brescanzin Lorenzo*	Programmatore	*Terrani Giulia*	Redazione §4.4.
0.4	2021_04_17	*Fantinato Filippo*	Progettista	*Terrani Giulia*	Redazione §4.2 e §4.3.
0.3	2021_04_16	*Fantinato Filippo*	Progettista	*Terrani Giulia*	Redazione §3 e §4.1.
0.2	2021_04_14	*Panighel Cristiano*	Programmatore	*Terrani Giulia*	Redazione §2.
0.1	2021_04_14	*Tredese Leonardo*	Programmatore	*Terrani Giulia*	Creazione del documento e redazione §1.

Indice

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Questo documento ha lo scopo di mostrare approfonditamente requisiti di sistema, tecnologie, architettura e struttura in generale della piattaforma *Emporioλambda*. Il documento è da considerarsi incompleto in quanto i contenuti verranno aggiornati e modificati avanzando con lo sviluppo del prodotto.

1.2 Scopo del prodotto

Il progetto *Emporioλambda* ha come scopo quello di rendere disponibile un servizio e-commerce[G] sfruttando tutti i vantaggi di un'architettura Serverless[G]:

- Gli sviluppatori potranno concentrare la propria attenzione sullo sviluppo del prodotto finale invece di focalizzarsi sulla gestione e sul funzionamento di server e di runtime, che siano nel cloud[G] o in locale;
- Comodità nel costruire un insieme di chiamate asincrone[G] che rispondono a diversi clienti contemporaneamente;
- Minori costi di sviluppo e di produzione;
- Semplicità nel suddividere il progetto in un insieme di microservizi[G].

1.3 Glossario

Per garantire che il documento sia chiaro e comprensibile; tutti i termini che richiedono una spiegazione aggiuntiva o che possono assumere un'interpretazione ambigua a seconda del contesto saranno indicati da una 'G' ad apice e riportati con il loro significato in appendice.

1.4 Riferimenti

1.4.1 Normativi

- **Capitolato d'appalto C2:**
  https://www.math.unipd.it/~tullio/IS-1/2020/Progetto/C2.pdf

1.4.2 Informativi

- **Slide *Prof. Cardin Riccardo* "Principi di programmazione SOLID":**
  https://www.math.unipd.it/%7Ercardin/swea/2021/SOLID%20Principles%20of%20Object-Oriented%20Design_4x4.pdf
- **Slide *Prof. Cardin Riccardo* "Stili Architetturali: Monolite e Microservizi":**
  https://www.math.unipd.it/~rcardin/sweb/2021/L03.pdf

2 Tecnologie, framework e librerie impiegate

Di seguito vengono descritte tecnologie, framework[G] e servizi di terze parti utilizzati per il progetto.

2.1 Amazon SQS

Amazon Simple Queue Service (SQS) è un servizio di accodamento messaggi completamente gestito che consente la separazione e la scalabilità di microservizi, sistemi distribuiti e applicazioni serverless. Con SQS, è possibile inviare, memorizzare e ricevere qualsiasi volume di messaggi tra componenti software senza perdite e senza dover impiegare altri servizi per mantenere la disponibilità. SQS offre due tipi di code di messaggi: le code standard offrono throughput massimo, ordinamento semplificato e distribuzione di tipo at-least-once mentre le code FIFO sono progettate per garantire che i messaggi vengano elaborati esattamente una sola volta, nell'ordine in cui sono inviati.

2.2 Amazon SNS

Amazon Simple Notification Service (Amazon SNS) è un servizio di messaggistica completamente gestito per la comunicazione application-to-person (A2P) e application-to-application (A2A). Le comunicazioni avvengono in modo asincrono[G], con un punto di accesso logico e un canale di comunicazione.

2.3 AWS Cognito

AWS Cognito permette di gestire autenticazione, autorizzazione e gestione degli utenti per le applicazioni web e mobili. Gli utenti possono accedere direttamente con un nome utente e una password, oppure tramite terze parti, ad esempio Facebook, Amazon, Google o Apple. I due componenti principali di Amazon Cognito da utilizzare separatamente o insieme sono i pool di utenti, directory utente che forniscono opzioni di registrazione e di accesso, e i pool di identità che consentono di concedere agli utenti l'accesso ad altri servizi AWS.

2.4 AWS DynamoDB

Amazon DynamoDB è un database NoSQL[G] che consente agli sviluppatori di scaricare gli oneri legati al funzionamento e al ridimensionamento di un database distribuito in modo da non doversi preoccupare del provisioning, dell'installazione e della configurazione dell'hardware, della replica, delle patch del software o del ridimensionamento del cluster[G] eliminando il carico operativo e la complessità coinvolti nella protezione dei dati sensibili.

- **Versione utilizzata:** 1.3.0;
- **Link:** https://www.npmjs.com/package/dynamodb.

2.5 AWS API Gateway

Amazon API Gateway semplifica per gli sviluppatori la creazione, la pubblicazione, la manutenzione, il monitoraggio e la protezione delle API RESTful su qualsiasi scala. Le API fungono da "porta di entrata" per consentire l'accesso delle applicazioni ai dati, alla logica aziendale o alle funzionalità dai servizi back end[G]. API Gateway gestisce tutte le attività di accettazione ed elaborazione relative alle chiamate API simultanee, inclusi gestione del traffico, controllo di accessi e autorizzazioni, monitoraggio e gestione delle versioni delle API.

2.6 AWS Lambda

AWS Lambda è un servizio di elaborazione serverless che permette di eseguire il codice senza effettuare il provisioning o gestire i server, creare una logica di dimensionamento dei cluster in funzione dei carichi di lavoro, mantenere integrazioni degli eventi o gestire i runtime. Con Lambda è possibile eseguire codice per qualsiasi tipo di applicazione o servizio di back end, senza alcuna amministrazione. È possibile scrivere le funzioni Lambda nel linguaggio preferiro (Node.js, Python, Go, Java e altri ancora) e utilizzare strumenti per creare, testare e distribuire le funzioni.

2.7 Amazon S3 Bucket

È un servizio di storage di oggetti che offre scalabilità, disponibilità dei dati, sicurezza e prestazioni all'avanguardia nel settore offrendo caratteristiche di gestione semplici da utilizzare che consentono di organizzare i dati e di configurare controlli di accesso ottimizzati per soddisfare requisiti aziendali, di pianificazione e di conformità specifici.

2.8 Dotenv

È un modulo a dipendenza zero che carica le variabili di ambiente da un file .env in process.env. La memorizzazione della configurazione nell'ambiente separato dal codice si basa sulla metodologia The Twelve-Factor App, dove le variabili d'ambiente non vengono mai raggruppate come "ambienti", ma vengono invece gestite in modo indipendente per ogni distribuzione.

- **Versione utilizzata:** 8.2.0;
- **Link:** https://www.npmjs.com/package/dotenv.

2.9 ESLint

ESLint è uno strumento di analisi statica del codice per identificare pattern problematici o codice che non rispetta certe linee guida predefinite nel codice JavaScript senza eseguirlo. Le regole in ESLint sono configurabili e le regole personalizzate possono essere definite e caricate. ESLint è scritto usando Node.js per fornire un ambiente a runtime veloce e di facile installazione attraverso npm.

- **Versione utilizzata:** 7.23.0;
- **Link:** https://www.npmjs.com/package/eslint.

2.10 Jest.js

Jest.js è un framework di test JavaScript con particolare attenzione alla semplicità e al supporto per applicazioni web di grandi dimensioni. Fornisce diverse funzionalità come la creazione automatizzata di mock, l'esecuzione di test in parallelo per aumentarne la velocità e la possibilità di testare il codice asincrono in modo sincrono. Jest trova automaticamente i test da eseguire nel codice sorgente, e funziona su progetti JavaScript che includono React, Babel, TypeScript, Node, Angular e Vue.

- **Versione utilizzata:** 26.6.3;
- **Link:** https://www.npmjs.com/package/jest.

2.11 Material-UI

Material-UI è una libreria di componenti React per sviluppare il design del proprio progetto o per poter utilizzare tutta una serie di componenti stabili predefiniti.

- **Versione utilizzata:** 4.11.4;
- **Link:** https://material-ui.com/.

2.12 Next.js

Next.js è un framework JavaScript back end per applicazioni React che non richiede alcun setup e che consente il rendering automatico lato server (SSR, server side rendering). Con Next.js si possono sviluppare applicazioni web, app mobile, desktop e web app progressive: è costruito secondo il principio di "Build once, run anywhere". Altre caratteristiche di Next.js sono suddivisione automatica del codice, routing automatico, hot code reloading (viene ricaricato solo il codice modificato) ed esportazione statica (con un solo comando può esportare un sito statico).

- **Versione utilizzata:** 10.2.0;
- **Link:** https://www.npmjs.com/package/next.

2.13 NodeJs

Node.js è un runtime system open source multipiattaforma orientato agli eventi per l'esecuzione di codice JavaScript, ha un'architettura orientata agli eventi che rende possibile l'I/O asincrono. Questo design punta ad ottimizzare il throughput e la scalabilità nelle applicazioni web con molte operazioni di input/output. Il modello di networking su cui si basa Node.js è I/O event-driven: ciò vuol dire che Node richiede al sistema operativo di ricevere notifiche al verificarsi di determinati eventi, e rimane quindi in sleep fino alla notifica stessa: solo in tale momento torna attivo per eseguire le istruzioni previste nella funzione di callback, così chiamata perché da eseguire una volta ricevuta la notifica che il risultato dell'elaborazione del sistema operativo è disponibile.

- **Versione utilizzata:** 15.4.0;
- **Link:** https://nodejs.org/it/.

2.14 Npm

Npm è un package manager per il linguaggio di programmazione JavaScript, il predefinito per l'ambiente di runtime Node.js. Consiste in un client da linea di comando, chiamato anch'esso npm, e un database online di moduli pubblici e privati che offrono diverse funzionalità: dalla gestione dell'upload di file, ai database MySQL, attraverso framework, sistemi di template e la gestione della comunicazione in tempo reale con i visitatori.

- **Versione utilizzata:** 7.0.15;
- **Link:** https://www.npmjs.com/.

2.15 React

React è una libreria Javascript open source utilizzata per creare interfacce utente o componenti UI. Può essere usato come base nello svillupo di applicazioni a pagina singola o mobile. React si occupa solamente della gestione dello stato e del rendering di tale stato nel DOM, quindi solitamente la creazione di applicazioni React richiede l'uso di librerie aggiuntive per il routing oltre ad alcune funzionalità lato client.

- **Versione utilizzata:** 17.0.2;
- **Link:** https://it.reactjs.org/.

2.16 Serverless Framework

Serverless Framework è un framework web open source e gratuito scritto utilizzando Node.js, sviluppato per la creazione di applicazioni su AWS Lambda. Il Serverless Framework è costituito da una CLI[G] open source e da una dashboard che insieme, forniscono una gestione completa del ciclo di vita delle applicazioni serverless.

- **Versione utilizzata:** 8.137.0;
- **Link:** https://www.npmjs.com/package/serverless.

2.17 Stripe

Stripe è una piattaforma esterna per i pagamenti online che consente all'e-commerce di accettare pagamenti con bancomat, carta ricaricabile o carta di credito. È una soluzione sicura e rapida, i pagamenti vengono elaborati utilizzando strumenti ad hoc sviluppati per la gestione dei flussi di pagamento con controlli anti-frode.

- **Versione utilizzata:** 8.145.0;
- **Link:** https://www.npmjs.com/package/stripe.

2.18 Swagger

Swagger è un linguaggio di descrizione dell'interfaccia per descrivere le API[G] RESTful[G] espresse utilizzando JSON[G]. Swagger viene utilizzato insieme a una serie di strumenti software open source[G] per progettare, creare, documentare e utilizzare i servizi web RESTful.

- **Versione utilizzata:** 3.0.3;
- **Link:** https://swagger.io/.

2.19 TypeScript

TypeScript è un linguaggio di programmazione open source sviluppato da Microsoft che estende la sintassi di JavaScript, aggiungendo o rendendo più flessibili e potenti varie sue caratteristiche, in modo che qualunque programma scritto in JavaScript sia anche in grado di funzionare con TypeScript senza nessuna modifica.

- **Versione utilizzata:** 4.0;
- **Link:** https://www.typescriptlang.org/.

3 Setup della piattaforma

3.1 Requisiti minimi

Per il deploy della piattaforma *Emporioλambda* è necessario disporre di un account AWS valido per ottenere le credenziali da includere nella configurazione del profilo del framework[G] serverless. Il sistema dove verrà effettuato il deploy deve avere installato:

- Npm;
- NodeJS;
- Serverless Framework.

Di seguito sono riportati i browser che supportano la piattaforma.

- **Mozilla Firefox**: versione per desktop 86.0.0 e successive, versione mobile 86.1.1 e successive;
- **Microsoft Edge**: versione per desktop 88.0.705.68 e successive, versione mobile 46.01.4.5140 e successive;
- **Google Chrome**: versione per desktop 88.0.4324.182 e successive, versione mobile 88.0.4324.181 e successive;
- **Safari**: versione per desktop 14.0.2 e successive, versione mobile 14.4 e successive.

3.2 Deploy

Per il deploy della piattaforma, seguire i seguenti passi:

- Aprire una CLI[G] e posizionarsi nella cartella dove sono contenuti i file;
- Installare le dipendenze usando il comando **npm install**;
- Creare una nuova applicazione serverless lanciando il comando **serverless**;
- Iniziare il deploy della piattaforma *Emporioλambda* lanciando il comando **serverless deploy**.

3.3 Test

L'analisi statica del codice redatto viene effettuata dal linter ESLint.
Per i test di unità è stato impiegato il framework Jest.js.

4 Backend

4.1 Descrizione generale

Le funzionalità previste dalla piattaforma hanno permesso di individuare i seguenti domini:

- **Products;**
- **Categories;**
- **Payments;**
- **Orders;**
- **Cart;**
- **Addresses;**
- **Users.**

I domini sono stati uniti riflettendo sulle loro interdipendenze al fine di individuare dei microservizi[G] utili per procedere con lo sviluppo, in particolare sono stati identificati i microservizi:

- **Products-categories service;**
- **Payments-orders service;**
- **Carts service;**
- **Addresses service;**
- **Users service.**

Lo sviluppo di singoli servizi permette una maggiore possibilità di sviluppo in parallelo dei singoli domini grazie alla loro struttura e all'agilità degli aggiornamenti e la possibilità di gestire i dati in modo decentralizzato.

4.2 Struttura dei microservizi

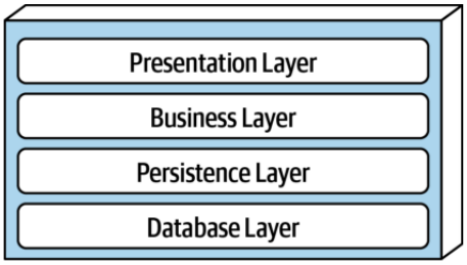

Figura 1: Layered architecture

La struttura di ogni singolo servizio si basa su una **layered architecture** dove i componenti sono organizzati in vari strati che comunicano tra loro, il principale vantaggio è che risulta molto più semplice procedere ai test attraverso mock[G].
Andando ad analizzare il singolo microservizio abbiamo:

Figura 2: DynamoDB

Figura 3: Lambda

Figura 4: API Gateway

- **DynamoDB:** per gestire i vari dati;
- **Lambda:** le funzioni che lavorano sui dati del database;
- **API Gateway:** riceve e gestisce le richieste del client richiamando le lambda e restituendone i risultati.

4.2.1 Products-categories service

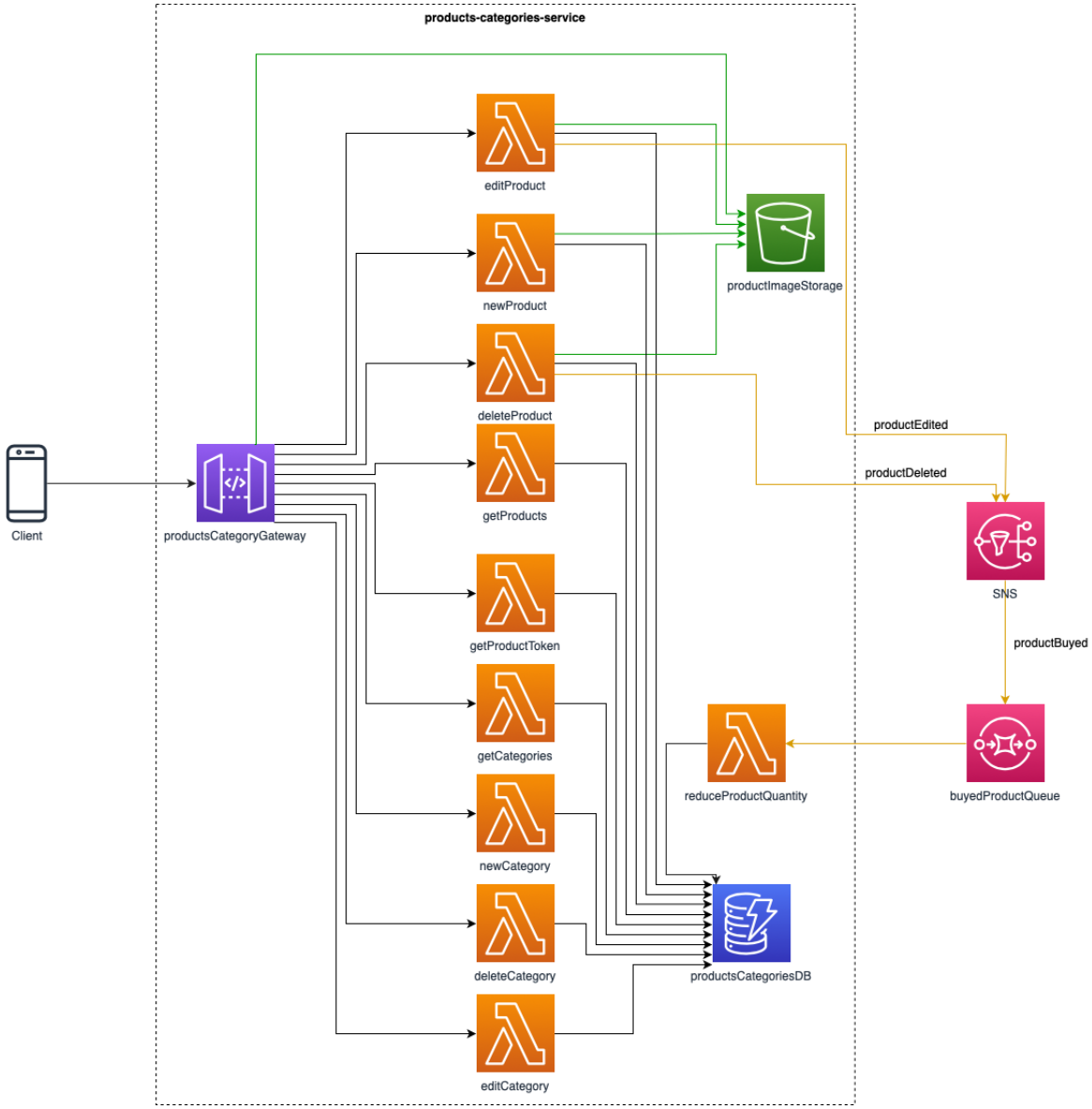

Figura 5: Products-categories service

Il microservizio **products-categories** fornisce le principali funzionalità per gestire i prodotti e le categorie alle quali vengono assegnati per le funzioni di ricerca. Sono esempi di funzioni usufruibili dall'utente:

- Inserimento di un nuovo prodotto;
- Eliminazione di un prodotto già presente nella piattaforma;
- Modifica delle specifiche di un prodotto;
- Inserimento di una nuova categoria per la classificazione dei vari prodotti in vendita.

Oltre all'utilizzo come per i successivi microservizi di AWS API Gateway, AWS Lambda e AWS DynamoDB, viene impiegato Amazon S3 bucket per gestire le immagini associate ad un prodotto, Amazon SNS che permette di notificare altri microservizi della modifica ed eliminazione di un prodotto e Amazon SQS che permette di gestire gli eventi in ingresso modificando la quantità disponibile dei prodotti nel caso di acquisto.

4.2.2 Payments-orders service

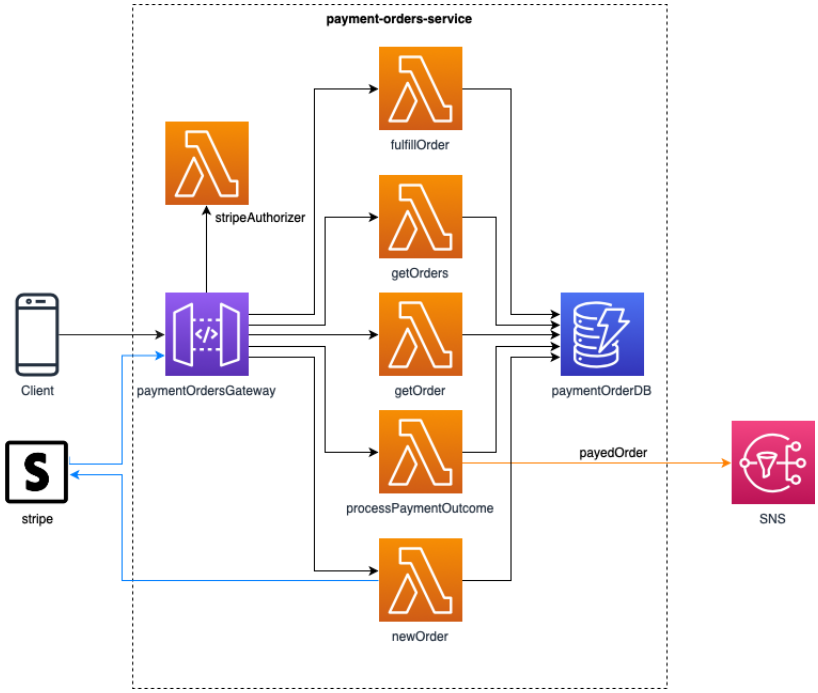

Figura 6: Payments-orders service

Il microservizio **payments-orders** fornisce le principali funzionalità per gestire il pagamento di un ordine e garantire la storicizzazione degli ordini effettuati. Anche in questo microservizio viene utilizzato Amazon SNS per notificare l'avvenuto pagamento il quale è gestito dal servizio esterno Stripe.

4.2.3 Carts service

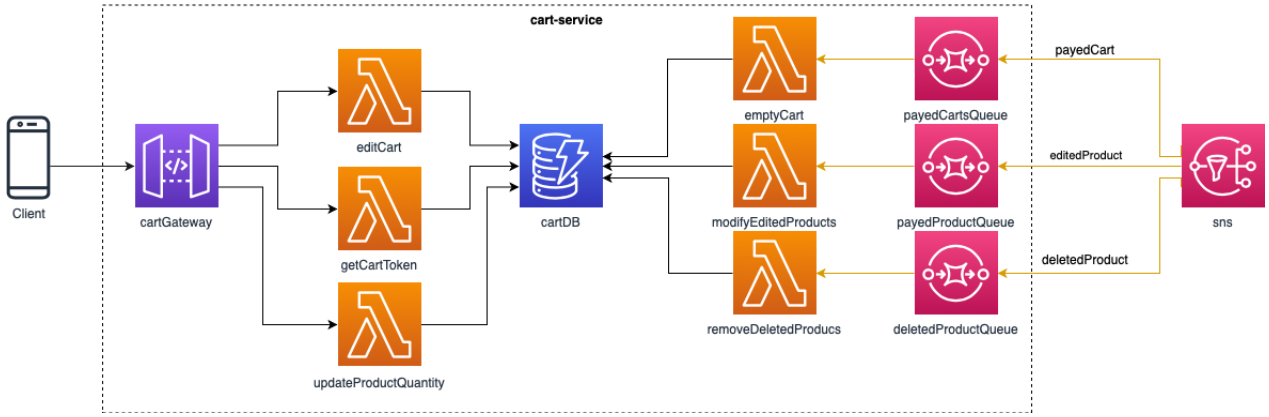

Figura 7: Carts service

Il microservizio **carts** fornisce le principali funzionalità per gestire il carrello dell'utente ovvero:

- Modificare la quantità di un prodotto;
- Aggiungere un prodotto;
- Visualizzare l'intero carrello.

Il micorservizio garantisce che il carrello sia sempre aggiornato su tutti i suoi dispositivi assicurando che eventuali modifiche fatte sui suoi prodotti esternamente, ad esempio il venditore modifica il prezzo di un prodotto, si riflettano anche all'interno dei carrelli degli utenti attraverso l'utilizzo di Amazon SNS e SQS.

4.2.4 Addresses service

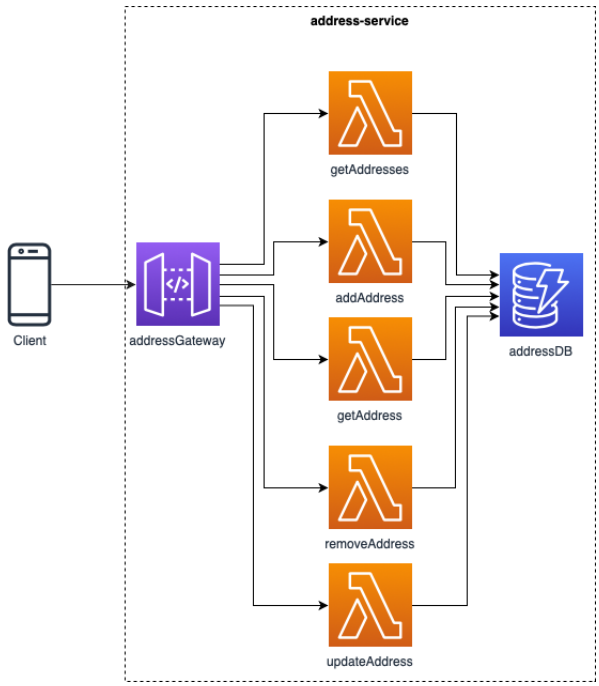

Figura 8: Addresses service

Il microservizio **addresses** fornisce le principali funzionalità per gestire gli indirizzi di spedizione di un utente come l'aggiunta, la modifica o la rimozione di un indirizzo. In questo microservizio vengono impiegati solo AWS API Gateway, AWS Lambda e AWS DynamoDB.

4.2.5 Users service

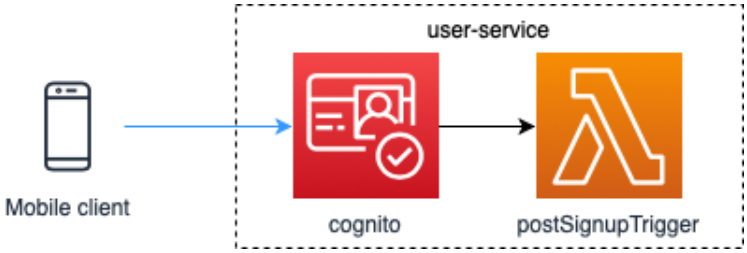

Figura 9: Users service

Il microservizio **users** fornisce le principali funzionalità per gestire la registrazione e l'autenticazione di un utente alla piattaforma attraverso l'utilizzo di AWS Cognito.

4.3 Struttura del backend

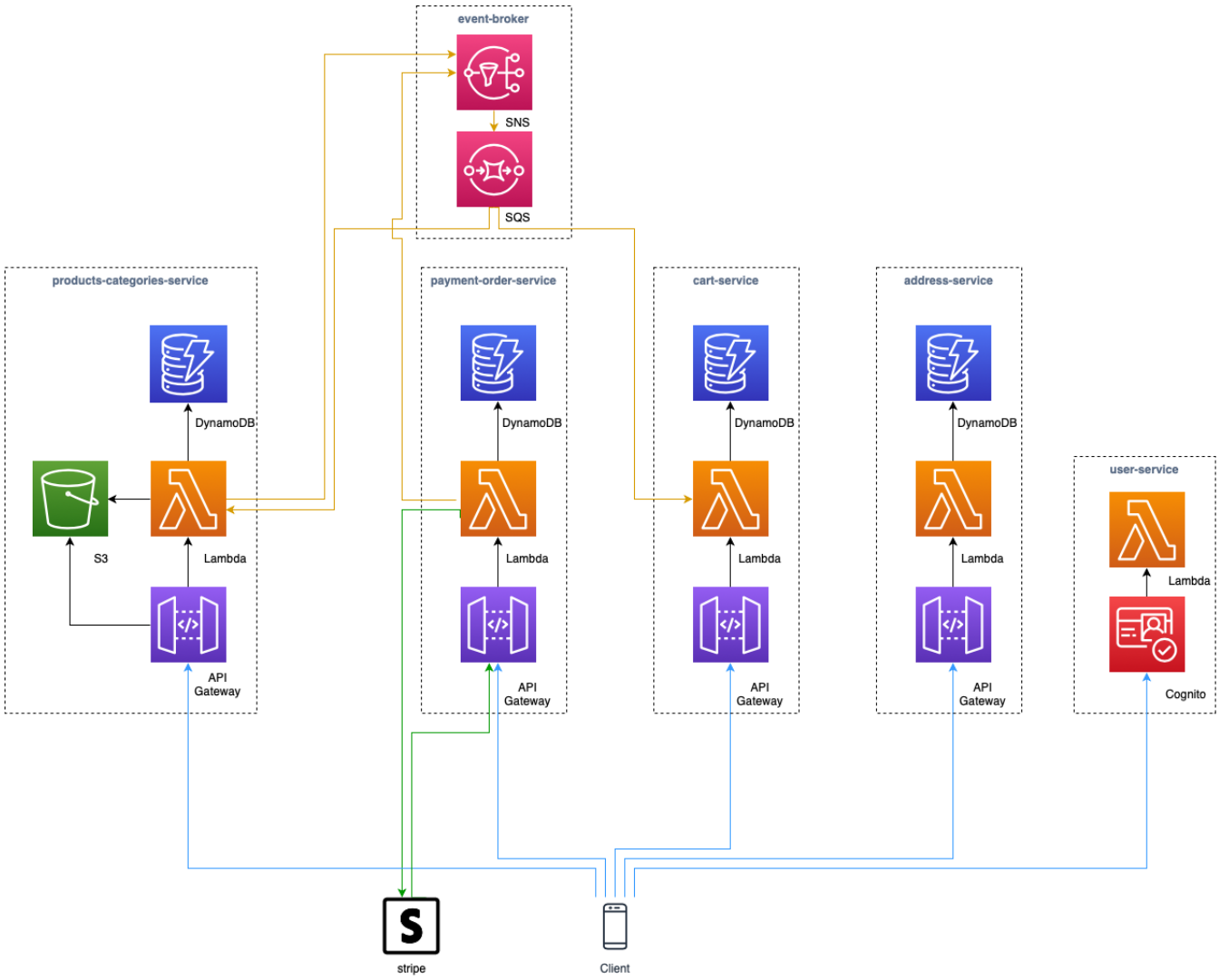

Figura 10: Architettura backend

Per slegare i microservizi tra di loro vengono utilizzate due strategie. La prima consiste nel gestire le chiamate asincrone attraverso SNS il servizio che fa da event broker[G] su AWS, eliminando così lo sviluppo di integrazioni tra microservizi e limitandosi a quelle per il message broker[G]. La seconda invece è pensata per eliminare le chiamate sincrone tra microservizi, come quelle di validazione. Per far ciò si utilizza la firma digitale legata a un microservizio per verificare l'integrità delle richieste del client, sostituendo così le varie integrazioni necessarie a un semplice check attraverso la chiave pubblica del microservizio.

4.4 Diagrammi di sequenza

Di seguito sono riportati i diagrammi di sequenza per descrivere alcune delle principali operazioni effettuabili all'interno della piattaforma.

4.4.1 Creazione ed eliminazione di un prodotto effettuata dal venditore

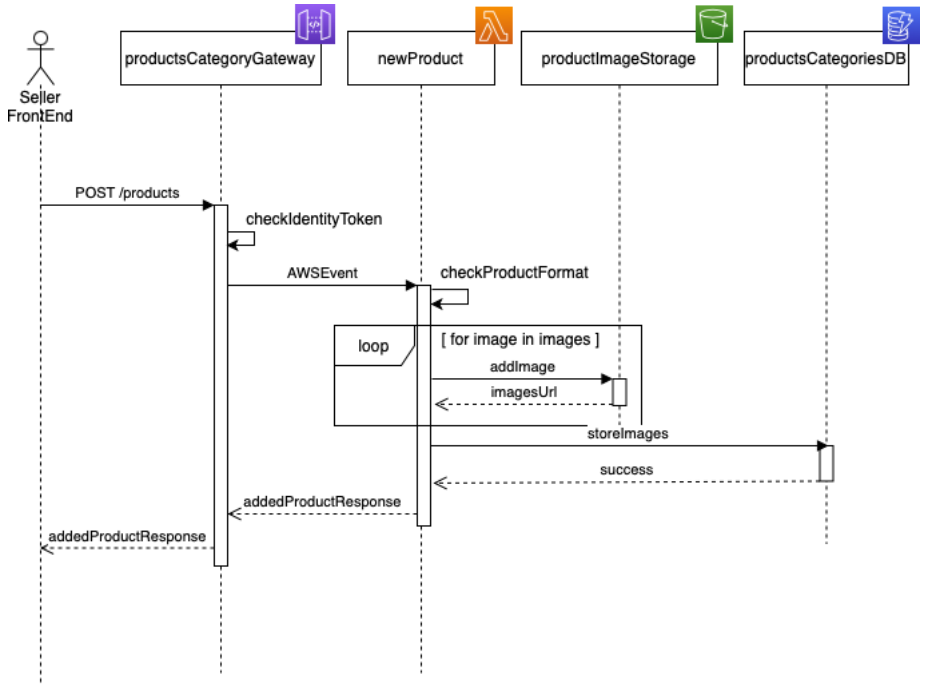

Figura 11: Creazione di un nuovo prodotto nella piattaforma

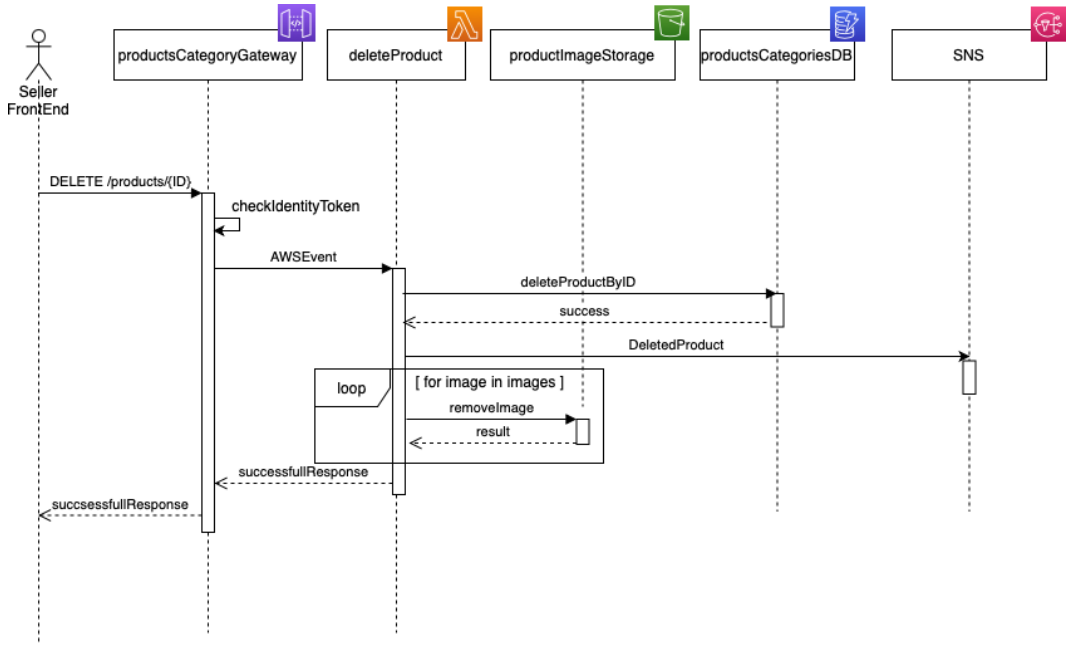

Figura 12: Eliminazione di un prodotto dalla piattaforma

I diagrammi rappresentano rispettivamente la creazione e l'eliminazione di un prodotto dalla piattaforma, in queste operazioni è necessario prevedere la gestione del salvataggio delle eventuali immagini associate ai prodotti tramite Amazon S3 bucket.

4.4.2 Aggiunta di un prodotto al carrello

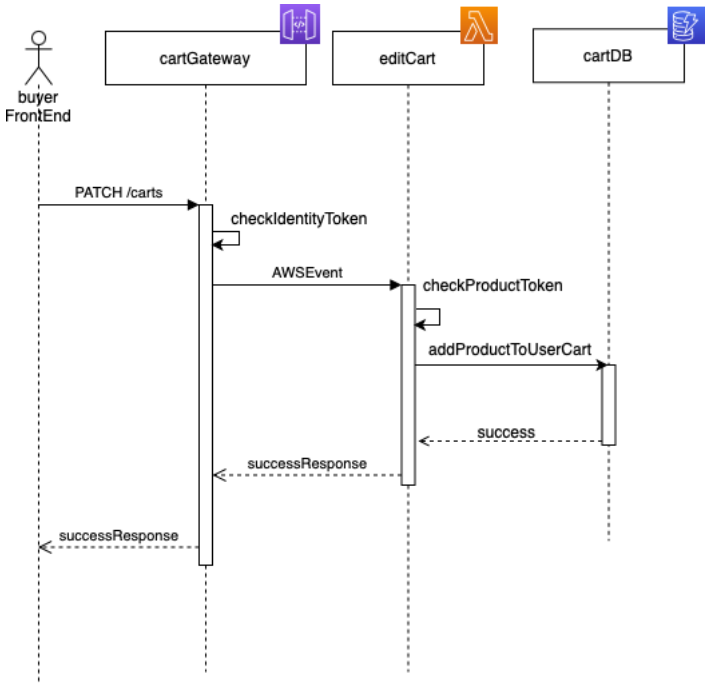

Figura 13: Aggiunta di un prodotto al carrello

Il diagramma rappresenta l'interazione delle tecnologie quando l'utente aggiunge un nuovo prodotto al carrello, in questo caso occorre modificare lo stato del carrello.

4.4.3 Modifica di un indirizzo di spedizione

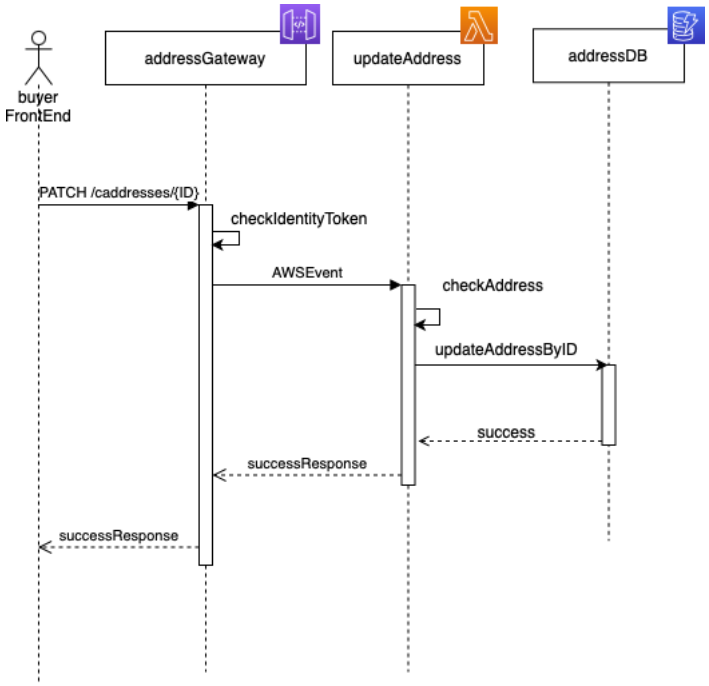

Figura 14: Modifica di un indirizzo di spedizione

All'interno della piattaforma l'acquirente può aggiungere, eliminare o modificare gli indirizzi di spedizione inseriti, il diagramma rappresenta la collaborazione delle tecnologie per effettuare quest'ultima operazione.

4.4.4 Pagamento e creazione di un ordine

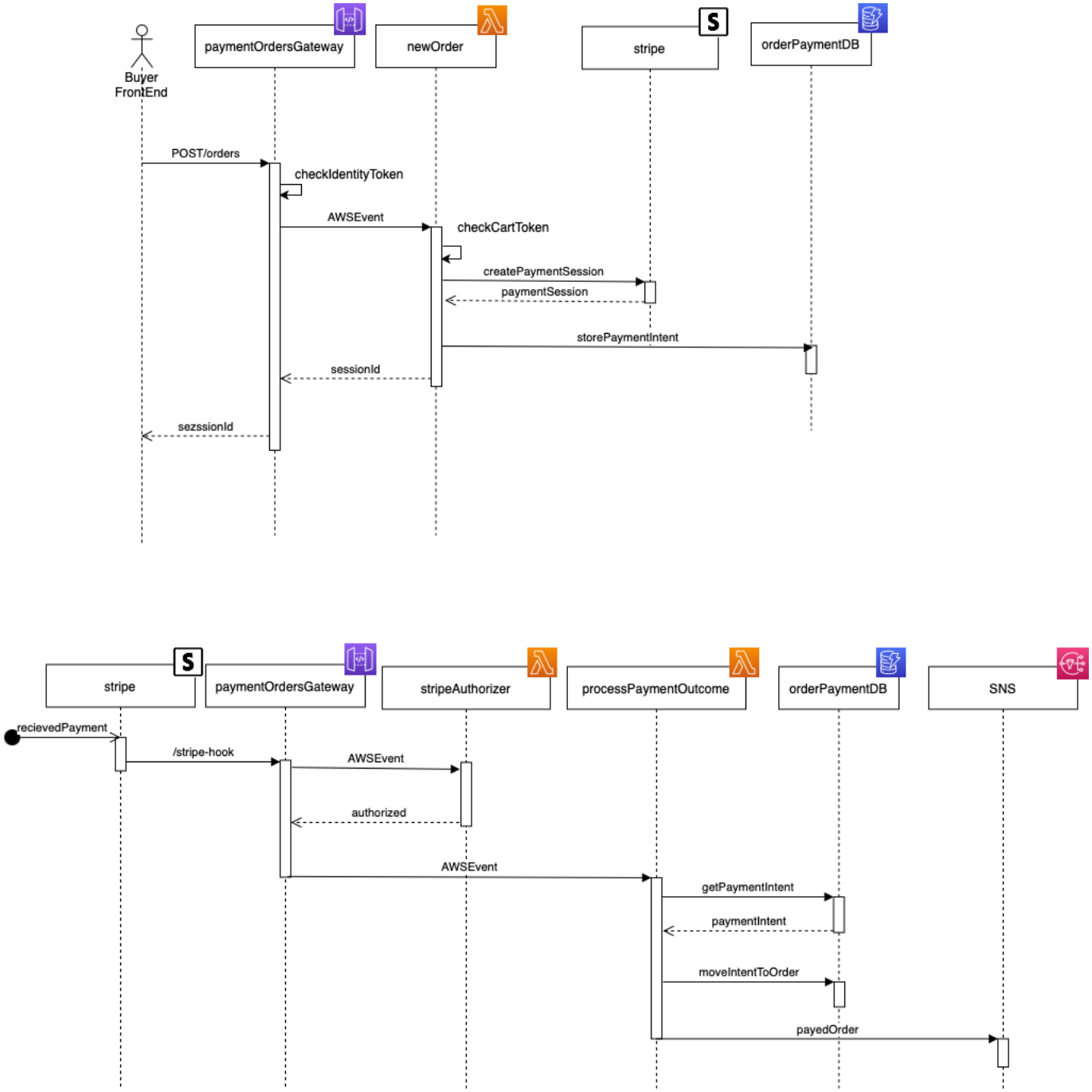

Figura 15: Nuovo ordine e pagamento dello stesso

I precedenti diagrammi rappresentano le tecnologie che entrano in gioco quando l'acquirente della piattaforma effettua un pagamento e, se questo è andato a buon fine, di conseguenza quanto inserito all'interno del carrello verrà trasformato in un'ordine effettuato e di conseguenza visualizzabile dall'utente all'interno dell'elenco degli ordini effettuati.

5 Frontend

5.1 Descrizione generale

L'architettura del front end pone come figura principale l'utente che interagisce con la piattaforma attraverso i componenti che costituiscono le schermate della stessa. Queste collaborano ed interagiscono a loro volta con le funzioni previste dai microservizi comunicando con il back end[G].

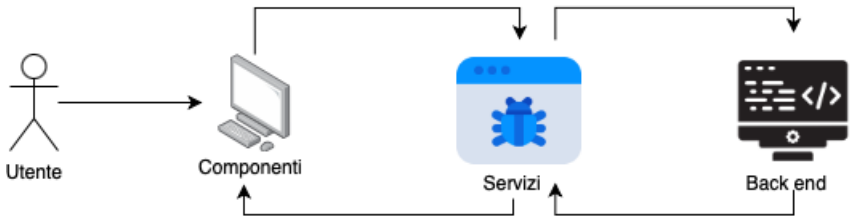

Figura 16: Architettura front end

Attraverso l'utilizzo di Next.js, vengono utilizzati entrambi i tipi di pre-rendering dei dati:

- **Static generation:** il sistema genera le pagine HTML[G] a build time e vengono riusate ad ogni richiesta;
- **Server-side rendering:** il sistema genera le pagine HTML a ogni richiesta.

In *Emporioλambda* vengono utilizzati i metodi:

- **getStaticProps:** utilizzata per il data pre-fetching di tipo static genertion;
- **getServerSideProps:** utilizzata per il data pre-fetching di tipo server-side rendering.

5.2 Diagramma dei package

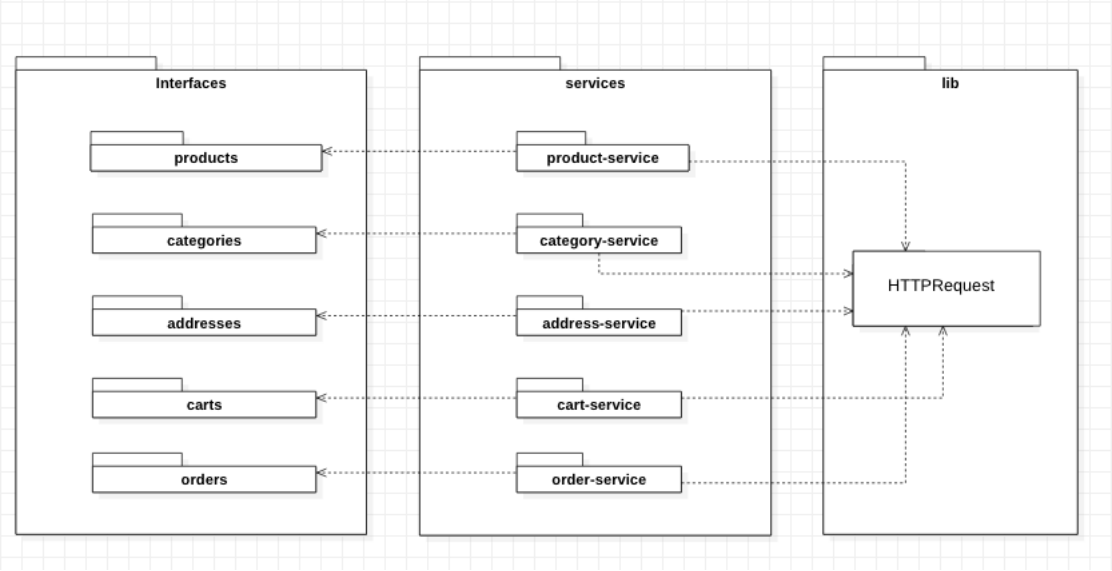

Figura 17: Diagramma dei package

Il diagramma rappresenta le dipendenze tra le classi che individuano i microservizi[G]. Il package **interfaces** contiene tutti i tipi che verranno utilizzati dal frontend. La classe **HTTPRequest** è stata implementata per centralizzare le chiamate HTTP[G] e permettere il controllo dei tipi. Consiste in un wrapper della libreria nativa fetch e ridefinisce i metodi GET, POST, PATCH, PUT e DELETE. Ottiene la variabile d'ambiente contenente l'url del server dal file di configurazione relativo di dotenv, dal costruttore gli verrà passato l'url alla specifica API[G] il quale verrà aggiunto all'url precedente e su questo sarà possibile invocare i metodi precedentemente dichiarati. Il package **services** rappresenta l'unico punto di collegamento tra i microservizi del backend e il frontend, tramite API[G] redatte utilizzando SwaggerHub, i vari servizi al suo interno andranno a invocare HTTPRequest per comunicare con il backend attraverso i metodi definiti. I microservizi individuati e utilizzati per il frontend sono:

- **Product;**
- **Address;**
- **Categories;**
- **Orders;**
- **Carts.**

5.3 Services

Ogni package presente all'interno del package services contiene il microservizio[G] che, invocando l'HTTPRequest, comunica con il back end. Ogni servizio contiene funzioni pubbliche asincrone e per questo motivo il tipo di ritorno è una **Promise¡T¿**.

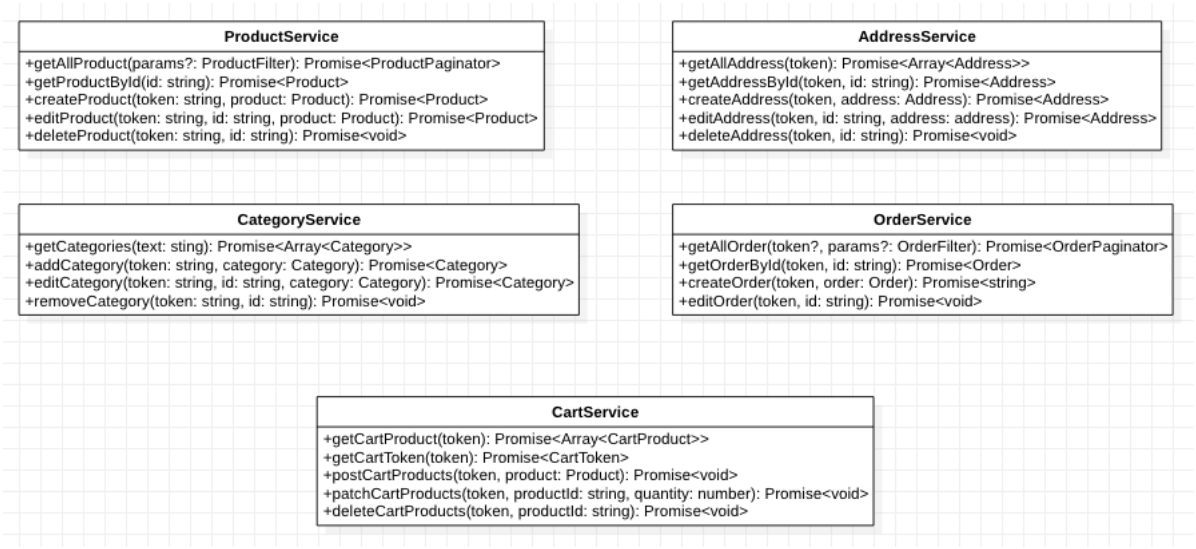

Figura 18: Classi Service

5.3.1 Esempio di funzionamento

Di seguito si riporta il diagramma delle classi relativo al funzionamento del microservizio **product**. Come si può vedere è presente un'interfaccia ProductService che viene implementata dalla classe ProductServiceFetch e ogni suo metodo istanzia un oggetto HTTPRequest per l'API di interesse. È possibile implementare l'interfaccia ProductService diversamente rispetto a quanto rappresentato senza la necessità di modificare il codice utilizzatore, attraverso la modifica della variabile d'ambiente NEXT_PUBLIC_SERVICE_METHOD sarà possibile scegliere quale implementazione utilizzare altrimenti verrà di default selezionata l'implementazione ProductServiceFetch. Per poter procedere con lo sviluppo della parte dei prodotti del frontend parallelamente con il backend, lo sviluppatore ha a disposizione la classe ProductServiceMock la quale ritornerà i dati predefiniti senza dover richiamare il server, basterà impostare la variabile NEXT_PUBLIC_SERVICE_METHOD="mock".

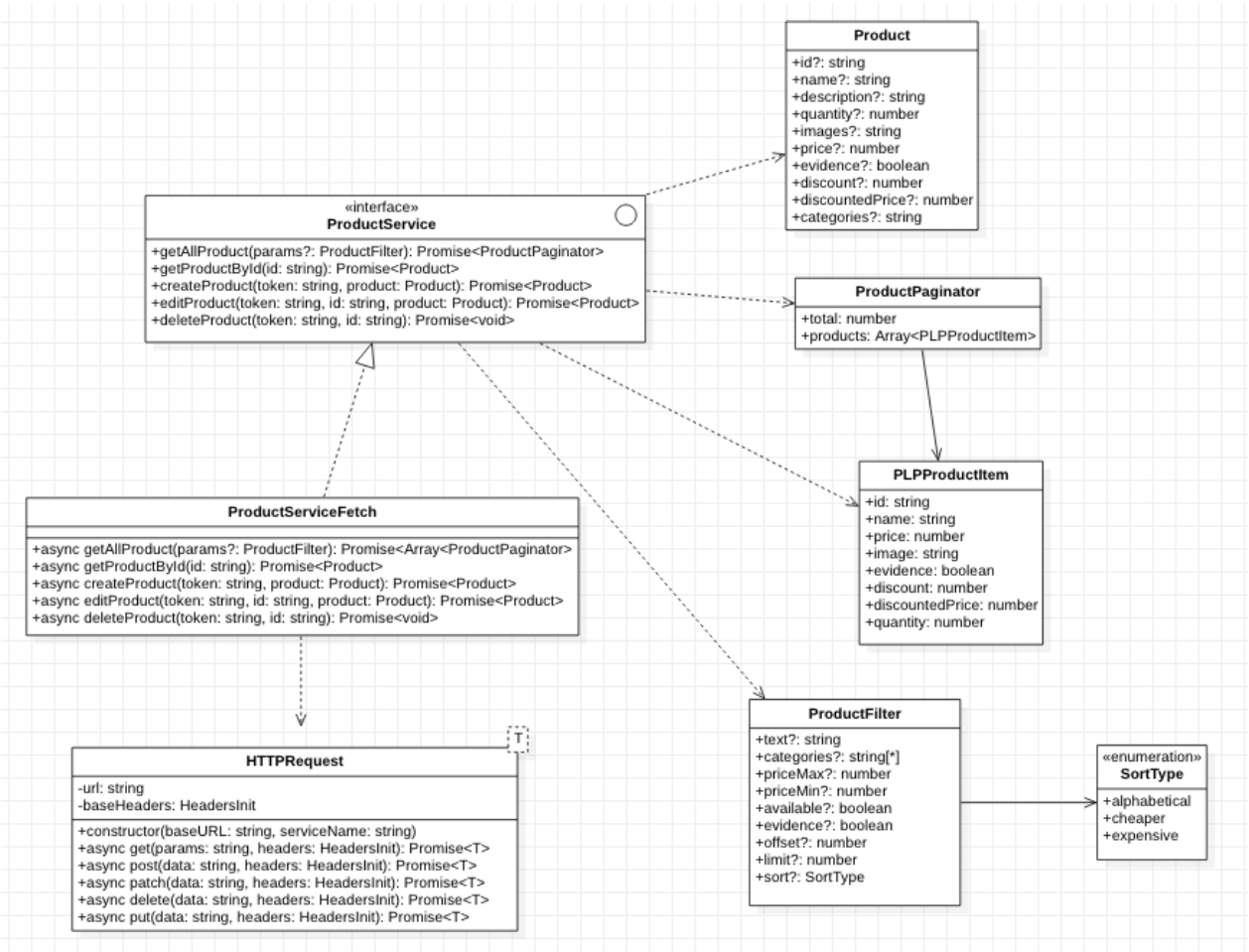

Figura 19: Microservizio Product

6 Glossario

A

API
Con Application Programming Interface (API) si indica un insieme di procedure per lo svolgimento di un dato compito. Tale termine spesso designa le librerie software disponibili in un linguaggio di programmazione.
Asincrone
I protocolli asincroni consentono la trasmissione per singolo carattere, nella trasmissione asincrona l'intervallo temporale tra il bit finale di un carattere e il bit iniziale del carattere successivo è indefinito.

B

Back end
Una applicazione back end è un programma con il quale l'utente interagisce indirettamente, in generale attraverso l'utilizzo di una applicazione front end[G]. In una struttura client/server il back end è il server.

C

CLI
CLI (command line interface) indica una interfaccia a riga di comando o anche console. A volte detta semplicemente riga di comando o prompt dei comandi, è un tipo di interfaccia utente caratterizzata da un'interazione testuale tra utente ed elaboratore.
Cloud
Con il termine inglese cloud, in alternativa anche cloud computing, si indica un paradigma di erogazione di servizi offerti on demand da un fornitore ad un cliente finale attraverso la rete internet.
Cluster
Insieme degli elementi connessi tra loro tramite una rete telematica.

E

E-commerce
Piattaforma web attraverso la quale commercianti e acquirenti vengono in contatto.
Event Broker
È un software middleware, un'appliance o SaaS utilizzato per trasmettere eventi tra produttori di eventi e consumatori in un modello di pubblicazione-sottoscrizione. Supportano lo sviluppo di app e microservizi cloud nativi e event-driven.

F

Framework
Sistema che consente di estendere le funzionalità del linguaggio di programmazione su cui è basato, fornendo allo sviluppatore una struttura coerente al fine di effettuare azioni e comandi in modo semplice e veloce.

H

HTML/HTML5
Linguaggio di markup per la strutturazione delle pagine web.

J

JSON
È una rappresentazione testuale e priva di schemi di dati strutturati basata su elenchi ordinati. Sebbene JSON sia derivato da JavaScript, è supportato in modo nativo o tramite librerie nella maggior parte dei linguaggi di programmazione. JSON è comunemente, ma non esclusivamente, utilizzato per scambiare informazioni tra client web e server web.

M

Message Broker
È un software che consente ad applicazioni, sistemi e servizi di comunicare tra loro e di scambiare informazioni. Il broker di messaggi esegue questa operazione traducendo i messaggi tra protocolli di messaggistica formali. Ciò consente a servizi interdipendenti di "dialogare" direttamente tra loro, anche se sono stati scritti in lingue diverse o implementati su piattaforme diverse.
Microservizio
Approccio architetturale alla realizzazione di applicazioni. Quello che distingue l'architettura basata su microservizi dagli approcci monolitici tradizionali è la suddivisione del prodotto nelle sue funzioni di base. Ciascuna funzione, denominata servizio, può essere compilata e implementata in modo indipendente. In questo modo i singoli servizi possono funzionare, o meno, senza compromettere gli altri.
Mock
Sono degli oggetti simulati che riproducono il comportamento degli oggetti reali in modo controllato. Un programmatore crea un oggetto mock per testare il comportamento di altri oggetti, reali, ma legati ad un oggetto inaccessibile o non implementato. Allora quest'ultimo verrà sostituito da un mock.

N

NoSQL
Database caratterizzati dal fatto di non utilizzare il modello relazionale, di solito usato dalle basi

di dati tradizionali.

O

Open source
Software non protetto da copyright, segue la filosofia della produzione collaborativa e dell'accesso pubblico al codice sorgente.

R

RESTful
È un'interfaccia di programmazione delle applicazioni conforme ai vincoli dell'architettura REST, abbreviazione per Representational State Transfer, è un insieme di principi architetturali atti alla creazione di servizi web. Si espongono operazioni per manipolare i servizi, appoggiandosi a protocolli web esistenti, tipicamente HTTP.

S

Serverless
Network la cui gestione non viene incentrata su dei server, ma viene dislocata fra i vari utenti che utilizzano il network stesso, quindi il lavoro necessario di gestione del network viene eseguito dagli stessi utilizzatori.