Università degli Studi di Modena e Reggio Emilia

Dipartimento di Scienze Fisiche, Informatiche e Matematiche

Corso di laurea in Informatica

Engim report service

Relatore:

Prof Canali

Candidato:

Dumitru Frunza

Anno academico 2021/2022

1 Presentazione della situazione esistente

1.1 Engim srl

Engim è una piccola azienda informatica che ha due prodotti principali: servizioGPS e TwiceTouch. ServizioGPS è un servizio di noleggio di tracker gps ed è il prodotto di punta. La maggior parte dei clienti sono comuni che usano i nostri dispositivi per tracciare il lavoro delle macchine pulisci neve e spargi sale. I tracker sono sia dispositivi fisici da collegare al veicolo che un app per android.
Twicetouch è un servizio di noleggio di dispositivi di sicurezza individuale. Il prodotto serve per i lavoratori in solitario, nel evenienza di un qualsiasi incidente viene inviata una segnalazione a dei numeri preposti. Questo segnalazione avviene nel caso in cui venga avvertito un grosso urto, oppure il lavoratore è fermo per un periodo prolungato, presumibilmente svenuto. Simile a servizioGPS c'è un prodotto fisico, il badge di TwiceTouch e un app per telefono.

1.2 Infrastruttura

Le tecnologie usate per servizioGPS sono le seguenti:

- Ruby on rails full stack
- Mariadb e Redis come database
- Python come back end di supporto
- Java per i prodotti app

Il servizio è inoltre diviso su molteplici server per evitare sovraccarico nelle giornate invernali, quando arrivato migliaia di punti contemporaneamente. Recentemente è stato un introdotto un server in AWS, aprendo la possibilità di estrarre logica comune a tutti i server tramite le lambda e quindi rimuovere il peso dai singoli server.

1.3 Microservizi

In alcuni casi non è necessario avere un servizio costantemente acceso che gira, nello specifico, se devo fare alcune operazioni periodiche o in maniera discontinua allora non è utile avere un servizio o un api che è sempre in ascolto per un input. Ecco che entrano in gioco i Microservizi.
A differenza dei servizi, un microservizio non ha un suo server, non ha un suo ambiente e non dipende da nessun altra tecnologia. Il Microservizio si crea l'ambiente ogni volta che viene richiamato (cold start), e rimane attivo per un lasso di tempo, svolge una specifica funzione, definita da un API precisa, e ritorna un messaggio HTML. Diventa molto vantaggiosa tale operazione soprattutto nel ambito web dove è molto naturale usare delle api per fare specifiche operazioni.

1.4 Sistema automatico di generazione di report

Le informazioni di ogni cliente vengono archiviate anno dopo anno. Nel caso in cui il cliente abbia necessità o interesse a salvare i dati del loro lavoro possono stampare o scaricare una serie di informazioni dal sito. La più importante stampa sono i punti che arrivano e le loro specifiche. Al momento questo è gestito internamente tramite una gemma di ruby. Il sistema attuale

crea un file HTML, che viene poi convertito in pdf. Questo crea una serie di gravi problemi:

- Fare il parsing simili è estremamente costoso in maniera computazionale
- Per quanto è facile scrivere HTML il controllo sul prodotto finale è basso
- La conversione è molto lenta

Questa operazione inoltre si presta molto bene ai microservizi. È un operazione ripetitiva, ben definita che ritorna sempre un file pdf. Quindi sarebbe possibile evitare calcoli extra di parsing e evitare di sovraccaricare il server se venisse esportata come microservizio. Inoltre una volta creata la logica per una singola stampa è possibile allargare la lamba per la stampa di un qualsiasi altro servizio.

2 Requisiti del progetto

2.1 Descrizione

Il progetto deve essere una lambda su AWS. La lambda deve essere in grado di ricevere un json, autenticare tramite token e whitelist, salvare il progetto su un bucket S3 e ritornare il link di dowload al pdf. Se qualcosa va storto deve ritornare 400 se è un problema di input oppure 500 se è un problema di connessione al bucket. Il programma inoltre deve essere modulare per permettere di espandere in futuro le stampe ad altri servizi oltre che servizioGPS e creare altri tipi di stampa come kml e exel.

2.2 Scelte del linguaggio

Le lambda su AWS hanno il supporto per i linguaggi più popolari del momento, in particolare ho preso in considerazione python, javascript e ruby. La libreria che faceva di più al mio caso era pdfKit di javascript, che è un port di pdfkit libreria di php per scrivere pdf senza dover far parsing di altri file. Ruby aveva più libreria con parsing intermedi e python aveva libreria non mantenute.

3 Sviluppo

3.1 Workflow

Il mio senior ha deciso di fare un implementazione a grossi step, ogni volta facendo un piccolo pezzo per poi controllare che tutto funzioni correttamente. Ogni issue è uno step e a ogni step il progetto viene testato sia manualmente su aws tramite simulazioni di richiesta sia con test automatizzati tramite mocha.

3.2 Creazione di un file e salvataggio su S3

La lamda è una funzione assincrona che viene eseguita ogni volta che riceve una chiamata https. Bisogna prima di tutto creare una connessione con il bucket di s3, una volta creata questa connessione si può salvare un file passando il filepath oppure uno stream di data e avremo salvato il file su S3

3.3 Stampa di un file pdf da un json

Le operazioni utili al mio scopo text per scrivere una stringa, stroke per disegnare una linea e infine è possibile manipolare colori e font. Inizialmente avevo deciso di programmare in maniera funzionale perché risultava una logica più semplice essendo un breve script, con l'aumentare della complessità e con il modo di esportare di javascript ho pensato che una classe fosse più adeguata. Quindi c'è un costruttore che definisce la grandezza del documento, questo mantiene la relazione di un foglio A4 (circa 1.41) in modo che stampa e visualizzazione sono sempre molto comodi. Dopo l'inizializzazione si inizia la scrittura dei blocchi. Ogni json potrebbe avere un header, body e table. Se uno dei 3 manca, non verrà scritto e gli altri si organizzeranno di conseguenza. Ogni metodo di scrittura controlla che non ci siano dati mancanti che possano nuocere alla stampa come ad esempio se manca un titolo non è una stampa valida, e ritornano 400. Se tutto va bene e la stampa va a buon fine, è possibile mandare l'oggetto con get document. Ogni possibile caso è testato con mocha, la connessione con AWS è mochata tramite aws-sdk e gran parte dei dati è generata casualmente tramite faker. Se tutto va a buon fine il programma ritorna 200 e il link del download.