

# 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. Claudia Canali

Candidato:

Dumitru Frunza

Anno accademico 2021/2022



## Elenco delle figure

1	Architettura . . . . .	5
2	Casi d'uso . . . . .	12
3	Lista di attività . . . . .	15
4	Stampa del elenco attività . . . . .	16
5	Attività visualizzazione . . . . .	17
6	Stampa di una singola attività . . . . .	18
7	Performance . . . . .	19

# Indice

<b>Introduzione</b>	<b>1</b>
<b>1 Engim Srl</b>	<b>4</b>
1.1 L'azienda . . . . .	4
1.2 Infrastruttura . . . . .	5
1.3 Microservizi . . . . .	6
1.4 Sistema automatico di generazione di report . . . . .	7
<b>2 Requisiti del progetto</b>	<b>8</b>
2.1 Descrizione . . . . .	8
2.2 Sicurezza . . . . .	8
2.3 Correttezza dei dati . . . . .	9
2.4 Problemi interni del server . . . . .	10
2.5 Presupposti e dipendenze . . . . .	11
2.6 Features . . . . .	11
<b>3 Implementazione</b>	<b>12</b>
3.1 Flusso di lavoro . . . . .	12
3.2 Creazione di un file e salvataggio su S3 . . . . .	13
3.3 Generazione di un PDF da un JSON . . . . .	13
3.4 Test automatizzati . . . . .	14
3.5 Ruby on Rails . . . . .	14
<b>4 Applicazione e performance</b>	<b>15</b>

4.1	Risultati . . . . .	15
4.2	Performance . . . . .	18
	<b>Conclusione</b>	<b>21</b>
	<b>Bibliografia</b>	<b>23</b>

## Introduzione

Un software gestionale è un sistema creato per aiutare un'azienda a organizzare e gestire il lavoro. Engim offre la possibilità di monitorare le mansioni di un lavoratore e garantire la sua sicurezza tramite ServizioGPS e Twice-Touch, i due prodotti di punta dell'azienda. In particolare, servizioGPS salva il percorso di una macchina lavoratrice e permette di calcolare il contributo dovuto. Ogni via attraversata viene salvata, ogni 5 secondi viene salvata una coordinata GPS.

Nell'affrontare una grande quantità di dati, diventa importante disporre di un metodo affidabile e intuitivo per l'archiviazione degli stessi. Un modo comune per salvare informazioni è tramite un file XLSX o PDF. I file PDF sono semplici da usare, da interpretare e sono estremamente diffusi rendendoli perfetti per un cliente senza capacità tecniche. Un servizio simile può essere parte integrale del software in quanto effettivamente è un'estensione dei servizi offerti. Questo è vero fin tanto che richiede minime risorse e tempo di calcolo.

Cosa succede quando un servizio secondario, importante per i clienti ma non abbastanza da giustificare un grosso impegno da parte del server, pesa gravemente sul software? È il problema che sta affrontando Engim; il servizio di conversione da dati a PDF richiede troppe risorse per periodi prolungati rendendo difficile la user experience ma soprattutto sovraccarica il server. Intuitivamente si potrebbe migliorare la situazione attuale, andando a riscrivere il codice incriminato. Questa soluzione però risolve solo il problema in termini di tempo e non di risorse, resta comunque un processo potenzialmen-

te gravoso sul server. Un altro possibile approccio sarebbe quello di scaricare la responsabilità del calcolo su qualcun altro.

È possibile effettivamente astrarre il processo di conversione, portarlo su un servizio di terze parti migliorando performance e risparmiando risorse.





# 1 Engim Srl

## 1.1 L'azienda

Engim è una società che si occupa di creare soluzioni tecnologiche in ambito ICT, telecomunicazioni, sistemi di gestione e mobilità. Da oltre 10 anni operano nel mercato della tracciabilità di flotte e attività e della sicurezza dei lavoratori in solitario.

ServizioGPS è il noleggio di tracker gps per veicoli lavoratori. Una prevalente parte dei clienti sono comuni che, tramite i prodotti Engim, tracciano il percorso delle macchine spazzaneve e spargisale. I tracker possono essere prodotti fisici oppure un app per smartphone. A loro volta i prodotti fisici si dividono in fissi e mobili. Il servizio include anche un gestionale per poter visualizzare, modificare o archiviare i propri dati.

Twicetouch è noleggio di dispositivi di sicurezza individuale. Il prodotto tutela i lavoratori in solitario mandando una segnalazione in caso di emergenza.

Esistono due tipi di rilevazione:

- caduta: l'accelerometro del dispositivo rileva un urto pericoloso
- assenza di movimento: il lavoratore non si è mosso per un lasso prolungato di tempo, quindi si presume che possa essere incosciente

Similmente a servizioGPS è possibile noleggiare un dispositivo fisico (badge) oppure l'app per android. In entrambi i casi è possibile impostare i numeri in caso di emergenza, che riceveranno una chiamata e un messaggio SMS.

## 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 il prodotti app

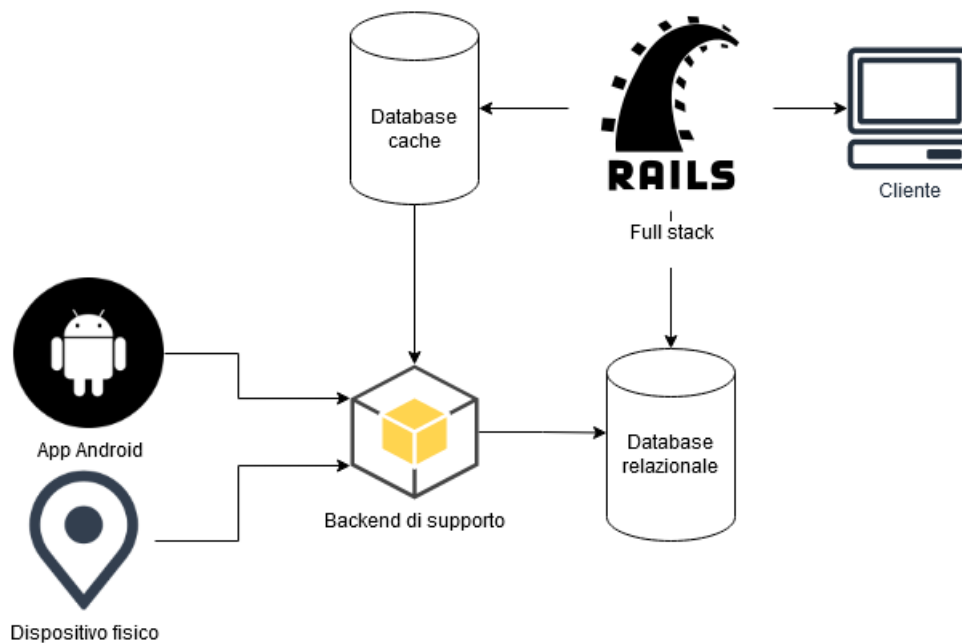


Figura 1: Architettura

Ruby on Rails è usato per front-end e gran parte del backend di servizioGPS. Il fetcher invece si dedica esclusivamente alla elaborazione di coordinate gps,

per poter alleviare il carico di lavoro da Rails. Allo stesso scopo il servizio si appoggia su molteplici server. Uno di questi server è Amazon Web Services, un servizio di cloud computing che oltre al noleggio di un server tradizionale rende possibile anche un architettura a microservizi.

### 1.3 Microservizi

Un servizio è un processo che: esegue specifiche operazioni autonomamente, risponde a eventi oppure rimane in attesa di una richiesta. Nel caso in cui queste operazioni vengano svolte continuamente, il servizio è estremamente vantaggioso. Non è però necessario che il servizio sia sempre in esecuzione se viene usato in maniera discontinua o per brevi periodi di tempo. I microservizi coprono questo ruolo, hanno le stesse caratteristiche di un servizio ma eseguono solo su richiesta.

L'ambiente di esecuzione è completamente gestito da AWS, l'unica requisito per creare un microservizio è caricare il proprio codice. In questo caso viene noleggiato il tempo di calcolo invece che una macchina fisica o virtuale. L'ambiente viene creato al momento della richiesta, esegue il codice e cessa di esistere. Quando il microservizio non è attivo non ci sono costi.

È bene tenere in mente due importanti caratteristiche dei microservizi:

- L'ambiente non ha spazio di archiviazione, qualora sia necessario salvare un file, bisogna caricarlo in un servizio di cloud computing come S3
- Le tecnologie devono essere compatibili con l'infrastruttura sottostante, il che limita le nostre scelte

Avendo in mente queste considerazioni, un caso d'uso adatto ai microservizi è un servizio API che viene usato in maniera occasionale oppure per brevi periodi fissi.

## 1.4 Sistema automatico di generazione di report

Engim esegue una manutenzione annuale di database, che consiste nell'archiviazione dei dati. Questi possono essere salvati dal cliente, nel caso fosse interessato o necessitato, sotto forma di pdf oppure xls.

Le informazioni più importanti sono l'elenco e le specifiche di tutte le "attività". Un'attività contiene una serie di dati, tra cui: coordinate gps, costi di lavoro, tempo di lavoro e altro. Al momento il servizio è implementato da Rails tramite una gemma di ruby. Il sistema attuale crea un'istanza di chrome, l'istanza contiene un HTML che si desidera convertire in pdf e infine avviene il parsing del documento.

Questo ha una serie di gravi problemi:

- La necessità di avviare un'istanza di Chrome e il parsing di un HTML è estremamente costoso dal punto di vista delle risorse
- Il parsing di un HTML è anche estremamente costoso in termini di tempo, aggravato dalle lunghe query dovute alla grande mole di dati
- Il parsing tende a essere poco affidabile

La natura del nostro problema rende molto facile la scelta di un microservizio. L'operazione è ripetitiva, ben definita e usata per brevi periodi. Altri vantaggi importanti sono il risparmio di risorse del server, che evita di gra-

vare sulle operazioni più critiche, e la possibilità di usare il microservizio per qualsiasi altro prodotto Engim.

## **2 Requisiti del progetto**

### **2.1 Descrizione**

Il progetto è una funzione lambda su AWS, ovvero un microservizio. La funzione viene chiamata in maniera diretta tramite una gemma di ruby. Questa richiede le credenziali IAM per effettuare l'accesso alla lambda e prende in input un JSON. Al suo interno abbiamo un token di autenticazione e il dominio di provenienza che vengono controllati dalla lambda.

I contenuti invece sono divisi in sezioni per permettere dinamicità di stampa, qualora uno dei blocchi fosse assente, semplicemente non verrà stampato.

Nel caso in cui la stampa sia avvenuta correttamente la funzione ritorna "200 OK" e il nome del file su S3. Se l'input della chiamata risulta errato, la funzione ritorna "400 BAD REQUEST" e un messaggio che descrive l'errore. Infine se avviene un errore di connessione al bucket S3, la funzione ritornerà un errore "500 INTERNAL ERROR".

Il progetto deve permettere l'implementazione di stampe diverse da quelle di servizioGPS e di altri file di output come KML e xls.

### **2.2 Sicurezza**

L'autenticazione avviene a livello di codice, sia sul server che sul microservizio.

La gemma `aws-sdk-lambda` permette di stabilire una connessione diretta con la lambda. È sufficiente fornire: nome della funzione, regione, credenziali e payload. Le credenziali sono salvati dentro un file crittografato yaml. Un ulteriore livello di sicurezza è integrato nella funzione sotto forma di token e whitelist.

Questi valori sono salvati nel ambiente di AWS, per evitare di averli scritti in chiaro nel codice. Il token è un lungo carattere alfanumerico.

La whitelist contiene tutti i domini di Engim, questi vengono caricati in una lista e confrontati con il mittente. Nel caso in cui l'operazione sia andato a buon fine, ritornerà un JSON in risposta.

Qualsiasi chiamata da dominio esterni oppure senza token verrà tratta come "400 BAD REQUEST".

## 2.3 Correttezza dei dati

Esistono 3 blocchi principali: "header", "body", "table". Fin tanto che almeno uno dei 3 è presente, la stampa risulta valida. Facoltativamente è possibile includere il blocco "media", che contiene eventuali immagini in "base64". Questo blocco evita la ripetizione delle immagini, alleggerendo l'invio dei dati.

Il header può contenere fino a 2 loghi, però deve necessariamente avere un titolo e una intestazione. Uno dei due loghi è sempre quello aziendale, mentre l'altro può essere una copia del precedente oppure appartenere al cliente. L'intestazione racchiude: dominio di provenienza, nome utente e data di oggi.

Il body può avere un numero arbitrario di blocchi. Ogni blocco è racchiuso

in un rettangolo e anche esso ha una serie di titoli con i loro dati. È valida l'assenza di un dato, ma invalida l'assenza di un titolo.

Il table è composto da una serie di titoli e una serie di righe. Non c'è limite al numero di righe che può contenere. Anche in questo caso non può mancare un titolo ma un dato può essere vuoto.

Tra i tipo di dato, di una tabella, è possibile avere un immagine che viene spedita in base64, se è assente non verrà scritto nulla. Onde evitare la duplicazione eccessiva di immagini, siccome una tabella potenzialmente contiene un numero illimitato di dati, le immagini sono salvate nel blocco media in singola coppia.

Il programma presume che i dati siano corretti, controlla solo la loro presenza. La funzione anche presume che ogni dato sia una stringa, questo evita conversioni e permette di stampare qualsiasi dato a prescindere dalla provenienza.

## 2.4 Problemi interni del server

Durante l'esecuzione vengono effettuate delle connessioni dirette tra i servizi. In un primo momento Rails si collega con la lambda, questa connessione potrebbe essere soggetta a timeout.

Lambda a sua volta crea un canale con il bucket, per poter effettuare il caricamento del file, qualsiasi tipo di errore viene catturato da javascript. Nel caso in cui il file venga salvato correttamente, la lambda restituisce la chiave del file, ovvero il suo nome.

Infine Rails si collega al bucket e cerca il link per aprire o scaricare il file direttamente dal bucket. Questa connessione può sembrare superflua, ma

aggiunge un livello di sicurezza ulteriore senza aumenti degni di nota sul costo temporale.

## 2.5 Presupposti e dipendenze

Siccome AWS mette a disposizione l'infrastruttura c'è una scelta limitata dei linguaggi di programmazione. Nella lista sono presenti i più popolari al momento, in particolare sono stati presi sotto esame python, ruby e javascript. Ruby ha numerose librerie che permettono la scrittura dei PDF che purtroppo prevedono il parsing di un HTML. Questa operazione è molto vantaggiosa per lo sviluppo ma non particolarmente per l'efficienza. Ogni conversione richiede un'istanza di chrome da cui convertire il file, creando una serie di processi non necessari.

Python offre molteplici librerie, ognuna con diverse funzionalità. Non è sufficiente avere una sola per le operazioni e inoltre molte di queste librerie non sono mantenute in maniera costante.

Javascript invece ha "pdfkit", una libreria che permette la creazione e manipolazione di un PDF senza intermezzi. È molto popolare, open source con una comunità attiva, oltre ad avere una documentazione breve e chiara.

La stampa ha una forma regolare, divisa per righe e colonne. Se i dati sono disposti in maniera più arbitraria è necessaria un'implementazione diversa. Non è richiesto la stampa in altri formati di file, ma è necessario lasciare la libertà di aggiungere diversi formati in futuro.

## 2.6 Features

La lambda ha un'unica funzionalità, stampa e salvataggio della stampa.



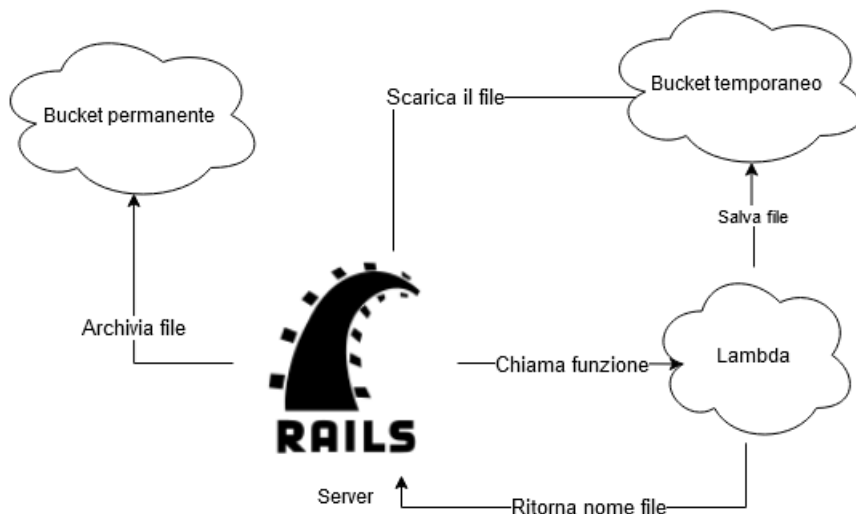


Figura 2: Casi d'uso

Ogni stampa deve seguire lo schema a blocchi definito, non è possibile ad esempio affiancare un'immagine in una tabella. Qualsiasi implementazione che non segua lo standard va aggiunta alla funzione.

È possibile stampare immagini. Queste devono essere prese in input siccome non esiste una memoria permanente per la funzione.

## 3 Implementazione

### 3.1 Flusso di lavoro

È stato deciso di lavorare con continua integrazione.

In prima battuta creare una funzione in grado di generare un PDF e salvarlo su S3. Testare ogni aspetto della funzione ed effettuare il deploy su AWS.

Successivamente integrare la funzionalità in rails, eliminando la vecchia fun-

zione. Assicurarsi che la stampa esegue correttamente ed in tempi ragionevoli.

### **3.2 Creazione di un file e salvataggio su S3**

È possibile interagire con un bucket S3 tramite due modi: connessione con link creato da un utente autorizzato oppure connessione IAM. In questo particolare caso è necessario stabilire una connessione diretta tramite IAM. Una volta stabilita è possibile accedere a una serie di metodi tra cui "upload()". Al termine del caricamento il bucket ritorna un dizionario con una serie di informazioni, la più importante è la chiave, che è il nome del file su S3. Nota importante, non usare "key" perché è un valore vecchio per retrocompatibilità, bensì "Key".

### **3.3 Generazione di un PDF da un JSON**

La libreria permette scrittura sul file, disegno elementare e manipolazione di font, posizione e colore. Inizialmente appariva conveniente un approccio funzionale, ma vedendo il modo particolare in cui viene esportato un file in "Nodejs" e la complessità sempre maggiore dei documenti, è stato deciso di creare una classe: PdfDocument.

Il costruttore va a definire la grandezza del documento in base al blocco più largo del file. Questa definisce la larghezza del documento e l'altezza viene calcolata di conseguenza (1.41 volte più grande, come un foglio A4). Il formato default è A4, per evitare stampe troppo piccole. Mantenendo queste proporzioni la visualizzazione e la stampa risultano molto semplici. Una volta definito il documento la classe chiama il metodo "writeData()".

Questo controlla la presenza dei dati e richiama altri metodi di scrittura se necessario. I metodi di scrittura controllano che non manchino dati vitali e se così non fosse scrivono sul documento. È possibile richiedere il documento tramite "getDocument()".

Il file è uno stream di dati e può essere spedito direttamente su S3.

### **3.4 Test automatizzati**

Si è deciso di effettuare del "unit testing" tramite la libreria mocha. Ogni singolo modulo è testato singolarmente con dati corretti ed errati. Ogni input è generato casualmente grazie a "faker-js". Infine le connessioni con AWS sono simulate tramite "aws-sdk-dev".

### **3.5 Ruby on Rails**

Il metodo di stampa corrente ritorna una pagina HTML e richiede la stampa oppure scarica la conversione di quella pagina. Innanzitutto è necessario cambiare il ritorno da HTML a PDF e in seguito passare un valore alla chiamata per distinguere la visualizzazione dallo scaricamento del file. La prossima fase è caricare tutti i dati in un hash e trasformarlo in un JSON. Ogni stampa ha disposizioni simili ma una conversione dei dati diversa, è quindi necessario creare due distinte funzioni: una per le attività e una per la singola attività

Una volta definito il JSON si può stabilire una connessione diretta con la lambda e chiamare il metodo interessato. Al termine della chiamata il metodo ritornerà il nome del file e stabiliamo un'ulteriore connessione, stavolta col bucket, per visualizzare o scaricare il file. Il ritorno di questa richiesta è un

JSON che contiene una serie di link tra cui il url di download e il url di visualizzazione. Il motivo per cui questo non viene ritornato direttamente dalla lambda è per avere un livello di sicurezza extra.

## 4 Applicazione e performance

### 4.1 Risultati

Non è stato necessario cambiare nessuna parte dell'interfaccia grafica. È possibile stampare una lista di attività cliccando sulla stampante oppure una scaricarle cliccando sul immagine alla sua destra.

ID	Status	First position time	Last position time	Activity type	Device	User	Economy	Total service
4		Wed 28 Dec 2022, 15:51:38	Wed 28 Dec 2022, 15:51:38	Tracking	123	admin	Englin	0s 0.0 km 15.00 km/h
5		Wed 28 Dec 2022, 15:51:38	Wed 28 Dec 2022, 15:51:38	Tracking	123	admin	Englin	0s 0.0 km 15.00 km/h
6		Wed 08 Mar 2023, 00:00:00	Mon 13 Mar 2023, 11:10:00	Tracking	123	admin	Englin	5d 11h 5m 59s 0.0 km 15.00 km/h
1		Wed 28 Dec 2022, 15:51:38	Wed 28 Dec 2022, 15:51:38	Tracking	123	admin	Englin	0s 0.0 km 15.00 km/h
2		Wed 28 Dec 2022, 15:51:38	Wed 28 Dec 2022, 15:51:38	Tracking	123	admin	Englin	0s 0.0 km 15.00 km/h
3		Wed 28 Dec 2022, 15:51:38	Wed 28 Dec 2022, 15:51:38	Tracking	123	admin	Englin	0s 0.0 km 15.00 km/h

Figura 3: Lista di attività

Di seguito è possibile vedere un esempio di stampa di attività.



Activities: 6

First activity time	Last activity time	Calculated distance	Total service
Wed 28 Dec 2022, 15:51:38	Wed 08 Mar 2023, 00:00:00	15.0 km	131.17 h
Not reduced km cost	Not reduced time cost	Not reduced time cost	
0.0 €	0.0 €	25.0 €	

Id	Status	First position time	Activity type	Device	User	Company	Total service
4		Wed 28 Dec 2022, 15:51:38	Tracking	123	admin	Engim	0.0 min - 5.0 km
5		Wed 28 Dec 2022, 15:51:38	Tracking	123	admin	Engim	0.0 min - 5.0 km
6		Wed 08 Mar 2023, 00:00:00	Tracking	123	admin	Engim	0.0 min - 0.0 km
1		Wed 28 Dec 2022, 15:51:38	Tracking	123	admin	Engim	0.0 min - 5.0 km
2		Wed 28 Dec 2022, 15:51:38	Tracking	123	admin	Engim	0.0 min - 5.0 km
3		Wed 28 Dec 2022, 15:51:38	Tracking	123	admin	Engim	0.0 min - 5.0 km

Figura 4: Stampa del elenco attività

Alternativamente è possibile stampare i dettagli di un unica attività.

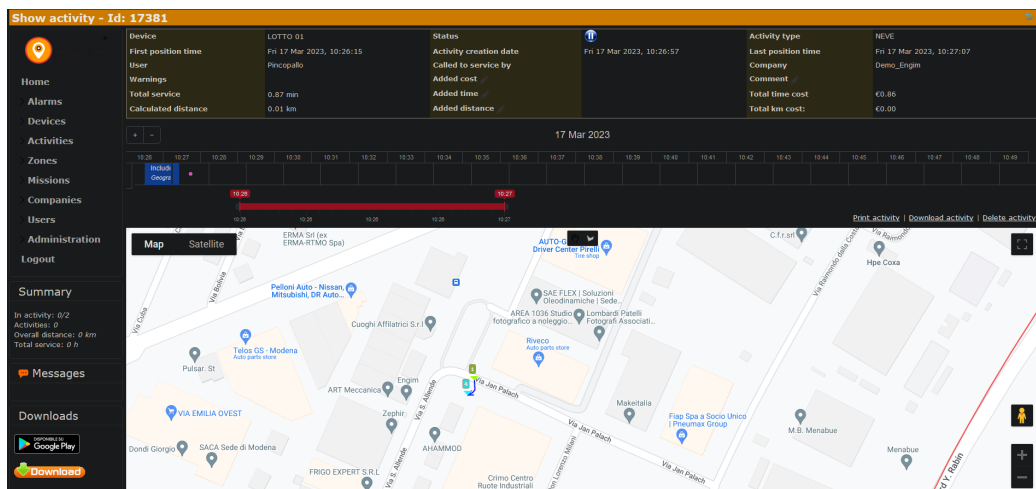




Figura 5: Attività visualizzazione

Esempio di stampa di un attività



Comune di Modena

# Activity report



Id	First position time	Last position time	Comment
1	2022-12-28 15:51:38 +0100	2022-12-28 15:51:38 +0100	

User	Device	IMEI	Called to service by
	123	122456789012345	admin

Total service
0s

Calculated distance
15.0 km of 0.0 km +5.00 km

Activity type	Total km cost	Total time cost
Tracking	0.0 €	0.0 €

Incl.	Position time	Duration	Address	Distance	Km/h
Included	2022-10-19 09:16:54 +0200	1 of 1 min	Via IV Novembre 1	0 of 0 m	min:0 mean:0 max:0

Figura 6: Stampa di una singola attività

## 4.2 Performance

Sono stati fatti test di performance sul elenco delle attività siccome è la stampa più impegnativa, perché contiene molte immagini.

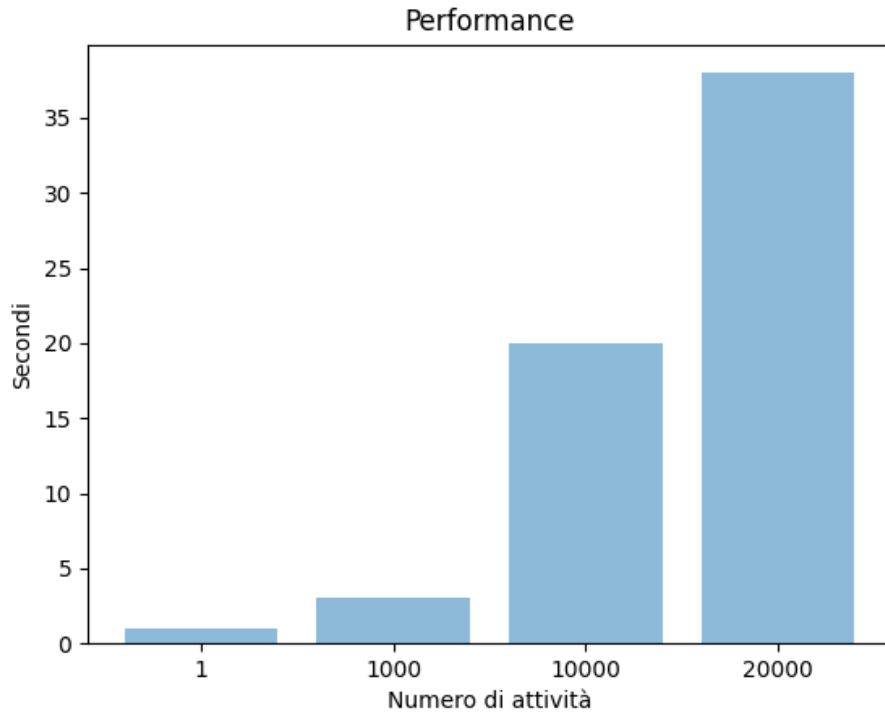


Figura 7: Performance

Al aumentare delle attività aumenta in maniera quasi lineare il tempo. Questo è dovuto al numero alto di cicli di scrittura. Oltre venti mila attività non è possibile effettuare una stampa perché raggiungiamo il limite del buffer di AWS. Se fosse necessario aumentare la performance sarebbe sufficiente trasformare le immagini in icone vettoriali.

Altre possibile ottimizzazione potrebbero essere: definire a priori le misure del foglio e salvarle come costanti oppure modificare il JSON per semplificare la struttura dati. Tuttavia non è necessario perché sono tempi ragionevoli. È più importante notare che il server usi risorse minime, in quanto deve solo



aspettare il ritorno della chiamata `https`. Questo era il fulcro del progetto, permettere una migliore e più sicura navigazione alleggerendo il lavoro compiuto dal software.

L'implementazione precedente aveva una performance ottima per un numero ridotto di informazioni, siccome non era legata a dei tempi fissi di connessioni. Allo stesso tempo al crescere delle attività aumentava esponenzialmente il tempo e rendeva alto il rischio di crash.

## Conclusione

I microservizi offrono un nuovo paradigma di programmazione, uno dove è possibile astrarre logica dal proprio software e eseguirla solo qual'ora fosse necessario. È possibile condividere questa logica su diverse piattaforme, e permette una scalabilità semplice e veloce. Naturalmente non è adatto a ogni situazione. Una serie di operazioni numerose o estremamente lunghe richiedono un servizio. Nel caso studiato invece una lambda è estremamente vantaggiosa.

In primo luogo risolve il problema del sovraccarico del server. Migliorare la logica e gli algoritmi sarebbe stato corretto ma non avrebbe alleggerito il carico di lavoro. L'esportazione del servizio invece permette di mantenere massima performance del sito siccome i calcoli vengono fatti altrove.

Il costo temporale non ha subito miglie in miglioramento nei casi più semplici, ma cresce in maniera lineare all'aumentare della mole di lavoro. Questo è molto più importante, perché ci garantisce usabilità anche nei casi più estremi, visto che non è poco comune avere una grossa mole di dati.

Il progetto si era concentrato sulle stampe più impegnative, siccome erano quelle responsabili del impatto negativo sul software e siccome sono le più comuni. Non sono però le uniche stampe presenti sul server. Esistono varie informazioni da stampare che sono meno vitali e meno usate. È però corretto avere una logica unica per tutte e portarla su lambda.

Inoltre non è possibile scaricare solo in formato PDF, bensì anche in XLSX e KML. È possibile introdurre queste tipologie in futuro, naturalmente ognuno dovrà avere una sua implementazione personale. Cosicché ogni operazione

di stampa è estratta dal server ed è in un unico progetto centrale. Questo permetterebbe al Rails di concentrarsi esclusivamente sull'esposizione dei dati ai clienti.

# Bibliografia

## Riferimenti bibliografici

- [1] AWS documentazione. <https://docs.aws.amazon.com/lambda/index.html>
- [2] Rails documentazione. <https://api.rubyonrails.org/>
- [3] Nodejs documentazione. <https://nodejs.org/it/docs>
- [4] Mozilla documentazione. <https://developer.mozilla.org/en-US/>
- [5] Pdf-kit documentazione. <https://pdfkit.org/>
- [6] Microservizi wiki. <https://en.wikipedia.org/wiki/Microservices>