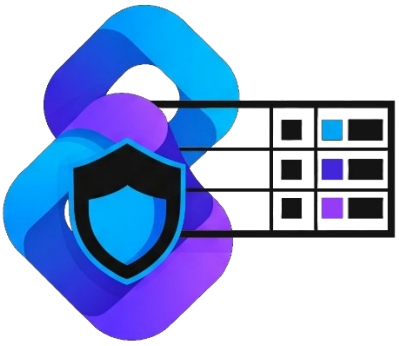

AnonimaData

Pietro Coloretti – Leonardo Gennaioli – Iacopo Sbalchiero

Scalable and Reliable Services – Ingegneria Informatica M
Università degli studi di Bologna - A.A. 2024-2025

Sommario

Abstract

AnonimaData ha avuto l'obiettivo di progettare e implementare un servizio scalabile e affidabile per l'anonimizzazione di dataset, garantendo la protezione della privacy dei dati sensibili. La piattaforma permette agli utenti di caricare dataset in formato CSV o JSON e di applicare algoritmi di anonimizzazione all'avanguardia come k-anonymity, l-diversity e differential privacy, configurandone dinamicamente i parametri.

AnonimaData è stato concepito per essere generalizzabile a dataset tabulari arbitrari, supportando schemi, tipi di dati e configurazioni di colonne diverse. Il servizio offre un'interfaccia utente web intuitiva che facilita la gestione del processo, con incluse funzionalità che mostrano l'anteprima del dataset anonimizzato. I dati anonimizzati vengono archiviati sia in un database per un accesso strutturato sia come file CSV nello storage a oggetti, garantendo disponibilità e facilità di download.

L'intera infrastruttura del sistema è stata distribuita su Google Cloud Platform (GCP), con la gestione e il provisioning delle risorse interamente automatizzati tramite Terraform, assicurando scalabilità, affidabilità e riproducibilità. Infine, il servizio integra un sistema di autenticazione utente (Google OAuth 2.0) per garantire un accesso sicuro e controllato. AnonimaData rappresenta una soluzione completa per la gestione della privacy dei dati, fornendo uno strumento robusto e flessibile per la conformità normativa e la protezione delle informazioni personali.

Algoritmi di anonimizzazione

Per comprendere il funzionamento degli algoritmi di anonimizzazione utilizzati da AnonimaData, è essenziale analizzare la natura degli attributi presenti nei dataset. Gli algoritmi operano infatti sulla differenza semantica e funzionale tra alcuni tipi di attributi, in particolare:

Quasi-Identificatori

I *quasi-identificatori* sono attributi che, presi singolarmente, non identificano necessariamente un individuo, ma che, se combinati tra loro o incrociati con fonti esterne, possono consentire la re-identificazione e rappresentano un rischio indiretto per la privacy (età, CAP, data di nascita, genere, professione).

Attributi Sensibili

Gli *attributi sensibili* contengono informazioni personali e riservate che, se esposte, possono comportare danni reputazionali, legali o psicologici per l'individuo, anche quando l'identità non è nota con certezza (diagnosi medica, salario, orientamento religioso o politico).

A partire da questa distinzione, di seguito vengono descritti gli algoritmi supportati dalla piattaforma:

K-Anonymity

K-anonymity garantisce che ogni record nel dataset sia indistinguibile da almeno altri $k-1$ rispetto a un insieme di quasi-identificatori. Si ottiene tramite generalizzazione e/o soppressione dei dati.

Vantaggi

- Semplice da comprendere e implementare.
- Preserva una buona quantità di dati utili, specialmente per valori elevati di k.
- Compatibile con una vasta gamma di dataset tabulari.

Svantaggi

- Non protegge contro inferenze se all'interno di un gruppo i dati sensibili sono omogenei.
- Vulnerabile a chi possiede conoscenze pregresse.
- Può richiedere eccessiva generalizzazione per dataset con outlier o alta cardinalità.

L-Diversity

Estende la k-anonymity richiedendo che ogni gruppo contenga almeno *l* valori distinti (o sufficientemente variegati) dell'attributo sensibile.

Vantaggi

- Migliore protezione rispetto a k-anonymity sui dati sensibili.
- Rafforza la diversità dei dati all'interno dei gruppi anonimizzati.
- Esistono diverse varianti che si adattano a contesti differenti.

Svantaggi

- Può essere difficile da ottenere su dataset con attributi sensibili poco vari.
- Maggior perdita di utilità del dato rispetto a k-anonymity.
- Non impedisce completamente la ricostruzione della distribuzione dei dati sensibili.

Differential Privacy

Fornisce una garanzia formale di privacy aggiungendo rumore casuale alle analisi o trasformazioni sui dati, rendendo trascurabile l'effetto di ogni singolo individuo sui risultati. È controllata da un parametro di privacy ε (epsilon).

Vantaggi

- Solida base matematica indipendente dal contenuto del dataset.
- Adatta a sistemi che forniscono risposte aggregate (es. API, report, query).
- Impedisce attacchi anche in presenza di conoscenze esterne.

Svantaggi

- Introduce rumore: la qualità dei risultati può degradare significativamente per bassi valori di ε.
- Richiede un'attenta calibrazione per bilanciare privacy e utility.
- Meno intuitivo rispetto ad altri modelli per utenti non esperti.

Requisiti Funzionali

Le funzionalità principali che il sistema deve fornire agli utenti includono:

- **Caricamento Dataset:** Gli utenti devono poter caricare dataset in formato **CSV** e **JSON**.
- **Selezione Algoritmo di Anonimizzazione:** Il sistema deve presentare una lista di algoritmi di anonimizzazione disponibili, tra cui:
  - k-anonimato
  - l-diversità
  - Privacy Differenziale
- **Configurazione Parametri Algoritmo:** Per ogni algoritmo selezionato, gli utenti devono poter configurare dinamicamente i parametri specifici (es. valore di *k* per k-anonimato, *epsilon* per la Privacy Differenziale).
- **Anteprima Dataset Anonimizzato:** Il sistema deve permettere agli utenti di visualizzare un'anteprima del dataset anonimizzato prima del salvataggio definitivo.
- **Archiviazione Dataset Anonimizzato:** I dataset anonimizzati devono essere archiviati in due modalità distinte:
  - **In un database:** Per consentire interrogazioni e gestione strutturata.
  - **Come file CSV:** Archiviati in uno storage a oggetti (es. Google Cloud Storage) per download e riutilizzo.
- **Interfaccia Utente Web:** Deve essere disponibile un'interfaccia web intuitiva e user-friendly per gestire l'intero processo di anonimizzazione.
- **Autenticazione Utente:** Il sistema deve supportare l'autenticazione degli utenti tramite un provider esterno **(Google OAuth 2.0)**.
- **Generalizzabilità Dataset:** La piattaforma deve essere in grado di elaborare dataset tabulari arbitrari, supportando diversi schemi, tipi di dati e configurazioni di colonne senza richiedere modifiche al codice.

Requisiti Non Funzionali

Il sistema deve rispettare alcuni vincoli principali, oltre che avere determinate qualità:

- **Scalabilità:** Il servizio deve essere in grado di gestire un aumento del numero di utenti e di dataset di grandi dimensioni senza degrado significativo delle prestazioni. Questo implica l'utilizzo di servizi cloud scalabili e architetture distribuite.
- **Affidabilità:** Il sistema deve essere resiliente ai guasti e garantire la disponibilità continua del servizio. I dati devono essere archiviati in modo ridondante e devono essere previsti meccanismi di recupero in caso di errore.
- **Performance:** Il tempo di elaborazione per l'anonimizzazione dei dataset deve essere ragionevole, anche per dataset di grandi dimensioni. L'interfaccia utente deve essere reattiva.
- Sicurezza:
    - **Protezione dei Dati:** Tutti i dati, sia quelli originali che quelli anonimizzati, devono essere protetti da accessi non autorizzati.
    - **Crittografia:** I dati sensibili devono essere crittografati sia a riposo che in transito.
    - **Conformità Privacy:** Il sistema deve aderire ai principi di protezione della privacy, garantendo che i dati anonimizzati non possano essere re-identificati.
- **Usabilità:** L'interfaccia utente web deve essere intuitiva, facile da navigare e offrire una buona esperienza d'uso agli utenti.
- **Deployabilità:** L'intero sistema deve essere completamente distribuibile su Google Cloud Platform (GCP) e la gestione dell'infrastruttura deve essere automatizzata tramite **Terraform**.
- **Robustezza:** Il sistema deve gestire errori e input non validi, fornendo feedback chiari all'utente.
- **Portabilità (Dati):** I dataset anonimizzati scaricabili (CSV) devono essere in un formato standard e facilmente importabile in altre applicazioni.

Tecnologie Utilizzate

Google Cloud Platform (GCP)

L'intero sistema è progettato per essere distribuito su GCP, fornendo l'infrastruttura sottostante necessaria per la scalabilità, l'affidabilità e la gestione dei servizi. GCP offre un'ampia gamma di servizi, inclusi calcolo (es. Cloud Run per i microservizi), storage (Cloud Storage per i file CSV e un database per i metadati/dati anonimizzati), e messaggistica (Pub/Sub) che sono fondamentali per l'interconnessione dei componenti del backend.

Terraform

Per la gestione e il provisioning dell'infrastruttura su GCP, viene utilizzato Terraform. Questo strumento di Infrastructure as Code (IaC) permette di definire, versionare e deployare l'intera infrastruttura cloud in modo dichiarativo e automatizzato, garantendo coerenza, riproducibilità e facilità di gestione dell'ambiente.

Docker

Ogni servizio è containerizzato utilizzando Docker. Questo permette di isolare l'ambiente di esecuzione di ciascun servizio, garantendo che le dipendenze siano gestite in modo coerente e che i servizi possano essere deployati in qualsiasi ambiente compatibile con Docker.

Python (e Flask per API REST)

Il linguaggio di programmazione principale per lo sviluppo dei servizi backend è Python. In particolare, il framework Flask viene utilizzato per costruire le API REST che espongono le funzionalità dei servizi (come l'upload di file, la richiesta di anonimizzazione e il recupero dello stato). Python, con le sue librerie ricche per la manipolazione dei dati (es. Pandas per i dataset), è ideale per le operazioni di analisi e anonimizzazione dei dati.

React (per Frontend)

React è una scelta popolare per la costruzione di Single Page Applications (SPA) dinamiche e interattive, ideale per la gestione del processo di upload, configurazione degli algoritmi e visualizzazione dei risultati. Comunica con il backend tramite le API REST esposte da Flask.

Firebase

Firebase è utilizzato per l'autenticazione degli utenti, in particolare tramite **Firebase Authentication (con Google OAuth 2.0)**. L'utilizzo di Firebase permette di delegare la gestione dell'identità e dell'autorizzazione a un servizio robusto e scalabile.

Architettura del Sistema

L'architettura di AnonimaData è modulare e distribuita, basata su microservizi che comunicano principalmente tramite Google Pub/Sub. Questa impostazione garantisce scalabilità, resilienza e manutenibilità. I componenti chiave includono il Frontend, l'Orchestratore, il Formatter e l'Anonymizer. Uno schema riassuntivo è visibile in Figura 1.

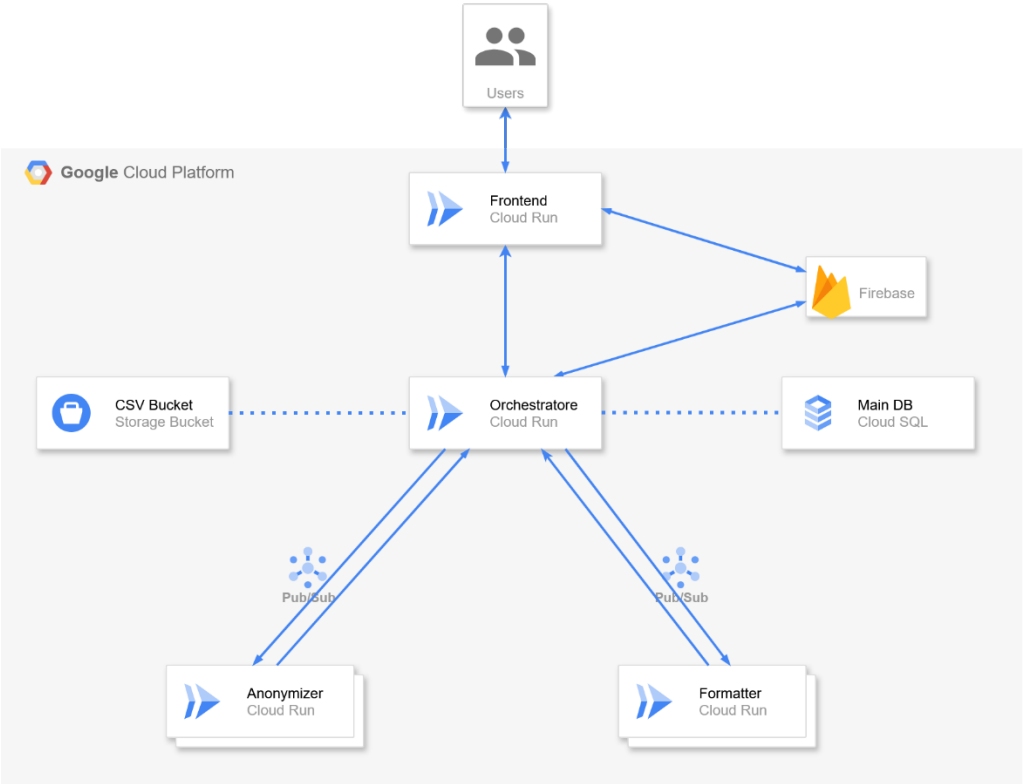

Figura 1 – Schema dell'architettura di AnonimaData

Orchestratore

L'Orchestratore è il cuore del backend di AnonimaData e si pone come punto di ingresso principale per le interazioni degli utenti e la gestione del flusso di lavoro complessivo. Le sue responsabilità principali sono le seguenti:

- **Interfaccia con il Frontend (API REST):** L'Orchestratore espone una serie di endpoint API REST che consentono al frontend di interagire con il sistema. Questi includono funzionalità per:
    - **Upload e avvio analisi (*/upload_and_analyze*):** Riceve i file caricati dagli utenti, li codifica in Base64 e pubblica un messaggio sul topic *DATA_UPLOAD_REQUESTS* di Pub/Sub per avviare il processo di analisi da parte del Formatter.

  - **Recupero dello stato delle operazioni (*/get_status/<job_id>*):** Permette al frontend di interrogare lo stato di un lavoro specifico (analisi o anonimizzazione) tramite un *job_id*.
  - **Richiesta di anonimizzazione (*/request_anonymization*):** Riceve le configurazioni e i parametri scelti dall'utente per l'anonimizzazione (metodo, parametri, selezioni colonne), recupera i dati preprocessati e i metadati associati al *job_id*, e pubblica un messaggio sul topic *ANONYMIZATION_REQUESTS* di Pub/Sub, destinato all'Anonymizer.
  - **Recupero elenco file e statistiche *(/get_files*):** Fornisce al frontend una lista dei file caricati dall'utente, inclusi il loro stato, le dimensioni, le informazioni sull'anonimizzazione e un URL per il download del file completo.
  - **Download file anonimizzati (*/download/<job_id>*):** Gestisce la richiesta di download di un dataset anonimizzato completo, recuperandolo dalla memoria (in questo caso, da un dizionario in-memory *job_status_map*) e inviandolo come file CSV.
  - **Rimozione dei file anonimizzati (*/delete/<job_id>*):** Permette all'utente di eliminare tutti i dati relativi ad un dataset, che esso sia stato anonimizzato o che sia ancora in fase di anonimizzazione.
- **Gestione dello Stato dei Job:** L'Orchestratore comunica con un database PostgreSQL e salva uno stato dettagliato di ogni job (es. *uploaded, analyzed, anonymization_requested, completed, error*), includendo anche l'avanzamento, i dettagli, i timestamp, i dati preprocessati, i metadati, i dati anonimizzati e i campioni anonimizzati.
- **Verifica Permessi Utente (Firebase Authentication):** Ogni richiesta API che implica l'accesso a risorse utente è protetta da un decoratore *@firebase_auth_required*. Questo decoratore estrae il token di autenticazione Firebase dall'header Authorization, lo verifica tramite *firebase_admin.auth.verify_id_token()*, e se valido, imposta l'ID utente (*request.user_id*) sulla richiesta. Questo garantisce che solo gli utenti autenticati e autorizzati possano accedere ai propri dati e processi. L'Orchestratore verifica che l'utente che effettua la richiesta sia lo stesso che ha avviato il job.
- **Comunicazione Inter-Servizio (Pub/Sub):** L'Orchestratore funge da hub di comunicazione Pub/Sub. Pubblica messaggi sui topic di Pub/Sub (es. *DATA_UPLOAD_REQUESTS* e *ANONYMIZATION_REQUESTS*) per delegare le operazioni di analisi e anonimizzazione ad altri servizi. Riceve anche notifiche da Pub/Sub su endpoint dedicati (es. */receive_analysis_results, /receive_anonymization_results, /receive_error_notifications*) quando il Formatter o l'Anonymizer completano le loro operazioni o riscontrano errori. Questo approccio asincrono e basato su eventi disaccoppia i servizi, migliorando la scalabilità e la robustezza.
- **Gestione Errori:** L'Orchestratore è configurato per ricevere notifiche di errore dal Pub/Sub (tramite il topic *ERROR_NOTIFICATIONS* e l'endpoint */receive_error_notifications*). In caso di errore in qualsiasi fase (analisi o

anonimizzazione), lo stato del job viene aggiornato a *'error'*, fornendo dettagli sullo stage e il messaggio di errore.

- **Preparazione Dati per l'Anonimizzazione:** Quando riceve una richiesta di anonimizzazione, l'Orchestratore recupera i DataFrame di Pandas contenenti i dati preprocessati e i metadati precedentemente salvati all'interno del database. Questi DataFrame vengono convertiti in stringhe CSV/JSON e codificati in Base64 prima di essere inviati come payload nel messaggio Pub/Sub all'Anonymizer. Questo assicura che l'Anonymizer riceva i dati necessari in un formato pronto per l'elaborazione.
- **Persistenza Dati (PostgreSQL e Google Cloud Storage):** L'Orchestratore è responsabile della gestione dello stato persistente di tutti i job di anonimizzazione e dei dati correlati. Nel nostro ambiente di produzione, la persistenza dei dati è gestita tramite database relazionale (PostgreSQL) e Storage ad Oggetti (Google Cloud Storage).

È stata presa la decisione strategica di adottare delle API REST pubbliche per l'interazione con il sistema. Questa scelta nasce dall'esigenza di garantire la massima flessibilità e futuro sviluppo, permettendo a eventuali evoluzioni o integrazioni di terze parti di interfacciarsi direttamente con l'orchestratore centrale, senza vincoli o dipendenze da componenti specifici dell'attuale architettura. L'intera struttura del sistema è gestita attraverso sei API principali, ciascuna dedicata a una funzionalità fondamentale: caricamento dei file (upload), invio e gestione dei parametri di anonimizzazione, ottenimento di informazioni dettagliate sul singolo job, recupero dell'elenco completo di tutti i job, download dei dataset anonimizzati ed eliminazione dei dataset non più necessari. Questa organizzazione modulare e standardizzata delle interfacce consente non solo una gestione ordinata e scalabile delle operazioni, ma anche una maggiore semplicità nell'estendere o adattare il sistema secondo nuove esigenze future.

Formatter

Il Formatter è un componente cruciale dell'architettura che si attiva dopo il caricamento iniziale di un dataset. Il suo compito principale è duplice: **standardizzazione del formato e analisi approfondita del dataset**.

- **Ingresso Dati (Pub/Sub):** Il Formatter non riceve le richieste direttamente dal frontend, ma si sottoscrive a un topic Pub/Sub. Quando l'Orchestratore riceve un nuovo upload di dataset, pubblica un messaggio su questo topic contenente l'ID del job, il nome del file e il contenuto del file codificato in Base64. Il Formatter, ricevendo questo messaggio tramite un endpoint push, decodifica il payload e inizia l'elaborazione.
- Standardizzazione del Dataset:
    - Il servizio è progettato per gestire dati in vari formati iniziali.

    - La funzione *read_dataset_for_web* è responsabile di leggere il contenuto del file (che l'Orchestratore ha inviato codificato in Base64) e di trasformarlo in un DataFrame di Pandas. Questo passaggio è fondamentale per standardizzare il formato dei dati, indipendentemente dal formato di input originale, in una struttura uniforme e facilmente manipolabile per le successive fasi.
    - Il dataset viene quindi convertito in un **CSV ben formattato**. Questa conversione interna garantisce che tutti i dati, una volta analizzati, siano in un formato coerente, facilitando il passaggio al servizio Anonymizer.
- Analisi delle Colonne e Rilevamento dei Tipi di Dato:
    - Dopo aver standardizzato il dataset, il Formatter esegue un'analisi approfondita di ogni colonna. La funzione *structure_dataset* è il fulcro di questa operazione.
    - L'obiettivo è identificare automaticamente il tipo di dato presente in ciascuna colonna. Tali tipi di dato possono essere:
        - **Numerico:** Numeri interi, decimali, float.
        - **Testuale:** Stringhe generiche.
        - **Data/Ora:** Date, timestamp, orari.
        - **Categorico:** Campi con un numero limitato di valori discreti (es. sesso, stato civile).
        - **Identificativi Sensibili (Quasi-Identificatori):** Campi che, se combinati, potrebbero portare alla re-identificazione (es. età, CAP, etnia).
        - **Attributi Sensitivi:** Campi che contengono informazioni private (es. salario, diagnosi medica).
        - **Email, Numeri Telefonici, Indirizzi, Codici Fiscali, ecc.:** Riconoscimento di pattern specifici che indicano dati personali altamente sensibili.
- Salvataggio dei Dati Preprocessati e Metadati:
    - Una volta completata l'analisi, il DataFrame preprocessato e i metadati generati (informazioni sulle colonne, tipi di dati, ecc.) vengono salvati.
    - Tali informazioni verranno quindi comunicate all'Orchestratore che procederà al salvataggio in memoria persistente, in particolare all'interno del **Google Cloud Storage** viene salvato il dataset formattato, mentre i metadati pertinenti vengono salvati in **PostgreSQL**, associati all'ID del job.
- **Notifica all'Orchestratore (Pub/Sub):** Dopo aver completato l'elaborazione e aver salvato i dati preprocessati e i metadati, il Formatter pubblica un messaggio su un topic Pub/Sub (*ANALYSIS_RESULTS*). Questo messaggio contiene l'ID del job, lo stato di successo, e le informazioni sui metadati generati. L'Orchestratore si sottoscrive a questo topic per ricevere la notifica e aggiornare lo stato del job, rendendo le informazioni disponibili al frontend.
- **Gestione Errori:** Se si verifica un errore durante l'elaborazione (es. file corrotto, formato non supportato), il Formatter cattura l'eccezione e pubblica un messaggio sul topic *ERROR_NOTIFICATIONS*, consentendo all'Orchestratore di aggiornare lo stato del job a "errore" e notificare l'utente.

Anonymizer

L'Anonymizer è il servizio backend dedicato all'applicazione degli algoritmi di protezione della privacy sui dataset. Riceve le richieste dall'Orchestratore, utilizzando i dati preprocessati e i metadati, genera il dataset anonimizzato finale.

- Ingresso Dati e Richiesta di Anonimizzazione (Pub/Sub):
  - Similmente al Formatter, l'Anonymizer non riceve richieste dirette dal frontend. Si sottoscrive al topic Pub/Sub *ANONYMIZATION_REQUESTS*.
  - Quando l'Orchestratore riceve una richiesta di anonimizzazione dall'utente (incluso l'algoritmo scelto e i suoi parametri), costruisce un messaggio Pub/Sub contenente l'ID del job, il dataset preprocessato (come stringa CSV/JSON codificata in Base64), i metadati associati (anche questi codificati), il metodo di anonimizzazione selezionato e i relativi parametri. Questo messaggio viene poi pubblicato sul topic *ANONYMIZATION_REQUESTS*.
  - L'Anonymizer, tramite un endpoint push, riceve il messaggio, decodifica il payload JSON e ne estrae il *job_id*, il *filename*, il *file_content_base64* (che rappresenta il dataset preprocessato), il *metadata_content_base64*, il *method* e i *params*.
- Processo di Anonimizzazione:
  - Il servizio *anonymization_service.py* invoca il modulo *anonymizer.py* per l'effettiva logica di anonimizzazione, tramite la funzione *process_anonymization*.
  - anonymizer.py è il cuore algoritmico del sistema. Le sue responsabilità includono:
    - **Lettura Dati e Metadati:** Decodifica il contenuto del dataset preprocessato (CSV) e dei metadati, trasformandoli in DataFrame Pandas utilizzabili.
    - **Implementazione degli Algoritmi:** Contiene le implementazioni robuste dei diversi algoritmi di anonimizzazione richiesti.
    - **Utilizzo dei Metadati:** Il modulo *anonymizer.py* fa un uso estensivo dei metadati generati dal Formatter, in particolare i tipi di dato e le indicazioni su quali colonne sono quasi-identificatori o attributi sensibili. La classe *Anonymizer* nel file *anonymizer.py* è inizializzata con il DataFrame e i metadati, e può anche considerare selezioni dell'utente (is_quasi_identifier, should_anonymize), che dovrebbero essere presenti nei metadati estesi. Questo permette una configurazione dinamica e intelligente dell'anonimizzazione.
    - **Generalizzazione e Soppressione:** Gli algoritmi applicano tecniche come la generalizzazione (sostituzione di valori specifici con intervalli o categorie più ampie, es. "20-30 anni" invece di "25 anni") e la soppressione (rimozione completa di valori sensibili o interi record).

        - **Gestione di Diversi Tipi di Dato:** Le implementazioni degli algoritmi sono in grado di trattare correttamente diversi tipi di dato (numerici, categorici, testuali) applicando le tecniche di anonimizzazione appropriate per ciascuno.
    - Salvataggio del Dataset Anonimizzato: Una volta completato il processo di anonimizzazione, il dataset risultante (un nuovo DataFrame Pandas) viene convertito in una stringa CSV per essere inviata all'Orchestratore.
- Notifica all'Orchestratore (Pub/Sub):
    - Al termine dell'anonimizzazione (successo o fallimento), l'Anonymizer pubblica un messaggio sul topic *ANONYMIZATION_RESULTS* (o *ERROR_NOTIFICATIONS* in caso di fallimento).
    - Questo messaggio include il *job_id*, lo stato (es. 'completed'), i dettagli del risultato (es. percorso GCS del file anonimizzato completo, un campione di dati anonimizzati per l'anteprima frontend, le statistiche sull'anonimizzazione), il metodo e i parametri utilizzati, e il timestamp di completamento. L'Orchestratore consumerà questo messaggio per aggiornare lo stato interno del job e notificare il frontend.
- **Gestione Errori:** Come gli altri servizi, l'Anonymizer implementa una robusta gestione degli errori. Qualsiasi eccezione durante il processo di anonimizzazione (es. parametri non validi, problemi di elaborazione dati) viene catturata, e un messaggio di errore viene pubblicato sul topic *ERROR_NOTIFICATIONS*, indirizzando l'errore all'Orchestratore per la notifica all'utente.

Frontend

L'interfaccia frontend della piattaforma è stata realizzata adottando l'architettura Single Page Application (SPA), utilizzando il linguaggio di programmazione JavaScript con il framework React. Questa scelta tecnologica è stata guidata dall'esigenza di offrire un'esperienza utente coerente, fluida e reattiva, riducendo al minimo i tempi di caricamento e garantendo una navigazione dinamica all'interno dell'applicazione, senza la necessità di ricaricare l'intera pagina ad ogni interazione.

L'interfaccia utente, frutto di un'attenta progettazione orientata alla semplicità e all'usabilità, si presenta con una struttura chiara e funzionale, pensata per facilitare l'interazione anche da parte di utenti non esperti. Tutte le principali funzionalità della piattaforma sono immediatamente accessibili, permettendo una rapida comprensione del flusso operativo e un utilizzo efficiente dell'applicazione fin dal primo accesso.

All'apertura del portale, l'utente viene invitato ad autenticarsi utilizzando le proprie credenziali Google. Questo meccanismo di autenticazione centralizzata, basato su OAuth, consente di semplificare la gestione dell'identità digitale, riducendo il numero di credenziali da ricordare e migliorando al contempo la sicurezza generale della piattaforma. Una volta completata con successo la procedura di login, l'utente viene reindirizzato alla dashboard principale dell'applicazione, che costituisce il punto di ingresso per tutte le operazioni successive.

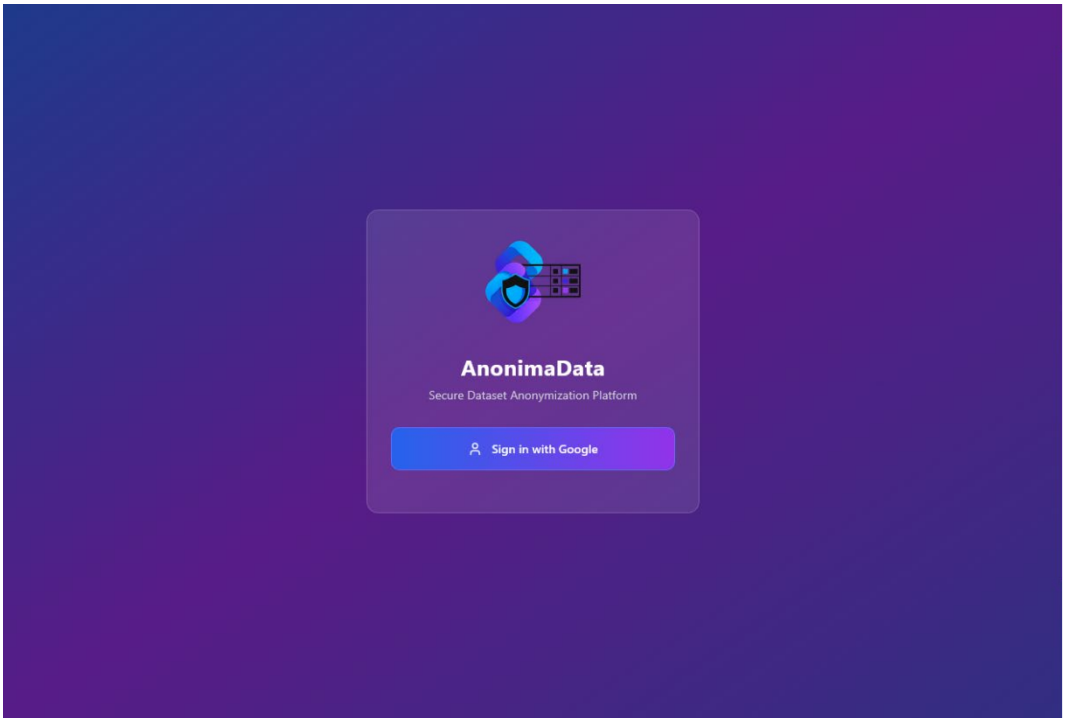

Figura 2 – Pagina di accesso ad AnonimaData

All’interno della dashboard, l’utente ha la possibilità di consultare l’elenco dei dataset precedentemente caricati e gestiti sulla piattaforma oppure avviare un nuovo processo di anonimizzazione. Quest’ultima operazione si avvia tramite la pressione del pulsante “Upload New Dataset”, che rappresenta il punto di partenza per un flusso guidato di caricamento ed elaborazione dei dati.

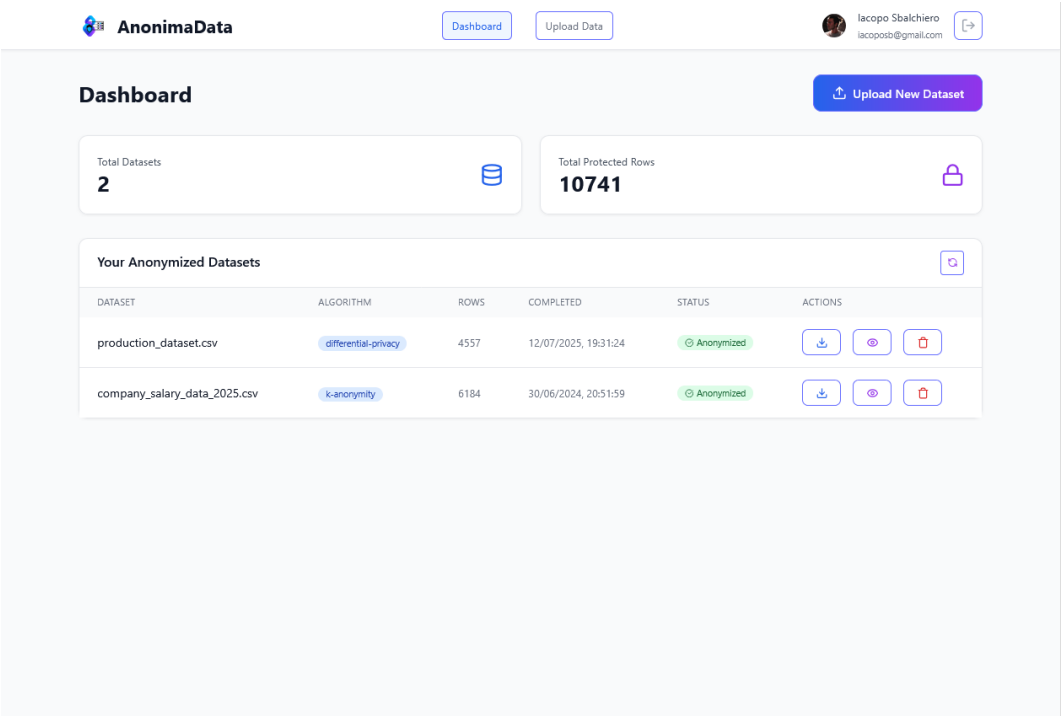

Figura 3 – Dashboard dell’applicazione

Una volta completato l’upload, viene presentata una schermata che consente all’utente di specificare i parametri desiderati per l’anonimizzazione dei dati. Tali parametri possono includere, a titolo esemplificativo, la scelta degli identificatori da mascherare, il livello di generalizzazione da applicare, o la modalità di perturbazione da utilizzare. L’interfaccia guida l’utente attraverso questa configurazione in modo strutturato, rendendo il processo il più chiaro e lineare possibile.

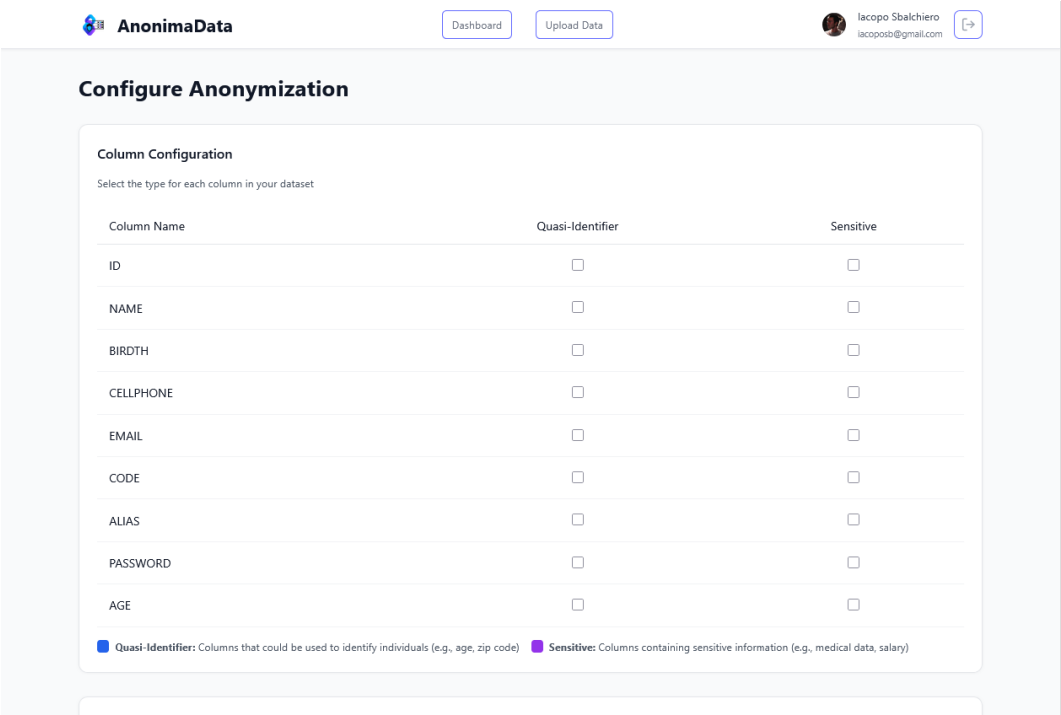

Figura 4 – Schermata di scelta del tipo di dato per colonna

Successivamente all'inserimento dei parametri di anonimizzazione, la piattaforma genera automaticamente un'anteprima parziale del risultato, che viene mostrata a video per consentire una prima verifica visiva da parte dell'utente. In aggiunta, viene fornita la possibilità di scaricare il file completo del dataset anonimizzato, qualora si desideri procedere immediatamente all'esportazione del risultato finale.

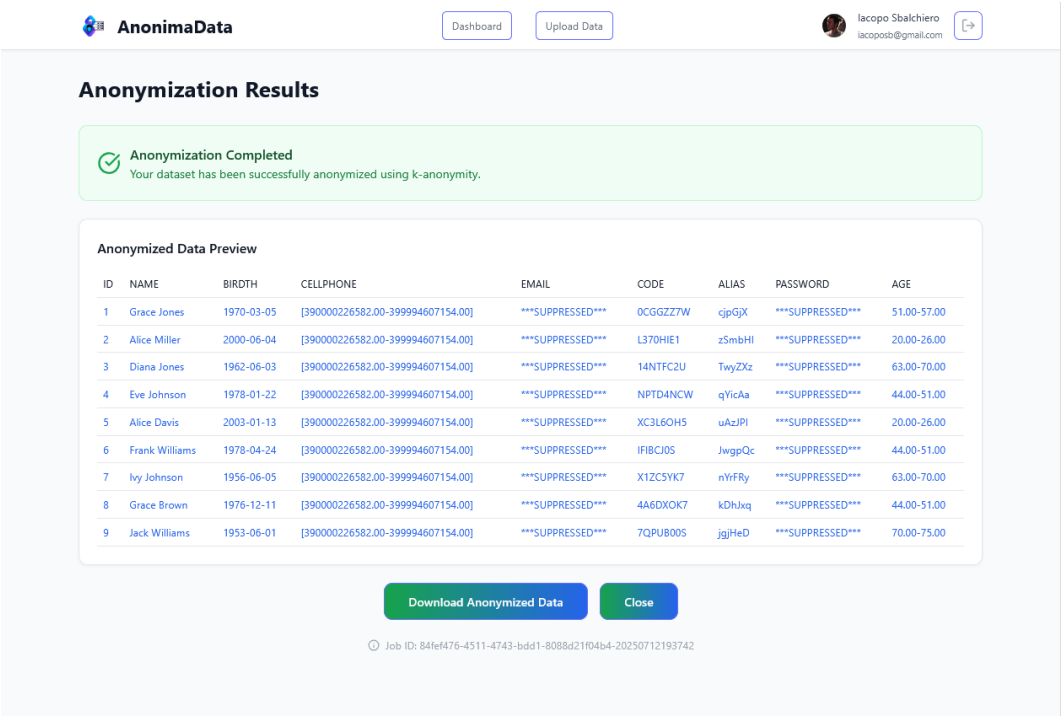

Figura 5 – Anteprima dei dati anonimizzati

La piattaforma è stata progettata per supportare anche scenari di utilizzo più frammentari o asincroni. Qualora l'utente, per qualsiasi motivo, decida di interrompere il processo di anonimizzazione dopo aver effettuato l'upload del dataset ma prima di completare la configurazione dei parametri, il sistema provvede comunque a salvare lo stato corrente del lavoro. All'interno della dashboard, tale dataset apparirà con un'indicazione chiara del suo stato incompleto, consentendo all'utente di riprendere successivamente il processo esattamente dal punto in cui era stato interrotto oppure, se lo ritiene opportuno, di eliminare definitivamente il dataset.

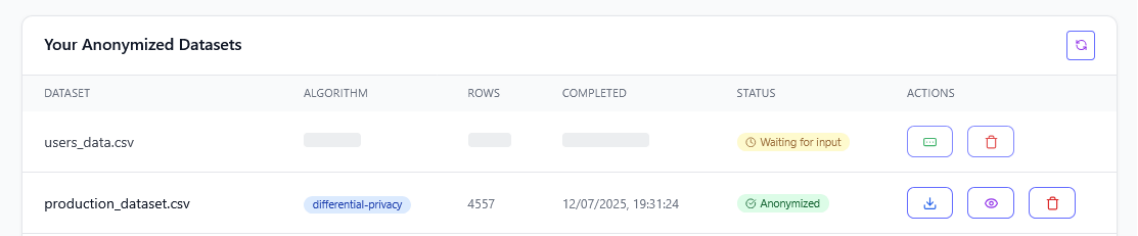

Figura 6 – Dettaglio della dashboard dove è possibile trovare un lavoro in attesa di essere configurato

In caso di errori durante una qualsiasi fase del processo, la piattaforma notifica immediatamente l'utente attraverso un sistema di avvisi visivi. In parallelo, il lavoro in corso viene automaticamente sospeso e salvato con uno stato di errore, permettendo all'utente di decidere in un secondo momento se proseguire con il completamento del flusso oppure eliminarlo. Questa scelta progettuale è stata adottata per garantire la tracciabilità delle operazioni fallite e fornire all'utente un quadro completo dello storico delle proprie attività. Inoltre, nel caso in cui l'utente venga disconnesso durante una fase critica del processo, al successivo accesso la piattaforma provvederà automaticamente a notificare lo stato dell'elaborazione interrotta, mantenendo intatte tutte le informazioni relative al contesto dell'errore.

Flusso di anonimizzazione

Dopo la descrizione delle componenti principali del sistema, possiamo ora esaminare il processo completo di anonimizzazione dei dati.
Il flusso si articola in una serie di passaggi ben definiti, rappresentati sinteticamente nello schema di Figura 2.
Per semplicità espositiva, alcuni dettagli tecnici, come l'autenticazione tramite Firebase, la gestione dei token o le interazioni con il database e i bucket, sono stati omessi nel diagramma.

1. Autenticazione utente: l'utente accede alla webapp (in esecuzione su Cloud Run) e viene reindirizzato a Google per l'autenticazione OAuth tramite Firebase.
2. Emissione del token: Firebase rilascia un ID token, che il frontend utilizza per firmare tutte le richieste successive.
3. Upload file: l'utente carica un file da anonimizzare attraverso l'interfaccia web. Il frontend allega il token all'invio.
4. Verifica del token e registrazione del job: il backend dell'Orchestratore (anch'esso su Cloud Run) riceve la richiesta, verifica l'autenticazione presso Firebase, assegna un job_id univoco, lo associa all'UUID dell'utente e registra tutto nel database.
5. Pubblicazione richiesta di analisi: l'Orchestratore invia una richiesta su Pub/Sub, includendo il file (in base64) e il job_id.
6. Intervento del Formatter: un servizio formatter (Cloud Run) sottoscrive il canale Pub/Sub, riceve il messaggio, decodifica il file, lo analizza (es. identificazione colonne, tipi di dato, struttura tabellare) e lo uniforma.
7. Risposta del Formatter: al termine, pubblica su un secondo canale Pub/Sub i file prodotti (dataset uniformato e descrizione delle colonne) e il job_id.
8. Polling stato da parte del frontend: mentre il processo è in corso, il frontend interroga periodicamente l'Orchestratore passando il job_id, per conoscere lo stato corrente del job.
9. Memorizzazione intermedia: quando l'Orchestratore riceve la risposta del Formatter, salva temporaneamente il file uniformato nel bucket, aggiorna le colonne nel database e imposta lo stato del job come "analizzato".
10. Configurazione dell'anonimizzazione: il frontend riceve lo stato aggiornato e visualizza i dettagli sulle colonne all'utente, che seleziona quali dati anonimizzare e con quali algoritmi e parametri.
11. Pubblicazione richiesta di anonimizzazione: il frontend invia la configurazione all'Orchestratore, che la pubblica (insieme al file formattato) su un terzo canale Pub/Sub.
12. Esecuzione degli Anonymizer: uno o più servizi Anonymizer (Cloud Run) ricevono la richiesta e applicano le trasformazioni richieste alle colonne selezionate (es. k-anonimato, mascheramento, hashing).

13. Polling stato anonimizzazione: anche durante questa fase, il frontend continua a interrogare periodicamente l'Orchestratore per aggiornamenti sul job.
14. Pubblicazione del risultato finale: al termine, l'Anonymizer pubblica il file anonimizzato e il relativo job_id su un quarto canale Pub/Sub.
15. Aggiornamento finale dell'Orchestratore: l'Orchestratore riceve il file anonimizzato, lo salva nel bucket, elimina il file formattato, aggiorna lo stato nel database e genera un'anteprima dei dati anonimizzati.
16. Restituzione dell'esito: alla successiva interrogazione del frontend, l'Orchestratore restituisce lo stato "completato" e l'anteprima.
17. Download finale: l'utente può visualizzare il risultato ed eventualmente scaricare il file anonimizzato. Il dataset rimane disponibile sulla piattaforma per future consultazioni o download.

Lo schema evidenzia l'architettura basata su microservizi Cloud Run orchestrati da Pub/Sub e supportati da Firebase, database e bucket GCP, assicurando scalabilità, tracciabilità dei job e separazione delle fasi (upload, analisi, anonimizzazione).

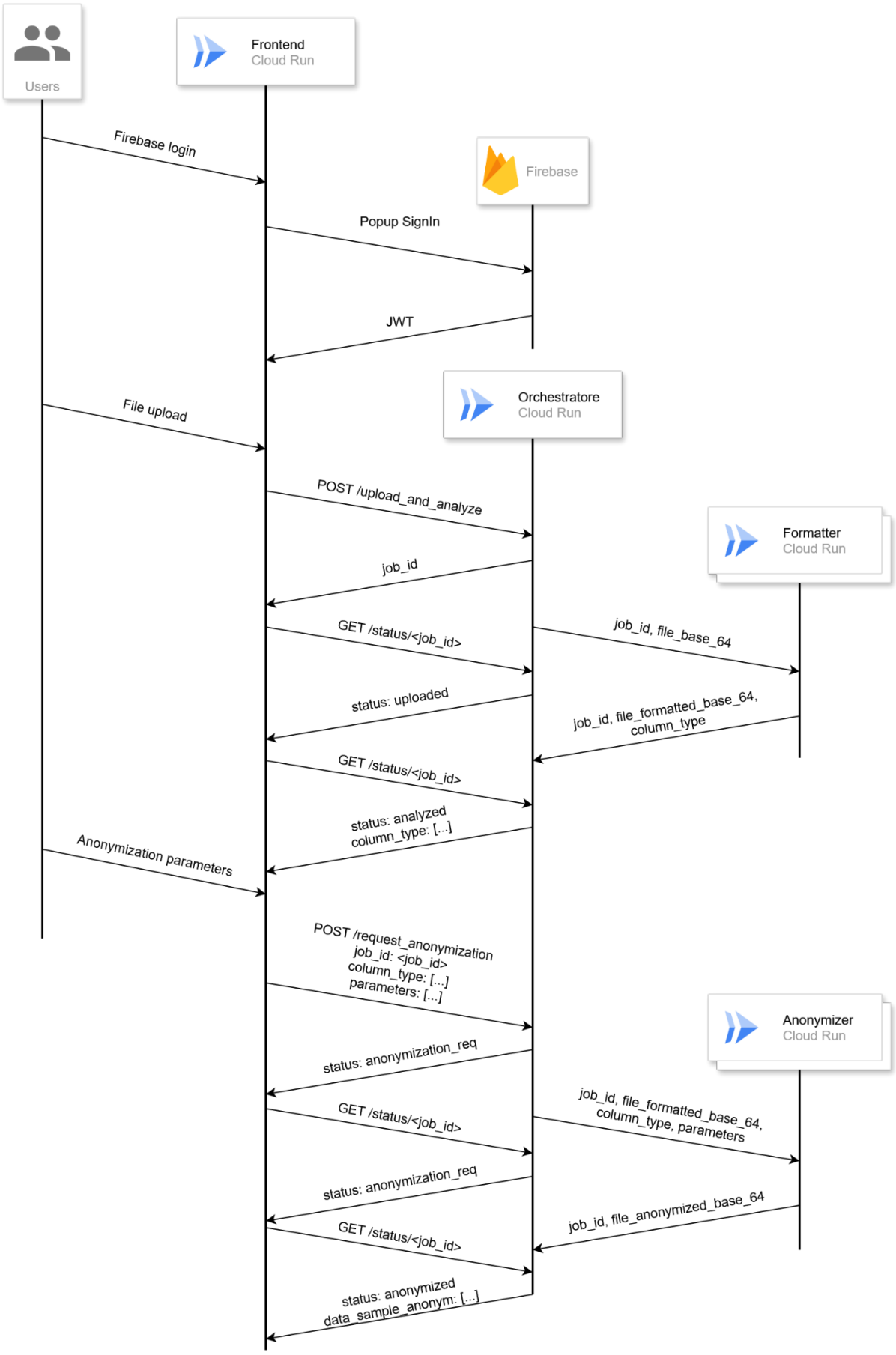

Figura 7 – Flusso di anonimizzazione di un dataset

Test di scalabilità e performance

Al fine di valutare in modo approfondito la robustezza e la scalabilità del sistema, sono stati condotti tre differenti tipologie di test: Stress Test, Spike Test e Soak Test. Questi test sono stati progettati per simulare situazioni realistiche di utilizzo da parte degli utenti, replicando condizioni operative che il sistema potrebbe affrontare in ambienti di produzione. Ogni test prevedeva un flusso standard (iterazione) che comprendeva le seguenti fasi:

- caricamento di un file composto da 1000 righe di dati,
- selezione casuale di uno degli algoritmi di anonimizzazione disponibili, con generazione automatica e randomica dei parametri associati,
- avvio del processo di anonimizzazione,
- download del dataset anonimizzato risultante.

Il test prevede che l'iterazione sia ripetuta ciclicamente dall'utente, fino allo scadere del tempo.

Per garantire che i test automatizzati non interferissero con le normali procedure di autenticazione degli utenti, basate su Firebase, sono stati creati endpoint dedicati esclusivamente all'esecuzione dei test. L'intera automazione dei test è stata realizzata utilizzando il tool k6, che ha consentito di programmare e gestire l'esecuzione dei diversi scenari di carico, monitorando costantemente le risposte del sistema, i tempi di elaborazione e il comportamento in presenza di picchi o carichi prolungati. Grazie a questo approccio, è stato possibile ottenere una panoramica completa delle capacità del sistema, individuare eventuali punti critici e raccogliere dati utili per ottimizzare ulteriormente le prestazioni e la resilienza di AnonimaData.

Il primo problema riscontrato in modo trasversale da tutti e tre i test effettuati (Stress Test, Spike Test e Soak Test) ha riguardato un evidente collo di bottiglia, causato dal limite massimo di connessioni simultanee al database Postgres. Questo limite ha rappresentato una criticità significativa per la scalabilità del sistema, in quanto il database è in grado di gestire un massimo teorico di 100 connessioni contemporanee. Per affrontare questa problematica, si è intervenuti inizialmente regolando i pool di connessione al database, in modo da adattare la configurazione del sistema e limitare la scalabilità al valore massimo consentito da Postgres.

Durante l'analisi del traffico generato dai test, è emerso che una parte consistente delle richieste proveniva dal controllo dello stato di avanzamento dei job, gestito tramite l'endpoint */get_status*. Questo endpoint, infatti, verifica lo stato dei job accedendo al database e viene chiamato periodicamente (sia dal frontend, che monitora in tempo reale lo stato dei processi di anonimizzazione per offrire agli utenti aggiornamenti costanti, sia dal tool di test, che emula il comportamento del frontend), generando un elevato numero di connessioni e

contribuendo al sovraccarico del sistema. Per mitigare questo fenomeno, si è scelto di rendere sincrono esclusivamente l'accesso al database relativo a questa operazione. Sebbene questa decisione comporti un rallentamento generale delle operazioni di verifica dello stato - dato che il controllo avviene tramite polling e gli utenti potrebbero dover attendere qualche istante in più per ricevere aggiornamenti - si è ritenuto preferibile garantire la coerenza e l'aggiornamento costante delle informazioni fornite.

L'introduzione di questa modifica ha prodotto un miglioramento significativo nei risultati dei test, con un aumento delle esecuzioni riuscite dello stress test dal 40% fino al 99%. Tuttavia, questo risultato ha comportato un incremento nella latenza delle richieste HTTP, in particolare per quelle legate alla verifica dello stato dei job. Per questo motivo, nell'analisi delle performance che segue, verranno presentate separatamente le latenze delle richieste HTTP generali e quelle relative alle richieste di controllo dello stato, in modo da fornire una panoramica più accurata dell'effettiva reattività del sistema.

Stress Test

L'obiettivo era determinare la **capacità massima del sistema** prima di iniziare a degradare in termini di performance o stabilità. Il test è stato eseguito in **quattro fasi progressive**:

- 50 utenti concorrenti per 5 minuti
- 100 utenti concorrenti per 5 minuti
- 150 utenti concorrenti per 5 minuti
- 200 utenti concorrenti per 5 minuti

Durante lo svolgimento dello stress test, il sistema è stato sottoposto a un carico molto elevato, con oltre 37.000 controlli effettuati sulle principali funzionalità. I risultati sono stati decisamente incoraggianti: la maggioranza delle operazioni è andata a buon fine, con il 99,66% dei check superati e solo una minima percentuale di errori.

Entrando più nel dettaglio delle singole fasi, il caricamento dei file (upload) ha funzionato in modo affidabile nella maggior parte dei casi, anche se qui si sono concentrati la maggior parte degli errori, con un tasso di successo del 98%. Le richieste di anonimizzazione e le operazioni di download si sono dimostrate ancora più solide, entrambe con una percentuale di successo superiore al 99%. A conferma della robustezza del sistema, tutte le verifiche legate all'esistenza dell'ID del job e ai cambiamenti di stato ("analyzed" e "anonymized") hanno dato esito positivo.

Per quanto riguarda le prestazioni, la durata media delle richieste HTTP si è attestata intorno ai 2 secondi, con una risposta minima davvero rapida e qualche picco di latenza nei momenti di massimo carico. Solo una piccola parte delle richieste HTTP è fallita, appena l'1,24%. Il grafico in Figura 8 mostra la latenza di tutte le richieste HTTP, ma osservando Figura 9 si nota come molta della latenza sia stata prodotta da queste richieste.

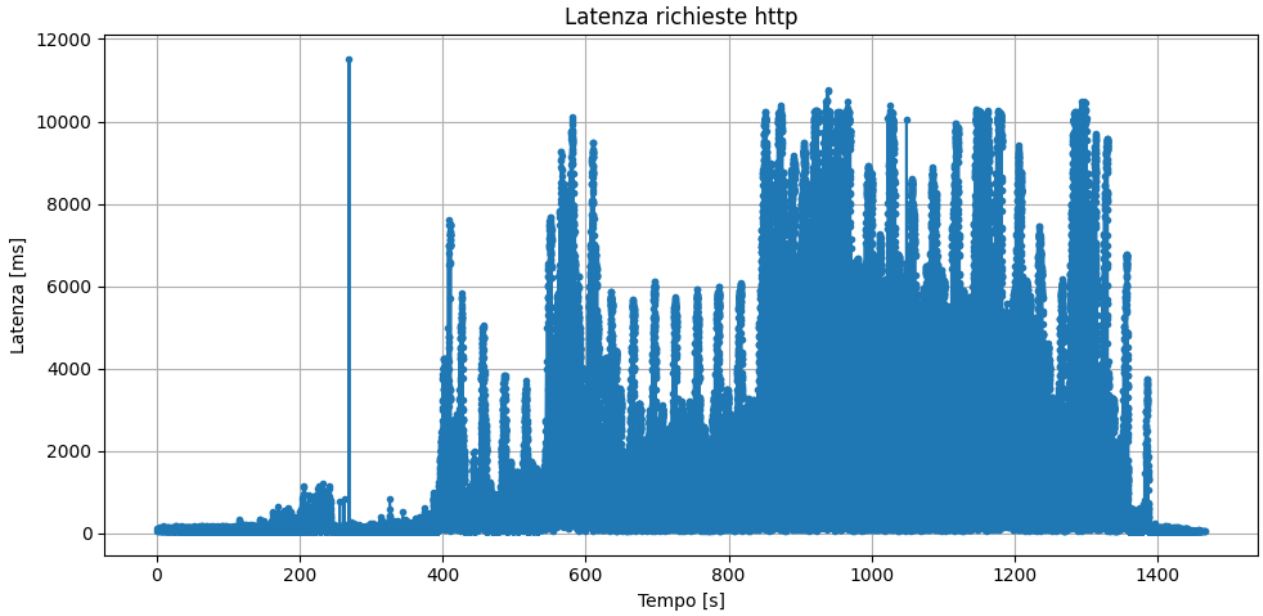

Figura 8 – Latenza richieste http durante lo stress test

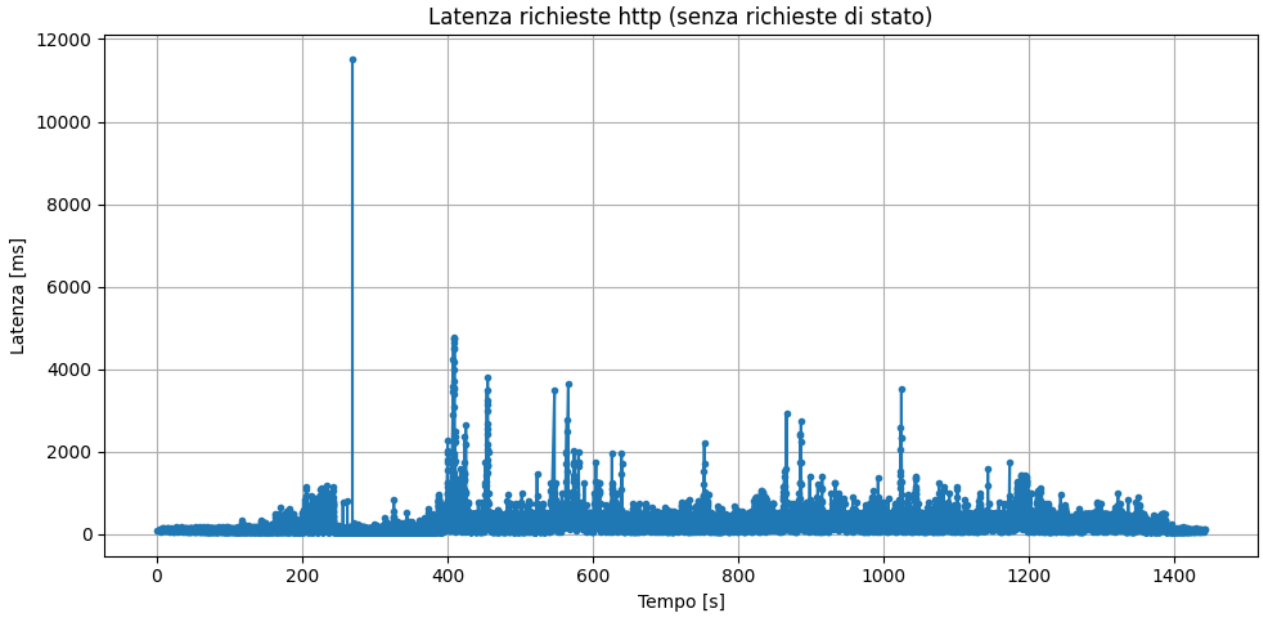

Figura 9 – Latenza richieste http durante lo stress test, filtrando le richieste per ottenere informazioni sullo stato attuale del job

Le iterazioni sono state oltre 6.200, con una durata media di circa 18 secondi e un massimo rilevato che supera i 13 minuti. Da notare che l'incremento della durata delle richieste è avvenuto a partire da 900 secondi (ovvero 15 minuti) dall'inizio del test, indicando che il sistema ha raggiunto notevoli rallentamenti nel momento in cui ha iniziato a gestire parallelamente 200 utenti contemporanei.

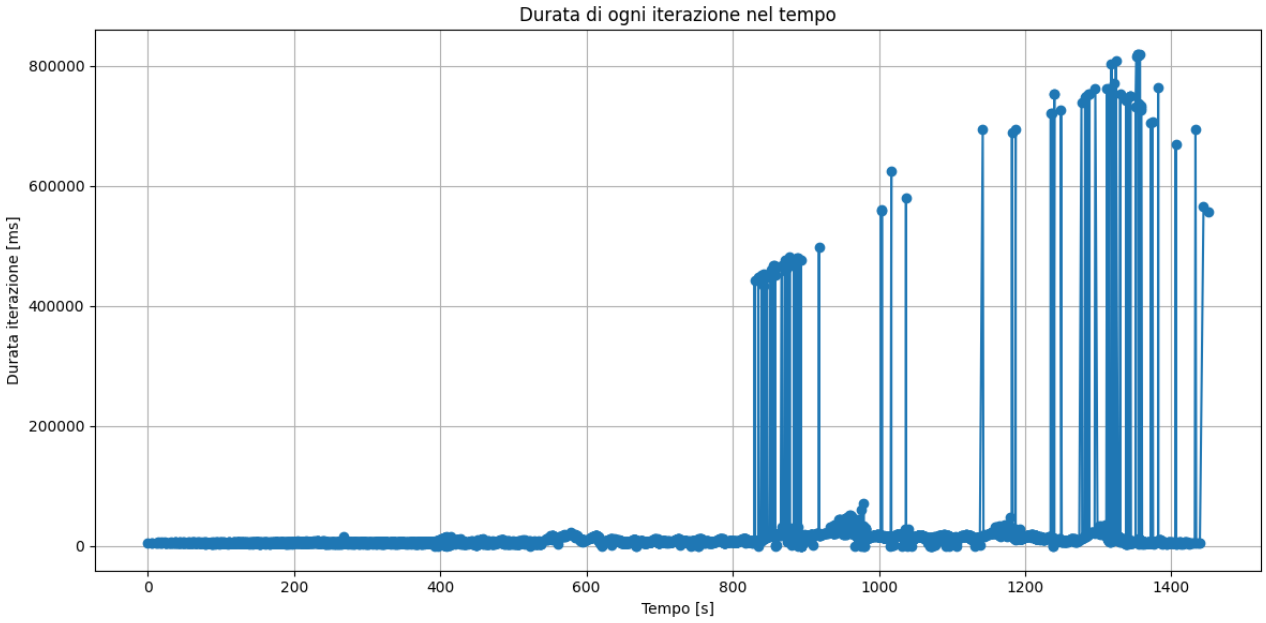

Figura 10 – Durata delle iterazioni durante lo stress test

Spike Test

Questo test ha valutato la **resilienza del sistema a un picco improvviso di traffico**. In particolare: 400 utenti concorrenti per 1 minuto.

Durante la prova sono state effettuate quasi 2.400 verifiche sulle funzionalità principali. Il comportamento generale è stato soddisfacente: il 98,91% delle operazioni è stato portato a termine con successo, mentre poco più dell'1% ha riscontrato degli errori. Nello specifico, il caricamento dei file e la richiesta di anonimizzazione hanno mantenuto percentuali di successo molto alte (99%), mentre la fase di download ha mostrato qualche difficoltà in più, con una percentuale di successo del 91% e 21 errori su 235 tentativi.

Un aspetto interessante riguarda le prestazioni del sistema sotto questo carico improvviso: la durata media delle richieste HTTP è salita a circa 6,3 secondi, con una mediana di 3,7 secondi e diversi picchi che hanno raggiunto quasi i 39 secondi. Il 90% delle richieste si è comunque concluso entro 14,4 secondi. Gli errori HTTP sono stati il 3,21% del totale, un valore superiore rispetto agli altri test, segno che il sistema ha accusato il colpo del picco improvviso di traffico.

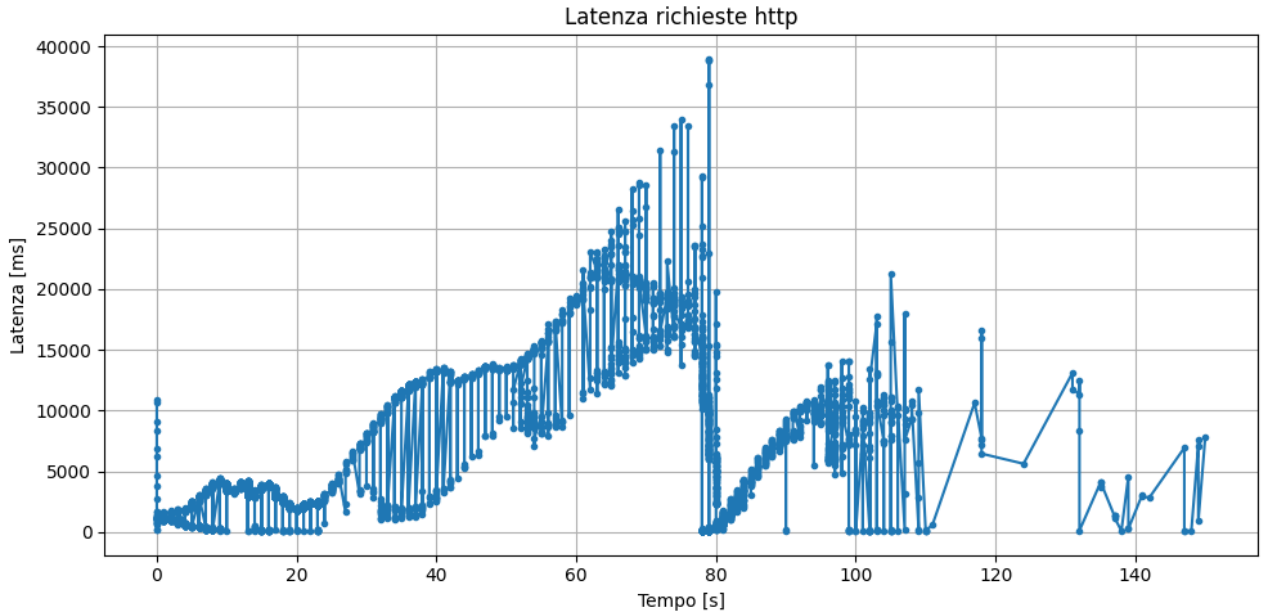

Figura 11 – Latenza richieste http durante lo spike test

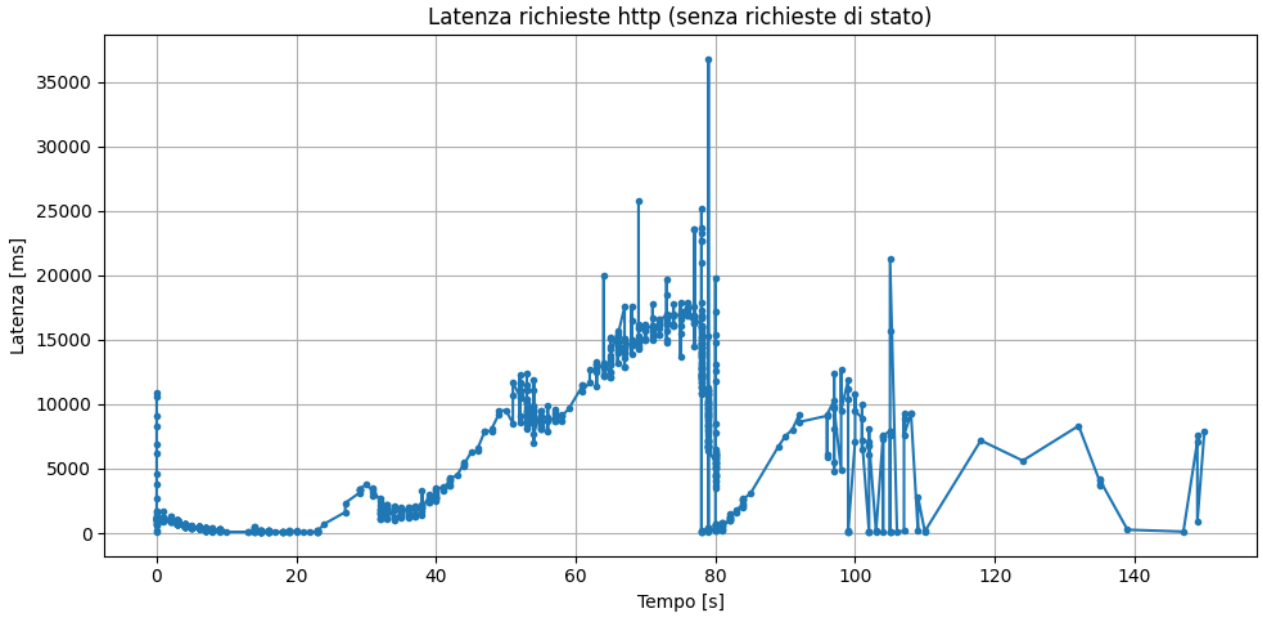

Figura 12 – Latenza richieste http durante lo spike test, filtrando le richieste per ottenere informazioni sullo stato attuale del job

Le iterazioni hanno mostrato una durata media piuttosto elevata: circa 1 minuto e 7 secondi. La mediana si attesta addirittura a 1 minuto e 16 secondi, mentre le iterazioni più lunghe hanno sfiorato i 2 minuti. Il 90% delle iterazioni si è concluso entro 1 minuto e 30 secondi, e il 95% entro 1 minuto e 40 secondi. Come aspettatosi, i tempi di esecuzione delle richieste sono stati crescenti a mano a mano che si procedeva nel test, come si può notare in Figura 13.

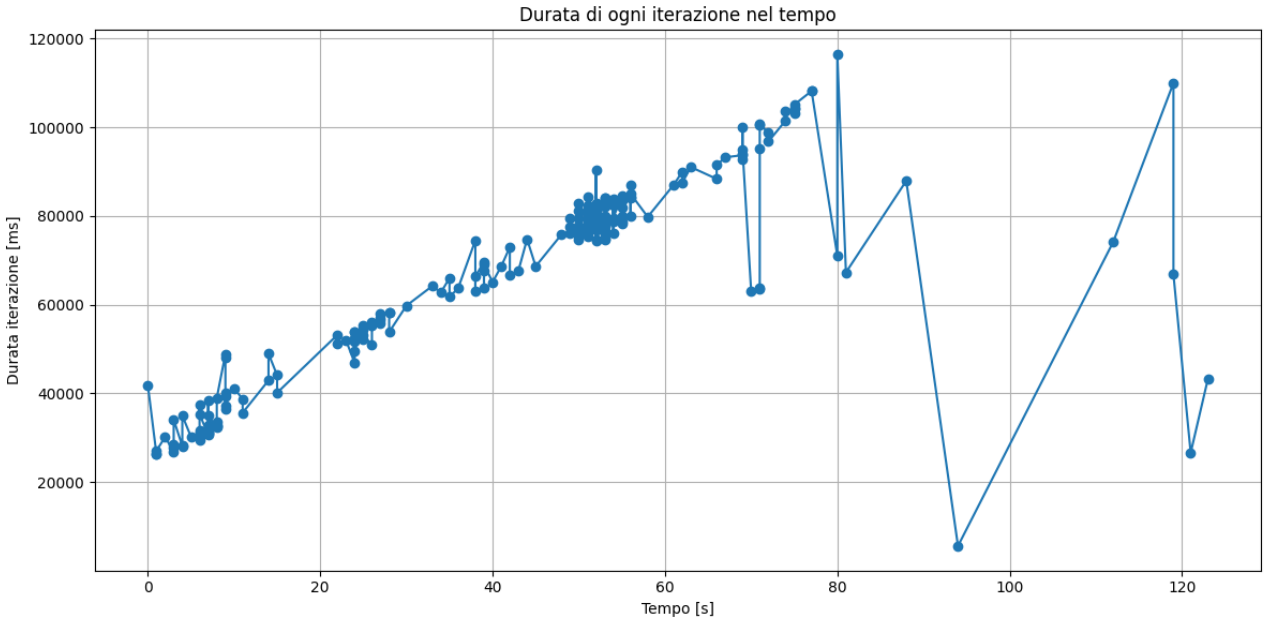

Figura 13 – Durata delle iterazioni durante lo spike test

Questi dati indicano che, sotto un carico improvviso e molto elevato, il sistema risponde in modo stabile ma con tempi di elaborazione sensibilmente più lunghi rispetto a condizioni normali o a test con carichi più graduali. L'aumento delle latenze è probabilmente dovuto alla saturazione delle risorse, in particolare del database e della rete, che diventano più evidenti quando molti utenti tentano di eseguire operazioni contemporaneamente.

Soak Test

Finalizzato a testare la **stabilità del sistema nel lungo periodo**, simulando un carico costante in un tempo prolungato: 50 utenti concorrenti per 1 ora continua.

Durante la prova, sono state effettuate quasi 95.000 verifiche sulle funzionalità principali, con un tasso di successo estremamente elevato: il 99,77% delle operazioni si è concluso correttamente, e solo lo 0,22% ha riscontrato errori.

Analizzando i dettagli delle varie fasi, il caricamento dei file (upload) ha avuto successo nel 99% dei casi, così come le richieste di anonimizzazione e le operazioni di download, che hanno mantenuto percentuali di successo molto alte e solo pochi errori. Le altre verifiche legate all'esistenza dell'ID del job e ai cambiamenti di stato ("analyzed" e "anonymized") si sono svolte senza alcun problema.

Sul piano delle prestazioni, si nota una reattività davvero notevole: la durata media delle richieste HTTP è stata di circa 435 ms, con una mediana sotto i 100 ms e picchi massimi di latenza che hanno raggiunto poco meno di 11 secondi solo in casi eccezionali. Il 90% delle

richieste si è concluso entro 1,2 secondi, e il 95% entro 2,26 secondi. Anche il tasso di fallimento delle richieste HTTP è stato molto contenuto, pari allo 0,21%.

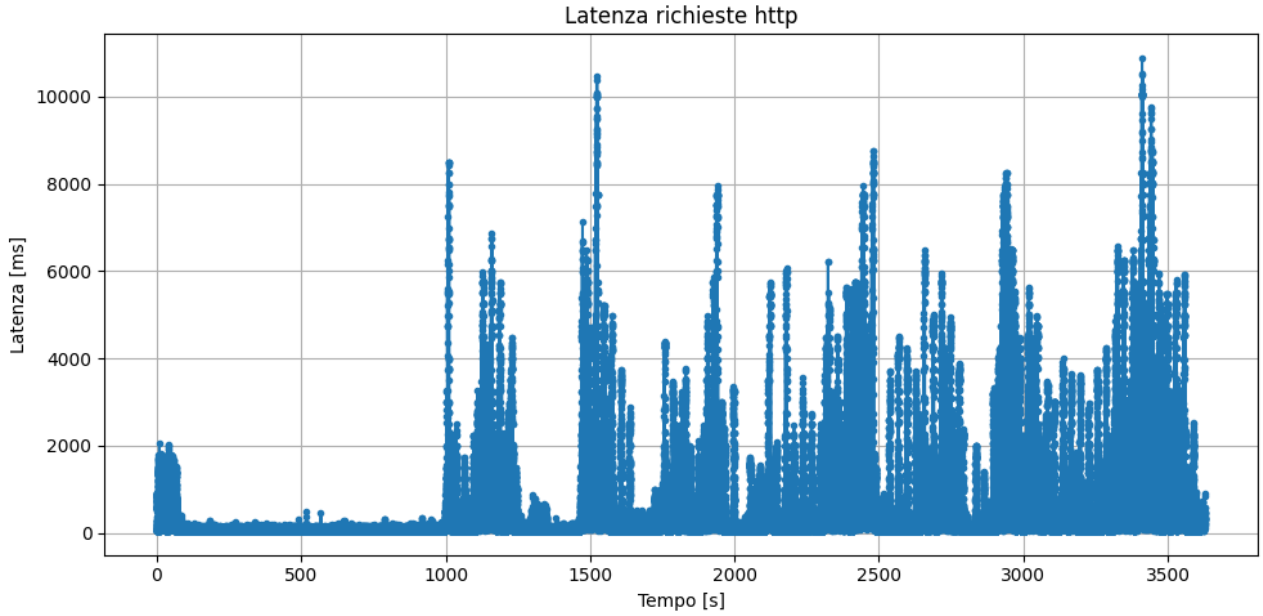

Figura 14 – Latenza richieste http durante il soak test

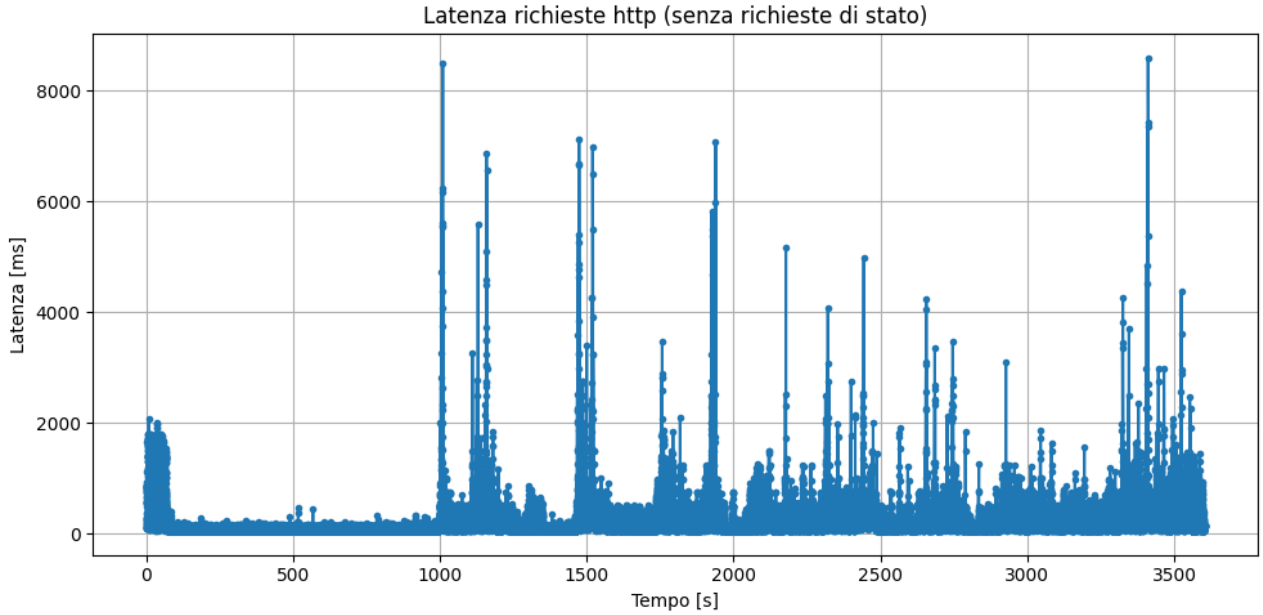

Figura 15 – Latenza richieste http durante il soak test, filtrando le richieste per ottenere informazioni sullo stato attuale del job

Per quanto riguarda la durata delle iterazioni, cioè il tempo necessario per completare l'intero ciclo di operazioni per ciascun utente virtuale. In media, ogni iterazione ha richiesto circa 11 secondi, con una mediana di poco più di 6 secondi. Il 90% delle iterazioni si è concluso entro

10 secondi e il 95% sotto i 12 secondi. Solo in rari casi si sono registrati tempi molto lunghi, fino a quasi 7 minuti, probabilmente dovuti a momentanei rallentamenti o congestioni.

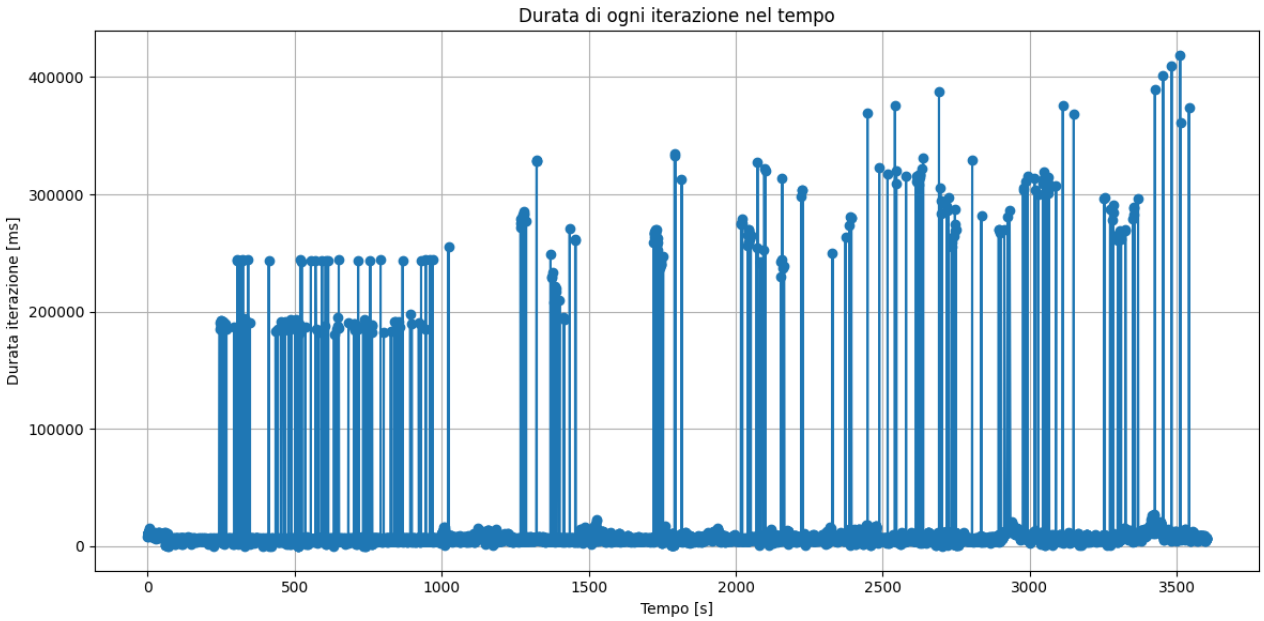

Figura 16 – Durata delle iterazioni durante il soak test

Riassunto dei test

		Stress test	Spike test	Soak test
Esito dei check delle iterazioni	Upload del file	98,64%	99,80%	99,32%
	Job id creato	100,00%	100,00%	100,00%
	File analizzato	100,00%	100,00%	100,00%
	Parametri caricati	99,50%	99,11%	99,50%
	File anonimizzato	100,00%	100,00%	100,00%
	Download del file	99,92%	91,06%	99,83%
Tempo di risposta http (incluse richieste di stato)	Medio	326,14ms	3,73s	93,55ms
	P(95)	8,48s	18,49s	2,26s
	Max	11,52s	38,97s	10,89s
Tempo di esecuzione di una iterazione	Medio	7s	1m16s	6,06s
	P(95)	20,75s	1m40s	12,21s
	Max	13m39s	1m56s	6m58s

Tabella 1 – Riassunto dei test eseguiti sull'orchestratore di AnonimaData