Real Time Collaboration Platform

Progetto finale di Architetture dei Sistemi Distribuiti

Ignazio Emanuele Piccichè
Mattia Castiello
Michela Di Simone

Anno accademico 2023/2024

Indice

1 Introduzione **2**
- 1.1 Obiettivi del progetto . . . 2
- 1.2 Sfide . . . 2

2 Back-End **3**
- 2.1 Gestione delle Connessioni . . . 3
  - 2.1.1 Metodo *file_server* . . . 3
- 2.2 Gestione dei Messaggi . . . 3
  - 2.2.1 Metodo *send_to_all* . . . 3
  - 2.2.2 Metodo *broadcast* . . . 3
  - 2.2.3 Metodo *send_user_list* . . . 4
- 2.3 Gestione del Documento . . . 4
  - 2.3.1 Metodo *load_file* . . . 4
  - 2.3.2 Metodo *save_file* . . . 4
  - 2.3.3 Metodo *network_partition_consistency* . . . 4
- 2.4 Gestione delle Operazioni CRDT . . . 4
  - 2.4.1 Metodo *crdt_operations* . . . 4

3 Front-End **5**
- 3.1 Connessione WebSocket . . . 5
- 3.2 Validazione JSON . . . 5
- 3.3 Gestione dei Timestamp . . . 5
- 3.4 Impostazione Username . . . 6
- 3.5 Gestione delle Modifiche all'Editor . . . 6
- 3.6 Gestione della Posizione del Cursore . . . 6
  - 3.6.1 Invio della Posizione del Cursore . . . 6
  - 3.6.2 Aggiornamento della Posizione del Cursore . . . 6
- 3.7 Simulazione della Partizione . . . 6
- 3.8 Riconnessione al Server . . . 6

4 Docker **7**
- 4.1 Dockerfile per Nginx . . . 7
- 4.2 Dockerfile per il Server Python . . . 7
- 4.3 File Docker-compose.yaml . . . 8

5 Compute Engine (GCP) **9**
- 5.1 Configurazione del Firewall . . . 9
- 5.2 Accesso al Servizio . . . 9

6 Sviluppi futuri **11**

1 Introduzione

Il progetto mira a sviluppare una piattaforma di collaborazione in tempo reale che non solo supporti la modifica condivisa di un documento, ma che sia anche resiliente alle partizioni di rete, garantendo così un'esperienza utente affidabile.

1.1 Obiettivi del progetto

Il principale obiettivo del progetto è creare un editor collaborativo in grado di:

- Consentire a più utenti di modificare lo stesso documento simultaneamente, sfruttando l'utilizzo delle WebSocket
- Mantenere coerenza dei dati e delle modifiche da parte degli utenti, in presenza di conflitti, utilizzando la tecnica *Conflict-free Replicated Data Types* (CRDTs)
- Gestire le partizioni di rete in modo da assicurare che tutte le modifiche vengano correttamente integrate una volta risolta la partizione
- Notificare in modo efficace la presenza degli utenti, informando chi è attualmente connesso al documento condiviso

1.2 Sfide

Per verificare la robustezza della piattaforma, sono stati condotti esperimenti mirati che includono:

- La simulazione delle partizioni di rete per testare la capacità del sistema di riconciliare le modifiche
- L'analisi dell'impatto delle diverse strategie di risoluzione dei conflitti sulla coerenza e la latenza del sistema

In conclusione, il progetto non solo esplorerà la complessità della collaborazione in tempo reale, ma cercherà anche di fornire soluzioni pratiche e innovative per garantire la continuità e l'integrità del lavoro condiviso in scenari di rete variabili.

2 Back-End

All'interno del *FileServer.py* che è stato sviluppato viene gestita la collaborazione in tempo reale su un documento condiviso tramite *WebSocket,* Consentendo a più client di connettersi e interagire simultaneamente con un file, aggiornandolo e mantenendo la consistenza tra le versioni sui diversi client.

L'applicazione utilizza un algoritmo di CRDT (*Conflict-free Replicated Data Type*) per risolvere conflitti e garantire che tutti i client visualizzino la stessa versione aggiornata del documento.

2.1 Gestione delle Connessioni

Questa sezione descrive come il server gestisce le connessioni WebSocket, mantenendo un registro dei client connessi e aggiornando gli altri client quando un nuovo utente si connette o si disconnette.

2.1.1 Metodo *file_server*

- Questo metodo gestisce la connessione e la comunicazione con i client. All'avvio della connessione, il server richiede il nome utente del client e lo aggiunge alla lista degli utenti connessi. In caso di disconnessione, il server rimuove il client dalla lista e informa gli altri utenti.

2.2 Gestione dei Messaggi

In questa sezione viene spiegato come il server gestisce e trasmette le modifiche tra i client. Questo include sia i messaggi di aggiornamento del documento che i messaggi di stato del sistema.

2.2.1 Metodo *send_to_all*

- Questo metodo invia un messaggio a tutti i client connessi. È utilizzato per notificare tutti gli utenti riguardo a eventi come l'ingresso o l'uscita di un utente.

2.2.2 Metodo *broadcast*

- Simile a *send_to_all,* ma specificamente progettato per inviare aggiornamenti del contenuto del file a tutti i client. Questo metodo utilizza JSON per serializzare i messaggi, facilitando l'invio di dati strutturati.

2.2.3 Metodo *send_user_list*

- Questo metodo aggiorna tutti i client con la lista attuale degli utenti connessi, mantenendo la visibilità sui partecipanti.

2.3 Gestione del Documento

Questa sezione esplora come il server carica, salva e aggiorna il contenuto del file condiviso.

2.3.1 Metodo *load_file*

- Carica il contenuto del file condiviso dal file system, se esiste. Se il file non esiste, restituisce una stringa vuota.

2.3.2 Metodo *save_file*

- Salva il contenuto del file condiviso nel file system, aggiornando il file con le modifiche recenti.

2.3.3 Metodo *network_partition_consistency*

- In fase di riconnessione di un utente, risolve i conflitti tra versioni del file ottenute da diverse fonti (utenti), utilizzando il metodo *SequenceMatcher* della libreria *difflib* per allineare i cambiamenti.

2.4 Gestione delle Operazioni CRDT

Questa sezione tratta il modo in cui il server utilizza il CRDT per applicare e sincronizzare le modifiche al documento.

2.4.1 Metodo *crdt_operations*

- Calcola le operazioni necessarie per trasformare un testo vecchio in uno nuovo, generando una lista di operazioni che rappresentano aggiunte, cancellazioni o modifiche. Queste operazioni sono poi applicate alla classe CRDT per mantenere la coerenza tra i client.

3 Front-End

All'interno del file ***style.css*** si definisce lo stile visivo e il layout della pagina web.
All'interno del file ***HomePage.html*** in cui è stata definita la struttura della pagina web, dove è possibile inserire il proprio nome utente , modificare in real time un documento e osservare lo stato e la lista degli utenti online. Inoltre, viene gestita la logica della comunicazione in tempo reale e l'interazione utente. Le funzionalità chiavi implementate sono:

- **Gestione degli utenti**: gli utenti possono inserire un nome utente e connettersi al server.
- **Modifica del documento**: gli utenti possono digitare contenuti in un'area di testo condivisa.
- **Collaborazione in tempo reale**: le modifiche apportate da un utente si riflettono nel documento per tutti gli utenti connessi.
- **Sincronizzazione della posizione del cursore**: la posizione del cursore di ogni utente viene monitorata.

3.1 Connessione WebSocket

La funzione *connectedWebSocket* stabilisce una connessione WebSocket con il server sulla porta 5555:

- **Onopen**: Quando la connessione viene aperta, se ci sono cambiamenti locali, vengono inviati al server.
- **Onmessage**: Gestisce i messaggi ricevuti dal server, aggiornando il contenuto del file di testo e la lista degli utenti online.

3.2 Validazione JSON

La funzione *isValidJSON* verifica se una stringa può essere convertita in un oggetto JSON restituendo *true*, altrimenti restituisce *false*.

3.3 Gestione dei Timestamp

La funzione *toggleTimestamps.onchange* gestisce la visualizzazione dei timestamp nei messaggi di stato, in base allo stato della checkbox.

3.4 Impostazione Username

La funzione *setUsernameButton.onclick* gestisce l'evento click sul pulsante *Set Username*, inviando il nome utente al server, abilitando l'editor se il nome utente non è vuoto e la possibilità di disconnettersi.

3.5 Gestione delle Modifiche all'Editor

La funzione *editor.addEventListener('input')* rileva le modifiche nell'editor, crea un oggetto messaggio contenente il contenuto aggiornato e il nome utente. Memorizza il messaggio localmente e lo invia al server, cattura la posizione corrente nell'editor e la memorizza in una variabile.

3.6 Gestione della Posizione del Cursore

3.6.1 Invio della Posizione del Cursore

La funzione *sendCursorPosition* recupera la posizione corrente del cursore dall'editor e invia il nome utente e la posizione del cursore alla funzione *setCursorPosition* lato server.

3.6.2 Aggiornamento della Posizione del Cursore

La funzione *setCursorPosition(position)* aggiorna lo stato del documento e la posizione del cursore per quell'utente.

3.7 Simulazione della Partizione

La funzione *simulatePartitionButton.onclick* simula una partizione di rete chiudendo la connessione WebSocket. Successivamente abilita il pulsante di riconnessione e disabilita il pulsante *Simulate Partition*. Inoltre, quando l'utente si disconnette, viene cancellata la visualizzazione dello stato e dell'elenco degli utenti.

3.8 Riconnessione al Server

La funzione *reconnectButton.onclick* gestisce l'evento click sul pulsante *Reconnect*, chiamando la funzione *connectedWebSocket* per ristabilire la connessione al server.

4 Docker

Nel progetto sono stati utilizzati *Docker* e *Docker Compose*. Docker è una piattaforma open-source che automatizza la distribuzione di applicazioni all'interno di container software. I container sono leggeri, portabili e autosufficienti, il che significa che contengono tutto il necessario per eseguire l'applicazione: codice, librerie di sistema e impostazioni.

La scelta dell'adozione di Docker in questo progetto è stata dettata dalle seguenti motivazioni:

- **Ambienti coerenti**: Garantire che l'applicazione si comporti in modo identico in fase di sviluppo, test e produzione
- **Facilità di distribuzione**: semplificare la distribuzione dell'applicazione su diversi ambienti senza dover riconfigurare il sistema
- **Scalabilità**: facilitare il ridimensionamento dei servizi, permettendo di gestire picchi di carico in modo più efficiente
- **Isolamento**: separare i diversi componenti dell'applicazione, migliorando la sicurezza e la gestione delle dipendenze

Di seguito, vengono descritti i file di configurazione utilizzati.

4.1 Dockerfile per Nginx

Il Dockerfile per Nginx è utilizzato per configurare un container che serve contenuti web statici. Il contenuto del Dockerfile è il seguente:

```
# Use the official Nginx image from the Docker Hub
FROM nginx:alpine

# Copy the custom Nginx configuration file into the container
COPY HomePage.html /usr/share/nginx/html/index.html
COPY style.css /usr/share/nginx/html/style.css
```

Listing 1: Dockerfile per Nginx

4.2 Dockerfile per il Server Python

Il Dockerfile per il server Python configura un container per eseguire un'applicazione Python. Il contenuto del Dockerfile è il seguente:

```
# Use the official Python image from the Docker Hub
FROM python:3.10

# Set the working directory in the container to /app
WORKDIR /app

# Copy the current directory contents into the
   container at /app
COPY . /app

# Install the dependencies
RUN pip install -r requirements.txt

# Run app.py when the container launches
ENTRYPOINT ["python", "FileServer.py"]
```

Listing 2: Dockerfile per il server Python

4.3 File Docker-compose.yaml

Il file ***docker-compose.yaml*** è stato utilizzato per orchestrare i due servizi definitivi nei Dockerfile sopra descritti. Il contenuto è il seguente:

```
version: '3.8' # version of docker-compose

services: # services that will be run
  web:
    build:
      context: ./web
    ports:
       - "8583:80"
    networks:
      - sharedDockNet

  server:
    build: # build the image from the current directory
      context: ./server
    ports:
      - "5555:6000"
    networks:
      - sharedDockNet

networks: # network named
  sharedDockNet:
```

Listing 3: Docker-compose

5 Compute Engine (GCP)

Il programma è stato implementato all'interno di una *Compute Engine* di *Google Cloud Platform,* utilizzando una macchina virtuale di piccole dimensioni con le seguenti caratteristiche:

- **Tipo di macchina**: e2-micro
- **Piattaforma CPU**: Intel Broadwell
- **Architettura**: x86/64
- **GPU**: Nessuna

La scelta di una macchina e2-micro è stata dettata dalla necessità di mantenere bassi i costi operativi, pur garantendo prestazioni adeguate per gestire la collaborazione in tempo reale su documenti condivisi. La piattaforma CPU Intel Broadwell offre una buona combinazione di efficienza energetica e potenza di calcolo, sufficienti per le nostre esigenze.

5.1 Configurazione del Firewall

Per assicurare il corretto funzionamento del servizio, è stato necessario configurare opportunamente i criteri di firewall nel servizio *VPC Network* di Google Cloud Platform. Questo ha incluso l'abilitazione delle seguenti porte in entrata per permettere la comunicazione tra i client e il server:

- **Porta TCP 5555**: Utilizzata per il servizio server (backend). Questa porta è cruciale per la gestione delle operazioni del CRDT e la sincronizzazione dei documenti tra i client.
- **Porta TCP 8583**: Utilizzata per il servizio web (client). Questa porta consente agli utenti di accedere all'interfaccia web per modificare i documenti.

Gli intervalli IP sono stati configurati su `0.0.0.0/0`, permettendo così l'accesso da qualsiasi indirizzo IP. Questo è stato necessario per garantire che i client possano connettersi al server indipendentemente dalla loro posizione geografica, facilitando una collaborazione globale senza restrizioni.

5.2 Accesso al Servizio

Il sito è visitabile al seguente indirizzo: `http://34.154.106.176:8583/`

Questa configurazione permette agli utenti di accedere al servizio web per la modifica collaborativa del documento in tempo reale.

Grazie all'infrastruttura scalabile di Google Cloud, il programma può gestire un numero crescente di utenti senza compromettere le prestazioni. Questo consente un'esperienza di collaborazione efficiente e senza interruzioni.

6 Sviluppi futuri

Nonostante il progetto attuale rappresenti una solida base per una piattaforma di collaborazione condivisa, è possibile individuare alcune aree di miglioramento ed eventuali funzionalità avanzate che potrebbero essere sviluppate in futuro al fine di migliorare l'esperienza dell'utente e la robustezza del sistema

1. **Tracciamento delle modifiche utente**

   Attualmente è consentito a ciascun utente online di modificare il documento del testo, senza tenere traccia di chi ha apportato una determinata modifica. Un possibile miglioramento potrebbe essere l'implementazione di sistema che tenga traccia delle modifiche effettuate da ciascun utente e permetta di:

   - Visualizzare il cursore di ciascun utente quando si sta modificando il testo in tempo reale
   - Utilizzare un colore specifico per ogni utente che evidenzi le modifiche che quell'utente ha apportato

2. **Possibilità di scaricare il documento**

   Un'ulteriore implementazione potrebbe essere la possibilità di scaricare il documento in vari formati (PDF, docx, txt), in modo tale da poterne usufruire anche offline e tenere traccia delle varie versioni del documento

3. **Risoluzione dei bug di merge**

   Quando un utente subisce una partizione della rete e poi si riconnette, il testo scritto offline subisce un merge con quello scritto dagli utenti online. Tuttavia, a volte possono verificarsi dei bug. Risulta quindi essenziale:

   - Implementare degli algoritmi di merge più robusti che possano gestire meglio i conflitti tra parole
   - Fornire la possibilità di notifica e di risoluzione manuale dei conflitti nei casi più complessi

4. **Stylish Text**

   Per rendere l'editor più versatile, sarebbe opportuno aggiungere la personalizzazione del testo. Questo prevederebbe:

   - La possibilità di formattazione di base come grassetto, corsivo, testo sottolineato, colore del testo, dimensione del carattere

- Integrazione di formattazione avanzata come elenchi puntati, enumerazioni, inserimento di link, immagini