Università degli Studi di Napoli Federico II
Esame di Advanced Computer Programming

Proff. De Simone, Della Corte

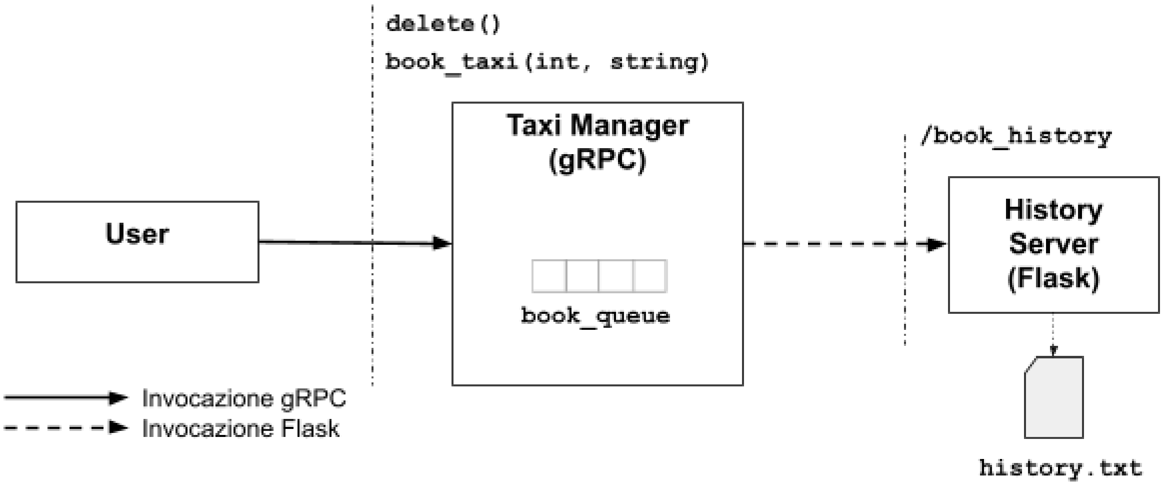
Prova pratica del giorno 11/04/2025
Durata della prova: 120 minuti

°°

Lo studente legga attentamente il testo e produca il programma ed i casi di test necessari per dimostrarne il funzionamento. Al termine della prova lo studente dovrà far verificare il funzionamento del programma ad un membro della Commissione.

°°

Testo della prova



Il candidato implementi un sistema distribuito in **Python** per la prenotazione di corse di taxi basato su **gRPC** e **Flask**. Il sistema è caratterizzato dai seguenti componenti.

User. E' un client utilizzato per richieste di prenotazione verso il **Taxi Manager**. L'invio di una richiesta di prenotazione consiste nella invocazione del metodo remoto *book_taxi(int)*, che prevede come parametro il **numero di_persone** (tipo intero) da trasportare, ed il luogo di ritiro (tipo stringa). L'invio di una richiesta di cancellazione di una prenotazione consiste nella invocazione del metodo remoto *delete()*, che non prevede parametri. User avvia 10 thread: ogni thread genera o una richiesta di prenotazione, invocando il metodo *book_taxi*, oppure una richiesta di cancellazione di una prenotazione, invocando il metodo *delete*. Nel caso di richiesta di prenotazione il *numero di persone* ed il *luogo di ritiro* vengono generati casualmente (intero tra 1 e 4 per il numero di persone, stringa nel formato *via Rossi X*, con X intero compreso tra 1 e 100).

Taxi Manager. E' un server gRPC che espone i metodi remoti *book_taxi* e *delete*. Il metodo *book_taxi* inserisce nella coda *book_queue* un dizionario con il *numero di persone* ed il *luogo di ritiro*. Il metodo *delete* invece consuma una prenotazione dalla coda *book_quque*. N.B: E' necessario utilizzare una lista Python (`list`) per implementare la coda, prevedendo i meccanismi di sincronizzazione necessari per il problema produttore/consumatore; la coda ha dimensione pari a 5. Prima di ritornare, i metodi *book_taxi* e *delete* generano un **richiesta di tipo POST** verso l'**History Server**, inserendo nel body il *tipo di operazione effettuata* (cioè book_taxi o delete) ed il *numero di persone* ed il *luogo di ritiro*, in formato JSON, e.g., `{"operation":"delete", "num_persone":4, "ritiro":"Via Rossi 50"}`, attendendo la risposta prima di ritornare. Il metodo *delete* ritorna al chiamante una tupla con il *numero di persone* ed il *luogo di ritiro* della prenotazione estratta dalla coda, mentre il metodo *book_taxi* un semplice ack (valore booleano).

History Server. Implementa un server Flask che espone una REST API con l'endpoint *book_history*. Tale endpoint accetta richieste di tipo POST, con payload in formato JSON (descritto in precedenza). Ricevuta una richiesta, l'History Server scrive (in append) sul file *history.txt* una stringa che è la concatenazione dei tre campi ricevuti tramite il payload della POST, cioè operation, numero di persone e luogo di ritiro, e.g., `book_taxi-4-Via Rossi 50`.