

Progetto LSO

Libreria

Ingenito Roberto - N86004077

Ingenito Simone - N86004063

Sequino Lorenzo - N86004367

Indice

1	Analisi dei requisiti	2
1.1	Obiettivo del progetto	3
1.2	Funzionalità principali	3
1.2.1	Registrazione e login utenti	3
1.2.2	Gestione del catalogo dei libri	3
1.2.3	Prestito di libri	3
1.2.4	Gestione del carrello e check-out	3
1.2.5	Notifiche agli utenti	3
2	Implementazione	4
2.1	Linguaggi utilizzati	5
2.2	Struttura del codice	6
2.3	Registrazione e login	7
2.4	Gestione dei libri e del catalogo	7
2.5	Prestito e restituzione libri	7
2.6	Carrello e check-out	8
2.7	Gestione delle notifiche	8
2.8	Funzioni speciali	8
3	Database	9
3.1	Schema	10
3.2	Descrizione delle tabelle	10
3.3	Query principali utilizzate	10

Analisi dei requisiti

1.1 Obiettivo del progetto

1.2 Funzionalità principali

1.2.1 Registrazione e login utenti

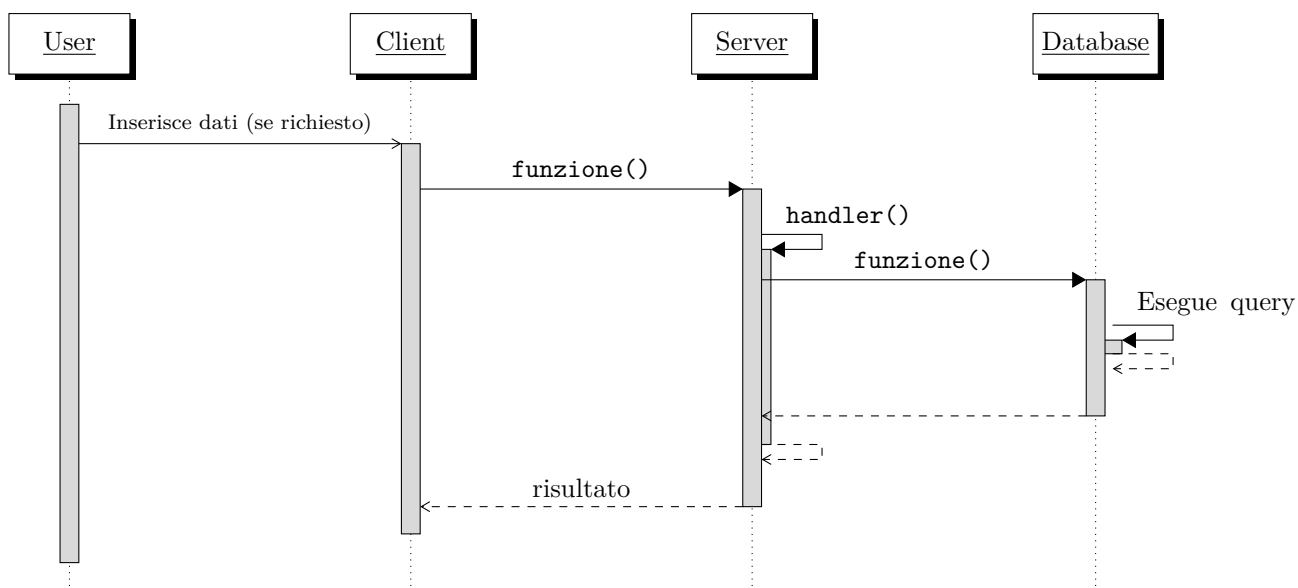
1.2.2 Gestione del catalogo dei libri

1.2.3 Prestito di libri

1.2.4 Gestione del carrello e check-out

1.2.5 Notifiche agli utenti

Implementazione



Questo *sequence diagram* descrive la maggior parte delle funzionalità presenti nel progetto. Viene fornito giusto un esempio per la registrazione nel paragrafo apposito.

2.1 Linguaggi utilizzati

Il progetto utilizza due linguaggi principali:

- **C:** Il linguaggio C è utilizzato per lo sviluppo dell'applicazione client-server. Il server e il client comunicano tramite socket, permettendo lo scambio di dati in rete.
- **SQL:** Il linguaggio SQL è utilizzato per la gestione del database PostgreSQL. Il server si connette al database per eseguire operazioni di lettura e scrittura, gestendo le informazioni richieste dai client.

2.2 Struttura del codice

Il progetto è organizzato in una struttura di cartelle che separa logicamente i vari componenti del progetto. Questa organizzazione aiuta a mantenere il codice pulito, modulare e facile da gestire.

Di seguito, è riportata la struttura gerarchica delle cartelle:

```
project_root
+-- client
|   +-- build
|   +-- include
|   +-- src
+-- config
+-- database
|   +-- include
|   +-- query
|   +-- src
+-- server
    +-- build
    +-- include
    +-- src
```

client / server

Le cartelle `client` e `server` hanno la stessa struttura:

- **build**: Contiene i file oggetto generati dalla compilazione del codice contenuto in `src`
- **include**: Contiene i file header
- **src**: Contiene il codice sorgente

La separazione tra `include` e `src` permette una chiara distinzione tra le interfacce e le implementazioni, migliorando la manutenibilità del codice.

La cartella `build` isola i file generati dalla compilazione, mantenendo pulito il repository.

config

Questa cartella possiede dei file di configurazione con implementazioni comuni sia per il *client* che per il *server*. Ad esempio la configurazione del database, le richieste che client e server possono (rispettivamente) fare e ricevere, oppure la configurazione degli indirizzi.

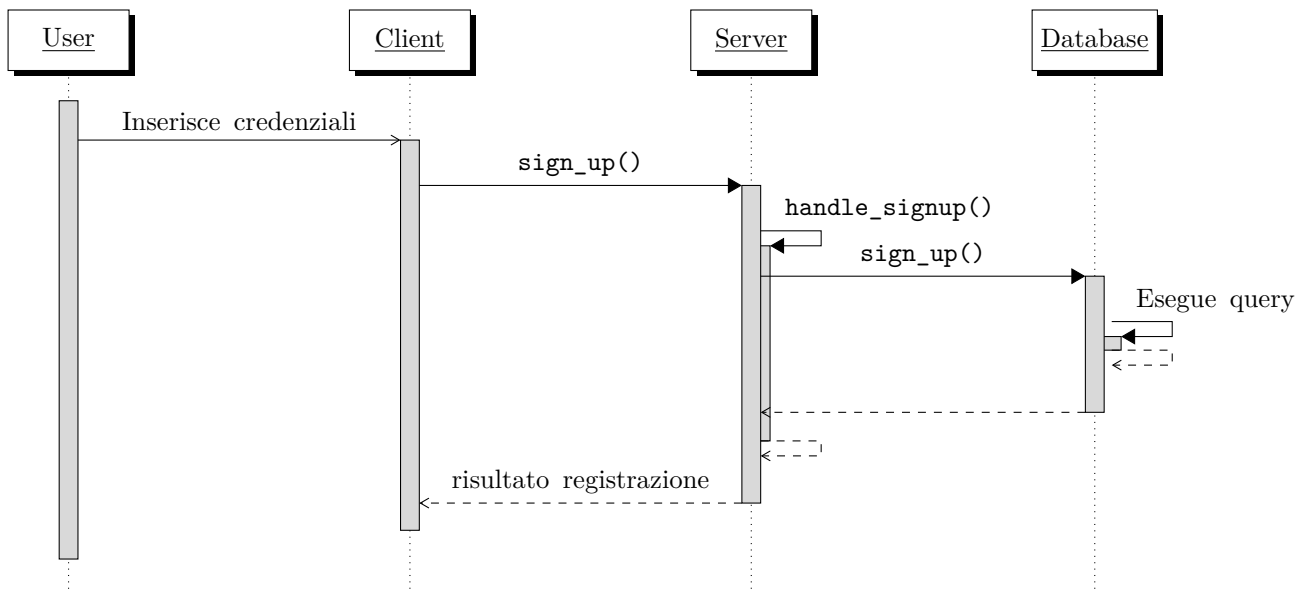
Centralizzare le configurazioni in una cartella dedicata facilita la gestione delle impostazioni del sistema e permette di modificare facilmente i parametri di configurazione senza dover cercare all'interno del codice sorgente.

database

- **include**: Contiene i file header per la gestione del database.
- **query**: Contiene i file SQL utilizzati per la gestione del database. (Creazione delle tabelle, trigger, funzioni, viste, ...)
- **src**: Contiene il codice sorgente per l'interazione con il database.

La separazione dei file SQL e del codice sorgente relativo al database aiuta a mantenere organizzate le operazioni di gestione del database e facilita eventuali modifiche o aggiornamenti.

2.3 Registrazione e login



L'utente inserisce le credenziali per la registrazione (*sign_up*) o l'accesso (*sign_in*).

Il client invia tre messaggi al server:

1. tipo di richiesta: **SIGN_IN** o **SIGN_UP**
2. username
3. password

Il server elabora la richiesta eseguendo una query con i dati ricevuti e successivamente invia un messaggio al client per notificare l'esito dell'operazione, indicando se la richiesta è stata eseguita con successo o meno.

2.4 Gestione dei libri e del catalogo

Il client può inviare diverse richieste al server per ottenere informazioni sui libri presenti nel sistema. Il server risponde a queste richieste eseguendo specifiche funzioni che interrogano il database.

Di seguito, una panoramica delle funzionalità disponibili:

1. Visualizzare l'intero catalogo
2. Visualizzare solo i libri disponibili
3. Cercare libri per genere
4. Cercare libri per titolo

Tutte queste funzioni restituiscono i risultati in formato JSON. Invece di inviare una stringa per volta per poi dover scomporre ogni campo, tramite JSON si facilita l'integrazione lato client.

2.5 Prestito e restituzione libri

Il client può interagire con il server per gestire i prestiti dei libri. Il server offre diverse funzionalità per supportare queste operazioni:

1. Creare un nuovo prestito
2. Visualizzare i libri attualmente in prestito
3. Restituire un libro

2.6 Carrello e check-out

L'utente può aggiungere libri al carrello, che viene mantenuto localmente sul client stesso. Quando l'utente decide di procedere al checkout, i libri presenti nel carrello vengono inviati al server. Il server poi elabora la richiesta, gestendo il processo di prestito.

La funzione `handle_loan_requests` gestisce le richieste di prestito di libri da parte del client. Essa svolge le seguenti operazioni principali:

- Legge gli ISBN dei libri richiesti dal client e li memorizza in una lista fino a quando non riceve il messaggio "STOP".
- Blocca un mutex (semaforo) per garantire l'accesso esclusivo alla sezione critica del codice che gestisce la creazione dei prestiti, poi chiama la funzione `create_loans` per processare le richieste di prestito.
- Dopo aver elaborato le richieste, invia un codice di stato al client per indicare se la query è riuscita o meno.
- Sblocca il mutex al termine dell'operazione.

La richiesta di prestito di un libro è una sezione critica del sistema, poiché coinvolge la modifica dello stato, sul database, dei prestiti. In particolare, se un libro ha solo una copia disponibile e più utenti tentano di richiedere contemporaneamente il prestito di quel libro, il database potrebbe diventare inconsistente. Questo può succedere se entrambi i processi di richiesta vedono la copia come disponibile e procedono con l'assegnazione, portando a una situazione in cui il libro appare come prestato a più utenti contemporaneamente. L'uso di mutex garantisce che solo un processo possa effettuare la richiesta di prestito per quel libro alla volta, assicurando che il database rimanga coerente.

2.7 Gestione delle notifiche

Quando l'utente accede al sistema, il server verifica se ha prestiti scaduti che devono essere restituiti. Se l'utente ha prestiti scaduti, il server invia una notifica al client informandolo della situazione.

2.8 Funzioni speciali

La funzione `send_string_segmented` invia una stringa al client tramite il socket, suddividendo la stringa in segmenti di lunghezza massima definita da `MAX_REQUEST_BUFFER_LENGTH`.

Iterativamente ogni segmento viene inviato al client in maniera ordinata.

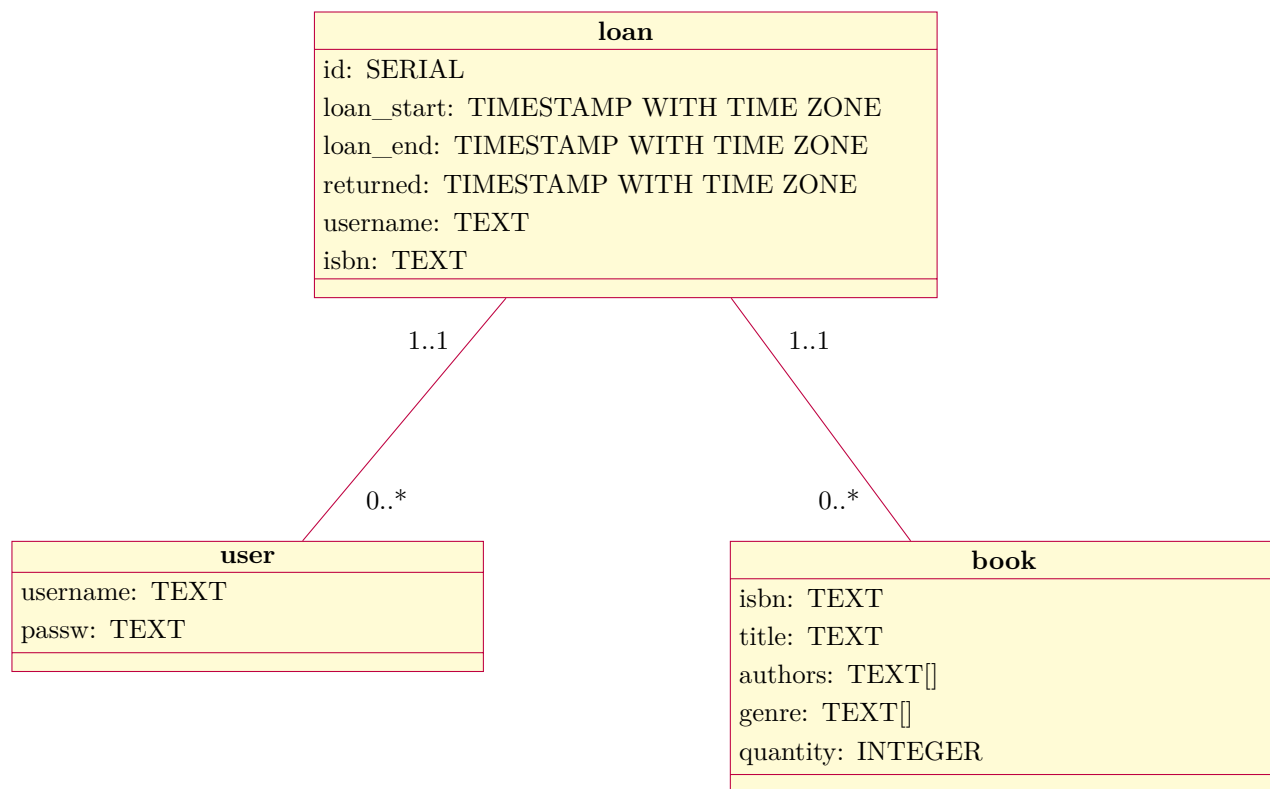
La funzione `recv_and_compose_segmented_string`, riceve una stringa di lunghezza massima `MAX_REQUEST_BUFFER_LENGTH`; esegue questo processo iterativamente, accodando il nuovo segmento al segmento precedente.

Al termine, la funzione restituisce l'intera stringa.

Queste funzioni sono necessarie per gestire l'invio e la ricezione di dati di lunghezza variabile poiché TCP non garantisce che tutti i dati vengano ricevuti in una singola operazione, specialmente se i dati sono di grandi dimensioni.

Database

3.1 Schema



3.2 Descrizione delle tabelle

3.3 Query principali utilizzate