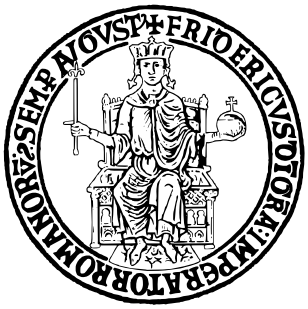
Università degli Studi di Napoli Federico II



Scuola Politecnica e delle Scienze di Base

Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione

Corso di Laurea Triennale in Informatica

Corso di Laboratorio di Sistemi Operativi

Progetto Client-Server: Videoteca

Prof.ssa
Alessandra Rossi

Studenti
Davide Galdiero N86004093
Francesco La Frazia N86002238

Anno Accademico 2024-2025

Indice

1 Analisi dei requisiti

1.1 Obiettivo del progetto

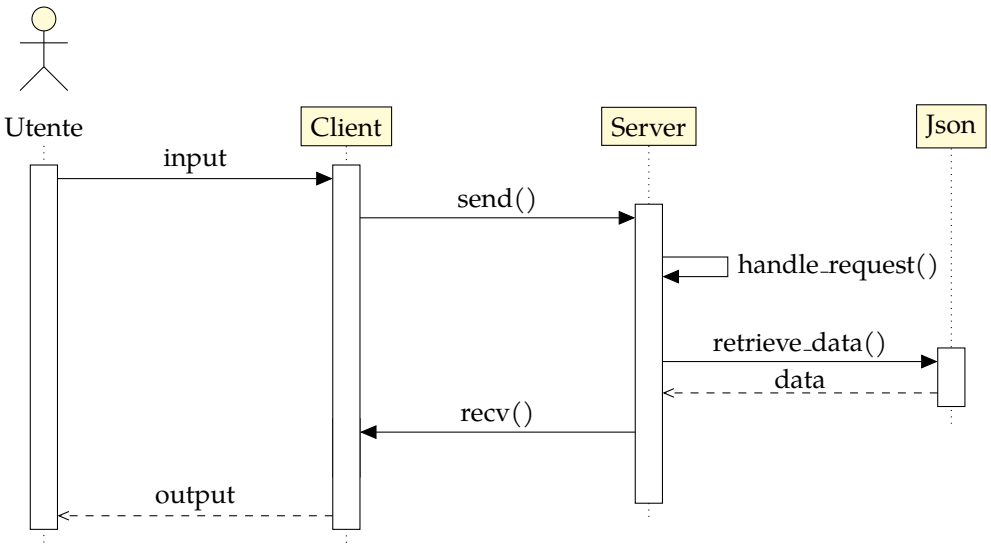
Il progetto consiste nella simulazione di un sistema che modella una videoteca per un numero non specificato di utenti. Gli utenti devono potersi registrare e accedere al server per interagire con la videoteca, che contiene una lista di film disponibili. Il sistema deve gestire il noleggio dei film, tenere traccia delle copie disponibili e delle copie prese in prestito, oltre a monitorare le scadenze di restituzione.

1.2 Funzionalità principali

- **Registrazione e Login degli Utenti**: Gli utenti devono potersi registrare e accedere per utilizzare il Sistema.
- **Ricerca dei film**: Gli utenti possono cercare il film disponibili. La ricerca può essere fatta per titolo o per genere.
- **Noleggio dei film**: Gli utenti possono noleggiare i film se ci sono ancora copie disponibili. Gli utenti possono mettere nel carrello tutti i film che intendono noleggiare e poi effettuare il check-out.
- **Restituire un film**: Gli utenti devono poter restituire un film.

2 Implementazione

Le funzionalità descritte sono rappresentate in maniera generale con il seguente sequence diagram:



2.1 Scelte tecnologiche

Il progetto utilizza il linguaggio C per lo sviluppo dell'applicazione client-server. Il server e il client comunicano tramite socket, permettendo lo scambio di dati in rete.

Data la natura accademica del progetto, per la persistenza dei dati si è scelto di usufruire di una soluzione che sfrutta i file JSON come tabelle di dati. Per poter garantire la stabilità di un server multi-client, il sistema sfrutta l'architettura multi-thread, con un controllo degli accessi ai file sincronizzato tramite l'utilizzo di mutex.

2.2 Struttura progetto

Il progetto è organizzato in una struttura di cartelle che separa logicamente le varie 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:

```
Videoteca
+-- videoteca-client/
|   +-- c/
|   |   +-- client.c
|   |   +-- hash.c
|   |   +-- socket.c
|   +-- h/
|   |   +-- client.h
|   |   +-- hash.h
|   |   +-- socket.h
|   +-- Dockerfile
|   +-- Makefile
|   +-- main.c
+-- videoteca-server/
|   +-- data/
|   |   +-- films.json
|   |   +-- users.json
|   +-- server/
|   |   +-- auth/
|   |       +-- auth.c
|   |       +-- auth.h
|   |   +-- cJSON/
|   |       +-- cJSON.c
|   |       +-- cJSON.h
|   |   +-- json_db/
|   |       +-- json_db.c
|   |       +-- json_db.h
|   |   +-- rental/
|   |       +-- rental.c
|   |       +-- rental.h
|   |   +-- Dockerfile
|   |   +-- Makefile
|   |   +-- main.c
+-- docker-compose.yml
```

Per il server è utilizzata la cartella `videoteca-server` contenente la cartella `data`, con i file JSON, e la cartella `server`, a sua volta contenente le cartelle che rappresentano i vari moduli che compongono il server, con i file di header e build.

Per il client è utilizzata la cartella `videoteca-client` contenente i file di header e di build divisi nelle cartelle rispettivamente `h` e `c`.

2.3 Comunicazione Client-Server

La comunicazione tra client e server avviene, come detto, attraverso l'utilizzo di apposite socket. Per i messaggi che il client inoltra al server, in base alle operazioni richieste, si è scelta una struttura ben precisata, come si mostrerà nel dettaglio. Tutte le risposte del server sono caratterizzate da un carattere di terminazione specifico, al fine di agevolare il lavoro del client nel recepire il messaggio nella sua interezza.

2.3.1 Registrazione e Login

L'utente inserisce le credenziali per la registrazione o per il login. Il client cripta la password inserita dall'utente e invia al server il tipo di richiesta, username e password crittografata. Successivamente, il server, controllando il file json relativo ai dati degli utenti, risponde con un messaggio o di errore o di conferma. Qualora l'utente abbia qualche noleggio in scadenza, il server inoltrerà anche un messaggio di avviso con il relativo titolo in scadenza e la data massima per la restituzione. Se la data del noleggio è stata superata, il server inoltrerà un messaggio di avviso relativo al titolo che deve essere restituito. La comunicazione dal client al server per le operazioni di registrazione e login è la seguente:

- **Login**: "`LOGIN|username|password`".
- **Registrazione**: "`REGISTER|username|password`"

2.3.2 Ricerca dei film

L'utente può effettuare delle ricerche sulla lista di film disponibili. Può cercare per titolo, genere, oppure in base alle votazioni che i film hanno registrato. Inserisce in input il tipo di ricerca che vuole effettuare e poi i parametri necessari per la ricerca. Il client invia al server il tipo di richiesta e il relativo parametro. Il server, dopo aver smistato correttamente la richiesta, risponde con una lista di film con tutti i dettagli, oppure una lista vuota come stringa. In caso di lista vuota, il client offre all'utente la possibilità di effettuare una nuova ricerca. La comunicazione verso il server è così costruita:

- **Ricerca per genere**: "`SEARCH|GENRE|genere_da_cercare`".
- **Ricerca per titolo**: "`SEARCH|TITLE|titolo_da_cercare`".
- **Ricerca per popolarità**: "`SEARCH|POPULAR|numero_elementi_da_visualizzare`".

2.3.3 Noleggio dei film

L'utente può noleggiare un film se ancora ci sono copie disponibili. Dopo aver cercato i film, può decidere se noleggiarne uno o più fino a un massimo di 5 alla volta. Dopo aver scelto, l'utente deve inserire in input l'id del film selezionato e poi decidere se

andare al checkout o selezionare altri film. In fase di checkout, una volta confermata l'operazione, l'utente deve inserire una data (in formato YYYY-MM-DD) entro la quale intende restituire il prestito. Il client invia questi dati al server. A meno di errori in fase di aggiornamento delle copie disponibili o in fase di salvataggio del prestito al relativo utente, il server ritorna una conferma di avvenuta operazione. La richiesta verso il server è composta come segue:

- **Noleggio**: "`RENT|username|film_id|return_date`".

2.3.4 Restituzione dei film

L'utente deve poter restituire un film. Quando l'utente va alla sezione per la restituzione di un film, il client inoltra verso il server prima una richiesta per ricevere la lista di titoli noleggiati dall'utente. Una volta ricevuta la lista dei film noleggiati come stringa, viene mostrata all'utente, che seleziona uno dei suoi film noleggiati per restituirlo, indicandone il relativo id; il client invia quindi al server una richiesta di restituzione con l'username dell'utente che sta effettuando la restituzione e l'id del film restituito. Il protocollo di comunicazione è il seguente:

- **Mostra la lista di noleggi**: "`MYRENTALS|username`".
- **Restituzione di un film**: "`RETURN|username|film_id`"

2.4 Gestione multi-thread

Per consentire al server di gestire molteplici richieste contemporaneamente, si è optato per una soluzione multi-thread. Alla ricezione di una nuova connessione, una volta validata, il server creerà un nuovo thread, che si occuperà delle richieste provenienti da quel singolo client. L'idea di creare nuovi thread, piuttosto che utilizzare altre strade, quali la fork del processo padre, è dovuta al fatto che con i thread le risorse sono condivise, e i singoli thread tenteranno di accedere alle medesime risorse (i file `films.json` e `users.json`) per poter elaborare correttamente le richieste dei rispettivi client. La concorrenzialità degli accessi alle risorse viene gestita tramite l'utilizzo di appositi mutex associati ai file.

3 Json

3.1 Scelte implementative

Al fine di preservare la persistenza dei dati, simulando l'utilizzo di un database, si è scelto di servirsi di file `users.json` e `films.json`, utilizzati nel sistema di gestione della videoteca. I due file rappresentano rispettivamente gli utenti registrati al sistema e il catalogo di titoli messi a disposizione dalla videoteca per il noleggio. Per poter elaborare correttamente i contenuti dei due file, è stata utilizzata la libreria `cJSON`, sfruttando blocchi di codice thread-safe, tramite l'utilizzo di appositi mutex, per gestire correttamente la concorrenza degli accessi durante le operazioni sui file.

3.2 File `users.json`

Il file `users.json` memorizza i dati relativi agli utenti registrati e ai loro noleggi. La struttura complessiva è un **array JSON** di oggetti, dove ogni oggetto rappresenta un singolo utente.

3.2.1 Struttura dell'Oggetto Utente

Un tipico oggetto Utente è così definito:

```
[{
    "username": "NomeUtente",
    "password": "passwordCriptata",
    "prestiti": [{
        "film_id": 2,
        "return_date": "2025-04-06"
    }]
}]
```

- **username** (stringa): Lo username univoco dell'utente.
- **password** (stringa): La password dell'utente, memorizzata in maniera criptata.
- **prestiti** (array): Un array di oggetti Prestito, ciascuno dei quali rappresenta un prestito effettuato dall'utente.

L'oggetto Prestito è definito dai campi:

- **film_id** (stringa): ID del film richiesto in prestito.
- **return_date** (stringa): Stringa rappresentante la data massima impostata per la restituzione del prestito.

3.3 File `films.json`

Il file `films.json` funge da base dati per il catalogo di film disponibili. La struttura complessiva è un **array JSON** di oggetti, dove ogni oggetto rappresenta un singolo titolo disponibile.

3.3.1 Struttura dell'Oggetto Film

Un tipico oggetto Film è così definito:

```
[{
    "film_id": 1,
    "titolo": "Pap-Music - Animation for Fashon",
    "genere": "Animazione",
    "copie_totali": 12,
    "copie_disponibili": 2,
    "rating": 10
}]
```

- **film_id** (stringa): Identificativo del film.
- **titolo** (stringa): Titolo del film.
- **copie_totali** (stringa): Intero che rappresenta il numero totale di copie a disposizione della videoteca.
- **copie_disponibili** (stringa): Intero che rappresenta il numero totale di copie ancora disponibili per il noleggio.

4 Docker

4.1 Client

Il dockerfile del client utilizza l'immagine di base gcc per compilare e prepara l'ambiente per la compilazione del client:

1. Installa le dipendenze tra cui la libreria per il crypting della password
2. Imposta la directory di lavoro a /app
3. Copia il codice sorgente nel container
4. Compila il programma
5. Lascia il container attivo

4.2 Server

Viene applicata la stessa logica di creazione del docker del client. In più, espone la porta 8080 per la comunicazione con i client.

4.3 Compose

Il docker compose è configurato per la gestione dei 2 container:

1. **Container del Server**: crea un'immagine a partire dal dockerfile del server, viene rinominato come video_server, espone la porta e monta i volumi per i file json.
2. **Container del Client**: crea un'immagine a partire dal dockerfile del client, viene rinominato come video_client e viene aggiunta la dipendenza al server.