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 in Informatica

Progetto di Laboratorio di Sistemi Operativi

Simulazione di un sistema modellante una videoteca

Autori
Simone Iavarone, N86004664
Simone Veniero, N86003919

Anno Accademico 2024–2025

Contents

Chapter 1

Introduzione

È stato richiesto lo sviluppo di un'applicazione client-server che si occupasse di realizzare la simulazione di un sistema che modella una videoteca per un numero imprecisato di utenti, che possono autenticarsi, consultare una lista di film e noleggiare questi ultimi.

È presente un venditore che fa da amministratore del sistema, le cui facoltà sono di aggiungere nuovi film al catalogo, visualizzare i noleggi effettuati dagli utenti ed eventualmente, qualora non abbiano restituito il film oltre la data di scadenza, ricordarlo con un messaggio all'interessato.

Gli utenti hanno facoltà di selezionare i film e di aggiungerli ad un carrello, dal quale possono poi finalizzare il noleggio con un sistema di Checkout che si occupa di verificare se, concorrentemente, altri utenti non abbiano già noleggiato l'ultima copia del film nel carrello.

1.1 Scelta delle tecnologie

Al gruppo di sviluppo è stato richiesto che il server sia sviluppato rigorosamente in linguaggio C, mentre è stata data libertà di scelta sul linguaggio (o i linguaggi) di sviluppo del client. Analogamente, il gruppo ha avuto facoltà di decidere autonomamente la tipologia di database.
Lo sviluppo del progetto è avvenuto quasi totalmente su dispositivi Windows

1.1.1 Backend

All'interno del perimetro del linguaggio C, si è fatto utilizzo delle librerie standard naturalmente più consolidate nella gestione di thread e di connessioni TCP, quali "pthread.h", "sys/types.h", "arpa/inet.h", "netinet/in.h/ e "sys/socket.h". In un momento iniziale si era contemplato l'utilizzo della libreria Winsock per rendere il sistema compatibile con Windows senza l'uso di WSL, ma il gruppo ha optato già dai primissimi momenti per l'alternativa più affermata.

Pressoché tutte le altre librerie utilizzate appartengono alla libreria standard - inclusa la gestione dei thread - eccezion fatta per la libreria "sqlite3.h" necessaria per la gestione del database e la libreria "models.h" realizzata dal gruppo stesso e contenente alcune struct utili nella comunicazione fra le tre parti del software.

1.1.2 Frontend

Data la libertà sulla scelta del linguaggio il gruppo di sviluppo si è trovato davanti alla necessità di individuare un linguaggio che consentisse la realizzazione di un'interfaccia utente pulita, semplice ed intuitiva.

La semplicità di Java ed in particolare del pacchetto JavaFX è giunta come una soluzione naturale, considerata anche la precedente esperienza del gruppo di sviluppo con il linguaggio. Questa scelta ha portato alla naturale conclusione di sviluppare un sistema desktop. Si è volutamente scelto un design lineare e diretto per mettere in risalto la funzionalità e la gestione della concorrenzialità.

L'utilizzo di FXML ha permesso la stesura di classi semplici, ciascuna dal ruolo ben definito ed inquadrato nel sistema grazie alla facoltà di separare la logica dall'interfaccia, facilitando l'eventuale manutenzione.

1.1.3 Database

Interpretato il sistema come la "virtualizzazione" di un esercizio commerciale, è stata una scelta altrettanto naturale quella di Sqlite come tecnologia DB per un sistema che non prevede di essere eccessivamente scalabile. La sua semplicità e velocità avrebbero permesso un recupero di dati rapido e l'ottima libreria integrativa col linguaggio C facilita molto un'interazione sicura fra il server ed il database.

Chapter 2

Realizzazione del Server

Lo sviluppo e l'espansione del server sono stati effettuati con una logica incrementale. Nella fase iniziale sono state realizzate le struct presenti in "models.h" ed il basilare codice di apertura della connessione col client.
La progettazione del client come un programma che gestisce una connessione locale col server ha portato allo sviluppo della funzione "gestione_client()", che gestisce nella sostanza tutto il ciclo vitale dell'applicazione.
Questa funzione è stata il cuore pulsante dello sviluppo del server e nelle sue iterazioni si è dapprima ingrandita per poi snellirsi, constatata la necessità di gestirla in modo più modulare, opportunamente rispetto ai diversi tipi di richieste che deve gestire.

2.0.1 gestione_client()

Mentre la connessione col client è aperta, il server riceve e gestisce una serie di richieste contrassegnate da un valore numerico fra 1 e 15, ciascuno dei quali associato ad un'azione (Registrazione, Login, aggiunta al carrello, checkout etc.).
L'applicazione si assicura al ricevimento di una delle suddette richieste di chiudere il mutex globale finché essa non viene processata. L'interazione fra il client ed il database è assolutamente evitata.
La gestione degli errori è delegata ad una piccola sezione della funzione denotata dalla label "cleanup", alla quale ogni errore è rimandato tramite un goto.

Chapter 3

Realizzazione del Client ed UI

Il client è stato sviluppato attraverso le conoscenze già acquisite nelle esperienze precedenti in termini di progettazione ed esecuzione.
La logica del client è divisa fra classi Model, che rappresentano con le dovute differenze le tabelle del database e i vari Controller relativi ai file FXML. I primi hanno i classici metodi tipici dei modelli: getter e setter, diversi costruttori tramite l'overloading e qualche metodo ausiliare.
I secondi gestiscono la comunicazione col server e la generazione di contenuto dinamico nelle varie finestre dell'interfaccia grafica.
Il client comunica col server attraverso dei metodi che gli inviano delle richieste numerate.

Chapter 4

Docker

L'utilizzo di Docker, richiesto dalla traccia, è stato complesso a causa dell'implementazione del progetto in quanto Docker non supporta nativamente interfacce grafiche. La soluzione è l'X11 forwarding, ovvero l'uso di un "server X" che gestisce il lato grafico dell'applicazione.
Ci siamo avvalsi di VcXsrv, un X server open source per Windows, ma utilizzare il programma su Linux richiede necessariamente l'uso di software diversi.
Il file yml nella root directory si occupa quindi sia di costruire il container per il server che per il client, oltre che specificare la presenza dell'X server.