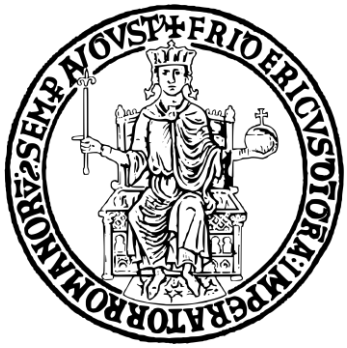

Laboratorio di Sistemi Operativi 2023 – 2024

La Libreria

Francesco Coppola & Alfredo D’Andrea

The CDs

Indice

Sommario

1 INTRODUZIONE AL SISTEMA

"La Libreria" è un sistema informativo che si occupa della gestione di una libreria offrendo funzionalità intuitive per raggiungere al meglio tale scopo. Il sistema è distribuito, infatti, dovrà comprendere due applicazioni: un client e una gestione server.

Requisiti assegnati:

1. Il proprietario può consultare l'elenco completo dei libri presenti nella libreria.

2. l proprietario può visualizzare il numero totale di copie disponibili per ogni libro. È possibile controllare quante copie di un libro sono attualmente disponibili, cioè non ancora prese in prestito. Per ogni copia prestata, il proprietario può accedere alle seguenti informazioni: identificativo del cliente che ha preso in prestito la copia, data in cui il cliente ha effettuato il prestito e data entro cui il cliente deve restituire il libro.

3. Il proprietario ha la possibilità di contattare direttamente gli utenti che hanno superato la data di scadenza del prestito senza ancora aver restituito il libro.

4. Gli utenti hanno la possibilità di registrarsi e accedere al server per utilizzare i servizi offerti dalla libreria.

5. Gli utenti possono cercare un libro utilizzando diversi criteri come genere, nome o dalla lista dei libri disponibili.
Per il prestito di libri, se ci sono copie disponibili di un libro, gli utenti possono prendere in prestito il libro desiderato.
Per il limite di prestito, gli utenti possono prendere in prestito un numero K di libri, dove K è determinato dal libraio.

6. Gli utenti possono aggiungere i libri desiderati al proprio carrello.
Per il check-out, una volta inseriti tutti i libri nel carrello, gli utenti possono procedere con il check-out per completare il processo di prestito.

7. Nel caso in cui un altro utente abbia già preso l'ultima copia disponibile di un libro, l'utente interessato riceverà una notifica al riguardo.

2 INDIVIDUAZIONE TARGET UTENTI

Il bacino di utenza del Sistema comprende:

- Proprietari
- Clienti

Secondo un'indagine condotta nel 2020 dall'American Booksellers Association (ABA), il 70% delle librerie indipendenti negli Stati Uniti ha implementato almeno un software di gestione per aiutare nelle operazioni quotidiane, come gestione delle scorte e del personale.

Alcune delle motivazioni dietro l'adozione di soluzioni software per la gestione delle librerie includono:

- La crescente necessità di adattarsi alle nuove esigenze dei clienti, comprese le modalità di acquisto online e la gestione degli ordini online durante la pandemia di COVID-19.
- L'importanza della trasformazione digitale per rimanere competitivi nel mercato in rapida evoluzione dell'editoria.
- Il desiderio di ottimizzare le operazioni interne, consentendo al personale di concentrarsi su compiti più strategici e di migliorare l'efficienza complessiva del negozio.

Inoltre, uno studio condotto dalla National Retail Federation (NRF) ha evidenziato che l'uso di cataloghi digitali e piattaforme online può influenzare positivamente le vendite in negozio, fornendo ai clienti un'esperienza di acquisto integrata e aumentando l'engagement con il pubblico attraverso canali digitali.

3 DOCUMENTO DI DESIGN DEL SISTEMA (PROGETTAZIONE)

3.1 System Design

3.1.1 Analisi dell'architettura con esplicita definizione dei criteri di design

CRITERI DI DESIGN

Sono stati scelti dei compromessi tra i seguenti 5 fondamentali criteri di design:

Performance: l'ottimizzazione delle prestazioni è stata una priorità fondamentale durante lo sviluppo del software gestionale per la libreria. Poiché il sistema coinvolge l'interazione di diversi processi o thread che comunicano attraverso socket, garantire elevate prestazioni è stato un obiettivo chiave.

Affidabilità: essendo un software di tipo gestionale che ha principalmente lo scopo di dividere i costi computazionali all'interno di uno stesso processo, l'affidabilità non è stato uno dei requisiti principali su cui ci siamo focalizzati. Nonostante ciò, è stato fondamentale separare gli utenti dall'avere un accesso diretto al database tramite la creazione di un lato server che potesse gestire le richieste.

Costi: è un progetto universitario ma supponendo non lo fosse avremo mantenuto dei costi abbastanza ridotti.

Mantenimento: investimento di un tempo maggiore di progettazione e sviluppo per avere una maggiore estendibilità e scalabilità. È necessario però trovare un trade-off che equilibri la qualità del software e il tempo a disposizione per realizzarlo.

Usabilità: un basilare studio di usabilità è stato da noi sviluppatori eseguito per garantire all'utente una buona esperienza.

Nel complesso quindi si sacrifica del tempo iniziale e usabilità per favorire maggiore performance, mantenimento e bassi costi.

ARCHITETTURA UTILIZZATA
Architettura Three-Tier
Il tipo di architettura scelta risulta quella a tre livelli:

Livello di presentazione: Frontend in Command Line Interface
Livello applicazione (business logic): Backend e Server del Frontend
Livello dati: Database

3.1.2 Descrizione motivazione delle scelte tecnologiche adottate

Docker
Per la realizzazione del software, abbiamo adottato un'architettura basata su container Docker, che offre un ambiente di sviluppo e distribuzione altamente scalabile e portabile sfruttando tutti i suoi vantaggi.

Il sistema è suddiviso in componenti distinti, ognuno dei quali viene eseguito all'interno di un container Docker separato. Questo permette di isolare e gestire in modo efficiente ogni parte dell'applicazione, garantendo coerenza e facilità di deployment.
Ogni container Docker ospita un'istanza dell'applicazione o un servizio specifico, come il backend ed il database (si poteva far partire anche il frontend ma per scopi didattici, non è stato aggiunto nei container). Questa suddivisione modulare favorisce la scalabilità orizzontale e la manutenibilità del sistema nel tempo.

Le motivazioni che ci hanno portato a fare uso di un approccio con i container Docker sono stati:

- Isolamento delle risorse: ogni container Docker ha le proprie risorse isolate, inclusi filesystem, memoria e CPU, garantendo che le applicazioni eseguite al loro interno non interferiscano tra loro.
- Portabilità: i container Docker sono leggeri e portabili, il che significa che l'applicazione può essere eseguita su qualsiasi piattaforma che supporta Docker, facilitando la migrazione tra ambienti e fornitori di servizi cloud.
- Scalabilità: grazie alla loro natura leggera e modulare, i container Docker consentono una scalabilità rapida e flessibile dell'applicazione. È possibile aumentare o diminuire il numero di istanze di un servizio in base alle esigenze di carico, senza dover ridimensionare l'intera infrastruttura.
- Gestione semplificata: Docker fornisce strumenti potenti per gestire i container, come Docker Compose per l'orchestrazione dei servizi che è uno strumento che consente di definire e gestire multi-container Docker applications.

In un file YAML denominato `docker-compose.yml`, sono stati specificati quasi tutti i servizi necessari per l'applicazione: il backend e il database.
Per ciascun servizio, sono stati definiti parametri come le dipendenze, le variabili d'ambiente, i volumi montati e le porte esposte. Questa configurazione permette di personalizzare il comportamento di ogni servizio in base alle esigenze dell'applicazione.
Docker Compose ha gestito l'avvio e l'interazione tra i container, garantendo che fossero avviati nell'ordine corretto e che le dipendenze tra di essi fossero gestite correttamente. Inoltre, ha semplificato operazioni comuni come la scalabilità dei servizi o la gestione dei log.

Il Dockerfile è un file di testo che contiene le istruzioni per la creazione di un'immagine Docker. È stato utilizzato per definire l'ambiente di esecuzione per ciascun servizio o componente dell'applicazione. Sono stati definiti due Dockerfile differenti: uno per gestire la creazione (in caso non esistesse) e l'avvio del database ed un altro per compilare ed eseguire il backend.

Il back-end svolge un ruolo importante nella gestione dei dati di un'applicazione. Quando un client invia una richiesta, il back-end la riceve e la elabora, quindi la propaga al database. Questo processo è fondamentale per garantire la sicurezza dei dati, poiché il back-end impedisce l'accesso diretto al database, fornendo una barriera protettiva e garantendo un sistema di gestione dei dati affidabile e robusto che supporta le esigenze dell'applicazione e garantisce che i dati siano sempre accessibili e sicuri.
In sintesi, il back-end è un componente essenziale per la gestione dei dati in un'applicazione, che fornisce sicurezza, flessibilità e affidabilità nella gestione dei dati.

Per la differenziazione dei vari client connessi, è stato fatto uso dei thread piuttosto che di nuovi processi principalmente per i seguenti motivi:

1. Efficienza delle risorse: i thread condividono lo stesso spazio di memoria del processo genitore, riducendo l'overhead rispetto alla creazione di processi separati, che richiederebbero la duplicazione dello spazio di memoria.
2. Semplificazione della gestione: utilizzando i thread, è più semplice coordinare e gestire le operazioni relative ai client loggati, poiché condividono le stesse risorse e lo stesso contesto di esecuzione.
3. Scalabilità: l'uso dei thread consente di gestire un grande numero di client contemporaneamente in modo efficiente, senza dover creare e gestire un processo separato per ciascuno di essi (operazione molto costosa computazionalmente).

Per la comunicazione backend-frontend sono state usate le socket per una comunicazione bidirezionale tra il backend e il frontend, consentendo loro di scambiare dati in tempo reale. Il protocollo usato è stato il TCP/IP per evitare una possibile perdita di informazioni e dati.

In sintesi, l'uso dei thread anziché dei processi per differenziare i client loggati offre vantaggi in termini di efficienza delle risorse, velocità di comunicazione e gestione semplificata, mentre l'uso delle socket per la comunicazione backend-frontend offre una soluzione affidabile, sicura e flessibile per lo scambio di dati tra le diverse componenti dell'applicazione.