Documentazione Cross

Martini Matteo 636694

20 marzo 2025

Indice

1 Scelte Progettuali **1**
- 1.1 Organizzazione delle unità di codice . . . 1
- 1.2 Comunicazione . . . 1
- 1.3 Gestione di Orderbook e Registrazioni . . . 1
- 1.4 Sincronizzazione . . . 2
- 1.5 Design Pattern . . . 2

2 Schema Generale dei Thread Attivati **2**

1 Scelte Progettuali

Il progetto è stato realizzato seguendo i principi SOLID per garantire modularità, scalabilità e manutenibilità

1.1 Organizzazione delle unità di codice

La suddivisione intensiva in package garantisce la modularità per le varie funzionalità e migliora la fruibilità del codice. Di seguito verranno elencati i package principali

- **Client**: contiene le task da eseguire lato Client e la classe che verrà istanziata per usufruire del servizio
- **Commands**: contiene tutti i comandi che possono essere utilizzati dagli utenti, insieme alla factory per istanziarli
- **Communication**: contiene i protocolli di comunicazione utilizzati insieme ai tipi di messaggio definiti per la comunicazione
- **Config**: contiene le classi per configurare Client e Server
- **Executables**: contiene ServerMain e ClientMain che eseguiranno rispettivamente Server e Client del servizio Cross
- **JsonAccessedData**: contiene tutti i vari file Json utilizzati per mantenere informazioni persistenti
- **Server**: contiene le task da eseguire lato Server e la classe che verrà istanziata per creare il Server centrale del servizio Cross
- **Utils**: contiene varie classi di utilità che semplificano l'esecuzione del codice

1.2 Comunicazione

La comunicazione definita nel package **Communication** contiene i due protocolli utilizzati:

- **TCP**: Protocollo maggiormente utilizzato per la comunicazione Client-Server mediante il quale il Client può inviare richieste al Server. La conessione viene instaurata all'avvio del Client e persiste fino alla chiusura dello stesso o di problemi che causano la chiusura del Server
- **UDP**: Protocollo utilizzato per notificare gli utenti della finalizzazione delle transazioni legate agli ordini da loro piazzati nell'orderbook

1.3 Gestione di Orderbook e Registrazioni

L'orderbook viene realizzato mediante un file Json diviso in due campi principali **askMap** e **bidMap** che contengono i relativi ordini Questa scelta consente di persistere i dati sugli ordini anche dopo la chiusura del Server, oltre a garantire una relativa semplicita nel caricamento in memoria degli ordini tramite una struttura dati apposita. La gestione delle registrazioni è stata gestita in maniera simile mediante un **"Userbook"** con una chiave usermap per rendere più agevole la traduzione da Json a struttura dati, inoltre per garantire maggiore sicurezza le password sono state criptate grazie alla libreria **BCrypt**.

1.4 Sincronizzazione

Per garantire consistenza in un ambiente concorrente vengono utilizzati metodi **synchronized** per proteggere sezioni critiche ed inoltre le varie strutture dati presenti nella collezione **java.util.concurrent**.

1.5 Design Pattern

Il progetto utilizza principalmente la **Simple Factory** per generare i comandi da inviare al Server. Questa generazione avviene sul Client sfruttando l'interfaccia **Values** la quale offre il metodo execute che il Server utilizzerà per eseguire correttamente i vari comandi disponibili. Inoltre questo approccio consente di mantenere invariato il formato di messaggi,**Message{Operation,Values}** che il Client invia al server e facilita l'aggiunta di comandi in quanto è sufficiente creare una nuova implementazione di Values.

2 Schema Generale dei Thread Attivati