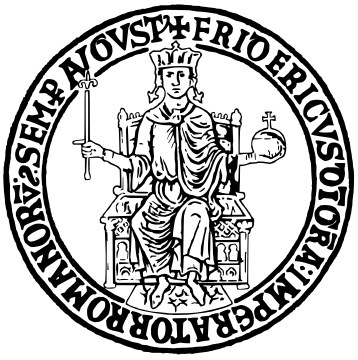
Laboratorio di Sistemi Operativi

Telefono Senza Fili



Documentazione Progetto

Davide Gargiulo *Matricola: N86004689*
Francesco Donnarumma *Matricola: N86004658*
Gennaro De Gregorio *Matricola: N86004591*

Anno Accademico 2024/2025
October 31, 2025

Contents

I

Introduzione

Questo progetto consiste nella realizzazione di un'applicazione client-server per poter permette di giocare con altri client al gioco del **telefono senza fili**.

Le tecnologie richieste per quest'applicazione sono le seguenti:

- **C**: linguaggio per il backend;
- **Java**: linguaggio scelto per il frontend;
- **JavaFx**: libreria per la realizzazione dell'interfaccia grafica in Java;
- **Mongoose**: libreria per la gestione delle connessioni WebSocket in C;
- **Docker**: strumento per la containerizzazione dell'applicazione;
- **Maven**: strumento di gestione delle dipendenze per Java;
- **PostgreSql**: sistema di gestione di database relazionali.

Ogni client può svolgere determinate azioni come:

- Registrarsi al server;
- Effettuare il login;
- Creare una nuova partita;
- Unirsi a una partita esistente;
- Unirsi a una partita già in corso come spettatore;
- Giocare.

Il server, invece, si occupa di:

- Gestire le connessioni dei client;
- Mantenere lo stato delle partite;
- Gestire la logica di gioco;
- Comunicare con i client tramite WebSocket.

II

Architettura Generale

L'architettura dell'applicazione è basata sul modello **client-server**, in cui il **server** gestisce la logica di gioco e le connessioni dei **client**, mentre i **client** forniscono l'interfaccia utente e comunicano con il **server** tramite WebSocket.

Il **server** è implementato in C utilizzando la libreria Mongoose per gestire le connessioni WebSocket. Si occupa di mantenere lo stato delle partite, gestire la logica di gioco e comunicare con i **client**.

La scelta della libreria **Mongoose** è stata dettata dalla sua leggerezza e facilità d'uso, inoltre supporta nativamente il protocollo WebSocket, rendendo più semplice l'implementazione della comunicazione in tempo reale tra **client** e **server**. Inoltre la comunicazione tramite WebSocket viene gestita tramite protocollo HTTP e oggetti JSON, facilitando l'interscambio di dati strutturati tra le due parti.

I dati rilevanti sono salvati in un database **PostgreSQL**, che offre robustezza e scalabilità per la gestione delle informazioni degli utenti e delle partite. Il nostro database contiene due tabelle principali, ovvero la tabella **users** e la tabella **lobby**.

- La tabella **users** contiene le informazioni degli utenti registrati, come username, password (hashata per motivi di sicurezza).
- La tabella **lobby** contiene le informazioni delle lobby create dai vari utenti, come l'ID della lobby, l'ID del creatore, lo stato della lobby (in attesa, in corso, terminata) e il numero di giocatori attuali.

Inoltre la scelta di PostgreSQL è derivata dal fatto che tutti i membri del team avevano già esperienza con questo sistema di gestione di database relazionali, facilitando così la fase di sviluppo e integrazione con il backend in C.

Per combinare le informazioni tra le due tabelle, utilizziamo una terza tabella **lobby_players**, che funge da tabella di collegamento tra gli utenti e le lobby. Questa tabella contiene gli ID degli utenti e gli ID delle lobby a cui sono associati, ma anche il loro status (attivo, spettatore).

III

Struttura del codice

In questo capitolo viene descritta l'architettura generale del codice, sia back-end scritto in C, sia front-end scritto in Java, illustrando le principali componenti e il loro funzionamento all'interno dell'applicazione.

Nella root del progetto è presente il file **docker-compose.yml** responsabile dell'orchestrazione dei vari container Docker che compongono l'applicazione. Questo file definisce i servizi necessari per eseguire l'applicazione, tra cui il servizio **backend** (server C), il servizio **frontend** (applicazione JavaFX) e il servizio **database** (PostgreSQL).

3.1

Struttura dei file back-end

La struttura dei file del codice back-end è organizzata in modo da separare le diverse funzionalità e facilitare la manutenzione del codice. Nella cartella principale del **back-end** troviamo il **Dockerfile** dove abbiamo la configurazione per la creazione dell'immagine Docker del server, dove sono specificate le dipendenze necessarie, come il sistema operativo, la libreria Mongoose per la gestione delle connessioni WebSocket, la libreria **libpq-fe.h** per l'interfacciamento con il database PostgreSQL e gcc come compilatore C, responsabile della compilazione di tutti i sorgenti C del progetto. Inoltre troviamo il file **server.c** che rappresenta il file principale dell'applicazione.

Troviamo poi una cartella **controllers** che contiene tutti i file responsabili del routing delle richieste ricevute dai client, gestendo le varie operazioni come la registrazione, il login, la creazione di lobby. Inoltre troviamo una cartella chiamata **moduli** che contiene i file responsabili della logica di business dell'applicazione, come la gestione delle lobby, degli utenti e delle partite. Questi moduli interagiscono con il database PostgreSQL per salvare e recuperare i dati necessari. Infine, nella cartella **moduli**, troviamo un'ultima sottocartella chiamata **websocket** che contiene i file specifici per la gestione delle comunicazioni WebSocket tra il server e i client, implementando le funzionalità d'invio e ricezione dei messaggi in tempo reale.

3.2

Struttura dei file front-end

La struttura dei file del codice front-end è organizzata in modo da separare le classi di gestione delle varie schermate dagli **utils** e i **DTO**.

All'interno della cartella principale del **front-end** troviamo la cartella **Controller** che contiene tutte le classi responsabili della gestione delle varie schermate dell'applicazione e dell'interazione con l'utente. Ogni classe controller è associata a una specifica schermata, come la schermata di login, la schermata di registrazione, la schermata della lobby e la schermata di gioco. Inoltre, troviamo la cartella **DTO** (Data Transfer Object) che contiene le classi utilizzate per il trasferimento dei dati tra il front-end e il back-end. Queste classi rappresentano le strutture dati necessarie per inviare e ricevere informazioni, come i dettagli dell'utente, le informazioni della lobby.

Tra gli **utils**, troviamo la classe **LobbyWebSocketClient** che si occupa di gestire la creazione di WebSocket e la ricezione/invio di messaggi tramite la stessa. Troviamo anche la classe **LanguageHelper** che si occupa della conversazione tra il client e l'API di **LibreTranslate** per la traduzione dell'UI e dei messaggi di gioco.

Infine possiamo trovare la cartella **resources** che contiene immagini, file fxml e css per la creazione e personalizzazione dell'interfaccia utente.