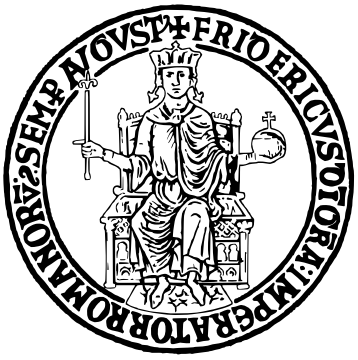
Laboratorio di Sistemi Operativi

Telefono Senza Fili



Documentazione Progetto

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

Anno Accademico 2024/2025
November 1, 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.

IV

Implementazione del Backend

4.1

Server C e Mongoose

Il server è implementato in C utilizzando la libreria Mongoose, che fornisce un framework leggero e efficiente per la gestione delle connessioni HTTP e WebSocket. Il file **server.c** rappresenta il punto di ingresso dell'applicazione e si occupa dell'inizializzazione del server, della configurazione delle route e della gestione del ciclo di vita dell'applicazione.

All'avvio, il server esegue le seguenti operazioni:

- Configura il gestore di eventi Mongoose per processare le richieste HTTP e WebSocket;
- Registra le route per le varie operazioni (registrazione, login, gestione lobby);
- Avvia il loop principale che rimane in ascolto delle connessioni in arrivo.

La comunicazione tra client e server avviene tramite messaggi JSON strutturati, che vengono parsati e processati dal server. Ogni messaggio contiene un campo `type` che identifica il tipo di operazione richiesta e eventuali parametri necessari per l'esecuzione dell'operazione.

4.2

Gestione del Database

La connessione al database PostgreSQL viene stabilita quando è necessario utilizzare il database. Il modulo di gestione del database espone funzioni per eseguire query SQL e recuperare i risultati in modo sicuro, prevenendo attacchi di SQL injection attraverso l'uso di prepared statements.

Le principali operazioni sul database includono:

- **Inserimento utenti**: quando un nuovo utente si registra, i suoi dati vengono inseriti nella tabella `users` con la password hashata utilizzando un algoritmo di hashing sicuro;
- **Autenticazione**: durante il login, viene verificata la corrispondenza tra username e password hashata;
- **Gestione lobby**: creazione, aggiornamento e cancellazione delle lobby nella tabella `lobby`;
- **Associazione giocatori**: gestione delle relazioni molti-a-molti tra utenti e lobby attraverso la tabella `lobby_players`.

4.3

Controller e Routing

La cartella **controllers** contiene i file che implementano la logica di routing delle richieste HTTP. Ogni controller è responsabile di gestire un insieme specifico di operazioni correlate. I principali controller implementati sono:

- **auth_controller.c**: gestisce le operazioni di autenticazione, incluse registrazione e login;
- **lobby_controller.c**: gestisce la creazione, l'aggiornamento e la cancellazione delle lobby;
- **user_controller.c**: gestisce le operazioni relative agli utenti, per recuperare le informazioni dell'utente.

Ogni controller implementa funzioni che ricevono come parametro la connessione Mongoose e il messaggio HTTP, estraggono i parametri necessari, invocano la logica di business appropriata e restituiscono una risposta JSON al client.

V

Implementazione del Frontend

5.1

Architettura JavaFX

Il frontend dell'applicazione è sviluppato in JavaFX, un framework moderno per la creazione di interfacce grafiche in Java. L'architettura segue il pattern MVC (Model-View-Controller), dove le viste sono definite in file FXML, i controller gestiscono la logica dell'interfaccia e i modelli rappresentano i dati.

La struttura dell'applicazione JavaFX è organizzata come segue:

- **View**: file FXML che definiscono la struttura e il layout delle schermate;
- **Controller**: classi Java che gestiscono gli eventi dell'interfaccia utente e coordinano l'interazione con il backend;
- **Model/DTO**: classi che rappresentano i dati trasferiti tra frontend e backend;
- **Utils**: classi di utilità per funzionalità comuni come la gestione WebSocket e la traduzione.

5.2

Schermate Principali

L'applicazione è composta da diverse schermate, ciascuna con una specifica funzionalità:

5.2.1 Schermata di Login

La schermata di login permette all'utente di autenticarsi inserendo username e password. Il controller associato valida i dati inseriti e invia una richiesta HTTP POST al server con le credenziali. In caso di successo, l'utente viene reindirizzato alla schermata principale; in caso di errore, viene visualizzato un messaggio appropriato.

5.2.2 Schermata di Registrazione

La schermata di registrazione consente a nuovi utenti di creare un account. Il controller verifica che i campi siano compilati correttamente, che la password soddisfi i requisiti minimi di sicurezza. Successivamente, invia una richiesta al server per creare il nuovo utente.

5.2.3 Schermata Lobby

La schermata lobby mostra l'elenco delle partite disponibili e permette all'utente di:

- Visualizzare le lobby esistenti con informazioni sul numero di giocatori e lo stato;
- Creare una nuova lobby specificando il senso di rotazione;
- Unirsi a una lobby esistente come giocatore;
- Unirsi a una partita in corso come spettatore.

La lista delle lobby viene aggiornata tramite un bottone dedicato, che invia richieste HTTP al server, garantendo che l'utente visualizzi sempre informazioni aggiornate.

5.2.4 Schermata di Gioco

La schermata di gioco rappresenta l'interfaccia principale durante una partita. Gli elementi principali includono:

- Area di visualizzazione della frase corrente;
- Barra per scrivere, a seconda del turno;
- Lista dei giocatori partecipanti;

5.3 Gestione WebSocket

La classe **LobbyWebSocketClient** estende la classe `WebSocketClient` fornita dalla libreria Java-WebSocket e implementa la logica per gestire la comunicazione in tempo reale con il server.

I principali metodi implementati sono:

- `onOpen()`: invocato quando la connessione WebSocket viene stabilita con successo;
- `onMessage()`: riceve i messaggi dal server e li processa in base al tipo;
- `onClose()`: gestisce la chiusura della connessione;
- `onError()`: gestisce eventuali errori durante la comunicazione;
- `send()`: invia messaggi JSON al server.

Durante una partita, il WebSocket viene utilizzato per:

- Ricevere aggiornamenti in tempo reale sullo stato della partita;
- Inviare le azioni del giocatore (frasi scritte);
- Sincronizzare i turni tra i giocatori;
- Notificare l'ingresso o l'uscita di giocatori.

VI

Logica di Gioco

6.1

Regole del Telefono Senza Fili

Il gioco del telefono senza fili è un gioco collaborativo in cui i giocatori si alternano nello scrivere frasi, creando una catena di interpretazioni che spesso porta a risultati divertenti e inaspettati.

Le regole del gioco sono le seguenti:

1. Il primo giocatore scrive una frase iniziale;
2. Il secondo giocatore visualizza la frase e deve continuarla;
3. Il terzo giocatore visualizza il complessivo e deve continuare a sua volta;
4. Il processo continua fino a quando tutti i giocatori hanno partecipato;
5. Al termine, viene mostrata la sequenza completa per vedere come la frase si è trasformata.

6.2

Fasi della Partita

Una partita attraversa diverse fasi, gestite dal server:

6.2.1 Fase di Attesa

La partita rimane in fase di attesa finché non si raggiunge il numero minimo di giocatori richiesto. Durante questa fase, i giocatori possono entrare e uscire liberamente dalla lobby. Il creatore della lobby può decidere di avviare la partita manualmente quando il numero minimo di giocatori è raggiunto (minimo 4 giocatori).

6.2.2 Fase di Inizializzazione

Quando la partita viene avviata, il server:

- Blocca l'ingresso di nuovi giocatori attivi (gli spettatori possono ancora unirsi);
- Invia a tutti i client le informazioni iniziali della partita;

6.2.3 Fase di Gioco

Durante ogni turno:

- Un giocatore riceve la frase dal turno precedente o, se è il primo turno, scrive la frase iniziale;
- Una volta completato, il contenuto viene inviato al server;
- Il server passa la frase al giocatore successivo;
- Gli altri giocatori e gli spettatori vedono un'indicazione di chi sta giocando.

6.2.4 Fase Finale

Quando tutti i giocatori hanno completato i loro turni, la partita termina e viene mostrata la sequenza completa di frasi.

6.3 Gestione dei Turni

Il server mantiene lo stato di ogni partita in una struttura dati che include:

- L'elenco dei giocatori e il loro ordine;
- L'indice del giocatore corrente;
- La catena di contenuti (frasi) prodotti finora;

Quando un giocatore completa il suo turno, il server:

1. Lo aggiunge alla catena;
2. Incrementa l'indice del turno;
3. Notifica il giocatore successivo tramite WebSocket;

6.4 Gestione delle Disconnessioni

Il sistema è progettato per gestire le disconnessioni improvvise dei giocatori. Quando un client si disconnette:

- Il server rileva la disconnessione tramite l'evento `onClose` del WebSocket;
- Se è il turno del giocatore disconnesso, viene saltato automaticamente;
- Gli altri giocatori vengono notificati della disconnessione;
- Se il numero di giocatori attivi scende sotto il minimo richiesto, la partita continua fino a quando i giocatori rimasti hanno partecipato.

VII

Sistema di Traduzione

7.1

Integrazione con LibreTranslate

Una caratteristica distintiva dell'applicazione è il supporto multilingua attraverso l'integrazione con l'API di LibreTranslate, un servizio di traduzione automatica open-source. Questo permette ai giocatori di utilizzare l'applicazione nella loro lingua preferita e facilita la comunicazione in partite con giocatori di diverse nazionalità.

7.2

LanguageHelper

La classe **LanguageHelper** implementa la logica di interazione con l'API di LibreTranslate. Le sue principali funzionalità includono:

- **Traduzione dall'inglese a una lingua target**: Utilizza l'endpoint di traduzione dell'API per convertire il testo dall'inglese alla lingua selezionata dall'utente.
- **Traduzione da una lingua sorgente all'inglese**: Utilizza l'endpoint di traduzione dell'API per convertire il testo da una lingua sorgente a inglese.

7.3

Flusso di Traduzione

Il processo di traduzione avviene come segue:

7.3.1 Traduzione dell'Interfaccia

In fase di registrazione, l'utente seleziona la lingua preferita. A questo punto:

1. Controlla se la lingua selezionata è inglese; in tal caso, non è necessaria alcuna traduzione;
2. Se la lingua è diversa dall'inglese, l'applicazione invia una richiesta all'API di LibreTranslate per tradurre tutti i testi dell'interfaccia;
3. Riceve i testi tradotti dall'API;
4. Aggiorna l'interfaccia con i testi tradotti.

7.3.2 Traduzione durante il Gioco

Durante una partita, quando un giocatore inserisce una frase:

1. L'applicazione verifica la lingua dell'utente;
2. Se la lingua è diversa dall'inglese, invia la frase all'API di LibreTranslate per tradurla in inglese;
3. Riceve la frase tradotta dall'API;
4. Invia la frase tradotta al server di gioco per la validazione;
5. Riceve la risposta dal server
6. Se necessario, traduce la risposta del server nella lingua dell'utente seguendo lo stesso processo inverso.
7. Mostra la risposta tradotta all'utente.

7.4 Gestione degli Errori di Traduzione

Il sistema implementa diverse strategie per gestire eventuali errori durante il processo di traduzione:

- Se l'API non è disponibile, viene mostrato il testo in inglese;
- Gli errori vengono loggati per permettere il debugging;
- L'utente viene notificato in caso di problemi persistenti.

VIII

Containerizzazione con Docker

8.1

Docker Compose

L'applicazione utilizza Docker Compose per orchestrare i vari servizi necessari al funzionamento del sistema. Il file **docker-compose.yml** definisce tre servizi principali:

8.1.1 Servizio Database

Il servizio database utilizza l'immagine ufficiale di PostgreSQL e configura:

- Le credenziali di accesso (username, password, nome del database);
- Il volume persistente per salvare i dati anche dopo il riavvio del container;
- La porta esposta per permettere la connessione dal servizio backend;

8.1.2 Servizio Backend

Il servizio backend compila ed esegue il server C:

- Utilizza un'immagine base con GCC e le librerie necessarie;
- Copia i file sorgente nel container;
- Compila l'applicazione all'avvio;
- Espone la porta per le connessioni WebSocket;
- Dipende dal servizio database e attende che sia pronto prima di avviarsi.

8.2

Dockerfile del Backend

Il Dockerfile del backend è strutturato in più stage per ottimizzare le dimensioni dell'immagine finale:

1. **Build stage**: installa le dipendenze di compilazione e compila il codice sorgente;
2. **Runtime stage**: copia solo i binari necessari dal build stage e le librerie runtime richieste.

Questo approccio multi-stage riduce significativamente le dimensioni dell'immagine finale, includendo solo ciò che è strettamente necessario per l'esecuzione dell'applicazione.

8.3

Networking

I container comunicano tra loro attraverso una rete Docker personalizzata. Questo garantisce:

- Isolamento dalla rete host;
- Risoluzione DNS automatica tra i servizi;
- Sicurezza migliorata limitando l'esposizione delle porte;
- Facilità di configurazione senza hard-coding degli IP.

8.4

Volumi Persistenti

L'applicazione utilizza volumi Docker per garantire la persistenza dei dati:

- **postgres_data**: memorizza i dati del database PostgreSQL.

Il volume è gestito da Docker e sopravvive alla rimozione dei container, garantendo che i dati degli utenti e le configurazioni non vengano persi.

IX

Sicurezza

9.1

Autenticazione e Autorizzazione

La sicurezza dell'applicazione è garantita attraverso diverse misure implementate sia lato server che lato client.

9.1.1 Hashing delle Password

Le password degli utenti non vengono mai salvate in chiaro nel database. Quando un utente si registra, la password viene processata attraverso un algoritmo di hashing crittografico sicuro prima di essere memorizzata. Durante il login, la password inserita viene hashata e confrontata con quella salvata nel database.

9.2

Prevenzione di Attacchi Comuni

9.2.1 SQL Injection

Tutte le query al database utilizzano prepared statements, che separano la logica SQL dai dati forniti dall'utente. Questo previene attacchi di SQL injection, poiché i dati dell'utente vengono sempre trattati come valori e mai come codice eseguibile.

X

Conclusioni e Sviluppi Futuri

10.1 Obiettivi Raggiunti

Il progetto ha raggiunto con successo tutti gli obiettivi prefissati:

- Implementazione completa di un'applicazione client-server funzionante;
- Interfaccia utente intuitiva e responsive realizzata con JavaFX;
- Backend robusto e performante implementato in C con Mongoose;
- Sistema di autenticazione sicuro con password hashate;
- Comunicazione in tempo reale tramite WebSocket;
- Supporto multilingua attraverso integrazione con LibreTranslate;
- Containerizzazione completa con Docker per facilità di deployment;
- Gestione persistente dei dati con PostgreSQL.

10.2 Difficoltà Incontrate

Durante lo sviluppo del progetto sono state affrontate diverse sfide:

- **Sincronizzazione**: garantire che tutti i client visualizzino lo stesso stato della partita in tempo reale ha richiesto un'attenta gestione dei messaggi WebSocket;
- **Gestione delle disconnessioni**: implementare un sistema robusto per gestire disconnessioni improvvise senza compromettere l'esperienza degli altri giocatori;

10.3

Considerazioni Finali

Questo progetto ha rappresentato un'opportunità significativa per applicare concetti di programmazione di sistema, networking e sviluppo di applicazioni distribuite. L'integrazione di tecnologie diverse (C, Java, PostgreSQL, Docker) ha permesso di acquisire esperienza pratica nella gestione di un sistema complesso multi-componente.

L'utilizzo di Docker ha semplificato notevolmente il processo di deployment, garantendo coerenza tra ambienti di sviluppo e produzione. Questa esperienza ha evidenziato l'importanza della containerizzazione nelle moderne architetture applicative.

Il supporto multilingua attraverso LibreTranslate ha dimostrato come l'integrazione con servizi esterni possa arricchire significativamente le funzionalità di un'applicazione, rendendola accessibile a un pubblico più ampio.

In conclusione, il progetto ha raggiunto tutti gli obiettivi prefissati, risultando in un'applicazione funzionale che offre un'esperienza di gioco divertente e coinvolgente. Le basi create permettono facilmente l'aggiunta di nuove funzionalità e miglioramenti futuri.

10.4

Ringraziamenti

Si ringraziano tutti i membri del team per il contributo allo sviluppo del progetto, per la collaborazione durante le fasi di progettazione e implementazione, e per il supporto reciproco nel superare le difficoltà incontrate. Un ringraziamento particolare va ai docenti del corso di Laboratorio di Sistemi Operativi per le competenze trasmesse e per il supporto fornito durante lo sviluppo del progetto.

Si ringrazia inoltre De Gregorio Gennaro per aver fornito il suo raspberry pi per l'hosting del server.