



UNIVERSITÀ DEGLI STUDI DI NAPOLI  
**FEDERICO II**

*Documentazione Laboratorio Di Sistemi Operativi*  
**TelefonoSenzaFilo**

Gruppo ThreadDripper

Florindo Zeconi (Matricola N86004544)

Francesco Trotti (Matricola N86004881)

Domenico Gagliotti (Matricola N86004536)

Anno 2024/2025

# Indice

<b>1</b>	<b>Introduzione</b>	<b>3</b>
<b>2</b>	<b>Documento Dei Requisiti</b>	<b>4</b>
2.1	Casi d'uso - Use Case Diagram . . . . .	4
2.2	Requisiti di Dominio . . . . .	5
2.3	Requisiti Non Funzionali . . . . .	5
<b>3</b>	<b>System Design</b>	<b>6</b>
3.1	Architettura . . . . .	6
3.1.1	Client-Server . . . . .	6
3.1.2	Monolitico . . . . .	6
3.2	Tecnologie . . . . .	6
3.3	Persistenza dati . . . . .	8
3.3.1	Class Diagram (Già Ristrutturato) . . . . .	8
3.3.2	Modello Logico . . . . .	8
3.3.3	Schema Fisico . . . . .	9
<b>4</b>	<b>Software Design</b>	<b>10</b>
4.1	Server Multithreading per gestire le richieste . . . . .	10
4.2	Creazione di un processo figlio per ogni partita . . . . .	10
4.3	Comunicazione tra server principale e processo partita con <code>unamed pipe</code> . . . . .	11
4.4	Uso di porte separate per le partite . . . . .	11
4.5	Server di partita multithreading . . . . .	11
4.6	Uso di <code>pthread_cond</code> per sincronizzare l'attesa dei client . . . . .	12
4.7	Pollin per evitare chiamate bloccanti . . . . .	12

# Elenco delle figure

2.1	Use Case Diagram - TelefonoSenzaFili . . . . .	4
3.1	Schema Database (Ristrutturato) . . . . .	8

# Capitolo 1

## Introduzione

Lo studente dovrà realizzare la simulazione di un sistema che modella il gioco **Telefono senza fili** tra *giocatori di diverse lingue*. Gli *utenti devono potersi registrare, specificando la propria lingua* di appartenenza, e loggare al server. Gli *utenti possono accedere a delle stanze di gioco* nella quale vogliono giocare, e *possono mandare messaggi* con altri utenti. Le *stanze devono avere un minimo numero di giocatori da attendere prima di iniziare la partita (minimo 4)*. Il *creatore decide se l'ordine di gioco deve essere orario o anti-orario*. Il *creatore della stanza è il primo che invia il messaggio*. Tale messaggio sarà tradotto nella lingua del secondo giocatore, il secondo giocatore aggiungerà un'altra parola a quella messaggiata, e la passerà al terzo giocatore. La frase verrà tradotta nella lingua del terzo giocatore, e così via finché non arriverà all'ultimo giocatore. Alla fine del turno, mostra a tutti i giocatori il percorso della frase.

*Utenti che si vogliono inserire in una partita già iniziata, vengono messi in coda fin quando un altro utente non lascia la stanza. Gli utenti vengono inseriti nella stanza con una logica FIFO.*

- Il dizionario si può realizzare come meglio si crede (es., file di testo, json, database sql o a grafo, etc..)
- Registrazione può essere sviluppata tramite database sql o file
- Opzionale: le stanze di gioco possono essere create dagli utenti

# Capitolo 2

## Documento Dei Requisiti

Il Documento dei Requisiti è un documento formale che descrive in modo chiaro e dettagliato le funzionalità, le caratteristiche e i vincoli di un sistema o software da sviluppare. Serve come riferimento per clienti, sviluppatori e stakeholder, garantendo un'interpretazione condivisa delle esigenze e degli obiettivi del progetto.

### 2.1 Casi d'uso - Use Case Diagram

Il Use Case Diagram di Telefono Senza Fili rappresenta le interazioni tra gli utenti e il sistema, evidenziando le funzionalità principali, come effettuare chiamate, ricevere chiamate e gestire la rubrica. Mostra gli attori coinvolti e le relazioni tra i diversi casi d'uso.

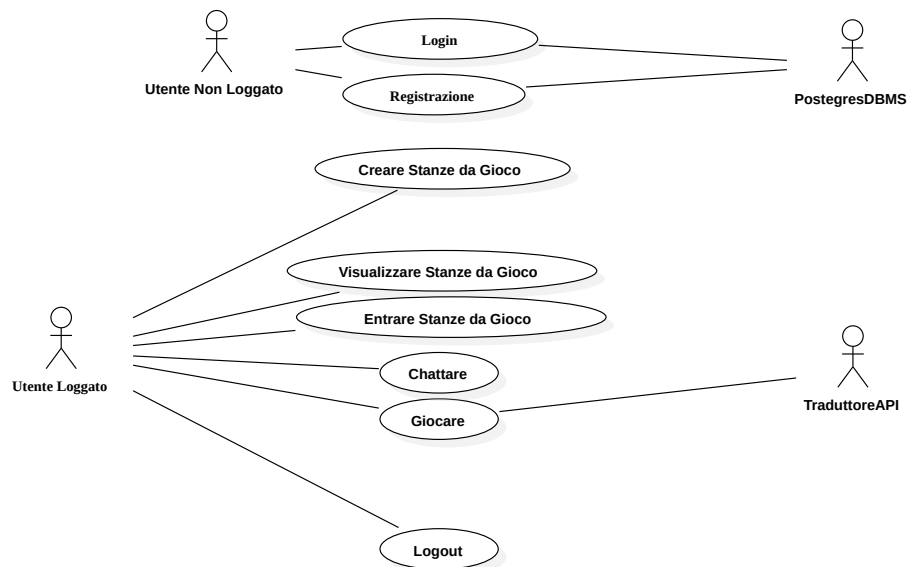


Figura 2.1: Use Case Diagram - TelefonoSenzaFili

## 2.2 Requisiti di Dominio

I requisiti di dominio descrivono le regole e le logiche specifiche del gioco Telefono Senza Fili, come la gestione delle lingue, il flusso dei messaggi e le dinamiche delle stanze.

- Il gioco si basa su una sequenza di traduzioni e aggiunte di parole, creando una frase evolutiva.
- Gli utenti devono specificare la propria lingua al momento della registrazione.
- Le stanze di gioco richiedono un minimo di 4 giocatori per iniziare una partita.
- Il creatore della stanza sceglie l'ordine di gioco (orario o anti-orario).
- Gli utenti possono unirsi a stanze in attesa o mettersi in coda per stanze già avviate.
- Il messaggio iniziale è inviato dal creatore della stanza e viene tradotto per ogni giocatore successivo.
- Alla fine del turno, il percorso della frase viene mostrato a tutti i giocatori.
- La gestione delle lingue e delle traduzioni è parte integrante del sistema.
- L'inserimento nelle stanze segue una logica FIFO (primo arrivato, primo servito).

## 2.3 Requisiti Non Funzionali

I requisiti non funzionali definiscono le qualità del sistema, come scalabilità, sicurezza, usabilità e prestazioni, per garantire un'esperienza di gioco efficiente e affidabile.

- **Scalabilità** : Il sistema deve supportare più stanze e più utenti simultaneamente.
- **Gestione della Concorrenza** : L'accesso alle stanze e la gestione dei turni devono avvenire senza conflitti tra utenti.
- **Opzionale** : Gli utenti possono creare stanze personalizzate.

# Capitolo 3

## System Design

In questa sezione viene descritta l'architettura del sistema, evidenziando le principali componenti, le loro interazioni e le scelte progettuali effettuate.

### 3.1 Architettura

Passiamo dunque alla descrizione dell'architettura passo dopo passo, elencando le scelte effettuate, i vantaggi e gli svantaggi, evidenziando i punti di forza e le caratteristiche su cui dobbiamo ancora lavorare.

#### 3.1.1 Client-Server

L'architettura Client-Server è un modello di comunicazione in cui un client invia richieste a un server, che elabora i dati e restituisce le risposte. Questo approccio è usato per distribuire risorse e servizi in rete, garantendo scalabilità e gestione centralizzata.

#### 3.1.2 Monolitico

L'architettura monolitica è un modello in cui un'applicazione è sviluppata come un unico blocco indivisibile, in cui tutti i componenti (interfaccia utente, logica di business e accesso ai dati) sono strettamente integrati. È semplice da sviluppare ma può diventare difficile da scalare e mantenere.

### 3.2 Tecnologie

- **Logica di presentazione :**
  - **C :** Gestisce l'interfaccia utente di un'applicazione, mostrando dati e raccogliendo input. Può essere implementata tramite console (`printf`, `scanf`).
- **Persistenza dati :**
  - **PostgreSQL:** Abbiamo scelto PostgreSQL come DBMS per il nostro progetto, in quanto si basa su un sistema relazionale che si adatta perfettamente al nostro dominio, il quale si presta al modello a tabelle. La scelta è motivata anche dalla disponibilità di licenze gratuite, a differenza del competitor Oracle, e dall'esperienza già consolidata del team con questa tecnologia.

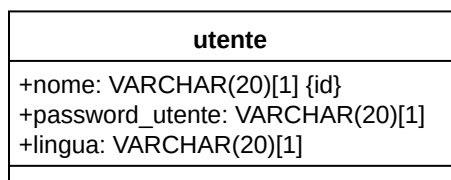
- **Logica di business :**
  - **C:** Come specificato nella traccia, garantendo efficienza e controllo diretto sulle risorse.
- **Comunicazione :**
  - **Socket POSIX:** La comunicazione con socket POSIX permette lo scambio di dati tra processi su reti locali o internet. Utilizziamo le API dei socket in C, come `socket()`, `bind()`, `listen()`, `accept()` per il server e `connect()` per il client, basandosi sul modello client-server.
  - **TCP :** Protocollo di trasporto affidabile che garantisce la consegna corretta dei dati tra client e server. Utilizza una connessione orientata al flusso di byte, assicurando che i dati siano ricevuti nell'ordine giusto e senza perdite.
- **Versionamento del codice :**
  - **Git :** Abbiamo scelto Git come sistema di versionamento per la sua affidabilità e diffusione nell'industria. Grazie a Git, possiamo gestire il lavoro in team in modo collaborativo, creare branch per sperimentazioni o nuove funzionalità, e tracciare facilmente ogni modifica al codice. Inoltre, la sua compatibilità con piattaforme come GitHub e GitLab semplifica l'integrazione con il nostro flusso di lavoro.
- **Containerizzazione :**
  - **Docker:** Come specificato nella traccia , consente di creare ambienti virtualizzati leggeri e indipendenti. Inoltre, Docker offre numerosi vantaggi, tra cui: indipendenza dall'hardware, facilità di memorizzazione, isolamento dei servizi e gestione semplificata dei container.



## 3.3 Persistenza dati

La sezione "Persistenza dei dati su un database" descrive il modello di persistenza adottato nel progetto, basato sui diagrammi delle classi e sul loro successivo miglioramento. Si approfondisce come le entità e le loro relazioni siano rappresentate nel database, analizzando l'evoluzione della struttura del modello per ottimizzare la gestione e il recupero dei dati. Vengono inoltre esplorate le scelte architetturali che assicurano integrità, coerenza e performance nel sistema.

### 3.3.1 Class Diagram (Già Ristrutturato)



The diagram shows a class named 'utente'. It has three attributes: '+nome: VARCHAR(20)[1] {id}', '+password\_utente: VARCHAR(20)[1]', and '+lingua: VARCHAR(20)[1]'. The attribute '+nome' is underlined, indicating it is a primary key. The attribute '+password\_utente' is double-underlined, indicating it is a foreign key. The attribute '+lingua' is single-underlined, indicating it is a foreign key.

utente
+nome: VARCHAR(20)[1] {id}
+password_utente: VARCHAR(20)[1]
+lingua: VARCHAR(20)[1]

Figura 3.1: Schema Database (Ristrutturato)

### 3.3.2 Modello Logico

Infine possiamo raggiungere la parte finale di questa progettazione, ovvero la parte dello schema logico. Indicazioni :

- Gli attributi sottolineati sono chiavi primarie.
- Gli attributi doppiamente sottolineati sono chiavi esterne.
- Gli attributi con \* possono essere NULL

---

**Utente** (nome, password\_utente , lingua)

---

Tabella 3.1: Schema logico - fine

### 3.3.3 Schema Fisico

```
1 CREATE TABLE utente (  
2     nome VARCHAR(20) PRIMARY KEY,  
3     password_utente VARCHAR(20) NOT NULL,  
4     lingua VARCHAR(20) NOT NULL  
5 );  
6  
7 ALTER TABLE utente  
8 ADD CONSTRAINT check_lingua  
9 CHECK (lingua IN ('it', 'en', 'es', 'fr', 'de'));
```

# Capitolo 4

## Software Design

Il Software Design Document (SDD) descrive l'architettura, i componenti e le decisioni di progettazione di un software. Serve come guida per sviluppatori e stakeholder, garantendo coerenza e chiarezza nella realizzazione del sistema.

### 4.1 Server Multithreading per gestire le richieste

- Il server principale accetta connessioni sulla welcome socket.
- Per ogni richiesta, il server crea un thread separato che gestisce la comunicazione.
- Motivazione:
  - Mantiene la welcome socket sempre disponibile per nuove connessioni.
  - Migliora la scalabilità, permettendo di gestire più client contemporaneamente.
- Vantaggio: Il server può accettare nuove connessioni senza blocchi, garantendo un'esperienza fluida.

### 4.2 Creazione di un processo figlio per ogni partita

- Quando un client crea una nuova partita, il server genera un processo figlio con `fork()`.
- Il figlio esegue un `exec` per caricare il programma che gestisce la partita.
- Motivazione:
  - Isolamento delle partite: Se un processo crasha, le altre partite e il server restano funzionanti.
  - Scalabilità: Ogni partita gira su un processo separato, sfruttando al meglio i multi-core.
  - Facilità di gestione: Il server principale si occupa solo della gestione delle richieste, mentre il processo partita gestisce la logica di gioco.
- Vantaggio: Stabilità e separazione delle partite, evitando che un problema in una partita blocchi l'intero sistema.

### 4.3 Comunicazione tra server principale e processo partita con unnamed pipe

- Il server principale e il processo partita comunicano tramite una pipe.
- Lo scambio dati include:
  - Dati della stanza (ID partita, numero giocatori, stato della partita).
  - Porta della socket della partita.
- Motivazione:
  - Pipe è una soluzione semplice ed efficiente per la comunicazione interprocesso (IPC).
  - Evita l'uso di file o database per condividere informazioni tra processi.
- Vantaggio: Comunicazione veloce e sicura tra server principale e processi partita senza overhead di rete.

### 4.4 Uso di porte separate per le partite

- Ogni partita ha una porta dedicata.
- Il client riceve dal server la porta corretta e si connette direttamente al processo partita.
- Motivazione:
  - Isolamento tra partite: Ogni partita ha il suo canale di comunicazione separato.
  - Efficienza: Evita di filtrare i messaggi in un unico server centralizzato.
  - Semplicità: Ogni client si connette direttamente al proprio server di gioco senza confusione.
- Vantaggio: Connessione diretta e pulita ai server di partita, evitando congestione sulla porta principale.

### 4.5 Server di partita multithreading

- Anche il processo partita usa thread separati per ogni richiesta.
- Motivazione:
  - Mantiene la welcome socket della partita libera, permettendo nuovi ingressi.
  - Gestisce più client simultaneamente, senza ritardi o blocchi.
  - Migliora la reattività, distribuendo il carico tra più thread.
- Vantaggio: I client possono interagire in tempo reale senza attese, migliorando l'esperienza di gioco.

## 4.6 Uso di pthread\_cond per sincronizzare l'attesa dei client

- I client si mettono in attesa della partita usando pthread\_cond\_wait() su un mutex.
- Il thread che gestisce la partita sveglia i client con pthread\_cond\_broadcast().
- Motivazione:
  - Evitare polling attivo: Senza pthread\_cond, i client dovrebbero controllare continuamente lo stato della partita, consumando CPU.
  - Sincronizzazione sicura: Garantisce che tutti i client siano pronti prima di avviare la partita.
- Vantaggio: Riduzione del consumo di risorse e gestione ordinata dell'ingresso in partita.

## 4.7 Pollin per evitare chiamate bloccanti

- Utilizziamo il pollin sulle receive per evitare chiamate bloccanti
- Utilizziamo il pollin per monitorare lo stato delle socket (IsSocketConnected)