Documentazione Completa Progetto: "Sistema di Gestione Campionato Serie A"

1. Panoramica del Progetto

1.1 Scopo e Ambito

Sistema: Applicazione client-server per la gestione di dati di un campionato di calcio (Serie A)
Funzionalità principale: Operazioni CRUD per squadre e giocatori con interfacce GUI/CLI
Architettura: Server TCP multithread con database SQLite

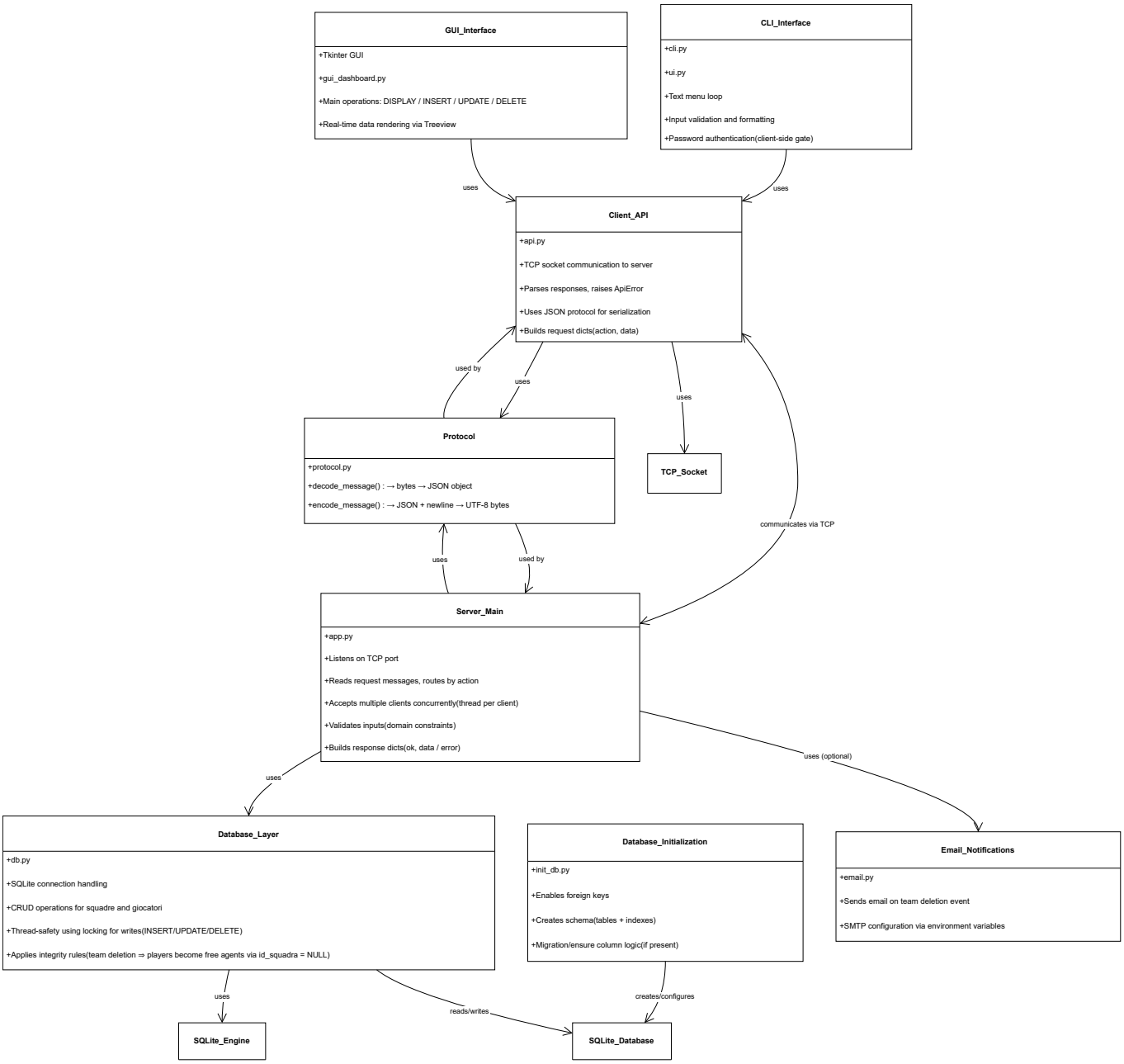
1.2 Stack Tecnologico

- **Linguaggio**: Python 3.8+
- **Database**: SQLite3 con vincoli di chiave esterna
- **Rete**: Socket TCP con protocollo JSON
- **GUI**: Tkinter (opzionale)
- **Comunicazione**: JSON su TCP con delimitazione newline

---

2. Documentazione dell'Architettura

2.1 Diagramma dell'Architettura del Sistema



2.2 Responsabilità dei Componenti

Lato Client

1. **Interfaccia CLI** (`cli.py` , `ui.py`):
   - Autenticazione **client-side** tramite password (controllo locale)
   - Sistema di menu testuale per la selezione delle operazioni
   - Raccolta, validazione e formattazione degli input utente
   - Visualizzazione strutturata dei risultati su terminale
2. **Interfaccia GUI** (`gui_dashboard.py`):
   - Interfaccia grafica basata su Tkinter con schermata di login
   - Dashboard principale con operazioni CRUD: DISPLAY, INSERT, UPDATE, DELETE
   - Visualizzazione e aggiornamento dei dati in tempo reale tramite widget Treeview

- Gestione selezione e sincronizzazione tra liste (squadre, giocatori, svincolati)

3. **Client API** (`api.py`):
   - Comunicazione tramite socket TCP con il server
   - Apertura di una connessione per ciascuna richiesta
   - Serializzazione e deserializzazione dei messaggi in formato JSON
   - Gestione di timeout, errori di connessione ed errori applicativi restituiti dal server
4. **Protocollo Client** (`protocol.py`):
   - Codifica e decodifica dei messaggi JSON
   - Serializzazione UTF-8 con delimitazione tramite newline (line-delimited JSON)

Lato Server

1. **Server Principale** (`app.py`):
   - Server socket multithread (un thread dedicato per ogni client)
   - Instradamento delle richieste in base all'azione richiesta
   - Validazione lato server delle regole di business (ruoli, vincoli temporali, coerenza dati)
   - Gestione centralizzata degli errori e invio di risposte strutturate al client
   - Integrazione con il modulo email per notifiche su eventi rilevanti
2. **Layer Database** (`db.py`):
   - Gestione delle connessioni SQLite
   - Operazioni thread-safe tramite meccanismi di locking
   - Applicazione dei vincoli di integrità referenziale
   - Implementazione delle regole applicative (es. svincolo automatico dei giocatori alla cancellazione di una squadra)
   - Sollevamento di eccezioni applicative per errori di dominio
3. **Protocollo Server** (`protocol.py`):
   - Decodifica delle richieste ricevute dal client
   - Codifica delle risposte in formato JSON UTF-8 con delimitazione newline

---

Moduli di Supporto

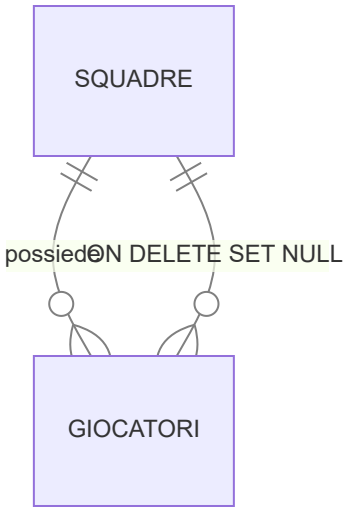
1. **Inizializzazione e Migrazione Database** (`init_db.py`):
   - Creazione dello schema iniziale del database
   - Definizione di tabelle, chiavi primarie e vincoli di chiave esterna
   - Gestione di migrazioni semplici per mantenere la compatibilità con database esistenti

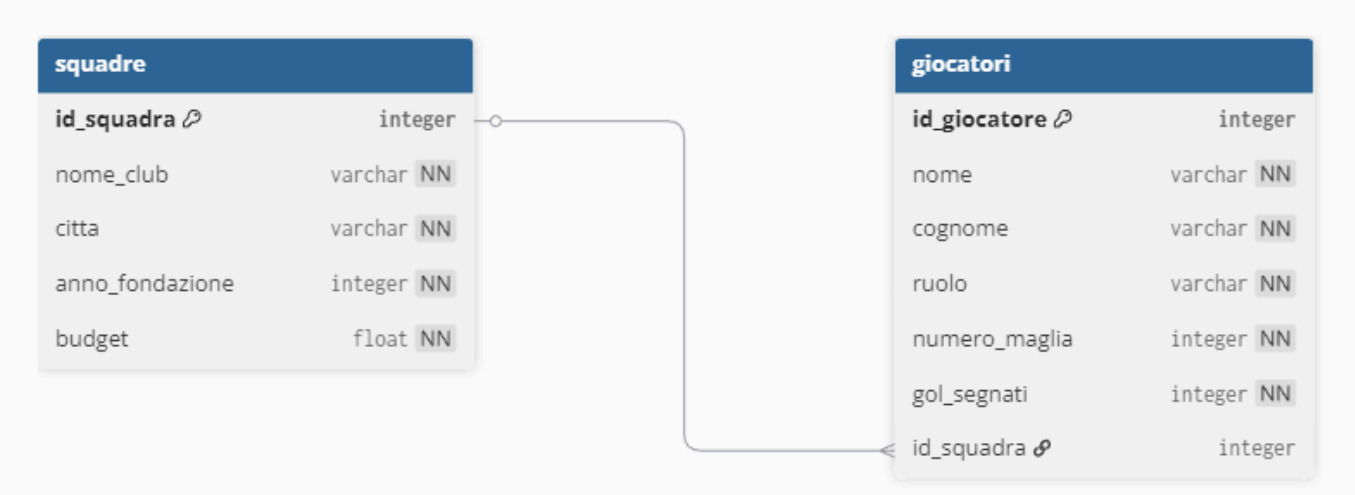
2. **Reset Database** (`reset_db.py`):
    - Utility per il ripristino dello stato iniziale del database
    - Supporto alle attività di test, debug e dimostrazione del sistema
3. **Notifiche Email** (`email.py`):
    - Invio di notifiche email in seguito a eventi specifici (es. cancellazione di una squadra)
    - Configurazione tramite variabili d'ambiente
    - Gestione non bloccante degli errori di invio (best-effort)

---

3. Documentazione Schema Database

3.1 Diagramma Entità-Relazione





3.2 Definizioni delle Tabelle

`squadre`

Colonna	Tipo	Vincoli	Descrizione
`id_squadra`	INTEGER	PRIMARY KEY AUTOINCREMENT	Identificatore unico della squadra
`nome_club`	STRING	NOT NULL UNIQUE	Nome assegnato alla squadra (univoco)
`citta`	STRING	NOT NULL	Città di origine
`anno_fondazione`	INTEGER	NOT NULL	Anno di fondazione (1850-anno corrente)
`budget`	REAL	NOT NULL DEFAULT 0	Budget della squadra in EUR

`giocatori`

Colonna	Tipo	Vincoli	Descrizione
`id_giocatore`	INTEGER	PRIMARY KEY AUTOINCREMENT	Identificatore unico del giocatore
`nome`	STRING	NOT NULL	Nome del giocatore
`cognome`	STRING	NOT NULL	Cognome del giocatore
`ruolo`	STRING	NOT NULL	Ruolo del giocatore: *Portiere/Difensore/Centrocampista/Attacc*
`numero_maglia`	INTEGER	NOT NULL	Numero maglia (1-99)
`gol_segnati`	INTEGER	NOT NULL DEFAULT 0	Gol segnati
`id_squadra`	INTEGER	FOREIGN KEY NULLABLE	Riferimento alla chiave della tabella squa (NULL = svincolato)

Foreign Key: `id_squadra` → `squadre(id_squadra)` ON DELETE SET NULL

---

4. Guida all'Installazione e Configurazione

4.1 Prerequisiti

```
# Requisiti di Sistema
- Python 3.8 o superiore
- SQLite3 (incluso con Python)
```

```
- Connessione internet (per funzionalità email)

# Pacchetti Python Richiesti
python-dotenv>=1.0.0  # Per configurazione ambiente
```

4.2 Installazione Passo-Passo

4.2 Download del Progetto

Il codice sorgente completo è disponibile sul mio repository GitHub personale:

🔗 **Repository GitHub**: https://github.com/spare1214/Campionato_Serie-A.git

Per scaricare e iniziare rapidamente *(via terminal):

```
git clone https://github.com/spare1214/Campionato_Serie-A.git
cd campionato
```

2. Struttura del Progetto

```
campionato_serie_a/
├── client/                    # Moduli lato client
│   ├── __init__.py
│   ├── api.py
│   ├── protocol.py
│   ├── cli.py                 # interfaccia a riga di comando
│   ├── gui_dashboard.py       # interfaccia grafica
│   └── ui.py
├── server/                    # Moduli lato server
│   ├── __init__.py
│   ├── app.py                 # Avviamento del server (main file)
│   ├── db.py
│   ├── email.py
│   └── protocol.py
├── db/                        # Directory database
│   ├── reset_db.py
│   ├── init_db.py             # Inizializzazione db
│   ├── campionato.db          # (creato al primo avvio)
├── assets/                    # Asset GUI
│   └── background_2.png       # Immagine di sfondo
└── .env                       # File configurazione per email (creare
manualmente)
```

3. Inizializzazione Database

```
# Inizializzare database con schema
python init_db.py

# Output atteso:
# DB initialized at db/campionato.db
```

4. Configurazione Ambiente

Creare file `.env` nella root del progetto:

```
# Configurazione SMTP (per notifiche email)
SMTP_SENDER="tua-email@gmail.com"
SMTP_PASSWORD="tua-password-app"     # Usa password app-specifica per Gmail
SMTP_RECIPIENT="admin@example.com"
SMTP_HOST="smtp.gmail.com"
SMTP_PORT="587"
```

5. Avvio del Sistema

Opzione A: Usando CLI (Interfaccia a Linea di Comando)

```
# Terminale 1: Avviare server
python -m server.app

# Terminale 2: Avviare client CLI
python -m client.cli
# Password: mypass (predefinita)
```

Opzione B: Usando GUI (Interfaccia Grafica)

```
# Terminale 1: Avviare server
python -m server.app

# Terminale 2: Avviare client GUI
python -m client.gui_dashboard
```

---

5. Documentazione del Protocollo di Comunicazione

5.1 Formato dei Messaggi

Tutti i messaggi sono oggetti JSON con terminazione newline (`\n`).

Formato Richiesta

```
{
  "action": "nome_azione",
  "data": {
    "campo1": "valore1",
    "campo2": 123
  }
}
```

Formato Risposta

```
// Successo
{
  "ok": true,
  "data": {
    // Dati di risposta
  }
}

// Errore
{
  "ok": false,
  "error": {
    "code": "CODICE_ERRORE",
    "message": "Messaggio errore leggibile"
  }
}
```

5.2 Azioni Disponibili

Azione	Descrizione	Campi Dati Richiesti
`list_teams`	Ottieni tutte le squadre	Nessuno
`create_team`	Crea nuova squadra	`nome_club` , `citta` , `anno_fondazione` , `budget`
`delete_team`	Elimina una squadra	`id_squadra`
`create_player`	Crea nuovo giocatore	`nome` , `cognome` , `ruolo` , `numero_maglia`

Azione	Descrizione	Campi Dati Richiesti
`list_players_by_team`	Ottieni giocatori per squadra	`id_squadra`
`update_player`	Aggiorna dati giocatore	`id_giocatore`, `nome`, `cognome`, `ruolo`, `numero_maglia`
`transfer_player`	Trasferisci giocatore tra squadre	`id_giocatore`, `id_squadra` (opzionale)
`delete_player`	Elimina un giocatore	`id_giocatore`
`list_free_agents`	Ottieni giocatori svincolati	Nessuno

5.3 Codici di Errore

Codice	Significato	Cause Tipiche
`BAD_JSON`	Formato JSON non valido	Richiesta malformata
`BAD_REQUEST`	Parametri richiesta non validi	Campi mancanti, tipi errati
`NOT_FOUND`	Risorsa non trovata	ID non validi
`INTEGRITY_ERROR`	Violazione vincolo database	Dati duplicati, violazione FK
`SERVER_ERROR`	Errore interno server	Problemi database, bug
`UNKNOWN_ACTION`	Nome azione non valido	Errore di battitura nel campo action

---

6. Guida per l'Utente

6.1 Interfaccia CLI

```
=== CAMPIONATO (CLIENT) ===
1) Inserisci nuova squadra
2) Mostra tutte le squadre
3) Cancella squadra (giocatori -> svincolati)
4) Tesserare nuovo giocatore
5) Mostra giocatori di una squadra
6) Modifica dati giocatore
7) Trasferisci giocatore
8) Cancella giocatore
9) Mostra giocatori svincolati
0 Esci
```

Esempio Sessione CLI

```
# 1. Crea una squadra
Nome club: Juventus FC
Città: Torino
Anno fondazione: 1897
Budget: 5000000.00
Squadra creata. ID = 1

# 4. Crea un giocatore
Nome: Cristiano
Cognome: Ronaldo
Ruolo: 1) Portiere 2) Difensore 3) Centrocampista 4) Attaccante
Scelta (1-4): 4
Numero maglia (1-99): 7
Assegna a squadra? (premi INVIO per svincolato)
ID squadra: 1
Giocatore creato. ID = 1

# 5. Visualizza giocatori squadra
ID squadra: 1
ID | Nome | Cognome | Ruolo | Maglia
1 | Cristiano | Ronaldo | Attaccante | 7
```

6.2 Interfaccia GUI

Schermata di Login

- Inserire password predefinita: `mypass`
- Sfondo: Immagine a tema Serie A

Dashboard

Quattro operazioni principali:

1. **DISPLAY**: Visualizza squadre, giocatori squadra e svincolati
2. **INSERT**: Crea nuove squadre e giocatori
3. **UPDATE**: Modifica dati giocatore e trasferimenti
4. **DELETE**: Rimuovi squadre e giocatori

Caratteristiche Chiave GUI

- **Aggiornamenti in tempo reale**: Clicca squadre per vedere giocatori
- **Validazione input**: Menu a tendina ruoli, validazione anno

- **Dialoghi di conferma**: Per operazioni distruttive
- **Notifiche email**: Su cancellazione squadra (se configurato)

---

7. Dettagli Implementazione Tecnica

7.1 Concorrenza e Modello Threading

```
# Architettura threading server
def client_thread(conn: socket.socket, addr):
    """Un thread per connessione client"""
    with conn:
        # Gestisci tutte le richieste da questo client
        for line in conn.makefile("rb"):
            # Processa richiesta
            resp = handle_request(decode_message(line))
            conn.sendall(encode_message(resp))

# Locking database
_write_lock = threading.Lock()  # Lock globale per tutte le operazioni di
scrittura

def create_team(...):
    with _write_lock, _connect() as conn:
        # Operazione thread-safe di scrittura
```

Strategia di Locking --> Single lock globale garantisce:

- Nessuna race condition su scritture database
- Implementazione semplice per bisogno di bassa concorrenza

7.2 Pipeline di Validazione Dati

```
Input Client → Validazione Client → Rete → Validazione Server → Vincoli
Database
      ↓                ↓                ↓          ↓                    ↓
   Controllo    Regole business    Parsing JSON  Validazione         Vincoli
FK
   tipo base   (maglia 1-99)                    ruolo/anno        (ON DELETE
SET NULL)
```

7.3 Sistema di Notifica Email

```
def send_delete_team_email(team_name: str, team_id: int):
    """
    Attivato quando una squadra viene eliminata
    Invia notifica all'email amministrativa configurata
    """
    # 1. Recupera configurazione SMTP da .env
    # 2. Costruisci messaggio email
    # 3. Invia via SMTP con TLS
    # 4. Log successo/fallimento (solo modalità debug)
```

Requisiti Configurazione:

- Utenti Gmail hanno bisogno di password app-specifica
- Altri server SMTP necessitano host/porta appropriati
- Funzionalità email è opzionale (saltata se non configurata)

---

8. Riferimento API

8.1 Classe `CampionatoAPI` (api.py)

```
class CampionatoAPI:
    """Wrapper API lato client per comunicazione server"""

    def __init__(self, host="127.0.0.1", port=5000):
        # Inizializza parametri connessione

    def _send(self, req: Dict[str, Any]) -> Dict[str, Any]:
        # Invia richiesta e analizza risposta

    # Operazioni squadre
    def create_team(self, nome_club: str, citta: str,
                    anno_fondazione: int, budget: float) -> int

    def list_teams(self) -> List[Tuple]

    def delete_team(self, id_squadra: int) -> None

    # Operazioni giocatori
    def create_player(self, nome: str, cognome: str, ruolo: str,
                      numero_maglia: int, id_squadra: Optional[int]) -> int

    def list_players_by_team(self, id_squadra: int) -> List[Tuple]
```

```
    def update_player(self, id_giocatore: int, nome: str, cognome: str,
                      ruolo: str, numero_maglia: int) -> None

    def transfer_player(self, id_giocatore: int,
                        id_squadra: Optional[int]) -> None

    def delete_player(self, id_giocatore: int) -> None

    def list_free_agents(self) -> List[Tuple]
```

8.2 Funzioni Layer Database (db.py)

```
# Funzioni core (tutte thread-safe)
create_team(nome_club, citta, anno_fondazione, budget) -> team_id
delete_team(id_squadra)  # Rende giocatori svincolati
create_player(nome, cognome, ruolo, numero_maglia, id_squadra, gol_segnati=0)
-> player_id
update_player(id_giocatore, nome, cognome, ruolo, numero_maglia,
gol_segnati=None)
transfer_player(id_giocatore, new_id_squadra)
delete_player(id_giocatore)

# Funzioni query (sola lettura, no locking necessario)
list_teams() -> List[Tuple]
list_players_by_team(id_squadra) -> List[Tuple]
list_free_agents() -> List[Tuple]
team_exists(team_id) -> bool
player_exists(player_id) -> bool
get_team_by_id(team_id) -> Tuple
```

---

9. Guida ai Test

9.1 Scenari di Test Manuale

Scenario 1: Ciclo di Vita Squadra

```
1. Crea squadra "AC Milan" (ID: 1)
2. Crea giocatore "Zlatan Ibrahimović" assegnato a squadra 1
3. Visualizza giocatori squadra 1 (dovrebbe mostrare Zlatan)
4. Elimina squadra 1 (giocatori diventano svincolati)
5. Visualizza svincolati (dovrebbe mostrare Zlatan)
```

```
6. Crea squadra "Inter Milan" (ID: 2)
7. Trasferisci Zlatan a squadra 2
8. Visualizza giocatori squadra 2 (dovrebbe mostrare Zlatan)
```

Scenario 2: Gestione Errori

```
1. Prova a creare squadra con nome duplicato (dovrebbe fallire)
2. Prova a eliminare squadra inesistente (dovrebbe mostrare errore)
3. Prova a creare giocatore con ruolo non valido (dovrebbe fallire)
4. Prova a trasferire giocatore a squadra inesistente (dovrebbe fallire)
```

9.2 Test Notifiche Email

```
1. Configura .env con credenziali SMTP valide
2. Crea una squadra di test
3. Elimina la squadra
4. Controlla la casella email per notifica
```

---

10. Risoluzione Problemi e FAQ

10.1 Problemi Comuni

Problema 1: Errore "Connection refused"

```
Causa: Server non in esecuzione o porta errata
Soluzione: Assicurarsi che il server sia in esecuzione: python -m server.app
Verificare che la porta 5000 sia disponibile: netstat -an | grep 5000
```

Problema 2: Errori database all'avvio

```
Causa: File database mancante o permessi insufficienti
Soluzione:  1) Eseguire init_db.py per creare database
            2) Eliminare file campionato.db esistente, eseguire reset_db.py
poi                    ritorna al passo no.1
Controllare permessi scrittura directory db/
```

Problema 3: GUI mostra sfondo bianco

```
Causa: Mancante assets/background_2.png
Soluzione: Assicurarsi che l'immagine di sfondo esista in assets/
Oppure modificare gui_dashboard.py per usare immagine diversa
```

Problema 4: Notifiche email non inviate

```
Causa: Configurazione SMTP incorretta
Soluzione: Controllare che file .env esista con valori corretti
Per Gmail: Usare password app-specifica, non password normale
Testare impostazioni SMTP con script autonomo
```

10.2 Checklist Configurazione

- [ ] Database inizializzato (`python init_db.py`)
- [ ] File `.env` creato (opzionale per email)
- [ ] Server avviato (`python -m server.app`)
- [ ] Porta 5000 non bloccata da firewall
- [ ] SQLite riesce a scrivere in directory `db/`

---

11. Considerazioni di Sicurezza

11.1 Misure di Sicurezza Attuali

1. **Prevenzione SQL Injection**: Tutte le query usano statement parametrizzati
2. **Validazione Input**: Validazione lato server di tutti gli input
3. **Sanitizzazione Errori**: Messaggi errore generici prevengono leakage informazioni
4. **Protezione Password**: Autenticazione base per accesso client (via cli or gui_dashboard). La password è hardcoded nella programma --> alternativo(opzionale) implementare e usare password hashata.

11.2 Miglioramenti Consigliati per Produzione

```python
# 1. Hashing password (invece di confronto plaintext)
import hashlib
def verify_password(input_password, stored_hash):
    return hashlib.sha256(input_password.encode()).hexdigest() == stored_hash

# 2. Crittografia connessioni (SSL/TLS per socket)
import ssl
```

```
context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
wrapped_socket = context.wrap_socket(conn, server_side=True)

# 3. Rate limiting
from collections import defaultdict
request_counts = defaultdict(int)
# Limita richieste per indirizzo IP
```

---

12. Estensione del Sistema

12.1 Aggiunta Nuove Funzionalità

Esempio: Aggiungi Statistiche Giocatore

```
# 1. Estendi schema database (init_db.py)
ALTER TABLE giocatori ADD COLUMN partite_giocate INTEGER DEFAULT 0;
ALTER TABLE giocatori ADD COLUMN assist INTEGER DEFAULT 0;

# 2. Aggiorna funzioni db.py
def update_player_stats(id_giocatore, gol_segnati=None,
                        partite_giocate=None, assist=None):
    # Aggiorna multiple statistiche

# 3. Aggiungi endpoint API
def update_player_stats(self, id_giocatore, **stats):
    # Nuovo metodo API

# 4. Aggiorna interfacce GUI/CLI
```

Esempio: Aggiungi Calendario Partite

```
# Nuova tabella in database
CREATE TABLE partite (
    id_partita INTEGER PRIMARY KEY AUTOINCREMENT,
    data DATE NOT NULL,
    squadra_casa INTEGER,
    squadra_ospite INTEGER,
    risultato TEXT,
    FOREIGN KEY (squadra_casa) REFERENCES squadre(id_squadra),
    FOREIGN KEY (squadra_ospite) REFERENCES squadre(id_squadra)
);
```

12.2 Migrazione a Database Diverso

```python
# Sostituisci funzione _connect() in db.py
def _connect():
    # Per PostgreSQL:
    import psycopg2
    return psycopg2.connect(
        host=os.getenv("DB_HOST"),
        database=os.getenv("DB_NAME"),
        user=os.getenv("DB_USER"),
        password=os.getenv("DB_PASSWORD")
    )
```

---

13. Appendice

13.1 Tabella Riferimento File

File	Scopo	Dipendenze
`client/cli.py`	Punto ingresso CLI	`api.py`, `ui.py`
`client/gui_dashboard.py`	Punto ingresso GUI	`api.py`, `tkinter`
`server/app.py`	Server principale	`db.py`, `protocol.py`, `email.py`
`server/db.py`	Operazioni database	`sqlite3`, `threading`
`server/email.py`	Notifiche email	`smtplib`, `os`
`client/api.py`	API client-server	`protocol.py`, `socket`
`client/ui.py`	Interfaccia utente CLI	`api.py`
`db/init_db.py`	Inizializzazione database	`sqlite3`, `pathlib`
`protocol.py`	Protocollo messaggi	`json`

13.2 Riferimento Variabili d'Ambiente

Variabile	Default	Scopo
`SMTP_HOST`	`smtp.gmail.com`	Hostname server SMTP
`SMTP_PORT`	`587`	Porta server SMTP
`SMTP_SENDER`	(richiesto)	Indirizzo email mittente
`SMTP_PASSWORD`	(richiesto)	Password autenticazione SMTP

Variabile	Default	Scopo
`SMTP_RECIPIENT`	(richiesto)	Indirizzo email destinatario
`SERVER_HOST`	`127.0.0.1`	Indirizzo bind server
`SERVER_PORT`	`5000`	Porta server
`APP_PASSWORD`	`mypass`	Password autenticazione client

---

14. Licenza e Crediti

14.1 Componenti di Terze Parti

- **Python Standard Library**: Funzionalità core
- **Tkinter**: Framework GUI (incluso con Python)
- **python-dotenv**: Configurazione ambiente (opzionale)
- **SMTP Library**: Configurazione della posta elettronica (opzionale)

14.2 Riconoscimenti e Fonti Utilizzate

1. Strumenti per Diagrammi Database

- **dbdiagram.io**: Utilizzato per creare il diagramma ER (Entità-Relazione) del database. Questo strumento online ha permesso di visualizzare chiaramente le relazioni tra le tabelle `squadre` e `giocatori`.

2. Assistenza AI per Sviluppo

- **ChatGPT/DeepSeek AI**: Utilizzati per:
  - Implementazione di funzionalità Tkinter avanzate
  - Sintassi Python complessa di cui non ero a conoscenza
  - Audit e debugging del codice
  - Ottimizzazione di algoritmi e strutture dati

3. Ispirazione GUI e Design

- **Progetto GitHub "Serie A Manager"**: Football_management_system Sports_management_system
  - Ispirazione per l'interfaccia grafica
  - Layout dashboard e organizzazione widget
  - Gestione visualizzazione dati in tabelle

- Implementazione funzionalità CRUD

4. Altri Progetti e Fonti di Riferimento

1. **Python Socket Programming Tutorial**: Real Python - Socket Programming
   - Implementazione client-server TCP
   - Gestione multi-threading
2. **SQLite with Python Guide**: SQLite Tutorial - Python
   - Operazioni CRUD con SQLite
   - Gestione transazioni e vincoli
3. **Email SMTP with Python**: Python Email SMTP Documentation
   - Implementazione sistema notifiche email
   - Configurazione SMTP con TLS

5. Strumenti per Documentazione

- **Markdown Language**: Documentazione inizialmente scritta in formato Markdown (.md)
- **Obsidian software**: Conversione della documentazione da Markdown a PDF per la consegna finale

6. Design e Grafica

- **Canva**: Utilizzato per modificare e personalizzare l'immagine di sfondo (`background_2.png`)

7. Gestione Progetto

- **Git/GitHub**: Version control e gestione codice sorgente
- **Draw.io**: Creazione diagrammi di flusso e architettura
