UNIVERSITÀ DEGLI STUDI DI BERGAMO

Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione

Corso di laurea in
Ingegneria Informatica

Classe n. L-8

Progettazione e sviluppo di un'applicazione web per lo scouting tecnico nel volley

Candidato:
Alessandro Rocco

Matricola n. 1081179

Relatore:
Chiar.ma Prof.ssa
Silvia Bonfanti

Anno accademico
2024/2025

Sommario

Capitolo 1: Introduzione **3**
1.1 Struttura e utilizzo dell'app 3
Capitolo 2: Progettazione **5**
2.1 Software lifecycle 5
2.2 Configuration management 6
2.3 Software quality 6
2.4 Requirement engineering 8
2.4.1 Registrazione dei dati 8
2.4.2 Visualizzazione dati (statistiche) 10
2.4.3 Altri requisiti funzionali 10
2.4.4 Possibili implementazioni future 10
2.4.5 MoSCoW 11
2.4.6 Requisiti non funzionali 12
2.5 Modeling UML 13
2.5.1 Use case diagram 13
2.5.2 Sequence diagram 14
2.5.3 Activity diagram 15
2.5.4 State Machine diagram 16
2.5.5 Class diagram 17
Capitolo 3: Implementazione **18**
3.1 Software architecture 18
3.1.1 Architettura 18
3.1.2 Tecnologie utilizzate 19
3.1.3 Database 20
3.2 Software development 21
3.3 Software testing 22
3.4 Software maintenance 22
Capitolo 4: Conclusioni **24**
4.1 Implementazioni future 24
Bibliografia **25**

Capitolo 1: Introduzione

Recentemente abbiamo vissuto un esponenziale aumento dell'importanza dei dati e del loro utilizzo negli ambiti decisionali più disparati, tra questi lo sport. Specificatamente, nella pallavolo dove ogni azione è caratterizzata da attività sostenute e decisioni prese velocemente e strategicamente, trova spazio l'analisi critica delle prestazioni dei singoli giocatori. Entra in gioco qui lo scouting tecnico: raccogliere, organizzare e interpretare i dati ricavati fornisce non solo informazioni riguardanti le performance individuali del giocatore o di squadra ma dà anche la possibilità di circoscrivere aree tecniche riguardanti i fondamentali della disciplina al fine di organizzare strategie migliorative e potenziarne la padronanza.

Lo stile di gioco del singolo componente della squadra, nel volley, è fortemente personalizzato e questo porta allenatori e staff tecnico ad avere giudizi soggettivi e quindi impressioni "falsate". Le performance del singolo variano poi leggermente da una partita all'altra e dunque avere informazioni oggettive basate sull'evidenza porta ad avere una visione di gioco più realistica e fondata. A questo si aggiunge anche un supporto decisionale in funzione della tattica avversaria che porta vantaggi nella lettura del gioco nel suo insieme.

La web app MatchVisionVB si inserisce proprio in questo frangente, offrendo uno strumento intuitivo e professionale per lo scouting. Nello specifico, con i dati raccolti in tempo reale durante le varie partite, si possono analizzare tabelle e statistiche successivamente generate, in modo da avere una visione di insieme quantitativa e precisa volta alla crescita tecnica e tattica del team.

1.1 Struttura e utilizzo dell'app

L'app appena aperta richiede il *login* da parte dell'utente. Questo perché, a causa dell'unico database su cui è basata, l'utilizzatore deve poter accedere soltanto ai propri

dati. Una volta loggato si troverà nella *dashboard*, da dove potrà decidere se *avviare una nuova partita, aggiungere nuovi giocatori*, o *creare una nuova squadra* con giocatori precedentemente creati. In questa "schermata home" si troveranno anche delle sezioni in cui consultare le proprie risorse, quali i giocatori inseriti in passato, le partite già analizzate e le squadre formate per altri match. Ognuna di queste sezioni ha una pagina di approfondimento, ad esempio i giocatori possiedono le statistiche generali per tutte le partite disputate, oltre a quelle per i singoli match.

La schermata di vero e proprio scouting si presenta invece con un campo da volley e le icone dei giocatori disposte secondo la formazione decisa inizialmente. Per registrare dati si seleziona il player di interesse e si sceglie il fondamentale oltre al livello di qualità con cui è stato eseguito, tramite un menu. Si possono registrare anche eventi generali come cartellini, cambi, time-out, doppi falli, e il punteggio si incrementa sia manualmente che anche in base al risultato di un tocco durante il gioco.

Capitolo 2: Progettazione

2.1 Software lifecycle

Per l'implementazione delle funzionalità dell'app è stato seguito un approccio *iterativo-incrementale* tenendo conto anche della caratteristica principale del Extreme Programming ovvero l'importanza dei test, e della prototipazione evolutiva.

Ogni nuova feature veniva progettata, realizzata e testata singolarmente eseguendo tutto il sistema, per poi essere integrata nell'ambiente esistente. Il modello di processo che più si avvicina a quello seguito è il RAD, con una deadline finale entro la quale i requisiti specificati dovevano essere soddisfatti.

Le 4 fasi in cui il RAD è suddiviso sono:

- **pianificazione dei requisiti**: il MoSCoW è il metodo utilizzato per la stesura dei requisiti ed è utile per suddividerli in base alla priorità e importanza che gli si vuole dare. Come primaria funzione ha comunque quella di analizzare i requisiti che l'app deve avere, in accordo col cliente futuro utilizzatore;

- **progettazione dell'applicazione**: in questa fase si creano dei prototipi dell'applicazione e in collaborazione col cliente si applicano modifiche e migliorie. Nel caso di MatchVisionVB i prototipi erano i singoli moduli o funzionalità che si andavano a sommare per creare il prodotto finale;

- **costruzione**: è la fase in cui si scrive il codice vero e proprio. Spesso è di nuovo alternata alla progettazione, sempre in collaborazione col cliente;

- **cutover**: qui il sistema viene rilasciato e il cliente formato al suo utilizzo.

2.2 Configuration management

Il codice e la documentazione sono stati inseriti nel repository di GitHub al link https://github.com/arocco3/MatchVision. L'utilizzo di GitHub ha permesso di tenere traccia delle modifiche e aggiunte fatte nel corso dello sviluppo, documentarle attraverso la descrizione dei commit e organizzare l'implementazione in modo sicuro tramite l'utilizzo di branch per separare la realizzazione delle varie funzionalità. Per ognuna di esse è stato creato un branch nella sezione "task/".

È stata utilizzata anche una Kanban Board denominata "To Do List" per annotare e classificare come "ToDo", "In Progress" o "Done" i task da implementare nell'app.

2.3 Software quality

Per lo sviluppo dell'app si sono seguiti gli standard di qualità definiti da McCall.

Funzionamento del programma:

- **Correttezza**: l'app soddisfa le specifiche e gli obiettivi definiti all'avvio del progetto, dunque i requisiti inizialmente previsti insieme alle funzionalità sono stati garantiti;
- **Affidabilità**: tutte le feature sono state sottoposte a test, garantendo la corretta esecuzione delle attività previste in condizioni di normale utilizzo. Tuttavia, come per qualsiasi applicazione, non si possono escludere del tutto eventuali malfunzionamenti imprevisti in scenari non considerati durante la fase di sviluppo e testing;
- **Efficienza**: l'app è pensata per avere un tempo di risposta rapido rispetto agli input ricevuti, requisito fondamentale in quanto il ritmo della pallavolo è anch'esso sostenuto;

- **Integrità**: Il programma è sicuro in quanto i dati che l'utente salva sul suo profilo sono protetti da password e l'accesso è gestito dal servizio di autenticazione;

- **Usabilità**: L'interfaccia dell'app è intuitiva e di facile comprensione. L'utilizzo si presenta lineare e l'unica difficoltà si potrebbe presentare nel prendere familiarità con la schermata di gioco.

Revisione del programma:

- **Manutenibilità**: l'app, nonostante la complessità, è risultata semplice da correggere in caso di errori o bug. Questo risultato è stato raggiunto grazie all'adozione di solide pratiche di ingegneria del software, divisione in MVC e l'inserimento di commenti dettagliati;

- **Testabilità**: l'app è testabile;

- **Flessibilità**: la struttura dell'app è stata sviluppata con un buon grado di modularità, consentendo l'aggiunta di nuove funzionalità in futuro, senza modificare eccessivamente il codice esistente ma semplicemente annettendo nuove righe;

Transizione del programma:

- **Portabilità**: il sistema di MatchVisionVB è pensato per offrire uno strumento intuitivo e di facile utilizzo, accessibile soprattutto da dispositivi come tablet, PC ma anche da smartphone sebbene, per motivi pratici, meno comodi;

- **Riutilizzabilità**: la struttura dell'app potrebbe essere riutilizzata per lo scouting in altri sport ma si dovrebbero apportare non poche modifiche alla parte modulare e alla struttura del database.

2.4 Requirement engineering

Per stilare l'elenco dei requisiti mi sono avvalso del supporto di esperti ed appassionati dello scouting tecnico, ma anche della mia esperienza diretta tramite l'utilizzo di applicazioni già pubblicate da terzi. Il confronto con i probabili futuri utilizzatori è stato di grande aiuto non solo per comprendere al meglio i punti di forza e le criticità di app simili già in circolazione, ma soprattutto per definire i requisiti cosiddetti "Must Have" e "Should Have" da inserire nel progetto.

Il loro elenco è suddiviso in due sezioni principali, ciascuna relativa a uno dei moduli funzionali fondamentali dell'app: il primo dedicato alla registrazione dati durante la partita (scouting); il secondo per la visualizzazione e analisi statistica dei dati.

Questi due moduli sono strutturalmente molto diversi tra loro per quanto riguarda l'implementazione, dunque la scissione è dovuta.

2.4.1 Registrazione dei dati

Il modulo per la raccolta dei dati sarà caratterizzato da un'interfaccia utente rappresentante il campo con la formazione dei giocatori (le loro icone rappresentano i numeri di maglia), alcune informazioni tecniche specificate poi nell'elenco e una sezione apposita per la valutazione qualitativa del fondamentale[1] eseguito, per ogni singolo giocatore. Da quest'ultima deriveranno le informazioni base per l'alimentazione della parte analitica del sistema.

Funzionalità previste:

- Creazione squadra:
    - Inserimento nome, numero e ruolo giocatori;

[1] Nota tecnica: per "fondamentale" si intende una tipologia di tocco eseguita dal giocatore (servizio, attacco, ricezione, alzata, difesa, muro), e il grado di qualità del tocco è lasciato al giudizio soggettivo dell'utente utilizzatore dell'app.

- Creazione partita:
    - Schieramento giocatori (suddivisi in titolari e panchina);
    - Selezione dei liberi;
- Scelta squadra al servizio;
- Animazione rotazione al “cambio palla” conquistato o manualmente;
- Rilevazione alzata avversaria (per il successivo dato di “distribuzione palla”);
- Visualizzazione di contatori per:
    - Time-out tecnici;
    - Cambi;
    - Doppi cambi;
    - Cambi medici;
    - Cartellini;
    - Punteggio (parziali set);
- Tasti per la registrazione di un punto generico (eseguito/subito);
- Tasto per il punto conteso (a nessuna squadra viene assegnato il punto, le relative statistiche vengono cancellate);
- Premendo sull’icona del giocatore in campo si aprirà un menu per la rilevazione della qualità del tocco;
- Possibilità di annullare le ultime azioni eseguite.

2.4.2 Visualizzazione dati (statistiche)

La visualizzazione dati sarà costituita da un numero di tabelle pari ai set giocati, divise per giocatore e fondamentale eseguito (righe e colonne), e in ogni cella sarà presente un numero esplicativo delle quantità e degli esiti dello stesso.

Funzionalità previste:

- Visualizzazione di tabelle suddivise in giocatori (colonne) e fondamentali (righe) con tocchi totali e parziali (per i diversi gradi di qualità);
- Possibilità di applicare filtri alle tabelle per una visualizzazione più mirata dei dati;
- Possibilità di esportare in PDF le tabelle;
- Possibilità di consultare match precedenti con le relative statistiche;
- Possibilità di consultare le statistiche generali del giocatore singolo.

2.4.3 Altri requisiti funzionali

Di seguito un elenco di requisiti non rientranti nelle due categorie precedenti:

- Login scouter;
- Salvataggio squadre per evitare in futuro di dover reinserire i giocatori;

2.4.4 Possibili implementazioni future

Di seguito un elenco di possibili modifiche o aggiunte per versioni successive a quella di rilascio.

- Selezione della zona del campo coinvolta per ogni tocco avversario (ad es. traiettoria d'attacco).

2.4.5 MoSCoW

Per stilare la lista dei requisiti si è seguito il *triage* definito come MoSCoW. I requisiti vengono divisi in base alle priorità che gli si dà. Nel caso di MatchVisionVB, la gran parte dei requisiti fa parte della sezione "must have" e, di fatti, sono stati implementati nella prima versione dell'app. Gli unici "could have" sarebbero i requisiti inseriti nella sezione delle possibili implementazioni future.

- **Must have**: in questa sezione vengono inseriti i requisiti senza i quali l'app non potrebbe essere definita funzionante;
- **Should have**: qui si trovano i requisiti importanti ma non di critica importanza per la prima versione;
- **Could have**: funzionalità opzionali che non variano significativamente il funzionamento dell'app e che vengono implementate solo se le risorse temporali lo consentono;
- **Won't have**: elementi esclusi momentaneamente dalla versione in sviluppo, ma considerabili successivamente.

Di seguito la tabella in cui sono organizzati i requisiti. Ad ognuno è stato abbinato un codice per evitare ripetizioni: "RD" sta per Registrazione dati, "VS" per Visualizzazione dati mentre "AF" per Altri requisiti funzionali. I requisiti sono numerati incrementalmente.

Must Have	Should Have	Could Have	Won’t Have
RD#1, RD#2, RD#3, RD#7, RD#8, RD#10, VD#1, VD#4, VD#5, AF#1, AF#2	RD#9, RD#5, RD#6, VD#2	RD#4, RD#11, VD#3	Gestione multiutente con sincronizzazione in tempo reale; integrazione supporto per la visualizzazione dei dati in streaming video

Tabella 1: MoSCoW

2.4.6 Requisiti non funzionali

Di seguito si trovano i requisiti non direttamente collegati alle funzioni che l’app deve svolgere ma che comunque migliorano l’esperienza di utilizzo da parte dell’utente.

- **Design**: deve privilegiare la semplicità d’uso con percorsi chiari e semplici permettendo all’utente azioni immediate. Deve anche essere *responsive* per adattarsi a diversi tipi di schermo;
- **Scalabilità**: la struttura dell’app deve permettere l’aggiunta di nuove funzionalità e la gestione di un numero crescente di utenti, e con essi i loro dati, il tutto senza perdere prestazioni;
- **Persistenza dei dati**: il sistema utilizza un database affidabile (relazionale);
- **Sicurezza**: le comunicazioni con il e dal database avvengono tramite protocollo HTTPS.

2.5 Modeling UML

UML (Unified Modeling Language) è un linguaggio di modellazione semi-formale creato per descrivere la struttura principalmente di progetti basati su linguaggi orientati agli oggetti. Descrive le regole per la costruzione di modelli che semplifichino lo sviluppo e visualizzazione d'insieme di progetti più o meno complessi, ed è fondamentale una crescente qualità di modellazione con quanto più è ampio il sistema da descrivere.

Come riportato sul sito della documentazione ufficiale di UML: "UML does not guarantee project success but it does improve many things" (OMG, 2025).

2.5.1 Use case diagram

Il diagramma dei casi d'uso esprime ciò che gli attori possono fare (i casi d'uso) quando interagiscono con il sistema.

I casi d'uso descrivono le funzionalità previste dal sistema in fase di sviluppo. Gli attori rappresentano i ruoli che gli utenti possono adottare.

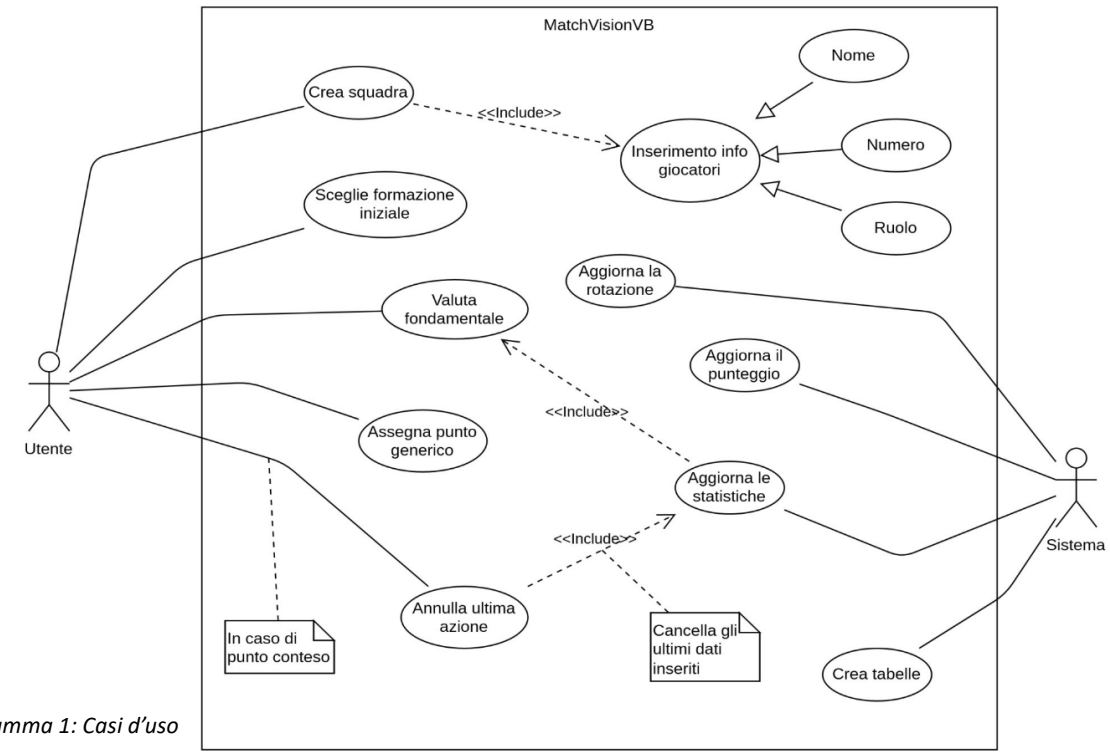
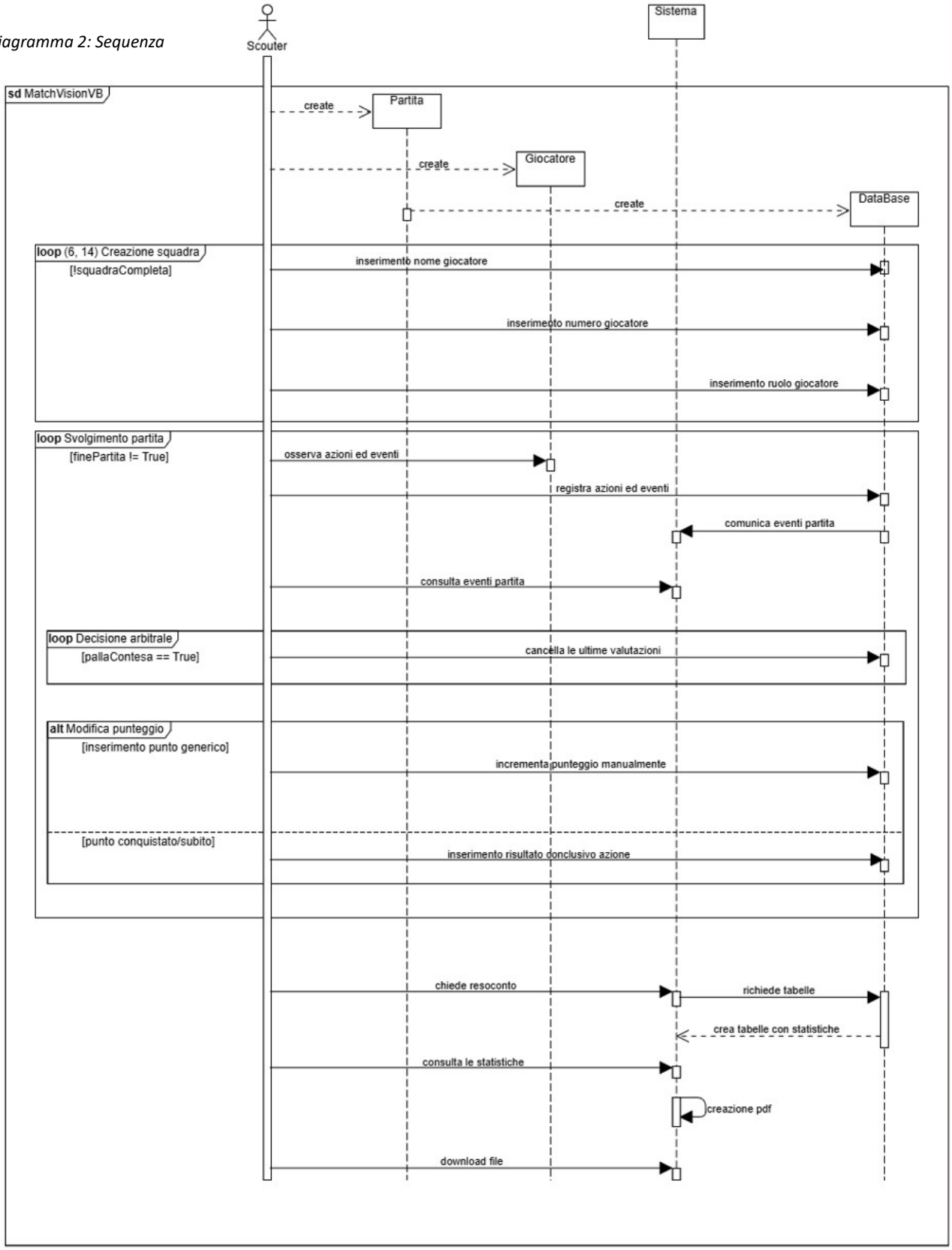


Diagramma 1: Casi d'uso

2.5.2 Sequence diagram

Il diagramma di sequenza specifica come vengono scambiati i messaggi e dati tra partner di interazione seguendo un ordine cronologico.

Diagramma 2: Sequenza



2.5.3 Activity diagram

Il diagramma delle attività specifica il comportamento dell'utente a diversi gradi di granularità. L'elemento base sono le azioni che vengono coordinate tra loro seguendo il token. I rami collegano le attività e le azioni tra loro definendo i flussi.

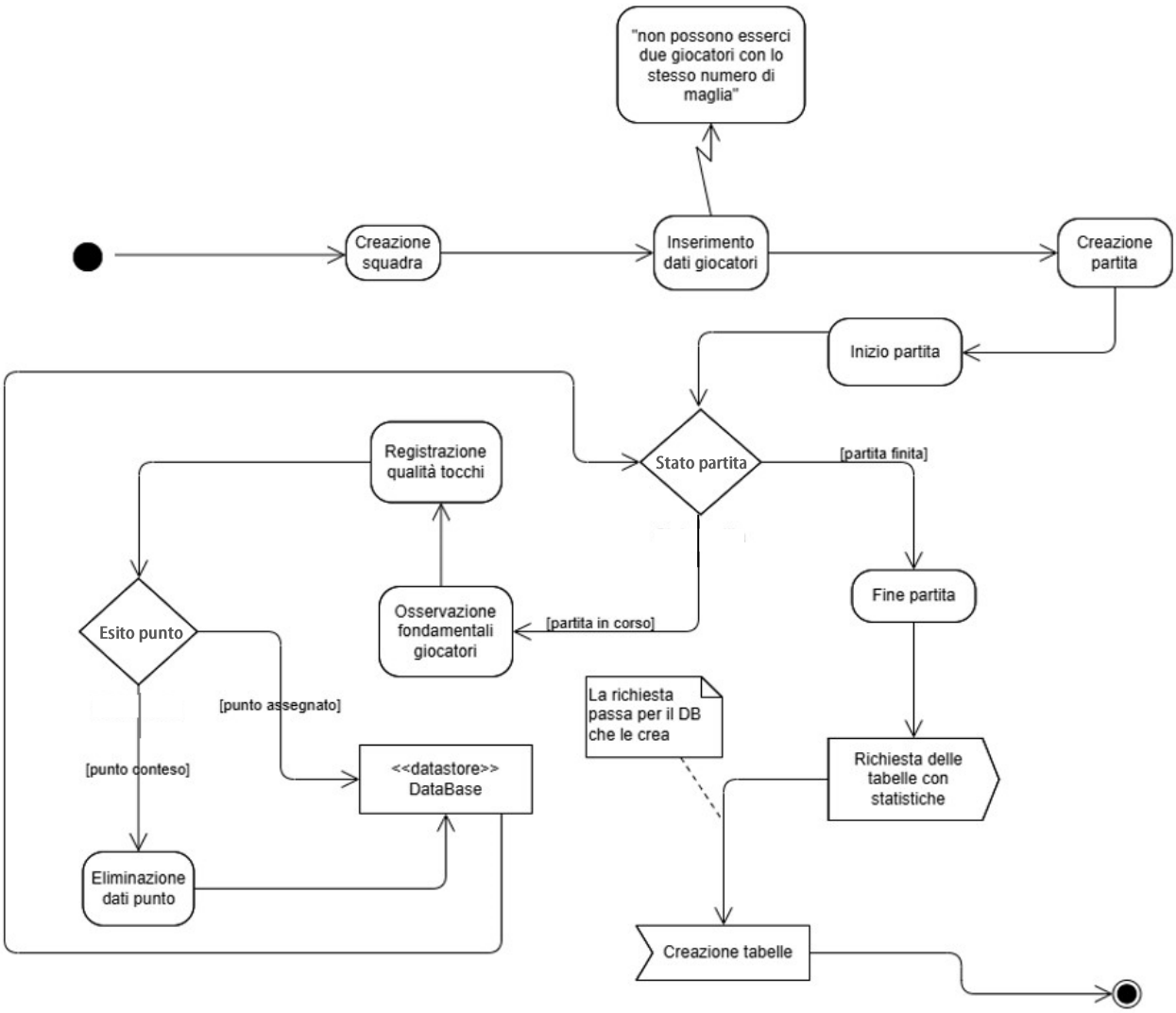


Diagramma 3: Attività

2.5.4 State Machine diagram

Il diagramma della macchina a stati descrive gli stati finiti che un oggetto assume durante la sua vita. Mostra come si verificano le transizioni da uno stato all'altro in funzione degli eventi e quale comportamento esibisce ogni oggetto.

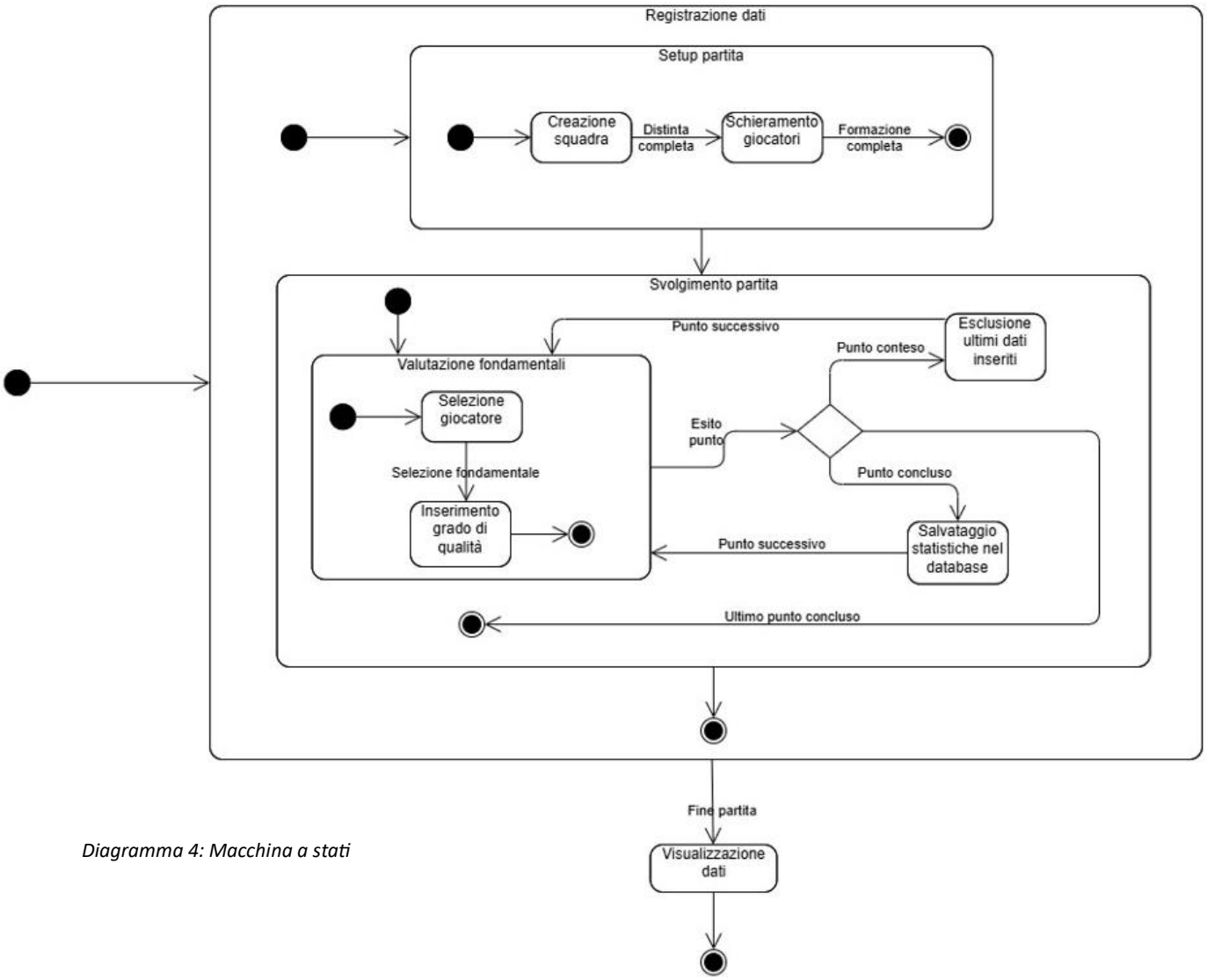


Diagramma 4: Macchina a stati

2.5.5 Class diagram

Nel diagramma delle classi si descrivono piani di costruzione per insiemi di oggetti simili di un sistema. Gli oggetti sono istanze delle classi, ed ognuno ha dei propri attributi (caratteristiche strutturali) e operazioni (comportamenti).

In questo caso il diagramma delle classi rappresenta precisamente anche la struttura del database sul quale il sistema si basa.

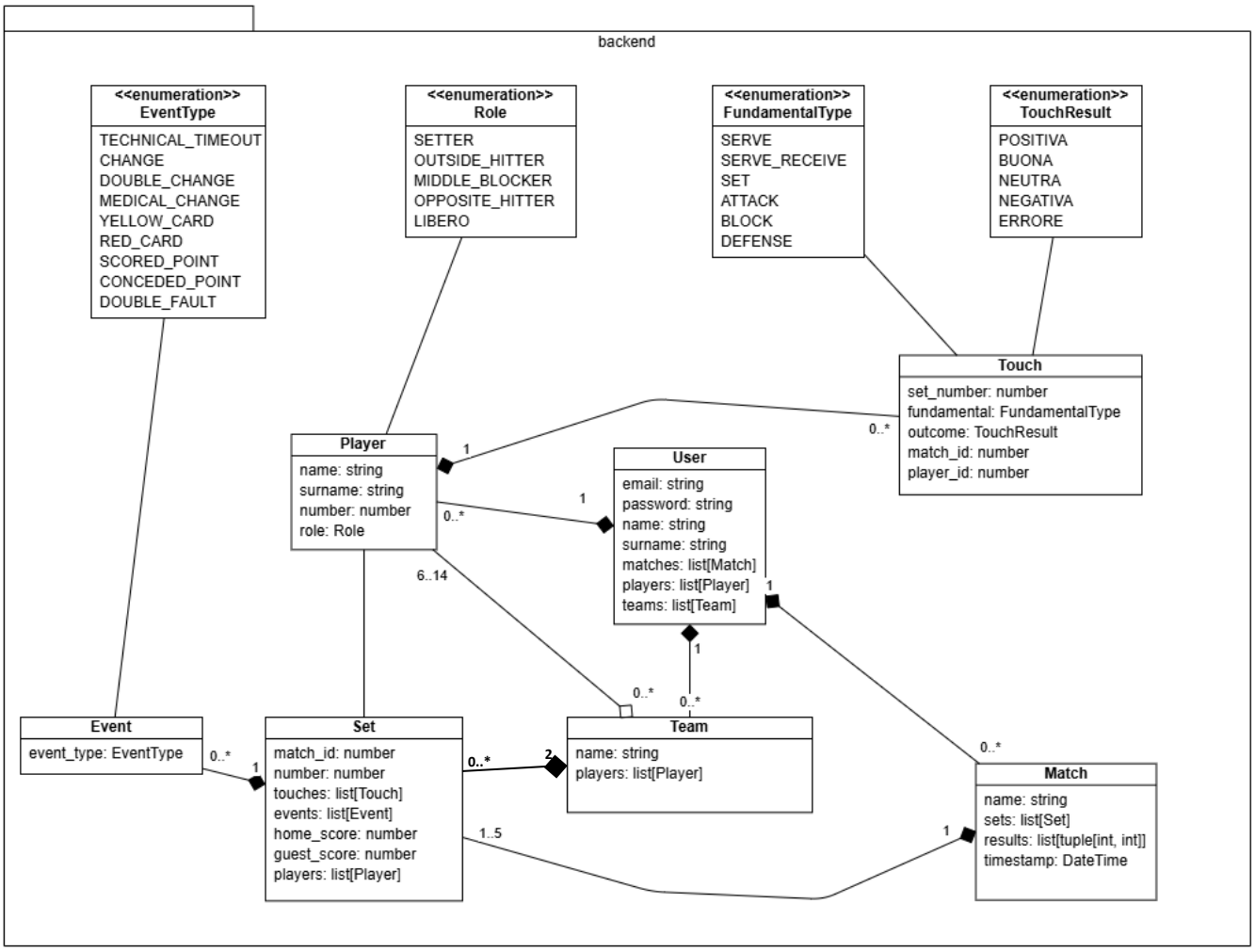


Diagramma 5: Classi

Capitolo 3: Implementazione

3.1 Software architecture

3.1.1 Architettura

L'architettura Model-View-Controller è un paradigma di progettazione software che suddivide la struttura del progetto in tre componenti principali, separando la logica di gestione dei dati, la loro presentazione e il controllo di flusso. Nel caso di MatchVisionVB la suddivisione è applicata così:

- **Model**: nel progetto il *Model* è rappresentato dal *backend* di Django, specificatamente dalla definizione delle classi in *models.py.* Queste classi definiscono la struttura dei dati e attraverso l'*Object-Relational Mapping* (ORM) mappano il database sottostante. In questo caso specifico la struttura dei modelli rispecchia appieno la composizione logica del DB;

- **View**: è gestita dal *frontend* sviluppato in Angular. Si occupa di presentare i dati all'utente e della sua interazione col sistema. Visualizza in modo dinamico le informazioni che riceve dal *Model* senza accedervi, aggiornando il contenuto della pagina evitando di ricaricarla ma rimpiazzando i componenti;

- **Controller**: questo ruolo è coperto dal sistema di API Rest implementato in Django. Le API fungono da intermediarie tra il Model e la View ricevendo richieste HTTP da parte di Angular, elaborandole, accedendo ai dati attraverso il model e restituendo il risultato della richiesta di nuovo al frontend. In questo modo si viene a creare un'architettura *client-server 3-tier* (perché è presente anche un database) che migliora la scalabilità dell'app consentendo in futuro di aggiungere altri client.

Di seguito il diagramma dei componenti, che descrive la struttura e il modo in cui i tre moduli comunicano tra loro.

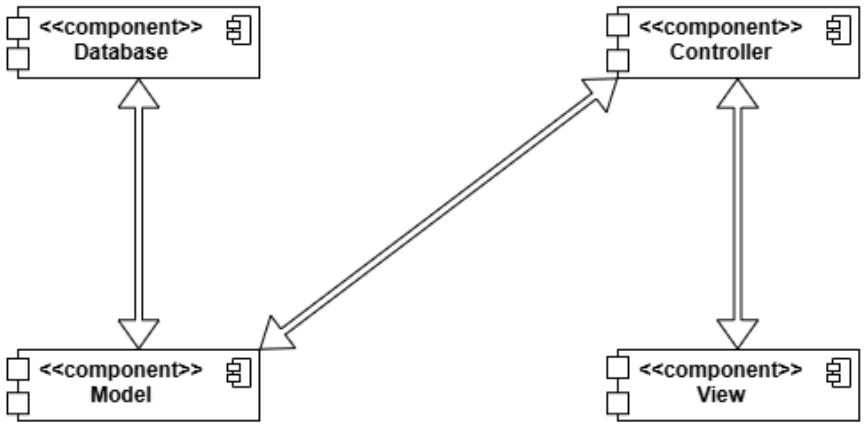


Diagramma 6: Componenti

3.1.2 Tecnologie utilizzate

Di seguito verranno indicati i linguaggi e i framework usati ed il loro ruolo.

- **Angular**: è un framework basato su TypeScript per lo sviluppo di applicazioni web *single-page*. Per MatchVisionVB è stato usato per costruire un *frontend* interattivo e *responsive* e per gestire il *routing* delle "pagine". È detto *component-based* ed ogni suo componente è articolato in logica, stile e struttura;

- **HTML**: è un linguaggio usato per strutturare il contenuto delle pagine web e gestirne la disposizione degli elementi che le costituiscono. Con Angular non si parla più di pagine web ma di componenti, e HTML ne definisce la struttura;

- **SCSS**: è stato impiegato per definire lo stile dei componenti dell'app. Il design dei componenti è reso responsivo grazie a questo linguaggio che permette una buona fruizione dell'app anche su schermi di diverse dimensioni. Ciò per cui differisce dal CSS è la possibilità di aggiungere variabili in modo da rendere più modulare e riutilizzabile il codice;

- **TypeScript**: per la logica dei componenti è stata impiegata questa estensione di JavaScript a cui aggiunge l'opportunità di creare dati (tipizzazione) ed è maggiormente orientata agli oggetti. Rende il codice più mantenibile;

- **Python**: in quanto linguaggio versatile di alto livello ha trovato impiego nello sviluppo della logica del *backend* dell'app con Django, nella gestione dei modelli e nella definizione delle REST API, quindi nella comunicazione con il *frontend*;

- **Django REST Framework (DRF)**: questa estensione di Django è servita a semplificare la scrittura di API RESTful per gestire lo scambio di dati con il *backend* usando il formato JSON.

- **PostgreSQL**: per la gestione dei dati è stato scelto questo sistema per basi di dati relazionali, molto ben integrabile con Django grazie all'ORM. PostgreSQL offre consistenza e sicurezza durante il suo utilizzo oltre all'affidabilità con grandi carichi di lavoro, scenario plausibile dato che l'app sarà pubblica e gratuita.

3.1.3 Database

Lo schema logico del database è il seguente:

USER (id, email, password, name, surname, matches, players, teams)

PLAYER (id, name, surname, number, role)

TEAM (id, name, players)

MATCH (id, sets, result)

SET (id, match_id, number, touches, events, home_score, guests_score, players)

EVENT (id, event_type)

TOUCH (id, set_number, fundamental, outcome, match_id, player_id)

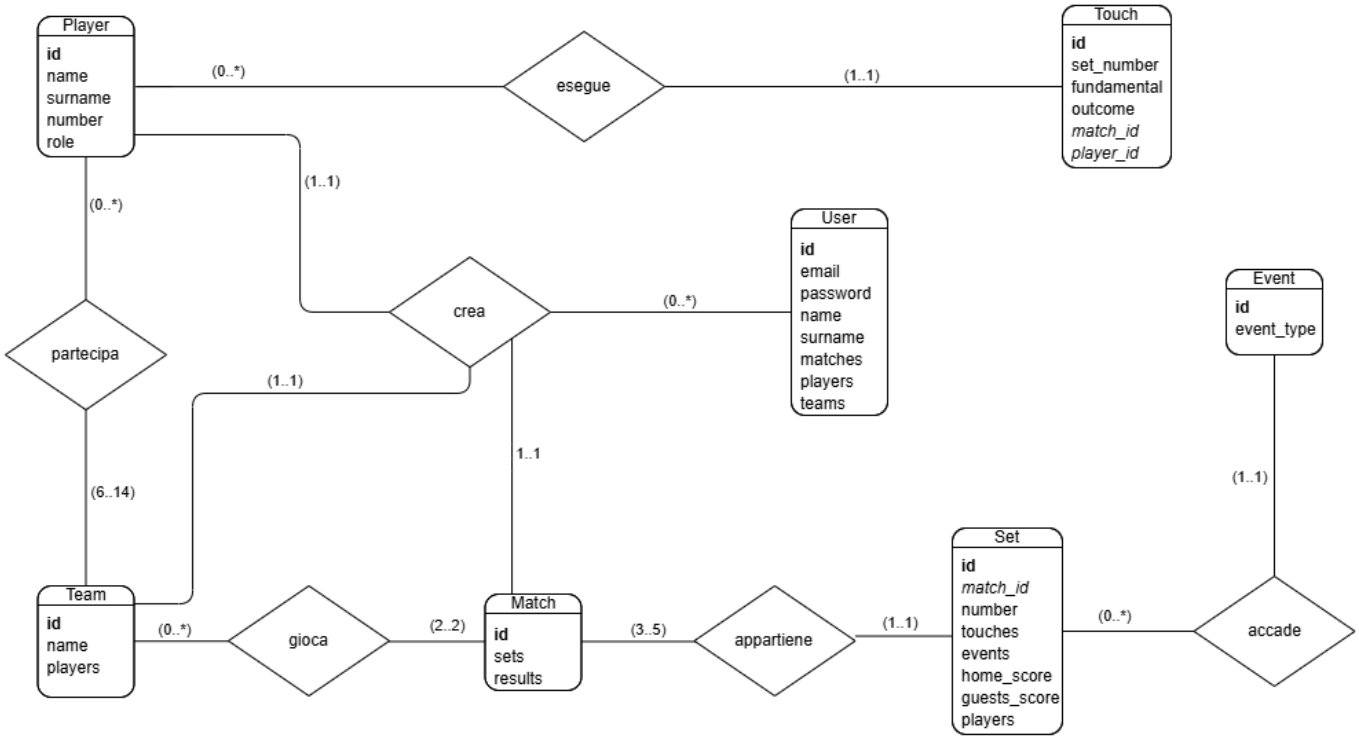


Diagramma 7: Entità-relazione database

3.2 Software development

Inserire screen app per ogni sezione (?)

3.3 Software testing

Essendo MatchVisionVB un sistema molto legato all'input dell'utente al quale non viene fornita troppa libertà d'azione, non potrebbe eseguire azioni pericolose che rischierebbero di rovinarne l'esperienza d'uso. Di fatto l'unico input libero che viene lasciato allo user sono le scelte dei nomi delle partite, giocatori e squadre. Le restanti interazioni sono molto guidate e dunque *safe* per il sistema, dato che si tratta di un'interfaccia costituita per la maggior parte da bottoni che svolgono operazioni semplici.

Per quanto la verifica del loro corretto funzionamento, ogni operazione è stata testata progressivamente durante lo sviluppo. Finché la logica dei comportamenti associati ai bottoni ed altri elementi di input non era completamente ultimata, i test erano svolti manualmente con strumenti tipici del debugging:

- istruzioni di *log* come $console.log()$ per le pagine web al fine di tenere sotto controllo il flusso dei dati e la corretta esecuzione degli eventi;
- creazione di dati fittizi in locale (direttamente nel codice) per controllare la visualizzazione e impaginazione degli stessi.

3.4 Software maintenance

L'importanza della manutenzione del software è pari a quella del suo sviluppo in quanto fondamentale per garantire nel tempo che l'app continui a fornire prestazioni qualitative e affidabili durante il suo utilizzo.

In particolare i tipi di manutenzione che verranno effettuati sono:

- **Manutenzione correttiva**: per la correzione di bug o malfunzionamenti eventualmente riscontrati e segnalati dagli utenti;
- **Manutenzione adattiva**: nel tempo i framework e le librerie (versioni di Angular e Django ad esempio) coinvolte nel progetto subiranno aggiornamenti e migliorie.

Per salvaguardare la compatibilità con queste tecnologie è necessario far stare al passo le funzioni su cui l'app si appoggia;

- **Manutenzione preventiva**: riguarda le attività svolte per la miglioria della qualità del codice, come l'aggiunta di commenti, *refactoring*, ottimizzando le API.

Capitolo 4: Conclusioni

4.1 Implementazioni future

Nonostante la *first release* dell'app soddisfi tutti i requisiti principali esistono alcune feature che se implementate darebbero un valore aggiunto al progetto.

Come già specificato nella sezione 5.4 (Possibili implementazioni future) una funzionalità che potrebbe esser aggiunta è l'opportunità di tenere traccia delle traiettorie dei colpi avversari in modo da arricchire il bacino di informazioni raccolte e potenziarne l'utilizzo.

Altre mansioni che il sistema potrebbe essere adattato ad offrire è una sezione in cui si trovano tutti i giocatori inseriti da qualunque *scouter* e consultare le statistiche generali di tutte le partite di anche stagioni sportive precedenti; si tratta dunque di avere uno storico completo delle sue *analytics*.

È considerabile un potenziamento dell'area Statistiche, inserendo funzioni maggiormente avanzate per metriche più complesse, come grafici più articolati.

Si può pensare anche ad una differenziazione di interfacce grafiche in base al ruolo dell'utente (scouter, allenatore, giocatore…) con diversi permessi e maggiori personalizzazioni.

Bibliografia

OMG, (2025). *Commento UML.* Object Management Group. Retrieved from https://www.omg.org/uml/uml-history-faq.htm