UNIVERSITÀ
DI TRENTO

Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea in
Informatica

Elaborato finale

SportIntelligence

Creazione di un sistema multipiattaforma per la fruizione di dati sportivi

Supervisore
Prof. Mauro Dragoni

Laureando
Davide Moletta

Anno accademico 2020/2021

Indice

Sommario

Il progetto descritto nel presente elaborato nasce con l'idea di creare un sistema per la fruizione di dati sportivi. L'utilità di tale sistema è presto spiegata, con il rapido progresso informatico degli ultimi anni e grazie all'evoluzione delle pagine Web e dei motori di ricerca è possibile trovare online una mole di informazioni inimmaginabile, gran parte di questi dati sono però difficili da raggiungere data la loro vastità e, a volte, inesatti o incompleti. Inoltre negli ultimi anni sono state riconosciute ufficialmente nuove categorie sportive, come gli sport elettronici, abbreviati con eSport. Tutto ciò ha contribuito alla continua crescita del numero di informazioni legate al mondo sportivo presenti su Internet.

L'obiettivo di questo progetto è quello di raggruppare grandi quantità di dati e fornire agli utenti un modo semplice e completo per visualizzarli. Con questo obiettivo in mente serviva quindi un sistema espandibile in grado di accogliere continuamente nuovi sport e informazioni relative ad essi senza mai risultare obsoleto.

Verrà descritto successivamente il lavoro svolto nel dettaglio che si compone di tre grandi fasi, la prima è quella di raccolta dati dove, tramite un Web Crawler, ho raccolto informazioni di varia natura dal sito Diretta.it [5]. Le restanti fasi sono dedicate alla visualizzazione di tali informazioni, per questo scopo sono stati utilizzati un bot Telegram [17] e un'applicazione mobile per il sistema operativo Android [1]. Tutti e tre i componenti sono stati realizzati tramite linguaggio Java, mi sono poi appoggiato a un database a grafo per il salvataggio delle informazioni raccolte dal Crawler e a Firestore [7], un database NoSQL, per la condivisione dei dati tra applicazione e bot.

Durante tutto lo sviluppo ho dato una grande importanza alla scalabilità, punto chiave per la corretta riuscita del progetto, ogni elemento creato è infatti espandibile in più modi dando così la possibilità di crescita futura del sistema.

1 Introduzione

Con la rapida evoluzione tecnologica si sono venute a creare immense collezioni virtuali di dati riguardanti qualsiasi ambito, provenienti da fonti ufficiali e non, ciò ha comportato un aumento esponenziale delle informazioni, coerenti con la realtà o meno, disponibili al pubblico.

Il tennis e tutti gli altri sport hanno subito lo stesso trattamento, sono infatti disponibili online enormi quantità di informazioni di vario tipo e natura relative ad essi.

Col tempo si è quindi rivelato necessario raggruppare tali informazioni in collezioni più specifiche andando a selezionare quali dati inserire e quali no, al giorno d'oggi esse vengono utilizzate per gli scopi più disparati data l'importanza economica e sociale degli sport ai nostri tempi. Tra i vari scopi figurano lo studio del comportamento umano negli sport o lo studio delle performance degli atleti sia da parte di coach o associazioni sia da parte di privati. Anche sul lato statistico sono stati svolti molti studi, uno tra tutti è quello pubblicato da Nate Silver dal nome *The Signal and The Noise: Why Most Predictions Fail – but Some Don't* [16]. Nel suo libro Silver non approfondisce solo il campo sportivo ma spazia anche in altri ambiti, tuttavia egli specifica come, a detta sua, siano necessarie grandi collezioni di dati selezionati e preferibilmente raccolti in lunghi archi temporali sulle quali utilizzare tecniche statistiche in grado di prevedere con maggior precisione avvenimenti futuri basandosi su quelli passati.

Data la quantità di studi già esistenti e calcolando le possibilità future possiamo concordare che l'utilità e le potenzialità di collezioni di dati ben strutturate siano immense sotto ogni punto di vista. Nel progetto che mi accingo a descrivere ho cercato di realizzare un sistema in grado di fornire una collezione di dati sportivi il più completa possibile e dei modi per visualizzarne il contenuto. Il progetto non fornisce stime o comparazioni in quanto lo scopo era quello di dare la possibilità all'utente di visualizzare liberamente i dati, ciò non toglie la possibilità di modificare il sistema in modo da portarlo in qualsiasi direzione si desidera.

2 Web Crawler

Il progetto è iniziato con la raccolta dei dati da Diretta.it, un sito italiano che contiene informazioni molto dettagliate relative ai tornei di vari sport oltre ad avere un archivio contenente tutte le edizioni svolte di ogni torneo. È stato deciso di limitare il progetto al tennis e, nello specifico, alla categoria ATP singolare, lasciando però spazio per eventuali aggiornamenti futuri relativi ad altri sport. Dopo un primo studio del sito per comprenderne la struttura ho cercato di trovare la soluzione migliore per raggiungere i dati oltre a selezionare quelli utili e quelli invece da scartare. Ho quindi iniziato lo sviluppo del Crawler utilizzando il linguaggio Java all'interno dell'IDE IntelliJ IDEA [9] mentre per il versioning del codice ho utilizzato il servizio offerto da GitHub [8].

2.1 Struttura

Il programma è diviso in parti: inizialmente vengono raccolte le informazioni relative ad ogni torneo, successivamente si passa alle informazioni delle edizioni dei tornei precedentemente trovati ed infine avviene l'effettiva raccolta dati di ogni partita svolta. Nello specifico le informazioni raccolte sono nome e anno per tornei ed edizioni, mentre per quanto riguarda le partite vengono raccolti tutti i dati basilari come data, partecipanti, durata, luogo, superficie, risultato e round. Oltre alle suddette informazioni vengono salvate, quando disponibili, le statistiche della partita e di ogni singolo set, lo storico di questi ultimi e le quote di vari siti di scommesse relative ad entrambi i giocatori.

L'estrazione dei dati è stata realizzata grazie a **Selenium** [13], un progetto composto da un insieme di strumenti sviluppati con l'obiettivo di simulare e automatizzare le interazioni di un utente con un Web Browser.

I suoi due utilizzi principali sono:

- **Automazione dei test**: Processo tramite il quale si creano le casistiche desiderate per scovare eventuali problemi di un sito oltre a verificarne il corretto funzionamento.
- **Web Crawling**: Si definisce Web crawling o scraping l'operazione di estrazione di dati da un sito Web tramite programmi software.

Nel mio caso ho realizzato un Crawler tramite il Selenium WebDriver [14], un'interfaccia che permette di avviare un'istanza del browser selezionato e di controllarlo tramite comandi impartiti da codice.

2.1.1 Selenium WebDriver

Il Selenium WebDriver utilizzato appunto per l'automazione delle comunicazioni con un Web Browser funziona nel seguente modo (*Figura 1.1*).

Figura 1.1: Funzionamento WebDriver di Selenium

Dove la libreria client di Selenium rappresenta il nostro programma, il **JSON Wire Protocol** è il mediatore che permette il trasferimento dei dati tra il client e il Browser Driver e quest'ultimo è un'interfaccia utilizzata per comunicare e controllare il relativo Browser. Nel mio caso ho deciso di utilizzare Google Chrome [3] e quindi il ChromeDriver [4] entrambi nella versione 95.0.4638.69.

2.2 Neo4j

Per quanto riguarda il salvataggio dei dati ho deciso di utilizzare Neo4j [10], un database a grafo NoSQL.

2.2.1 Implementazione

L'implementazione è avvenuta tramite Neo4j Desktop, un'applicazione installabile fornita dall'azienda stessa che permette di lavorare con i loro database, il programma comunica quindi con l'applicativo tramite gli appositi driver compatibili con Java. La connessione è avvenuta tramite Bolt, un protocollo di rete per le comunicazioni client-server utilizzato anche da Amazon Neptune [11].

I dati raccolti vengono, di volta in volta, inseriti all'interno del database seguendo la struttura visibile in *Figura 1.2*.

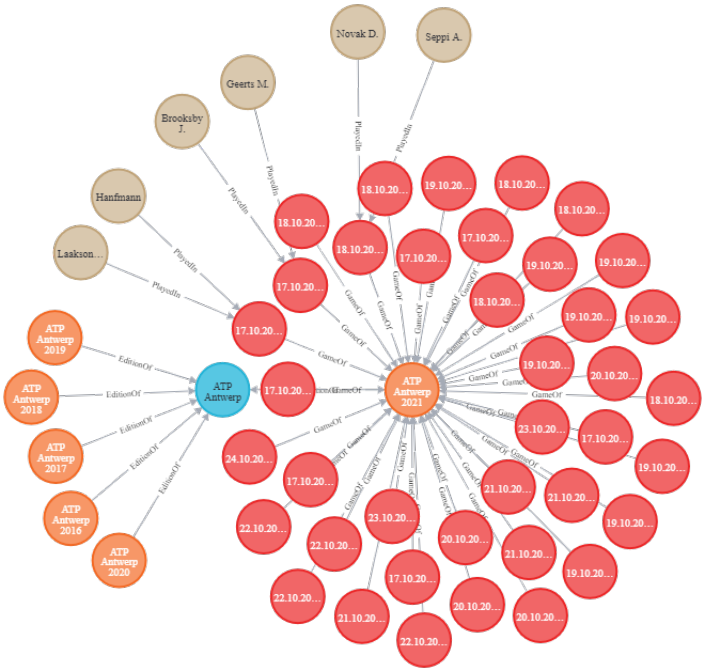

Figura 1.2: Struttura del grafo Neo4j dove:

- Il nodo blu rappresenta un torneo
- I nodi arancioni rappresentano le edizioni del torneo
- I nodi rossi rappresentano le partite dell'edizione a cui sono collegati
- I nodi marroni rappresentano i giocatori che hanno partecipato nelle partite

2.2.2 Database a grafo e relazionali

Le principali differenze tra un database a grafo e un più classico database relazionale sono le seguenti:

- Il database a grafo è composto da nodi e collegamenti mentre il database relazionale si compone di tabelle.
- Le relazioni in un database a grafo sono considerate dati e sono rappresentate tramite collegamenti tra nodi mentre in un database relazionale sono realizzate tramite chiavi.
- In un database relazionale è necessario realizzare complessi join per query complesse mentre nella controparte a grafo questo non è richiesto.

Ho deciso quindi di utilizzare un database a grafo, e nello specifico Neo4j, per la sua velocità e comodità nel raggiungere qualsiasi dato necessario, nonché per la sua struttura perfettamente adattabile al problema da me affrontato.

2.3 Difficoltà riscontrate

Durante lo sviluppo ho riscontrato due difficoltà principali oltre a soffrire di inesperienza data dal mio primo approccio a programmi di questo tipo.

Il primo problema riscontrato è relativo ad un aggiornamento del sito Diretta.it arrivato circa a metà sviluppo. L'aggiornamento in questione non ha stravolto il sito ma ha cambiato in parte la struttura dei file HTML, il che ha richiesto una piccola riprogettazione del Crawler per adattarsi ai cambiamenti dato che il driver di Selenium lavora per l'appunto tramite tag HTML. La soluzione non è stata complicata da trovare in quanto è bastato controllare i campi interessati e aggiornarli seguendo la logica del sito.

La seconda problematica invece riguarda il WebDriver di Selenium e la sua velocità. A progetto terminato mi sono reso conto che il Crawler richiedeva molto tempo per essere eseguito, si è così rivelato necessario cercare di trovare una soluzione per migliorarlo. Tramite la documentazione ufficiale di Selenium ho scoperto che alcuni tag sono preferibili ad altri se e quando disponibili in quanto sono più veloci da trovare all'interno della pagina rispetto ad altri. Ho quindi modificato il codice utilizzando i tag migliori tra quelli disponibili oltre a snellirlo migliorandone la struttura. Tale modifica mi ha permesso di ridurre il tempo di esecuzione del 40% circa portando quindi a notevoli miglioramenti del programma.

3 Applicazione Android

Conclusa la fase di raccolta dati sono passato alla progettazione dell'applicazione Android. Tuttavia, prima di passare alla vera e propria scrittura del codice, ho lavorato alla prototipazione, processo tramite il quale si definisce un prototipo parziale di quello che si intende sviluppare in modo da eliminare gran parte dei problemi prima dell'effettiva realizzazione del prodotto finale. Tale processo è stato interamente eseguito tramite l'editor grafico online Figma [6].

3.1 Prototipazione

Il primo passaggio è stato quindi quello di capire quali funzioni erano necessarie e quali superflue per creare un'applicazione usabile senza sovraccaricare l'utente con funzioni inutili o inutilmente complesse. Una volta identificate le funzionalità da implementare ho creato un wireframe low-fidelity, un prototipo molto semplificato del risultato che si vuole ottenere all'interno del quale vengono inserite solo le componenti base dell'interfaccia senza badare a elementi estetici. Ho quindi utilizzato tale wireframe per trovare la migliore soluzione di implementazione delle funzionalità che volevo inserire. Dopo aver ideato un primo modello e dopo alcune modifiche sono arrivato ad un prototipo soddisfacente (*Figura 2.1*) che mi avrebbe permesso di procedere con lo sviluppo dell'applicazione.

Figura 2.1: Schermate create durante lo sviluppo del wireframe low-fidelity

Successivamente ho trasformato il wirerfame low-fidelity in una controparte high-fidelity lavorando questa volta più sulla grafica che sulle funzionalità, sono andato quindi a rimuovere elementi indesiderati dell'interfaccia in modo da renderla sempre più semplice ma allo stesso tempo più comoda da utilizzare e meno complessa oltre a modificare alcune piccole imperfezioni della versione precedentemente creata.

A lavoro terminato ho ottenuto un prototipo praticamente identico alla versione attuale dell'applicazione se non per alcuni piccoli dettagli rivisti in corso d'opera. Non posso negare che il processo di prototipazione ha reso più semplice lo sviluppo effettivo del codice e mi ha permesso di evitare eventuali problemi di inconsistenza e usabilità, portandomi così a risparmiare tempo e a creare un prodotto migliore rispetto ad un possibile risultato ottenuto saltando il suddetto processo.

3.2 Progettazione

Terminata la fase di prototipazione sono passato alla progettazione vera e propria dell'applicazione. Per questo passaggio ho utilizzato Android Studio [2] come ambiente di sviluppo, l'IDE più diffuso e

utilizzato per la creazione di applicazioni mobili, accompagnato dal linguaggio originario di Android, Java. Per il versioning del codice ho utilizzato GitHub come per il Crawler.

3.3 Struttura

La struttura è definita da un'Activity principale contenente un Fragment container all'interno del quale vengono caricate tutte le pagine che sono definite come Fragment che a loro volta contengono i vari layout delle schermate. Scelta dettata dal fatto che, secondo la documentazione Android, le Activity sono elementi complicati da gestire, richiedono un grande sforzo computazionale e non sono consigliati per sviluppi simili a quello di questo progetto. I Fragment invece sono spesso utilizzati per la creazione di interfacce dinamiche grazie alla loro semplicità di implementazione e al minor tempo richiesto per istanziarli.

3.3.1 Home

La prima schermata presentata all'utente è la Home, pagina dalla quale è possibile selezionare uno sport tra tennis, calcio, pallavolo e basket. La versione attuale dell'applicazione permette l'accesso solo alla sezione tennis in quanto gli altri sport sono stati delegati ad aggiornamenti futuri. Oltre alla selezione dello sport è possibile, tramite il pulsante raffigurante l'icona di Telegram, accedere alla schermata di collegamento con il Bot, che approfondirò nel *capitolo 3*.

3.3.2 Tennis

Una volta selezionato il tennis l'utente verrà portato alla schermata principale del suddetto sport (*Figura 2.2*) dalla quale è possibile effettuare ricerche di specifiche partite, creare e gestire filtri di ricerca e visualizzare e gestire le partite preferite.

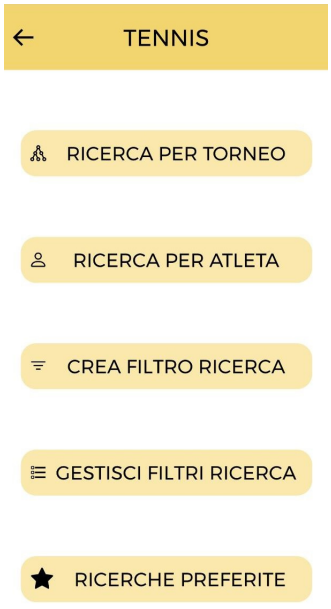

Figura 2.2: Schermata principale della categoria tennis

3.3.3 Database

Prima di descrivere la ricerca desidero spiegare come l'applicazione si interfaccia con il database Neo4j che contiene i dati da mostrare all'utente. La connessione avviene in maniera simile a quella vista per il Crawler, l'applicazione si connette da remoto allo stesso applicativo desktop di Neo4j tramite gli appositi driver e interroga tramite CypherQuery, un linguaggio simile a SQL ma ideato appositamente per Neo4j, il database per richiedere i dati necessari. Anche il metodo di comunicazione utilizzato è quello visto in precedenza, ovvero il protocollo Bolt.

3.3.4 Ricerca

La ricerca è stata il punto cardine dello sviluppo ed anche il più delicato, c'era il rischio di renderla inutilmente complessa e dispersiva. Data la presenza di più di 2.000 edizioni e oltre 150.000 partite

disputate all'interno del database sarebbe stato impossibile lasciare l'utente in balia di tutti questi dati e permettergli navigare liberamente all'interno di essi. Ho deciso quindi di creare una ricerca a passi, inizialmente è possibile scegliere se ricercare una specifica partita partendo da un giocatore oppure da un torneo (*Figura 2.3*), fatta la propria scelta vengono caricate a schermo solamente le edizioni di tale torneo o quelle in cui ha giocato il giocatore. A questo punto è necessario scegliere l'edizione desiderata e solo una volta selezionata verranno mostrate le relative partite (*Figura 2.4*).

Anche seguendo questa strada capita che, per tornei molto importanti come l'Australian Open, i dati mostrati a schermo risultino numerosi, motivo per cui ho deciso di rendere disponibile, durante tutta questa fase, una barra di ricerca che permette di facilitare ulteriormente il ritrovamento di specifici dati.

Ho scelto di impostare la ricerca in maniera simile ad un setaccio perché la trovo la soluzione più adatta in questo caso, si accompagna l'utente cercando di aiutarlo passo per passo a trovare i dati da lui ricercati cercando tuttavia di limitarlo il meno possibile, tutto questo permettendo comunque la completa visualizzazione del contenuto del database.

Ci tengo inoltre a sottolineare che la ricerca viene "salvata" passo per passo, ciò permette di ripercorrerla all'indietro evitando l'obbligo all'utilizzatore di ricominciare da capo il processo a seguito di un errore.

Figura 2.3: Ricerca per torneo e giocatore

Figura 2.4: Selezione dell'edizione

3.3.5 Selezione partita

A ricerca compiuta l'utente si trova davanti una lista di partite (*Figura 2.5*) che rispettano i criteri precedentemente scelti, all'interno di questa schermata tuttavia è possibile vedere solo parte dei dati relativi alla partita. Per la visualizzazione completa delle informazioni è necessario selezionare un match, si verrà quindi reindirizzati alla pagina descritta nella sezione successiva ovvero quella relativa alle informazioni della partita.

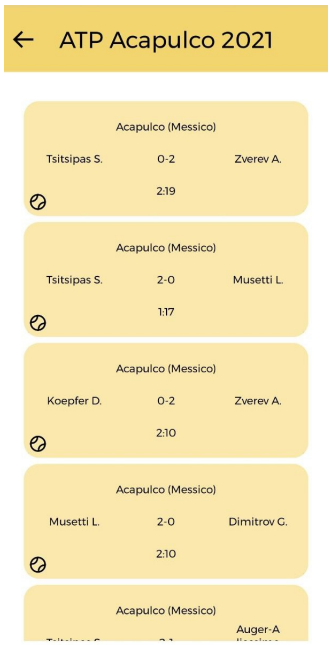

Figura 2.5: Schermata per la selezione della partita

3.3.6 Informazioni partita

Giunti al termine della ricerca e selezionata una partita possiamo visualizzarne tutte le informazioni disponibili (*Figura 2.6*). All'interno di questo Fragment è inoltre possibile rendere preferito il match grazie all'icona della stella oppure rimuoverlo dai preferiti se già ne fa parte. Grazie al menù a tendina posizionato a sinistra della stella l'utente può applicare dei filtri, che vedremo nella prossima sezione, per visionare i dati a cui è interessato. La pagina è poi strutturata nel seguente modo, la parte superiore contenente le informazioni della partita è fissa mentre la parte inferiore è inserita all'interno di una **ScrollView**, un elemento Android che permette di scorrere il suo contenuto in modo da visualizzare grandi quantità di dati senza sforare i limiti dello schermo.

Figura 2.6: Schermata per la visualizzazione dei dati della partita

3.3.7 Creazione filtri

Come citato prima è possibile creare dei filtri tramite l'apposita schermata (*Figura 2.7*), essi permettono a chi li crea di selezionare quali dati vuole visualizzare e quali no, è inoltre richiesto di assegnare un nome al filtro creato in modo da rendere più semplice l'applicazione futura. Per il salvataggio dei filtri ho deciso di utilizzare un file di testo all'interno del quale vengono salvati nome e tipologie di dati selezionati, il file verrà poi letto in fase di applicazione del filtro. Ho deciso di seguire questo approccio perché mi è sembrato il più comodo per salvare potenzialmente un gran numero di filtri e per permetterne la creazione anche in assenza di connessione ad internet.

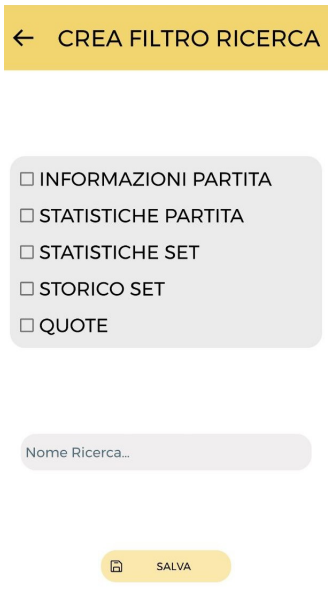

Figura 2.7: Schermata di creazione dei filtri

3.3.8 Modifica filtri

Con la creazione dei filtri ho ritenuto necessario aggiungere una schermata per la gestione degli stessi (*Figura 2.8*) all'interno della quale è possibile visualizzare tutti i filtri creati, modificarli, eliminarli singolarmente oppure eliminarli in massa. Se vengono cliccati i pulsanti per eliminare i filtri il risultato

è ovvio mentre se l'utente clicca il pulsante di modifica viene portato allo stesso Fragment della creazione dei filtri ma, invece di trovare i campi vuoti, essi vengono riempiti in modo da eguagliare quelli del filtro selezionato, l'utente è poi libero di modificare il filtro a piacimento e una volta cliccato su salva i cambiamenti vengono salvati sul file di testo.

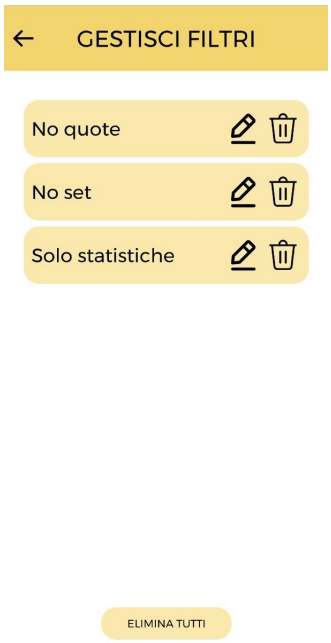

Figura 2.8: Schermata per la gestione dei filtri

3.3.9 Partite preferite

La possibilità di salvare dei preferiti nasce per permettere all'utente di accedere velocemente e con semplicità a tali partite. L'utente può salvare un match come preferito tramite la stella vista nella *sezione 3.3.6* e può poi trovarlo nella schermata dei preferiti (*Figura 2.9*) dalla quale è possibile rimuovere una partita precedentemente salvata oppure visualizzarne i dati completi. Il funzionamento è simile a quello dei filtri infatti i preferiti vengono salvati all'interno di un secondo file di testo che verrà poi letto e modificato seguendo le istruzione impartite dall'utilizzatore.

Figura 2.9: Schermata dei preferiti

3.3.10 Collegamento Bot

Questa schermata (*Figura 2.10*) è stata inserita per permettere la condivisione dei dati tra l'applicazione Android e il Bot Telegram.

Viene infatti richiesto all'utente di inserire il suo chat ID Telegram, codice univoco per l'identificazione degli utenti all'interno del servizio di messaggistica, una volta inserito il valore viene salvato in locale tramite **SharedPreferences** [15] in modo da non doverlo richiedere ad ogni accesso e i dati dell'applicazione vengono condivisi col Bot. All'interno del Fragment è anche presente la descrizione passo per passo su come trovare il proprio chat ID in modo da non creare confusione negli utenti che non capiscono cosa gli viene richiesto.

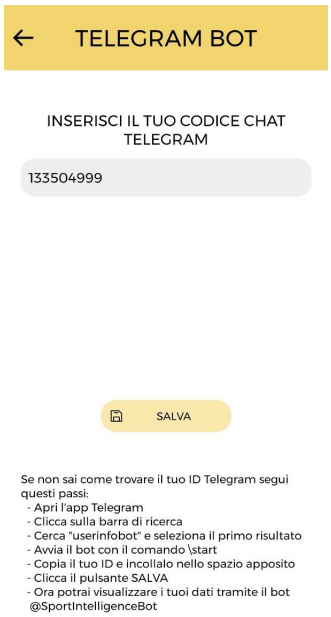

Figura 2.10: Schermata per la connessione con il Bot

3.3.11 Condivisione

Ho deciso di appoggiarmi a Cloud Firestore, un database NoSQL fornito da Google, per la condivisione dei dati tra Bot e applicazione.

Una volta inserito il chat ID le informazioni relative a filtri e match preferiti vengono salvate su Firestore. Ad ogni aggiornamento dei dati sopracitati oltre al salvataggio in locale nei file di testo è previsto un aggiornamento simultaneo dei dati contenuti nel database nel caso in cui l'utente abbia attivato la condivisione.

L'implementazione è avvenuta seguendo alla lettera la documentazione ufficiale.

L'inserimento del codice di chat è ovviamente opzionale e non comporta nessun tipo di svantaggio all'interno dell'applicazione, tutte le funzioni sono infatti disponibili per entrambe le tipologie di utenti.

3.3.12 RecyclerView

Voglio relegare una sezione alla spiegazione delle **RecyclerView** [12], elemento fondamentale per la costruzione dell'applicazione appena descritta. Esse sono implementate tramite una libreria e permettono di mostrare a schermo grandi quantità di dati efficientemente. Per l'implementazione è necessario fornire alla **RecyclerView** un layout grafico sul quale verranno strutturati gli elementi e l'insieme di dati da mostrare.

All'interno del progetto questi elementi sono stati utilizzati durante tutta la fase di ricerca, nella pagina di selezione della partita e nelle pagine relative a filtri salvati e preferiti.

Prendendo come esempio la realizzazione della pagina di gestione dei filtri sopracitata posso dire di aver proceduto nella seguente maniera. Come prima cosa ho creato l'item **saved_filter.xml** (*Figura 2.11*), elemento di layout contenente un campo di testo generico e le icone di modifica e cancellazione mentre all'interno del Fragment stesso ho inserito una **RecyclerView**. Terminata la fase grafica sono passato al codice attraverso il quale ho creato la classe **SavedFilterAdapter**, essa permette di gestire

l'inserimento dei dati nell'item grafico, ho quindi specificato alla **RecyclerView** i valori da inserire, ovvero i filtri ottenuti dalla lettura su file, tramite una lista di stringhe. Oltre all'inserimento dei dati la classe sopracitata permette di gestire eventuali comportamenti e risposte degli elementi creati, ciò mi ha permesso di impostare degli **OnClickListener** su entrambe le icone per dare la possibilità all'utente di modificare o eliminare il filtro selezionato.

Il procedimento appena descritto è stato utilizzato in tutte le pagine che presentano elementi di questo tipo adattando ovviamente i dati inseriti e gli item da visualizzare.

Figura 2.11: Item grafico utilizzato per visualizzare i filtri salvati

3.4 Sezioni

Come accennato prima, ho dato grande importanza alla scalabilità del progetto e, per permettere l'espansione dell'applicazione, essa è stata strutturata in sezioni permettendo quindi l'aggiunta di un qualsiasi numero di sport in qualsiasi momento.

3.4.1 Aggiunta nuova sezione

L'aggiunta di una nuova sezione è stata resa molto semplice, bisognerà infatti creare i propri Fragment strutturandoli come si preferisce per la corretta fruizione dei dati relativi allo sport che si sta aggiungendo. È stato purtroppo impossibile standardizzare la visualizzazione dei dati, tale impedimento è dovuto alla pesante eterogeneità degli stessi e alla diversa strutturazione dei vari sport. Una volta aggiunti tutti i Fragment necessari e creata una specie di sotto applicazione a sé stante sarà richiesto di modificare solamente due file già esistenti all'interno del progetto ovvero la **Home** e il **NavGraph**. Le modifiche richieste prevedono di aggiungere tutte le schermate create all'interno del **NavGraph** tramite l'apposita funzione fornita da Android Studio (*Figura 2.12*) e collegare la schermata **Home** già presente al primo Fragment della sezione che si vuole mostrare all'utente, nel caso del tennis ho collegato la schermata vista nella *sezione 3.3.2*. Effettuata tale modifica l'ultima cosa da fare è aggiungere la funzione di navigazione legata al pulsante relativo al nuovo sport nel codice della pagina **Home** come da esempio.

```
newSportButton.setOnClickListener(v -> {
          navController.navigate(R.id.action_homeFragment_to_newSportFirstFragment);
      });
```

Una volta eseguiti questi passaggi la nuova sezione dell'applicazione è pronta per essere utilizzata.

Per il salvataggio dei dati è possibile utilizzare lo stesso database di Neo4j utilizzato per il tennis seguendo una struttura simile a quella citata nella *sezione 2.2.1*. Un altro motivo per cui ho deciso di utilizzare Neo4j infatti è perché esso permette, dalla sua versione 3.0, di salvare grafi di qualsiasi dimensione abbattendo il precedente limite di 34 Miliardi di nodi, ciò è possibile grazie alla **Dynamic pointer compression**, tecnica tramite la quale ogni nodo può trovare i suoi vicini tramite un solo "salto" di puntatore.

Anche per la comunicazione con il Bot è possibile utilizzare lo stesso sistema visto in precedenza connettendosi a Firestore.

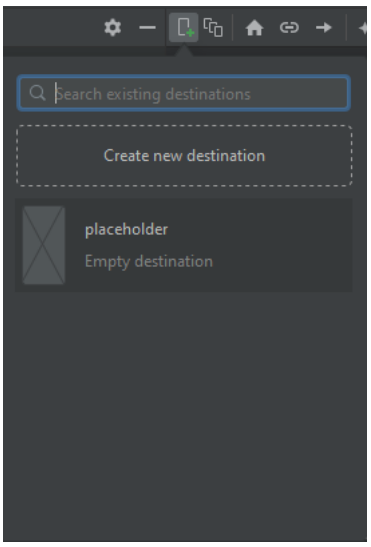

Figura 2.12: Tramite il menù a tendina è possibile selezionare tutti i Fragment che si vogliono aggiungere

3.5 Difficoltà e Test

3.5.1 Difficoltà riscontrate

L'unica difficoltà riscontrata durante lo sviluppo è stata la connessione da remoto al database di Neo4j, inizialmente infatti l'applicazione funzionava solo all'interno della mia rete locale.

Anche seguendo alla lettera la documentazione ufficiale risultava impossibile riuscire a connettersi tuttavia, dopo aver imparato a conoscere il file di configurazione del database e dopo aver provato molteplici impostazioni, sono riuscito a trovare quella ideale per me e ad abilitare la connessione da remoto.

3.5.2 Test

Per la fase di testing ho deciso di consegnare un versione completa dell'applicazione a vari conoscenti, alcuni appartenenti all'ambiente informatico e altri no in modo da avere riscontri di diversa natura. Tali test mi hanno permesso di scovare alcuni piccoli problemi di usabilità e alcuni bug minori.

Grazie a questo progetto ho compreso l'importanza dei test e del feedback degli utenti, ad esempio ho scoperto che con l'utilizzo del tema scuro di Android risultava difficile leggere porzioni di testo per via della colorazione inadeguata, problema del quale non mi ero reso conto dato che durante lo sviluppo ho utilizzato il tema chiaro.

Ogni problematica portata alla luce durante questa fase è stata corretta tempestivamente portando l'applicazione alla versione attuale.

4 Bot Telegram

Terminato lo sviluppo dell'applicazione sono passato all'ultimo componente del progetto, ovvero il Bot.

4.1 Ambiente

L'ambiente in questo caso era identico a quello utilizzato per il Crawler quindi linguaggio Java sviluppato all'interno dell'IDE IntelliJ IDEA e GitHub per il versioning del codice. L'unica differenza è che, trattandosi di un bot, c'era bisogno di avere un supporto per l'invio e la ricezione dei messaggi con l'utente, per ovviare a questo problema ho deciso di appoggiarmi a Telegram, noto servizio di messaggistica istantanea famoso anche per la numerosa presenza di bot di svariato tipo. L'idea era quella di creare un metodo alternativo per visualizzare i preferiti e i filtri salvati sull'applicazione, una volta condivisi i dati dalla stessa non rimaneva che sviluppare il Bot attorno ai dati stessi.

4.2 Progettazione

Come primo passaggio ho studiato il funzionamento dei bot in quanto non mi ero mai trovato a svilupparne uno, una volta appresi i concetti fondamentali ho provato a fare un piccolo test slegato dal concetto di questo progetto per mettere in pratica ciò che avevo imparato.

4.2.1 Creazione del Bot

Grazie all'utilizzo di Telegram è stato molto semplice creare la base del progetto, è bastato seguire la documentazione e tramite il **BotFather**, il vero e proprio padre di tutti i bot, digitare il comando di creazione. Fatto questo si riceve un Token utilizzato per collegare un qualsiasi programma al Bot in modo da poter implementare le funzioni desiderate dato che al momento della nascita non ne esistono.

Dopo il primo test e dopo aver gettato la base del progetto sono passato allo sviluppo del prodotto finale, come prima cosa ho connesso il Bot sia al database Neo4j per la lettura dei dati delle partite sia a Firestore per la lettura dei dati condivisi dall'applicazione. Le implementazioni sono state identiche a quelle utilizzate per la connessione all'applicativo Android. Stabilite le connessioni e verificato il corretto funzionamento di entrambe ho scritto il codice per la comunicazione vera e propria del Bot con l'utente, per fare questo ho utilizzato la **Telegram Bot Java Library** [18], una libreria gratuita e open source che permette di creare programmi di questo tipo utilizzando il linguaggio Java.

4.3 Stato attuale

Al momento attuale il Bot risponde a tre comandi

- **/help**: invia un messaggio all'utente con una lista di tutti i comandi disponibili.
- **/filtri**: invia tramite messaggio tutti i filtri e i relativi campi selezionati che sono stati precedentemente creati.
- **/partite**: invia tramite messaggio tutti i match preferiti salvati sull'applicazione.

4.3.1 Aggiunta nuovi comandi

È ovviamente possibile implementare altri comandi in maniera molto semplice, basterà infatti modificare il codice della classe **Bot.Java** aggiungendo una funzione simile a quella visibile qui sotto o sovrascrivendo quelle già esistenti.

```
public Ability newAbility() {
       return Ability
              .builder()
```

```
            .name("Ability name")
            .info("Ability info")
            .locality(Locality.ALL)
            .privacy(Privacy.PUBLIC)
            .action(ctx -> silent.send("messageToSend"), ctx.chatId()))
            .build();
    }
```

Cambiando i valori desiderati sarà possibile modificare la funzione adattandola all'utilizzo che se ne vuole fare.

5 Futuro

Durante tutto lo sviluppo ho cercato di modellare sia l'applicazione che il Bot in maniera tale da lasciare la possibilità di espandere ogni componente ove possibile.

La natura del sistema inoltre permette un ampio numero di modifiche e aggiornamenti volti al progresso dello stesso nonché alla sua completezza. Mi appresto ora a citarne alcuni secondo me degni di nota e che, se implementati, porterebbero grandi miglioramenti al progetto.

5.1 Futuro applicazione

Come citato prima all'interno dell'applicazione sono già presenti dei pulsanti relativi a calcio, pallavolo e basket, in futuro infatti si potrebbero implementare questi sport o aggiungerne altri ancora grazie alla struttura a sezioni citata precedentemente. Con l'aggiunta di grandi quantità di sport però rischia di nascere un problema, ovvero di sovraccaricare la pagina **Home** rendendola dispersiva. Per ovviare a ciò ho pensato che sarebbe utile dare la possibilità all'utente di visualizzare solo gli sport a cui è interessato e di nascondere quindi quelli inutili, ciò potrebbe essere implementato tramite una pagina di impostazioni o direttamente nella **Home** con apposito pulsante. Un'altra possibile modifica potrebbe essere quella di aggiungere altre tipologie di ricerche in modo da permettere all'utente di cercare, per esempio, tutti i vincitori delle varie edizioni di un torneo o tutte le vittorie di uno specifico giocatore. Il che permetterebbe di effettuare ricerche più specifiche e dettagliate.

5.2 Futuro Bot

Per quanto riguarda il Bot invece ho previsto due strade principali percorribili nel futuro.

La prima, quella intrapresa da me, prevede di continuare con l'impostazione attuale e mantenere il Bot come un "supporto" all'applicazione per fornire un'alternativa alla visualizzazione dei dati. Seguendo questa strada si potrebbero apportare varie modifiche al Bot, ad esempio si potrebbe permettere all'utente di modificare i filtri e i preferiti in modo da riflettere la modifica anche sull'applicazione. Un'altra idea sarebbe quella di dare la possibilità all'utente di creare nuovi filtri. Con l'aggiunta di nuovi sport si potrebbe implementare anche una navigazione a sezioni simile a quella dell'app in modo da mantenere i due progetti simili tra loro nell'utilizzo. Inoltre seguendo questa strada ogni modifica effettuata all'applicazione potrebbe essere aggiunta anche al Bot.

La seconda strada prevede invece di scindere completamente Bot e applicazione rendendoli due elementi separati. Si potrebbe infatti eliminare il collegamento Firestore attualmente esistente e modificare il Bot in maniera tale da renderlo simile all'applicazione implementandone quindi tutte le funzioni. Il che permetterebbe ai potenziali utenti di decidere in quale modo sfruttare i dati raccolti visto che avrebbero la possibilità di accedervi in due diverse maniere. Seguendo questa idea non ci sarebbe l'obbligo di scaricare l'applicazione per permettere l'utilizzo del Bot, tale libertà andrebbe incontro alle esigenze di quegli utenti che non vogliono scaricare applicazioni oppure a quelli che vogliono utilizzare il Bot tramite PC grazie all'applicativo Telegram Web [19], corrispettivo utilizzabile tramite Browser dell'omonima applicazione.

Entrambe le strade hanno grandi potenzialità di sviluppo tuttavia ho scelto di percorrere la prima dato che non trovo comodo l'ambiente fornito da un Bot per la visualizzazione di grandi quantità di dati. Soprattutto la parte di ricerca ha generato molti dubbi ed è stato il fattore principale che mi ha portato allo sviluppo attuale del sistema in quanto non avendo la libertà e gli strumenti forniti da un'applicazione risulterebbe complicato navigare tra le varie edizioni e partite. Sarebbe tuttavia possibile creare un Bot del genere stando molto attenti a non cadere in errori simili a quello citato, progettando quindi il tutto al meglio e creando dei comandi adatti allo scopo in modo da evitare di rendere l'ambiente inutilizzabile dall'utente.

Un'ultima possibilità sarebbe quella di seguire la seconda strada sopracitata mantenendo comunque il collegamento opzionale tra app e Bot.

5.3 Futuro del sistema

Oltre alle strade sopra descritte che prevedono la modifica e l'aggiornamento di componenti attualmente esistenti sarebbe possibile espandere il sistema in altri modi.

Un esempio potrebbe essere quello di creare una pagina Web con lo stesso funzionamento dell'applicazione così da fornire ancora più libertà all'utente nella scelta del metodo di visualizzazione dei dati. In questo caso bisognerebbe creare il sito in maniera tale da permettere una corretta navigazione tra le varie schermate e tra i vari sport seguendo magari il concetto di ricerca introdotto nella *sezione 3.3.4*.

Inoltre il database Neo4j potrebbe essere ampliato inserendo dati provenienti da più fonti e di diversa natura. L'inserimento di articoli sportivi o comunicati ufficiali delle associazioni sportive per esempio potrebbe risultare utile per gli utenti che ricercano tali informazioni.

6 Conclusioni

In conclusione mi ritengo soddisfatto per l'attuale stato del progetto, l'ambizione era quella di creare un sistema per la fruizione di dati specifici riguardanti il tennis e tale obiettivo è stato raggiunto. Ci tengo comunque a sottolineare che, come visto nel *capitolo 5*, il progetto potrebbe essere ampliato e migliorato notevolmente grazie alla sua struttura volta alla continua espansione.

Possiamo riassumere il progetto in tre passaggi principali, il primo è stato quello di raccolta dei dati tramite Web Crawler, nella seconda fase invece sono passato allo sviluppo dell'applicazione Android suddivisa a sua volta nelle fasi di prototipazione, progettazione e sviluppo. Nell'ultimo passaggio invece mi sono concentrato nella creazione di un Bot Telegram utilizzato per la condivisione dei dati.

Il lavoro svolto mi ha permesso, in primo luogo, di maturare come sviluppatore, ho infatti appreso nuove informazioni a livello di programmazione e ho rafforzato le mie conoscenze pregresse. In secondo luogo durante lo sviluppo di ogni componente sono riuscito a capire l'importanza di seguire le linee guida della creazione di un buon progetto, ad esempio ho compreso l'utilità dei prototipi e di quanto possano migliorare e velocizzare lo sviluppo di un'applicazione.

Tale miglioramento deriva anche dalla mia inesperienza e dalle mie lacune in alcuni di questi campi, fino ad ora infatti non mi ero mai trovato a realizzare Crawler o Bot ed essendo programmi leggermente particolari al primo approccio mi sono trovato completamente spaesato, tutti i dubbi e le difficoltà sono tuttavia svaniti durante lo sviluppo anche grazie alle documentazioni online. Per l'applicazione invece il discorso è un po' diverso in quanto, sia alle scuole superiori sia durante il mio percorso universitario, ho avuto modo di approcciarmi allo sviluppo Android creando qualche piccolo progetto.

Bibliografia

[1] Android. https://www.android.com/.

[2] Android studio: doc for app developers. https://developer.android.com/docs.

[3] Google chrome. https://www.google.it/intl/it/chrome/.

[4] Chromedriver. https://chromedriver.chromium.org.

[5] Diretta.it. https://www.diretta.it.

[6] Figma. https://www.figma.com/.

[7] Firestore, store and sync app data at global scale. https://firebase.google.com/docs/firestore.

[8] Github: Where the world builds software. https://github.com.

[9] Intellij idea: The capable and ergonomic java ide by jetbrains. https://www.jetbrains.com/idea/.

[10] Neo4j: The fastest path to graph. https://neo4j.com.

[11] Amazon neptune. https://aws.amazon.com/it/neptune/.

[12] Recyclerview doc. https://developer.android.com/guide/topics/ui/layout/recyclerview.

[13] Selenium automates browsers. that's it! https://www.selenium.dev.

[14] Selenium webdriver developer documentation. https://www.selenium.dev/documentation/webdriver/.

[15] Sharedpreferences doc. https://developer.android.com/reference/android/content/SharedPreferences.

[16] Nate Silver. *The Signal and the Noise*. Penguin Group, 2012.

[17] Telegram. https://play.google.com/store/apps/details?id=org.telegram.messenger&hl=it&gl=US.

[18] Telegram bot java library. https://github.com/rubenlagus/TelegramBots.

[19] Telegram web. https://web.telegram.org/k/.