FIFA CHATBOT

DOCENTI:

PROF. DOMENICO URSINO

DOTT. MICHELE MARCHETTI

STUDENTI

CIVITARESE ANDREA

LONGARINI LORENZO

PASQUINI GIOELE

RAMOVINI LORIS

Sommario

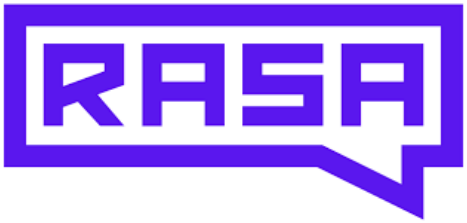
Introduzione 3
Tecnologie e Librerie 3
Rasa 3
Pandas 3
FuzzyWuzzy 3
Dataset 4
ETL 4
RASA 6
Struttura 6
Rasa NLU 6
Rasa CORE 7
Chatbot 8
Funzioni utili 8
Trovare i calciatori di una determinata squadra 8
Trovare le statistiche di un calciatore 11
Trovare l'immagine di un calciatore 13
Confrontare le caratteristiche di due calciatori 14
Trovare il miglior calciatore per ruolo/piede/campionato 17
Creare un team inserendo modulo/età/nazionalità 20
Telegram 25
Trovare i calciatori di una determinata squadra 25
Trovare le statistiche di un calciatore 27
Trovare l'immagine di un calciatore 27
Confrontare le caratteristiche di due calciatori 28
Trovare il miglior calciatore per ruolo/piede/campionato 28
Creare un team inserendo modulo/età/nazionalità 28

Introduzione

Il FIFA Chatbot è una soluzione innovativa che mira a fornire agli appassionati del famoso videogioco FIFA un assistente virtuale in grado di offrire informazioni dettagliate sui calciatori presenti nel gioco. Attraverso un'interfaccia conversazionale, gli utenti possono interrogare il chatbot per ottenere dati aggiornati sulle valutazioni dei giocatori, le loro posizioni in campo, le statistiche delle squadre e molto altro ancora.

Tecnologie e Librerie

Rasa



Rasa è un framework open source per la costruzione di chatbot e assistenti virtuali. Offre un insieme completo di strumenti per progettare, costruire e addestrare chatbot sofisticati capaci di comprendere e rispondere alle richieste degli utenti in linguaggio naturale. Rasa si distingue per la sua flessibilità e la possibilità di essere eseguito localmente o in un ambiente cloud, garantendo la piena proprietà e controllo dei dati. In seguito, approfondiremo la struttura del framework in maniera più approfondita.

Pandas

Pandas è una libreria Python ampiamente utilizzata per l'analisi e la manipolazione dei dati. Nel nostro progetto, Pandas viene impiegata per gestire il dataset "FIFA 23 Complete Player Dataset", consentendo di estrarre, filtrare e organizzare le informazioni sui giocatori in modo efficiente. Grazie a Pandas, possiamo facilmente selezionare le colonne di interesse, eseguire operazioni sui dati e preparare le informazioni per essere utilizzate dal chatbot.

FuzzyWuzzy

FuzzyWuzzy è una libreria Python che fornisce metodi per confrontare stringhe in modo "fuzzy", cioè tenendo conto di piccole differenze o errori di battitura. Questa libreria si rivela particolarmente utile nel contesto del FIFA Chatbot per gestire le query degli utenti che potrebbero non essere formulate in modo esatto. Grazie a FuzzyWuzzy, il chatbot è in grado di interpretare le richieste anche se contengono lievi imprecisioni, migliorando l'esperienza dell'utente e la qualità delle risposte fornite.

Dataset

Il dataset scelto è "FIFA 23 Complete Player Dataset", il quale rappresenta una risorsa completa e dettagliata per l'analisi di calciatori, squadre e allenatori nel contesto del famoso videogioco FIFA. Il dataset è disponibile su Kaggle[1], questo dataset copre le versioni del gioco che vanno dal 2015 al 2023, fornendo una panoramica evolutiva delle caratteristiche dei soggetti coinvolti nel corso degli anni.
Il dataset si compone di diversi file CSV, ognuno dei quali è dedicato a una specifica categoria:

- **Calciatori**: Ogni file relativo ai giocatori contiene 110 attributi che ne descrivono le abilità, le statistiche e le caratteristiche personali. Questi attributi includono, ad esempio, la valutazione complessiva, il potenziale, la nazionalità, la posizione in campo, le abilità tecniche e fisiche, e molti altri dettagli che ne definiscono il profilo.
- **Squadre**: I file relativi alle squadre presentano 54 attributi per ciascuna squadra, fornendo informazioni sulla loro composizione, sulle performance, sui valori di mercato e su altri aspetti che ne caratterizzano la forza e la reputazione.
- **Allenatori**: Infine, i file dedicati agli allenatori contengono 8 attributi per ciascun allenatore, tra cui il nome, la nazionalità, la squadra di appartenenza e altre informazioni che ne descrivono la carriera e le competenze.

Questo dataset rappresenta quindi una risorsa preziosa per analisi approfondite nel campo del calcio virtuale, ed in particolare risulta interessante da utilizzare in combinazione con Rasa, al fine di ottenere facilmente statistiche e creare la propria squadra personalizzata.

ETL

Nell'elaborazione del dataset "FIFA 23 Complete Player Dataset", ci siamo concentrati esclusivamente sul file CSV relativo ai calciatori maschi. Per semplificare l'analisi e l'utilizzo dei dati nel contesto di un chatbot, abbiamo selezionato un insieme specifico di colonne che ritenevamo più rilevanti. Queste colonne includono informazioni essenziali e sono descritte nella tabella di seguito:

Attributo	Descrizione
`player_id`	Identificativo univoco del giocatore
`player_face_url`	URL dell'immagine del volto del giocatore
`fifa_version`	Versione del gioco FIFA a cui si riferiscono i dati
`short_name`	Nome breve del giocatore
`long_name`	Nome completo del giocatore
`player_positions`	Posizioni in campo del giocatore
`overall`	Valutazione complessiva del giocatore
`potential`	Potenziale massimo del giocatore
`value_eur`	Valore di mercato del giocatore in euro
`age`	Età del giocatore
`height_cm`	Altezza del giocatore in centimetri
`weight_kg`	Peso del giocatore in chilogrammi
`league_name`	Nome della lega in cui gioca il giocatore
`league_id`	Identificativo univoco della lega
`club_name`	Nome del club di appartenenza del giocatore
`club_team_id`	Identificativo univoco del club di appartenenza
`club_jersey_number`	Numero di maglia del giocatore nel club
`nationality_name`	Nazionalità del giocatore

[1] https://www.kaggle.com/datasets/stefanoleone992/fifa-23-complete-player-dataset

`nationality_id`	Identificativo univoco della nazionalità
`preferred_foot`	Piede preferito del giocatore (Destro/Sinistro)
`pace`	Velocità del giocatore
`shooting`	Abilità di tiro del giocatore
`passing`	Abilità di passaggio del giocatore
`dribbling`	Abilità di dribbling del giocatore
`defending`	Abilità difensive del giocatore
`physic`	Condizione fisica del giocatore

Per facilitare la comprensione e la gestione dei ruoli dei giocatori, abbiamo introdotto una mappatura che semplifica le diverse posizioni in campo in categorie più ampie: "Portiere", "Difensore", "Centrocampista" e "Attaccante". Questa mappatura ci permette di raggruppare i giocatori in base alle loro funzioni principali in campo e di rendere le query più intuitive, in particolare:

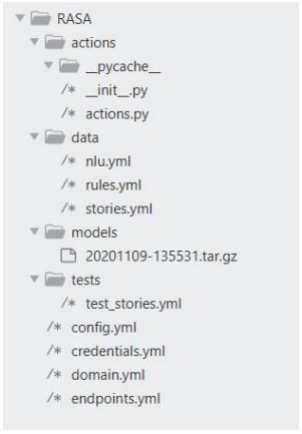
```
map_roles = {
    'GK': 'Portiere',
    'RB': 'Difensore',
    'CB': 'Difensore',
    'LB': 'Difensore',
    'RWB': 'Difensore',
    'LWB': 'Difensore',
    'CDM': 'Centrocampista',
    'CM': 'Centrocampista',
    'CAM': 'Centrocampista',
    'RM': 'Centrocampista',
    'LM': 'Centrocampista',
    'RW': 'Attaccante',
    'LW': 'Attaccante',
    'CF': 'Attaccante',
    'ST': 'Attaccante'
}
```

Questa mappatura è stata fondamentale in ottica di comprensione per l'utilizzatore finale, semplificando notevolmente la ricerca dei ruoli e dei calciatori associati agli stessi.
Infine, abbiamo effettuato alcune modifiche ai valori delle colonne per adattarli meglio all'interazione con il chatbot. Ad esempio, abbiamo tradotto i valori della colonna "preferred_foot" da "Right" e "Left" a "Destro" e "Sinistro", rispettivamente, per garantire una maggiore chiarezza nella comunicazione con gli utenti italiani.

RASA

Struttura

Dopo aver creato un ambiente virtuale tramite Conda e una directory dedicata allo sviluppo del progetto, è possibile inizializzare il progetto Rasa utilizzando il comando rasa init. Questo comando genera la struttura base del progetto, includendo tutti i componenti necessari per lo sviluppo di un chatbot.



- **actions.py**: Uno script Python che contiene le definizioni di tutte le classi corrispondenti alle azioni specificate nel file domain.yml. Queste classi permettono di generare risposte basate su elaborazioni complesse, anziché risposte statiche predefinite.
- **nlu.yml**: Un file che definisce gli esempi utilizzati per l'addestramento degli intenti e delle entità.
- **rules.yml**: Un file che contiene regole predefinite per le policy di dialogo.
- **stories.yml**: Un file che include esempi di conversazioni, chiamate "path", che il chatbot seguirà.
- **config.yml**: Un file che contiene la configurazione del modello di machine learning.
- **domain.yml**: Un file che raccoglie tutte le informazioni necessarie per il funzionamento del chatbot. In questo file sono elencati tutti gli intenti che possono essere espressi durante una conversazione, insieme alle entità, gli slot per memorizzare i dati necessari all'elaborazione, le risposte predefinite (utterances), l'elenco di tutte le azioni e le definizioni delle form.

Il framework si compone principalmente di due parti: Rasa NLU (Natural Language Understanding) per l'interpretazione dei messaggi degli utenti, e Rasa Core per gestire il flusso di conversazione e decidere cosa fare successivamente in base al contesto. Grazie alla sua architettura modulare e alla possibilità di integrarsi con diverse piattaforme di messaggistica come Telegram, Slack e altri, Rasa è ideale per lo sviluppo di chatbot personalizzati come il chatbot FIFA, progettato per fornire informazioni specifiche e assistenza relativa al mondo del videogioco FIFA, migliorando l'esperienza utente attraverso interazioni naturali e intuitive.

Rasa NLU

Rasa NLU (Natural Language Understanding) è la componente del framework Rasa responsabile dell'interpretazione dei messaggi degli utenti. Utilizza modelli di machine learning per estrarre intenzioni (intents) e entità (entities) dai testi in input. Le intenzioni rappresentano l'obiettivo dell'utente, come ad esempio la richiesta di informazioni o l'esecuzione di un'azione, mentre le entità sono elementi specifici all'interno del messaggio che possono essere utili per comprendere il contesto o per eseguire azioni, come nomi di luoghi, date o numeri.

Per addestrare Rasa NLU, è necessario fornire un insieme di esempi di messaggi degli utenti annotati con le corrispondenti intenzioni e entità. Una volta addestrato, il modello sarà in grado di analizzare nuovi messaggi e identificare le intenzioni e le entità con una certa accuratezza.

Rasa CORE

Rasa Core è la componente del framework Rasa che gestisce il flusso della conversazione. Si basa su un modello di machine learning che prende decisioni su cosa fare successivamente in base al contesto attuale della conversazione. Questo contesto include le intenzioni e le entità estratte da Rasa NLU, nonché lo storico delle interazioni precedenti con l'utente.

Rasa Core utilizza storie (stories) per apprendere come gestire le conversazioni. Le storie sono esempi di conversazioni annotate che mostrano come il bot dovrebbe rispondere a determinate sequenze di intenzioni dell'utente. Ad esempio, se l'utente esprime l'intenzione di ottenere informazioni su un videogioco e poi chiede dettagli su una specifica caratteristica, la storia mostrerà come il bot dovrebbe rispondere a queste intenzioni in sequenza.

Insieme, Rasa NLU e Rasa Core offrono un potente strumento per la creazione di chatbot avanzati e personalizzati, come il chatbot FIFA, che può comprendere le richieste degli utenti e gestire le conversazioni in modo naturale e intuitivo.

Chatbot

Il chatbot che abbiamo deciso di sviluppare si occupa di fornire agli appassionati di calcio un accesso facile e intuitivo alle informazioni statistiche relative ai giocatori per FIFA 22. I dati consentono confronti multipli per gli stessi giocatori del videogioco. Il chatbot è stato realizzato solamente per parlare e rispondere in italiano.
Abbiamo posto particolare attenzione su un insieme di informazioni che potrebbero essere di interesse per gli utenti, tenendo conto anche dei dati disponibili nel dataset utilizzato. A partire da questo dataset, abbiamo selezionato solo le informazioni ritenute più rilevanti e significative per le nostre analisi. Questo processo ci ha permesso di concentrare le nostre risorse e l'attenzione solo sulle variabili che sono cruciali per la nostra ricerca, garantendo al contempo una gestione efficiente dei dati e un'analisi mirata e approfondita.
Nello specifico gli utenti potranno:

- Trovare i calciatori di una determinata squadra
- Trovare le statistiche di un calciatore
- Trovare l'immagine di un calciatore
- Confrontare le caratteristiche di due calciatori
- Trovare il miglior calciatore per ruolo/piede/campionato
- Creare un team inserendo modulo/età/nazionalità

Funzioni utili

Inizialmente, abbiamo apportato delle modifiche alla threshold del FallbackClassifier nel file di configurazione (config.yml) per ottimizzare la gestione delle richieste non riconosciute.
Successivamente, è stata implementata una risposta predefinita del chatbot da utilizzare in scenari in cui l'intent dell'utente non viene identificato con chiarezza. In tali casi, il sistema risponderà cortesemente con la seguente frase: "Mi dispiace ma non credo di aver capito. Potresti riformulare la frase?".
Successivamente abbiamo creato le vere e proprie funzioni che verranno descritte nel seguito.

Trovare i calciatori di una determinata squadra

Questa funzionalità permette agli utenti di trovare i calciatori di una determinata squadra di calcio. L'utente può fornire il nome della squadra tramite l'intento find_team_intent, e il chatbot utilizzerà l'azione find_team_action per recuperare le informazioni sui giocatori di quella squadra.

L'intent find_team_intent raccoglie esempi di frasi che l'utente potrebbe utilizzare per chiedere informazioni su una specifica squadra. Questi esempi aiutano il modello di Rasa a riconoscere l'intenzione dell'utente quando esprime il desiderio di conoscere i calciatori di una squadra.

```
- intent: find_team_intent
  examples: |
    - Vorrei informazioni su [Inter](team)
    - Dati su [Lazio](team)
    ...
    - Chi gioca per il [Manchester City](team)?
    - Rosa della squadra del [FC Porto](team)
    - Formazione per l'[AS Monaco](team)
```

La classe ActionFindTeam è responsabile della ricerca dei calciatori di una specifica squadra. La funzione run estrae il nome della squadra dall'entità team rilevata nel messaggio dell'utente. Se il nome della squadra è presente, viene effettuata una ricerca nel dataset per trovare i calciatori che appartengono a quella squadra. Il risultato viene formattato e inviato all'utente tramite il dispatcher. Se il nome della squadra non è specificato o non è chiaro, viene inviato un messaggio di errore all'utente.

```
class ActionFindTeam(Action):
    def name(self) -> Text:
```

```
        return "find_team_action"

    def run(self,
            dispatcher: CollectingDispatcher,
            tracker: Tracker,
            domain: DomainDict):

        team = next(tracker.get_latest_entity_values('team'), None)

        if team is not None:
            if len(team) == 0:
                dispatcher.utter_message(text='Esistono più squadre con questo nome, devi essere più specifico. \n Se la squadra ha due nomi, scrivili senza spazi.')
            else:
                finded = find_values(team, search='club_name', dispatcher = dispatcher)
                teams =  df[df['club_name'] == (str(finded))]
                team_infos = teams[columns].to_dict('records')
                response = f"I giocatori della squadra {team} sono:\n" + "\n".join([f"-{info['club_name']} : {info['long_name']}" for info in team_infos])
                dispatcher.utter_message(text=response)
        else:
                dispatcher.utter_message(text='Non ho capito a quale squadra ti riferisci')
```

La rule "Find a player team" definisce la sequenza di azioni per gestire la ricerca dei calciatori di una squadra. Una volta rilevata l'intenzione dell'utente di trovare informazioni su una squadra, viene eseguita l'azione find_team_action per effettuare la ricerca.

```
- rule: Find team players
  steps:
    - intent: find_team_intent
    - action: find_team_action
```

In questa prima casistica è possibile osservare come inserendo il nome corretto, il chatbot restituisca tutti i calciatori appartenenti alla squadra indicata:

```
Il risultato viene mostrato di seguito:

Your input ->  qual 'è la rosa della juventus?
I giocatori della squadra juventus sono:
- Juventus : Paulo Bruno Exequiel Dybala
- Juventus : Wojciech Tomasz Szczęsny
- Juventus : Leonardo Bonucci
- Juventus : Matthijs de Ligt
- Juventus : Federico Chiesa
- Juventus : Juan Guillermo Cuadrado Bello
- Juventus : Dušan Vlahović
- Juventus : Manuel Locatelli
- Juventus : Alex Sandro Lobo Silva
- Juventus : Álvaro Borja Morata Martín
- Juventus : Adrien Rabiot
- Juventus : Denis Lemi Zakaria Lako Lado
- Juventus : Arthur Henrique Ramos de Oliveira Melo
- Juventus : Mattia Perin
- Juventus : Danilo Luiz da Silva
- Juventus : Federico Bernardeschi
- Juventus : Moise Bioty Kean
- Juventus : Weston James Earl McKennie
- Juventus : Mattia De Sciglio
- Juventus : Daniele Rugani
- Juventus : Carlo Pinsoglio
- Juventus : Kaio Jorge Pinto Ramos
- Juventus : Matìas Soulé
- Juventus : Fabio Miretti
- Juventus : Koni De Winter
- Juventus : Nicolò Fagioli
- Juventus : Ferdinando Del Sole
```

```
- Juventus : Luca Zanimacchia
- Juventus : Hans Nicolussi Caviglia
- Juventus : Alessandro Di Pardo
```

Viceversa, se non si specifica correttamente la squadra, il chatbot invita all'utente a riprovare. È, inoltre, importante osservare che nel caso in cui due squadre condividano una parte del nome, venga presa solamente il primo dei risultati con match superiore alla treshold inserita:

```
Your input ->  cosa puoi dirmi sul manch
Mi dispiace ma non credo di aver capito. Potresti riformulare la frase?
Your input ->  cosa puoi dirmi sul manchester?
I giocatori della squadra manchester sono:
- Manchester United : Cristiano Ronaldo dos Santos Aveiro
- Manchester United : David De Gea Quintana
- Manchester United : Bruno Miguel Borges Fernandes
- Manchester United : Paul Pogba
- Manchester United : Raphaël Varane
- Manchester United : Jadon Sancho
- Manchester United : Edinson Roberto Cavani Gómez
- Manchester United : Luke Shaw
- Manchester United : Marcus Rashford
- Manchester United : Harry Maguire
- Manchester United : Scott McTominay
- Manchester United : Frederico Rodrigues de Paula Santos
- Manchester United : Alex Nicolao Telles
- Manchester United : Victor Jörgen Nilsson Lindelöf
- Manchester United : Aaron Wan-Bissaka
- Manchester United : Dean Henderson
- Manchester United : Nemanja Matić
- Manchester United : Jesse Lingard
- Manchester United : José Diogo Dalot Teixeira
- Manchester United : Eric Bertrand Bailly
- Manchester United : Juan Manuel Mata García
- Manchester United : Tom Heaton
- Manchester United : Phil Jones
- Manchester United : Anthony David Junior Elanga
- Manchester United : Hannibal Mejbri
- Manchester United : Björn Bryan Hardley
- Manchester United : Alejandro Garnacho Ferreyra
- Manchester United : Shola Maxwell Shoretire
- Manchester United : Zidane Iqbal
- Manchester United : Charlie Wellens
- Manchester United : Charlie Savage
- Manchester United : D'Mani Mellor
- Manchester United : Will Fish
- Manchester United : Lee Grant
- Manchester United : Martin Šviderský
- Manchester United : Charlie McNeill
- Manchester United : Marc Jurado Gomes
- Manchester United : Noam Fritz Emeran
- Manchester United : Willy Kambwala
- Manchester United : Isak Hansen-Aarøen
- Manchester United : Toby Collyer
- Manchester United : José Meteo Mejía Piedrahita
- Manchester United : Logan Pye
- Manchester United : Sam Mather
- Manchester United : Connor Stanley
- Manchester United : Omari Forson
- Manchester United : Ethan Ennis
- Manchester United : Daniel Gore
- Manchester United : Ondřej Mastný
- Manchester United : Dermot Mee
- Manchester United : Mason Greenwood
- Manchester United : Dillon Hoogewerf
- Manchester United : Andreas Hugo Hoelgebaum Pereira

```

Trovare le statistiche di un calciatore

Questa funzione consente di visualizzare le caratteristiche di un giocatore di FIFA22, fornendo informazioni utili come i suoi possibili ruoli in campo, il nome del club in cui attualmente gioca, il suo overall, il valore di mercato in euro e la sua età. Queste informazioni sono cruciali per valutare il giocatore in termini di prestazioni, adattabilità tattica, valore di mercato e prospettive di carriera nel gioco. Con queste caratteristiche a disposizione, gli utenti possono prendere decisioni strategiche più informate riguardo alla composizione della propria squadra e alle scelte di mercato durante la gestione della loro carriera virtuale nel gioco FIFA22.

```
- rule: find info player
  steps:
    - intent: find_info_player_intent
    - action: player_form
    - active_loop: player_form
    - slot_was_set:
      - requested_slot: player
    - active_loop: null
    - action: utter_submit
    - action: get_player
    - action: action_reset_slot
```

Questa funzione permette agli utenti di ottenere informazioni dettagliate su un giocatore di calcio. Quando un utente esprime il desiderio di conoscere informazioni su un giocatore, il sistema attiva l'intent find_info_player_intent, come ad esempio “parlami di un player” o “informazioni giocatore”. Successivamente, una regola, find info player, guida l'interazione, avviando un modulo di raccolta informazioni tramite l'azione player_form. Qui l'utente fornisce il nome del giocatore di interesse. Se il nome del giocatore non viene trovato nel database, viene inviato un messaggio di errore indicando che il giocatore non è stato trovato. Una volta completato il modulo, le informazioni richieste vengono recuperate dal database dei giocatori attraverso l'azione GetPlayer.

```
class GetPlayer(Action):
    def name(self) -> Text:
        return "get_player"

    def run(
            self,
            dispatcher: CollectingDispatcher,
            tracker: Tracker,
            domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

        player = tracker.get_slot('player')
        players = df[(df['long_name'].str.contains(player, case=False, na=False))]

        if len(players) == 0:
            dispatcher.utter_message("Non ci sono calciatori che rispettano queste condizioni!")
        else:
            one_player = players.iloc[0]
            response = f"Il giocatore con nome {one_player['long_name']} ha le seguenti caratteristiche:\n" + "\n".join([f"- Positions: {one_player['player_positions']} \n- Club name: {one_player['club_name']} \n- Overall: {one_player['overall']} \n- Value in euros: {one_player['value_eur']} \n- Age: {one_player['age']}"])
            dispatcher.utter_message(text=response)
        return []

```

Se il giocatore non è trovato nel database, viene inviato un messaggio di errore specificando che non ci sono giocatori che corrispondono alla richiesta. Infine, l'azione ValidatePlayerForm verifica se il nome del giocatore fornito è valido.

```python
class ValidatePlayerForm(FormValidationAction):
    def name(self) -> Text:
        return "validate_player_form"

    def validate_player(
        self,
        slot_value: Any,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: DomainDict,
    ) -> Dict[Text, Any]:

        print(slot_value)

        finded = find_values(slot_value, search='long_name', dispatcher = dispatcher)
        if len(finded) == 0 or finded is None:
            dispatcher.utter_message("Non ho trovato questo giocatore")
            return {"player": None}
        else:
            return {"player": finded}
```

In caso positivo, le informazioni del giocatore vengono restituite all'utente; altrimenti, viene inviato un messaggio di errore. Infine, si resetta lo stato dei dati raccolti con l'action: action_reset_slot. Successivamente abbiamo creato una story per gestire come l'utente può interrompere il form in qualsiasi momento. Se durante la raccolta dei dati l'utente esprime l'intenzione di fermarsi (ad esempio scrivendo 'stop'), il form viene disattivato, i dati raccolti vengono cancellati, e lo stato viene resettato.

Ecco un esempio di come potremmo presentare un'interazione con la shell di Rasa per illustrare una procedura corretta:

```
Your input ->  parlami di un player
Qual è il nome del giocatore?
Your input ->  locatelli
Ok hai inserito i dati correttamente!
Il giocatore con nome Manuel Locatelli ha le seguenti caratteristiche:
- Ruolo: Centrocampista
- Squadra: Juventus
- Punteggio complessivo: 82
- Valore in euro: 40000000.0
- Età: 23
Ora puoi chiedermi qualcos'altro!
```

Qui invece possiamo vedere come vengono gestiti eventuali errori, dovuti ad un input impreciso dell'utente:

```
Your input ->  informazione giocatore
Qual è il nome del giocatore?
Your input ->  fgbrfgbfrgrbs
Non ho trovato questo giocatore
Qual è il nome del giocatore?
Your input ->  maia
Ok hai inserito i dati correttamente!
Il giocatore con nome Victor Martin Maia Padilha ha le seguenti caratteristiche:
- Ruolo: Difensore
- Squadra: Palmeiras
- Punteggio complessivo: 77
- Valore in euro: 10500000.0
- Età: 25
Ora puoi chiedermi qualcos'altro!
```

Trovare l’immagine di un calciatore

Questa funzione consente di visualizzare l'immagine di un giocatore di calcio. Dopo aver fornito al chatbot il nome del giocatore di interesse, il sistema restituirà l'immagine associata a quel giocatore. Il processo per realizzare questa funzionalità ha coinvolto l'addestramento di un modello di machine learning con diversi esempi di frasi che gli utenti potrebbero scrivere per richiedere l'immagine di un giocatore specifico. Una volta completato questo primo step, il chatbot è in grado di comprendere e rispondere alle richieste degli utenti, fornendo loro l'immagine del giocatore di calcio richiesto.

```yaml
- rule: view image player
  steps:
  - intent: view_image_intent
  - action: player_image_form
  - active_loop: player_image_form
  - slot_was_set:
    - requested_slot: player_image
  - active_loop: null
  - action: utter_submit
  - action: get_player_image
  - action: action_reset_slot
```

Questa funzione consente agli utenti di visualizzare l'immagine di un giocatore di calcio. Quando un utente esprime il desiderio di vedere un'immagine di un giocatore, il sistema attiva l'intent view_image_intent, ad esempio “immagine del giocatore” o “mostrami una foto”. Successivamente, una rule, view image player, guida l'interazione, avviando un modulo di raccolta informazioni tramite l'azione player_image_form.
Qui l'utente fornisce il nome del giocatore di interesse. Se il nome del giocatore non viene trovato nel database, viene inviato un messaggio di errore indicando che il giocatore non è stato trovato. Una volta completato il modulo, l'URL dell'immagine del giocatore richiesto viene recuperato dal database dei giocatori attraverso l'azione GetPlayerImage.

```python
class GetPlayerImage(Action):
    def name(self) -> Text:
        return "get_player_image"

    def run(
            self,
            dispatcher: CollectingDispatcher,
            tracker: Tracker,
            domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

        player = tracker.get_slot('player_image')
        print(player)
        players = df[(df['long_name'].str.contains(player, case=False, na=False))]

        if len(players) == 0:
            dispatcher.utter_message("Non ci sono calciatori che rispettano queste condizioni!")
        else:
            one_player = players.iloc[0]
            response = f"Ecco l'immagine del giocatore con nome {one_player['long_name']}: URL -> {one_player['player_face_url']}\n"
            dispatcher.utter_message(text=response)
        return []
```

Se l'immagine del giocatore non è disponibile nel database, viene inviato un messaggio di errore specificando che non ci sono giocatori che corrispondono alla richiesta.

```python
class ValidatePlayerImageForm(FormValidationAction):
    def name(self) -> Text:
        return "validate_player_image_form"

```

```
    def validate_player_image(
        self,
        slot_value: Any,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: DomainDict,
    ) -> Dict[Text, Any]:

        print(slot_value)

        finded = find_values(slot_value, search='long_name', dispatcher = dispatcher)
        if len(finded) == 0 or finded is None:
            dispatcher.utter_message("Non ho trovato questo giocatore")
            return {"player_image": None}
        else:
            return {"player_image": finded}
```

Infine, l'azione ValidatePlayerImageForm verifica se il nome del giocatore fornito è valido. In caso positivo, l'URL dell'immagine del giocatore viene restituito all'utente; altrimenti, viene inviato un messaggio di errore comunicando che il giocatore non è stato trovato nel database.
Successivamente abbiamo creato una story per gestire come l'utente può interrompere il form in qualsiasi momento. Se durante la raccolta dei dati l'utente esprime l'intenzione di fermarsi (ad esempio scrivendo 'stop'), il form viene disattivato, i dati raccolti vengono cancellati, e lo stato viene resettato.

Ecco un esempio di come potremmo presentare un'interazione con la shell di Rasa, mettendo in evidenza come potrebbe avvenire un'interazione in maniera precisa e accurata:

```
1. Your input ->  mostrami una foto
2. Qual è il nome del giocatore di cui vuoi vedere la foto?
3. Your input ->  benzema
4. Ok hai inserito i dati correttamente!
5. Ecco l'immagine del giocatore con nome Karim Benzema: URL ->
https://cdn.sofifa.net/players/165/153/22_120.png
6. Ora puoi chiedermi qualcos'altro!
```

Ora mostriamo invece un possibile errore dell'input dell'utente:

```
 1. Your input ->  immagine giocatore
 2. Qual è il nome del giocatore di cui vuoi vedere la foto?
 3. Your input ->  qwwecatatc
 4. Non ho trovato questo giocatore
 5. Qual è il nome del giocatore di cui vuoi vedere la foto?
 6. Your input ->  giroud
 7. Ok hai inserito i dati correttamente!
 8. Ecco l'immagine del giocatore con nome Olivier Giroud: URL ->
https://cdn.sofifa.net/players/178/509/22_120.png
 9. Ora puoi chiedermi qualcos'altro!
```

Confrontare le caratteristiche di due calciatori

Questa funzione consente di confrontare le caratteristiche di due giocatori in FIFA22. Gli utenti possono inserire i nomi dei due giocatori di loro interesse e ottenere una valutazione comparativa delle loro abilità, includendo parametri come la velocità (Pace), il tiro (Shooting), il passaggio (Passing), il dribbling (Dribbling), la difesa (Defending) e il fisico (Physical). Tale confronto fornisce una panoramica dettagliata delle prestazioni relative dei giocatori in campo, aiutando gli utenti a prendere decisioni strategiche riguardo alla composizione della propria squadra, alla selezione dei giocatori per determinati ruoli e alle strategie di gioco. Il modello di machine learning è stato

addestrato su un ampio set di dati per comprendere e rispondere a richieste di confronto di giocatori, garantendo così una valutazione accurata e informativa.

```
- rule: compare two players
  steps:
    - intent: compare_players_intent
    - action: compare_players_form
    - active_loop: compare_players_form
    - slot_was_set:
      - requested_slot: player_one
    - slot_was_set:
      - requested_slot: player_two
    - active_loop: null
    - action: utter_submit
    - action: compare_two_players
    - action: action_reset_slot
```

Questa funzione permette agli utenti di mettere a confronto le abilità di due calciatori in FIFA22. Quando un utente desidera confrontare due giocatori, il sistema attiva l'intent compare_players_intent.

```
- intent: compare_players_intent
  examples: |
    - confronta questi due giocatori
    - che giocatori vuoi confrontare?
    - confronta due giocatori di calcio
    - voglio paragonare due calciatori famosi
    - quali sono le differenze tra due grandi giocatori di calcio?
    - confronta le statistiche di due attaccanti
    - che players vuoi confrontare?
    - voglio confrontare due grandi nomi del calcio
    - paragona le abilità di due players di calcio
    - quali sono i punti di forza e di debolezza di due calciatori?
```

Successivamente, una rule, denominata compare two players, guida l'interazione, avviando un processo di selezione tramite l'azione compare_players_form. Qui l'utente fornisce i nomi dei due giocatori da confrontare. Se uno dei nomi forniti non corrisponde a un giocatore nel database, viene emesso un messaggio di avviso indicando che il giocatore non è stato trovato. Una volta completata la selezione, le statistiche dei due giocatori richiesti vengono estratte dal database tramite l'azione CompareTwoPlayers.

```
class CompareTwoPlayers(Action):
    def name(self) -> Text:
        return "compare_two_players"

    def run(
            self,
            dispatcher: CollectingDispatcher,
            tracker: Tracker,
            domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

        player_one = tracker.get_slot('player_one')
        player_two = tracker.get_slot('player_two')
        players_one = df[(df['long_name'].str.contains(player_one, case=False, na=False))]
        players_two = df[(df['long_name'].str.contains(player_two, case=False, na=False))]

        if len(players_one) == 0 or len(players_two) == 0:
            dispatcher.utter_message("Non ci sono calciatori che rispettano queste
condizioni!")
        else:
            one_player = players_one.iloc[0]
            two_player = players_two.iloc[0]
            response = f"Ecco i due giocatori player_one: {one_player['long_name']} e
player_two: {two_player['long_name']}\nPace -> {one_player['pace']} vs
{two_player['pace']}\nShooting -> {one_player['shooting']} vs {two_player['shooting']}\nPassing
```

```
-> {one_player['passing']} vs {two_player['passing']}\nDribbling -> {one_player['dribbling']} vs
{two_player['dribbling']}\nDefending -> {one_player['defending']} vs
{two_player['defending']}\nPhysic -> {one_player['physic']} vs {two_player['physic']}\n"
            dispatcher.utter_message(text=response)
        return []
```

Se uno o entrambi i giocatori non sono presenti nel database, viene inviato un messaggio di avviso specificando che non ci sono giocatori corrispondenti alla richiesta.

```
class ValidateComparePlayersForm(FormValidationAction):
    def name(self) -> Text:
        return "validate_compare_players_form"

    def validate_player_one(
        self,
        slot_value: Any,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: DomainDict,
    ) -> Dict[Text, Any]:

        print(slot_value)
        if slot_value == 'prosegui':
            return {"player_one": ''}
        else:
            finded = find_values(slot_value, search='long_name', dispatcher = dispatcher)
            if len(finded) == 0:
                dispatcher.utter_message("Non ho trovato questo giocatore")
                return {"player_one": None}
            else:
                return {"player_one": finded}

    def validate_player_two(
        self,
        slot_value: Any,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: DomainDict,
    ) -> Dict[Text, Any]:

        print(slot_value)
        if slot_value == 'prosegui':
            return {"player_two": ''}
        else:
            finded = find_values(slot_value, search='long_name', dispatcher = dispatcher)
            if len(finded) == 0:
                dispatcher.utter_message("Non ho trovato questo giocatore")
                return {"player_two": None}
            else:
                return {"player_two": finded}
```

Infine, l'azione ValidateComparePlayersForm verifica la validità dei nomi dei giocatori forniti. Se i nomi sono validi, le statistiche dei due giocatori vengono restituite per il confronto; altrimenti, viene emesso un messaggio di avviso comunicando che uno o entrambi i giocatori non sono stati trovati nel database.
Successivamente abbiamo creato una story per gestire come l'utente può interrompere il form in qualsiasi momento. Se durante la raccolta dei dati l'utente esprime l'intenzione di fermarsi (ad esempio scrivendo ‘stop’), il form viene disattivato, i dati raccolti vengono cancellati, e lo stato viene resettato.

Ecco un esempio di come potremmo dimostrare un'interazione con la shell di Rasa per illustrare come potrebbe avvenire un'interazione corretta:

```
Your input ->  confronta due giocatori
```

```
 2. Qual è il nome del primo giocatore?
 3. Your input ->  Messi
 4. Qual è il nome del secondo giocatore?
 5. Your input ->  Mbappe
 6. Ok hai inserito i dati correttamente!
 7. Ecco i due giocatori player_one: Lionel Andrés Messi Cuccittini e player_two: Kylian Mbappé
Lottin
 8. Velocità -> 83.0 vs 97.0
 9. Tiro -> 90.0 vs 88.0
10. Passaggio -> 91.0 vs 80.0
11. Dribbling -> 95.0 vs 92.0
12. Difesa -> 34.0 vs 36.0
13. Fisico -> 64.0 vs 77.0
14. Ora puoi chiedermi qualcos'altro!
```

Qui ne facciamo vedere una in cui l'interazione non avviene completamente corretta:

```
 1. Your input ->  confronta abilità giocatori
 2. Qual è il nome del primo giocatore?
 3. Your input ->  oefngvnerfkve
 4. Non ho trovato questo giocatore
 5. Qual è il nome del primo giocatore?
 6. Your input ->  popoerikefef
 7. Non ho trovato questo giocatore
 8. Qual è il nome del primo giocatore?
 9. Your input ->  ronaldo
10. Qual è il nome del secondo giocatore?
11. Your input ->  pofjvisndvssssss
12. Non ho trovato questo giocatore
13. Qual è il nome del secondo giocatore?
14. Your input ->  kane
15. Ok hai inserito i dati correttamente!
16. Ecco i due giocatori player_one: Cristiano Ronaldo dos Santos Aveiro e player_two: Harry
Kane
17. Velocità -> 85.0 vs 68.0
18. Tiro -> 93.0 vs 92.0
19. Passaggio -> 80.0 vs 83.0
20. Dribbling -> 86.0 vs 82.0
21. Difesa -> 34.0 vs 47.0
22. Fisico -> 75.0 vs 83.0
23. Ora puoi chiedermi qualcos'altro!
```

Trovare il miglior calciatore per ruolo/piede/campionato

Questa funzionalità permette agli utenti di cercare il miglior giocatore di calcio in base al ruolo, al piede preferito e al campionato in cui gioca. Il processo è guidato da un form che raccoglie queste tre informazioni dall'utente attraverso una serie di domande interattive.
In primo luogo, abbiamo definito gli intent necessari per poter lanciare la rule che attiva l'effettiva form. Successivamente abbiamo definito la rule che descrive il comportamento che il chatbot deve seguire:

```
 1. - rule: Activate form find bplayer
 2.   steps:
 3.     - intent: find_best_player
 4.     - action: bplayer_form
 5.     - active_loop: bplayer_form
 6.     - slot_was_set:
 7.         - requested_slot: role
 8.     - slot_was_set:
 9.         - requested_slot: league
10.     - slot_was_set:
11.         - requested_slot: preferred_foot
12.     - active_loop: null
13.     - action: utter_submit
```

```
14.     - action: get_bplayer
15.     - action: action_reset_slot
```

Una volta rilevata l'intenzione dell'utente di trovare il miglior giocatore (intent: find_best_player, come ad esempio ‘Cerca il miglior giocatore’ oppure ‘Dimmi qual è il calciatore più forte’), il form bplayer_form verrà attivato. Questo form rimane attivo finché non ha raccolto tutti i dati necessari: ruolo, campionato e piede preferito. Una volta raccolte queste informazioni, il form viene chiuso, viene inviato un messaggio di conferma e si procede con l'azione get_bplayer per recuperare le informazioni sul giocatore. Infine, si resetta lo stato dei dati raccolti.
Successivamente abbiamo creato una story per gestire come l'utente può interrompere il form in qualsiasi momento. Se durante la raccolta dei dati l'utente esprime l'intenzione di fermarsi (ad esempio scrivendo ‘stop’), il form viene disattivato, i dati raccolti vengono cancellati, e lo stato viene resettato.
La classe GetBPlayer implementa la logica per cercare il miglior giocatore in base ai criteri specificati dall'utente. Recupera i valori dagli slot del form e utilizza questi per filtrare un dataset dei giocatori, selezionando quello con il rating più alto che corrisponde ai criteri.

```
 1. class GetBPlayer(Action):
 2.     def name(self) -> Text:
 3.         return "get_bplayer"
 4.
 5.     def run(
 6.             self,
 7.             dispatcher: CollectingDispatcher,
 8.             tracker: Tracker,
 9.             domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
10.
11.         league = tracker.get_slot('league')
12.         role = tracker.get_slot('role')
13.         preferred_foot = tracker.get_slot('preferred_foot')
14.         best_players = df[
15.             (df['league_name'].str.contains(league, case=False, na=False)) &
16.             (df['player_positions'].str.contains(role, case=False, na=False)) &
17.             (df['preferred_foot'].str.contains(preferred_foot, case=False, na=False))
18.         ].sort_values(by='overall', ascending=False)
19.
20.         if len(best_players) == 0:
21.             dispatcher.utter_message("Non ci sono calciatori che rispettano queste
condizioni!")
22.         else:
23.             best_player = best_players.iloc[0]
24.             response = (f"Il miglior giocatore è: {best_player['short_name']}, "
25.                         f"che gioca in {best_player['league_name']} come
{best_player['player_positions']},"
26.                         f"con piede preferito {best_player['preferred_foot']} e overall
{best_player['overall']}.")
27.             dispatcher.utter_message(text=response)
28.         return []
```

La classe ValidateSearchBPlayerForm è responsabile della validazione dei valori inseriti dall'utente in ogni step del form. Se l'utente decide di interrompere la ricerca (stop_intent), i dati vengono resettati. Altrimenti, verifica che i dati inseriti corrispondano a valori validi nel dataset e, in caso di dati non validi, informa l'utente e richiede di inserire nuovamente i dati.

```
 1. class ValidateSearchBPlayerForm(FormValidationAction):
 2.     def name(self) -> Text:
 3.         return "validate_bplayer_form"
 4.
 5.     def validate_role(
 6.         self,
 7.         slot_value: Any,
 8.         dispatcher: CollectingDispatcher,
 9.         tracker: Tracker,
```

```
        domain: DomainDict,
    ) -> Dict[Text, Any]:
        if tracker.latest_message["intent"]["name"] == "stop_intent":
            print(tracker.latest_message["intent"]["name"])
            # dispatcher.utter_message(text="Hai interroto la ricerca!")
            return [AllSlotsReset(), Restarted()]
        print(slot_value)
        if slot_value == 'prosegui' or slot_value == '':
            return {"role": ''}
        else:
            finded = find_values(slot_value, search='player_positions', dispatcher =
dispatcher)
            if len(finded) == 0:
                dispatcher.utter_message("Non ho trovato questo ruolo")
                return {"role": None}
            else:
                return {"role": finded}

    def validate_league(
        self,
        slot_value: Any,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: DomainDict,
    ) -> Dict[Text, Any]:
        print(tracker.latest_message["intent"]["name"])
        if tracker.latest_message["intent"]["name"] == "stop_intent":
            # dispatcher.utter_message(text="Hai interroto la ricerca!")
            return [AllSlotsReset(), Restarted()]
        print(slot_value)
        if slot_value == 'prosegui' or slot_value == '':
            return {"league": ''}
        else:
            finded = find_values(slot_value, search='league_name', dispatcher = dispatcher)
            if len(finded) == 0:
                dispatcher.utter_message("Non ho trovato una lega con questo nome")
                return {"league": None}
            else:
                return {"league": finded}

    def validate_preferred_foot(
        self,
        slot_value: Any,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: DomainDict,
    ) -> Dict[Text, Any]:
        print(tracker.latest_message["intent"]["name"])
        if tracker.latest_message["intent"]["name"] == "stop_intent":
            # dispatcher.utter_message(text="Hai interroto la ricerca!")
            return [AllSlotsReset(), Restarted()]
        print(slot_value)
        if slot_value == 'prosegui' or slot_value == '':
            return {"preferred_foot": ''}
        else:
            finded = find_values(slot_value, search='preferred_foot', dispatcher =
dispatcher)
            if len(finded) == 0:
                dispatcher.utter_message("Inserire Destro o Sinistro")
                return {"preferred_foot": None}
            else:
                {"preferred_foot": finded}
                return [FollowupAction("get_bplayer")]
```

Mostriamo ora un'interazione con Rasa Shell per far capire come potrebbe avvenire un'interazione in maniera del tutto corretto:

```
1. Your input ->  Chi è il miglior giocatore di sempre?
```

```
 2. Qual è il ruolo del calciatore? - Digita prosegui se non hai preferenze
 3. Your input ->  Attaccante
 4. In quale campionato gioca? - Digita prosegui se non hai preferenze
 5. Your input ->  Serie A
 6. Qual è il piede preferito del calciatore? - Digita prosegui se non hai preferenze
 7. Your input ->  Destro
 8. Ok hai inserito i dati correttamente!
 9. Il miglior giocatore è: C. Immobile, che gioca in Serie A come Attaccante,con piede
preferito Destro e overall 86.
10. Ora puoi chiedermi qualcos'altro!
```

E ne mostriamo ora una in cui invece non avviene in maniera completamente corretta:

```
 1. Your input ->  chi è il più forte
 2. Qual è il ruolo del calciatore? - Digita prosegui se non hai preferenze
 3. Your input -> prosegui
 4. In quale campionato gioca? - Digita prosegui se non hai preferenze
 5. Your input ->  prema
 6. Non ho trovato una lega con questo nome
 7. In quale campionato gioca? - Digita prosegui se non hai preferenze
 8. Your input ->  premie leg
 9. Qual è il piede preferito del calciatore? - Digita prosegui se non hai preferenze
10. Your input ->  des
11. Ok hai inserito i dati correttamente!
12. Il miglior giocatore è: Cristiano Ronaldo, che gioca in Premier League come Attaccante,con
piede preferito Destro e overall 91.
13. Ora puoi chiedermi qualcos'altro!
```

Creare un team inserendo modulo/età/nazionalità

Questa funzionalità permette agli utenti di creare una squadra di calcio in base a specifici criteri come modulo, età media, nazionalità e caratteristiche dei giocatori. Il processo è guidato da un form che raccoglie queste informazioni attraverso una serie di domande interattive.
La rule "Activate form create team" definisce la sequenza di azioni per gestire la creazione della squadra. Una volta rilevata l'intent dell'utente di creare una squadra (come 'Crea una squadra'), il form create_team_form viene attivato. Questo form rimane attivo finché non ha raccolto tutti i dati necessari: modulo, età media, nazionalità e caratteristica. Una volta raccolte queste informazioni, il form viene chiuso, viene inviato un messaggio di conferma e si procede con l'azione get_team per creare la squadra. Infine, si resetta lo stato dei dati raccolti.

```
 1. - rule: Activate form create team
 2.   steps:
 3.     - intent: create_team_intent
 4.     - action: create_team_form
 5.     - active_loop: create_team_form
 6.     - slot_was_set:
 7.         - requested_slot: module
 8.     - slot_was_set:
 9.         - requested_slot: age
10.     - slot_was_set:
11.         - requested_slot: nationality
12.     - slot_was_set:
13.         - requested_slot: characteristic
14.     - active_loop: null
15.     - action: utter_submit
16.     - action: get_team
17.     - action: action_reset_slot
```

Nelle stories abbiamo definito come l'utente può interrompere il form in qualsiasi momento. Se durante la raccolta dei dati l'utente esprime l'intenzione di fermarsi, il form viene disattivato i dati raccolti vengono cancellati e lo stato viene resettato.

Per quanto riguarda le azioni, è stata creata la classe GetTeam la quale implementa la logica per creare la squadra in base ai criteri specificati dall'utente. Recupera i valori dai slot del form e utilizza questi per selezionare i giocatori che corrispondono ai criteri, organizzandoli nel modulo richiesto.

```
class GetTeam(Action):
    def name(self) -> Text:
        return "get_team"

    def run(
            self,
            dispatcher: CollectingDispatcher,
            tracker: Tracker,
            domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
        module = tracker.get_slot('module')
        age = tracker.get_slot('age')
        age = int(age)
        nationality = tracker.get_slot('nationality')
        characteristic = tracker.get_slot('characteristic')
        if modules[0] in module:
            goalkeepers = find_team_players(player_positions = 'Portiere', age = age,
nationality = nationality, characteristic = characteristic, head = 1)
            defenders = find_team_players(player_positions = 'Difensore', age = age,
nationality = nationality, characteristic = characteristic, head = 4)
            midfielders = find_team_players(player_positions = 'Centrocampista', age = age,
nationality = nationality, characteristic = characteristic, head = 3)
            strikers = find_team_players(player_positions = 'Attaccante', age = age,
nationality = nationality, characteristic = characteristic, head = 3)
            if len(goalkeepers) == 0 or len(defenders) < 4 or len(midfielders) < 3 or
len(strikers) < 3:
                dispatcher.utter_message("Non è possibile creare una squadra rispettando
queste condizioni!")
            else:

                response = f"Portiere: \n"+
f"\n".join([f"{goalkeeper['long_name']}:{goalkeeper['age']}, {goalkeeper['nationality_name']},
{characteristic} : {goalkeeper[characteristic]}" for goalkeeper in goalkeepers])
                response += "\nDifensori: \n" +
f"\n".join([f"{defender['long_name']}:{defender['age']}, {defender['nationality_name']},
{characteristic} : {defender[characteristic]}" for defender in defenders])
                response += "\nCentrocampisti: \n" +
f"\n".join([f"{midfielder['long_name']}:{midfielder['age']}, {midfielder['nationality_name']},
{characteristic} : {midfielder[characteristic]}" for midfielder in midfielders])
                response += "\nAttaccanti: \n" +
f"\n".join([f"{striker['long_name']}:{striker['age']}, {striker['nationality_name']},
{characteristic} : {striker[characteristic]}" for striker in strikers])
                dispatcher.utter_message(text=response)

        elif module == modules[1]:

            goalkeepers = find_team_players(player_positions = 'Portiere', age = age,
nationality = nationality, characteristic = characteristic, head = 1)
            defenders = find_team_players(player_positions = 'Difensore', age = age,
nationality = nationality, characteristic = characteristic, head = 4)
            midfielders = find_team_players(player_positions = 'Centrocampista', age = age,
nationality = nationality, characteristic = characteristic, head = 4)
            strikers = find_team_players(player_positions = 'Attaccante', age = age,
nationality = nationality, characteristic = characteristic, head = 2)
            if len(goalkeepers) == 0 or len(defenders) < 4 or len(midfielders) < 4 or
len(strikers) < 2:
                dispatcher.utter_message("Non è possibile creare una squadra rispettando
queste condizioni!")
            else:

                response = f"Portiere: \n"+
f"\n".join([f"{goalkeeper['long_name']}:{goalkeeper['age']}, {goalkeeper['nationality_name']},
{characteristic} : {goalkeeper[characteristic]}" for goalkeeper in goalkeepers])
```

```
                response += "\nDifensori: \n" +
f"\n".join([f"{defender['long_name']}:{defender['age']}, {defender['nationality_name']},
{characteristic} : {defender[characteristic]}" for defender in defenders])
                response += "\nCentrocampisti: \n" +
f"\n".join([f"{midfielder['long_name']}:{midfielder['age']}, {midfielder['nationality_name']},
{characteristic} : {midfielder[characteristic]}" for midfielder in midfielders])
                response += "\nAttaccanti: \n" +
f"\n".join([f"{striker['long_name']}:{striker['age']}, {striker['nationality_name']},
{characteristic} : {striker[characteristic]}" for striker in strikers])
                dispatcher.utter_message(text=response)
        elif module == modules[2]:

            goalkeepers = find_team_players(player_positions = 'Portiere', age = age,
nationality = nationality, characteristic = characteristic, head = 1)
            defenders = find_team_players(player_positions = 'Difensore', age = age,
nationality = nationality, characteristic = characteristic, head = 3)
            midfielders = find_team_players(player_positions = 'Centrocampista', age = age,
nationality = nationality, characteristic = characteristic, head = 5)
            strikers = find_team_players(player_positions = 'Attaccante', age = age,
nationality = nationality, characteristic = characteristic, head = 2)
            if len(goalkeepers) == 0 or len(defenders) < 3 or len(midfielders) < 5 or
len(strikers) < 2:
                dispatcher.utter_message("Non è possibile creare una squadra rispettando
queste condizioni!")
            else:

                response = f"\nPortiere: \n"+
f"\n".join([f"{goalkeeper['long_name']}:{goalkeeper['age']}, {goalkeeper['nationality_name']},
{characteristic} : {goalkeeper[characteristic]}" for goalkeeper in goalkeepers])
                response += "\nDifensori: \n" +
f"\n".join([f"{defender['long_name']}:{defender['age']}, {defender['nationality_name']},
{characteristic} : {defender[characteristic]}" for defender in defenders])
                response += "\nCentrocampisti: \n" +
f"\n".join([f"{midfielder['long_name']}:{midfielder['age']}, {midfielder['nationality_name']},
{characteristic} : {midfielder[characteristic]}" for midfielder in midfielders])
                response += "\nAttaccanti: \n" +
f"\n".join([f"{striker['long_name']}:{striker['age']}, {striker['nationality_name']},
{characteristic} : {striker[characteristic]}" for striker in strikers])
                dispatcher.utter_message(text=response)
        return []
```

La classe ValidateCreateTeamForm è responsabile della validazione dei valori inseriti dall'utente in ogni step del form. Se l'utente decide di interrompere la creazione della squadra (stop_intent), i dati vengono resettati. Altrimenti, verifica che i dati inseriti corrispondano a valori validi e, in caso di dati non validi, informa l'utente e richiede di inserire nuovamente i dati.

```
class ValidateCreateTeamForm(FormValidationAction):
    def name(self) -> Text:
        return "validate_create_team_form"

    def validate_module(
        self,
        slot_value: Any,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: DomainDict,
    ) -> Dict[Text, Any]:


        if slot_value not in modules:
            dispatcher.utter_message(text = f"Devi inserire un modulo valido!")
            return {"module": None}
        else:
            return {"module": slot_value}

    def validate_age(
        self,
```

```
        slot_value: Any,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: DomainDict,
    ) -> Dict[Text, Any]:

        if slot_value == 'prosegui' or slot_value == '':
            return {"age": df['age'].max()}
        elif not slot_value.isdigit():
            dispatcher.utter_message("Inserisci un valore numerico")
            return {"age": None}
        else:
            return {"age": slot_value}

    def validate_nationality(
        self,
        slot_value: Any,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: DomainDict,
    ) -> Dict[Text, Any]:

        if slot_value == 'prosegui' or slot_value == '':
            return {"nationality": ''}
        else:
            finded = find_values(slot_value, search='nationality_name', dispatcher =
dispatcher)
            if len(finded) == 0:
                dispatcher.utter_message("Non ho trovato questa nazionalità")
                return {"nationality": None}
            else:
                return {"nationality": finded}

    def validate_characteristic(
        self,
        slot_value: Any,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: DomainDict,
    ) -> Dict[Text, Any]:

        if slot_value == 'prosegui' or slot_value == '':
            random_value = random.choice(char)
            return {"characteristic": random_value}
        else:
            finded = process.extract(str(slot_value), char, limit= 1)
            if finded[0][1] > 80:
                return {"characteristic": finded[0][0]}
            else:
                dispatcher.utter_message("Non ho capito, devi inserire \'Overall\' oppure
\'Potential\'")
                return {"characteristic": None}

```

A titolo di esempio, mostriamo l'interazione con la shell, prima in una casistica in cui tutto va a buon fine:

```

Your input ->  Ciao!
Ehila!
Your input ->  Crea una squadra!
? Quale sarà il modulo della squadra? 2: 4-4-2 (/ask_module{"module":"4-4-2"})
Inserisci l'età massima. - Premi invio o inserisci prosegui se non hai preferenze
Your input ->  27
Inserisci la nazionalità. - Premi invio o inserisci prosegui se non hai preferenze
Your input ->  Argentina
Inserisci una caratteristica (Overall oppure Potential). - Premi invio o inserisci prosegui
se non hai preferenze
```

```
Your input ->  Overall
Ok hai inserito i dati correttamente!
Portiere:
Juan Agustín Musso:27, Argentina, overall : 81
Difensori:
Cristian Gabriel Romero:23, Argentina, overall : 82
Gonzalo Ariel Montiel:24, Argentina, overall : 79
Marcos Nicolás Senesi Barón:24, Argentina, overall : 78
Juan Marcos Foyth:23, Argentina, overall : 77
Centrocampisti:
Rodrigo Javier De Paul:27, Argentina, overall : 83
Giovani Lo Celso:25, Argentina, overall : 80
Leandro Daniel Paredes:27, Argentina, overall : 80
Emiliano Buendía:24, Argentina, overall : 79
Attaccanti:
Paulo Bruno Exequiel Dybala:27, Argentina, overall : 87
Lautaro Javier Martínez:23, Argentina, overall : 85
Ora puoi chiedermi qualcos'altro!
```

Mentre ora mostriamo un esempio di story andata male:

```
Your input ->  squadra crea
? Quale sarà il modulo della squadra? 3: 3-5-2 (/ask_module{"module":"3-5-2"})
Inserisci l'età massima. - Premi invio o inserisci prosegui se non hai preferenze
Your input ->  trenta
Inserisci un valore numerico
Inserisci l'età massima. – Digita prosegui se non hai preferenze
Your input ->  30
Inserisci la nazionalità. - Digita prosegui se non hai preferenze
Your input ->
Inserisci una caratteristica (Overall oppure Potential). - Digita prosegui se non hai
preferenze
Your input ->  stop
Ora puoi chiedermi qualcos'altro!

```

Telegram

L'ultima fase di sviluppo del nostro progetto prevede l'integrazione del chatbot con una piattaforma di messaggistica istantanea. Questo permetterà agli utenti di interagire con il chatbot attraverso dispositivi mobili, beneficiando di un'interfaccia grafica moderna e intuitiva. Per questo scopo, abbiamo selezionato Telegram come piattaforma di riferimento.

Per realizzare l'integrazione, abbiamo utilizzato il servizio ngrok, che consente di stabilire una connessione sicura dal nostro server locale alla piattaforma Telegram. I dettagli di questa configurazione sono stati definiti nel file credentials.yml, necessario per il deployment del chatbot.

```
1. telegram:
2.   access_token: 7039441013:AAFLoD1pmkhos_6vLobX7fzeebPd-jBBEd4
3.   verify: fifa22chat_bot
4.   webhook_url: https://bb3d-45-11-80-147.ngrok-free.app/webhooks/telegram/webhook
```

Le configurazioni specifiche includono:

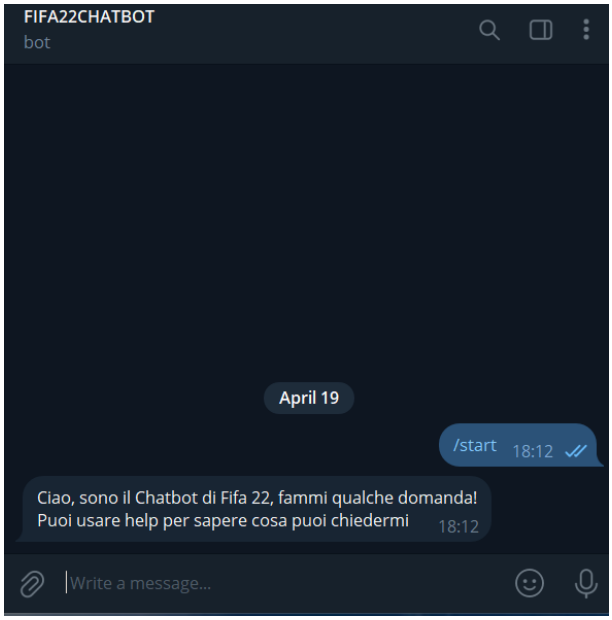
- **access_token**: Il token ottenuto seguendo la procedura guidata di Telegram tramite il bot BotFather;
- **verify**: Nome assegnato al bot durante la fase di configurazione su Telegram;
- **webhook_url**: L'URL di callback utilizzato per abilitare la comunicazione tra il server locale e il chatbot via ngrok.

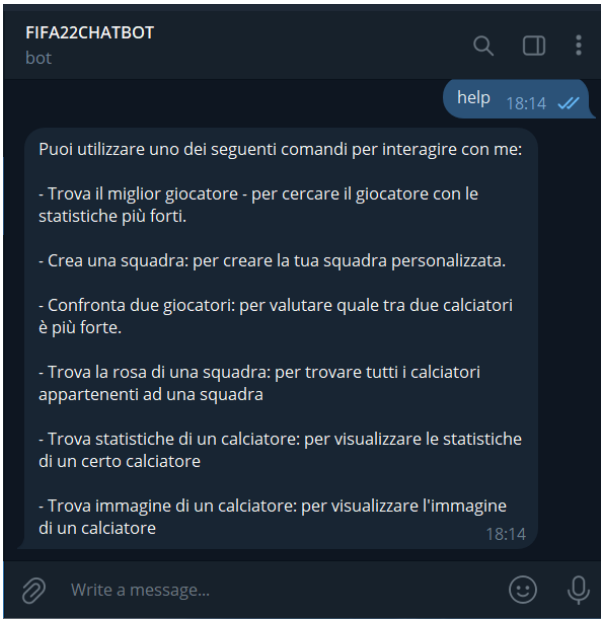
Dopo aver completato l'implementazione e la connessione del chatbot alla piattaforma di messaggistica, abbiamo proceduto con la fase di testing. Questo ha incluso la simulazione di interazioni tipiche degli utenti con il chatbot, così come scenari con input errati per testare la robustezza dei controlli di validazione implementati. Questo processo ci ha permesso di assicurare il corretto funzionamento del chatbot in vari contesti d'uso.

Nei paragrafi seguenti mostreremo le interazioni descritte in precedenza.

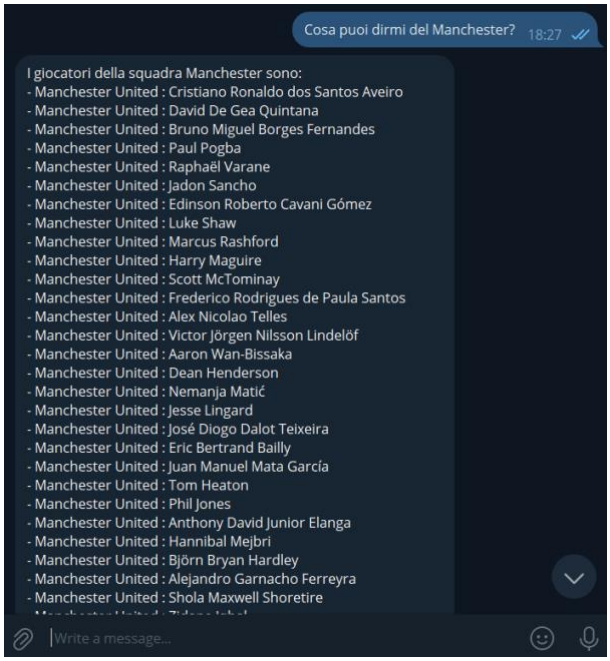
Avvio del bot

In primo luogo, abbiamo avviato il bot:

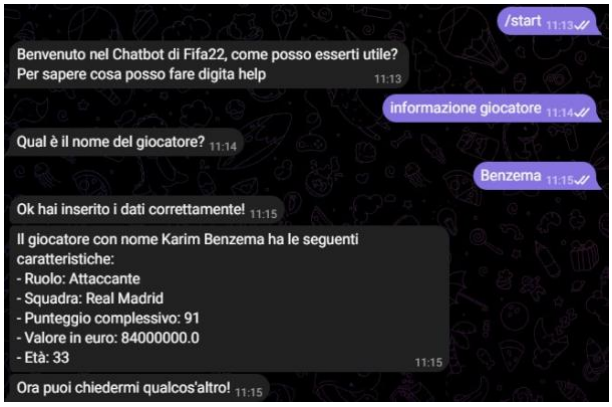


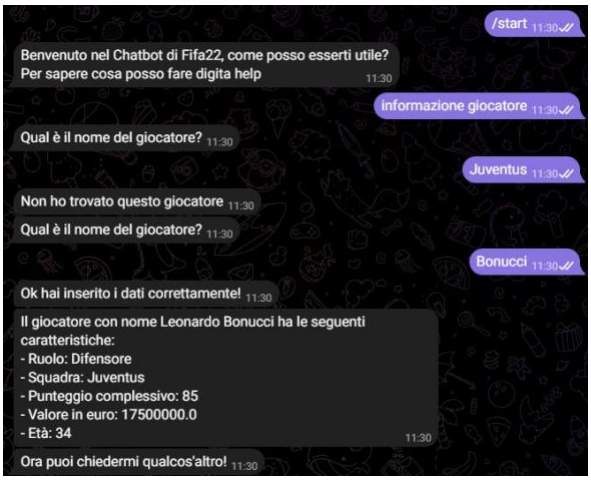


Trovare i calciatori di una determinata squadra

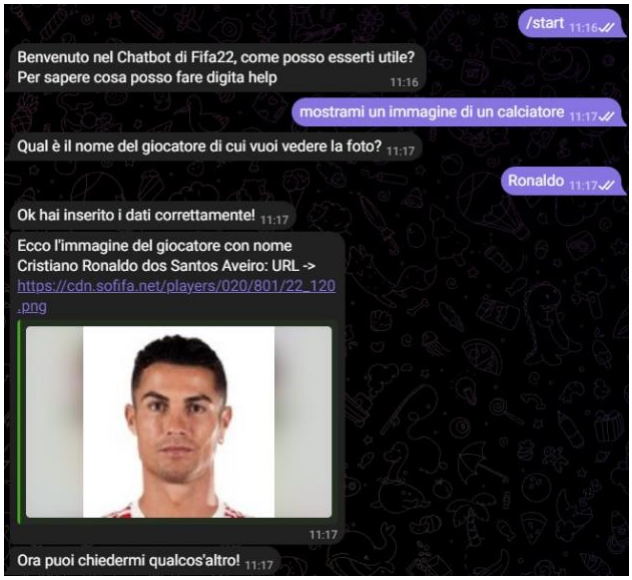


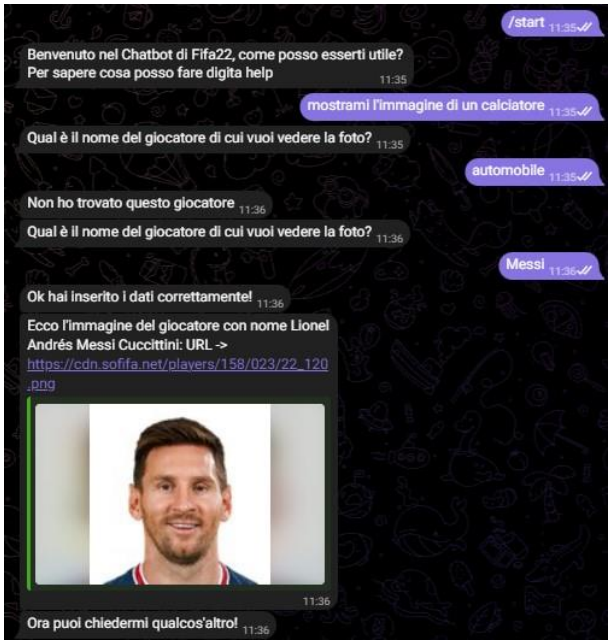
Trovare le statistiche di un calciatore



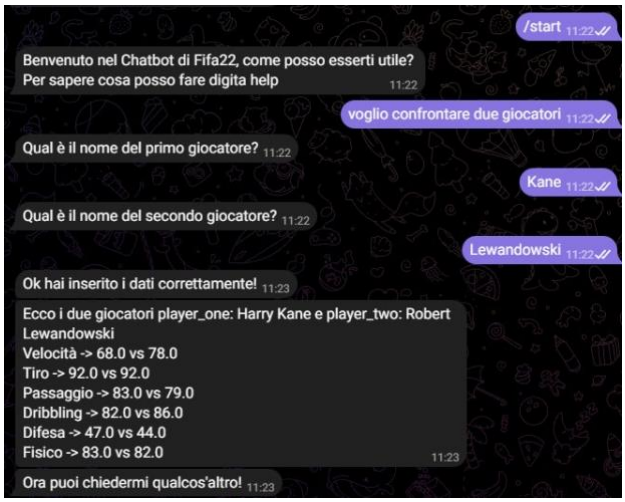


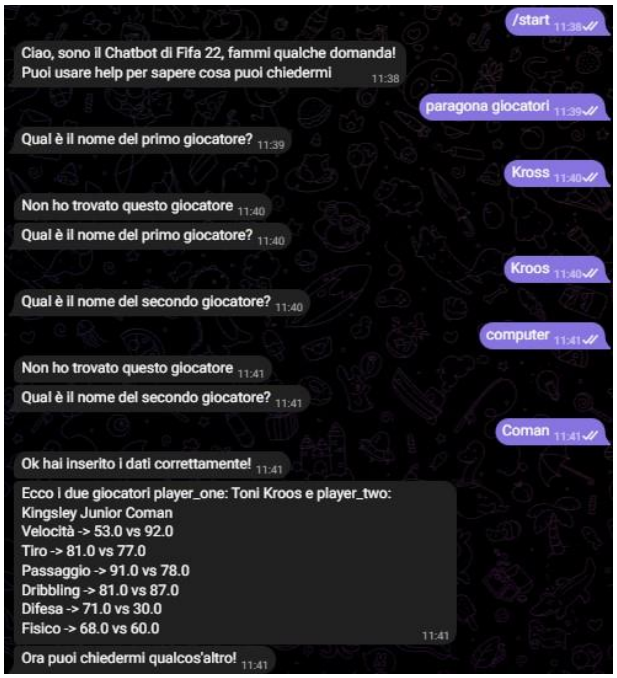
Trovare l'immagine di un calciatore



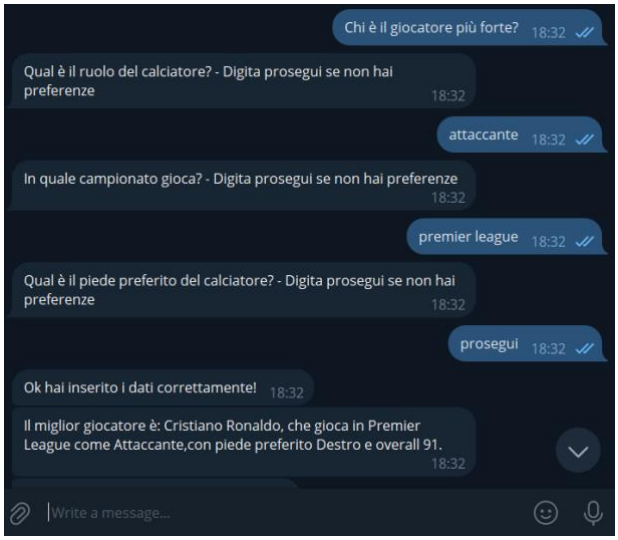


Confrontare le caratteristiche di due calciatori

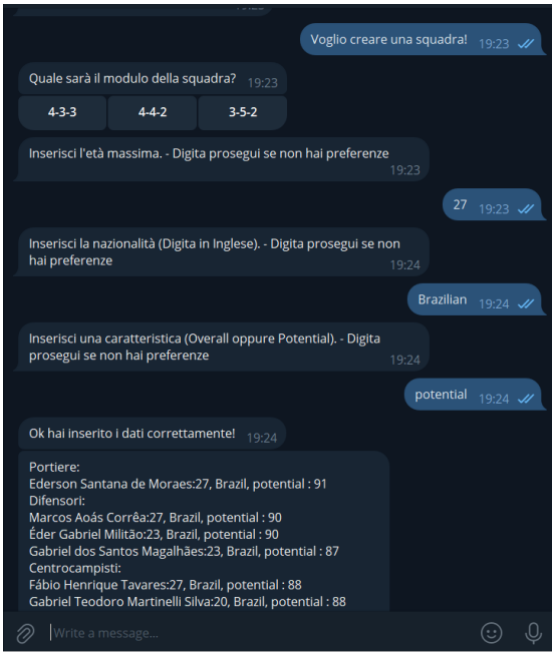


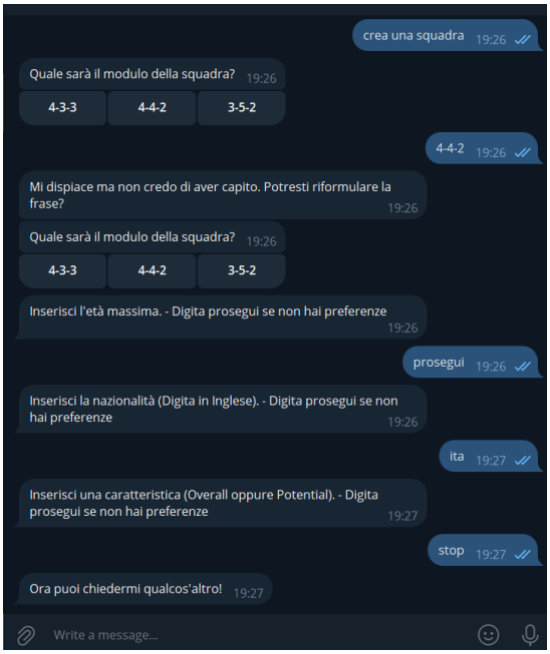


Trovare il miglior calciatore per ruolo/piede/campionato



Creare un team inserendo modulo/età/nazionalità


Voglio creare una squadra! 19:23
Quale sarà il modulo della squadra? 19:23
4-3-3
4-4-2
3-5-2
Inserisci l'età massima. - Digita prosegui se non hai preferenze 19:23
27 19:23
Inserisci la nazionalità (Digita in Inglese). - Digita prosegui se non hai preferenze 19:24
Brazilian 19:24
Inserisci una caratteristica (Overall oppure Potential). - Digita prosegui se non hai preferenze 19:24
potential 19:24
Ok hai inserito i dati correttamente! 19:24
Portiere:
Ederson Santana de Moraes:27, Brazil, potential : 91
Difensori:
Marcos Aoás Corrêa:27, Brazil, potential : 90
Éder Gabriel Militão:23, Brazil, potential : 90
Gabriel dos Santos Magalhães:23, Brazil, potential : 87
Centrocampisti:
Fábio Henrique Tavares:27, Brazil, potential : 88
Gabriel Teodoro Martinelli Silva:20, Brazil, potential : 88
Write a message...


crea una squadra 19:26
Quale sarà il modulo della squadra? 19:26
4-3-3
4-4-2
3-5-2
4-4-2 19:26
Mi dispiace ma non credo di aver capito. Potresti riformulare la frase? 19:26
Quale sarà il modulo della squadra? 19:26
4-3-3
4-4-2
3-5-2
Inserisci l'età massima. - Digita prosegui se non hai preferenze 19:26
prosegui 19:26
Inserisci la nazionalità (Digita in Inglese). - Digita prosegui se non hai preferenze 19:26
ita 19:27
Inserisci una caratteristica (Overall oppure Potential). - Digita prosegui se non hai preferenze 19:27
stop 19:27
Ora puoi chiedermi qualcos'altro! 19:27
Write a message...