Chatbot informativo sui prodotti della Mulino Bianco

Implementazione di un chatbot mediante il framework RASA

Autori:
Chiara Amalia Caporusso
Margherita Galeazzi
Simone Scalella
Zhang Yihang

Sommario

1 Introduzione

In questa relazione illustreremo gli aspetti peculiari della progettazione e dell'implementazione di un chatbot che ha come scopo finale quello di informare i consumatori circa i prodotti della Mulino Bianco. Entrando nel dettaglio esso permetterà di conoscere le informazioni nutrizionali dei vari prodotti, i loro ingredienti, i loro allergeni e simili, oltre a consentire all'utente finale di visionare un'immagine del prodotto.

1.1 Chatbot

Un chatbot è un software che simula ed elabora le conversazioni umane (scritte o parlate), consentendo agli utenti di interagire con i dispositivi digitali come se stessero comunicando con altri esseri umani. Per rendere tutto ciò possibile un chatbot sfrutta diverse tecniche di intelligenza artificiale, insieme a tecniche di machine learning e alla teoria del Natural Language Processing.
Ci sono alcune caratteristiche di base che un buon assistente virtuale dovrebbe avere e queste sono:

- ***Interattività***: devono permettere un interazione bidirezionale, in quanto dovrebbero comprendere l'input fornito dall'utente e fornire ad esso dei risultati utilizzando il deep learning e l'elaborazione del linguaggio naturale;
- ***Iteratività***: il chatbot dovrebbe avere memoria delle precedenti iterazioni avute con l'utente e restituire le informazioni adatte all'iterazione specifica che si sta avendo in un dato momento;
- ***Additività***: un chatbot non deve essere programmato per un singolo task, ma dovrebbe essere dinamico imitando le capacità di un cervello umano ed adattandosi all'ambiente circostante;
- ***Contestualizzazione***: al fine di fornire un output adeguato secondo quanto richiesto dall'utente, i chatbot devono essere in grado di identificare ed estrarre gli elementi contestuali dall'input come il task, l'obiettivo, il tempo, il luogo.

I chatbot portano numerosi vantaggi alle aziende che li utilizzano in quanto sono disponibili 24/7 e impiegano meno tempo di un operatore umano per completare un task.

Esistono due principali tipi di chatbot.

- ***I chatbot dedicati alle attività (dichiarativi)*** sono programmi monouso che si concentrano sull'esecuzione di una funzione. Usando regole, NLP e pochissima ML, generano risposte automatizzate ma colloquiali alle richieste degli utenti. Le interazioni con questi chatbots sono altamente specifiche e strutturate e sono per lo più applicabili alle funzioni di assistenza e di servizio: pensa a domande frequenti interattive e consolidate. I chatbot dedicati alle attività sono in grado di gestire domande comuni, ad esempio query riguardo gli orari lavorativi o semplici transazioni che non coinvolgono una varietà di variabili. Sebbene utilizzino la NLP in modo tale che gli utenti finali possano sperimentarli in modo semplice, le loro capacità sono abbastanza basilari. Attualmente, questi sono i chatbot più usati.
- ***I chatbot predittivi basati sui dati (di conversazione)*** sono spesso indicati come assistenti virtuali o assistenti digitali e sono molto più sofisticati, interattivi e personalizzati rispetto ai chatbot dedicati alle attività. Questi chatbot sono consapevoli del contesto di riferimento e sfruttano la comprensione della lingua naturale (NLU), la NLP e la ML per imparare. Applicano intelligenza predittiva e analisi dei dati per consentire la personalizzazione in base ai profili degli utenti e al comportamento degli utenti precedenti. Gli assistenti digitali possono imparare nel tempo le preferenze di un utente, fornire raccomandazioni e persino anticipare le esigenze. Oltre a monitorare i dati e le linee guida, possono avviare conversazioni. Siri di Apple e Alexa di Amazon sono esempi di chatbot predittivi orientati al consumatore e basati sui dati.

Gli assistenti digitali avanzati sono inoltre in grado di connettere diversi chatbot monouso sotto un unico gruppo, estrarre diverse informazioni da ognuno e combinare queste informazioni per eseguire un'attività mantenendo comunque il contesto: in tal modo il chatbot non si "confonde".

1.2 Il framework RASA

RASA è un framework open source in Python dedicato alla creazione di chatbot conversazionali, si basa sul machine learning supervisionato, insieme alle tecniche di Natural Language Processing (NLP), per comprendere gli intenti degli utenti e fornire ad essi una risposta coerente.
A fronte del grande successo riscontato dai chatbot nel contesto aziendale, sono stati sviluppati molti altri framework per creare chatbot (come Dialogflow, Amazon Lex e Luis), ma i vantaggi nell'utilizzo di Rasa sono diversi:

1. RASA è open source, e ciò quindi permette di avere a disposizione il codice e poter intervenire manualmente sul bot in caso di necessità;
2. RASA è un progetto in continuo sviluppo, con una vasta community di sviluppatori a supporto. Gli stessi creatori sono disponibili a fornire chiarimenti e a risolvere bug e problemi vari;
3. RASA non è un servizio cloud e poter quindi essere ospitato in locale sulle macchine aziendali e questo è un vantaggio soprattutto nel caso in cui nelle chat con gli utenti passassero dati sensibili che, per motivi di privacy, non dovrebbero essere esposti all'esterno. La privacy dell'utente viene così garantita, avendo tutti i dati in locale.

L'architettura RASA è costituita da due componenti fondamentali, che sono:

- RASA NLU: si occupa di capire e classificare la volontà dell'utente (chiamato anche intente), prendendo come input del testo "libero" scritto da quest'ultimo e restituendo dati strutturati;
- RASA CORE: dopo aver classificato il messaggio dell'utente, elabora quella che sarà la risposta sulla base di ciò che è stato recepito al momento presente, ma anche in passato.

Come detto in precedenza RASA si basa sull'apprendimento supervisionato e quindi è necessario fornirgli dati di training, in particolare i cosiddetti ***intent***, che altro non sono che frasi di esempio che mappano le possibili motivazioni che un utente può avere per utilizzare il chatbot in questione.
Ad ogni frase scritta dall'utente, infatti, corrisponde un'intenzione e l'obiettivo del chatbot, in prima istanza, sarà proprio quello di individuare e classificare correttamente tale aspetto, quindi si dovrà addestrare il chatbot fornendogli un opportuno numero di esempi per ciascuna tipologia di richiesta
Una volta aver incluso tutti i possibili intent relativi al dominio di interesse, il chatbot sarà in grado di classificare un input ricevuto associandogli l'intent che più si addice, sulla base di un punteggio.

Quindi una volta che la componente RASA NLU rileva l'intent dell'input, si attiva il RASA Core che crea un modello di machine learning per imparare dagli esempi forniti e predire la risposta più adatta da restituire all'utente, scegliendola tra le ***utterences***.
Se invece la risposta prevede un'elaborazione più articolata, come ad esempio l'invocazione di una API o una query su database si deve ricorrere alle ***Actions***.
Tutti i file da utilizzare hanno estensione .yml, tranne le actions che sono un file Phyton.

2 Chatbot mulino bianco

Figura 2.1 - Logo Chatbot

Il chatbot oggetto di questo progetto ha l'obiettivo di informare i consumatori sui prodotti della Mulino Bianco (e delle altre imprese ad essa legate: Pan di Stelle e Gran Cereale). Il chatbot è stato realizzato per parlare e rispondere in italiano.
Ci si è focalizzati su un set di possibili informazioni che l'utente potrebbe voler conoscere, anche in base ai dati che si avevano a disposizione nel dataset utilizzato.
Con il dataset a disposizione sono state individuate tutte le informazioni che maggiormente interessano i consumatori. Essendo molte le informazioni si è deciso di raggrupparle in più funzionalità, in maniera tale da non avere delle conversazioni troppo lunghe. Le informazioni considerate sono relative ai gusti del consumatore, come ad esempio la categoria o un certo tipo di ingredienti. Inoltre, si è tenuta in considerazione anche la salute del consumatore, inserendo tra le informazioni anche gli allergeni.
Le possibili azioni che gli utenti potranno fare sono le seguenti:

- Visualizzare tutte le categorie dei prodotti presenti (ad es. torte, biscotti, …);
- Visualizzare come è composto il packaging del prodotto (ad es. plastica, carta, …);
- Visualizzare un prodotto, le diverse dimensioni dei pacchi disponibili, la porzione consigliata, la categoria in cui rientra il prodotto e gli ingredienti;
- Visualizzare un prodotto, i suoi ingredienti, i suoi allergeni e le sue kcal;
- Visualizzare i vari brand presenti, e una volta selezionato uno di essi tutti i prodotti di tale brand;
- Visualizzare tutti i prodotti che hanno meno calorie di quelle inserite dall'utente, per porzione;
- Comprare un dato prodotto.

Le funzioni che abbiamo descritto prima richiedono di accedere ad un dataset che consente al chatbot di trovare le informazioni che l'utente richiede e di registrare quelle relative alle vendite dei prodotti.
Il dataset è stato memorizzato in un database con all'interno due tabelle, una per gli ingredienti e una per gli acquisti.

2.1 Il dataset

Ai fini del progetto si è utilizzato un database gestito con MySQL che è un *relational database management system* (RDBMS) composto da un client a riga di comando che permette l'interrogazione del database e un server nel quale il database è ospitato.
I dati per popolare tale database sono stati reperiti effettuando una ricerca sul sito https://it.openfoodfacts.org/ e scaricando i risultati.
Si è poi provveduto ad aggiornare il dataset così ottenuto aggiungendo i prodotti nuovi e andando a rimuovere quelli che non sono più in commercio, inoltre si è effettuato anche una fase di ETL che è consistita soprattutto nella rimozione delle colonne inutili e nella rimozione delle righe riguardanti i prodotti non esistenti in Italia.
Infine, è stata aggiunto un attributo ad ogni riga per associare ad ogni prodotto la sua immagine.

Sostanzialmente il nostro dataset è così composto:

mulino_bianco	
Attributo	**Descrizione**
id	Codice identificativo del prodotto
name	Nome del prodotto
quantity	Dimensioni (in grammi) del pacchetto del prodotto

servin_size	Porzione (in grammi) consigliata
packaging	Materiali di cui è composto l'imballaggio dei prodotti
brand	L'azienda (tutte controllate da Mulino Bianco) che produce il prodotto
category	Categoria del prodotto (ad es. torte, biscotti, …)
ingredients	Ingredienti del prodotto
allergens	Allergeni contenuti nel prodotto
energy_kcal_value	Kcal per 100g di prodotto
fat_value	Grassi in grammi contenuti in 100g di prodotto
saturated_fat_value	Grassi saturi in grammi contenuti in 100g di prodotto
carbohydrates_value	Carboidrati in grammi contenuti in 100g di prodotto
sugars_value	Zuccheri in grammi contenuti in 100g di prodotto
fiber_value	Fibre in grammi contenuti in 100g di prodotto
proteins_value	Proteine in grammi contenuti in 100g di prodotto
salt_value	Sale in grammi contenuti in 100g di prodotto
off_nova_groups	Categoria NOVA a cui appartiene il prodotto. NOVA è una classificazione che divide gli alimenti in: 1) Cibi non trasformati o minimamente lavorati; 2) Ingredienti per la cucina domestica; 3) Alimenti trasformati; 4) Alimenti ultraprocessati.
off_nova_groups_tags	Categoria NOVA a parole
image	Link ad un immagine del prodotto

2.2 La struttura

Una volta che si è creato un a ambiente virtuale tramite conda e una directory per lo sviluppo del progetto, si può inizializzare il progetto mediante il comando `rasa init` (messo a disposizione del framework) che crea la struttura per il progetto completa di tutte le componenti che sono necessarie per poter sviluppare un chatbot.

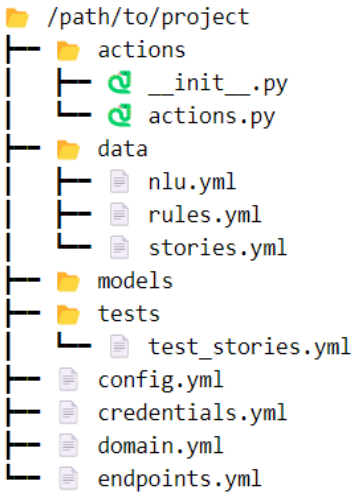


Figura 2.2 - Struttura di un progetto RASA

Le componenti sono quelle illustrate nella *Figura 2.2* e sono:

- ***actions.py*** che è uno script Python in cui sono definite tutte le classi che mappano le azioni che sono riportate nel file *domain.yml* e che richiedono un'elaborazione maggiore per costruire la risposta, quindi permettono di avere risposte che sono frutti di calcoli e non già predefinite;
- ***nlu.yml*** è un file nel quale vengono definiti gli esempi che saranno poi usati in fase di addestramento sia per gli *intent* che per le *entities*;

- ***rules.yml*** file che contiene regole predefinite per le policy di dialogo;
- ***stories.yml*** file che contiene esempi di conversazione, chiamati *path*, che il chatbot seguirà;
- ***config.yml*** file che contiene la configurazione del modello di machine learning;
- ***domain.yml*** file che contiene tutte le informazioni che sono necessarie al chatbot per poter operare, nello specifico in questo file è possibile vedere una lista di tutti gli *intent* che potranno essere espressi durante una conversazione insieme alle *entities* che saranno valorizzate, gli *slot* utilizzati per memorizzare i dati necessari per l'elaborazione, le *utterences* con le risposte predefinite che il chatbot fornirà, la lista contenente tutte le azioni e la lista dove vengono definite le *form*.

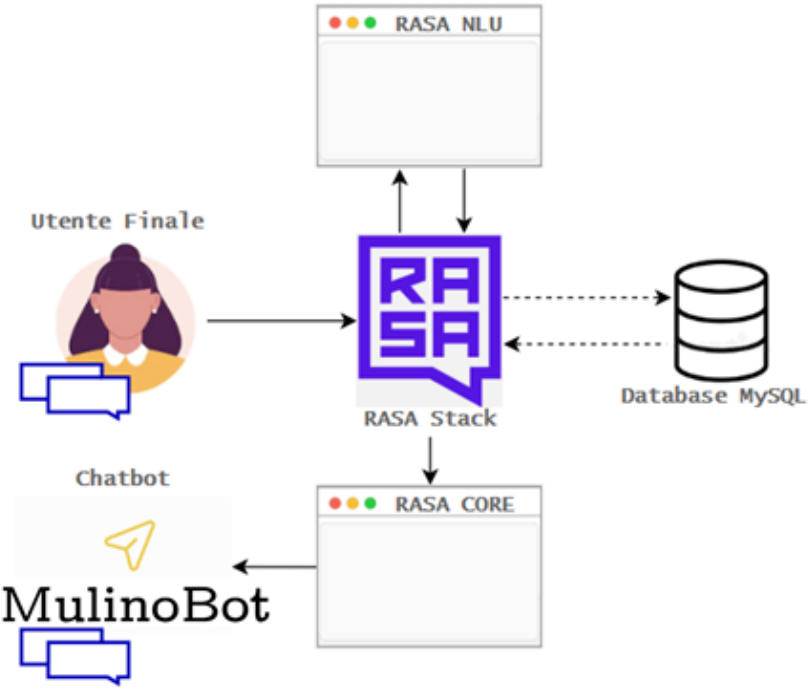


Figura 2.3 - Struttura del progetto

3 Funzionalità del chatbot

Procederemo in questo capitolo illustrando le varie funzionalità del chatbot e per ognuna di esse spiegando le scelte e le strategie applicate per svilupparle.

3.1 Visualizzare tutte le categorie dei prodotti presenti

Funzione del chatbot che permette di visualizzare tutte le categorie dei prodotti presenti (ad es. torte, biscotti, …).
Il primo step per realizzare tale funzionalità è stato fornire al modello di machine learning alcuni esempi di frasi che l'utente potrebbe scrivere al chatbot con l'intento di vedere le categorie dei vari prodotti e su questi il modello è stato poi allenato.

```
- intent: categoria
  examples: |
    - categorie disponibili
    - categorie, per favore
    - mostrami le categorie
    - mostami le tipologie
```

Figura 3.1 - Intent visualizzazione categoria

Per realizzare tale funzione è stata definita una classe di azione che, mediante una connessione al database, effettua una query in grado di estrarre distintamente tutte le categorie presenti all'interno del database ed associate a ciascun prodotto. In Figura 3.2 viene riportato il codice implementato:

```python
#Azione che permette di visualizzare l'immagine di un dato prodotto
class GetCategorie(Action):
    def name(self) -> Text:
        return "action_categoria"

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

        query = 'SELECT DISTINCT category FROM mulino_bianco'

        cursor.execute(query)
        result = cursor.fetchall()
        if len(result) == 0:
            dispatcher.utter_message("Non ci sono categorie di prodotto!")
        else:
            cat='Le categorie di prodotti sono le seguenti: \n'
            for elem in result:
                cat=cat+f' - {elem[0]}\n'
            dispatcher.utter_message(text=cat)

        return []
```

Figura 3.2 - Action associata alla visualizzazione delle categorie

Il risultato che si ottiene è una lista riportante tutte le categorie presenti nel database, riportata in Figura 3.3:

```
Your input ->  categorie, perfavore
Le categorie di prodotti sono le seguenti:
 - biscotti
 - biscotti ripieni
 - gelati
 - merendine
 - cereali
 - snack salati
 - crema
 - torte
 - pane e sostituti
 - fette biscottate
 - barrette dolci
 - torta
```

Figura 3.3 - Output visualizzazione categorie

3.2 Visualizzare da cosa è composto il packaging del prodotto

Funzione del chatbot che permette di visualizzare come è composto il packaging del prodotto (ad es. plastica, carta, …).

Il primo step per realizzare tale funzionalità è stato fornire al modello di machine learning alcuni esempi di frasi che l'utente potrebbe scrivere al chatbot con l'intento di vedere il packaging (utile ai fini di un corretto riciclo) di un dato prodotto e su questi il modello è stato poi allenato.

```
- intent: vedi_packaging
  examples: |
    - packaging prodotto
    - mostra il packaging
    - vorrei vedere il packaging di un prodotto
```

Figura 3.4 - Intent visualizzazione packaging

Per realizzare tale funzione è stata definita una classe di azione che, mediante una connessione al database, effettua una query in grado di estrarre il packaging associato ad un dato prodotto, passato in input dall'utente. Viene effettuato un ulteriore controllo sulla presenza del prodotto nel database. In

```python
#Azione che permette di visualizzare il packaging di un dato prodotto
class GetPackagingFromDB(Action):
    def name(self) -> Text:
        return "action_packaging"

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

        nome_prodotto=tracker.get_slot('prodotto').lower()
        query = 'SELECT packaging FROM mulino_bianco WHERE name=%s'

        cursor.execute(query,(nome_prodotto,))
        result = cursor.fetchall()
        if len(result) == 0:
            dispatcher.utter_message("Questo prodotto non esiste!")
        else:
            dispatcher.utter_message(text=f"Ecco a te il packaging di {nome_prodotto}: {result[0][0]}")

        return [{"name":"prodotto","event":"slot","value":None}]
```

Figura 3.5 - Action associata alla visualizzazione del packaging di un prodotto

In Figura 3.6 - Output visualizzazione packagingFigura 3.6 viene riportata l'interazione con il chatbot

```
Your input ->  vorrei vedere il packaging di un prodotto
Di che prodotto vuoi vedere il packaging?
Your input ->  baiocchi
Ecco a te il packaging di baiocchi: plastica,carta,polipropilene
```

Figura 3.6 - Output visualizzazione packaging

3.3 Visualizzare informazioni sul prodotto

Funzione del chatbot che permette di visualizzare un prodotto, le diverse dimensioni dei pacchi disponibili, la porzione consigliata, la categoria in cui rientra il prodotto e gli ingredienti.

```
- intent: prodotti_con_info
  examples: |
    - lista prodotti
    - fammi vedere i vostri prodotti
    - elenco dei prodotti
    - vorrei vedere elenco dei prodotti
```

Figura 3.7 - Intent visualizzazione informazioni

Questa funzione è una funzione articolata e richiede diverse informazioni all'utente. In fase di implementazione si è tenuta in considerazione la possibilità che l'utente non volesse specificare tutte le informazioni che sono state considerate, quindi, su quattro all'utente ne interessano due. Di conseguenza si è deciso di realizzare questa funzione rendendo le richieste di informazioni facoltative.
Sono state realizzate due classi di azioni, una per l'inserimento di valori da parte dell'utente e una per costruire la query da fare al database. Tutte le azioni sono state riportate nel file domain.
In Figura 3.8 è riportata un'immagine del codice scritto per realizzare queste azioni.

```python
#Validazione informazione prodotti
class ValidateProdottoInfoForm(FormValidationAction):
    def name(self) -> Text:
        return "validate_prodotto_info_form"

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

        print("validate info ingredienti")
        if(slot_value[0] == 'no' or slot_value[0] == 'skip'):
            print('skip slot')
            return {"info_ingredienti": ['no']}

        arrayfied = slot_value[0].replace(' ',',')
        arrayfied = arrayfied.split(',')
        arrayfied = list(filter(lambda x: len(x)>0,arrayfied))
        return {"info_ingredienti": arrayfied}

    def validate_info_serving(
```

Figura 3.8 - Action associata alla visualizzazione di informazioni (1)

La prima azione è quella di verifica dell'input inserito dall'utente. All'utente viene comunicata la possibilità di inserire tutte o solo una parte delle informazioni richieste. L'input inserito dall'utente viene controllato, infatti, se inserisce la stringa "no" o "skip" lo slot corrispondente verrà valorizzato con un valore di controllo. Tale valore è la stringa "no" che useremo per fare ulteriori controlli in fase di invio della query.
Si offre la possibilità di inserire più ingredienti nella richiesta, infatti, lo slot degli ingredienti è stato definito come una lista. Infine, sono stati realizzati alcuni controlli di sicurezza, ad esempio controlliamo se per le calorie è stato inserito un numero, altrimenti, avvisiamo l'utente, oppure controlliamo se la categoria inserita esiste, tramite una semplice query e poi, successivamente, controllandone il risultato.

Al termine della storia viene attivata l'action di invio della query. Riportiamo in Figura 3.9 l'immagine del codice.

```python
# risultato delle informazioni prodotti
class RisultatoDellaRicerca(Action):
    def name(self) -> Text:
        return "query_result"

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

        emptyQuery = True
        beofore = False
        BasicQueryString = 'SELECT name,quantity,servin_size,category,ingredients FROM mulino_bianco WHERE '
        info_ingredienti=tracker.get_slot('info_ingredienti')
        if(len(info_ingredienti)>0 and info_ingredienti[0] != 'no'):
            emptyQuery = False
            beofore = True
            regexStr = '\''      (variable) info_ingredienti: Any | None
            for ingredienti in info_ingredienti:
                regexStr += ingredienti.lower()+'|'
            BasicQueryString += 'ingredients REGEXP ' + regexStr[:-1]+'\''

        info_serving=tracker.get_slot('info_serving')
        if(info_serving.lower() != 'no'):
            emptyQuery = False
            if(beofore):
```

Figura 3.9 - Action associata alla visualizzazione di informazioni (2)

In questa azione, in base a ciò che è stato inserito all'utente, viene realizzata la query che verrà inviata al database.

La query viene costruita in maniera incrementale, controllando il valore presente nello slot. Se nello slot è presente la stringa "no", allora, non si aggiunge la condizione nel where relativa a quell'informazione. Invece, se è presente l'input dell'utente si costruisce un pezzo della query, e lo si aggiunge alla where.

Per evitare errori nella costruzione della query sono state utilizzate diverse guardie, in quanto l'utente ha il massimo grado di libertà nel rispondere al chatbot. Ad esempio, un caso limite, è quello in cui l'utente inizia la story ma inserisce solo no o skip, quindi, in questo caso il chatbot non deve inviare nessuna query al database.

In Figura 3.10 riportiamo un'immagine dell'output ottenuto al termine della storia.

```
Your input ->  Vorrei la lista dei vostri prodotti
A che tipo di ingredienti sei interessato? puoi rispondere skip o no se non ti interessa il filtro su questo campo
Your input ->  latte
Hai una preferenza di serving size minimo? puoi rispondere skip o no se non ti interessa il filtro su questo campo
Your input ->  skip
A che tipo di categoria sei interessato? puoi rispondere skip o no se non ti interessa il filtro su questo campo
Your input ->  biscotti
Hai una preferenza relativamente alla quantita minima richiesta? puoi rispondere skip o no se non ti interessa il filtro su questo campo
Your input ->  skip
I prodotti con la condizione proposta sono:
- abbracci, quantita: 350, servin size: 11.0, categoria: biscotti, ingredienti:farina di frumento, zucchero, burro, olio di girasole, uova fresche, cacao, p
anna fresca pastorizzata, latte scremato in polvere, miele, sale, agenti lievitanti, aroma vanillina
- abbracci, quantita: 700, servin size: 11.0, categoria: biscotti, ingredienti:farina di frumento, zucchero, burro, olio di girasole, uova fresche, cacao, p
anna fresca pastorizzata, latte scremato in polvere, miele, sale, agenti lievitanti, aroma vanillina
- abbracci, quantita: 1000, servin size: 11.0, categoria: biscotti, ingredienti:farina di frumento, zucchero, burro, olio di girasole, uova fresche, cacao,
panna fresca pastorizzata, latte scremato in polvere, miele, sale, agenti lievitanti, aroma vanillina
- baiocchi, quantita: 168, servin size: 9.0, categoria: biscotti ripieni, ingredienti:biscotto: farina di frumento, zucchero, olio di girasole, uova fresche
, latte scremato in polvere, latte fresco pastorizzato, agenti lievitanti, sale, amido di frumento, aroma. crema alla nocciola e cacao: zucchero, grassi ed
oli vegetali, nocciole, cacao, latte scremato in polvere, amido di frumento, aroma
- baiocchi, quantita: 60, servin size: 9.0, categoria: biscotti ripieni, ingredienti:biscotto: farina di frumento, zucchero, olio di girasole, uova fresche,
 latte scremato in polvere, latte fresco pastorizzato, agenti lievitanti, sale, amido di frumento, aroma. crema alla nocciola e cacao: zucchero, grassi ed o
li vegetali, nocciole, cacao, latte scremato in polvere, amido di frumento, aroma
- baiocchi al pistacchio, quantita: 168, servin size: 28.0, categoria: biscotti ripieni, ingredienti:biscotto: farina di frumento, zucchero, olio di girasol
e, uova fresche, latte scremato in polvere, latte fresco pastorizzato, agenti lievitanti, sale, amido di frumento, aroma, crema al pistacchio
- baiocchi choco, quantita: 144, servin size: 24.0, categoria: biscotti ripieni, ingredienti:farina di frumento, zucchero, oli vegetali, amido di frumento,
burro, sciroppo di glucosio, uova fresche,  agenti lievitanti , sale, aroma, cioccolato al latte con nocciole
- batticuori, quantita: 350, servin size: 9.8, categoria: biscotti, ingredienti:farina di frumento, zucchero, olio di girasole, amido di frumento, burro, ca
cao, latte fresco pastorizzato, cioccolato fondente, emulsionante: lecitina di soia, aromi, agenti lievitanti, sale
- biscocrema, quantita: 168, servin size: 28.0, categoria: biscotti ripieni, ingredienti: farina di frumento, zucchero, oli vegetali, amido di frumento, cac
ao magro, sciroppo di glucosio, latte intero in polvere, burro, latte concentrato zuccherato, uova fresche, agenti lievitanti, sale, aromi, crema alle nocci
ole e cacao con granella di biscotto, cioccolato, crema al latte
- biscottoni, quantita: 700, servin size: 20.0, categoria: biscotti, ingredienti:farina di frumento, zucchero, olio di girasole, uova fresche, latte fresco
pastorizzato, burro, zucchero di canna, agenti lievitanti, sale, aromi
- buongrano, quantita: 350, servin size: 8.0, categoria: biscotti, ingredienti:farina integrale di frumento, olio di girasole, zucchero, zucchero di canna,
farro (grano) integrale croccante, latte fresco pastorizzato, uova fresche, estratto di malto d'orzo, agenti lievitanti, aromi (latte), sale
- cioccoavena, quantita: 220, servin size: 27.5, categoria: biscotti, ingredienti:farina di frumento, cioccolato fondente, zuchero, fiocchi di avena, olio d
i girasole, amido di frumento, scorze di arancia candite, fibra di frumento, latte scremato in polvere, agenti lievitanti, cacao magro, aromi, emulsionanti
- cioccograno, quantita: 330, servin size: 12.5, categoria: biscotti, ingredienti:farina integrale di frumento, farina di frumento, zucchero, burro, olio di
 girasole, uova fresche, cioccolato, cacao, fibra di frumento, miele, latte scremato in polvere, sale, aromi, agenti lievitanti
```

Figura 3.10 - Output visualizzazione informazioni

3.4 Visualizzare un prodotto, i suoi ingredienti, i suoi allergeni e le sue kcal

Descriviamo la funzione che realizza la visualizzazione dei prodotti, prendendo in considerazione gli allergeni, gli ingredienti, il tipo di prodotto e le sue kcal.
Per realizzare questa funzione si è deciso di inserire un nuovo intent, nel quale andiamo a riportare gli esempi di frasi che una persona con allergie o intolleranze alimentari può scrivere al chatbot.

```
- intent: info_problem
  examples: |
    - Ho dei problemi e mi servono informazioni
    - Ho dei problemi di salute
    - Voglio comprare un prodotto, ma mi servono informazioni per la mia salute
    - Non posso mangiare, ho dei problemi
    - Voglio un prodotto particolare
    - Ho delle richieste da fare
```

Figura 3.11 - Intent per richiedere informazioni su un prodotto

Successivamente sono state realizzate le due azioni che servono per recuperare l'input inserito dall'utente e per realizzare la query da inviare al database.
Durante la realizzazione di questa funzione si è tenuto conto di tutte le considerazioni fatte precedentemente, quindi, l'utente ha la facoltà di inserire tutte o solo alcune delle informazioni richieste.
Di seguito riportiamo un'immagine contenente il codice utilizzato per realizzare la funzione.

```
    if(slot_value == 'no' or slot_value == 'skip'):
        print('skip slot')
        dispatcher.utter_message("A che tipo di ingredienti sei interessato? Puoi rispondere skip, o no, se non ti intere
        return {"allergeni_slot": 'no'}

    query = f'SELECT name,allergens FROM mulino_bianco WHERE allergens LIKE \'%{slot_value}%\''

    cursor.execute(query)
    result = cursor.fetchall()
    if len(result) == 0:
        dispatcher.utter_message("L'allergene non è presente nei nostri prodotti"+"\n")
        dispatcher.utter_message("A che tipo di ingredienti sei interessato ? Puoi rispondere skip, o no, se non ti inter
        return {'allergeni_slot': 'no'}

    dispatcher.utter_message("A che tipo di ingredienti sei interessato ? Puoi rispondere skip, o no, se non ti interessa
    return {'allergeni_slot': slot_value}

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

    print("validate info ingredienti")
    if(slot_value[0] == 'no' or slot_value[0] == 'skip'):
        print('skip slot')
        dispatcher.utter_message("Quante calorie deve contenere al massimo ? Puoi rispondere skip, o no, se non interessa
        return {"ingrediente_slot": ['no']}
```

Figura 3.12 - Action associata alla visualizzazione di un prodotto filtrando sugli allergeni (1)

Anche in questa funzione sono stati inseriti dei controlli, ad esempio se l'utente inserisce un allergene che non esiste, viene allertato con un messaggio, che gli comunica l'assenza di quell'allergene in tutti i nostri prodotti. Le informazioni richieste all'utente sono l'allergene, gli ingredienti che gli interessano, infatti, l'utente può inserire più ingredienti che vengono inseriti in uno slot di tipo List, il valore massimo di kcal che deve contenere, e infine, il tipo di prodotto che gli interessa. L'azione termina quando l'utente ha inserito tutte le informazioni richieste.
Al termine di questa storia viene attivata la funzione di invio della query al database. Di seguito riportiamo un'immagine contente il codice utilizzato per realizzare questa funzione.

```python
class SubmitAllergeni(FormValidationAction):
    def name(self) -> Text:
        return "submit_allergeni"

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

        emptyQuery = True
        beofore = False
        BasicQueryString = 'SELECT name,ingredients,energy_kcal_value,allergens FROM mulino_bianco WHERE '
        info_ingredienti=tracker.get_slot('ingrediente_slot')
        if(len(info_ingredienti)>0 and info_ingredienti[0] != 'no'):
            emptyQuery = False
            beofore = True
            regexStr = '\''
            for ingredienti in info_ingredienti:
                regexStr += ingredienti.lower()+'|'
            BasicQueryString += 'ingredients REGEXP ' + regexStr[:-1]+'\''

        info_calorie=tracker.get_slot('calorie_slot')
        if(info_calorie.lower() != 'no'):
            emptyQuery = False
            if(beofore):
                BasicQueryString += ' AND energy_kcal_value <=' + info_calorie
            else:
                beofore = True
```

Figura 3.13 - Action associata alla visualizzazione di un prodotto filtrando sugli allergeni (2)

Questa funzione costruisce la query in maniera incrementale, basandosi sulle informazioni inserite dall'utente. Si controllano gli slot valorizzati con gli input dell'utente, se il valore è "no", perché l'utente ha deciso di non inserire quell'informazione, allora, la funzione non inserirà nella query la condizione corrispondente, altrimenti, se è stato inserito un valore corretto, la funzione aggiunge la condizione corrispondente alla where della query.
Una volta terminata la query si mostra il risultato all'utente e a tutti gli slot viene assegnato il valore "None", altrimenti, continuerebbero a contenere i vecchi valori, e se l'utente riattiva la storia ottiene sempre lo stesso risultato. In questo modo gli slot sono pronti per essere utilizzati nella riattivazione successiva della storia.
Di seguito riportiamo un'immagine dell'output ottenuto al termine della storia.

```
Your input ->  Ho dei problemi di salute
Sei allergico a qualcosa ?
Your input ->  glutine
A che tipo di ingredienti sei interessato ? Puoi rispondere skip, o no, se non ti interessa il filtro su questo campo
Your input ->  latte
Quante calorie deve contenere al massimo ? Puoi rispondere skip o no se non interessa il filtro su questo campo
Your input ->  skip
A che prodotto sei interessato ? Puoi rispondere skip, o no, se non ti interessa il filtro su questo campo
Your input ->  skip
I prodotti con la condizione proposta sono:
- frollini al miele millefiori senza glutine, ingredienti: fecola di patata, amido, farina di mais, zucchero, olio di se
mi di girasole, miele, uova fresche, aromi, latte scremato in polvere, agenti lievitanti, proteine del latte, sale, emul
sionante. Kcal: 457, allergeni: soia,latte,
```

Figura 3.14 - Output richiesta informazioni prodotto

3.5 Visualizzare i vari brand presenti e i relativi prodotti

Funzione del chatbot che permette di visualizzare i vari brand presenti, e una volta selezionato uno di essi tutti i prodotti di tale brand.
Il primo step per realizzare tale funzionalità e stato fornire al modello di machine learning alcuni esempi di frasi che l'utente potrebbe scrivere al chatbot con l'intento di conoscere i brand che sono presenti nel dataset e su questi il modello è stato poi allenato.

```yaml
- intent: brand
  examples: |
    - quali sono i brand?
    - mostra i brand
```

Figura 3.15 - Intent visualizzazione brand e loro prodotti

Successivamente si è definita l'azione che connettendosi al database prende tutti i diversi brand (mediante una query SQL) e fa restituire al chatbot tali brand sottoforma di bottone.

```python
#Azione che permette di visualizzare lista dei brand
class GetBrand(Action):
    def name(self) -> Text:
        return "get_brand"

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


        query = 'SELECT DISTINCT brands FROM mulino_bianco'

        cursor.execute(query)
        result = cursor.fetchall()
        if len(result) == 0:
            dispatcher.utter_message("Non ci sono brand!")
        else:
            brd='I brand presenti sono: \n'
            buttons=[]
            for elem in result:
                brd=brd+f' - {elem[0]}\n'
                buttons.append({"title": elem[0], "payload": f'/viewBrandProduct{{"brand":"{elem[0]}"}}'})
            dispatcher.utter_button_message(brd,buttons)
        return []
```

Figura 3.16 - Action associata alla visualizzazione dei brand presenti e dei loro prodotti

Si è deciso di utilizzare un bottone per rendere più user-friendly lo step successivo ovvero quello di conoscere tutti i prodotti di tale brand, cosicché l'utente non debba digitare nuovamente il brand rischiando di fare un errore.

```
Your input ->  quali sono i brand?
? I brand presenti sono:
 - mulino bianco
 - pan di stelle
 - gran cereale
 (Use arrow keys)
   1: mulino bianco
 » 2: pan di stelle
   3: gran cereale
```

```
  2: pan di stelle
Ecco i prodotti di pan di stelle:
 - biscocrema
 - cono pan di stelle
 - gelato biscotto pan di stelle
 - merenda pan di stelle
 - mooncake
 - torta pan di stelle
```

Figura 3.17 - Output funzione visualizzazione dei brand e dei loro prodotti

3.6 Visualizzare prodotti in relazione alle calorie

Funzione del chatbot che permette di visualizzare tutti i prodotti che hanno meno calorie di quelle inserite dall'utente, per porzione.

Il primo step per realizzare tale funzionalità e stato fornire al modello di machine learning alcuni esempi di frasi che l'utente potrebbe scrivere al chatbot con l'intento di visualizzare la lista delle merendine che hanno meno calorie di quelle specificate dall'utente per porzione consigliata. Il modello è stato poi allenato con questi esempi.

A differenza degli altri intent qua si può notare che vi è un valore tra parentesi quadre, affiancato a uno tra parentesi tonde. Questo è fatto per creare l'entità "calorie", mediante la quale il chatbot capirà che tale valore riguarda le calorie che saranno poi utilizzate per fare la query.

```
- intent: snack_meno_calorie
  examples: |
      - quali sono le merendine con meno di [100](calorie) kcal?
      - quali sono gli snack con meno di [254](calorie) kcal?
      - fammi vedere le merendine con meno di [310](calorie) kcal
      - fammi vedere gli snack con meno di [155](calorie) kcal
      - mostrami le merendine con meno di [450](calorie) kcal
      - mostrami gli snack con meno di [267](calorie) kcal
```

Figura 3.18 - Intent visualizzazione prodotti con meno di X kcal

Successivamente si è definita l'azione che connettendosi al database prende tutti i prodotti che rispettano la condizione posta (mediante una query SQL) e fa restituire al chatbot la lista di tali prodotti con il nome, la porzione (in grammi) e le kcal per porzione.

```python
#Azione che permette di visualizzare le merendine con meno di X calorie per porzione
You, 2 weeks ago | 2 authors (You and others)
class GetSnackCalLessFromDB(Action):
    def name(self) -> Text:
        return "get_snack_meno_cal"

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

        cal_serSize=tracker.latest_message['entities'][0]['value']
        query = f'SELECT name, servin_size, energy_kcal_value FROM mulino_bianco WHERE energy_kcal_value<({cal_serSize}*100)/servin_size'

        cursor.execute(query)
        result = cursor.fetchall()
        if len(result) == 0:
            dispatcher.utter_message("Non ci sono prodotti che rispettano questa condizione!")
        else:
            pl=f"I prodotti con meno di {cal_serSize} a porzione sono: \n"
            for elem in result:
                if not elem[0] in pl:
                    pl=pl+f' - {elem[0]}, porzione: {elem[1]}, kcal per porzione: {(elem[1]*elem[2])/100}\n'
            dispatcher.utter_message(text=pl)
        return [{"name":"calorie_slot","event":"slot","value":None}]
```

Figura 3.19 - Action associata alla visualizzazione degli snack con meno di X calorie

Di seguito è riportata l'interazione con il chatbot:

```
Your input ->  fammi vedere gli snack con meno di 35 kcal
I prodotti con medo di 35 a porzione sono:
 - baiocchi, porzione: 9.0, kcal per porzione: 28.71
 - fette biscottate dorate, porzione: 8.8, kcal per porzione: 34.408
 - fette biscottate integrali, porzione: 8.8, kcal per porzione: 33.792
 - fiori d acqua con farina integrale, porzione: 3.0, kcal per porzione: 12.06
 - fiori d'acqua, porzione: 3.0, kcal per porzione: 12.24
 - galletti, porzione: 6.0, kcal per porzione: 28.5
 - pan di stelle, porzione: 7.0, kcal per porzione: 33.81
 - rigoli con miele millefiori italiano, porzione: 7.4, kcal per porzione: 33.522
```

Figura 3.20 - Output funzione visualizzazione prodotti con meno di X kcal

3.7 Visualizzare l'immagine di un prodotto

Funzione del chatbot che permette di visualizzare l'immagine di un prodotto.
Il primo step per realizzare tale funzionalità e stato fornire al modello di machine learning alcuni esempi di frasi che l'utente potrebbe scrivere al chatbot con l'intento di visualizzare un'immagine di un particolare prodotto, che sarà specificato dall'utente nell'interazione successiva. Il modello è stato poi allenato con questi esempi.

```yaml
- intent: vedi_foto
  examples: |
    - vorrei vedere
    - voglio vedere
    - fammi vedere
    - mostrami
    - mostrami la foto
    - mostrami l'immagine
    - vorrei vedere una foto
    - voglio vedere una foto
    - fammi vedere una foto
    - mostrami una foto
    - vorrei vedere un immagine
    - voglio vedere un immagine
    - fammi vedere un immagine
    - mostrami un immagine
```

Figura 3.21 - Intent visualizzazione immagine del prodotto

Successivamente si è definita l'azione che connettendosi al database prende il link del prodotto di interesse (mediante una query SQL) e fa restituire al chatbot un messaggio di testo insieme al link dell'immagine.

```python
#Azione che permette di visualizzare l'immagine di un dato prodotto
class GetImageFromDB(Action):
    def name(self) -> Text:
        return "get_image_from_db"

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

        nome_prodotto=tracker.get_slot('prodotto').lower()
        query = 'SELECT image FROM mulino_bianco WHERE name=%s'

        cursor.execute(query,(nome_prodotto,))
        result = cursor.fetchall()
        if len(result) == 0:
            dispatcher.utter_message("Questo prodotto non esiste!")
        else:
            dispatcher.utter_message(text=f"Ecco a te un immagine di {nome_prodotto}:",image=result[0][0])

        return [{"name":"prodotto","event":"slot","value":None}]
```

Figura 3.22 - Action associata alla visualizzazione di un'immagine del prodotto

A differenza delle altre risposte qui si ha un attributo image, che serve per consentire la visualizzazione dell'immagine.

```
Your input ->  mostrami una foto
Di che prodotto vuoi vedere l'immagine?
Your input ->  abbracci
Ecco a te un immagine di abbracci:
Image: https://drive.google.com/file/d/1A95wE5TlGgK9jwPud-mmVJX9pf1l0Ntm/view?usp=share_link
```

Figura 3.23 - Output visualizzazione immagine del prodotto

3.8 Acquistare un dato prodotto

Funzione del chatbot che permette di effettuare l'acquisto di un prodotto.
Il primo step per realizzare tale funzionalità è stato fornire al modello di machine learning alcuni esempi di frasi che l'utente potrebbe scrivere al chatbot con l'intento di acquistare un prodotto, che sarà specificato dall'utente nell'interazione successiva, insieme alla dimensione della confezione desiderata. Il modello è stato poi allenato con questi esempi.

```yaml
- intent: acquisto_prodotto
  examples: |
    - comprare prodotti
    - voglio una confezione
    - vorrei una confezione
    - acquistare prodotti
    - acquisto confezione
    - acquisto regali
    - desidero acquisti
    - comprare un prodotto
    - volerre prodotto
    - comprare qualcosa
    - volere qualcosa
    - fare reagali
    - volere spendere
```

Figura 3.24 - Intent acquisto prodotto

Per consentire l'acquisto mediante il chatbot si è dovuto effettuare un primo step nel quale si effettuavano le validazioni degli slot, della form di acquisto riempiti dall'utente; ovvero si andava a vedere se nel database tali prodotti esistevano e se la dimensione del pacco che l'utente richiedeva era effettivamente presente.

```python
# Azione di validazione della form per l'acquisto di un prodotto
class ValidateProdottoForm(FormValidationAction):
    def name(self) -> Text:
        return "validate_prodotto_form"

    def validate_tipo_prodotto(
        self,
        slot_value: Any,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: DomainDict):

        nome_prodotto=tracker.get_slot('tipo_prodotto').lower()
        query = 'SELECT DISTINCT name FROM mulino_bianco'

        cursor.execute(query)
        result = [row for [row] in cursor.fetchall()]

        #print(result)
        print(nome_prodotto)
        if nome_prodotto not in result:
            dispatcher.utter_message(text="prodotto inesistente")
            return {'tipo_prodotto': None}
        dispatcher.utter_message(text=f'Ok, hai scelto il prodotto {nome_prodotto}')
        return {'tipo_prodotto': nome_prodotto}

    def validate_dimensione_prodotto(
        self,
        slot_value: Any,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: DomainDict):

        dimensione=tracker.get_slot('dimensione_prodotto')
        tipo_prodotto = tracker.get_slot('tipo_prodotto').lower()
        query = f'SELECT quantity FROM mulino_bianco WHERE name = \'{tipo_prodotto}\''
        cursor.execute(query)
        result = [row for [row] in cursor.fetchall()]
        if dimensione not in result:
            print("validazione fallita")
            dispatcher.utter_message(text="Il valore della dimensione inserita non va bene")
            validStr = ''
            for content in result:
                validStr += content+','
            dispatcher.utter_message(text="I valori ammissibili sono: "+validStr[:-1])
            return {"dimensione_prodotto": None}
        print("validazione successa")
        dispatcher.utter_message(text=f"Ok, hai scelto la dimensione {dimensione}")
        return {"dimensione_prodotto": dimensione}
```

Figura 3.25 - Action acquisto prodotto

Terminata la prima azione, che controlla l'input inserito dall'utente, la storia attiva la seconda azione realizzata per questa funzione, quella che effettua l'invio della query al database.
La query viene costruita inserendo nel campo values i valori definiti dall'utente. La submit della query viene realizzata all'interno di un costrutto try except, il quale serve ad avvisare l'utente qualora si presentino problemi legati al database. Se non si sono verificati problemi, l'utente viene avvisato dell'esito positivo, relativo al termine dell'azione.

```python
# Azione che effettua la submit di un acquisto
class SubmitAcquisto(FormValidationAction):
    def name(self) -> Text:
        return "submit_acquisto"

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

        tipo_prod = tracker.get_slot('tipo_prodotto')
        dim_prod = tracker.get_slot('dimensione_prodotto')
        query = "INSERT INTO acquisti (prodotto, dimensione) VALUES (%s,%s);"
        try:
            cursor.execute(query,(tipo_prod,dim_prod))
            mydb.commit()
            dispatcher.utter_message(text="Acquisto avvenuto con successo !!")
        except:
            dispatcher.utter_message(text="Errore database")
        #'tipo_prodotto': None


        return [{"name":"dimensione_prodotto","event":"slot","value":None},{"name":"tipo_prodotto","event":"slot","value":None}]
```

Figura 3.26 - Inserimento dell'acquisto nel database

Di seguito viene riportata l'interazione con il chatbot.

```
Your input ->  ciao
Ciao! Come stai
Your input ->  Vorrei effettuare degli acquisti
Perfetto, procediamo con l'acquisto
Che tipo di prodotto vuoi acquistare ?
Your input ->  baiocchi al pistacchio
Ok, hai scelto il prodotto baiocchi al pistacchio
Di che dimensione deve essere la confezione richiesta ?
Your input ->  168
Ok, hai scelto la dimensione 168
Io adesso ordinerò il prodotto per te
Io adesso ordinerò un baiocchi al pistacchio da 168
Acquisto avvenuto con successo !!
```

Figura 3.27 - Output acquisto prodotto

4 Connessione a Telegram e Test

4.1 Connessione

L'ultima fase del progetto consiste nel collegamento del chatbot realizzato ad una piattaforma di messaggistica istantanea, in maniera tale che gli utenti possano interagirci in maniera semplice, da smartphone e soprattutto con un'interfaccia grafica moderna.
La piattaforma che si è deciso di utilizzare è Telegram, si è utilizzato il servizio ngrok per finalizzare tale connessione e mandare in esecuzione il chatbot su server locale.

```
telegram:
  access_token: 5934480459:AAGXd10Hf3rJxSzQVrP0J2bTYqZ9y-FKyl8
  verify: MulinoWhiteBot_bot
  webhook_url: "https://9472-93-41-36-21.eu.ngrok.io/webhooks/telegram/webhook"
```

Figura 4.1 - configurazione connessione

La Figura sopra riporta la configurazione che è stato necessario impostare all'interno del file credentials.yml per poter effettuare il deploy su Telegram.
Gli elementi che sono stati impostati sono i seguenti:

- ***access_token***: Token che si è ottenuto mediante la procedura guidata di deploy su Telegram tramite il bot BotFather;
- ***verify***: nome del bot in questione assegnato in fase di deploy su Telegram;
- ***webhook_url***: endpoint che permette alla macchina locale e al chatbot di comunicare. Tale endopoint è stato realizzato mediante ngrok.

4.2 Testing

Finita l'implementazione e la connessione alla piattaforma di messaggistica istantanea scelta, si è proceduto a testare il chatbot simulando le possibili interazioni che l'utente potrebbe avere con esso. Si è proceduto a simulare anche il comportamento di utente sbadato, cioè, di utente che inserisce, in modo sbadato, degli input errati. Questo ci è servito per verificare il corretto funzionamento dei controlli implementati.
Di seguito riportiamo alcune immagini con i risultati ottenuti.

4.2.1 Avvio del Bot

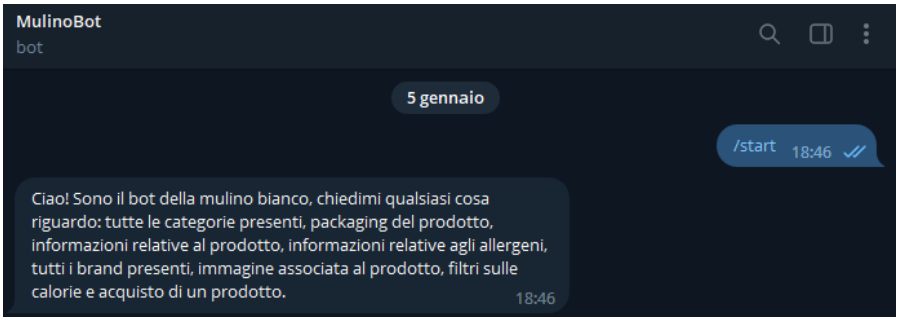


Figura 4.2 - Avvio del Bot

4.2.2 Visualizzare tutte le categorie del prodotto presenti

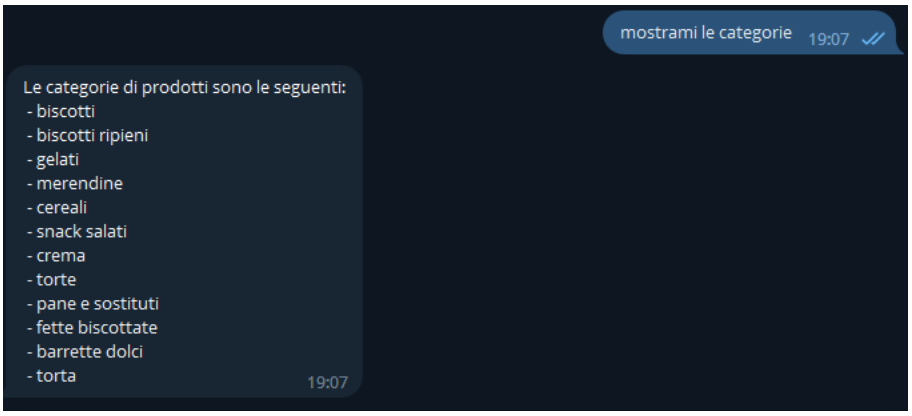
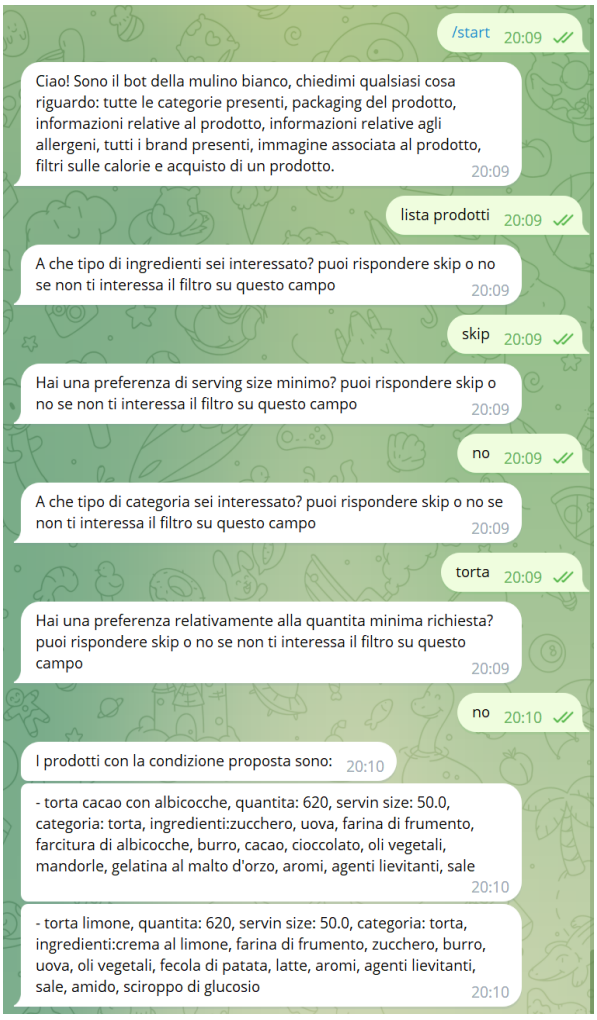


Figura 4.3 - categorie dei prodotti

4.2.3 Visualizzare informazioni prodotto



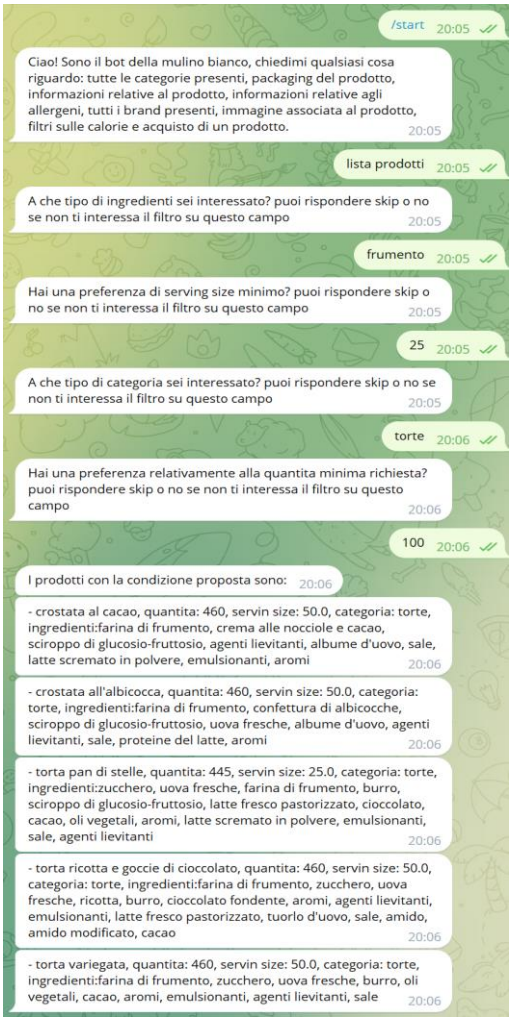


Figura 4.4 - Visualizzazione informazioni prodotto

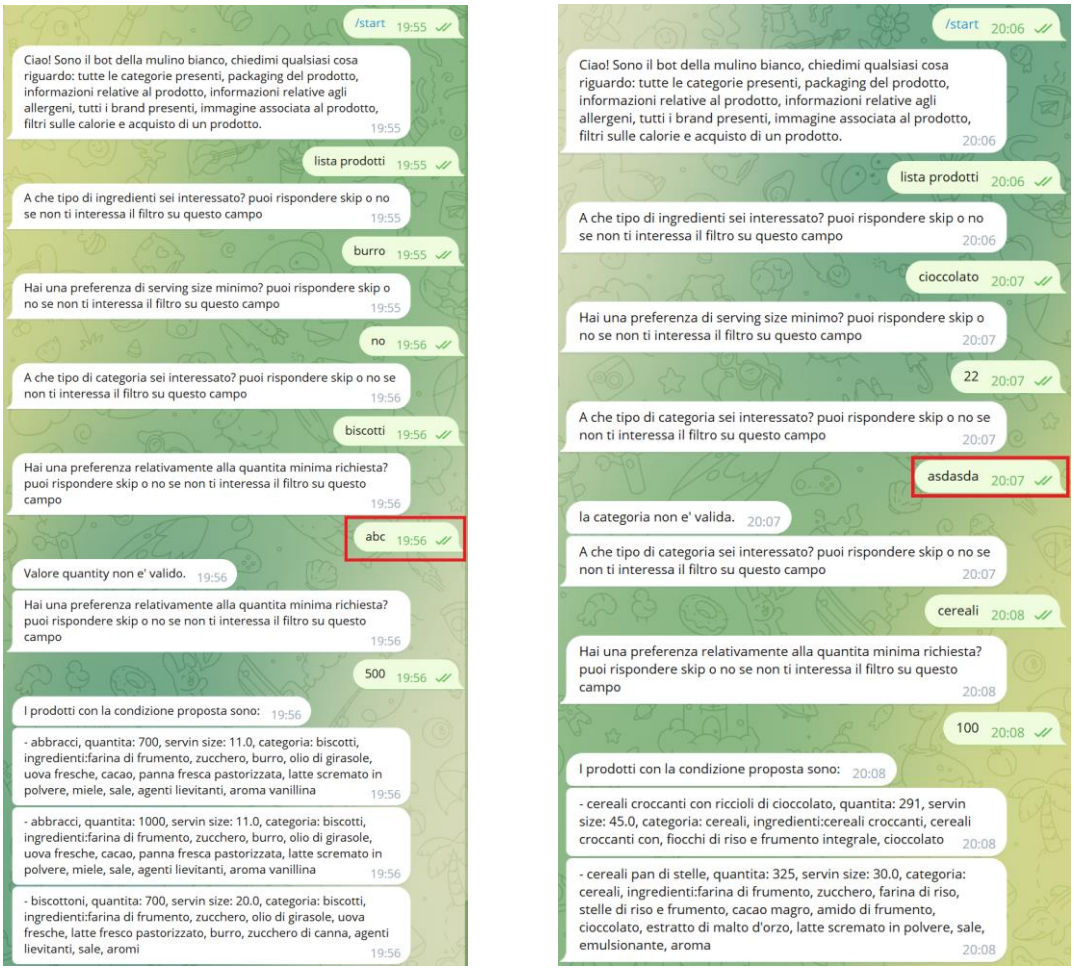


Figura 4.5 - Check sul valore della quantità e della categoria inseriti dall'utente

4.2.4 Visualizzare da cosa è composto il packaging del prodotto

Figura 4.6 - packaging prodotto

4.2.5 Visualizzare un prodotto, i suoi ingredienti, i suoi allergeni e le sue kcal

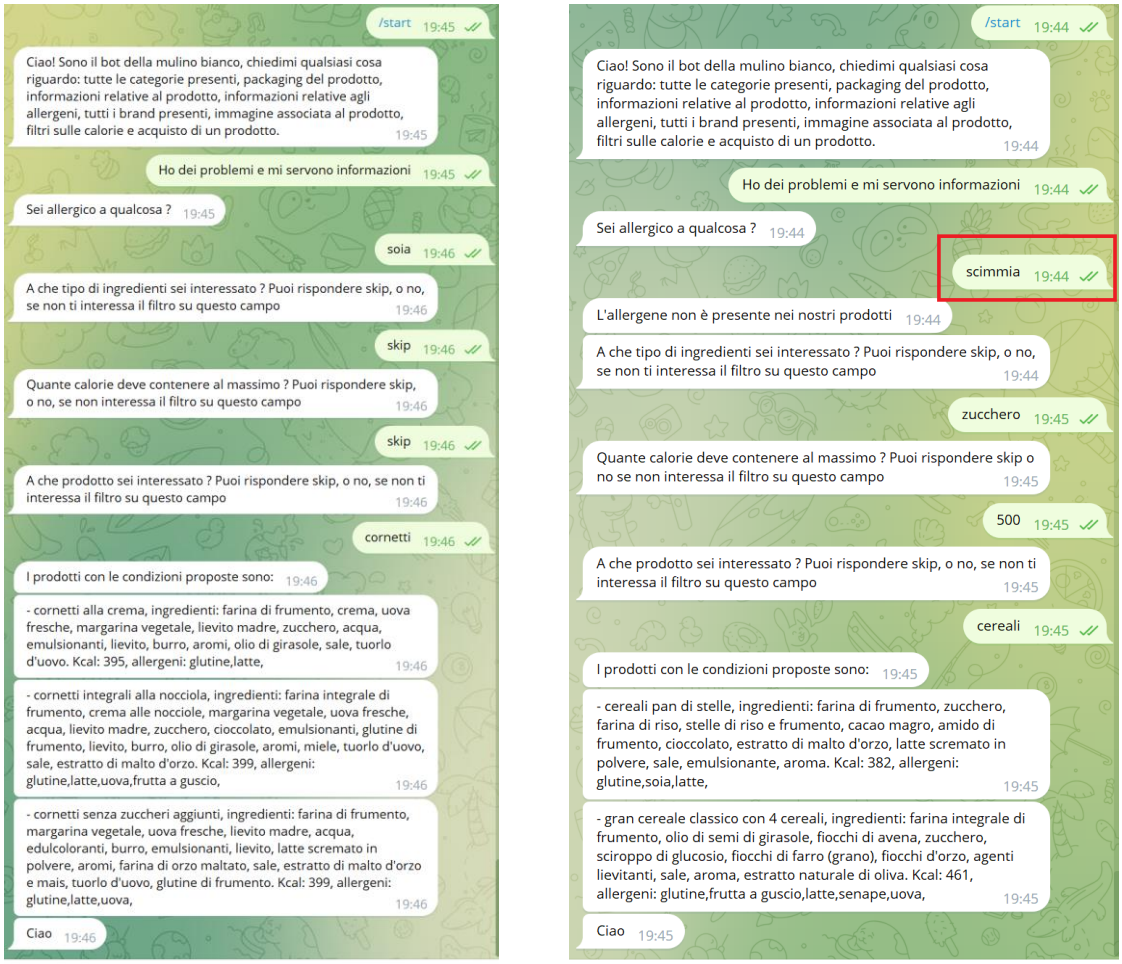


Figura 4.7 - filtro sulle intolleranze e kcal con check sulle intolleranze

4.2.6 Visualizzare i vari brand presenti e i relativi prodotti

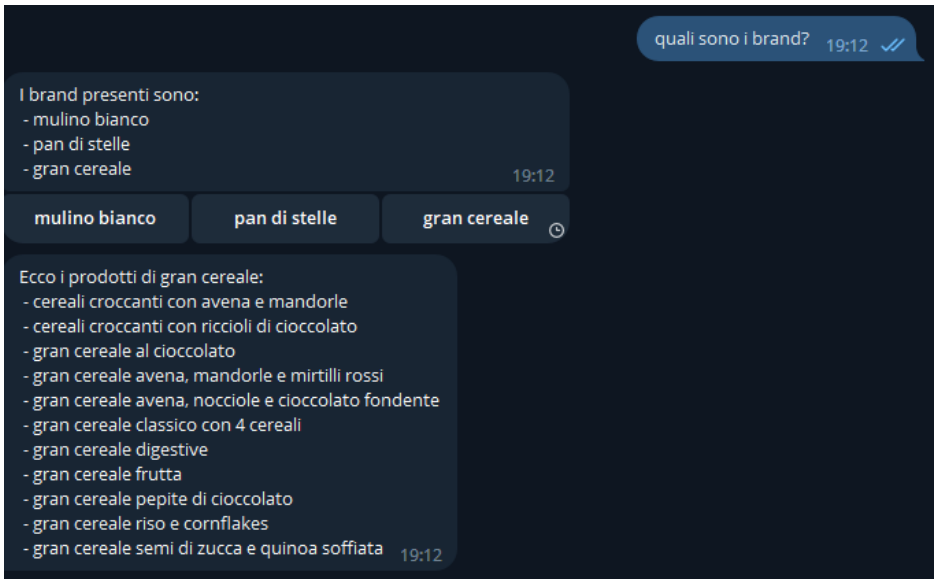


Figura 4.8 - Brand presenti e relativi prodotti

4.2.7 Visualizzare i prodotti in relazione alle calorie

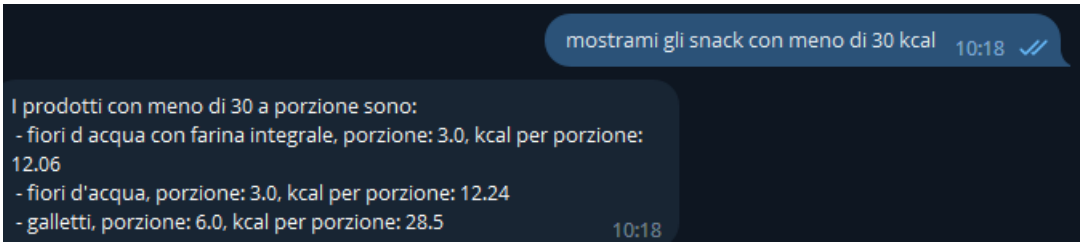


Figura 4.9 - snack con meno di 30 kcal

4.2.8 Visualizzare l'immagine di un prodotto

Figura 4.10 - Immagine tegolino

4.2.9 Acquistare un dato prodotto

Figura 4.4 - Acquisto prodotto con check sul prodotto inserito dall'utente

5 Conclusioni e sviluppi futuri

Si è utilizzato il framework RASA (AI) per la realizzazione di un chatbot come supporto al cliente, nella fattispecie un chatbot che guidi l'utente nella scelta dei prodotti fra quelli venduti dall'azienda Mulino Bianco.
Le funzionalità implementate (descritte nel Capitolo 3 di tale relazione) del chatbot sono rivolte principalmente ai prodotti della Mulino Bianco e di alcune delle sue controllate.
Si potrebbe pensare come un possibile sviluppo futuro di ampliare il database dei prodotti, per estenderlo pure a quelli di altre aziende oltre che creare altre funzionalità più incentrate sulle informazioni nutrizionali.

Link al repository GitHub:
https://github.com/Simone-Scalella/ChatBotRasa