Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

SmartPantry: La dispensa intelligente

Relatore:
Dott.
Federico Montori Correlatore:
Dott.
Luca Sciullo

Presentata da:
Gabriele Fogu

II Sessione
Anno Accademico 2021/2022

Sommario

SmartPantry è un applicazione per Android che si pone come obiettivo quello di rendere semplice e pratica la gestione virtuale delle dispense degli utenti. Oltre a questo implementa un recommender system dedicato al suggerimento di ricette adatte ai prodotti contenuti nella dispensa, per farlo l'algoritmo si avvale della distanza di Damerau-Levenshtein per eseguire Natural Language Processing in modo tale da interpretare gli ingredienti delle dispense degli utenti e poterli mappare ad una collezione di ingredienti mantenuti in un database remoto.

All'interno di questo elaborato andremo ad analizzare i dettagli di progettazione ed implementativi di SmartPantry e degli algoritmi che la sostengono ponendo particolare attenzione agli aspetti qualitativi degli algoritmi di NLP e raccomandazione raccogliendo dati sufficienti a trarre conclusioni oggettive sulla precisione ed efficacia dei suddetti. Nell'ultimo capitolo vedremo come nonostante la presenza di margini di miglioramento, come versione 1.0, gli algoritmi abbiano restituito dei risultati più che discreti.

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Ci troviamo nell'era dell'informazione: i sistemi informatici di comunicazione costituiscono il fondamento delle società industrializzate attorno al globo, le persone che vivono all'interno di queste società sono abiutate ad utilizzare mezzi tecnologici di ogni tipo come smartphone e computer e ad avere accesso alle informazioni di cui hanno bisogno in qualsiasi momento. Quasi tutti hanno uno smartphone e lo utilizzano per svolgere qualsiasi attività possa essere svolta online o comunque con l'ausilio della tecnologia: effettuare un bonifico, comunicare con le persone che conoscono, prenotare un treno o ascoltare musica sono solo alcuni dei tantissimi servizi che un applicazione può fornire. Questo non è un fenomeno sconosciuto ad istituzioni pubbliche e private che investono risorse per sviluppare applicazioni che migliorino la qualità dei servizi che offrono. Va da se che oltre a soddisfare necessità preesistenti si cerca di sviluppare applicazioni che offrano servizi di cui le persone non sanno di avere bisogno. I fattori di successo per queste applicazioni sono vari e cambiano in base a innumerevoli aspetti come l'area geografica di rilascio, il mercato di riferimento, le tecnologie impiegate, i *competitors* già presenti sul mercato e via dicendo.
Nonostante il mercato delle applicazioni mobili possa sembrare saturo ci sono sempre delle idee innovative da sviluppare o applicazioni già esistenti con le quali competere sviluppando un'app che possa essere "migliore".
Nasce così SmartPantry.
All'interno di questo trattato verranno analizzate le motivazioni dietro lo sviluppo di SmartPantry e le scelte implementative e di progettazione che

sono state compiute durante il processo di sviluppo. Il **primo capitolo** è un'*overview* sulle tecnologie impiegate nello sviluppo: perché sono state impiegate, in che maniera e con quale approccio. Vengono analizzati gli aspetti, rilevanti ai fini dello sviluppo, del sistema operativo **Android** e della scienza dietro gli algoritmi di **Natural Language Processing** e dei **Recommender Systems** e viene spiegato come questi algoritmi verranno valutati tramite ***crowdsourcing***.

Nel **secondo capitolo** si passa ad una analisi approfondita delle scelte implementative e dell'architettura (**MVC**) dell'applicazione. Si esaminano macroscopicamente la struttura e la logica delle basi di dati e delle API che l'applicazione utilizza, per poi, nel **terzo capitolo**, analizzare a livello microscopico scelte e dettagli di progettazione. Viene eseguita l'analisi della parte *frontend* e di tutte le *features* che l'applicazione implementa e viene discussa la parte *backend*, dunque gli algoritmi di **NLP** e **Recommender System**, anche tramite l'analisi del codice scritto.

L'**ultimo capitolo** è dedicato all'analisi dei dati raccolti, questi dati vengono riportati e vengono tratte le conclusioni finali sull'efficacia e la qualità degli algoritmi implementati.

Capitolo 1

Stato dell'arte

1.1 SmartPantry - Nativa Android

I sistemi operativi per dispositivi mobili determinano le funzionalità, la sicurezza e le *features* di cui il dispositivo dispone. Attualmente i due competitor principali nel mondo dei sistemi operativi per dispositivi mobili, sono **Google** con **Android** e **Apple** con **iOS**. Nello sviluppo di applicazioni mobili la prima scelta che bisogna affrontare è decidere se si vuole sviluppare un'applicazione **nativa** oppure **ibrida**.

1.1.1 *Perché un'applicazione nativa?*

Mentre le applicazioni ibride sono sostanzialmente delle applicazioni web contenute da un *wrapper* nativo, le applicazioni native vengono sviluppate utilizzando il linguaggio di programmazione specifico del sistema operativo in uso (**Objective C o Swift** per iOS e **Java** per Android), il che consente un controllo capillare delle funzionalità di sistema (Notifiche, Fotocamera, GPS e in generale tutte le funzionalità *built-in*). Un'applicazione che viene sviluppata in un ecosistema nativo, seguendo le linee guida tecniche e riguardanti la *User Experience*, dimostra avere performance migliori e porta l'utente ad avvertire come "corretto" il design generale del software. In gergo il "*feels right*", ovvero la sensazione che l'estetica e le interazione *in-app* siano coerenti con le altre applicazioni native del dispositivo. Il vantaggio che si trae da un'applicazione che rispetta questo canone è che l'utente finale troverà più immediata la navigazione nell'app. e imparerà più velocemente ad utilizzarla. [1]

1.1.2 *Perché Android?*

Android è il sistema operativo **Open Source** per smartphone e tablet più diffuso al mondo, secondo *Stat Counter (Azienda che si occupa di analisi del traffico web)* ad oggi Android alimenta il 71.21% dei dispositivi mobili presenti sul mercato [2], rendendo di fatto lo sviluppo nativo ed esclusivo per Android una buona alternativa allo sviluppo ibrido.

Di seguito viene riportato un grafico che mostra l'andamento della popolarità dei sistemi operativi per dispositivi mobili dal 2012 al 2022. Come è evidente gli attori principali del mercato sono sempre stati iOS e Android e ad oggi, Android è il leader indiscusso in termini di *market share*.

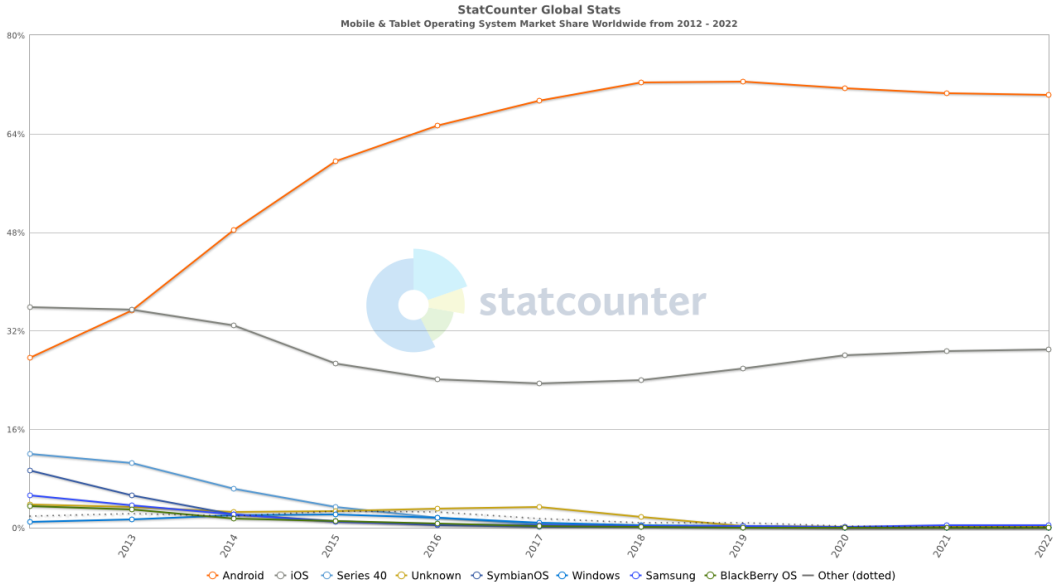

Figura 1.1: Mobile & Tablet Operating System Market Share Worldwide (2012-2022)

1.2 Natural Language Processing

L'interpretazione del linguaggio naturale - o **N**atural **L**anguage **P**rocessing (NLP) - è un ramo delle Intelligenze artificiali che consiste nell'interpretazione e analisi del linguaggio umano in modo tale da renderlo computabile e dunque rappresentabile e comprensibile alle macchine.

Natural Language Processing (NLP) is a tract of Artificial Intelligence and Linguistics, devoted to make computers understand the statements or words written in human languages. Natural language processing came into existence to ease the user's work and to satisfy the wish to communicate with the computer in natural language [3]

Il NLP è largamente utilizzato nell'ambiente dello sviluppo software ad esempio come strumento di *Email Spam Detection*, per l'estrapolazione di informazioni, per eseguire *Fake news Detection* [4], e per molti altri scopi.

Un algoritmo particolarmente utilizzato nel NLP è quello che calcola la **distanza di Damerau-Levenshtein** fra una parola **A** ed una parola **B**. L'idea alla base dell'algoritmo è quella di trovare il minimo numero di modifiche da apportare alla stringa A per renderla uguale a B. Per calcolare la distanza di Demerau-Levenshtein viene considerato il set di operazioni costituito da:

1. Sostituzione di una lettera in A
2. Inserimento di una lettera in A
3. Cancellazione di una lettera in A
4. Trasposizione di due lettere adiacenti in A

Nel caso in cui si considerino valide solo le operazioni di sostituzione, inserimento e cancellazione, si parla di **distanza di Levenshtein**.

Spelling error correction, data clustering and data mining, comparing packet traces, quantifying the similarity of DNA/RNA/protein sequences, gene finding, and gene function prediction are some of the applications of the DL distance. [5]

L'impiego più comune della distanza di Damerau-Levenshtein la *spelling correction*: viene calcolata la distanza minima fra la parola di cui si vuole fare

spelling correction con parole simili prese da un dizionario e si scelgono se presenti le parole con distanza 1. Damerau dimostrò che l'80% degli errori tipografici hanno distanza 1 dalla parola originale [6]. Viene ancora oggi fatta ricerca per migliorare gli algoritmi di spelling correction basati su distanza di DL e ad oggi l'algoritmo più veloce conosciuto è quello di **Lawrence e Wagner** [7] che utilizza tempo e spazio lineari nella dimensione delle due parole in analisi.
Detto questo però, la distanza di DL viene impiegata anche nel *data clustering* [8], nella biologia per calcolare la similarità fra due sequenze di proteine [9] ma anche, come nel caso di SmartPantry, per estrapolare informazioni sulla base della similarità fra due parole [10].

1.3 Recommender Systems

I Recommender Systems vengono impiegati principalmente nell'*e-commerce* per prevedere il gradimento degli utenti rispetto a prodotti per cui non hanno mai espresso interesse in modo tale da poterne consigliare l'acquisto.
Più formalmente:

[...] The recommender problem can be interpreted as determining the mapping (c, i) → R where c denotes a user, i denotes an item, and R is the utility of the user being recommended with the item. Items are then sorted by utility and top N items are presented to user as recommendation. [11]

Il valore R non rappresenta necessariamente il ritorno economico per l'azienda che impiega il Recommender System, ma può essere un indicatore dell'efficacia della raccomandazione eseguita.

All'interno di SmartPantry ad esempio R è strettamente legato all'identificazione degli ingredienti: se gli ingredienti dell'utente vengono identificati correttamente, anche le ricette consigliate saranno adeguate alla disponibi-

lità di ingredienti. Oltre a questo l'efficacia delle raccomandazioni è definita anche dal numero di ingredienti necessari per la ricetta in relazione a quelli disponibili all'utente.
SmartPantry, inoltre, utilizza un sistema di raccomandazione con filtraggio *Content-Based*, ovvero, che pone **enfasi sugli attributi degli oggetti** nello spazio di ricerca piuttosto sulle **preferenze espresse** in precedenza dall'utente come accade invece nel *Collaborative filtering* [12].

1.4 Valutazione degli algoritmi

E' necessario verificare l'efficacia degli algoritmi di NLP e di raccomandazione, per farlo è necessario raccogliere dati sulla correttezza dell'interpretazione degli ingredienti e sull'adeguatezza delle ricette consigliate agli utenti. Questi dati vengono richiesti proprio agli utenti in ciò che viene chiamato *crowdsourcing*: un processo di reperimento delle informazioni necessarie ad uno scopo, nel nostro caso la valutazione degli algoritmi, all'interno di una folla, ovvero un gruppo di persone le cui caratteristiche, il numero, l'eterogeneità e le conoscenze dipendono dall'obiettivo dell'iniziativa per cui si fa crowdsourcing [13]. A questa folla viene richiesto di completare delle *task*, nello specifico, in questo caso verrà richiesto alla folla (il più eterogenea possibile) di svolgere delle operazioni sull'applicazione ed infine di inviare le proprie valutazioni sempre tramite l'app.

1.4.1 *Valutazione - Algoritmi NLP*

Solitamente nella valutazione di un sistema di NLP si utilizzano i valori di *Precision, Recall, F-score (o F-measure/F1-measure)* e *Accuracy* come misure della qualità dell'algoritmo [14].

- Precision: Dato un insieme di risultati (parole interpretate), la **precisione** è il valore percentuale che indica quanti di questi risultati sono

corretti rispetto all'output atteso, il valore complementare indica i **falsi positivi**.

- Recall: In italiano può essere tradotto come **sensibilità**, ed è, dato un insieme di risultati, il valore percentuale che indica quanti risultati corretti vengono trovati, l'opposto della *Recall* è il valore che indica i **falsi negativi**.

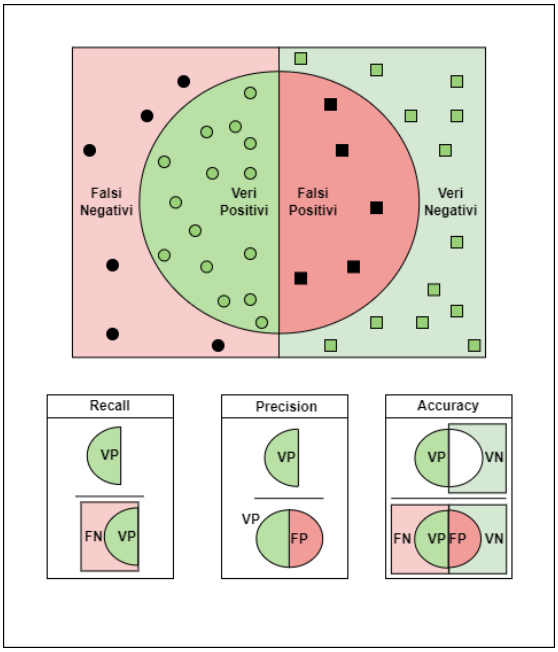

Figura 1.2: Relazione fra Falsi/Veri Positivi/Negativi e Precision, Recall e Accuracy.

- F-score: E' un'armonizzazione dei valori *Precision* e *Recall* di un sistema che viene calcolato come

$$Fscore = 2 \times \left(\frac{Precision \times Recall}{Precision + Recall} \right)$$

Un valore moderatamente alto di questa misura può indicare uno sbilanciamento tra Precision (**P**) e Recall (**R**) senza però fornire indicazioni complete su quale dei due valori crei questo squilibrio; di conseguenza

l'utilizzo del F-score per determinare la qualità di un sistema è oggetto di critiche.
Nella maggior parte dei sistemi migliorare uno dei due valori fra P ed R senza influenzare negativamente l'altro non è banale, per questo esistono delle variazioni della formula che attribuiscono più valore ad una misura piuttosto che all'altra (*F2-measure* maggior peso alla Recall e *F0.5-measure* maggior peso alla Precision).

- Accuracy: L'**accuratezza** di un sistema è un valore di classificazione **binario** che indica se un output è corretto oppure no. Si calcola come

$$Accuracy = \left(\frac{V.Positivi + V.Negativi}{V.Positivi + V.Negativi + F.Positivi + F.Negativi} \right)$$

Per ottenere i dati che ci servono a definire la Precision, la Recall e l'Accuracy dell'algoritmo di riconoscimento dei prodotti in SmartPantry è stato implementato un sistema di voto destinato agli utenti finali.

Nel contesto dell'applicazione sono definiti:

- **Vero Positivo**: Un match che l'utente giudica adeguato.
- **Falso Positivo**: Un match che l'utente non giudica adeguato.
- **Vero Negativo**: Un match che non viene consigliato e non è adeguato.
- **Falso Negativo**: Un match adeguato che non viene suggerito all'utente.

Nel capitolo 4 vengono analizzati i dati raccolti e vengono calcolati i valori appena descritti.

1.4.2 *Valutazione - Recommender Systems*

La qualità di un sistema di raccomandazione è definita tramite due parametri: *Accuracy* e *Coverage* [15]. L'Accuracy è definita, parallelamente

al caso del NLP, come il rapporto fra **raccomandazioni corrette** e **raccomandazioni possibili totali**. La Coverage invece indica la frazione di oggetti nello spazio di ricerca che il sistema è in grado di inserire in una raccomandazione. Per com'è strutturato il Recommender System di SmartPantry la Coverage non rappresenta un valore di interesse in quanto sotto le giuste condizioni tutti gli oggetti dello spazio di ricerca possono essere consigliati all'utente, di conseguenza il valore che verrà utilizzato per valutare la qualità del Recommender System di SmartPantry è l'Accuracy. L'accuracy può essere calcolata tramite metriche **statistiche** o di ***decision support*** [16].

- *Statistical Accuracy Metrics*: E' la metrica che compara il *rating* previsto con quello assegnato dall'utente. Il *Mean Absolute Error* (MAE) [17], ovvero la media dei rapporti tra il valore assoluto degli errori di previsione è la metrica statistica più utilizzata è viene calcolata come segue [18]:

  $$\frac{1}{N}\sum_{u,i}|p_{u,i} - r_{u,i}|$$

  dove $p_{u,i}$ è la previsione per l'utente u sull'oggetto i, $r_{u,i}$ è il rating effettivo ed N è la cardinalità dell'insieme delle predizioni.
  Più basso è il MAE, migliore è l'accuratezza del Recommender System. Verrà utilizzato il MAE per il calcolo dell'accuracy del R.S. di SmartPantry. Delle alternative valide al MAE sono il *Root Mean Square Error* (RMSE) e la *Correlation*.

- *Decision Support Accuracy Metrics*: Letteralmente "metriche di accuratezza del supporto alla decisione" sono metriche che aiutano gli utenti a scegliere oggetti che abbiano una qualità maggiore rispetto agli altri [19]. Alcune delle più popolari sono *Reversal rate*, *Weighted Errors*, *Receiver Operating Characteristics (ROC) e Precision Recall Curve (PRC)* oltre a *Precision, Recall* ed *F-measure* [15] di cui già abbiamo discusso.
  Questo tipo di metriche interpretano la previsione relativa ad un de-

terminato prodotto come il risultato di un output binario che indica se l'oggetto in analisi sia di qualità o meno.

1.5 Perchè SmartPantry?

SmartPantry nasce con lo scopo di provare ad utilizzare alcune delle tecnologie più moderne per rendere la vita più semplice alle persone. Di applicazioni per la gestione della propria dispensa ne esistono, sicuramente ne esisteranno di migliori in termini tecnici.
Ciò che manca, e che SmartPantry ambisce ad essere, è un applicazione versatile che ti permetta la gestione della dispensa e della spesa senza dover rinunciare all'utilizzo del linguaggio naturale e contemporaneamente fornendo un sistema di suggerimento di ricette per offrire all'utente un numero notevole di ricette sulla base di ciò che egli già possiede.
Ad oggi SmartPantry non è *finita*, ma bensì è *completa* alla sua versione 1.0. Di feature da aggiungere e migliorie da apportare ce ne sarebbero tantissime, ma per ora, vista la pienezza dei contenuti che offre, è giusto fermarsi ad analizzare lo stato dell'arte, raccogliere dati ed analizzarli, per comprendere le performance e l'efficacia del lavoro svolto. Nel capitolo seguente andrò ad analizzare l'architettura dell'applicazione descrivendone alcune scelte implementative e la logica che c'è dietro.

Capitolo 2

L'architettura di SmartPantry

In questo capitolo verrà discussa l'architettura delle principali componenti di SmartPantry ponendo particolare attenzione alla logica generale piuttosto che soffermandosi sui dettagli implementativi che verranno discussi nel **cap.3**. Il client di SmartPantry, durante il *lifecycle* dell'applicazione, interagisce continuamente con numerosi database (descritti nella sezione 2.1) tramite numerose API (descritte nella sezione 2.2) con scopi molto diversi tra loro: Il database remoto è interamente dedicato a ricette e match di prodotti ed alla raccolta dei feedback, fornisce API per invocare gli algoritmi di match e recommender system e per memorizzare i voti espressi.
Il database locale gestisce tutti i dati relativi ai prodotti di interesse per ogni utente. L'ultimo database, ovvero quello fornito dai titolari del corso, espone API di autenticazione e fornisce meccanismi per la visualizzazione e modifica della raccolta globale di prodotti aggiunti tramite l'applicazione. Questo definisce una separazione netta degli ambienti sulla base delle funzionalità che sostengono portando, oltre ad un alleggerimento del carico di lavoro dei server, ad una divisione logica che si è rivelata fondamentale durante la scrittura del codice e il *debugging* dell'applicazione.

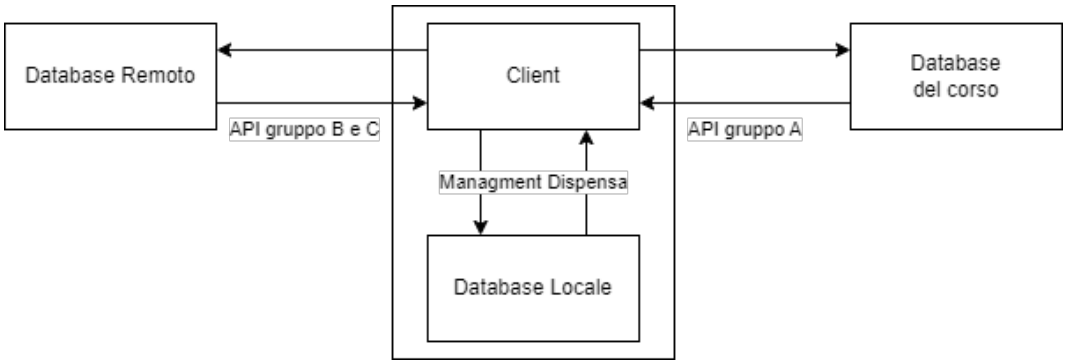

Figura 2.1: Schema architetturale dell'applicazione.

2.1 Database

SmartPantry dispone di un database locale nel quale salva informazioni riguardanti i prodotti e le ricette con cui l'utente interagisce e di un database remoto sul quale vengono mantenuti i dati riguardanti ricette ed ingredienti oltre che ad un database per la gestione degli utenti dell'applicazione creato dal Dott. *F. Montori* e dal Dott. *L. Sciullo* per il progetto di applicazioni mobili del corso di Informatica dell'Università di Bologna (anno di corso 2021/2022).

2.1.1 *Database locale*

Il database locale di SmartPantry è costituito da tre tabelle: *products*, *preferences* e *recipesRatings*.

- **Preferences**: La tabella adibita alla memorizzazione locale dei voti espressi dall'utente nei confronti dei prodotti inseriti da altri utenti. Lo scopo di questa tabella è quello di mostrare all'utente il voto che ha attribuito ad ogni scheda prodotto e di conseguenza ottenere un meccanismo per impedirgli di votare più volte. I voti vengono memorizzati sul **database del corso** è servono ad implementare un meccanismo di eliminazione delle schede prodotto di qualità scadente.

- **Products**: La tabella responsabile della memorizzazione di tutti i dati relativi ai prodotti che l'utente possiede o ha posseduto in dispensa. Distinguiamo i seguenti campi all'interno della tabella:
  - **ID**: Un identificativo associato al prodotto al momento dell'aggiunta lato server.*entries*.
  - **Barcode**: Il codice a barre del prodotto.
  - **productName**: Il nome del prodotto.
  - **productDescription**: La descrizione assegnata al prodotto.
  - **quantity**: Intero maggiore di zero che rappresenta la quantià di cui l'utente dispone.
  - **toBuyQuantity**: La quantità che l'utente intende acquistare di questo prodotto.
  - **expireDate**: La data di scadenza del prodotto, se presente.
  - **icon**: Il *path* dell'icona assegnata al prodotto.
  - **inPantry**: Booleano che indica se il prodotto è presente in dispensa oppure è solamente da mantenere memorizzato.
  - **favorite**: Booleano che identifica i prodotti che l'utente segna come preferiti.
- **RecipesRatings**: Questa tabella è stata aggiunta nella fase finale di sviluppo durante l'implementazione dei meccanismi di valutazione dell'algoritmo per il Recommender System. Si tratta di una tabella che per ogni voto che l'utente esprime nei confronti di una ricetta suggerita, memorizza la coppia $(rec\_id, rating)$ in modo da impedire all'utente di votare due volte senza dover eseguire controlli sul database remoto.

2.1.2 *Database remoto*

Il database remoto di SmartPantry è accessibile tramite un server dedicato che rappresenta il fulcro del sistema di *matching* dei prodotti agli

ingredienti e del sistema di raccomandazione delle ricette. E' un **database relazionale** costituito da tre tabelle:

1. **Ingredienti** (`ingredients`): L'oggetto "ingrediente" è caratterizzato esclusivamente dal nome e dall'id ad esso associato, La tabella ingredienti conta poco meno di 15.000 entries.

2. **Ricette** (`recipes`): La tabella che mantiene tutte le informazioni riguardanti le circa 28.200 ricette disponibili, eccezione fatta per gli ingredienti che le costituiscono. Ad ogni ricetta, oltre all'identificativo, è associata una folta collezione di informazioni:

   - **Nome**
   - **Tipo** di piatto che essa descrive: Bevanda, Antipasto, Primo, Carne, Pesce, Pollame, Contorno e Dessert.
   - **Ingrediente Principale** della ricetta
   - **Preparazione**
   - **Porzioni** per le quali è descritta la preparazione
   - **Note** come ad esempio il luogo d'origine, curiosità sulla ricetta e quant'altro.

3. **Ingredienti delle ricette** (`recipes_ingredients`): Ovvero la tabella che mette in relazione gli identificativi di ricette e ingredienti a coppie del tipo `(rec_id, ing_id)`. La tabella è costituita da 206.000 coppie, ogni coppia indica che alla ricetta identificata da `rec_id` è associato l'ingrediente identificato da `ing_id`.

Queste tabelle sono state generate tramite il *parsing* di file di testo contenenti ricette con procedure ed ingredienti, questi file di testo sono proprietà intellettuale di *G. Musilli* e sono disponibili con licenza *freeware/creative common*. Dopo essere state riempite sono state pulite programmaticamente e in alcuni casi manualmente tramite delle query SQL.

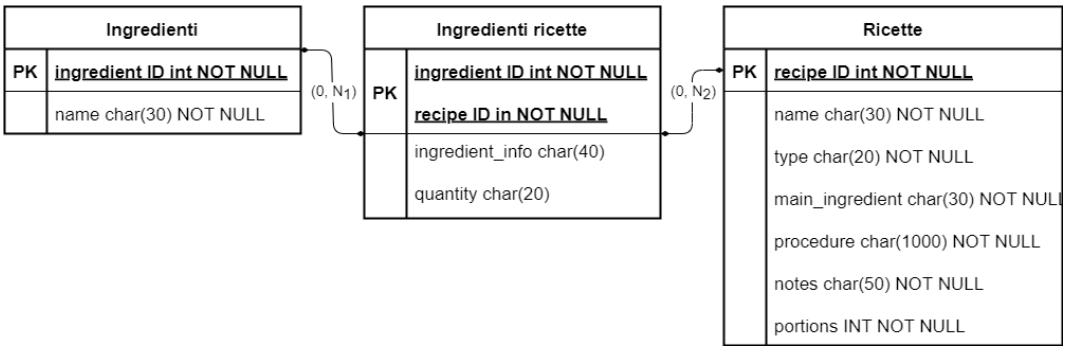

Figura 2.2: Schema relazionale del database remoto

Tabelle per la valutazione della qualità

Oltre alle tabelle appena descritte è stato necessario aggiungere allo stesso database due ulteriori tabelle dedicate alla registrazione dei voti espressi degli utenti nei confronti di match e raccomandazioni:

1. **match_votes**: Tabella adibita alla memorizzazione dei voti espressi dagli utenti nei confronti dei match eseguiti dall'applicazione sui prodotti inseriti in dispensa. La tabella è costituita da:

    - **id**: Identificativo intero auto-generato.
    - **u_id**: Identificativo dell'utente che ha espresso il voto.
    - **match_id**: Identificativo dell'ingrediente del database che è stato associato al prodotto dell'utente.
    - **match_name**: Nome dell'ingrediente associato al prodotto dell'utente.
    - **to_match**: Nome del prodotto per cui si è eseguito il match.
    - **is_correct**: Booleano che indica se il match è stato considerato corretto o meno dall'utente.

2. **recipes_ratings**: Tabella che contiene il voto attribuito dall'utente alla ricetta che gli è stata consigliata, la tabella è composta da:

   - **id**: Identificativo intero auto-generato.
   - **u_id**: Identificativo dell'utente che ha espresso il voto.
   - **rec_id**: Identificativo della ricetta raccomandata all'utente.
   - **expected_rating**: Voto previsto da parte dell'utente per la raccomandazione.
   - **rating**: Voto effettivo dato dall'utente per la raccomandazione.

2.2 API dell'applicazione

L'applicazione fa frequentemente uso di API che possono essere suddivise in tre macro-gruppi, ovvero: il gruppo di API definite nel server fornito per il progetto (**gruppo A**), quelle definite per match degli ingredienti e raccomandazioni di ricette (**gruppo B**) e quelle dedicate al raccoglimento dei *feedback* delle funzionalità sopracitate (**gruppo C**).

2.2.1 *API - Gruppo A*

Per lo svolgimento del progetto del corso di Applicazioni Mobili è stato fornito un database remoto accessibile tramite delle chiamate API che ha lo scopo di memorizzare i dati di accesso degli utenti e le schede prodotto da essi aggiunte, oltre che agli eventuali voti che essi decidono di dare alle schede prodotto.
Data l'estensiva documentazione fornita dai titolari del corso, nella descrizione delle API di questo gruppo si ricerca, oltre alla chiarezza, la concisione.

REGISTER: `POST /users`
Chiamata che accetta come parametri un `username`, una `e-mail` ed una `password` e ritorna informazioni sulla data di creazione, la versione criptata

della password l'`id` che il server ha assegnato all'utente e l'*echo* dei parametri ricevuti.

GET_USER_ID: `GET /users/me`
Chiamata che richiede e-mail e password di un utente come parametri e in caso di riscontro positivo sul server, restituisce l'id dell'utente associato alle credenziali.

LOGIN: `POST /auth/login`
Accetta `email` e `password` e ritorna un'`accessToken` necessario ad eseguire qualsiasi altra operazione sul server.

GET_PRODUCTS_BY_BARCODE: `GET /products?barcode=XXXXXX`
Accetta come parametro un barcode da inserire nel url al posto di `XXXXXX` e richiede l'`accessToken` nell'*header*. Ritorna un *array* di prodotti trovati corrispondenti al *barcode* e un `token` necessario a votare i prodotti fra i risultati o a postare una nuova scheda prodotto.

POST_PRODUCT_DETAILS: `POST /products`
Richiede l'`accessToken` nell'header della richiesta ed un **JSON** i cui campi devono comprendere il `token` e le informazioni necessarie alla scheda prodotto, ovvero: nome, descrizione e codice a barre. Ritorna un echo dei dati inviati, informazioni sulla data di creazione e l'`id` assegnato al prodotto

POST_PRODUCT_PREFERENCE: `POST /votes`
Richiede l'`accessToken` ed un JSON contenente il `token`, il `rating` attribuito dall'utente e l'`id` della scheda prodotto. Ritorna un echo dei dati inviati ed un **timestamp** del momento del voto.

2.2.2 *API - Gruppo B*

Le API del gruppo B, come detto, sono quelle che agiscono da interfaccia alle basi di dati necessarie ad elaborare i match e consigliare le ricette. Di seguito le descrizioni[1].

GET_AVAILABLE_RECIPES:
`GET /recipes/get_avail_recipes/:ingredients`
Questa API ritorna una lista di JSON contenenti le ricette composte da ogni campo che le descrive e in aggiunta due campi:

- `ing_list`: Lista di JSON che descrivono gli ingredienti relativi alla ricetta

- `score`: Questo score assume un valore ($0 < \texttt{score} \leq 1$) che indica la qualità della raccomandazione, ricette per cui si possiedono più ingredienti avranno $\texttt{score} \simeq 1$.

La chiamata a questa funzione va eseguita ponendo al posto di `:ingredients` i nomi (in linguaggio naturale, non necessariamente matchati) dei prodotti per i quali si vogliono cercare le ricette, separati dalla sequenza di caratteri "`$$$`".

GET_MATCHING: `GET /ingredients/get_matching/:name`
API che ritorna fino a 5 dei migliori match trovati nel database dato il nome di un prodotto in input. Il nome del prodotto per il quale si vogliono ottenere i match va inserito nel url al posto di `:name`.

[1]Nonostante non vengano utilizzate, sono state definite nel codice del server alcune API riguardanti match e raccomandazioni che sono logicamente conseguenti alle funzionalità che il server implementa, come ad esempio, l'API che ritorna tutte le ricette disponibili o tutti gli ingredienti disponibili. Dato che non vengono mai invocate, in questo trattato non verranno descritte.

2.2.3 *API - Gruppo C*

Gruppo di API costituito da tutte le funzioni necessarie a raccogliere i dati di valutazione delle *features* da parte degli utenti. Si tratta di API molto semplici che semplicemente immagazzinano i voti dati dagli utenti nelle tabelle descritte nel paragrafo precedente.

POST_MATCH_VOTE: `POST /ingredients/vote_match/:uid`
Richiede l'identificativo dell'utente al posto di `:uid` e come parametro del corpo un JSON che abbia come parametri:

- `toMatch`: Il nome per il quale si cercano i match.
- `voteList`: Una lista di oggetti JSON con parametri `id`, `name` e `isCorrect`, ovvero un booleano che esprime il voto dell'utente sulla correttezza del match (`isCorrect=true` per un match corretto).

POST_RECIPE_RATING: `POST /recipes/rate_recipe/:uid`
Richiede come parametri l'id utente al posto di `:uid` nel url e un JSON con i seguenti campi:

- `rating`: Il voto compreso tra zero e 5 inclusi a passi di 0.5.
- `expected`: Il voto calcolato come probabile per la ricetta suggerita.
- `rec_id`: L'id della ricetta per cui si sta esprimendo un voto.

Capitolo 3

Progettazione e implementazione dell'applicazione mobile

In questo capitolo vengono analizzate la progettazione dell'applicazione e le funzionalità principali che implementa. L'applicazione è divisa in sette attività che invocano a loro volta dodici frammenti diversi. Di seguito vengono esposti i ruoli delle componenti più importanti.
Spesso in questo capitolo verranno descritti metodi che fanno riferimento a delle costanti nella forma `Global.PROPERTY_NAME`, queste costanti sono definite all'interno del file `Global.java`.
Delle altra caratteristiche che verranno considerate sottintese durante la stesura di questo capitolo è che tutte le risorse di testo di SmartPantry sono disponibili in lingua italiana ed inglese e che tutte le chiamate HTTP e le query SQL sono eseguite all'interno di *thread* dedicati.

3.1 MVC Pattern

Il **Model-View-Controller** è un design pattern che descrive l'architettura di un'applicazione dividendo in tre aree il funzionamento della stessa:

1. **Model**: fornisce metodi di accesso e modifica dei dati
2. **View**: l'interfaccia grafica con la quale l'utente finale interagisce.
3. **Controller**: il legante fra le prime due componenti, riceve gli input da parte dell'utente, gli elabora e se necessario modifica lo stato del Model e della View.

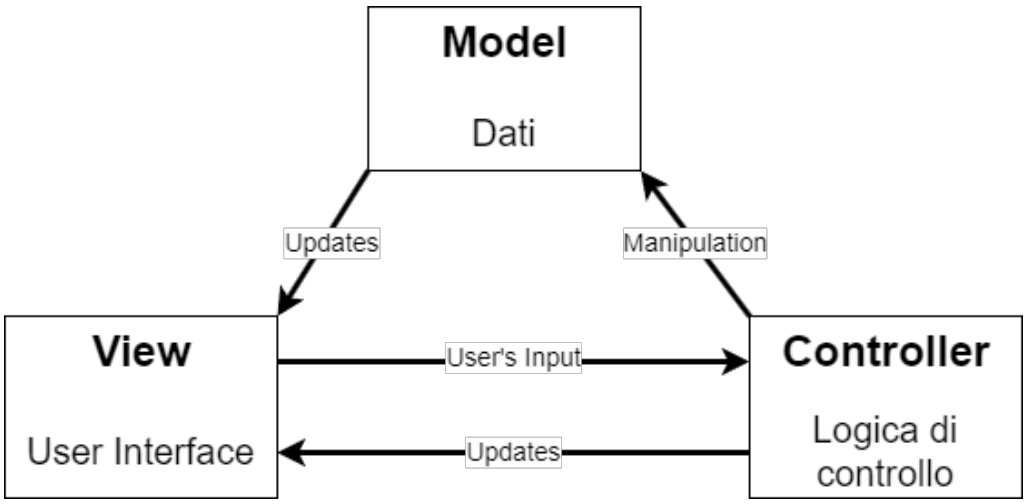

Figura 3.1: Model-View-Controller pattern.

L'architettura di SmartPantry segue il modello MVC che rende lo sviluppo dell'applicazione più rapido. Oltre a questo il codice è completamente modulare rendendo le modifiche e il *debugging* operazioni più semplici ed immediate.

3.1.1 *Global*

Prima di addentrarci nei dettagli implementativi che costituiscono l'architettura di SmartPantry andiamo ad analizzare il file `Global.java`.

All'interno di Global sono definiti:

- Gli *endpoint* delle API che l'applicazione utilizza (discussi nel capitolo precedente).
- La nomenclatura e quindi la gerarchia logica delle `SharedPreferences` dell'applicazione.
- Metodi statici necessari a più componenti dell'applicazione.
- *Tag* da associare ai fragment quando vengono aggiunti allo *stack*.
- ID per attività, canali di notifica e *request code* per la camera ed i canali di notifica.
- *Path* di locazione delle icone utilizzate per le ricette e per i prodotti.
- il formato data utilizzato dal database, ovvero `yyyy-MM-dd`.
- Il numero di giorni dopo il quale considerare scaduto il token.
- *Status* di login dell'utente.

Andiamo a vedere più nello specifico la struttura delle `SharedPreferences` e il corpo dei metodi definiti in *Global*.

Shared Preferences

Le `SharedPreferences` che l'applicazione utilizza sono impiegate per memorizzare informazioni riguardanti informazioni di ordinamento delle liste, sull'utente e di utilità:

1. `LIST_ORDER`: Informazioni legate all'ordinamento delle liste prodotti, dispensa e lista della spesa:
    - `ORDER`: La proprietà sulla base della quale ordinare, ad esempio nome, data di scadenza ecc.
    - `FLOW`: Direzione dell'ordinamento fra ascendente e discendente.

- `TEMP_ORDER`: Versione temporanea di `ORDER`.
- `TEMP_FLOW`: Versione temporanea di `FLOW`.
- `NOT_IN_PANTRY`: Visualizzazione dei soli elementi non in dispensa (Solo per la raccolta prodotti).

2. `LOGIN`: Informazioni relative alla modalità e allo stato del login dell'utente:
   - `STAY_LOGGED`: Opzione per mantenere l'accesso dell'utente.
   - `ACCESS_TOKEN`: L'`accessToken` dell'utente.
   - `CURRENT_SESSION`: Opzione per indicare che l'utente ha appena eseguito il login.
   - `VALID_DATE`: Data fino alla quale il token è da considerarsi valido.
3. `USER_DATA`: Credenziali dell'utente:
   - `EMAIL`
   - `PASSWORD`
   - `USERNAME`
   - `ID`
4. `UTILITY`: Informazione di utilità necessarie al funzionamento di alcune componenti:
   - `SESSION_TOKEN`: Token per l'interazione con il database remoto dei prodotti.
   - `NOTIFY_EXPIRED`: Booleano per indicare la necessità di inviare una notifica per prodotti scaduti.
   - `NOTIFY_FAVORITES`: Booleano per indicare la necessità di inviare una notifica per prodotti preferiti non presenti in dispensa.

Metodi globali

I metodi della classe `Global` sono *statici* ovvero associati alla classe stessa e non alle istanze, il che li rende utilizzabili senza dover inizializzare un'istanza della classe. Sono definiti quattro metodi all'interno della classe:

1. `checkConnectionAvailability`: funzione che viene invocata ogni qual volta un componente ha necessità di utilizzare internet per verificare che sia disponibile una connesione:

```
boolean checkConnectionAvailability(Context context) {
    ConnectivityManager connMgr = (ConnectivityManager)
        context.getSystemService(Context.
           CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo = connMgr.
       getActiveNetworkInfo();
    return (networkInfo != null && networkInfo.
       isConnected());
}
```

2. `isDateBeforeToday`: Metodo che verifica se la stringa rappresentante una data passata in input come parametro sia successiva alla data odierna.

3. `getNewValidDate`: Metodo che restituisce una stringa rappresentante la prossima data valida per il token di login, ottenuta come aggiungendo i giorni di validità del token alla data odierna.

```
String getNewValidDate() {
    ...
    try {
        c.setTime(sdf.parse(today));
        c.add(Calendar.DATE, LOGIN_TOKEN_VALID_DAYS);
        validDate = sdf.format(c.getTime());
    } catch (ParseException e) {
        ...
    }
    return validDate;
```

```
}
```

4. `changeDateFormat()`: Metodo per cambiare il formato di una data. Necessario per convertire il formato delle date del database remoto al formato locale all'applicazione utilizzato per presentare le date all'utente.
   Prende in input tre stringhe: una data, un formato di input e un formato di output e converte la data fornita da un formato all'altro.

3.2 Frontend - L'applicazione

L'attività visualizzata quando l'utente lancia l'applicazione è `ActivityMain`, che all'interno del metodo `onCreate` imposta gli *Event Handler* dell'interfaccia grafica, imposta la `RecylcerView` della dispensa, inizializza un *pool* di thread, apre un'istanza del database ed invoca la funzione `handleLogin()` che verifica lo stato di autenticazione dell'utente.
La maggior parte delle operazioni sui prodotti che l'utente può compiere su SmartPantry (es. Voto di un prodotto, aggiunta o eliminazione di un prodotto, ecc...) richiedono l'utilizzo di un `accessToken`, ovvero un token che viene inviato al client al momento dell'autenticazione. Gli `accessToken` hanno una durata limitata e di conseguenza, nel caso in cui l'utente abbia espresso la volontà che le proprie credenziali vengano ricordate, è necessario verificare la validità dell'`accessToken` ed eventualmente richiedere un nuovo token al server. Il metodo `handleLogin` come prima cosa tramite la funzione `status` identifica lo stato di accesso fra i seguenti:

- L'utente ha appena eseguito il login.
- L'utente non ha richiesto che le sue credenziali venissero ricordate o nessun utente ha mai eseguito il login sull'app.
- L'utente ha richiesto che le sue credenziali venissero ricordate ed il suo token d'accesso non è scaduto.

- L'utente ha richiesto che le sue credenziali venissero ricordate ed il suo token d'accesso è scaduto.

Le condizioni appena definite comportano uno dei seguenti codici e le rispettive azioni da eseguire.

1. SHOW_LOGIN: All'utente viene imposto di eseguire il login tramite l'attività `ActivityLogin` per continuare ad utilizzare l'applicazione.

2. REQUEST_TOKEN: L'applicazione provvede a richiedere un nuovo token di accesso per sostituire quello scaduto.

3. OK: Nessuna azione da eseguire, l'utente può continuare.

Il codice `SHOW_LOGIN` prevede il lancio di un `Intent` verso l'attività `ActivityLogin` e la chiusura immediata dell'attività principale, in modo da impedire all'utente di compiere qualsiasi azione senza prima aver effettuato il login.

L'operazione di rinnovo del token (codice `REQUEST_TOKEN`) viene eseguita su un thread a parte dal metodo `updateToken()` nel quale viene eseguita la API `LOGIN` tramite le credenziali salvate in precedenza in modalità privata all'interno delle shared preferences con chiave `Global.LOGIN`.

Di seguito il corpo di `handleLogin()`.

```
private void handleLogin() {
    SharedPreferences sp = getSharedPreferences(Global.LOGIN,
        MODE_PRIVATE);
    int status = status(sp);
    switch (status) {
        case Global.LOGIN_STATUS_SHOW_LOGIN:
           Intent login = new Intent(this, ActivityLogin.
              class);
           startActivity(login);
           this.finish();
           break;
        case Global.LOGIN_STATUS_REQUEST_TOKEN:
            threadPool.execute(this::updateToken);
```

```
                break;
            case Global.LOGIN_STATUS_OK:
            default:
                break;
        }
    }
```

Come detto, con il codice `OK` non è necessario compiere nessun'altra operazione e quindi il *lifecycle* dell'applicazione può proseguire.

Una volta gestito l'accesso dell'utente, per ultimo, viene inizializzato il `JobIntentService IntentServiceSetAlarm`, che si occupa di impostare la verifica periodica della presenza di prodotti in scadenza all'interno della dispensa e il controllo sui prodotti preferiti dell'utente in modo da avvisarlo ogni 4 giorni nel caso in cui uno di essi non sia in dispensa. Ciò avviene tramite i metodi `setFavoritesAlarm` e `setExpiringAlarm`, all'interno del metodo `onHandleWork` del `JobIntentService`, che periodicamente lanciano degli `Intent` per il controllo sui prodotti da notificare a dei `BroadcastRecevier` dedicati: `BroadcastReceiverExpireCheck` e `BroadcastReceiverFavoritesCheck` i quali eseguono delle query e in caso di positività al controllo inviano al dispositivo dell'utente le notifiche.

L'interfaccia che `ActivityMain` espone all'utente è abbastanza semplice e gli permette di navigare verso tutte le schermate dell'applicazione in maniera facile e veloce.
E' composta da:

- Un **Header** che comprende una barra di ricerca per navigare e filtrare i prodotti all'interno della propria dispensa ed un pulsante per inserire manualmente il codice a barre di un prodotto.
- La **dispensa** costruita tramite una `RecyclerView` che espone all'utente i prodotti dandogli la possibilità di gestirli.
- Un **Footer** che fornisce quattro tasti di navigazione per :

- **Raccolta Prodotti**: Attività impiegata per la visualizzazione della raccolta prodotti dell'utente (`ActivityShowProducts`).
- **Lista della Spesa**: Attività per la gestione della lista della spesa (`ActivityShoppingList`).
- **Ricettario**: Attività per la raccomandazione di ricette (`ActivityRecipes`).
- **Impostazioni**: Navigation Drawer per la gestione delle impostazioni quali log-out, eliminazione dei dati salvati e gestione notifiche.
- **Scan Barcode**: Attività preposta alla scansione dei barcode tramite la fotocamera che consente all'utente di ricercare un prodotto inquadrando il suo codice a barre (`ActivityCamera`).

Aggiunta di elementi alla dispensa

L'applicazione è stata concepita per utilizzare i codici a barre dei prodotti come identificativi degli stessi, di conseguenza è necessario fornirne uno per aggiungere un prodotto in dispensa. SmartPantry prevede due modalità di inserimento di un codice a barre:

1. Aggiunta Manuale: il codice a barre va inserito all'interno di un campo di testo che viene mostrato in una finestra di dialogo alla pressione del tasto **Aggiungi Prodotto** posto nell'header della dispensa.

2. Aggiunta Automatica: tramite il pulsante **Scan Barcode** viene aperta la fotocamera del dispositivo, a questo punto basterà scattare una fotografia (temporanea, che quindi non verrà salvata sulla memoria del dispositivo) e una finestra di dialogo mostrerà il risultato della scansione.

Una volta fornito il codice a barre all'applicazione, esso viene inviato al server che risponde con una lista di tutte le schede prodotto che vengono mostrate tramite `FragmentBarcodeListProducts` ed un token per eseguire azioni

come l'aggiunta, l'eliminazione (possibile solo nel caso in cui il prodotto sia stato aggiunto dall'utente stesso) o il voto di un prodotto. A questo punto l'utente può selezionare un prodotto già esistente o definirne uno nuovo da aggiungere alla propria dispensa.
Nel caso in cui venga inserito un barcode già presente fra i prodotti dell'utente l'applicazione proporrà all'utente, tramite `FragmentAlreadySavedBarcode`, una lista dei prodotti con lo stesso barcode e l'utente potrà scegliere se continuare con l'aggiunta del prodotto o se selezionare uno dei prodotti esistenti.

Aggiunta di un prodotto

Come già accennato, per svolgere operazioni sul database dei prodotti è necessario che l'utente sia in possesso dell'`accessToken`, un token che viene restituito al client dopo l'accesso all'applicazione e che va inserito nell'header delle richieste affinché il server le consideri valide. L'aggiunta del prodotto però richiede l'utilizzo di un altro token che è quello restituito dal utilizzo dell'API `GET_PRODUCTS_BY_BARCODE`.
Ciò definisce l'iter dell'aggiunta di un prodotto come segue:

1. L'utente accede all'applicazione ed ottiene l'`accessToken`.

2. Inserisce il `barcode` dell'oggetto che vuole aggiungere alla propria dispensa.

3. Viene eseguita `GET_PRODUCTS_BY_BARCODE` con parametri `barcode` e `accessToken`.

4. `GET_PRODUCTS_BY_BARCODE` restituisce il `token` assieme alla lista di prodotti corrispondenti al `barcode`.

5. L'utente aggiunge i dettagli del prodotto che desidera inserire ed utilizza il `token` per poter eseguire la `POST_PRODUCT_DETAILS` (o sceglie un prodotto già esistente).

6. Il prodotto è stato correttamente aggiunto alla dispensa.

Nel caso in cui fra il punto 4 ed il punto 5 l'utente voti un prodotto è quindi consumi il `token`, l'applicazione eseguirà automaticamente una nuova chiamata all'API, senza mostrare i risultati ottenuti, in modo da ottenere un nuovo `token` valido. Questo meccanismo è implementato tramite il booleano `show`, argomento della funzione
`getProductsByBarcode(String barcode, boolean show)` che esegue la chiamata HTTP al server per ricevere il `token` e la lista di prodotti da mostrare all'utente solo se `show` è uguale a `true`.
L'aggiunta del prodotto avviene tramite il fragment `FragmentAddProduct` che può essere invocato in modalità "nuovo prodotto" oppure "prodotto esistente". Questà modalità è definita da un booleano che viene passato come argomento del frammento è definisce se il frammento fa riferimento ad un prodotto che deve essere creato da zero oppure ad un prodotto scelto fra quelli della community. I nomi e le descrizioni dei prodotti della community non possono essere modificati mentre può essere modificata l'icona, la data di scadenza e la quantità. Il fragment espone l'interfaccia `onProductAddedListener` che viene implementata da `ActivityMain` così che l'attività possa rispondere adeguatamente all'aggiunta di un prodotto (ad esempio aggiornando la lista dei prodotti in dispensa).

Rating del match dei prodotti

Al fine di ottenere dati utili all'analisi qualitativa dell'algoritmo di matching dei prodotti è stata implementata un sistema di valutazione molto basilare subito dopo l'aggiunta di un prodotto alla dispensa: una volta aggiunto il prodotto, l'applicazione utilizza l'API `GET_MATCHING` ed ottiene n match (con $n <= 5$). Questi match vengono mostrati in una finestra di dialogo costituita da `FragmentBestMatch` tramite l'adapter `AdapterBestMatch`. L'adapter mostra il nome di ogni match affiancato da un `ToggleButton` il cui stato definisce la correttezza del match.
A questo punto l'utente seleziona i match che ritiene corretti e conferma la

Figura 3.2: Schermata della lista di prodotti ottenuti tramite il barcode

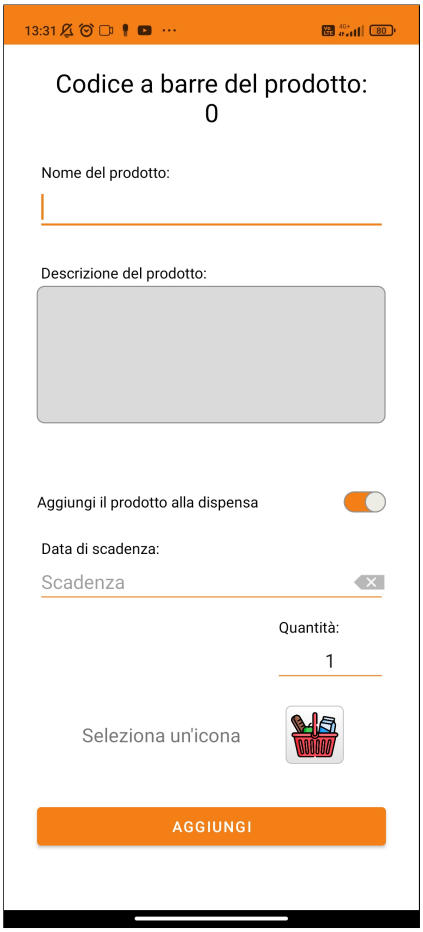

Figura 3.3: Schermata dell'aggiunta di un prodotto

scelta. Alla pressione del tasto di conferma viene costruito uno `JSONArray` costituito da oggetti contenenti **id**, **nome** e **correttezza** dell'ingrediente proposto come match. L'array viene a sua volta inserito in un `JSONObject` assieme al nome del prodotto appena inserito dall'utente `toMatch` (di cui si è eseguito il match). Il `JSONObject` appena definito viene inviato al server tramite l'API `POST_MATCH_VOTE`, assieme all'id utente, dove verrà inserito nel database all'interno della tabella dedicata.

3.2.1 *Dispensa*

La dispensa è costituita da una `RecyclerView` a cui viene associato un adapter (`AdapterPantryList`). Per mostrare all'utente tutti i prodotti della dispensa, l'attività `ActivityMain` mantiene una `ArrayList`, ovvero `pantryProducts`, di prodotti rappresentati tramite la classe `ProductPantryItem`, classe che mantiene tutte le informazioni necessarie alla gestione del prodotto da parte dell'utente.
Per riempire `pantryProducts` viene eseguita su un thread dedicato il metodo `populatePantryList`:

```
private void populatePantryList(String order, String flow) {
    pantryProducts.clear();
    Cursor cursor = database.getPantryProducts(order, flow);
    cursor.moveToFirst();
    while (!cursor.isAfterLast()) {
        pantryProducts.add(new ProductPantryItem(
            ...
        ));
        cursor.moveToNext();
    }
    cursor.close();
}
```

La funzione prende come parametri l'ordinamento e la direzione (ascendente o discendente) con i quali mostrare i prodotti, (l'ordine di default è "preferiti prima"), svuota la lista nel caso in cui contenga degli elementi e invoca il metodo `getPantryProducts(order, flow)` della classe `DBHelper` che esegue la query SQL e restituisce un cursore su cui iterare per riempire la lista. A questo punto l'adapter viene inizializzato ed assegnato e la dispensa diventa operativa.
Ogni scheda prodotto mostrata in dispensa espone un'interfaccia con cui l'utente può gestire il prodotto: modificarne la data di scadenza e la quantità, eliminarlo dalla dispensa o aggiungerlo alla lista della spesa o ai prodotti preferiti. Tutto ciò è gestito all'interno dell'adapter nel metodo

`onBindViewHolder`.

Ogni attività che presenta una recycler view, al fine di fornire un sistema di ordinamento e filtraggio, deve implementare l'interfaccia del fragment `FragmentFilters` "`onApplyFilters`", e ciò avviene per `ActivityMain`, `ActivityShoppingList` e `ActivityShowProducts`.

Ordinamento dei prodotti

L'ordinamento è gestito tramite il fragment `FragmentFilters` che prende come argomento in input un intero che indica l'attività chiamante, a seconda dell'attività viene mostrata o meno l'opzione per visualizzare solo i prodotti non presenti in dispensa (disponibile solo nell'attività `ActivityShowProducts`) e l'opzione per visualizzare i prodotti in scadenza per primi (assente nell'attività `ActivityShoppingList`. Il frammento espone l'interfaccia per i filtri e gli ordinamenti e quando l'utente conferma la sua selezione il frammento invoca un metodo che le attività chiamanti devono implementare che applica i filtri selezionati al recycler di riferimento. La selezione dei filtri viene inoltre salvata nelle SharedPreferences in maniera duratura o temporanea (a seconda che l'utente abbia selezionato "Save and Apply" o solo "Apply") in modo tale da presentare in futuro i prodotti con lo stesso ordinamento e gli stessi filtri selezionati nel caso in cui l'utente abbia espresso tale intenzione.

Ricerca nelle liste di prodotti

L'implementazione della ricerca avviene tramite il metodo `setSearchBox` che imposta il listener `onQueryTextChange(String query)` in modo tale da ricercare fra tutti i prodotti nella lista del recycler di riferimento dei prodotti il cui nome contenga la stringa `query`, quando ne trova uno aggiunge ad una lista dei risultati il prodotto. Questa lista viene poi adattata ad una `ListView` tramite l'adapter `AdapterSuggestionList` e in seguito, quando l'utente tocca il risultato a cui è interessato viene eseguito un con-

trollo sulla visibilità nel recycler dell'elemento cercato utilizzando la posizione dell'elemento e i metodi `findFirstCompletelyVisibleItemPosition()` e `findLastCompletelyVisibleItemPosition()` implementati dalla classe `LinearLayoutManager`.
Se l'elemento è completamente visibile verrà segnalato facendolo lampeggiare di giallo, altresì il recycler scorre automaticamente al risultato della ricerca in modo da renderlo completamente visibile.

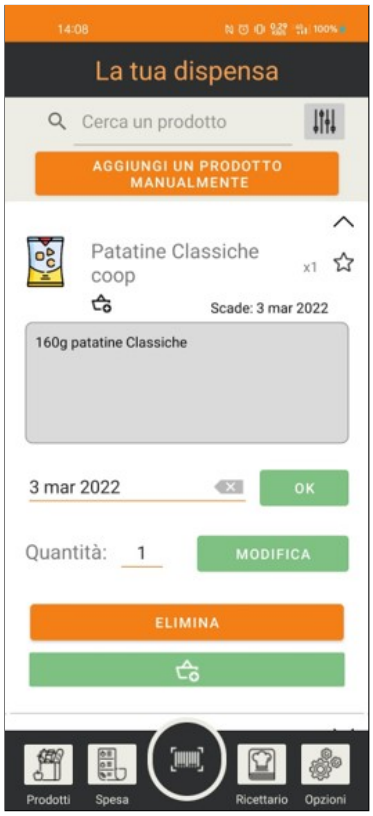

Figura 3.4: Schermata principale della dispensa,

Figura 3.5: Modale per ordinamento dei prodotti.

3.2.2 *Raccolta Prodotti*

La raccolta dei prodotti è un'attività pensata per dare all'utente la possibilità di gestire prodotti che ha aggiunto in passato alla sua dispensa e che non possiede più. Si tratta di un'attività molto semplice che espone all'utente una lista di **schede prodotto** con cui può interagire come nel caso della dispensa con la possibilità aggiunta di modificarne le icone. Lo scopo di questa attività è di fornire un meccanismo di memoria all'utente per i prodotti con cui ha interagito anche nel caso non li possieda più in dispensa.
Come per la dispensa l'adapter `AdapterProductsList` dispone i prodotti all'interno di un recycler.

3.2.3 *Lista della Spesa*

La lista della spesa è una delle featureprincipali di SmartPantry e da all'utente la possibilità di segnarsi i prodotti di cui ha bisogno assieme alla quantità della quale ha bisogno. Un prodotto può essere aggiunto alla lista della spesa tramite la Dispensa o la Raccolta Prodotti tramite un bottone dedicato che apre una finestra di dialogo nella quale è possibile scrivere la quantità che si desidera aggiungere alla lista della spesa. L'attività dispone la lista della spesa come una lista di schede prodotto `ProductShopping`, alleggerite dalla GUI usuale, presentate all'interno di una recycler view tramite l'adapter `AdapterShoppingList` come nel caso della dispensa. La lista può essere gestita selezionando con una check-box i prodotti che si vogliono eliminare in quanto "acquistati" o con uno *swipe* per eliminarli (senza quindi aggiungere la quantità segnata in dispensa).

3.2.4 *Impostazioni*

La gestione delle impostazioni avviene all'interno di una `NavigationView` definita nel file `activity_main.xml`. Il drawer può essere aperto con uno swipe sullo schermo da sinistra verso destra, o alternativamente, tramite la pressione del tasto dedicato nella schermata di Home. La schermata impo-

stazioni si presenta come un pannello laterale che copre 3/4 dello schermo e offre all'utente le seguenti quattro possibilità:

- **Logout**: Il pulsante di logout invoca il metodo`logout`, che modifica il valore della chiave `Global.STAY_LOGGED` all'interno delle Shared Preferences `Global.LOGIN` a `false`, in modo da imporre il login ad ogni avvio successivo dell'app. Dopodiché `MainActivity` viene terminata e viene mostrata la schermata di login.

- **Pulisci la dispensa**: La pressione di questo pulsante mostra una finestra di dialogo all'utente nella quale gli vengono spiegate le conseguenze dell'azione e gli viene chiesto di digitare la e-mail associata all'account per confermare la scelta. La pulizia della dispensa, eseguita tramite la funzione `askToClearPantry`, comporterà lo spostamento di tutti i prodotti dalla **Dispensa** alla **Raccolta prodotti** e di conseguenza perdere quantità e date di scadenza associate ai prodotti.
  L'operazione è irreversibile.

- **Gestione notifiche**: E' possibile gestire le notifiche che l'applicazione invierà al dispositivo direttamente dall'applicazione. Viene mostrata una finestra di dialogo tramite `FragmentNotificationManager` che consente all'utente di attivare e disattivare sia le notifiche dei prodotti in scadenza che le notifiche dei prodotti preferiti mancanti.

- **Elimina tutti i prodotti salvati**: Opzione ubicata nel footer del pannello, consente tramite il metodo `askToDeleteAllProducts` di eliminare tutti i prodotti salvati nell'app, anche in questo caso viene chiesto all'utente di confermare la scelta digitando la propria e-mail in una casella di testo.
  L'operazione è irreversibile.

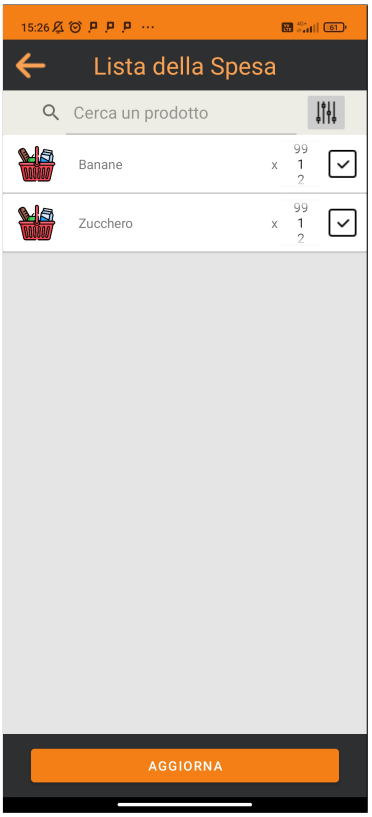

Figura 3.6: Schermata lista della spesa.

Figura 3.7: Schermata del drawer delle impostazioni.

3.2.5 *Scanner*

L'attività dedicata alla scansione: `ActivityCamera`. Implementata mediante l'utilizzo di **ML kit**, il *package* di Google dedicato al Machine Learning, che fornisce metodi e classi per la scansione della maggior parte dei codici a barre, in grado di riconoscere automaticamente il formato del barcode.
I formati supportati da SmartPantry sono: **Codebar**, **Code 39**, **Code 93**, **Code 128**, **EAN-8**, **EAN-13**, **ITF**, **UPC-A** ed **UPC-E**.
L'attività viene invocata da `ActivityMain` dopo aver verificato che l'utente abbia fornito l'autorizzazione per l'utilizzo della fotocamera.
Il pulsante funziona in modo da richiedere il permesso per l'utilizzo della

telecamera la prima volta che l'utente prova ad aprire lo scanner:

```
if (hasCameraPermission()) {
    enableCamera();
} else {
    requestPermission();
}
```

La funzione `hasCameraPermission()` verifica che il permesso per la camera dichiarato nel **Manifest** sia stato concesso, se così non è viene richiesta l'autorizzazione all'utente tramite `requestPermission()`, se invece il permesso è stato confermato `enableCamera()` lancia l'attività `ActivityCamera` aspettandone il risultato tramite il metodo `startActivityForResult(...)`. All'interno della `onCreate` di `ActivityCamera` viene inizializzata un'istanza del `ProcessCameraProvider`, **singleton** della libreria per la gestione della fotocamera **CameraX**, che viene poi legato al *lifecycle* dell'attività assieme ad un `CameraSelector` che specifica l'utilizzo della fotocamera frontale ed agli `UseCases` relativi all'utilizzo automatico del flash, alla `Preview` della fotocamera ed all'`ImageAnalyzer` che fornisce immagini accessibili dalla CPU per essere analizzate.

```
bindImageAnalysis(@NonNull ProcessCameraProvider
    cameraProvider) {
    ...
    int height = displayMetrics.heightPixels;
    int width = displayMetrics.widthPixels;
    ImageAnalysis imageAnalysis =
        new ImageAnalysis.Builder()
        .setTargetResolution(new Size(width, height))
        .setBackpressureStrategy(ImageAnalysis.
            STRATEGY_KEEP_ONLY_LATEST).build()
        .setAnalyzer(ContextCompat.getMainExecutor(this),
            ImageProxy::close);

    Preview preview = new Preview.Builder().build();
    preview.setSurfaceProvider(previewView.
        createSurfaceProvider());
```

```
    CameraSelector cameraSelector = new CameraSelector.
       Builder()
            .requireLensFacing(CameraSelector.
               LENS_FACING_BACK).build();
    imageCapture = new ImageCapture.Builder()
            .setFlashMode(ImageCapture.FLASH_MODE_AUTO)
            .build();

    cameraProvider
    .bindToLifecycle(this,
        cameraSelector, imageCapture, imageAnalysis, preview)
           ;
    ...
}
```

Quando l'utente preme il pulsante per scansionare il codice a barre, viene invocato il metodo `takePicture` su di un thread dedicato, in caso di cattura avvenuta con successo, l'`ImageProxy` dell'handler `onCaptureSuccess()` viene passato come parametro al metodo dell'attività `scanBarcode(ImageProxy imageProxy)` affinché ne venga analizzato il contenuto. L'analisi avviene tramite il `BarcodeScanner` che esegue il metodo `process()` sull'immagine ottenuta grazie all'`ImageProxy`. Se nell'immagine viene riconosciuto un codice a barre, questo viene mostrato all'utente per confermarne la correttezza, l'`ImageProxy` viene chiusa ed il fragment appena invocato restituisce ad `ActivityMain` il segnale `Activity.RESULT_OK` ed il codice a barre scansionato in un extra associato all'`Intent` di ritorno.

3.2.6 *Ricette*

L'attività per la visualizzazione delle ricette è `ActivityRecipes` la quale all'interno della `onCreate` invoca il metodo `getRecipes()`. L'intero corpo del metodo è eseguito su di un thread dedicato, in quanto vengono eseguite operazioni come chiamate HTTP e query. Per prima cosa viene invocato il metodo del database

`getPantryProducts(DBHlpr.COLUMN_PRODUCT_NAME,Global.DESC_ORDER)` che prende come parametro la proprietà su cui ordinare gli elementi e l'ordine (ascendente/discendente) e restituisce un cursore.
Viene poi definito un comparatore per la classe `Recipe` che ovviamente rappresenta le ricette. A questo punto viene costruita la stringa contenente i nomi dei prodotti in dispensa separati dalla sequenza `"$$$"` iterando sul cursore e viene eseguita la `GET_AVAILABLE_RECIPES`. La chiamata invia al client una lista di ricette in forma di `JSONObject` dai quali vengono prelevate le informazioni per generare degli oggetti `Recipe` con i quali viene generata una lista. La lista ottenuta viene ordinata tramite il metodo `Collections.sort(recipesList, comparator)` utilizzando il comparatore definito in precedenza in questo modo:

```
Comparator<Recipe> comparator  =
    (a, b) -> Double.valueOf((b.score - a.score)*1000).
       intValue();
```

che fa si che le ricette con una percentuale più alta di ingredienti posseduti vengano mostrate per prime.
A questo punto l'attività è in possesso di una lista ordinata sulla base degli ingredienti che si possiedono (in percentuale) che viene mostrata all'interno di una recycler view tramite l'adapter `AdapterRecipesList` il quale si occupa di legare l'handler che mostra il frammento `FragmentShowRecipe` all'evento della pressione di una delle ricette mostrate.
Il sistema di navigazione della lista di ricette è diverso da quello delle altre attività: per navigare le ricette, l'utente ha a disposizione una barra di ricerca associata al listener `onQueryTextChange` il quale ad ogni modifica del testo di ricerca crea una lista provvisoria contenente solo i prodotti il cui nome contiene il testo inserito. L'effetto finale è che la lista di ricette diminuisce di cardinalità per mostrare solo le ricette ricercate, nel caso in cui non vengano trovate ricette corrispondenti alla ricerca viene mostrato un messaggio al centro della view.

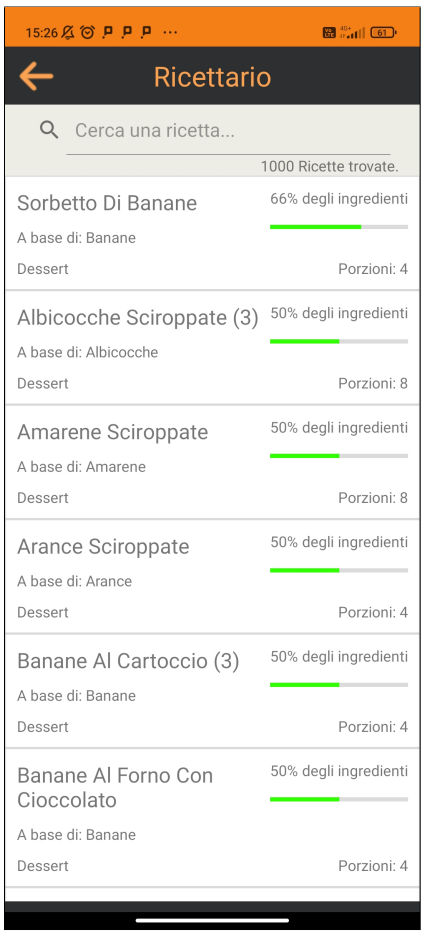

Figura 3.8: Schermata della lista delle ricette a seguito di una ricerca.

Figura 3.9: Schermata modale di una ricetta aperta.

Rating della raccomandazione ricette

Il frammento `FragmentShowRecipes` mostra tutte le informazioni riguardanti la ricetta e fornisce un sistema di rating nel quale viene chiesto di indicare il numero di prodotti che effettivamente si possiedono rispetto al numero totale degli ingredienti tramite una `SeekBar` con valore massimo pari al numero totale degli ingredienti. Alla creazione del frammento dall'event listener, associato alle carte ricetta dall'adapter, viene passata come argomento del frammento la percentuale di ingredienti posseduti prevista dall'algoritmo che viene poi mostrata come *score* della ricetta. Assieme allo

score, vengono mostrati il nome, tutti gli ingredienti della ricetta assieme ad eventuali note e quantità, il numero di porzioni ed il procedimento.
Una volta selezionato il voto da attribuire, questo viene trasformato in percentuale di completamento (1 ingrediente su 4 viene trasformato in 25%, quindi 25) e tramite la chiamata API `POST_RECIPE_RATING` viene inviato assieme al voto previsto dall'algoritmo, anch'esso tradotto in percentuale, e assieme all'id dell'utente che sta esprimendo il voto. Questi dati vengono poi inseriti nel database remoto all'interno della tabella `recipes_rating`.

3.2.7 *Autenticazione*

Registrazione

`ActivityRegister` è un'attività che espone una classica interfaccia di registrazione utente che richiede nome utente, email e password (con conferma della password). Per confermare la registrazione basta premere il pulsante con la scritta "Registrati" sulla parte bassa dello schermo il che attiverà l'handler dell'evento che controllerà lo stato dei campi di testo tramite la funzione `checkFields` la quale verifica che ogni campo del *form* di registrazione sia stato riempito in maniera adeguata. Nel caso in cui uno o più campi non siano stati riempiti correttamente ciò viene notificato all'utente tramite il metodo `setError(String errorText)` degli elementi `EditText`, altrimenti, se il form è stato riempito correttamente, viene eseguita la chiamata POST `REGISTER` per registrare l'utente.

Login

Dalla schermata di registrazione è possibile passare alla schermata di Login e viceversa. Per eseguire il Login in SmartPantry è necessario fornire il nome utente e la password ed è possibile selezionare un check-box per far si che le credenziali d'accesso vengano ricordate dall'applicazione. Quando l'utente preme sul tasto per effettuare il login, se entrambi i campi per le credenziali sono stati riempiti, viene lanciato un thread per eseguire la chiamata

API `LOGIN`. Se l'autenticazione va a buon fine, nell'handler della *response* alla chiamata viene eseguita la funzione `verifyUserIsTheSameOne(...)` alla quale vengono passati come parametri i dati appena inseriti dall'utente. La funzione ha lo scopo di verificare che l'utente che sta cercando di eseguire l'accesso sia lo stesso che l'aveva eseguito in precedenza (nel caso ce ne fosse uno). Per farlo viene confrontata la e-mail inserita con quella salvata nelle shared preferences dell'applicazione e a seconda dell'esito del controllo si rientra in uno dei seguenti casi:

- **L'utente è lo stesso**: Nel caso le e-mail coincidano viene eseguita la funzione `login` a cui viene passato il token ottenuto al momento dell'autenticazione, vengono aggiornate le shared preferences relative al login (`Global.LOGIN`) ed ai dati utente (`Global.USER_DATA`), viene richiesto e salvato l'id utente tramite l'API `GET_USER_ID` ed infine viene lanciata l'attività principale `ActivityMain` terminando l'attività corrente e di conseguenza la fase di login.
  Di seguito la funzione `login` per mostrare le `SharedPreferences` che vengono modificate:

```
private void login(String token, String password,
                   String email, boolean remember) {
    SharedPreferences.Editor loginEdit =
    getApplicationContext()
        .getSharedPreferences(Global.LOGIN, MODE_PRIVATE)
        .edit();

    //Edit Login SharedPrefs.
    loginEdit.putBoolean(Global.STAY_LOGGED, remember);
    loginEdit.putString(Global.ACCESS_TOKEN, token);
    loginEdit.putBoolean(Global.CURRENT_SESSION, true);
    loginEdit
        .putString(Global.VALID_DATE, Global.
            getNewValidDate());
    loginEdit.apply();

    //Get user ID
```

```
        threadPool.execute(()-> getUserID(email, password));

        //Edit UserData SharedPrefs.
        SharedPreferences.Editor userDataEdit =
        getApplicationContext()
            .getSharedPreferences(Global.USER_DATA,
               MODE_PRIVATE)
            .edit();
        userDataEdit.putString(Global.EMAIL, email);
        userDataEdit.putString(Global.PASSWORD, password);
        userDataEdit.apply();

        //Launch ActivityMain
        Intent main = new Intent(this, ActivityMain.class);
        startActivity(main);
        this.finish();
    }
```

- **L'utente *non* è lo stesso**: Se le credenziali inserite per eseguire l'accesso non sono le stesse salvate all'interno delle shared preferences significa che un utente diverso dall'utlimo che ha utilizzato l'app sta cercando di eseguire l'accesso.
  Data la necessità, dettata dall'architettura del server fornito per il progetto, di mantenere le informazioni legate ai prodotti posseduti dall'utente in locale, è necessario nel caso cambi l'utente eliminare la dispensa e la raccolta prodotti dell'utente sostituito. Per fare ciò viene invocata la funzione `askToDropTables(...)` tramite la quale viene mostrato un *dialog* all'utente nel quale gli vengono spiegate le conseguenze dell'accesso da un nuovo account. Nel caso in cui il nuovo utente confermi la scelta vengono svuotate da tutti i dati salvati le shared preferences e vengono pulite tutte le tabelle del database tramite i metodi `deleteSharedPreferences()` e `DBHelper.dropAllTables()`. A questo punto viene invocata la funzione `login()` descritta sopra e l'applicazione è pronta per essere utilizzata.

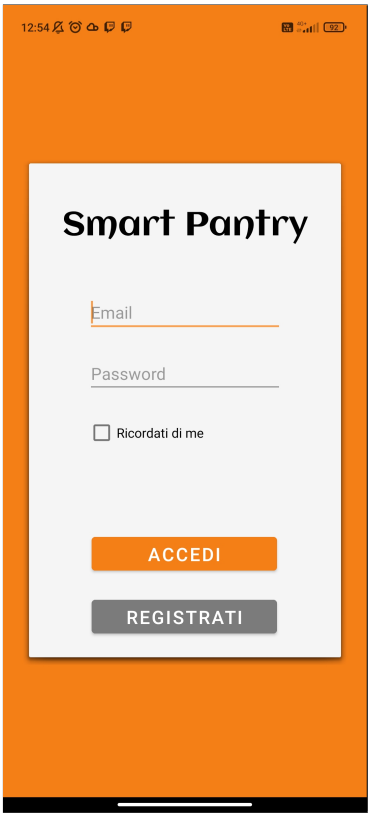

Figura 3.10: Schermata di accesso dell'app.

Figura 3.11: Schermata di registrazione dell'app.

3.3 Backend - Match e Raccomandazioni

Il backend dell'applicazione è programmato tramite il framework Javascript **Node.js**. Il server espone delle *route*, definite nella sezione dedicata alle API dell'applicazione (gruppo **B** e **C** par. 2.2.2 e 2.2.3). Tutte le operazioni sul database eseguite nel codice backend vengono eseguite tramite i metodi esposti dai DAO definiti per ogni tabella tramite le *promise* di **Bluebird**.

3.3.1 *Implementazione API gruppo B*

Per analizzare l'implementazione delle API in questione procediamo seguendo l'ordine con le quali vengono definite.
`GET_AVAIL_RECIPES`, come detto, riceve come parametro una lista di nomi di prodotti fornita dall'utente tramite la sua dispensa. Questi nomi devono essere interpretati e in un certo senso mappati dal linguaggio naturale (nel quale un nome può contenere un numero arbitrario di parole senza rilevanza ai fini del riconoscimento) al linguaggio riconosciuto dall'applicazione costituito da tutti i nomi di ingredienti presenti nel database ricette, ciò avviene tramite una funzione **asincrona** utilizzata anche da `GET_MATCHING` per eseguire il match vero e proprio fra prodotti client e ingredienti nel database. Nella funzione viene per prima cosa iterato sulla lista di nomi fornita dal client e viene inizializzato un vettore che verrà riempito con gli ingredienti che *matchano*.
A questo punto viene definito il vettore `toMatch` costituito dalle parole che compongono il nome che si cerca di mappare (tutte le parole vengono poste in *lower case* e vengono rimosse le parole "ignorate", ovvero **preposizioni articolate** e **semplici**, **articoli** e altre componenti lessicali da ignorare. Viene inizializzata a 0 la variabile `total_distance` e vengono eseguite le operazioni svolte su `toMatch` anche per l'iterato della lista degli ingredienti del database definendo così il vettore di parole che compongono l'ingrediente `ingredientWords`. La situazione generale in questo punto del codice al momento dell'esecuzione è la seguente:

```
var listOfMatchedIngredients = [];
let matching_ings = [];
let toMatch = removeIgnoredWords(product.toLowerCase()).split
    (" ");
let total_distance = 0;
let ingredientWords = removeIgnoredWords(ingredients[i].name
    .toLowerCase()).split(" ");
```

Adesso inizia il codice per il calcolo della distanza tra l'ingrediente **i** ($0 <= i < database\_ingredients.length$) ed il prodotto client in analisi calcolando

la distanza di *Demerau-Levenshtein* per ogni parola confrontata. Se la distanza di D-L è significativa (> 1) viene sommata a `total_distance` ovvero la distanza fra i due prodotti altrimenti se non lo è, a `total_distance` viene sottratta una costante. Ciò fa si che gli ingredienti il cui nome si assomiglia abbiano una `total_distance` o punteggio molto piccolo, spesso negli ingredienti che vengono matchati, `total_distance` < 0. Il codice che esegue i confronti ed assegna il valore a `total_distance` è il seguente:

```
//Per ogni parola che compone il nome ricercato
for (let j = 0; j < toMatch.length; j++) {
    //Su ogni parola che compone il nome dell'ingrediente da
       matchare
    for (let k = 0; k < ingredientWords.length; k++) {
        /*Valuta la distanza, se e' significativa (> 1)
           aggiungila al conteggio della distanza altrimenti
           match perfetto: rimuovere costante*/
        let distance = DamLev.distance(toMatch[j],
           ingredientWords[k]);
        if (distance > 0) {
            total_distance += distance;
        } else {
            total_distance -= DAMLEV_THRESHOLD * 4;
        }
    }
}
```

Se a questo punto `total_distance`, quindi il punteggio, è minore o al più uguale ad un *treshold* stabilito `DAMLEV_TRESHOLD`, l'ingrediente viene considerato un match per il prodotto client e viene inserito nella lista `matching_ings`. Questo procedimento viene eseguito per ogni prodotto client e per ogni prodotto.
Ora `matching_ings` viene ordinata e vengono presi solo i 5 match migliori (con punteggio più basso). Viene dunque inserito in
`listOfMatchedIngredients` ogni ingrediente la cui distanza è minore o uguale alla distanza del secondo ingrediente migliore, se questo secondo

ingrediente migliore non esiste viene inserito solo il match migliore.

```
matching_ings.forEach((ing) => {
    if (matching_ings.length > 1
      && ing.distance <= matching_ings[1].distance) {
          listOfMatchedIngredients.push(ing)
    } else if (matching_ings.length == 1){
        listOfMatchedIngredients.push(ing)
    }
});
```

Mentre nel caso di `GET_MATCHING` l'elaborazione dei risultati è conclusa e si può restituire al client la lista dei 10 ingredienti migliori tramite `matching_ings.slice(0, 10)`, nel caso di `GET_AVAIL_RECIPES` l'elaborazione continua e la lista `listOfMatchedIngredients` viene passata alla funzione **asincrona** `ingredientsLoop` il cui valore di ritorno (In questo caso un dizionario di oggetti costituito da una `Map`) viene restituito come risultato della chiamata API.
`ingredientsLoop` è una funzione che per ogni match ottenuto richiede tutte le ricette che possono essere eseguite utilizzando detto ingrediente. Viene dunque definito un oggetto rappresentante una ricetta che possiede un parametro `score` il quale viene incrementato per ogni ingrediente che l'algoritmo ha riconosciuto fra quelli ottenuti dal client. Questo score viene poi diviso per il numero di ingredienti in modo da ottenere un numero compreso tra 0 ed 1. Alla fine della computazione, tutti gli oggetti ricetta che possiedono uno score maggiore di 0.1 vengono aggiunti alla mappa da restituire al client sotto forma di lista.
Viene utilizzata una mappa per rendere computazionalmente meno costoso l'accesso a delle ricette già aggiunte alla mappa nel momento in cui, per una ricetta di cui il client già possiede uno o più ingredienti, venga trovato un nuovo ingrediente che il client possiede. A questo punto la mappa viene ordinata sulla base dello score e le 1000 ricette con punteggio più alto vengono restituite come risultato al client.

3.3.2 *Implementazione API gruppo C*

L'implementazione delle API del gruppo C `POST_MATCH_VOTE` e `POST_RECIPE_RATING` non richiede particolari spiegazioni, vengono esposte due *route* dedicate che recuperano dai JSON di parametro le informazioni necessarie a creare una nuova entry nel database e invocano tramite i DAO delle rispettive tabelle il metodo `create` per aggiungere una nuova entry alla base di dati.

Capitolo 4

Analisi dei dati raccolti

I dati che andremo ad analizzare in questo capitolo sono stati raccolti tramite crowdsourcing: ad un gruppo eterogeneo di 20 persone è stato chiesto di utilizzare SmartPantry per eseguire delle task, al termine delle quali esprimere un voto.

4.1 Le task da completare

"Dopo aver eseguito la registrazione/accesso a SmartPantry, utilizza il tasto «AGGIUNGI UN PRODOTTO MANUALMENTE» oppure il tasto di ***scansione codici a barre*** *per aggiungere* ***almeno 5 prodotti****. Al termine di ogni aggiunta, ti verrà mostrata una finestra di dialogo contenente una lista di prodotti. Seleziona fra i prodotti mostrati quelli che possono corrispondere al prodotto appena inserito.*
Una volta aggiunti ***almeno 5 prodotti****, vai alla sezione «Ricettario», apri* ***almeno 5 ricette*** *e votale seguendo le istruzioni riportate all'interno della finestra di che ti verrà mostrata.*
Hai finito, grazie per aver partecipato alla raccolta dati!"

Il testo sopra riportato è il testo guida che è stato inviato a tutti i partecipanti al crowdsourcing. Il testo deve essere chiaro e conciso in modo tale

che le task da eseguire siano comprese dall'individuo che deve svolgerle indipendentemente dalle sue caratteristiche personali (Ad esempio: estrazione sociale, età, affinità all'utilizzo della tecnologia, ecc...).

Nessuno dei partecipanti a riscontrato difficoltà nel seguire le indicazioni fornite e nessuno ha palesato la necessità di ricevere assistenza.

4.2 *Dati raccolti*

Di seguito vengono riportate in maniera parziale delle tabelle contenenti i dati raccolti tramite il crowdsourcing. E' da segnalare che in alcuni casi gli utenti hanno fornito un numero minore o maggiore di votazioni rispetto a quello indicato nel testo che gli è stato fornito. Ciò non dovrebbe rappresentare un problema e l'analisi dei risultati verrà eseguita sulla totalità dei dati raccolti.

4.2.1 Feedback sull'algoritmo di NLP

I feedback raccolti sulla qualità dei match eseguiti dall'applicazione consiste in una collezione di 249 *record*.

#	Ingredient to match	Ingredient matched	Is correct?
1	Mela	Mela	1
2	Mela	Semi Di Mela	0
3	Macinato misto	Macinato Misto	1
4	Macinato misto	Arrosto Misto Macinato	0
5	Sugo	Sugo	1
6	Sugo	Sugo Di Ragù	0
7	Cipolla	Cipolla	1
8	Cipolla	Cipolla Viola	0
9	sedano	Sedano	1
...			
241	peperoncino	Peperoncino	1
242	peperoncino	Pepe O Peperoncino	1
243	spaghetti	Pasta Tipo Spaghetti	1
244	spaghetti	Spaghetti Trafilati Al Bronzo	1
245	aglio	Aglio	1
246	aglio	Aglio Pulito	1
247	sale	Sale	1
248	sale	Sale Alle Erbe	0
249	sale	Sale A Scaglie	0

Tabella 4.1: *Parziale della tabella **A.1** contenente i feedback sui match eseguiti tramite NLP*

Per estrapolare informazioni utili a comprendere la qualità dell'algoritmo di NLP, dobbiamo dare una definizione insiemistica di match **Vero Positivo**, **Vero Negativo**, **Falso Positivo** e **Falso Negativo**.

- **Veri positivi** - L'insieme dei match segnati come corretti dagli utenti. Questo valore è ottenuto con l'interrogazione SQL "`SELECT count(*) from match_votes WHERE is_correct=1`" ed è **175**.

- **Falsi positivi** - L'insieme dei match segnati come errati dagli utenti. Questo valore è ottenuto con l'interrogazione SQL "`SELECT count(*) from match_votes WHERE is_correct=0`" ed è **74**.

La definizione insiemistica di **Veri Negativi** e **Falsi Negativi** invece, non è così ovvia.
Nel caso dell'insieme dei falsi negativi bisogna considerare che l'algoritmo consiglia al massimo 5 ingredienti per ogni prodotto, di conseguenza, spesso si avranno dei falsi negativi nonostante l'algoritmo li avesse identificati come positivi al match.

Per quanto riguarda i veri negativi invece, se fossimo in grado di identificare i falsi negativi, basterebbe considerare l'insieme complementare all'unione degli insiemi di positivi e falsi negativi (insieme che comunque avrebbe una cardinalità molto elevata e poco realistica dato che il numero di ingredienti a disposizione è circa 15.000 a fronte delle sole 5 richieste fatte da ogni partecipante al crowdsourcing).
Non sarà dunque possibile calcolare l'accuratezza dell'algoritmo come era stata definita nel **cap. 1.4.1**.:

$$Accuracy = \frac{V.Positivi + V.Negativi}{V.Positivi + V.Negativi + F.Positivi + F.Negativi}$$

E' tuttavia possibile definire la Precision come:

$$Precision = \frac{V.Positivi}{V.Positivi + F.Positivi} = \frac{175}{175 + 74} = 0.70281124498$$

il che significa che l'algoritmo ha una precisione di circa il **70%**, questo però, considerando tutti i dati ottenuti dalla tabella dei match indistintamente. Tuttavia, analizzando i dati ottenuti ci si rende conto che, spesso, il primo risultato di quelli proposti (massimo 5) è un match corretto. Se dunque considerassimo per ogni "`Ingredient to match`" solamente il primo risultato, otterremmo la tabella seguente.

#	Ingredient to match	Ingredient matched	Is correct?
1	Mela	Mela	1
2	Macinato misto	Macinato Misto	1
3	Sugo	Sugo	1
4	Cipolla	Cipolla	1
5	sedano	Sedano	1
6	petto di pollo	Petto Di Pollo	1
7	scalogno	Scalogno	1
8	pomodori	Pomodori	1
9	peperoncino	Pepe O Peperoncino	1
10	limone	Limone	1
11	timo	Timo	1
12	spaghetti	Pasta Tipo Spaghetti	1
13	tonno	Tonno	1
...			
85	cioccolato fondente	Cioccolato Fondente	1
86	farina 00	Farina 00	1
87	zucchero	Zucchero	1
88	pere	Pere	1
89	mele	Mele	1
90	vino rosso	Vino Rosso Borgogna Rosso	0
91	bistecca di manzo	Bistecca Di Manzo	1
92	scalogno	Scalogno	1
93	pomodori datterini	Pomodori Datterini	1
94	patate	Patate	1
95	peperoncino	Peperoncino	1
96	spaghetti	Pasta Tipo Spaghetti	1
97	aglio	Aglio	1
98	sale	Sale	1

Tabella 4.2: *Parziale della tabella **A.2** contenente il primo match (quello migliore) per ogni prodotto*

La tabella ottenuta è composta da 98 elementi dei quali, solamente 6 sono segnati come errati. Se quindi proviamo a calcolare la precisione dell'algoritmo che, contrariamente a quello implementato, restituisce solo il primo match migliore, otteniamo:

$$Precision = \frac{92}{92+6} = 0.9387755102$$

Il che mostra un incremento della precisione a circa il **93%**.

4.2.2 Dati sul Recommender System

I 20 utenti che hanno partecipato al crowdsourcing hanno prodotto **109** record di tabella, uno per ogni raccomandazione di ricetta votata. Di seguito viene mostrata una vista parziale sui dati raccolti tramite l'applicazione che mostra un'accuracy calcolata su metrica statistica del 100% in quanto, la dove l'algoritmo prevedeva il possesso del $x\%$ di ingredienti, quella previsione si è rivelata corretta **108 volte su 109** con **valore di *MAE* pari a 0.91743119266**.

Ricordiamo infatti la formula matematica per il calcolo del MAE:

$$MAE = \frac{1}{N}\sum_{u,i} |p_{u,i} - r_{u,i}|$$

dove $p_{u,i}$ è la previsione per l'utente, `Expected Rating` in tabella, che indica il rating che l'algoritmo prevede di ricevere sulla raccomandazione i dall'utente u. $r_{u,i}$ invece è il rating effettivo ottenuto da parte dell'utente u sull'oggetto i, `Effective Rating` in tabella. N è la cardinalità dell'insieme delle predizioni.
Applicando i valori dei dati raccolti:

$$MAE = \frac{1}{109} * |0 + ... + 0 + (100 - 0) + 0 + ... + 0| = 0.91743119266$$

dato il risultato della valutazione n°90 (vedere tabella **A.3**, n°=90). Il ruolo del MAE è quello di mettere a confronto le aspettative di accuratezza

dell'algoritmo con l'effettiva accuratezza riscontrata. Per esempio, se l'algoritmo suggerisce una certa ricetta perché riscontra un'accuratezza del 80% ma quella ricetta viene valutata come accurata al 10% da parte dell'utente, significa l'algoritmo ha "commesso" un errore. Se la media assoluta degli errori (MAE) è troppo alta allora l'algoritmo troppo spesso non è accurato e bisogna ricalibrarlo.
E' tuttavia probabile che eseguendo una raccolta dati su un campione più ampio di utenti e raccomandazioni il MAE salirebbe, ciò non toglie però che l'algoritmo di raccomandazione finale funziona bene consigliando agli utenti le ricette per le quali possiedono il maggior numero di ingredienti.
Per ottenere maggiori informazioni sul funzionamento dell'algoritmo è stato creato un istogramma relativo alla *Distribuzione di Probabilità Discreta* dei valori dell'`Expected Rating` nell'algoritmo.

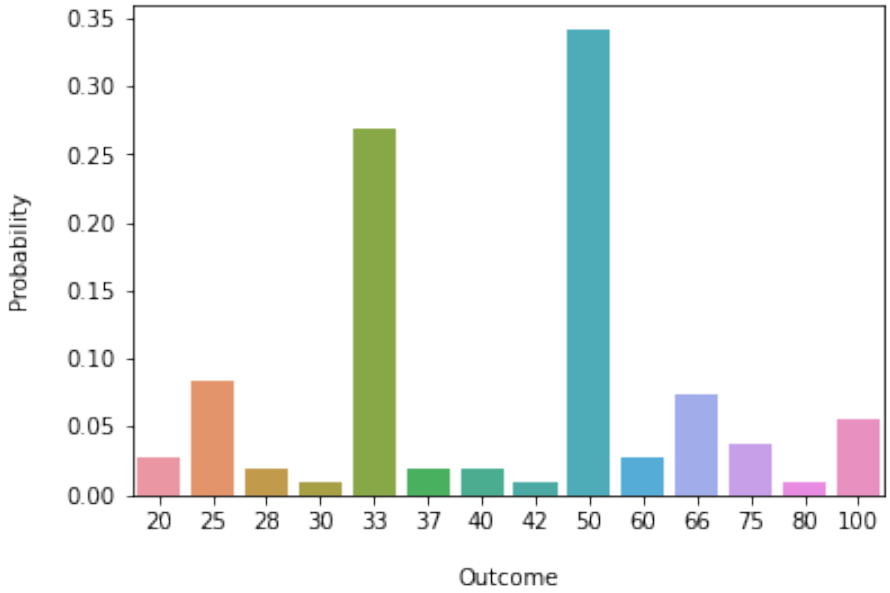

Figura 4.1: DPD del recommender system.

Osserviamo come il valore con probabilità maggiore sia il 50% con una probabilità di circa 0.35 (35%), il che è comprensibile dato che il ricettario è

costituito da moltissime ricette da due soli ingredienti e dunque è sufficiente un solo ingrediente per ottenere un buon `Expected Rating` sulle ricette in questione. A seguire 33% con probabilità di circa 0.28 e con un distacco notevole il 25% con una probabilità attorno a 0.08.

#	Recipe ID	Expected Rating	Effective Rating
1	7959	100	100
2	2035	50	50
3	22655	50	50
4	10353	33	33
5	12654	30	30
6	732	50	50
7	4080	50	50
8	6605	50	50
...			
102	25531	28	28
103	5846	25	25
104	16195	100	100
105	11903	80	80
106	3334	75	75
107	3339	75	75
108	3476	75	75
109	21583	75	75

Tabella 4.3: *Parziale della tabella **A.3** contenente i feedback sull'efficacia recommender system di ricette.*

Conclusioni

In conclusione, abbiamo osservato come l'algoritmo di raccomandazione implementato all'interno di SmartPantry abbia superato le aspettative con un **Mean Absolute Error** pari a 0.91743119266. Sicuramente c'è margine di miglioramento in termini di logica dell'algoritmo.
Non è detto che, per ogni potenziale utente dell'applicazione, la scelta di suggerire ricette per le quali si ha la maggior parte degli ingredienti sia corretta. Ad esempio ricette con un solo ingrediente (che si possiede) vengono preferite dall'algoritmo a ricette con più ingredienti dei quali si possiede il 60/75% degli ingredienti. Questo genere di comportamento potrebbe non essere quello più indicato, questo però è difficile da decidere senza il feedback di una base di utenza che utilizzi con costanza l'applicazione nella quotidianità. Lo stato attuale dell'algoritmo è sicuramente un ottimo punto di partenza.
Per quanto riguarda l'algoritmo di **Natural Language Processing** invece, il discorso cambia: l'algoritmo in se funziona bene con una precisione del 70,02%, il margine di miglioramento in questo caso è ancora più ampio. Le criticità maggiori si riscontrano ad esempio eseguendo il match di prodotti o ingredienti nei quali si ripetono delle parole, ad esempio cercando una corrispondenza per "latte" i risultati sono "Latte Oppure Siero Di Latte" e "Latte Fresco Mescolato A 1 Cucchiaio Di Latte". Entrambi i risultati hanno acquisito un punteggio di corrispondenza più alto di "Latte" o "Latte Caldo" perché nei primi due si ripete la parola "Latte" che migliora la corrispondenza. Un comportamento del genere può essere modificato mantenendo memorizzate le parole già analizzate per non analizzarle più volte.

Un'alternativa a questa soluzione è utilizzare un *dataset* migliore nel quale gli ingredienti che possono essere sostituiti ad altri vengano segnalati in maniera diversa (in modo tale da poter gestire "Latte Oppure Siero Di Latte" come i due ingredienti separati "Latte" e "Siero Di Latte").
Come già detto nella sezione **1.5**, SmartPantry non è da considerarsi finita ma bensì completa. E' un'applicazione completa perché fornisce agli utenti una collezione di funzionalità ben armonizzate fra loro che nell'insieme definiscono un ecosistema che implementa un servizio utile e ben costruito.
E' un'applicazione non finita perché non c'è limite agli aspetti di un software che possono essere migliorati: interfaccia, usabilità, sicurezza dell'applicazione in generale e complessità degli algoritmi implementati sono aspetti che possono essere migliorati (non senza un'ampia raccolta di feedback da parte degli utenti).

Tirando le somme SmartPantry non è solo un esercizio di stile, ma bensì è un prototipo di applicazione che propone delle basi solide su cui vale la pena investire risorse di tempo, lavoro e denaro con l'obiettivo di maturare un'applicazione valida che potenzialmente risponde a delle necessità reali degli utenti, delle persone.

Appendice A

Tabelle complete

Tabella A.1: *Tabella contenente la raccolta completa dei feedback sull'algoritmo di NLP per il matching dei prodotti.*

n°	Ingredient to match	Ingredient matched	Is correct?
0	Mela	Mela	1
1	Mela	Semi Di Mela	0
2	Macinato misto	Macinato Misto	1
3	Macinato misto	Arrosto Misto Macinato	0
4	Sugo	Sugo	1
5	Sugo	Sugo Di Ragù	0
6	Cipolla	Cipolla	1
7	Cipolla	Cipolla Viola	1
8	sedano	Sedano	1
9	sedano	Sedano Tenero	1
10	sedano	Sale Di Sedano	0
11	sedano	Sedano Di Verona	0
12	sedano	Sedano Bianco	1
13	petto di pollo	Petto Di Pollo	1

(Continua)

n°	Ingredient to match	Ingredient matched	Is correct?
14	petto di pollo	Petto Di Pollo Con Pelle	1
15	scalogno	Scalogno	1
16	scalogno	Scalogno Piccolo	1
17	scalogno	Scalogno Stufato	0
18	pomodori	Pomodori	1
19	pomodori	Pomodori Polposi	1
20	peperoncino	Pepe O Peperoncino	1
21	peperoncino	Peperoncino	1
22	limone	Limone	1
23	limone	Timo Limone	0
24	limone	Limone	1
25	limone	Timo Limone	0
26	timo	Timo	1
27	timo	Timo Limone	0
28	spaghetti	Pasta Tipo Spaghetti	1
29	spaghetti	Spaghetti Trafilati Al Bronzo	1
30	tonno	Tonno	1
31	tonno	Tonno Rosso	0
32	cipolla	Cipolla	1
33	cipolla	Cipolla Viola	0
34	gelato alla fragola	Gelato Alla Fragola	1
35	gelato alla fragola	Gelato Di Fragola	1
36	latte	Latte Oppure Siero Di Latte	1
37	latte	Latte Fresco Mescolato A 1 Cucchiaio Di Latte Condensato	0

(Continua)

n°	Ingredient to match	Ingredient matched	Is correct?
38	uova	Uova	1
39	uova	Uova Sode	0
40	uova	Uova Di Trota	0
41	banane	Banane	1
42	banane	Arance E Banane	0
43	anguria	Anguria	1
44	anguria	Anguria Matura	1
45	pane	Pane	1
46	pane	Pasta Per Pane	0
47	pane	Pasta Da Pane	0
48	pane	Pane Pita	0
49	pane	Pasta Di Pane	0
50	uova	Uova	1
51	uova	Uova Sode	0
52	uova	Uova Di Trota	0
53	latte	Latte Oppure Siero Di Latte	1
54	latte	Latte Fresco Mescolato A 1 Cucchiaio Di Latte Condensato	0
55	marmellata d'arancia	Marmellata D'arancia E Marmellata Di Limone	0
56	marmellata d'arancia	Marmellata D'arancia	1
57	lime	Lime	1
58	lime	Succo Di Lime	1
59	lime	Fettine Di Lime	1
60	lime	Scorze Di Lime	1
61	mozzarella	Mozzarella	1

(Continua)

n°	Ingredient to match	Ingredient matched	Is correct?
62	mozzarella	Mozzarella Di Bufala	0
63	sugo	Sugo	1
64	sugo	Sugo Di Ragù	0
65	farina	Farina	1
66	farina	Farina Di Farro	0
67	basilico	Basilico Secco	0
68	basilico	Basilico	1
69	basilico	Basilico Riccio	0
70	olio d'oliva	Olio Di Sesamo O Olio D'oliva	0
71	olio d'oliva	Olio D'oliva	1
72	cosce di pollo	Cosce Di Pollo	1
73	cosce di pollo	Polpa Di Cosce Di Pollo	1
74	insalata iceberg	Insalata Iceberg	1
75	insalata iceberg	Insalata	1
76	pomodorini	Pomodorini	1
77	pomodorini	Pomodorini Perini	0
78	cetrioli	Cetrioli Piccoli	1
79	cetrioli	Cetrioli	1
80	cetrioli	Cetrioli Tritati	1
81	cetrioli	Cetrioli Verdi	1
82	soia	Soia	1
83	soia	Salsa Di Soia	1
84	soia	Olio Di Soia	1
85	soia	Semi Di Soia	1
86	arance rosse	Arance Rosse	1
87	arance rosse	Arance Rosse Sode	1
88	pennette	Pennette	1

(Continua)

n°	Ingredient to match	Ingredient matched	Is correct?
89	pennette	Pennette Rigate	1
90	sugo di pomodoro	Sugo Di Pomodoro	1
91	sugo di pomodoro	Sugo Al Pomodoro	1
92	pesto	Pesto	1
93	pesto	Pesto Pronto	1
94	aglio	Aglio	1
95	aglio	Aglio Pulito	1
96	triglia	Triglia	1
97	triglia	Triglia Grande	1
98	scampi	Scampi	1
99	scampi	Scampi Cotti	0
100	scampi	Scampi Grandi	1
101	scampi	Scampi Reali	0
102	pomodori	Pomodori	1
103	pomodori	Pomodori Polposi	1
104	olive	Olive Verdi E Olive Nere	1
105	olive	Olive Nere E Olive Verdi	1
106	aglio	Aglio	1
107	aglio	Aglio Pulito	1
108	melanzane	Melanzane	1
109	melanzane	Melanzane Lunghe	1
110	melanzane	Melanzane Medie	1
111	melanzane	Caviale Di Melanzane	0
112	passata di pomodoro	Passata Di Pomodoro	1
113	passata di pomodoro	Passata Pomodoro	1
114	basilico	Basilico	1

(Continua)

n°	Ingredient to match	Ingredient matched	Is correct?
115	basilico	Basilico Secco	1
116	basilico	Basilico Riccio	0
117	ricotta	Ricotta	1
118	ricotta	Ricotta Di Pecora	0
119	fusilli	Pasta Tipo Fusilli	1
120	fusilli	Pasta Tipo Fusilli Bucati	1
121	fusilli	Pasta Tipo Fusilli Lunghi	1
122	miele	Miele	1
123	miele	Miele Di Tiglio	0
124	fragole	Fragole	1
125	fragole	Fragole Fresche	1
126	fragole	Fragole Piccole	1
127	cioccolato	Cioccolato	1
128	cioccolato	Cioccolato Sbriciolato	0
129	cioccolato	Cioccolato Sciolto	0
130	pollo petto	Petto Di Pollo	1
131	pollo petto	Petto Di Pollo Con Pelle	1
132	menta	Menta	1
133	menta	Menta Secca	0
134	pesto	Pesto	1
135	pesto	Pesto Pronto	1
136	fettina di vitello	Vitello	1
137	fettina di vitello	Fettine Di Vitello	1
138	fettina di vitello	Vitello A Fettine	1
139	tonno	Tonno	1
140	tonno	Tonno Rosso	0

(Continua)

n°	Ingredient to match	Ingredient matched	Is correct?
141	zucchine	Zucchine	1
142	zucchine	Zucchine Piccole	1
143	zucchine	Succo Di Zucchine	0
144	zucchine	Zucchine Lunghe	1
145	coca cola	Coca Cola	1
146	coca cola	Coca Buton	0
147	cioccolato	Cioccolato	1
148	cioccolato	Cioccolato Sbriciolato	1
149	cioccolato	Cioccolato Sciolto	0
150	petto d'anatra	Petto D'anatra	1
151	petto d'anatra	Petti D'anatra	1
152	sugo pronto	Pesto Pronto	0
153	sugo pronto	Sugo	1
154	sugo pronto	Pollo Pronto	0
155	spaghetti	Pasta Tipo Spaghetti	1
156	spaghetti	Spaghetti Trafilati Al Bronzo	1
157	calamari	Calamari Tagliati	1
158	calamari	Calamari	1
159	rape	Rape	1
160	rape	Cime Di Rape	1
161	rape	Rape Rosse	0
162	fave	Fave	1
163	fave	Fave E Ceci	0
164	orecchiette	Orecchiette	1
165	orecchiette	Funghi Orecchiette	0
166	zucchine	Zucchine	1
167	zucchine	Zucchine Piccole	0

(Continua)

n°	Ingredient to match	Ingredient matched	Is correct?
168	zucchine	Succo Di Zucchine	0
169	zucchine	Zucchine Lunghe	0
170	pomodorini sott'olio	Pomodorini Sott'olio	1
171	pomodorini sott'olio	Pomodorini Secchi Sott'olio	1
172	melanzane	Melanzane	1
173	melanzane	Melanzane Lunghe	1
174	melanzane	Melanzane Medie	1
175	melanzane	Caviale Di Melanzane	0
176	costine di maiale	Costine Di Maiale	1
177	costine di maiale	Costine Di Maiale Affumicate	0
178	pomodorini	Pomodorini	1
179	pomodorini	Pomodorini Perini	0
180	aglio	Aglio	1
181	aglio	Aglio Pulito	1
182	olio d'oliva	Olio Di Sesamo O Olio D'oliva	0
183	olio d'oliva	Olio D'oliva	1
184	pecorino	Formaggio Pecorino	1
185	pecorino	Formaggio Pecorino Romano	0
186	pecorino	Formaggio Pecorino Siciliano	0
187	pecorino	Formaggio Pecorino Marzolino	0
188	capocollo	Capocollo	1
189	capocollo	Capocollo Di Maiale	1

(Continua)

n°	Ingredient to match	Ingredient matched	Is correct?
190	uova	Uova	1
191	uova	Uova Sode	0
192	uova	Uova Di Trota	0
193	pennette rigate	Pennette Rigate	1
194	pennette rigate	Pasta Tipo Pennette Rigate	1
195	birra	Birra	1
196	birra	Birra Chiara	1
197	birra	Birra Scura	0
198	birra	Birra Bionda	1
199	birra	Birra Amara	0
200	macinato di manzo	Macinato Di Manzo	1
201	macinato di manzo	Manzo Macinato	1
202	cipolla	Cipolla	1
203	cipolla	Cipolla Viola	0
204	sugo di pomodoro	Sugo Di Pomodoro	1
205	sugo di pomodoro	Sugo Al Pomodoro	1
206	dado di brodo	Brodo Di Manzo O Brodo Di Dado Vegetale	1
207	dado di brodo	Dado Di Brodo	1
208	dado di brodo	Dado Per Brodo	1
209	dado di brodo	Dado Da Brodo	1
210	dado di brodo	Brodo Di Dado	1
211	carote	Carote	1
212	carote	Carote Crude	1
213	carote	Carote Cotte	0
214	cioccolato fondente	Cioccolato Fondente	1

(Continua)

n°	Ingredient to match	Ingredient matched	Is correct?
215	cioccolato fondente	Cioccolato Fondente A Tocchetti	1
216	farina 00	Farina 00	1
217	farina 00	Farina Di Grano 00	1
218	farina 00	Farina Tipo 00	1
219	zucchero	Zucchero	1
220	zucchero	Zucchero Scuro	0
221	zucchero	Burro E Zucchero	0
222	zucchero	Zucchero Nero	0
223	zucchero	Zucchero A Piacere	1
224	pere	Pere	1
225	pere	Pere Dure	1
226	pere	Mele E Pere	0
227	mele	Mele	1
228	mele	Mele E Pere	0
229	vino rosso	Vino Rosso Borgogna Rosso	0
230	vino rosso	Vino Rosso	1
231	bistecca di manzo	Bistecca Di Manzo	1
232	bistecca di manzo	Bistecca Di Entrecote Di Manzo	0
233	scalogno	Scalogno	1
234	scalogno	Scalogno Piccolo	1
235	scalogno	Scalogno Stufato	0
236	pomodori datterini	Pomodori Datterini	1
237	pomodori datterini	Pomodori	1
238	patate	Patate	1
239	patate	Patate Pelate	1

(Continua)

n°	Ingredient to match	Ingredient matched	Is correct?
240	peperoncino	Peperoncino	1
241	peperoncino	Pepe O Peperoncino	1
242	spaghetti	Pasta Tipo Spaghetti	1
243	spaghetti	Spaghetti Trafilati Al Bronzo	1
244	aglio	Aglio	1
245	aglio	Aglio Pulito	1
246	sale	Sale	1
247	sale	Sale Alle Erbe	0
248	sale	Sale A Scaglie	0

Tabella A.2: *Tabella contenente la raccolta completa dei feedback sull'algoritmo di NLP per il matching dei prodotti fermandosi al match migliore. "*`New n°`*" è l'indice dei match nella tabella in questione, "*`n°`*" è l'indice associato ai match nella tabella* ***A.1***

`New n°`	`n°`	`Ingredient to match`	`Ingredient matched`	`Is correct?`
0	0	Mela	Mela	1
1	2	Macinato misto	Macinato Misto	1
2	4	Sugo	Sugo	1
3	6	Cipolla	Cipolla	1
4	8	sedano	Sedano	1
5	13	petto di pollo	Petto Di Pollo	1
6	15	scalogno	Scalogno	1
7	18	pomodori	Pomodori	1
8	20	peperoncino	Pepe O Peperoncino	1
9	22	limone	Limone	1
10	26	timo	Timo	dai1
11	28	spaghetti	Pasta Tipo Spaghetti	1
12	30	tonno	Tonno	1
13	32	cipolla	Cipolla	1
14	34	gelato alla fragola	Gelato Alla Fragola	1
15	36	latte	Latte Oppure Siero Di Latte	1
16	38	uova	Uova	1
17	41	banane	Banane	1
18	43	anguria	Anguria	1
19	45	pane	Pane	1
20	50	uova	Uova	1

(Continua)

New	n°	Ingredient to match	Ingredient matched	Is correct?
21	53	latte	Latte Oppure Siero Di Latte	1
22	55	marmellata d'arancia	Marmellata D'arancia E Marmellata Di Limone	0
23	57	lime	Lime	1
24	61	mozzarella	Mozzarella	1
25	63	sugo	Sugo	1
26	65	farina	Farina	1
27	67	basilico	Basilico Secco	0
28	70	olio d'oliva	Olio Di Sesamo O Olio D'oliva	0
29	72	cosce di pollo	Cosce Di Pollo	1
30	74	insalata iceberg	Insalata Iceberg	1
31	76	pomodorini	Pomodorini	1
32	78	cetrioli	Cetrioli Piccoli	1
33	82	soia	Soia	1
34	86	arance rosse	Arance Rosse	1
35	88	pennette	Pennette	1
36	90	sugo di pomodoro	Sugo Di Pomodoro	1
37	92	pesto	Pesto	1
38	94	aglio	Aglio	1
39	96	triglia	Triglia	1
40	98	scampi	Scampi	1
41	102	pomodori	Pomodori	1
42	104	olive	Olive Verdi E Olive Nere	1
43	106	aglio	Aglio	1

(Continua)

New	n°	Ingredient to match	Ingredient matched	Is correct?
44	108	melanzane	Melanzane	1
45	112	passata di pomodoro	Passata Di Pomodoro	1
46	114	basilico	Basilico	1
47	117	ricotta	Ricotta	1
48	119	fusilli	Pasta Tipo Fusilli	1
49	122	miele	Miele	1
50	124	fragole	Fragole	1
51	127	cioccolato	Cioccolato	1
52	130	pollo petto	Petto Di Pollo	1
53	132	menta	Menta	1
54	134	pesto	Pesto	1
55	136	fettina di vitello	Vitello	1
56	139	tonno	Tonno	1
57	141	zucchine	Zucchine	1
58	145	coca cola	Coca Cola	1
59	147	cioccolato	Cioccolato	1
60	150	petto d'anatra	Petto D'anatra	1
61	152	sugo pronto	Pesto Pronto	0
62	155	spaghetti	Pasta Tipo Spaghetti	1
63	157	calamari	Calamari Tagliati	1
64	159	rape	Rape	1
65	162	fave	Fave	1
66	164	orecchiette	Orecchiette	1
67	166	zucchine	Zucchine	1
68	170	pomodorini sott'olio	Pomodorini Sott'olio	1
69	172	melanzane	Melanzane	1
70	176	costine di maiale	Costine Di Maiale	1
71	178	pomodorini	Pomodorini	1

(Continua)

New	n°	Ingredient to match	Ingredient matched	Is correct?
72	180	aglio	Aglio	1
73	182	olio d'oliva	Olio Di Sesamo O Olio D'oliva	0
74	184	pecorino	Formaggio Pecorino	1
75	188	capocollo	Capocollo	1
76	190	uova	Uova	1
77	193	pennette rigate	Pennette Rigate	1
78	195	birra	Birra	1
79	200	macinato di manzo	Macinato Di Manzo	1
80	202	cipolla	Cipolla	1
81	204	sugo di pomodoro	Sugo Di Pomodoro	1
82	206	dado di brodo	Brodo Di Manzo O Brodo Di Dado Vegetale	1
83	211	carote	Carote	1
84	214	cioccolato fondente	Cioccolato Fondente	1
85	216	farina 00	Farina 00	1
86	219	zucchero	Zucchero	1
87	224	pere	Pere	1
88	227	mele	Mele	1
89	229	vino rosso	Vino Rosso Borgogna Rosso	0
90	231	bistecca di manzo	Bistecca Di Manzo	1
91	233	scalogno	Scalogno	1
92	236	pomodori datterini	Pomodori Datterini	1
93	238	patate	Patate	1
94	240	peperoncino	Peperoncino	1
95	242	spaghetti	Pasta Tipo Spaghetti	1

(Continua)

New	n°	Ingredient to match	Ingredient matched	Is correct?
96	244	aglio	Aglio	1
97	246	sale	Sale	1

Tabella A.3: *Tabella contenente la raccolta completa dei feedback sull'efficacia dell'algoritmo del recommender system.*

n°	Recipe ID	Expected rating	Actual rating
0	7959	100	100
1	2035	50	50
2	22655	50	50
3	10353	33	33
4	12654	30	30
5	732	50	50
6	4080	50	50
7	6605	50	50
8	14432	40	40
9	2863	33	33
10	4050	33	33
11	23495	66	66
12	5843	50	50
13	21524	50	50
14	23611	50	50
15	24698	50	50
16	1279	33	33
17	1887	50	50
18	10539	50	50
19	26909	50	50
20	807	33	33
21	958	33	33
22	731	50	50
23	15687	50	50
24	15738	50	50

(Continua)

n°	Recipe ID	Expected rating	Actual rating
25	27057	50	50
26	1633	33	33
27	9889	66	66
28	18852	60	60
29	4157	50	50
30	10404	50	50
31	13328	50	50
32	17285	66	66
33	13465	50	50
34	18535	50	50
35	5361	25	25
36	20139	25	25
37	2893	20	20
38	214	50	50
39	18506	33	33
40	22172	33	33
41	248	25	25
42	1806	25	25
43	6292	40	40
44	3361	37	37
45	14779	33	33
46	9470	28	28
47	23483	33	33
48	10770	50	50
49	17325	42	42
50	10786	37	37
51	1030	33	33
52	14731	33	33

(Continua)

n°	Recipe ID	Expected rating	Actual rating
53	9975	66	66
54	2230	50	50
55	8078	50	50
56	14889	33	33
57	25342	33	33
58	24011	33	33
59	7321	33	33
60	24812	33	33
61	7898	33	33
62	24902	33	33
63	27834	33	33
64	1881	50	50
65	4077	50	50
66	18283	33	33
67	4182	25	25
68	9306	25	25
69	9195	33	33
70	12745	25	25
71	16405	20	20
72	21673	20	20
73	24132	25	25
74	16192	100	100
75	5247	66	66
76	14686	60	60
77	18167	66	66
78	21829	66	66
79	563	50	50
80	1463	50	50

(Continua)

n°	Recipe ID	Expected rating	Actual rating
81	11855	50	50
82	17281	50	50
883	8336	50	50
84	4973	50	50
85	9893	50	50
86	6059	33	33
87	8130	33	33
88	28139	33	33
89	11364	33	33
90	14843	100	0
91	14869	100	100
92	14846	66	66
93	18254	60	60
94	715	50	50
95	7461	50	50
96	5327	50	50
97	17799	100	100
98	13129	33	33
99	17790	33	33
100	25531	28	28
101	5846	25	25
102	16195	100	100
103	11903	80	80
104	3334	75	75
105	3339	75	75
106	3476	75	75
107	21583	75	75

Bibliografia

[1] YML. *Native VS Hybrid Mobile Apps — Here's How To Choose*. Lug. 2021. URL: https://yml.co/native-vs-hybrid-mobile-apps-heres-how-to-choose (visitato il 18/07/2022).

[2] StatCounter. *Mobile & Tablet Operating System Market Share Worldwide*. URL: https://gs.statcounter.com/os-market-share/mobile-tablet/worldwide/#yearly-2012-2022 (visitato il 18/07/2022).

[3] Diksha Khurana et al. «Natural Language Processing: State of The Art, Current Trends and Challenges». In: *CoRR* abs/1708.05148 (2017). arXiv: 1708.05148. URL: http://arxiv.org/abs/1708.05148.

[4] Abdullah Hamid et al. *Fake News Detection in Social Media using Graph Neural Networks and NLP Techniques: A COVID-19 Use-case*. 2020. DOI: 10.48550/ARXIV.2012.07517. URL: https://arxiv.org/abs/2012.07517.

[5] Chunchun Zhao e Sartaj Sahni. «String correction using the Damerau-Levenshtein distance». In: *BMC Bioinformatics* 20.11 (giu. 2019), p. 277. ISSN: 1471-2105. DOI: 10.1186/s12859-019-2819-0. URL: https://doi.org/10.1186/s12859-019-2819-0.

[6] Gregory V. Bard. *Spelling-Error Tolerant, Order-Independent Pass-Phrases via the Damerau-Levenshtein String-Edit Distance Metric*. Cryptology ePrint Archive, Paper 2006/364. https://eprint.iacr.org/2006/364. 2006. URL: https://eprint.iacr.org/2006/364.

[7] Roy Lowrance e Robert A. Wagner. «An Extension of the String-to-String Correction Problem». In: *J. ACM* 22 (1975), pp. 177–183.

[8] Desmond Bala Bisandu, Rajesh Prasad e Musa Muhammad Liman. «Data clustering using efficient similarity measures». In: *Journal of Statistics and Management Systems* 22.5 (2019), pp. 901–922. DOI: 10.1080/09720510.2019.1565443. eprint: https://doi.org/10.1080/09720510.2019.1565443. URL: https://doi.org/10.1080/09720510.2019.1565443.

[9] Sovan Saha et al. «Protein function prediction from dynamic protein interaction network using gene expression data». In: *Journal of Bioinformatics and Computational Biology* 17.04 (2019). PMID: 31617461, p. 1950025. DOI: 10.1142/S0219720019500252. eprint: https://doi.org/10.1142/S0219720019500252. URL: https://doi.org/10.1142/S0219720019500252.

[10] Daw Khin Po. «Similarity Based Information Retrieval Using Levenshtein Distance Algorithm». In: *Int. J. Adv. Sci. Res. Eng* 6.04 (2020), pp. 06–10.

[11] Shi. *Recommendation Systems: A Review, A summary of recommender system methods*. A cura di towardsdatascience.com. [Online; posted 23-February-2020]. Feb. 2020. URL: https://towardsdatascience.com/recommendation-systems-a-review-d4592b6caf4b.

[12] Jonathan L. Herlocker et al. «Evaluating Collaborative Filtering Recommender Systems». In: *ACM Trans. Inf. Syst.* 22.1 (gen. 2004), pp. 5–53. ISSN: 1046-8188. DOI: 10.1145/963770.963772. URL: https://doi.org/10.1145/963770.963772.

[13] Enrique Estellés-Arolas e Fernando González-Ladrón-de-Guevara. «Towards an integrated crowdsourcing definition». In: *Journal of Information Science* 38.2 (2012), pp. 189–200. DOI: 10.1177/0165551512437638. eprint: https://doi.org/10.1177/0165551512437638. URL: https://doi.org/10.1177/0165551512437638.

[14] expert.ai - femanuele. *Precision and Recall, F-score and Accuracy – Measuring NLP performance*. Ott. 2021. URL: https://community.expert.ai/articles-showcase-56/precision-and-recall-f-score-and-accuracy-measuring-nlp-performance-191?postid=366#post366 (visitato il 23/07/2022).

[15] F.O. Isinkaye, Y.O. Folajimi e B.A. Ojokoh. «Recommendation systems: Principles, methods and evaluation». In: *Egyptian Informatics Journal* 16.3 (2015), pp. 261–273. ISSN: 1110-8665. DOI: https://doi.org/10.1016/j.eij.2015.06.005. URL: https://www.sciencedirect.com/science/article/pii/S1110866515000341.

[16] Badrul Sarwar et al. «Item-Based Collaborative Filtering Recommendation Algorithms». In: *Proceedings of the 10th International Conference on World Wide Web*. WWW '01. Hong Kong, Hong Kong: Association for Computing Machinery, 2001, pp. 285–295. ISBN: 1581133480. DOI: 10.1145/371920.372071. URL: https://doi.org/10.1145/371920.372071.

[17] Ken Goldberg et al. «Eigentaste: A Constant Time Collaborative Filtering Algorithm». In: *Information Retrieval* 4.2 (lug. 2001), pp. 133–151. DOI: 10.1023/A:1011419012209. URL: https://doi.org/10.1023/A:1011419012209.

[18] Tim Miranda et al. «Combining Content-Based and Collaborative Filters in an Online Newspaper». In: *In Proceedings of ACM SIGIR Workshop on Recommender Systems*. 1999.

[19] Badrul M Sarwar et al. «Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system». In: *Proceedings of the 1998 ACM conference on Computer supported cooperative work*. 1998, pp. 345–354.