



JustCook

03/12/22

Sviluppo applicazione



Indice

SCOPO DEL DOCUMENTO	2
1. USER FLOWS	3
1.1. REGISTRAZIONE, LOGIN, PAGINA ACCOUNT	3
1.2. DISPENSA	4
1.3. RICERCA RICETTE, PAGINA RICETTA	4
2. APPLICATION IMPLEMENTATION AND DOCUMENTATION	5
2.1. PROJECT STRUCTURE	5
2.2. PROJECT DEPENDENCIES	6
2.3. PROJECT DATA OR DB	7
2.4. PROJECT APIs	11
2.4.1. RESOURCES EXTRACTION	11
2.4.2. RESOURCES MODEL	21
2.5. SVILUPPO API	33
3. API DOCUMENTATION	48
4. FRONTEND IMPLEMENTATION	50
4.1. PAGINA HOME	50
4.2. PAGINA CAMBIO PASSWORD	52
4.3. PAGINA RICETTA	53
5. GITHUB REPOSITORY AND DEPLOYMENT INFO	54
5.1. GITHUB REPOSITORY	54
5.2. DEPLOYMENT INFO	57
6. TESTING	58
6.1. CASI DI TEST	58
6.2. COVERAGE	62



SCOPO DEL DOCUMENTO

Questo documento riporta tutte le informazioni necessarie per lo sviluppo di una parte di JustCook. Ovviamente si è tenuto conto di tutte le analisi eseguite e mostrate nei documenti precedenti. Il “prototipo” realizzato riguarda principalmente le funzionalità dell’utente. Infatti, prima di analizzare l’implementazione vera e propria, viene presentato l’user flow dell’utente. Successivamente vengono descritte le API ideate (con Resources Extraction Diagram e Resources Model) e il loro codice. È anche stata realizzata la documentazione delle API e una parte di testing. Infine viene presentato il Front-end, la Github repository e il deployment dell’applicazione.

1. USER FLOWS

In questo paragrafo viene presentato lo user flows dell'utente. Il grafico viene diviso in più parti per permettere al lettore di analizzarlo e visualizzarlo al meglio. Per ogni parte ci sarà una breve descrizione.

1.1. REGISTRAZIONE, LOGIN, PAGINA ACCOUNT

Dalla pagina della home l'utente può effettuare la registrazione se non possiede ancora un account. Quando si crea l'account si va direttamente alla pagina corrispondente.

Invece, se la registrazione è già stata effettuata, si può eseguire il login nella rispettiva pagina. In quest'ultima si può anche effettuare il cambio password. Se l'autenticazione viene eseguita con successo, l'utente accede alla propria pagina dell'account. Infine, se il login è già stato eseguito, si può accedere direttamente alla pagina dell'account dalla home.

Nella pagina della home è possibile: visualizzare la lista delle ricette preferite, guardare le proprie ricreazioni dei piatti, modificare i propri dati e caricare nuove ricette.

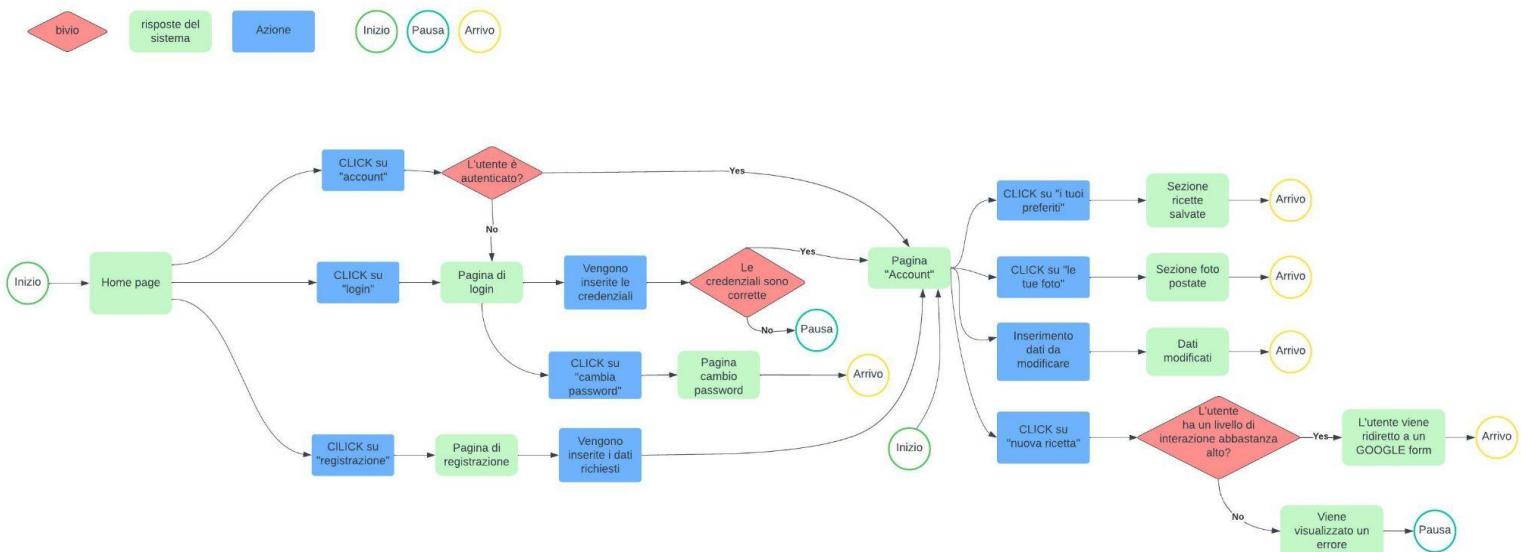


Figura 1: User flows registrazione, login, pagina account

1.2.DISPENSA

Dalla pagina della home l'utente può accedere alla propria dispensa. Può aggiungere gli ingredienti inserendo i loro nomi nella barra di ricerca. Inoltre si può specificarne la quantità. L'utente può anche modificare la quantità degli ingredienti e/o eliminarli dalla dispensa.

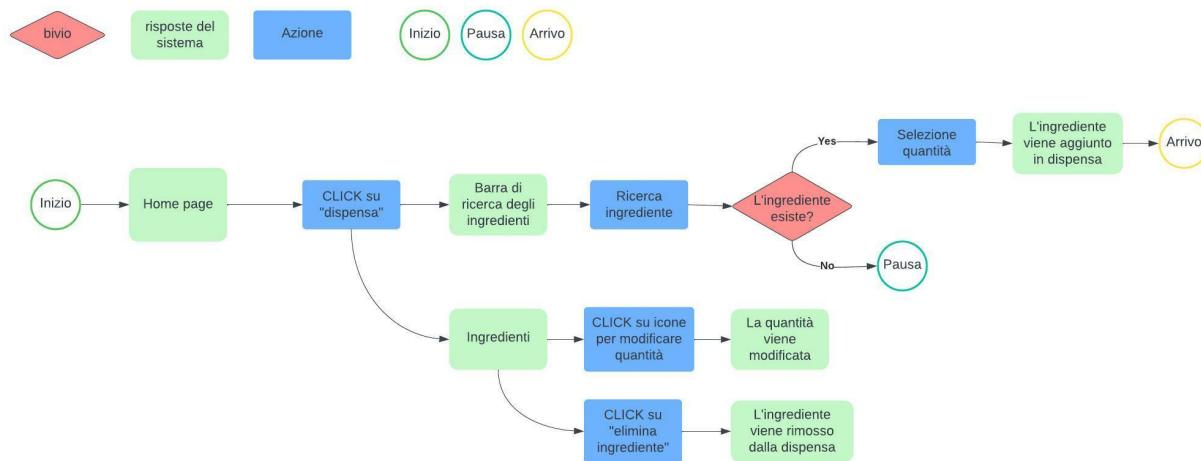


Figura 2: User flows dispensa

1.3.RICERCA RICETTE, PAGINA RICETTA

Dalla pagina della home l'utente può cercare le ricette. Può farlo utilizzando (oltre agli ingredienti presenti in dispensa) il nome del piatto, i parametri di ricerca e/o i filtri. L'utente cliccando su un risultato va alla pagina della ricetta corrispondente. In quest'ultima l'utente può: commentare allegando una immagine, segnalare gli altri commenti, aggiungere o eliminare la ricetta ai preferiti, completare il piatto, lasciare una votazione.

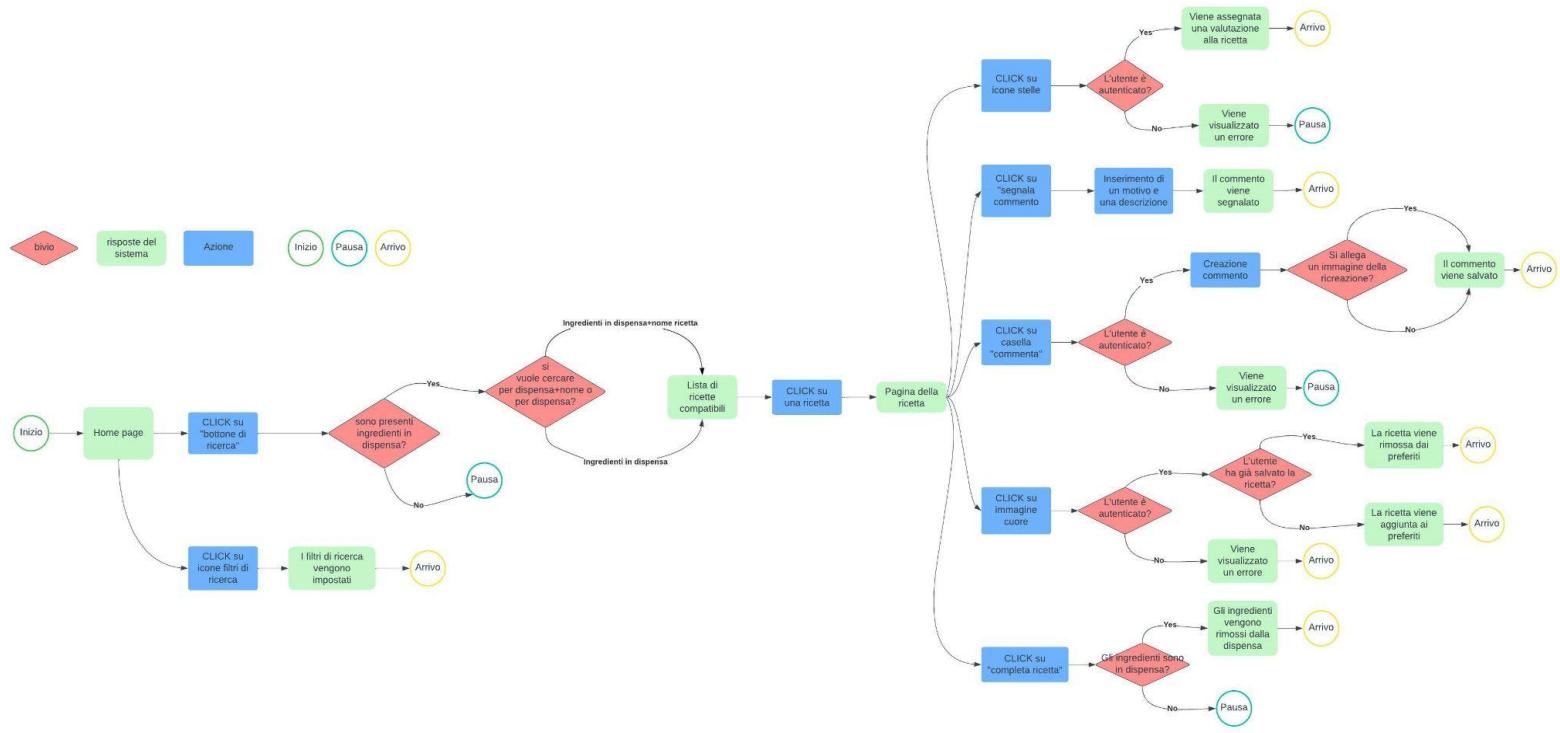


Figura 3: User flows ricerca ricette, pagina ricetta

2. APPLICATION IMPLEMENTATION AND DOCUMENTATION

2.1. PROJECT STRUCTURE

La cartella principale è JustCook. La struttura del progetto si divide in cinque cartelle: **controllers**, **grafica**, **models**, **routes**, **test**. La cartella **models** contiene i modelli delle strutture dati presenti nel database. Le cartelle **controllers** e **routes** servono per la gestione delle API. La cartella **grafica** contiene il codice per il Front-end. Infine la cartella **test** contiene il testing delle API. Questa suddivisione è visibile in [figura 4](#). Inoltre nel [paragrafo 5.1](#) verrà analizzata più nel dettaglio la Github repository.

```
▽ DOCUMENTAZIONE_D4
  > controllers
  > models
  > routes
  > static
  > test
  ♦ .gitignore
  JS app.js
  {} package-lock.json
  {} package.json
  ↵ Procfile
  ⓘ README.md
  JS server.js
  {} swagger.json
```

Figura 4: Struttura del progetto

2.2. PROJECT DEPENDENCIES

Oltre ai moduli Node standard che vengono scaricati al momento dell'installazione, sono stati utilizzati altri moduli e aggiunti al file Package.Json. Quest'ultimi sono:

- Express
- Mongoose
- Cors
- Dotenv
- Multer
- Swagger -ui -express
- Jest
- Ms
- Supertest

2.3. PROJECT DATA OR DB

In questo paragrafo vengono presentate le strutture dati presenti nel database di JustCook. Non sono state ideate tutte quelle necessarie per la completa realizzazione dell'applicazione Web, ma quelle relative ai dati utili per la “creazione di un prototipo”. Nell'immagine seguente ([figura 4](#)) sono mostrate le collezioni dati appena introdotte. Successivamente per ogni struttura verrà riportata una descrizione con un esempio.

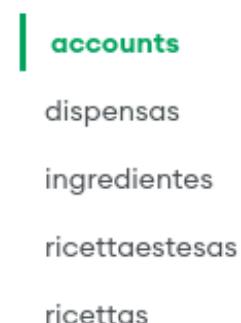


Figura 5: Collezioni Dati

Esempi tipi di dato

1. ACCOUNT

Per rappresentare gli account è stato creato il modello account con le seguenti “proprietà”:

- **preferiti**: contiene gli id delle ricette preferite dall’utente
- **password**: la password dell’account
- **ratingDati** : contiene gli id delle ricette e i rating associati ad esse dati dagli utenti
- **primiCompletamenti**: contiene gli id delle ricette completate (per la prima volta) dall’utente
- **indirizzoEmail**: l’indirizzo email associato all’account
- **username**: il nome utente associato all’account

```
_id: ObjectId('63a6fa25db2b4423e4c6e5de')
  preferiti: Array
    0: ObjectId('639d9961f5edd649e010000e')
    1: ObjectId('639cd417d099f3b10a5e238c')
    password: "eadJ039kl^^"
  ratingDati: Array
    0: Object
      ricetta: ObjectId('639cd417d099f3b10a5e238c')
      ratingData: 3
      _id: ObjectId('63aeab3dd17d3be57b478d53')
    1: Object
      ricetta: ObjectId('639d9961f5edd649e010000e')
      ratingData: 3
      _id: ObjectId('63aeab79d17d3be57b478d62')
  primiCompletamenti: Array
    0: ObjectId('639cd417d099f3b10a5e238c')
  indirizzoEmail: "babbonatale:)@gmail.com"
  username: "BabboNatale01"
```

Figura 6: Tipo di dato Account

2. DISPENSA

Per rappresentare le dispense è stato creato il modello dispensa con le seguenti “proprietà”:

- ingredienti: contiene gli ingredienti presenti nella dispensa
- account: è l'id dell'account associato ad una determinata dispensa
- quantita: contiene le quantità degli ingredienti presenti in dispensa

```
_id: ObjectId('63aee34a40713fc321b4d6cd')
  ingredienti: Array
    0: ObjectId('639cd6ee6335dc6ce6a36672')
  account: ObjectId('63a6fa25db2b4423e4c6e5de')
  quantita: Array
    0: 1
```

Figura 7: Tipo di dato Dispensa

3. INGREDIENTE

Per rappresentare gli ingredienti è stato creato il modello ingrediente con le seguenti “proprietà”:

- nome: è il nome dell'ingrediente
- tipo: è il tipo (grammi, numero, etti) dell'ingrediente

```
_id: ObjectId('639d9672f5edd649e0100005')
nome: "Yogurt greco"
tipo: "g"
```

Figura 8: Tipo di dato `ingrediente`

4. RICETTA ESTESA

Per rappresentare le ricette della pagina della ricetta è stato creato il modello `ricettaEstesa` con le seguenti “proprietà”:

- `ricetta`: è l'id della ricetta corrispondente
- `descrizione`: è la descrizione della ricetta in breve
- `ingredienti`: contiene gli ingredienti utilizzati nella ricetta corrispondente
- `passaggi`: è l'insieme delle istruzioni per la creazione del piatto
- `quantita`: contiene le quantità degli ingredienti della ricetta

```
_id: ObjectId('639cd58cd099f3b10a600972')
ricetta: ObjectId('639cd417d099f3b10a5e238c')
descrizione: "Il tiramisù è uno dei capisaldi della cucina italiana, uno dei dolci a..."
▼ ingredienti: Array
  0: ObjectId('639cd6dc099f3b10a615c4a')
  1: ObjectId('639cd6ee6335dc6ce6a36672')
▼ passaggi: Array
  0: "Per preparare il tiramisù preparate il caffè con la moka per ottenerne..."
  1: "Montate i tuorli con le fruste elettriche, versando solo metà dose di ..."
  2: "Non appena il composto sarà diventato chiaro e spumoso, e con le frust..."
  3: "Incorporato tutto il formaggio avrete ottenuto una crema densa e compa..."
  4: "Pulite molto bene le fruste e passate a montare gli albumi"
  5: "Quando saranno schiumosi versate il restante zucchero un po' alla volt..."
  6: "Dovrete montarli a neve ben ferma "
  7: "Prendete una cucchiainata di albumi e versatela nella ciotola con la cr..."
  8: " Mescolate energicamente con una spatola"
  9: "Dopodiché procedete ad aggiungere la restante parte di albumi, poco al..."
▼ quantita: Array
  0: 750
  1: 250
```

Figura 9: Tipo di dato `ricettaEstesa`

5. RICETTA

Per rappresentare le ricette è stato ideato il modello `ricetta` con le seguenti “proprietà”:

- `nome`: è il nome della ricetta
- `autore`: è l’username dell’autore della ricetta
- `statistica`: contiene i dati della ricetta (tempo, difficoltà, costo)
- `filtrri`: contiene i filtri della ricetta
- `rating`: votazione degli utenti alla ricetta

```
_id: ObjectId('639cd417d099f3b10a5e238c')
nome: "Tiramisù"
autore: "Maria"
✓ statistica: Array
  0: 100
  1: 1
  2: 1
✓ filtrri: Array
  0: "senza uova"
  1: "dessert"
rating: 4
```

Figura 10: Tipo di dato `ricetta`

2.4. PROJECT APIs

In questo paragrafo vengono presentate le API dell'applicazione JustCook. Prima vengono descritte le risorse API attraverso la lingua italiana e dei diagrammi in Unified Modeling Language (UML). Quest'ultimi sono il Resources Extraction Diagram (dal diagramma delle classi) e il Resources Model. Successivamente viene mostrato il codice con cui vengono implementate alcune delle API descritte.

2.4.1. RESOURCES EXTRACTION

In questa sezione vengono presentate e descritte le risorse API ideate a partire dai metodi delle classi del Class Diagram (analizzato nel documento precedente). Successivamente viene mostrato il Resources Extraction Diagram.

Descrizione risorse API estratte

1. CREDENZIALI INIZIALI - METODO GET CREDENZIALI

Analizzando la classe PaginaRegistrazione e il metodo `getCredenziali` è stata ideata l'API **Credenziali iniziali**. L'utente nella fase di creazione del proprio account deve inserire l'username, la password e l'indirizzo email associato in delle apposite barre. L'API analizzata possiede il metodo HTTP GET per ottenere e validare le prime credenziali appena inserite.

2. REGISTRAZIONE - METODO CREAZIONE ACCOUNT

Analizzando il metodo `creazioneAcount` della classe Registrazione è stata ideata l'API **Registrazione**. Quest'ultima possiede il metodo HTTP POST perchè deve creare un nuovo account per l'utente che ha effettuato la registrazione correttamente.

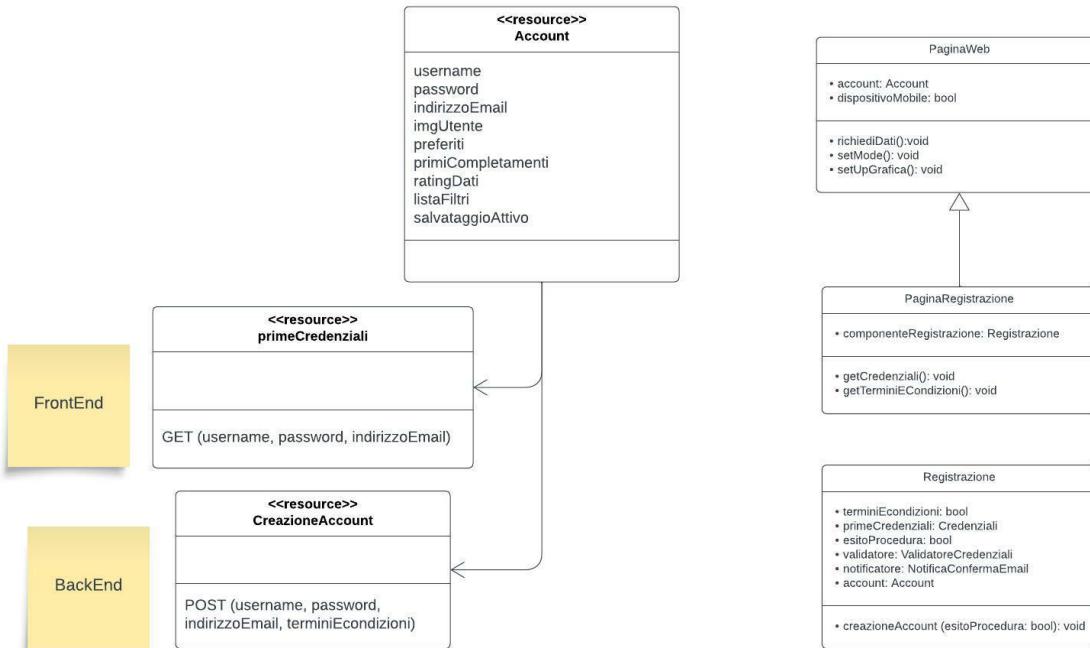


Figura 11: Classi e Resources Extraction Diagram registrazione

3. NUOVE CREDENZIALI - METODO GET CREDENZIALI

Dalla pagina dell'account è possibile modificare le credenziali (username, password, indirizzo email). Si possono cambiare inserendo i nuovi dati nelle apposite barre. Per questo è stato creato il metodo `getCredenziali` della classe `Account`. Di conseguenza è stata ideata l'API **Nuove credenziali** con il metodo HTTP `GET` per ottenere i dati e valutarli.

4. MODIFICA CREDENZIALI - METODO MODIFICA CREDENZIALI

La classe `GestoreModificaDati` possiede il metodo `modificaCredenziali`. Per questo è stata ideata l'API **Modifica credenziali** che si occupa di cambiare i dati in questione con quelli inseriti nella pagina dell'account. Possiede il metodo HTTP `PUT`.

5. MODIFICA IMMAGINE PROFILO - METODO MODIFICA IMG PROFILO

Analizzando la classe `GestoreModificaDati` e il metodo `modificaImgProfilo` è stata ideata l'API **Modifica Immagine profilo**. L'utente nella propria pagina dell'account può modificare la foto in questione. Per cambiarla l'API considerata utilizza il metodo HTTP `PATCH`.

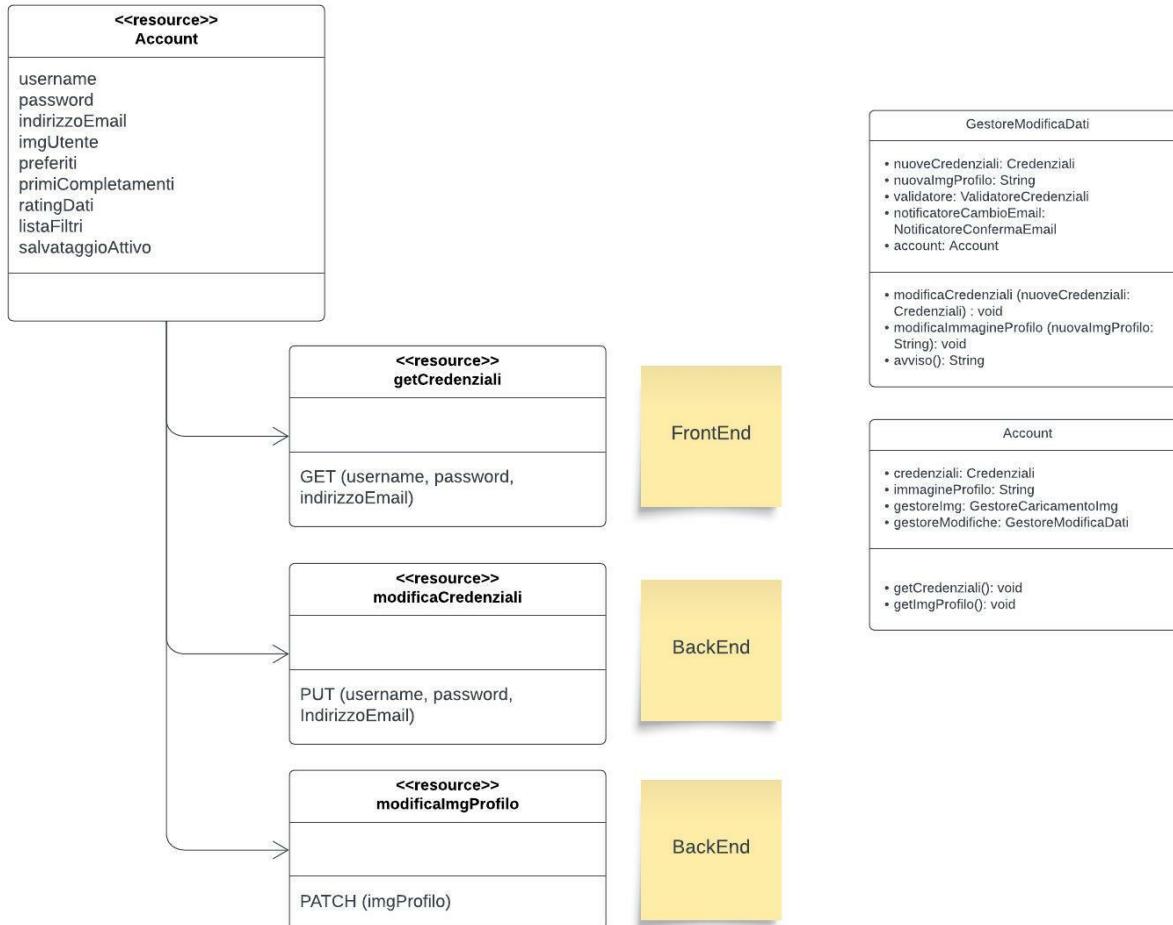


Figura 12: Classi e Resources Extraction Diagram modifica dati

6. SALVATAGGIO FILTRI - METODO SALVA FILTRI

Analizzando la classe `SalvataggioFiltri` e il suo metodo `salvaFiltrri` è stata individuata l'API **Salvataggio filtri**. L'utente nella pagina dell'account può decidere se avere dei filtri predefiniti nella ricerca delle ricette oppure no. Questa modifica viene effettuata con il metodo HTTP PATCH.

7. SELEZIONA FILTRI - METODO SELEZIONA FILTRO

L'utente quando aziona il salvataggio dei filtri può selezionare quelli che vuole avere attivi nella ricerca delle ricette. È possibile farlo attraverso il metodo `selezionaFiltro` della classe `SalvataggioFiltri`. Di conseguenza è stata ideata l'API **Selezione filtri** con il metodo HTTP PUT.

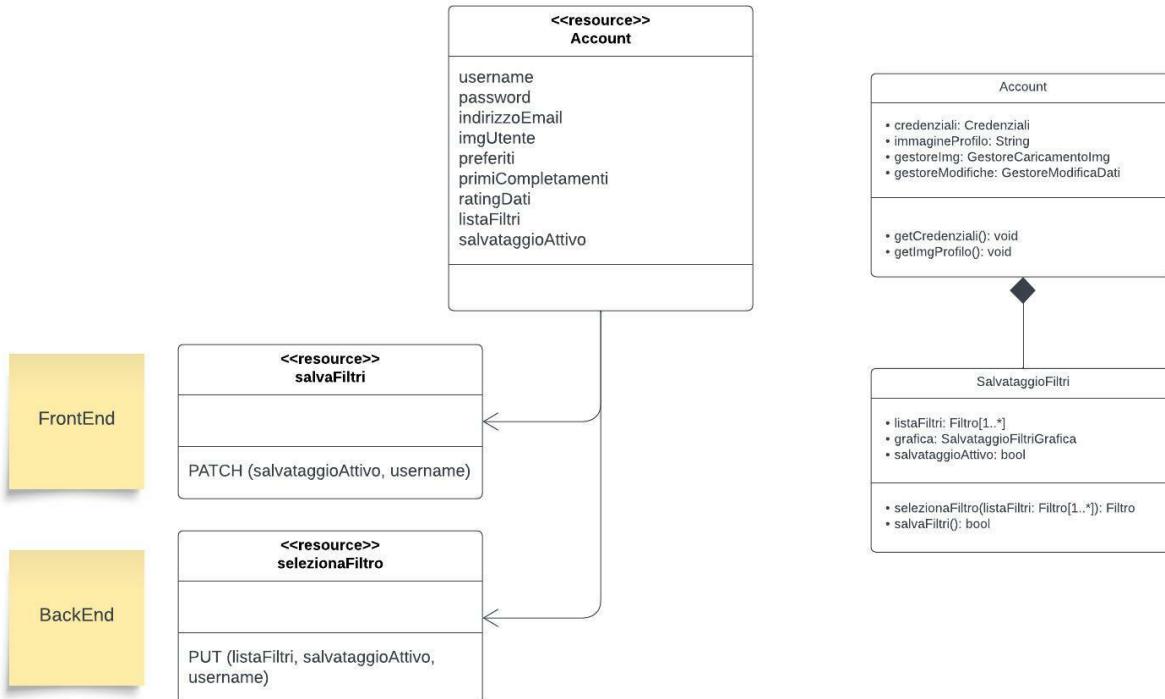


Figura 13: Classi e Resources Extraction Diagram salvataggio filtri

8. LOGIN - METODO FORNISCI ACCESSO

L'utente per accedere al proprio account deve inserire l'username e la password corretti. Per verificarne l'esattezza viene utilizzato il metodo `fornisciAccesso` della classe `Login`. Di conseguenza è stata creata l'API `Login` con il metodo HTTP `POST`.

9. EMAIL CAMBIO PASSWORD - METODO GET MAIL

Analizzando la classe `PaginaLogin` e il suo metodo `getMail` è stata ideata l'API **Email cambio password**. Quando l'utente desidera cambiare la password nella pagina del login deve inserire anche l'indirizzo email associato in un'apposita barra. Per prelevare l'indirizzo email e vedere se è associato ad un account, l'API considerata utilizza il metodo HTTP `GET`.

10. CAMBIO PASSWORD LOGIN - METODO CAMBIO CREDENZIALI

L'utente può cambiare la password dalla pagina del login. Per questo è stato ideato il metodo `cambioCredenziali` della classe `Login`. A quest'ultimo viene associata l'API **Cambio password login** con il metodo HTTP `PATCH`.

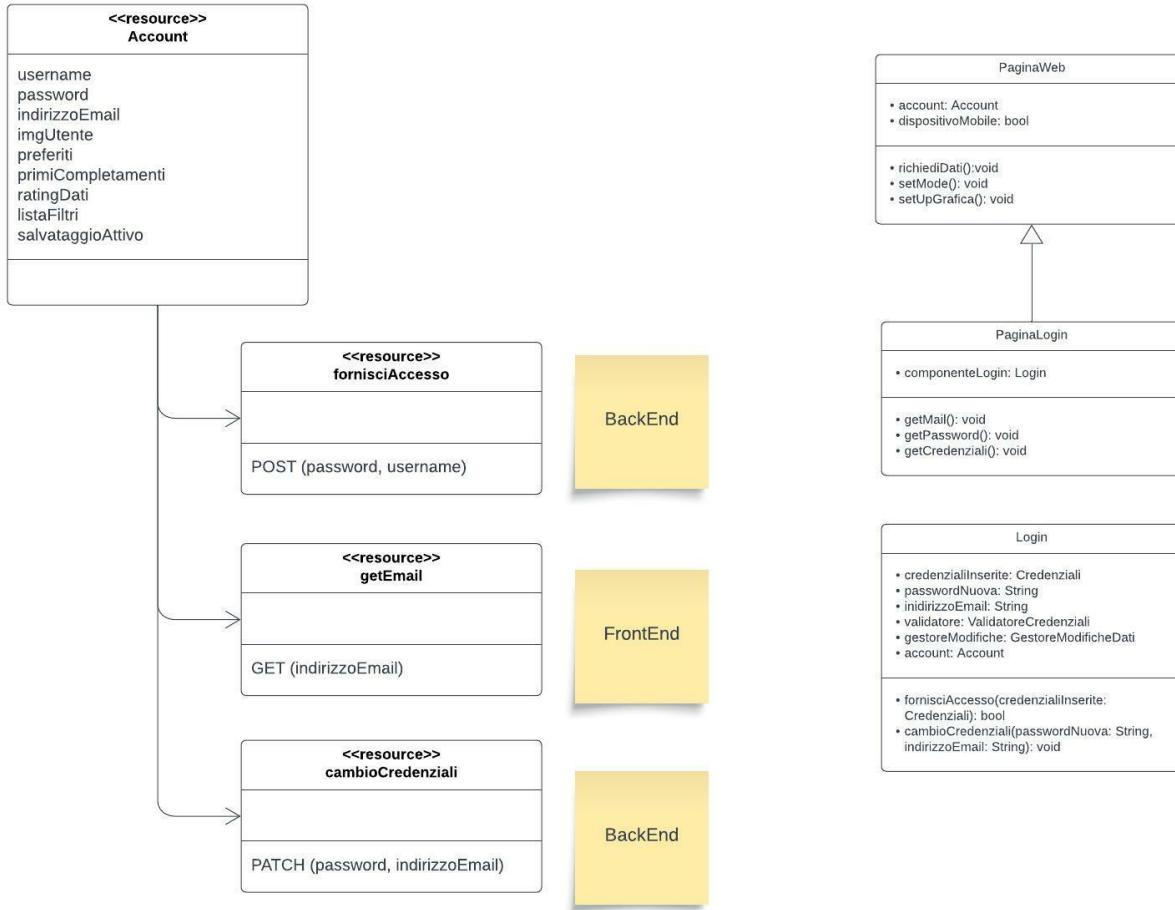


Figura 14: Classi e Resources Extraction Diagram login

11. CREA COMMENTO - METODO CREA COMMENTO

L'API **Crea commento** corrisponde al metodo `creaCommento` della classe `InterfacciaCreaCommento`. Attraverso il metodo HTTP `POST` aggiunge un nuovo commento alla relativa ricetta nel database.

12. ELIMINA COMMENTO - METODO ELIMINA COMMENTO

L'API **Elimina commento** corrisponde al metodo `eliminaCommento` della classe `InterfacciaCommento`. Attraverso il metodo HTTP `DELETE` viene eliminato il commento che corrisponde all'id passato dalla relativa ricetta nel database.

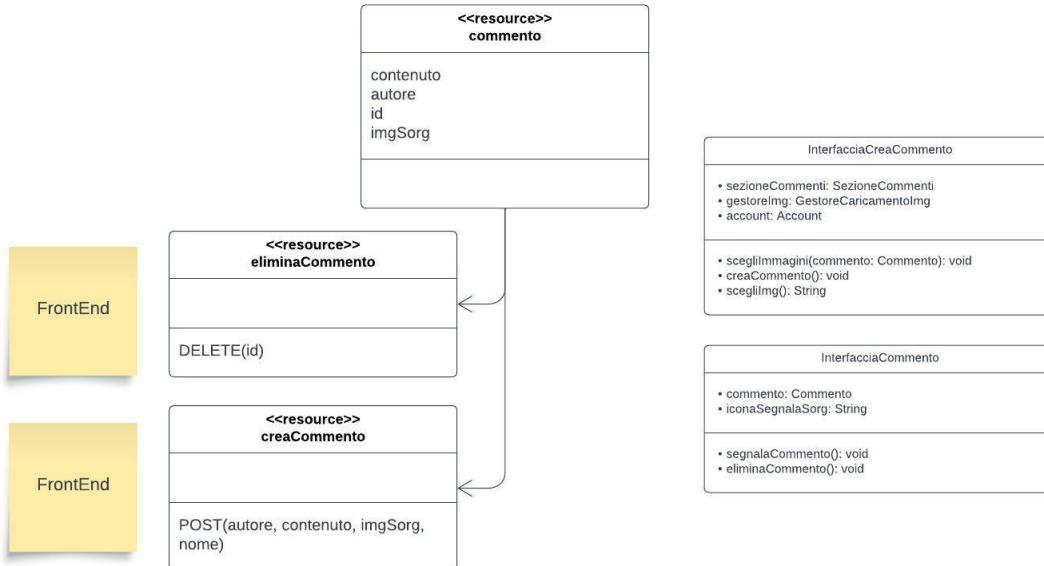


Figura 15: Classi e Resources Extraction Diagram commenti

13. MODIFICA QUANTITÀ - METODO MODIFICA QUANTITÀ

L'API **Modifica quantità** estratta dalla classe `Dispensa` è stata ideata allo scopo di modificare la quantità di un dato ingrediente in dispensa alla pressione di un tasto. Per implementare questa interazione dell'utente con il database si utilizza un metodo HTTP `PATCH` in modo che il resto della dispensa rimanga invariato.

14. AGGIUNGI INGREDIENTE - METODO AGGIUNGI INGREDIENTE

Sempre parte della classe `Dispensa`, analizzando il metodo `aggiungiIngrediente` si decide di implementare l'API **Aggiungi ingrediente**. Quest'ultima ha lo scopo di, una volta trovato l'ingrediente interessato fra tutti quelli disponibili, aggiungerlo alla propria dispensa. Per fare ciò si utilizza il metodo HTTP `PATCH`, in quanto nonostante un ingrediente venga aggiunto non si vuole perdere la dispensa già salvata.

15. ELIMINA INGREDIENTE - METODO ELIMINA INGREDIENTE

L'API **Elimina ingrediente** viene implementata analizzando la necessità, da parte sia dell'utente che del sistema, di eliminare un ingrediente dalla dispensa attuale. Per farlo viene usato il metodo HTTP `PATCH`, che viene chiamato alla pressione di un pulsante.

16. CERCA INGREDIENTE - METODO CERCA INGREDIENTE

L'API **Cerca ingrediente** viene ideata a partire dal metodo `cercaIngrediente` della classe `Dispensa`. Serve per verificare che l'ingrediente che si vuole aggiungere nella dispensa sia presente nel database di JustCook. Per fare questo utilizza il metodo HTTP GET.

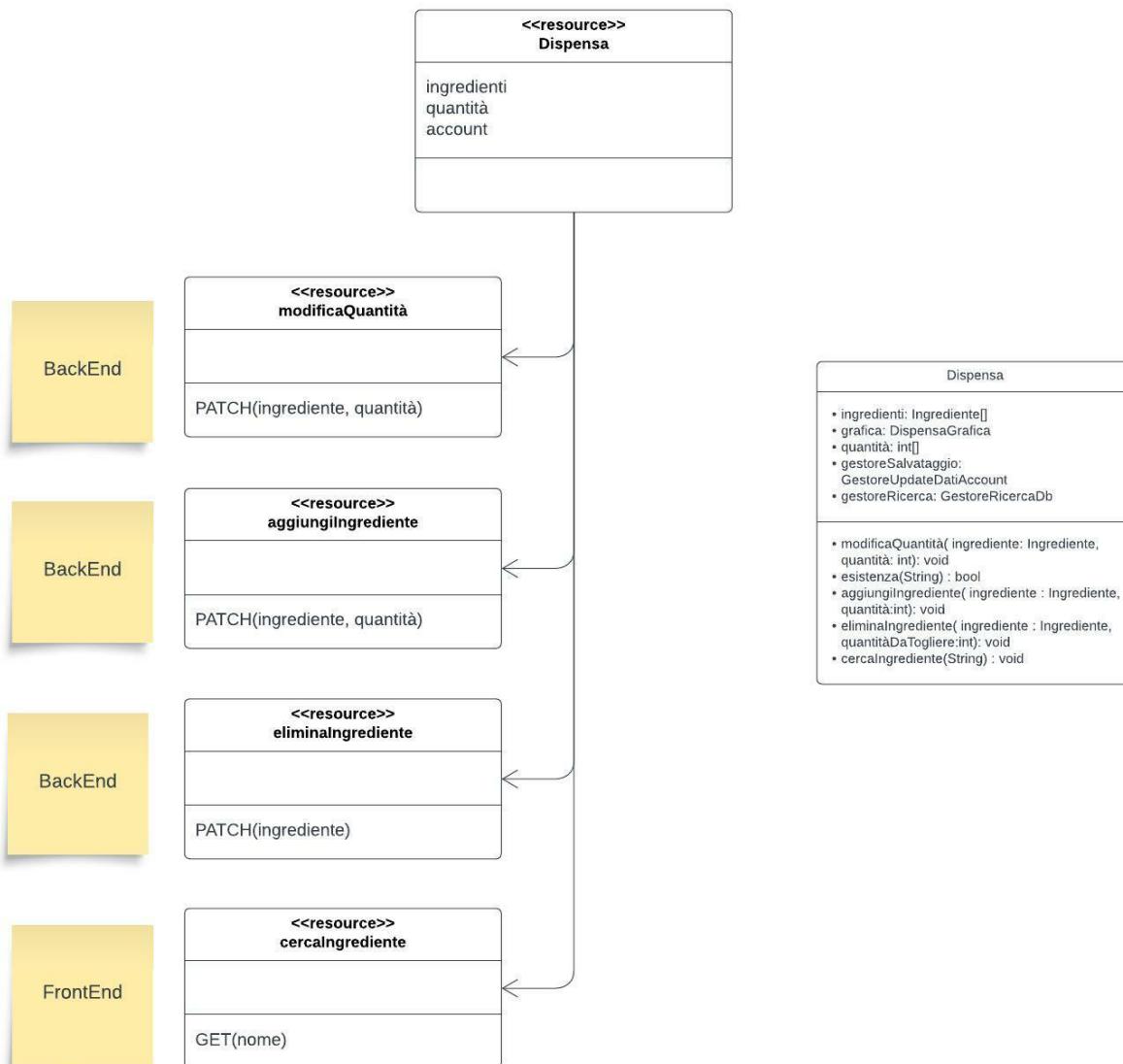


Figura 16: Classe e Resources Extraction Diagram interazione con dispensa

17. CERCA RICETTE - METODO CERCA RICETTE

Il metodo `cercaRicette` viene implementato con l'API **Cerca ricette** a partire dalla classe `RicercaRicette`. Analizzando la classe infatti si può notare la necessità di ricevere in ingresso alcune ricette a partire da dei criteri di ricerca. Questo metodo permette anche di cercare a partire dal nome della ricetta che si desidera leggere. Per implementare questa funzione viene usato un metodo HTTP GET.

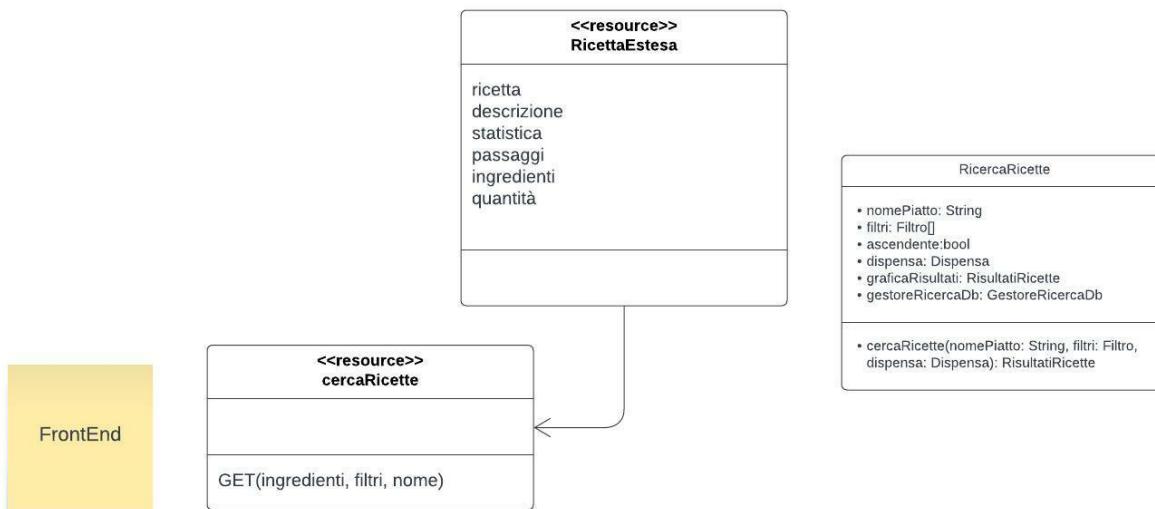


Figura 17: Classe e Resources Extraction Diagram ricerca ricette

18. COMPLETA RICETTA - METODO COMPLETA RICETTA

L'API **Completa ricetta** corrisponde, nel diagramma delle classi, al metodo `completaRicetta` della classe `RicettaEstesa`. Utilizza il metodo HTTP PATCH per aggiornare il database quando avviene il primo completamento di una determinata ricetta.

19. AGGIUNGI AI PREFERITI - METODO AGGIUNGI A PREFERITI

L'API **Aggiungi ai preferiti** corrisponde al metodo `aggiungiAPreferiti` della classe `RicettaEstesa`. Attraverso il metodo HTTP PATCH, modifica lo stato di una determinata ricetta per segnalare la sua presenza tra i preferiti dell'utente.

20. TOGLI DAI PREFERITI - METODO TOGLI DAI PREFERITI

L'API **Togli dai preferiti** corrisponde al metodo `togliDaiPreferiti` della classe `RicettaEstesa`. Attraverso il metodo HTTP `PATCH`, aggiorna lo stato della ricetta nel database per segnalare la sua assenza tra i preferiti dell'account.

21. AGGIUNGI RATING - METODO DAI RATING

L'API **Aggiungi rating** corrisponde al metodo `daiRating` della classe `ricettaEstesa`. Attraverso il metodo HTTP `PATCH` aggiunge il nuovo rating alla ricetta nel database.

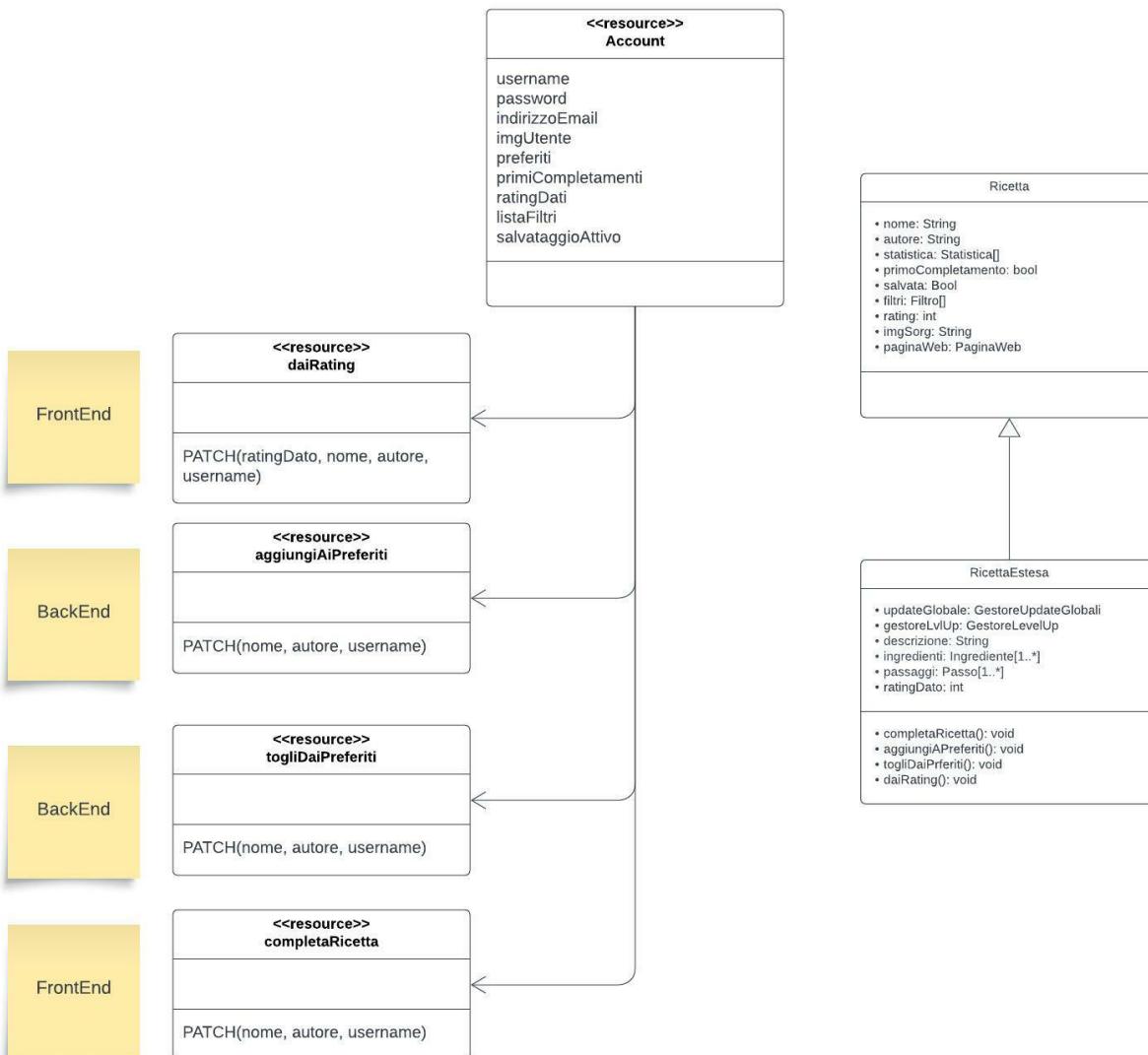


Figura 18: Classi e Resources Extraction Diagram interazione con ricetta

Diagramma Resources Extraction Diagram

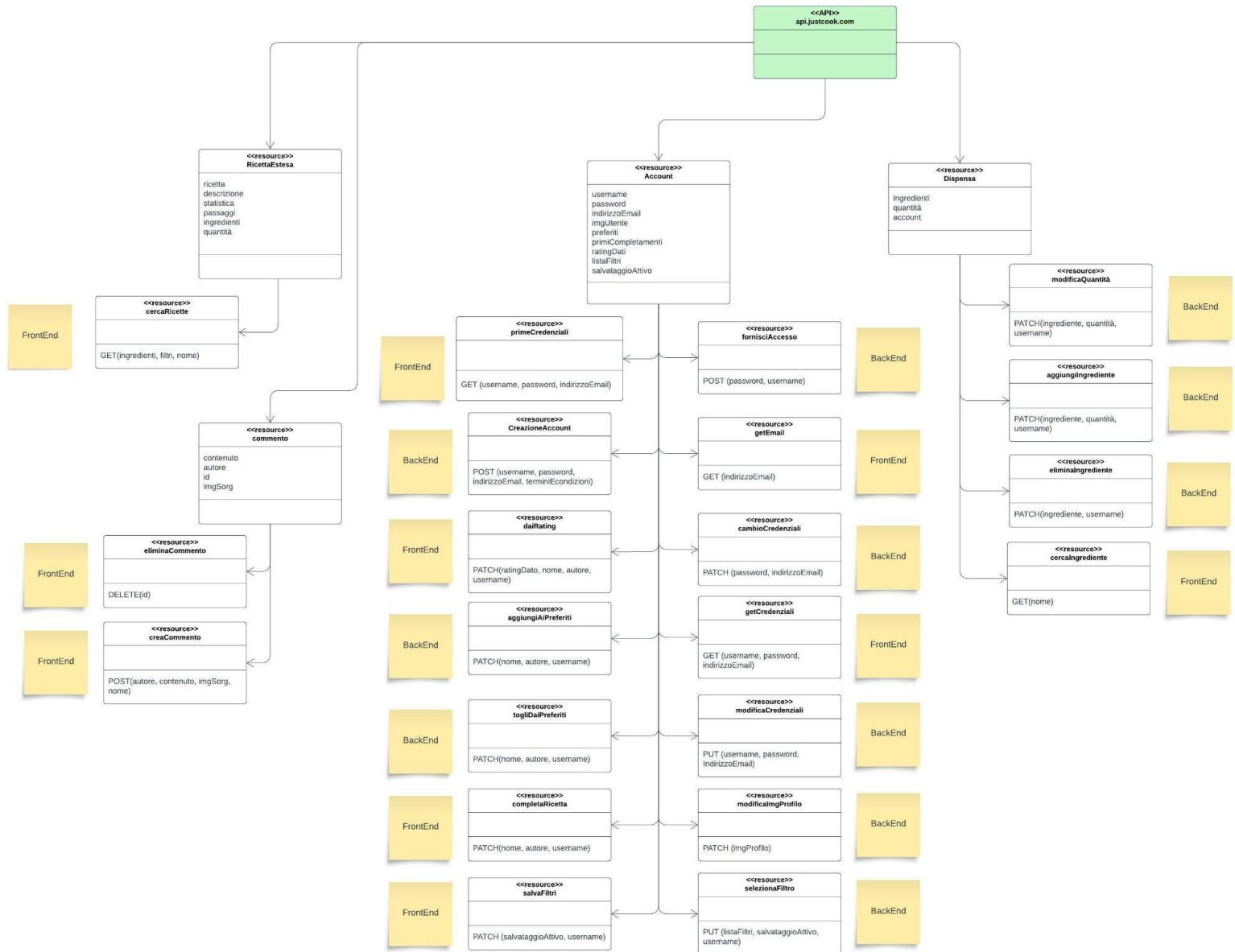


Figura 19: Resources Extraction Diagram

2.4.2. RESOURCES MODEL

In questa sezione vengono analizzati nel dettaglio i modelli delle risorse API descritte nel paragrafo precedente. I modelli vengono prima analizzati, successivamente ne vengono mostrati i diagrammi.

Descrizione modelli risorse API

1. CREDENZIALI INIZIALI

L'API **Credenziali iniziali** prende in input primeCredenziali che corrisponde alla password, allo username e all'indirizzo email inseriti dall'utente nella pagina di registrazione. La preleva attraverso il metodo HTTP GET. Se le istruzioni vengono eseguite correttamente viene ritornato il messaggio 200 OK e vengono passate le credenziali a chi si occupa di gestire la creazione dell'account ([risorsa API 2](#)). Nel caso non venissero trovati i dati ricercati viene ritornato il codice di risposta 404 NOT FOUND. Invece se la procedura presenta degli errori o i dati non sono validi il codice è 400 BAD REQUEST.

2. CREAZIONE ACCOUNT

L'API **Registrazione** si occupa di creare l'account attraverso il metodo HTTP POST. Per farlo prende in input primeCrenziali ([risorsa API 1](#)) e terminiEcondizioni (variabili booleana). Quest'ultima ha valore true se l'utente ha accettato i termini e le condizioni nella pagina della registrazione, false altrimenti. Se la procedura è corretta il codice di risposta 201 CREATED e verrà ritornato il nuovo account . Invece se sono presenti degli errori viene ritornato il messaggio 400 BAD REQUEST.

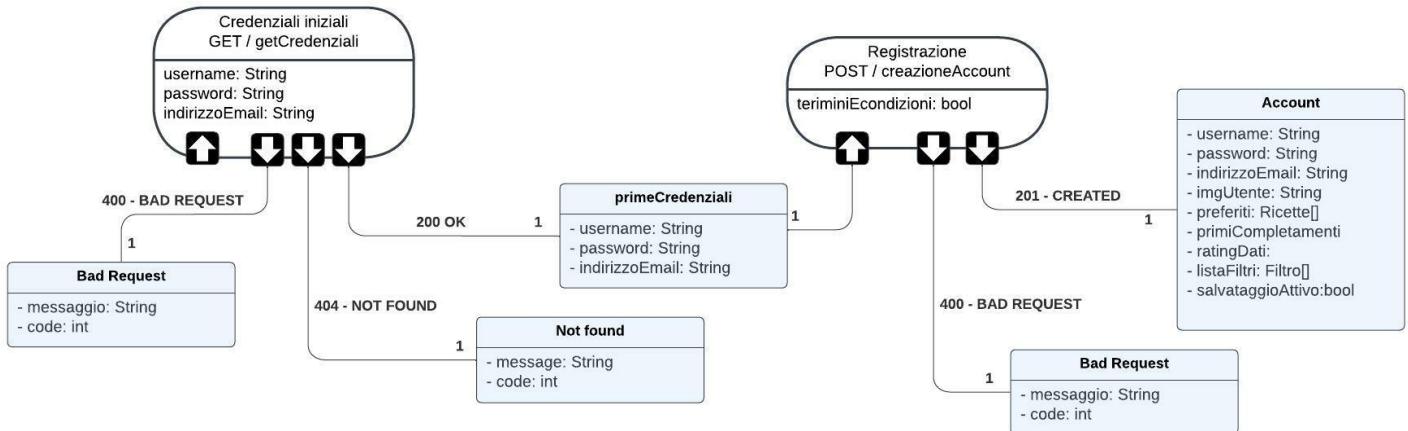


Figura 20: Resources Model Credenziali iniziali, Registrazione

3. NUOVE CREDENZIALI

L'API **Nuove credenziali** prende in input le credenziali dell'utente. Alcune di queste contengono i nuovi dati inseriti dall'utente nella pagina dell'account. Li recupera attraverso il metodo `HTTP GET`. Successivamente se l'operazione è stata eseguita correttamente verrà ritornato il messaggio `200 OK` e `nuoveCredenziali` verranno passate a chi si occupa di modificare i dati dell'account ([risorsa API 4](#)). Se i nuovi dati non vengono trovati sarà ritornato il codice di risposta `404 NOT FOUND`. Invece se l'operazione presenta degli errori o le nuove credenziali non sono valide il codice sarà `400 BAD REQUEST`.

4. MODIFICA CREDENZIALI

L'API **Modifica Credenziali** si occupa di modificare i dati dell'account attraverso il metodo HTTP `PUT`. Prende in input `nuoveCredenziali` ([risorsa API 3](#)). Se l'operazione avviene correttamente viene ricevuto il messaggio `200 OK` e `Account` con i dati modificati, altrimenti `400 BAD REQUEST`.

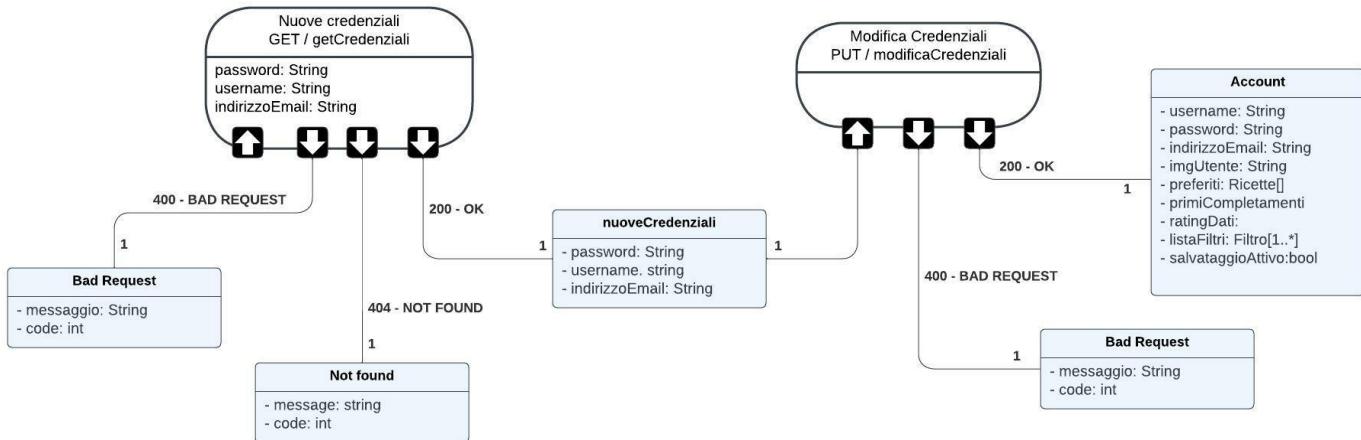


Figura 21: Resources Model Nuove credenziali, Modifica credenziali

5. MODIFICA IMMAGINE PROFILO

L'API **Modifica Immagine profilo** attraverso il metodo HTTP PATCH modifica l'immagine profilo. Prende in input nuovaImgProfilo che corrisponde alla nuova foto e username per identificare l'account da modificare. Se l'operazione è eseguita correttamente viene ritornato il messaggio 200 OK e Account con la foto modificata. Se l'operazione non è eseguita correttamente viene ritornato il codice di risposta 400 BAD REQUEST. Se l'immagine non viene trovata il codice è 404 NOT FOUND.

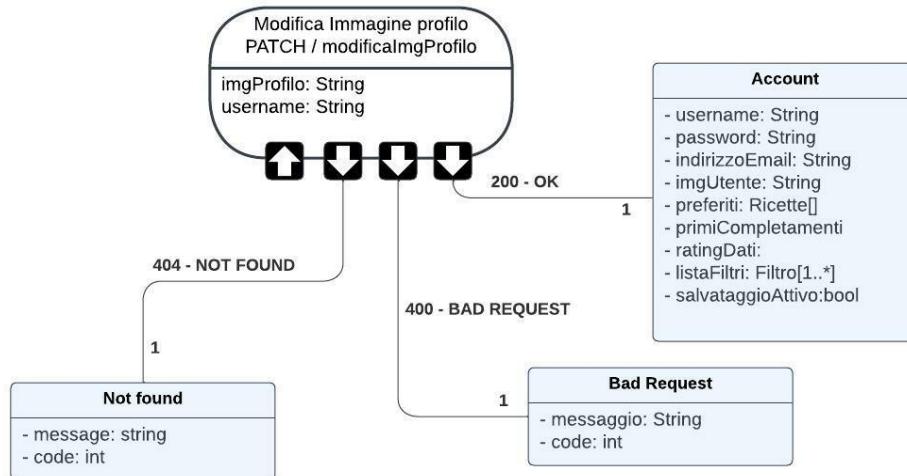


Figura 22: Resources Model Modifica Immagine profilo

6. SALVATAGGIO FILTRI

L'API **Salvataggio filtri** prende in input la variabile booleana `salvataggioAttivo`. Quest'ultima ha valore `true` se l'utente attiva il salvataggio dei filtri di ricerca, `false` altrimenti. L'altro parametro di input è `username`. Serve per identificare l'account di cui si vuole cambiare il salvataggio dei filtri. La modifica avviene attraverso il metodo HTTP `PATCH`. Se l'operazione eseguita è corretta viene ritornato il messaggio `200 OK` e `salvataggioAttivo` viene mandato a chi si occupa della selezione dei filtri ([risorsa API 7](#)). Invece, se il processo è errato, viene inviato il codice di risposta `400 BAD REQUEST`.

7. SELEZIONA FILTRI

L'API **Selezione filtri** utilizza il metodo HTTP `PUT` per cambiare i filtri da tenere nella ricerca delle ricette. Prende in input `salvataggioAttivo` ([API 6](#)), `username` (per identificare l'account da modificare) e `listaFiltri`. Quest'ultima contiene tutti i filtri presenti in JustCook. Se l'operazione di selezionamento va a buon fine viene ritornato il messaggio `200 OK` e `Account` con `listaFiltri` modificata. Invece, se l'operazione non va a buon fine, viene ricevuto il codice di risposta `400 BAD REQUEST`.

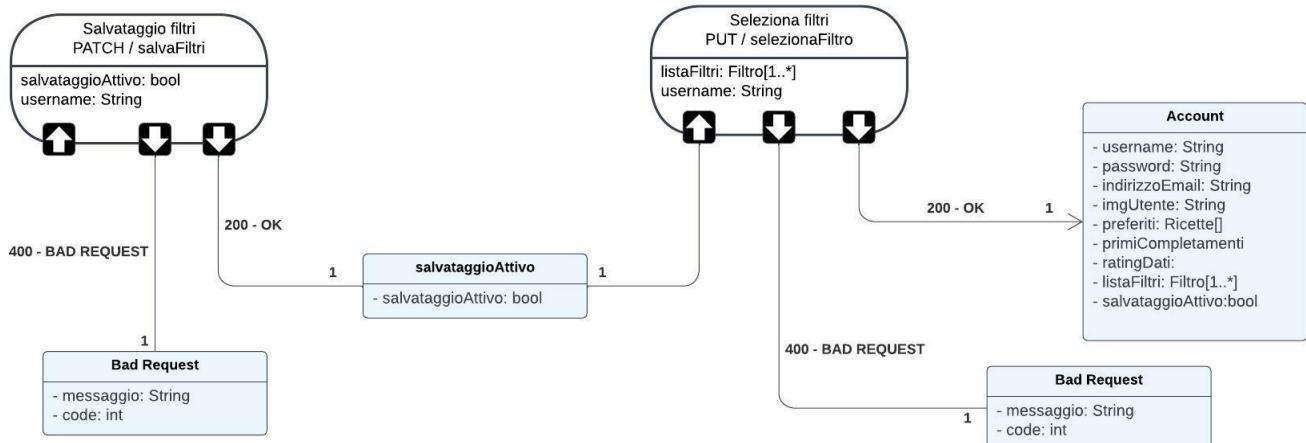


Figura 23: Resources Model Salvataggio filtri, Selezione filtri

8. LOGIN

L'API **Login** fornisce l'accesso all'utente nella fase di login attraverso il metodo HTTP POST. Riceve in input password e username. Se quest'ultime sono corrette (corrispondono ad un account) viene ritornato 200 OK con Account, altrimenti il codice di risposta è 400 BAD REQUEST. Se le credenziali inserite nella pagina del login dall'utente non vengono trovate il messaggio di risposta è 404 NOT FOUND.

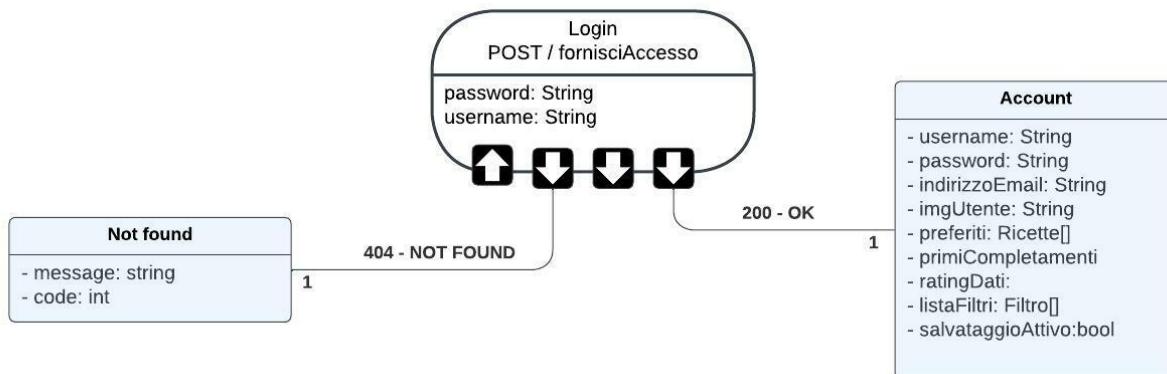


Figura 24: Resources Model Login

9. EMAIL CAMBIO PASSWORD

L'API **Email cambio password** prende in input indirizzoEmail contenente l'indirizzo email dell'utente inserito nella pagina del login. Viene prelevato attraverso il metodo HTTP GET. Se viene trovato il codice di risposta è 200 OK e indirizzoEmail viene inviato a chi si occupa di gestire il cambio password ([risorsa API 10](#)). Invece, se non viene trovato l'indirizzo email, il codice di risposta è 404 NOT FOUND. Se l'indirizzo email non è associato ad un account il codice è 400 BAD REQUEST.

10. CAMBIO PASSWORD LOGIN

L'API **Cambio password** login modifica la password dell'utente attraverso il metodo HTTP PATCH. Prende in input password (contiene la nuova password) e indirizzoEmail ([risorsa API 9](#)). Se l'operazione è corretta viene ritornato 200 OK con Account, altrimenti 400 BAD REQUEST (password nuova non valida). Se la password non è stata inserita il codice è 404 NOT FOUND.

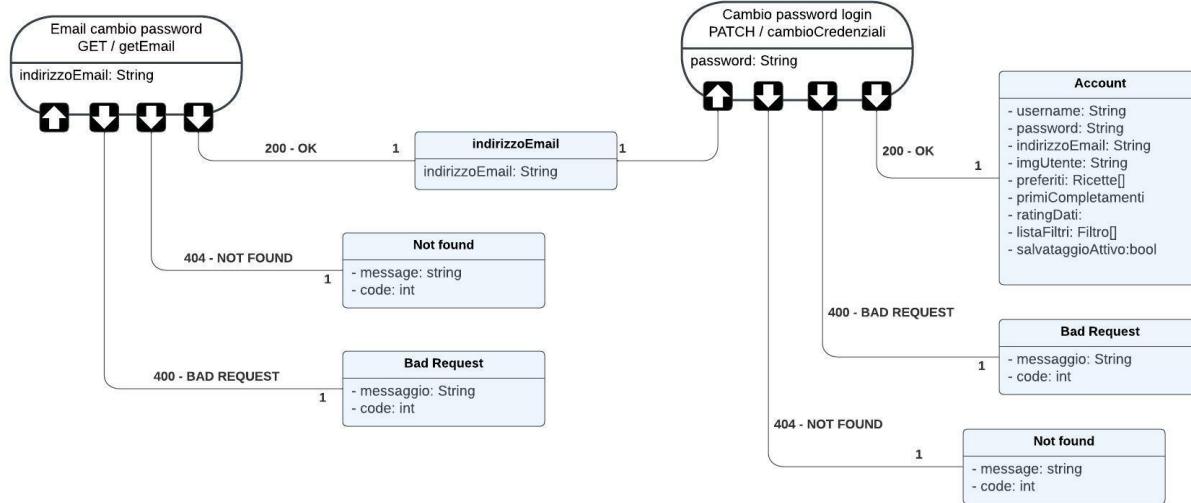


Figura 25: Resources Model Cambio password login, Email cambio password

11. CREA COMMENTO

L'API **Crea commento** prende in input il nome della ricetta su cui commentare. Inoltre prende in input il contenuto del commento, l'autore di quest'ultimo e l'eventuale immagine allegata. Poi attraverso il metodo HTTP POST viene aggiunto un nuovo commento alla ricetta nel database. Viene ritornato il messaggio di 201 CREATED e Commento alla buona riuscita dell'operazione e l'errore 404 NOT FOUND se non viene trovata la ricetta da commentare. Se l'operazione presenta degli errori il messaggio ritornato è 400 BAD REQUEST.

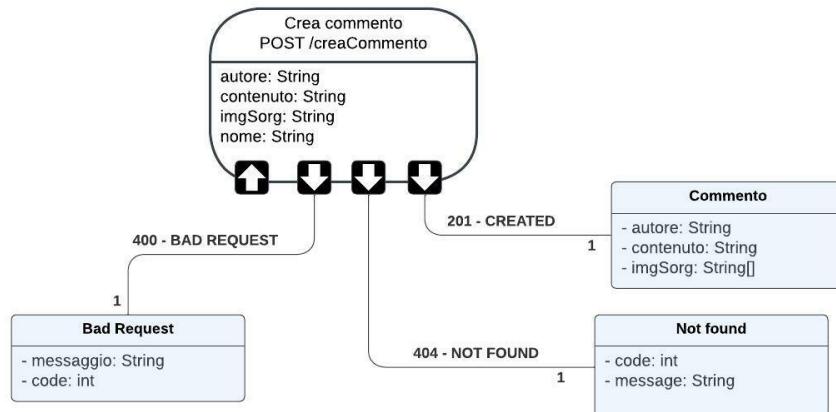


Figura 26: Resources Model Crea commento

12. ELIMINA COMMENTO

L'API **Elimina commento** prende in input l'id del commento da eliminare, poi attraverso il metodo HTTP `DELETE` viene eliminato il commento che corrisponde all'id passato dalla relativa ricetta nel database. Viene ritornato un messaggio di `200 OK` alla buona riuscita dell'operazione e l'errore `404 NOT FOUND` se non viene trovata la ricetta in cui è presente il commento. Invece se l'operazione presenta degli errori il codice è `400 BAD REQUEST`.

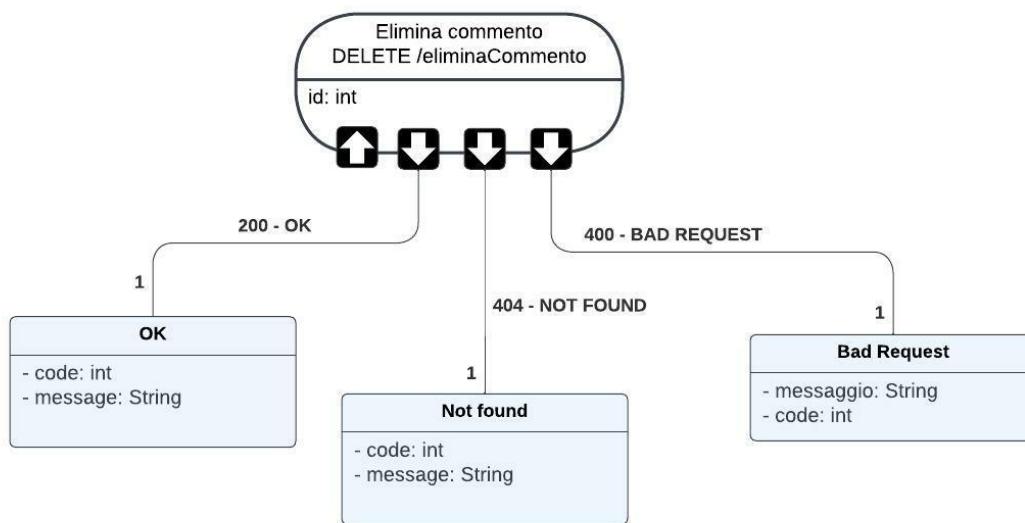


Figura 27: Resources Model Elimina commento

13. MODIFICA QUANTITÀ

L'API **Modifica quantità** viene implementata per permettere all'utente di diminuire o aumentare la quantità di un ingrediente alla pressione di un tasto sviluppato in front-end. Per funzionare richiede che in input gli venga passato l'ingrediente e la nuova quantità. Inoltre prende in ingresso `username`. Quest'ultimo serve per individuare la dispensa dell'utente che sta effettuando la modifica. Procede a fare una HTTP `PATCH` sulla quantità di un ingrediente. A operazione eseguita viene restituito un codice `200 OK` per confermare l'esito e restituire la dispensa modificata. In caso l'operazione subisca degli errori verrà restituito il codice `500 INTERNAL SERVER ERROR` con collegato messaggio di errore. In caso in cui si provi a modificare la quantità di un ingrediente non in dispensa il codice ritornato sarà `404 NOT FOUND`.

14. AGGIUNGI INGREDIENTE

Per aggiungere un determinato ingrediente nella dispensa di un utente si sviluppa l'API **Aggiungi ingrediente** di tipo HTTP PATCH. In ingresso viene richiesto l'ingrediente, la quantità associata da aggiungere e l'username dell'utente che sta effettuando l'aggiunta.. La funzione PATCH verrà applicata alla quantità legata a un determinato ingrediente e, una volta svolta la business logic, viene restituito in output il codice 200 OK per confermare l'operazione, allegando la dispensa modificata. Come per altri metodi, in caso l'operazione subisca degli errori verrà invece restituito il codice 500 INTERNAL SERVER ERROR. Nel caso in cui l'ingrediente che si prova ad aggiungere non esista, verrà restituito codice 404 NOT FOUND con allegato messaggio di avvenuto errore.

15. ELIMINA INGREDIENTE

L'API **Elimina ingrediente** utilizza il metodo HTTP PATCH per modificare i dati della dispensa. A questa API viene passato in input l'ingrediente che si desidera eliminare e l'username dell'utente che sta effettuando l'operazione. Al termine dell'esecuzione si otterrà un codice 200 OK che conferma l'eliminazione dell'ingrediente e restituisce la dispensa modificata. In caso la procedura sia errata si ottiene il codice 500 INTERNAL SERVER ERROR, se invece si passa un ingrediente o una dispensa non validi viene ritornato codice 404 NOT FOUND.

16. RICERCA INGREDIENTE

L'API **Cerca ingrediente** prende in input il nome dell'ingrediente inserito dall'utente nell'apposita barra di ricerca per aggiungerlo nella dispensa . Per fare questo l'API utilizza il metodo HTTP GET. Se l'operazione va a buon fine viene restituito il messaggio 200 OK con Ingrediente. Se il nome del cibo non è stato inserito correttamente nella barra di ricerca o l'ingrediente cercato non è presente nel database viene ritornato l'errore 404 NOT FOUND.

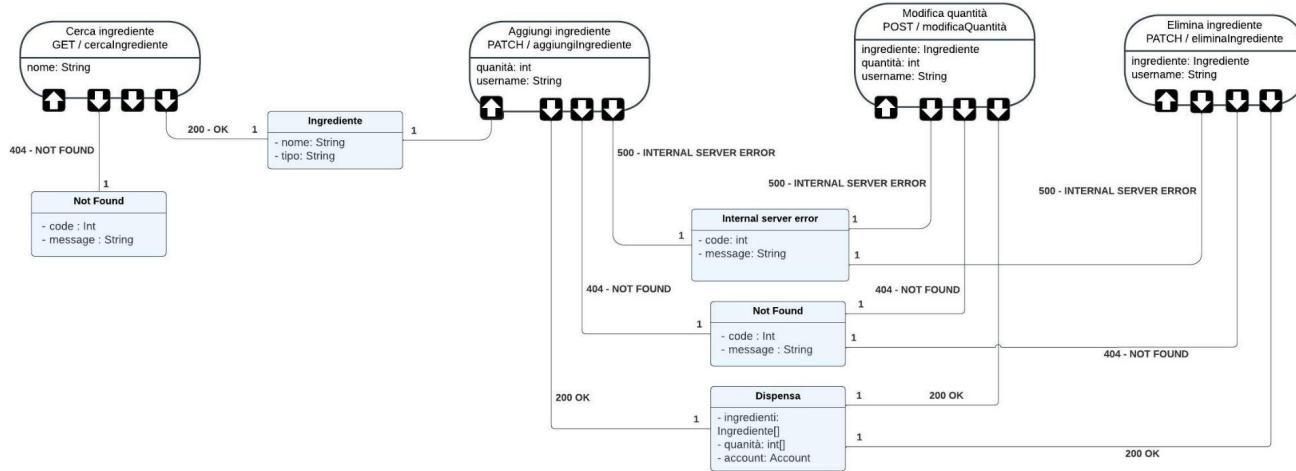


Figura 28: Resources Model Cerca ingrediente, Aggiungi ingrediente, Modifica quantità, Elimina ingrediente

17. CERCA RICETTE

L'API **Cerca ricette** prende in input `ingredienti` che rappresenta gli ingredienti nella dispensa dell'utente, `filtri` che corrisponde ai filtri selezionati nella ricerca e `nome` che è il nome del piatto ricercato (può anche non essere specificato). Se l'operazione non presenta errori viene ricevuto il codice di risposta 200 OK con `listaRicette` che rappresenta i risultati della ricerca. Se ci sono stati degli errori durante l'esecuzione il codice ricevuto è 500 INTERNAL SERVER ERROR.

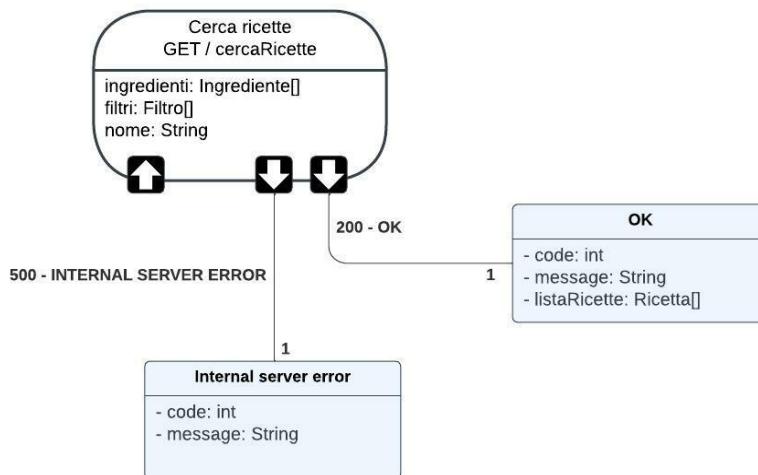


Figura 29: Resources Model Cerca ricette

18. COMPLETA RICETTA

L'API **Completa ricetta** prende in input il nome e l'autore della ricetta che è stata appena "completata". Prende in ingresso anche `username` per identificare l'utente che ha completato la ricetta. L'API attraverso il metodo `HTTP PATCH` aggiorna i completamenti dell'account dell'utente. All'esecuzione corretta delle istruzioni viene ritornato un messaggio di `200 OK`, mentre se non viene trovata la ricetta cercata si riscontra l'errore `404 NOT FOUND`. Inoltre se la procedura presenta errori viene ritornato il codice `500 INTERNAL SERVER ERROR`.

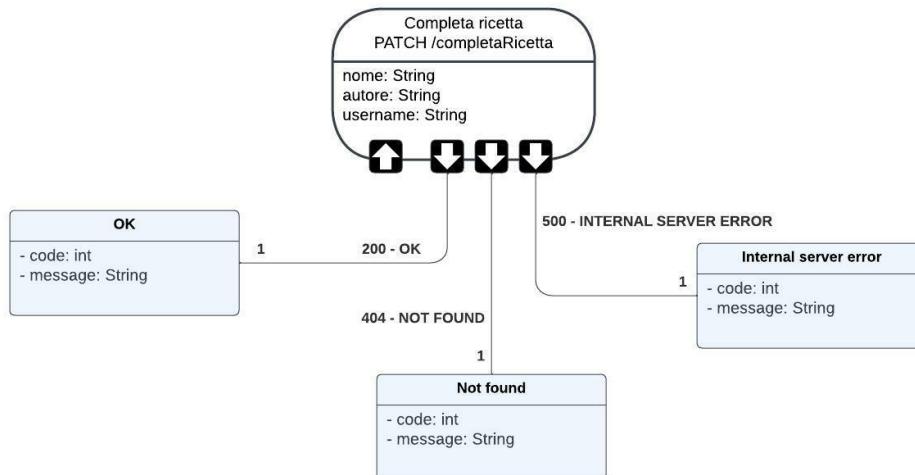


Figura 30: Resources Model Completa ricetta

19. AGGIUNGI AI PREFERITI

L'API **Aggiungi ai preferiti** prende in input il nome e l'autore della ricetta da aggiungere tra i preferiti dell'utente. Inoltre prende in ingresso `username` per identificare l'utente che ha aggiunto la ricetta come preferita. Attraverso il metodo `HTTP PATCH` viene aggiornato l'elenco delle ricette preferite dell'account dell'utente. Viene ritornato un messaggio di `200 OK` alla buona riuscita dell'operazione e l'errore `404 NOT FOUND` se non viene trovata la ricetta cercata. Se la ricetta è già presente nei preferiti viene ritornato il codice `400 BAD REQUEST`. Invece se la procedura presenta degli errori il codice ritornato è `500 INTERNAL SERVER ERROR`.

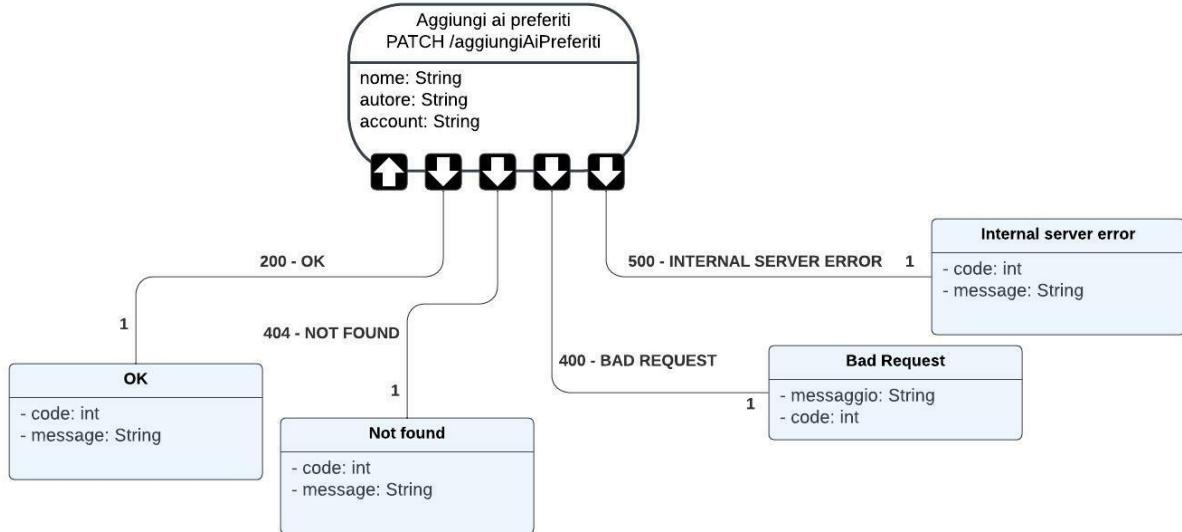


Figura 31: Resources Diagram Aggiungi ai preferiti

20. TOGLI DAI PREFERITI

L'API **Togli dai preferiti** prende in input il nome e l'autore della ricetta da togliere dai preferiti dell'utente (identificato attraverso `username`). Attraverso il metodo HTTP PATCH viene aggiornata la lista dei preferiti dell'utente rimuovendo la ricetta in questione. Viene ritornato un messaggio 200 OK alla buona riuscita dell'operazione e l'errore 404 NOT FOUND se non viene trovata la ricetta da togliere dai preferiti. Inoltre se l'operazione presenta degli errori la risposta ricevuta è 500 INTERNAL SERVER ERROR.

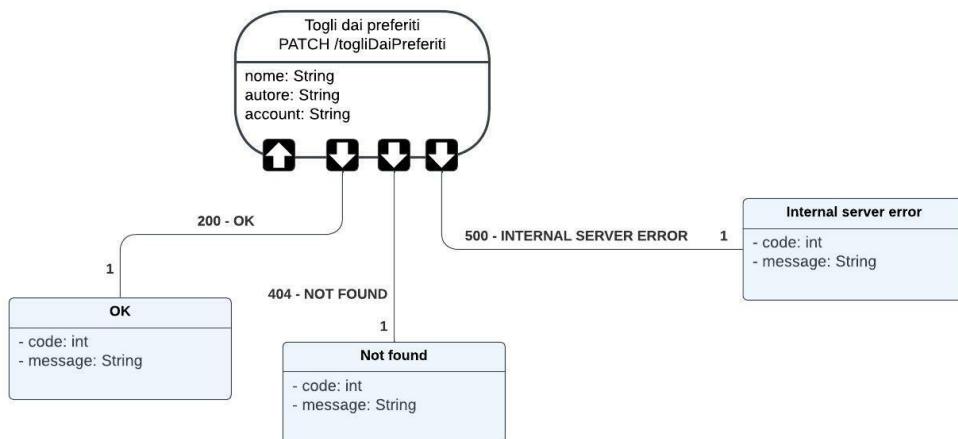


Figura 32: Resources Model Togli dai preferiti

21. AGGIUNGI RATING

L'API **Aggiungi rating** prende in input il nome e l'autore della ricetta di cui salvare il rating dato dall'utente (identificato da `username`) e la valutazione stessa appena fornita come intero da 0 a 5. Attraverso il metodo `HTTP PATCH` viene aggiunto il nuovo rating alla ricetta nel database e aggiunto il rating all'account dell'utente. Al buon termine dell'operazione viene inviato un messaggio di `200 OK`. Se non viene trovata la ricetta cercata si riscontra l'errore `404 NOT FOUND`. Se il rating è già stato dato viene ritornato il codice `400 BAD REQUEST`. Infine se la procedura presenta degli errori il messaggio di risposta è `500 INTERNAL SERVER ERROR`.

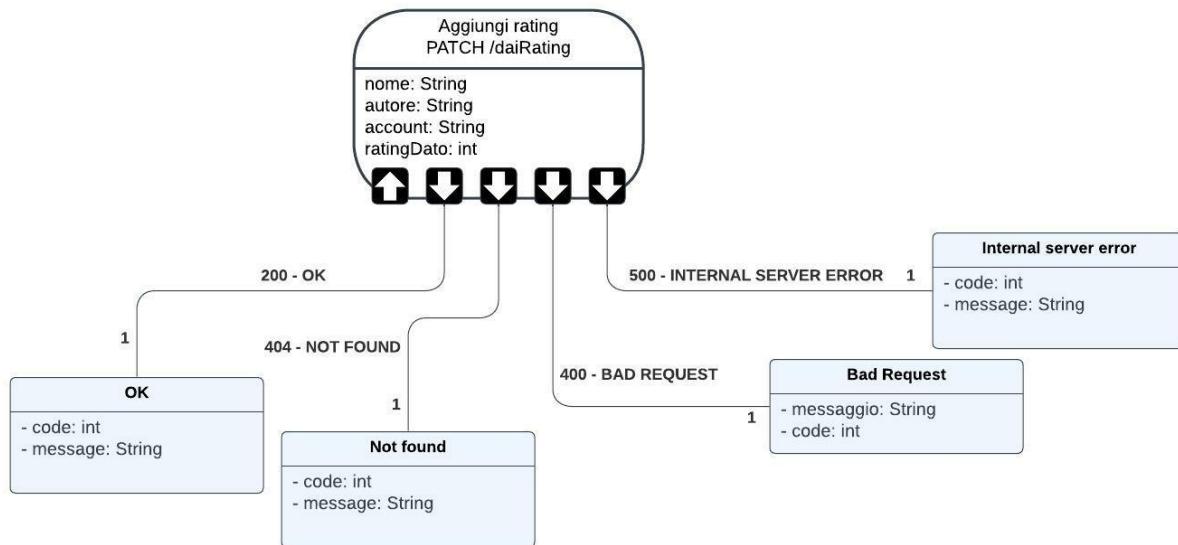


Figura 33: Resources Model Aggiungi rating

2.5. SVILUPPO API

In questo paragrafo vengono presentate le API sviluppate a partire dai Resources Extraction Diagram e dai Resources Model analizzati precedentemente. Viene mostrato il codice accompagnato da una breve descrizione.

Descrizione API sviluppate

1. LOGIN - RISORSA LOGIN

L'API **login** consente agli utenti che possiedono un account di accedervi. Preleva `password` e `username` inseriti dall'utente dal body. Successivamente verifica che esista un account con quelle credenziali. Nel caso non esistesse l'account viene ritornato un errore, altrimenti l'account trovato.

```
router.post('/login', authenticationController.login);
```

Figura 34: API login router

```
const login = async (req, res) => {
  const {username, password} = req.body;
  const account = await Account.findOne({username: username, password: password});
  if (account) {
    res.json(account);
  } else {
    res.json({error: "Account non trovato", code: 404});
  }
};
```

Figura 35: API login controller

2. GET MAIL - RISORSA EMAIL CAMBIO PASSWORD

L'API **getMail** preleva e analizza l'indirizzo email inserito dall'utente nella pagina del cambio password. Prende `indirizzoEmail` come parametro e verifica che esista nel database un account associato alla mail. Se non esiste quest'ultimo o l'utente non inserisce l'indirizzo email viene ritornato errore. Se l'operazione non subisce errori viene ritornato l'indirizzo email.



```
router.get('/account/:indirizzoEmail', accountController.getMail);
```

Figura 36: API getMail router

```
const getMail = (req, res) => {
  let mail = req.params.indirizzoEmail;
  if(!mail) return res.json({error:"Email non inserita", code:404});
  Account.findOne({indirizzoEmail: mail}, (err, Account) =>{
    if (!Account|| err) return res.json({error: "Email non valida", code: 400});
    | return res.json(Account.indirizzoEmail);
  });
};
```

Figura 37: API getMail controller

3. PATCH PASSWORD - RISORSA CAMBIO PASSWORD LOGIN

L'API **patchPassowrd** è necessaria per cambiare la password dell'utente prima di aver effettuato il login. Prende come parametri l'indirizzo email passato da [getMail \(punto 2\)](#) e la password nuova. Quest'ultima viene analizzata per capire se è valida (minimo 8 caratteri, almeno una lettera minuscola, maiuscola, un carattere speciale e un numero). Se non è valida o se la password non è stata inserita, se ci sono problemi con l'indirizzo email o la modifica non avviene correttamente vengono ritornati dei codici di errore. Altrimenti la nuova password viene salvata nel database.

```
router.patch('/account/:indirizzoEmail/:password', accountController.patchPassword);
```

Figura 38: API patchPassword router

```
const patchPassword = (req, res) => {

  let mail = req.params.indirizzoEmail;
  let password = req.params.password;

  if(!password) return res.json({error: "Password nuova non inserita", code: 404});

  var strongRegex = new RegExp("^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.*[!@#$%^&*])(?=.{8,})");

  if (!strongRegex.test(password)) return res.json({error: "Nuova password non valida", code: 400});

  Account.findOne({indirizzoEmail: mail}, (err, Account) =>{
    if (err) return res.json({error: "Operazione non riuscita", code: 400});
    Account.password = password;
    Account.save((err, data) => {
      if (err) return res.json({error: "Operazione non riuscita", code: 400});
      res.json(data);
    });
  });
};
```

Figura 39: API patchPassword controller

4. AGGIUNGI INGREDIENTE - RISORSA AGGIUNGI INGREDIENTE

L'API **aggiungiIngrediente** si occupa di gestire l'aggiunta degli ingredienti nella dispensa di un utente. Preleva dal body l'username dell'account e l'id dell'ingrediente da aggiungere. Se quest'ultimo non è presente nella dispensa dell'utente viene inserito con la quantità (prelevata sempre dal body) corrispondente. Quest'ultima viene aggiunta a quella presente nella dispensa se l'ingrediente è già stato aggiunto. Possono essere ritornati dei messaggi di errore e dei codici di risposta se gli elementi analizzati precedentemente non vengono trovati o se ci sono dei problemi durante l'esecuzione del codice. Nel caso non ci fossero errori viene aggiornata la dispensa nel database.

```
router.patch('/dispensa/aggiungiIngrediente', dispensaController.aggiungiIngrediente);
```

Figura 40: API aggiungiIngrediente router

```
const aggiungiIngrediente = (req, res) => {
  let nomeAccount = req.body.nome;
  let idIngrediente = mongoose.Types.ObjectId(req.body.idIngrediente);

  Account.findOne({username: nomeAccount}, (err, account) => {
    if (!account || err) return res.json({error: "Account non trovato", code: 404});
    Dispensa.findOne({account: account._id}, (err, dispensa) => {
      if (!dispensa || err) return res.json({error: "Dispensa non trovata", code: 404});
      let index = dispensa.ingredienti.indexOf(idIngrediente);
      if (index == -1) {
        dispensa.ingredienti.push(idIngrediente);
        dispensa.quantita.push(req.body.quantita);
      } else {
        let quantita_attuale = dispensa.quantita[index];
        dispensa.quantita[index] = Number(quantita_attuale) + Number(req.body.quantita);
      }
      dispensa.save((err, data) => {
        if (err) return res.json({error: "Errore nell'aggiunta", code: 500});
        res.json(data);
      });
    });
  });
};
```

Figura 41: API aggiungiIngrediente controller

5. ELIMINA INGREDIENTE - RISORSA ELIMINA INGREDIENTE

L'API **eliminaIngrediente** è necessaria per l'eliminazione degli ingredienti dalle dispense degli utenti. Preleva dal body lo username dell'utente che sta eseguendo l'operazione e l'id dell'ingrediente da eliminare. Se trova l'account e la dispensa associata toglie l'ingrediente con la sua quantità. Infine ritorna la dispensa aggiornata al database. Se la procedura presenta dei problemi vengono ritornati dei messaggi di errore con i rispettivi codici.

```
router.delete('/dispensa/eliminaIngrediente', dispensaController.eliminaIngrediente);
```

Figura 42: API eliminaIngrediente router

```
const eliminaIngrediente = (req, res) => {
  let nomeAccount = req.body.nome;
  console.log(nomeAccount);
  let idIngrediente = req.body.idIngrediente;
  Account.findOne({username: nomeAccount}, (err, account) => {
    if (!account || err) return res.json({error: "Account non trovato", code: 404});
    Dispensa.findOne({account: account._id}, (err, dispensa) => {
      if (!dispensa || err) return res.json({error: "Dispensa non trovata", code: 404});
      let index = dispensa.ingredienti.indexOf(idIngrediente);
      dispensa.ingredienti.splice(index, 1);
      dispensa.quantita.splice(index, 1);
      dispensa.save((err, data) => {
        if (err) return res.json({error: "Errore nell'aggiunta", code: 500});
        res.json(data);
      });
    });
  });
};
```

Figura 43: API eliminaIngrediente controller

6. CERCA INGREDIENTE - RISORSA RICERCA INGREDIENTE

L'API `cercaIngrediente` serve per verificare che un ingrediente che si vuole inserire nella dispensa esista nel database. Prende come parametro il nome inserito dall'utente nella barra di ricerca della dispensa. Se l'ingrediente è presente nel database viene restituito, altrimenti viene ritornato un messaggio di errore con il codice corrispondente.

```
router.get('/ingrediente/:nome', ingredienteController.cercaIngrediente);
```

Figura 44: API cercaIngrediente route

```
const cercaIngrediente = (req, res) => {
  let nomeIngrediente = req.params.nome;
  Ingrediente.findOne({nome: nomeIngrediente}, (err, Ingrediente) => {
    if (!Ingrediente || err) return res.json({error: "Ingrediente non trovato", code: 404});
    res.json(Ingrediente);
  });
};
```

Figura 45: API cercaIngrediente controller

7. CERCA RICETTE - RISORSA CERCA RICETTE

L'API **cercaRicette** è utilizzata per la ricerca delle ricette nel db. Estraе dalla query una lista di stringhe per gli ingredienti, una lista di stringhe per i filtri e una stringa per il nome della ricetta. Se la lista degli ingredienti è vuota allora ritorna un warning indicando all'utente che la dispensa è vuota. La lista di filtri e il nome della ricetta non sono necessari e possono dunque essere vuoti o nulli. Le ricette nel database vengono poi filtrate e ritornate come risultato.

```
router.get('/cercaRicette/:cerca', ricettaEstesaController.cercaRicette);
```

Figura 46: API cercaRicette router

```

const cercaRicette = (req, res, next) => {
  let ingredientiLista;
  let filtriLista;

  if(req.query.ingredienti == undefined)
    ingredientiLista = []
  else{
    ingredientiLista = req.query.ingredienti.split(',');
    if(!Array.isArray(ingredientiLista))
      ingredientiLista = [ingredientiLista]
  }

  if(req.query.filtri == undefined)
    filtriLista = []
  else{
    filtriLista = req.query.filtri.split(',');
    if(!Array.isArray(filtriLista))
      filtriLista = [filtriLista]
  }

  let nomeRicetta = req.query.nome;
  let ingredientiBasici = ["Sale", "Zucchero", "Acqua", "Olio"];

  if(ingredientiLista.length == 0){
    console.log("Ricerca impossibile, non esiste alcun ingrediente nella dispensa")
    return res.json({message: "Warning: ricerca impossibile, non esiste alcun ingrediente nella dispensa"})
  }else{
    ingredientiLista = ingredientiLista.concat(ingredientiBasici);
    ingredientiLista.sort();
    let stringaPrec = ingredientiLista[0];

    for(let i = 0; i < ingredientiLista.length -1; i++){
      if(ingredientiLista[i+1] == stringaPrec){
        ingredientiLista.splice(i, 1);
      }
      stringaPrec = ingredientiLista[i];
    }

    RicettaEstesa.aggregate([
      { $lookup: {from: "ingredientes", localField: "ingredienti", foreignField: "_id", as: "ingredientiInfo"}},
      { $match: { $expr: {$setIsSubset: ["$ingredienteInfo.nome", ingredientiLista]}} },
      { $lookup: {from: "ricettas", localField: "ricetta", foreignField: "_id", as: "ricettaInfo"}},
      { $project: {nome: {$first: "$ricettaInfo.nome"}, autore: {$first: "$ricettaInfo.autore"}, statistica: {$first: "$ricettaInfo.statistica"}, filtri: {$first: "$ricettaInfo.filtri"}, rating:{$first: "$ricettaInfo.rating"} }},
      { $match: { $expr: {$setIsSubset: [filtriLista, "$filtri"]}} },
      { $match: { $expr: { $eq: ["$nome", {$ifNull: [nomeRicetta, "$nome"]} ]}}}
    ])

    , (error, data) => {
      if(error){
        console.log(error)
        return res.status(500).json({error: "Errore: qualcosa è andato storto nella ricerca"})
      }else{
        console.log(data)
        return res.status(200).json(data)
      }
    }
  )
};

}
;
```

Figura 47: API cercaRicette controller

8. COMPLETA RICETTA - RISORSA COMPLETA RICETTA

L'API **completaRicette** è utilizzata per completare una ricetta, ovvero togliere la quantità degli ingredienti usati per realizzarla dalla dispensa dell'utente. Eventualmente la ricetta in questione viene aggiunta tra la lista dei piatti completati per la prima volta dall'utente.

Prende dal body della richiesta i seguenti parametri: il nome e l'autore della ricetta, l'username dell'utente che ha fatto l'accesso.

L'API cerca l'id della ricetta corrispondente all'autore e al nome inserito. Se non viene trovato un risultato viene restituito un errore `404 NOT FOUND`, altrimenti viene confrontato l'id con la lista dei piatti completati per la prima volta dall'account. Se non è presente tra questi viene aggiunto. In ogni caso vengono confrontati gli ingredienti nella dispensa con quelli della ricetta. Se un ingrediente del piatto non è presente nella dispensa o la quantità di un ingrediente non è sufficiente per realizzare la ricetta viene ritornato un errore, altrimenti la dispensa viene aggiornata con le nuove quantità degli ingredienti dopo il loro utilizzo. Nel caso in cui un ingrediente venga esaurito, ovvero la sua quantità raggiunge lo 0, viene tolto dalla dispensa.

```
router.patch('/completaRicetta', accountController.completaRicetta);
```

Figura 48: API completaRicetta router

```
const completaRicetta = (req, res) => {

  let nomeRicetta = req.body.ricetta;
  let autoreRicetta = req.body.autore;
  let nomeAccount = req.body.account;

  RicettaEstesa.aggregate([
    { $lookup: {from: "ricettas", localField: "ricetta", foreignField: "_id", as: "ricettaInfo"}},
    { $project: {nome: "$ricettaInfo.nome", autore: "$ricettaInfo.autore", ingredienti: "$ingredienti",
      quantita: "$quantita",
      "_id": { $first: "$ricettaInfo._id" } },
    { $match: {nome: nomeRicetta,autore: autoreRicetta} }
  ], (error, dataRicetta) => {
    if(error){
      console.log(error)
      return res.status(500).json({error: "Errore: qualcosa è andato storto nella fase di completamento della ricetta"})
    }else{
      if(dataRicetta.length == 0)
        return res.status(404).json({error: "Errore: ricetta da completare non presente nel database"})
      else{
        dataRicetta = dataRicetta[0]
        let id_ricetta = mongoose.Types.ObjectId(dataRicetta._id)

        Account.aggregate([
          { $match: {username: nomeAccount}},
          { $lookup: {from: "dispensas", localField: "_id", foreignField: "account", as: "dispensaInfo"}},
          { $project: {ingredientiDispensa: {$first:"$dispensaInfo.ingredienti"}, 
            quantitaDispensa: {$first:"$dispensaInfo.quantita"}, id_account: "$_id",
            completata: {$in: [id_ricetta, "$primiComplematamenti"] } }}
        ],
        (error, dataDisp) => {
          if(error){
            console.log(error)
            return res.status(500).json({error: "Errore: errore nella fase di completamento della ricetta"})
          }else{
            dataDisp = dataDisp[0]

            let ingredientiSufficienti = true
            let nuoveQuantita = dataDisp.quantitaDispensa
            let nuoviIngredienti = dataDisp.ingredientiDispensa
            console.log(dataRicetta.ingredienti)

            for(let k = 0; ingredientiSufficienti && k < dataRicetta.ingredienti.length; k++){
              let continua = true
              let j = 0
              for(j = 0; continua && j < dataDisp.ingredientiDispensa.length; j++){
                if(""+dataDisp.ingredientiDispensa[j] == ""+dataRicetta.ingredienti[k])
                  | continua = false
              }
              j--
            }
          }
        })
      }
    }
  })
}
```

Figura 49: API completaRicetta controller parte 1

```
        if(continua){
            ingredientiSufficienti = false
        }else{
            if(dataDisp.quantitaDispensa[j] < dataRicetta.quantita[k])
                ingredientiSufficienti = false;
            else{
                nuoveQuantita[j] = dataDisp.quantitaDispensa[j] - dataRicetta.quantita[k]
                if(nuoveQuantita[j] == 0){
                    nuoveQuantita.splice(j, 1)
                    nuoviIngredienti.splice(j, 1)
                }
            }
        }
        if(ingredientiSufficienti){
            Dispensa.updateOne( { "account": dataDisp.id_account},
            { $set: {"ingredienti":  nuoviIngredienti,"quantita": nuoveQuantita} }
            , (error, data) => {
                if(error){
                    console.log(error)
                    return res.status(500).json({error: "Errore: errore nell'update della dispensa"})
                }else{
                    if(dataDisp.completata)
                        return res.json({message: "OK: ricetta completata di nuovo"})
                    else{
                        Account.updateOne( { "username": nomeAccount},
                        { $push: {"primiCompletamenti": id_ricetta }}

                        , (error) => {
                            if(error){
                                console.log(error)
                                return res.status(500).json({error: "Errore completamento della ricetta"})
                            }else{
                                return res.status(200).json({messaggio: "Ok"})
                            }
                        });
                    }
                }
            });
        }else
            return res.json({error: "Errore: non hai gli ingredienti per completare la ricetta"});
    }
}
};

};
```

Figura 50: API completaRicetta controller parte 2



9. AGGIUNGI AI PREFERITI - RISORSA AGGIUNGI AI PREFERITI

L'API `aggiungiAiPreferiti` è utilizzata per aggiungere un certo piatto tra i preferiti dell'utente. Prende dal body della richiesta i seguenti parametri: il nome e l'autore della ricetta, l'username dell'utente che ha fatto l'accesso.

L'API cerca l'id della ricetta corrispondente all'autore e al nome inseriti. Se non viene trovato un risultato viene restituito un errore `404 NOT FOUND`, altrimenti viene poi confrontato l'id con la lista dei piatti preferiti dell'account. Nel caso in cui non sia presente tra questi viene aggiunto, altrimenti viene restituito un errore.

```
router.patch('/aggiungiAiPreferiti', accountController.aggiungiAiPreferiti);
```

Figura 51: API aggiungiAiPreferiti router

```
const aggiungiAiPreferiti = (req, res) => {

  let nomeRicetta = req.body.ricetta;
  let autoreRicetta = req.body.autore;
  let nomeAccount = req.body.account;

  Ricetta.findOne({nome: nomeRicetta, autore: autoreRicetta}, {"_id": true})
  , (error, data) => {

    if(error){
      console.log(error)
      return res.status(500).json({error: "Errore: qualcosa è andato storto nell'aggiugere la ricetta"})
    }else{
      if(data == null){
        return res.status(404).json({error: "Errore: ricetta da aggiungere non presente nel database"})
      }else{
        let id_ricetta = mongoose.Types.ObjectId(data._id)

        Account.findOne( { username: nomeAccount, preferiti: id_ricetta } )
        , (error, data) => {
          if(error){
            console.log(error)
            return res.status(500).json({error: "Errore: qualcosa è andato storto nell'aggiugere la ricetta"})
          }else{
            if(data == null){

              Account.updateOne( { "username": nomeAccount},
              { $push: {"preferiti": id_ricetta } }
              , (error) => {
                if(error){
                  console.log(error)
                  return res.status(404).json({error: "Errore: account indicato non trovato"})
                }else{
                  return res.status(200).json({message: "Ok: ricetta aggiunta ai preferiti"})
                }
              })
            }else
              return res.status(400).json({error: "Errore: ricetta già presente nei preferiti"})
          }
        }
      }
    }
  );
};
```

Figura 52: API aggiungiAiPreferiti controller

10. TOGLI DAI PREFERITI - RISORSA TOGLI DAI PREFERITI

L'API `togliDaiPreferiti` è utilizzata per togliere un certo piatto dai preferiti dell'utente. Prende dal body della richiesta i seguenti parametri: il nome e l'autore della ricetta, l'username dell'utente che ha fatto l'accesso.

L'API cerca l'id della ricetta corrispondente all'autore e al nome inseriti. Se non viene trovato un risultato viene restituito un errore `404 NOT FOUND`, altrimenti viene confrontato l'id con la lista dei piatti preferiti dell'account. Nel caso in cui sia presente tra questi viene tolta.

```
router.patch('/togliDaiPreferiti', accountController.togliDaiPreferiti);
```

Figura 53: API `togliDaiPreferiti` router

```
const togliDaiPreferiti = (req, res) => {

  let nomeRicetta = req.body.ricetta;
  let autoreRicetta = req.body.autore;
  let nomeAccount = req.body.account;

  Ricetta.findOne({nome: nomeRicetta, autore: autoreRicetta}, {"_id": true})
    , (error, data) => {

      if(error){
        console.log(error)
        return res.status(500).json({error: "Errore: qualcosa è andato storto nell'eliminazione della ricetta"})
      }else{
        console.log(data)
        if(data == null){
          return res.status(404).json({error: "Errore: ricetta da togliere non presente nel database"})
        }else{
          let id = mongoose.Types.ObjectId(data._id)
          Account.updateOne( { "username": nomeAccount}, { $pull: {"preferiti": id} })
            , (error, dataAcc) => {
              console.log(dataAcc)
              if(error){
                console.log(error)
                return res.status(500).json({error: "Errore: qualcosa è andato storto nell'eliminazione della ricetta"})
              }else{
                return res.status(200).json({message: "Ok: ricetta tolta dai preferiti"})
              }
            );
        }
      }
    }
};

};
```

Figura 54: API `togliDaiPreferiti` controller



11. AGGIUNGI RATING - RISORSA AGGIUNGI RATING

L'API **aggiungiRating** è utilizzata per aggiungere una valutazione ad una ricetta. Prende dal body della richiesta i seguenti parametri: il nome e l'autore della ricetta, l'username dell'utente che ha fatto l'accesso, il rating dato. Se il rating non è fornito o non è compreso tra 0 e 5 viene restituito un errore, altrimenti l'API cerca l'id della ricetta corrispondente all'autore e al nome inseriti. Nel caso non venga trovato viene restituito un errore 404 NOT FOUND, altrimenti viene confrontato l'id con la lista dei rating già dati dall'account. Se non è presente tra questi viene aggiunto il riferimento alla ricetta e la nuova valutazione alla lista, altrimenti viene restituito un errore.

```
router.patch('/aggiungiRating', accountController.aggiungiRating);
```

Figura 55: API aggiungiRating router

```
const aggiungiRating = (req, res) => {
  let nomeRicetta = req.body.ricetta;
  let autoreRicetta = req.body.autore;
  let nomeAccount = req.body.account;
  let rating = req.body.rating;

  if(rating < 0 || rating > 5 || rating == undefined)
    return res.status(400).json({error: "Errore: rating non valido"});
  else{
    Ricetta.findOne({nome: nomeRicetta, autore: autoreRicetta})
      , (error, data) => {
        if(error){
          return res.status(500).json({error: "Errore: qualcosa è andato storto nell'aggiungere il nuovo rating"});
        }else{
          if(data == null){
            return res.status(404).json({error: "Errore: la ricetta inserita non esiste nel database"});
          }else{
            let id_ricetta = mongoose.Types.ObjectId(data._id)
            console.log(data)
            Account.findOne({username: nomeAccount, "ratingDati.ricetta": id_ricetta })
              , (error, data) => {
                if(error){
                  return res.status(500).json({error: "Errore: qualcosa è andato storto nell'aggiungere il nuovo rating"});
                }else{
                  if(data == null){
                    Account.updateOne( { "username": nomeAccount}, { $push: { "ratingDati": {ricetta: id_ricetta, ratingDato: rating} } })
                      , (error, dataAcc) => {
                        console.log(dataAcc)
                        if(error){
                          console.log(error)
                          return res.status(500).json({error: "Errore: qualcosa è andato storto nell'aggiunta del nuovo rating"});
                        }else{
                          return res.status(200).json({message: "Ok: nuovo rating aggiunto"});
                        }
                      };
                  }else
                    return res.status(400).json({error: "Errore: rating già registrato per questa ricetta"});
                }
              };
            });
          });
        };
      };
    );
};

};
```

Figura 56: API aggiungiRating controller



3. API DOCUMENTATION

In questa sezione viene presentata la documentazione delle API descritte e analizzate nel paragrafo precedente. La documentazione è stata realizzata grazie all modulo NodeJS **Swagger UI Express**. È stato usato quest'ultimo perché rende possibile osservare la documentazione a chiunque possieda il codice sorgente. Inoltre Swagger UI crea una pagina Web dalle definizioni contenute nel file `swagger.js`.

Queste sono le API analizzate con i metodi HTTP associati:

- login - POST
- getMail - GET
- patchPassword - PATCH
- aggiungiIngrediente - PATCH
- eliminaIngrediente - DELETE
- cercaIngrediente - GET
- cercaRicette - GET
- completaRicetta - PATCH
- aggiungiAiPreferiti - PATCH
- togliDaiPreferiti - PATCH
- aggiungiRating - PATCH

Inoltre sono stati definiti i seguenti modelli:

- Ricette
- Account
- Dispense
- Ingredienti

Successivamente verranno mostrate delle immagini delle API e dei modelli appena presentati.

L'endpoint da invocare per raggiungere la documentazione è:

<http://localhost:8080/api-docs>

 Swagger
Supported by SMARTBEAR

JustCook 1.0.0

[Base URL: localhost:8080 /]

Implementazione web app JustCook

MIT

Schemes

Giulio Bazzoli, Alessandra Dalla Verde, Denise Comincioli API per JustCook

Ricette

GET /cercaRicette/:cerca Cerca tra le ricette nel database a seconda dei parametri in input che possono essere: gli ingredienti nella dispensa, i vari filtri e/o il nome della ricetta

Account

PATCH /aggiungiAiPreferiti Dato il nome e l'autore della ricetta e l'account dell'utente aggiunge il piatto passato ai preferiti

PATCH /togliDaiPreferiti Dato il nome e l'autore della ricetta e l'account dell'utente toglie il piatto passato dai preferiti

PATCH /completaRicetta Dato il nome e l'autore della ricetta e l'account dell'utente aggiunge il piatto tra i primi completati

PATCH /aggiungiRating Dato il nome e l'autore della ricetta, l'account dell'utente e il rating da aggiungere, il rating nuovo viene aggiunto

GET /account/:indirizzoEmail Preleva l'indirizzo email per il cambio password del login e verifica se è associato ad un account

PATCH /account/:indirizzoEmail/:password Preleva indirizzo email e nuova password per il cambio password del login

Dispensa

PATCH /dispensa/:aggiungiIngrediente Aggiunge alla dispensa l'ingrediente e la quantità specificati dall'utente

DELETE /dispensa/eliminaIngrediente Elimina l'ingrediente specificato dall'utente dalla dispensa

Authentication

POST /login Fornisce l'accesso agli account degli utenti

Ingredienti

GET /ingrediente/:nome Cerca l'ingrediente inserito nella barra di ricerca della dispensa e verifica che esista nel database

Figura 57: Pagina Web documentazione API

Models

- Ricette >
- Account >
- Dispense >
- Ingredienti >

Figura 58: Pagina Web documentazione modelli

4. FRONTEND IMPLEMENTATION

In questo paragrafo viene presentato il Front-end dell'applicazione JustCook. È stato sviluppato per desktop. Sono state implementate la pagina della home (con la dispensa, il login, il cambio password e la ricerca delle ricette), la pagina della ricetta (con la possibilità di aggiungerla/toglierla dai preferiti, di dare un rating e di completare il piatto) e la pagina per il cambio password. Verranno descritte le due pagine con delle immagini.

4.1. PAGINA HOME

1. PAGINA HOME SENZA LOGIN

La pagina della home senza aver effettuato il login si presenta con a sinistra la dispensa e al centro la sezione della ricerca delle ricette. In alto a destra c'è il login e il link per la pagina del cambio password se si sono dimenticate le credenziali. Nella [figura 59](#) l'utente ha già inserito l'username e la password per effettuare l'accesso.



Figura 59: Pagina home senza login

2. PAGINA HOME CON LOGIN - DISPENSA

Nella figura 60 è presente la pagina della home dopo aver effettuato il login. In alto a destra compare la scritta di benvenuto con a fianco lo username di chi ha effettuato l'accesso. Inoltre nella sezione della dispensa l'utente può vedere gli ingredienti che erano presenti alla fine della sessione precedente. Può aggiungere nuovi ingredienti o rimuoverli.



Figura 60: Pagina home con login - dispensa

3. PAGINA HOME CON LOGIN - RICERCA RICETTE

Nella figura 61 l'utente ha sempre effettuato il login e la sua dispensa contiene degli ingredienti. Questa volta è stata effettuata una ricerca delle ricette. Questa è possibile farla specificando il nome del piatto nell'apposita barra di ricerca, selezionando dei filtri e/o inserendo i parametri di ordinamento. Dopo aver premuto su "cerca", il sistema manda come risposta all'utente i piatti possibili da realizzare con gli ingredienti disponibili in dispensa (in questo caso solo "Tiramisù"). Le ricette risultanti si presentano con il loro titolo, la valutazione, lo username di chi ha ideato il piatto, il tempo, il costo e la difficoltà.

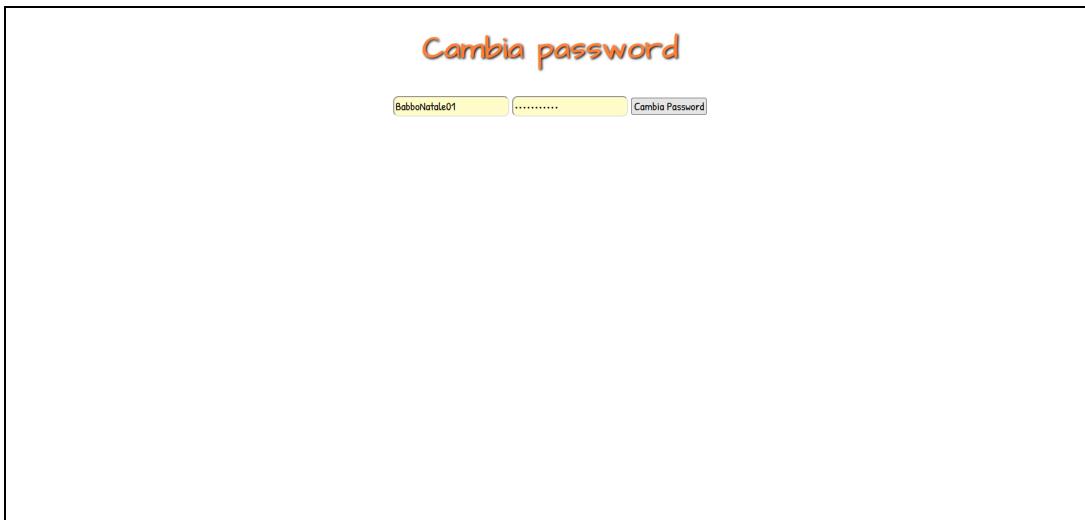


The screenshot shows the JustCook homepage. On the left, a green sidebar titled "La tua dispensa" (Your pantry) contains a search bar ("Inserisci un ingrediente") and a button ("Aggiungi"). Below it is a section titled "I tuoi ingredienti" (Your ingredients) with a list of items: "Savoiardi 1" and "Mascarpone 100". There is also a "Rimuovi un ingrediente dalla dispensa" (Remove an ingredient from your pantry) section with a search bar and a "Rimuovi" button. On the right, a large orange area titled "JustCook" displays a search interface with a search bar ("Cerca una ricetta per nome"), filters for "Dieta: vegana, vegetariana", "Senza: latte, uova, glutine, frutta secca", "Tipo: antipasto, primo, secondo, dessert", sorting options ("Ordina per: tempo, costo, difficoltà, rating"), and a specific result for "Tiramisu di Maria" with a 4-star rating, low time and cost, and low difficulty.

Figura 61: Pagina home con login - ricerca ricette

4.2.PAGINA CAMBIO PASSWORD

Cliccando il link per il cambio password ([figura 59](#)) si arriva alla pagina corrispondente. Ci sono le barre dove l'utente può inserire il proprio indirizzo email e la nuova password. Se la procedura presenta degli errori verrà mostrato all'utente un messaggio di errore, altrimenti la password viene cambiata.



The screenshot shows a simple form titled "Cambia password". It has three input fields: a yellow one containing "BabboNatale01", a red one containing ".....", and a blue one containing "Cambia Password".

Figura 62: Pagina cambio password

4.3.PAGINA RICETTA

Nella figura 63 è presente la pagina della ricetta. Vi sono tutte le informazione del piatto, ovvero: il nome, chi l'ha ideato, la valutazione (in questo caso quella che ha dato l'utente, tre stelle, diversa da quella presente in figura 61), il tempo, il costo, la difficoltà, i filtri, la descrizione, gli ingredienti e il procedimento. Inoltre di fianco al nome della ricetta ci sono i pulsanti per completare la ricetta e/o aggiungerla ai preferiti. Nel caso si cliccasse quest'ultimo la scritta diverrebbe “togli dai preferiti”.

Tiramisù di Maria completa aggiungi ai preferiti

★★★
tempo: 100 min
costo: basso
difficoltà: bassa
Filtri: senza uova dessert

Descrizione
Il tiramisù è uno dei capisaldi della cucina italiana, uno dei dolci al cucchiaio più amati e realizzati in tutto il mondo

Ingredienti

- Mascarpone: 750g
- Savoiardi: 250g

Procedimento

1. Per preparare il tiramisù preparate il caffè con la moka per ottenerne 300 g, poi zuccherate a piacere (noi abbiamo messo un cucchiaino) e lasciatelo raffreddare in una ciotolina bassa e ampia. Separate le uova dividendo gli albumi dai tuorli
2. Montate i tuorli con le fruste elettriche, versando solo metà dose di zucchero
3. Non appena il composto sarà diventato chiaro e spumoso, e con le fruste ancora in funzione, potrete aggiungere il mascarpone, poco alla volta
4. Incorporato tutto il formaggio avrete ottenuto una crema densa e compatta

Figura 63: Pagina ricetta



5. GITHUB REPOSITORY AND DEPLOYMENT INFO

In questo paragrafo viene presentata la github repository (la struttura del progetto viene analizzata e descritta più nel dettaglio rispetto alla [sezione 2.1](#)). Dopo aver mostrato il link della repository, verranno descritte le varie cartelle che la compongono. Successivamente verranno analizzate le informazioni sul deployment e il link da utilizzare per eseguire l'applicazione su Heroku.

5.1. GITHUB REPOSITORY

Il link per raggiungere la github repository ([figura 64](#)) è il seguente:

https://github.com/JustCook1/Documentazione_D4/tree/main/JustCook

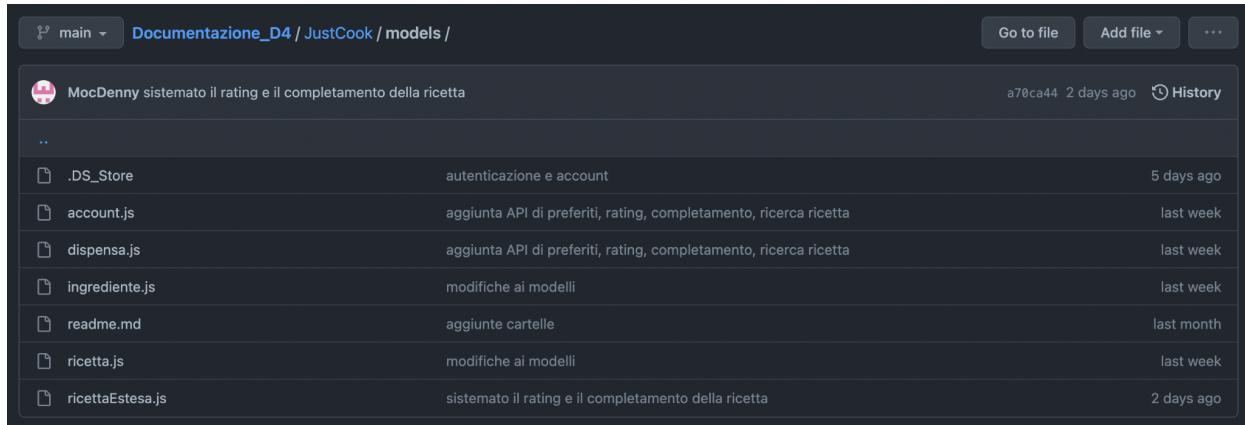
The screenshot shows the GitHub repository page for 'JustCook'. At the top, it displays the branch 'main' (1 branch, 0 tags), a 'Go to file' button, an 'Add file' dropdown, and a 'Code' button. Below this is a list of 96 commits from a user named 'MocDenny'. Each commit includes the file name, a brief description, and the time of the commit. The commits are as follows:

File	Description	Time
.github/workflows	update	7 minutes ago
Istruzioni_documento_D4	rearrangement dei documenti per il deployment	2 hours ago
controllers	rearrangement dei documenti per il deployment	2 hours ago
models	rearrangement dei documenti per il deployment	2 hours ago
routes	rearrangement dei documenti per il deployment	2 hours ago
static	update	20 minutes ago
test	rearrangement dei documenti per il deployment	2 hours ago
.gitignore	rimosso DS_Store	12 hours ago
Procfile	update struttura	2 hours ago
README.md	update sul funzionamento di MongoDB durante i test su Postman	3 weeks ago
app.js	rearrangement dei documenti per il deployment	2 hours ago
package-lock.json	rearrangement dei documenti per il deployment	2 hours ago
package.json	rearrangement dei documenti per il deployment	2 hours ago
server.js	update file	42 minutes ago
swagger.json	rearrangement dei documenti per il deployment	2 hours ago

Figura 64: Github repository JustCook

1. MODELS

La cartella **models** contiene i modelli che rappresentano le strutture dati (analizzate nel paragrafo 2.3) presenti nel database. Sono stati realizzati i modelli dell'account, della ricetta base, della ricetta completa, dell'ingrediente e della dispensa.



The screenshot shows a GitHub repository interface for the 'models' folder of the 'JustCook' project. The folder contains several files: '.DS_Store', 'account.js', 'dispensa.js', 'ingrediente.js', 'readme.md', 'ricetta.js', and 'ricettaEstesa.js'. Each file has a commit history:

File	Description	Date
.DS_Store	autenticazione e account	5 days ago
account.js	aggiunta API di preferiti, rating, completamento, ricerca ricetta	last week
dispensa.js	aggiunta API di preferiti, rating, completamento, ricerca ricetta	last week
ingrediente.js	modifiche ai modelli	last week
readme.md	aggiunge cartelle	last month
ricetta.js	modifiche ai modelli	last week
ricettaEstesa.js	sistemato il rating e il completamento della ricetta	2 days ago

Figura 65: Github repository JustCook - models

2. ROUTES

La cartella **routes** contiene le definizioni dei percorsi delle API. Sono stati realizzati i file per l'account, la dispensa, l'ingrediente, la ricetta, la ricetta estesa e l'autenticazione.



The screenshot shows a GitHub repository interface for the 'routes' folder of the 'JustCook' project. The folder contains several files: 'account.js', 'authentication.js', 'dispensa.js', 'ingrediente.js', 'readme.md', 'ricetta.js', and 'ricettaEstesa.js'. Each file has a commit history:

File	Description	Date
account.js	sistemato il rating	3 days ago
authentication.js	eseguito merge	2 days ago
dispensa.js	eseguito merge	2 days ago
ingrediente.js	eseguito merge	2 days ago
readme.md	aggiunge cartelle	last month
ricetta.js	aggiunta API di preferiti, rating, completamento, ricerca ricetta	last week
ricettaEstesa.js	aggiunta pagina ricetta	3 days ago

Figura 66: Github repository JustCook - routes

3. CONTROLLERS

La cartella **controllers** è essenzialmente la business logic di JustCook. Contiene il codice delle API. Sono stati realizzati i file per l'account, l'autenticazione, la ricetta, la ricetta completa, l'ingrediente e la dispensa.

Documentazione_D4 / JustCook / controllers /		
 alessandradallaverde	Delete tokenChecker.js	...
		4424ce5 13 minutes ago
..		History
 .DS_Store	autenticazione e account	5 days ago
 account.js	sistemato il rating e il completamento della ricetta	3 days ago
 authentication.js	update grafico 2 pagien	14 hours ago
 dispensa.js	eseguito merge	2 days ago
 ingrediente.js	eseguito merge	2 days ago
 readme.md	aggiunta cartelle	last month
 ricetta.js	aggiunta API di preferiti, rating, completamento, ricerca ricetta	last week
 ricettaEstesa.js	feature per ordinare i risultati aggiunta	2 days ago

Figura 67: Github repository JustCook - controllers

4. STATIC

La cartella **static** è quella che racchiude tutto il codice del Front-end. Presenta anche una cartella **img** per la memorizzazione di immagini di supporto.

Documentazione_D4 / static /		
 McDenny	update	7c8ec56 22 minutes ago
		History
..		
 img	update	1 hour ago
 cambiaPassword.html	update	1 hour ago
 index.html	update file	44 minutes ago
 ricetta.html	update	1 hour ago
 ricetta_style.css	update	1 hour ago
 script.js	update	22 minutes ago
 script_cambiapass.js	update	1 hour ago
 script_ricetta.js	update	1 hour ago
 style.css	update	1 hour ago
 style_cambiapsw.css	update	1 hour ago

Figura 68: Github repository JustCook - static

5. TEST

La cartella **test** contiene il testing per i file (codice API) della cartella **controllers**. Infatti c'è il file relativo all'account, alla dispensa, all'ingrediente, alle ricette e all'autenticazione.

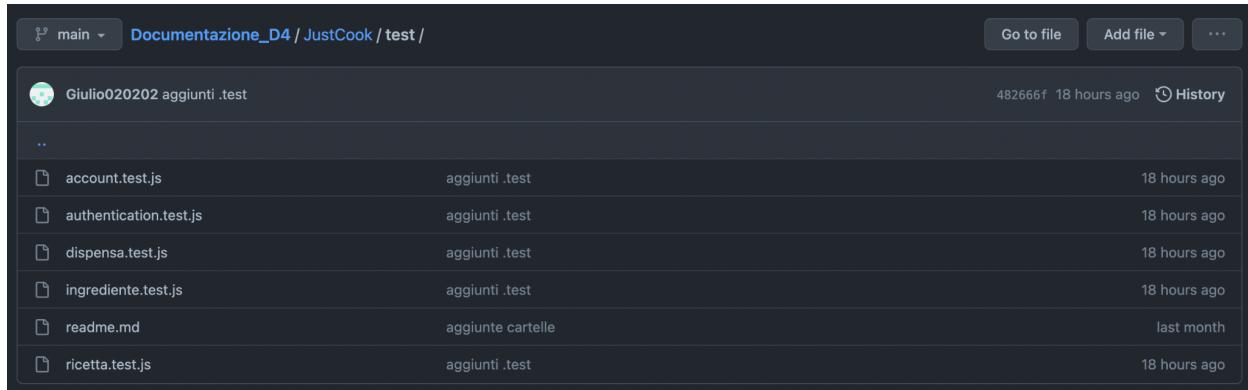


Figura 69: Github repository JustCook - test

5.2. DEPLOYMENT INFO

In questa sezione viene analizzata la fase di deployment. Durante quest'ultima si è utilizzata la piattaforma di **Heroku**. Dopo aver seguito tutte le procedure per creare l'app web “justcook” sulla piattaforma e collegarla al progetto github, è stata creata una nuova action per fare il build dell'applicazione e il deployment (sempre attraverso Heroku).

Il link seguente è quello per lanciare l'applicazione:

<https://justcook.herokuapp.com/>



6. TESTING

In questo paragrafo viene presentato il testing delle API. Per l'applicazione web JustCook viene testato il funzionamento delle diverse API fondamentali al funzionamento. In particolare si procede al testing delle API usate dall'implementazione descritta precedentemente, quindi omettendo diverse funzionalità non presenti in questa versione del sito. I test vengono eseguiti con **JEST** e **supertest**. Questi sono strumenti utili al testing di applicazioni JavaScript, in particolare delle API. Di seguito si descrivono i diversi casi di test, divisi in base ai modelli fondamentali utilizzati nel progetto. Tutti i test vengono eseguiti collegandosi al database MongoDB in localhost, quindi su oggetti inseriti, manualmente o meno, dagli sviluppatori. Infine viene mostrato il file “coverage” con delle osservazioni.

6.1.CASI DI TEST

1. AUTHENTICATION

Poiché `login` è l'unico metodo di autenticazione il test unico è quello in [figura 70](#). Il testing consiste nel verificare se un tentativo di login vada a buon fine.

```
describe('Test per le api di authentication', () => {  
  //aspetto che la connessione al db sia stabilita  
  beforeAll( async () => { jest.setTimeout(8000);  
    app.locals.db = await mongoose.connect(process.env.MONGODB_URI); });  
  afterAll( () => { mongoose.connection.close(true); });  
  
  //test per login  
  test('Test per login', async () => {  
    const res = await request(app).post('/login').send({username : "prova", password : "prova"});  
    expect(res.statusCode).toEqual(200);  
    expect(res.body).toHaveProperty('username');  
  });  
});
```

Figura 70: Test authentication

2. ACCOUNT

I test per il modello account sono quelli mostrati in [figura 71](#). Per ogni funzione ci si limita a testare che il codice di risposta da parte del server sia quello atteso.

```
describe('Test per le api di account', () => [
  //aspetto che la connessione al db sia stabilita
  beforeAll( async () => { jest.setTimeout(8000);
    | app.locals.db = await mongoose.connect(process.env.MONGODB_URI); });
  afterAll( () => { mongoose.connection.close(true); });

  //test per aggiungiAiPreferiti
  test('Test per aggiungiAiPreferiti', async () => {
    const res = await request(app).patch('/aggiungiAiPreferiti').send({ricetta : "formaggio con le pere", autore : "juj", account : "pippo"});
    expect(res.statusCode).toEqual(200);
  });

  //test per togliDaiPreferiti
  test('Test per togliDaiPreferiti', async () => {
    const res = await request(app).patch('/togliDaiPreferiti').send({ricetta : "formaggio con le pere", autore : "juj", account : "pippo"});
    expect(res.statusCode).toEqual(200);
  });

  //test per aggiungiRating (mi aspetto un errore perchè ho già dato un rating)
  test('Test per aggiungiRating', async () => {
    const res = await request(app).patch('/aggiungiRating').send({ricetta : "formaggio con le pere", autore : "juj", account : "pippo", rating : 5});
    expect(res.statusCode).toEqual(400);
  });

  //test completaRicetta (mi aspetto un errore perchè ho già completato la ricetta)
  test('Test per completaRicetta', async () => {
    const res = await request(app).patch('/completaRicetta').send({ricetta : "formaggio con le pere", autore : "juj", account : "pippo"});
    expect(res.statusCode).toEqual(404);
  });

  //test per getMail
  test('Test per getMail', async () => {
    const res = await request(app).get('/account/sos@mail.cccccccccccccc');
    expect(res.statusCode).toEqual(200);
  });

  //test per patchPassword
  test('Test per patchPassword', async () => {
    const res = await request(app).patch('/account/sos@mail.cccccccccccc/NuovaPassword123!');
    expect(res.statusCode).toEqual(200);
  });
]);
```

Figura 71: Test account

3. DISPENSA

Come per i modelli precedenti, per i metodi del modello dispensa si verifica se l'API funziona correttamente controllando anche la validità dei parametri restituiti.

```
//test per le api di dispensa
describe('Test per le api di dispensa', () => {

    //aspetto che la connessione al db sia stabilita
    beforeAll( async () => { jest.setTimeout(8000);
        app.locals.db = await mongoose.connect(process.env.MONGODB_URI); });
    afterAll( () => { mongoose.connection.close(true); });

    //test per la get di una dispensa
    test('Test per la get di una dispensa', async () => {
        const res = await request(app).get('/dispensa/pippo');
        expect(res.statusCode).toEqual(200);
        expect(res.body).toHaveProperty('ingredienti');
        expect(res.body).toHaveProperty('quantita');
    });

    //test per nuova dispensa
    test('Test per la creazione di una nuova dispensa', async () => {
        const res = await request(app).post('/dispensa').send({nome: 'pippo'});
        expect(res.statusCode).toEqual(200);
    });

    //test per aggiungere un ingrediente alla dispensa
    test('Test per aggiungere un ingrediente alla dispensa', async () => [
        const res = await request(app).patch('/dispensa/aggiungiIngrediente').send({nome: 'pippo', idIngrediente: '63a72e3fdbf4ec92157444b8', quantita: 100});
        expect(res.statusCode).toEqual(200);
        expect(res.body).toHaveProperty('ingredienti');
        expect(res.body).toHaveProperty('quantita');
    ]);

    //test per rimuovere un ingrediente alla dispensa
    test('Test per rimuovere un ingrediente alla dispensa', async () => {
        const res = await request(app).delete('/dispensa/eliminaIngrediente').send({nome: 'pippo', idIngrediente: '63a72e3fdbf4ec92157444b8'});
        expect(res.statusCode).toEqual(200);
        expect(res.body).toHaveProperty('ingredienti');
        expect(res.body).toHaveProperty('quantita');
    });

});
```

Figura 72: Test dispensa

4. INGREDIENTE

I metodi testati per il modello `ingrediente` sono fondamentalmente analoghi, si tratta infatti di metodi GET, ma sfruttando parametri diversi. Questo porta anche i test a essere praticamente identici.

```
describe('Test per le api di ingrediente', () => {

    //aspetto che la connessione al db sia stabilita
    beforeAll( async () => { jest.setTimeout(8000);
        app.locals.db = ...mongoose.connect(process.env.MONGODB_URI); });
    afterAll( () => { mongoose.connection.close(true); });

    //test per getIngredienti
    test('Test per getIngredienti', async () => {
        const res = await request(app).get('/ingrediente/formaggio');
        expect(res.statusCode).toEqual(200);
    });

    //test per creaIngredienteId
    test('Test per creaIngredienteId', async () => {
        const res = await request(app).get('/ingrediente/id/63a72e3fdbf4ec92157444b8');
        expect(res.statusCode).toEqual(200);
    });

});
```

Figura 73: Test ingrediente

5. RICETTA

Per il metodo `getRicetta` si esegue un testing per la correttezza, verificando il codice ritornato dalla funzione.

```
//test per le api di ricetta
describe('Test per le api di ricetta', () => {

    //aspetto che la connessione al db sia stabilita
    beforeAll(async () => {
        jest.setTimeout(8000);
        app.locals.db = ...mongoose.connect(process.env.MONGODB_URI);
    });
    afterAll(() => { mongoose.connection.close(true); });

    //test per getRicetta
    test('Test per getRicetta', async () => {
        const res = await request(app).get('/ricetta/formaggio con le pere/juj');
        expect(res.statusCode).toEqual(200);
    });

});
```

Figura 74: Test ricetta

6. RICETTA ESTESA

Come per gli altri metodi descritti fino ad ora anche in questo caso si procede a verificare il caso in cui l'API abbia successo, e restituisca quindi codice 200.

```
describe('Test per le api di ricettaEstesa', () => {  
  
    //aspetto che la connessione al db sia stabilita  
    beforeAll(async () => {  
        jest.setTimeout(8000);  
        app.locals.db = await mongoose.connect(process.env.MONGODB_URI);  
    });  
    afterAll(() => { mongoose.connection.close(true); });  
  
    //test per cercaRicette  
    test('Test per cercaRicette', async () => {  
        const res = await request(app).get('/cercaRicette/cerca?ingredienti=formaggio,pere&filtrati=primo');  
        expect(res.statusCode).toEqual(200);  
    });  
  
    //test per trovaInfoRicette  
    test('Test per trovaInfoRicette', async () => {  
        const res = await request(app).get('/trovaRicetta/ricettaEstesa?ricetta=formaggio con le pere');  
        expect(res.statusCode).toEqual(200);  
    });  
});
```

Figura 75: Test ricettaEstesa

6.2.COVERAGE

Dal file “coverage”(figura 76) generato dal pacchetto **jest**, si può notare come il testing effettivo sia spesso scarso. Il motivo è la scelta di testare solo le API necessarie al funzionamento dell'applicazione web, tralasciando quindi quelle implementate per fare testing durante lo sviluppo grazie allo strumento Postman e quelle non sfruttate nella versione finale del sito. Inoltre, come si può vedere dal codice del testing riportato sopra, nella maggior parte dei casi si procede a verificare il corretto funzionamento delle API nel caso di successo, tralasciando il testing sui diversi casi di errore poiché forzare determinati errori di funzioni così complessi richiederebbe una fase di testing decisamente più formale e estensiva.

All files JustCook/controllers

46.77% Statements 167/357 34.78% Branches 64/184 58.46% Functions 38/65 51.07% Lines 166/325

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File ▲	Statements ▲	Branches ▲	Functions ▲	Lines ▲
account.js		46.29%	75/162	32.09%
authentication.js		91.66%	11/12	75%
dispensa.js		62.19%	51/82	52%
ingrediente.js		43.75%	14/32	33.33%
ricetta.js		50%	9/18	37.5%
ricettaEstesa.js		10%	4/40	0%
tokenChecker.js		27.27%	3/11	0%

Figura 76: Coverage