Ingegneria del Software
UniTN - Dipartimento di Ingegneria e Scienza dell'Informazione
Anno accademico 2022 - 2023
Bevilacqua L. - Sartore A. - Tecchio L.

UNIVERSITY OF TRENTO

Nome del progetto:

Hungry Everywhere

Titolo del documento:

Sviluppo Applicazione

INDICE

1. SCOPO DEL DOCUMENTO

Il presente documento riporta tutte le informazioni necessarie per lo sviluppo di una parte dell'applicazione Hungry Everywhere. In particolare, presenta tutti gli artefatti necessari per realizzare i servizi di gestione dei clienti e dei gestori dei ristoranti dell'applicazione Hungry Everywhere. Partendo dalla descrizione degli user flow legati al cliente e al gestore, il documento prosegue con la presentazione delle API necessarie (tramite l'API Model e il Modello delle risorse) per poter aggiungere prodotti al carrello del cliente, aggiungere, modificare e rimuovere ristoranti, gestire gli ordini e recuperare le informazioni riguardanti la posizione, il tutto all'interno di Hungry Everywhere. Per ogni API realizzata, oltre ad una descrizione delle funzionalità fornite, il documento presenta la sua documentazione e i test effettuati. Infine una sezione è dedicata alle informazioni del Git Repository e il deployment dell'applicazione stessa.

2. USER FLOWS

Cliente

In questa sezione del documento di sviluppo riportiamo gli “user flows” per il ruolo del Cliente nella nostra applicazione. La figura 1 descrive il flow di operazioni che un utente può fare. Una volta registrato e fatto il login viene considerato come Cliente. Dalla pagina principale il Cliente inserisce il proprio indirizzo (o utilizza le API di Google) e visualizza i ristoranti vicini a sé. Da qui in poi l’utente, se ha effettuato l’accesso come Cliente, può terminare la sessione e fare il logout. Dalla visualizzazione dei ristoranti è possibile selezionare il singolo ristorante da cui sarà possibile visualizzare il menù, modificare e aggiungere pietanze al carrello. Per questo passaggio viene eseguito un controllo sulla sessione per verificare se l’utente è autenticato o meno, nel caso non lo sia viene riportato nella pagina di login. Una volta finito l’ordine, è possibile visionare il carrello per rimuovere o modificare le pietanze inserite. Il Cliente può tornare alla pagina dei ristoranti senza pagare l’ordine, ma ciò cancella il carrello corrente. Infine il Cliente può passare alla consegna e al pagamento dell’ordine. In ogni modo nella figura 1 sono illustrate tutte le varie interazioni tra il Cliente e l’applicazione, ma anche le varie relazioni tra le azioni. Nella figura 1 è anche presente una legenda che descrive i simboli usati nello user flow.

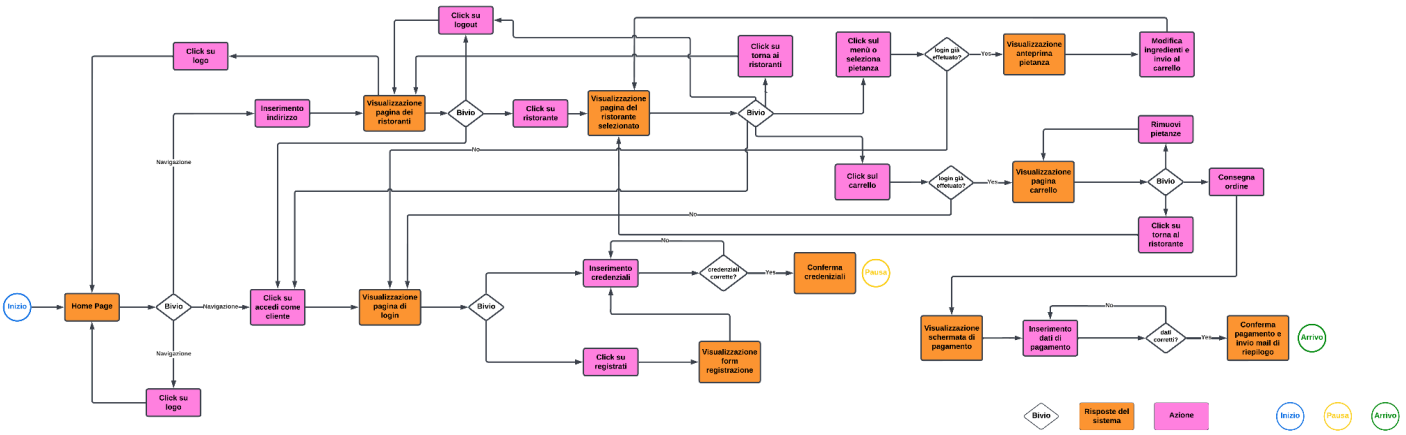


Figura 1. User Flow Cliente

GestoreRistorante

In questa sezione del documento di sviluppo riportiamo gli “user flows” per il ruolo del GestoreRistorante nella nostra applicazione. La figura 2 descrive lo user flow relativo alle operazioni che un gestore può svolgere all’interno dell’applicazione Hungry Everywhere. Il gestore, per essere considerato tale e distinguersi dal cliente, deve necessariamente effettuare il login come ristoratore nella home page del sito. Nel caso possieda già le credenziali le dovrà inserire negli appositi campi, altrimenti dovrà effettuare una registrazione mediante un form. Per poter gestire i propri ristoranti, il gestore deve sottoscrivere un abbonamento che gli permette di aggiungere, modificare tramite apposito form o eliminare i propri ristoranti. Per ogni ordine il gestore potrà scegliere se accettarlo o rifiutarlo, in tal caso però invierà anche il rimborso, ed invierà una mail per notificare al cliente l’esito dell’operazione. Nella figura 2 sono illustrate le varie interazioni tra il gestore e l’applicazione, ma anche le varie relazioni tra le azioni. Nella figura 1 è anche presente una legenda che descrive i simboli usati nello user flow.

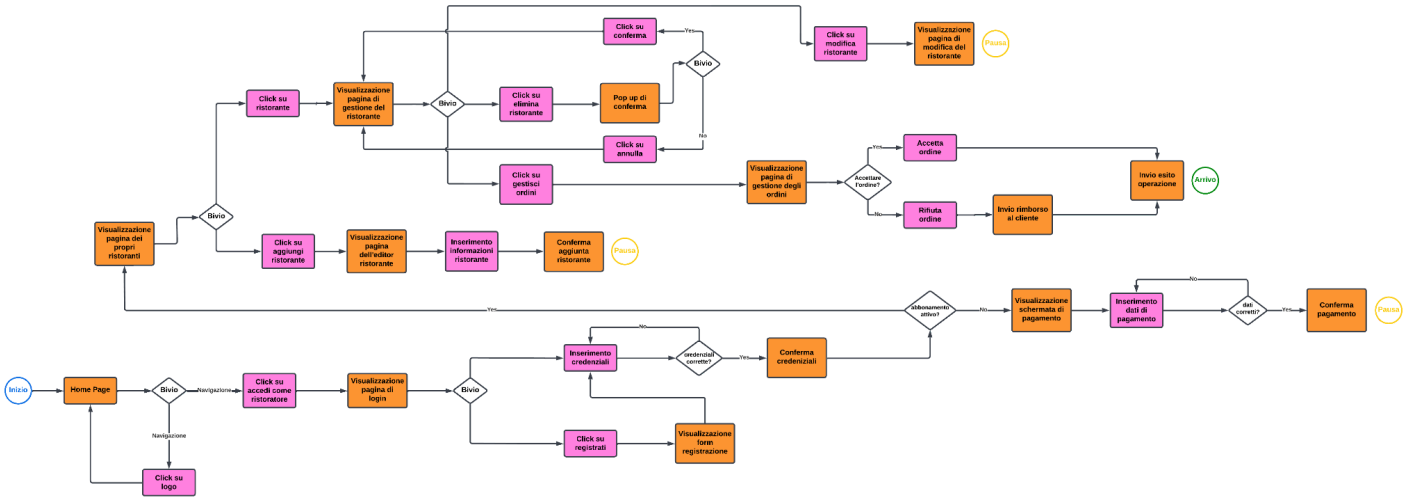


Figura 2. User Flow GestoreRistorante

3. APPLICATION IMPLEMENTATION AND DOCUMENTATION

Nelle sezioni precedenti abbiamo identificato le varie features che devono essere implementate per la nostra applicazione con un'idea di come il nostro utente finale (RA) può utilizzarle nel suo flusso applicativo. L'applicazione è stata sviluppata utilizzando NodeJS e VueJS. Per la gestione dei dati abbiamo utilizzato MongoDB.

3.1 PROJECT STRUCTURE

La struttura del progetto è presentata in Figura 3 ed è composto di una cartella api per la gestione delle api locali, di una cartella ui per la parte del front-end, e di una cartella Photos per la memorizzazione di immagini a supporto.

3.2 PROJECT DEPENDENCIES

I seguenti moduli Node sono stati utilizzati e aggiunti al file Package.Json

- Express:
- Express-file upload
- MongoDB
- Cors
- Body-parser

3.3 PROJECT DATA OR DB

Per la gestione dei dati utili all'applicazione abbiamo definito due principali strutture dati come illustrato in Figura 4. Una collezione di "dipartimenti" e una collezioni di "dipendenti".

Per rappresentare i dipartimenti e i dipendenti abbiamo definito i seguenti tipi di dati (vedi figure 5 e 6).

3.4 PROJECT APIs

3.4.1 RESOURCES EXTRACTION FROM THE CLASS DIAGRAM

Abbiamo individuato le seguenti risorse: *SistemaPagamento*, *Cliente*, *GestoreRistornate*, *Ordine*, *Ristorante*, *Carrello*. Per ogni risorsa sono state individuate delle funzionalità.

SistemaPagamento

Questa risorsa contiene i seguenti attributi: idMittente e idDestinatario. La risorsa implementa la seguente API:

- validaPagamento: Utilizza il metodo POST passando come parametri idMittente, idDestinatario e importo. La funzione è quella di validare un pagamento.

Cliente

Questa risorsa contiene i seguenti attributi: password, email, nome e cognome. La risorsa implementa le seguenti API:

- login: Utilizza il metodo POST passando come parametri email e password. La funzione è quella di effettuare il login dell'utente.
- logout: Utilizza il metodo GET per terminare la sessione dell'utente e fargli eseguire il logout.
- register: Utilizza il metodo POST passando come parametri email e password. La funzione è quella di registrare l'utente nel database.

GestoreRistorante

Questa risorsa contiene i seguenti attributi: password, email, nome, cognome e ristoranti. La risorsa implementa le seguenti API:

- login: Utilizza il metodo POST passando come parametri email e password. La funzione è quella di effettuare il login dell'utente.
- logout: Utilizza il metodo GET per terminare la sessione dell'utente e fargli eseguire il logout.

- register: Utilizza il metodo POST passando come parametri email e password. La funzione è quella di registrare l'utente nel database.
- gestioneOrdine: Utilizza il metodo POST passando come parametri ordine e accettazione. La funzione è quella di confermare o rifiutare un ordine.
- getOrdine: Utilizza il metodo GET per recuperare un'ordinazione specifica.
- aggiungiRistorante: Utilizza il metodo POST passando come parametri le informazioni del ristorante da aggiungere.
- modificaRistorante: Utilizza il metodo PUT passando come parametri le informazioni del ristorante che devono venir modificate.
- rimuoviRistorante: Utilizza il metodo DELETE passando come parametri le informazioni del ristorante che si vuole rimuovere.

Ordine

Questa risorsa contiene i seguenti attributi: ordine e isValid. La risorsa implementa la seguente API:

- inviaOrdine: Utilizza il metodo POST passando come parametro utente. La funzione è quella di inviare l'ordinazione di quell'utente passato come parametro.

Ristorante

Questa risorsa contiene i seguenti attributi: nome, indirizzo e menù. La risorsa implementa le seguenti API:

- getInfo: Utilizza il metodo GET passando come parametro ristorante. La funzione è quella di ritornare tutte le informazioni del singolo ristorante passato come parametro.
- getMenu: Utilizza il metodo GET passando come parametro ristorante. La funzione è quella di ritornare il menu del singolo ristorante passato come parametro.
- getThumbnail: Utilizza il metodo GET passando come parametro ristorante. La funzione è quella di ritornare la copertina del singolo ristorante passato come parametro.

Carrello

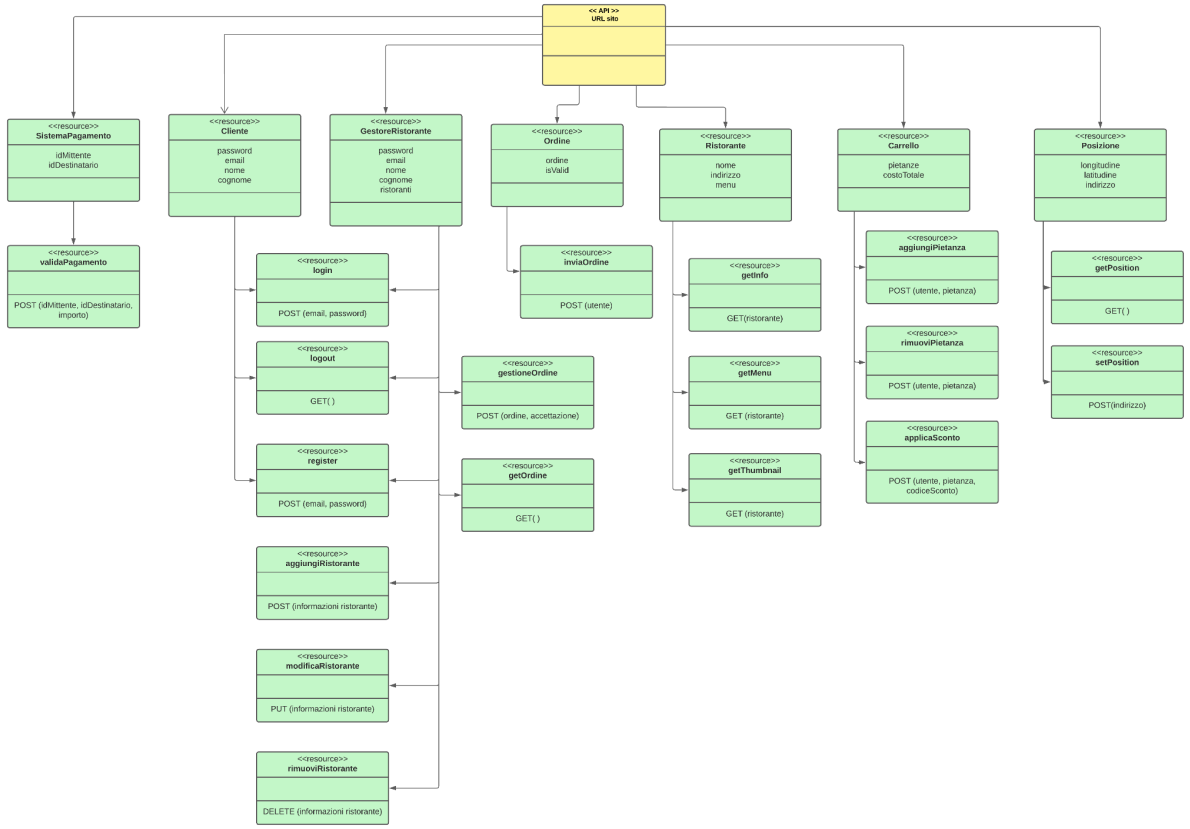
Questa risorsa contiene i seguenti attributi: pietanza e costoTotale. La risorsa implementa le seguenti API:

- aggiungiPietanza: Utilizza il metodo POST passando come parametri utente e pietanza. La funzione è quella di aggiungere la pietanza passata come parametro al carrello dell'utente passato come parametro.
- rimuoviPietanza: Utilizza il metodo POST passando come parametri utente e pietanza. La funzione è quella di rimuovere la pietanza passata come parametro al carrello dell'utente passato come parametro.
- applicaSconto: Utilizza il metodo POST passando come parametri utente, pietanza e codiceSconto. La funzione è quella di controllare come l'utente è autenticato e controllare che per quella pietanza ci sia uno sconto applicabile. Il codiceSconto può essere utilizzato come alternativa per applicare lo sconto.

Posizione

Questa risorsa contiene i seguenti attributi: longitudine, latitudine e indirizzo. La risorsa implementa le seguenti API:

- getPosition: Utilizza il metodo GET per ritornare la posizione dell'utente nel caso questi abbia permesso l'accesso alla propria posizione.
- setPosition: Utilizza il metodo POST passando come parametro indirizzo. La funzione è quella di impostare l'indirizzo come posizione dell'utente.

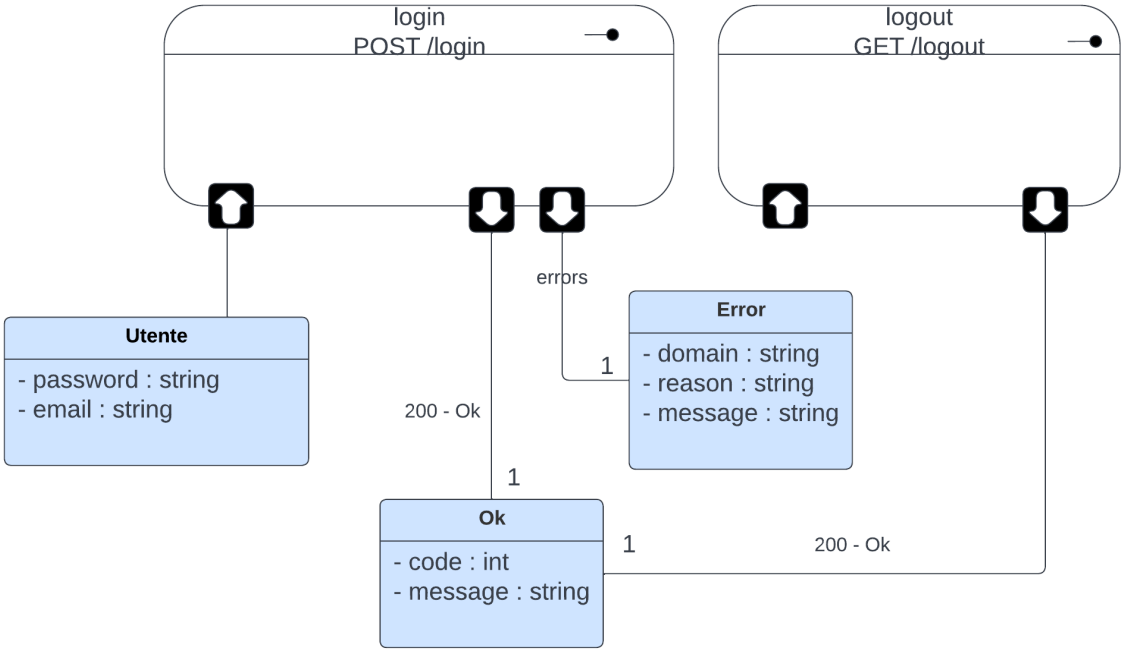

<< API >>
URL sito
<<resource>>
SistemaPagamento
idMittente
idDestinatario
<<resource>>
validaPagamento
POST (idMittente, idDestinatario, importo)
<<resource>>
Cliente
password
email
nome
cognome
<<resource>>
login
POST (email, password)
<<resource>>
logout
GET( )
<<resource>>
register
POST (email, password)
<<resource>>
aggiungiRistorante
POST (informazioni ristorante)
<<resource>>
modificaRistorante
PUT (informazioni ristorante)
<<resource>>
rimuoviRistorante
DELETE (informazioni ristorante)
<<resource>>
GestoreRistorante
password
email
nome
cognome
ristoranti
<<resource>>
gestioneOrdine
POST (ordine, accettazione)
<<resource>>
getOrdine
GET( )
<<resource>>
Ordine
ordine
isValid
<<resource>>
inviaOrdine
POST (utente)
<<resource>>
Ristorante
nome
indirizzo
menu
<<resource>>
getInfo
GET(ristorante)
<<resource>>
getMenu
GET (ristorante)
<<resource>>
getThumbnail
GET (ristorante)
<<resource>>
Carrello
pietanze
costoTotale
<<resource>>
aggiungiPietanza
POST (utente, pietanza)
<<resource>>
rimuoviPietanza
POST (utente, pietanza)
<<resource>>
applicaSconto
POST (utente, pietanza, codiceSconto)
<<resource>>
Posizione
longitudine
latitudine
indirizzo
<<resource>>
getPosition
GET( )
<<resource>>
setPosition
POST(indirizzo)

3.4.2 Resources Model

GestoreRistorante

La risorsa GestoreRistorante implementa le seguenti API, descritte attraverso il Resource Model sotto indicato:

- **login**: Utilizza il metodo POST prendendo in input un Utente. Restituisce un messaggio di OK attraverso il codice 200 nel caso tutto sia andato a buon fine. Nel caso dovessero accadere degli errori viene restituito un messaggio con la motivazione dell'errore.
- **logout**: Utilizza il metodo GET e restituisce un messaggio di OK attraverso il codice 200 informando l'utente che ha eseguito il logout.



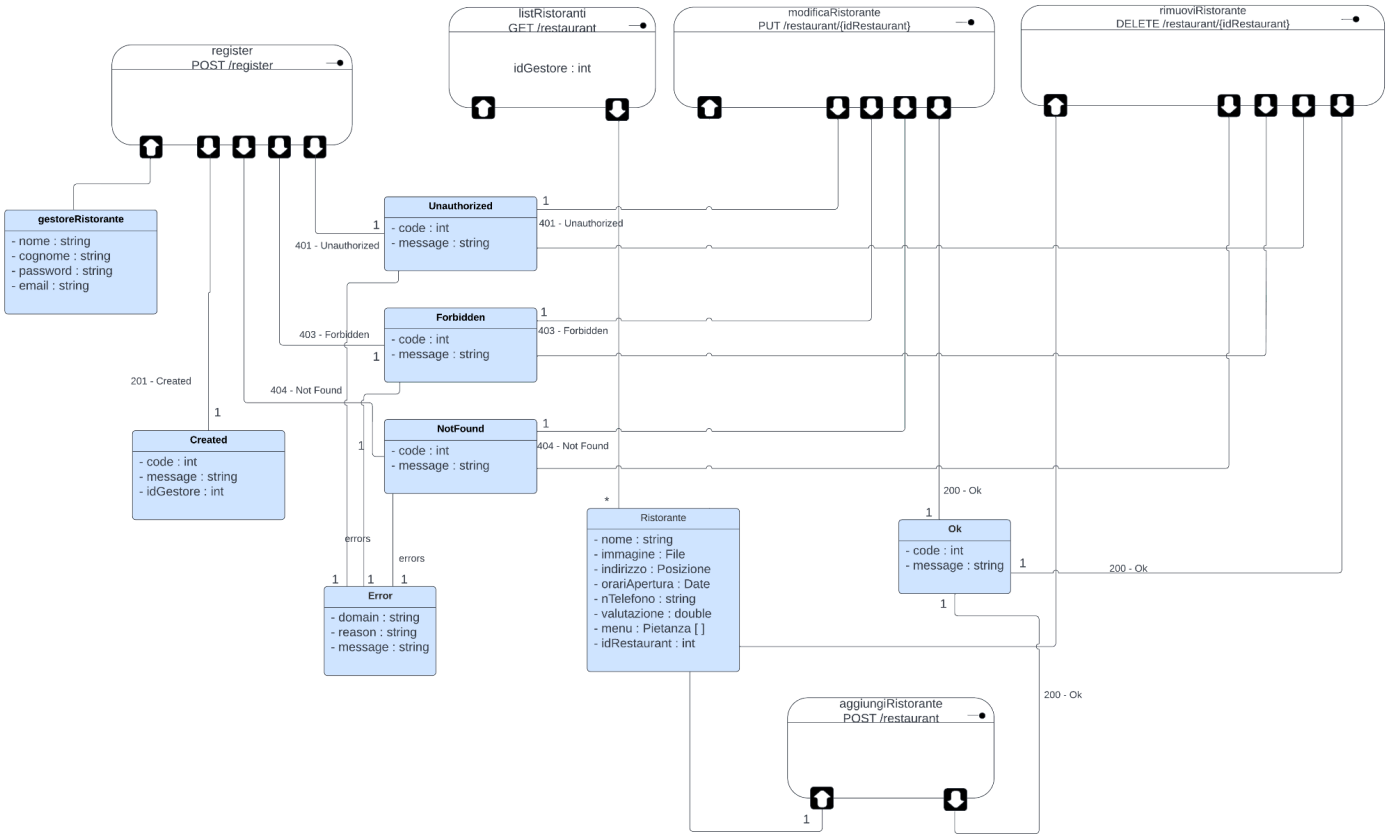
- **register**: Utilizza il metodo POST con in input gestoreRistorante. Se tutto va a buon fine viene creato il gestore e viene notificato attraverso il codice 201 - Created con relativo messaggio. Può succedere che ci siano degli errori:
  - 401 - Unauthorized, nel caso in cui l'utente non abbia il permesso di registrarsi;
  - 403 - Forbidden, nel caso in cui all'utente non sia permesso di registrarsi;

    - 404 - Not Found, nel caso in cui la pagina di registrazione non sia raggiungibile.

  Tutti gli errori vengono notificati all'utente attraverso opportuni messaggi di errore.
- **listRistoranti**: Utilizza il metodo GET con parametro idGestore e ritorna tutti i ristoranti di quel gestore.
- **modificaRistorante**: Utilizza il metodo PUT con input il Ristorante da modificare. Se tutto va a buon fine viene restituito un messaggio di avvenuta modifica attraverso il codice 200 - OK. Può succedere che ci siano degli errori:
    - 401 - Unauthorized, nel caso in cui l'utente non possa modificare quel ristorante;
    - 403 - Forbidden, nel caso in cui all'utente non sia permessa la modifica del ristorante;
    - 404 - Not Found, nel caso in cui la pagina di modifica non sia raggiungibile.

  Tutti gli errori vengono notificati all'utente attraverso opportuni messaggi di errore.
- **aggiungiRistorante**: Utilizza il metodo POST con input il ristorante da aggiungere. Se tutto va a buon fine viene restituito un messaggio "aggiunto correttamente" attraverso il codice 200 - OK.
- **rimuoviRistorante**: Utilizza il metodo DELETE con input il ristorante da rimuovere. Se tutto va a buon fine viene restituito un messaggio di avvenuta rimozione attraverso il codice 200 - OK. Può succedere che ci siano degli errori:
    - 401 - Unauthorized, nel caso in cui l'utente non possa rimuovere quel ristorante;
    - 403 - Forbidden, nel caso in cui all'utente non sia permessa la rimozione del ristorante;
    - 404 - Not Found, nel caso in cui la pagina di rimozione non sia raggiungibile.

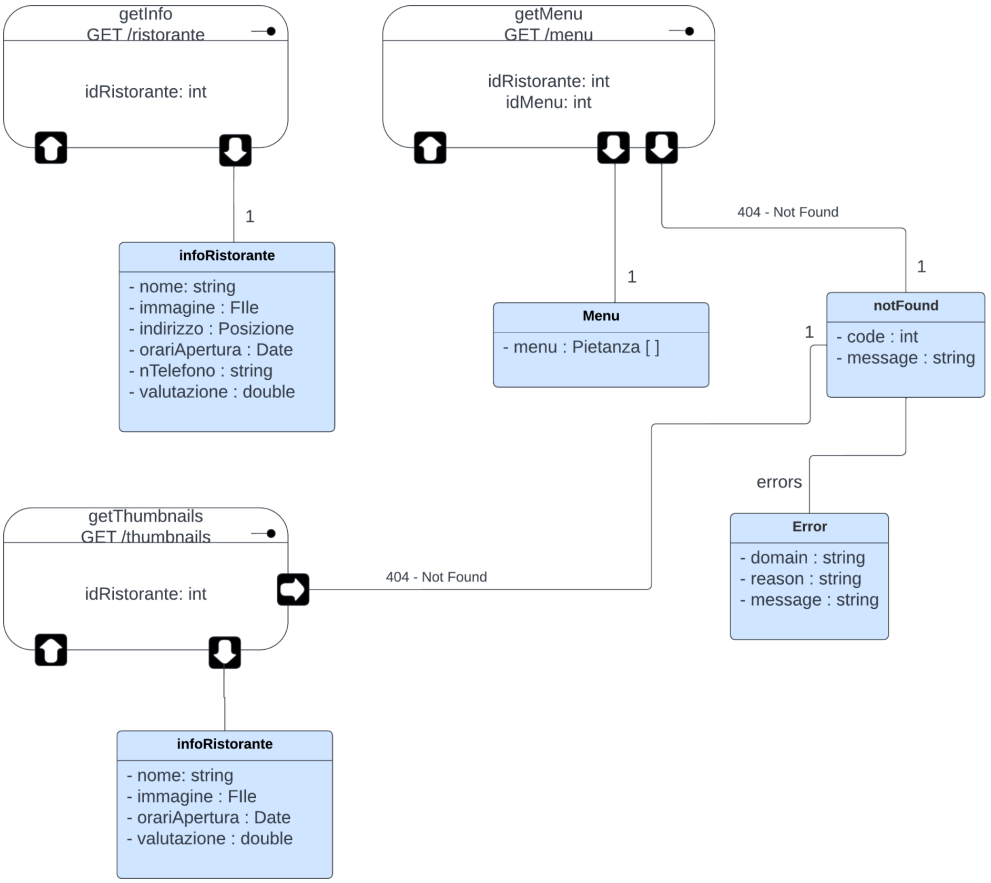
  Tutti gli errori vengono notificati all'utente attraverso opportuni messaggi di errore.



Ristorante

La risorsa Ristorante implementa le seguenti API, descritte attraverso il Resource Model sotto indicato:

- **getInfo**: Utilizza il metodo GET con parametro idRistorante e ritorna infoRistorante che contiene tutte le informazioni del ristorante con quell'id.
- **getMenu**: Utilizza il metodo GET con parametri idRistorante e idMenu e ritorna il menù corrispondente di quel ristorante. Nel caso non venga trovato viene notificato all'utente un messaggio di errore con codice 404 - Not Found.
- **getThumbnails**: Utilizza il metodo GET con parametro idRistorante e ritorna infoRistorante che contiene le informazioni del ristorante che sono presenti nella pagina di tutti i ristoranti vicini. Nel caso non venga trovato viene notificato all'utente un messaggio di errore con codice 404 - Not Found.



Carrello

La risorsa Carrello implementa le seguenti API, descritte attraverso il Resource Model sotto indicato:

- **aggiungiPietanza**: Utilizza il metodo POST con input la pietanza da aggiungere. Se tutto va a buon fine viene aggiunta la pietanza al carrello e viene notificato attraverso il codice 201 - Created con relativo messaggio. Può succedere che ci siano degli errori:
    - 401 - Unauthorized, nel caso in cui l'utente non abbia effettuato l'accesso e provi ad aggiungere la pietanza;
    - 403 - Forbidden, nel caso in cui all'utente non sia permesso di aggiungere la pietanza;

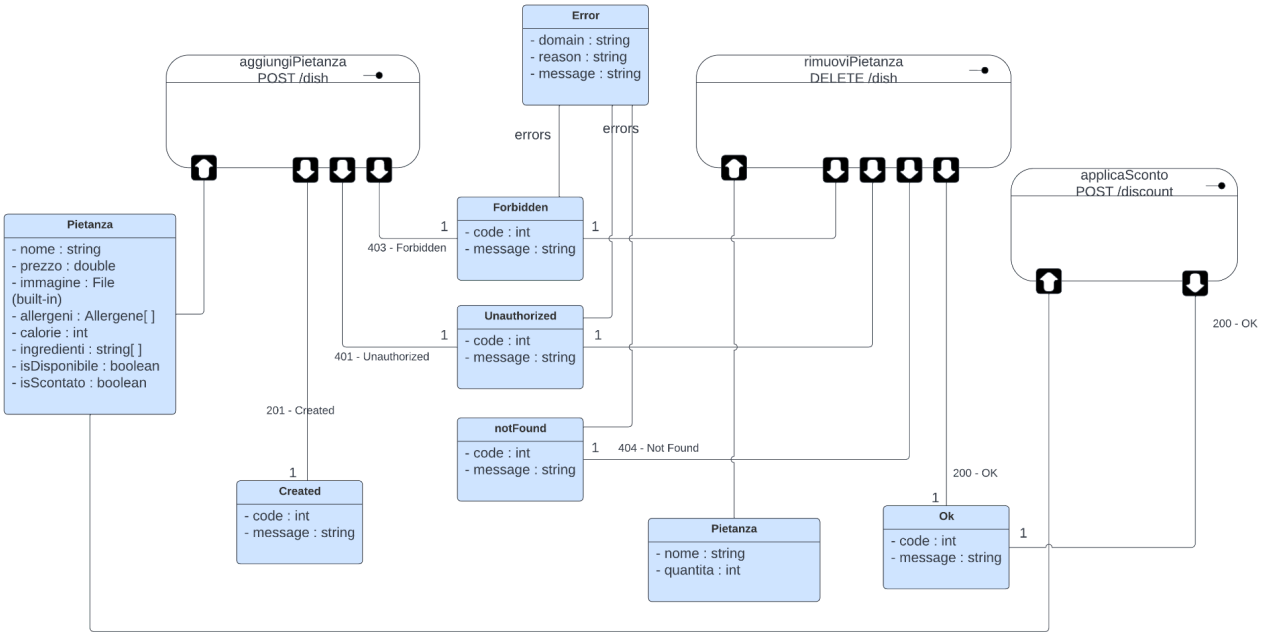
  Tutti gli errori vengono notificati all'utente attraverso opportuni messaggi di errore.
- **rimuoviPietanza**: Utilizza il metodo DELETE con input la pietanza da rimuovere e la quantità. Se tutto va a buon fine

viene notificato all'utente la corretta rimozione della pietanza attraverso il codice 200 - OK. Può succedere che ci siano degli errori:

- 401 - Unauthorized, nel caso in cui l'utente non possa rimuovere quella pietanza;
- 403 - Forbidden, nel caso in cui all'utente non sia permessa la rimozione della pietanza;
- 404 - Not Found, nel caso in cui si provi a rimuovere una pietanza inesistente.

Tutti gli errori vengono notificati all'utente attraverso opportuni messaggi di errore.

- **applicaSconto**: Utilizza il metodo POST con input la Pietanza da scontare. Se tutto va a buon fine viene notificato all'utente che lo sconto è stato applicato attraverso il codice 200 - OK.

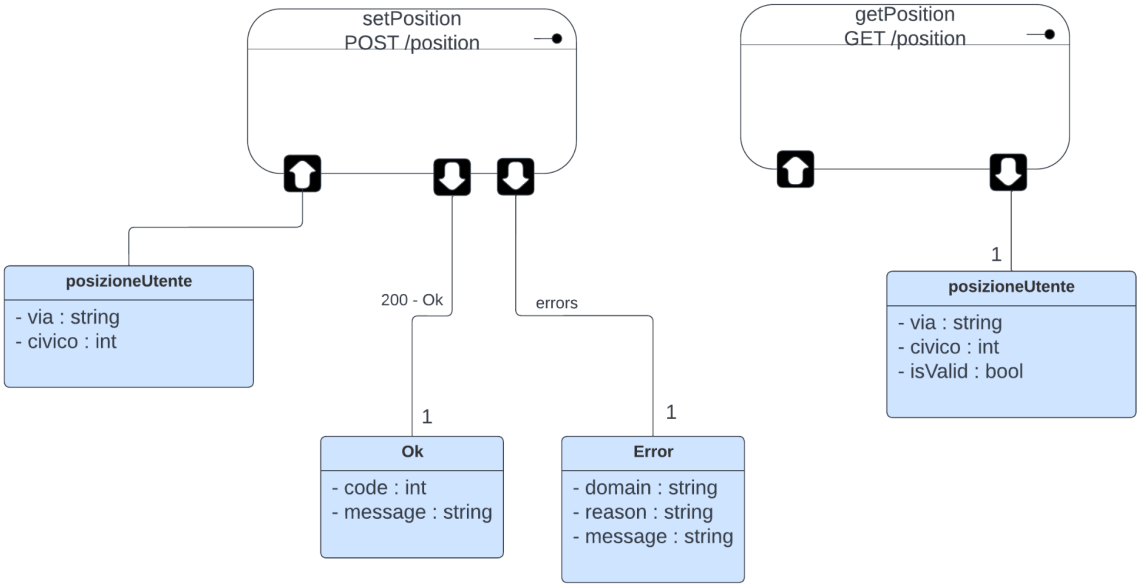


Posizione

La risorsa Posizione implementa le seguenti API, descritte attraverso il Resource Model sotto indicato:

- **setPosition**: Utilizza il metodo POST con input posizioneUtente. Se la via dell'utente esiste viene notificato all'utente la conferma della posizione con annesso codice 200 - Ok. Viene notificato un messaggio di errore nel caso la via non esista.

- **getPosition**: Utilizza il metodo GET e restituisce posizioneUtente.



3.5 Sviluppo API

4. API documentation

5. FrontEnd Implementation

6. GitHub Repository and Deployment Info

<< Descrizione di come e' strutturato il Repo Git >>
<< Info sul deployment e sul link da utilizzare per eseguire l'applicazione su Heroku >>

7. Testing

<< Descrizione dei casi di test e del loro esito su alcune funzionalità dell'applicazione>>