Relazione Tecnica sul Progetto di Gestione Ristoranti

Giacomo Di Giuseppe

6 luglio 2024

Indice

1 Introduzione

Questa relazione tecnica fornisce una panoramica di ReservEZ, un'applicazione web progettata per semplificare la prenotazione e l'ordinazione presso una vasta gamma di ristoranti. In questa sezione di introduzione, verranno presentati gli obiettivi del progetto, le funzionalità richieste e le scelte implementative adottate.

1.1 Obiettivi del Progetto

Il principale obiettivo del progetto di Gestione Ristoranti è fornire agli utenti un'esperienza intuitiva e efficiente per prenotare tavoli e effettuare ordini presso una varietà di ristoranti. Gli obiettivi specifici includono:

- Sviluppare un'interfaccia utente per la ricerca, la prenotazione e l'ordine in diversi ristoranti.
- Implementare un sistema di gestione delle prenotazioni e degli ordini per i ristoratori.
- Incorporare un sistema di raccomandazione per suggerire ai utenti ristoranti in base ai loro gusti e la loro posizione.
- Dare la possibilità ai ristoratori di avere una propria pagina sulla quale caricare tutte le informazioni, delle immagini e il menù del locale
- Avere un sistema di notifiche real-time per prenotazioni, ordini per il ristoratore e dell'aggiornamento dello stato dell'ordine per l'utente

1.2 Funzionalità Richieste

Le funzionalità principali richieste per il sistema di ReserveZ includono:

- Registrazione e accesso per gli utenti e i ristoratori.
- Ricerca e visualizzazione dettagliata dei ristoranti, compresi menu, orari di apertura e foto.
- Prenotazione di tavoli e ordinazione di cibo e bevande (da ritirare).
- Gestione delle pagine dei ristoranti da parte dei proprietari, inclusa la modifica delle informazioni e la gestione delle prenotazioni.
- Sistema di raccomandazione per suggerire ristoranti agli utenti in base alle loro preferenze.
- Sistema di notifiche real-time per prenotazioni, ordini per il ristoratore e dell'aggiornamento dello stato dell'ordine per l'utente
- Avere una dashboard per la gestione delle informazioni dell'utente, come indirizzo o nome.

1.3 Struttura del Documento

La relazione tecnica seguirà una struttura organizzata, divisa in varie sezioni per esaminare dettagliatamente ciascun aspetto del progetto di Gestione Ristoranti. Le sezioni successive copriranno i seguenti argomenti:

1. Organizzazione logica dell'applicazione
2. Dettagli delle funzionalità implementate
3. Test effettuati e risultati ottenuti
4. Analisi dei requisiti non funzionali

2 Descrizione del Progetto

2.1 Use Case UML

Il diagramma dei casi d'uso (Use Case UML) rappresenta le funzionalità disponibili per ciascun tipo di utente: utenti anonimi, utenti registrati, e ristoratori.

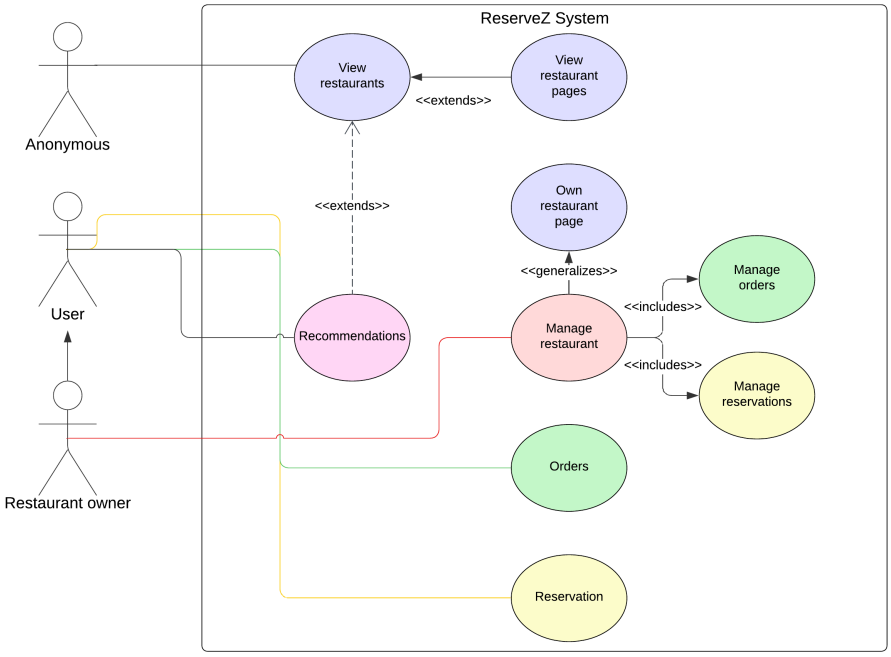

Figura 1: Use Case Diagram

2.2 Diagramma delle Classi

Il diagramma delle classi descrive la struttura statica del sistema, mostrando le classi, i loro attributi, metodi e le relazioni tra di esse.

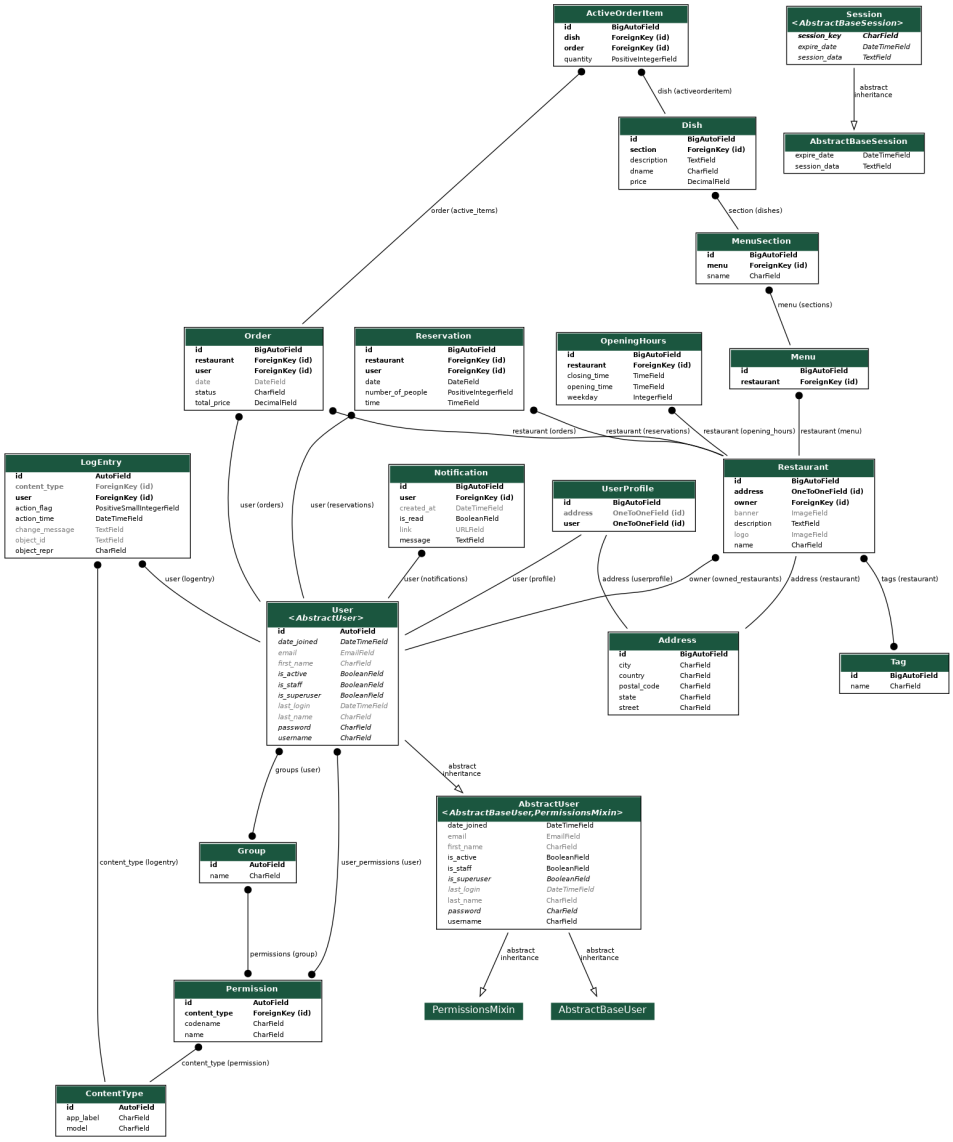

Figura 2: Diagramma delle Classi

2.3 Activity Diagram

L'activity diagram illustra i principali algoritmi e i passaggi logici del sistema. Questo diagramma aiuta a capire il flusso delle operazioni come la gestione degli ordini e delle prenotazioni.

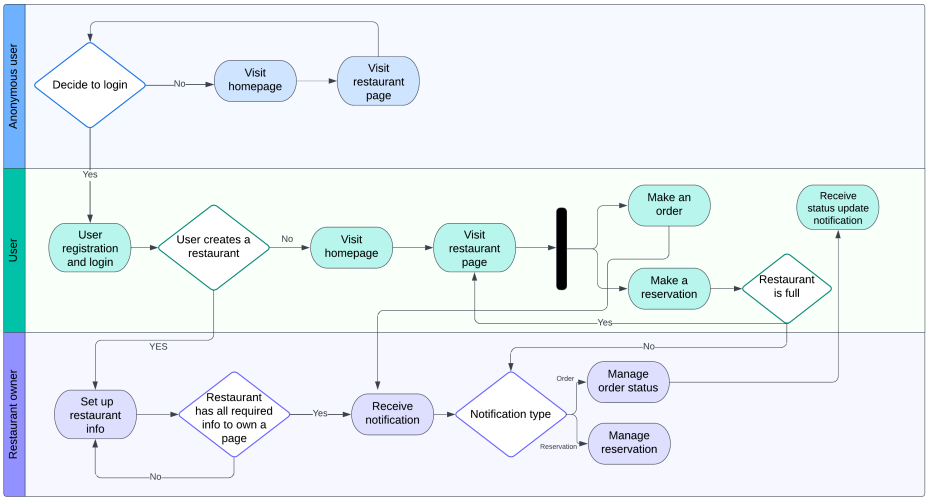

Figura 3: Activity Diagram

3 Scelte Implementative

3.1 Tecnologie usate

Per implementare le funzionalità richieste, ho adottato un'architettura web basata su **Django**, un framework **Python** per lo sviluppo di applicazioni web. Django offre un'ampia gamma di funzionalità, inclusi modelli, views e URL routes, che mi hanno permesso di creare velocemente un sistema robusto e scalabile.

Inoltre, abbiamo ho utilizzato per la parte di **front-end** i **templates HTML** di django, **CSS** e **JavaScript** per creare una semplice interfaccia utente, utilizzando i principi del design **responsive** per garantire che il sito sia accessibile da diverse risoluzioni schermo e dispositivi. Per velocizzare quest'ultimo processo mi sono aiutato con il framework **Bootstrap**.

Per quanto riguarda il **DB**, ho scelto **PostgreSQL**, un sistema di gestione di database relazionali, sfruttando il concetto di **ORM** di Django che permette di manipolare il DB utilizzando oggetti Python invece di dover scrivere SQL query direttamente.

Infine, ho implementato un sistema di autenticazione e autorizzazione utilizzando l'**authentication framework** di Django. Questo ha permesso di proteggere le aree riservate del sito e assicurare che solo gli utenti registrati possano accedere alle funzionalità avanzate, come la prenotazione dei tavoli e la gestione delle pagine dei ristoranti.

Inoltre, per gestire le notifiche in tempo reale, ho integrato l'uso dei **WebSocket channels** per Django, che consentono l'invio di notifiche istantanee tra il ristoratore e l'utente quando vengono effettuate prenotazioni o ordini, e quando lo stato dell'ordine viene aggiornato (ad esempio, da "awaiting acceptance" a "accepted" o "ready").

Infine, per la gestione dei **form**, ho adottato **Crispy Forms**, un'applicazione Django che semplifica la creazione e la gestione dei form HTML, offrendo un controllo più granulare sull'aspetto e sul comportamento dei form nel sito web.

4 Organizzazione logica dell'applicazione

Nel processo di sviluppo di ReservEZ, l'organizzazione logica del codice è stata pianificata per garantire una struttura chiara e modulare.

4.1 Struttura della directory

La directory del progetto ReservEZ è organizzata in modo da riflettere le diverse funzionalità e i moduli principali dell'applicazione. Di seguito è riportata una panoramica della struttura della directory:

- **accounts**: Contiene tutti i moduli relativi alla gestione degli account utente, inclusi modelli, viste, form e template per l'autenticazione, la registrazione e la gestione dei dati dell'utente.
- **restaurants**: Racchiude i moduli dedicati alla gestione dei ristoranti, compresi quelli per le viste dell'utente (homepage e pagina del ristorante) e la dashboard del ristoratore.
- **reservez**: Include le impostazioni principali del progetto e le route dell'URL globali.
- **static**: Contiene i file statici come stylesheet CSS.
- **templates**: Contiene i template HTML di base utilizzati dall'applicazione.
- **media**: Contiene i file di default per le card dei ristoranti nella homepage, ma anche le immagini caricate da ogni utente, contentute in cartelle chiamate "user_nome-utente".

4.2 Moduli principali

All'interno di ciascuna cartella principale, sono presenti diversi moduli che svolgono ruoli specifici nell'applicazione. Ad esempio:

- In **accounts**, il modulo `models.py` definisce i modelli degli utenti e dei profili, mentre `views.py` gestisce le viste relative all'autenticazione e alla gestione del profilo utente.
- In **restaurants**, i moduli `views.py` e `forms.py` gestiscono rispettivamente le viste e i form associati alla gestione dei ristoranti e delle prenotazioni.
- Il modulo `settings.py` all'interno di **reservez** contiene le impostazioni globali dell'applicazione Django, come le configurazioni del database e le app installate.

4.3 Divisione delle responsabilità

La divisione delle responsabilità è stata un principio guida nella progettazione dell'architettura dell'applicazione. Le responsabilità sono state suddivise in modo da separare chiaramente le funzionalità dell'utente da quelle del ristoratore. Ad esempio:

- Le viste e i template all'interno di **accounts** gestiscono le operazioni relative all'autenticazione e alla gestione del profilo dell'utente.
- Le viste e i template all'interno di **restaurants** sono responsabili della visualizzazione delle informazioni sui ristoranti e della gestione delle prenotazioni da parte dei ristoratori.

4.4 Benefici dell'approccio

L'approccio adottato nell'organizzazione del codice offre diversi vantaggi:

- **Manutenibilità**: La suddivisione modulare facilita la manutenzione del codice, consentendo ai team di lavorare in modo collaborativo su aree specifiche dell'applicazione.
- **Scalabilità**: La separazione delle responsabilità rende più semplice aggiungere nuove funzionalità e adattare l'applicazione a esigenze future.
- **Chiarezza**: La struttura ben definita rende più facile per i nuovi sviluppatori comprendere il codice e contribuire al progetto.

4.5 Possibili miglioramenti

Nonostante la struttura attuale sia stata progettata con cura, ci sono sempre spazi per miglioramenti futuri. Alcune possibili aree di miglioramento includono:

- Ulteriore modularizzazione: Alcuni moduli potrebbero essere suddivisi ulteriormente per promuovere una maggiore coesione e un accoppiamento più basso.
- Ottimizzazione delle performance: Potrebbero essere adottate tecniche per migliorare le prestazioni dell'applicazione, come la cache dei dati o l'ottimizzazione delle query al database.

5 Scelte fatte

In questa sezione, verranno descritte le principali scelte e le motivazioni dietro, relative alle funzionalità implementate nel sistema di gestione dei ristoranti. Le scelte riguardano la gestione delle liste di attesa, il tipo di sistema di raccomandazione adottato e altre funzionalità rilevanti.

5.1 Registrazione di un ristorante

Ogni utente può decidere di registrare il proprio locale sull'applicazione. L'idea iniziale era di permettere ad ogni utente di registrare un numero maggiore di locali (ad esempio nel caso di una catena), però ho deciso di optare per un unico locale per ogni utente, per semplificare i controlli e la gestione della dashboard.

Agli gli utenti che decidono di registrare un ristorante, verranno richieste poche, essenziali informazioni:

- Nome del ristorante
- Descrizione
- Indirizzo

che potranno modificare in seguito, oltre a poter aggiungere ulteriori informazioni in una comoda dashboard user-friendly.

5.2 Homepage

Nella homepage c'è un carosello con i ristoranti preferiti, mostrati solo se l'utente ha effettuato il login, e ha inserito il proprio indirizzo. Questo verrà approfondito nella sezione successiva. Sotto i "recommended" abbiamo un elenco di ristoranti e ad ognuno è associata una pagina. Tutti i ristoranti che compaiono in homepage devono rispettare alcuni criteri, che consistono nell'inserimento di informazioni necessarie al fine di poter rendere la loro pagina utilizzabile dall'utente. Tra i dati richiesti abbiamo:

- Nome del locale
- Nome del proprietario
- Descrizione del locale
- Orari di apertura
- Indirizzo
- Almeno 1 tag

Nell'homepage di un utente registrato, vengono mostrati i locali in un determinato ordine:

1. Per primi i locali che sono nella stessa città dell'utente, che sono **aperti**.
2. Poi i locali che sono nella stessa città, ma sono chiusi
3. I ristoranti che sono in una città diversa che sono aperti
4. I ristoranti in città diversa che sono chiusi

L'idea iniziale era di mostrare solo quelli nella stessa città dell'utente, però sarebbe stato limitativo in quanto inserire un alto numero di locali con città diverse, per fare dei test sull'interfaccia è un processo che richiede molto tempo.

Inoltre è presente una searchbar, che permette di effettuare una ricerca testuale, che filtra tutte le corrispondenze per nome locale, descrizione e tags. A questa sono affiancati due filtri aggiuntivi dove si possono selezionare la città e i tag

5.3 Decoratori per la verifica dello stato del ristorante

Come annunciato nella sezione precedente, ho implementato dei decoratori (*check_restaurant_complete, filter_complete_restaurants*) per garantire che solo i ristoranti completi e validi vengano visualizzati agli utenti. Questa scelta è stata fatta per assicurare che l'esperienza utente non sia compromessa da informazioni incomplete o errate sui ristoranti. I decoratori semplificano il codice delle view (*user_homepage, restaurant_page*) e garantiscono la consistenza nei controlli di validità.

5.4 Sistema di raccomandazione

Per il sistema di raccomandazione, abbiamo adottato un approccio basato sulla posizione e sulle preferenze dell'utente e le interazioni passate. In particolare:

- **Posizione**: Alla base di tutto, un ristorante raccomandato per un certo utente, deve necessariamente trovarsi nella stessa città dell'utente.
- **Ristoranti recenti**: se un certo utente ha ordinato da un ristorante recentemente, è molto probabile che quel ristorante sia in cima alla lista di ristoranti consigliati
- **Ristoranti con tag simili**: ogni ristorante ha un numero di tag compreso tra 1 e 3. Per ogni utente sono calcolate 3 tag che in base agli ordini e le prenotazioni fatte in precedenza. Se un ristorante possiede uno o più dei tag preferiti dell'utente, il suo peso nel sistema di raccomandazione aumenta. Più tag di un ristorante corrispondono ai tag preferiti dell'utente, maggiore sarà il peso del ristorante nel sistema di raccomandazione.

5.5 Gestione degli stati degli ordini

Ho definito diversi stati per gli ordini (*Not Sent, Awaiting Acceptance, Accepted, Ready, Retired*) per permettere un tracciamento dettagliato del processo degli ordini. Questa scelta è stata motivata dalla necessità di fornire chiarezza e trasparenza sia per i gestori dei ristoranti che per i clienti. La chiara distinzione degli stati aiuta a gestire e monitorare gli ordini in modo efficiente.

5.6 Notifiche in tempo reale

Per le notifiche, abbiamo scelto di implementare un sistema basato su WebSocket per garantire aggiornamenti in tempo reale agli utenti. Questa scelta è stata motivata dall'esigenza di fornire un'esperienza utente reattiva e informata, specialmente per aggiornamenti critici come l'arrivo di ordini, il loro cambio di stato e le prenotazioni. La libreria *channels* di Django è stata utilizzata per gestire le comunicazioni in tempo reale.

5.7 Caricamento delle immagini

Per il caricamento delle immagini, utilizziamo una cartella locale denominata *media*. Inizialmente, avevo previsto di utilizzare Amazon S3 per lo storage delle immagini, ma a causa di problemi con l'account AWS, ho dovuto optare per una soluzione locale.

6 Test

6.1 Test 1: `FilterCompleteRestaurantsTestCase`

Questo test verifica che la vista per la homepage degli utenti mostri solo i ristoranti "completi", ovvero quelli che hanno tutte le informazioni richieste, inclusi gli orari di apertura e tags.

- **Setup:**
  - Creazione di un utente test.
  - Creazione di un ristorante completo con tutte le informazioni necessarie, inclusi indirizzo, descrizione, orari di apertura e tag.
  - Creazione di un ristorante incompleto, mancante degli orari di apertura, ma con indirizzo, descrizione e tag.
- **Test:**
  - Login dell'utente test.
  - Richiesta della homepage dell'utente.
  - Verifica che la risposta contenga solo i ristoranti completi.
  - Verifica che il ristorante incompleto non sia presente nella risposta.

6.2 Test 2: `GenerateRecommendationsTestCase`

Questo test verifica che la funzione `generate_recommendations` restituisca solo i primi tre ristoranti e rispetti altri criteri di filtraggio:

- **Setup:**
  - Creazione di una lista di tag.
  - Creazione di una lista di ristoranti completi con tutte le informazioni necessarie, inclusi indirizzo, descrizione, orari di apertura e tag.
  - Creazione di ordini e prenotazioni per un utente test per i primi tre ristoranti nella lista dei ristoranti completi.
- **Test:**
  - Invocazione della funzione `generate_recommendations` con l'utente test e l'elenco completo dei ristoranti.
  - Verifica che solo i primi tre ristoranti siano raccomandati.
  - Verifica che le raccomandazioni includano solo ristoranti nella città "Test City" con i tag preferiti dell'utente.
  - Verifica che l'ordine delle raccomandazioni sia basato sulle interazioni recenti dell'utente.

7 Presentazione UI

In questa sezione verranno allegati degli screenshot delle principali pagine di ReservEZ.

7.1 Homepage per utente autenticato

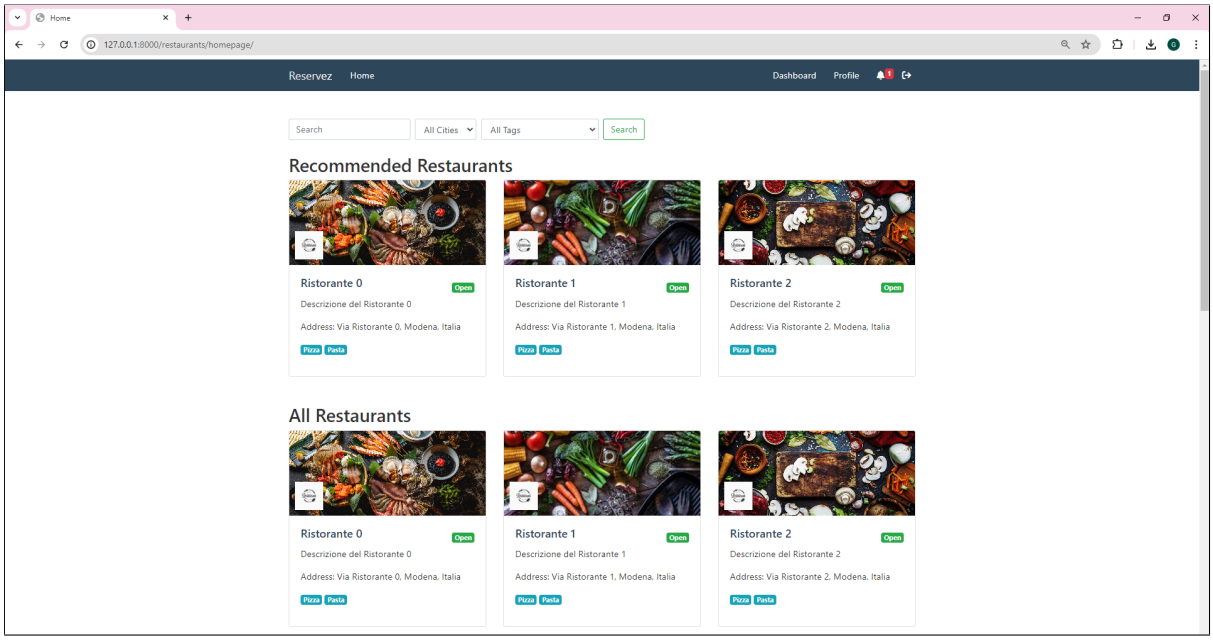

Figura 4: Homepage

7.2 Pagina di un ristorante

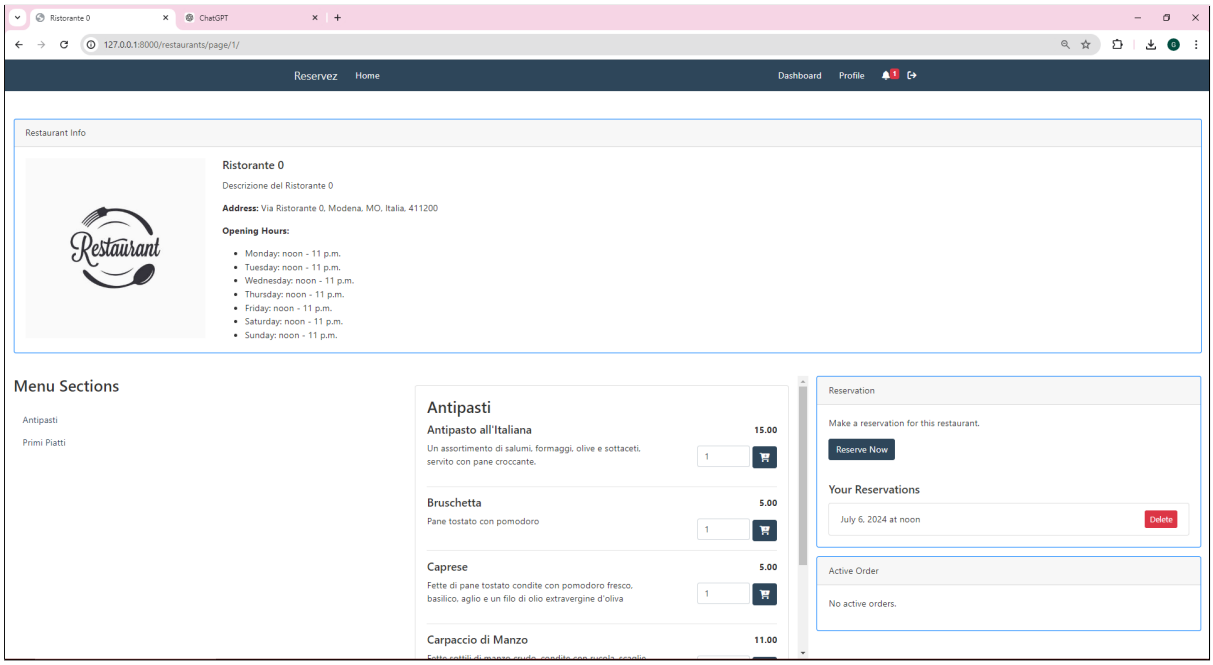

Figura 5: Pagina ristorante

7.3 Pagina di prenotazione per un ristorante

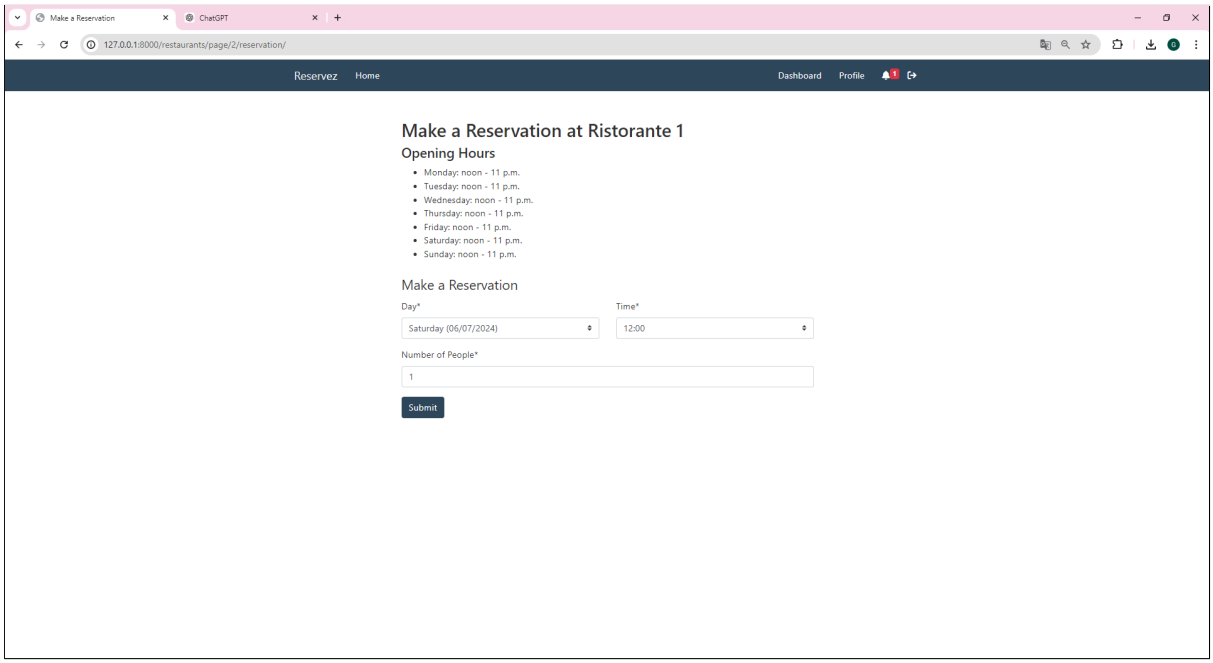

Figura 6: Prenotazione

7.4 Dashboard per gestione di un ristorante

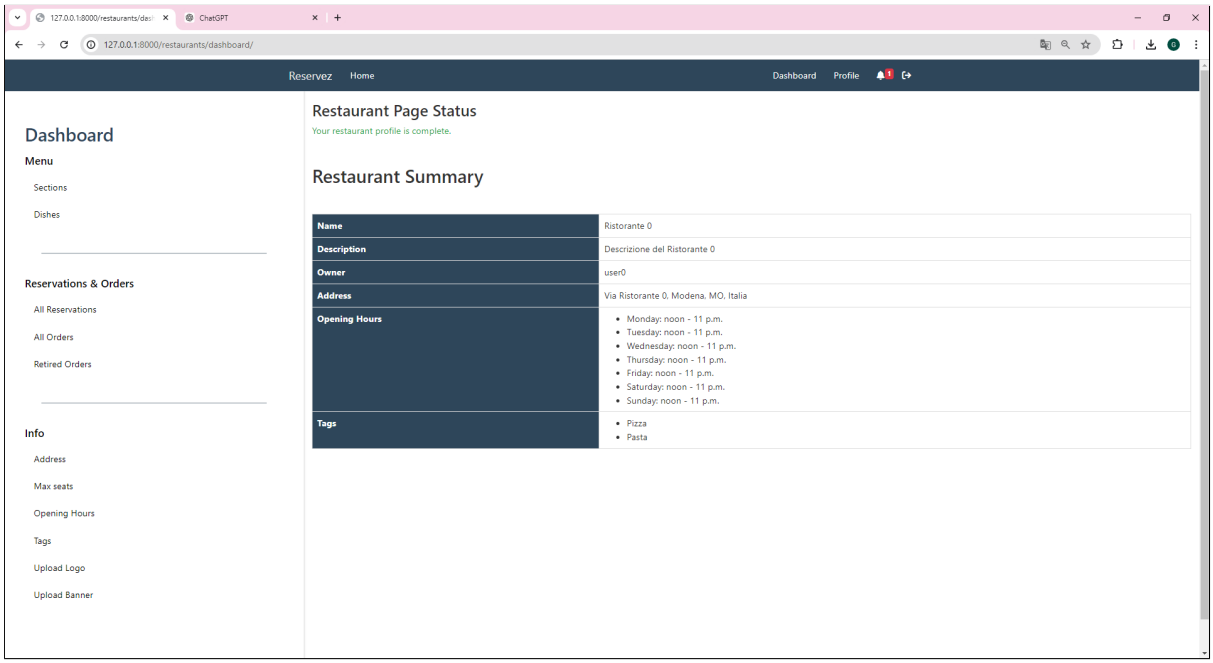

Figura 7: Dashboard