



UNIVERSITÀ DEGLI STUDI DI PALERMO
DIPARTIMENTO DI INGEGNERIA
CORSO DI LAUREA TRIENNALE IN INGEGNERIA INFORMATICA



CISCUSASSI

Documentazione Tecnica

Studenti

Leonardo Giovanni Caiezza
Diego Corona
Luca Gaetani
Daniele Orazio Susino

Docente

Prof. Luca Cruciata

Anno accademico

2024 - 2025

Indice

| | |
|--|-----------|
| 1 Introduzione | 3 |
| 1.1 Descrizione del sistema | 3 |
| 1.2 Vista degli attori | 3 |
| 1.3 Componenti del sistema | 4 |
| 1.3.1 Backend | 4 |
| 1.3.2 Frontend | 5 |
| 1.3.3 Embedded | 6 |
| 1.4 Struttura del progetto | 7 |
| 1.4.1 Guida alla configurazione e avvio del progetto | 8 |
| 1.4.1.1 Variabili d'ambiente | 8 |
| 1.4.1.2 Avvio dell'applicazione | 9 |
| 1.4.1.3 Test locale del progetto | 9 |
| 1.4.1.4 Test della rotta /otp con client REST | 9 |
| 1.5 Disegno tecnico Torretta | 10 |
| 2 Componenti | 12 |
| 2.1 Componenti e librerie Ionic utilizzati | 12 |
| 2.2 Componenti e librerie Angular utilizzati | 13 |
| 2.3 Root-Component: AppComponent | 13 |
| 2.4 Componenti personalizzati utilizzati | 14 |
| 2.4.1 PiattoAmministratoreComponent | 14 |
| 2.4.2 FilialeAmministratoreComponent | 14 |
| 2.4.3 ImpiegatoAmministratoreComponent | 14 |
| 2.4.4 PiattoDelGiornoComponent | 15 |
| 2.4.5 PrenotazioneCardComponent | 15 |
| 2.4.6 ProdottoMenuComponent | 15 |
| 2.4.7 MenuDividerComponent | 16 |
| 2.4.8 ListaMenuComponent | 16 |
| 2.4.9 FooterComponent | 16 |
| 2.4.10 HeaderComponent | 17 |
| 2.4.11 HeroComponent | 17 |
| 2.4.12 RicevutaComponent | 17 |
| 2.4.13 NumeroPostiButtonComponent | 17 |
| 2.4.14 FilialeCardComponent | 18 |
| 2.4.15 LeafletMapComponent | 18 |
| 2.4.16 ProdottoOrdineComponent | 18 |
| 2.4.17 ListaOrdiniComponent | 19 |
| 3 Gestione dei dati persistenti | 20 |
| 3.1 Persistenza remota: Database | 20 |
| 3.1.1 Modello E/R | 20 |
| 3.1.2 Modello Relazionale | 21 |
| 3.1.3 DBMS | 21 |
| 3.2 Persistenza locale: Ionic storage | 21 |
| 3.3 Creazione e testing del DB | 22 |
| 3.3.1 Migrations | 22 |
| 3.3.2 Seeders | 22 |
| 3.3.3 File di migrations | 22 |
| 4 Rotte | 23 |
| 4.1 Gestione delle rotte | 23 |
| 4.1.1 Middleware (Backend) | 23 |
| 4.1.1.1 Autenticazione | 23 |
| 4.1.1.2 Ruoli | 23 |
| 4.1.2 Guardie (Frontend) | 23 |
| 4.1.3 Interceptor (Frontend) | 23 |
| 4.2 Rotte utilizzate | 24 |

| | |
|--|------------|
| 5 Funzionalità | 28 |
| 5.1 Introduzione alle funzionalità offerte dal sistema e note generali | 28 |
| 5.2 Diagramma degli stati | 28 |
| 5.3 Flusso creativo pre-mockup | 30 |
| 5.4 Gestione Account | 33 |
| 5.4.1 Login | 33 |
| 5.4.2 Registrazione | 33 |
| 5.4.3 Cambio Password | 34 |
| 5.4.4 Cambio Email | 35 |
| 5.4.5 Recupera Password | 36 |
| 5.4.6 Visualizza Schermata Account | 37 |
| 5.4.7 Visualizza Schermata Gestione Account | 38 |
| 5.5 Gestione Amministrazione | 39 |
| 5.5.1 Visualizza Schermata Amministrazione | 39 |
| 5.5.2 Gestisci Piatti | 40 |
| 5.5.3 Gestisci Filiali | 41 |
| 5.5.4 Gestisci Impiegati | 42 |
| 5.5.5 Visualizza Utili | 43 |
| 5.5.6 Aggiungi Piatti | 44 |
| 5.5.7 Aggiungi Filiali | 45 |
| 5.5.8 Aggiungi Impiegati | 46 |
| 5.5.9 Modifica Impiegati | 47 |
| 5.5.10 Modifica Filiali | 48 |
| 5.5.11 Modifica Piatti | 49 |
| 5.6 Gestione Cameriere | 50 |
| 5.6.1 Visualizza Tavoli Cameriere | 50 |
| 5.6.2 Visualizza Ordini Cameriere | 52 |
| 5.7 Gestione Chef | 53 |
| 5.7.1 Visualizza Tavoli Chef | 53 |
| 5.7.2 Visualizza Ordini Chef | 54 |
| 5.8 Visualizza Schermata Ristoranti | 56 |
| 5.9 Gestione Prenotazioni | 57 |
| 5.9.1 Imposta Numero Persone | 57 |
| 5.9.2 Prenota | 58 |
| 5.9.3 Scelta Giorno | 60 |
| 5.10 Gestione Ordinazione | 62 |
| 5.10.1 Visualizza Menu Asporto | 62 |
| 5.10.2 Visualizza Menu Tavolo | 64 |
| 5.10.3 Ordina al Tavolo | 66 |
| 5.10.4 Ordina Asporto | 67 |
| 5.10.5 Ordina Ora | 68 |
| 5.10.6 Visualizza Ringraziamenti Asporto | 69 |
| 5.10.7 Visualizza Ordini | 70 |
| 5.10.8 Pagamento Asporto | 71 |
| 5.10.9 Pagamento Carta | 72 |
| 5.10.10 Pagamento Cassa | 73 |
| 5.10.11 Pagamento Tavolo | 74 |
| 5.11 Visualizza Schermata Menu | 75 |
| 5.12 Visualizza Schermata Home | 77 |
| 6 Design | 79 |
| 6.1 Scelte di design | 79 |
| 6.2 Tema e palette colori | 79 |
| 6.3 Fonts | 79 |
| 6.4 Mockup | 80 |
| 7 Copyright e diritto d'autore | 111 |

1 Introduzione

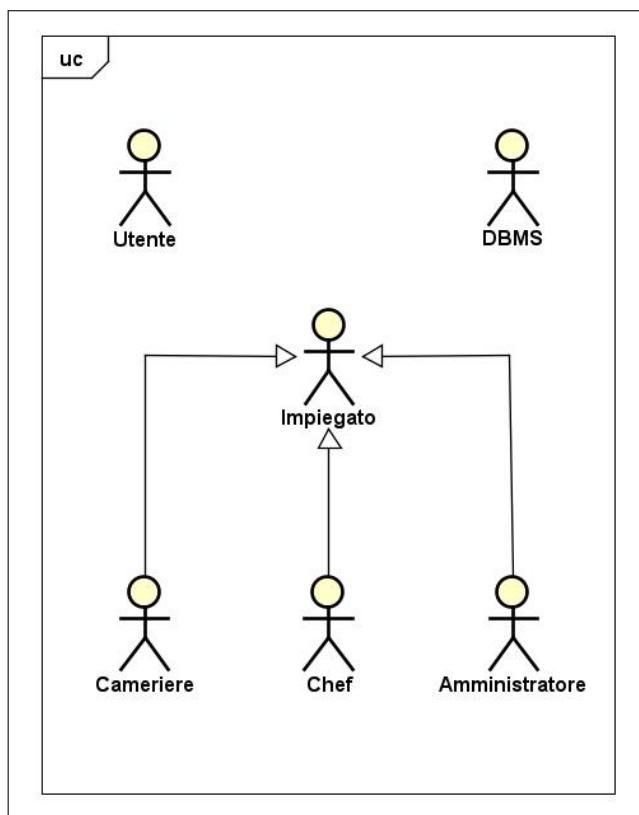
1.1 Descrizione del sistema

Ciscusassi è una piattaforma web progettata per la **gestione di una catena di ristoranti** specializzati in pasta.

La **caratteristica distintiva** del sistema è l'**integrazione di torrette fisiche** posizionate su ciascun tavolo, le quali permettono ai clienti di **scannerizzare un codice QR** per accedere autonomamente al servizio di ordinazione. Attraverso tale interfaccia, è possibile effettuare ordini, monitorarne lo stato di avanzamento e accumulare punti fedeltà.

Ogni torretta è inoltre dotata di un **piccolo schermo che mostra un codice OTP**, il quale consente di validare la presenza del cliente nel locale, garantendo così l'**associazione corretta tra tavolo fisico e ordine digitale**.

1.2 Vista degli attori



Il sistema prevede **quattro principali tipologie di utenti autenticati**:

- **Cliente:** può ordinare da asporto, prenotare un tavolo in presenza, effettuare ordini al tavolo e procedere al pagamento.
- **Cameriere:** può assegnare tavoli ai clienti presenti fisicamente nel locale, visualizzare le prenotazioni e gestire le comande in consegna. Inoltre, ha la possibilità di segnalare se un piatto è stato servito correttamente o se è stato necessario cestinarlo.
- **Chef:** ha accesso all'elenco delle comande ricevute e può segnalare l'inizio e la conclusione della preparazione dei piatti.
- **Amministratore:** ha il compito di gestire il catalogo dei piatti, le filiali e l'organico del personale, potendo effettuare operazioni di creazione, modifica e rimozione su ciascuna di queste sezioni. Inoltre, può consultare gli utili annuali di ogni filiale e scaricarne un report in formato Excel.

Ogni utente autenticato può inoltre accedere al proprio profilo personale, modificarne l'indirizzo e-mail, aggiornare la password e disconnettersi dal sistema.

L'accesso al sito è consentito anche agli **utenti non autenticati**, i quali possono **visualizzare il contenuto informativo** del portale. Tuttavia, la possibilità di effettuare prenotazioni o ordini d'asporto è riservata esclusivamente ai clienti autenticati.

1.3 Componenti del sistema

Il sistema è suddiviso in **tre componenti principali: backend, frontend ed embedded (la tortetta)**. Ognuna di queste parti impiega framework e/o librerie specifiche, scelti in funzione delle esigenze progettuali di leggerezza, modularità e integrazione con l'ambiente di sviluppo. Di seguito vengono riportate le librerie di maggiore rilevanza.

1.3.1 Backend

Il **backend** è sviluppato in **Node.js** con architettura **basata su Express**.

Il **pattern architettonico** scelto è l'**MVCS (Model-View-Controller-Service)**, che permette una chiara separazione delle responsabilità:

- **Model**: Contiene lo stato dell'applicazione e fornisce un'API per manipolare i dati. Gestisce l'incapsulamento dei dati.
- **View**: Rappresenta come i dati saranno visualizzati all'utente. È costituita da pagine e widget che interagiscono principalmente con il Controller.
- **Controller**: Gestisce la logica applicativa, comunicando tra Model, Service e View. Si occupa di elaborare le richieste http.
- **Service**: Funziona come intermediario tra Controller e Model. Recupera i dati dal model e li fornisce al Controller. Gestisce anche la logica di dominio e le referenze dei model.

Il Backend è **scritto in TypeScript** e utilizza una serie di librerie per la validazione, la gestione del database e la gestione delle email:

- **express**: framework web principale per la definizione delle rotte e middleware.
- **express-validator**: libreria middleware per Express.js che fornisce un set completo di strumenti per la validazione e la sanitizzazione dei dati delle richieste HTTP, basata su validator.js, facile da integrare e altamente configurabile.
- **sqlite3**: driver ufficiale per la gestione del database relazionale locale.
- **dotenv**: libreria per caricare variabili d'ambiente da un file .env nel process.env, semplice e utile per la gestione della configurazione in ambienti di sviluppo e produzione.
- **bcriptjs**: libreria JavaScript per la cifratura di password basata sull'algoritmo bcrypt, leggera e compatibile con Node.js e browser, supporta hashing, salting e verifica delle password..
- **nodemailer**: libreria per Node.js che permette di inviare email tramite SMTP o servizi come Gmail, Outlook, ecc.; adatta a contesti server-side.
- **html-to-text**: libreria per convertire contenuti HTML in testo semplice; utile per creare versioni leggibili delle email o per elaborazioni testuali prive di markup.
- **handlebars**: motore di template JavaScript basato su Mustache; consente di generare HTML dinamico a partire da dati e template, mantenendo separata la logica dalla presentazione. È stato utilizzato per generare email con contenuto variabile.
- **jsonwebtoken**: libreria per la creazione e verifica di JSON Web Token (JWT) in Node.js, utilizzata per l'autenticazione e l'autorizzazione, semplice e sicura da integrare nelle API.
- **@faker-js/faker**: libreria per la generazione di valori casuali realistici e coerenti tra di loro.
- **Axios**: libreria per effettuare richieste HTTP, usata per scaricare immagini convertite in Base64.

1.3.2 Frontend

Il frontend è sviluppato con **Ionic 7.2** e **Angular 19**, sfruttando **componenti standalone**. Il sistema offre un’interfaccia moderna, reattiva e compatibile con dispositivi mobili.

- **leaflet**, **@types/leaflet**: libreria open source per la creazione di mappe interattive, leggera e facile da usare, supporta marker, layer, popup e molto altro.
- **xlsx**: libreria JavaScript per leggere, scrivere e manipolare file Excel (.xlsx).
- **xlsx-js-style**: libreria JavaScript per creare e modificare file Excel (.xlsx) che estende **xlsx** aggiungendo il supporto per la formattazione avanzata delle celle, come colori, bordi, font e stili personalizzati.
- **file-saver**: libreria JavaScript per salvare file lato client, consente il download di blob o dati generati dinamicamente direttamente dal browser.
- **rxjs**: libreria per la programmazione reattiva basata su observable; consente la gestione asincrona di eventi.
- **jspdf**: libreria JavaScript per generare file PDF direttamente dal browser, utile per esportare contenuti dinamici in formato stampabile o scaricabile.
- **html2canvas**: libreria JavaScript che cattura screenshot di elementi HTML e li converte in immagini canvas, utile per trasformare contenuti web in formato immagine (da usare, ad esempio, insieme a **jspdf** per creare PDF visivi). Usata nella creazione della ricevuta.
- **date-fns**: libreria per la manipolazione di date in JavaScript, modulare e performante, offre funzioni pure per parsing, formattazione, confronto e calcoli su date.
- **jwt-decode**: libreria JavaScript per decodificare JSON Web Token (JWT) lato client, utile per leggere il payload di un token senza necessità di una chiave segreta o verifica della firma.

1.3.3 Embedded

La **torretta fisica**, presente su ogni tavolo, è basata su **ESP32** ed è programmata tramite **Arduino**.

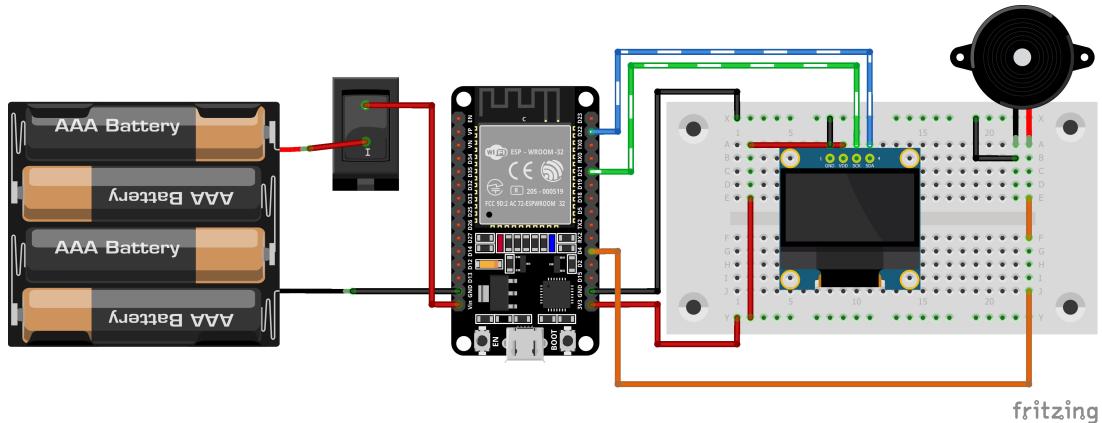
Essa gestisce la connessione WiFi, la comunicazione con il server e la visualizzazione dell'OTP su display OLED.

Per garantire la sicurezza, le credenziali WiFi e gli endpoint API sono contenuti in un file separato () non versionato.

Librerie utilizzate:

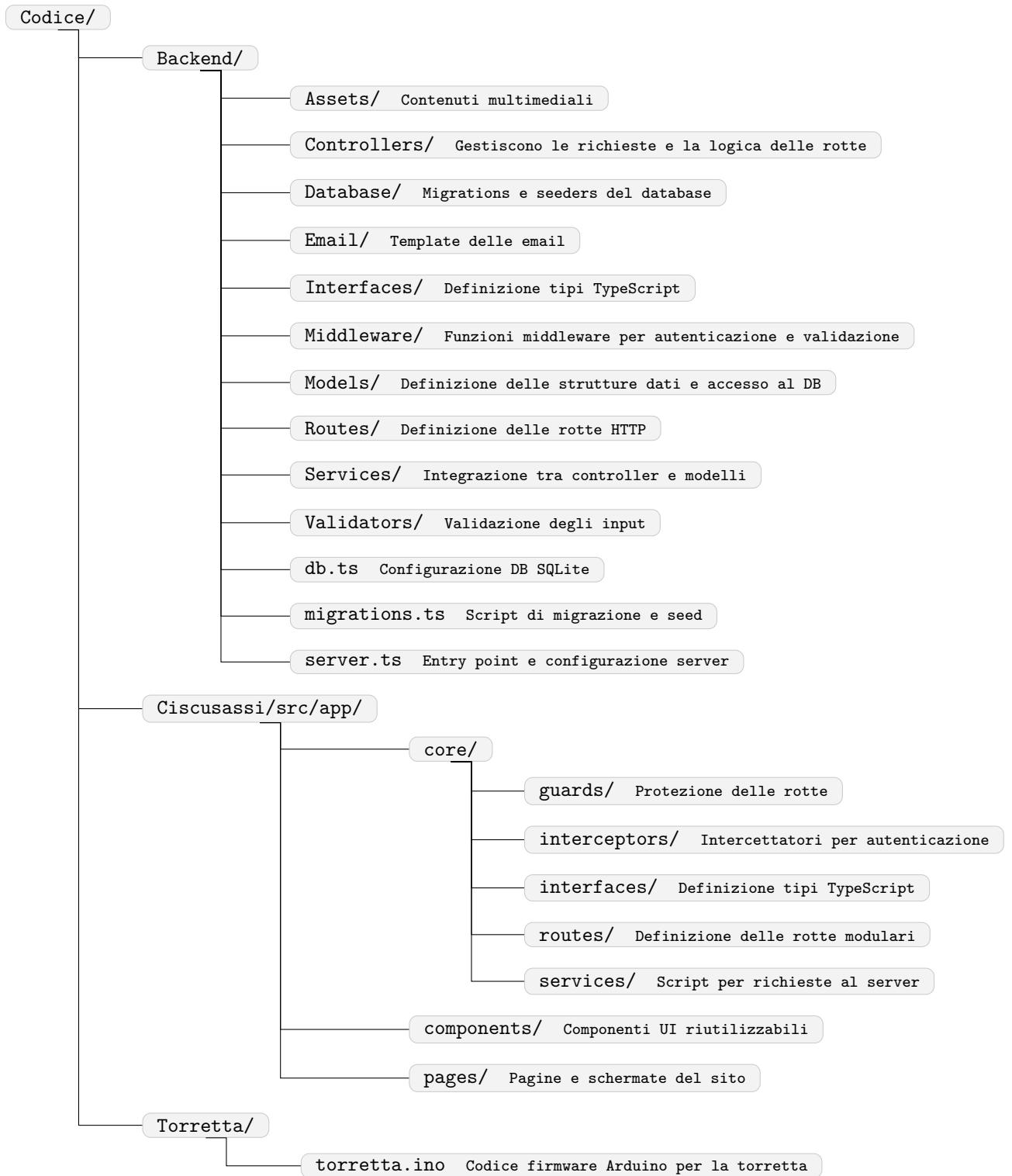
- **WiFi.h, esp_wpa2.h**: per la connessione alla rete WiFi ed eventualmente reti Enterprise (come quella di UNIPA).
- **HTTPClient.h**: per le chiamate HTTP al server Express.
- **Adafruit_SSD1306, Adafruit_GFX, Wire.h**: per la gestione del display OLED.
- **ArduinoJson**: Per il parsing del body di ritorno del server.

Di seguito è riportato lo **schema elettrico** della **torretta**, che mostra i collegamenti tra il microcontrollore ESP32, il display OLED, il buzzer e i relativi pin di alimentazione e comunicazione.



1.4 Struttura del progetto

Viene elencata di seguito la struttura del progetto, in modo tale da **facilitare la ricerca dei file** alla quale si è interessati:



1.4.1 Guida alla configurazione e avvio del progetto

Per il corretto funzionamento dell'applicazione, lato **frontend**, **backend**, ed anche **embedded** è necessario effettuare l'installazione delle librerie e l'impostazione delle variabili d'ambiente.

1.4.1.1 Variabili d'ambiente

Le variabili d'ambiente del frontend sono definite nel file `src/environments/environment.ts`:

```
export const environment = {
  production: false,
  tomtomApiKey: 'API_KEY_TOMTOM', // API_KEY di TOMTOM per le mappe
  apiURL: 'http://localhost:3000', // L'endpoint base del server express
  JWT_SECRET_KEY: 'PASSWORD DEL TOKEN', // La password JWT che deve essere uguale nel
                                         backend
};
```

Mentre nel backend sono contenute nel file `.env`, con la seguente struttura:

```
# Porta del server backend
PORT=3000

# URL per le richieste CORS dal frontend
CORS_ORIGIN='indirizzo del frontend' # Modifica con l'URL reale del tuo frontend

# Chiave segreta per JWT
JWT_SECRET_KEY='jwt_secret_key' # Cambia con una chiave sicura

# Credenziali per l'email
MAIL_USER='tua@email.com' # La tua email
GOOGLE_APP_PASSWORD='tuapasswordapp' # La tua app password o password email

# Auth Torrette
TORRETTA_HEADER='header-torretta'
TORRETTA_SECRET='token-torretta'
```

Lato embedded sono contenute nel file `arduino_secrets.h`, con la seguente struttura:

```
#define SECRET_SSID "" // Il nome della rete alla quale connettersi
#define SECRET_USERNAME "" // In caso di EAP inserire l'username
#define SECRET_PASSWORD "" // La password della rete
#define SECRET_IS_EAP true // Se la rete è EAP (Extensible Authentication Protocol)
#define SECRET_OTP "" // La rotta esplosa dal backend per ricevere l'OTP
#define SECRET_HEADER_NAME "" // Il nome dell'header di autenticazione
#define SECRET_HEADER_VALUE "" // Il valore dell'header di autenticazione
```

1.4.1.2 Avvio dell'applicazione

Per avviare correttamente l'applicazione è necessario installare le dipendenze in entrambe le parti del progetto. Dalla root delle rispettive cartelle, eseguire:

```
npm install
```

Nel caso in cui il database non sia ancora stato creato o popolato, è necessario eseguire preventivamente la procedura di **migration** e **seeding**:

```
npm run migrate
```

Questo comando inizializza le tabelle e inserisce i dati di esempio necessari per il corretto funzionamento e testing dell'applicazione.

I comandi principali per l'esecuzione dei due ambienti sono già definiti all'interno dei file `package.json`:

- **Backend:**

```
"scripts": {  
    "start": "node server.js",  
    "dev": "nodemon",  
    "migrate": "npx ts-node migrations.ts --reset --seed"  
}
```

- **Frontend:**

```
"scripts": {  
    "dev": "ionic serve --external -- --disable-host-check"  
}
```

1.4.1.3 Test locale del progetto

Una volta configurato l'ambiente ed eseguite eventuali migrazioni, è possibile testare localmente l'applicazione. Per avviare il backend ed il frontend in modalità sviluppo, richiamare nelle rispettive cartelle:

```
npm run dev
```

1.4.1.4 Test della rotta /otp con client REST

Per testare la rotta GET `/:id_torretta/otp`, è possibile utilizzare strumenti come **Insomnia**, **Postman** o **Thunder Client** (estensione per Visual Studio Code). Di seguito vengono riportati gli step da compiere per testare la rotta in uno di questi software:

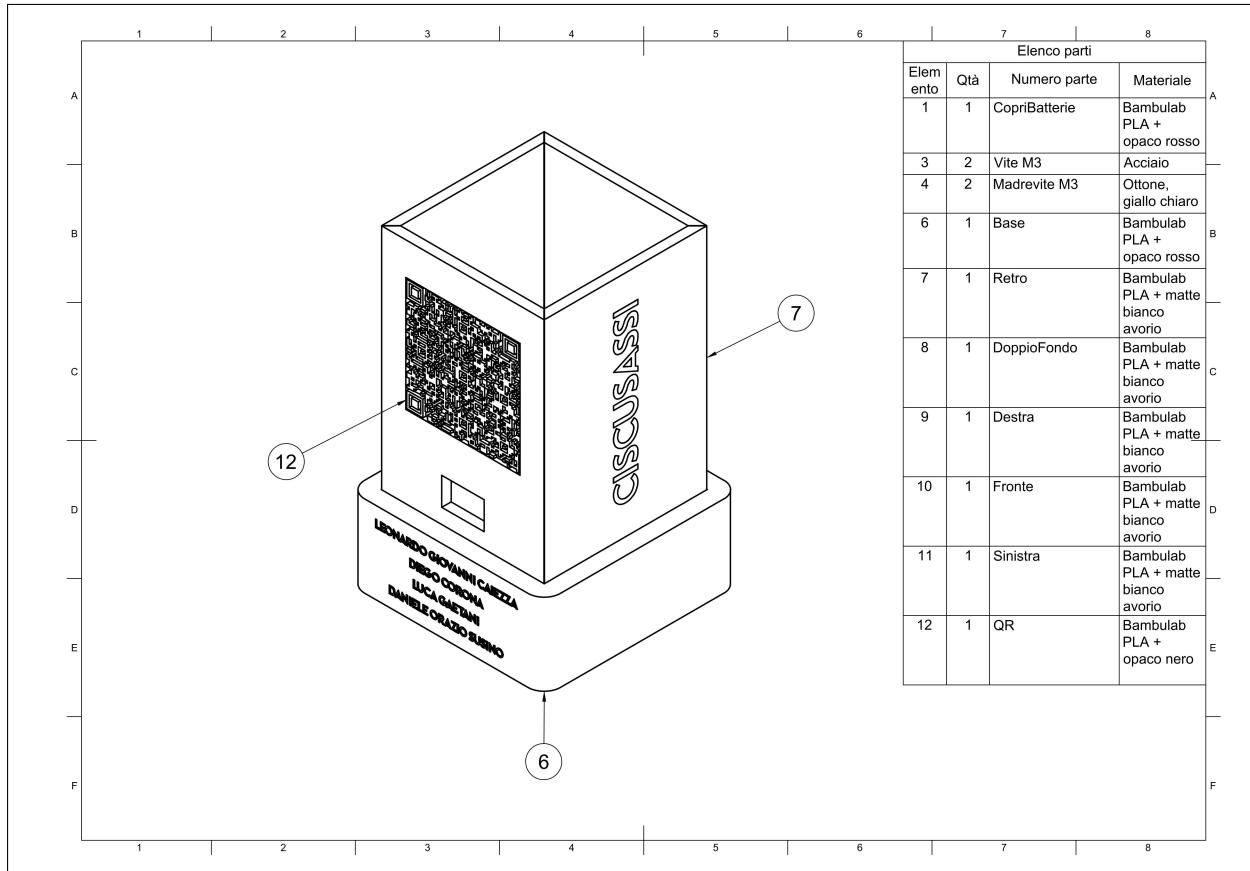
1. Selezionare il metodo **GET**.
2. Inserire l'URL completo della rotta, ad esempio: "`http://localhost:3000/3/otp`" dove 3 rappresenta l'ID della torretta.
3. Nella sezione **Headers**, aggiungere un nuovo header:
 - `x-torretta-auth` come chiave (corrispondente al valore della variabile `TORRETTA_HEADER` nel file `.env`)
 - `token-torretta` come valore (corrispondente al valore della variabile `TORRETTA_SECRET` nel file `.env`)
4. Inviare la richiesta. Se l'ID è valido, il token è corretto e ci si trova in un orario successivo a una delle fasce previste (es. 13:30) ma precedente alla fascia successiva (es. 19:30), il server restituirà in risposta l'**OTP associato alla prenotazione con orario esattamente pari all'inizio della fascia individuata**.

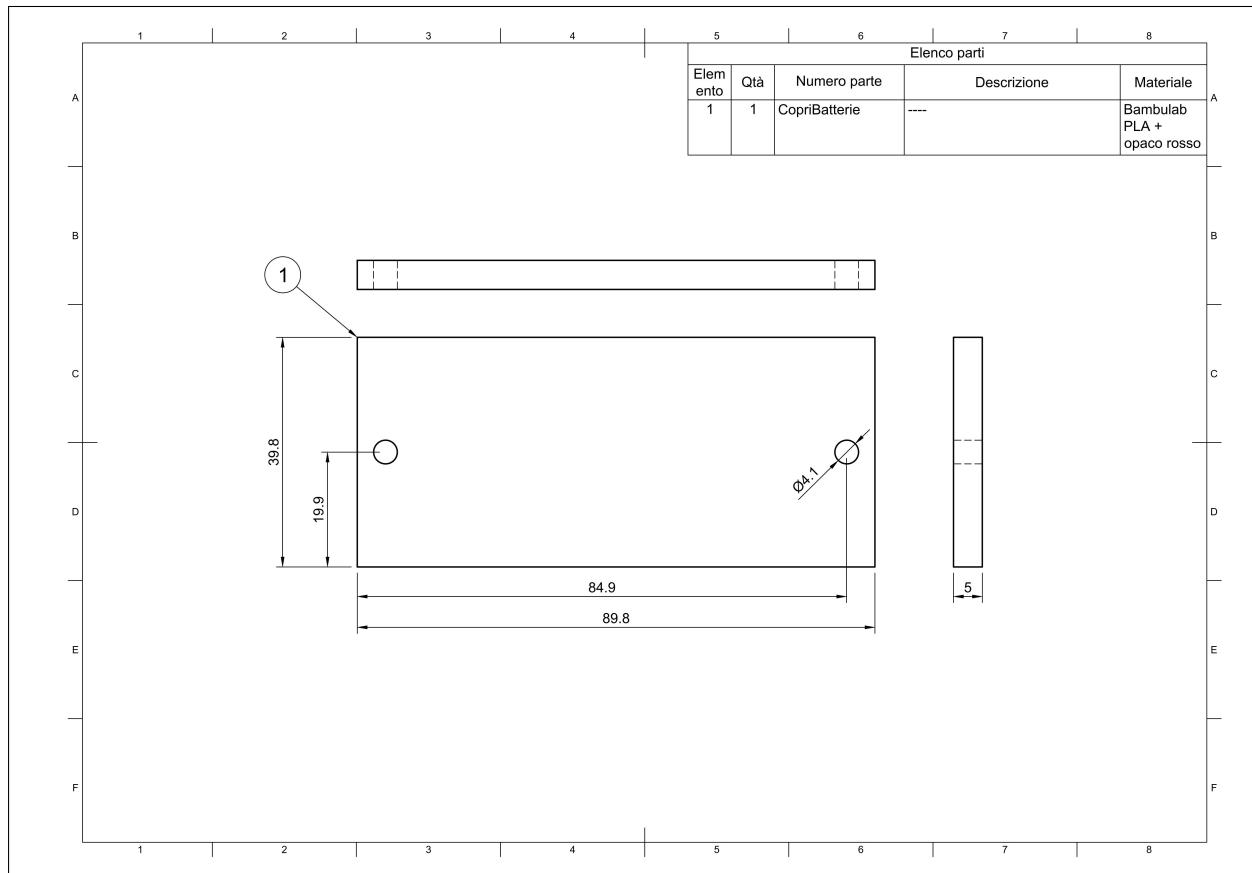
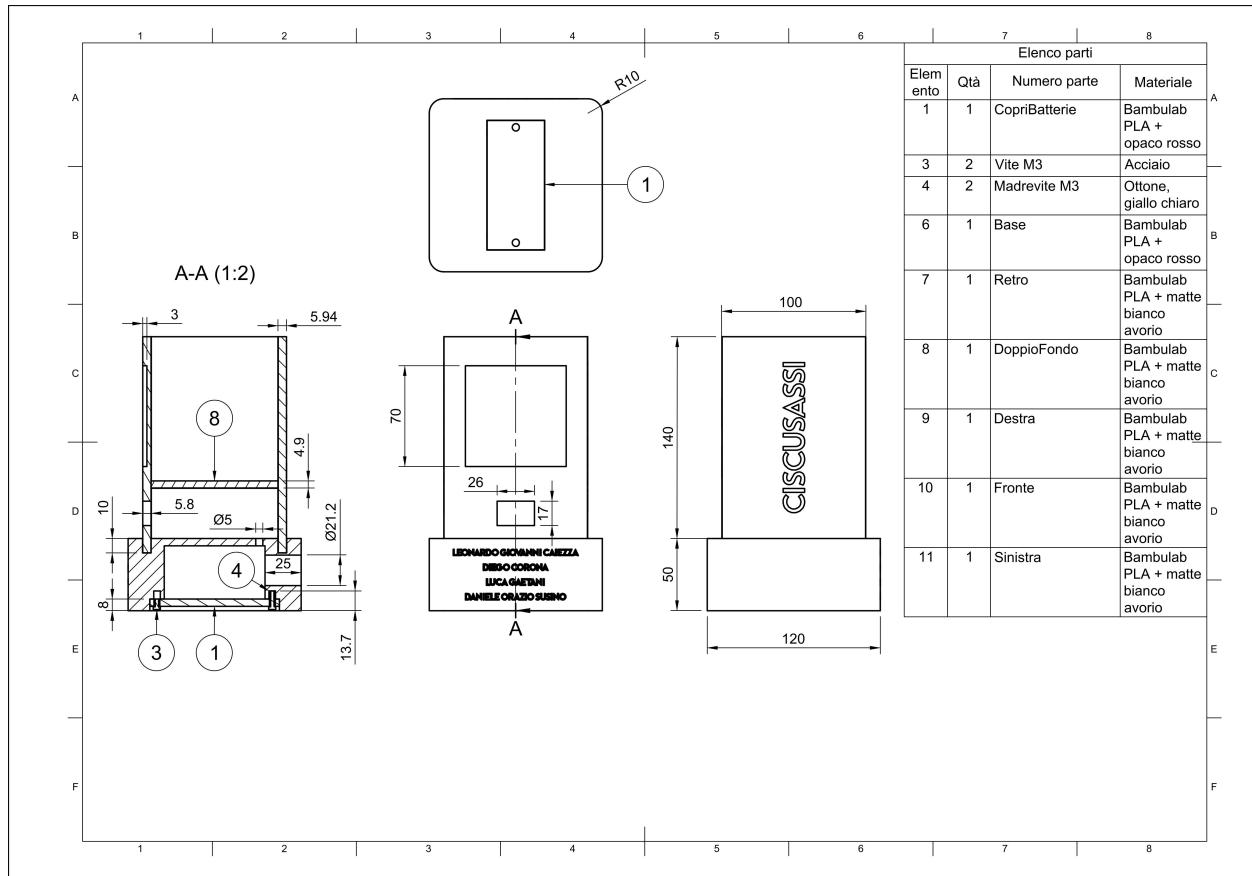
In questo modo è possibile simulare il comportamento della torretta fisica direttamente da un'interfaccia di test HTTP.

1.5 Disegno tecnico Torretta

Il **disegno tecnico** della torretta illustra in dettaglio la struttura del dispositivo.

La torretta è dotata di uno **schermo integrato** che visualizza in tempo reale un **codice OTP (One-Time Password)** generato al momento della prenotazione. Questo codice rappresenta la **chiave d'accesso sicura e temporanea per poter effettuare ordini al tavolo**, garantendo un sistema di autenticazione semplice ma efficace.





2 Componenti

2.1 Componenti e librerie Ionic utilizzati

- **@ionic/angular/standalone**: componenti UI e strumenti per la piattaforma Ionic.
- **@ionic/storage-angular**: per la gestione della persistenza locale lato utente.
- **ionicons**: libreria di icone open source sviluppata da Ionic, usata per aggiungere icone vettoriali e personalizzabili nelle applicazioni web e mobile.
- **ion-app**: il contenitore principale dell'applicazione Ionic, che gestisce il ciclo di vita e l'ambiente generale dell'app.
- **ion-avatar**: contenitore circolare per mostrare un'immagine del profilo o simile.
- **ion-button**: pulsante personalizzabile con supporto per routing, eventi e stili.
- **ion-card**: contenitore a forma di "scheda", usato per evidenziare un blocco di contenuto.
- **ion-card-content**: sezione interna del **ion-card** che contiene il contenuto principale della card.
- **ion-chip**: componente per mostrare etichette cliccabili, spesso usato per filtri o categorie con colori e stato outline per selezione.
- **ion-col**: rappresenta una colonna nella griglia. Può essere ridimensionata e adattata alle varie dimensioni dello schermo.
- **ion-content**: contiene il contenuto principale della pagina. È scrollabile per impostazione predefinita e si adatta alla visualizzazione su dispositivi mobili.
- **ion-datetime**: componente per selezionare data e/o ora, con varie modalità di presentazione (ad esempio solo data, solo ora o entrambe).
- **ion-header**: rappresenta l'intestazione della pagina. Generalmente contiene toolbar, titoli o pulsanti di navigazione.
- **ion-icon**: componente per mostrare icone vettoriali.
- **ion-img**: componente per visualizzare un'immagine ottimizzata in Ionic.
- **ion-input**: componente per l'inserimento di testo (email, password, ecc.). Supporta varie proprietà come tipo, etichetta, placeholder e reactive forms.
- **ion-input-otp**: componente specializzato per l'inserimento di codici OTP (one-time-password), tipicamente con un numero fisso di caratteri e stile specifico.
- **ion-input-password-toggle**: piccolo componente aggiuntivo inseribile dentro un **ion-input** per permettere all'utente di mostrare/nascondere la password. Tipicamente posizionato nello slot.
- **ion-item**: contenitore usato per allineare e strutturare elementi di input, testo, icone, ecc. Molto comune nei form per creare righe coerenti dal punto di vista visivo.
- **ion-label**: testo descrittivo associato a un elemento come **ion-item**, usato per titoli o etichette.
- **ion-list**: componente usato per elencare un insieme di elementi, spesso in combinazione con **ion-item**. Viene usato qui per mostrare liste di suggerimenti (autocomplete).
- **ion-menu**: componente per un menu laterale (drawer) che può essere aperto o chiuso; tipicamente usato per la navigazione principale nelle app.
- **ion-menu-toggle**: wrapper che permette di chiudere automaticamente il menu quando si clicca su un elemento al suo interno.
- **ion-row**: rappresenta una riga nella griglia. Contiene uno o più **ion-col**.
- **ion-router-outlet**: contenitore per la navigazione con routing, che carica i componenti delle pagine in base alla route attiva.
- **ion-searchbar**: componente per la barra di ricerca con supporto a debounce, animazioni e pulsante per cancellare il testo. Utile per filtri dinamici.
- **ion-select**: componente per selezionare un'opzione da una lista, funziona come una dropdown. Supporta varie interfacce (popover, alert, etc.) ed è usato per selezioni singole o multiple.
- **ion-select-option**: componente figlio di **ion-select** che rappresenta ogni singola opzione selezionabile.

- **ion-spinner**: indicatore di caricamento animato.
- **ion-text**: usato per applicare stili (come colore o allineamento) al testo in modo dichiarativo. Utile per rispettare il design system di Ionic, soprattutto con temi e colori dinamici.
- **ion-textarea**: componente per input testuale multilinea, utile per inserire descrizioni o testi lunghi.
- **ion-title**: visualizza un titolo testuale, tipicamente all'interno di una toolbar.
- **ion-toolbar**: barra degli strumenti che può includere titoli, pulsanti, segmenti, ecc. Usato comunemente all'interno di **ion-header**.
- **ion-grid**: sistema di layout a griglia usato per organizzare il contenuto in righe e colonne.
- **ion-modal**: componente modale usato per finestre popup con contenuto personalizzato, gestisce apertura, chiusura e backdrop.
- **NavController**: controller per la gestione della navigazione tra pagine o viste in un'app; fornisce metodi come `navigateForward()`, `navigateBack()` per muoversi nello stack di navigazione.
- **ToastController**: controller per la creazione e visualizzazione di messaggi toast temporanei; utile per fornire feedback rapido all'utente senza interrompere il flusso dell'app.
- **AlertController**: controller per mostrare finestre di dialogo modali con messaggi, pulsanti e input; adatto per notifiche, conferme o richieste di interazione da parte dell'utente.
- **IonicStorageModule**: modulo di Ionic che fornisce un sistema di storage asincrono e persistente, compatibile con vari engine (IndexedDB, SQLite, localStorage).

2.2 Componenti e librerie Angular utilizzati

- `@angular/*`: core framework Angular, aggiornato alla versione 19.
- **CommonModule**: modulo Angular che fornisce direttive comuni come `ngIf` e `ngFor`, necessario nei moduli funzionali.
- **FormsModule**: modulo che abilita l'uso di form template-driven e il two-way data binding con `[(ngModel)]`.
- **RouterModule**: modulo che abilita la gestione delle route e la navigazione tra pagine in un'app Angular.
- **DomSanitizer**, **SafeHtml**: componenti di sicurezza di Angular per la gestione sicura di contenuti HTML; **DomSanitizer** consente di sanificare dati potenzialmente pericolosi (come HTML o URL) per prevenire attacchi XSS, mentre **SafeHtml** rappresenta una versione sicura di contenuto HTML da usare nel template. Fanno parte del modulo platform-browser di Angular.

2.3 Root-Component: AppComponent

Il componente principale del software AppComponent si presenta in `src/app` e funge da punto di ingresso dell'app. All'interno del suo template (`app.component.html`), troviamo un layout classico Ionic composto da `ion-app`, un'intestazione (`app-header`), un menu laterale (`ion-menu`) visibile solo su schermi piccoli, e un `ion-router-outlet` centrale dove vengono caricate dinamicamente le pagine in base al routing. A chiusura c'è un `app-footer` che accompagna ogni schermata.

Il menu laterale contiene diverse voci (Home, Menu, Ristoranti, Ordina Ora, Prenota), ciascuna associata a un'icona e a una route specifica. Il menu è progettato per essere mobile-friendly, dato che viene nascosto sui dispositivi di dimensioni medie o superiori, come indicato dalla classe `ion-hide-md-up`. Dal lato TypeScript, AppComponent gestisce due responsabilità principali. In primo luogo, inizializza lo storage locale, e in secondo luogo, imposta un listener sugli eventi del router per aggiornare dinamicamente il titolo della pagina, rendendolo coerente con l'ultima sezione visitata. Questo migliora l'usabilità e l'accessibilità, soprattutto su dispositivi mobili o in ambienti in cui il titolo della pagina è visibile. Un'altra parte importante dell'inizializzazione è la registrazione delle icone di Ionicons utilizzate nel menu: vengono caricate solo quelle effettivamente usate, migliorando l'efficienza. Per quanto riguarda il file di routing, l'app utilizza una configurazione basata su lazy loading, in cui ciascuna route specifica il componente da caricare solo quando necessario. Ciò è essenziale per mantenere i tempi di caricamento rapidi, specialmente in applicazioni con molte pagine come questa. Ogni route è associata anche, in

alcuni casi, a un guard di autenticazione (authGuard) che controlla il ruolo dell'utente prima di permettere l'accesso a determinate sezioni dell'app. Ad esempio, le pagine di ordinazione o prenotazione sono accessibili solo ai clienti, mentre la dashboard amministrativa è riservata agli amministratori.

L'applicazione è modulare e segue buone pratiche di sviluppo Angular e Ionic. È progettata per offrire un'esperienza fluida agli utenti con ruoli diversi, mantenendo una gestione rigorosa dei permessi e un'architettura scalabile per l'evoluzione futura.

2.4 Componenti personalizzati utilizzati

In questa sezione, tutti i componenti saranno sotto la cartella `src/app/components/`.

2.4.1 PiattoAmministratoreComponent

Il componente PiattoAmministratore è progettato per mostrare le informazioni dettagliate di un singolo piatto all'interno dell'interfaccia amministrativa, consentendo la gestione del piatto stesso e la selezione del piatto del giorno. Riceve in input un oggetto prodotto di tipo `ProdottoRecord`, che contiene tutte le informazioni rilevanti come nome, immagine, descrizione, categoria e costo. Inoltre, riceve l'identificativo `idPiattoDelGiorno` per determinare se il piatto attualmente visualizzato è selezionato come piatto del giorno. La visualizzazione è organizzata tramite componenti Ionic come `ion-card`, `ion-icon`, `ion-img`, e `ion-button` per mantenere un design coerente e responsivo. Il componente mostra l'immagine del piatto, il nome, il prezzo, la categoria e una descrizione dettagliata. Se il piatto è quello del giorno, la card assume uno sfondo verde e l'icona della stella viene evidenziata in modo diverso, con animazioni come la rotazione dell'immagine. L'icona a forma di stella funge da bottone per selezionare o deselezionare il piatto del giorno. Quando viene cliccata, il componente emette un evento contenente l'oggetto prodotto tramite `changePiattoDelGiornoEmitter`, così da permettere al componente genitore di aggiornare la selezione. I pulsanti sottostanti consentono di navigare alla pagina di modifica del piatto, passando il piatto stesso nello stato della navigazione, o di aprire un alert di conferma per la cancellazione tramite l'emissione di un evento con `showAlertDeletePiattoEmitter`. Questo componente fornisce un'interfaccia chiara e funzionale per la gestione amministrativa dei piatti, mantenendo separata la logica di presentazione dalla gestione degli eventi e delle interazioni.

2.4.2 FilialeAmministratoreComponent

Il componente FilialeAmministratore rappresenta una scheda informativa dedicata alla gestione di una singola filiale da parte dell'amministratore. Lo scopo è mostrare in modo chiaro i dati principali della filiale e permettere all'utente di accedere rapidamente alle funzionalità di modifica, gestione del personale e rimozione. Il componente riceve in input un oggetto filiale di tipo `FilialeRecord`, che contiene tutte le informazioni necessarie per popolare la card, come l'immagine, l'indirizzo, il comune e il numero totale di tavoli. L'interfaccia è costruita con componenti Ionic per garantire un aspetto coerente e responsivo, utilizzando `ion-card`, `ion-text`, `ion-img` e `ion-button`. Nella parte inferiore della card si trovano tre pulsanti che consentono rispettivamente di modificare la filiale, gestire gli impiegati e rimuovere la filiale stessa. Il pulsante per modificare la filiale reindirizza alla pagina di modifica passando l'intero oggetto filiale nello stato di navigazione, mentre quello per la gestione degli impiegati passa l'identificativo della filiale sia tramite stato che come query parameter nella navigazione. Il pulsante per rimuovere la filiale chiama un metodo che emette un evento con l'oggetto filiale, consentendo al componente genitore di aprire un alert di conferma o gestire la cancellazione. Questo componente si propone come una soluzione modulare e chiara per consentire all'amministratore di gestire più sedi attraverso un'interfaccia compatta e intuitiva, mantenendo separata la logica di presentazione dalla gestione degli eventi.

2.4.3 ImpiegatoAmministratoreComponent

Il componente ImpiegatoAmministratore è progettato per visualizzare i dettagli di un singolo dipendente all'interno dell'interfaccia amministrativa. Riceve in input un oggetto impiegato di tipo `ImpiegatoRecord` che contiene informazioni quali nome, cognome, matricola, ruolo, email, data di nascita e foto del dipendente. Questi dati vengono mostrati in una card Ionic, con un layout chiaro che mette in evidenza le informazioni personali e professionali del dipendente. L'interfaccia utilizza componenti Ionic come `ion-card`, `ion-img`, `ion-text` e `ion-button` per garantire un aspetto coerente e responsivo. Viene visualizzata la foto del dipendente in cima alla card, seguita dal nome completo e da dettagli aggiuntivi come matricola,

ruolo, email e data di nascita. Sono presenti due pulsanti: uno per navigare alla pagina di modifica dei dati del dipendente, passando l'oggetto impiegato nello stato della navigazione, e un altro che, una volta cliccato, emette un evento per mostrare un alert di conferma prima di procedere con la rimozione o licenziamento del dipendente. Questo evento viene emesso tramite showAlertDeleteImpiegatoEmitter, permettendo al componente genitore di gestire l'azione di cancellazione.

2.4.4 PiattoDelGiornoComponent

Il componente PiattoDelGiorno si occupa di mostrare dinamicamente il piatto del giorno prelevandolo da un servizio esterno. All'avvio, durante il caricamento dei dati, viene mostrato uno spinner animato per indicare all'utente che l'operazione è in corso. Se la chiamata al servizio va a buon fine e riceve i dati corretti, il componente visualizza il nome del piatto (convertito in maiuscolo), la sua immagine, il prezzo e una breve descrizione, il tutto presentato con uno stile chiaro e leggibile su sfondo scuro. Inoltre, viene segnalato un vantaggio per il cliente: acquistando il piatto del giorno si guadagnano 10 punti fedeltà bonus. Nel caso si verifichi un errore durante il caricamento, viene mostrato un messaggio di errore chiaro che invita a riprovare, mantenendo comunque la stessa coerenza visiva. Se non è stato impostato nessun piatto del giorno, viene mostrato un messaggio specifico. La gestione dello stato interno prevede variabili booleane per il controllo del caricamento e degli errori, mentre la logica di ricezione e validazione dei dati è delegata a un metodo dedicato che aggiorna lo stato e gestisce eventuali problemi. Questo approccio garantisce un'esperienza utente fluida e informativa, mantenendo il codice ben organizzato e facilmente manutenibile.

2.4.5 PrenotazioneCardComponent

Il componente PrenotazioneCard è stato progettato per mostrare in modo chiaro e riassuntivo le informazioni relative a una prenotazione effettuata da un utente. Si tratta di un componente Angular standalone, costruito utilizzando i componenti UI di Ionic, come IonCard, IonIcon, IonText e IonButton, ed è pensato per essere facilmente riutilizzabile all'interno di liste o schermate personali. Il componente riceve un oggetto di tipo PrenotazioneWithFiliale tramite l'input prenotazione, che rappresenta i dettagli completi di una prenotazione, incluso l'indirizzo della filiale, il comune, la data e ora, e il numero di posti prenotati. La scheda è costruita per essere visivamente ordinata: ogni riga della card mostra un'icona (location, calendario, persona o persone) seguita dal dato corrispondente. In particolare, viene mostrata l'icona "person" se il numero di persone è uno, oppure l'icona "people" se superiore, adattando dinamicamente anche il testo ("posto" o "posti"). Per quanto riguarda la data, viene gestita tramite una funzione formattaData, che riceve una stringa o un numero, la trasforma in oggetto Date, e la restituisce formattata nel formato gg/mm/aaaa - hh:mm. Questo garantisce coerenza nella presentazione dell'orario su tutta l'interfaccia. Se la data ricevuta non è valida, la funzione restituisce una stringa vuota e registra un errore nella console. Una funzionalità centrale del componente è il pulsante "Cancella Prenotazione", posizionato in fondo alla card. Quando viene cliccato, attiva un EventEmitter (confermaCancellazioneEmitter) che comunica al componente padre l'intenzione dell'utente di cancellare la prenotazione, passando come parametro l'ID corrispondente. In questo modo, la logica di cancellazione effettiva può essere gestita esternamente, mantenendo il componente semplice e focalizzato solo sulla presentazione e interazione di base.

La cancellazione di una prenotazione è sottoposta a un meccanismo di conferma tramite un alert, che evita cancellazioni accidentali. La conferma è condizionata allo stato della prenotazione stessa, verificando che non sia già stata assegnata a un tavolo; in caso contrario, viene mostrato un toast che informa l'utente dell'impossibilità di cancellazione. La procedura di cancellazione invia una richiesta al backend e aggiorna la lista delle prenotazioni sul client, offrendo un feedback visivo chiaro sia in caso di successo che di errore.

L'architettura complessiva del file TypeScript riflette una chiara separazione delle responsabilità, con metodi distinti per la gestione del caricamento dati, del filtraggio, della navigazione e delle azioni utente. Il codice si avvale di meccanismi tipici di Angular e Ionic per il binding, l'aggiornamento reattivo della UI e la gestione asincrona dei dati, garantendo un'esperienza utente fluida e coerente.

2.4.6 ProdottoMenuComponent

Il componente ProdottoMenu ha il compito di visualizzare un piatto all'interno del menù e permettere all'utente di interagire con esso, ad esempio aggiungendolo o rimuovendolo dal carrello. Si tratta di un

componente standalone, semplice ma versatile, progettato per adattarsi dinamicamente in base al contesto in cui viene utilizzato.

Al suo interno riceve tre input principali. Il primo è il prodotto stesso, rappresentato da un oggetto ProdottoRecord contenente tutte le informazioni necessarie come nome, descrizione, immagine, prezzo e identificativo. Il secondo, isPiattoGiorno, consente di distinguere visivamente se il piatto rappresentato è il piatto del giorno: in tal caso, viene applicata una classe CSS diversa e lo stile della card risulta più evidenziato. Il terzo input, isOrdinazione, determina se l'utente si trova in fase di ordinazione: se vero, vengono mostrati i pulsanti per aggiungere o rimuovere il prodotto dal carrello e il conteggio aggiornato della quantità selezionata. Il componente si sottoscrive al flusso prodotti\$ del servizio CarrelloService, che rappresenta lo stato corrente del carrello. Ogni volta che il contenuto del carrello cambia, viene calcolato in tempo reale quante istanze del piatto corrente sono presenti e il valore viene assegnato alla variabile quantity, che viene poi mostrata all'utente. Le azioni di aggiunta e rimozione sono delegate direttamente al servizio CarrelloService, il quale gestisce la logica di aggiornamento del carrello centralizzato. Questo approccio consente di mantenere la logica business fuori dal componente, lasciando a quest'ultimo il solo compito di mostrare i dati e rispondere agli eventi. Infine, il componente si prende cura di liberare la sottoscrizione quando viene distrutto, evitando così memory leak. Nel suo insieme, ProdottoMenuComponent rappresenta un elemento centrale e riutilizzabile per la composizione del menù interattivo dell'app, adattandosi sia a contesti informativi che funzionali.

2.4.7 MenuDividerComponent

Il componente MenuDividerComponent rappresenta un pulsante visivo che funge da divisore e titolo per le varie sezioni del menu, mostrando un'immagine di sfondo personalizzata e un testo descrittivo. Riceve come input il titolo da visualizzare e l'URL dell'immagine di sfondo, applicata dinamicamente tramite binding allo stile del div interno. Quando l'utente clicca sul pulsante, il componente emette un evento (ApriMenuEmit) che notifica al componente genitore di aprire o chiudere la sezione corrispondente del menu. Questa struttura semplice e modulare consente una facile integrazione nel menu principale, facilitando la gestione interattiva delle categorie tramite un'interfaccia chiara e accattivante. Il componente è standalone, utilizza IonButton di Ionic per lo stile e il comportamento del pulsante, e non mantiene stato interno oltre a quello necessario per l'input e l'emissione dell'evento.

2.4.8 ListaMenuComponent

Il componente ListaMenu gestisce la visualizzazione dinamica di un menu suddiviso in categorie come antipasti, primi, dolci e bevande. All'avvio, recupera tramite un servizio esterno la lista completa dei piatti e li filtra in base alla categoria di appartenenza, aggiornando lo stato interno per riflettere il caricamento o eventuali errori. L'interfaccia presenta ogni categoria con un componente separato (MenuDividerComponent) che funge da titolo cliccabile con immagine di sfondo. Quando l'utente seleziona una categoria, il componente mostra o nasconde la lista dei piatti appartenenti, utilizzando il metodo AperturaLista per gestire lo stato di apertura. Ogni piatto viene rappresentato tramite il componente ProdottoMenuComponent, che riceve informazioni sul prodotto e sul contesto di ordinazione. Il componente utilizza variabili booleane per gestire lo stato di caricamento e di errore, garantendo una corretta gestione delle chiamate asincrone. Il design modulare e l'uso di componenti standalone facilitano la manutenibilità e la riusabilità del codice, offrendo all'utente una navigazione chiara e interattiva tra le diverse sezioni del menu.

2.4.9 FooterComponent

Il componente Footer è un elemento semplice, realizzato con ion-footer e una toolbar rossa che contiene a sinistra il copyright “© CISCUSASSI 2025” e a destra i link ai profili LinkedIn degli sviluppatori, mostrati con testo bianco per garantire contrasto e leggibilità. Questo permette di mantenere una presenza discreta ma sempre visibile delle informazioni legali e degli autori in tutte le pagine dell'app.

2.4.10 HeaderComponent

Il componente Header gestisce la barra di navigazione principale dell'app Ionic, adattandosi sia a dispositivi mobili sia a schermi più grandi. Al suo interno, presenta un menu hamburger per schermi piccoli e una serie di pulsanti di navigazione visibili su desktop, che permettono di accedere alle pagine principali come Home, Menu, Ristoranti, Ordina Ora, Prenota e il pulsante Indietro per la navigazione delle pagine. Il pulsante "Ordina Ora" include anche la funzione di svuotare il carrello prima della navigazione. Sulla destra, il componente mostra pulsanti dinamici basati sul ruolo dell'utente autenticato, indirizzandolo verso pagine dedicate come amministrazione, pannello chef o camerieri. La gestione dello stato dell'utente (ruolo e avatar) avviene tramite sottoscrizioni ai servizi di autenticazione, mentre la funzione di svuotare il carrello è delegata a un servizio dedicato.

2.4.11 HeroComponent

Il componente Hero è responsabile della visualizzazione dell'intestazione grafica introduttiva della pagina, comunemente chiamata "hero section". Questo componente accetta tre input: un titolo, una descrizione e il nome dell'immagine di sfondo, che viene applicata dinamicamente tramite una proprietà di stile inline. Il contenuto testuale, composto dal titolo e dalla descrizione, viene centrato orizzontalmente e reso visivamente leggibile grazie a uno stile tipografico chiaro su sfondo scuro o semi-trasparente. Nella parte inferiore del componente è presente un'icona a forma di chevron orientata verso il basso, utilizzata per suggerire all'utente la possibilità di scorrere verso il contenuto successivo della pagina. Questa icona viene importata dinamicamente e aggiunta al set di icone disponibili tramite la funzione addIcons. La struttura del layout è pensata per offrire un impatto visivo accattivante e adattabile a diversi dispositivi, mentre l'interazione con l'utente resta minimale, limitata alla semplice navigazione visiva. L'intero componente è standalone, sfrutta i moduli standalone di Angular e Ionic, e non mantiene stato interno né effettua chiamate a servizi esterni.

2.4.12 RicevutaComponent

Il componente Ricevuta è incaricato della generazione e visualizzazione del riepilogo finale di un ordine sotto forma di ricevuta, adattandosi sia al servizio al tavolo sia all'asporto. Il componente accetta tre input principali: la lista dei prodotti acquistati, un logo da visualizzare nell'intestazione, e la modalità del servizio selezionato. All'interno dell'interfaccia, viene mostrata una testata contenente il logo, la scritta "Ricevuta" e la data di generazione, seguita da una tabella con l'elenco dei prodotti raggruppati per nome, la quantità ordinata e il prezzo calcolato in base alla moltiplicazione tra costo unitario e quantità. Nel caso di servizio al tavolo, viene mostrata una voce aggiuntiva che evidenzia eventuali sconti fidelity e la divisione del totale tra i commensali (romana), con l'importo sottratto visibile accanto alla descrizione. Il calcolo della differenza tra il totale effettivo e quello ridotto viene gestito da una proprietà dedicata, mentre il valore del totale per il tavolo viene recuperato al momento dell'inizializzazione del componente tramite il TavoloService. Il totale viene calcolato dinamicamente sia per l'asporto che per il tavolo, a seconda della modalità impostata, e viene visualizzato in fondo alla sezione centrale, prima del piè di pagina. Quest'ultimo contiene le informazioni legali e i nomi dei membri del team di sviluppo. La logica di raggruppamento dei prodotti consente di consolidare più istanze dello stesso prodotto in un'unica riga per semplificare la lettura. Il componente non effettua operazioni asincrone complesse ma si basa su dati in input e su chiamate sincrone a metodi dei servizi TavoloService e PrenotazioneService, mantenendo un comportamento reattivo e semplice, pensato per esportazioni o stampe in formato PDF.

2.4.13 NumeroPostiButtonComponent

Il componente NumeroPostiButton rappresenta un singolo pulsante a forma di cerchio, utilizzato per selezionare un numero di posti (persone) in fase di prenotazione o configurazione del tavolo. È progettato per essere altamente riutilizzabile e comunicare in modo semplice con il componente genitore, mantenendo la propria responsabilità ben isolata. Riceve in input un valore numerico number, che rappresenta il numero specifico associato a quel cerchio. Questo numero è mostrato all'interno del cerchio e utilizzato anche come valore di confronto per determinare lo stato visuale del pulsante. Il componente accetta anche personeSelezionate, cioè il valore attualmente selezionato tramite clic, e inputManuale, che rappresenta un eventuale valore inserito manualmente dall'utente: questi due input permettono di applicare dinamicamente delle classi CSS (selected o match-input) per evidenziare il cerchio selezionato o quello

corrispondente al valore inserito. Il componente espone un EventEmitter<number> chiamato selectCircleEmitter, che viene invocato ogni volta che l'utente clicca sul cerchio. In questo modo, il componente genitore può ricevere l'informazione e aggiornare lo stato dell'applicazione di conseguenza. Nel metodo ngOnInit, è presente un controllo difensivo che impedisce l'inizializzazione del componente con valore zero, poiché un numero di posti pari a zero è considerato non valido.

2.4.14 FilialeCardComponent

Il componente FilialeCard ha il compito di rappresentare graficamente una singola filiale attraverso una card interattiva. Il suo scopo è mostrare all'utente in modo chiaro e sintetico le principali informazioni relative a una sede disponibile, come la località, l'indirizzo, gli orari di apertura e l'immagine rappresentativa. L'intero contenitore è racchiuso all'interno di un <ion-button>: cliccare sulla card corrisponde quindi a selezionare la filiale. La card è composta da un'immagine (visualizzata tramite <ion-img>) e tre righe informative, ognuna con un'icona identificativa: una per la posizione (icona location), una per gli orari di apertura (time) e una per le eventuali informazioni aggiuntive come i giorni di chiusura (information-circle). Questi elementi sono strutturati con una disposizione a righe e allineati in modo responsivo grazie a classi personalizzate e utility Ionic. Quando l'utente clicca sulla card, viene chiamato il metodo salvaFiliale(), che emette l'ID della filiale attraverso l'EventEmitter selectFilialeEmitter. Questo consente al componente genitore di reagire all'azione dell'utente, ad esempio salvando la sede scelta o navigando verso un'altra pagina. Infine, il costruttore del componente registra le icone necessarie tramite la funzione addIcons, così da poterle utilizzare nei rispettivi tag <ion-icon> all'interno del template.

2.4.15 LeafletMapComponent

Il componente LeafletMap gestisce la visualizzazione di una mappa interattiva tramite la libreria Leaflet, mostrando le filiali ottenute da un servizio esterno. All'avvio recupera le informazioni delle filiali e, in caso di successo, inizializza la mappa centrata su coordinate predefinite. Ogni filiale viene rappresentata con un marker personalizzato che mostra un'immagine circolare e un tooltip con l'indirizzo. Se l'input redirect è attivo, il marker presenta un popup con un pulsante per prenotare, che al clic salva la filiale selezionata tramite un servizio di prenotazione e reindirizza l'utente a una pagina dedicata; altrimenti il click sul marker apre Google Maps con la posizione della filiale in una nuova scheda. Durante il caricamento mostra uno spinner, mentre in caso di errore visualizza un messaggio di fallback. Il componente gestisce inoltre la corretta pulizia della mappa alla distruzione per evitare perdite di memoria. L'interazione con il DOM per i pulsanti di prenotazione viene gestita tramite eventi dinamici dopo l'apertura dei popup, garantendo un'esperienza utente fluida e integrata.

2.4.16 ProdottoOrdineComponent

Il componente ProdottoOrdine rappresenta un piatto all'interno di un ordine e serve a mostrare tutte le sue informazioni principali (nome, immagine, descrizione, costo, stato attuale) con un'interfaccia che si adatta al ruolo dell'utente che sta visualizzando il contenuto. Può essere utilizzato in contesti in cui si desidera permettere la gestione dei piatti da parte dello staff (ad esempio chef e camerieri) o semplicemente mostrarli al cliente in modo visivo e interattivo. Appena inizializzato, il componente si sottoscrive a un observable fornito dal servizio di autenticazione. Questo consente di rilevare il ruolo dell'utente loggato e salvare questa informazione localmente nella variabile ruolo. Tale informazione è poi utilizzata all'interno del template per decidere dinamicamente quali controlli rendere visibili e con quale comportamento. Ad esempio, se l'utente è uno chef, verranno mostrati i pulsanti per avviare o terminare la lavorazione del piatto. Se è un cameriere, saranno disponibili i controlli per consegnare o cestinare il piatto. Se invece l'utente è un cliente, non potrà interagire con lo stato ma potrà vedere il piatto e modificarne l'opzione "Romana" tramite un checkbox, se disponibile.

Ogni piatto ha uno stato associato (non-in-lavorazione, in-lavorazione, in-consegna, consegnato). Questo stato viene inizializzato nella fase ngOnChanges, dove si verifica se l'input prodotto è stato aggiornato e si aggiorna la variabile statoPiatto di conseguenza. Tutte le operazioni di modifica dello stato vengono poi gestite attraverso chiamate a metodi del servizio OrdineService, che aggiorna il backend e, in caso di successo, aggiorna lo stato localmente nel componente. Infine, il metodo cambiaRomana consente al cliente di modificare l'opzione "Romana" per il piatto selezionato. Anche in questo caso l'aggiornamento

avviene attraverso una chiamata HTTP e il valore booleano viene aggiornato in modo reattivo solo in caso di successo della richiesta.

2.4.17 ListaOrdiniComponent

Il componente ListaOrdini, definito ha il compito di visualizzare una lista reattiva di prodotti associati a uno o più ordini, rendendosi particolarmente utile all'interno di pannelli di gestione come quelli per chef e camerieri. Il componente riceve come input uno stream Observable contenente un array di oggetti OrdProdEstended, che rappresentano le istanze estese di prodotti ordinati. La visualizzazione avviene tramite un ciclo @for, che sfrutta la trackBy basata sull'identificativo dell'associazione ordine-prodotto, per migliorare l'efficienza del rendering. Ogni elemento della lista viene delegato al componente figlio ProdottoOrdineComponent, che si occupa della rappresentazione visiva del singolo prodotto d'ordine. La struttura del componente è pensata per mantenere una separazione pulita tra logica di presentazione e gestione dei dati asincroni, senza introdurre complessità nel ciclo di vita del componente, che infatti non implementa alcuna logica interna nel metodo ngOnInit. L'approccio standalone garantisce maggiore modularità e riusabilità in diversi contesti dell'applicazione. La visualizzazione è reattiva e si aggiorna automaticamente ogni volta che lo stream prodotti\$ emette nuovi valori.

3 Gestione dei dati persistenti

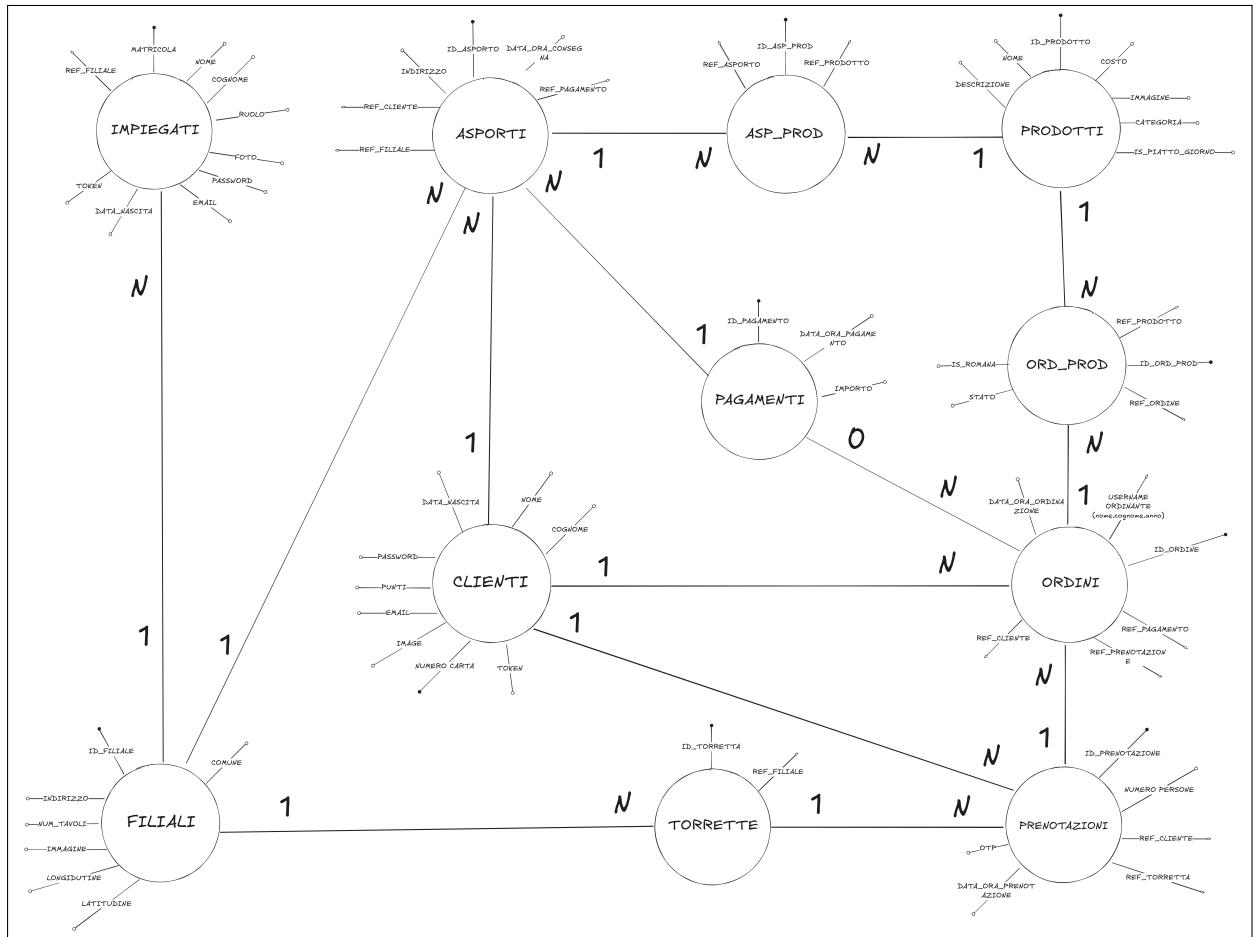
Il sistema prevede una **gestione strutturata e affidabile dei dati persistenti**, fondamentale per garantire la coerenza, l'integrità e l'accessibilità delle informazioni.

La persistenza dei dati è **implementata sia a livello remoto**, attraverso un database centralizzato gestito dal server, **sia a livello locale**, tramite il meccanismo di storage fornito dal framework Ionic, utilizzato nel frontend.

3.1 Persistenza remota: Database

La **persistenza remota** è affidata a un database centralizzato gestito lato server. Essa viene utilizzata per memorizzare tutte le **informazioni condivise** del sistema.

3.1.1 Modello E/R



3.1.2 Modello Relazionale



3.1.3 DBMS

Per la **gestione della persistenza remota** è stato scelto **SQLite** come **Database Management System** (DBMS). Questa scelta si è rivelata particolarmente adatta per il progetto in quanto:

- Semplicità di utilizzo:** SQLite è un DBMS leggero e server-less, consentendo un'integrazione rapida e diretta con applicazioni basate su Node.js ed Express.
- Assenza di configurazione:** non richiede l'installazione o la gestione di un server dedicato, semplificando notevolmente la fase di deployment, specialmente in ambienti di sviluppo o prototipazione.
- Performance adeguate:** per applicazioni di piccola e media scala con accessi concorrenti limitati (come nel caso di una gestione ristorativa centralizzata), SQLite offre performance più che sufficienti.
- Ampio supporto:** è ampiamente supportato da librerie TypeScript, garantendo compatibilità con l'intero stack tecnologico utilizzato.

In contesti più avanzati o con requisiti di scalabilità elevata, sarà eventualmente possibile **sostituire SQLite con un DBMS relazionale più strutturato** (ad esempio MySQL o MariaDB) senza modificare in modo sostanziale la logica applicativa.

3.2 Persistenza locale: Ionic storage

La **persistenza locale** è limitata al dispositivo dell'utente e viene utilizzata esclusivamente per **memorizzare informazioni relative alla sessione** in corso, al fine di garantire una navigazione fluida e contestualizzata all'interno dell'applicazione.

I dati salvati localmente, servono quindi a **mantenere lo stato dell'utente autenticato**, anche in caso di aggiornamento o riavvio dell'interfaccia.

Nel sistema, lo **storage locale** è gestito nel file `core/services/authentication.service.ts` tramite una serie di oggetti `BehaviorSubject`, che permettono di propagare le modifiche ai componenti sottoscritti in tempo reale.

I dati memorizzati sono i seguenti:

- **id Utente**: Id dell’utente, oppure la matricola nel caso dell’impiegato.
- **Token di accesso**: utilizzato per autenticare le richieste al backend.
- **Ruolo dell’utente**: necessario per abilitare le funzionalità specifiche del ruolo (cliente, cameriere, chef, amministratore).
- **Username**: visualizzato nell’interfaccia utente.
- **Avatar**: immagine profilo dell’utente, eventualmente personalizzabile.
- **idFiliale**: Id della filiale (se disponibile) dell’impiegato.

Il sistema di persistenza locale, permette di **migliorare l’esperienza utente** attraverso un’**interfaccia più reattiva e immediata**, mantenendo lo stato dell’utente tra le varie schermate dell’app.

3.3 Creazione e testing del DB

La fase di creazione e testing del database viene realizzata tramite un sistema modulare composto da:

- File di **migration**, per la definizione e creazione delle tabelle.
- File di **seeding**, per l’inserimento di dati di esempio utili al testing e allo sviluppo.
- File **migrations.ts**: Uno script centralizzato per avviare la procedura completa.

3.3.1 Migrations

Le migrations gestiscono in modo strutturato lo schema del database. Ogni file definisce la logica per creare ed eliminare una specifica tabella, mantenendo il sistema coerente e aggiornabile anche in caso di modifiche future.

Tutti i file di migration del progetto si trovano in: `/Backend/Database/Migrations`.

Ogni migration segue uno schema chiaro e ripetibile, esportando due funzioni:

- `createIfDoesntExists()`: crea la tabella usando ”CREATE TABLE IF NOT EXISTS”;
- `dropTable()`: per rimuovere la tabella se necessario.

3.3.2 Seeders

I seeders sono strumenti fondamentali per il **popolamento controllato del database** con dati di test realistici e coerenti. Sono particolarmente utili durante lo sviluppo per simulare casi d’uso concreti, verificare le funzionalità applicative o collaudare l’interfaccia utente.

Ogni seeder è specializzato in una determinata entità (clienti, prodotti, ecc.) e può sfruttare generatori di dati casuali come `@faker-js/faker` per creare record variabili e plausibili, anche corredati da immagini e descrizioni.

Questa fase consente di verificare il comportamento del sistema in scenari realistici, migliorando l’affidabilità e riducendo i rischi di malfunzionamenti in produzione.

Tutti i file di seeding si trovano in `/Backend/Database/Seeders`.

3.3.3 File di migrations

Il file `/Backend/migrations.ts` rappresenta lo **script centrale** per l’inizializzazione del database. In base ai parametri forniti all’avvio, può eseguire diverse azioni:

- **Creazione** di tutte le tabelle del database;
- **Drop** completo delle tabelle esistenti;
- **Esecuzione dei seeder** per popolare il sistema con dati di esempio;
- **Reinizializzazione completa** del sistema in un ambiente di test o sviluppo.

Questo approccio centralizzato garantisce ripetibilità e controllo completo sullo stato del database, rendendo il processo di deploy e testing affidabile e coerente.

Per avviare il processo di reinizializzazione completo, è possibile utilizzare il comando `npm run migrate` all’interno della cartella Backend.

4 Rotte

4.1 Gestione delle rotte

L'applicazione adotta un'organizzazione modulare delle rotte, sia lato frontend che backend, al fine di garantire una struttura chiara, scalabile e facilmente manutenibile.

In particolare, le rotte backend sono raccolte nella cartella Routes, dove ogni file è dedicato a un ambito funzionale specifico del sistema.

Questa segmentazione consente di isolare le responsabilità, migliorando al contempo la leggibilità del codice,

L'applicazione implementa un **sistema di protezione delle rotte** sia lato **backend** (tramite middleware Express) che lato **frontend** (tramite guardie e interceptor Angular). Questo garantisce che solo gli utenti correttamente autenticati e autorizzati possano accedere alle risorse sensibili del sistema.

4.1.1 Middleware (Backend)

4.1.1.1 Autenticazione

Il middleware `Middleware/authMiddleware.ts` verifica la presenza e validità del bearer token, implementato con JWT (JSON Web Token), incluso nella richiesta. In particolare:

- Se il token è valido, viene determinato se l'utente è un cliente o un Impiegato, e le relative informazioni vengono iniettate nell'oggetto `req.user`.
- In caso contrario, la richiesta viene respinta con errore `401 Unauthorized`.

4.1.1.2 Ruoli

Il middleware `Middleware/roleMiddleware.ts` verifica che il ruolo dell'utente sia incluso tra quelli autorizzati per la specifica rotta. Qualora non lo fosse ritorna un errore `403 Forbidden` se il ruolo non è incluso tra quelli autorizzati.

4.1.2 Guardie (Frontend)

Nel **frontend Angular**, la protezione delle rotte è implementata tramite la guardia `/core/guards/authGuard`, che utilizza `CanActivateFn`. La guardia:

- Verifica la presenza di un token valido tramite la service `core/services/AuthenticationService`.
- Controlla che il ruolo dell'utente sia incluso tra quelli ammessi per la rotta.
- In caso contrario, reindirizza automaticamente alla pagina di login, eseguendo il logout se il token è scaduto.

4.1.3 Interceptor (Frontend)

Il frontend utilizza l'interceptor `core/interceptors/AuthInterceptor` per:

- Iniettare automaticamente il bearer token nell'header `Authorization` delle richieste HTTP.
- Intercettare eventuali errori 401 o 403 e, in tal caso, eseguire il logout e redirigere l'utente alla login.

Questo meccanismo garantisce una **protezione centralizzata delle richieste** e una migliore gestione della sessione utente.

4.2 Rotte utilizzate

| asporto | | |
|---------|-------------|---------------------|
| POST | /addAsporto | Aggiunge un asporto |

| auth | | |
|------|-------------------------|--------------------|
| POST | /auth/login | Endpoint di login |
| GET | /auth/logout | Endpoint di logout |
| POST | /auth/recupera_password | Recupero password |

| cliente | | |
|---------|-------------------------|---------------------------------------|
| POST | /cliente/register | Registrazione di un nuovo cliente |
| GET | /cliente/points | Recupera punti cliente |
| PUT | /cliente/updateDati | Aggiorna i dati personali del cliente |
| PUT | /cliente/nuova_password | Aggiorna la password del cliente |
| PUT | /cliente/nuova_email | Aggiorna l'email del cliente |
| GET | /cliente/prenotazioni | Recupera le prenotazioni del cliente |

filiale

^

GET /filiale Recupera tutte le filiali

▼

POST /filiale/addFiliale Aggiungi una nuova filiale

▼

PUT /filiale/updateFiliale/{id_filiale} Aggiorna una filiale esistente

▼

DELETE /filiale/deleteFiliale/{id_filiale} Elimina una filiale

▼

POST /filiale/addImpiegato Aggiungi un impiegato alla filiale

▼

PUT /filiale/updateImpiegato/{matricola} Aggiorna un impiegato

▼

DELETE /filiale/deleteImpiegato/{matricola} Elimina un impiegato

▼

GET /filiale/{id_filiale}/impiegati Recupera tutti gli impiegati di una filiale

▼

PUT /filiale/impiegato/{nuova_password} Aggiorna la password di un impiegato

▼

menu

^

GET /menu Recupera tutti i prodotti

▼

POST /menu/addProdotto Aggiungi un nuovo prodotto

▼

PUT /menu/updateProdotto/{id_prodotto} Aggiorna un prodotto esistente

▼

DELETE /menu/deleteProdotto/{id_prodotto} Elimina un prodotto

▼

GET /menu/piattoDelGiorno Recupera il piatto del giorno

▼

PUT /menu/changePiattoDelGiorno/{id_prodotto} Aggiorna il piatto del giorno

▼

pagamenti

^

GET /pagamenti/{year} Ottiene i pagamenti per anno

▼

prenotazione

| | | | |
|--------|--|---|---|
| GET | /prenotazione | Ottieni tutte le prenotazioni | ▼ |
| GET | /prenotazione/{id_prenotazione} | Ottieni una prenotazione per ID | ▼ |
| GET | /prenotazione/cameriere/{id_cliente} | Ottieni le prenotazioni di un cliente (cameriere/admin) | ▼ |
| GET | /prenotazione/filiale/prenotazioni | Ottieni le prenotazioni del giorno per filiale | ▼ |
| GET | /prenotazione/filiale/tavoli-in-uso | Ottieni tavoli attualmente in uso | ▼ |
| POST | /prenotazione/check-otp | Verifica OTP | ▼ |
| GET | /prenotazione/{id_prenotazione}/cameriere/stato | Ottieni stato prenotazione per cameriere | ▼ |
| GET | /prenotazione/{id_prenotazione}/chef/stato | Ottieni stato prenotazione per chef | ▼ |
| POST | /prenotazione/prenota | Crea una prenotazione | ▼ |
| POST | /prenotazione/prenotaLoco | Prenota in loco | ▼ |
| PUT | /prenotazione/modificaPrenotazione | Modifica una prenotazione | ▼ |
| DELETE | /prenotazione/eliminaPrenotazione/{id_prenotazione} | Elimina una prenotazione | ▼ |
| POST | /prenotazione/conferma | Conferma una prenotazione | ▼ |
| POST | /prenotazione/addOrdine | Aggiungi un ordine | ▼ |
| POST | /prenotazione/ordine/pay | Aggiungi un pagamento all'ordine | ▼ |
| GET | /prenotazione/ordine/{idOrdine}/totale | Calcola il totale di un ordine | ▼ |
| GET | /prenotazione/{id_prenotazione}/ordini/{username} | Ottieni ordine per prenotazione e username | ▼ |
| POST | /prenotazione/ordine/addProdotti | Aggiungi prodotti a un ordine | ▼ |
| GET | /prenotazione/ordine/{id_ordine}/prodotti/ | Ottieni prodotti per prenotazione | ▼ |
| GET | /prenotazione/{id_prenotazione}/prodotti | Ottieni prodotti per prenotazione | ▼ |
| PUT | /prenotazione/ordine/prodotto/{id_ordprod}/cambiaStato | Cambia stato prodotto ordine | ▼ |
| PUT | /prenotazione/ordine/prodotto/{id_ordprod}/isRomana | Aggiorna stato "Romana" per un prodotto | ▼ |

torrette

^

GET /torrette/{id_torretta} Ottiene la torretta per ID

▼

GET /{id_torretta}/otp Ottiene OTP per torretta e data

▼

5 Funzionalità

5.1 Introduzione alle funzionalità offerte dal sistema e note generali

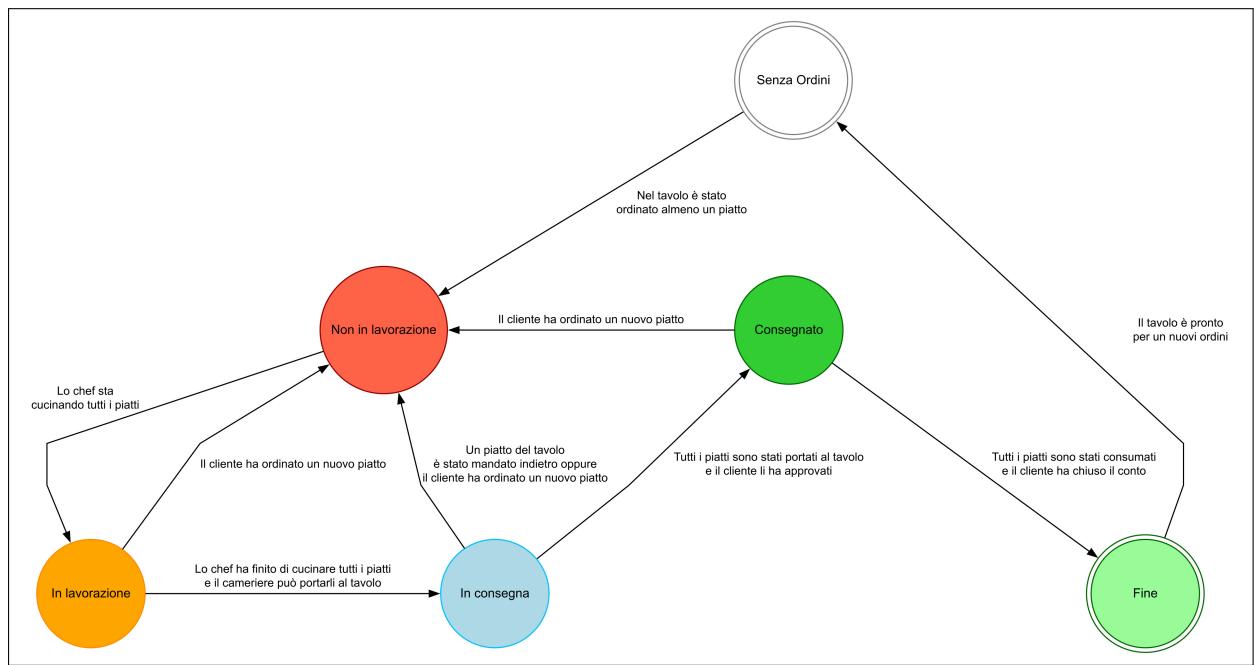
Il sistema offre numerose funzionalità relative alla gestione del servizio di ristorazione della catena di ristoranti Ciscusassi. Quando si fa riferimento a una funzionalità, con particolare riferimento al componente Angular/Ionic, viene indicato un *path* che al suo interno contiene quattro file relativi al componente pagina o al componente Ionic personalizzato:

- ***.ts**: contiene la logica del componente, inclusi metodi, proprietà, eventi e interazioni con servizi.
- ***.html**: definisce la struttura e il contenuto visivo del componente (template).
- ***.css**: gestisce lo stile del componente (colori, layout, spaziature, ecc.).
- ***.spec.ts**: contiene i test automatici per verificare il corretto funzionamento del componente.

In qualsiasi circostanza, se il server è irraggiungibile o non riesce a soddisfare la richiesta, le pagine che effettuano richieste al backend subiscono una modifica dell'interfaccia e un mostrano un messaggio di errore che chiede all'utente di riprovare successivamente.

Ciscusassi è chiuso il martedì, le fasce di apertura sono 11:50-16:30 e 19:20-00:00, mentre i turni di prenotazione sono 12:00-13:30, 13:30-15:00, 19:30-21:00, 21:00-22:30.

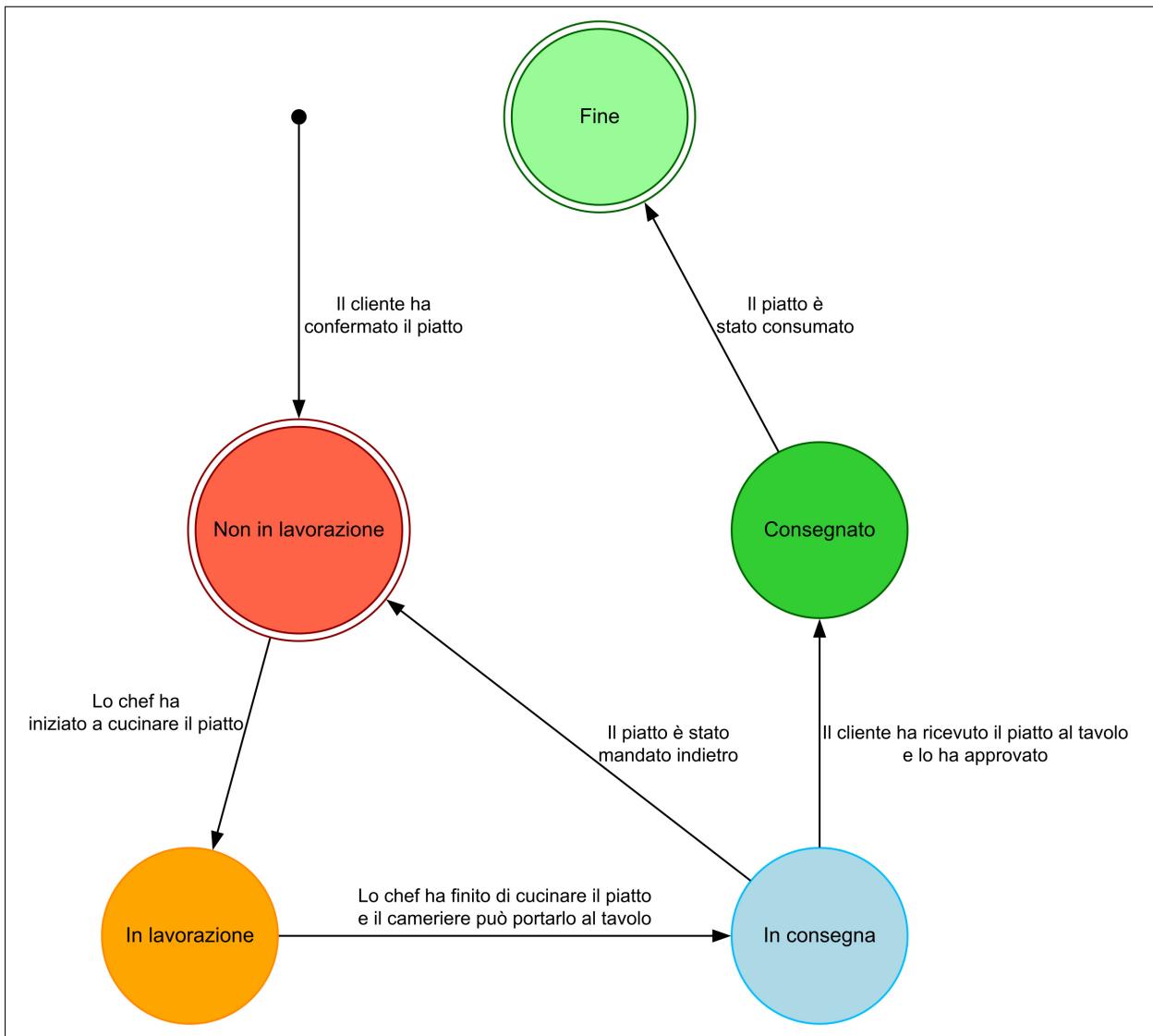
5.2 Diagramma degli stati



Il diagramma rappresenta il ciclo di vita degli ordini associati a un tavolo nel ristorante, focalizzandosi sul punto di vista del tavolo del ristorante. Si parte dallo stato iniziale "Senza Ordini", che indica un tavolo libero, su cui non è stato ancora effettuato alcun ordine. Appena il cliente effettua una richiesta, il sistema transita nello stato "Non in lavorazione", che rappresenta una situazione in cui l'ordine è stato registrato ma la cucina non ha ancora iniziato a preparare i piatti. Quando lo chef inizia a cucinare, il tavolo passa nello stato "In lavorazione", e vi rimane fino al completamento della preparazione. Una volta pronti, i piatti possono essere serviti e lo stato diventa "In consegna". Questo momento è delicato, poiché da qui sono possibili più evoluzioni: se i piatti vengono accettati dal cliente, il sistema progredisce verso lo stato "Consegnato". Tuttavia, se qualcosa va storto (ad esempio un piatto viene mandato indietro o viene ordinato qualcosa in più), si torna allo stato "Non in lavorazione", riprendendo così il ciclo. Quando tutti i piatti sono stati consegnati e approvati, il tavolo rimane nello stato "Consegnato" finché il cliente non termina il pasto e chiude il conto. A quel punto si raggiunge lo stato "Fine", che rappresenta la conclusione del servizio per quel tavolo. Da questo stato si può ritornare a "Senza Ordini", nel caso il

tavolo venga liberato e reso disponibile per nuovi clienti.

Il diagramma modella perfettamente il caso della prenotazione in loco, ovvero quella che il cameriere effettua per un cliente già registrato sulla piattaforma quando si presenta fisicamente al ristorante senza aver effettuato alcuna prenotazione anticipata. Nel caso in cui il cliente effettui invece una prenotazione, la modellazione sarebbe leggermente diversa, in quanto è previsto un nuovo stato, denominato "Attesa Arrivo", che rappresenta la fase di attesa tra la prenotazione e l'effettivo arrivo del cliente al ristorante. Quando il cliente arriva, il cameriere ne conferma la presenza, provocando la transizione dello stato del tavolo da "Attesa Arrivo" a "Senza Ordini". Da questo punto in poi, la modellazione prosegue invariata secondo lo schema già definito.

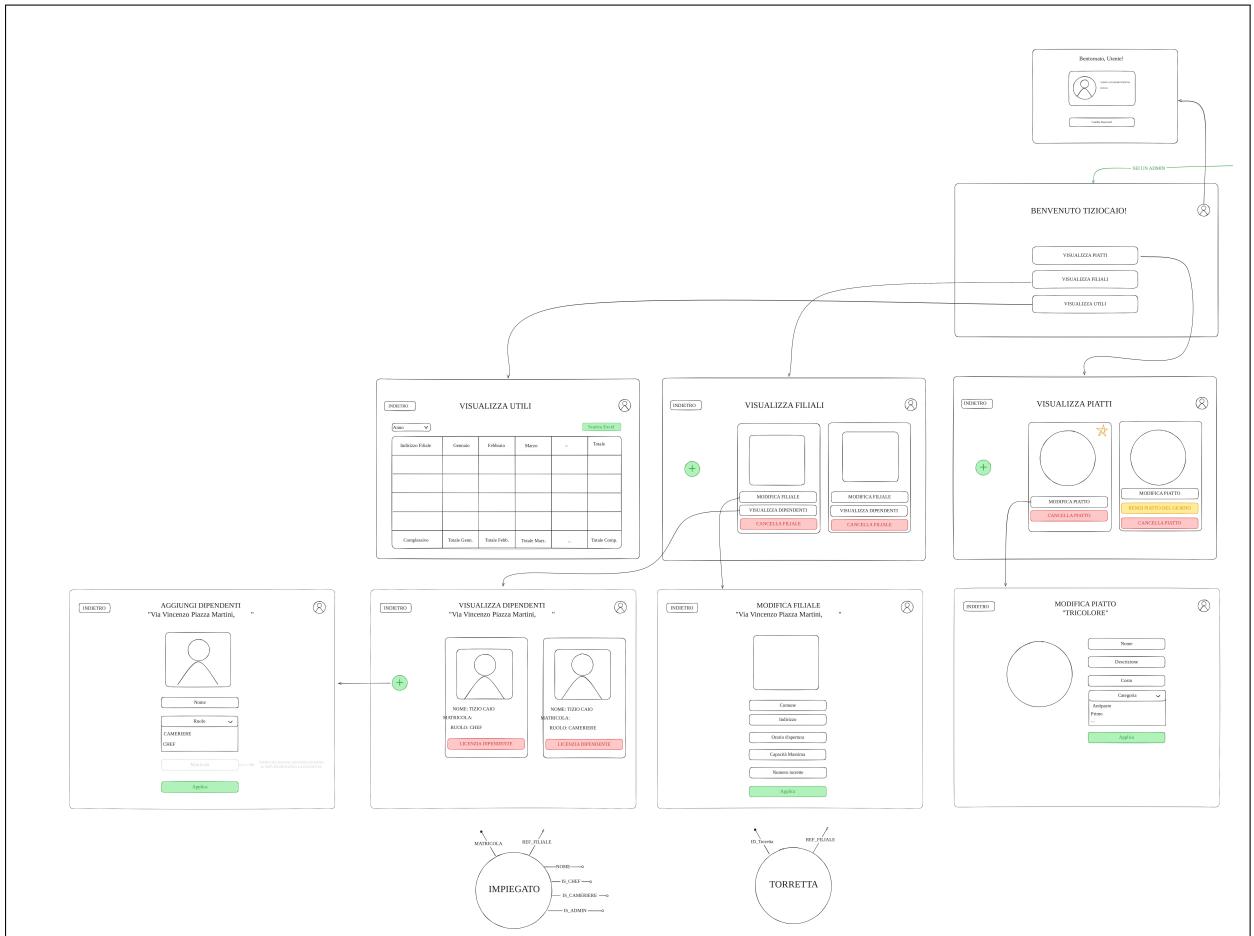


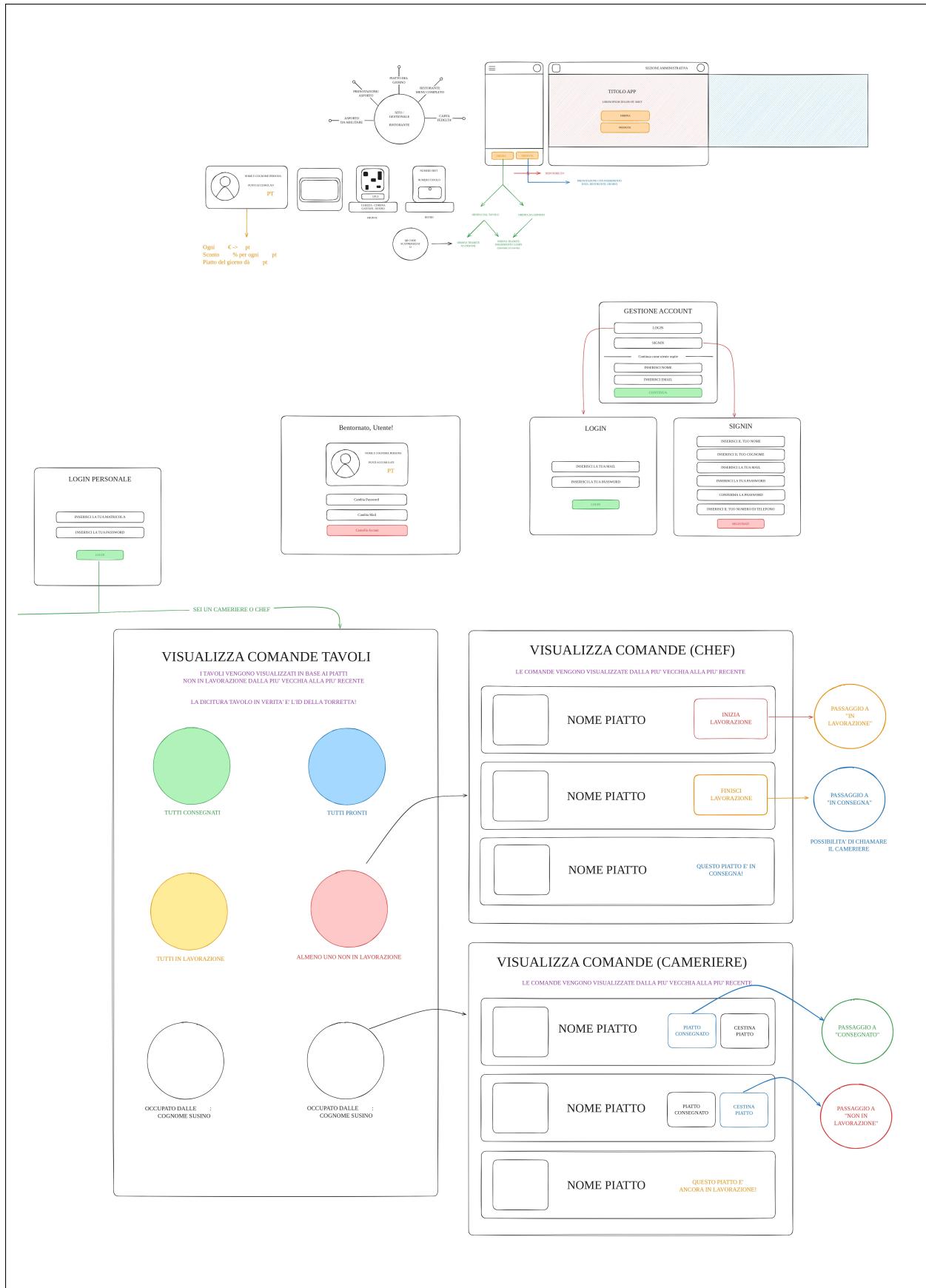
Il diagramma degli stati descrive in dettaglio il ciclo di vita di un singolo piatto all'interno di un ordine, dal momento in cui viene confermato dal cliente fino al suo eventuale consumo. Rispetto al diagramma precedente, che si concentrava sul tavolo come unità principale, qui l'attenzione è rivolta al singolo elemento ordinato, modellando il percorso operativo e decisionale che lo accompagna. Il ciclo ha inizio nello stato "Non in lavorazione", che rappresenta un piatto ordinato ma non ancora preso in carico dalla cucina. Quando lo chef inizia a cucinarlo, si passa allo stato "In lavorazione", il quale permane fino a quando la preparazione è completata. A quel punto, il piatto viene affidato al cameriere e si entra nello stato "In consegna". Questo è un momento cruciale, perché rappresenta l'interazione diretta tra il cliente e il piatto: se il cliente lo riceve e lo approva, allora il sistema transita nello stato "Consegnato". Tuttavia, se qualcosa non va, il piatto può essere "mandato indietro", e in tal caso ritorna allo stato "Non in lavorazione", riavviando così una parte del ciclo. Una volta che il piatto è stato approvato e consegnato, non c'è più nulla da fare, quindi si arriva alla fine.

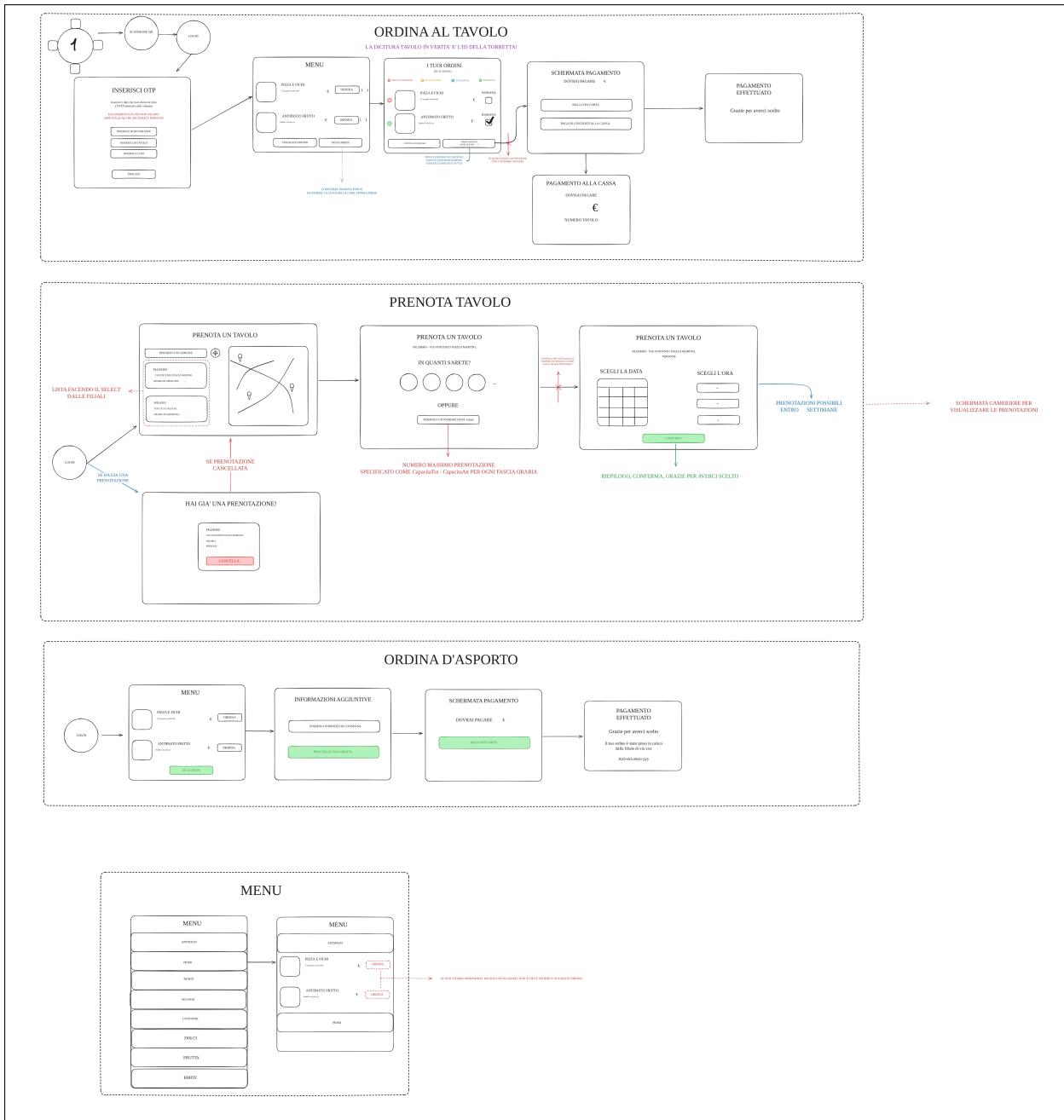
nato, il passaggio allo stato finale "Fine" avviene nel momento in cui il cliente lo consuma completamente.

Il diagramma modella in modo efficace sia il caso della prenotazione in loco sia quello della prenotazione anticipata, tenendo conto anche di eventuali situazioni impreviste, come il rifiuto del piatto da parte del cliente, e proponendo una gestione coerente per ciascuna di esse.

5.3 Flusso creativo pre-mockup





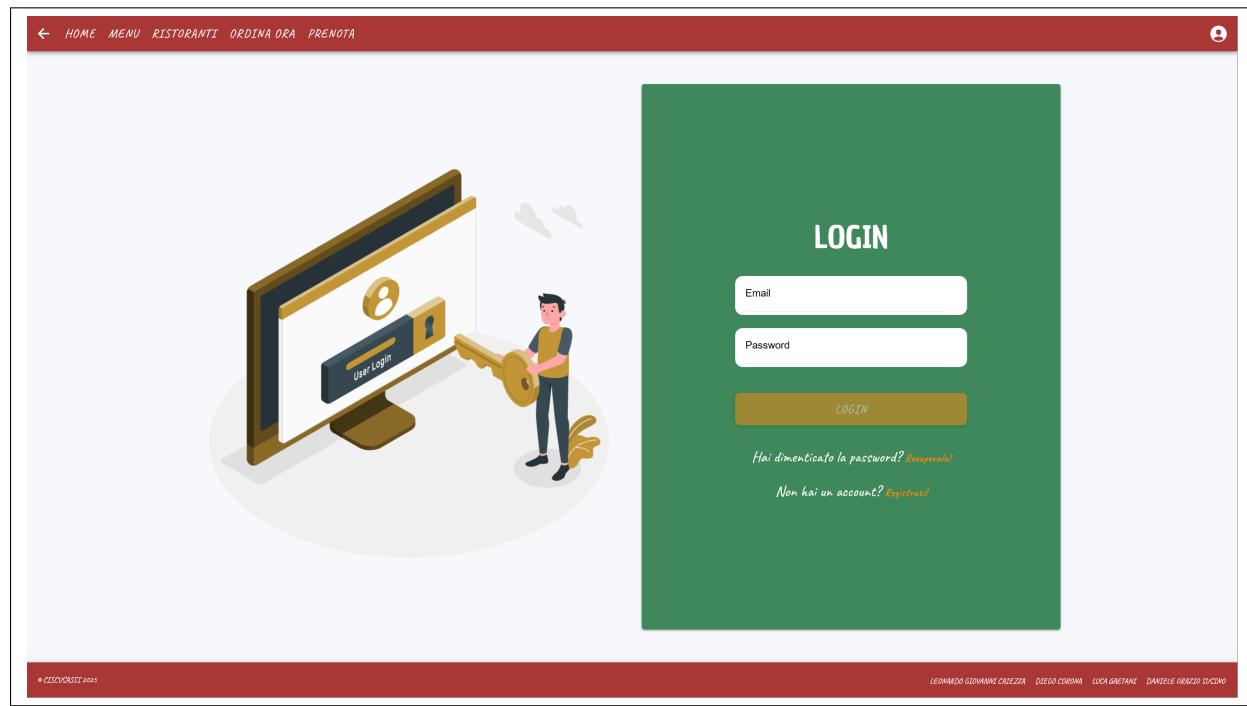


5.4 Gestione Account

5.4.1 Login

La funzionalità Login si presenta in pages/account/login e consente all'utente di accedere al sistema attraverso l'inserimento delle proprie credenziali, ovvero una coppia formata da email e password. Una volta aperta la schermata di login, l'utente si trova di fronte a un'interfaccia chiara e centrata, realizzata con i componenti offerti da Ionic come ion-grid, ion-card, ion-input e ion-button. La pagina è costruita nel file login.page.html e il relativo comportamento è gestito nel componente LoginPage attraverso l'uso del modulo ReactiveFormsModule, che permette di strutturare il form in modo dinamico e sicuro. Il modulo formLogin, istanziato come FormGroup, contiene due controlli: uno per l'email e uno per la password. L'email è validata attraverso la funzione Validators.pattern tramite una regex personalizzata, allo stesso modo, anche la password è validata tramite una espressione regolare che impone la presenza di almeno 6 caratteri, una lettera maiuscola, una minuscola, un numero e un carattere speciale. Se i dati non rispettano questi criteri, il bottone di login viene disattivato, impedendo l'invio del form.

Nel momento in cui l'utente preme il pulsante di login, viene attivato il metodo onSubmit(). Se il form è valido, i dati vengono inviati al servizio di autenticazione, il quale comunica con il back-end attraverso l'API di login. In caso di risposta positiva da parte del server, il metodo handleResponse() estrae il token JWT restituito e lo decodifica per ottenere informazioni come l'id utente, il ruolo, lo username, l'id della filiale e altri dati necessari alla sessione. Questi dati vengono quindi salvati local storage di ionic attraverso apposite funzioni asincrone, e l'utente viene reindirizzato alla homepage dell'applicazione. Qualora si verificasse un errore nella risposta, oppure ci fossero problemi nel salvataggio dei dati, viene mostrato un messaggio di errore specifico e il form rimane attivo per permettere un nuovo tentativo di login. L'interfaccia mostra inoltre un link che consente all'utente di recuperare la password nel caso l'avesse dimenticata, indirizzandolo alla pagina apposita, e un altro link che consente all'utente di registrarsi, reindirizzandolo alla pagina apposita. L'intera gestione è pensata per garantire sicurezza, chiarezza visiva e compatibilità con i tipi richiesti dalle API del sistema.



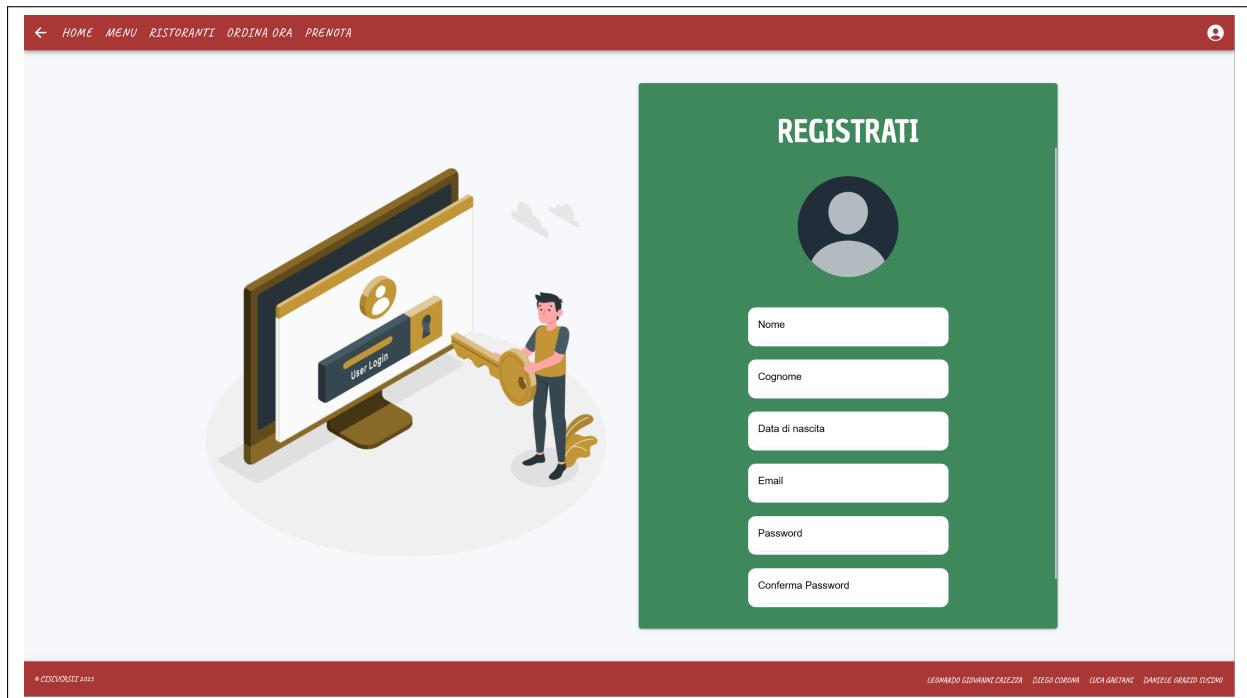
5.4.2 Registrazione

La funzionalità Registrazione si presenta in pages/account/signin. L'interfaccia, realizzata con i componenti Ionic (ion-grid, ion-card, ion-input, ion-avatar, ion-button). L'utente inserisce nome, cognome, data di nascita, email, password e la conferma della password in un form reattivo denominato formRegistrazione, gestito tramite FormBuilder e validato con controlli obbligatori. Il campo data di nascita presenta un validatore personalizzato validDateOfBirthValidator(), che verifica che la data di nascita non

sia futura e che sia successiva all'anno 1900, l'email è soggetta a una validazione tramite espressione regolare, la password è soggetta a una validazione avanzata tramite espressione regolare che impone almeno una maiuscola, una minuscola, un numero, un carattere speciale e una lunghezza minima di sei caratteri. Le due password vengono confrontate attraverso un validatore personalizzato `matchPassword()` per assicurare che coincidano. Il form disabilita il pulsante di invio finché non è valido.

Durante la registrazione, l'utente può caricare una propria immagine cliccando sull'ion-avatar. In assenza di immagine, viene caricata automaticamente una di default convertita in Base64 tramite `FileReader`. Alla pressione del pulsante “Registrati”, viene invocato `onSubmit()`, che verifica la validità del form, popola l'oggetto credenziali e chiama il metodo `registrati()` del servizio `AuthenticationService`. In caso di risposta positiva, viene mostrato un messaggio di successo tramite `ToastController` e l'utente viene reindirizzato al login. In caso di errore, viene visualizzato un messaggio specifico.

Il componente pagina gestisce anche lo stato di caricamento, la visualizzazione degli errori e il caricamento asincrono dell'immagine. Il form viene costruito al momento dell'inizializzazione del componente (`ngOnInit()`), sfruttando `ReactiveFormsModule` per garantire sicurezza, modularità e facilità di manutenzione.

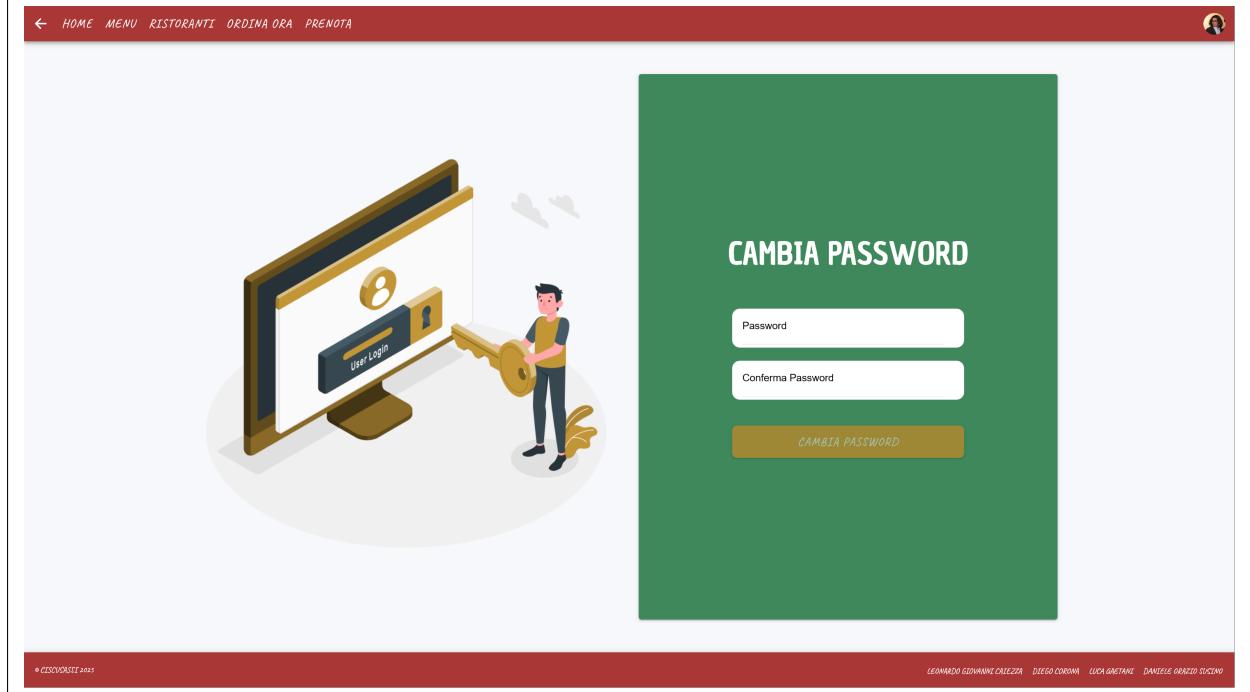


5.4.3 Cambio Password

La funzionalità Cambio Password si presenta in `pages/account/cambia-password` e consente all'utente di avviare il processo di recupero credenziali in caso di smarrimento. L'interfaccia è costruita con i componenti standard di Ionic come `ion-grid`, `ion-card`, `ion-input` e `ion-button`, disposti in modo centrale e responsivo all'interno di una griglia. All'interno della card compare il titolo “RECUPERA PASSWORD”, accompagnato da un campo input che consente di inserire l'indirizzo email associato al proprio account. Il modulo è costruito tramite `ReactiveFormsModule` e contiene un solo campo `email`, validato sia come obbligatorio che nel formato, tramite `Validators.required` e `Validators.pattern`, utilizzando una espressione regolare personalizzata. Il bottone per l'invio è disabilitato fino a quando il campo non è correttamente compilato. Durante l'invio, un `ion-spinner` segnala il caricamento.

Il form, creato tramite `ReactiveFormsModule`, è definito da un `FormGroup` che include due controlli: `password` e `confermaPassword`. Il campo `password` è sottoposto a una validazione avanzata tramite espressione regolare, che impone la presenza di almeno una lettera maiuscola, una minuscola, un numero, un carattere speciale e una lunghezza minima di sei caratteri. Inoltre, viene applicato un validatore personalizzato `matchPassword()`, che verifica la corrispondenza tra i due campi. Solo quando entrambe le condizioni sono soddisfatte, il bottone per il cambio password diventa attivo. Quando l'utente conferma l'operazione premendo il pulsante, viene eseguito il metodo `onSubmit()`, che controlla la validità del

form e, se tutto è corretto, chiama il metodo `authenticationService.cambiaPassword()` passando le due password. La risposta del server viene gestita da `handleResponse()`: se l'operazione va a buon fine, viene mostrato un messaggio di conferma tramite un toast e l'utente viene reindirizzato alla pagina di gestione dati account. In caso contrario, viene visualizzato un messaggio di errore e il form rimane disponibile per tentativi successivi. L'intero flusso è pensato per massimizzare la sicurezza e l'usabilità, con un'attenzione particolare alla protezione delle credenziali e alla chiarezza dei feedback forniti all'utente.

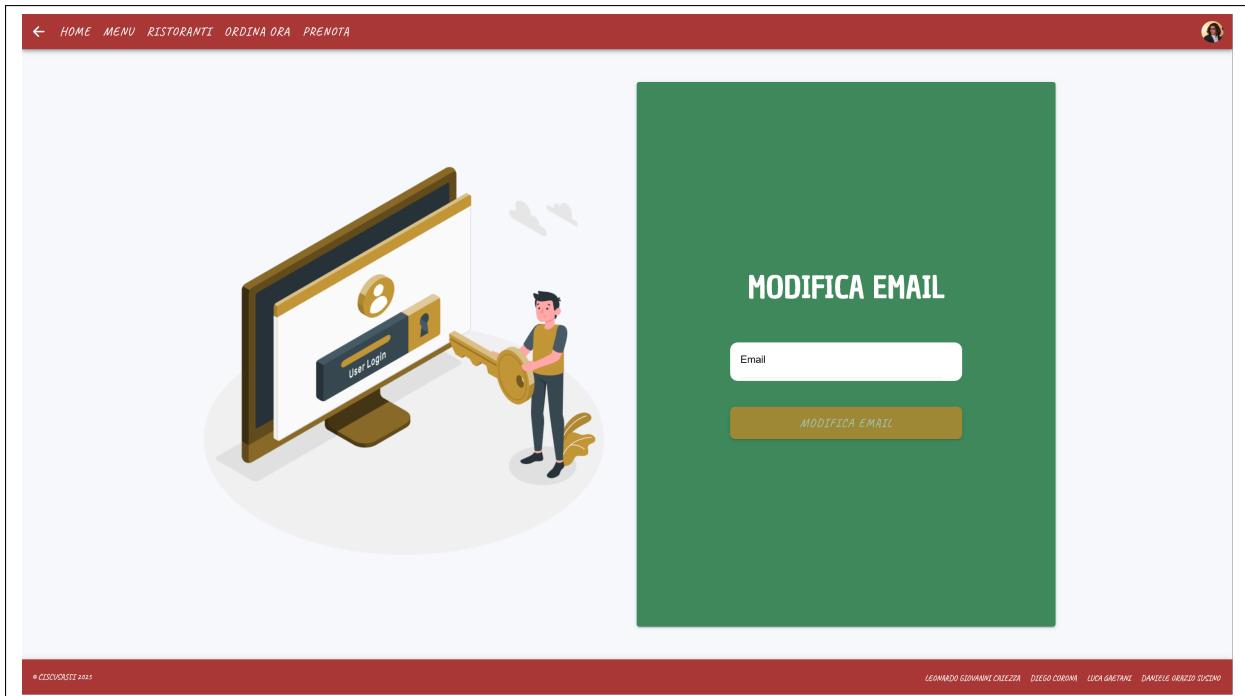


5.4.4 Cambio Email

La funzionalità Cambio Email si presenta in `pages/account/cambia-email` e consente all'utente di aggiornare l'indirizzo email associato al proprio profilo. L'interfaccia si presenta coerente con il resto dell'applicazione, adottando una struttura centrata e responsiva basata su ion-grid, con l'aggiunta di un'illustrazione decorativa posizionata a sinistra per i dispositivi con schermi ampi. La parte destra dello schermo contiene una ion-card con sfondo verde chiaro, che ospita un modulo semplice e chiaro con un solo campo input per l'inserimento della nuova email. L'etichetta "MODIFICA EMAIL" è mostrata in alto, e l'intero layout sfrutta i componenti di Ionic come ion-input, ion-button e ion-spinner per offrire un'interazione intuitiva e accessibile.

Il comportamento dinamico della pagina è gestito all'interno del componente `CambiaEmailPage`, dove viene inizializzato un `FormGroup` tramite `FormBuilder`. Il modulo contiene un singolo controllo chiamato `email`, che è sottoposto a doppia validazione: è obbligatorio (`Validators.required`) e deve rispettare il formato di un indirizzo email valido, che viene validato tramite `Validators.pattern` e un'espressione regolare personalizzata. Quando il form è valido, l'utente può premere il pulsante "MODIFICA EMAIL", che attiva il metodo `onSubmit()`. Questo metodo imposta lo stato di caricamento, legge il valore della nuova email dal form e lo invia al servizio `authenticationService.cambiaEmail()`.

Il risultato della chiamata viene poi gestito dalla funzione `handleResponse()`, la quale analizza la risposta dell'API. In caso di successo, viene mostrato un messaggio toast in alto con conferma dell'avvenuta modifica e l'utente viene reindirizzato alla pagina dei dati account. Se la risposta invece segnala un errore, viene impostato uno stato di errore interno e mostrato un messaggio all'utente, ad esempio nel caso in cui l'email risulti già registrata o il formato non sia corretto. L'interazione si conclude sempre con la possibilità per l'utente di correggere i dati inseriti e riprovare. Questo flusso offre una modalità sicura ed efficace per aggiornare l'email, garantendo una buona esperienza utente grazie a feedback visivi immediati e controlli formali robusti.



5.4.5 Recupera Password

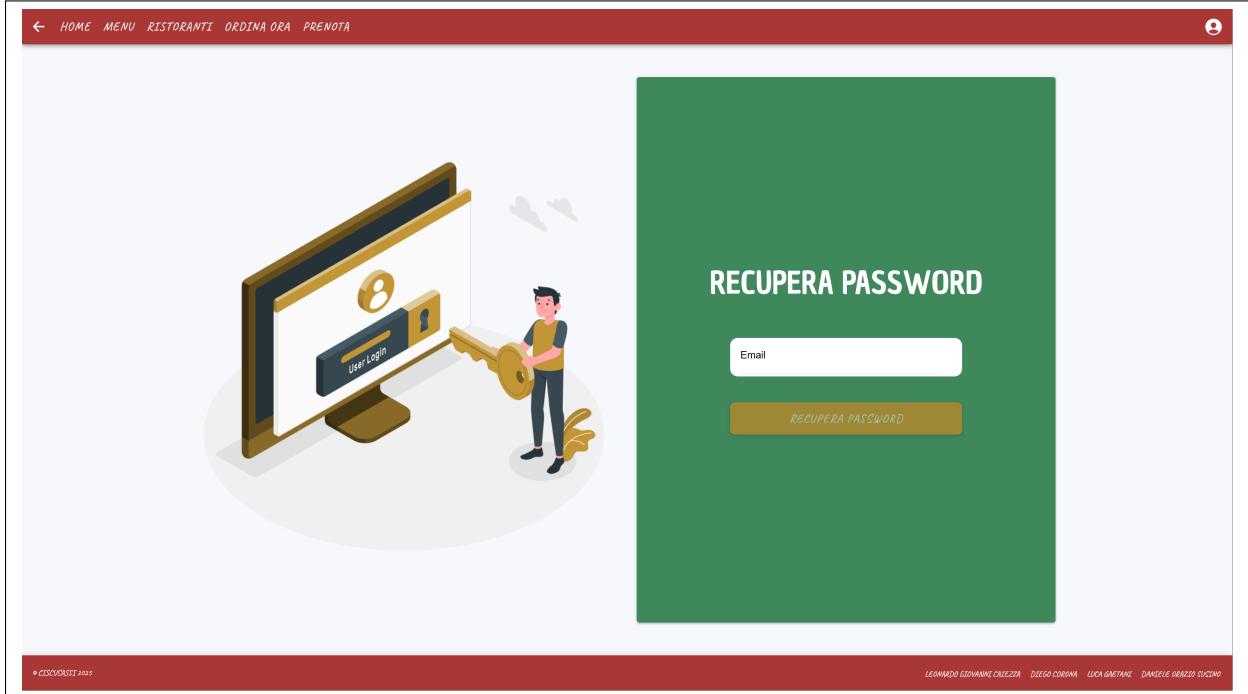
La funzionalità Recupera Password si presenta in pages/account/recupera-password, organizzata all'interno di una ion-grid che divide il contenuto in due colonne: una con un'immagine illustrativa visibile solo su schermi di grandi dimensioni, e una con una ion-card contenente il modulo per l'inserimento dell'email. Lo stile grafico utilizza colori personalizzati allineati con la paletta predefinita.

Sotto il titolo è presente un paragrafo condizionale che mostra eventuali messaggi di errore, legati alla validazione o a problemi di invio della richiesta. Il form utilizza Angular Reactive Forms e contiene un unico campo ion-input associato al controllo email, validato con il validator standard required e un validator pattern che sfrutta un'espressione regolare personalizzata per controllare il corretto formato dell'email. Il bottone di invio è disabilitato se il form è invalido o se è in corso una richiesta (loading). Durante il caricamento, il testo del bottone viene sostituito da uno spinner (ion-spinner). Alla sottomissione del modulo, viene invocato il metodo onSubmit.

Quando l'utente preme su “Recupera Password”, il metodo onSubmit() verifica la validità del form e, se tutto è corretto, chiama il servizio authenticationService.recuperaPassword() passando l'email inserita. La risposta del server viene gestita dal metodo handleResponse(), che nel caso di successo mostra un toast informativo all'utente, confermando che, se l'indirizzo esiste, riceverà una password temporanea. In caso di errore, viene mostrato un messaggio all'interno dell'interfaccia, e il form resta attivo per consentire nuovi tentativi. L'obiettivo principale di questa schermata è garantire un'interazione semplice, chiara e rassicurante per l'utente, riducendo la frustrazione tipica delle operazioni di recupero account e mantenendo la coerenza visiva e funzionale con il resto dell'applicazione. La funzionalità Cambia Password è disponibile tramite la pagina cambia-password, che consente all'utente di modificare la propria password, tipicamente in seguito a una procedura di recupero. La struttura dell'interfaccia ricalca quella delle altre schermate di autenticazione, con un layout centrato all'interno di una ion-grid che presenta una colonna con immagine illustrativa e una colonna con un modulo contenuto in una ion-card di colore verde. Il titolo della schermata, “CAMBIA PASSWORD”, è seguito da due campi input che raccolgono rispettivamente la nuova password e la conferma della stessa. Entrambi i campi sono protetti da ion-input-password-toggle, che consente all'utente di visualizzare o nascondere il contenuto digitato.

Nel componente TypeScript, viene inizializzato il form nel metodo ngViewWillEnter, richiamato da ngOnInit, assicurandosi che ogni volta che si entra nella vista, il form venga resettato e lo stato interno ripristinato. Il metodo onSubmit imposta lo stato di loading e, se il form è valido, invia la richiesta al servizio AuthenticationService, chiamando il metodo recuperaPassword. La risposta viene gestita da handleResponse, che mostra un messaggio di successo tramite ToastController e reindirizza l'utente alla

pagina di login se la richiesta è andata a buon fine. In caso contrario, viene mostrato un messaggio di errore. Se la chiamata al server fallisce per problemi di rete o formattazione errata dell'email, viene mostrato un messaggio generico di errore e lo stato error viene aggiornato. Infine, il metodo presentToast utilizza il ToastController di Ionic per mostrare una notifica in alto allo schermo con un messaggio personalizzato e un colore coerente con lo stato della risposta (successo, errore, o avviso).



5.4.6 Visualizza Schermata Account

La funzionalità Visualizza Schermata Account si presenta in pages/account/dati-account e consente all'utente autenticato di visualizzare e gestire le informazioni principali relative al proprio profilo. Una volta effettuato l'accesso, l'utente può accedere a questa pagina per consultare il proprio nome utente, l'avatar e – se previsto – il numero di punti accumulati. La pagina è costruita sfruttando componenti dell'interfaccia di Ionic, con una struttura chiara e responsive, che si adatta a diverse risoluzioni di schermo.

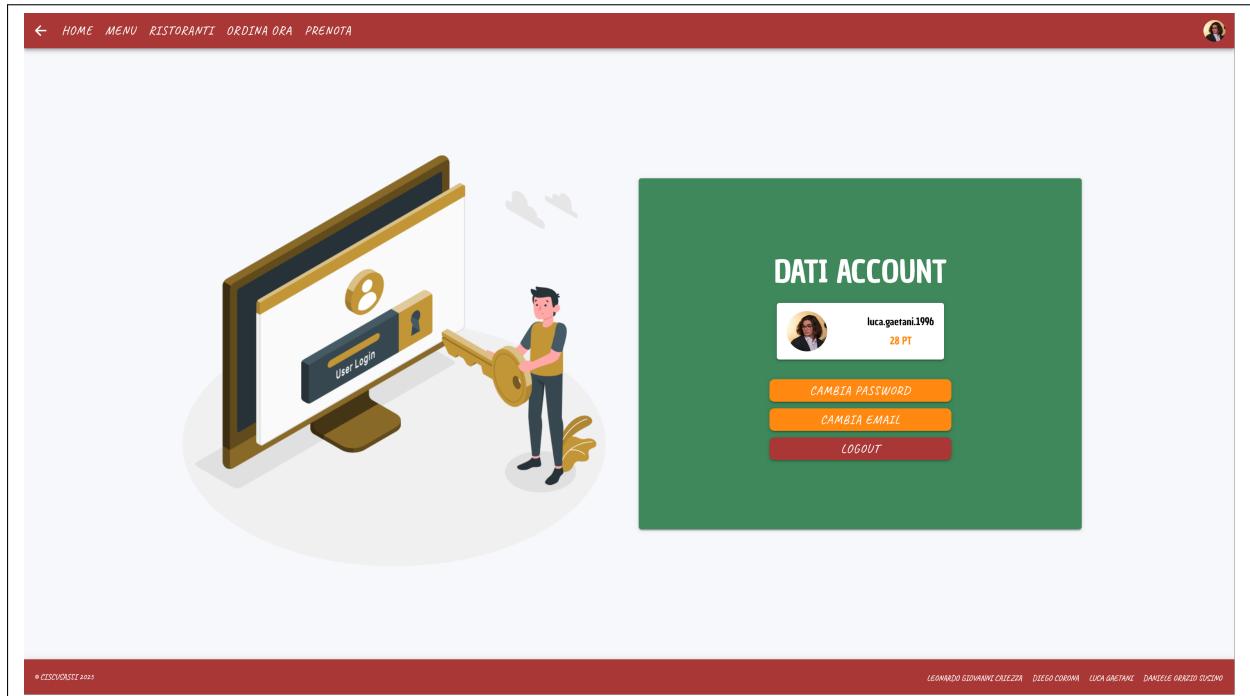
Nella parte sinistra della schermata viene mostrata un'immagine illustrativa visibile solo su dispositivi con schermo extra-large, mentre sulla destra è presente il contenuto principale racchiuso all'interno di una card. All'interno di questa card si trova l'intestazione “DATI ACCOUNT”, seguita da una sezione centrale che mostra l'avatar dell'utente e il relativo nome utente. Se il ruolo dell'utente è “cliente”, viene anche eseguita una chiamata asincrona per ottenere i punti accumulati, che vengono visualizzati accanto allo username non appena la richiesta viene completata. Durante il caricamento, compare uno spinner che segnala l'attesa della risposta.

Il caricamento dei dati è gestito all'interno del metodo ngOnInit(), dove vengono sottoscritti gli observable username\$, role\$ e avatar\$ esposti dal servizio di autenticazione. In questo modo, la pagina rimane sempre sincronizzata con le informazioni correnti dell'utente. Se il ruolo è “cliente”, viene effettuata anche una richiesta all'API per recuperare i punti, attraverso il metodo getPoints() dell'AuthenticationService. L'esito della chiamata viene gestito nel metodo handleResponse(), che aggiorna lo stato della pagina in base alla risposta ricevuta: se positiva, assegna i punti all'utente; se negativa, mostra un messaggio d'errore.

Oltre alla semplice visualizzazione, la pagina fornisce anche una serie di azioni che l'utente può eseguire tramite due pulsanti, “CAMBIA PASSWORD” e “CAMBIA EMAIL”, che reindirizzano a pagine dedicate dove l'utente può aggiornare le proprie credenziali. Infine, il pulsante “LOGOUT” permette all'utente di uscire dall'applicazione in modo sicuro. Quando viene premuto, viene mostrato uno spinner finché l'operazione non è completata. Se il logout va a buon fine, l'utente viene reindirizzato alla schermata di

login; in caso di errore, viene visualizzato un messaggio che invita a riprovare.

L'intera funzionalità è pensata per essere intuitiva, reattiva e accessibile, con una particolare attenzione alla gestione degli stati di errore e al feedback visivo per ogni operazione. Grazie all'integrazione tra Angular e Ionic, è possibile garantire un'esperienza utente coerente su tutte le piattaforme supportate.

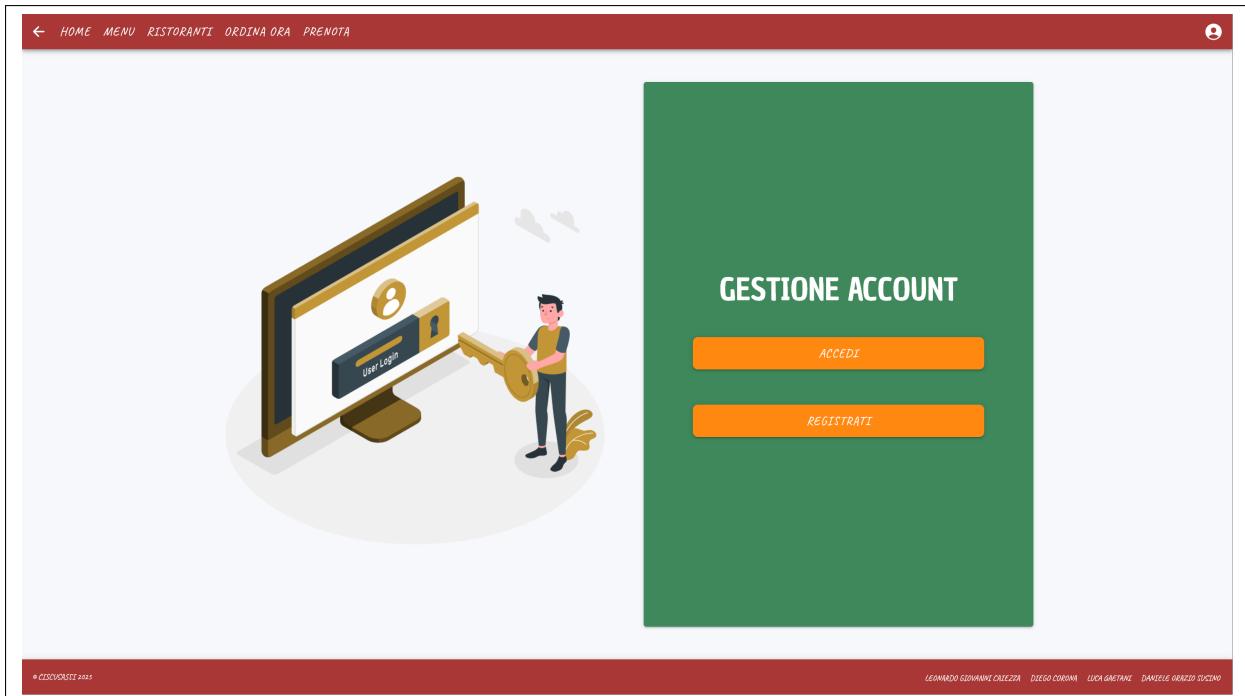


5.4.7 Visualizza Schermata Gestione Account

La funzionalità Visualizza Schermata Gestione Account si trova in pages/account/gestione-account e rappresenta la schermata iniziale da cui l'utente può scegliere la modalità di accesso al sistema. L'interfaccia, chiara e centrata, è costruita con i componenti offerti da Ionic, tra cui ion-grid, ion-row, ion-col, ion-card, ion-text, ion-img e ion-button, e si adatta in modo responsivo ai diversi formati di schermo.

La pagina è composta da una griglia a due colonne. Nella parte sinistra, visibile solo su dispositivi extra-large, è presente un'immagine illustrativa, mentre la colonna destra contiene una card con il titolo "GESTIONE ACCOUNT" e due pulsanti. Il primo pulsante consente all'utente di accedere al sistema tramite la pagina di login (/login). Il secondo, con la dicitura "REGISTRATI", indirizza alla pagina di registrazione (/signin).

Il file TypeScript (gestione-account.page.ts) non contiene logica applicativa, limitandosi all'inizializzazione del componente tramite il metodo ngOnInit(). La gestione dei pulsanti è interamente demandata al routing, definito tramite l'attributo [routerLink].



5.5 Gestione Amministrazione

5.5.1 Visualizza Schermata Amministrazione

La funzionalità Visualizza Schermata Amministrazione si presenta in pages/admin/amministrazione e consente all’utente con ruolo amministrativo di accedere a una schermata riepilogativa da cui può gestire le principali operazioni relative al sistema. L’accesso a questa pagina è riservato agli utenti autenticati con il ruolo di amministratore. Una volta autenticato, l’utente visualizza una schermata divisa in due sezioni principali. Nella parte sinistra viene mostrata un’immagine illustrativa, visibile esclusivamente su dispositivi con schermi extra-large, mentre nella parte destra è presente una card centrale che accoglie l’utente con il messaggio “Benvenuto [username]!”, dove il nome utente è recuperato dinamicamente dal servizio di autenticazione.

Il caricamento del nome utente è gestito all’interno del metodo ngOnInit() del componente AmministrazionePage, il quale sottoscrive l’observable username\$ esposto dall’AuthenticationService. In questo modo, il valore del nome utente viene aggiornato in tempo reale in base allo stato corrente della sessione. L’interfaccia, costruita con componenti Ionic come ion-grid, ion-card, ion-button, ion-text e ion-img, presenta una struttura responsive che garantisce una buona fruibilità su tutte le risoluzioni di schermo. All’interno della card sono disponibili tre pulsanti principali, ognuno dei quali permette l’accesso a una sezione specifica dell’area amministrativa:

- “Gestisci piatti”, che reindirizza alla pagina dedicata alla gestione del menù;
- “Gestisci filiali”, che permette di visualizzare e modificare le filiali attive;
- “Visualizza utili”, che conduce a una sezione dedicata alla consultazione dei dati economici.

Ciascun pulsante è implementato utilizzando il componente ion-button con attributo [routerLink], che consente una navigazione fluida tra le varie sezioni dell’applicazione. Lo stile è personalizzato con classi CSS e colori coerenti con la paletta del sistema per mantenere un’identità visiva chiara e uniforme. L’intera funzionalità è progettata per offrire un’interazione immediata e intuitiva all’utente amministratore, fornendo un punto di accesso rapido alle operazioni fondamentali del sistema.



5.5.2 Gestisci Piatti

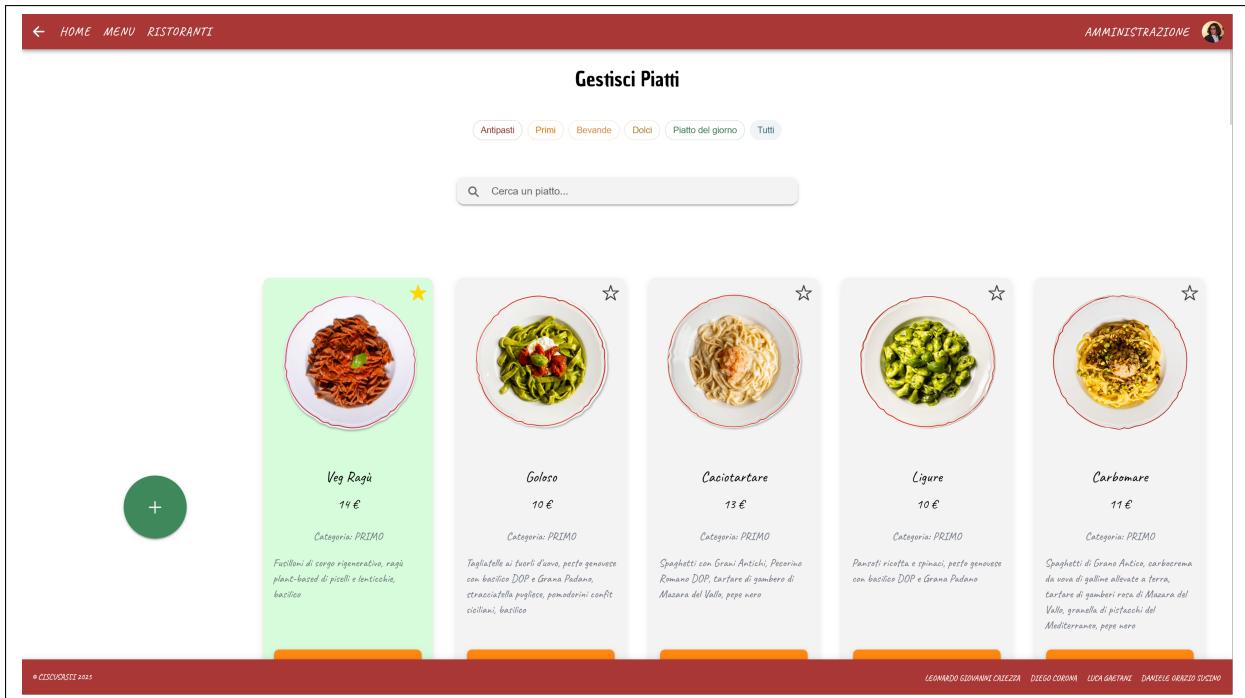
La funzionalità Gestisci Piatti si trova in pages/admin/gestisci-piatti e permette all’utente amministratore di gestire il catalogo dei piatti disponibili nel sistema. La pagina offre un’interfaccia intuitiva e responsiva, costruita utilizzando componenti Ionic come ion-content, ion-chip, ion-searchbar, ion-button, ion-spinner, ion-text e un componente personalizzato app-piatto-amministratore. All’apertura della pagina, il componente GestisciPiattiPage esegue, tramite il metodo ngOnInit(), il caricamento sia del piatto del giorno che della lista completa dei piatti dal backend, interfacciandosi con il servizio ProdottoService.

L’utente può filtrare i piatti in base a categorie predefinite (Antipasti, Primi, Bevande, Dolci, Piatto del giorno, Tutti) selezionando gli appositi pulsanti ion-chip. Inoltre, è possibile cercare un piatto specifico tramite la barra di ricerca ion-searchbar, che aggiorna dinamicamente la lista mostrata. Durante il caricamento dei dati viene mostrato uno spinner circolare, mentre in caso di errore viene visualizzato un messaggio specifico invitando l’utente a riprovare. Ogni piatto è rappresentato dal componente app-piatto-amministratore, che consente di visualizzare i dettagli e modificare eventuali proprietà. È possibile anche impostare un piatto come “Piatto del giorno” oppure eliminarlo. La cancellazione viene confermata tramite un alert modale che richiede la conferma dell’utente prima di procedere.

Le principali funzionalità offerte sono:

- Caricamento asincrono della lista completa dei piatti e del piatto del giorno dal backend.
- Filtraggio dinamico per categoria.
- Ricerca testuale con debounce per migliorare le performance.
- Impostazione del piatto del giorno con controllo per evitare rimozioni non consentite.
- Eliminazione di un piatto con conferma tramite alert.
- Gestione di stati di caricamento, errore e aggiornamento della lista dopo modifiche.
- Visualizzazione di messaggi toast per notificare all’utente l’esito delle operazioni.

L’interazione con le API è gestita con observable RxJS per una gestione reattiva ed efficiente dei dati.



5.5.3 Gestisci Filiali

La funzionalità Gestisci Filiali si presenta in pages/admin/gestisci-filiali e consente all’utente amministratore di gestire il catalogo delle filiali registrate nel sistema. L’interfaccia è progettata per essere intuitiva e responsiva, sfruttando componenti standalone di Ionic come ion-content, ion-searchbar, ion-button, ion-spinner e ion-text, oltre a un componente personalizzato chiamato app-filiale-amministratore.

Al caricamento della pagina, il componente GestisciFilialiPage esegue automaticamente, all’interno del metodo ngOnInit(), una chiamata asincrona al backend tramite il servizio FilialeService per ottenere la lista completa delle filiali. Durante questo processo di caricamento, viene mostrato uno spinner circolare per informare l’utente che i dati sono in fase di recupero. Nel caso in cui si verifichi un errore durante questa operazione, la pagina visualizza un messaggio esplicito che invita l’utente a riprovare più tardi. La barra di ricerca, realizzata con il componente ion-searchbar, permette all’utente di filtrare le filiali in base all’indirizzo inserito. Questa funzionalità implementa un meccanismo di debounce, così da evitare di applicare il filtro ad ogni singola digitazione e migliorare così le prestazioni complessive. Il filtro agisce in tempo reale sulla lista delle filiali, aggiornandola dinamicamente in base al testo inserito.

La lista filtrata delle filiali viene mostrata tramite il componente app-filiale-amministratore, che visualizza i dettagli di ciascuna filiale e fornisce strumenti per modificarne le proprietà o avviare la procedura di cancellazione. Nel caso in cui si desideri eliminare una filiale, viene mostrato un alert modale che richiede la conferma dell’utente, evitando così cancellazioni accidentali. Se l’eliminazione viene confermata, viene effettuata una chiamata al servizio per rimuovere la filiale dal backend. In seguito, la lista delle filiali viene aggiornata localmente per riflettere la modifica, e viene mostrato un messaggio toast che comunica l’esito positivo o negativo dell’operazione. Oltre alla gestione delle filiali esistenti, la pagina include un pulsante con icona “+” che consente all’utente di navigare alla pagina dedicata all’aggiunta di nuove filiali. Inoltre, è possibile modificare i dati di una filiale selezionata tramite la funzione di navigazione che passa i dati necessari alla pagina di modifica.

Nel complesso, l’interazione con il backend e la gestione dello stato della pagina (caricamento, errore, aggiornamento dati) è realizzata in modo reattivo sfruttando gli observable di RxJS. Questo garantisce una gestione efficiente e fluida dei dati e una migliore esperienza utente. Tutte le operazioni di modifica e cancellazione sono corredate da messaggi di feedback tramite toast, per tenere sempre l’utente informato sull’esito delle azioni intraprese.

Gestisci Filiali

Cerca una filiale...

| Via Vincenzo Piazza Martini, 45 Località: Palermo | Via Palmerino, 52/A Località: Palermo | Via Catania, 17 Località: Palermo | Via Saitta Longhi, 116B Località: Palermo |
|---|--|--------------------------------------|--|
| Numero di tavoli totali: 22 | Numero di tavoli totali: 15 | Numero di tavoli totali: 17 | Numero di tavoli totali: 16 |
| MODIFICA FILIALE | MODIFICA FILIALE | MODIFICA FILIALE | MODIFICA FILIALE |
| GESTISCI IMPIEGATI | GESTISCI IMPIEGATI | GESTISCI IMPIEGATI | GESTISCI IMPIEGATI |
| RIMUOVI FILIALE | RIMUOVI FILIALE | RIMUOVI FILIALE | RIMUOVI FILIALE |

LEONARDO GIOVANNI CICIZZA DIEGO CORONA LUCA GRESTANI DANIELE ORAZIO SUCINO

5.5.4 Gestisci Impiegati

La funzionalità Gestisci Impiegati si presenta in pages/admin/gestisci-impiegati e consente all’utente amministratore di gestire il personale di una specifica filiale. La pagina presenta un’interfaccia chiara e responsiva, realizzata con componenti Ionic standalone come ion-content, ion-chip, ion-searchbar, ion-button, ion-spinner, ion-text e il componente personalizzato app-impiegato-amministratore.

All’apertura della pagina, il componente GestisciImpiegatiPage utilizza il metodo ngOnInit() per recuperare l’elenco completo degli impiegati appartenenti alla filiale selezionata. L’identificativo della filiale viene estratto dai parametri della query string o, in alternativa, dallo stato di navigazione del router. Viene quindi effettuata una chiamata asincrona al backend tramite il servizio ImpiegatoService per ottenere i dati. Durante il caricamento dei dati viene mostrato uno spinner circolare, mentre in caso di errore viene visualizzato un messaggio che invita l’utente a riprovare più tardi. L’utente può filtrare il personale visualizzato scegliendo una categoria tramite pulsanti ion-chip corrispondenti ai ruoli “Chef”, “Camerieri”, “Amministratori” oppure “Tutti”. Questo filtro agisce dinamicamente sulla lista interna degli impiegati. Inoltre, la barra di ricerca ion-searchbar consente di effettuare una ricerca testuale tra i dipendenti per nome, cognome o email, applicando un debounce per ottimizzare le prestazioni. La combinazione di filtro per ruolo e ricerca testuale aggiorna in tempo reale la lista degli impiegati mostrati.

Gli impiegati filtrati vengono visualizzati attraverso il componente app-impiegato-amministratore, che permette di visualizzare i dettagli di ciascun dipendente e di interagire con essi. Tramite questo componente è possibile avviare la procedura di licenziamento di un impiegato. Prima di procedere con la cancellazione, viene mostrato un alert modale di conferma, dove l’utente deve confermare esplicitamente l’azione per evitare cancellazioni accidentali. In caso di conferma, la pagina effettua una rimozione ottimistica dall’elenco locale e invia una chiamata al backend per eliminare definitivamente il dipendente. A seguito dell’operazione, viene mostrato un messaggio toast che notifica all’utente l’esito positivo o negativo dell’azione. Un pulsante “+” consente di navigare alla pagina per aggiungere un nuovo impiegato, facilitando la gestione del personale. L’interazione con le API è gestita tramite observable RxJS per garantire un flusso dati reattivo ed efficiente, mantenendo sincronizzato lo stato dell’interfaccia con le operazioni backend.

Tutte le operazioni di modifica e cancellazione sono corredate da messaggi di feedback tramite toast, per tenere sempre l’utente informato sull’esito delle azioni intraprese.

Gestisci impiegati

Cerca un dipendente...

| Arnold Fahey | Leticia Reichert | Mathew Boehm | Rene Morar | Linda Johnson |
|--|--|--|--|---|
| Matricola: 1 Ruolo: Chef Email: Arnold.Fahey9@yahoo.com Data di nascita: 1990-03-15 | Matricola: 2 Ruolo: Cameriere Email: Leticia.Reichert62@hotmail.com Data di nascita: 1987-06-30 | Matricola: 3 Ruolo: Cameriere Email: Mathew.Boehm96@hotmail.com Data di nascita: 1985-06-18 | Matricola: 4 Ruolo: Cameriere Email: Rene_Morar69@gmail.com Data di nascita: 1994-06-15 | Matricola: 5 Ruolo: Cameriere Email: Linda.Johnson49@gmail.com Data di nascita: 2002-03-22 |
| MODIFICA DATI IMPIEGATO | MODIFICA DATI IMPIEGATO | MODIFICA DATI IMPIEGATO | MODIFICA DATI IMPIEGATO | MODIFICA DATI IMPIEGATO |

© CISCUSASSI 2023

AMMINISTRAZIONE

LEONARDO GIOVANNI CICIZZA DIEGO CORONA LUCA GAETANI DANIELE ORAZIO SUCINO

5.5.5 Visualizza Utili

La funzionalità VisualizzaUtiliPage si presenta in pages/admin/visualizza-utili e consente la visualizzazione dettagliata per un amministratore degli utili mensili delle filiali per un anno selezionato, con la possibilità di esportare i dati in formato Excel. L'interfaccia presenta un contenuto a schermo intero con sfondo chiaro allo scopo di facilitare la visualizzazione della tabella degli utili. È disponibile un filtro per la selezione dell'anno tramite un menu a tendina che mostra gli anni disponibili (Ciscusassi è nato nel 2023, per questo motivo gli anni disponibili partono dal 2023) e un pulsante per scaricare un file Excel contenente i dati visualizzati.

La tabella è implementata utilizzando il componente <ion-grid> di Ionic, che consente una struttura a griglia flessibile e responsiva. La prima riga della griglia (<ion-row>) è dedicata all'intestazione, con una colonna per l'indirizzo della filiale e una colonna per ciascuno dei 12 mesi, generata tramite un ciclo sull'array months. L'ultima colonna è riservata al totale annuale. Ogni riga dati è anch'essa un <ion-row>, generata tramite un ciclo sull'array rows. All'interno di ogni riga, la prima colonna visualizza l'indirizzo della filiale, mentre le 12 colonne successive mostrano gli utili mensili. Questi valori numerici sono formattati usando il pipe Angular number con due decimali e il simbolo dell'euro aggiunto manualmente. L'ultima colonna della riga mostra il totale annuale calcolato tramite la funzione getRowTotal(row). Per migliorare la leggibilità, ad ogni riga viene assegnata una classe dinamica che alterna gli stili "even" e "odd" in base all'indice della riga. Infine, una riga aggiuntiva posta al fondo della tabella calcola e visualizza i totali per ogni mese (colonne mensili) e il totale complessivo (ultima colonna). Anche in questa riga i valori sono formattati in euro con due decimali e il testo della prima colonna è marcato in grassetto per indicare chiaramente la somma totale.

La tabella mostra le informazioni organizzate per filiale, con colonne che rappresentano i 12 mesi dell'anno più una colonna finale per il totale annuale. Gli indirizzi delle filiali sono visualizzati nella prima colonna, mentre le celle successive riportano gli utili mensili formattati in euro con due decimali. La tabella alterna lo stile delle righe per una migliore leggibilità e include una riga finale che calcola i totali per ogni mese e il totale complessivo.

All'avvio (ngOnInit), il componente imposta come anno selezionato il più recente disponibile (il primo elemento dell'array years) e avvia il caricamento dei dati corrispondenti chiamando il metodo loadDataForYear(selectedYear). Ogni volta che l'utente modifica la selezione dell'anno tramite il filtro (ion-select), viene invocato il metodo onYearChange(), che a sua volta richiama nuovamente loadDataForYear con il nuovo anno selezionato. Il metodo loadDataForYear(year: number) esegue due chiamate asincrone in sequenza. Prima chiama filialeService.GetSedi() per recuperare la lista delle filiali, da cui costruisce una

mappa filialiMap che associa l'ID di ogni filiale all'indirizzo completo, ottenuto concatenando il campo indirizzo e comune. Questa mappa viene utilizzata per visualizzare indirizzi leggibili all'interno della tabella. Subito dopo, richiama pagamentoService.GetUtiliMensili(year) per ottenere i dati relativi agli utili mensili dell'anno selezionato. I dati ricevuti contengono per ogni record il nome del mese e l'importo dell'utile associato a una specifica filiale.

Per organizzare queste informazioni, viene creato un oggetto grouped in cui ogni chiave corrisponde all'ID di una filiale e il valore è un array di 12 elementi inizializzati a zero, rappresentanti i mesi dell'anno. Viene quindi iterato l'array di dati degli utili, determinando per ciascun elemento l'indice corrispondente al mese all'interno dell'array months. L'importo viene assegnato alla posizione appropriata nell'array mensile di quella filiale. Nel caso in cui una filiale non sia presente in grouped (situazione possibile con dati incompleti), viene inizializzato un nuovo array di zeri per assicurare la coerenza della struttura. Una volta completata la popolazione dei dati, l'oggetto grouped viene trasformato in un array rows, dove ogni elemento contiene l'indirizzo completo della filiale e l'array degli importi mensili, pronto per essere visualizzato nella tabella. Se si verificano errori nelle chiamate ai servizi, i dati della tabella vengono resettati a un array vuoto per evitare la visualizzazione di informazioni inconsistenti.

La pagina rileva inoltre il browser in uso per applicare classi CSS specifiche che migliorano la resa grafica su Chrome e Firefox. La funzione di esportazione genera un file Excel formattato in modo leggibile, con intestazioni colorate, righe alternate e formule di somma per righe, colonne e totale generale. Le larghezze delle colonne sono automaticamente adattate in base al contenuto, assicurando una visualizzazione ordinata e professionale nel file esportato.

| Indirizzo filiale | Gennaio | Febbraio | Marzo | Aprile | Maggio | Giugno | Luglio |
|--|------------|------------|------------|------------|------------|------------|----------|
| Via Vincenzo Piazza Martini, 45, Palermo | € 707.92 | € 896.35 | € 824.46 | € 723.45 | € 1.028.33 | € 815.71 | € 201.00 |
| Via Palmerino, 52/A, Palermo | € 498.32 | € 910.16 | € 630.87 | € 590.48 | € 1.001.63 | € 977.92 | € 0.00 |
| Via Catania, 17, Palermo | € 575.08 | € 693.34 | € 804.62 | € 476.67 | € 520.30 | € 701.64 | € 0.00 |
| Via Saitta Longhi, 116G, Palermo | € 901.65 | € 633.37 | € 627.40 | € 677.06 | € 514.42 | € 1.083.28 | € 0.00 |
| TOTALE | € 2.682.97 | € 3.133.22 | € 2.887.35 | € 2.467.66 | € 3.064.68 | € 3.578.55 | € 201.00 |

5.5.6 Aggiungi Piatti

La funzionalità Aggiungi Piatti si presenta in pages/admin/aggiungi-piatti e consente all'utente amministratore di creare un nuovo piatto da inserire nel catalogo dei prodotti. L'interfaccia è sviluppata interamente utilizzando componenti standalone di Ionic, come ion-input, ion-textarea, ion-select, ion-img, ion-card e ion-button, ed è progettata per essere responsiva e accessibile da qualsiasi dispositivo. Il componente AggiungiPiattiPage gestisce sia il layout della pagina sia la logica di business associata alla creazione di un piatto.

Al caricamento della pagina, l'utente visualizza un modulo diviso in due colonne: una per il caricamento dell'immagine rappresentativa del piatto e l'altra per l'inserimento delle informazioni principali come nome, descrizione, costo e categoria. Il caricamento dell'immagine è gestito localmente tramite FileReader, che converte l'immagine selezionata in formato base64 per permetterne una visualizzazione im-

mediata e l'invio al backend. L'utente può selezionare una categoria tra quelle disponibili (ANTIPASTO, PRIMO, BEVANDA, DOLCE) tramite una select a interfaccia popover, e indicare se si tratta di un “piatto del giorno” tramite un campo booleano. Al momento dell'invio, il metodo creaPiatto() verifica che tutti i campi obbligatori siano compilati. In caso contrario, viene mostrato un messaggio toast di avviso. Se i dati sono validi, viene costruito un oggetto ProdottoInput e inviato al backend tramite il servizio ProdottoService.

La risposta positiva del backend attiva un messaggio toast di conferma e la navigazione automatica verso la pagina di gestione dei piatti (/gestisci-piatti), mentre in caso di errore viene mostrato un messaggio di errore dettagliato. Dopo ogni operazione completata, il form viene resettato per permettere l'inserimento di un nuovo piatto. L'interazione con il backend è completamente asincrona e reattiva, sfruttando gli observable di RxJS per garantire fluidità e reattività durante tutto il processo di creazione del piatto. L'uso combinato di feedback visivi tramite toast, immagini dinamiche e un'interfaccia modulare assicura una user experience chiara, efficiente e immediata.

5.5.7 Aggiungi Filiali

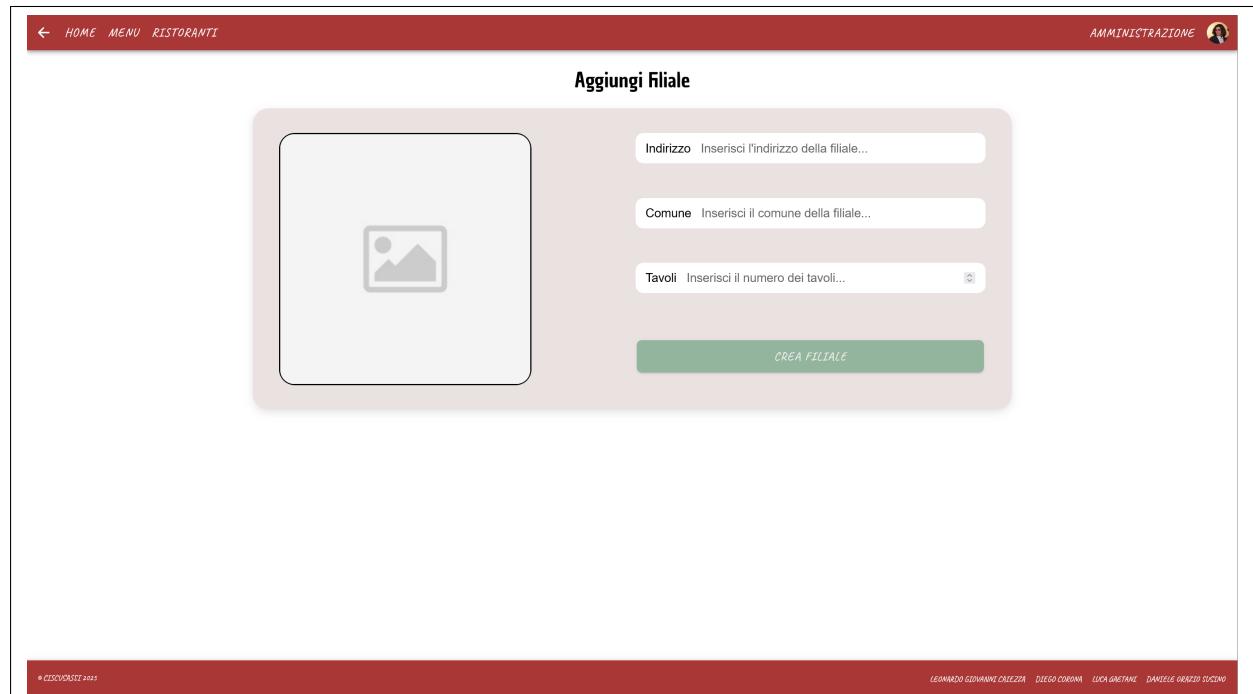
La funzionalità Aggiungi Filiali si presenta in pages/admin/aggiungi-filiali e consente all'utente amministratore di registrare una nuova filiale specificando i dati fondamentali come l'indirizzo, il comune, il numero di tavoli disponibili e un'immagine rappresentativa della sede. L'interfaccia è realizzata esclusivamente con componenti standalone di Ionic, tra cui ion-input, ion-img, ion-card, ion-button, ion-text, ion-list e ion-item, e si adatta in modo fluido a qualsiasi risoluzione grazie a un layout responsivo basato su ion-grid.

All'apertura della pagina, l'utente visualizza una card contenente un form suddiviso in due colonne: a sinistra si trova un box per il caricamento dell'immagine della filiale, mentre a destra sono presenti i campi di input testuale. Il caricamento dell'immagine è gestito localmente attraverso un componente input[type=file] nascosto, il cui trigger avviene cliccando sull'area visiva. L'immagine viene immediatamente convertita in formato Base64 tramite FileReader per essere visualizzata come anteprima e allegata al payload destinato al backend.

L'utente può inserire l'indirizzo della filiale tramite un campo ion-input con funzionalità di autocomplete basata sull'API di TomTom Search. Una volta iniziata la digitazione, il sistema effettua un debounce e propone un elenco di suggerimenti aggiornati che possono essere selezionati con un semplice tap. Selezionando un suggerimento, il campo viene automaticamente aggiornato. Altri campi disponibili includono il comune della filiale e il numero di tavoli (gestito come input numerico con validazione lato client). Tutti

i campi sono obbligatori: il pulsante CREA FILIALE è abilitato solo quando tutti i valori richiesti sono stati correttamente inseriti e validati. Il metodo creaFiliale() si occupa della logica di creazione. Dopo aver verificato che tutti i dati siano presenti, la funzione concatena indirizzo e comune per effettuare una richiesta di geocoding all'API di TomTom, ottenendo così latitudine e longitudine precise della nuova filiale. Se le coordinate sono valide, viene costruito un oggetto FilialeInput che include anche l'immagine convertita, e il tutto viene inviato al backend tramite il servizio FilialeService.

Il processo è asincrono e reattivo: sfrutta l'approccio observable per rispondere in modo efficace sia in caso di successo (mostrando un toast di conferma e reindirizzando l'utente alla pagina /gestisci-filiali) sia in caso di errore (con toast di errore dettagliati). Al termine dell'operazione, il form viene azzerato automaticamente per consentire l'inserimento rapido di un'ulteriore filiale. Il feedback visuale viene gestito tramite il ToastController di Ionic, garantendo un'interazione immediata e coerente con l'utente. L'approccio modulare, l'integrazione con servizi esterni e l'interfaccia dinamica assicurano un'esperienza utente intuitiva, efficiente e professionale.



5.5.8 Aggiungi Impiegati

La funzionalità Aggiungi Impiegati si presenta in pages/admin/aggiungi-impiegati e consente all'utente amministratore di inserire nuovi dipendenti in una filiale. L'interfaccia è completamente realizzata con componenti standalone di Ionic, come ion-card, ion-input, ion-select, ion-img, ion-button, ion-text, ion-grid e altri, e presenta un layout responsivo strutturato in due colonne grazie all'utilizzo del sistema a griglia ion-grid. All'apertura, l'utente visualizza un'intestazione che introduce il form e una card centrale suddivisa in due sezioni. Nella colonna sinistra è presente un'area per il caricamento dell'immagine del dipendente, visualizzata tramite ion-img. L'immagine viene selezionata cliccando sull'area visiva, che attiva un input di tipo file nascosto. Il file selezionato viene immediatamente convertito in formato Base64 tramite FileReader e mostrato in anteprima nell'interfaccia. Il valore base64 viene poi incluso nel payload inviato al backend. La colonna destra del form contiene tutti i campi necessari per l'inserimento del dipendente, tra cui nome, cognome, data di nascita, email, ruolo e filiale di assegnazione. I campi sono implementati con componenti ion-input e ion-select, e la selezione delle filiali disponibili viene popolata dinamicamente tramite una chiamata asincrona al servizio FilialeService. La validazione è eseguita localmente: l'indirizzo email viene controllato tramite espressione regolare e il pulsante per confermare l'assunzione risulta disabilitato fino a quando tutti i campi richiesti non sono compilati correttamente, ad eccezione della data di nascita e l'immagine del dipendente, che vengono controllati successivamente alla pressione del pulsante e presentano dei toast personalizzati in caso di errore.

Il metodo aggiungiImpiegato() implementa la logica principale: crea un oggetto ImpiegatoInput con i dati forniti, valida la correttezza dell'email e la selezione della filiale, quindi invia la richiesta al backend tramite il ImpiegatoService. L'operazione è asincrona e gestita tramite Observable. In caso di successo, viene mostrato un Toast di conferma e l'utente viene riportato alla vista precedente grazie a NavController. In caso di errore, viene notificato tramite un Toast con messaggio personalizzato. Al termine dell'operazione, il form viene resettato per consentire l'inserimento di nuovi dati. La struttura è modulare, l'interfaccia chiara e l'esperienza utente fluida ed efficiente grazie all'uso di feedback tramite ToastController visivi immediati e una distribuzione intelligente dei contenuti nella pagina.

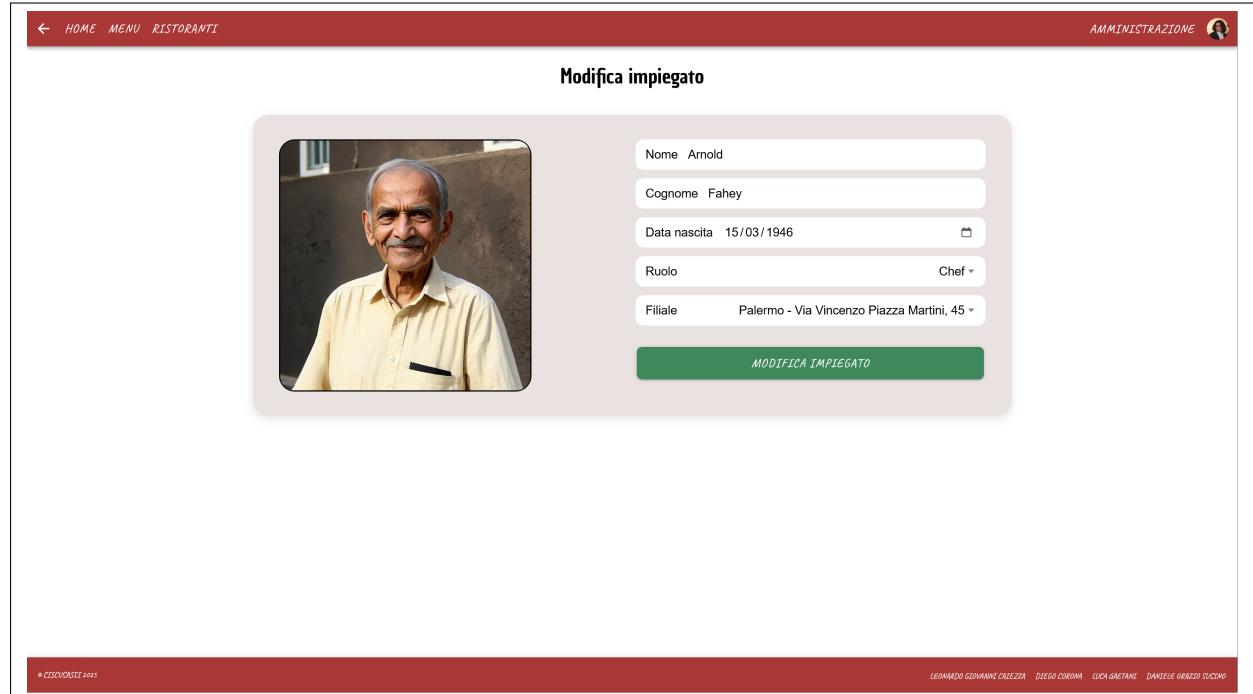
5.5.9 Modifica Impiegati

La funzionalità Modifica Impiegati si presenta in pages/admin/modifica-impiegati e consente all'utente amministratore di aggiornare i dati anagrafici e professionali di un dipendente esistente. L'interfaccia utente è sviluppata interamente utilizzando componenti standalone di Ionic, come ion-card, ion-input, ion-select, ion-img, ion-button, ion-grid e ion-text, organizzati in un layout responsive a due colonne grazie al sistema ion-grid. All'apertura della vista, viene visualizzata un'intestazione centrale che guida l'utente nell'operazione di modifica. All'interno della ion-card, la parte sinistra del layout ospita una sezione dedicata al caricamento o aggiornamento dell'immagine del dipendente. L'immagine può essere selezionata cliccando direttamente sull'area visiva, la quale attiva un input file nascosto che consente il caricamento di immagini nei formati JPG, JPEG, PNG o WebP. L'immagine selezionata viene immediatamente convertita in stringa Base64 tramite FileReader e visualizzata in anteprima tramite ion-img. Il valore codificato in base64 viene poi assegnato alla proprietà foto e incluso nel payload della richiesta HTTP per l'aggiornamento del profilo.

La colonna destra del form contiene i campi modificabili relativi al dipendente: nome, cognome, data di nascita, ruolo e il campo filiale di appartenenza che viene automaticamente caricato all'avvio della pagina. I campi sono implementati tramite ion-input e ion-select, utilizzano il binding bidirezionale ngModel e presentano una stilizzazione coerente con il design generale grazie a classi CSS personalizzate. I ruoli sono predefiniti e disponibili tramite un menu a tendina, mentre l'elenco delle filiali viene recuperato in modo asincrono dal backend mediante una chiamata al FilialeService, con la risposta gestita come Observable. La select filiale mostra dinamicamente l'indirizzo e il comune di ogni sede disponibile. Tutti i campi sono soggetti a validazione lato client. Il pulsante per salvare le modifiche è abilitato solo quando tutti i campi richiesti (inclusa la foto) sono correttamente compilati. Il metodo salvaModifiche() incapsula la logica operativa: verifica la presenza della matricola del dipendente (necessaria per l'aggiornamento), la completezza dei dati inseriti e la correttezza dei riferimenti selezionati. Se la validazione è superata, viene costruito un oggetto ImpiegatoData e inviato al backend attraverso il servizio ImpiegatoService,

utilizzando una richiesta HTTP asincrona. In caso di successo, viene mostrato un Toast di conferma e l'utente viene riportato alla schermata precedente tramite NavController. In caso di errore, un messaggio Toast con colore rosso informa l'utente del fallimento dell'operazione. L'intero flusso è progettato per garantire fluidità, chiarezza e reattività nell'esperienza di modifica dei dati.

Il componente è implementato come standalone e include, nella sua dichiarazione, tutti i moduli e componenti Ionic necessari al funzionamento, rendendolo riutilizzabile e indipendente da moduli Angular tradizionali. I dati del dipendente da modificare vengono ricevuti tramite il NavigationExtras del Router, e inizializzati nel ciclo di vita ngOnInit, permettendo un caricamento dinamico del form sulla base del contesto. Il codice è ben strutturato, modulare e mantiene una forte separazione tra logica di presentazione, logica di business e servizi dati, con una UX coerente rispetto alle altre sezioni del sistema di gestione.



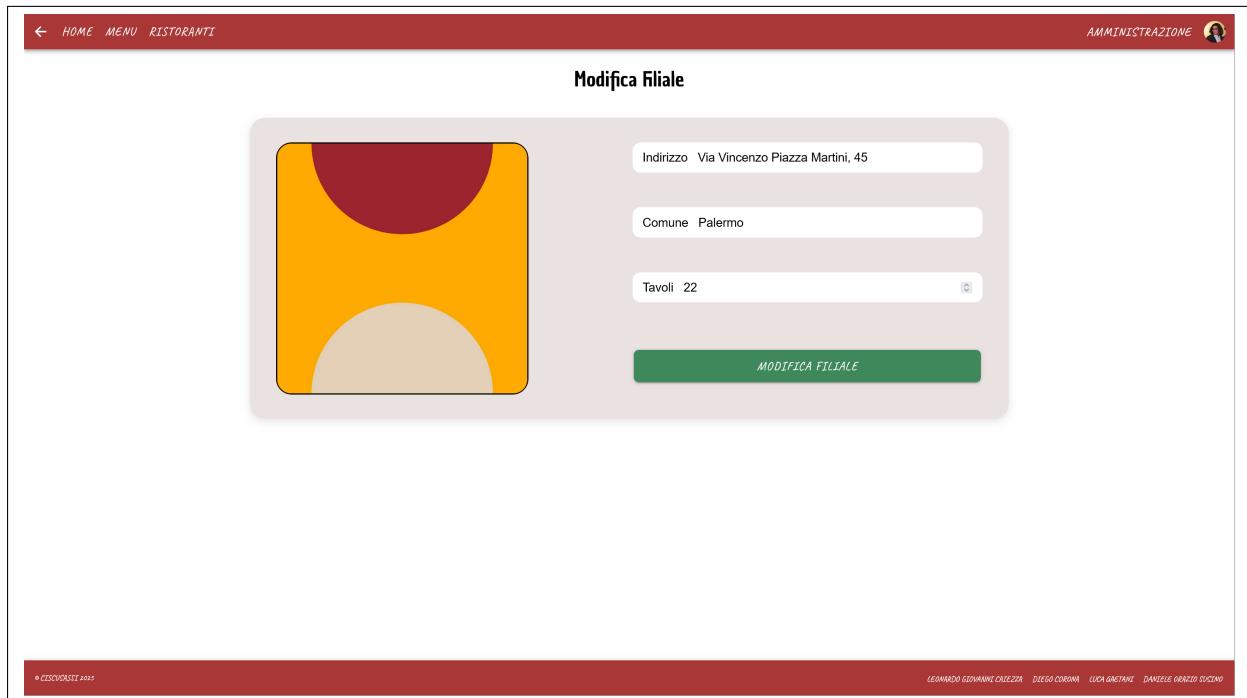
5.5.10 Modifica Filiali

La funzionalità Modifica Filiali si presenta in pages/admin/modifica-filiali e consente all'utente amministratore di aggiornare le informazioni principali relative a una filiale esistente. L'interfaccia utente è realizzata interamente con componenti standalone di Ionic, tra cui ion-card, ion-input, ion-button, ion-grid, ion-text, ion-img, e ion-list, organizzati all'interno di un layout responsive a due colonne tramite il sistema ion-grid.

All'apertura della vista, viene visualizzata un'intestazione centrale che introduce l'operazione di modifica. Il contenuto principale è racchiuso all'interno di una ion-card, la cui colonna sinistra è dedicata alla gestione dell'immagine della filiale. In quest'area, un contenitore cliccabile consente all'utente di caricare o sostituire l'immagine tramite un input file nascosto, compatibile con i formati .jpg, .jpeg, .png e .webp. Il file selezionato viene elaborato in tempo reale e convertito in una stringa base64 attraverso l'oggetto FileReader. La stringa risultante viene immediatamente assegnata alla proprietà filiale.immagine, rendendo disponibile un'anteprima aggiornata tramite il componente ion-img. La colonna destra del layout contiene i campi modificabili associati alla filiale: indirizzo, comune e numero di tavoli. I campi sono implementati con ion-input e collegati direttamente all'oggetto filiale attraverso il binding bidirezionale ngModel, con una formattazione coerente garantita da classi CSS personalizzate. La digitazione dell'indirizzo attiva dinamicamente il metodo cercaIndirizzo(), che interroga l'API TomTom per ottenere suggerimenti auto-completati. I risultati vengono visualizzati in una lista interattiva: selezionandone uno, l'indirizzo e il comune vengono automaticamente compilati, insieme alle coordinate geografiche associate, aggiornando i campi latitudine e longitudine. Il campo "Comune" può essere eventualmente modificato manualmente,

mentre il numero dei tavoli è vincolato a valori numerici validi. Tutti i campi del form sono soggetti a validazione lato client, e il pulsante di salvataggio rimane disabilitato finché non viene fornita un'immagine valida e tutti i dati richiesti non risultano compilati correttamente.

Il metodo modificaFiliale() incapsula l'intera logica operativa. Prima di procedere, verifica che tutti i campi richiesti siano stati completati e che siano presenti le coordinate geografiche. In caso contrario, l'utente viene avvisato tramite un messaggio Toast con colore warning. Se i dati sono corretti, il metodo geocodificaIndirizzo() utilizza l'API TomTom per aggiornare latitudine e longitudine, e quindi costruisce l'oggetto FilialeData da inviare al backend. L'operazione di aggiornamento viene eseguita attraverso il servizio FilialeService, mediante una richiesta HTTP asincrona. In caso di esito positivo, un Toast verde conferma l'aggiornamento e l'utente viene reindirizzato alla schermata di gestione filiali tramite NavController. In caso di errore, viene visualizzato un Toast. I dati della filiale da modificare vengono recuperati attraverso la proprietà navigation.extras.state del Router, e inizializzati nel ciclo di vita ngOnInit, rendendo possibile la precompilazione dinamica dei campi in base alla filiale selezionata in precedenza. L'intera implementazione è orientata a garantire fluidità, immediatezza e coerenza con l'esperienza utente delle altre funzionalità del sistema gestionale.



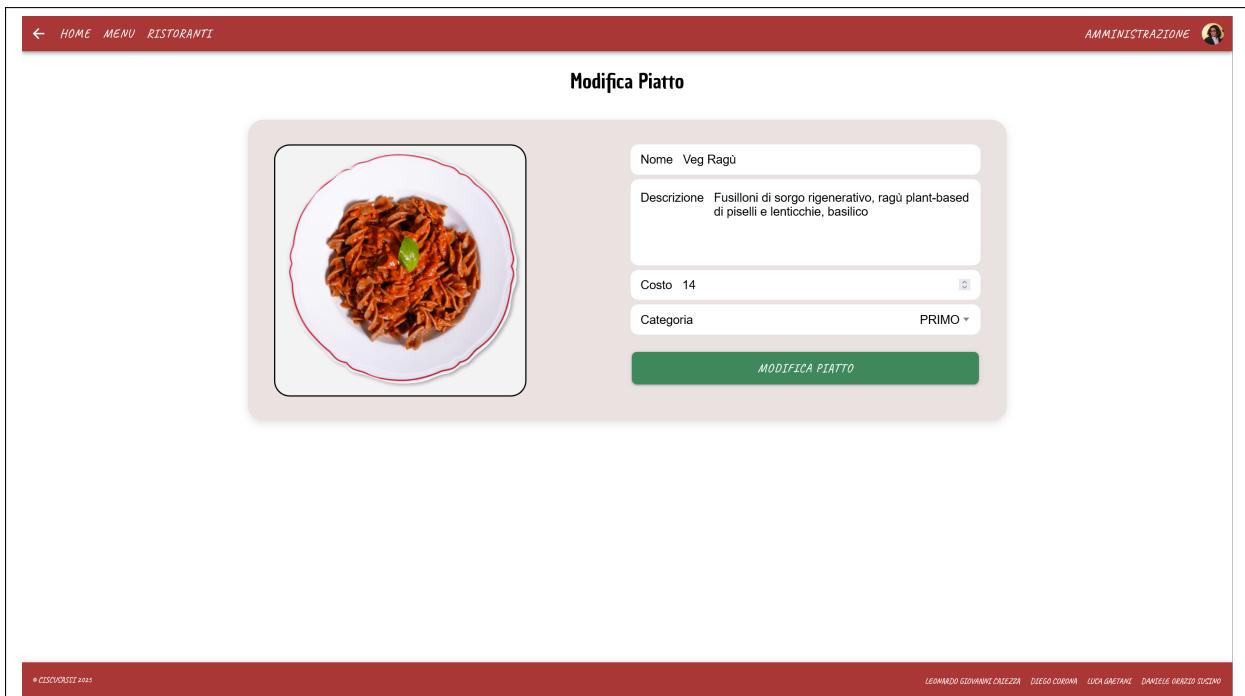
5.5.11 Modifica Piatti

La funzionalità Modifica Piatti si presenta in pages/admin/modifica-piatti e consente all'utente amministratore di aggiornare le informazioni relative a un piatto esistente nel menù. L'interfaccia utente è sviluppata interamente con componenti standalone di Ionic, come ion-card, ion-input, ion-textarea, ion-select, ion-img, ion-button, ion-grid, ion-row e ion-col, distribuiti in un layout responsive a due colonne grazie al sistema ion-grid.

All'apertura della vista viene visualizzato un titolo centrale che guida l'utente nell'operazione di modifica. L'intero form è racchiuso all'interno di una ion-card, divisa in due sezioni principali. Nella colonna sinistra, l'utente può aggiornare l'immagine associata al piatto: cliccando sull'area designata, viene attivato un campo input di tipo file, nascosto, che accetta immagini nei formati jpg, jpeg, .png e .webp. L'immagine selezionata viene immediatamente convertita in stringa Base64 tramite FileReader e visualizzata in anteprima nel componente ion-img. Il valore codificato viene poi assegnato alla proprietà immagine dell'oggetto piatto e incluso nel payload della richiesta HTTP per il salvataggio. Nella colonna destra del layout, sono presenti i campi modificabili del piatto: nome, descrizione, costo e categoria. Tutti i campi sono implementati con componenti ion-input, ion-textarea e ion-select, con binding bidirezionale ngModel per mantenere la sincronizzazione automatica tra l'interfaccia e l'oggetto dati. Le categorie disponibili (ad esempio: ANTIPASTO, PRIMO, BEVANDA, DOLCE) sono predefinite e visualizzate in

un menu a tendina. Ogni campo è stilizzato coerentemente con il resto dell’interfaccia grazie a classi CSS personalizzate. Il pulsante ”Modifica Piatto”, posizionato alla base del form, è attivo solo quando i dati sono stati correttamente compilati. Al clic, viene invocato il metodo `modificaPiatto()`, che incapsula tutta la logica operativa: prima verifica la presenza dell’`id_prodotto`, quindi costruisce un oggetto `ProdottoInput` contenente i nuovi valori e lo invia al backend tramite il servizio `ProdottoService`. L’operazione è gestita in maniera asincrona, e in caso di successo viene mostrato un `Toast` verde di conferma seguito da un reindirizzamento automatico alla schermata precedente tramite `NavController`. In caso di errore, un `Toast` rosso informa l’utente del fallimento dell’operazione, con messaggi differenziati per errori lato server o problemi di comunicazione.

Il componente è implementato come standalone, includendo al proprio interno tutti i moduli Ionic e Angular necessari, e non dipende da moduli Angular tradizionali, risultando pienamente riutilizzabile e indipendente. I dati del piatto da modificare vengono passati tramite lo stato di navigazione (`NavigationExtras`) e inizializzati nel ciclo di vita `ngOnInit`, permettendo il caricamento dinamico e contestuale del form. L’intera interfaccia è pensata per offrire chiarezza, immediatezza e coerenza visiva con le altre funzionalità del sistema gestionale.



5.6 Gestione Cameriere

5.6.1 Visualizza Tavoli Cameriere

La funzionalità Visualizza Tavoli Cameriere si presenta in `pages/cameriere/visualizza-tavoli-cameriere` e consente all’utente cameriere di gestire le prenotazioni in un fiale rispetto al suo ruolo. La logica è interamente costruita con componenti standalone Ionic, combinati con direttive strutturali Angular per il controllo del flusso e l’interpolazione dati.

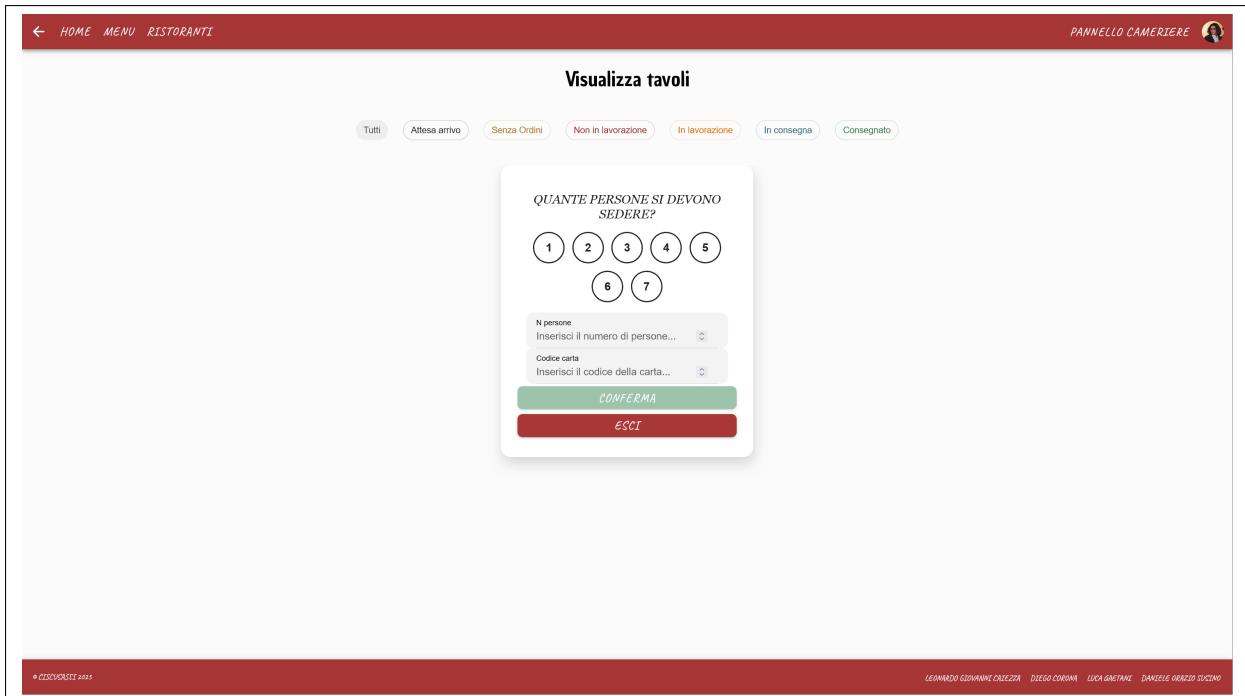
Se il locale non è aperto , viene mostrato un messaggio ben visibile che indica la chiusura dell’attività, evidenziato con una classe stilizzata `text-color-rosso`. Se invece il locale è attivo, viene resa disponibile un’interfaccia interattiva per la visualizzazione e gestione dei tavoli. In alto nella vista viene mostrata una barra filtri implementata tramite `ion-chip`, che permette di filtrare i tavoli in base al loro stato (ad esempio: attesa arrivo, senza ordini, in lavorazione, in consegna, ecc.). Ogni chip attiva un filtro invocando il metodo `filtroTavoliPerStato()` con un parametro identificativo. Lo stato attivo è rappresentato graficamente tramite l’attributo `[outline]`. Se il risultato del filtro non produce tavoli, viene mostrato un messaggio che comunica l’assenza di prenotazioni correnti. Al centro dell’interfaccia si trova una griglia responsive gestita con `ion-grid` che visualizza, per ciascun tavolo, il numero, la prenotazione, l’orario e il numero di persone. Ogni tavolo è rappresentato da un cerchio colorato il cui stile dipende dallo stato

associato. L'ultimo elemento della griglia è un'icona di aggiunta (ion-icon con nome add) che, se cliccata, apre un modale per la creazione di una nuova prenotazione.

La creazione di una nuova prenotazione avviene tramite un ion-modal che richiede di selezionare il numero di persone (sia da scelta rapida sia tramite input manuale) e il codice cliente. Il form è strutturato in due sezioni: un elenco di opzioni predefinite visualizzate in cerchi cliccabili e un input numerico per l'inserimento manuale. I campi sono gestiti tramite [(ngModel)] per la sincronizzazione automatica dei dati. Il pulsante "CONFERMA" viene abilitato solo quando entrambi i campi contengono valori validi, garantendo una validazione formale a livello di interfaccia. Un secondo ion-modal consente all'utente di confermare l'arrivo del cliente associato a un tavolo specifico. Anche in questo caso, la conferma è gestita da un pulsante che richiama il metodo confermaArrivo(), mentre il pulsante "Indietro" chiude il modale corrente.

Tutta la logica di controllo come apertura e chiusura dei modali, gestione dei dati in input e selezioni, e aggiornamento dinamico dei tavoli filtrati è implementata nel file TypeScript associato, il quale gestisce lo stato dell'interfaccia attraverso proprietà reattive e metodi dichiarativi. Il file TS utilizza binding bidirezionale e approccio dichiarativo Angular per la gestione dello stato UI, sincronizzazione del modello dei dati e invocazione delle operazioni sul backend tramite servizi. Il file TypeScript controlla l'interazione dell'utente con l'interfaccia "Visualizza Tavoli" e gestisce il comportamento dei componenti collegati. Quando l'utente apre la pagina, vengono inizializzati i dati relativi al locale e ai tavoli disponibili. Le prenotazioni vengono caricate e filtrate secondo vari stati gestionali come "attesa arrivo", "senza ordini" e "in lavorazione", "non in lavorazione", "in consegna", "consegnato". Le proprietà come tavoliFiltrati, selectedFilter, localeAperto e personePossibili sono utilizzate per tenere traccia dello stato attuale dell'interfaccia e per alimentare dinamicamente il contenuto dei componenti. I metodi principali includono filtraTavoliPerStato() per l'applicazione di filtri, visualizzaModaleAggiungiPrenotazione() per l'apertura del modale, confermaModaleInserimentoPrenotazione() per la creazione di una nuova prenotazione dopo la verifica dei dati inseriti, e confermaArrivo() per confermare la presenza del cliente.

Le interazioni utente sono guidate da controlli di validazione che prevengono l'invio di dati incompleti o errati. Le azioni più critiche sono accompagnate da feedback visivo tramite toast o da conferme modali, garantendo un'esperienza utente guidata e intuitiva. L'intero componente è progettato in modalità standalone, includendo al proprio interno le dipendenze Ionic e Angular necessarie, e si integra pienamente nel flusso gestionale dell'applicazione.



5.6.2 Visualizza Ordini Cameriere

La funzionalità Visualizza Ordini Cameriere si presenta in pages/cameriere/visualizza-ordini-cameriere e consente all’utente cameriere di gestire in tempo reale lo stato delle comande relative a un tavolo specifico. Sviluppato come componente standalone, integra direttamente tutti i moduli e componenti necessari, inclusi quelli di Ionic, Angular core e moduli condivisi, rendendolo completamente autonomo e riutilizzabile. Alla sua inizializzazione, il componente recupera l’oggetto tavolo attualmente selezionato dal servizio TavoloService, estraendo contestualmente il codice prenotazione, che viene utilizzato come chiave per interrogare il backend attraverso OrdineService e ottenere l’elenco dei prodotti ordinati.

Il caricamento dei dati è asincrono e gestito tramite Observable, encapsulato in un BehaviorSubject per garantire un flusso reattivo all’interno dell’interfaccia. I dati ottenuti vengono filtrati e strutturati secondo l’interfaccia OrdProdEstended e visualizzati nella parte inferiore della pagina tramite il componente figlio ListaOrdiniComponent, che riceve l’Observable come input per il binding dinamico. Ogni 30 secondi, il sistema esegue un refresh automatico dei dati tramite setInterval, assicurando l’aggiornamento continuo dell’elenco comande, senza necessità di interazione manuale. L’interfaccia include anche un pulsante “CONSEGNA TUTTO” che consente al cameriere di cambiare lo stato di tutti i prodotti attualmente in consegna associati alla prenotazione del tavolo. La logica è gestita nel metodo consegnaTutto(), che cicla sui prodotti filtrando quelli con stato “in-consegna” ed esegue un aggiornamento asincrono verso lo stato “consegnato” mediante chiamata al backend. Al termine di ogni aggiornamento, viene eseguito un nuovo caricamento dei dati per riflettere i cambiamenti in tempo reale.

La schermata è progettata con attenzione all’usabilità e alla chiarezza: in caso di caricamento in corso, viene mostrato uno spinner di tipo ion-spinner colorato di bianco, mentre in stato pronto, la lista comande viene visualizzata. L’interfaccia è interamente costruita con componenti Ionic standalone, all’interno di un contenitore ion-content a schermo pieno, con uno sfondo colorato di verde chiaro per indicare visivamente il contesto operativo attivo. Nella parte superiore viene mostrato un titolo centrato con il testo “VISUALIZZA COMANDE” e il numero del tavolo attualmente in gestione, con grafica coerente allo stile dell’applicazione. Al centro dell’interfaccia è posizionato un pulsante “CONSEGNA TUTTO”, che consente all’utente di notificare la consegna totale dei prodotti in lavorazione. Il pulsante è disabilitato automaticamente quando il sistema è in fase di caricamento, prevenendo interazioni multiple e condizioni di concorrenza. In fase di caricamento iniziale o durante aggiornamenti periodici, viene mostrato un indicatore di caricamento di tipo ion-spinner con animazione “crescent”, altrimenti viene renderizzato dinamicamente il componente figlio app-lista-ordini, che riceve in input il flusso prodotti\$ e visualizza l’elenco aggiornato dei piatti ordinati.

L'intera UI è progettata per essere chiara, diretta e facilmente interpretabile in ambienti ad alta operatività, come quello di un ristorante, facilitando l'interazione rapida da parte del cameriere e integrandosi perfettamente con il flusso gestionale del sistema.



5.7 Gestione Chef

5.7.1 Visualizza Tavoli Chef

La funzionalità Visualizza Tavoli Chef si presenta in pages/chef/visualizza-tavoli-chef e consente all'utente chef di visualizzare e monitorare in tempo reale le prenotazioni dei tavoli di una determinata filiale. L'interfaccia si presenta in modo pulito e ordinato, con un'intestazione centrale che introduce la funzionalità. Se il locale risulta chiuso, un messaggio informativo evidenziato in rosso notifica chiaramente lo stato corrente. In caso contrario, l'utente ha accesso a una dashboard interattiva per la gestione delle prenotazioni.

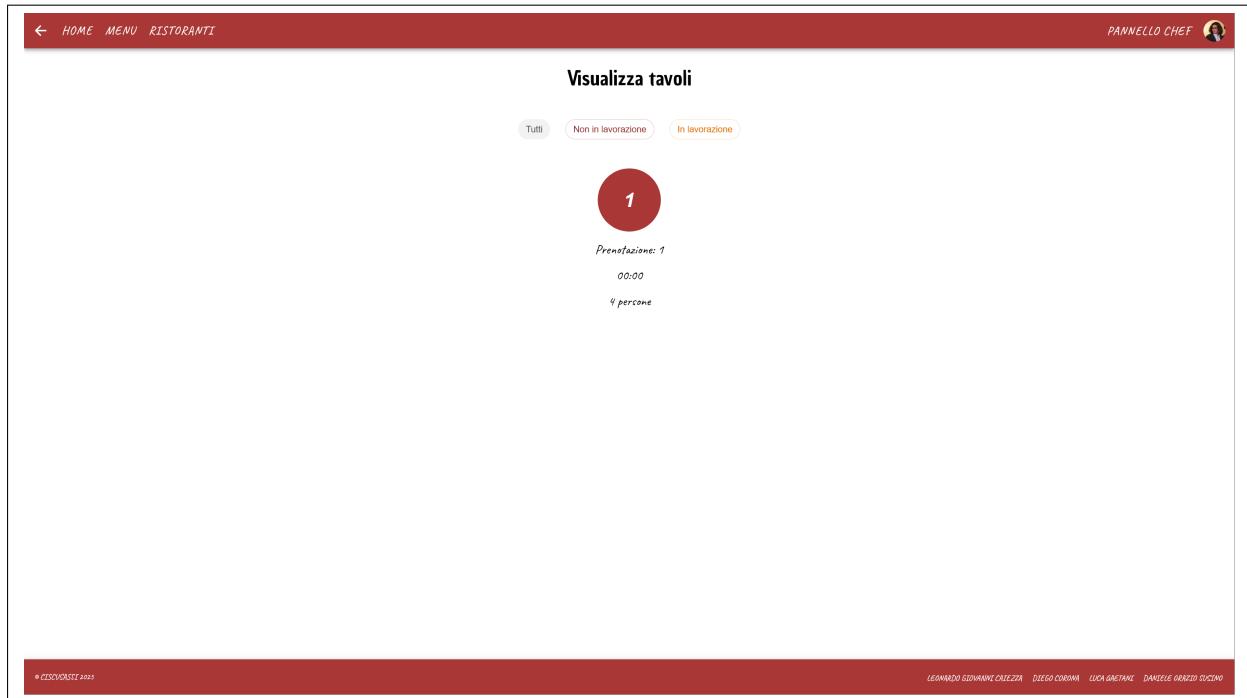
L'interfaccia grafica integra una barra filtri attraverso l'uso di ion-chip, consentendo all'utente di selezionare e visualizzare i tavoli in base allo stato della prenotazione, come "in lavorazione" o "non in lavorazione". I filtri sono dinamici e modificano in tempo reale la vista dei tavoli disponibili. Se non ci sono prenotazioni attive, viene visualizzato un messaggio che informa l'utente dell'assenza di ordini correnti. I tavoli sono organizzati all'interno di una griglia responsiva gestita da ion-grid, in cui ogni cella rappresenta un tavolo con le relative informazioni: numero, ID prenotazione, orario e numero di persone. Ogni tavolo è stilizzato con un cerchio colorato che ne riflette lo stato. Cliccando su un tavolo, si viene reindirizzati alla pagina di dettaglio degli ordini associati, mediante l'invocazione del metodo handleClick() che utilizza il router Angular per la navigazione e salva temporaneamente lo stato del tavolo selezionato nel servizio TavoloService.

Nel file TypeScript associato, la logica è centralizzata per gestire sia lo stato dell'interfaccia che l'interazione con i servizi backend. Alla prima apertura della pagina, viene verificato se il locale è aperto e, se positivo, vengono caricati automaticamente i dati dei tavoli tramite loadTavoli(), che a sua volta effettua chiamate ai servizi per ottenere prenotazioni e relativo stato operativo. La funzione checkOrariAperitura() è responsabile del controllo dell'orario di attività, con aggiornamento regolare tramite setInterval e disattivazione dei timer nella fase di ngOnDestroy() per evitare memory leak. L'interfaccia si adatta allo stato del locale, mostrando solo se necessario i contenuti interattivi. La struttura è dichiarativa e completamente gestita attraverso binding bidirezionale e direttive strutturali Angular, con un approccio reattivo per aggiornare i tavoli in base ai filtri selezionati. I messaggi di errore o stato sono forniti tramite ToastController, migliorando la chiarezza e l'interazione dell'utente con il sistema. Questo componente è

perfettamente integrato nella logica gestionale del sistema di prenotazioni e rappresenta uno strumento efficace per la supervisione e gestione operativa da parte dello chef.

Parallelamente alla versione cameriere della pagina, localizzata in pages/cameriere/visualizza-tavoli-cameriere, fornisce una struttura simile nella gestione dei tavoli, ma con funzionalità specifiche specifiche per il ruolo di chef. Anche in questo caso, la pagina è costruita utilizzando componenti standalone di Ionic e Angular, con flusso dati gestito tramite binding dichiarativi. La vista è introdotta da un titolo centrale, seguito da un messaggio condizionato che segnala l'eventuale chiusura del locale, sempre evidenziato in rosso. In caso di apertura del locale, la pagina mostra una serie di filtri in alto che consentono la visualizzazione dei tavoli per stato.

Tutta la logica di validazione, sincronizzazione dei dati e aggiornamento dei tavoli filtrati è contenuta nel file TypeScript associato. Le proprietà reattive mantengono lo stato coerente, mentre i metodi associati consentono una gestione chiara e modulare delle interazioni. Le operazioni utente sono guidate da controlli visivi, feedback attraverso toast e rendendo l'esperienza utente fluida, chiara e coerente. L'approccio dichiarativo, insieme alla separazione delle responsabilità tra vista e logica, rende questa pagina una parte robusta e ben strutturata dell'applicazione.



5.7.2 Visualizza Ordini Chef

La funzionalità Visualizza Ordini Chef si presenta in pages/chef/visualizza-ordini-chef e consente all'utente chef di visualizzare in dettaglio le comande relative a un tavolo selezionato, precedentemente scelto nella vista principale dei tavoli. L'interfaccia si presenta con un layout centrato, ordinato e responsive, dominato da un'intestazione che mostra il titolo della pagina e il numero del tavolo attualmente in lavorazione. La grafica utilizza colori personalizzati per garantire chiarezza visiva, mentre i pulsanti di azione "INIZIA LAVORAZIONE TOTALE" e "FINE LAVORAZIONE TOTALE", rispettivamente in arancione e rosso, permettono di gestire in blocco lo stato dei prodotti ordinati per l'intero tavolo.

La pagina utilizza un approccio dichiarativo e reattivo per mostrare dinamicamente i dati. Al caricamento iniziale e a intervalli regolari di 30 secondi, viene richiamato il metodo loadOrdini() che interroga il servizio OrdineService per ottenere la lista aggiornata dei prodotti ordinati associati alla prenotazione del tavolo. Questo metodo, oltre a gestire lo stato di caricamento tramite lo spinner ion-spinner, effettua controlli sulla struttura della risposta per garantire l'integrità dei dati prima di aggiornarli nella proprietà osservabile prodotti\$, tramite BehaviorSubject. In caso di errore, viene registrato un log e l'interfaccia viene ripulita dai dati precedenti. Il componente app-lista-ordini, integrato come child component, è

responsabile della visualizzazione delle singole comande, e riceve in input lo stream prodotti\$, aggiornandosi automaticamente ogni volta che i dati vengono modificati. I metodi iniziaLavorazioneTotale() e fineLavorazioneTotale() rappresentano le azioni operative principali della vista. Essi iterano attraverso i prodotti ordinati, verificano il loro stato attuale e invocano il metodo cambiaStato() sul servizio backend per modificarne il flusso operativo, rispettivamente verso “in-lavorazione” o “in-consegna”. Al termine di ciascuna operazione, viene effettuato un nuovo caricamento per sincronizzare lo stato dei dati.

La logica sottostante del file TypeScript è fortemente modulare, con netta separazione tra interfaccia e gestione dati. Il ciclo di vita del componente è gestito attraverso gli hook ngOnInit e ngOnDestroy, in cui vengono avviati o interrotti i processi temporizzati. Inoltre, il servizio TavoloService funge da contenitore temporaneo dello stato del tavolo selezionato e permette di accedere ai dati della prenotazione in corso. La struttura del componente è standalone e sfrutta i moduli di Ionic per garantire compatibilità, performance e reattività. Questa vista rappresenta un'estensione verticale dell'esperienza chef, focalizzata sulla gestione approfondita delle comande di un singolo tavolo. L'integrazione tra logica asincrona, aggiornamento reattivo dei dati e un'interfaccia chiara e interattiva, rende il componente uno strumento essenziale per la supervisione diretta da parte dello chef, perfettamente in linea con l'architettura modulare e scalabile dell'applicazione.



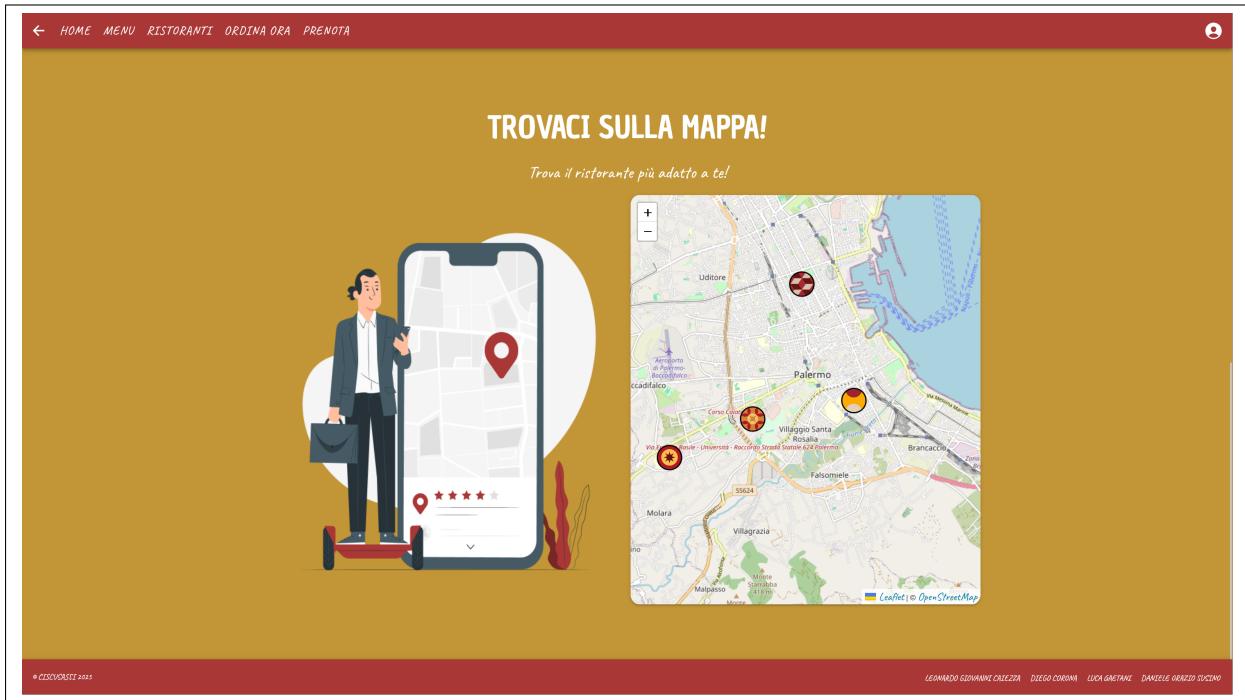
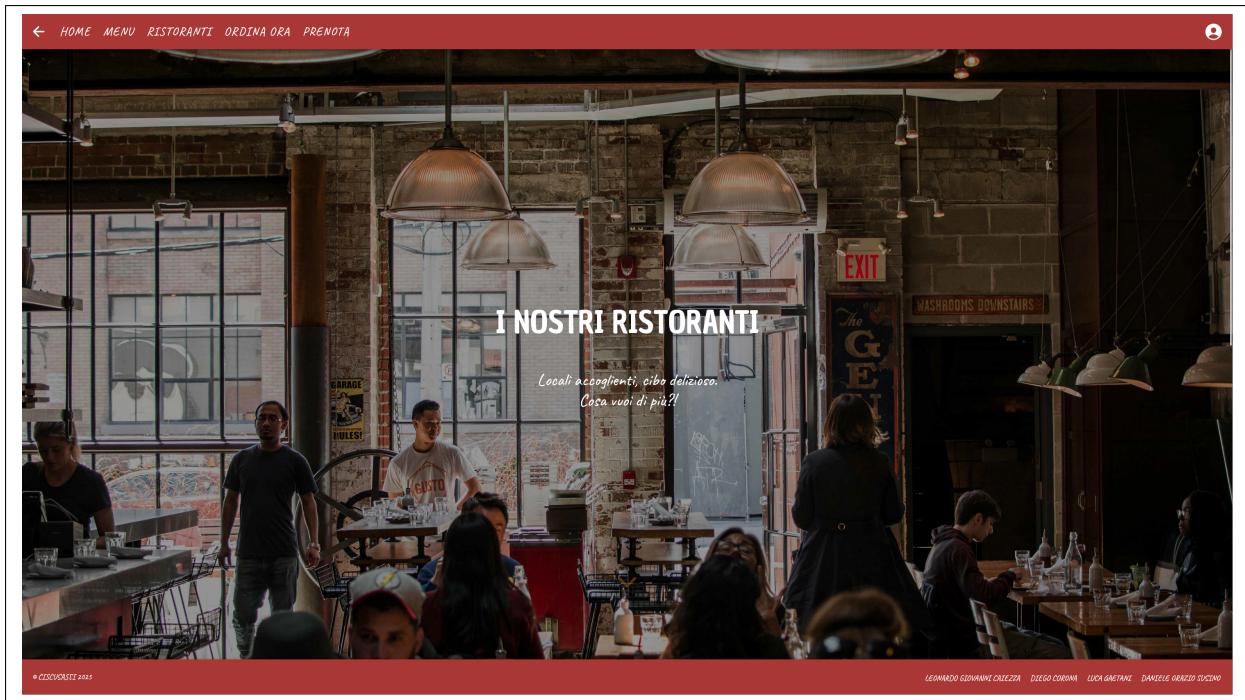


5.8 Visualizza Schermata Ristoranti

La funzionalità Visualizza Schermata Ristoranti si presenta in pages/ristoranti e consente all’utente di visualizzare la presentazione dei ristoranti. La struttura visiva è costruita con componenti standalone di Angular e Ionic, seguendo un layout moderno e responsivo. All’apertura della pagina, l’utente viene accolto da una sezione app-hero personalizzata, che fornisce un titolo evocativo e una descrizione coinvolgente, accompagnati da un’immagine di sfondo tematica. Questa intestazione centrale serve a contestualizzare l’argomento e introdurre la funzionalità della mappa. La sezione seguente è strutturata all’interno di un contenitore che include un titolo secondario e un sottotitolo stilizzati con ion-text, progettati per guidare l’utente verso l’uso della mappa. La mappa è presentata attraverso un layout ion-grid, responsivo e centrato, dove due colonne affiancano un’illustrazione rappresentativa e il contenuto interattivo.

Nel dettaglio, la colonna sinistra (visibile solo su schermi XL e superiori) mostra un’immagine illustrativa statica, mentre la colonna destra contiene una ion-card che ospita il componente app-leaflet-map. Quest’ultimo, importato come componente standalone, rappresenta l’integrazione con Leaflet.js per visualizzare i ristoranti sulla mappa. Il caricamento del CSS di Leaflet è gestito separatamente tramite link nel file HTML, garantendo il corretto rendering dell’interfaccia geografica.

La pagina risulta efficace come punto di accesso alla rete di ristoranti, fornendo una panoramica visiva semplice ma coinvolgente. Qualora si premesse su uno dei ristoranti presenti sulla mappa, si verrebbe reindirizzati alla pagina di google maps dell’indirizzo specificato. L’integrazione con la mappa interattiva consente una navigazione intuitiva, mentre l’impostazione grafica curata ne rafforza l’usabilità e l’impatto estetico complessivo. La mancanza di logica interna rende questa pagina facilmente estendibile per future funzionalità, mantenendo nel contempo una separazione chiara tra presentazione e logica operativa.



5.9 Gestione Prenotazioni

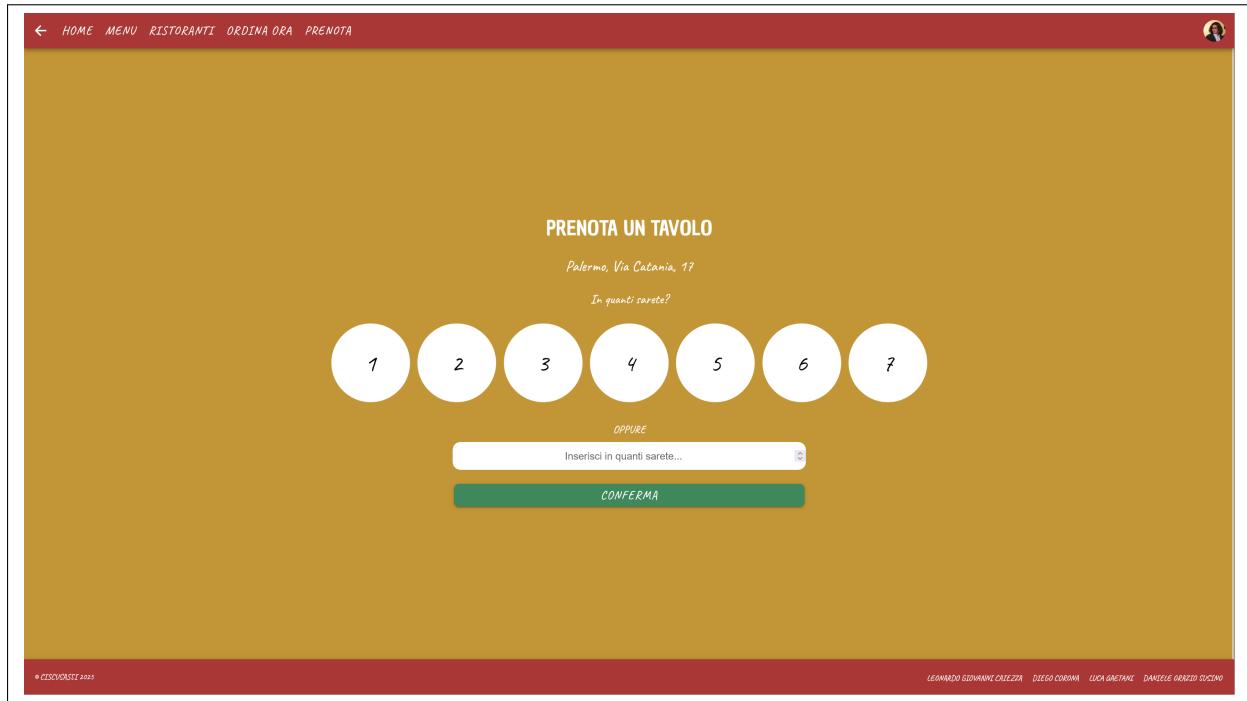
5.9.1 Imposta Numero Persone

La funzionalità Imposta Numero Persone si presenta in pages/prenotazioni/numero-persone e consente all'utente cliente di selezionare il numero di persone per una prenotazione in una specifica filiale. L'interfaccia si presenta con un design chiaro e coerente, caratterizzata da uno sfondo con colore senape e una struttura basata su ion-grid per un layout ordinato e responsivo. Al centro della pagina, un titolo in evidenza introduce la funzionalità "PRENOTA UN TAVOLO", seguito dall'indicazione dinamica della filiale selezionata, che mostra il comune e l'indirizzo, fornendo così un riferimento chiaro all'utente. In caso di caricamento dei dati, un componente ion-spinner comunica lo stato di attesa, mentre in condizioni normali l'utente può scegliere il numero di persone tramite due modalità di input: una selezione rapida tramite

pulsanti rappresentati da componenti app-numero-posti-button e un campo di input manuale numerico. La selezione rapida è visibile solo su dispositivi di dimensioni medie o superiori, mentre l'input manuale è sempre accessibile. La pagina prevede inoltre un pulsante di conferma che attiva la validazione e il salvataggio del dato selezionato.

Dal punto di vista logico, il file TypeScript associato gestisce lo stato della pagina attraverso proprietà reattive, come filiale, personeSelezzionate, inputManuale e loading. All'avvio (ngOnInit), la pagina resetta i valori di input e recupera l'identificativo della filiale tramite un servizio di prenotazione, quindi carica i dati relativi alla filiale con una chiamata asincrona al servizio FilialeService. La risposta viene gestita in modo robusto, con controlli per la presenza e validità dei dati, e l'eventuale segnalazione di errori è tracciata tramite console. La selezione delle persone è sincronizzata tra i pulsanti rapidi e l'input manuale, garantendo una UX fluida: la modifica di uno azzerà l'altro, evitando conflitti. Il metodo salvaPersone applica una logica di controllo che verifica che il numero inserito sia positivo e confronta la richiesta con la disponibilità reale dei tavoli nella filiale. In caso di insufficienza dei tavoli, viene mostrato un messaggio di errore tramite il servizio ToastController. Quando la validazione ha successo, il numero di persone viene salvato nel servizio di prenotazione e l'utente viene reindirizzato alla pagina successiva per la scelta del giorno della prenotazione.

L'approccio complessivo è caratterizzato da una separazione chiara tra la vista e la logica di controllo, con un uso consistente di componenti standalone e binding Angular. La pagina offre un'esperienza utente intuitiva, integrando feedback visivi e messaggi chiari per supportare il processo di prenotazione in modo efficace e affidabile.

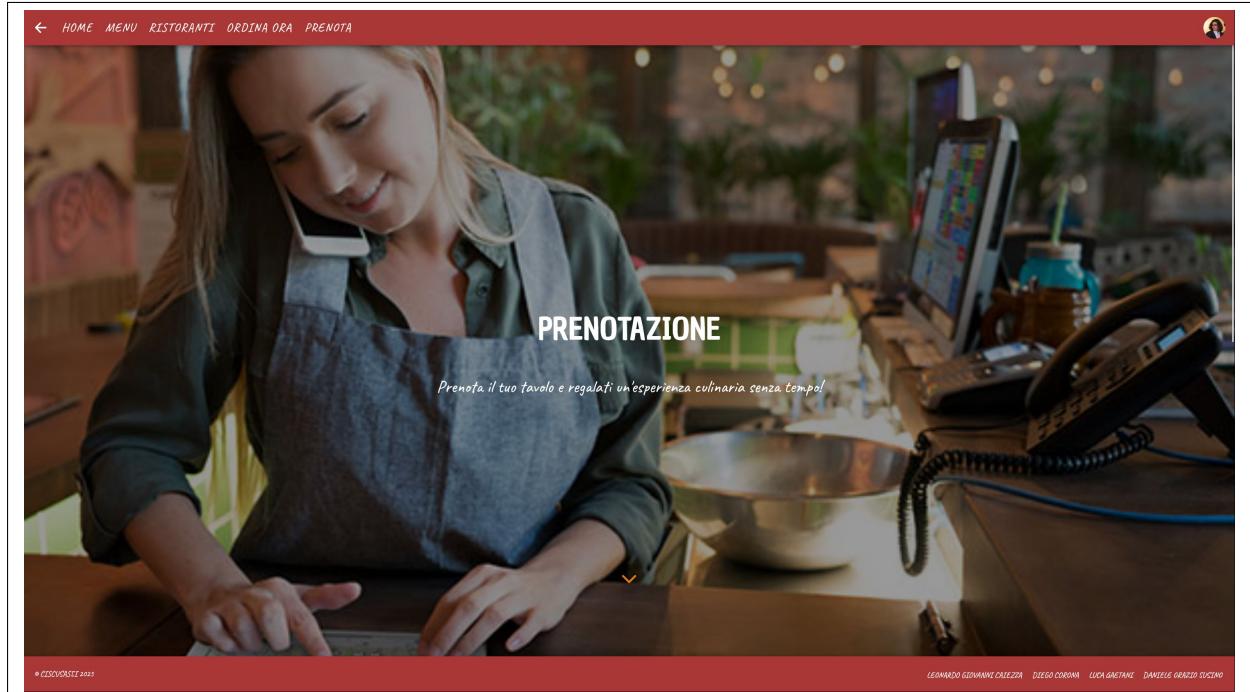


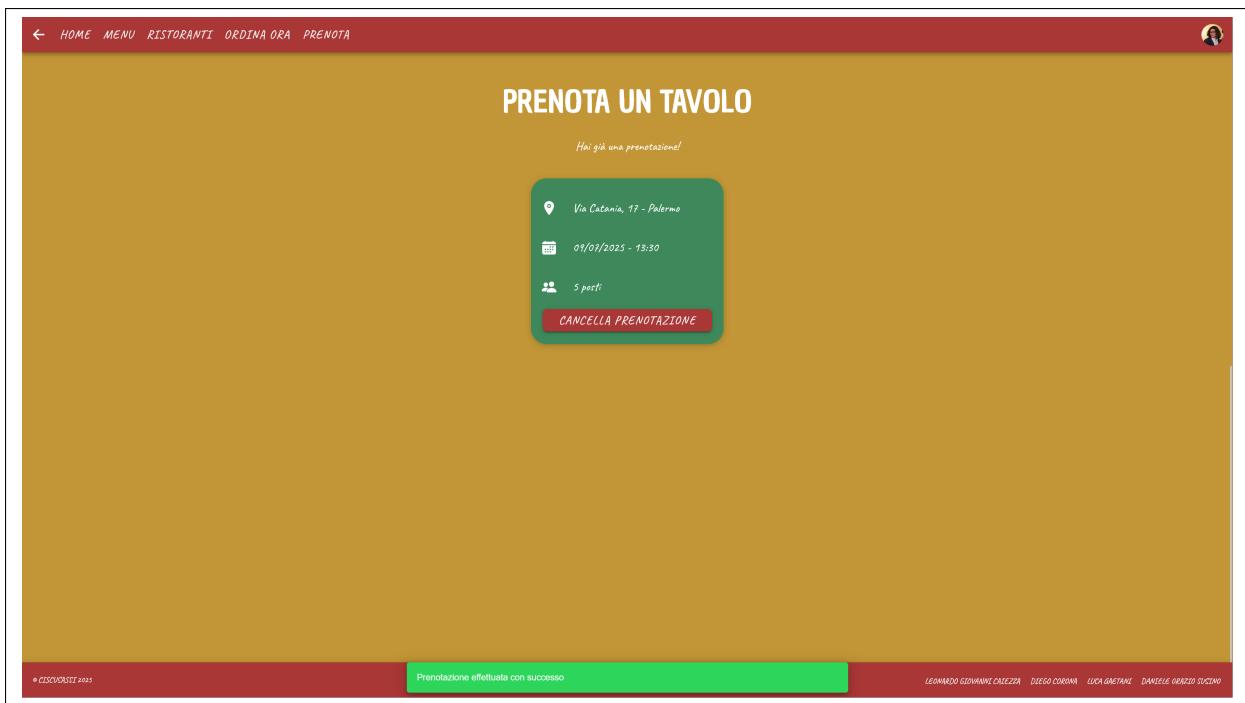
5.9.2 Prenota

La funzionalità Prenota si presenta in pages/prenotazioni/prenota e consente all'utente cliente di effettuare una prenotazione in una filiale. Il file HTML della pagina di prenotazione presenta una struttura chiara e ben organizzata, finalizzata a gestire l'esperienza utente per la prenotazione di un tavolo. La pagina è racchiusa all'interno del componente <ion-content> impostato in modalità fullscreen, per sfruttare al meglio l'area disponibile. All'inizio è presente un componente eroe, <app-hero>, che mostra un titolo e una descrizione esplicativa, accompagnati da un'immagine di sfondo che contribuisce a creare un'atmosfera coinvolgente. La sezione principale della pagina è definita da un contenitore con sfondo color senape, in cui viene gestita la visualizzazione dinamica del contenuto in base allo stato di caricamento delle prenotazioni.

Nel caso in cui i dati siano stati caricati correttamente, la pagina verifica se esistono prenotazioni attive per l'utente. Se sì, viene mostrato un messaggio che informa l'utente della presenza di una prenotazione, seguito da una lista di card che rappresentano ciascuna prenotazione tramite il componente <app-prenotazione-card>. Questa lista è organizzata all'interno di una griglia responsive che assicura una buona resa visiva su diversi dispositivi. Nel caso contrario, cioè quando non sono presenti prenotazioni, l'utente viene invitato a scegliere una filiale tramite una barra di ricerca testuale e una lista filtrata di filiali, dove ogni filiale è rappresentata da un componente <app-filiale-card>. Accanto alla lista di filiali, su dispositivi con schermi grandi, viene mostrata una mappa interattiva integrata tramite il componente <app-leaflet-map>, che aiuta a visualizzare geograficamente le sedi disponibili. Questo approccio permette all'utente di selezionare rapidamente la filiale più comoda e proseguire nella prenotazione direttamente dalla mappa, cliccando sulla filiale di interesse. Durante il caricamento dei dati, invece, la pagina mostra un indicatore di caricamento con un componente spinner, comunicando all'utente che l'operazione è in corso.

Il file TypeScript associato alla pagina di prenotazione gestisce in modo efficiente la logica dell'interfaccia e il ciclo di vita del componente. Tramite la dichiarazione di numerose proprietà reattive, il file tiene traccia sia delle liste di filiali e prenotazioni, sia degli stati di caricamento e di errore. All'interno del metodo ion-ViewWillEnter, che viene eseguito ogni volta che la pagina sta per diventare visibile, vengono attivate due chiamate asincrone: una per caricare tutte le filiali disponibili tramite il servizio FilialeService, e l'altra per recuperare le prenotazioni attive dell'utente tramite il PrenotazioneService. Prima di procedere, viene inoltre svuotata qualsiasi prenotazione in corso per garantire uno stato pulito. Il caricamento delle filiali avviene con la chiamata a GetSedi, la cui risposta viene gestita con attenzione per verificare il successo dell'operazione e l'effettiva presenza di dati validi. In caso di errore, viene impostato un flag per segnalare lo stato di errore e per interrompere il caricamento. Per quanto riguarda le prenotazioni, una volta ricevuti i dati dal backend, viene effettuato un filtro che esclude le prenotazioni già scadute, basandosi sulla conversione della stringa della data e ora in oggetto Date. Tale parsing avviene in modo robusto, con controlli per assicurarsi che la stringa rispetti il formato previsto e che la conversione generi una data valida. L'interazione con la barra di ricerca è gestita tramite il metodo onFiltraFilialiChange, che aggiorna dinamicamente la lista filtrata di filiali in base al testo inserito dall'utente, migliorando l'esperienza di ricerca. Quando l'utente seleziona una filiale, il metodo salvaFiliale memorizza l'identificativo scelto nel servizio di prenotazione e reindirizza l'utente alla pagina successiva per la selezione del numero di persone.





5.9.3 Scelta Giorno

La funzionalità Scelta Giorno si presenta in pages/prenotazioni/scelta-giorno e consente all'utente cliente di selezionare il giorno e la fascia oraria per la prenotazione di un tavolo in una specifica filiale. La pagina si presenta con un'interfaccia pulita e reattiva, costruita con componenti Ionic standalone e strutturata tramite ion-grid per mantenere un layout ordinato e responsivo. Al caricamento, la pagina recupera l'identificativo della filiale dal servizio di prenotazione e inizializza la data selezionata, applicando una logica particolare per il giorno di martedì (che è il giorno di chiusura), che imposta la data al giorno successivo, mentre negli altri casi la data è impostata a oggi. Successivamente, vengono caricati i dati relativi alla filiale tramite una chiamata asincrona al servizio FilialeService, che viene gestita con attenzione a eventuali errori, segnalati tramite console.

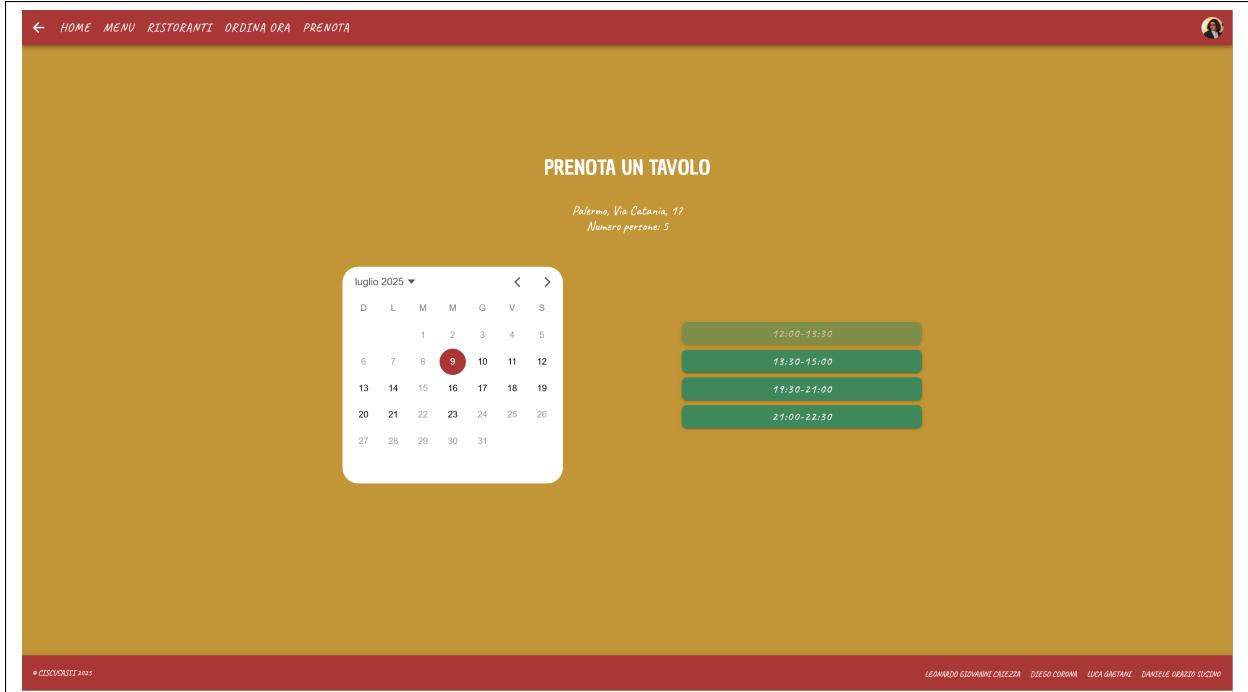
La logica di controllo prevede che le fasce orarie non disponibili siano disabilitate nell’interfaccia utente, garantendo così un’esperienza di prenotazione chiara e affidabile. La validazione delle date impedisce la selezione del martedì e limita la scelta a un intervallo massimo di due settimane. Al cambio della data, la pagina effettua un controllo sulle prenotazioni già in uso in modo da aggiornare lo stato delle fasce orarie disponibili. Viene inoltre prevista una funzione di controllo per escludere la possibilità di prenotare fasce orarie già passate nel caso in cui la data selezionata sia quella odierna. Il metodo per la conferma della prenotazione verifica che il cliente non abbia prenotazioni future attive, mostrando un messaggio di errore in caso contrario, e tenta di salvare la nuova prenotazione, notificando all’utente l’esito dell’operazione tramite toast informativi. La pagina integra una robusta gestione degli errori e un feedback visivo continuo per accompagnare l’utente durante il processo di prenotazione. La separazione tra la logica e la vista è mantenuta con un uso appropriato dei binding Angular e dei servizi dedicati.

Il componente HTML della pagina SceltaGiornoPage è costruito con ion-content che occupa tutto lo schermo e presenta uno sfondo color senape, coerente con il design generale dell’applicazione. La struttura è definita tramite ion-grid che ospita una serie di righe e colonne, garantendo un layout ordinato e adattabile a diverse dimensioni di dispositivo. In cima alla pagina, un titolo centrale e ben visibile introduce la funzionalità “PRENOTA UN TAVOLO”, seguito dall’indicazione dinamica della filiale selezionata che mostra il comune e l’indirizzo, accompagnati dal numero di persone selezionate. Durante il caricamento dei dati, la pagina visualizza un componente ion-spinner al centro, comunicando in modo chiaro lo stato di attesa. In condizioni normali, la pagina presenta un ion-datetime per la selezione della data, con restrizioni personalizzate per escludere il martedì e limitare la scelta a due settimane future. Alla destra della selezione data, o sotto di essa su dispositivi più piccoli, si trovano i pulsanti per scegliere rapidamente una fascia oraria di prenotazione. Questi pulsanti sono disabilitati in base alla disponibilità delle fasce e al superamento dell’orario. L’interfaccia è progettata per offrire un’esperienza utente semplice e immediata, con pulsanti ben visibili e messaggi chiari, integrando una gestione dinamica dello stato tramite Angular e Ionic, che assicura aggiornamenti in tempo reale e feedback appropriati durante l’interazione dell’utente con la pagina.

Il file TypeScript associato alla pagina “Scelta Giorno” gestisce la logica dell’interfaccia attraverso una classe component Angular con decoratore standalone, che integra vari moduli e componenti Ionic per realizzare una UI reattiva e accessibile. Al momento dell’inizializzazione (ngOnInit), il componente recupera l’identificativo della filiale dal servizio di prenotazione, imposta una data di default tenendo conto di una particolare regola (se il giorno corrente è martedì, viene selezionato il giorno successivo), e procede al caricamento delle informazioni della filiale tramite il servizio dedicato FilialeService, che effettua una chiamata asincrona per ottenere i dati delle sedi. La gestione della risposta del servizio FilialeService è effettuata con controlli robusti per verificare la presenza e validità dei dati, impostando eventuali flag di errore per la corretta comunicazione dello stato all’utente. Il metodo privato loadFiliale attiva uno stato di caricamento e resetta gli errori prima di eseguire la richiesta, mentre handleResponse filtra la lista delle filiali per identificare quella corrispondente all’id recuperato, aggiornando lo stato del componente e richiamando la funzione alCambioData per aggiornare la disponibilità dei tavoli. La funzione noMartediEMaxDueSettimane consente di abilitare solo le date valide per la prenotazione, escludendo i martedì e limitando la scelta a un intervallo di due settimane a partire dal giorno corrente. La data selezionata è formattata in formato standard ISO tramite formatDateToYYYYMMDD, che implementa controlli per gestire eventuali dati non validi. La logica di aggiornamento alla modifica della data selezionata prevede il reset delle fasce orarie disponibili e una chiamata al servizio prenotazioneService per ottenere il numero di tavoli già occupati nelle varie fasce orarie, elaborando quindi la disponibilità residua in base al numero totale di tavoli della filiale e al numero di tavoli richiesti per la prenotazione. Tale aggiornamento consente di abilitare o disabilitare i pulsanti corrispondenti alle fasce orarie di prenotazione, tenendo conto anche del fatto che fasce già trascorse nella giornata corrente risultino disabilitate tramite la funzione isFasciaPassata. Per garantire una corretta gestione delle prenotazioni, la pagina carica preventivamente le prenotazioni future del cliente tramite una Promise in caricaPrenotazioniCliente, filtrando quelle non ancora scadute e memorizzandole nello stato locale. Prima di confermare una nuova prenotazione, il metodo confermaPrenotazione verifica se sono già presenti prenotazioni attive, e in caso positivo informa l’utente tramite un Toast di errore, bloccando l’operazione. Se non ci sono conflitti, procede con la richiesta di prenotazione tramite il servizio dedicato, gestendo gli esiti con messaggi di feedback chiari: in caso di successo l’utente viene notificato e reindirizzato alla pagina principale di prenotazione, mentre eventuali errori, inclusa la mancanza di tavoli disponibili, vengono comunicati tramite Toast di colore rosso. Il metodo privato parseDateTime si occupa di convertire le stringhe di data e ora in oggetti Date,

con controlli di validità per prevenire errori dovuti a formati non corretti, supportando così la logica di filtro delle prenotazioni future.

L'architettura del componente si caratterizza quindi per una chiara separazione tra la gestione dello stato, la comunicazione asincrona con i servizi backend e il supporto all'interfaccia utente, garantendo una UX fluida, reattiva e informativa nel processo di scelta e conferma della prenotazione, integrando efficacemente controlli di validità, gestione degli errori e feedback visivi.



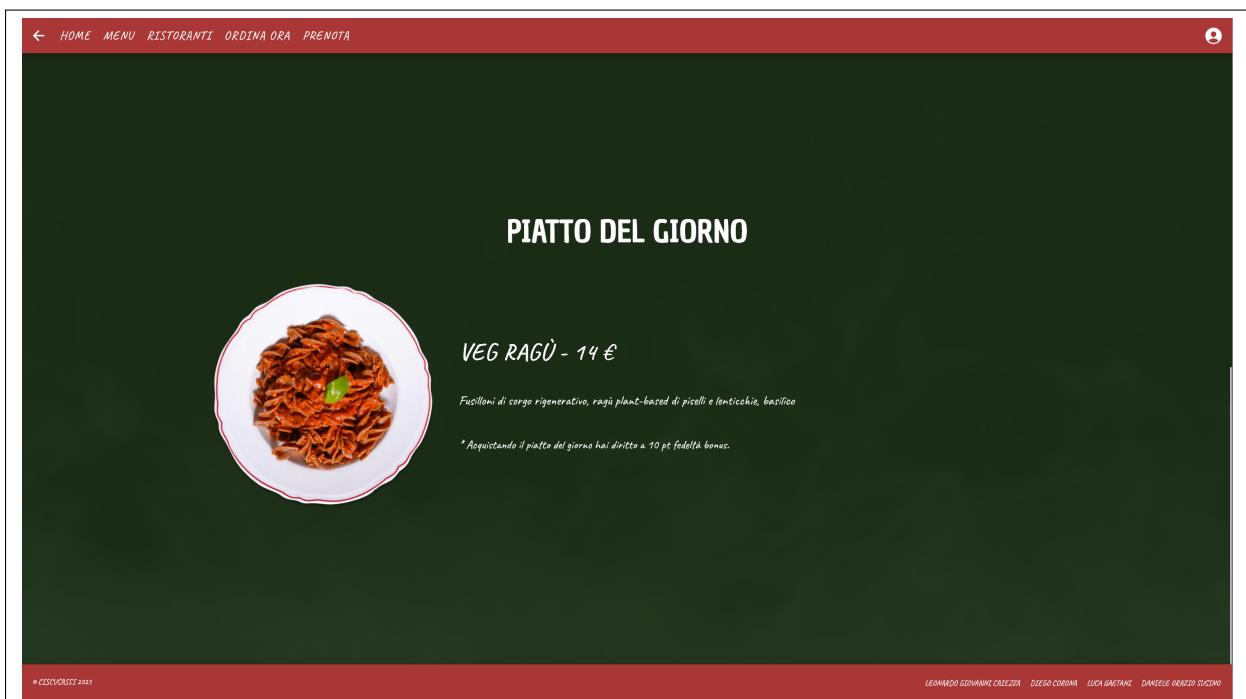
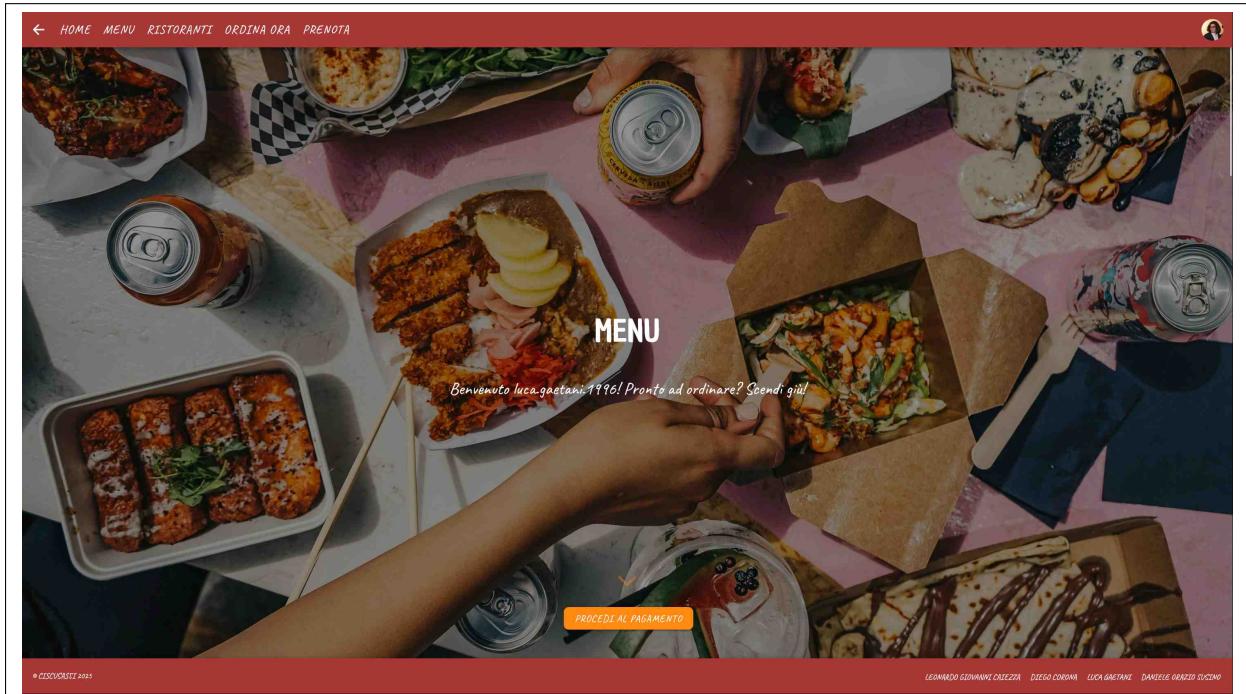
5.10 Gestione Ordinazione

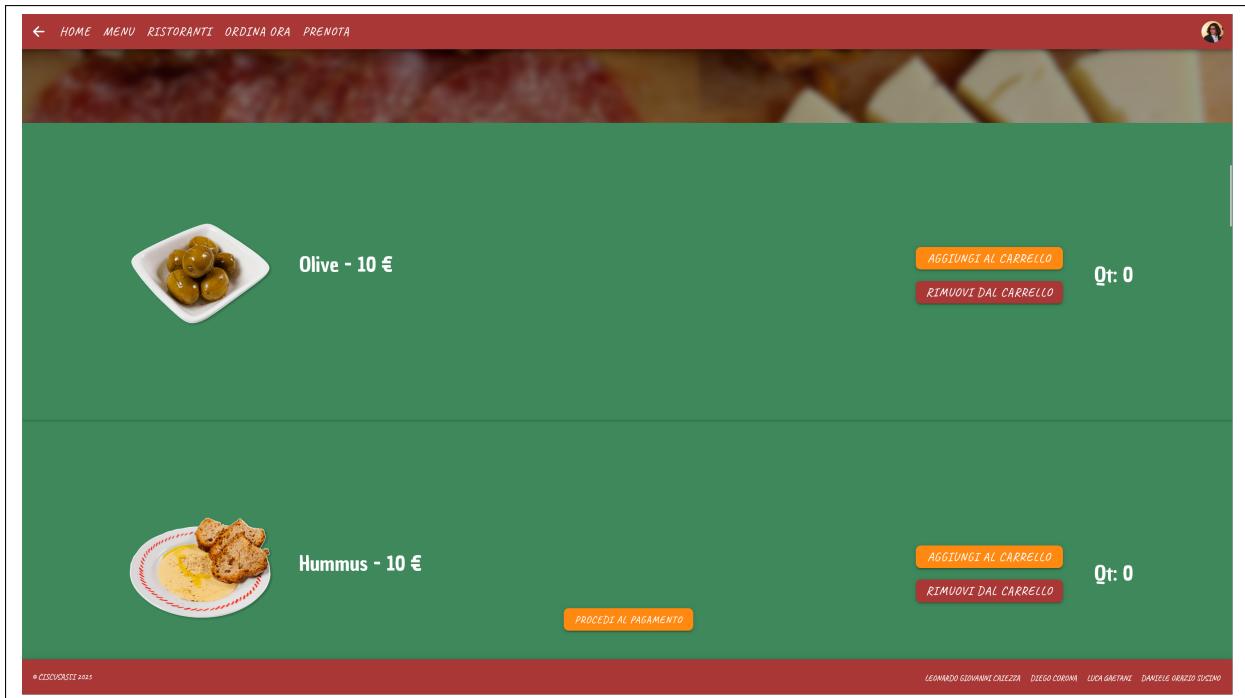
5.10.1 Visualizza Menu Asporto

La funzionalità Visualizza Schermata Menu Asporto si presenta in pages/ordinazioni/menu-asporto, che funge da interfaccia principale per consentire agli utenti autenticati di visualizzare e ordinare piatti per l'asporto. All'interno del template HTML, l'intero contenuto è racchiuso in un elemento ion-content con la proprietà fullscreen attiva, garantendo l'uso completo dello schermo disponibile per una migliore esperienza utente. La pagina è introdotta dal componente app-hero, che presenta il titolo "MENU" e una descrizione dinamica personalizzata con il nome dell'utente autenticato, ottenuto dal servizio di autenticazione. L'atmosfera visiva è arricchita da uno sfondo specificato tramite la proprietà backgroundURL impostata su "MenuAsportoBackground.jpg". Segue il componente app-piatto-del-giorno, che evidenzia la specialità quotidiana del ristorante. Questo elemento rappresenta un incentivo promozionale, poiché suggerisce ai clienti di ordinare il piatto del giorno per ottenere eventuali vantaggi aggiuntivi. Più sotto, il componente app-lista-menu elenca tutti i piatti disponibili per l'ordinazione, con la proprietà isOrdinazione impostata su true, indicando che la schermata corrente consente agli utenti di effettuare un ordine. Alla fine della pagina, un pulsante ion-button consente di avviare la procedura di pagamento attraverso il metodo checkTotale().

Dal punto di vista TypeScript, la classe MenuAsportoPage è definita come standalone e importa moduli personalizzati HeroComponent, PiattoDelGiornoComponent e ListaMenuComponent. Inoltre, viene incluso IonButton per l'interazione utente. All'interno della logica della pagina, viene iniettato il servizio CarrelloService per accedere ai prodotti selezionati nel carrello e calcolare il totale dell'ordine, mentre il ToastController è utilizzato per mostrare messaggi informativi. Il metodo checkTotale() controlla se il valore complessivo dei prodotti nel carrello supera la soglia minima di 30 euro, necessaria per procedere con la consegna: se il requisito non è soddisfatto, viene mostrato un messaggio di avviso; in caso contrario, l'utente viene reindirizzato alla pagina di pagamento. Il metodo ngOnInit() inizializza la proprietà nomeUtente recuperando il nome dell'utente autenticato tramite AuthenticationService, favorendo

un'interazione personalizzata e contestuale. L'intera struttura conferma un'organizzazione modulare e scalabile, in cui la logica di presentazione e la gestione dei dati rimangono ben separate per garantire facilità di manutenzione e futura espandibilità.





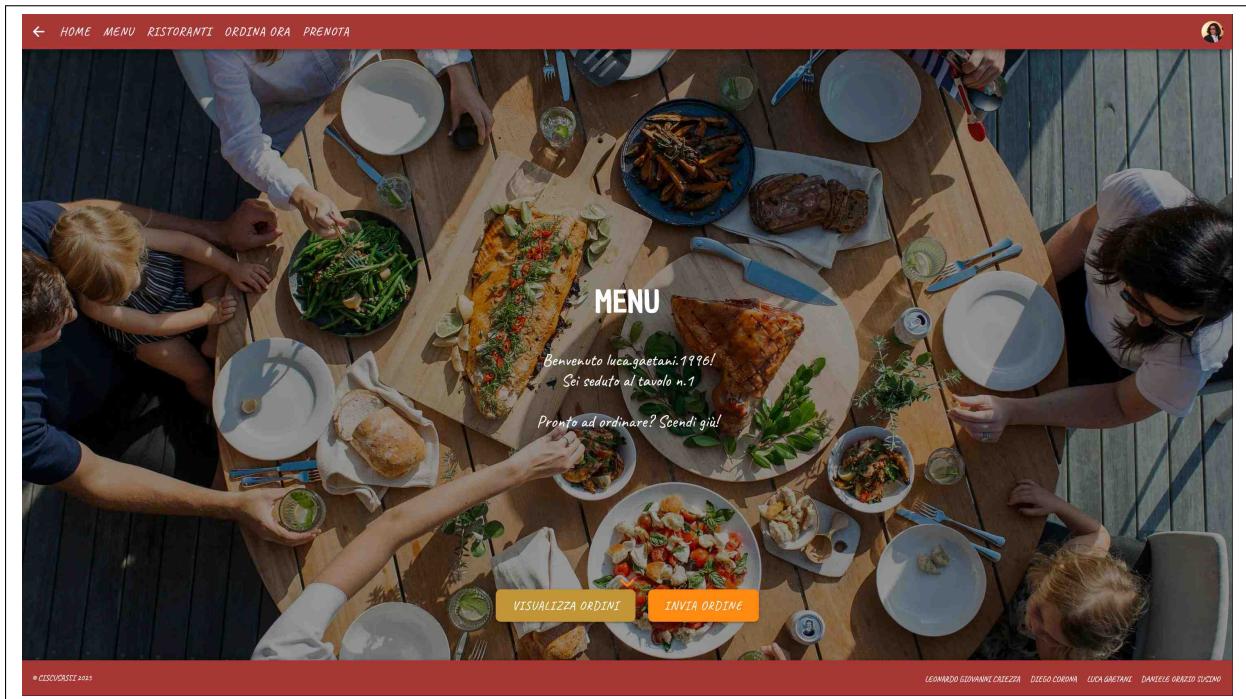
5.10.2 Visualizza Menu Tavolo

La funzionalità Visualizza Schermata Menu Tavolo in pages/menu-tavolo rappresenta l’interfaccia principale per gli utenti che stanno ordinando dal tavolo di un ristorante. All’interno del template HTML, l’intero contenuto è racchiuso in un elemento ion-content con la proprietà fullscreen abilitata, assicurando un’esperienza a schermo intero. L’apertura visiva è affidata al componente app-hero, che presenta il titolo “MENU” e una descrizione dinamica che include il nome dell’utente e il numero del tavolo associato alla prenotazione, ottenuti attraverso i servizi di autenticazione e gestione tavoli. Lo sfondo è personalizzato tramite la proprietà backgroundURL impostata su “MenuTavoloBackground.jpg”, contribuendo a contestualizzare l’esperienza. Segue il componente app-piatto-del-giorno, che evidenzia la specialità quotidiana, e il componente app-lista-menu, che mostra tutti i piatti disponibili, con la proprietà isOrdinazione impostata su true, indicando che la pagina è attiva per la selezione di prodotti. In fondo alla schermata, due pulsanti gestiti tramite eventi click consentono di visualizzare lo stato degli ordini attivi (visualizzaOrdini()) o procedere all’invio dell’ordine selezionato (checkTotale()). I pulsanti sono stilizzati con colori distintivi e layout ottimizzato per l’interazione da dispositivi mobili.

Dal punto di vista TypeScript, il componente importa moduli personalizzati HeroComponent, PiattoDelGiornoComponent e ListaMenuComponent. La logica principale è distribuita in vari metodi e si basa su una stretta collaborazione con diversi servizi iniettati tramite dependency injection: AuthenticationService per il recupero dell’utente, CarrelloService per la gestione dei prodotti selezionati, OrdineService per l’invio e recupero degli ordini e TavoloService per la gestione del numero tavolo e prenotazione corrente. Durante l’inizializzazione (ngOnInit), viene richiamato ngViewWillEnter(), che popola i dati iniziali tra cui nome utente, numero tavolo e un eventuale ordine già esistente associato all’utente e alla prenotazione attiva. Il metodo checkTotale() verifica se l’utente ha già pagato e calcola il totale del carrello; se il carrello è vuoto o non è presente una prenotazione valida, viene mostrato un messaggio di avviso. Se necessario, viene creato un nuovo ordine e i prodotti selezionati vengono associati ad esso. L’operazione si conclude svuotando il carrello e notificando l’utente con un messaggio di successo. Il metodo visualizzaOrdini() consente di accedere allo stato attuale dell’ordine. Se l’utente ha già pagato, oppure non ha ancora effettuato un ordine, viene mostrata una notifica contestuale. In caso contrario, viene eseguita una chiamata al backend per recuperare i prodotti associati al numero d’ordine, e l’utente viene reindirizzato alla pagina di visualizzazione ordini.

L’intera struttura del componente promuove un approccio modulare, fortemente orientato alla separazione delle responsabilità e al riuso di servizi e componenti. La logica di presentazione è ben distinta dalla logica di business e ogni operazione è accompagnata da un feedback visuale tramite toast, migliorando

significativamente l'esperienza utente. Questa architettura si presta a essere facilmente estesa o modificata per supportare funzionalità future.



← HOME MENU RISTORANTI ORDINA ORA PRENOTA

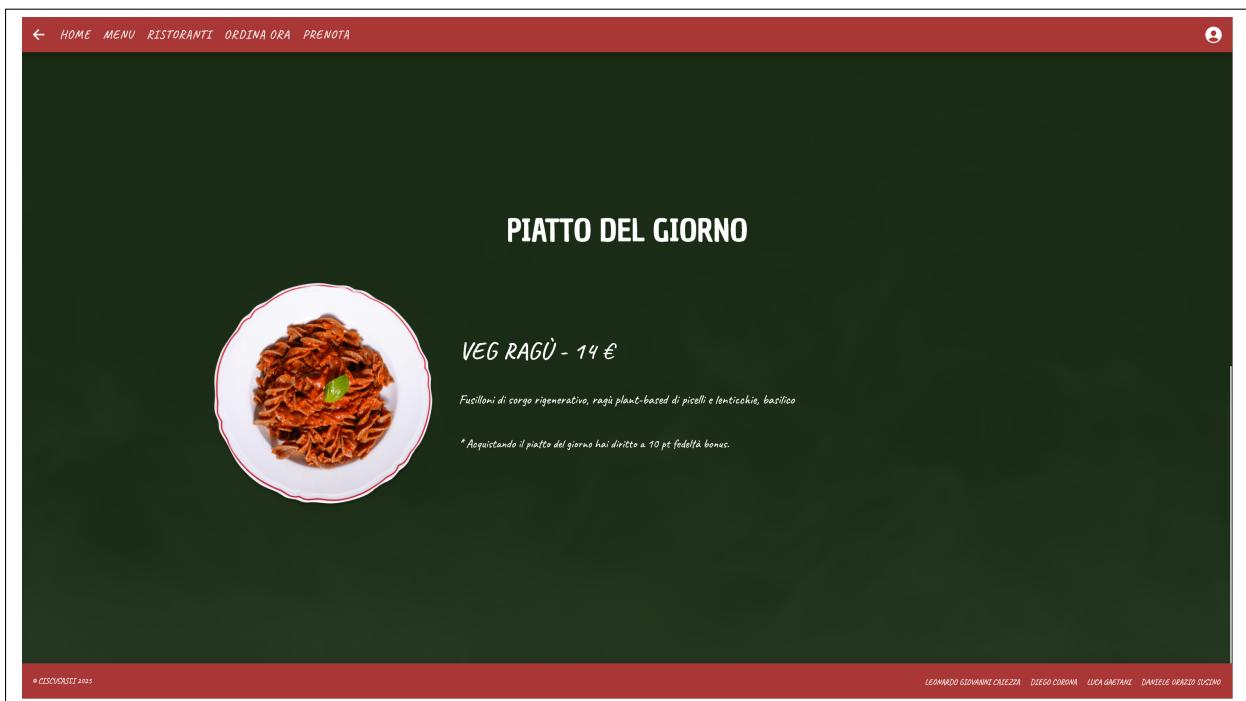
MENU

Benvenuto luca.gastani.1996!
Sei seduto al tavolo n.1

Pronto ad ordinare? Scendi giù!

VISUALIZZA ORDINI INVIA ORDINE

© CICCHERONI 2020 LEONARDO GIOVANNI CALEZZA DIEGO CORINA LUCA GASTANI DANIELE ORAZIO SUCINO



← HOME MENU RISTORANTI ORDINA ORA PRENOTA

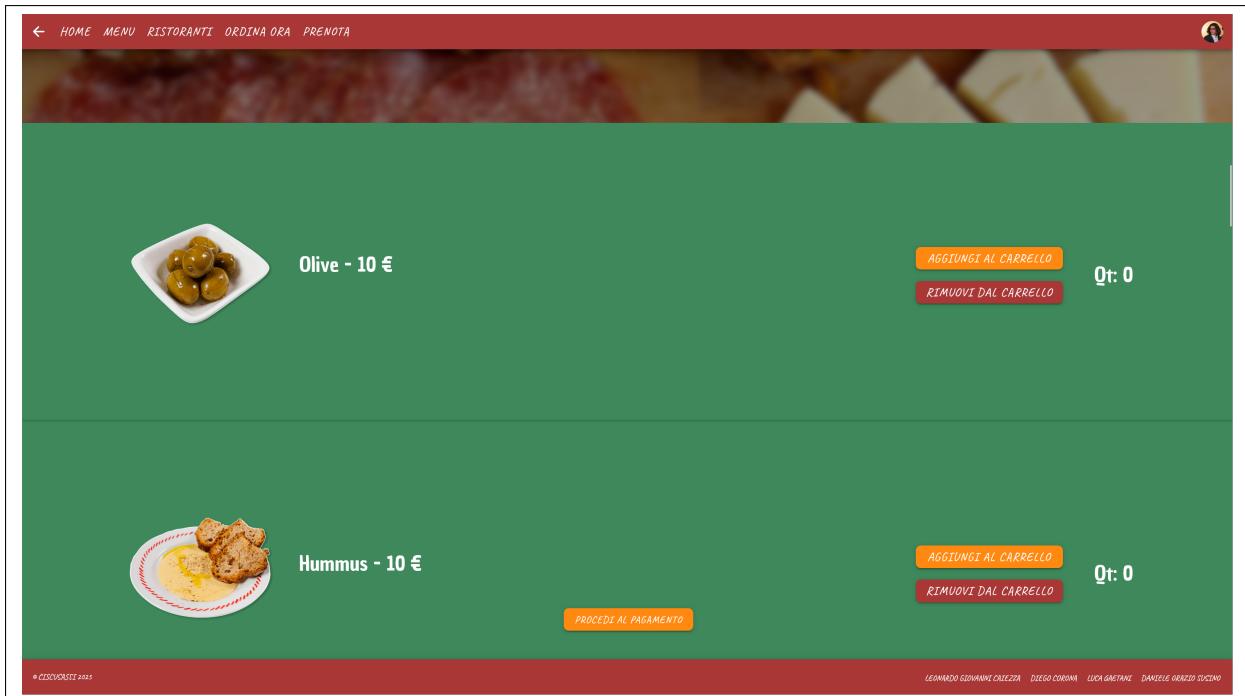
PIATTO DEL GIORNO

VEG RAGÙ - 14 €

Fusilli di sorgo rigenerativo, ragù plant-based di piselli e lenticchie, basilico

* Acquistando il piatto del giorno hai diritto a 10 pt fedeltà bonus.

© CICCHERONI 2020 LEONARDO GIOVANNI CALEZZA DIEGO CORINA LUCA GASTANI DANIELE ORAZIO SUCINO



5.10.3 Ordina al Tavolo

La funzionalità Ordina al Tavolo in pages/ordinazioni/ordina-al-tavolo definisce il punto d'ingresso per gli utenti che desiderano ordinare dal proprio tavolo in un ristorante tramite identificazione con torretta e codice OTP. All'interno del template HTML, tutto il contenuto è encapsulato in un elemento ion-content con la proprietà fullscreen attivata, garantendo una visualizzazione immersiva. La struttura centrale è affidata a un ion-card, posizionato in modo centrato all'interno di una griglia ion-grid, e personalizzato con uno sfondo colorato e testi stilizzati per migliorare la leggibilità.

Il card introduce la schermata con un titolo (“INFORMAZIONI DEL TAVOLO”) e due brevi descrizioni: una per istruire l’utente sull’inserimento del numero torretta e una seconda per la OTP. Il numero torretta viene acquisito tramite un componente ion-input, mentre l’OTP viene inserita attraverso il componente ion-input-otp, configurato per accettare stringhe alfanumeriche lunghe 6 caratteri. Entrambi i campi sono gestiti tramite data binding bidirezionale ([ngModel]), consentendo un tracciamento reattivo dei valori inseriti. Il pulsante “PROCEDEI”, situato alla fine del card, è attivo solo quando entrambi i campi sono validi e l’OTP è lunga esattamente sei caratteri. Alla pressione del pulsante, viene eseguito il metodo onProceedClick(), che rappresenta il cuore della logica del componente. Il comportamento del bottone è inoltre vincolato a condizioni dinamiche di validazione e presenta un feedback visivo (classe errore) in caso di input OTP non conforme. Dal punto di vista TypeScript, il componente OrdinaAlTavoloPage è dichiarato come standalone e importa moduli essenziali come IonContent, IonInput, IonInputOtp, IonButton, FormsModule, CommonModule, RouterModule, oltre ai servizi specifici del dominio: TavoloService, PrenotazioneService, e ToastController. I due campi del form, otp e numeroTorretta, sono inizializzati e gestiti localmente, mentre la funzione ngViewWillEnter() viene richiamata all'avvio (ngOnInit) per resettare eventuali stati precedenti e assicurare un contesto pulito a ogni accesso alla pagina. La logica del metodo onProceedClick() esegue diverse validazioni in sequenza. Dopo aver verificato la presenza dei dati, viene controllato il formato dell’OTP mediante espressione regolare. Successivamente, viene determinata la fascia oraria valida corrente per la prenotazione, scegliendo tra intervalli orari prestabiliti. Se viene trovata una fascia compatibile (in corso o entro 10 minuti dall'inizio), viene composta una stringa di data e ora da usare per le verifiche successive. In caso contrario, viene notificato l’utente con un messaggio di avviso. Quando la fascia è valida, viene invocato il metodo checkOtp() del PrenotazioneService, che invia i dati al backend per la verifica dell’OTP. Se la risposta è positiva, i dati della prenotazione vengono salvati tramite TavoloService e l’utente viene reindirizzato alla pagina successiva (/menu-tavolo). Se invece l’OTP è errata o la verifica fallisce, viene mostrato un messaggio di errore. Tutte le operazioni sono accompagnate da notifiche toast, che forniscono un riscontro immediato all’utente.

L’intera architettura del componente è orientata alla chiarezza e alla modularità: l’interfaccia utente è ben

separata dalla logica applicativa, ogni servizio ha una responsabilità chiara e i feedback visivi migliorano l’usabilità. Il design si presta a estensioni future come la personalizzazione del layout o l’introduzione di metodi alternativi di autenticazione.

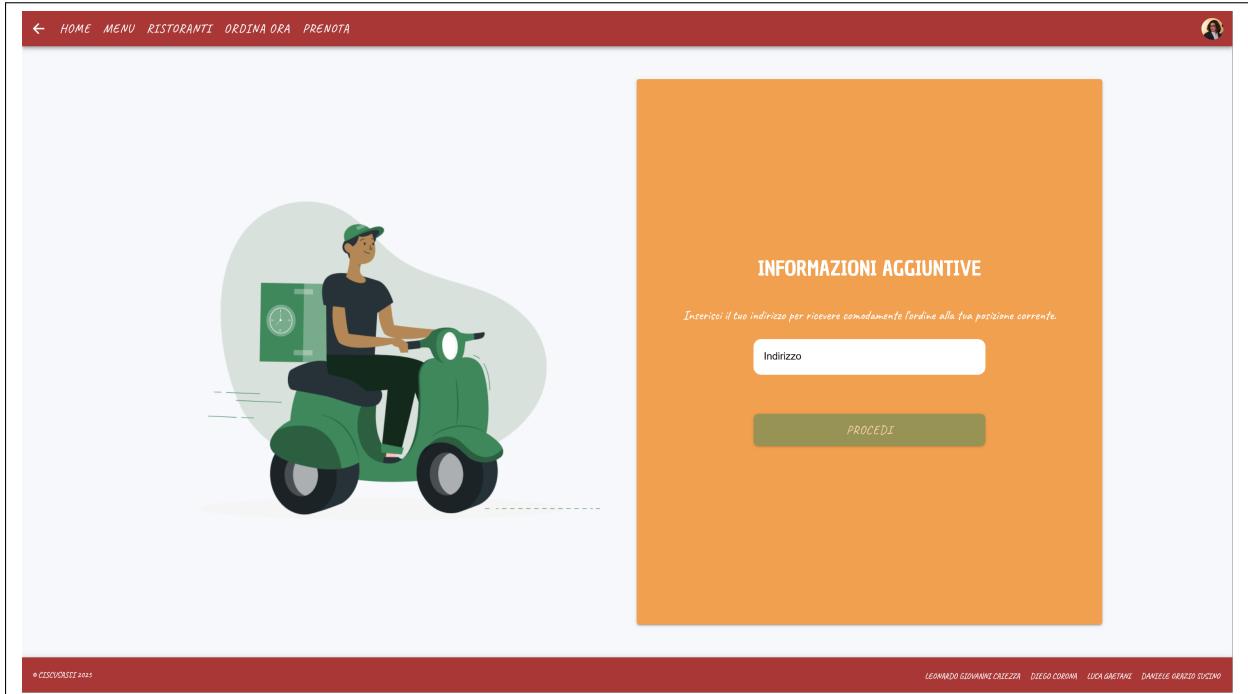


5.10.4 Ordina Asporto

La funzionalità Ordina Asporto in pages/ordinazioni/ordina-asporto rappresenta il punto di contatto tra l’utente e il sistema per effettuare un ordine da ricevere comodamente presso la propria posizione. L’interfaccia utente è racchiusa all’interno di un elemento ion-content con fullscreen abilitato, garantendo così un’esperienza a schermo intero e un design pulito. La struttura della pagina è organizzata tramite una griglia ion-grid che suddivide la schermata in due colonne principali: la colonna sinistra, visibile solo su schermi grandi, ospita un’illustrazione grafica rappresentativa, mentre la colonna destra contiene un ion-card che funge da contenitore per il form di inserimento indirizzo e l’interazione utente. Il card, caratterizzato da uno sfondo color “salmone”, presenta un titolo ben evidente e una descrizione sintetica che istruisce l’utente a inserire l’indirizzo per la consegna dell’ordine. L’input per l’indirizzo è gestito tramite un componente ion-input dotato di binding bidirezionale con la proprietà testoRicerca, che aggiorna dinamicamente il valore mentre l’utente digita. La ricerca dell’indirizzo è ottimizzata attraverso un meccanismo di debounce che ritarda la chiamata al servizio di geocoding per ridurre il numero di richieste. I risultati vengono mostrati in un elenco di suggerimenti ottenuti tramite l’API di TomTom, da cui l’utente può selezionare l’indirizzo desiderato. Alla selezione, vengono memorizzate le coordinate geografiche associate, abilitando così il calcolo della filiale più vicina in termini di tempo di viaggio.

Dal punto di vista TypeScript, il componente è dichiarato come standalone e importa moduli chiave di Ionic e Angular, oltre ai servizi dedicati alla gestione delle filiali, del carrello e alla navigazione. La logica prevede innanzitutto il caricamento delle filiali disponibili all’avvio della pagina. Quando l’utente inserisce l’indirizzo, la ricerca viene inviata con un debounce per evitare chiamate ridondanti. Se la ricerca è già stata effettuata, i risultati sono recuperati da una cache locale per migliorare la reattività. L’API di TomTom fornisce i risultati di geocoding, che sono visualizzati come suggerimenti cliccabili. Quando l’indirizzo è selezionato, il sistema esegue un calcolo di routing per determinare quale filiale è raggiungibile nel minor tempo possibile, utilizzando l’API di routing di TomTom per ottenere distanza e durata del viaggio. Questa informazione è fondamentale per valutare se il luogo scelto rientra nelle condizioni di servizio. L’utente può quindi procedere all’ordine solo se la distanza è compatibile con i limiti prefissati, altrimenti riceve una notifica visiva tramite toast che informa della lontananza e impedisce la navigazione successiva. Nel caso di esito positivo, i dati della filiale, dell’indirizzo selezionato e del tempo stimato vengono salvati in un servizio dedicato, e l’utente viene reindirizzato alla pagina del menu per asporto.

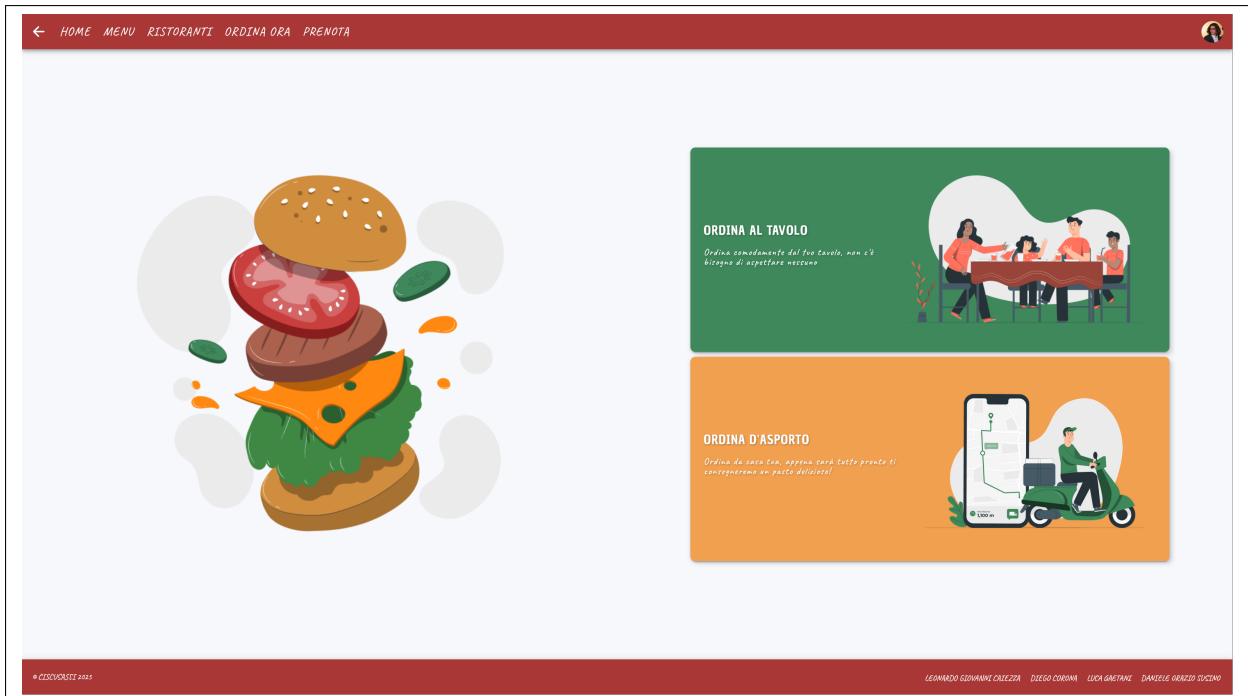
Il pulsante “PROCEDE” è dinamicamente abilitato solo quando l’indirizzo è stato selezionato e le coordinate sono disponibili. Prima di navigare, viene anche svuotato il carrello per garantire un contesto pulito per il nuovo ordine. L’intero flusso si caratterizza per un’esperienza utente fluida e assistita, dove feedback visivi e caching migliorano l’efficienza e la reattività, e dove la separazione delle responsabilità tra interfaccia, servizi di backend e logica di routing rende il codice facilmente manutenibile e estensibile.



5.10.5 Ordina Ora

La funzionalità Ordina Ora si presenta in pages/prenotazioni/ordina-ora e consente all’utente di scegliere tra due modalità di ordinazione: al tavolo e d’asporto. Il template HTML utilizza un elemento ion-content con la proprietà fullscreen attivata e una classe personalizzata per mantenere una struttura fissa e centrata. All’interno di un ion-row configurato per l’allineamento verticale e orizzontale, la pagina si suddivide in due colonne principali: la prima, visibile solo su schermi extra-large, ospita un’immagine rappresentativa di un panino, mentre la seconda contiene due pulsanti di navigazione che indirizzano rispettivamente alle pagine di ordinazione al tavolo e di ordinazione da asporto. I pulsanti sono realizzati tramite ion-button con stili distintivi di colore e classi per personalizzare l’aspetto e la disposizione dei contenuti. Ciascun bottone presenta un contenuto strutturato in due parti: un blocco testuale con titolo e descrizione, formattati per mantenere un testo chiaro e leggibile con colori a contrasto, e un’immagine illustrativa associata al tipo di ordinazione. Le rotte sono gestite tramite la direttiva [routerLink], che consente una navigazione fluida tra le pagine dell’applicazione.

Dal punto di vista TypeScript, OrdinaOraPage è un componente standalone che importa moduli essenziali di Ionic, Angular e routing. La classe è semplice e minimalista, con un costruttore vuoto e un metodo ngOnInit senza logica aggiuntiva, indicativo del fatto che la pagina svolge principalmente una funzione di interfaccia statica per la navigazione. L’architettura enfatizza la chiarezza e la semplicità, lasciando spazio per future estensioni o integrazioni senza complicare la struttura attuale.

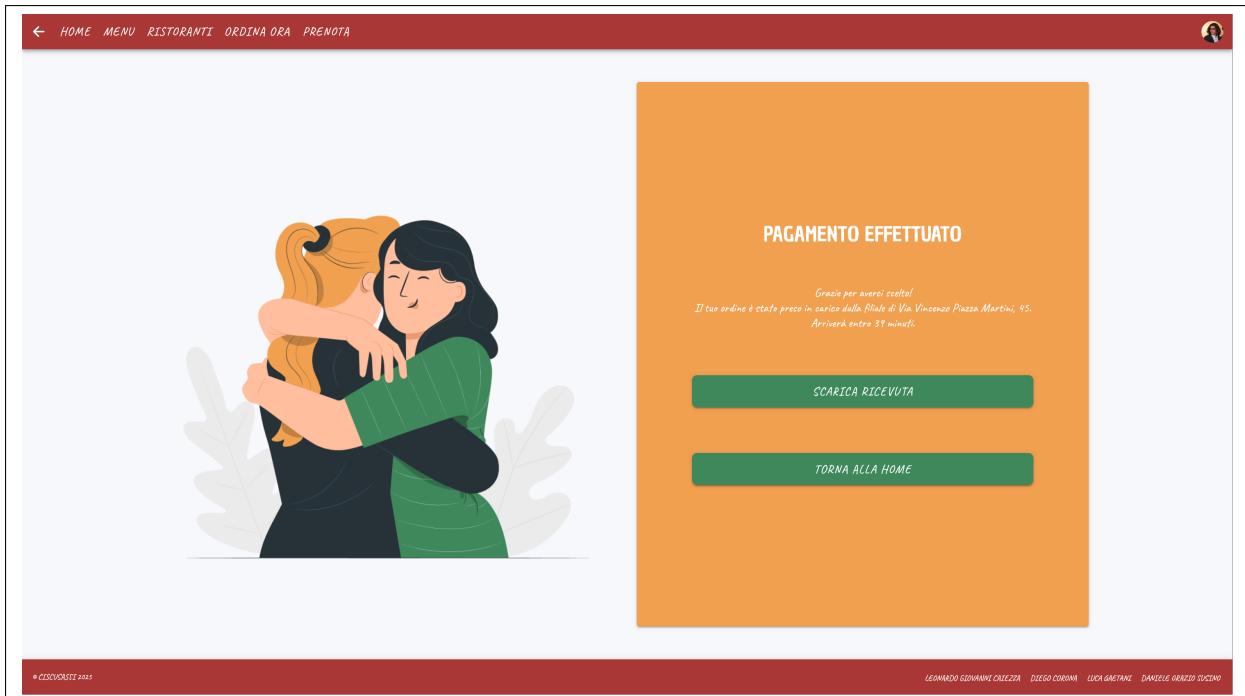


5.10.6 Visualizza Ringraziamenti Asporto

La funzionalità Visualizza Ringraziamenti Asporto in pages/ordinazioni/ringraziamenti-asporto consente di confermare il pagamento per un ordine d'asporto, offrendo all'utente un messaggio di ringraziamento e le informazioni essenziali relative alla filiale che gestirà l'ordine e ai tempi stimati di consegna. Il layout si struttura su una griglia responsive che dispone, su schermi ampi, un'immagine illustrativa nella colonna sinistra e, sulla destra, una card con il messaggio di conferma, un pulsante per scaricare la ricevuta in formato PDF e un pulsante per tornare alla schermata home. Il design utilizza colori caldi e coerenti con il tema dell'app, garantendo leggibilità e un'esperienza utente accogliente.

Nel codice TypeScript, il componente si avvale di due servizi principali per recuperare le informazioni della filiale più vicina e gestire lo stato attuale del carrello. Al momento dell'inizializzazione, si ottiene la filiale selezionata tramite il servizio dedicato e si calcola un tempo stimato di consegna sommando un margine fisso (che indica il tempo di preparazione) ai minuti di viaggio. La funzionalità principale del componente consiste nella generazione dinamica di una ricevuta PDF: tramite la creazione di un componente `RicevutaComponent` in modo programmatico, si passa ad esso il contenuto necessario (carrello, logo e tipo di servizio) e si procede alla sua resa in un elemento HTML nascosto fuori dallo schermo.

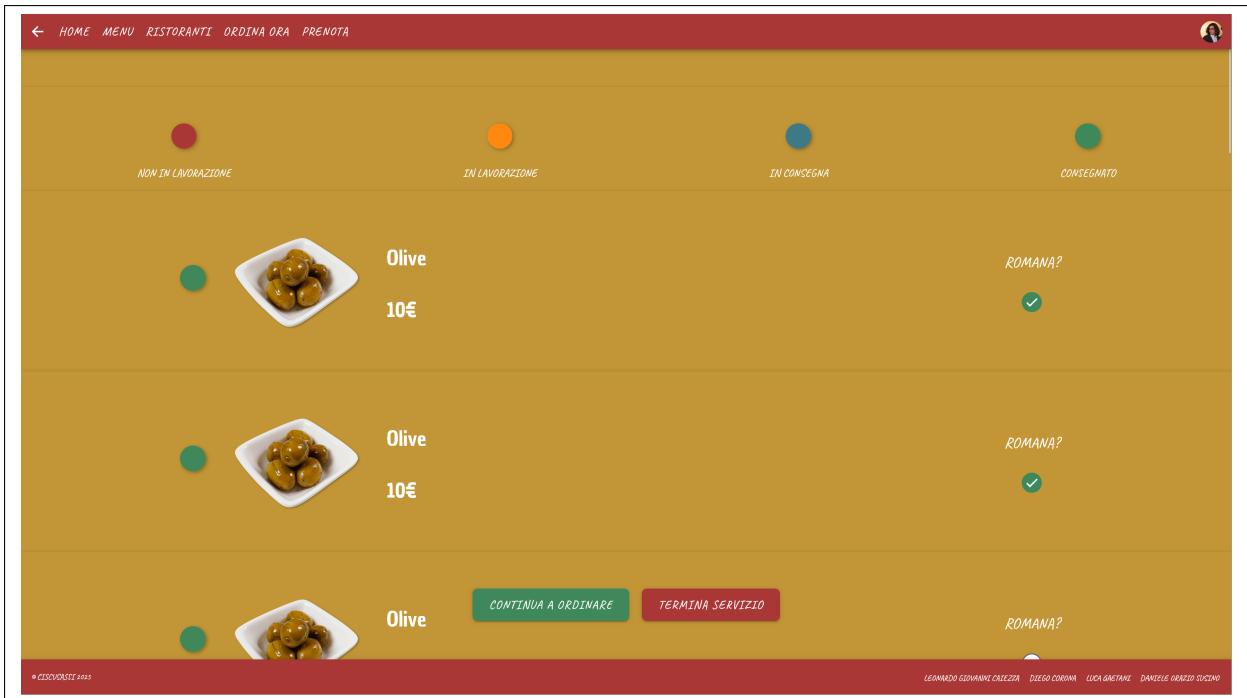
Utilizzando `html2canvas`, il componente cattura visivamente il contenuto della ricevuta in un canvas ad alta risoluzione, che viene successivamente suddiviso in pagine di formato A4 all'interno di un documento PDF creato con `jsPDF`. Questo processo permette di gestire anche ricevute che superano la lunghezza di una singola pagina, assicurando una stampa corretta e professionale. Al termine, il PDF viene automaticamente scaricato dall'utente e il componente temporaneo viene rimosso e distrutto per liberare risorse. Infine, il metodo `ngOnDestroy` si occupa di svuotare il carrello al termine dell'esperienza, mantenendo lo stato dell'app sempre aggiornato e coerente.



5.10.7 Visualizza Ordini

La funzionalità Visualizza Ordini in pages/ordinazioni/visualizza-ordini gestisce la visualizzazione in tempo reale dello stato degli ordini effettuati da un utente al tavolo, mostrando in modo chiaro e immediato le fasi di lavorazione attraverso indicatori visivi colorati e una lista dettagliata dei prodotti ordinati. L’interfaccia si compone di una serie di cerchi colorati che rappresentano i diversi stati degli ordini — non in lavorazione, in lavorazione, in consegna e consegnato — accompagnati da etichette testuali, seguiti da una lista dinamica dei prodotti ordinati, caricata e aggiornata periodicamente. Il layout utilizza colori contrastanti per mantenere una buona leggibilità e una gerarchia visiva chiara, e offre in basso due pulsanti per continuare a ordinare o terminare il servizio, guidando l’utente nelle azioni successive. Dal punto di vista funzionale, il componente utilizza un observable per gestire i dati degli ordini, inizialmente vuoto, e una variabile booleana per indicare lo stato di caricamento. Al caricamento della pagina, tramite il metodo ngOnInit, si attiva un ciclo di aggiornamento automatico che ricarica la lista degli ordini ogni 30 secondi chiamando il metodo loadOrdini, il quale recupera i dati dal backend attraverso un servizio dedicato. Nel metodo terminaServizio viene controllato che tutti i prodotti siano stati consegnati prima di consentire all’utente di procedere al pagamento; in caso contrario, viene mostrato un messaggio di avviso tramite un toast. Inoltre, alla distruzione del componente viene cancellato l’intervallo di aggiornamento per evitare perdite di memoria.

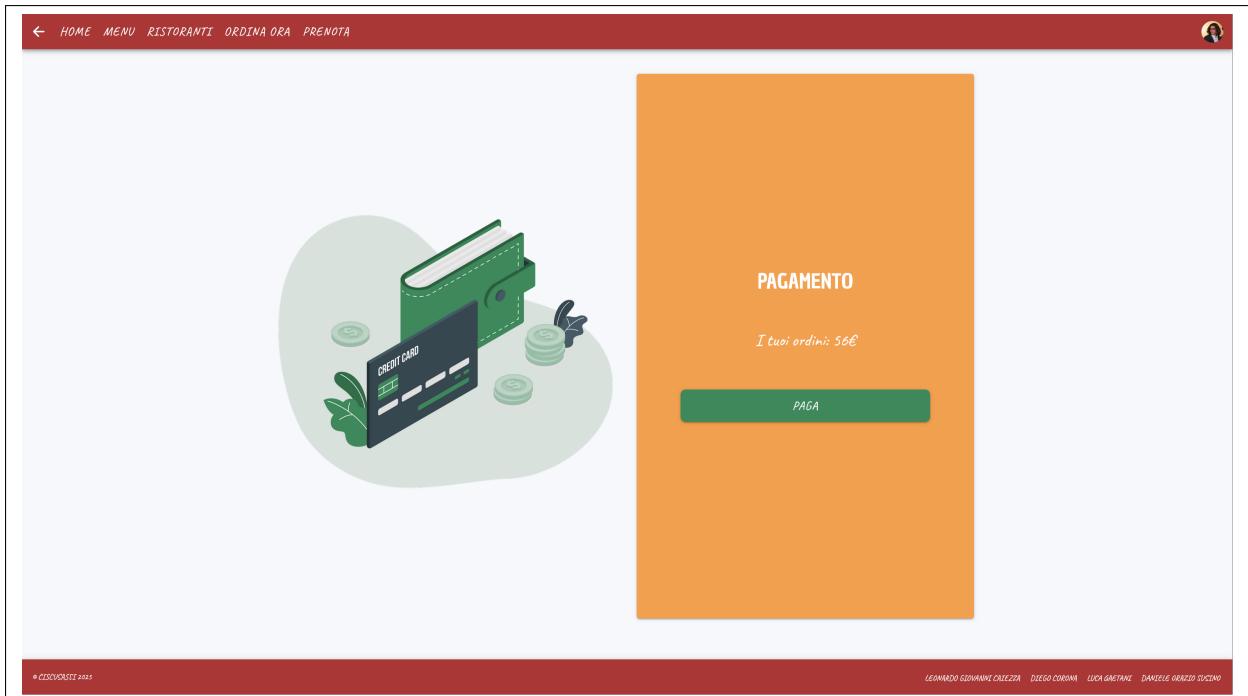
La gestione dei dati è robusta e prevede il trattamento di possibili errori o risposte inattese dal backend, mantenendo la lista degli ordini aggiornata e coerente con lo stato reale. L’uso di componenti stand-alone come ListaOrdiniComponent permette di isolare la logica di visualizzazione della lista prodotti, garantendo modularità e riusabilità del codice. Questo approccio offre all’utente un’esperienza fluida e informativa, con aggiornamenti continui che riflettono lo stato corrente del suo ordine e un’interfaccia chiara per navigare tra le diverse fasi del servizio.



5.10.8 Pagamento Asporto

La funzionalità Pagamento Asporto in pages/ordinazioni/pagamento-asporto consente all’utente di effettuare il pagamento per un ordine da asporto. Graficamente è suddiviso in due colonne all’interno di una griglia centrale: a sinistra viene mostrata un’illustrazione informativa (“SceltaPagamentoAsporto.png”) che arricchisce l’esperienza utente, mentre a destra viene presentata una scheda (ion-card) contenente il totale dell’ordine e un pulsante per completare il pagamento. L’interfaccia è studiata per guidare in modo chiaro e intuitivo l’utente alla conferma finale dell’ordine. All’avvio del componente (ngOnInit), vengono recuperati i prodotti presenti nel carrello tramite il servizio CarrelloService e ne viene calcolato il totale, formattato con due cifre decimali. Successivamente vengono acquisiti l’indirizzo dell’utente e il tempo stimato di viaggio fino alla filiale più vicina attraverso il FilialeAsportoService. In base a queste informazioni vengono calcolate due variabili temporali: l’ora attuale e l’ora stimata di consegna, ottenuta sommando al tempo di viaggio un piccolo margine. Quando l’utente preme il pulsante “PAGA”, viene eseguito il metodo aggiungiAsporto(). In questo metodo le informazioni vengono aggiornate con nuovi timestamp calcolati al momento della conferma. I dati raccolti — indirizzo, orari, costo totale, prodotti ordinati e riferimento alla filiale più vicina — vengono inseriti all’interno di un oggetto AsportoInput e inviati al backend tramite il servizio AsportoService. Il sistema registra l’ordine e, in caso di successo, viene automaticamente eseguita la navigazione verso la pagina di ringraziamento.

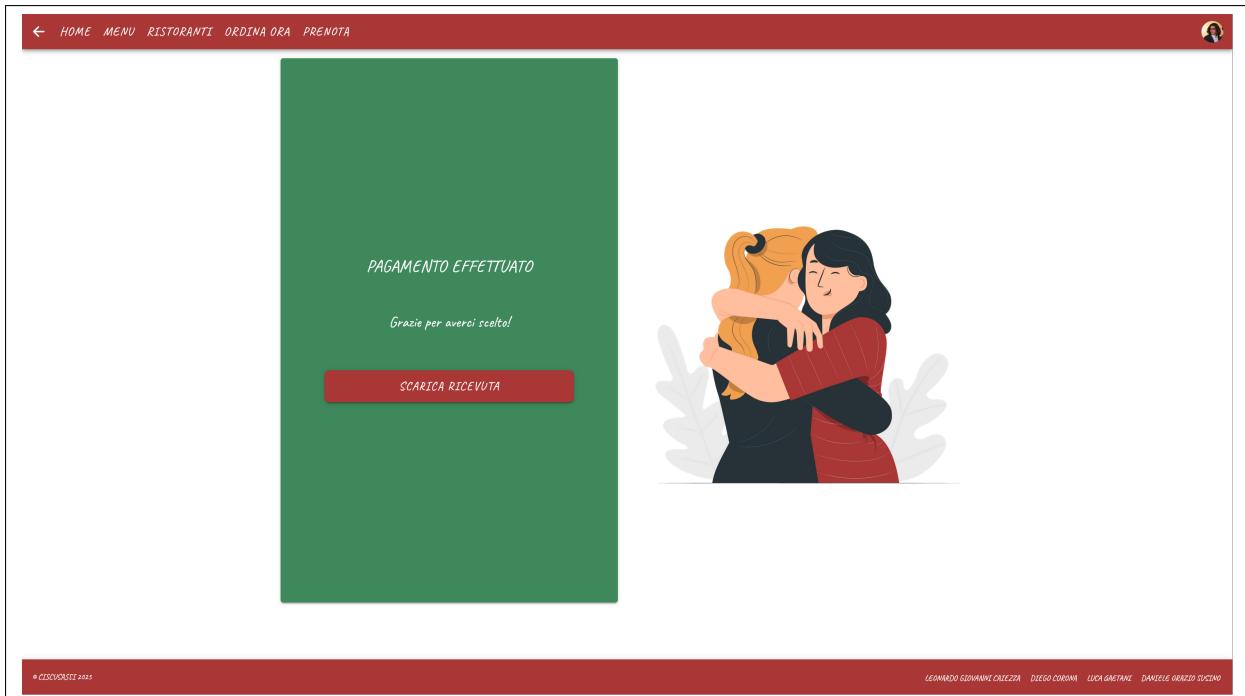
La logica complessiva è pensata per essere completamente automatica e trasparente all’utente: tutti i dati sono precompilati e l’unica azione richiesta è la conferma. L’uso del colore (salmon per lo sfondo del card, verde chiaro per il bottone) e il testo in evidenza rafforzano la gerarchia visiva, migliorando l’esperienza d’uso anche su schermi di grandi dimensioni.



5.10.9 Pagamento Carta

La funzionalità Pagamento Carta si presenta in pages/ordinazioni/pagamento-carta e permette all’utente di concludere un ordine effettuato al tavolo e di scaricare la relativa ricevuta in formato PDF. L’interfaccia è organizzata attraverso una griglia centrata (ion-grid) suddivisa in due colonne. Nella colonna sinistra è presente una card (ion-card) che conferma l’avvenuto pagamento e invita l’utente a scaricare la ricevuta tramite un pulsante ben visibile. Nella colonna destra, visibile solo su schermi di grandi dimensioni, viene mostrata un’immagine illustrativa (“PagamentoCarta.png”) che arricchisce visivamente la pagina e rafforza la sensazione di conclusione positiva dell’esperienza.

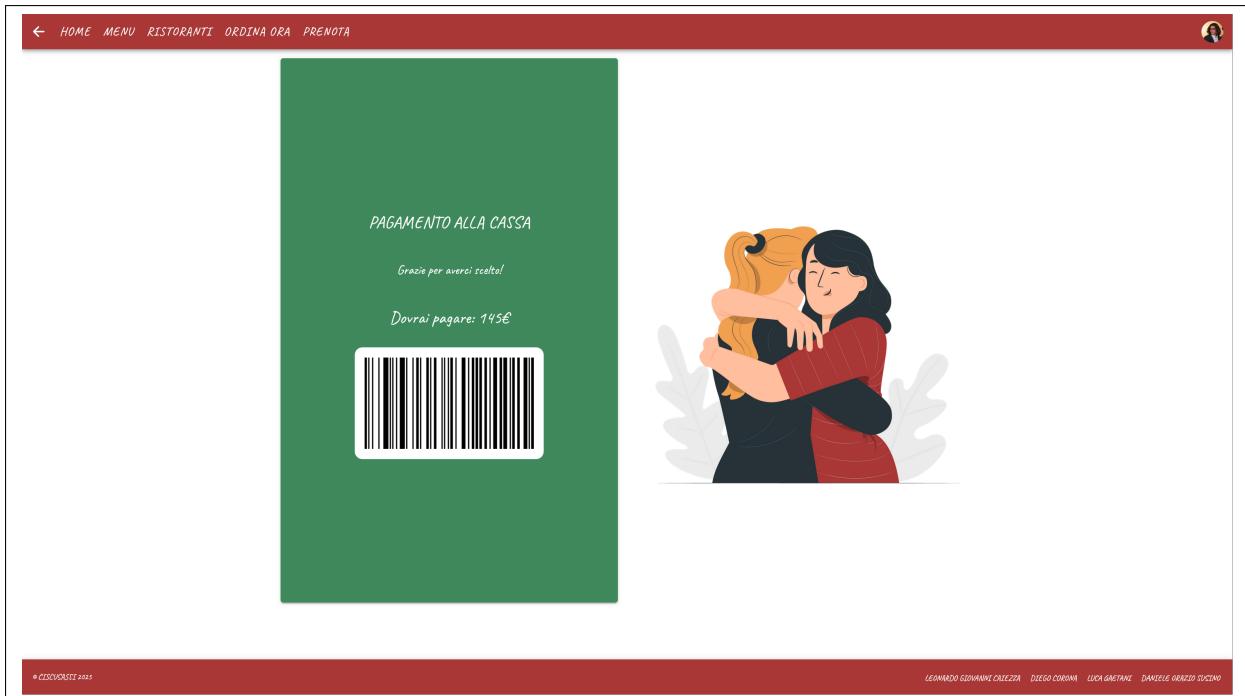
Al caricamento del componente (ngOnInit), viene eseguito il metodo ngViewWillEnter che si occupa di recuperare dal servizio TavoloService i prodotti attualmente presenti nel carrello. Questi dati vengono salvati all’interno della proprietà carrello del componente e costituiscono la base delle informazioni che verranno utilizzate nella ricevuta. Quando l’utente preme il pulsante “SCARICA RICEVUTA”, viene eseguito il metodo generaRicevuta. Questo metodo crea dinamicamente il componente RicevutaComponent, al quale vengono passati il contenuto del carrello, il logo e il tipo di servizio (“tavolo”). Il componente viene poi inserito temporaneamente nel DOM in una posizione invisibile, per garantirne il rendering completo senza interferire con l’interfaccia principale. Dopo un breve intervallo e l’attesa del caricamento delle immagini, il contenuto HTML del componente viene catturato con html2canvas ad alta risoluzione. L’immagine generata viene suddivisa in pagine, se necessario, e inserita all’interno di un oggetto jsPDF, formattato in dimensione A4. Una volta completata la generazione, il PDF viene automaticamente salvato con il nome “ricevuta.pdf” e il componente temporaneo viene rimosso e distrutto per liberare risorse.



5.10.10 Pagamento Cassa

La funzionalità Pagamento Cassa si presenta in pages/ordinazioni/pagamento-cassa e consente all’utente di visualizzare i dettagli finali del proprio ordine effettuato al tavolo e di recarsi fisicamente alla cassa per completare il pagamento. L’interfaccia è progettata per fornire in modo immediato tutte le informazioni necessarie, attraverso una griglia centrata (ion-grid) suddivisa in due colonne. Nella colonna sinistra viene mostrata una card (ion-card) con il messaggio di conferma, l’importo da pagare e un codice a barre generato dinamicamente, utile per identificare rapidamente l’ordine al momento del pagamento. La colonna destra ospita un’illustrazione decorativa (“PagamentoCarta.png”) che arricchisce visivamente la pagina ed è visibile esclusivamente su schermi di dimensioni elevate.

All’avvio del componente (ngOnInit), viene invocato il metodo ngViewWillEnter, che si occupa di recuperare i dati dell’ordine attualmente in corso attraverso il servizio TavoloService. In particolare, vengono ottenuti il numero del tavolo, il numero dell’ordine e il totale da pagare. Sulla base del numero dell’ordine viene generata una stringa testuale che viene poi utilizzata per creare il codice a barre tramite un’immagine remota fornita dall’API di bwip-js. Il caricamento del codice a barre è gestito con uno spinner che garantisce una transizione fluida: lo spinner viene visualizzato fino al caricamento completo dell’immagine, dopodiché quest’ultima viene resa visibile al posto dell’indicatore di caricamento. L’interfaccia è progettata per essere semplice, chiara e adatta a un utilizzo rapido in contesto fisico. Il colore verde chiaro del ion-card comunica un messaggio positivo di conferma, mentre il testo centrale in bianco guida l’utente in modo rassicurante. L’inserimento del codice a barre automatizza e velocizza la procedura di riconoscimento dell’ordine alla cassa, riducendo al minimo le interazioni necessarie. L’esperienza complessiva è pensata per accompagnare l’utente fino alla conclusione dell’ordine con il minimo sforzo cognitivo, sfruttando un’interfaccia essenziale ma funzionale.

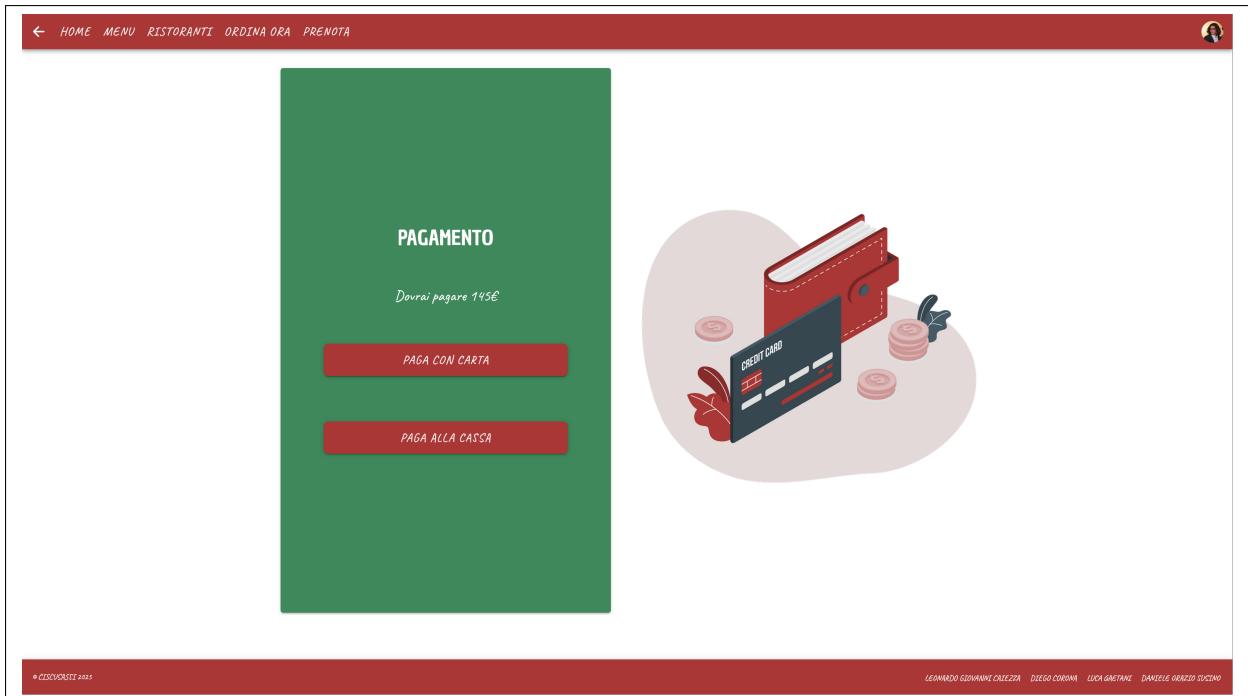


5.10.11 Pagamento Tavolo

La pagina Pagamento Tavolo si presenta in pages/ordinazioni/pagamento-tavolo e realizza l'ultimo passaggio del flusso di ordinazione da tavolo. In questa schermata, l'utente visualizza l'importo totale da pagare per l'ordine corrente e può scegliere in modo chiaro tra due modalità di pagamento: con carta oppure alla cassa. L'interfaccia è composta da una griglia suddivisa in due colonne: nella parte sinistra, un box con sfondo verde chiaro presenta l'importo da pagare e due pulsanti ben evidenziati per selezionare la modalità preferita. Nella colonna destra, visibile solo su schermi extra-large, è mostrata un'illustrazione grafica a supporto.

Durante l'inizializzazione del componente (ngOnInit), viene recuperato il numero dell'ordine attraverso il TavoloService. Se l'ordine risulta valido, il sistema interroga il PrenotazioneService per ottenere l'importo totale associato all'ordine e ne aggiorna lo stato interno e il servizio condiviso. In caso di errore o numero ordine non valido, il totale viene forzato a zero e viene loggato un messaggio di warning. Entrambi i metodi di pagamento (pagaCarta e pagaCassa) eseguono una chiamata al servizio PagamentoService, passando il numero dell'ordine, il totale e la data formattata (ottenuta tramite getFormattedDate). Se la chiamata ha successo, l'utente viene reindirizzato alla rispettiva pagina di conferma del pagamento (/pagamento-carta o /pagamento-cassa). In caso di errore, viene mostrato un toast con un messaggio di avviso, utile per comunicare eventuali problemi come ordini già pagati o errori lato server.

La progettazione della pagina è focalizzata sulla chiarezza visiva e operativa: i pulsanti sono grandi, ben separati, con etichette dirette e uno stile coerente con il resto dell'applicazione. Il colore rosso dei pulsanti contrasta volutamente con il verde del contenitore per rendere evidenti le azioni disponibili. L'approccio modulare del codice assicura una buona manutenibilità, separando nettamente la logica di calcolo, la gestione dell'interfaccia e l'invocazione dei servizi.

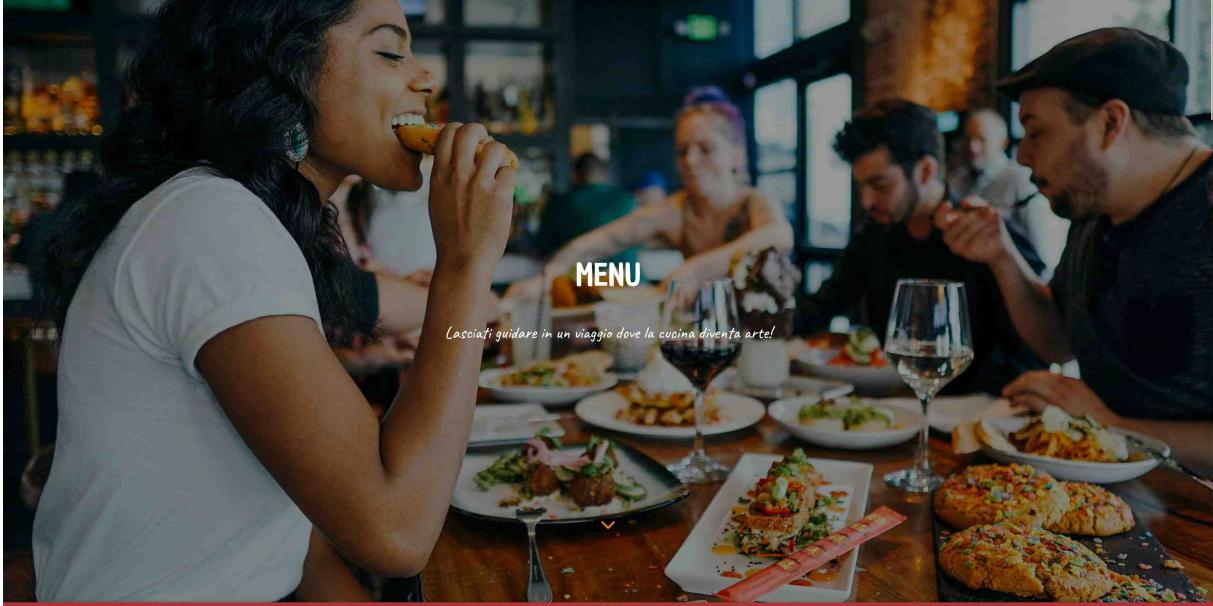


5.11 Visualizza Schermata Menu

La funzionalità Visualizza Schermata Menù si presenta in pages/menu che funge da contenitore principale per la visualizzazione del menu di un ristorante. Nel template HTML, un elemento ion-content con fullscreen attivato avvolge l'intero contenuto della pagina, garantendo che utilizzi tutto lo spazio disponibile sullo schermo. Al suo interno, il componente app-hero introduce la pagina con un titolo "MENU" e una descrizione evocativa, accompagnata da uno sfondo specificato tramite la proprietà backgroundURL impostata su "MenuBackground.jpg". Successivamente, il componente app-piatto-del-giorno mostra il piatto del giorno, una sezione dedicata alla specialità quotidiana, che offre più punti cliente se ordinata. Infine, il componente app-lista-menu presenta la lista dei piatti disponibili, con una proprietà isOrdinazione impostata su false per indicare che la pagina non si trova in modalità ordinazione.

Dal punto di vista TypeScript, il componente MenuPage è dichiarato come standalone e importa moduli fondamentali come IonContent e CommonModule, oltre ai componenti personalizzati HeroComponent, PiattoDelGiornoComponent e ListaMenuComponent necessari per la composizione della pagina. Questa struttura sottolinea un approccio modulare e riutilizzabile, dove la pagina funge da shell che aggrega e mostra componenti funzionali già definiti, mantenendo separata la logica di presentazione dalla logica di business e facilitando la manutenzione e l'estensione futura.

← HOME MENU RISTORANTI ORDINA ORA PRENOTA



MENU

Lasciati guidare in un viaggio dove la cucina diventa arte!

© CICCOVORSE 2023

LEONARDO GIOVANNI CALEZZA DIEGO CORONA LUCA GRATTANI DANIELE ORAZIO SUCINO

← HOME MENU RISTORANTI ORDINA ORA PRENOTA



PIATTO DEL GIORNO

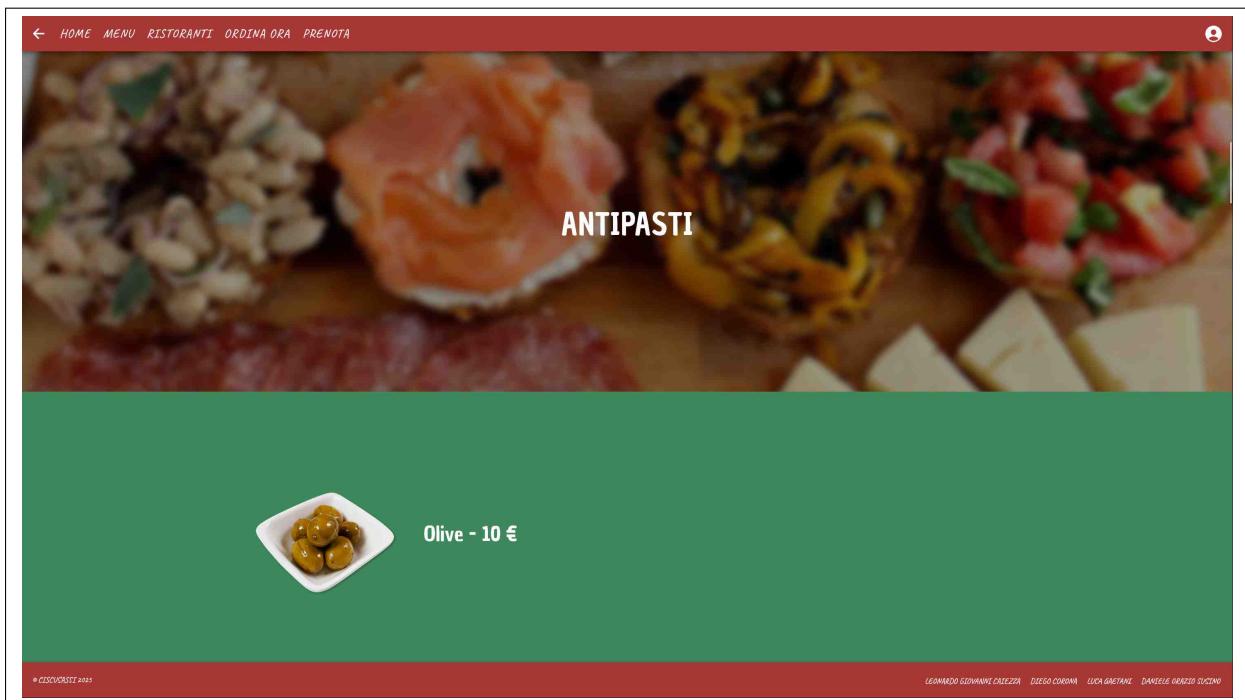
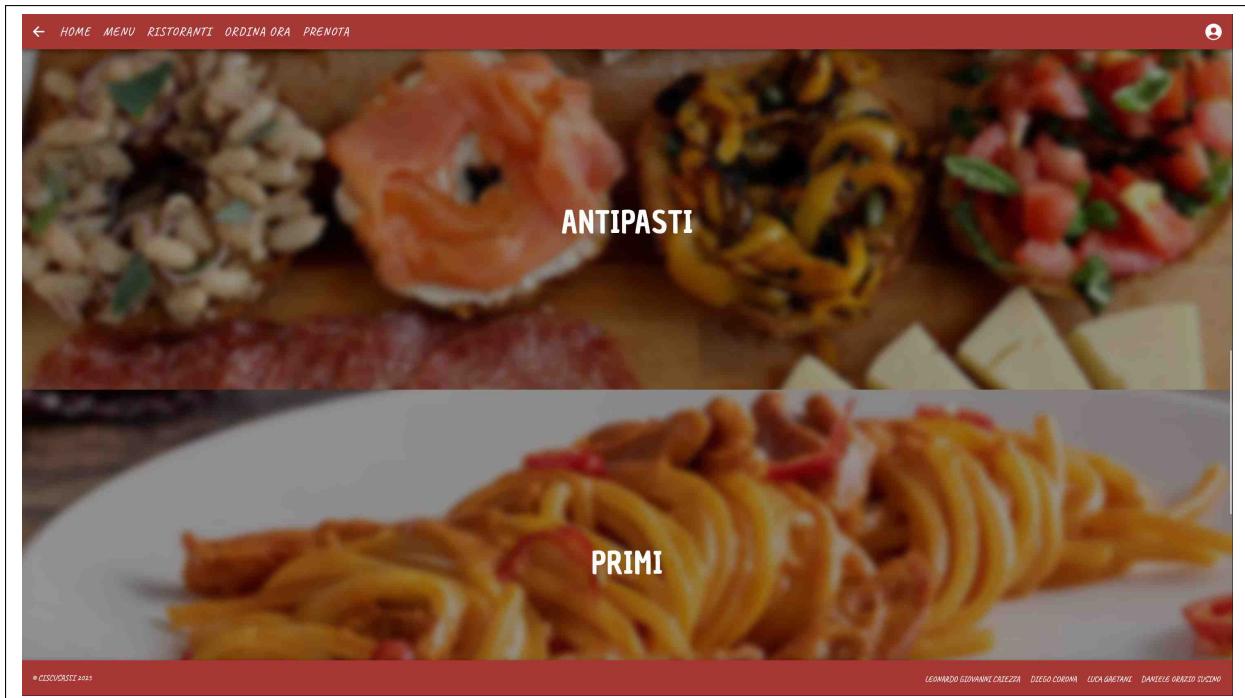
VEG RAGÙ - 14 €

Fusilli di sorgo regenerativo, ragù plant-based di piselli e lenticchie, basilico

* Acquistando il piatto del giorno hai diritto a 10 pt fedeltà bonus.

© CICCOVORSE 2023

LEONARDO GIOVANNI CALEZZA DIEGO CORONA LUCA GRATTANI DANIELE ORAZIO SUCINO

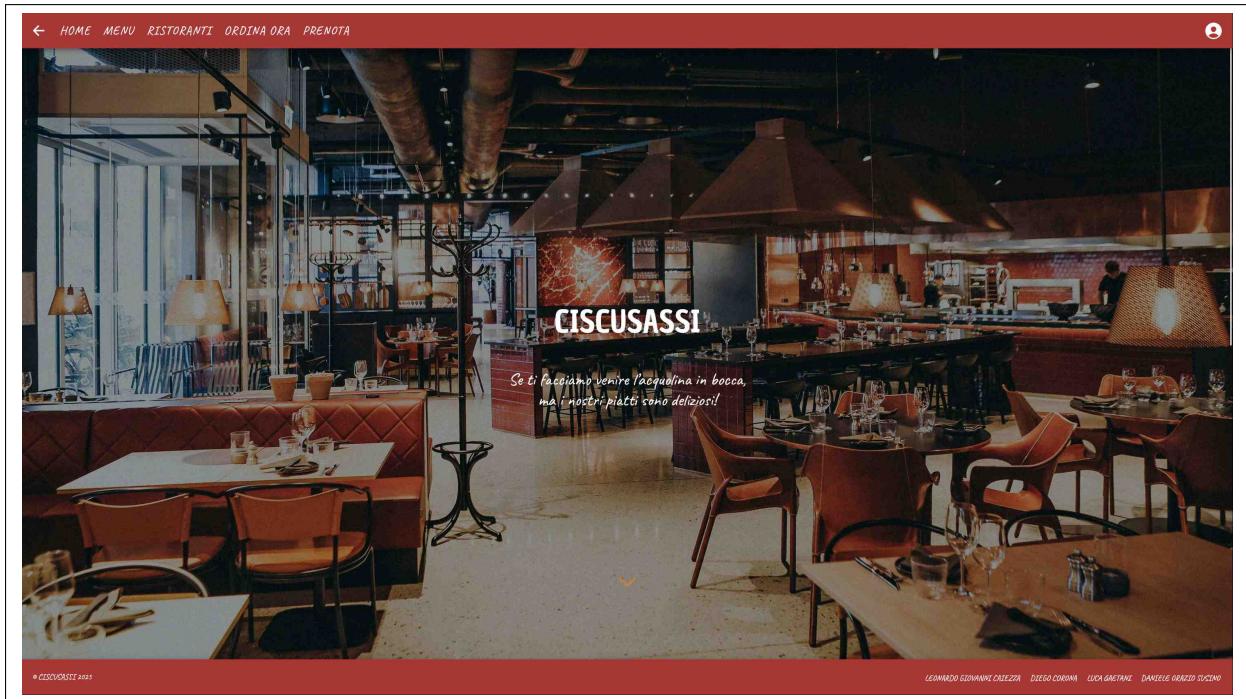


5.12 Visualizza Schermata Home

La funzionalità Visualizza Schermata Home si presenta in pages/Home e consente di visualizzare la schermata principale dell'applicazione. Il suo scopo è accogliere l'utente con un messaggio introduttivo e presentare il piatto del giorno, creando così un'esperienza coinvolgente fin dall'inizio.

Nel template HTML, la pagina utilizza un contenitore <ion-content> impostato in modalità fullscreen, che occupa l'intera area visibile dello schermo per offrire una visualizzazione senza interruzioni o margini. All'interno di questo contenitore si trova il componente app-hero, che funge da sezione hero della pagina. Questo componente riceve un titolo e una descrizione testuale. La sezione hero ha anche uno sfondo personalizzato tramite la proprietà backgroundURL, impostato su "HomeBackground.jpg". Sotto la sezione hero è presente il componente app-piatto-del-giorno, che mostra dinamicamente il piatto speciale o del

giorno, rendendo la pagina più accattivante e focalizzata sui contenuti culinari.



6 Design

6.1 Scelte di design

Il **design dell'applicazione** è stato pensato per coniugare semplicità, riconoscibilità e un'**identità visiva coerente con l'ambiente moderno e accogliente dei ristoranti**.

Particolare attenzione è stata riservata all'accessibilità, all'usabilità mobile e alla leggibilità dei contenuti, con un'interfaccia reattiva e intuitiva per tutte le tipologie di utenti.

6.2 Tema e palette colori

Il **tema visivo** è stato costruito attorno a una **paletta di colori caldi e naturali**, che **evocano l'ambiente di un ristorante accogliente e artigianale**.

Ogni colore della paletta ha un **ruolo semantico ben definito** all'interno dell'interfaccia utente:

Rosso #a83736 – Colore primario, usato per header, footer, elementi chiave e pulsanti principali.

Arancione #ff8811 – Colore secondario, utilizzato per evidenziare azioni rilevanti.

Verde scuro #21351b – Colore terziario associato ai piatti del giorno e sezioni speciali.

Verde chiaro #3e885b – Utilizzato come sfondo per schermate secondarie (login, registrazione).

Senape #c29637 – Colore neutro per sfondi meno importanti.

Salmone #f1a050 – Colore di supporto per sfondi meno importanti.

Azzurro #3E7988 – Colore di supporto per gli stati dei tavoli.

Bianco neve #f3f3f4 – Sfondo principale delle schermate.

Bianco #ffffff – Usato per contenuti, testi, e sfondi puri.

Questa struttura cromatica assicura una **gerarchia visiva chiara** e aiuta l'utente a distinguere sezioni, azioni e livelli informativi all'interno dell'applicazione.

6.3 Fonts

L'**identità visiva** dell'applicazione è definita anche attraverso un'**attenta selezione tipografica**, che bilancia eleganza, leggibilità e carattere distintivo. Sono stati scelti **tre font principali**, ciascuno con un ruolo ben preciso all'interno dell'interfaccia:

- **Chau Philomene One:** Impiegato per gli header di primo e secondo livello (**h1, h2**), questo font sans-serif dalla struttura compatta ed elegante suggerisce una gerarchia visiva ben definita. La forma leggermente squadrata delle lettere bilancia il minimalismo con una presenza autorevole, guidando lo sguardo dell'utente attraverso le sezioni fondamentali dell'interfaccia.
- **Caveat:** Utilizzato per tutto il contenuto testuale secondario: paragrafi, descrizioni, pulsanti e link. Questo font handwritten infonde un tono informale e coinvolgente. La sua calligrafia dinamica, dal

tratto fluido e personale, trasmette empatia e calore comunicativo, pur mantenendo leggibilità e coerenza nel layout generale.

Il **contrasto** tra la precisione semantica di *Chau Philomene One* e la spontaneità espressiva di *Caveat* genera un'esperienza visiva equilibrata, umanamente accessibile e gerarchicamente chiara, dove la struttura informativa è al servizio di una narrazione visuale empatica ma rigorosa.

6.4 Mockup

I **mockup** allegati rappresentano una **testimonianza concreta del processo creativo e progettuale** che ha guidato lo sviluppo dell'applicazione.

Attraverso queste interfacce preliminari è stato possibile **esplorare soluzioni visive**, definire la gerarchia informativa e **verificare la coerenza tra l'identità grafica e le funzionalità previste**.

Ogni schermata è stata pensata per **ottimizzare l'esperienza utente**, tenendo conto dell'usabilità, della chiarezza comunicativa e della responsività del design.

- I mockup hanno quindi costituito una **base solida per il successivo sviluppo tecnico**, permettendo un **passaggio fluido dall'ideazione all'implementazione**.



HOME MENU RISTORANTI ORDINA ORA PRENOTA

AMMINISTRAZIONE

MENU

*Benvenuto Daniele!
Pronto ad ordinare? Scendi giù!*

V

PIATTO DEL GIORNO



TRICOLORÈ - 12 €

Paccheri con Grani Antichi, salsa di pomodoro datterino siciliano, pesto e granella di pistacchi del Mediterraneo, stracciatella pugliese, olio EVO, basilico

AGGIUNGI AL CARRELLO **QT: 69** **RIMUOVI DAL CARRELLO**

* Acquistando il piatto del giorno hai diritto a 10 pt fedeltà bonus.

ANTIPASTI

PRIMI

TRICOLORÈ - 12€

Paccheri con Grani Antichi, salsa di pomodoro datterino siciliano, pesto e granella di pistacchi del Mediterraneo, stracciatella pugliese, olio EVO, basilico

AGGIUNGI AL CARRELLO **QT: 69** **RIMUOVI DAL CARRELLO**

PANINI

SECONDI

PROCEDE AL PAGAMENTO

© CINCUSASSI 2025
Tutti i diritti riservati

LEONARDO GIOVANNI CAZZA - DIEGO CORONA - LUCA GATTANI - DANTE E ORAZIO SUSINO

HOME MENU RISTORANTI ORDINA ORA PRENOTA AMMINISTRAZIONE  



PAGAMENTO

I tuoi ordini: 36€
Romana: 33€
Totale: 69€

PAGA 69€

© CISCUSASSI 2025
Tutti i diritti riservati

LEONARDO GIOVANNI CAIEZZA DIEGO CORONA LUCA GAETANI DANIELE ORAZIO SUSINO

HOME MENU RISTORANTI ORDINA ORA PRENOTA AMMINISTRAZIONE  



INFORMAZIONI AGGIUNTIVE

Inserisci il tuo indirizzo per ricevere comodamente l'ordine alla tua posizione corrente

ORDINA

© CISCUSASSI 2025
Tutti i diritti riservati

LEONARDO GIOVANNI CAIEZZA DIEGO CORONA LUCA GAETANI DANIELE ORAZIO SUSINO



Modifica Dipendente



Inserisci il nome

Inserisci il ruolo

Amministratore

Chef

Cameriere

MODIFICA
DIPENDENTE



VISUALIZZA COMANDE
TAVOLO 8

[CONSEGNA TUTTO](#)

TRICOLORE

[CONSEGNA](#)[CESTINA](#)

Paccheri con Grani Antichi, salsa di pomodoro datterino siciliano, pesto e granella di pistacchi del Mediterraneo, stracciatella pugliese, olio EVO, basilico



TRICOLORE

IN LAVORAZIONE

Paccheri con Grani Antichi, salsa di pomodoro datterino siciliano, pesto e granella di pistacchi del Mediterraneo, stracciatella pugliese, olio EVO, basilico



TRICOLORE

[CONSEGNA](#)

Paccheri con Grani Antichi, salsa di pomodoro datterino siciliano, pesto e granella di pistacchi del Mediterraneo, stracciatella pugliese, olio EVO, basilico



VISUALIZZA COMANDE
TAVOLO 8

[INIZIA LAVORAZIONE TOTALE](#)[FINE LAVORAZIONE TOTALE](#)

TRICOLORE

[INIZIA LAVORAZIONE](#)

Paccheri con Grani Antichi, salsa di pomodoro datterino siciliano, pesto e granella di pistacchi del Mediterraneo, stracciatella pugliese, olio EVO, basilico



TRICOLORE

[FINE LAVORAZIONE](#)

Paccheri con Grani Antichi, salsa di pomodoro datterino siciliano, pesto e granella di pistacchi del Mediterraneo, stracciatella pugliese, olio EVO, basilico



TRICOLORE

[IN CONSEGNA](#)

Paccheri con Grani Antichi, salsa di pomodoro datterino siciliano, pesto e granella di pistacchi del Mediterraneo, stracciatella pugliese, olio EVO, basilico



CREA FILIALE



Inserisci il comune

Inserisci l'indirizzo

Numero tavoli

CREA FILIALE

© CISCUSASSI 2025

LEONARDO GIOVANNI CAIEZZA DIEGO CORONA LUCA GAETANI DANIELE ORAZIO SUSINO

Tutti i diritti riservati



CREA PIATTO



Inserisci il nome

Inserisci una descrizione

Costo

Categoria

CREA PIATTO

© CISCUSASSI 2025

LEONARDO GIOVANNI CAIEZZA DIEGO CORONA LUCA GAETANI DANIELE ORAZIO SUSINO

Tutti i diritti riservati

HOME MENU RISTORANTI ORDINA ORA PRENOTA

AMMINISTRAZIONE  



**BENVENUTO
LEONARDO!**

GESTISCI PIATTI
GESTISCI FILIALI
VISUALIZZA UTILI

© CISCUSASSI 2025
Tutti i diritti riservati

LEONARDO GIOVANNI CAIEZZA DIEGO CORONA LUCA GAETANI DANIELE ORAZIO SUSINO

HOME MENU RISTORANTI ORDINA ORA PRENOTA

AMMINISTRAZIONE  



GESTIONE ACCOUNT

LOGIN
SIGNIN

© CISCUSASSI 2025
Tutti i diritti riservati

LEONARDO GIOVANNI CAIEZZA DIEGO CORONA LUCA GAETANI DANIELE ORAZIO SUSINO

HOME MENU RISTORANTI ORDINA ORA PRENOTA

AMMINISTRAZIONE  



GESTIONE ACCOUNT



Daniele Susino
Punti accumulati
20000 PT

CAMBIA PASSWORD

CAMBIA EMAIL

CANCELLA ACCOUNT

LOGOUT

© CISCUSASSI 2025
Tutti i diritti riservati

LEONARDO GIOVANNI CAIEZZA DIEGO CORONA LUCA GAETANI DANIELE ORAZIO SUSINO

HOME MENU RISTORANTI ORDINA ORA PRENOTA

AMMINISTRAZIONE  



GESTIONE ACCOUNT



Daniele Susino
cuoco
007689432

CAMBIA PASSWORD

CAMBIA EMAIL

LOGOUT

© CISCUSASSI 2025
Tutti i diritti riservati

LEONARDO GIOVANNI CAIEZZA DIEGO CORONA LUCA GAETANI DANIELE ORAZIO SUSINO



Gestisci Impiegati



LEONARDO GIOVANNI
CAEZZA

007689432

CUOCO

[MODIFICA IMPIEGATO](#)

[LICENZIA](#)



DIEGO CORONA

007688745

CAMERIERE

[MODIFICA IMPIEGATO](#)

[LICENZIA](#)

© CISCUSASSI 2025

LEONARDO GIOVANNI CAEZZA DIEGO CORONA LUCA GAETANI DANIELE ORAZIO SUSINO

Tutti i diritti riservati



GESTISCI FILIALI



PALERMO

Via Carciofi, 85

[MODIFICA FILIALE](#)

[GESTISCI IMPIEGATO](#)

[CANCELLA FILIALE](#)



PALERMO

Via Vincenzo Piazza Martini, 45

[MODIFICA FILIALE](#)

[GESTISCI IMPIEGATO](#)

[CANCELLA FILIALE](#)

© CISCUSASSI 2025

LEONARDO GIOVANNI CAEZZA DIEGO CORONA LUCA GAETANI DANIELE ORAZIO SUSINO

Tutti i diritti riservati



GESTISCI PIATTI

Filtra per nome



| | | |
|---|---|---|
| TRICOLORE MODIFICA PIATTO CANCELLA PIATTO | TRICOLORE MODIFICA PIATTO CANCELLA PIATTO | TRICOLORE MODIFICA PIATTO CANCELLA PIATTO |
|---|---|---|

© CISCUSASSI 2025

Tutti i diritti riservati

LEONARDO GIOVANNI CAEZZA DIEGO CORONA LUCA GAETANI DANIELE ORAZIO SUSINO



CISCUSASSI

*Se ti facciamo venire l'acquolina in bocca,
ma i nostri piatti sono deliziosi!*

[ORDINA ORA](#)[PRENOTA](#)

© CISCUSASSI 2025
Tutti i diritti riservati

LEONARDO GIOVANNI CAPEZZA DIEGO CORONA LUCA GAETANI DANIELE ORAZIO SUSINO



PIATTO DEL GIORNO



TRICOLORI - 12 €

Paccheri con Grani Antichi, salsa di pomodoro datterino siciliano, pesto e granella di pistacchi del Mediterraneo, stracciatella pugliese, olio EVO, basilico

* Acquistando il piatto del giorno hai diritto a 10 pt fedeltà bonus.



PRENOTAZIONE

*Prenota il tuo tavolo e regalati
un'esperienza culinaria senza tempo!*

V

PRENOTA UN TAVOLO

Hai già una prenotazione!

📍 Via Vincenzo Piazza Martini, 45, Palermo

⌚ 27/06/2025 - 20:30

✖ 4 posti

[Cancella Prenotazione](#)



LOGIN

Inserisci la tua email

Inserisci la tua password

PROSEGUO

Hai dimenticato la password? [Recupera la tua password](#)

MENU

*Lasciati guidare in un viaggio
dove la cucina diventa arte!*

V

PIATTO DEL GIORNO

TRICOLORE - 12 €



Paccheri con Grani Antichi, salsa di pomodoro datterino siciliano, pesto e granella di pistacchi del Mediterraneo, stracciatella pugliese, olio EVO, basilico

* Acquistando il piatto del giorno hai diritto a 10 pt fedeltà bonus.

ANTIPASTI

PRIMI

TRICOLORE - 12 €



Paccheri con Grani Antichi, salsa di pomodoro datterino siciliano, pesto e granella di pistacchi del Mediterraneo, stracciatella pugliese, olio EVO, basilico

PANINI

SECONDI

Assumi Dipendente



Inserisci il nome

Inserisci il ruolo

Amministratore

Chef

Cameriere

ASSUMI DIPENDENTE

© CISCUSASSI 2025

Tutti i diritti riservati

LEONARDO GIOVANNI CAIEZZA DIEGO CORONA LUCA GAETANI DANIELE ORAZIO SUSINO

MODIFICA FILIALE



Inserisci il comune

Inserisci l'indirizzo

Numero Tavoli

CREA FILIALE

© CISCUSASSI 2025

Tutti i diritti riservati

LEONARDO GIOVANNI CAIEZZA DIEGO CORONA LUCA GAETANI DANIELE ORAZIO SUSINO



MODIFICA PIATTO



Tricolore

Inserisci una descrizione

Costo

Categoria

APPLICA MODIFICHE

© CISCUSASSI 2025

LEONARDO GIOVANNI CAPEZZA DIEGO CORONA LUCA GAETANI DANTELE ORAZIO SUSINO

Tutti i diritti riservati



ORDINA AL TAVOLO

Ordina comodamente dal tuo tavolo, non c'è bisogno di aspettare nessuno!



ORDINA D'ASPORTO

Ordina da casa tua, appena sarà tutto pronto ti consegneremo un pasto delizioso!



© CISCUSASSI 2025

LEONARDO GIOVANNI CAPEZZA DIEGO CORONA LUCA GAETANI DANTELE ORAZIO SUSINO

Tutti i diritti riservati



PAGAMENTO ALLA CASSA

Dovrai pagare

69€

Numero tavolo

1



© CISCUSASSI 2025

LEONARDO GIOVANNI CAIEZZA DIEGO CORONA LUCA GAETANI DANIELE ORAZIO SUSINO

Tutti i diritti riservati



PAGAMENTO EFFETTUATO

Grazie per averci scelto!

SCARICA RICEVUTA



© CISCUSASSI 2025

LEONARDO GIOVANNI CAIEZZA DIEGO CORONA LUCA GAETANI DANIELE ORAZIO SUSINO

Tutti i diritti riservati

HOME MENU RISTORANTI ORDINA ORA PRENOTA

AMMINISTRAZIONE  



PAGAMENTO EFFETTUATO

Grazie per averci scelto!
Il tuo ordine è stato preso in carico
dalla filiale di via xxx
Arriverà entro 69 minuti

© CISCUSASSI 2025
Tutti i diritti riservati

LEONARDO GIOVANNI CAPEZZA DIEGO CORONA LUCA GAETANI DANIELE ORAZIO SUSINO

HOME MENU RISTORANTI ORDINA ORA PRENOTA

AMMINISTRAZIONE  



PAGAMENTO

Dovrai pagare 69€

PAGA CON CARTA

PAGA ALLA CASSA

© CISCUSASSI 2025
Tutti i diritti riservati

LEONARDO GIOVANNI CAPEZZA DIEGO CORONA LUCA GAETANI DANIELE ORAZIO SUSINO



PRENOTAZIONE

*Prenota il tuo tavolo e regalati
un'esperienza culinaria senza tempo!*

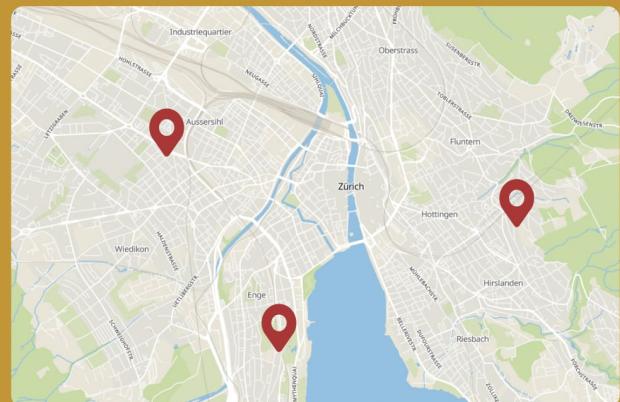
PRENOTA UN TAVOLO

Trova il ristorante più adatto a te!

Inserisci un comune



- **PALERMO**
Via Vincenzo Piazza Martini, 45
Orario Apertura: 15:00
- **PALERMO**
Via Palmerino, 52/A
Orario Apertura: 11:00
- **PALERMO**
Via Catania, 17
Orario Apertura: 12:00
-  **PALERMO**





PRENOTA UN TAVOLO

Palermo, Via Vincenzo Piazza Martini, 45

In quanti sarete?

1

2

3

4

5

6

7

OPPURE

Inserisci in quanti sarete...

CONFERMA

© CISCUSASSI 2025

LEONARDO GIOVANNI CAIEZZA DIEGO CORONA LUCA GAETANI DANIELE ORAZIO SUSINO

Tutti i diritti riservati



PRENOTA UN TAVOLO

Palermo, Via Vincenzo Piazza Martini, 45

5 persone

Scegli una data



Scegli un orario

18-19

18-19

18-19

CONFERMA

© CISCUSASSI 2025

LEONARDO GIOVANNI CAIEZZA DIEGO CORONA LUCA GAETANI DANIELE ORAZIO SUSINO

Tutti i diritti riservati



RECUPERA PASSWORD

Inserisci la tua email

PROSEGUO

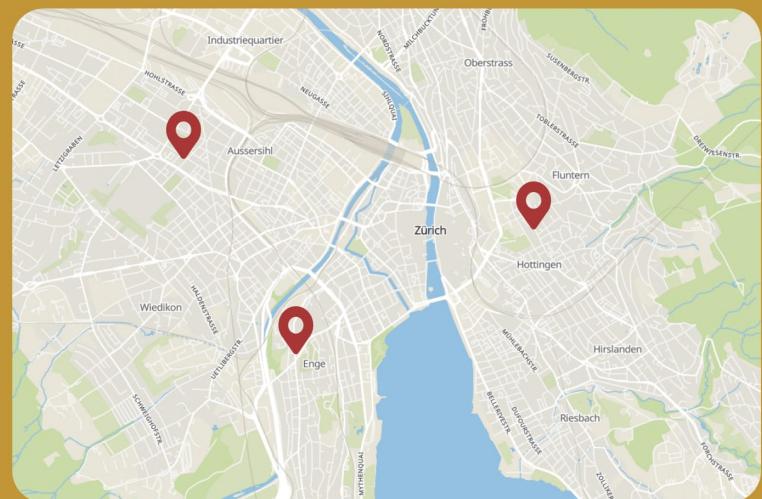
Te la sei ricordata? [Torna indietro!](#)

I NOSTRI RISTORANTI

*Locali accoglienti, cibo delizioso.
Cosa vuoi di più?*

V

TROVACI SULLA MAPPA!





SIGNIN

Inserisci il tuo nome

Inserisci il tuo cognome

Inserisci la tua email

Inserisci la tua password

Conferma la tua password

 PROSEGUO



VISUALIZZA TAVOLI

NON IN LAVORAZIONE *IN LAVORAZIONE*

IN CONSEGNA *CONSEGNATO*



VISUALIZZA TAVOLI

IN CONSEGNA CONSEGNATO

QUANTE PERSONE SI DEVONO
SEDERE?

1

2

3

4

5

6

7

OPPURE

Inserisci in quanti sarete...

CONFERMA

ESCI

9

Susino

20-22

4 persone

10

Gaetani

19-21

8 persone

+



VISUALIZZA TAVOLI

IN CONSEGNA CONSEGNATO

6

3

7

2

8

1

5

4

9

10

11

+

Susino

20-22

4 persone

Gaetani

19-21

8 persone

19-21

8 persone



VISUALIZZA TAVOLI



NON IN LAVORAZIONE



IN LAVORAZIONE

8

1

5

4

HOME MENU RISTORANTI ORDINA ORA PRENOTA AMMINISTRAZIONE

MENU

*Benvenuto Daniele!
Sei al tavolo xxx
Pronto ad ordinare? Scendi giù!*

V

PIATTO DEL GIORNO



TRICOLORÈ - 12 €

Paccheri con Grani Antichi, salsa di pomodoro datterino siciliano, pesto e granella di pistacchi del Mediterraneo, stracciatella pugliese, olio EVO, basilico

AGGIUNGI AL CARRELLO **QT: 69** **RIMUOVI DAL CARRELLO**

* Acquistando il piatto del giorno hai diritto a 10 pt fedeltà bonus.

ANTIPASTI



PRIMI



TRICOLORÈ - 12€

Paccheri con Grani Antichi, salsa di pomodoro datterino siciliano, pesto e granella di pistacchi del Mediterraneo, stracciatella pugliese, olio EVO, basilico

AGGIUNGI AL CARRELLO **QT: 69** **RIMUOVI DAL CARRELLO**

PANINI



SECONDI



VISUALIZZA ORDINI **INVIA ORDINE**

© CUSCUS ASSI 2025
Tutti i diritti riservati

LEONARDO GIOVANNI CAZZA - DIEGO CORONA - LUCA GAETANI - DANIELE OBIAZZO SUSTINO

 *NON IN LAVORAZIONE* *IN LAVORAZIONE* *IN CONSEGNA* *CONSEGNATO*

TRICOLORE

*ROMANA?***12€**

Paccheri con Grani Antichi, salsa di pomodoro datterino siciliano, pesto e granella di pistacchi del Mediterraneo, stracciatella pugliese, olio EVO, basilico



TRICOLORE

*ROMANA?***12€**

Paccheri con Grani Antichi, salsa di pomodoro datterino siciliano, pesto e granella di pistacchi del Mediterraneo, stracciatella pugliese, olio EVO, basilico



TRICOLORE

*ROMANA?***12€**

Paccheri con Grani Antichi, salsa di pomodoro datterino siciliano, pesto e granella di pistacchi del Mediterraneo, stracciatella pugliese, olio EVO, basilico

TOTALE DA PAGARE: 36€[CONTINUA AD ORDINARE](#)[TERMINA SERVIZIO](#)



VISUALIZZA UTILI

Inserisci l'anno

SCARICA EXCEL

| INDIRIZZO FILIALE | Gennaio | Febbraio | Marzo | Aprile | Maggio | Giugno | Luglio | Agosto | Settembre | Ottobre | Novembre | Dicembre | Totale |
|---------------------------------|---------|----------|-------|--------|--------|--------|--------|--------|-----------|---------|----------|----------|--------|
| Via Vincenzo Piazza Martini, 45 | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € |
| Via Vincenzo Piazza Martini, 46 | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € |
| Via Vincenzo Piazza Martini, 47 | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € |
| Via Vincenzo Piazza Martini, 48 | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € |
| Via Vincenzo Piazza Martini, 49 | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € |
| Via Vincenzo Piazza Martini, 50 | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € |
| Via Vincenzo Piazza Martini, 51 | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € |
| Via Vincenzo Piazza Martini, 52 | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € |
| TOTALE | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € | - € |

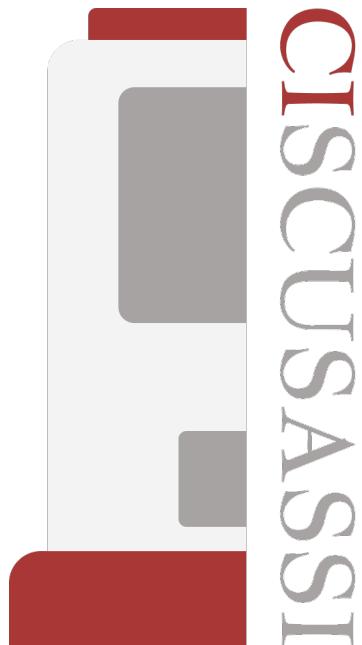
© CISCUSASSI 2025

LEONARDO GIOVANNI CAIEZZA - DIEGO CORONA - LUCA GAETANI - DANTELE ORAZIO SUSINO

Tutti i diritti riservati

7 Copyright e diritto d'autore

La presente **documentazione** è protetta dalle **leggi sul diritto d'autore**. Nessuna parte di questo documento può essere riprodotta, distribuita o trasmessa in alcuna forma o con alcun mezzo, elettronico o meccanico, inclusa la fotocopia, la registrazione o altri sistemi di memorizzazione o recupero di informazioni, senza il **previo consenso scritto degli autori**.



Ciscusassi © 2025

Leonardo Giovanni Coazza

Diego Corona

Alice Gaetano

Daniela Rizzo Scammarco