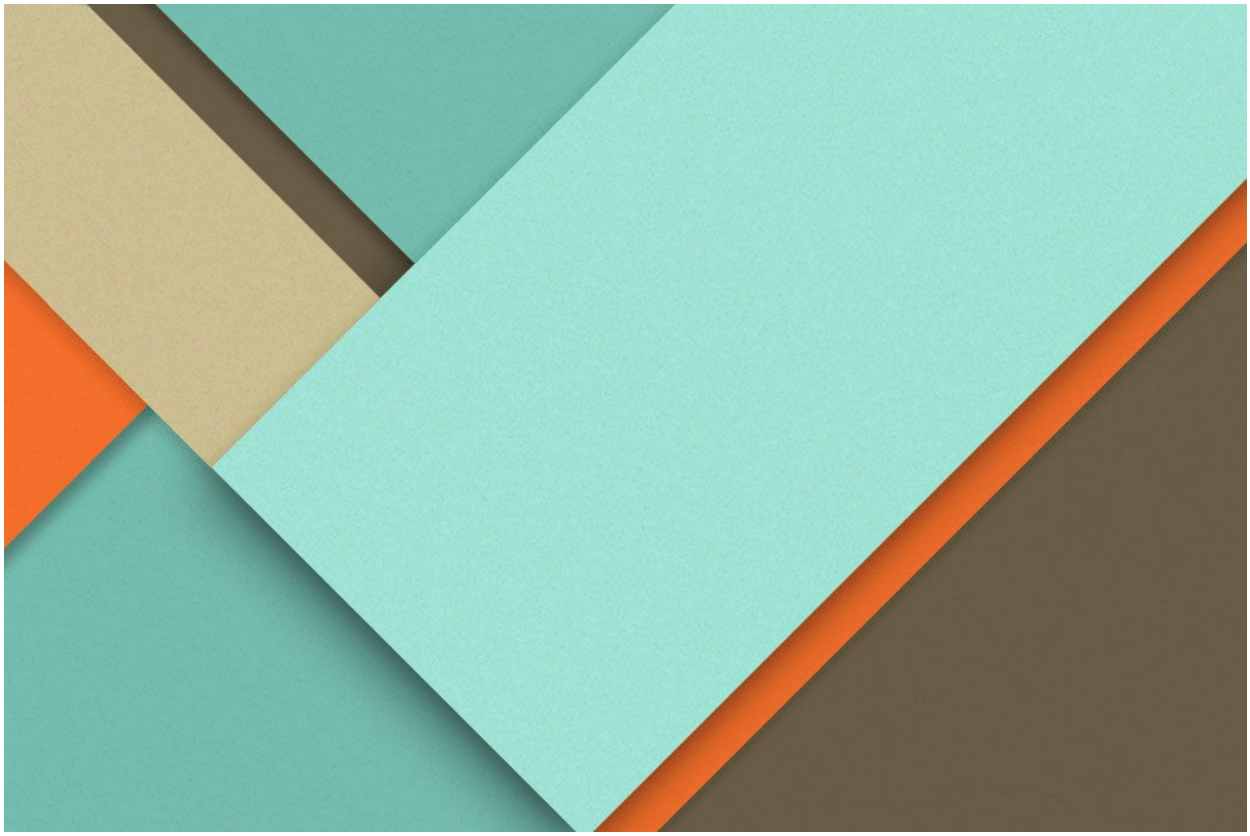


Restaurant App

Scodeller Giovanni 864906
Weeb Dev
Unversità Ca' Foscari Venezia

Overview

Restaurant App è un'applicazione multipiattaforma (web, desktop e mobile) creata per facilitare la gestione di un'attività ristorativa. L'applicazione può essere installata su i dispositivi con cui si ritiene più opportuno lavorare, è stata pensata per un utilizzo su tablet e/o mobile per i cameriere, tablet e/o desktop per i barman e cuochi e desktop e/o web per la cassa.

Inoltre per l'utilizzo dell'applicazione è necessaria una connessione a internet per la connessione al server ed il database utilizzati per il salvataggio dei dati e la sua gestione.

Architettura

L' architettura del sistema è divisa fondamentalmente in due parti:

- Un ***Backend*****:** scritto in **javascript** con l'utilizzo di **API REST** il tutto sotto l'ambiente **Node.js**, il routing viene gestito tramite **Express.js**[1]**.** Per il salvataggio dei dati viene utilizzato il **DBMS MongoDB** sulla piattaforma Atlas.
- Un ***Frontend*****:** realizzato in stile **SPA** (Single Page Application) creato con l'utilizzo di **Angular**[2].

  L'applicazione web inoltre può essere esportata in client desktop tramite l'utilizzo di **Electron** e in un'applicazione mobile utilizzando **Apache Cordova.**

Ogni utente una volta loggato vedrà un'interfaccia utente diversa rispetto al ruolo che gli viene assegnato (chef, bartender, waiter o cashier), inoltre è stato anche implementato un sistema di notifica, attraverso dei **socket**[3], ai camerieri quando i piatti o le bevande sono pronte per essere servite al tavolo.

Modello dei dati

Per la gestione ed il salvataggio dei dati è stato scelto il DBMS MongoDB e viene utilizzato **mongoose**[4] per facilitare l'utilizzo per la connessione, le query e la creazione dei modelli.

I documenti per essere inseriti nel database inoltre devono passare il controllo di **Joi**[5] una libreria di node.js che abbiamo utilizzato per la validazione dei dati.

[1] Express.js: https://expressjs.com/
[2] Angular: https://angular.io/
[3] Socket.io: https://socket.io/
[4] Mongoose: https://mongoosejs.com/
[5] hapi/joi library: https://github.com/hapijs/joi

Descrizione dei documenti

Ora verranno descritte le collection specificando le strutture dei relativi document (i campi "*_id*" e "*__v*" vengono autogenerati da MongoDB):

- **Users:** In questa collection sono contenuti tutti gli utenti del sistema.


```
_id: ObjectId("5d4d6595f07bac3d843ec30c")
totalPlate: 0
dailyPlate: 16
name: "chef"
email: "chef@gmail.com"
password: "$2b$10$sQET3XiUqHfFPooTG7r.9u33/9Pa9GsivQuFPXr.KYxy9O8L1/cPy"
role: "chef"
userId: 4
__v: 0
```


  - **name** (String): Nome dell'utente.
  - **email** (String): Email utilizzata per la registrazione ed il login.
  - **password** (String): La password dell'utente salvata dopo essere stata cryptata.
  - **role** (String): Il ruolo che viene assegnato all'utente, esso può essere: chef, bartender, waiter o cashier.
  - **dailyPlate** (Number): Numero di piatti serviti, nel caso del cameriere, o preparati nel caso del cuoco o barman.
  - **totalPlate** (Number): Numero totale di piatti che sono stati serviti o preparati da quell'utente.
  - **userId** (Number): Id numerico per facilitare la gestione.

- **Tables:** Collection contenente i tavoli del ristorante.


```
_id: ObjectId("5d361e9bee67d03428ed916b")
free: true
seats: 5
number: 11
waiter: ""
__v: 0
```


  - **number** (Number): Numero del tavolo, viene anche utilizzato come id poichè consideriamo che non possa esistere un altro tavolo con lo stesso numero.

- **free** (Boolean): Serve ad identificare se il tavolo è libero oppure già occupato da dei commensali.
- **seats** (Number): Numero di posti a sedere nel tavolo.
- **waiter** (String): Cameriere che impegna il tavolo.

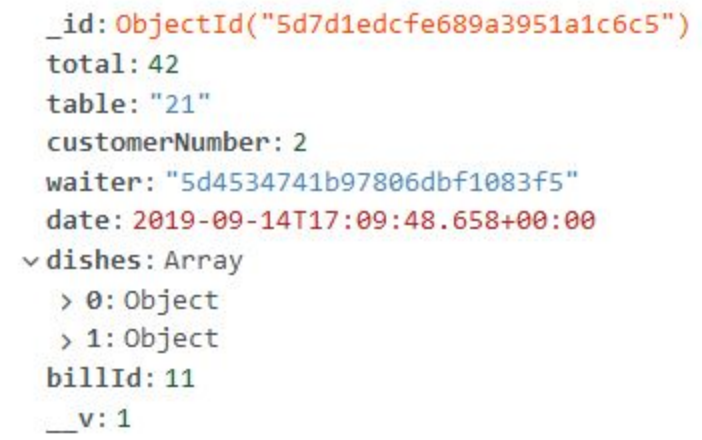
- **Courses:** In questa collection viene racchiuso il menù del ristorante con i piatti che sono divisi per le rispettive categorie.

- **category** (String): Categoria della tipologia del piatto, attualmente sono: antipasti, primi, secondi e bevande.
- **dishes** (Dish[]): Vengono descritti nel dettaglio i piatti che fanno parte di quella categoria.
    - **Dish:** Descrizione del singolo piatto:
        - **name** (String): Nome del piatto.
        - **price** (Number): Prezzo del piatto.
        - **preparation** (Number): Tempo di preparazione del piatto in minuti.
        - **dishId** (Number): Id numerico per facilitare la gestione.
        - **category** (String): Categoria del piatto.
        - **quantity** (Number): Viene utilizzato per descrivere la quantità che viene ordinata dal tavolo.
        - **status** (Number): Utilizzato per descrivere lo stato del piatto:
            - 0, l'ordine è stato creato.
            - 1, l'ordine sta venendo preparato.
            - 2, l'ordine è pronto per essere servito.

- **Orders:** La collection racchiude tutti gli ordini che vengono acquisiti dai camerieri.

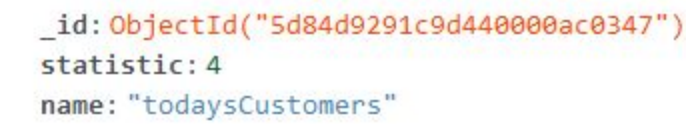
  - **dishes** (Dish[]): Array di piatti e bevande che appartengono a quel particolare ordine, i piatti contengono anche l'informazione della quantità che è stata ordinata.
  - **foodStatus** (Number): Stato dell'ordine per quanto riguarda le pietanze, viene codificato nel seguente modo:
    - 0, l'ordine è stato creato.
    - 1, l'ordine è in preparazione.
    - 2, l'ordine è pronto per essere servito al tavolo.
    - 3, l'ordine è stato servito al tavolo.
  - **drinkStatus** (Number): Simile a foodStatus, contiene lo stato delle bevande dell'intero ordine.
  - **table** (Number): Numero del tavolo a cui appartiene all'ordine.
  - **waiter** (String): Id del cameriere che sta seguendo e servirà il tavolo.
  - **orderId** (Number): Id numerico per facilitare la gestione.

- **Bills:** In questa collection sono contenute tutte le ricevute battute dalla cassa.



- **total** (Number): Il totale espresso in euro del costo del pasto.
- **table** (Number): Numero del tavolo dove erano seduti i clienti.
- **customerNumber** (Number): Numero effettivo delle persone che si sono sedute al tavolo. Si è presupposto che un numero inferiore di clienti rispetto alla capienza massima può sedersi lo stesso in quel tavolo.
- **waiter** (Number): Id del cameriere che ha servito il tavolo.
- **date** (Date): Orario in cui lo scontrino viene creato.
- **dishes** (Dish[]): Tutti i piatti che sono stati ordinati nel corso del pasto.
- **billId** (Number): Id numerico per facilitare la gestione.

- **Statistics:** Utilizzata per tenere traccia delle statistiche del ristorante.



- **todaysProfit** (Number): Profitto totale della giornata lavorativa.
- **todaysCustomers** (Number): Numero totale dei clienti che hanno mangiato al ristorante.

REST API

Di seguito sono descritti tutti gli endpoint utilizzati per creare, gestire e organizzare le collection e le interazioni client-server. L'indirizzo del server in locale è il seguente http://localhost:3000/.

/users/login

Method	Body	Query Params	Description	Users
POST	{email:String, password:String}		Utilizzato per il login dell'utente	

/users/register

Method	Body	Query Params	Description	Users
POST	{email:String, password:String, role: String}		Utilizzato per registrare un utente	

/users/:id

Method	Body	Query Params	Description	Users
GET		{id: Number}	Ricevi il dipendente :id	Cassa
DELETE		{id: Number}	Cancella il dipendente :id	Cassa

/users/refresh-token

Method	Body	Query Params	Description	Users
POST	{AccessToken: String}		Ricevi il dipendente :id	Cassa Cameriere Barman Chef

/bills

Method	Body	Query Params	Description	Users
GET			Ricevi tutti i conti	Cassa
POST	{ table: Number, customer: Number}		Crea un nuovo conto vuoto per il tavolo Tablenumber, occupandolo	Cameriere

/bills/:id

Method	Body	Query Params	Description	Users
GET		{id: Number}	Ricevi il conto :id	Cassa

/bills/:tableId

Method	Body	Query Params	Description	Users
POST	{total: Number, dishes: Dish[]}	{tableId: Number}	Ricevi il conto :id	Cassa

/orders

Method	Body	Query Params	Description	Users
GET			Ricevi tutti gli ordini in base al ruolo assegnato	Barman Cuoco Cameriere
POST	{table: Number, dishes: Dish[] }		Posta un ordine di piatti nel tavolo specificato	Cameriere

/orders/:id

Method	Body	Query Params	Description	Users
GET		{id: Number}	Ricevi l'ordine :id	Cassa Cameriere Barman Chef
POST	{ food*: FoodStatus, drink*: DrinkStatus }		Cambia lo stato dell'ordine	Barman Chef Cameriere

/orders/tables/:id

Method	Body	Query Params	Description	Users
GET		{table: Number}	Ricevi l'ordine del tavolo :table	Cassa Cameriere Barman Chef

/orders/:id/:dish

Method	Body	Query Params	Description	Users
GET			Ricevi il piatto :dish dell'ordine :id	Tutti
POST	{status:DishStatus}		Cambia lo stato del piatto :dish dell'ordine :id	Barman Cuoco

/courses

Method	Body	Query Params	Description	Users
GET			Ricevi la lista di menù	Cassa Cameriere Barman Chef

/courses/createCourse

Method	Body	Query Params	Description	Users
POST	{category:String}		Crea una nuova categoria per il menù	

/courses/newPlate

Method	Body	Query Params	Description	Users
POST	{Dish}		Crea un nuovo piatto per il menù	

/tables

Method	Body	Query Params	Description	Users
GET			Ricevi tutti i miei tavoli	Cameriere Cassa
POST	{ Table }		Crea un nuovo tavolo	Cassa

/tables/:id

Method	Body	Query Params	Description	Users
GET			Ricevi il tavolo :id	Cassa Cameriere Barman Chef

/tables/freeTables

Method	Body	Query Params	Description	Users
GET			Ritorna tutti i tavoli liberi	Cameriere Cassa

/tables/myTables

Method	Body	Query Params	Description	Users
GET			Ricevi i tavoli che sono impegnati dal quel cameriere	Cameriere

/statistics/daily

Method	Body	Query Params	Description	Users
GET			Ricevi le statistiche della giornata	Cassa

/statistics/user/:id

Method	Body	Query Params	Description	Users
GET		{id: String}	Ricevi le statistiche dell'utente :id	Cassa

Frontend Angular

Il frontend è stato realizzato tramite l'utilizzo di **Angular** che permette di realizzare applicazioni con stile **SPA**. Il framework lavora con i *component* che permettono un facile riutilizzo del codice a seconda dell'utente che è loggato. Inoltre per il frontend è stato utilizzato **Angular Material**[6] per migliorare l'interfaccia utente e renderne più facile l'utilizzo.

Routes

I components sono gestiti dal seguente routing:

```
const routes: Routes = [
  {
    path: "",
    redirectTo: "login",
    pathMatch: "full"
  },
  {
```

[6] Angular Material: https://material.angular.io/

```
  path: "login",
  component: LoginComponent,
  pathMatch: "full"
},
{
  path: "logout",
  component: LogoutComponent,
  pathMatch: "full"
},
{
  path: "waiter/new-table",
  canActivate: [AuthGuard],
  data: {
    expectedRole: "waiter"
  },
  component: WaiterNewTableComponent
},
{
  path: "waiter/new-order/:id",
  canActivate: [AuthGuard],
  data: {
    expectedRole: "waiter"
  },
  component: WaiterNewOrderComponent
},
{
  path: "waiter",
  component: WaiterDashboardComponent,
  canActivate: [AuthGuard],
```

```
    data: {
      expectedRole: "waiter"
    },
    children: [
      {
        path: "",
        component: DummyComponent,
        pathMatch: "full"
      },
      { path: ":id", component: WaiterTableDetailComponent },
      { path: "**", redirectTo: "" }
    ]
  },
  {
    path: "bartender",
    component: BartenderDashboardComponent,
    canActivate: [AuthGuard],
    data: {
      expectedRole: "bartender"
    },
    children: [
      {
        path: "",
        component: DummyComponent,
        pathMatch: "full"
      },
      { path: ":id", component: BartenderOrderDetailComponent },
      { path: "**", redirectTo: "" }
    ]
```

```
},
{
  path: "chef",
  component: ChefDashboardComponent,
  canActivate: [AuthGuard],
  data: {
    expectedRole: "chef"
  },
  children: [
    {
      path: "",
      component: DummyComponent,
      pathMatch: "full"
    },
    { path: ":id", component: ChefOrderDetailComponent },
    { path: "**", redirectTo: "" }
  ]
},
{
  path: "cashier",
  canActivate: [AuthGuard],
  data: {
    expectedRole: "cashier"
  },
  children: [
    { path: "", redirectTo: "dashboard", pathMatch: "full" },
    { path: "dashboard", component: CashierDashboardComponent },
    { path: "tables/:table", component: CashierViewOrderComponent },
    { path: "tables/:table/bill", component: CashierBillComponent },
```

```
      { path: "statistics", component: CashierStatisticsComponent },
      {
        path: "statistics/users/new",
        component: CashierNewUserComponent
      },
      {
        path: "statistics/users/:user",
        component: CashierStatisticsUserComponent
      },
      { path: "**", redirectTo: "dashboard" }
    ]
  },
  { path: "**", redirectTo: "logout" }
];
```

Il primo component che viene renderizzato quando l'applicazione viene aperta è il *LoginComponent*, inoltre se l'utente ha un token non valido o scaduto viene reindirizzato nella pagina di login.

Components

Nell'applicazione possiamo distinguere dei macrocomponent a seconda del ruolo con cui si ha effettuato l'accesso.

Sottostante sono riportati tutti i component utilizzati nell'applicazione (posso essere trovati nella cartella "frontend-angular-app/src/app/components"):

- bartender
  - bartender-dashboard
  - bartender-order-detail
- cashier
  - cashier-bill
  - cashier-dashboard
    - cashier-new-user
    - cashier-statistics
      - cashier-statistics-user
    - cashier-view-orders

- chef
    - chef-dashboard
        - chef-order-detail
- dummy: viene utilizzato come placeholder
- header: comune a tutti
- login: component dove viene effettuato il login
- logout: serve per poi reinderizzare al login
- order-list
- sidenav: comune a tutti gli utenti, il contenuto dipende dal ruolo
- snackbar: viene utilizzata per i messaggi
- waiter
    - waiter-dashboard
        - waiter-new-order
        - waiter-new-table
        - waiter-table-detail
        - Waiter-table-list

Services

Vengono utilizzati in **Angular** per poter comunicare con il *backend* e eseguire le operazioni CRUD[7].

Abbiamo utilizzato i seguenti *services* per gestire al meglio l'applicazione (tutti i services sono raggruppati sotto la cartella "frontend-angular-app/src/app/services"):

- auth-guard: utilizzato per controllare gli accessi
- auth: utilizzato per il login
- bill: contiene i servizi necessari per gestire le ricevute
- dish: utilizzato per i menu
- notification: viene utilizzato per gestire le notifiche
- order: contiene tutti i servizi per gestire gli ordini
- socket: servizio creato per inizializzare i socket
- statistics: viene utilizzato dalla cassa per la visualizzazione delle notifiche
- table: viene utilizzato per accedere agli endpoint dei tavoli
- user: utilizzato dalla cassa per la gestione degli utenti
- utils: contiene servizi generici come l'impostazione del titolo, sidebar e progress bar

[7] CRUD: Create Read Update Delete

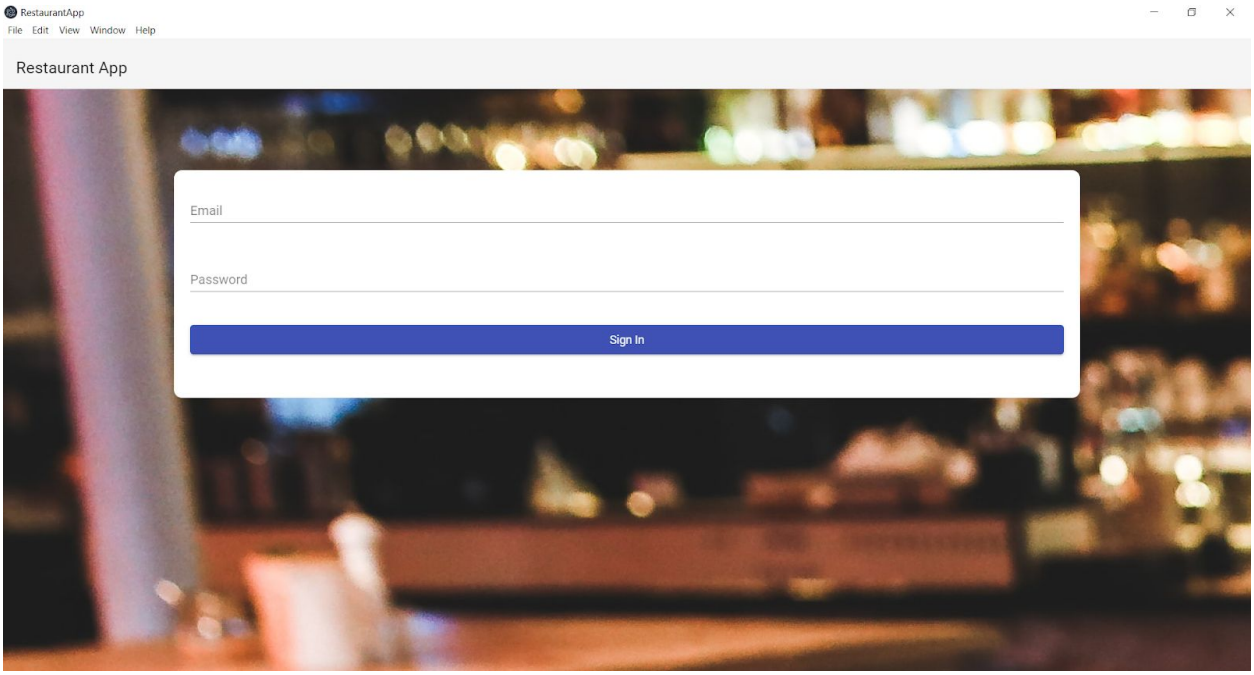
Workflow

Workflow Autenticazione con chef

Per mostrare il workflow dell'autenticazione possiamo anche descrivere il workflow di una giornata tipica lavorativa di uno chef, ipotizziamo che la sua postazione lavorativa è costituita da un computer con schermo touch.

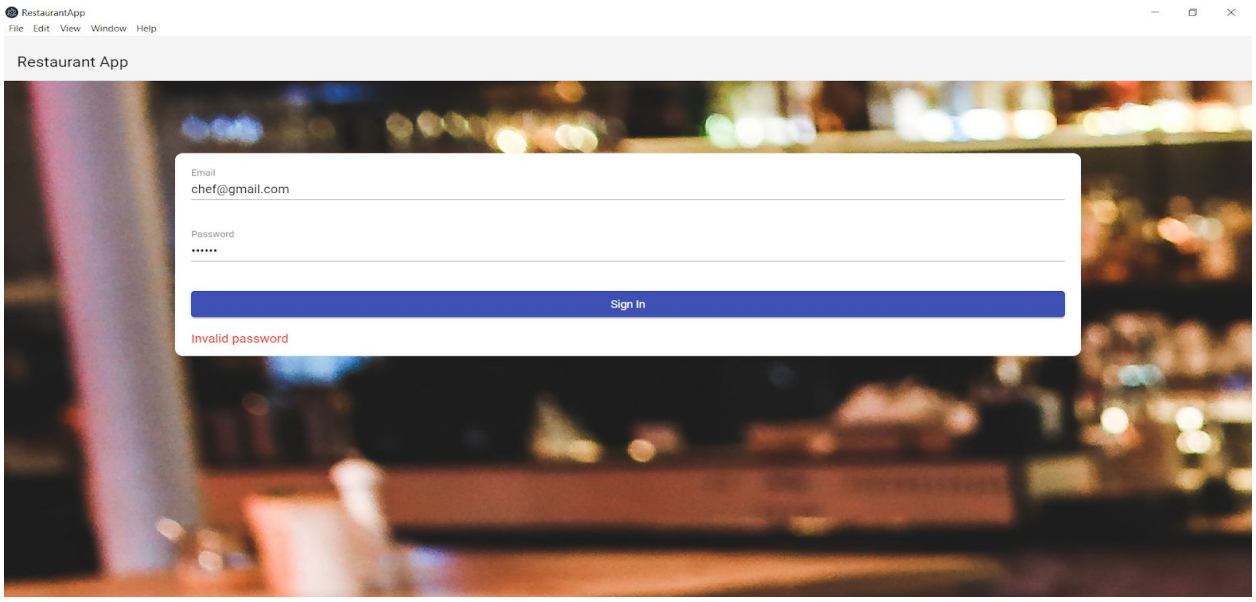
Quando viene avviata l'applicazione desktop questa è la prima schermata che l'utente vede.

In questa schermata l'utente già registrato potrà effettuare il login inserendo le proprie credenziali, nel caso *email* e *password* siano corrette verrà creato un **JWT**[8] che utilizzerà per le autenticazioni con il backend e potrà accedere alla schermata successiva che dipenderà dal ruolo che ha.

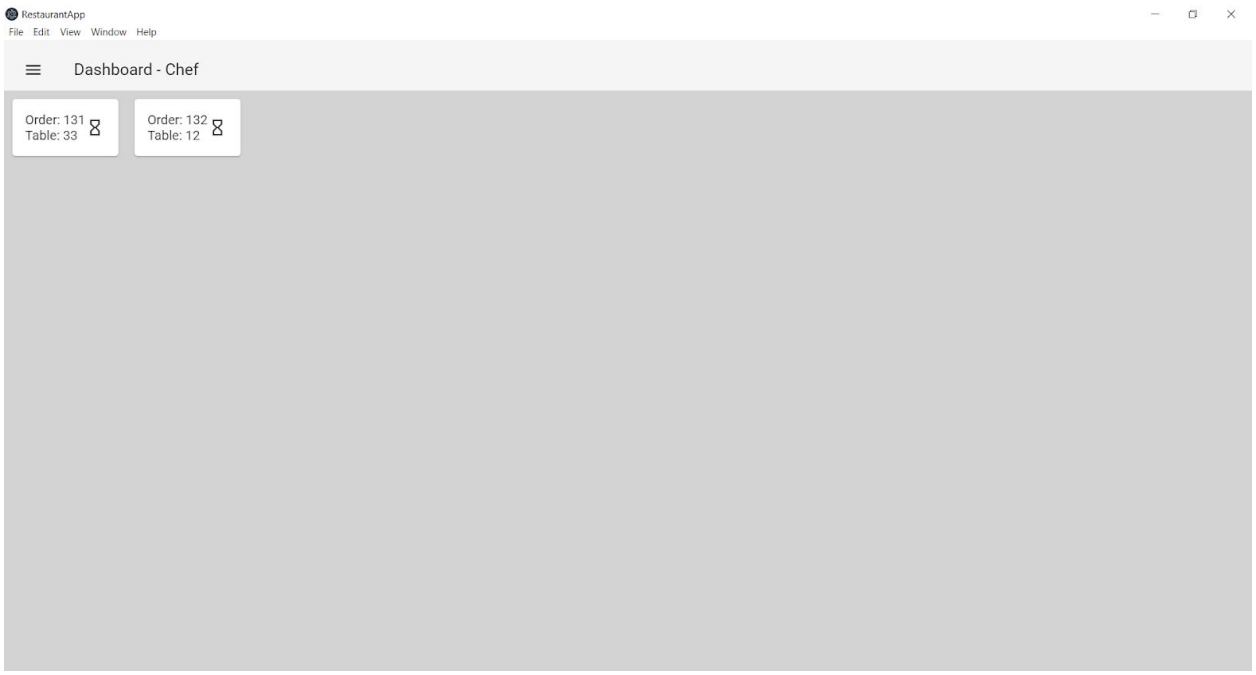


[8] JWT: JSON Web Token, standard per gestire l'autenticazione basata su token.

Nel caso vengono inserite delle credenziali sbagliate viene notificato all'utente che il login non è andato a buon fine, come nell'immagine seguente. In questo modo l'utente può ritentare di fare il login.

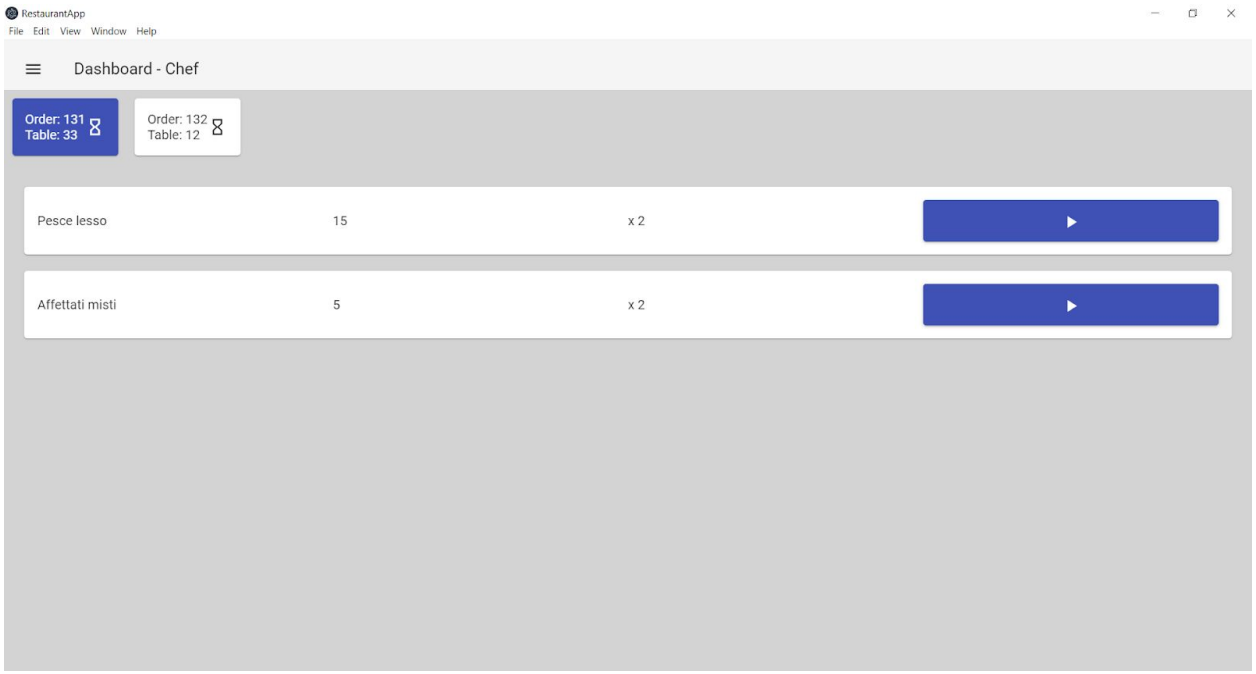


Una volta effettuato il login con successo l'utente passerà alla schermata principale che corrisponde alla dashboard del ruolo a cui appartiene, in questo caso alla dashboard del cuoco.



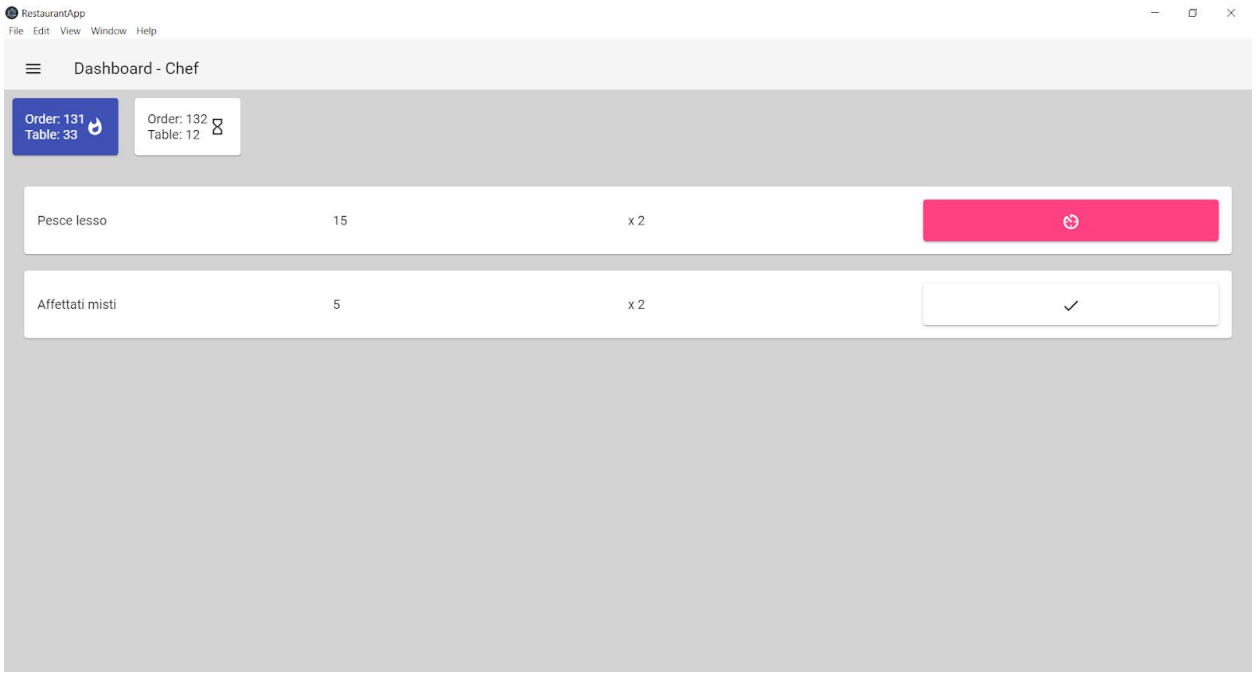
In questo caso lo chef visualizza gli ordini delle pietanze che sono state richieste dai clienti in sala, sono divisi in una coda dove il primo ordine che viene preparato è il primo ordine che viene ricevuto.

Cliccando sopra l'ordine lo chef visualizzerà i piatti che dovrà preparare con il relativo tempo di preparazione.



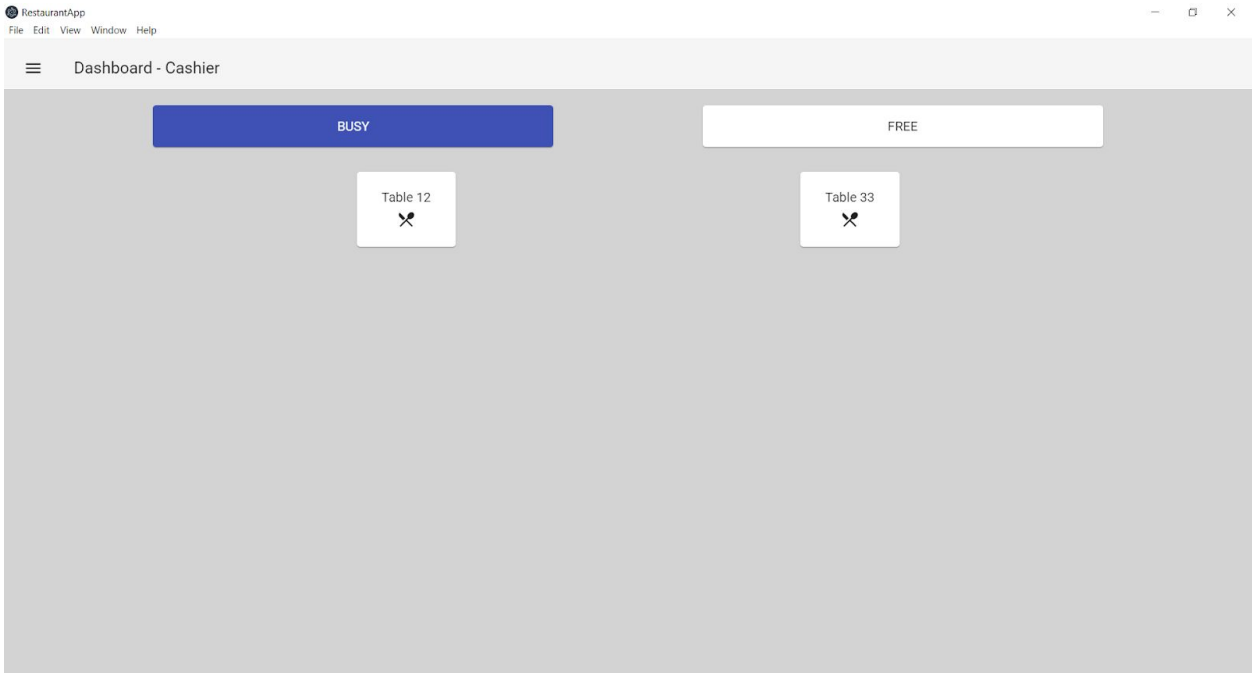
In questo modo quando vuole potrà iniziare il piatto cliccando sul bottone della pietanza ed una volta completata ricliccando sul bottone lo contrassegnerà come finito. Quando tutte le pietanze sono state preparate l'ordine viene rimosso dal cuoco e al cameriere che deve servire quel tavolo viene inviata una notifica per segnalargli che i piatti possono essere portati al tavolo.

Nell'immagine seguente si può vedere che il primo piatto è contrassegnato come *in preparazione* ed il secondo *finito.*

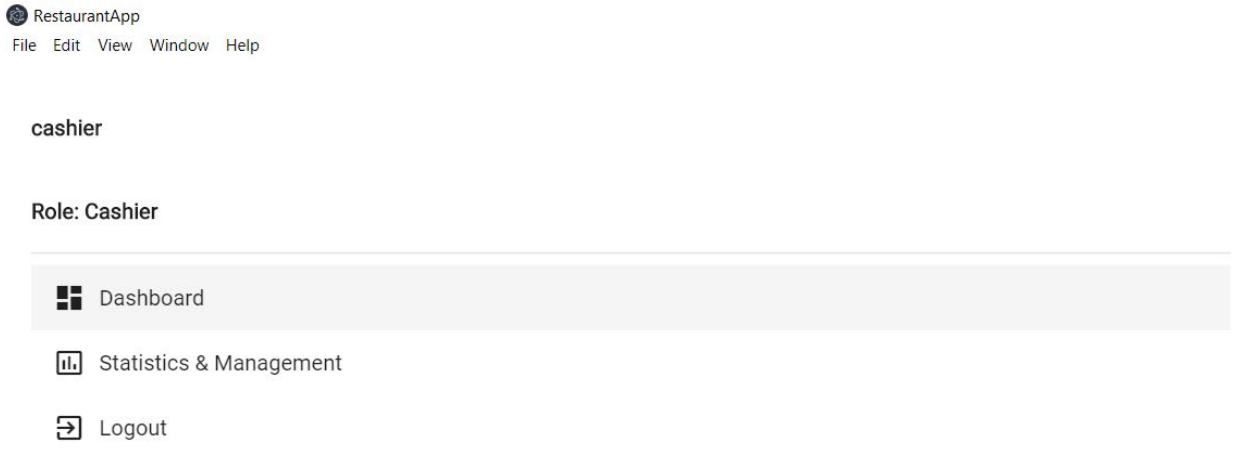


Workflow cassiere

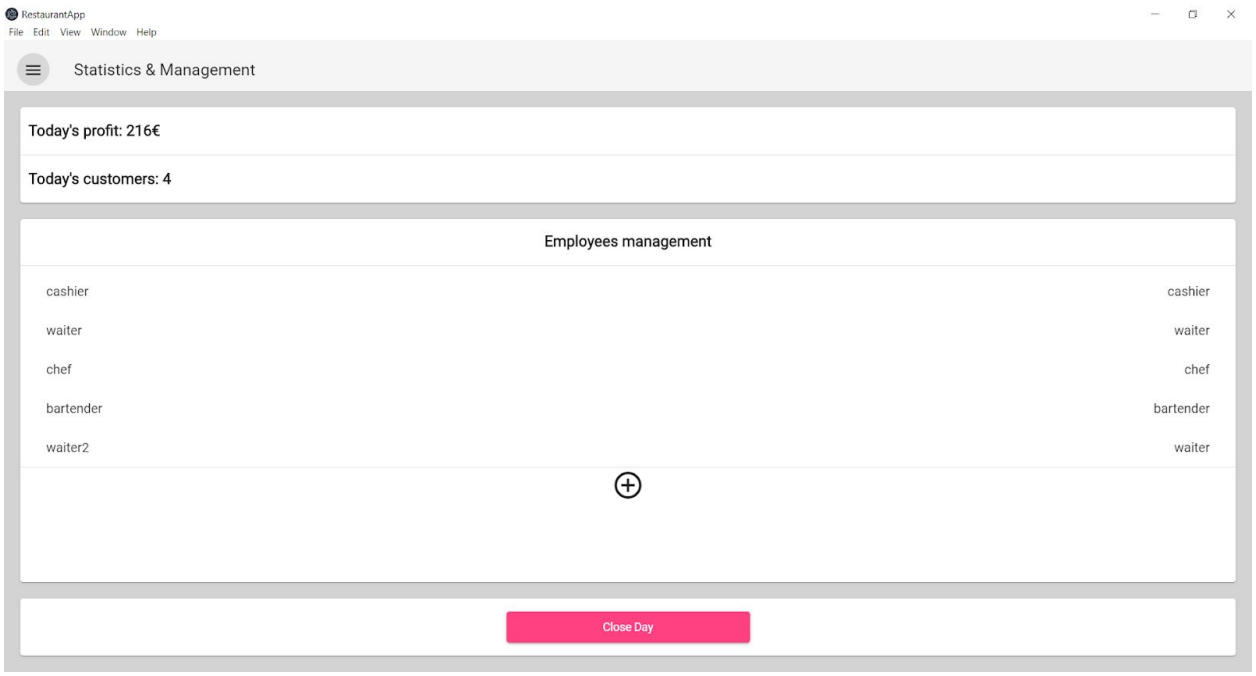
Dopo aver eseguito con successo l'autenticazione il cassiere si troverà davanti alla propria dashboard dove può già visualizzare lo stato della sala dove vengono indicati i tavoli liberi ed i tavoli occupati.



Inoltre cliccando in alto a destra il cassiere potrà accedere alla gestione degli utenti e alla visualizzazione delle statistiche giornaliere.



Se accediamo alla voce *Statistics & Management* verrà visualizzata questa schermata.



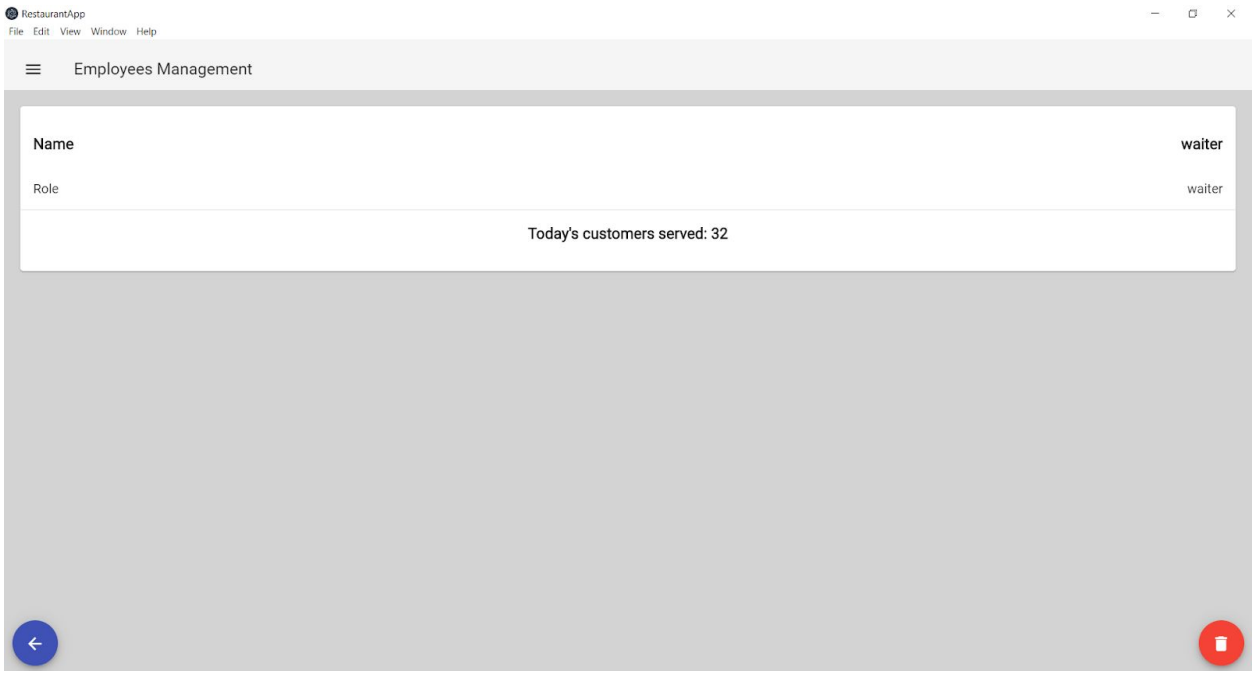
Qui è possibile visualizzare vari dati, tra i quali:

- Il profitto totale della giornata
- Quanti clienti sono stati serviti
- La lista di tutti gli utenti del sistema

Inoltre è possibile chiudere la giornata lavorativa del ristorante resettando le statistiche e cliccando il bottone + è possibile aggiungere nuovi utenti al sistema.

Di seguito viene riportata la schermata dell'aggiunta di un nuovo utente al sistema. La creazione è resa effettiva quando viene cliccato il bottone in basso a destra.

Qui sotto è riportata una semplice schermata di statistiche di un singolo impiegato.



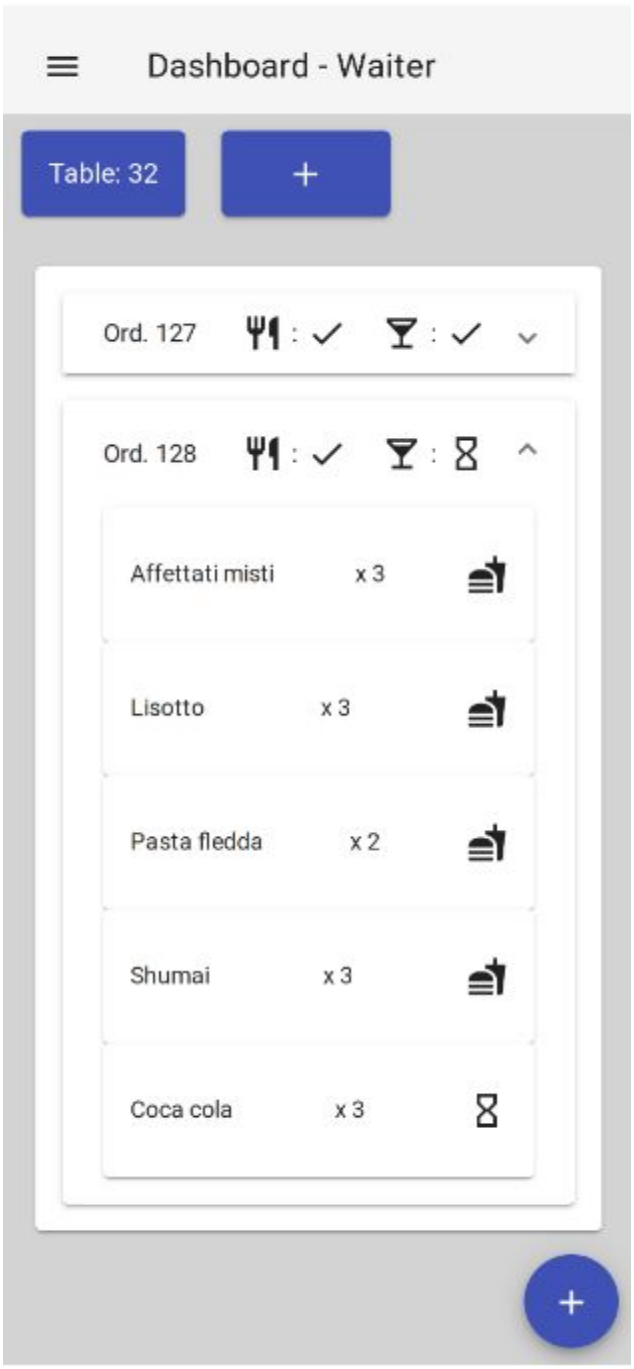
Inoltre c'è la possibilità di eliminare l'utente cliccando il bottone in basso a destra.

Workflow cameriere

Per mostrare il workflow del cameriere utilizziamo l'interfaccia mobile dell'applicazione.

Dopo aver eseguito l'accesso il cameriere si troverà nella dashboard dell'applicazione dove troverà i suoi tavoli, nel caso ne stia già seguendo qualcuno, altrimenti sarà vuota.

In questo caso il cameriere sta già servendo il tavolo 32, infatti è possibile visualizzare i due ordini che i clienti hanno già effettuato. Cliccando sopra ad un ordine inoltre apparirà in dettaglio ciò che stato ordinato.

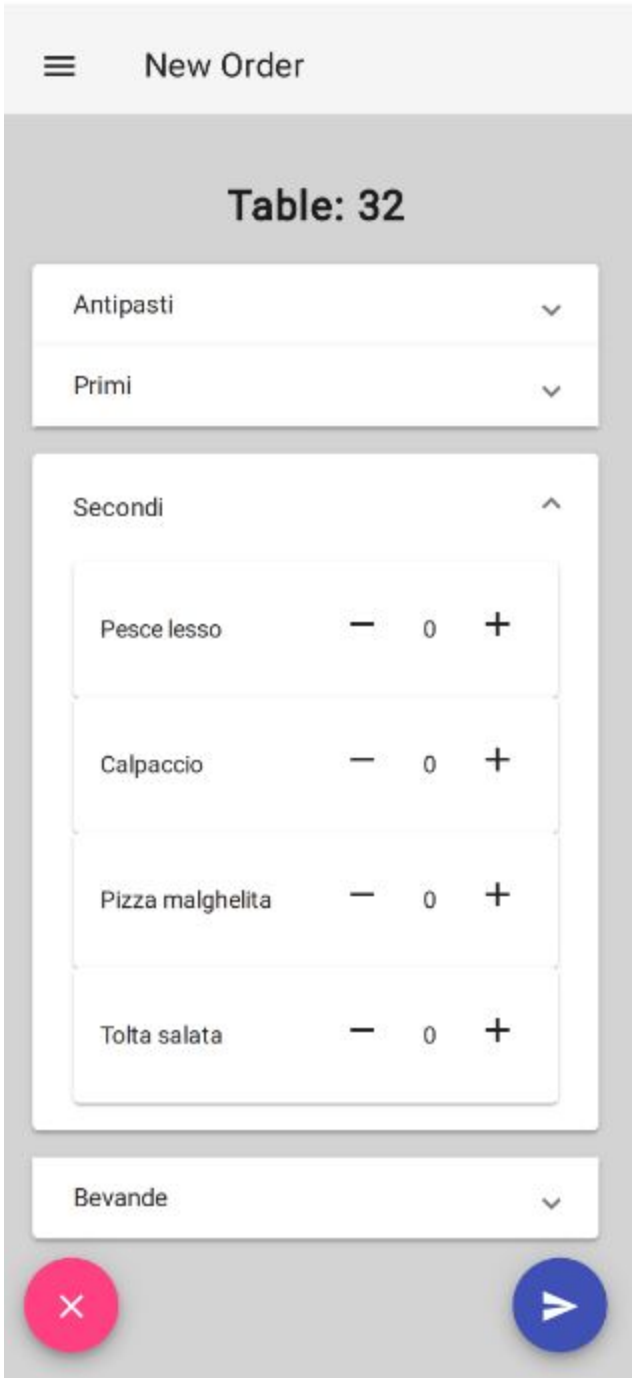


Si può osservare che nel dettaglio dell'*ordine 128* tutte le pietanze sono state completate e si può osservare dall'immagine a destra della quantità che ne contraddistingue lo stato.

Quando tutte i piatti o tutte le bevande sono pronte per essere servite arriva una notifica al cameriere che viene visualizzata in alto a destra, inoltre compare un ulteriore notifica nell'ordine stesso per mostrare se bisogna andare a prendere le pietanze o le bevande rispettivamente dal barman o dal cuoco.

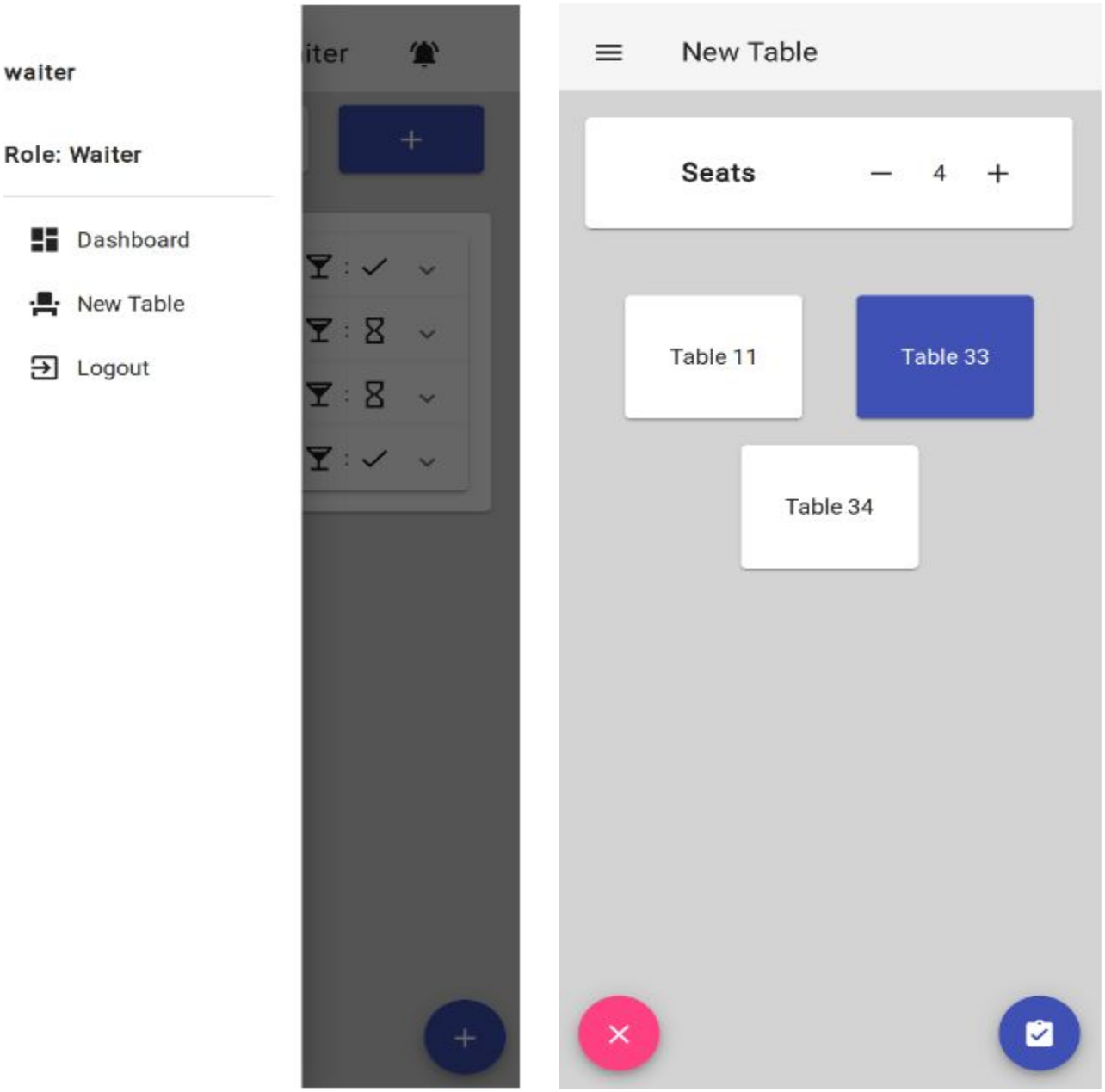
Basta cliccare nella notifica vicino alla *forchetta e coltello* per confermare la consegna dei piatti al tavolo.

Il cameriere può effettuare un nuovo ordine in qualsiasi momento cliccando il pulsante + in basso a destra. Comparirà la schermata con il menù diviso per categorie e si potrà scegliere quale pietanza e la quantità da ordinare.



Inoltre il cameriere può accomodare nuovi clienti nel ristorante. Quando arriveranno e verranno accolti gli verrà chiesto il numero di persone che mangerà al ristorante, in questo modo il cameriere potrà inserire il numero di persone e scegliere il tavolo più opportuno.

Per aggiungere un nuovo tavolo il cameriere può selezionare indifferentemente la voce *new table* nella sidebar, oppure cliccare il tasto + nella lista dei tavoli.



Una volta selezionato il tavolo basta cliccare sulla spunta in basso a destra e verrà contrassegnato come occupato, quel cameriere potrà prendere le ordinazioni e servire anche quel tavolo.