



Università  
Ca'Foscari  
Venezia

Università Ca' Foscari  
Laurea Triennale in Informatica

**Tecnologie e applicazioni web (CT0142)**

**A.A. 2018-19**

**Relazione sulla realizzazione del progetto**

**Pizzeria**

VISENTIN Thomas (869438)

## **Sommario**

<b><u>1.</u></b>	Architettura del sistema.....	pag.2
<b><u>2.</u></b>	Gestione della memoria persistente (MongoDB).....	pag.4
<b><u>3.</u></b>	Lista e descrizione dettagliata degli endpoints.....	pag.8
<b><u>4.</u></b>	Gestione dell'autenticazione degli utenti (Passport).....	pag.14
<b><u>5.</u></b>	Descrizione dettagliata della web app realizzata con Angular.....	pag.16
<b><u>6.</u></b>	Screenshot e workflow dell'applicazione.....	pag.18

# 1. Architettura del sistema

Il sistema implementato al fine di soddisfare i requisiti di progetto risulta essere composto da 4 parti fondamentali, di seguito illustrate:

- **Server:** sul quale gira Node.js, una runtime di JavaScript Open source multiplatforma orientata agli eventi per l'esecuzione di codice JavaScript Server-side.  
Per la gestione degli endpoint è stato usato il framework Express, il quale consente di gestire agevolmente i vari parametri delle richieste ricevute dal client (es. metodo, header e body della richiesta).  
Il framework Passport è stato invece usato per gestire l'autenticazione (bearer authentication) come illustrato nel dettaglio nella sezione 5.  
La libreria jsonwebtoken per la generazione e la gestione del token usato dalla bearer authentication, in quanto tale libreria consente facilmente di rendere "sicuro" il token firmandolo.  
La libreria Socket.io per la "comunicazione" in tempo reale tra gli utenti, in quanto permettendo tale libreria una comunicazione bidirezionale tra client e server, consente che i dati visualizzati dagli utenti siano sempre quelli "più aggiornati".
- **Client:** che non è altro che un'interfaccia grafica richiamante le API del server, creata con Angular e responsiva grazie al framework Bootstrap, anche qui viene utilizzata la libreria Socket.io per comunicare con il server.
- **Applicazione mobile (Android):** creata con Apache Cordova, che consiste sostanzialmente in un browser "bloccato" sulle view della web app Angular, che tuttavia dà l'impressione all'utente di avere a che fare un'app vera e propria con un'interfaccia identica a quella client visualizzabile con un normalissimo browser.
- **Programma desktop:** creato con il tool Electron, che consiste in un file eseguibile (.exe per Windows) con al suo interno una versione "ridotta" del browser Google Chrome, che consente la visualizzazione delle sole pagine riguardanti la web app. Tale soluzione risulta molto adatta ad attività commerciali come appunto pizzerie, in quanto potrebbe essere installata come unica applicazione nel device interessato, bloccandolo, ovvero permettendo di usarlo a soli scopi "lavorativi".

**N.B.: Così come scritto nel file README.txt, il progetto consegnato così come configurato funziona nel seguente modo:**

**Le parti di progetto contenute nelle cartelle lato\_server e lato\_client lavorano "insieme" (il server resta in ascolto sul localhost:80 mentre il lato client è raggiungibile all'indirizzo localhost:4200).**

**Le parti contenute nelle cartelle Mobile e Desktop invece, "fanno" le loro richieste in un'istanza del server raggiungibile all'indirizzo IP: 80.211.139.226:80 (VPS Aruba).**

**Tale scelta è dovuta al fatto di voler rendere “indipendenti” l’applicazione mobile e desktop dal localhost, in modo da poterle testare in qualsiasi momento.**

## 2. Gestione della memoria persistente

Come da direttive del progetto, per la gestione della memoria persistente è stato usato il DBMS non relazionale (NoSql) MongoDB.



In particolare le “tabelle” del database (chiamate “collections”) sono state strutturate nel seguente modo:

Elenco_tavoli	
_id	ObjectId
numero	number
posti	number

La “collection” Elenco\_tavoli, contiene il campo \_id (generato automaticamente dal DBMS), il campo numero che rappresenta il numero del tavolo e il campo posti che indica il numero massimo di commensali che possono pranzare/cenare in quel tavolo.

Listino_pizze	
_id	ObjectId
nome	string
prezzo	number
descrizione	string
tempo_di_preparazione	number

La “collection” Listino\_pizze rappresenta i tipi di pizza ordinabili, contiene il campo \_id (generato automaticamente dal DBMS), il campo nome che rappresenta il nome della pizza e il campo prezzo che indica il relativo prezzo, il campo descrizione che contiene usualmente ed eventualmente gli ingredienti della pizza.

Bibite	
_id	ObjectId
nome	string
prezzo	number
descrizione	string

La “collection” Bibite rappresenta le bevande ordinabili, contiene il campo *id* (generato automaticamente dal DBMS), il campo *nome* che rappresenta il nome della bibita e il campo *prezzo* indica il relativo prezzo e il campo *descrizione* che contiene eventualmente la quantità o altre caratteristiche della bevanda.

Elenco_aggiunte	
<u><i>_id</i></u>	ObjectId
<u><i>nome</i></u>	string
<u><i>prezzo</i></u>	number
<u><i>descrizione</i></u>	string
<u><i>pizza_bibita</i></u>	boolean

La “collection” Elenco\_aggiunte rappresenta le aggiunte disponibili sia per le bevande che per le pizze, contiene il campo *id* (*generato automaticamente dal DBMS*), il campo *nome* che rappresenta il nome dell’aggiunta e il campo *prezzo* indica il relativo prezzo da sommare al prodotto a cui viene aggiunta, il campo *descrizione* che contiene eventualmente altre caratteristiche dell’aggiunta e il campo *booleano* *pizza\_bibita* che indica se l’aggiunta riguarda le pizze o le bibite.

Ordini	
<u><i>_id</i></u>	ObjectId
<u><i>tavolo</i></u>	number
<u><i>persone_da_servire</i></u>	number
<u><i>ordine_completato</i></u>	boolean
<u><i>stato_bibite</i></u>	number
<u><i>stato_pizze</i></u>	number
<u><i>ordine_pagato</i></u>	boolean
<u><i>ora_ordine</i></u>	datetime
<u><i>cameriere</i></u>	string
<u><i>barista</i></u>	string
<u><i>cassiere</i></u>	string

Legenda:

	stato_bibite	stato_pizze
<b>0</b>	Bibite non ancora in preparazione	Pizze non ancora pronte
<b>1</b>	Bibite in preparazione	Pizze pronte
<b>2</b>	Bibite pronte	Pizze consegnate
<b>3</b>	Bibite consegnate	//

La “collection” Ordini rappresenta le aggiunte disponibili sia per le bevande che per le pizze, contiene il campo id (*generato automaticamente dal DBMS*), il campo tavolo che rappresenta il tavolo a cui si riferisce l'ordine e il campo persone\_da\_servire indica il numero di persone sedute al tavolo a cui l'ordine si riferisce, il campo booleano ordine\_completato che indica se l'acquisizione dell'ordine è stata completata da parte del cameriere, il campo stato\_bibite che indica lo stato di preparazione delle bibite dell'ordine con il significato indicato nella legenda, il campo stato\_pizze che indica lo stato di preparazione delle pizze dell'ordine con il significato indicato nella legenda, il campo booleano ordine\_pagato che indica se l'ordine è stato pagato, il campo ora\_ordine che indica l'ora di acquisizione dell'ordine, il campo cameriere che indica l'username del cameriere che ha acquisito l'ordine, il campo barista che indica l'username del barista che ha preparato le bibite presenti nell'ordine, il campo cassiere che indica l'username del cassiere che ha riscosso il pagamento dell'ordine.

Prodotti_ordinati	
<u>_id</u>	ObjectId
nome_prodotto	string
aggiunte	Array(string)
prezzo	number
pizzaiolo	string
bibita_pizza	boolean
ordine	string
stato_preparazione	number
tempo_di_preparazione	number

La “collection” Prodotti\_ordinati rappresenta ogni singolo prodotti aggiunto ad un ordine con le eventuali aggiunte, contiene il campo id (*generato automaticamente dal DBMS*), il campo nome\_prodotto che rappresenta il nome del prodotto e il campo aggiunte indica le aggiunte al prodotto, il campo prezzo che contiene il prezzo del prodotto comprensivo delle aggiunte, il campo pizzaiolo che indica l'username del pizzaiolo che ha preparato la pizza (se il prodotto in questione è una pizza), il campo booleano bibita\_pizza che indica se il prodotto in questione è una bibita o una pizza, il campo ordine che indica l'ordine a cui appartiene il prodotto, il campo stato\_preparazione che indica lo stato di preparazione di un certo prodotto, il valore del campo è da interpretarsi con la legenda vista nella collection precedente e il campo tempo\_di\_preparazione che indica il tempo previsto per la preparazione dei prodotti (presente solo se si tratta si pizze).

Users	
<u>_id</u>	ObjectId
username	string
cognome	string
nome	string
ruolo	string
salt	string
digest	string

La “collection” Users gli utenti che con i diversi ruoli e privilegi possono accedere al sistema, contiene il campo id (*generato automaticamente dal DBMS*), il campo username è univoco e rappresenta il “soprannome” che l’utente deve inserire al momento del login, il campo cognome indica il cognome dell’utente, il campo nome che contiene il nome dell’utente, il campo ruolo che può assumere i seguenti valori: “cameriere”, “barista”, “pizzaiolo”, “cassiere” e indica la mansione che ha l’utente nella pizzeria e infine vi sono i campi salt e digest che servono al fine di autenticazione per verificare che la password immessa dall’utente sia corretta.



### 3. Lista e descrizione dettagliata degli endpoint

Endpoint	Metodo	Parametri e/o body	Accessibile a:	Descrizione
/	GET	nessuno	tutti	Restituisce la versione della API e la lista degli endpoint disponibili
/listino_pizze	GET	nessuno	tutti	Restituisce la lista delle pizze disponibili con il loro rispettivo prezzo, descrizione e tempo di preparazione
/listino_bibite	GET	nessuno	tutti	Restituisce la lista delle bibite disponibili con il loro rispettivo prezzo e descrizione
/listino_aggiunte_pizze	GET	nessuno	tutti	Restituisce la lista delle aggiunte disponibili per le pizze e la loro relativa descrizione
/listino_aggiunte_bibite	GET	nessuno	tutti	Restituisce la lista delle aggiunte disponibili per le bibite e la loro relativa descrizione
/elenco_tavoli	GET	nessuno	tutti	Restituisce la lista dei tavoli presenti nel locale con la loro rispettiva capienza

/users_camerieri	GET	nessuno	cassieri	Restituisce la lista dei camerieri registrati nel sistema
/users_pizzaiolo	GET	nessuno	cassieri	Restituisce la lista dei pizzaioli registrati nel sistema
/users_baristi	GET	nessuno	cassieri	Restituisce la lista dei baristi registrati nel sistema
/users_camerieri	GET	nessuno	cassieri	Restituisce la lista dei camerieri registrati nel sistema
/users_cassiere	GET	nessuno	cassieri	Restituisce la lista dei camerieri registrati nel sistema
/renew	GET	nessuno	Qualsiasi utente registrato al sistema	Restituisce un nuovo token corrispondente allo stesso utente che inoltra la richiesta con la scadenza posticipata di un'ora
/utente	POST	Body: {username : "example", nome : "example", cognome : "example", ruolo: "cameriere", password : "example"}	cassieri	Crea un nuovo utente con le caratteristiche indicati nel body della richiesta. Il JSON nel body della richiesta deve avere le proprietà presenti nell'esempio qui a sinistra

/utente/:id	DELETE	Parametro id: deve corrispondere all'id di un utente registrato nel sistema	cassieri	Rimuove un utente dal sistema
/crea_ordine	POST	Body: { "tavolo": 2 "persone_da_servire": 2 }	camerieri	Crea un nuovo ordine e ne restituisce l'id. Il campo tavolo del JSON passato nel body deve corrispondere a quello di un tavolo esistente e le persone da servire deve essere minore o uguale alla capienza del tavolo
/aggiungi_prodotto/:id_ordine	POST	Parametro: id_ordine deve corrispondere all'id di un ordine esistente Body: { "nome_prodotto": "Margherita", "aggiunte": ["Mozzarella di bufala"] }	camerieri	Aggiunge un prodotto all'ordine passato come parametro. Il campo prodotto deve corrispondere al un prodotto esistente e le relative aggiunte devono essere previste per quel tipo di prodotto (pizza o bibita)
/chiudi_ordine/:id_ordine	POST	Parametro: id_ordine deve corrispondere all'id di un ordine esistente	camerieri	Chiude l'ordine passato come parametro rendendolo così visibile a pizzaioli e baristi.

				Restituisce le bibite del
--	--	--	--	---------------------------

/prossima_ord_barista	GET	nessuno	baristi	prossimo ordine da preparare
/abbandona_ordine_bibite/:id_ordine	POST	Parametro: id_ordine deve corrispondere ad un ordine esistente	baristi	Abbandona la preparazione dell'ordine corrispondente all'id passato e lo assegna ad un altro barista
/bibite_pronte/:id_ordine	POST	Parametro: id_ordine deve corrispondere ad un ordine esistente	baristi	Contrassegna un le bibite dell'ordine corrispondente all'id passato come pronte, tali bibite vengono così segnalate ai camerieri per la consegna
/prossima_ord_pizzaiolo	GET	nessuno	pizzaioli	Restituisce le pizze del prossimo ordine da preparare
/pizza_inizio_preparazione/:id_prodotto	POST	Parametro: id_prodotto deve corrispondere all'id di un prodotto (pizza) ordinato	pizzaioli	Serve a segnalare che la pizza con l'id passato è in fase di preparazione
/pizza_fine_preparazione/:id_prodotto	POST	Parametro: id_prodotto deve corrispondere all'id di un prodotto (pizza) ordinato	pizzaioli	Serve a segnalare che la pizza con l'id passato è pronta
/pizze_pronte/:id_ordine	POST	Parametro: id_ordine deve corrispondere ad un ordine esistente	pizzaioli	Serve a segnalare che tutte le pizze dell'ordine corrispondente all'id passato sono pronte

				Restituisce il prossimo
--	--	--	--	-------------------------

/ordini_da_consegnare	GET	nessuno	camerieri	ordine pronto da consegnare
/bibite_consegnate/ :id_ordine	POST	Parametro: id_ordine deve corrispondere ad un ordine esistente	camerieri	Serve a segnalare che le bibite appartenenti all'ordine passato come id_ordine sono state consegnate
/pizze_consegnate/ :id_ordine	POST	Parametro: id_ordine deve corrispondere ad un ordine esistente	camerieri	Serve a segnalare che le pizze appartenenti all'ordine passato come id_ordine sono state consegnate
/tavoli_occupati	GET	nessuno	cassieri	Restituisce i tavoli che attualmente sono occupati
/informazioni_ordine/ :numero_tavolo	GET	Parametro: numero_tavolo deve corrispondere ad un tavolo esistente	cassieri	Restituisce informazioni riguardo all'ordine (stato preparazione, cameriere che ha preso l'ordine etc.) nel tavolo passato come parametro
/informazioni_prodotti/ :id_ordine	GET	Parametro: id_ordine deve corrispondere ad un ordine esistente	cassieri	Restituisce i prodotti presenti nell'ordine corrispondente all'id passato come parametro

/calcolo_totale/ :numero_tavolo	GET	Parametro: numero_tavolo deve corrispondere ad un tavolo esistente	cassieri	Calcola il totale dell'ordine aperto al numero di tavolo passato come parametro
/pagamento_ordine/ :numero_tavolo	POST	Parametro: numero_tavolo deve corrispondere ad un tavolo esistente	cassieri	Paga il totale dell'ordine aperto al tavolo passato come parametro e libera il tavolo
/incasso_giorno	GET	nessuno	cassieri	Restituisce l'incasso del giorno corrente
/incasso_settimana	GET	nessuno	cassieri	Restituisce l'incasso degli ultimi 7 giorni
/incasso_mese	GET	nessuno	cassieri	Restituisce l'incasso degli ultimi 30 giorni
/statistiche_dipendente/:id	GET	Parametro: id deve essere l'id di un utente registrato nel sistema	cassiere	Restituisce le statiche relative all'utente corrispondente all'id passato come parametro

## 4. Gestione dell'autenticazione degli utenti

Per la gestione l'autenticazione degli utenti è stato usato il framework Passport, che consente attraverso le sue molte API di usare differenti metodi di autenticazione, nel seguito verrà spiegato come è stato usato in dettaglio in questo progetto.

Nel progetto è stata utilizzata la c.d. "bearer authentication", nel dettaglio:

- la stringa username + : + password dopo essere stata codificata in base64 viene ricevuta dal server (fino a qui come in una normalissima basic authentication)
- il server verifica che nel database sia presente l'username ricevuto, nel caso sia presente verifica che la password ricevuta sia compatibile con quella presente nel database, in caso negativo restituisce un errore di autenticazione
- una volta verificate le credenziali crea un oggetto JSON contenente le informazioni relative all'utente memorizzate nel database (es. ruolo dell'utente)
- prima di restituire all'utente il JSON creato precedentemente ad esso viene aggiunta una "data di scadenza" (nel nostro caso impostata ad un'ora da quanto il token viene creato) e viene firmato digitalmente.

A questo punto, l'utente ha "in mano" un token firmato dal server contenente le informazioni dell'utente stesso, che può essere usato dal server stesso per essere certo dell'identità e del ruolo dell'utente senza dover costantemente ad ogni singola richiesta verificare nel database la correttezza dei dati e di conseguenza dei privilegi dell'utente.

Come scritto prima, il token ha una "data di scadenza", essa viene ovviamente impostata per motivi di sicurezza, e comunque ogni qualvolta l'utente si "logga" tramite il token, ne viene rilasciato uno di nuovo con la data di scadenza "spostata" sempre in avanti di 1 ora rispetto al momento della richiesta.

È utile inoltre far notare che nel caso vi fosse il sospetto che la chiave con cui vengono firmati i messaggi fosse stata in qualche modo scoperta, è sempre possibile cambiarla (modificando il file .env), rendendo così invalidi tutti i token in circolazione e "costringendo" tutti gli utenti ad inserire nuovamente username e password per avere accesso al sistema.

**N.B. : nel progetto da noi realizzato a scopo didattico e stata usata una connessione HTTP e non HTTPS, ovviamente affinché il tutto possa veramente considerarsi sicuro è necessario utilizzare una connessione HTTPS che dà certezza dell'identità dell'interlocutore e dell'integrità dei dati, in quanto a titolo esemplificativo ma non esaustivo potrebbe accadere che in un attacco "man in the middle" un server "falso" risponda al posto di quello autentico impossessandosi così dei dati dell'utente.**

## **Gestione delle credenziali degli utenti nel database**

Come spiegato precedentemente utilizzando la tecnica della “bearer authentication” si ottiene un buon compromesso tra sicurezza e carico di lavoro che deve affrontare il server per essere certo dell’identità dell’utente, tuttavia un altro elemento fondamentale da proteggere sono i dati degli utenti salvati nel database del server, pensiamo infatti a cosa potrebbe succedere se un malintenzionato riuscisse ad accedere o a scaricare il database e qui trovasse le password di ogni singolo utente salvate in chiaro: beh saprebbe ovvio che riuscirebbe ad accedere non solo per un’ora come accadrebbe nel caso rubasse il token ma riuscirebbe ad accedere fino a che la password non viene cambiata.

Nel database del server quindi, per evitare i problemi descritti sopra sono state memorizzati per ogni utenti un salt e un digest che semplificando e senza entrare in dettagli crittografici: la password fornita dall’utente viene concatenata al salt e così processata da una funzione crittografica che ritorna una stringa che se risulta essere uguale al digest salvato nel database conferma che la password inserita dall’utente era quella corretta.



## 5. Descrizione dettagliata della web app realizzata con Angular

Il client web come da specifiche di progetto è stato realizzato con il framework Angular.

### Lista dei components:

- **user-login**: è il component “iniziale” al quale tutti gli utenti vengono “indirizzati” inizialmente, esso consente di effettuare il login inserendo username e password
- **cameriere**: è il component a cui viene reindirizzato il cameriere non appena ha effettuato il login, esso consente di procedere con l'acquisizione di un nuovo ordine o in alternativa di procedere alla consegna di un ordine già preparato (questa eventualità è messa in evidenza dal fatto che vi è un bottone giallo lampeggiante a segnalarela)
- **cameriere-prende-ordinazioni**: è il component a cui viene indirizzato il cameriere non appena ha scelto il tavolo per il quale procedere con l'ordinazione, da qui è possibile ordinare 1 o più bibite (o nessuna) ed le eventuali aggiunte (es. ghiaccio, limone)
- **cameriere-prende-ordinazioni2**: è il component a cui viene indirizzato il cameriere non appena ha provveduto a ordinare le bibite, da qui è possibile ordinare 1 o più pizze (o nessuna) ed le eventuali aggiunte (es. mozzarella di bufala)
- **cameriere-consegna-ordinazioni**: è il component a cui il cameriere può arrivare da **cameriere**, da qui è possibile vedere quali sono le prossime bibite/pizze pronte da consegnare nel caso ve ne siano
- **barista**: è il component a cui viene reindirizzato il barista non appena ha effettuato il login, esso consente di visualizzare le bibite del prossimo ordine da preparare
- **cassiere**: è il component a cui viene reindirizzato il cassiere non appena ha effettuato il login, esso consente di avere una panoramica sui tavoli liberi e occupati, e per questi ultimi consente di vedere il dettaglio dell'ordine aperto
- **cassiere-incasso**: è il component a cui il cameriere può arrivare da **cassiere** cliccando il bottone incassi, da qui è possibile vedere gli incassi odierni, relativi agli ultimi 7 e degli ultimi 30.

- **cassiere-gestione-utenti**: è il component a cui il cassiere può arrivare da **cameriere**, da qui è possibile visualizzare le statistiche per ogni singolo utente registrato al sistema, eliminarlo o creane uno nuovo attraverso il bottone “Crea nuovo utente”
- **user-signup**: è il component a cui il cassiere può arrivare da **cassiere-gestione utenti** premendo il bottone “Crea nuovo utente”, da qui è possibile creare un nuovo utente scegliendone il ruolo e assegnandogli una password

### Lista delle routes:

La tabella sottostante mostra, quale component sia associato ad ogni route:

Route	Compent
/	UserLoginComponent
/login	UserLoginComponent
/signup	UserSignupComponent
/cameriere	CameriereComponent
/barista	BaristaComponent
/pizzaiolo	PizzaioloComponent
/cassiere	CassiereComponent
/cameriereOrd	CamerierePrendeOrdinazioniComponent
/cameriereOrd2	CamerierePrendeOrdinazioni2Component
/cameriereCons	CameriereConsegnaOrdinazioniComponent
/incasso	CassiereIncassoComponent
/gestioneutenti	CassiereGestioneUtentiComponent

### I service

Nel progetto sono stati creati 2 service:

- **prodotti.service.ts**: esso contiene tutte le richieste verso il server riguardanti la manipolazione di prodotti e aggregazioni di prodotti (es.ordini)
- **user.service.ts**: esso contiene tutte le richieste verso il server riguardanti la manipolazione di utenti e le loro relative informazioni (es. statistiche)

## 6. Screenshot e workflow dell'applicazione

Schermata di login comune a tutte le tipologie di utente:



Per favore, inserisci le tue credenziali:

Username

Password

☐ Ricordami

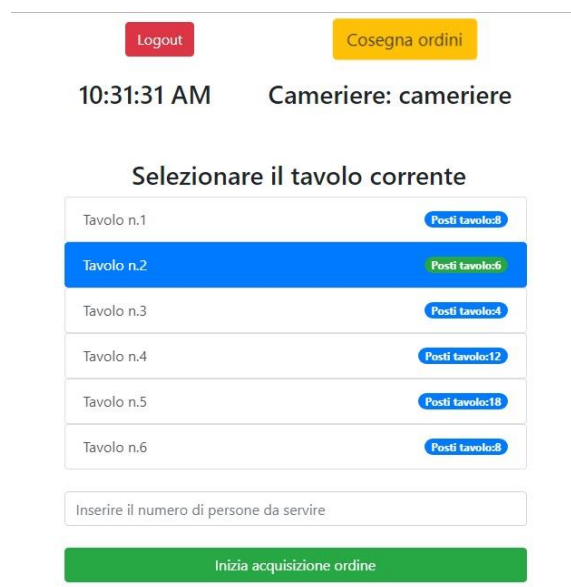
Accedi

© 2019

Nicolas Pietro Martignon & Thomas Visentin

*Figura 1*

Le prossime schermate riguardano invece il workflow tipico di un cameriere:



Logout Cosegna ordini

10:31:31 AM Cameriere: cameriere

Selezionare il tavolo corrente

Tavolo n.1	Posti tavolo:8
<b>Tavolo n.2</b>	<b>Posti tavolo:6</b>
Tavolo n.3	Posti tavolo:4
Tavolo n.4	Posti tavolo:12
Tavolo n.5	Posti tavolo:18
Tavolo n.6	Posti tavolo:8

Inserire il numero di persone da servire

Inizia acquisizione ordine

*Figura 2*

Appena effettuato il login, il cameriere si trova di fronte alla schermata mostrata nella Figura 2, da qui il percorso di “biforca”:

- il cameriere procede con l’acquisizione di un nuovo ordine
- il cameriere procede con la consegna di ordini già pronti

Vediamo ora, il “percorso” della prima ipotesi:

Tavolo N°: 2      Numero persone: 2

Ordinazione bibite

Articoli ordinati:

Aranciata	Quantità: 1
<b>Ghiaccio</b>	

Chiudi ordinazione bibite

Listino Articoli :

Aranciata	2.8€
Birra piccola	3€
<b>CocaCola</b>	2.8€

Listino Aggiunte :

Ghiaccio	0€
<b>Limone</b>	0€

2      Aggiungi

Figura 3

Il cameriere dopo aver selezionato un tavolo, aver inserito il numero di persone da servire e aver premuto il pulsante “Inizia acquisizione ordine” nella figura 2, viene reindirizzato alla schermata nella figura 3, dalla quale aggiungere delle bibite all’ordine premendo il tasto “Aggiungi” e una volta aggiunte tutte le bibite chiudere la sezione “bibite” dell’ordine premendo il pulsante “Chiudi ordinazione bibite”.

Analoga operazione viene fatta successivamente con le pizze come mostrato qui sotto in figura 4.

Tavolo N°: 2
Numero persone: 2

Ordinazione pizze

Articoli ordinati:

Capricciosa
Mozzarella di bufala

Quantità: 1

Chiudi ordinazione

Listino Articoli :

Capricciosa
6€

Diavola
6€

Margherita
4€

Listino Aggiunte :

Mozzarella di bufala
1.5€

1
Aggiungi

Figura 4

Una volta chiusa l'ordinazione attraverso il bottone "Chiudi ordinazione", il cameriere verrà indirizzato nuovamente alla schermata mostrata in figura 2.

Supponiamo ora che il cameriere voglia procedere con la consegna di eventuali ordini già pronti, trovandosi alla schermata mostrata in figura 2 può farlo premendo il tasto "Consegna ordine" (se il tasto è di color giallo lampeggiante, in caso contrario significa che nessun ordine è pronto da consegnare)

Dopo aver premuto il pulsante precedentemente nominato, il cameriere si troverà di fronte alla schermata rappresentata nella figura 5.

Da qui potrà scegliere quali bibite o quali pizze consegnare.

<-
Indietro

10:47:09 AM
Cameriere: cameriere

Bibite da consegnare

Tavolo n.4
Persone: 2

Ora ordinazione: 10:25:00 AM
Bibite consegnate

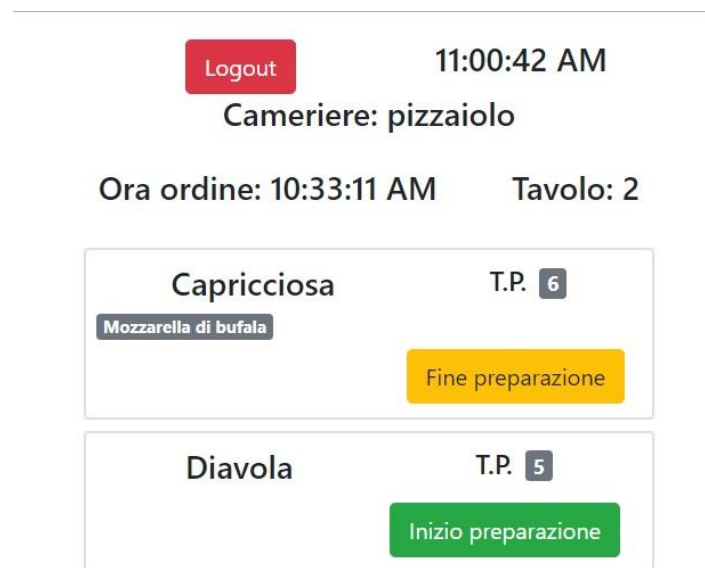
Pizze da consegnare

Tavolo n.4
Persone: 2

Ora ordinazione: 10:25:00 AM
Pizze consegnate

Figura 5

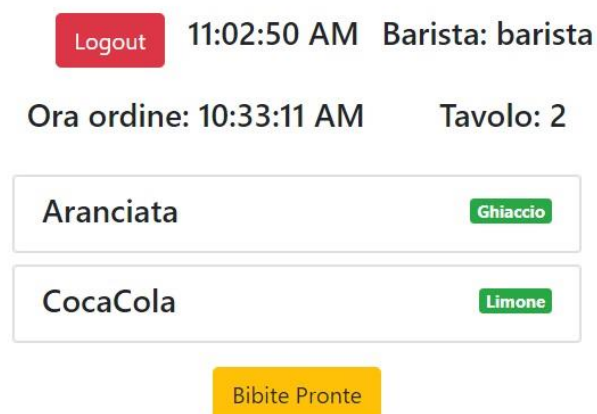
La prossima schermata riguarda invece i pizzaioli:



*Figura 6*

La schermata molto semplice ed intuitiva consente a ciascun pizzaiolo di vedere le pizze da preparare segnalare il fatto che sono in preparazione o che questa è finita.

La prossima schermata riguarda i baristi:



*Figura 7*

La schermata come è possibile vedere consente di segnalare che l'ordine è pronto "in blocco" (ovvero non come avveniva per le pizze in cui si poteva compiere una segnalazione per ogni singola pizza) quando viene premuto il tasto "Bibite Pronte" inoltre vengono visualizzate le bibite dell'ordine successivo.

La seguente serie di schermate riguarda invece il cassiere:



Figura 8

Da figura 8 mostra invece la schermata a cui si trova davanti un cassiere subito dopo aver effettuato il login: come si può notare vi è una dashboard tramite la quale il cassiere può avere sott'occhio la situazione dei tavoli (verdi quelli liberi, gialli quelli occupati), il cassiere può inoltre cliccando un tavolo visualizzare l'ordine "aperto" su quel tavolo e visualizzarne i dettagli, successivamente il cameriere ha 2 possibilità:

- visionare l'incasso giornaliero, settimanale o mensile (cliccando il bottone "Incasso")

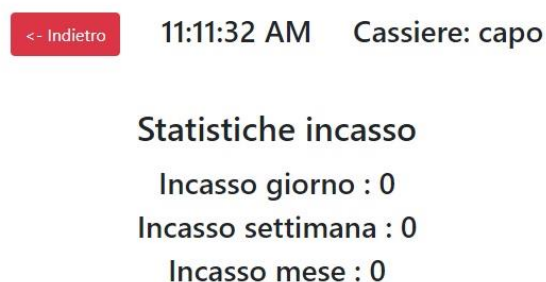


Figura 9

- "gestire" gli utenti visualizzandone le statistiche o creandone di nuovi

<- Indietro

Crea nuovo utente

11:13:11 AM

Cassiere: capo

Camerieri:

Username: cameriere

Nome Cognome: Mario Verdi

Elimina

Pizzaioli:

Username: pizzaiolo

Nome Cognome: Mario Visentin

Elimina

Baristi:

Username: barista

Nome Cognome: Mario martignon

Elimina

Cassieri:

Username: capo

Nome Cognome: Mario Rossi

Elimina

Statistiche

- Persone a cui e stato preparato da bere : 2
- Singoli prodotti preparati : 2
- Tavoli a cui e stato preparato da bere : 1

Figura 10

Com'è possibile vedere dalla figura 10 cliccando sul nome di un utente ne compaiono le statistiche (diverse a seconda del ruolo dell'utente stesso).

Per creare un nuovo utente invece, basta cliccare sul bottone "Crea nuovo utente", e si verrà indirizzati alla pagina mostrata in figura 11.



Registrazione nuovo utente

Username

Enter username

Cognome

Inserisci cognome

I tuoi dati personali non verranno condivisi con nessuno.

Nome

Inserisci nome

I tuoi dati personali non verranno condivisi con nessuno.

Ruolo

Password

Password

Registrati!

Figura 11

Una volta giunti a questa schermata basta inserire i campi richiesti e il nuovo utente è creato.