Relazione del progetto d’esame

Tecnologie e applicazioni web, a.a. 2018/2019

Riccardo Dentico, matricola: 864881

Indice

Architettura del sistema

Il sistema ha un'architettura di tipo Client/Server ossia un
client si connette ad un server per richiedere delle risorse. Il protocollo di comunicazione è HTTP. Di seguito sono riportate le informazioni riguardanti il server e le app dei client.

Back-end

Il server è stato realizzato in javascript, sfruttando il run-time system Node.js (versione 12.7.5) , ed è ospitato presso il servizio di hosting online Heroku. Il database è stato realizzato con il DBMS MongoDB (versione 3.3.2) ed è hostato dal relativo servizio di cloud Atlas.

Per quanto riguarda la struttura del server, sono presenti tre file principali:

- server-ristorante.js, che contiene il codice principale del server
- socket.js, che serve a gestire socket.io (dichiarando il socket, gli eventi e i destinatari della comunicazione)
- validation.js, che si occupa della validazione dei dati ricevuti in input tramite joi

Di seguito sono riportati i moduli esterni usati nel server (con relativa versione):

- express (4.17.1), fa da middleware e serve a gestire la pipeline di richieste e risposte e il routing fra gli endpoint e le API
- cors (2.8.5), si interfaccia con express per permettere le Cross-origin resource sharing (un meccanismo che non permette su una pagina web di richiedere risorse provenienti da un altro dominio)
- joi (14.3.1), per verificare se i tipi di dato inseriti nei campi di input siano corretti
- mongoose (5.7.0), per modellare i dati del database MongoDB ed è adatto a lavorare in un ambiente asincrono
- body-parser (1.19.0), per effettuare il parsing sulle query (necessario al funzionamento di Express)
- express-jwt (5.3.1), che serve ad autenticare le richieste HTTP usando jsonwebtoken
- passport (0.4.0), per creare i meccanismi di autenticazione
- passport-http (0.3.0), per creare gli schemi di autenticazione di passport
- jsonwebtoken (8.5.1), per scambiare i dati dell'autenticazione tramite un web token che usa il formato json

- socket.io (2.1.2), per inviare le notifiche agli utenti (permette l'invio in modo asincrono di eventi da client a server e viceversa)

Front-end

I client sono stati realizzati con Angular (versione 8.1.3) insieme al framework Ionic (versione 4.7.1), in modo da creare una progressive web app. Queste funzionano a prescindere dal browser scelto e si adattano alle varie dimensioni dello schermo (responsive), simulando anche un comportamento da app native. Il porting all'app desktop è stato effettuato con Electron, quello della versione mobile con Cordova.

E' presente una cartella per ogni categoria di utenti (bartender, cashier, cook, waiter), una per le collezioni di dati e una per i servizi.

Modello dei dati

Nel database sono presenti 5 collezioni.
Di seguito è riportata la lista delle collezioni con i relativi campi e il tipo dei dati presenti in essi:

Items contiene i documenti in cui sono salvate le informazioni
dei prodotti (piatti e bavande) presenti, fra cui un codice numerico univoco incrementale da 100 (nel caso in cui sia una bevanda) e da 200 (nel caso in cui si tratti di un piatto).

Items
- code: number
- name: string
- requiredTime: number
- price: number

Orders contiene i documenti in cui sono salvate le informazioni
degli ordini presenti, fra cui due array di numeri per l'elenco dei piatti e delle bevande, un array di numeri a rappresentare lo stato di preparazione dei piatti (0, se il piatto non è ancora in fase di preparazione; 1, se il piatto è in preparazione; 2, se il

piatto è stato preparato), un booleano che rappresenta lo stato di preparazione delle bevande, la data in giorno, mese, anno rappresentata da un numero intero.

Orders

- orderNumber: number
- beverageList: [number]
- dishList: [number]
- dishState: [number]
- beverageState: boolean
- numberPeople: number
- tableNumber: number
- date: number
- orderStatus: number
- userNameWaiter: string

Stats contiene i documenti in cui sono salvate le informazioni relative ai cuochi e ai camerieri, fra cui un numero di clienti e di ordini serviti per i camerieri e numero e prezzo dei piatti preparati per i cuochi.

Stats

- username: string
- numberOfClients: number
- numberOfPlates: number
- priceOfPlates: number
- numberOfOrders: number

Tables contiene i documenti in cui sono salvate le informazioni dei tavoli presenti, fra cui numero di posti del tavolo, un numero che rappresenta l'id dell'ordine attualmente associato al tavolo e un booleano per stabilire se il tavolo è occupato.

Tables

- tableNumber: number
- seats: number
- orderId: number
- occupied: boolean

Users contiene i documenti in cui sono salvate le informazioni
degli utenti presenti, fra cui un numero per il ruolo (1, per il cassiere; 2, per il cameriere; 3, per il cuoco; 4, per il barista), un numero (campo salt) che rappresenta una sequenza casuale di bit, che usata insieme alla password, viene data in input alla funzione hash, un numero (digest) risultato di una funzione di hashing per criptare la password che viene confrontata con quella ricevuta dal client per completare l'autenticazione.

Users

- username: string
- password: string
- role: number
- salt: number
- digest: number

API

Di seguito è riportata la lista delle API:

endpoint: /users/login
metodo: get
descrizione: restituisce il token per l'autenticazione

endpoint: /users
metodo: get
descrizione: restituisce la lista di tutti gli utenti

endpoint: /renew
metodo: get
descrizione: restituisci un nuovo token

endpoint: /users
metodo: post
formato richiesta: {username: string, password: string, role: number }
descrizione: crea un nuovo utente

endpoint: /tables
metodo: post

formato richiesta: {tableNumber: number, seats: number}
descrizione: crea un nuovo tavolo

endpoint: /tables
metodo: get
descrizione: restituisce la lista di tutti i tavoli

endpoint: /tables/:id
metodo: get
descrizione: restituisce un determinato tavolo (usato per la ricerca del tavolo in base all'id)

endpoint: /tables/:id
metodo: patch
descrizione: modifica lo stato (il campo occupied; 0, se il tavolo è libero; 1, se il tavolo è occupato) di un determinato tavolo

endpoint: /items
metodo: get
descrizione: restituisce la lista di tutti i prodotti

endpoint: /items
metodo: post
formato richiesta: {code: number, name: string, requiredTime: number, price: number}
descrizione: crea un nuovo prodotto

endpoint: /items/:code
metodo: delete
descrizione: cancella un determinato prodotto

endpoint: /items/:code
metodo: get
descrizione: restituisce un determinato prodotto (usato per la ricerca di un prodotto in base al codice)

endpoint: /orders
metodo: get
descrizione: restituisce la lista di tutti gli ordini

endpoint: /orders

metodo: post
formato richiesta: {orderNumber: number, beverageList: [number], dishList: [number], dishState: [number], numberPeople: number, tableNumber: number, date: number, userNameWaiter: string}
descrizione: crea un nuovo ordine

endpoint: /orders/status/:id
metodo: patch
descrizione: modifica lo stato (0, se l'ordine è stato creato ma non è in preaparazione; 1, se l'ordine è stato preparato ma non è ancora stato pagato; 2, se l'ordine è già stato pagato)

endpoint: /orders/:id
metodo: get
descrizione: restituisce un singolo ordine (usato per la ricerca di un ordine in base all'id)

endpoint: /orders/dishes/:id
metodo: patch
formato richiesta: {index: number}
descrizione: modifica lo stato di preparazione di un determinato piatto ordinato in un determinato ordine

endpoint: /orders/beverages/:id
metodo: patch
descrizione: modifica lo stato di preparazione di una determinata bevanda ordinata in un determinato ordine

endpoint: /orders/tickets/day/:date/:type
metodo: get
descrizione: restituisce la lista degli ordini di una determinata giornata un determinato in un determinato stato (0, se l'ordine è stato creato ma non è in preaparazione; 1, se l'ordine è stato preparato ma non è ancora stato pagato; 2, se l'ordine è già stato pagato)

endpoint: /orders/tickets/:id
metodo: get
descrizione: restituisce un determinato ordine (usato per la ricerca di un ordine in base all'id)

endpoint: /stats
metodo: get
descrizione: restituisce la lista delle statistiche di tutti gli utenti

Autenticazione

Per il meccanismo di autenticazione degli utenti è stato usato passport,un modulo di Node.js, il quale fornisce varie strategie di autenticazione. Fra queste, basic authentication che codifica i dati inviati (username:password) in base 64. Successivamente viene usato il meccanismo della bearer authentication.
Il server crea un token, per il quale è stato usato il modulo JsonWebToken, che viene firmato dal server stesso e viene inviato al client. Una volta ricevuto esso può generare la stringa da spedire al server come risposta. Il server procederà quindi a verificare se la stringa ricevuta coincide con quella nel server allora l'autenticazione dell'utente sarà stata completata correttamente.

Client web

La web app è stata realizzata con Angular e Ionic. Dato l'uso di quest'ultimo framework i component sono anche rinominati come page, di seguito è riportata la lista:

Bartender

- bartender: visualizza la lista degli ordini assegnati a un barista, organizzati in una coda in base alla data con cui essi sono arrivati
- bartender details: visualizza la lista delle bevande di un ordine e il pulsante per notificare al cameriere assegnato a quell'ordine che questo è pronto

Cashier

- cashier menù: mostra il menù con all'interno le funzionalità a disposizione del cassiere

- table status: mostra la visualizzazione dei tavoli con relative informazioni. Il colore indica se sono liberi (verde) o occupati (rosso) e sono riportati anche il numero di posti a sedere. Cliccando sui tavoli si avrà una pagina (table details) con altre informazioni
- table details: visualizza la lista dei piatti e delle bevande presenti nell'ordine associate a quel tavolo e il cameriere associato
- kitchen queue: mostra la coda di preparazione della cucina, ossia gli ordini che saranno preparati in base all'ordine di arrivo
- kitchen queue details: visualizza la lista di piatti e bevande, attualmente in preparazione o in attesa di essere preparati, relativa a un ordine. In base al colore questi sono in attesa di preparazione (rosso), in preparazione (giallo) o preparati (verde). Piatti e bevande sono ordinati secondo la coda di cuochi e baristi. I cuochi cominceranno a preparare prima i piatti che richiedono più tempo
- tickets: mostra il totale incassato nella giornata e gli scontrini in attesa di essere pagati
- tickets details: visualizza la lista di piatti e bevande con il relativo prezzo e il totale (a cui è stato aggiunto il costo del coperto per le persone che erano al tavolo) di un ordine e il pulsante per chiudere il conto
- users: visualizza il form per la creazione di un nuovo utente (in cui inserire username, password e ruolo) e il pulsante per l'invio
- tables: visualizza il form per la creazione di un nuovo tavolo (in cui inserire numero del tavolo e posti a sedere) e il pulsante per l'invio
- items: visualizza il form di creazione di un nuovo prodotto (in cui inserire codice, nome del prodotto, tempo di preparazione e prezzo) e il pulsante per l'invio
- stats: visualizza le statistiche relative a cuochi (numero di piatti e prezzo totale dei piatti preparati) e camerieri (numero di clienti e di ordini serviti)

Cook

- cook: visualizza la lista degli ordini assegnati a un cuoco, organizzati in una coda in base alla data con cui essi sono arrivati
- cook details: visualizza la lista dei piatti di un ordine ordinati, in base al tempo di preparazione (prima i piatti che richiedono un maggior tempo di preaprazione) e il pulsante per notificare al cameriere assegnato a quell'ordine che questo è pronto. Ogni piatto è cliccabile in modo da indicare lo stato di preparazione (cliccando una volta diventerà giallo e due volte diventerà verde)

Waiter

- waiter: mostra lo stato dei tavoli in base al colore, sono liberi (verde) o occupati (rosso), e se sono liberi sarà possibile effettuare un ordine cliccandoli
- waiter order: visualizza il form per effettuare un nuovo ordine (contenente numero di persone, lista delle bevande e lista dei piatti)e il pulsante per inviarlo

Di seguito è riportata la lista dei servizi con le relative funzioni:

- item-http: fornisce i metodi relativi alla gestione dei prodotti
- order-http: fornisce i metodi relativi alla gestione degli ordini
- stat-http: fornisce i metodi relativi alla gestione delle statistiche (in questo caso solo getStats)
- table-http: fornisce i metodi relativi alla gestione dei tavoli
- user-http: fornisce i metodi relativi alla gestione degli utenti
- http-interceptor: incapsula la richiesta di un altro servizio, aggiungendo il necessario affinchè venga ricevuta correttamente dal serve
- socketio: fornisce i metodi realtivi alla gestione del socket (in questo caso solo una get che invia il socket al client)

Di seguito è riportata la lista delle route:

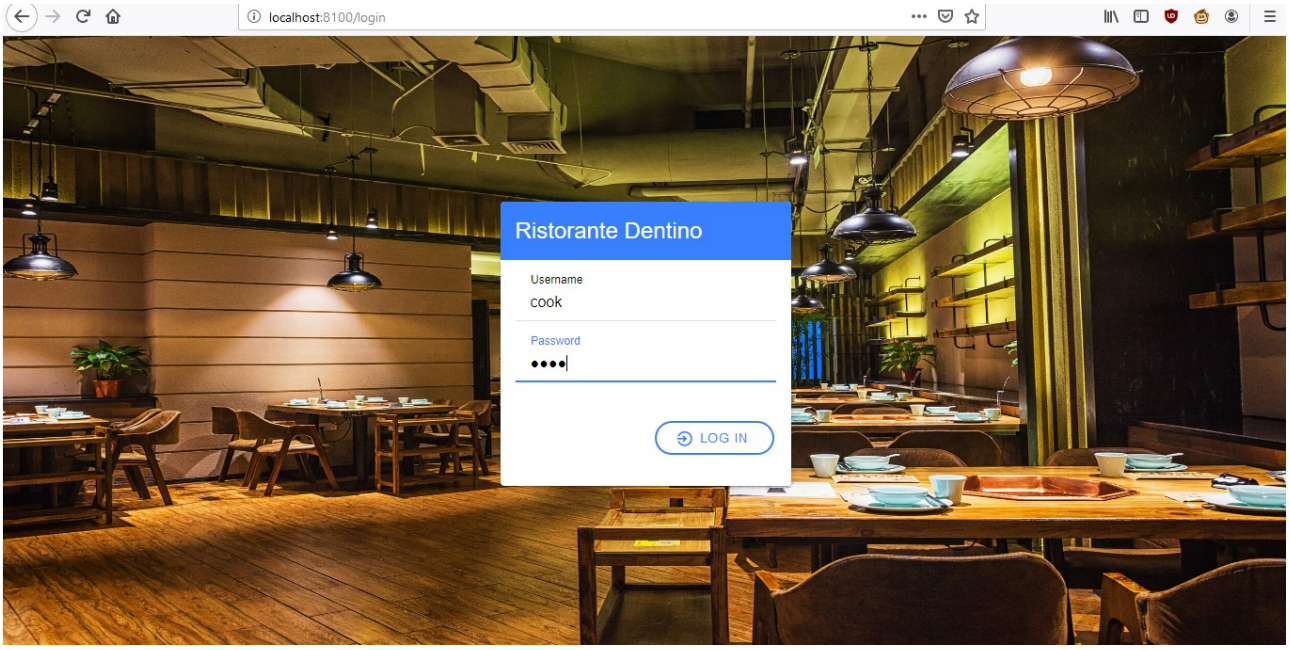
- /
- /login
- /bartender
- /bartender/:orderId
- /cashier, reindirizza a /cashier/menu/tablestatus
- /cashier/menu/tablestatus
- /cashier/menu/kitchenque
- /cashier/menu/ticket
- /cashier/menu/gestusers
- /cashier/menu/gesttable
- /cashier/menu/gestitem
- /cashier/menu/stats
- /cook
- /cook/:orderId

- /waiter
- /waiter/:tableId

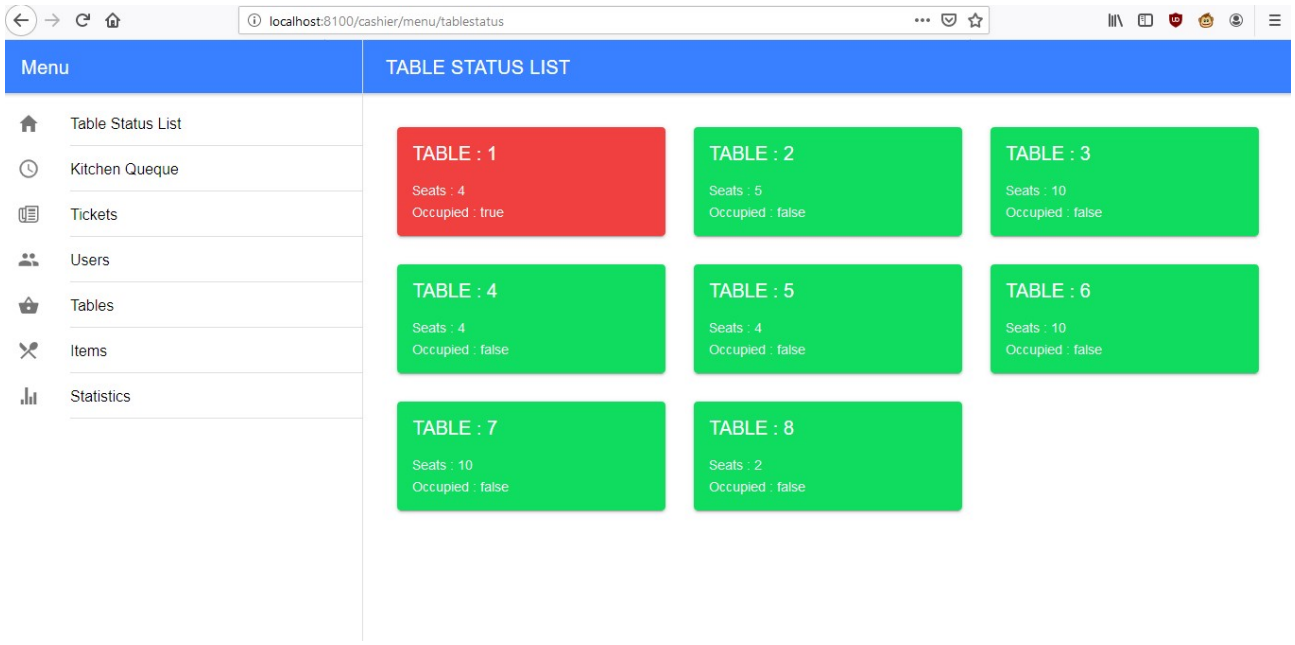
Esempi d’uso

Di seguito sono riportati alcuni screenshot relativi ai casi d’uso:

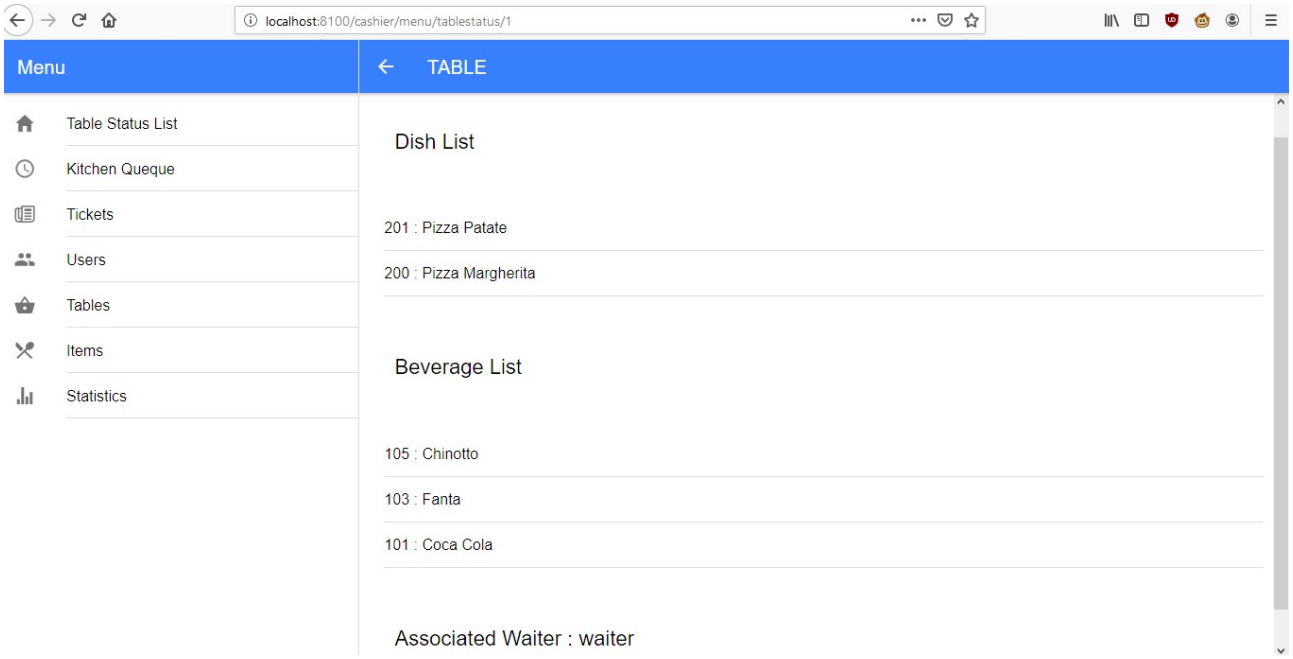
. pagina di login, si presenta a tutti gli utenti che accedono all’app



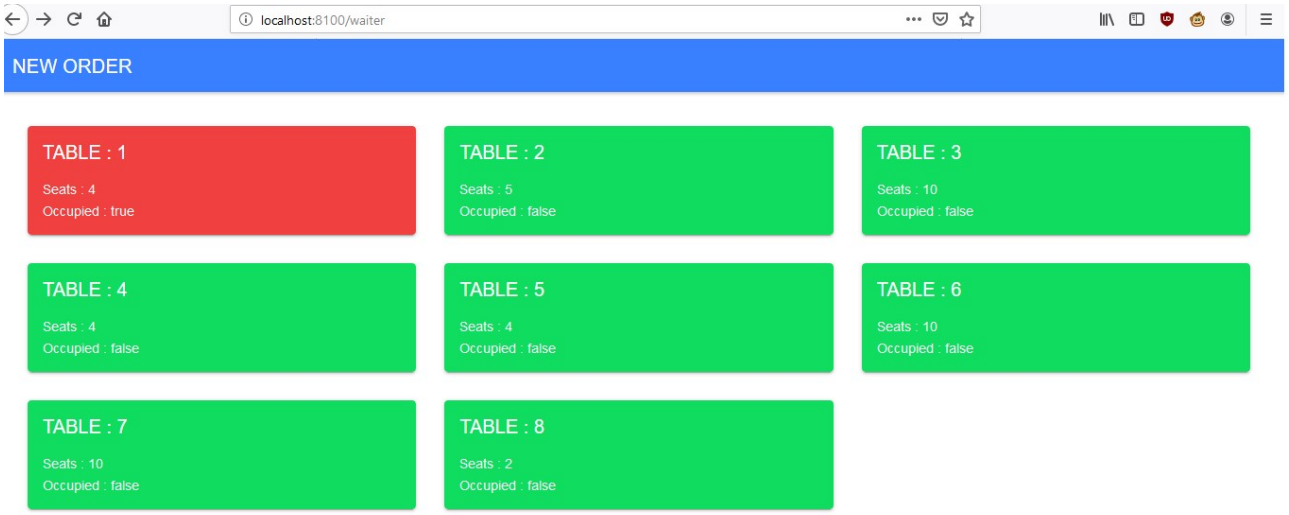
. un cassiere appena effettuato l’accesso verrà indirizzato alla pagina dello stato dei tavoli, da qualsiasi pagina può selezionare le voci del menù



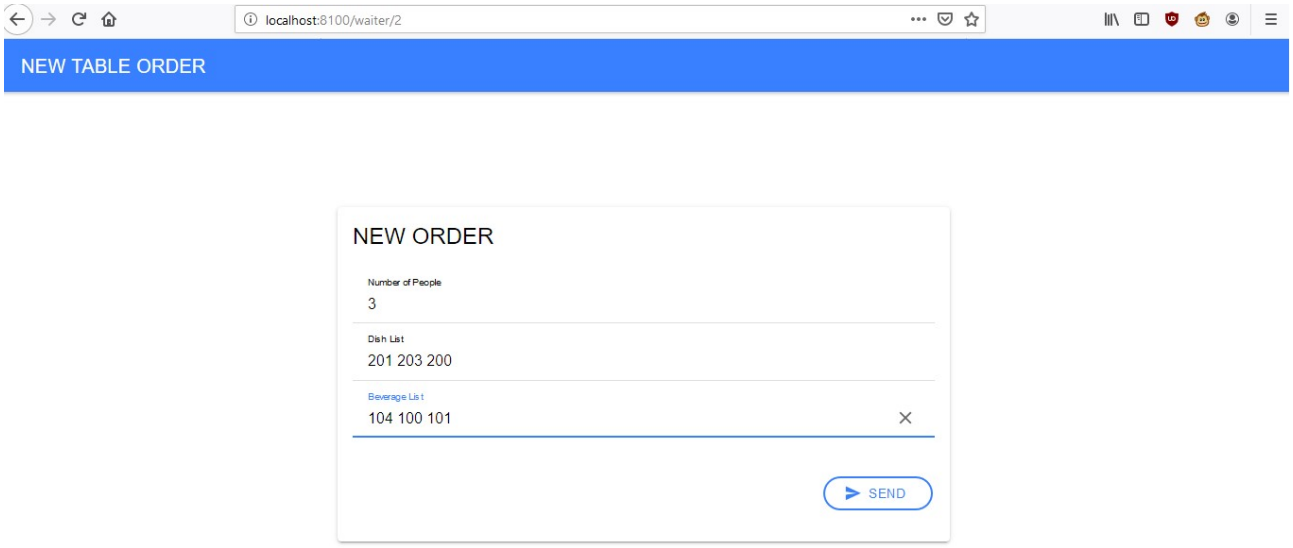
. dalla schermata precedente, selezionando un tavolo occupato il cassiere potrà vedere il dettaglio dell'ordine



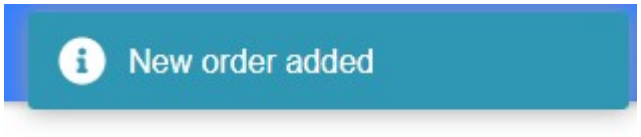
. un utente loggato come cameriere visualizzerà la pagina con i tavoli e il relativo stato, e potrà cliccare solo su quelli liberi per effettuare un nuovo ordine in quel tavolo



. dalla schermata precedente selezionando su un tavolo libero visualizzerà il form per la creazione di un nuovo ordine



. dopo che un cameriere avrà effettuato l'ordine il cuoco e il barista che riceveranno in consegna l'ordine riceveranno una notifica


New order added