



# Relazione di progetto

Dipartimento di Scienze Ambientali, Informatica e Statistica

Tecnologie e Applicazioni Web - A.A. 2018-2019 CT0142

Docente: Filippo Bergamasco

Paolo Porcellato, 870945

Gruppo: WATTENE

Paolo Porcellato 870945

Matteo Schizzerotto 867704

Goran Gajic 870592

<b>Sommario</b>	<b>4</b>
<b>Introduzione</b>	<b>4</b>
<b>Architettura del sistema</b>	<b>4</b>
<b>Back-end</b>	<b>5</b>
La directory _helpers	6
Autenticazione degli utenti	6
Refresh del token	7
La directory controllers	7
APIs	8
Drinks endpoints	8
Foods endpoints	10
Orders endpoints	12
Tables endpoints	17
Users endpoints	19
La directory models	23
Drink model	23
Food model	24
Order model	26
Suborder model	28
orderedFoodSchema	29
orderedDrinkSchema	31
Table model	33
User model	34
La directory services	35
<b>Front-End</b>	<b>36</b>
Components	36
App	36
Login	36
Navbar	36
Dashboard	36
Order	36
Back office	37
Stats	37
Services	37
Guard	37
Services per le api	38
Routes	38

<b>Workflow</b>	<b>39</b>
Autenticazione	39
Backoffice	40
Gestione utenti	40
Gestione cibi	41
Visualizzazione statistiche	41
Gestione tavoli	42
Gestione coda di preparazione	44

# Sommario

Il progetto realizzato ha l'obiettivo di gestire le operazioni riguardanti le ordinazioni che possono avvenire in un locale appartenente al settore ristorativo in modo digitalizzato. Per adempiere a tale scopo è stato realizzato un applicativo utilizzabile da dispositivi mobile e desktop per la presa delle comande e fornisce, al gestore, una visione complessiva della situazione dei tavoli ed un'interfaccia dedicata per inserire nuovi cibi e bibite con il relativo prezzo e dati statistici

## Introduzione

Il prodotto finale è un'applicazione web che permette un'interazione dinamica da parte dell'utente. Si tratta di una web app SPA realizzata grazie ad una comunicazione tramite il protocollo Websockets implementata nel sistema con la libreria [Socket.IO](#).

Gli utilizzatori finali possono utilizzare dispositivi mobili Android e iOS o desktop.

## Architettura del sistema

L'architettura del sistema è composta da un front-end ed un back-end.

Il front-end rappresenta l'interfaccia, è stato realizzato tramite il framework [Angular](#) sviluppato da Google.

Mentre il back-end permette il funzionamento delle interazioni dell'utente nella pagina, esso fornisce i dati, li inserisce e li elabora interrogando il database, il tutto tramite delle request HTTP inviate in modo asincrono a cui segue una response.

Il back-end è stato realizzato tramite:

- [NodeJS](#) runtime environment che consente di scrivere applicazioni in JavaScript lato server;
- [Express](#) o Express.js è un framework per la realizzazione di web app ed API gestendo il routing delle request in arrivo al server.
- [MongoDB](#) è un database NoSql orientato ai documenti con lo scopo di mantenere i dati sotto forma di JSON e di fornirli quando richiesti.

La soluzione descritta appartiene allo stack MEAN (**M**ongoDB, **E**xpress, **A**ngular, **N**ode.js).

Per definire e validare i modelli è stata realizzata l'ODM [mongoose](#).

Per lo sviluppo dell'applicativo mobile è stato utilizzato il framework [Apache Cordova](#) mentre per l'ambiente desktop il framework [Electron](#), entrambi consentono lo sviluppo di GUI utilizzando tecnologie web.

# Back-end

Struttura della directory backend:

## **backend**

```
|-- src
|  |-- _helpers
|  |  |-- authorize.js
|  |  |-- db-init.js
|  |  |-- db.js
|  |  |-- enum.ts
|  |  |-- error-handler.js
|  |  |-- jwt.js
|  |  `-- utility.js
|  |-- controllers
|  |  |-- drink.controller.ts
|  |  |-- food.controller.ts
|  |  |-- order.controller.ts
|  |  |-- table.controller.ts
|  |  `-- user.controller.ts
|  |-- models
|  |  |-- drink.model.ts
|  |  |-- food.model.ts
|  |  |-- order.model.ts
|  |  |-- suborder.model.ts
|  |  |-- table.model.ts
|  |  `-- user.model.ts
|  |-- services
|  |  |-- drink.service.ts
|  |  |-- food.service.ts
|  |  |-- order.service.ts
|  |  |-- suborder.service.ts
|  |  |-- table.service.ts
|  |  `-- user.service.ts
|  |-- config.json
|  `-- index.ts
|-- .env
|-- package.json
`-- cleaner.js
```

## La directory \_helpers

Tale directory si occupa, principalmente, di fornire metodi di utilità come:

- connessione al database ed inizializzazione con dei dati;
- controllo del token di ogni request in arrivo e del ruolo;
- definizione degli enum per il ruolo degli utenti, categorie e stati di preparazione dei cibi e delle bibite.

## Autenticazione degli utenti

L'autenticazione al sistema avviene mediante lo standard **JSON Web Token (JWT)** RFC 7519.

L'utente deve inviare una request all'apposito endpoint includendo nel body il suo username e password:

```
{ url: "/users/login", methods: ["POST"] }
```

Se essi risultano corretti viene generato un token per l'utente che gli verrà restituito nella response, esso viene salvato localmente, nel nostro caso nel local storage ma potrebbe essere salvato anche in un cookie.

Nel payload del token vengono salvati alcuni statement dell'utente come il role (cassa, cameriere, cuoco, password) e il suo id nel campo "sub" (subject).

Il campo "exp" indica il periodo di validità del token, attualmente la validità dei token generata dal back-end corrisponde ad 1 giorno, mentre "iat" indica la data e l'ora in cui questo token è stato generato.

Token generato dal server, dopo un login con successo da parte della cassa:	
Header	<pre>{   "alg": "HS256",   "typ": "JWT" }</pre>
Payload	<pre>{   "sub": "5e4d4099334d442bd4291b65",   "role": "cassa",   "iat": 1582121479,   "exp": 1582207879 }</pre>
Signature	<pre>HMAC-SHA256(   base64urlEncoding(header) + '.' +   base64urlEncoding(payload),   secret )</pre>

Nell'applicativo sviluppato il token ha una scadenza, questa scelta è dovuta a motivi di sicurezza.

Se il token venisse rubato da un utente malevolo il suo utilizzo per i fini di autenticazione verrebbe limitato fino allo scadere.

Poiché il traffico nelle request http è trasmesso in chiaro è necessaria una connessione sicura (https) per effettuare la comunicazione del token tra client e server.

## Refresh del token

In aggiunta al token JWT è presente un sistema per rinnovare il token dopo sua scadenza. La durata standard del token usato per l'autenticazione è di 1 giorno, allo scadere esso può essere rinnovato entro 5 giorni.

Per implementare questo meccanismo, dopo aver effettuato il login con successo viene salvato nel document relativo all'utente un refresh token (un token firmato dal server che scade dopo 5 giorni). Quest'ultimo viene restituito anche dalla response insieme al token di accesso e salvato localmente. Se l'utente, utilizzando il servizio, riceve come risposta un errore da uno degli endpoint (err. 401 unauthorized) viene effettuata una request:

```
{ url: "/users/refresh", methods: ["POST"] }
```

Nel body di questa request viene inserito il refresh token e se il rinnovo ha successo si riceve una response con un nuovo token ed un nuovo refresh token.

## La directory controllers

Contiene le routes per le API. Alcune routes sono accessibili solo se l'utente è autenticato e autorizzato ad effettuare particolari operazioni; questo controllo avviene decodificando il token dell'utente presente nell'header della request e verificandone il ruolo, ciò avviene mediante i controlli definiti nella directory [\\_helpers](#)

Le varie route sono rispettivamente:

- [/drinks](#)
- [/foods](#)
- [/orders](#)
- [/tables](#)
- [/users](#)

L'entry point del back-end è il file **index.ts**, presente in radice della cartella src. In tale file vengono inizializzate le routes definite nei controllers.

```
app.use('/drinks', new DrinkController().router);
app.use('/foods', new FoodController().router);
app.use('/tables', new TableController().router);
app.use('/orders', new OrderController().router);
app.use("/users", new UserController().router);
```

## APIs

Per una visione complessiva di tutte le routes definite è necessario lanciare una richiesta GET all'url base del server (in fase di sviluppo <http://localhost:4000/>).

Nelle seguenti tabelle sono descritti tutti gli endpoint.

**N.B.:** ove indicato Formato dei dati della risposta DrinkModel si intende che la response è quella definita nel file backend/src/models/drink.model.ts

### Drinks endpoints

Endpoint	/drinks
Metodo	GET
Parametri di query	
Formato dei dati della richiesta	
Formato dei dati della risposta	<a href="#">DrinkModel[]</a>
Descrizione	Restituisce un array con la lista di tutti i drink

Endpoint	/drinks/:id
Metodo	GET
Parametri di query	
Formato dei dati della richiesta	
Formato dei dati della risposta	{ "_id": ObjectId, "name": string, "price": number, "time": number, "category": string, "available": boolean }
Descrizione	Restituisce il drink, se presente, che corrisponde all'id :id

Endpoint	/drinks/categories
Metodo	GET
Parametri di query	



Formato dei dati della richiesta	
Formato dei dati della risposta	{ "Analcolico", "Alcolico" }
Descrizione	Restituisce le categorie di drink. Se nessun drink appartiene ad una categoria, essa non viene restituita. Esempio: se nessuna bibita alcolica presente allora la response restituisce solo "Analcolico"

Endpoint	/drinks
Metodo	POST
Parametri di query	
Formato dei dati della richiesta	<a href="#">DrinkModel</a> N.B. non specificare_id
Formato dei dati della risposta	<a href="#">DrinkModel</a>
Descrizione	Crea una nuova bibita e ritorna un JSON che la descrive

Endpoint	/drinks/:id
Metodo	DELETE
Parametri	
Formato dei dati della richiesta	
Formato dei dati della risposta	{ }
Descrizione	Elimina il drink con id uguale a quello specificato nel parametro dell'endpoint

Endpoint	/drinks/:id
Metodo	PUT
Parametri	
Formato dei dati della richiesta	<a href="#">DrinkModel</a> N.B. non specificare_id

Formato dei dati della risposta	<a href="#">DrinkModel</a>
Descrizione	Aggiorna i dati del drink con id uguale a quello specificato nel parametro dell'endpoint con i dati specificati nel body della request

---

## Foods endpoints

Endpoint	/foods
Metodo	GET
Parametri di query	
Formato dei dati della richiesta	
Formato dei dati della risposta	<a href="#">FoodModel[]</a>
Descrizione	Restituisce un array con la lista di tutti i cibi

Endpoint	/foods/:id
Metodo	GET
Parametri di query	
Formato dei dati della richiesta	
Formato dei dati della risposta	<a href="#">FoodModel</a>
Descrizione	Restituisce il cibo, se presente, che corrisponde all'id :id

Endpoint	/foods/categories
Metodo	GET
Parametri di query	
Body della request	
Formato dei dati della risposta	<pre>{   "Dessert",   "Antipasto",   ... }</pre>

Descrizione	Restituisce le categorie di cibi presenti. Se nessun cibo appartiene ad una categoria definita nel file enum, essa non viene restituita. Esempio: se non è presente nessun Primo Piatto allora non verrà restituita tale categoria nella response
-------------	--

Endpoint	/foods
Metodo	POST
Parametri di query	
Body della request	<a href="#">FoodModel</a> N.B. non specificare_id
Dati della response	<a href="#">FoodModel</a>
Descrizione	Crea un nuovo cibo e ritorna un JSON che lo descrive

Endpoint	/foods/:id
Metodo	DELETE
Parametri	
Formato dei dati della richiesta	
Formato dei dati della risposta	{ }
Descrizione	Elimina il cibo con id uguale a quello specificato nel parametro dell'endpoint

Endpoint	/foods/:id
Metodo	PUT
Parametri	
Formato dei dati della richiesta	<a href="#">FoodModel</a> N.B. non specificare_id
Formato dei dati della risposta	<a href="#">FoodModel</a>
Descrizione	Aggiorna i dati del cibo con id uguale a quello specificato nel parametro dell'endpoint con i dati specificati nel body della request

---

## Orders endpoints

Endpoint	/orders
Metodo	GET
Parametri	
Formato dei dati della richiesta	
Formato dei dati della risposta	<a href="#">OrderModel[]</a>
Descrizione	Restituisce un array con la lista di tutti gli ordini

Endpoint	/orders/:id
Metodo	GET
Parametri	
Formato dei dati della richiesta	
Formato dei dati della risposta	<a href="#">OrderModel</a>
Descrizione	Restituisce l'ordine con id uguale a quello specificato nel parametro dell'endpoint

Endpoint	/orders/suborders
Metodo	GET
Parametri	?type=food ?ready=<boolean> - default=false
Formato dei dati della richiesta	
Formato dei dati della risposta	<a href="#">SuborderModel[]</a>
Descrizione	Restituisce un array con la lista di tutti i suborder che hanno del cibo ordinato e non risulta pronto

Endpoint	/orders/suborders
Metodo	GET
Parametri	type=drink
Formato dei dati della richiesta	
Formato dei dati della risposta	<a href="#">SuborderModel[]</a>
Descrizione	Restituisce un array con la lista di tutti i suborder che hanno delle bibite ordinate e non risultano pronte

Endpoint	/orders/:order_id/suborders
Metodo	GET
Parametri	
Formato dei dati della richiesta	
Formato dei dati della risposta	<a href="#">SuborderModel[]</a>
Descrizione	Restituisce i suborders legati all'ordine con id uguale a quello specificato nel parametro dell'endpoint

Endpoint	/orders
Metodo	POST
Parametri	
Formato dei dati della richiesta	<a href="#">OrderModel</a> N.B. non specificare_id
Formato dei dati della risposta	<a href="#">OrderModel</a>
Descrizione	Crea un nuovo ordine e ritorna un JSON che lo descrive

Endpoint	/orders/:order_id/suborders
Metodo	POST
Parametri	
Formato dei dati della richiesta	<a href="#">SuborderModel</a>
Formato dei dati della risposta	<a href="#">SuborderModel</a>
Descrizione	Crea un nuovo suborder e ritorna un JSON che lo describe

Endpoint	/orders/:id
Metodo	PUT
Parametri	
Formato dei dati della richiesta	<a href="#">OrderModel</a> N.B. non specificare_id
Formato dei dati della risposta	<a href="#">OrderModel</a>
Descrizione	Aggiorna un ordine con id uguale a quello specificato nel parametro dell'endpoint con i dati del body della request

Endpoint	/orders/:order_id/suborders/:suborder_id/drinks/:drink_id
Metodo	PUT
Parametri	
Formato dei dati della richiesta	{ "ready": boolean }
Formato dei dati della risposta	<a href="#">SuborderModel</a>
Descrizione	Indica come servita la bibita con id uguale a quello specificato nel parametro dell'endpoint e presente all'interno del suborder con id uguale a quello specificato nel parametro dell'endpoint

Endpoint	/orders/:order_id/suborders/:suborder_id/foods/:food_id
Metodo	PUT
Parametri	
Formato dei dati della richiesta	{ "ready": boolean, }
Formato dei dati della risposta	<a href="#">SuborderModel</a>
Descrizione	Indica come servita un cibo con id uguale a quello specificato nel parametro dell'endpoint e presente all'interno del suborder con id uguale a quello specificato nel parametro dell'endpoint

Endpoint	/orders/:order_id/suborders/:suborder_id
Metodo	PUT
Parametri	
Formato dei dati della richiesta	{ "foods_served": boolean, "drinks_served": boolean }
Formato dei dati della risposta	<a href="#">SuborderModel</a>
Descrizione	Aggiorna lo stato di cibi e/o bevande di un suborder corrispondente con id uguale a quello specificato nel parametro dell'endpoint ed appartenente all'ordine order_id. Imposta rispettivamente i cibi come "Servito" e le bibite come "Servito"

Endpoint	/orders/:id
Metodo	DELETE
Parametri	
Formato dei dati della richiesta	
Formato dei dati della risposta	{}
Descrizione	Elimina l'ordine con id uguale a quello specificato nel parametro dell'endpoint

### Tables endpoints

Endpoint	/tables
Metodo	GET
Parametri	
Formato dei dati della richiesta	
Formato dei dati della risposta	<a href="#">TableModel[]</a>
Descrizione	Restituisce un array con la lista di tutti i tavoli

Endpoint	/tables/:id
Metodo	GET
Parametri	
Formato dei dati della richiesta	
Formato dei dati della risposta	<a href="#">TableModel</a>
Descrizione	Restituisce il tavolo, se presente, con id uguale a quello specificato nel parametro dell'endpoint



Endpoint	/tables
Metodo	POST
Parametri	
Formato dei dati della richiesta	<a href="#">TableModel</a> N.B. non specificare_id
Formato dei dati della risposta	<a href="#">TableModel</a>
Descrizione	Crea un nuovo tavolo con i parametri specificati nel body

Endpoint	/tables/:id
Metodo	PUT
Parametri	
Formato dei dati della richiesta	<a href="#">TableModel</a> N.B. non specificare_id
Formato dei dati della risposta	<a href="#">TableModel</a> N.B. non specificare_id
Descrizione	Aggiorna i dati del tavolo con id uguale a quello specificato nel parametro dell'endpoint con i dati del body della request

Endpoint	/tables/:id
Metodo	DELETE
Parametri	
Formato dei dati della richiesta	
Formato dei dati della risposta	{ }
Descrizione	Elimina il tavolo con id uguale a quello specificato nel parametro dell'endpoint

## Users endpoints

Endpoint	/users
Metodo	GET
Parametri	
Formato dei dati della richiesta	
Formato dei dati della risposta	<a href="#">UserModel[]</a>
Descrizione	Restituisce la lista di tutti gli utenti

Endpoint	/users/current
Metodo	GET
Parametri	
Formato dei dati della richiesta	
Formato dei dati della risposta	<a href="#">UserModel</a>
Descrizione	Restituisce un JSON che descrive l'utente autenticato che invia la request

Endpoint	/users/:id
Metodo	GET
Parametri	
Formato dei dati della richiesta	
Formato dei dati della risposta	<a href="#">UserModel</a>
Descrizione	Restituisce un JSON che descrive l'utente che corrisponde all'id specificato nell'URL

Endpoint	/users
Metodo	POST
Parametri	
Formato dei dati della richiesta	<pre>{   "username": string,   "firstname": string,   "lastname": string,   "role": string,   "password": string }</pre>
Formato dei dati della risposta	<a href="#">UserModel</a> N.B. l'hash della password non viene restituito TODO verifica se viene fornito il token
Descrizione	Crea un nuovo utente e restituisce un JSON che lo descrive

Endpoint	/users/login
Metodo	POST
Parametri	
Formato dei dati della richiesta	<pre>{   "username": string,   "password": string }</pre>
Formato dei dati della risposta	<a href="#">UserModel</a> N.B. l'hash della password non viene restituito
Descrizione	Effettua il login ed aggiorna il refresh_token dell'utente nel database

Endpoint	/users/refresh
Metodo	POST
Parametri	
Formato dei dati della richiesta	{ "user_id": ObjectId, "refresh_token": string }
Formato dei dati della risposta	{ "token": string, "refresh_token": string }
Descrizione	Viene verificato che l'utente con id uguale a user_id abbia, nel database, la stessa stringa di refresh_token ricevuta nel body, in tal caso il refresh token viene aggiornato nel database.

Endpoint	/users/:id
Metodo	PUT
Parametri	
Formato dei dati della richiesta	{ "username": string, "firstname": string, "lastname": string, "role": string, "password" }
Formato dei dati della risposta	<a href="#">UserModel</a> N.B. l'hash della password non viene restituito
Descrizione	Aggiorna i dati dell'utente corrisponde all'id :id (specificato nell'URL) con i dati contenuti nel body

Endpoint	/users/:id
Metodo	DELETE
Parametri	
Formato dei dati della richiesta	
Formato dei dati della risposta	{ }
Descrizione	<p>Imposta il deletion_date dell'utente alla data in cui viene lanciata la request.</p> <p>In questo modo gli si impedisce l'accesso al sistema a partire da tale data-ora</p>

## La directory models

Le collection mantenute nel database MongoDB hanno la struttura definita nei sei file presenti.

- **drink**
- **food**
- **order**
- **suborder**
- **table**
- **user**

### Drink model

Le categorie valide per le bibite sono descritte nel file enum.ts, sono:

- Analcolico
- Alcolico

drink.model.ts				
Scopo: la collection rappresenta le bibite venduti dal ristorante.				
Field	Tipo	Vincoli	Valore default	Descrizione
name	String	Lunghezza minima = 1	/	Il nome del drink
price	Number	required = true	/	Il prezzo di vendita del drink
time	Number	required = true	/	Il tempo di preparazione stimato (minuti)
category	String	la categoria deve appartenere ad una categoria di bevande valida	/	
available	Boolean	required = true	/	Indica se una bibita è disponibile. Es. rimozione temporanea di una bibita perché sono esaurite le scorte

## Food model

Le categorie valide per le bibite sono descritte nel file enum.ts, sono:

- Dessert
- Antipasto
- Primo Piatto
- Secondo Piatto
- Contorno
- Dolce

food.model.ts				
Scopo: la collection rappresenta i cibi venduti dal ristorante.				
Field	Tipo	Vincoli	Valore default	Descrizione
name	String	Lunghezza minima = 1	/	Il nome del cibo
price	Number	required = true	/	Il prezzo di vendita del cibo
time	Number	required = true	/	Il tempo di preparazione stimato (minuti)
category	String	la categoria deve appartenere ad una categoria di cibi valida	/	
available	Boolean	required = true	/	Indica se un cibo è disponibile. Es. rimozione temporanea di un cibo perché non è un prodotto di stagione

## Order model

order.model.ts				
Scopo: la collection rappresenta la testata degli ordini effettuati attualmente ed in passato.				
Field	Tipo	Vincoli	Valore default	Descrizione
waiter_id	ObjectId	required = true	/	Il cameriere associato a questo ordine
table_id	ObjectId	required = true	/	Il tavolo da cui l'ordine è stato effettuato
clients	Number	minimo = 1 required = true	/	Il numero di clienti serviti in questo ordine
paid	Boolean		false	Indica se l'ordine è stato pagato o meno
served	Boolean		false	Indica se l'ordine è stato servito meno
amount	Number	minimo = 0	0	Il totale da pagare temporanea di un cibo perché non è un prodotto di stagione
createdAt TODo	Date	/	La data di creazione dell'ordine	Indica quando è stato effettuato l'ordine
updatedAt TODo	Date	/	La data di creazione dell'ordine	Indica quando è stato aggiornato l'ordine



## Suborder model

Gli stati validi per il cibo e bibite sono descritti nel file enum.ts, attualmente sono:

- In coda
- In preparazione
- Servito
- Pagato

suborder.model.ts				
<p><b>Scopo:</b> la collection rappresenta il dettaglio degli ordini, essi descrivono il cibo e le bibite che sono state richieste in un ordine (order model).</p> <p>In ogni ordine possono esserci più suborder, se del cibo viene ordinato quando lo stato è ancora "In coda" non vengono creati nuovi sotto ordini ma il cibo viene aggiunto al sotto ordine già presente, lo stesso vale per le bibite.</p>				
Field	Tipo	Vincoli	Valore default	Descrizione
order_id	ObjectId	required = true	/	Indica a quale ordine è legato questo suborder
ordered_foods	[orderedFoodSchema]		[]	Indica il cibo ordinato dai clienti
ordered_drinks	[orderedDrinkSchema]		[]	Indica le bibite ordinate dai clienti
state_foods	String	required: true; lo stato del cibo deve appartenere ad uno stato valido	"In coda"	Indica lo stato del cibo
state_drinks	Boolean	required: true; lo stato del cibo deve appartenere ad uno stato valido;	"In coda"	Indica lo stato delle bibite
createdAt TODO	Date	/	La data di creazione dell'ordine	Indica quando è stato effettuato l'ordine
updatedAt TODO	Date	/	La data di creazione dell'ordine	Indica quando è stato aggiornato l'ordine

## orderedFoodSchema

orderedFoodSchema				
I sotto ordini contengono un array di questo tipo per descrivere i cibi ordinati				
Field	Tipo	Vincoli	Valore default	Descrizione
food_id	ObjectId	required = true	/	Indica l'id del cibo che è stato ordinato
quantity	Number	minimo = 1 required = true	/	Indica la quantità ordinata per tale cibo
time	Number	minimo = 0	/	Indica il tempo di preparazione del cibo
name	String	required: true; lo stato del cibo deve appartenere ad uno stato valido	/	Indica il nome del cibo
price	Number	minimo = 0 required = true	/	Indica il prezzo del cibo, utile ai fini statistici , altrimenti se il prezzo di questo cibo varia risultano dati incoerenti
prepared_by	ObjectId	/	null	Indica l'id del cuoco che ha preparato la quantità di cibi richiesta

## orderedDrinkSchema

orderedDrinkSchema				
I sotto ordini contengono un array di questo tipo per descrivere le bibite ordinate				
Field	Tipo	Vincoli	Valore default	Descrizione
drink_id	ObjectId	required = true	/	Indica l'id della bibita che è stato ordinata
quantity	Number	minimo = 1 required = true	/	Indica la quantità ordinata per tale bibita
time	Number	minimo = 0	/	Indica il tempo di preparazione della singola bibita
name	String	required: true; lo stato del cibo deve appartenere ad uno stato valido	/	Indica il nome della bibita
price	Number	minimo = 0 required = true	/	Indica il prezzo della bibita, utile ai fini statistici , altrimenti se il prezzo di questa bibita varia risultano dati incoerenti
prepared_by	ObjectId	/	null	Indica l'id del barista che ha preparato la quantità di bibite richieste

## Table model

table.model.ts				
Scopo: la collection rappresenta i tavoli presenti nel ristorante				
Field	Tipo	Vincoli	Valore default	Descrizione
table_number	Number	required = true, unique = true	/	Indica il numero del tavolo
available	[orderedFoodSchema]	required = true,	true	Indica se il tavolo è disponibile oppure occupato
total_seats	Number	required = true, minimo = 1	[]	Indica il numero massimo di posti per il tavolo
clients	Number	minimo = 0, massimo = total_seats	0	Indica il numero di clienti seduti, altrimenti 0
waiter_id	ObjectId		null	Indica l'id del cameriere che si occupa del tavolo
order_id	ObjectId		null	Indica l'id della testata dell'ordine fatto in questo tavolo

## User model

I ruoli validi per gli utenti sono descritte nel file enum.ts, essi sono:

- cassa
- cameriere
- cuoco
- barista

user.model.ts				
Scopo: la collection rappresenta i gli utenti registrati				
Field	Tipo	Vincoli	Valore default	Descrizione
username	String	required = true, unique = true, lunghezza minima = 1		Indica lo username, univoco dell'utente
hash	String	required = true	[]	Indica l'hash della password, comprensivo di salt
firstname	String	required = true, lunghezza minima = 1	"In coda"	Indica il nome dell'utente
lastname	String	required = true, lunghezza minima = 1		Indica il cognome dell'utente
role	String	required = true, il ruolo deve appartenere ad uno dei ruoli validi definiti nell'enum		Indica il ruolo dell'utente
delation_date	Date		null	Se l'utente viene eliminato dalla cassa delation_date assume il valore della data corrente
refresh_token	String		null	Al momento del login viene assegnato un

				nuovo refresh token all'utente. Se il token scade di accesso scade il refresh_token sarà utile nel processo di rinnovamento
--	--	--	--	---

## La directory services

I services definiscono i metodi invocabili dai controllers e hanno lo scopo di effettuare le operazioni CRUD (Create, Read, Update, Delete) interagendo direttamente con la base dati. Ogni services ha un area di competenza che può essere: bibite, cibi, ordini, sotto ordini, tavoli e utenti e si occupa di soddisfare le request delle [api](#) descritte

# Front-End

Il Front end, realizzato con il framework Angular è stato organizzato con 3 cartelle Components, Services e Models. I Models rispecchiano quelli definiti nel back-end.

## Components

Di seguito sono elencati i vari components definiti per l'interfaccia utente.

### App

È la radice dell'applicazione, contiene il componente per la visualizzazione delle pagine in modo dipendente dallo stato corrente del router.

### Login

Componente per il login al sistema in cui è necessario inserire username e password validi per accedere.

### Navbar

Barra di navigazione visibile solo se autenticati. Se l'utente è cassa potrà accedere al backoffice cliccando sull'apposita icona (ingranaggio).

### Dashboard

Componente che appare dopo aver effettuato il login, il contenuto varia in base al ruolo dell'utente.

La cassa e i camerieri vedono i tavoli con la relativa situazione attuale.

- **serve-queue:**  
Questo componente, è visibile esclusivamente ai camerieri e indica tramite due liste se ci sono cibi o bibite da servire.
- **Preparation-queue:**  
Questo componente rappresenta la lista degli ordini da preparare per i cuochi ed i baristi, i cuochi vedranno solo i cibi mentre i baristi solo le bevande.
- **Dialog-payment:**  
Finestra di dialogo con un riepilogo di tutti i cibi e le bevande acquistati dal tavolo che sta pagando.
- **Dialog-table:**  
Finestra di dialogo per la creazione (da parte della cassa) e per l'occupazione dei tavoli (cassa e camerieri).

### Order

Componente dove il cameriere prende le ordinazioni per un tavolo.

## Back office

Componente per gestire gli utenti (registrazione ed eliminazione), gestire i cibi e le bevande ed infine visualizzare le statistiche per la cassa.

## Stats

Questo componente viene incluso nel back office e tramite 6 tabs e mostra varie statistiche elencate di seguito.

- **Money-stats**  
Componente per visualizzare quanti clienti sono stati serviti ed il guadagno della giornata.
- **Food-stats e Drink-stats**  
Componente per visualizzare le statistiche relative a cibi e bibite, viene indicata la quantità venduta di ogni cibo e bibita presente.
- **Waiter-stats**  
Componente per visualizzare le statistiche dei camerieri quindi vedere quanti clienti hanno servito.
- **Barman-stats e Cook-stats**  
Componente per statistiche dei baristi e dei cuochi per vedere rispettivamente quante bibite e cibi hanno preparato.

## Services

Di seguito sono elencati i vari services utilizzati per permettere l'interazione.

## Guard

Sono presenti tre services che implementano CanActivate per stabilire se una route può essere mostrata ad un utilizzatore del sistema.

- **auth-guard.service.ts** guardia del router '' e /dashboard impedendo l'accesso a tale route se l'utente non è autenticato.
- **role-guard.service.ts** guardia del router '/backoffice' e 'orders/:table\_id' che verifica se l'utente ha il ruolo adatto per accedere ad una router, in caso contrario viene rimandato alla route di login.
- **no-auth-guard.service.ts** guardia del router '/login' che verifica se l'utente si è autenticato. Se risulta autenticato, lo reindirizza alla route '/dashboard'.



## Services per le api

- **drink.service.ts** :  
servizio per effettuare request agli endpoint delle bibite;
- **food.service.ts**:  
servizio per effettuare request agli endpoint dei cibi;
- **order.service.ts**:  
servizio per effettuare request agli endpoint degli ordini;
- **table.service.ts**:  
servizio per effettuare request agli endpoint dei tavoli;
- **user.service.ts**:  
servizio per effettuare request agli endpoint degli utenti;
- **req-interceptor.service.ts**:  
servizio che intercetta le richieste tutte le request HTTP, aggiungendo ad ogni request (eccetto quella per il login) il Bearer Token dell'utente nell'header.  
In caso di response con errore questo service prova a rinnovare il token, poiché l'errore potrebbe essere causato da una richiesta con token scaduto.
- **SocketIO.service.ts**:  
servizio che inizializza socket.io per mandare le notifiche ai vari utenti che si mettono in ascolto, quando riceve un emit dal server lo comunica ai client connessi.

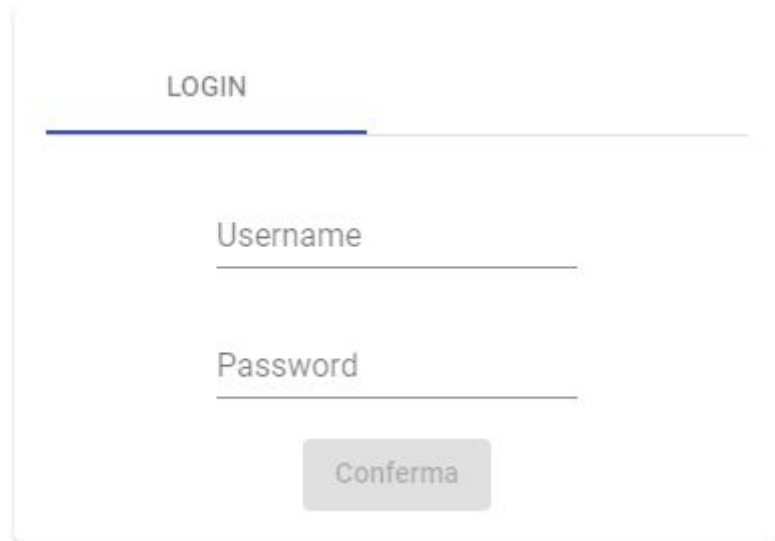
## Routes

- **"/"**: reindirizza al path **"/dashboard"** solo se l'utente è autenticato, altrimenti va al path **'login'**;
- **"/login"**: path del componente di login, se l'utente risulta autenticato, esso viene reindirizzato al path della  **'/dashboard'**;
- **"/dashboard"**: path della pagina dashboard. Se l'utente non è autenticato, esso viene reindirizzato al path di **'login'**;
- **"/backoffice"**: path della pagina backoffice. L'utente deve essere autenticato come cassa altrimenti esso viene reindirizzato al path di login;
- **"/orders/:id"**: path della pagina dell'ordine. L'utente deve essere autenticato come cameriere altrimenti viene reindirizzato al path di login.

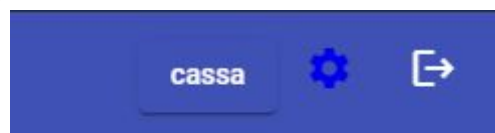
# Workflow

## Autenticazione

Per prima cosa l'utente dovrà accedere al sistema con un username ed una password corretti, altrimenti apparirà una snack bar ad indicare che i dati immessi sono errati.

A login form with a title 'LOGIN' at the top. Below the title are two input fields: 'Username' and 'Password'. At the bottom of the form is a button labeled 'Conferma'.

Dopo il login l'utente vede nella navbar il proprio ruolo. Solo se l'utente è cassa potrà accedere al backoffice per la gestione.



## Backoffice

Dal backoffice la cassa può gestire gli utenti, i cibi, le bibite e vedere varie statistiche.

### Gestione utenti

<

REGISTRAZIONE

UTENTI

CIBI

>

Nome

Cognome





Username

Ruolo

Password

Conferma

Filter

Username	Nome	Ruolo	
barista	Paolo Porcellato	barista	
cameriere	Matteo Schizzerotto	cameriere	
cuoco	Goran Gajic	cuoco	
cassa	admin admin	cassa	

## Gestione cibi

I cibi presenti possono essere eliminati oppure indicati come non disponibili, inoltre è possibile aggiungerne di nuovi indicando nome, categoria, prezzo e tempo di preparazione.

Filter

Nome	Categoria	Prezzo	Tempo	Disponibile	
Cotoletta e patatine fritte	Secondo Piatto	7.5	15	<input checked="" type="checkbox"/>	
Fagioli in salsa	Contorno	3.5	4	<input checked="" type="checkbox"/>	
Pasta allo scoglio	Primo Piatto	14	6	<input checked="" type="checkbox"/>	

Aggiungi un cibo

Clicca per visualizzare



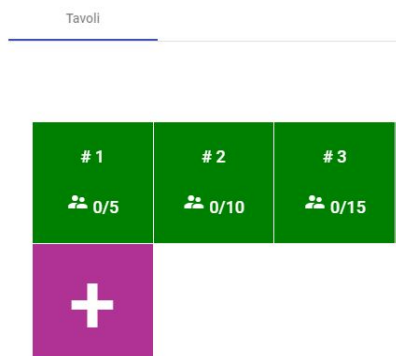
## Visualizzazione statistiche

Sono visualizzabili 6 diversi tipi di statistiche, nell'immagine seguente è indicato il numero totale dei clienti con il relativo guadagno.

<	Clients	Cibi più venduti	Bibite più vendute	Camerieri	Cuochi	Baristi	>
Numero clienti		Guadagni della giornata					
18		709.5					

## Gestione tavoli

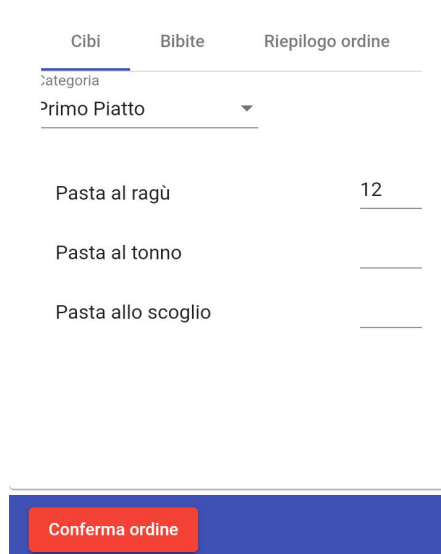
La cassa può aggiungere un nuovo tavolo alla sala cliccando l'apposito pulsante "+", dovrà indicare il numero del tavolo ed il numero di posti a sedere.



Quando dei clienti vogliono ordine, il cameriere può occupare un tavolo cliccando sopra ad esso o essere assegnato dalla cassa.



In seguito i clienti comunicano cosa vogliono ordinare ed il cameriere lo indica nell'apposita schermata.



Alla fine dell'ordinazione il cameriere può vedere un riepilogo e volendo, può leggerlo ai clienti e chiedere se è corretto.

**WATTENE** cameriere 

Cibi

Bibite

**Riepilogo ordine**

x12 Pasta al ragù  
x22 Acqua naturale

Conferma ordine

La cassa vede in ogni momento la situazione dei tavoli vedendo qual è lo stato dei cibi e delle bibite chi è il cameriere assegnato all'ordine.

**Matteo Schizzerotto**

**Cibo In coda**

**Bibite In coda**

**# 1**

 **2/5**

Quando i clienti decidono di pagare la cassa dovrà cliccare sopra ad un tavolo e selezionare “Paga” oppure “Paga e libera tavolo” ed apparirà un riepilogo con il prezzo.

Elimina ordine

Paga

Paga e libera tavolo

**Pagamento**

72€ x12 Pasta al ragù  
55€ x22 Acqua naturale

**Totale: 127€**

Paga

## Gestione coda di preparazione

I cuochi ed i baristi vedono in tempo reale la coda di preparazione.

Il colore rosso indica che il cibo/bibita non è stato preparato mentre il verde indica che è pronto.

Quando tutto è pronto sarà possibile cliccare il tasto “Pronto” ed il cameriere provvederà a servire l’ordine.

### Ordini

Pronto	Acqua frizzante x3 Tempo totale: 3 min	Sprite x2 Tempo totale: 2 min	Tè alla pesca x1 Tempo totale: 1 min	
Pronto	Acqua frizzante x1 Tempo totale: 1 min	Acqua naturale x3 Tempo totale: 3 min	Coca cola x4 Tempo totale: 4 min	Fanta x5 Tempo totale: 5 min

### Ordini

Pronto	Acqua frizzante x3 Tempo totale: 3 min	Sprite x2 Tempo totale: 2 min	Tè alla pesca x1 Tempo totale: 1 min	
Pronto	Acqua frizzante x1 Tempo totale: 1 min	Acqua naturale x3 Tempo totale: 3 min	Coca cola x4 Tempo totale: 4 min	Fanta x5 Tempo totale: 5 min