ADVANCED PROGRAMMING LANGUAGES

Ingegneria Informatica LM-32

A.A. 2021/2022

DinerHub

Sviluppo di un'applicazione per la gestione degli ordini nei ristoranti

Studenti:

Amenta Daniele
Matricola: 1000027022

D'Agosta Daniele
Matricola: 1000027035

GitHub: DinerHub

Sommario

1. Introduzione

Lo scopo dell'elaborato è quello di realizzare un'applicazione che consenta la gestione degli ordini nei ristoranti.

Il sistema sarà composto da:

- **Client**: si occuperà di interagire con le utente e di visualizzare l'interfaccia grafica;
- **Server**: si occuperà di rispondere alle richieste degli utenti, come ad esempio, login, registrazione, effettuazione ordini, e dell'interazione con il database per la gestione della persistenza;
- **Stats Server**: si occuperà di rendere disponibili le statistiche di sistema per le utenze disponibili.

Sono previste tre tipologie di utenze:

- **User**: deve potersi registrare/loggare al sistema tramite l'interfaccia grafica, visualizzare e modificare le proprie informazioni personali, visualizzare la lista dei ristoranti disponibili e i propri menù, poter effettuare ordini e visualizzare statistiche sulla sua interazione con il sistema;
- **Admin**: si occupa della gestione (inserimento, modifica e rimozione) dei ristoranti e degli utenti, deve poter visualizzare le proprie informazioni personali ed eventualmente modificarle e visualizzare le statistiche del sistema;
- **Restaurant**: deve poter gestire gli ordini che riceve, visualizzare e modificare il proprio menù e visualizzare statistiche sugli ordini ricevuti.

2. Soluzioni tecnologiche

Per lo sviluppo dei moduli che compongono il sistema sono state selezionate le seguenti soluzioni tecnologiche:

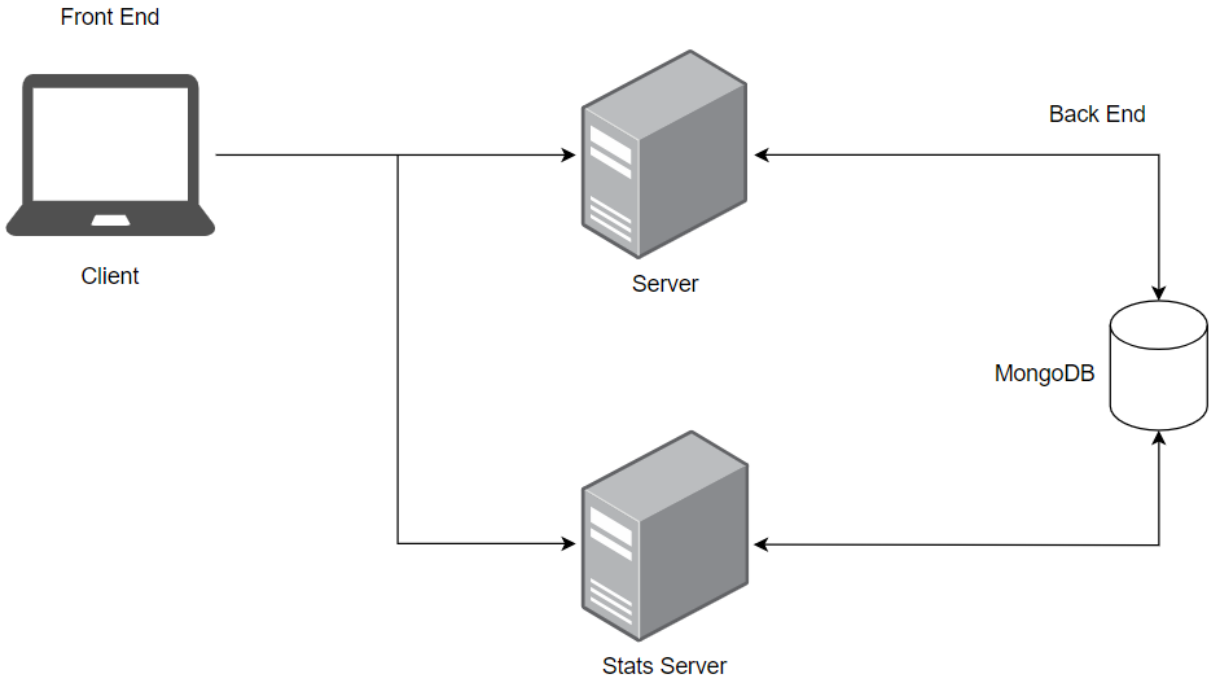
- **Client**: sviluppato in C++ con relativo utilizzo dell'estensione QT Visual Studio Tools per Visual Studio per realizzare l'interfaccia grafica e della libreria CPR per effettuare le richieste http ai REST server;
- **Server**: sviluppato in C#, facendo uso dei seguenti pacchetti NuGet:
  - **Microsoft.AspNetCore.Mvc.NewtonofJson**: utile per la gestione delle API response in formato JSON;
  - **MongoDB.Driver**: necessario per l'interazione con il database MongoDB;
  - **Microsoft.AspNetCore.Authentication.JwtBearer**, **Microsoft.IdentityModel.Tokens** e **System.IdentityModel.Tokens.Jwt**: necessari per generare e validare i token JWT e proteggere le route esposte dai controller.
- **Stats Server**, sviluppato in Python v3.10 con utilizzo delle seguenti librerie:
  - **pymongo**;
  - **matplotlib**;
  - **numpy**;
  - **flask**;
  - **pyjwt**.

L'elaborato è stato sviluppato in ambiente Windows, utilizzando **Visual Studio 2022** come IDE per il Client C++ e il Server C# e **PyCharm** come ambiente di sviluppo per lo Stats Server in Python. Per lo sviluppo della GUI è stato utilizzato **QTDesigner**.

3. Architettura

L'architettura di riferimento è mostrata in figura. Il front end dell'applicazione è costituito dal client e della GUI eseguiti sugli end points, tramite i quali gli utenti interagiscono con il sistema. Il back end invece è costituito da due server.

La comunicazione tra il client ed i server è bidirezionale e basata sull'architettura REST.



3.1 Client [C++]

Questa entità è composta da due parti: la GUI che permette all'utente di poter utilizzare l'applicativo e visualizzare le informazioni e la logica applicativa che implementa la comunicazione del Client con i due server e implementa le reali funzionalità del sistema associandole agli elementi dell'interfaccia grafica.

All'interno della directory `src/Client` sono presenti i file che rappresentano l'implementazione del modulo.

3.1.1 GUI

L'interfaccia grafica è stata realizzata utilizzando **QTDesigner**. Le interfacce grafiche prodotte sono le seguenti:

- ***Client.ui***: il file contiene l'interfaccia che contente a utenti, admin e ristoranti di effettuare il login; permette inoltre il passaggio all'interfaccia per effettuare la registrazione;
- ***RegisterGUI.ui***: il file contiene l'interfaccia che consente la registrazione degli utenti;
- ***UserGUI.ui***: il file contiene l'interfaccia grafica che consente di effettuare le operazioni ad utente registrato al sistema. Tale interfaccia grafica è organizzata in tab tramite uno QStackedWidget. La GUI fornisce le seguenti tab:
    - ***Order***: permette all'utente di visualizzare la lista di ristoranti presenti. Se l'utente clicca su un ristorante, nella seconda tabella verrà mostrato il menù del ristorante scelto. Per effettuare un ordine, bisogna scegliere le quantità delle portate mostrate nel menù;

    - ***Order Summary***: mostra il riepilogo degli ordini effettuati dall'utente, ordinati dal più recente al meno recente. Visualizza inoltre, in un'apposita colonna, lo stato dell'ordine;
    - ***Profile***: visualizza tutti i dati personali di un utente, consentendo di modificarli. Ogni qual volta si deve modificare un dato, bisogna inserire la password. Da questa tab è anche possibile apportare modifiche al saldo dell'utente, sia in positivo che in negativo;
    - ***Stats***: mostra le statistiche circa l'interazione dell'utente con il sistema, mostrando anche i relativi grafici;
    - ***Logout***: permette all'utente di terminare la sessione in corso, re-indirizzandolo all'interfaccia del login.
- ***AdminGUI.ui***: il file contiene l'interfaccia grafica che consente di effettuare le operazioni all'admin del sistema. Tale interfaccia grafica è organizzata in tab tramite uno QStackedWidget. La GUI fornisce le seguenti tab:
    - ***User List***: permette di visualizzare la lista degli utenti registrati al sistema e permette, cliccando sull'utente, di eliminare il suo profilo;
    - ***Restaurant List***: mostra sulla destra la lista dei ristoranti presenti nel sistema; cliccando su uno di essi si può procedere all'eliminazione del relativo profilo. Sulla parte sinistra è invece permessa la creazione di un nuovo ristorante; questo è l'unico punto da cui è permessa tale creazione;
    - ***Profile***: visualizza i dati dell'admin (e-mail e password), consentendo di modificarli;
    - ***Stats***: mostra le statistiche generali del sistema, mostrando anche i relativi grafici;
    - ***Logout***: permette all'admin di terminare la sessione in corso, re-indirizzandolo all'interfaccia del login.
- ***RestaurantGUI.ui***: il file contiene l'interfaccia grafica che consente di effettuare le operazioni di gestione degli ordini ad un ristorante. Tale interfaccia grafica è organizzata in tab tramite uno QStackedWidget. La GUI fornisce le seguenti tab:
    - ***Orders to prepare***: mostra la lista degli ordini ancora da preparare; cliccando su uno di essi è possibile cambiare il suo stato in pronto al ritiro;
    - ***Orders to collect***: mostra la lista degli ordini pronti al ritiro; cliccando su uno di essi è possibile cambiare il suo stato in ritirato;
    - ***Orders history***: mostra la lista degli ordini già ritirati;
    - ***Profile***: visualizza tutti i dati del ristorante, consentendo di modificarli. Ogni qual volta si deve modificare un dato, bisogna inserire la password;
    - ***Restaurant Menu***: nella parte destra dell'interfaccia si può visualizzare il menu del ristorante e cliccando su uno dei piatti, è possibile rimuoverlo dal menu. Nella parte sinistra è invece permessa la creazione di nuovi piatti da aggiungere al menù;
    - ***Stats***: mostra le statistiche sugli ordini ricevuti dal ristorante, mostrando anche i relativi grafici;
    - ***Logout***: permette al ristorante di terminare la sessione in corso, re-indirizzandolo all'interfaccia del login.

3.1.2 Logica applicativa

La logica è stata suddivisa tra diversi file.

Da sottolineare è il funzionamento delle variabili con il prefisso `session_`, utilizzate per salvare, per tutta la durata dell'utilizzo dell'interfaccia da parte dell'utente, i dati quali il suo identificativo, il token e informazioni utili per il funzionamento dell'interfaccia.

Il token, insieme all'identificativo del profilo, viene restituito, indipendentemente dal ruolo dell'utente che fa uso del sistema, a seguito del login andato a buon fine, quindi letto dalla risposta della chiamata effettuata al server.

Nel file di header **Utils.h** si trova la definizione dei prototipi di quattro funzioni:

- **string getOrderStatus(int status)**: fornendo in ingresso l'id dello stato dell'ordine, restituisce l'apposita stringa, in base al mappamento definito;
- **string getDishType(int type)**: fornendo in ingresso l'id del tipo di portata, restituisce l'apposita stringa, in base al mappamento definito;
- **string sha256(const string str)**: restituisce il corrispettivo della stringa passata come parametro della funzione, codificata con l'algoritmo SHA-256. Questa funzione viene utilizzata ogni qual volta bisogna inserire la password nella chiamata di login verso il server;
- **string currentISO8601TimeUTC()**: restituisce la data attuale in formato ISO-8601. Viene utilizzata per fornire al server la data di creazione dell'ordine.

Tutte le volte che un utente scrive dei dati all'interno di un input field, essi vengono validati grazie all'utilizzo delle espressioni regolari (regex), mediante l'uso della funzione `regex_match`, che ritorna un valore booleano: `true` se la condizione è soddisfatta, `false` altrimenti.

3.2 Server [C#]

3.2.1 Models

L'entità Server è stata sviluppata rispettando il pattern Model View Controller (MVC). Sono presenti i seguenti Models con i relativi attributi:

- **User.cs**
  - **UserId** (integer): identificativo univoco dell'utente;
  - **Name** (string): nome dell'utente;
  - **Surname** (string): cognome dell'utente;
  - **Email** (string): email del profilo utente;
  - **Password** (string): password del profilo utente;
  - **Phone** (string): numero di telefono dell'utente;
  - **Balance** (double): credito a disposizione dell'utente per poter effettuare gli ordini;
  - **BirthDate** (datetime): data di nascita dell'utente;
  - **IsAdmin** (boolean): variabile booleana che indica se si tratta di un amministratore o meno.
- **ReturnUser.cs**: entità che viene restituita a seguito di login di un utente/admin
  - **UserId** (integer): identificativo univoco dell'utente;
  - **Token** (string): token dell'utente;
  - **IsAdmin** (boolean): variabile booleana che indica se si tratta di un amministratore o meno.
- **Restaurant.cs**
  - **RestaurantId** (integer): identificativo univoco del ristorante;
  - **Name** (string): nome del ristorante;
  - **Address** (string): indirizzo del ristorante;
  - **Email** (string): email del profilo del ristorante;
  - **Password** (string): password del profilo del ristorante;
  - **Phone** (string): numero di telefono del locale;
  - **IsRestaurant** (boolean): variabile booleana sempre settata a `true`. Utile per le logiche di login.

- **ReturnRestaurant.cs**: entità che viene restituita a seguito di login di un ristorante
  - **RestaurantId** (integer): identificativo univoco del ristorante;
  - **Token** (string): token dell'utente;
  - **IsRestaurant** (boolean): variabile booleana sempre settata a `true`. Utile per le logiche di login.
- **Dish.cs**
  - **DishId** (integer): identificativo univoco della portata;
  - **Name** (string): nome della portata;
  - **Type** (DishTypes): tipo di portata;
  - **Price** (double): prezzo della portata;
  - **RestaurantId** (integer): identificativo univoco del ristorante a cui appartiene la portata.
- **DishTypes.cs**: enum che indica il tipo di portata
  - **0**: Appetizer
  - **1**: First Course
  - **2**: Second Course
  - **3**: Dessert
  - **4**: Drink
- **Order.cs**
  - **OrderId** (integer): identificativo univoco dell'ordine;
  - **Status** (OrderStatus): stato dell'ordine;
  - **Total** (double): importo totale dell'ordine;
  - **UserId** (int): identificativo univoco dell'utente che ha effettuato l'ordine;
  - **RestaurantId** (int): identificativo univoco del ristorante a cui si riferisce l'ordine;
  - **Dishes** (array - OrderDish): portate presenti nell'ordine;
  - **Date** (datetime): data e ora in cui è stato effettuato l'ordine.
- **OrderDish.cs**
  - **Dish** (Dish): portata;
  - **quantity** (int): quantità della portata.
- **OrderStatus.cs** enum che indica lo stato dell'ordine
  - **0**: Created
  - **1**: ReadyToPickup
  - **2**: Collected

3.2.2 Controllers

Di seguito i controller presenti:

- **ServerController.cs**: espone solamente l'API `/ping`, utile a fini di debug e per verificare liveness del server;
- **UserController.cs**: tale controller risponde al path `/api/user/` ed espone le seguenti API:

GET	`/user/all`	Restituisce la lista di tutti gli utenti registrati al sistema; da tale lista vengono esclusi i profili che presentano l'attributo IsAdmin pari a true. Restituisce tutti i dati solo se la richiesta viene effettuata da un admin, altrimenti restituisce solo il profilo del singolo utente.
GET	`/user/{id}`	Restituisce i dati dell'utente cui UserId è fornito come parametro della richiesta.

POST	/user/registration	Registra un utente nel sistema; se uno dei parametri forniti non rispetta le regex, viene restituito il codice di errore 400 (Bad Request). Ogni utente viene inserito con un credito iniziale 0.00. L'email fornita inoltre deve essere univoca e l'UserId viene assegnato dal server.
POST	/user/login	Permette l'autenticazione degli utenti, sia che siano admin, sia che siano ristoranti. In base all'utente fornito, varierà la risposta alla chiamata.
PUT	/user/update	Aggiorna i dati di uno specifico utente.
DELETE	/user/delete/{id}	Cancella l'utente cui UserId è fornito come parametro della chiamata.
PATCH	/user/update/balance	Aggiorna il credito dell'utente, sia in positivo che in negativo. Se il credito scende sotto la soglia dello 0.00, la richiesta restituirà il codice di errore 400 (Bad Request).
GET	/user/balance/{id}	Restituisce il credito di un utente cui UserId è fornito come parametro della chiamata.

All'interno di questo controller è inoltre presente la funzione **GetUserTokenDetails()**, utilizzata in fase di login, che restituisce l'utente a cui appartiene il token, decodificando i claims contenuti in esso.

- **OrderController.cs**: tale controller risponde al path /api/order/ ed espone le seguenti API:

GET	/order/all	Restituisce la lista di tutti gli ordini effettuati nella piattaforma. Tale API è autorizzata solo per gli admin.
GET	/order/{id}	Restituisce i dati dell'utente cui OrderId è fornito come parametro della richiesta
POST	/order/create	Consente l'inserimento di un nuovo ordine nel sistema; le regex lato server controllano solamente la validità dei campi quali UserId e RestaurantId, poiché gli altri, quali OrderId, Status e Total verranno sovrascritti dal server. Tale API calcola automaticamente il totale dell'ordine e crea l'ordine nello stato 'Created'.
PUT	/order/update	Aggiorna i dati di uno specifico ordine. Non sarà però possibile cambiare UserId e RestaurantId.
DELETE	/order/delete/{id}	Cancella l'ordine cui OrderId è fornito come parametro della richiesta. Tale API è autorizzata solo per gli admin.
PATCH	/order/update/status	Aggiorna lo stato di uno specifico ordine.
GET	/order/status/{id}	Restituisce lo stato dell'ordine cui OrderId è fornito come parametro della chiamata.
GET	/order/dish/{id}	Restituisce tutti i piatti contenuti nell'ordine cui OrderId è fornito come parametro della chiamata.
GET	/order/user/{id}	Restituisce tutti gli ordini di un utente cui UserId è fornito come parametro della chiamata.
GET	/order/restaurant/status	Restituisce tutti gli ordini di un ristorante con uno specifico stato.

- **RestaurantController.cs**: tale controller risponde al path /api/restaurant/ ed espone le seguenti API:

GET	`/restaurant/all`	Restituisce la lista di tutti i ristoranti presenti nel sistema.
GET	`/restaurant/{id}`	Restituisce i dati del ristorante cui RestaurantId è fornito come parametro della richiesta
POST	`/restaurant/create`	Consente l'inserimento di un nuovo ristorante nel sistema. Se uno dei parametri forniti non rispetta le regex, viene restituito il codice di errore 400 (Bad Request). L'email fornita inoltre deve essere univoca e il RestaurantId viene assegnato dal server.
PUT	`/restaurant/update`	Aggiorna i dati di uno specifico ristorante. Non sarà però possibile sovrascrivere la variabile IsRestaurant.
DELETE	`/restaurant/delete/{id}`	Cancella il ristorante cui RestaurantId è fornito come parametro della richiesta. Tale API è autorizzata solo per gli admin.

- **DishController.cs**: tale controller risponde al path `/api/dish/` ed espone le seguenti API:

GET	`/dish/all`	Restituisce la lista di tutte le portate presenti nel sistema.
GET	`/dish/{id}`	Restituisce i dati della portata cui DishId è fornito come parametro della richiesta
POST	`/dish/create`	Inserisce una nuova portata nel sistema.
PUT	`/dish/update`	Aggiorna i dati di una specifica portata. Non sarà però possibile sovrascrivere il RestaurantId.
DELETE	`/dish/delete/{id}`	Cancella la portata cui DishId è fornito come parametro della richiesta.
GET	`/dish/restaurant/{id}`	Ritorna tutte le portate cui RestaurantId è fornito come parametro della richiesta.

Qualsiasi altro path che non coincide con i precedenti, restituirà un response code 404 (Not Found).

3.2.3 Altre informazioni

All'interno della directory `src/Server/JwtManager` si trova la classe `JwtFunctions` che contiene le funzioni che permettono la generazione del token, sia se si tratti di un admin/utente, sia che si tratti di un ristorante.

Nel file `appsettings.json` sono memorizzate le variabili utili al funzionamento del sistema, quali le regex adottate, l'indirizzo del database MongoDB e le informazioni utili per generare il token JWT.

3.3 Stats Server [Python]

Questa entità si occupa di collezionare ed elaborare le statistiche relative al sistema e agli utenti durante l'esecuzione dell'applicazione. Inoltre, si occupa di creare grafici ad hoc per utenti, ristoranti e amministratori di sistema al fine di rendere visibili graficamente alcune statistiche.

3.3.1 API

Per le API esposte dallo Stats Server, eccetto la /ping, è necessario fornire come parametro della richiesta il token che si è ricevuto a seguito del login al sistema. Le API contenute nel file `UserStas.py` sono autorizzate solo per amministratori ed utenti; quelle contenute nel file

`RestaurantStats.py` sono autorizzate solo per amministratori e ristoranti, mentre quelle contenute nel file `AdminStats.py` sono autorizzate solo per gli amministratori.

3.3.2 UserStats.py

Per l'utente, vengono esposte le seguenti API, che calcolano le varie statistiche:

GET	`/userStats/countOrder/{id}`	Restituisce il numero di ordini effettuati dall'utente, cui UserId è fornito come parametro della richiesta.
GET	`/userStats/totalOrder/{id}`	Restituisce la somma di denaro spesa dal cliente in base agli ordini effettuati, cui UserId è fornito come parametro della richiesta.
GET	`/userStats/favoriteRestaurant/{id}`	Restituisce il ristorante preferito dell'utente, cui UserId è fornito come parametro della richiesta. Si intende 'ristorante preferito' quello in cui l'utente ha speso la cifra maggiore.
GET	`/userStats/orderfordayofweek/{id}`	Restituisce un grafico (*istogramma*) che mostra il numero di ordini che sono stati effettuati in ogni giorno della settimana da un utente, cui UserId è fornito come parametro della richiesta.
GET	`/userStats/orderCost/{id}`	Restituisce un grafico a linee che mostra la comparazione tra il totale degli ordini effettuati da un utente, cui UserId è fornito come parametro della richiesta.
GET	`/userStats/bestDish/{id}`	Restituisce la portata preferita dell'utente, cui UserId è fornito come parametro della richiesta. Per 'portata preferita' si intende quella che è stata ordinata in quantità maggiore.

3.3.3 AdminStats.py

Per l'amministratore del sistema, vengono esposte le seguenti API, che calcolano le varie statistiche:

GET	`/adminStats/allUsers`	Restituisce il numero totale di utenti registrati alla piattaforma.
GET	`/adminStats/totalOrder`	Restituisce il numero totale di ordini effettuati dal rilascio della piattaforma. Per convenzione si è scelto di partire dal 07/12/2021.
GET	`/adminStats/totals`	Restituisce l'ammontare totale di denaro speso nella piattaforma.
GET	`/adminStats/dailyOrders`	Restituisce la media di ordini giornalieri effettuati.
GET	`/adminStats/bestCustomer`	Restituisce l'utente che ha speso più soldi nella piattaforma da sempre.
GET	`/adminStats/bestCustomerOfMonth`	Restituisce l'utente che ha speso più soldi nella piattaforma nel mese corrente.
GET	`/adminStats/topgrossing`	Restituisce il ristorante che possiede gli incassi maggiori.

GET	`/adminStats/plotOrdersForHour`	Restituisce un grafico (*istogramma*) che mostra il numero di ordini divisi per fascia oraria.
GET	`/adminStats/plotAge`	Restituisce un grafico a torta che mostra i clienti suddivisi per fascia di età.

3.3.4 RestaurantStats.py

Per i ristoranti, vengono esposte le seguenti API, che calcolano le varie statistiche:

GET	`/restaurantStats/orderReceived/{id}`	Restituisce il numero totale di ordini ricevuti da un dato ristorante, cui RestaurantId è fornito come parametro della richiesta.
GET	`/restaurantStats/totalOrder/{id}`	Restituisce il totale degli incassi ottenuti da un dato ristorante, cui RestaurantId è fornito come parametro della richiesta.
GET	`/restaurantStats/bestCustomer/{id}`	Restituisce il miglior cliente di un dato ristorante, cui RestaurantId è fornito come parametro della richiesta. Per 'miglior cliente' si intende quello che ha speso la quantità maggiore di denaro presso il locale.
GET	`/restaurantStats/plotAge/{id}`	Restituisce un grafico a torta che mostra i clienti del ristorante suddivisi per fasce d'età, cui RestaurantId è fornito come parametro della richiesta.
GET	`/restaurantStats/plotOrderForHour/{id}`	Restituisce un grafico (*istogramma*) che mostra gli ordini suddivisi per fascia oraria di un dato ristorante, cui RestaurantId è fornito come parametro della richiesta.
GET	`/restaurantStats/customer/{id}`	Restituisce il numero di clienti 'fidelizzati' dal ristorante, cui RestaurantId è fornito come parametro della richiesta. Per 'cliente fidelizzato' si intende un cliente che ha effettuato almeno un ordine presso il locale.
GET	`/restaurantStats/bestDish/{id}`	Restituisce il piatto più venduto da un dato ristorante, cui RestaurantId è fornito come parametro della richiesta.

3.3.5 Altre informazioni

Lo storage delle immagini dei grafici avviene localmente. I grafici vengono suddivisi nella varie directory in base al tipo di API che le ha generate. Le directory sono: `/AdminStats`, `/RestaurantStats`, `/UserStats`.

Le immagini vengono identificate dal nome, spesso concatenato con l'ID dell'utenza che ne ha fatto richiesta, in modo che si eviti di generarne troppe. In questo caso, se un'immagine già esiste, viene sovrascritta da quella appena generata.

Quando un'immagine deve essere ritornata come risposta di un'API, viene restituito l'URL relativo della directory in cui è salvata l'immagine. Il client si occuperà solo della visualizzazione.

All'interno del file `variables.py` si trovano le funzioni che permettono la validazione del token inserito come parametro della richiesta. Tali funzioni saranno poi wrappate prima di ogni API esposta verso l'esterno del server. Le funzioni restituiscono come codice di errore:

- 403 (Forbidden) quando il token è assente;
- 401 (Unathorized) quando il token è scaduto o non valido per quella richiesta.

4. Avvio

Prima di avviare l'applicazione assicurarsi che le porte 5276, 5000 e 27017 siano disponibili perché utilizzate rispettivamente da Server, Stats Server e MongoDB.

Per avviare correttamente l'applicazione occorre:

- Avviare un'istanza di **MongoDB** sulla propria macchina;
- Compilare e lanciare il Server;
- Lanciare lo **Stats Server**, digitando il comando all'interno della directory del progetto: `python .\src\Stats-Server\main.py`
- Compilare e lanciare il **Client**.