UNIVERSITÀ DEGLI STUDI DI CATANIA

SoccerBet

Advanced Programming Languages a.a. 2019/2020

Raiti Mario O55000434
Nardo Gabriele Salvatore O55000430

Sommario

Elaborato disponibile al link: https://github.com/GabrieleNardo/SoccerBet

Introduzione

Lo scopo dell'elaborato è quello di realizzare un sistema di betting on line, che permetta agli utenti sottoscritti di poter piazzare scommesse calcistiche attraverso il portale. Il sistema sarà composto da: Client che si occuperà dell'interazione con le utenze e la visualizzazione dell'interfaccia grafica, il Server si occuperà delle operazioni di autenticazione degli utenti, validazione delle scommesse e interazione con il database per la gestione della persistenza e lo Stats Server che si occuperà di rendere disponibili le statistiche di sistema per l'Admin e le statistiche personali per i singoli iscritti.

Sono previste due tipologie di utenze:

- ***Admin***, che si occupa della gestione (inserimento, modifica e rimozione) degli eventi e quote, deve poter visualizzare informazioni relative al sistema (informazioni statistiche relative alle vittorie e giocate) e validazione delle giocate.
- ***Utente***, che deve potersi registrare/loggare al sistema attraverso la GUI, visualizzare le proprie informazioni personali relative al proprio profilo (dati personali, storico giocate e saldo), e poter piazzare la scommessa.

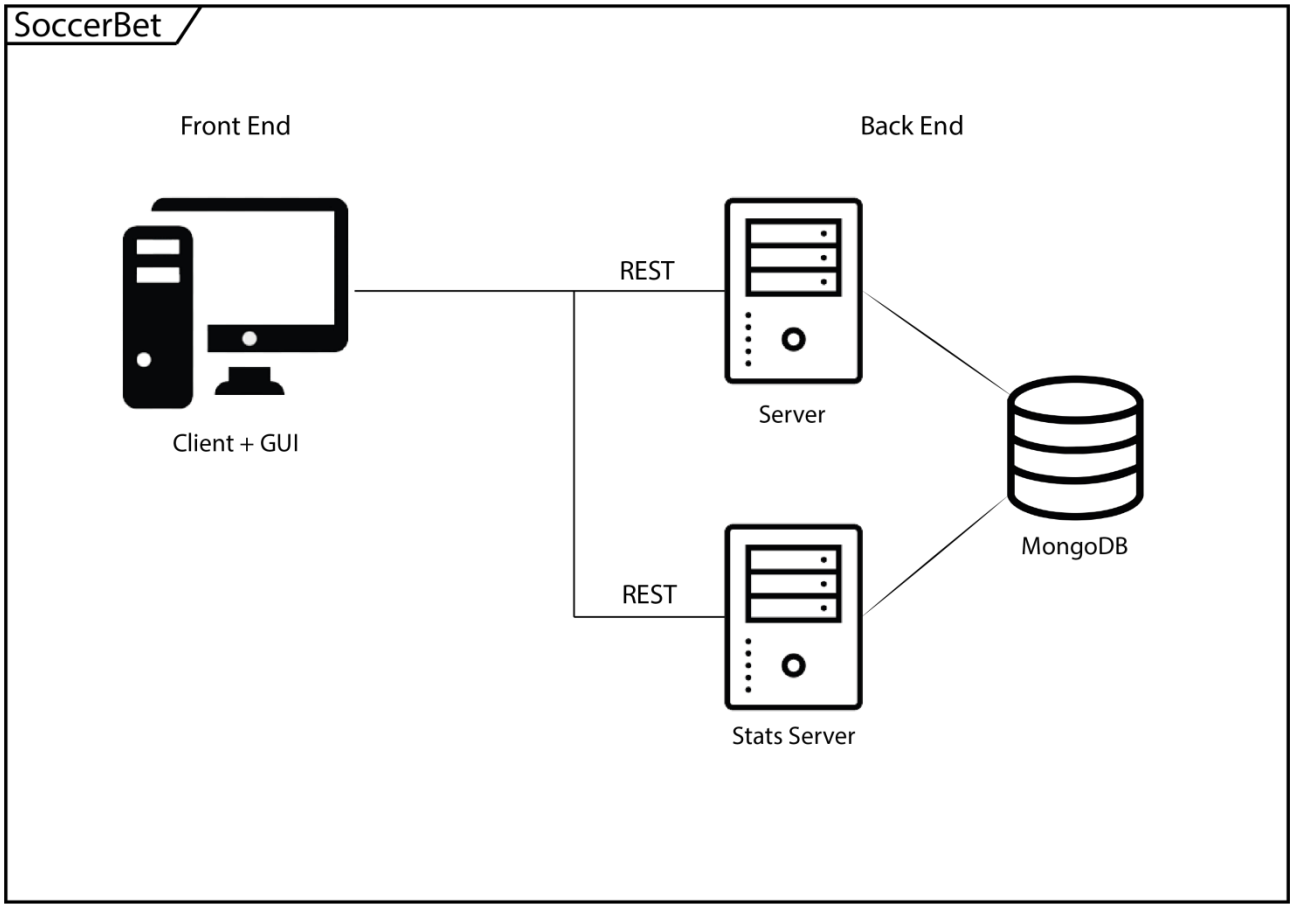
Soluzioni Tecnologiche

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

- **`Client`**: sviluppato in ***python***(ver. 3) con relativo utilizzo della libreria ***PyQT***(ver. 5) per l'implementazione della GUI.
- **`DB`**: come database è stato scelto ***MongoDB.***
- **`Server`**: sviluppato in ***C++*** con relativo utilizzo del driver ***mongocxx*** per l'integrazione col DB, della libreria ***RESTinio*** per implementare un'architettura REST per la comunicazione.
- **`Stats Server`**: sviluppato in ***R***, con relativo utilizzo della libreria ***mongolite*** per l'interazione col DB e la libreria ***plumber*** per rendere disponibili le statistiche sotto forma di API Rest.

L'elaborato inoltre è stato sviluppato in ambiente Windows, utilizzando ***VSCode*** come ambiente editor per il Client e Stats Server, e ***Visual Studio 2019*** come ambiente di Sviluppo per il Server. Per lo sviluppo della GUI è stato utilizzato ***QTDesigner***.

Architettura



L'architettura di riferimento per l'elaborato è mostrata nella figura in alto. Il front end dell'applicazione è costituito dal client e dalla GUI eseguiti sugli end point, tramite i quali gli utenti interagiscono col sistema. Il back end invece è

costituito da due server e il database in cui verranno memorizzate le informazioni di sistema, i dati degli utenti e i dati relativi agli eventi sportivi.

Sono presenti due server: il primo (che nella figura viene chiamato Server), si occupa di implementare la comunicazione con i client, le operazioni di autenticazione, la gestione delle scommesse e la gestione del database; il secondo, lo Stats Server, invece si occupa di elaborare le statistiche di sistema e dei singoli utenti interagendo col DB e di fornirle al client.

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

Client

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.

Come dipendenze esterne sono state utilizzate le seguenti librerie:

- **PyQt5** per la traduzione dell'interfaccia grafica in codice python;
- **Request** per implementare la comunicazione REST tra client e i server;
- **Json** per avere una rappresentazione standard dei dati scambiati tramite le chiamate;
- **Re** per utilizzare le espressioni regolari nelle form validator.

GUI

L'interfaccia grafica è stata realizzata utilizzando **QTDesigner**. I file ui prodotti sono stati poi tradotti in file python al fine di renderli utilizzabili dalla logica applicativa.

La traduzione di tali file è stata realizzata attraverso il comando ***pyuic5*** file.ui **–o** file.py.

Gli elementi grafici prodotti si trovano all'interno della directory `src/Client/UI_Files` e sono i seguenti:

- ***SoccerBet.ui***: il file contiene l'interfaccia grafica completa dell'applicativo. Tale interfaccia grafica è organizzata in pagine tramite uno QStackedWidget L'intera GUI è resa responsive applicando i diversi tipi di layout di QtCreator e applicando spacers tra i vari elementi. La GUI fornisce le seguenti pagine:
    - **Login***: permette di effettuare il login o di spostarsi alla pagina di registrazione;*
    - ***Registration****: contiene il form per la registrazione di un nuovo utente o admin e consente di tornare alla schermata di login;*
    - **AdminHome***:* home page dell'admin, fornisce una tabella che mostra tutti gli eventi presenti in sistema e il form per l'inserimento di un nuovo evento*;*
    - **AdminUsers***:* contiene una tabella che indica le informazioni di tutti gli utenti (no Admin) presenti nel sistema*;*
    - **AdminProfile***:* pagina che mostra le informazioni di profilo dell'admin e fornisce il form per la modifica delle informazioni correnti. Prevede inoltre un tasto per l'eliminazione del profilo*;*
    - **AdminBets***:* contiene una tabella che mostra le scommesse in corso nel sistema in attesa di essere validate. Inoltre sono presenti altre due tabelle che indicano quali delle precedenti scommesse sono state validate come vinte o perse*;*
    - **AdminStats***:* pagina contenente le informazioni statistiche di sistema e visualizzazione dei grafici relativi a tali statistiche*;*
    - **UserHome***:* home page dell'utente, fornisce una tabella che mostra tutti gli eventi presenti in sistema e tramite essa permette di creare la scommessa da giocare cliccando sulla quota all'interno della tabella e settando l'importo da giocare*;*
    - **UserProfile***:* pagina che mostra le informazioni di profilo dell'utente e fornisce il form per la modifica delle informazioni correnti. Prevede inoltre un tasto per l'eliminazione del profilo.
    - **UserStats***:* pagina contenente le informazioni statistiche dell'utente e visualizzazione dei grafici relativi a tali statistiche*;*
    - **UserBets***:* contiene una tabella che mostra le scommesse in corso per lo specifico utente in attesa di essere validate. Inoltre, sono presenti altre due tabelle che indicano quali delle precedenti scommesse sono state validate come vinte o perse.

- ***UserInfoDialog.ui***: il file contiene una finestra di dialogo (QDialog) con una QLabel all'interno della quale viene visualizzato il grafico delle finanze per lo specifico utente;
- ***EditEventDialog.ui***: il file contiene la finestra di dialogo (QDialog) contenente il form per l'inserimento delle informazioni per aggiornare gli eventi presenti nel sistema;
- ***EventBetDialog.ui***: il file contiene una finestra di dialogo (QDialog) contenente una tabella (QTableWidget) che andrà a visualizzare le informazioni degli eventi che costituiscono una specifica scommessa.

All'interno della directory `docs/imgs` sono presenti dei file png che mostrano gli elementi grafici sopra descritti.

Al seguente link è presente uno slide show che mostra una preview l'interfaccia grafica.

Logica Applicativa

La logica dell'applicativo è stata implementata in python e suddivisa su diversi file.

Un gruppo di file viene utilizzato per gestire gli elementi grafici. I file *Ui_EditEventsDialog.py*, *Ui_EventBetDialog.py*, *Ui_UserInfoDialog.py* rappresentano le traduzioni degli elementi grafici descritti sopra. Tali file vengono tutti importati all'interno del file ***GuiTools.py*** dove le classi che rappresentano gli elementi grafici vengono estese attraverso delle classi controller al fine di definire meccanismi per istanziarle e personalizzarle a run time. In tale file viene inoltre definita la classe *MessageBox* che si occupa di creare le message box utilizzate dal sistema per fornire avvisi o eventuali errori.

Sono presenti, inoltre, due file che implementano le chiamate REST per la comunicazione tra client e i due server. Tali file fanno uso della libreria requests di python:

- ***Rest_Request.py*** presenta le funzioni utilizzate per comunicare con il server, quindi implementa le chiamate alle varie APIs esposte. Inoltre, estendendo la classe *AuthBase* di *requests* viene definita la classe *BearerAuth* che permette di utilizzare l'autenticazione tramite Bearer Token prevista dal server.

- ***Stast_Rest_Requests.py*** presenta le funzioni utilizzate per comunicare con lo stats server, quindi implementa le chiamate alle varie APIs esposte.

Il file ***FormValidator.py*** implementa le funzioni che vengono utilizzate per validare gli input provenienti dai form di registrazione, di inserimento degli eventi. Tali funzioni si avvalgono del modulo ***re*** di python al fine di poter utilizzare delle espressioni regolari per validare i valori. Tali funzioni prevedono una tupla come valori di ritorno, un booleano che rappresenta il risultato della validazione ed un messaggio testuale.

Il file ***SoccerBet.py*** rappresenta il cuore del client, perché si fa carico della gestione dell'intera interfaccia grafica e della logica essendo il main per l'applicazione.

Viene definita la classe ***SoccerBetUiController*** che si occupa di istanziare e settare l'interfaccia grafica definita in SoccerBet.ui e associare ad essa la logica definita nei file precedenti. Tale classe è corredata da numerosi metodi.

Tra di essi, quelli che permettono la navigazione tra le varie pagine della GUI presentano tutti la seguente nomenclatura `show`*`NomePagina`*`()`. Tali metodi si occupano di settare come pagina corrente l'elemento desiderato dello stacked widget ed eseguire le eventuali chiamate per ottenere e settare i dati da visualizzare.

Degni di nota anche i metodi `initVariables()` e `connectSignals()` che vengono invocati all'interno dell'init della classe rispettivamente per inizializzare le variabili "globali" necessarie per il funzionamento del sistema e per associare agli elementi dell'interfaccia grafica (di norma bottoni) delle funzioni che ne definiscono i comportamenti in risposta agli eventi.

Sono inoltre presenti dei metodi che presentano tutti la seguente nomenclatura `get`*`NomeEntità`*`()`, si occupano di invocare le chiamate get definite all'interno degli specifici file, gestire i casi relativi ad i codici di ritorno e fornire i dati risultanti in json.

Tutti i metodi con le seguenti nomenclatura `on`*`AzioneNomeEntità`*`()`, `on`*`AzioneNomeEntità`*`Btn()`, si occupano di implementare le operazioni richieste dalla specifica azione (come ad esempio ottenere i data da un form, validarli e fare le rispettive chiamate gestendo i codici di ritorno e le eventuali

routine per gestire gli errori) che vengono poi associate a pulsanti o ad azioni specifiche sull'interfaccia grafica.

Infine sono presenti i metodi che seguono la seguente nomenclatura `setTable`*`TipoUtente`*`_get`*`NomeEntità`*`()`. Tali metodi si occupano di creare la specifica tabelle e riempirle con i dati in json forniti come parametro, settare i vari pulsanti se presenti ed associarne i comportamenti richiesti.

Server

L'entità Server principale è stata sviluppata in C++ su ambiente Visual Studio e contiene due file .h, contenenti la definizione delle strutture e funzioni utilizzate all'interno del Server, e due file .cpp, contenenti la logica del Server e l'interazione con il DB. La tecnologia utilizzata per la comunicazione con il DB Mongo è il driver mongocxx. Di seguito viene descritto dettagliatamente il contenuto dei file sopra citati:

- ***DocumentStruct.h:*** questo file contiene la definizione delle struct utilizzate per modellare le entità del nostro progetto. Queste entità sono: User (nome, cognome, email, password, balance, admin), Date (g, m, a, h, mm), Event (nome, data_inizio, data_fine, quote), Event_Outcome (event_id, nome_evento, nome_esito, quota), Bet(user_id, esito_eventi, ammontare_scommesso, potenziale_vincita), EndedBet (user_id, bet_id, importo_vinto, vinta). Inoltre, per la struct Date, è stato effettuato un overload degli operatori <, > e == per definire operazioni proprietarie sulle date.
- ***DBController.h:*** questo file contiene il costruttore/distruttore della classe Controller e la definizione dei prototipi delle funzioni che verranno implementate nella sua controparte .cpp. Queste funzioni riguardano le azioni da effettuare sulle strutture User, Event, Bet ed EndedBet.
- ***DBController.cpp:*** questo file contiene la logica della classe Controller. Esso implementa i metodi di inserimento, aggiornamento, cancellazione, ricerca mirata o generica di utenti, eventi, scommesse, scommesse vinte/perse. Inoltre, vi sono metodi aggiuntivi per aggiornare il saldo dell'utente, aggiornare le quote delle scommesse IN CORSO se un evento viene cancellato e le varie OPERATIONS che contengono l'esecuzione di più azioni attraverso chiamate a più funzioni consecutivamente. Importante dire che tutte le varie funzioni di

inserimento, aggiornamento, ricerca e cancellazione di elementi nel DB, vengono eseguite attraverso metodi specifici del modulo mongocxx, che ne permette l'interazione.

- ***Server.cpp:*** questo file rappresenta il fulcro del server. Esso istanzia il controller, istanzia la sessione e, attraverso il modulo RESTinio, risponde alle chiamate REST effettuate ai particolari path definiti. La creazione del token è gestita alla richiesta di login dell'utente, per poi essere ritornato al client. Ogni chiamata ha inoltre bisogno di un token valido per poter essere eseguita, motivo per cui viene utilizzata la funzione *on_request* per recuperare il token dalla richiesta e verificare che esso sia valido (in caso contrario l'utente non viene autorizzato a eseguire la chiamata). Per la creazione e la verifica del token jwt si è utilizzata la libreria jwt-cpp.

Di seguito vengono elencati tutti i path esposti dal Server per eseguire le varie azioni sul DB:

USERS

GET	`/user`	Ritorna un json contenente tutti gli utenti iscritti, se esistono utenti.
GET	`/user/:id`	Ritorna un json con l'utente che ha user_id uguale a quello specificato da id, se esiste.
POST	`/login`	Ritorna l'utente specificato dai campi email e password riempiti nel body, se esiste, e crea il token che viene ritornato al client.
POST	`/register`	Inserisce l'utente nel DB con i vari campi specificati nel body (name, surname, email, password, admin).
PUT	`/user/:id`	Aggiorna l'utente con user_id uguale a id nel DB, con i vari campi specificati nel body (name, surname, email, password, admin).
DELETE	`/user/:id`	Elimina l'utente dal DB se id è uguale ad un user_id degli utenti presenti nel DB.
PUT	`/user/balance/:id`	Aggiorna il balance dell'utente che ha user id uguale a id, se esiste, con il campo specificato nel body.

EVENTS

GET	`/event`	Ritorna un json contenente tutti gli eventi, se esistono.
GET	`/event/:id`	Ritorna un json con l'evento che ha event_id uguale a quello specificato da id, se esiste.
POST	`/event`	Inserisce l'evento nel DB con i vari campi specificati nel body

		(name, start_date, end_date, odds).
PUT	`/event/:id`	Aggiorna l'evento che ha event_id uguale a id nel DB, con i vari campi specificati nel body (name, start_date, end_date, odds).
DELETE	`/event/:id`	Elimina l'evento dal DB se id è uguale ad un event_id degli eventi presenti nel DB.

BETS

GET	`/bet`	Ritorna un json contenente tutte le scommesse, se esistono.
GET	`/bet/:id`	Ritorna un json con la scommessa che ha bet_id uguale a quello specificato da id, se esiste.
GET	`/bet/user/:id`	Ritorna un json contenente tutte le scommesse appartenenti all'utente che ha user_id uguale a id, se esistono.
POST	`/bet`	Inserisce la scommessa nel DB con i vari campi specificati nel body (user_id, event_outcomes, betted_amount, potential_win).
DELETE	`/bet/:id`	Elimina la scommessa dal DB se id è uguale ad un bet_id delle scommesse presenti nel DB.

ENDED BETS

GET	`/endedbet/won`	Ritorna un json contenente tutte le scommesse vinte, se esistono.
GET	`/endedbet/lost`	Ritorna un json contenente tutte le scommesse perse, se esistono.
GET	`/endedbet/:id`	Ritorna un json con la scommessa conclusa (vinta o persa) che ha bet_id uguale a quello specificato da id, se esiste.
GET	`/endedbet/won/user/:id`	Ritorna un json contenente tutte le scommesse vinte appartenenti all'utente che ha user_id uguale a id, se esistono.
GET	`/endedbet/lost/user/:id`	Ritorna un json contenente tutte le scommesse perse appartenenti all'utente che ha user_id uguale a id, se esistono.
POST	`/endedbet`	Inserisce la scommessa conclusa nel DB con i vari campi specificati nel body (bet_id, won).
DELETE	`/endedbet/:id`	Elimina la scommessa conclusa dal DB se id è uguale ad un bet_id delle

scommesse presenti nel DB.

Qualsiasi altro path non congruo con i precedenti, non sarà matchato e quindi il response code sarà 404 NOT FOUND.

Per quanto riguarda l'utilizzo del tipo di dato json è stata utilizzata la libreria nlohmann-json.

Stats Server

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 e amministratori di sistema al fine di rendere visibili gli andamenti di alcune statistiche.

Lo Stats Server è stato sviluppato in R. Attraverso la libreria ***mongolite*** si è gestita l'interazione col DB al fine di creare le collezioni ***userstats*** e ***sistemstats*** e poter elaborare, aggiornare e collezionare i valori delle statistiche di cui si è voluto tener traccia.

Per l'utente si è scelto di tener traccia delle seguenti statistiche:

- Id utente;
- Denaro ricaricato;
- Denaro scommessi;
- Denaro vinti;
- Numero di scommesse vinte;
- Numero di scommesse perse.

Per il sistema si è scelto di tener traccia delle seguenti statistiche:

- Numero di utenti registrati;
- Numero totale di scommesse nel sistema;
- Numero di scommesse vinte;
- Numero di scommesse perse;
- Entrate;
- Uscite.

L'interazione con il Client e la GUI avviene attraverso delle REST APIs. Questo è reso possibile dalla libreria ***plumber***. Tale libreria permette, tramite annotazioni, di rendere delle funzioni di R disponili come Api Rest attraverso un endpoint.

All'interno della directory `src/Stats_Server` sono presenti due file R che rappresentano l'implementazione del modulo. Tali file sono :

- ***StatsServer.R:*** rappresenta l'elemento attivo dello StatsServer e utilizzando la libreria plumber, attiva il server eseguendolo all'indirizzo `localhost:9090` ed esponendo le API definite nel file StatsAPIs.R
- ***StatsAPIs.R:*** contiene delle funzioni che si occupano di interagire col DB al fine di calcolare, creare ed aggiornare le statistiche di sistema e degli utenti. Si occupa di graficare i valori di tali statistiche usando le funzioni di plot standard fornite da R. Tali funzioni vengono poi esposte come APIs Rest attraverso delle particolari annotazioni.

Per la gestione delle statistiche relative agli utenti sono state rese disponibili le seguenti APIs:

POST	`/userstats`	Crea il documento nel DB
GET	`/userstats`	Ritorna un json con le statistiche di tutti gli utenti nel sistema
GET	`/userstats/:id`	Ritorna le statistiche di uno specifico utente in json
GET	`/userstats/:id/piegraph`	Ritorna un file png contente un grafico a torta (Scommesse vinte vs perse)
GET	`/userstats/:id/bargraph`	Ritorna un file png contente un grafico a barre (entrate e uscite)
PUT	`/userstats/:id/paid`	Aggiorna la quantità di soldi totali ricaricati
PUT	`/userstats/:id/won`	Aggiorna il numero totale di scommesse vinte e la quantità totale di entrate
PUT	`/userstats/:id/lose`	Aggiorna il numero totale di scommesse perse
PUT	`/userstats/:id/bet`	Aggiorna il numero totale di soldi scommessi
DELETE	`/userstats/:id`	Elimina il documento nel DB quando l'utente cancella il proprio profilo

Per la gestione delle statistiche di sistema sono state rese disponili le seguenti APIs:

POST	`/sistemstats`	Crea il documento nel DB
PUT	`/sistemstats/user/register`	Incrementa il numero di utenti registrati alla registrazione di un nuovo utente
PUT	`/sistemstats/user/delete`	Decrementa il numero di utenti registrati quando un utente

		rimuove il proprio profilo
PUT	`/sistemstats/bet`	Aggiorna il numero totale di scommesse e la quantità totale di entrate
PUT	`/sistemstats/wbet`	Aggiorna il numero totale di scommesse vinte e la quantità totale di uscite
PUT	`/sistemstats/fbet`	Aggiorna il numero totale di scommesse perse
GET	`/sistemstats`	Ritorna le statistiche di sistema in json
GET	`/sistemstats/piegraph`	Ritorna un file png contente un grafico a torta (scommesse vinte vs perse)
GET	`/sistemstats/bargraph`	Ritorna un file png contente un grafico a barre(entrate e uscite)
DELETE	`/sistemstats`	Elimina il documento nel DB

Avvio

Prima di avviare l'applicazione assicurarsi che le porte 8080, 9090 e 27017 siano disponibili perché utilizzate rispettivamente da *Server*, *StatsServer* e *MongoDB*.

Eseguire i seguenti passi per avviare correttamente l'applicazione Soccer Bet:

- Avviare un'istanza di **MongoDB** sulla propria macchina;
- Compilare e lanciare il ***Rest Server*** (aggiungere alle direttive di compilazione in Visual Studio la seguente macro **_CRT_SECURE_NO_WARNINGS**);
- Lanciare lo ***StatsServer***, digitando il seguente comando da terminale all'interno della directory del progetto:
  `Rscript .\src\Stats_Server\StatsServer.R`
- Lanciare il ***Client***, digitando il seguente comando da terminale all'interno della directory del progetto `src\Client`
  `python .\SoccerBet.py`