

UNIVERSITÀ DEGLI STUDI DI CATANIA

DIPARTIMENTO DI INGEGNERIA ETETTRICA ELETTRONICA E NFORMATICA

CORSO DI LAUREA IN INFORMATICA, SECONDO LIVELLO

*Placido Papotto 1000013479*

*Restivo Luca Pesce 1000008725*

*Russo Salvatore 1000008934*

Progetto Adavanced Programming Languages

RELAZIONE PROGETTO FINALE

Ch.ma prof.ssa Vincenza Carchiolo

Ch.mo prof. Giuseppe Mangioni

Anno Accademico 2021 – 2022

Indice

[1. Introduzione 3](#_Toc89270990)

[2. Client (C#) 4](#_Toc89270991)

[1. Alcuni aspetti fondamentali 4](#_Toc89270992)

[1. Gestione utente loggato 4](#_Toc89270993)

[2. Gestione dei form come “figli” 4](#_Toc89270994)

[2. Galleria screen 5](#_Toc89270995)

[3. Backend (Go) 6](#_Toc89270996)

[1. Configurazione MongoDB 6](#_Toc89270997)

[2. Services 6](#_Toc89270998)

[3. Models 7](#_Toc89270999)

[4. Web Scraping (Python) 8](#_Toc89271000)

[1. Estrazione dei dati 8](#_Toc89271001)

[2. Salvataggio dei dati 9](#_Toc89271002)

[5. Generazione grafici (R) 10](#_Toc89271003)

# Introduzione

L’applicazione da noi sviluppata nasce con l’intendo di permettere agli utenti registrati, di poter gestire, tramite una applicazione Desktop, la lista dei loro film preferiti con la possibilità di assegnare ad ognuno di questi un voto personale che va da 0 a 5; il voto personale permette quindi di esprimere la preferenza per un film.

La nostra applicazione per poter essere utilizzata prevede una fase di registrazione e solo dopo questa sarà possibile sfruttare i servizi.

Il software si compone di quattro diverse componenti:

* Il client (FE), scritto interamente in **C#** e si basa su una gestione degli eventi implementata per mezzo di Windows Forms, cioè una user interface basata su .NET Framework.
* Il BE del nostro progetto è stato realizzato interamente in **Go** ed espone delle API REST grazie all'utilizzo del framwork "Gin Gonic Web Framework"
* Un modulo per la generazione di dati statistici (due grafici con esattezza) scritto in **R**
* Un modulo per il Web Scraping, scritto in **Python** che legge le informazioni sulle pagine del sito IMDb e le salva sia sul DB che su un CSV letto dal modulo di statistica

Lo strato di persistenza si appoggia su un DB di tipo non relazionale, ovvero MongoDB, quindi parliamo di un DB document oriented.

Il progetto è disponibile su GitHub:

<https://github.com/Papotto97/APL>

# Client (C#)

Lo sviluppo del client si basa interamente sulla gestione di eventi, infatti ad ogni azione dell’utente corrisponderà un evento che grazie agli handler, presenti nei vari forms, sarà gestito e permetterà di eseguire un’azione/operazione specifica.

Il client si compone di diversi form, di cui 2 principali e 3 chiamati “figli”:

* Welcome (utilizzato per il login e per la registrazione) [Figura 1]
* Dashboard (richiama altri forms "figli")

Gli altri form invece, quelli richiamati come "figli" dal form Dashboard sono:

* DashboardPlotForm (utilizzato per mostrare i grafici ottenuti chiamando l'API esposta dal modulo in R)
* SearchFilmsForm (utilizzato per effettuare ricerche di film mediante una chiamata alle API esposte da IMDB)
* YourFavouritesMovies (utilizzato per mostrare i film che sono stati aggiunti ai preferiti dall'utente loggato; i film sono recuperati mediante una chiamata HTTP al BE)

## Alcuni aspetti fondamentali

### Gestione utente loggato

Per la gestione dell’utente loggato, dal momento che nelle applicazioni Desktop non è previsto il concetto di sessione, abbiamo pensato di utilizzare una classe statica “UserInfo” che sarà popolata al momento del login e permetterà di mantenere le informazioni sull’utente loggato. Si tratta chiaramente di una soluzione poco efficiente e poco sicura, visto che in qualsiasi punto del codice è possibile modificare l’istanza della classe, ma per il nostro scopo questa soluzione è sembrata ottima.

### Gestione dei form come “figli”

Per fare in modo che i form “figli” vengano aperti all’interno di un pannello presente nel form principale Dashboard, è stato realizzato un metodo privato, come membro della classe Dashboard, ovvero **private void OpenPanelFormCentral(object child)**:

Immagine che contiene testo

Descrizione generata automaticamenteIl metodo in questione controlla prima di tutto che il pannello centrale non abbia alcun componente attivo al suo interno (sfruttando il fatto che la classe Form estende la classe Control, così come la maggior parte dei componenti di Windows Forms); nel caso in cui siano presenti dei componenti richiama per ognuno di essi il metodo Dispose() che consente di liberare le risorse occupate, mentre se non è presente alcun componente all’interno del pannello, viene aggiunto il componente “child” passato come parametro in ingresso.

Il metodo appena visto viene richiamato per esempio dal metodo **searchFilmsButton\_Click** che è sostanzialmente un handler che gestisce l’evento click sul pulsante “**searchFilmsButton**” presente nella dashboard. Nel corpo di quest’ultimo metodo viene quindi istanziato un componente di tipo SearchFilmsForm utilizzando il suo costruttore che prevede in ingresso un oggetto di tipo “Form” in qui, mediante la keyword *this* viene passato il form Dashboard come padre.

Sfruttando questo meccanismo, grazie al metodo **public void SetResultPanel(Control control)** presente nella classe Dashboard, è possibile andare ad aggiungere nel pannello dei risultati, componenti (oggetti) la cui classe è una estensione del tipo “**Control**” (nel nostro caso viene quasi sempre generato uno componente di tipo “**DataGridView**” e viene quindi passato in ingresso al metodo in questione).

## Galleria screen

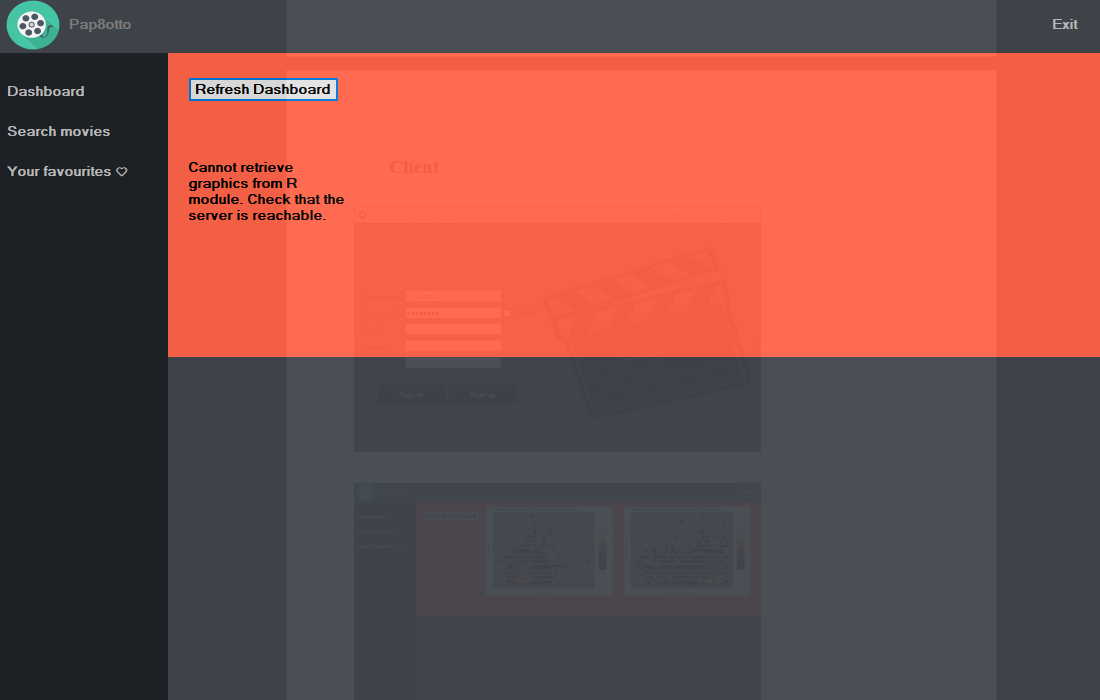
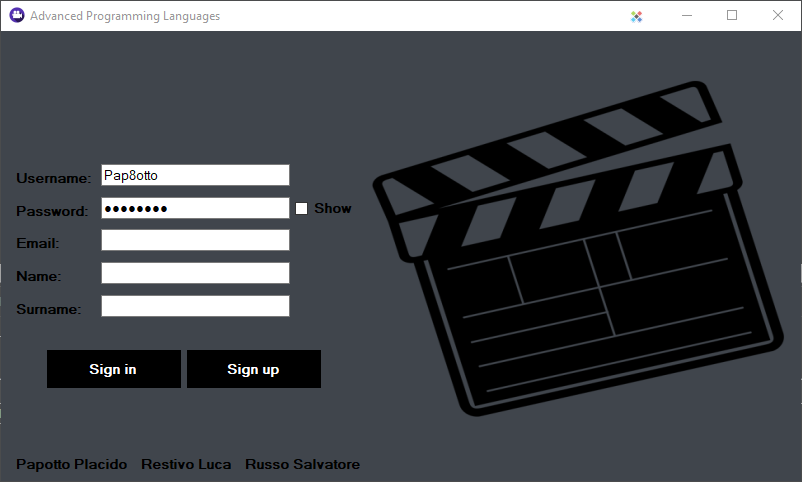


Immagine che contiene testo

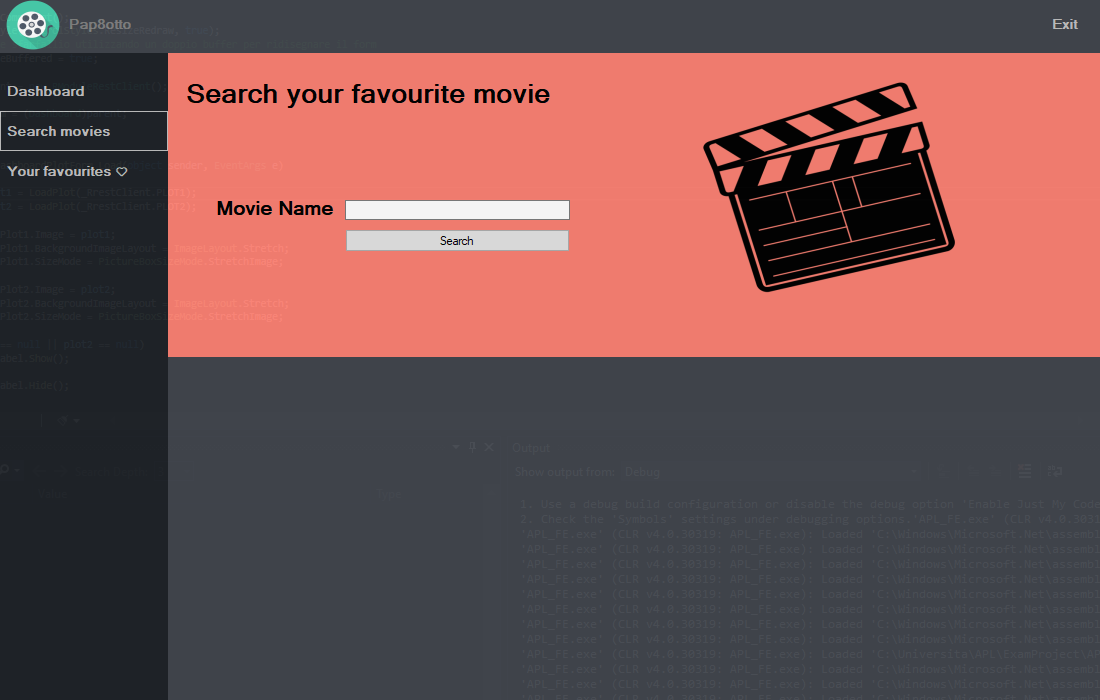
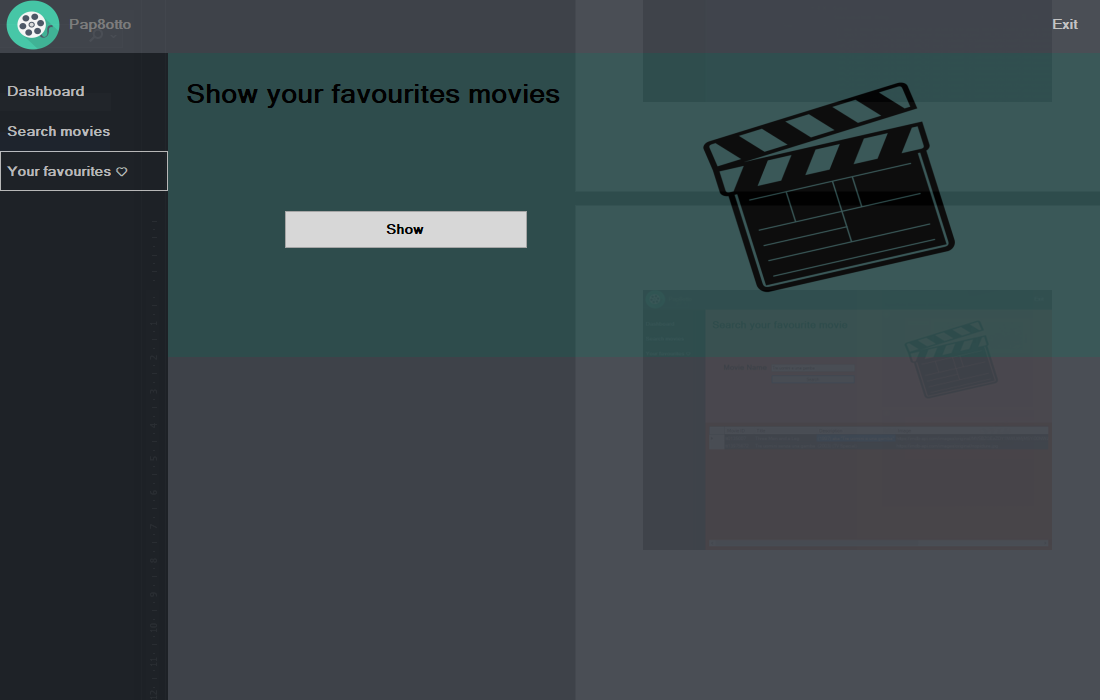
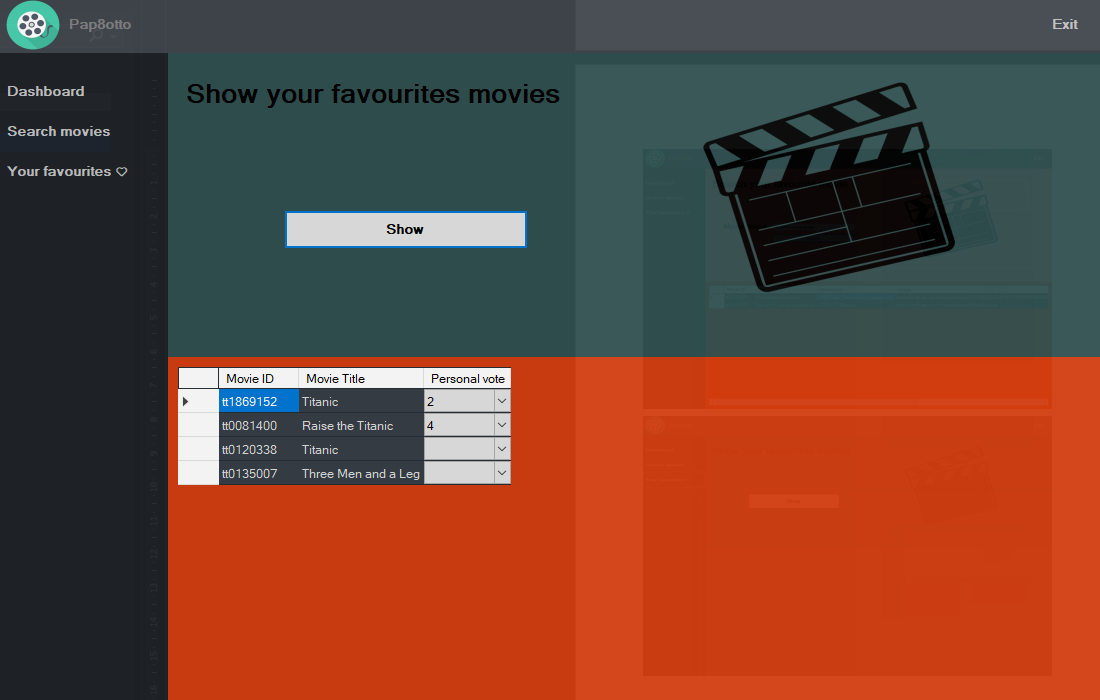
Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente



# Backend (Go)

Il BE del nostro progetto è stato realizzato interamente in Go ed espone delle API REST grazie all'utilizzo del framwork "Gin Gonic Web Framework" che consente di creare un web server in grado di esporre le API utilizzando delle route specifiche. Il FE quindi non effettuerà delle chiamate dirette verso MongoDB, ma grazie alle API esposte dal BE sarà in grado di effettuare le CRUD sulle varie entità gestite dall'applicazione. Si è cercato quindi di rispettare le caratteristiche dei metodi HTTP:

* **GET**: utilizzato semplicemente per recuperare dati dal DB
* **POST**: effettuare update di entità (usata per assegnare un voto ad uno dei film tra i preferiti)
* **PUT**: inserire nuove entità (usata per la registrazione di nuovi utenti o per l'aggiunta di un film tra i preferiti)

Immagine che contiene testo

Descrizione generata automaticamenteCome possiamo vedere, la funzione main, presente nel file *main.go* (che rappresenta l’entry point del BE), al suo interno non fa altro che effettuare una operazione di dichiarazione ed assegnazione della variabile r mediante il metodo **gin.Default()** che ritorna un oggetto di tipo **\*gin.Engine** ovvero, un puntatore all’oggetto **gin.Engine** che rappresenta il motore del framework, ovvero quell’oggetto al quale è possibile registrare le route ed i loro rispettivi metodi per la gestione.

Nel nostro caso, infatti, come possiamo vedere, abbiamo registrato diversi path (diverse route) e ad ognuno di essi abbiamo assegnato un handler per la gestione della richiesta HTTP. Sono stati definiti metodi GET, POST e PUT.

## Configurazione MongoDB

Il file sorgente di configurazione *mongo.go* contiente una funzione pubblica **GetConnection()** che ritorna un client mongo, un contesto ed una **CancelFunc** (una funzione che consente di comunicare ad una operazione di abbandonare il proprio lavoro attuale).

La funzione viene quindi richiamata in tutti i service per configurare la connessione al DB.

## Services

La logica delle varie API esposte dal BE non viene svolta direttamente nel file *main.go*, ma per avere una corretta suddivisione delle responsabilità sono stati creati dei services, cioè dei file in cui sono presenti delle funzioni pubbliche che consentono di gestire le entità (nel nostro caso Collezioni mongo) presenti sul DB.

Immagine che contiene testo

Descrizione generata automaticamenteSono stati creati infatti 4 diversi tipi di services, ognuno dei quali gestisce una specifica collezione mongo (i modelli sono definiti all’interno della directory “models/”).

In ognuno dei service sono quindi presenti le funzioni pubbliche che vengono richiamate dagli handler definiti nel *main.go*; vediamo come esempio la funzione che gestisce l’inserimento su MongoDb di un nuovo user, richiamata dall’handler che gestisce la richiesta PUT sulla route “*/user/*”.  
Come vediamo da questa schermata, la funzione prende in ingresso un puntatore ad un oggetto di tipo “User” e ritorna un oggetto di tipo ResponseDTO, utile per gestire lo status HTTP nella response.

## Models

Per creare classi (come ad esempio le classi C#) e quindi lavorare ad oggetti, in Go è necessario usare la sintassi seguente:

Immagine che contiene testo

Descrizione generata automaticamenteIn questo modo abbiamo definito, all’interno della directory “*models/*” diversi oggetti/modelli utilizzati in tutta l’applicazione.

# Web Scraping (Python)

Il modulo per effettuare lo scraping sul sito IMDB è realizzato interamente in Python e sfrutta la libreria esterna BeautifulSoup che offre svariati metodi per recuperare informazioni dai DOM ricavati mediante richieste HTTP. Infatti come è possibile vedere dal file sorgente "scraper.py", una volta istanziato un ogetto di tipo BeautifulSoup passando in ingresso la risposta HTTP (che sarebbe una pagina HTML) e il tipo di parser da adottare, mediante questo oggetto possiamo ricavare tutte le informazioni che servono. Finita la fase di scraping, i dati vengono poi conservati sia in un CSV che su una collezione di MongoDB; il CSV sarà successivamente analizzato dal modulo R per generare i grafici che saranno forniti al FE.

Per lanciare il modulo di Web Scraping è necessario prima di tutto scaricare tutte le dipendenze utilizzate dagli script in python; le dipendenze sono specificate nel file *requirements.txt* e tramite il comando

`pip3 install -r requirements.txt`

è possibile scaricare tutte le dipendenze necessarie al modulo.

Una volta che tutte le librerie esterne sono state scaricante mediante il gestore dei pacchetti di python, pip3, è possibile lanciare lo scraper con *python main.py*.

Lo script stamperà nella console diverse righe che danno informazioni sulle richieste HTTP che esso effettua per recuperare le pagine HTML dal portale IMDB.

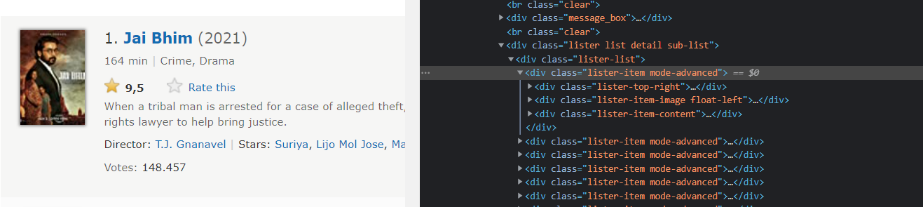
## Estrazione dei dati

Il cuore dello scraping si ha nel file *scraper.py* che dopo aver effettuato una richiesta HTTP, verifica innanzitutto che la risposta sia ok, ovvero lo stato della risposta sia 200 e successivamente inizia la vera e propria estrazione dei dati.

Immagine che contiene testo

Descrizione generata automaticamente

Grazie ad un oggetto di tipo **BeautifulSoup** (fornito dalla libreria *bs4*) istanziato mediante il suo costruttore che vuole in ingresso una risorsa su cui effettuare il parsing (in questo caso la risposta HTTP ottenuta precedentemente, ovvero la pagina HTML <https://www.imdb.com/search/title/?groups=top_1000&sort=user_rating,desc&start=1&ref_=adv_nxt> ) ed il tipo di parser da usare.

Successivamente, ispezionando la pagina del portale tramite browser abbiamo avuto modo di vedere che le informazioni che ci interessano sono tutte contenute in un div di classi “lister-item mode-advanced”

e quindi in ogni div con questa classe siamo sicuri di trovare tutto ciò che ci interessa.

Scavando quindi dentro questo elemento vengono recuperate le informazioni utili per creare oggetti di tipo “**Movie**”, cioè:

* Immagine che contiene testo

  Descrizione generata automaticamenteimdb\_id
* title
* rating
* date
* summary
* runtime
* genres

La pagina su cui viene effettuato lo scraping delle informazioni, contiene la lista dei Top 1000 film presenti su IMDb ordinati in base alla votazione degli utenti del portale.

## Salvataggio dei dati

Una volta finito lo scraping dei dati, questi verranno ritornati (come una lista di oggetti di tipo Movie) al metodo che chiama la funzione **movie\_scraper()**, nel nostro caso il main, e sarà il main stesso a chiamare prima la funzione di salvataggio dei dati su MongoDB e successivamente chiamerà la funzione **movie\_printer()** definita all’interno di *printer.py* che consente di salvare un file CSV che sarà successivamente letto da R per la generazione dei grafici.

Immagine che contiene testo

Descrizione generata automaticamente

# Generazione grafici (R)

Grazie ad uno script realizzato in R e all'utilizzo della libreria "Plumber" ricavata da CRAN ( <https://cran.r-project.org/web/packages/plumber/index.html> ) siamo riusciti a realizzare due diversi grafici:

* Relationship between Movie Runtime and IMDb Movie Rating
* Relationship between Movie Release Year and IMDb Movie Rating

I grafici in questione, una volta generati e salvati nel server in cui viene lanciata l'applicazione in R, vengono recuperati dal FE mediante delle API REST che vengono esposte grazie all'utilizzo della libreria "***Plumber***" citata sopra; grazie a questo meccanismo è infatti possibile sfruttare il protocollo HTTP in maniera rapida e veloce per far comunicare i vari moduli.

Le librerie usate dal modulo sono:

Immagine che contiene testo

Descrizione generata automaticamente

Per esporre delle API con plumber è necessario utilizzare dei metadati sulla funzione (come delle annotazioni Java) che indicano a plumber come gestire quella particolare funzione. Nel nostro caso sono state quindi create due funzioni differenti, con due endpoint differenti, che permettono di generare rispettivamente i due grafici elencati sopra.

Immagine che contiene testo

Descrizione generata automaticamente