

# Progetto PWM

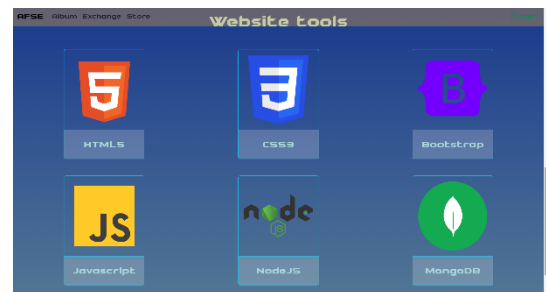
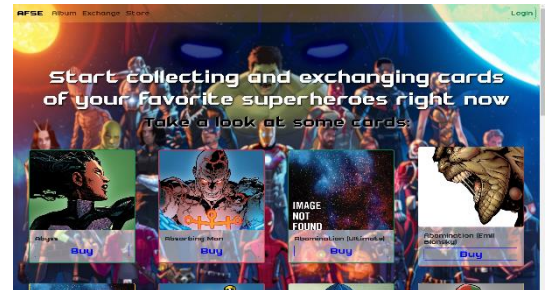
Giulio Cacciapuoti (28970A)

## Front End

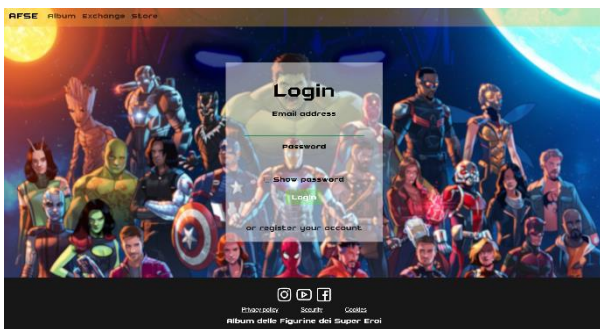
La prima schermata visualizzabile è quella di benvenuto (AFSE.html), presenta una navbar con fixed position (così che scorra man mano che l'utente scrolla la pagina) e mostra alcune delle figurine disponibili sul sito, ciascuna figurina è modellata automaticamente dalla funzione *mostraFigurine*.

Scorrendo vengono mostrati i tools usati per la realizzazione del sito, questi non vengono però modellati da una funzione ma sono già pronti all'avvio del sito.

Infine, troviamo un footer con tutti gli eventuali collegamenti secondari del sito.



Dalla navbar in alto è possibile raggiungere varie sezioni del sito ma molte di queste controlleranno che l'utente sia loggato prima di essere pienamente utilizzabili.



Cliccando sul tasto login si apre una pagina dove inserire le informazioni del proprio account (login.html), queste verranno inviate al server grazie alla funzione *loginUser* che in caso di successo risponderà con un codice http 200 e un JSON contenente tutte le informazioni sull'utente (ad eccezione della password per garantire

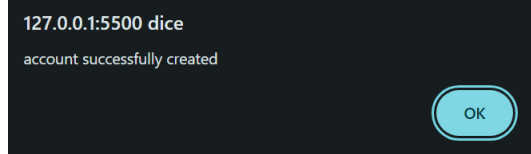
maggiore sicurezza) mentre risponderà con un codice http 400 e un messaggio di errore in caso di credenziali errate.

Nel caso in cui invece ci si voglia registrare è presente nella stessa pagina il bottone registrati, una volta cliccato si aprirà la schermata per inserire tutte le informazioni necessarie (register.html) per poi inviarle al server con la funzione *registerUser*, in caso di creazione corretta il server risponderà con un 201 e si aprirà la pagina di login, altrimenti verrà restituito un codice 400 e un messaggio d'errore, ciò accade se si sceglie una password troppo corta, un username troppo corto o un email già utilizzata.

A screenshot of a web form titled "Register". It has a background image of Spider-Man. The form contains fields for "Username" (filled with "giulio"), "Email address" (filled with "giulio@unimi.it"), and "Password" (filled with "\*\*\*\*\*"). There are also checkboxes for "Show password" and "Favorite superhero" (filled with "spiderman"), and a green "Register" button at the bottom.

Tutte le informazioni sugli utenti saranno poi salvate sul db hashando la password.

Ogni volta che si riceve un aggiornamento dal server l'utente viene allertato mediante un *alert*.

A dark-themed alert dialog box with a title bar. The text inside says "127.0.0.1:5500 dice" and "account successfully created". There is an "OK" button in the bottom right corner.

Una volta eseguito l'accesso il

server restituisce

un JSON con tutti i dati necessari che verrà conservato nel *localStorage* col nome di *user* così da averlo sempre a disposizione nel corso della navigazione.

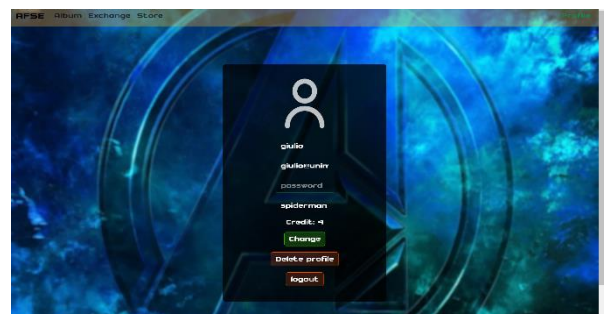
```
▼ {sessionId: "6661024b854784bf527501ed", username: "giulio", email: "giulio@unimi.it", ...}
  cards: []
  credit: 0
  email: "giulio@unimi.it"
  favhero: "spiderman"
  sessionId: "6661024b854784bf527501ed"
  username: "giulio"
```

Il sessionId corrisponde all'objectId personale di

ogni utente conservato sul db e permette di identificarlo singolarmente.

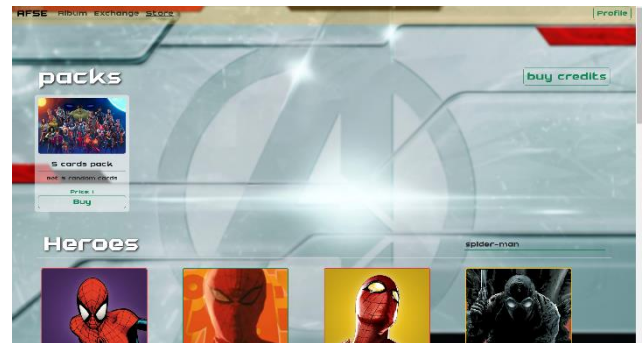
Dopo essersi loggati si verrà reindirizzati alla pagina di benvenuto (AFSE.html) con la possibilità di accedere pienamente a tutte le sezioni del sito.

Il pulsante login d'ora in avanti sarà sostituito da un pulsante profile (grazie alla funzione *checkIfLogged*) che ci permetterà di accedere a *userprofile.html*, per accedere a questa pagina è sempre necessario un id, nel caso in cui l'id passato coincida con l'id presente nel *localStorage* (sessionId) allora l'utente potrà modificare le proprie credenziali (funzione *change*), cancellare il proprio profilo (funzione *delUser*), ed effettuare il logout (funzione *logout*), altrimenti saranno solo mostrare informazioni generali sull'utente selezionato, queste funzioni sfruttano l'id salvato nel *localStorage* per funzionare (*logout* cancella il parametro *user* del *localStorage*)

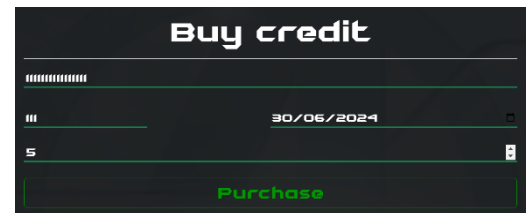
A screenshot of a user profile page. It has a dark blue background with a circular pattern. In the center, there's a white circle with a person icon. Below it, the text "giulio" is displayed. Underneath, there's a list of fields: "giulio@unimi.it", "password", "spiderman", and "Credit: 4". There are buttons for "Change", "Delete profile", and "logout".

Accendendo allo store (store.html) sarà possibile acquistare pacchetti, singole figurine o crediti

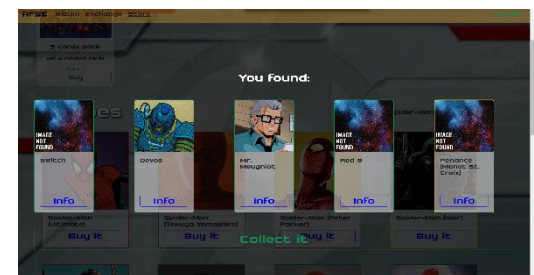
I pacchetti costano 1 credito e contengono 5 figurine, se invece si vuole scegliere una specifica figurina il prezzo varierà da 1 a 3 crediti in base alla rarità della figurina, quest'ultima è calcolata basandosi sulla quantità di comics in cui l'eroe compare.



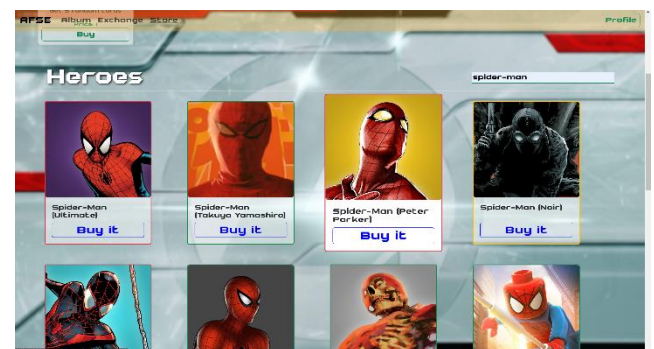
Cliccando su buy credits si aprirà un modulo per inserire le informazioni relative al pagamento con la funzione *compraCrediti* e verrà aggiornato il profilo dell'utente col nuovo credito (funzione *transazione*) una volta terminato di inserire le informazioni.



Acquistando un pacchetto verranno mostrate le figurine trovate (funzione *compraPacchetto*) e una volta cliccato il pulsante Collect it si verrà reindirizzati all'album (album.html) dove si visualizzeranno tutte le figurine possedute.

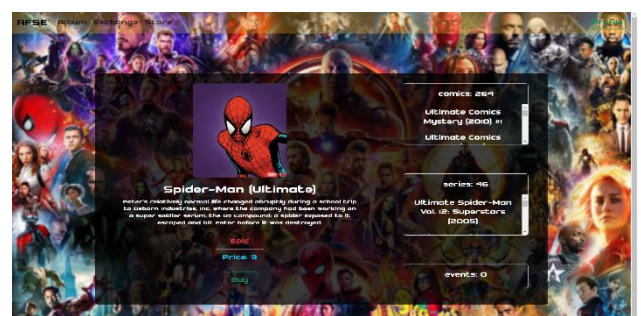


Per cercare una figurina attraverso il nome dell'eroe ad essa associato è presente una barra di ricerca che aggiorna i risultati ad ogni cambiamento rilevato attraverso la funzione *cercaFigurineDaCedere* (utilizzata anche nella pagina exchange), questa ad ogni cambiamento nella barra di ricerca esegue una specifica API call e mostra i risultati, per modellare le figurine viene riutilizzata la funzione *mostraFigurine*.



Ogni volta che viene mostrata una figurina non in dettaglio sono sempre mostrate alcune informazioni base come immagine, nome e rarità, quest'ultima si stabilisce dal colore dei bordi (verde=comune gialla=rara rosso=molto rara)

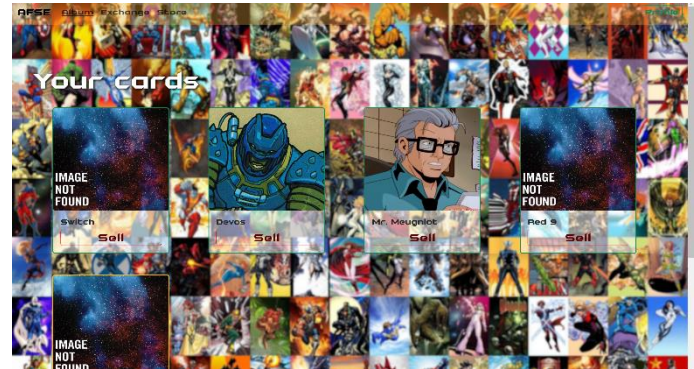
Cliccando su ogni figurina si aprirà una pagina con informazioni più dettagliate sull'eroe (heroprofile.html) da cui sarà possibile acquistarla, per identificare ogni eroe il sito associa ad ogni figurina un id che poi passa alla pagina heroprofile così da recuperare solo le informazioni sull'eroe scelto (con una API call).



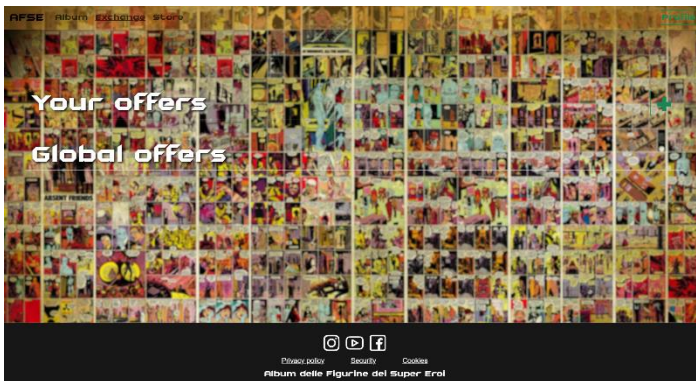
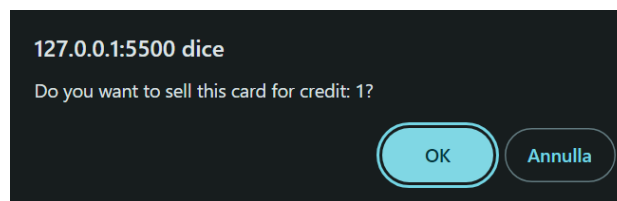


Cliccando sul tasto buy la figurina verrà acquistata e inserita nella collezione dell'utente

Nell'album (album.html) saranno caricate e mostrate tutte le figurine possedute dall'utente, ogni volta che si carica questa pagina viene aggiornato il localStorage in quanto viene fatta una nuova richiesta al server che legge dal db (funzione *mostraFigurineUtente*), questo viene fatto così da poter aggiornare il localStorage senza doversi necessariamente riloggar ogni volta che un utente conclude uno scambio di figurine.

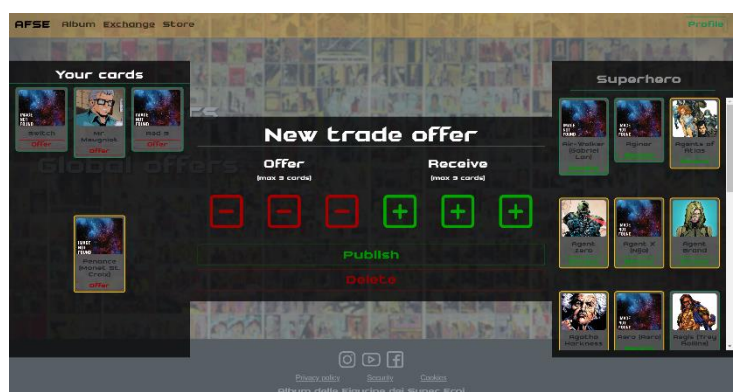


Sotto ogni figurina è presente il tasto Sell che permette di venderla in cambio di crediti (funzione *vendiFigurina*), anche in questo caso a ogni figurina è associato il relativo id e ciò permette di vendere quella corretta, cliccando sulla figurina poi si potrà di nuovo accedere alla sezione heroprofile per vedere tutte le informazioni sull'eroe

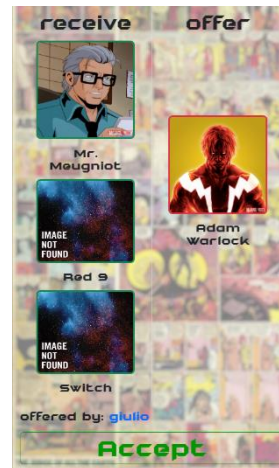


Nella pagina exchange.html è possibile pubblicare, cancellare e accettare richieste di scambio tra gli utenti, la pagina caricherà innanzitutto gli scambi disponibili (funzione *showTrades*) richiedendo al server gli scambi pubblicati e poi darà la possibilità di crearne di nuovi (mostrando il bottone apposta a destra di "Your offers")

Cliccando sul pulsante si aprirà una sezione (funzione *apriMenu*) dove configurare una nuova proposta di scambio, cliccando sui relativi pulsanti si apriranno le sezioni per scegliere le carte da ricevere (funzione *toggleModuloRicevi*) e quelle da offrire (funzione *toggleModuloOffri*), si possono scegliere più figurine da offrire



o ricevere creando quindi scambi più complessi (massimo 3 per tenere gli scambi equi), cliccando sul tasto delete si andrà a cancellare lo scambio creato (funzione *chiudiMenu*) e cliccando sul tasto publish si pubblicherà l'offerta (funzione *pubblicaOfferta*), prima di pubblicarla però il sito verifica che non si stiano ricevendo o offrendo figurine uguali e che non si stiano ricevendo figurine già presenti nella collezione, se questo non accade lo scambio viene salvato sul server e, finché l'utente resta loggato, viene visualizzato come suo (con possibilità di rimuoverla), altrimenti verrà visualizzata come offerta globale (con possibilità di accettarla solo in caso si sia loggati con un altro account e si posseggano le figurine necessarie a completare lo scambio), sotto la scritta "receive" sono mostrate le figurine che l'utente riceverà accettando lo scambio, sotto "offer" invece ci sono quelle che l'utente dovrà offrire per completare quello scambio.



Ogni volta che si crea uno scambio le figurine offerte vengono rimosse dall'account dell'utente per evitare che si possano creare offerte infinite e quindi guadagnare più figurine di quante dovute con 1 sola figurina, se l'utente cancella l'offerta però queste gli vengono restituite (funzione *removeOffer*).

Se un utente invece accetta un'offerta pubblicata da un altro utente (funzione *acceptTrade*) questa viene cancellata dal server e ad entrambi gli utenti vengono distribuite le figurine dovute aggiornando le loro informazioni sul database.

Per ogni offerta è possibile cliccare sulle figurine o sul nome dell'utente che la ha pubblicata prima di decidere se accettarla o rimuoverla così da avere più informazioni.

Nel caso in cui un utente cancelli il suo profilo con delle proposte di scambio ancora attive il server si occuperà di cancellare tutte le offerte pubblicate da quell'utente, per farlo in ogni scambio viene salvato anche l'id dell'utente che lo ha creato, così da poter eliminare quelli corretti

## Back End

Il backend è gestito dal server che si occupa di ricevere le richieste dagli utenti e scrivere/leggere sul database quando necessario, alla pagina localhost:5500/api-docs è possibile trovare lo swagger con tutte le possibili richieste che si possono fare al server, l'unica variazione è per i trades, questi ultimi infatti vengono salvati in un json (trades.json) e quando vengono richiesti vengono letti direttamente dal json, in questo modo si cerca di diminuire la latenza tra la richiesta al server node e la richiesta a mongodb eliminando

quest'ultima, è comunque presente la pagina /saveTrades per sincronizzare le proposte salvate nel json con quelle sul db così da mantenere anche il db sempre aggiornato, quest'ultima viene chiamata ogni volta che si aggiunge o rimuove una nuova proposta (o quando si rimuove un account).

Nella funzione *acceptTrade* inoltre vengono utilizzati dei timeout di 1 secondo per non mandare troppe richieste tutte insieme al db, questo infatti potrebbe provocare un crash del db se fatto da molti utenti contemporaneamente.

## Files

Nella cartella front end sono immagazzinati tutti i file relativi alla struttura (html) e allo stile (css) ed è presente lib.js che contiene le funzioni più utilizzate dal sito o quelle più complesse, le funzioni più semplici o quelle relative solo a una pagina invece si trovano in degli script all'interno dei file html.

Nella cartella images sono presenti le immagini utilizzate dal sito eccetto quelle ricavate dalle API call.

Nella cartella Server sono presenti i file index.js per il funzionamento del server e swagger.js per la definizione dello swagger oltre a tutti i file necessari per il corretto funzionamento di node e npm.

Nella cartella è anche presente un font esterno (Bruce Forever.ttf) utilizzato per lo stile del sito.

All'esterno della cartella codice infine è presente il file trades.json (dove vengono salvati i trades dal server oltre che nel db), questo deve essere posizionato fuori dalla cartella che si andrà a hostare in localhost in quanto ogni sua modifica comporta il reload della pagina (problema riscontrato con alcune estensioni per il localhost su visual studio code) e ciò provocherebbe malfunzionamenti nel sito, dunque è consigliabile non metterla nella stessa cartella