

Relazione Progettazione Web e Mobile 2025

Progetto: Album delle Figurine dei Supereroi

CORSO: Progettazione Web e Mobile

Autrice: Veronica Falgiani (Matricola: 21191A)

Indice

- [Introduzione](#)
- [Implementazione](#)
 - [Front-end](#)
 - [HTML](#)
 - [CSS](#)
 - [JavaScript](#)
 - [Back-end](#)
 - [NodeJS + Express](#)
 - [MongoDB](#)
 - [Moduli aggiuntivi](#)
 - [.env](#)
 - [uuid \(Session Tokens\)](#)
 - [Swagger](#)
- [Funzionamento](#)
 - [Homepage](#)
 - [Registrazione](#)
 - [Login](#)
 - [Modifica profilo](#)
 - [Acquisto crediti](#)
 - [Acquisto pacchetti](#)
 - [Album](#)
 - [Visualizzazione carte](#)
 - [Vendita carte](#)
 - [Scambi](#)
 - [Visualizzazione scambi](#)
 - [Creazione scambio](#)
 - [Eliminazione scambio](#)
 - [Accetta scambio](#)
- [Note extra](#)

Introduzione

Questo progetto è stato svolto per il corso di Programmazione Web e Mobile.

La seguente relazione descrive il procedimento per la creazione del sito web "Album delle Figurine dei Supereroi" e il suo funzionamento in modo dettagliato.

Il progetto è stato scritto interamente in lingua inglese come esercitazione personale e per seguire lo "standard di programmazione" che utilizza principalmente l'inglese.

Implementazione

Per questo progetto ho usato diverse tecnologie:

- **Front-end**
 - HTML
 - CSS
 - JavaScript
- **Back-end**
 - NodeJS + Express
 - MongoDB

Per semplicità ho utilizzato lo standard **JSON** per gestire le richieste e le risposte tra client e server.

FRONT-END

HTML

I file HTML sono contenuti all'interno della cartella `/src/html`:

- `album.html`: mostra tutte le carte collezionabili suddivise in pagine da 50 carte. E' possibile visionare solo le carte collezionate dall'utente tramite un bottone.
- `card.html`: mostra tutte le informazioni di una carta. Se non è stata ottenuta mostra solo la descrizione, altrimenti anche fumetti, serie e collezioni.
- `credits.html`: permette all'utente di comprare crediti tramite vari metodi di pagamento (ho omesso i pagamenti per ragioni pratiche e di sicurezza).
- `login.html`: mostra all'utente una pagina per il log in in cui deve inserire username e password per poter accedere al sito.
- `packs.html`: mostra i crediti dell'utente e permette di acquistare pacchetti tramite 3 bottoni.
- `profile.html`: mostra il profilo con tutte le informazioni dell'utente. Ha due bottoni che permettono la modifica e l'eliminazione del profilo.
- `register.html`: mostra la pagina di registrazione per l'utente con i campi username, email, password, eroe preferito e serie preferita.
- `trade.html`: mostra tutte le informazioni per uno scambio, ovvero titolo, username di chi ha creato lo scambio e le carte richieste e inviate.
- `trades.html`: mostra tutti gli scambi possibili meno quelli dell'utente suddivisi in pagine da 20 scambi. In questa schermata è possibile cliccare uno scambio per avere più informazioni. Premendo un bottone invece verranno mostrati solo gli scambi dell'utente. Qui è possibile eliminare scambi presenti o creare nuovi tramite un bottone.

Molte pagine possono sembrare spoglie, ma vengono popolate tramite scripts solo quando sono caricate. Questo permette di avere dati sempre aggiornati e non dover creare tanti file HTML per rappresentare uno stesso gruppo di informazioni, ma che hanno leggere differenze.

All'interno di ogni pagina è stato inserito un `<div id="alert"></div>` che si popola ogni volta che vengono ricevute delle risposte dal server. In questo modo l'utente è sempre aggiornato su quello che sta

succedono all'interno del sito.

Stesso principio si applica al menù in cima alla pagina che è contenuto in `<div id="menu"></div>`.

CSS

Il file CSS è all'interno di `/scr/css` e descrive principalmente:

- L'immagine di sfondo da usare per le pagine
- La dimensione di alcuni contenitori
- Il font e i colori da utilizzare per titoli e testo
- La visualizzazione corretta delle carte

Non ho modificato molto il file CSS perchè all'interno di questo progetto mi sono aiutata con **Bootstrap** per creare stili e animazioni.

Questo toolkit permette di avere stili già preimpostati che rendono il sito web molto piacevole alla vista senza troppo sforzo.

JAVASCRIPT

I file javascript sono all'interno di `/src/html/scripts` e sono:

- `album.js`: contiene tutti gli script per richiedere alle API AFSE tutte le carte e per popolare le pagine. Inoltre ci sono funzioni che permettono di visualizzare 50 carte per pagina e anche di spostarsi tra una pagina e l'altra.
- `card.js`: popola la pagina con informazioni più o meno dettagliate in base a quali carte abbiamo collezionato. Per ricevere le informazioni utilizziamo le API Marvel. Inoltre le carte ottenute possono essere vendute per 0.1 credito richiamando le API AFSE.
- `credits.js`: aggiorna la pagina per mostrare i crediti correnti di un utente e permette di incrementarne il valore tramite API AFSE.
- `login.js`: contiene la funzione per fare il login sul server tramite API AFSE.
- `menu.js`: contiene lo script per popolare il menù di navigazione all'interno delle pagine modificando `<div id="menu"></div>`. Contiene anche lo script per il log out dell'utente e per mostrare alert sulla pagina modificando `<div id="alert"></div>` tramite l'utilizzo di API AFSE.
- `packs.js`: contiene funzioni per aprire pacchetti di carte che richiamano l'API Marvel per prendere eroi random e le API AFSE per inserire le carte nell'utente. Mostra poi a schermo le carte ottenute dall'utente.
- `profile.js`: ripopola la pagina del profilo ad ogni aggiornamento e poi contiene funzioni per modificare username e password di utente. Questo avviene tramite chiamata ad API AFSE.
- `register.js`: recupera tutte le informazioni inserite dall'utente e le invia alle API AFSE per registrare l'utente.
- `trade.js`: popola lo scambio interessato mandando una richiesta alle API AFSE.
- `trades.js`: richiede all'API AFSE le informazioni per tutti gli scambi nel database. Permette di visualizzare gli scambi creati dagli altri o i nostri scambi. Possiamo cliccare su uno scambio per visualizzare le informazioni, oppure creare o eliminare uno scambio sempre mandando richieste ad API AFSE

NODEJS + EXPRESS

Le funzionalità di NodeJS+Express sono contenute nei seguenti file:

- `app.js`: è l'entrypoint dell'applicazione. Contiene le funzioni per avviare il server e definire gli endpoint.
- `src/lib/cards.js`: contiene le funzioni principali per gli endpoint che lavorano sulle carte.
- `src/lib/credits.js`: contiene le funzioni per gli endpoint che lavorano sui crediti.
- `src/lib/marvel.js`: contiene la funzione che permette di dialogare con le API Marvel server-side.
- `src/lib/session.js`: contiene le funzioni per gestire i token di sessione degli utenti (salvati come cookie) e le funzioni per gli endpoint di login, register, logout.
- `src/lib/trades.js`: contiene le funzioni per gli endpoint che lavorano sugli scambi.
- `src/lib/user.js`: contiene le funzioni per gli endpoint che lavorano sugli utenti.

MONGODB

Per poter utilizzare il sito in modo efficiente ho creato un database con MongoDB per immagazzinare i dati di utenti e scambi.

Il database si chiama AFSE e contiene al suo interno 3 collezioni:

- Users
- Cards
- Trades

Users: contiene tutte le informazioni principali dell'utente e le informazioni delle carte che possiede.

Esempio:

```
{
  "_id": {
    "$oid": "67a0b28fcaef7c74fa397d19"
  },
  "username": "veronica",
  "email": "veronica@mail.com",
  "password": "faa82036538b8f367fcf7bfd4c63b789",
  "hero": "1011006",
  "series": "261",
  "credits": 34.6,
  "cards": [
    {
      "id": 1010352,
      "name": "Madame Masque",
      "thumbnail":
"http://i.annihil.us/u/prod/marvel/i/mg/5/00/4ce5a251c1100.jpg",
      "number": 2,
      "inTrade": true
    },
    {
      "id": 1009438,
      "name": "Medusa",
      "thumbnail": "
```

```

    "http://i.annihil.us/u/prod/marvel/i/mg/c/30/537bb549bebd0.jpg",
        "number": 1,
        "inTrade": false
    },
    {
        "id": 1010915,
        "name": "Captain Britain (Ultimate)",
        "thumbnail":
    "http://i.annihil.us/u/prod/marvel/i/mg/6/a0/4c003574e99fb.jpg",
        "number": 1,
        "inTrade": false
    }
}

```

Campi:

- `_id`: id assegnato da MongoDB
- `username`: username dell'utente
- `email`: email dell'utente
- `password`: password dell'utente salvato in md5
- `hero`: id eroe preferito
- `series`: id serie preferita
- `credits`: crediti dell'utente
- `cards`: informazioni sulle carte
 - `id`: id dell'eroe
 - `name`: nome dell'eroe
 - `thumbnail`: immagine dell'eroe
 - `number`: copie possedute
 - `inTrade`: determina se la carta è impegnata in uno scambio o no

Cards: contiene le informazioni principali su tutti gli eroi marvel. Questa collezione è stata creata facendo fetch ad uno ad uno di tutti gli eroi sulla Marvel API.

Esempio:

```

{
    "_id": {
        "$oid": "67a1e057c80912e46a012eff"
    },
    "id": 1011334,
    "name": "3-D Man",
    "thumbnail": "http://i.annihil.us/u/prod/marvel/i/mg/c/e0/535fecbbb9784.jpg"
}

```

Campi:

- `_id`: id assegnato da MongoDB
- `id`: id dell'eroe

- **name**: nome dell'eroe
- **thumbnail**: immagine dell'eroe

Trades: contiene tutti gli scambi creati dagli utenti del sito. All'interno si trovano le informazioni per le carte da inviare e le carte da ricevere.

Esempio:

```
{
  "_id": {
    "$oid": "67ae6ab79e68c6aef63b3b27"
  },
  "username": "Pippo",
  "name": "Need Hulk",
  "receive": [
    {
      "id": 1009351,
      "name": "Hulk",
      "thumbnail": "http://i.annihil.us/u/prod/marvel/i/mg/5/a0/538615ca33ab0.jpg"
    }
  ],
  "send": [
    {
      "id": 1009262,
      "name": "Daredevil",
      "thumbnail": "http://i.annihil.us/u/prod/marvel/i/mg/d/50/50febb79985ee.jpg"
    },
    {
      "id": 1017320,
      "name": "Iron Man (Iron Man 3 - The Official Game)",
      "thumbnail": "http://i.annihil.us/u/prod/marvel/i/mg/9/03/5239c1408c936.jpg"
    }
  ]
}
```

Campi:

- **_id**: id assegnato da MongoDB
- **username**: username dell'utente che ha creato lo scambio
- **name**: nome dello scambio
- **receive**: informazioni sulle carte da ricevere
 - **id**: id dell'eroe da ricevere
 - **name**: nome dell'eroe da ricevere
 - **thumbnail**: immagine dell'eroe da ricevere
- **send**: informazioni sulle carte da inviare
 - **id**: id dell'eroe da inviare
 - **name**: nome dell'eroe da inviare
 - **thumbnail**: immagine dell'eroe da inviare

Moduli aggiuntivi

.ENV

Questo modulo viene utilizzato per salvare credenziali, chiavi e dati sensibili in locale, senza esporli direttamente sul server. Per fare ciò ho salvato nella directory principale un file `.env` con il seguente contenuto (le informazioni originale sono state omesse per sicurezza):

```
# Server setup
HOST = "localhost"
PORT = "3000"

# Mongodb connection string
MONGODB_URI = "your mongodb connection string"

# Marvel API keys
PUBLIC_MARVEL_KEY = "your public api key"
PRIVATE_MARVEL_KEY = "your private api key"
```

UUID (SESSION TOKENS)

Per gestire le sessioni degli utenti sul sito e evitare che si acceda a pagine senza previa autorizzazione, ho utilizzato il modulo UUID. Questo mi permette di creare token da passare come cookie al browser dell'utente per fornire autenticazione.

Le funzioni principali e le classi sono contenute in `/src/lib/session.js`.

Ho creato una classe che contiene un costruttore con username e data di scadenza del token sessione e una funzione che verifica se il token è scaduto:

```
class Session {
    constructor(username, expiresAt) {
        this.username = username
        this.expiresAt = expiresAt
    }

    isExpired() {
        this.expiresAt < (new Date())
    }
}

/* We save locally all the session tokens */
const sessions = {}
```

Successivamente ho scritto delle funzioni che gestiscono i vari passaggi di autenticazione e si possono riassumere in:

- `generateSession(username)`: genero un token che dura 30 minuti, creo una sessione con l'username e la data di scadenza e poi salvo l'informazione dentro `sessions`. Alla fine ritorno il token di sessione e il server lo invierà all'utente sotto forma di cookie.

- `validateSession(cookies)`: riceve il cookie dell'utente che contiene la sessione e verifica che sia valida e non scaduta confrontandola con quella salvata localmente su `sessions`. Se il token è valido ritorna true, altrimenti false.
- `refreshSession(cookies)`: ogni volta che si carica una pagina il cookie, dopo essere verificato, viene rigenerato e inviato nuovamente all'utente. Questo permette di allungare la scadenza del token, senza far perdere la sessione all'utente.
- `logoutSession(cookies)`: quando l'utente vuole fare il logout viene ricevuto il cookie di sessione per poter eliminare il token salvato localmente in `sessions`, in modo tale che non possa più essere utilizzato. Ritorna true se il token è stato eliminato, false se il token era errato.

SWAGGER

Per testare la funzionalità delle API e fornire una documentazione dettagliata ho utilizzato il modulo Swagger. Per configuralo ho creato un file `src/api/swagger.js` che contiene le configurazioni principali e la descrizione degli endpoint.

Il file si presenta in questo modo (ho omesso del codice per leggibilità):

```
const swaggerJSDoc = require("swagger-jsdoc");

const swaggerDefinition = {
    "openapi": "3.0.3",
    "info": {
        "title": "Album delle Figurine dei Supereroi API",
        "description": "Documentation for the \"Album delle Figurine dei Supereroi\" API",
        "version": "1.0.11"
    },
    "host": "localhost:3000",
    "basePath": "/",
    "tags": [
        {
            "name": "session",
            "description": "Manages log in, register and log out of users"
        },
        {
            "name": "user",
            "description": "Manages user information"
        },
        {
            "name": "credits",
            "description": "Manages user credits"
        },
        {
            "name": "cards",
            "description": "Manages getting, selling and updating cards"
        },
        {
            "name": "trades",
            "description": "Manages trades between users"
        }
    ]
}
```

```
[  
  "paths": {  
    /* SESSION */  
    "/register": {  
      "post": {  
        "tags": [  
          "session"  
        ],  
        "description": "Registers a user with the info given in the body",  
        "requestBody": {  
          "description": "tuple used for registration",  
          "required": true,  
          "content": {  
            "application/Json": {  
              "schema": {  
                "$ref": "#/definitions/userRegister"  
              }  
            }  
          }  
        }  
      },  
      "responses": {  
        "200": {  
          "description": "Returns user info",  
        },  
        "400": {  
          "description": "User input missing",  
        },  
        "500": {  
          "description": "Error fetching the database",  
        },  
      }  
    },  
  },  
  ...  
  ...  
  const options = {  
    swaggerDefinition,  
    apis: ["../../app.js"],  
  };  
  
  const swaggerSpec = swaggerJSDoc(options);  
  module.exports = swaggerSpec;
```

Per poter visualizzare la pagina creata da Swagger basterà navigare alla pagina [/api-docs](#):

The screenshot shows the Swagger UI interface for the "Album delle Figurine dei Supereroi API". The title bar includes the Swagger logo and "Supported by SMARTBEAR". Below the title, it says "Album delle Figurine dei Supereroi API 1.0.11 OAS 3.0 Documentation for the 'Album delle Figurine dei Supereroi' API". The interface is organized into sections:

- session**: Manages log in, register and log out of users
 - POST /register**
 - POST /login**
- user**: Manages user information
 - GET /user/{username}**
 - PUT /user/{username}**
 - DELETE /user/{username}**
- credits**: Manages user credits
 - GET /credits/{username}**
 - POST /credits/{username}**
- cards**: Manages getting, selling and updating cards
 - GET /cards**

Inoltre si potranno anche visualizzare e testare i vari endpoint:

POST /login

user Manages user information

GET /user/{username}

Gets the info of the specified user

Parameters

Name	Description
username <small>required (path)</small>	Username Veronica

Responses

Curl

```
curl -X 'GET' \
'http://localhost:3000/user/Veronica' \
-H 'accept: */*'
```

Request URL

<http://localhost:3000/user/Veronica>

Server response

Code	Details
200	Response body

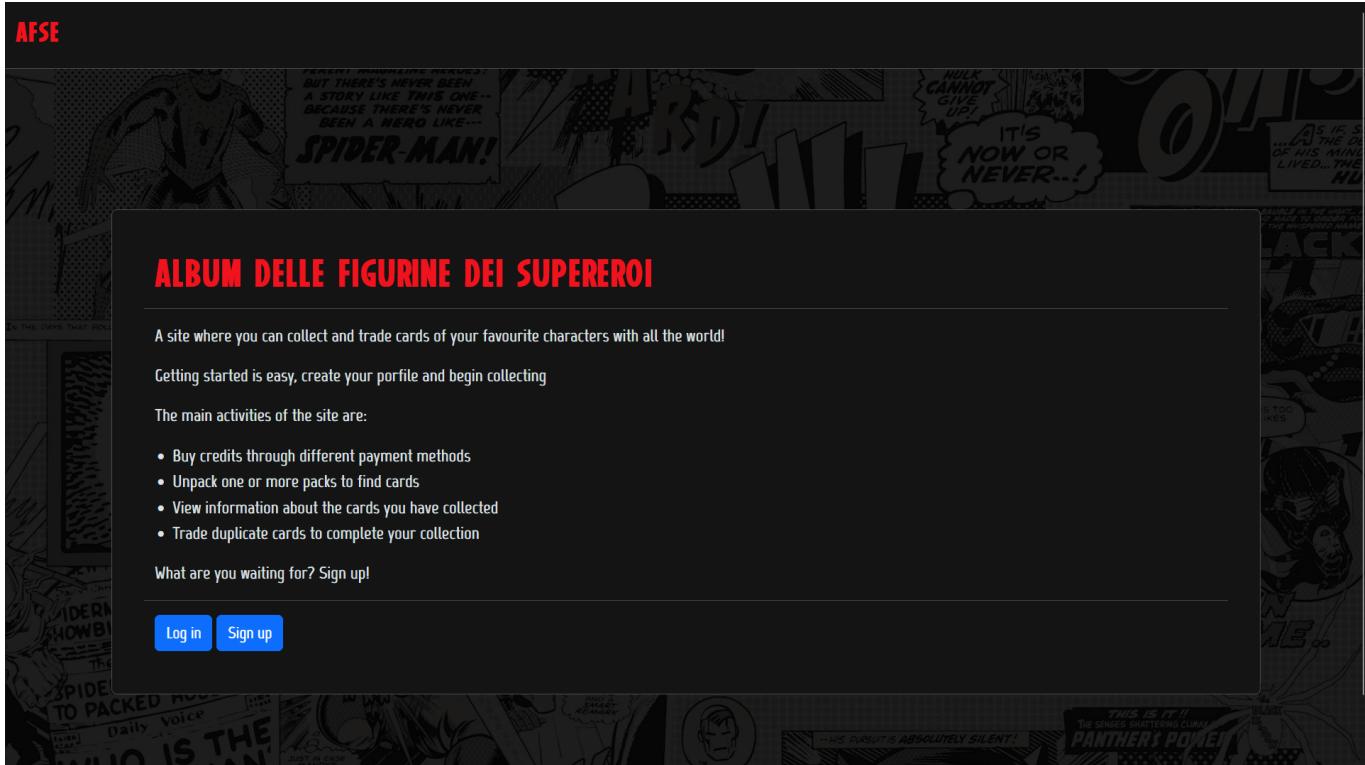
```
{
  "_id": "67b3593173cf42d420316a399",
  "username": "Veronica",
  "email": "veronica@mail.com",
  "password": "fa8a2036538b8f367fcf7bf4c63b789",
  "name": "Veronica",
  "series": "469",
  "credits": 0.1,
  "cards": [
    {
      "id": 1011237,
      "name": "Tiger's Beautiful Daughter",
      "thumbnail": "http://i.annihil.us/u/prod/marvel/i/mg/9/80/4ce5a59d27a81.jpg",
      "number": 2,
      "inTrade": false
    },
    {
      "id": 1011283,
      "name": "Lords of Avalon",
      "thumbnail": "http://i.annihil.us/u/prod/marvel/i/mg/b/40/image_not_available.jpg",
      "number": 1,
      "inTrade": false
    },
    {
      "id": 1009439,
      "name": "The Shadowed Throne",
      "thumbnail": "http://i.annihil.us/u/prod/marvel/i/mg/b/40/image_not_available.jpg",
      "number": 1,
      "inTrade": false
    }
  ],
  "responseHeaders": {
    "access-control-allow-origin": "*",
    "connection": "Keep-Alive",
    "content-length": 4232,
    "content-type": "application/json; charset=UTF-8",
    "date": "Tue, 28 Feb 2023 16:35:12 GMT",
    "etag": "W/\"1088-00NWWWW15xMz2lly30XLdKtIw\"",
    "keep-alive": "timeout=5",
    "x-powered-by": "Express"
  }
}
```

PUT /user/{username}

DELETE /user/{username}

Funzionamento

HOME PAGE



REGISTRAZIONE

SIGN UP

Username
Veronica

Email
veronica@mail.com

Password

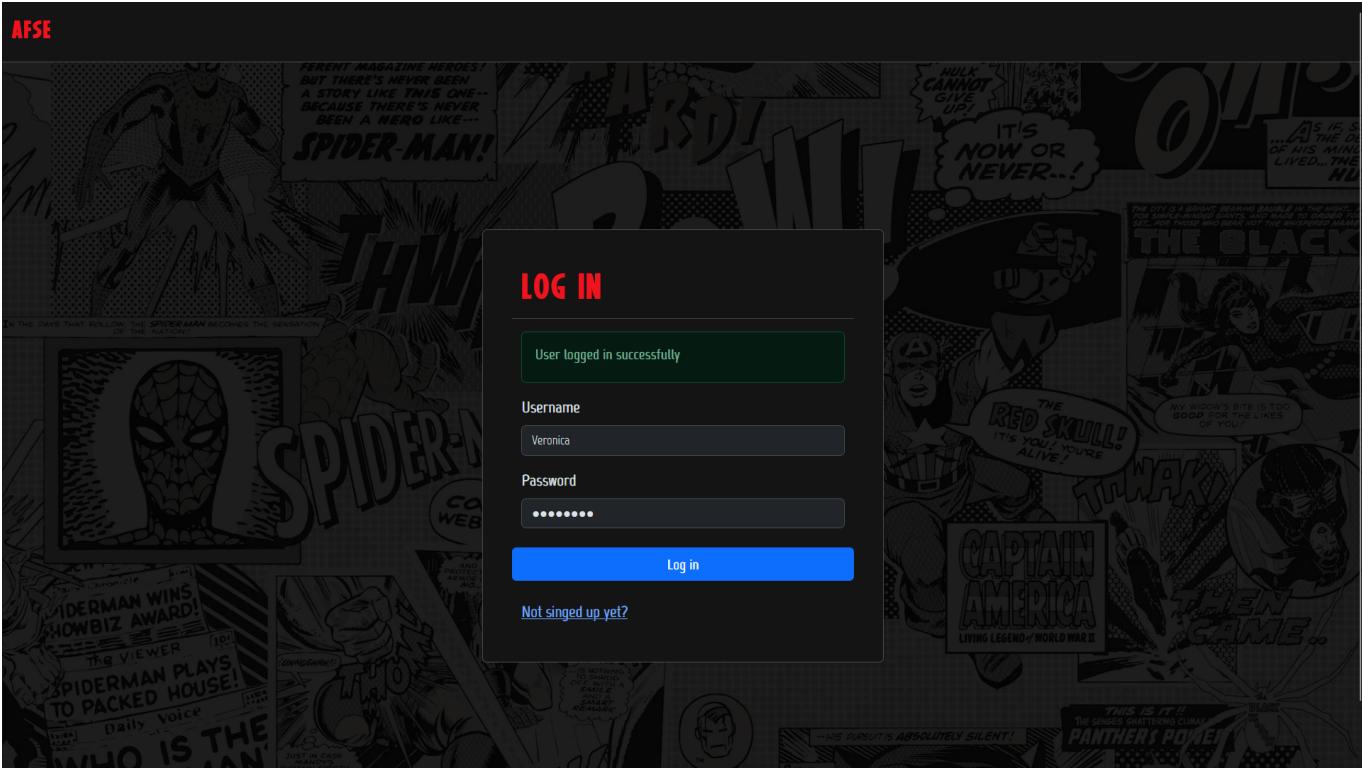
Favourite Hero
Enter the first letters of the hero to search
wol Wolverine (LEGO Marvel Super Heroes)

Favourite Series
Enter the first letters of the series to search
ult Infinity [2013]

[Sign up](#)

[Already registered?](#)

LOGIN



PROFILO

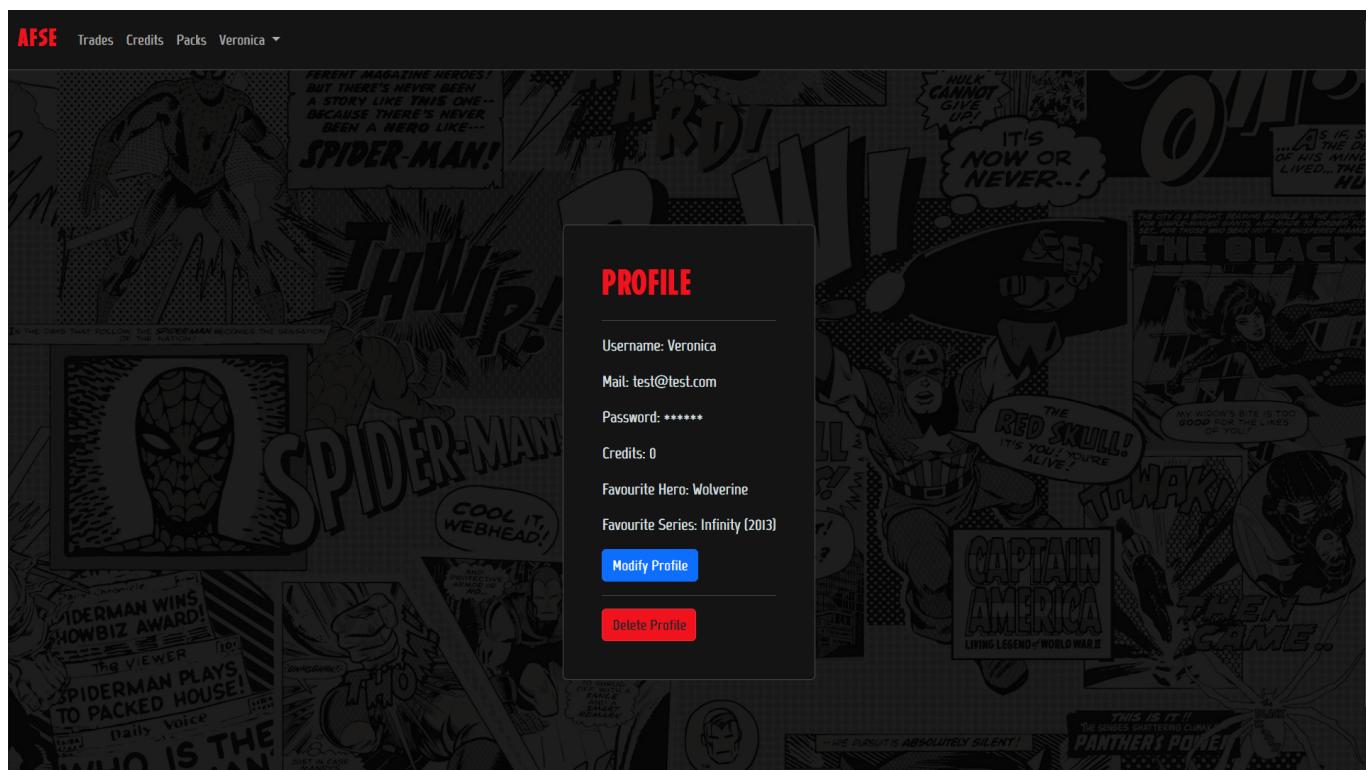
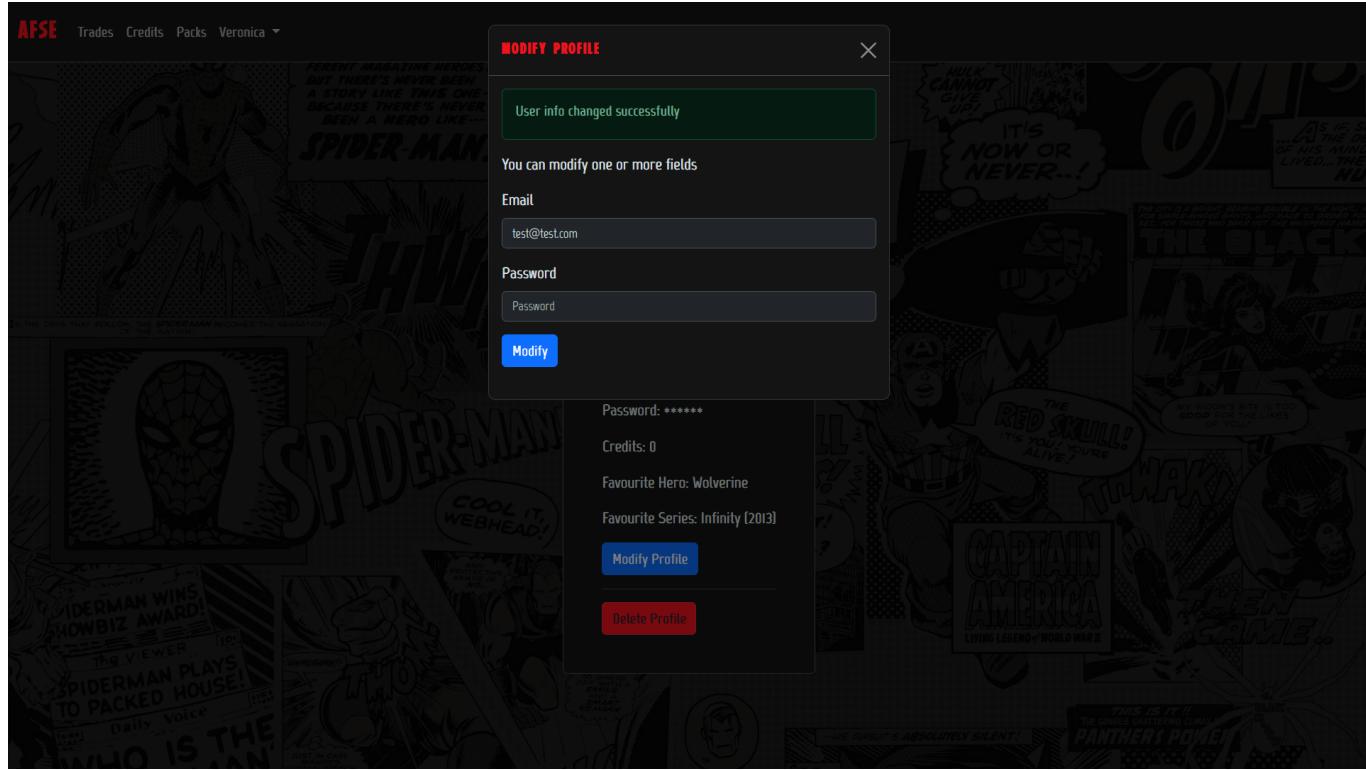
PROFILE

Username: Veronica
Mail: veronica@mail.com
Password: *****
Credits: 0
Favourite Hero: Wolverine
Favourite Series: Infinity (2013)

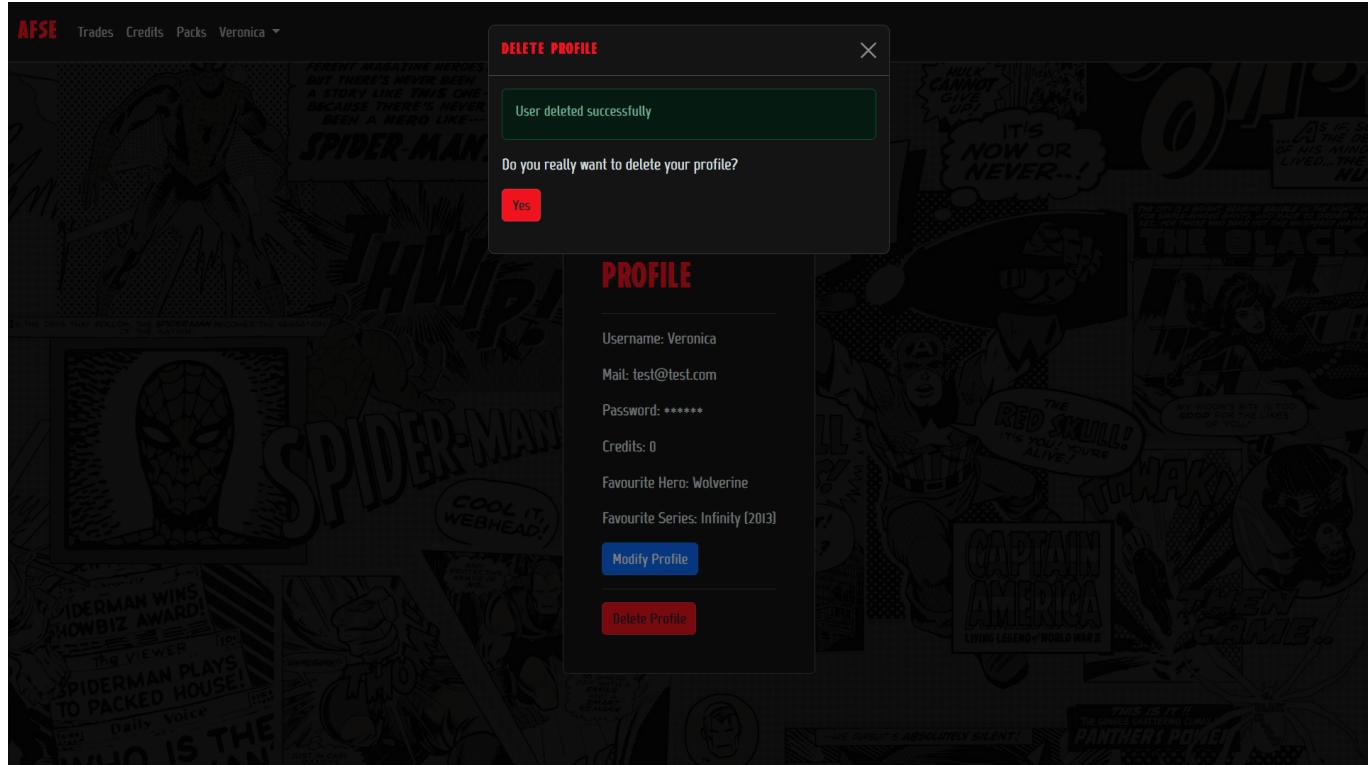
[Modify Profile](#)

[Delete Profile](#)

MODIFICA PROFILO

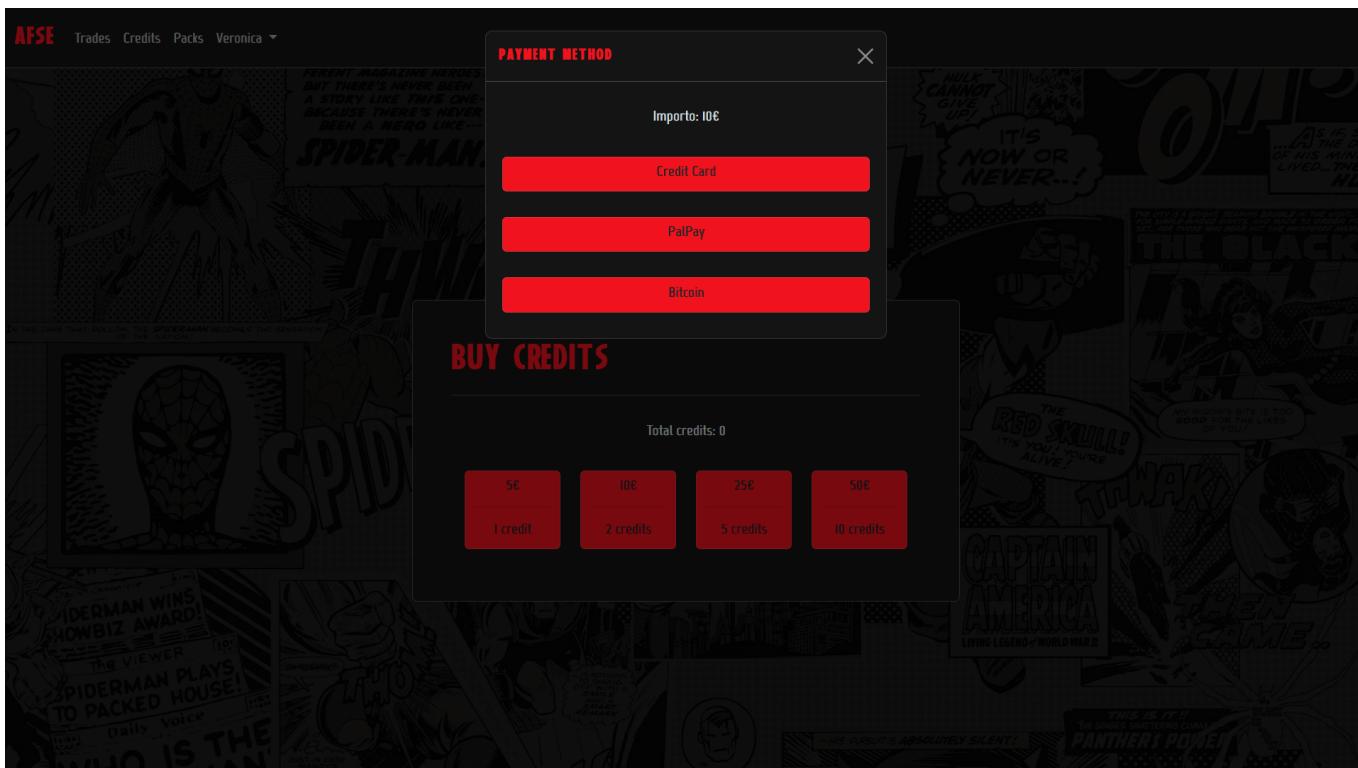
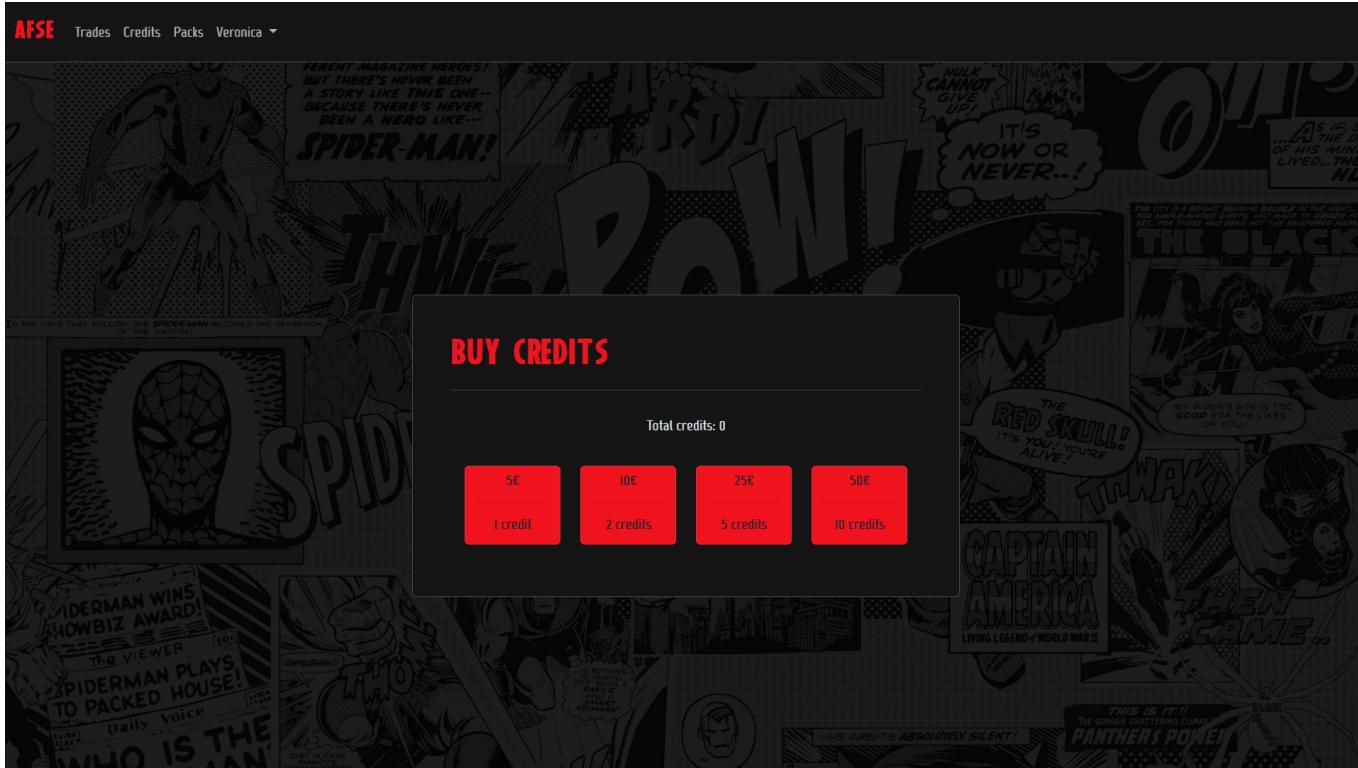


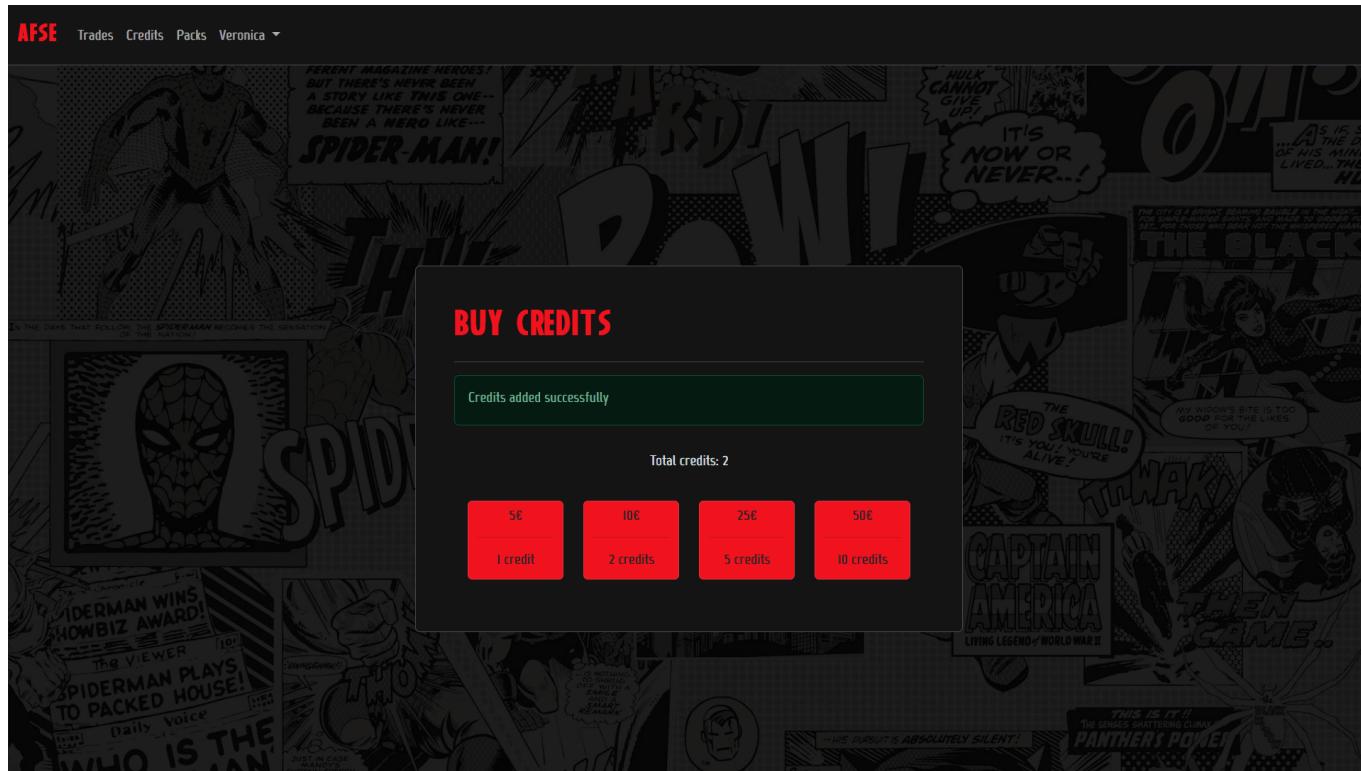
ELIMINA PROFILO



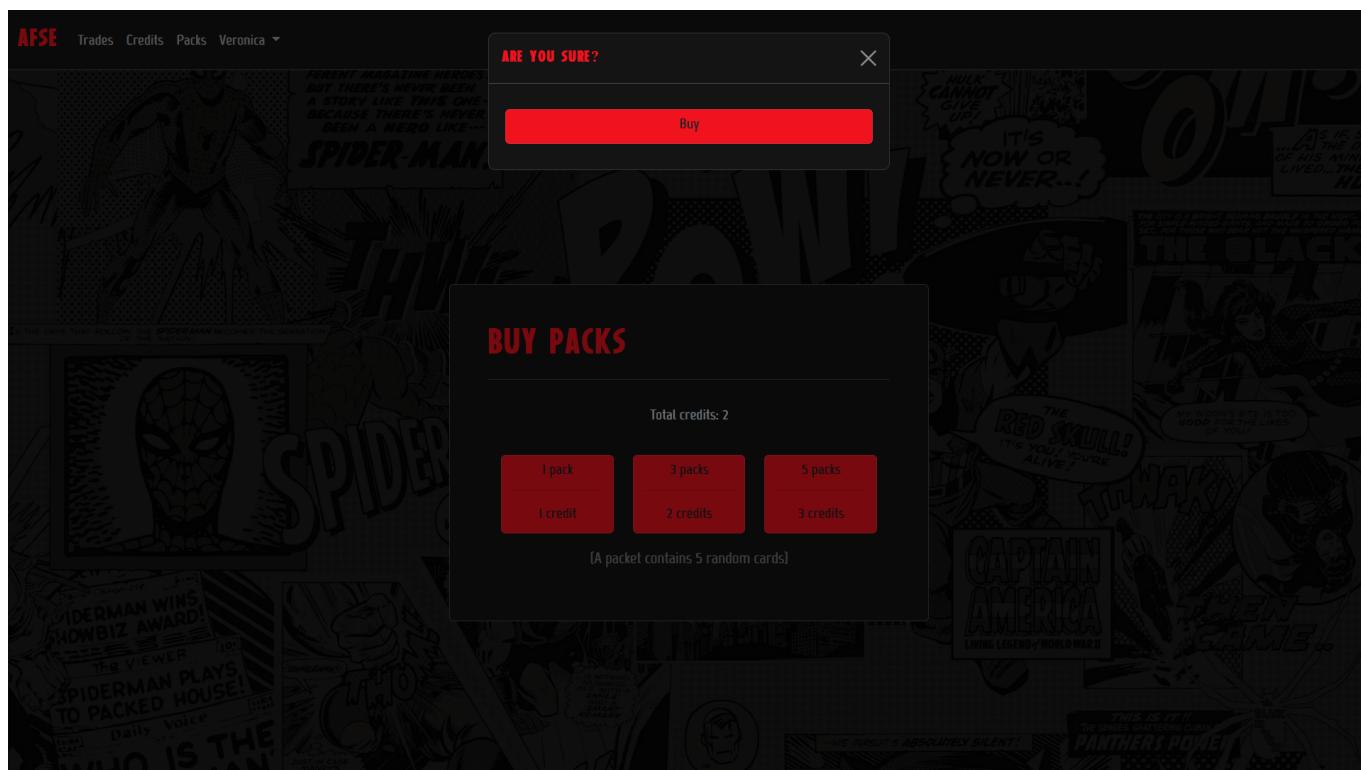
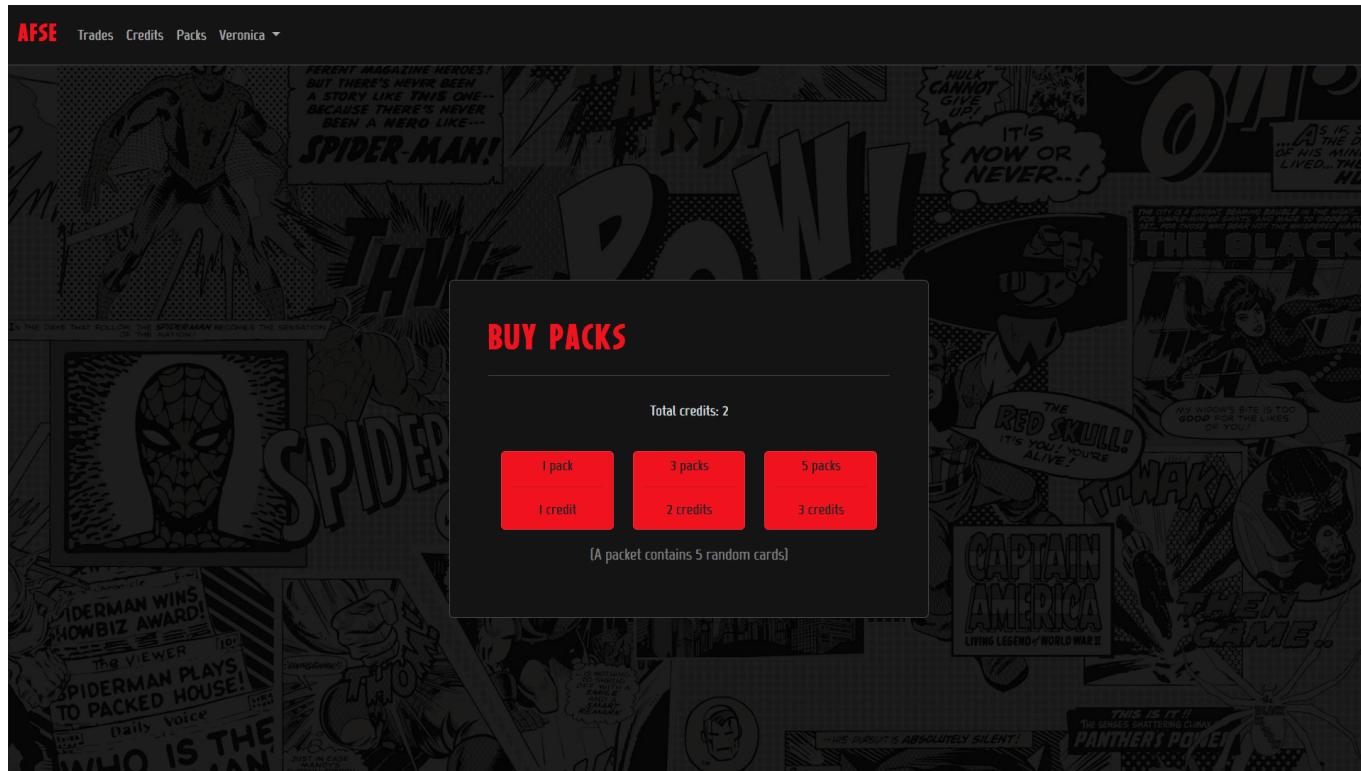
Si verrà riportati alla homepage

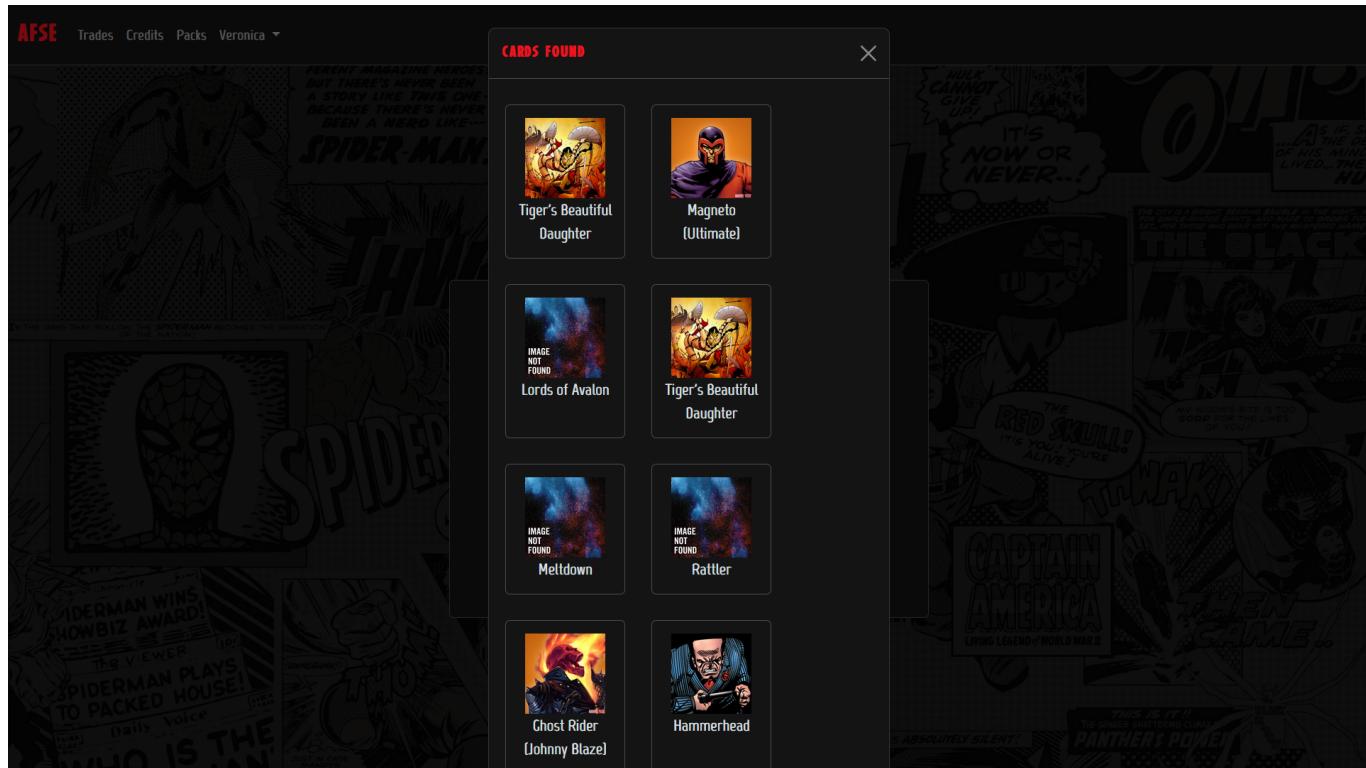
ACQUISTO CREDITI





ACQUISTO PACCHETTI





ALBUM

Tutti gli eroi:

ALBUM

Obtained cards (without doubles): 29/156

Show user's cards

3-D Man	A-Bomb (HAS)	A.I.M.	IMAGE NOT FOUND Aaron Stack	Abomination (Emil Blonsky)	Abomination (Ultimate)
Absorbing Man	Abyss	Abyss (Age of Apocalypse)	IMAGE NOT FOUND Adam Destine	Adam Warlock	Aegis (Trey Rollins)

Eroi collezionati:

AFSE Trades Credits Packs Veronica ▾

ALBUM

« ▶ »

Show all the cards

Obtained cards (without doubles): 29/1564

(2) Tiger's Beautiful Daughter	(1) Magneto (Ultimate)	IMAGE NOT FOUND	IMAGE NOT FOUND	IMAGE NOT FOUND	(1) Rattler
(1) Hammashad	IMAGE NOT FOUND	IMAGE NOT FOUND	(1) Silver	(1) M (Monet St. Iron Man 2 - The) Iron Man 2 - The	(1) Titanium Man

VISUALIZZAZIONE CARTE

Carte non ottenute:

AFSE Trades Credits Packs Veronica ▾

ADAM WARLOCK



Description:
Adam Warlock is an artificially created human who was born in a cocoon at a scientific complex called The Beehive.

Carte ottenute:



MAGNETO (ULTIMATE)



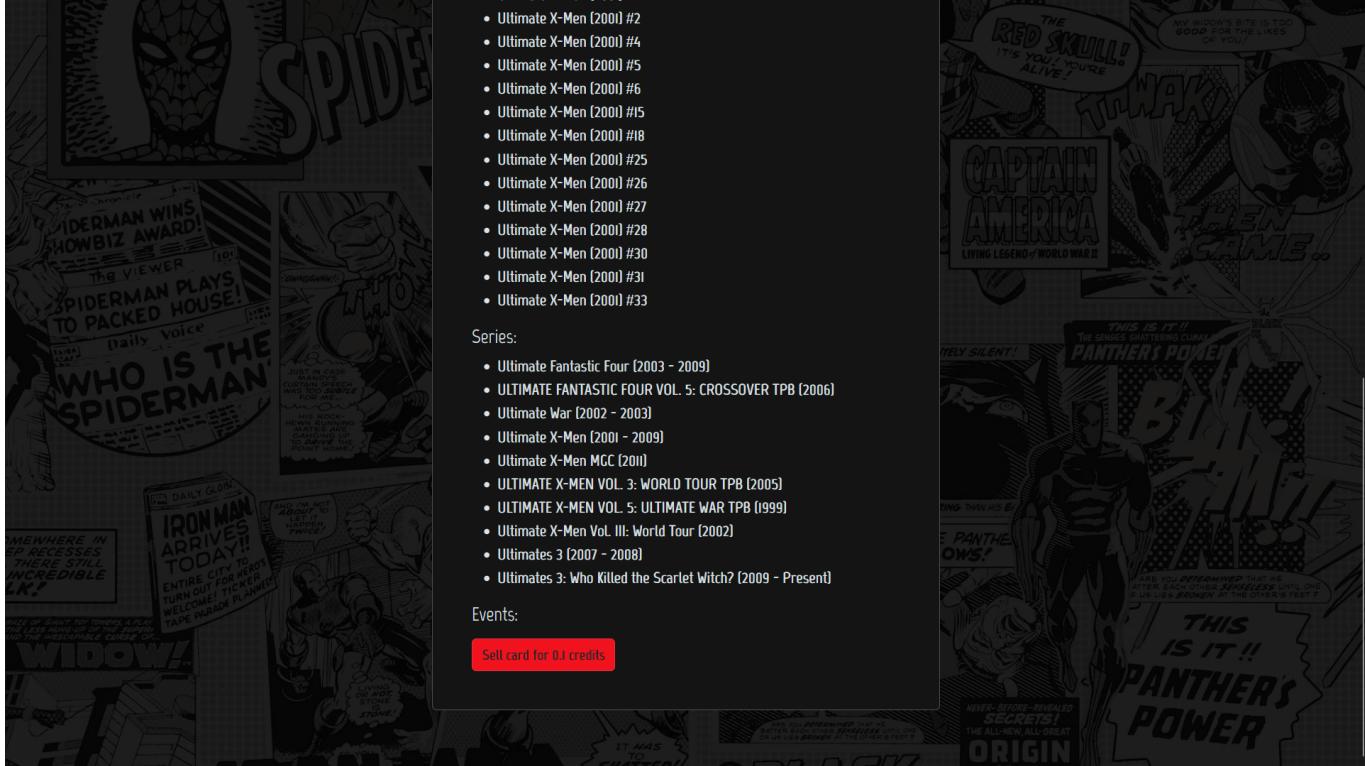
Description:

Comics:

- Ultimate Fantastic Four (2003) #22
- ULTIMATE FANTASTIC FOUR VOL. 5: CROSSOVER TPB (Trade Paperback)
- Ultimate War (2002) #1
- Ultimate War (2002) #2
- Ultimate War (2002) #3
- Ultimate War (2002) #4
- Ultimate X-Men (2001) #1
- Ultimate X-Men (2001) #2
- Ultimate X-Men (2001) #4
- Ultimate X-Men (2001) #5
- Ultimate X-Men (2001) #6
- Ultimate X-Men (2001) #8
- Ultimate X-Men (2001) #18
- Ultimate X-Men (2001) #25



VENDITA CARTE



AFSE Trades Credits Packs Veronica ▾

MAGNETO (ULTIMATE)

Card sold successfully

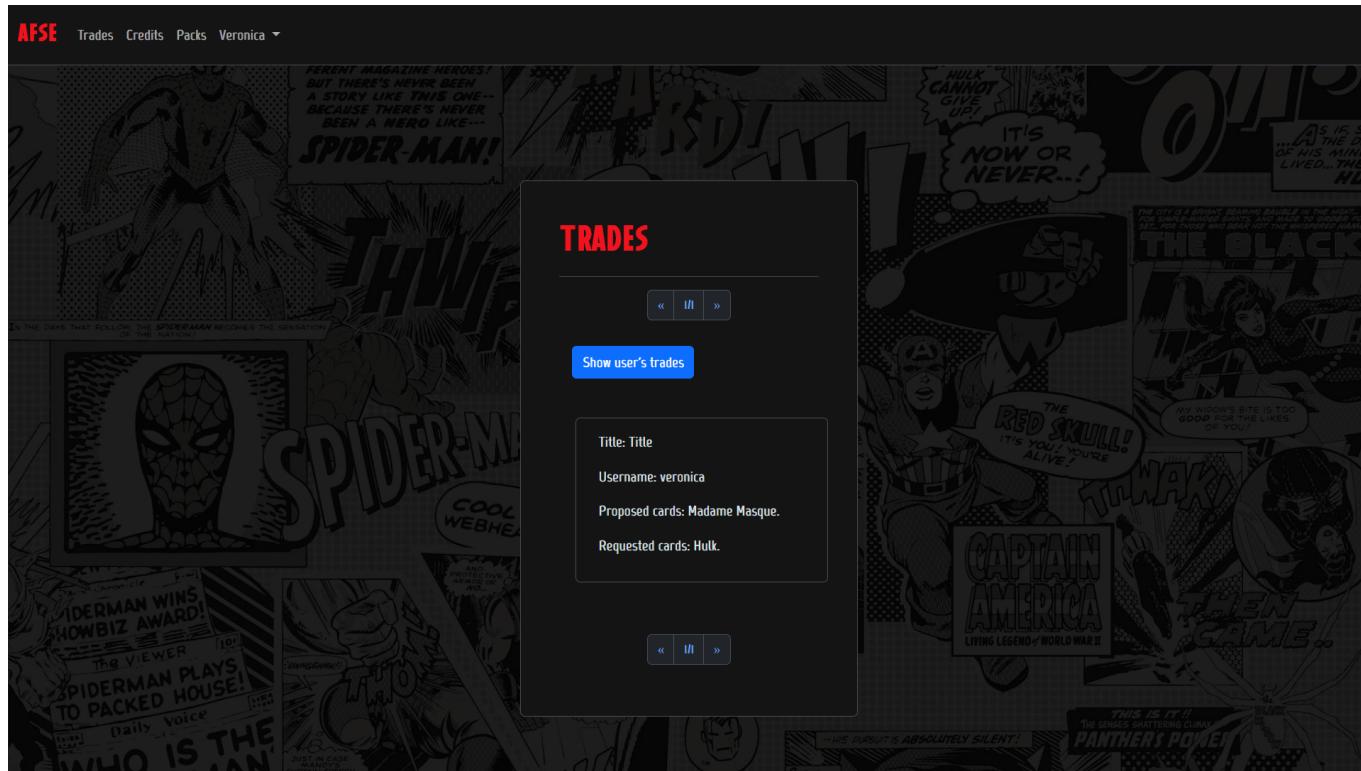
Description:

Comics:

- Ultimate Fantastic Four (2003) #22
- ULTIMATE FANTASTIC FOUR VOL. 5: CROSSOVER TPB (Trade Paperback)
- Ultimate War (2002) #1
- Ultimate War (2002) #2
- Ultimate War (2002) #3
- Ultimate War (2002) #4
- Ultimate X-Men (2001) #1

SCAMBI

VISUALIZZAZIONE SCAMBI



CREAZIONE SCAMBI

The interface shows a central modal window titled "TRADES". Inside, there are two sections: "Show all trades" and "[+] Add trade". Below these are navigation arrows and a search bar labeled "Hero".

TRADES

Show all trades

[+] Add trade

Hero

ADD TRADE

Title: Need Wolverine

Heroes to receive: Wolverine

Enter the first letters of the hero to search: Hero Find

Heroes to send: Tiger's Beautiful Daughter

Enter the first letters of the hero to search: Hero Find

Add trade

The screenshot shows the AFSE mobile application interface. At the top, there's a navigation bar with the AFSE logo and links for Trades, Credits, Packs, and Veronica. The main content area is titled "TRADES". A specific trade proposal is displayed:

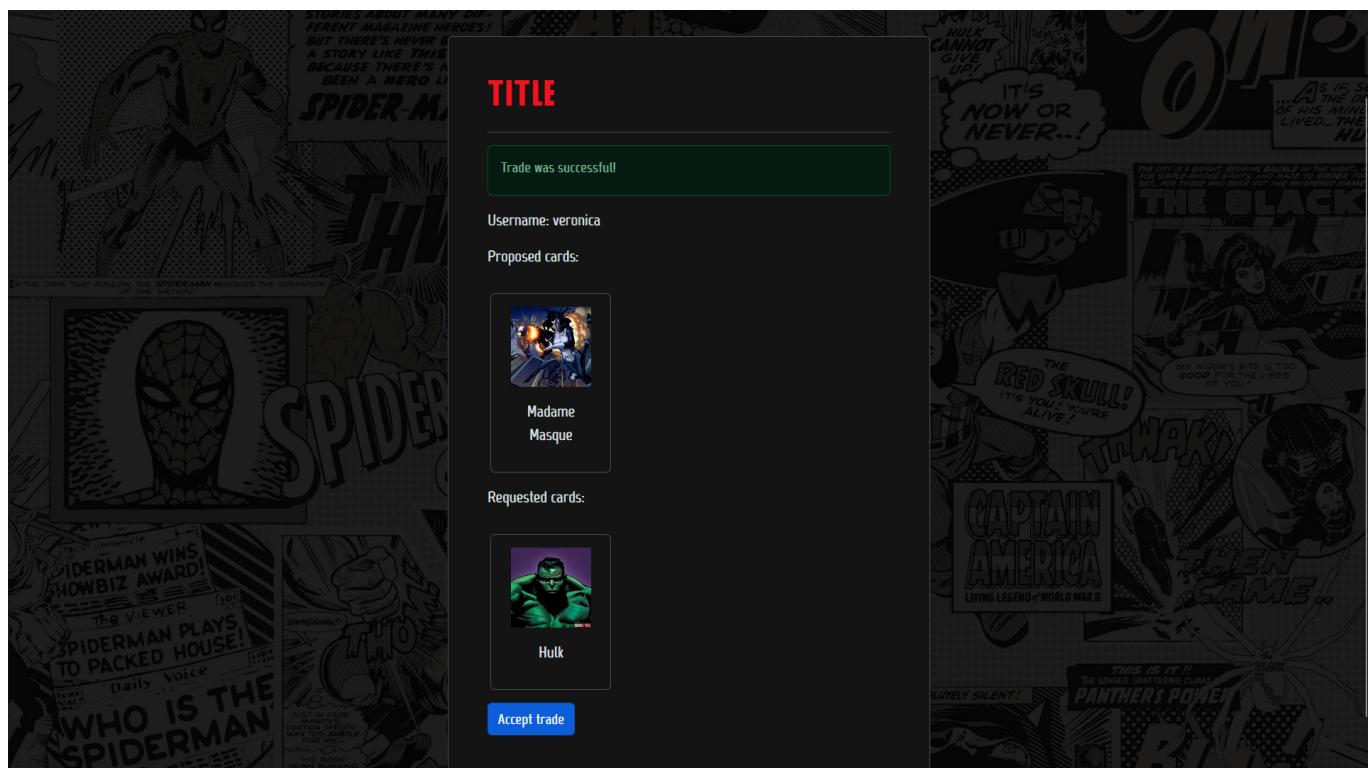
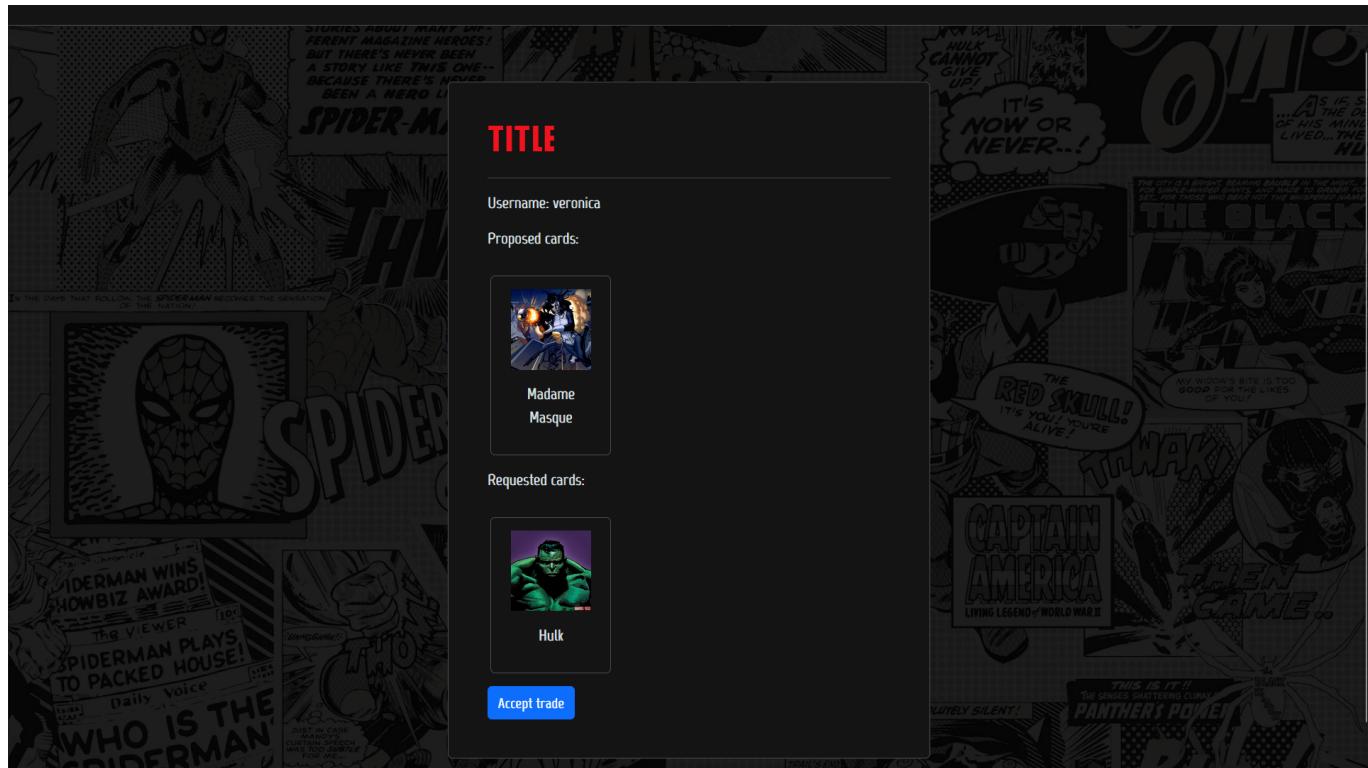
- Title:** Need Wolverine
- Username:** Veronica
- Requested cards:** Wolverine.
- Proposed cards:** Tiger's Beautiful Daughter.

Below the proposal is a red button labeled "Delete trade". At the bottom of the screen, there are navigation controls: a blue "Show all trades" button, a search bar with the placeholder "Search", and a red "[+ Add trade]" button. The background features a collage of comic book panels with characters like Spider-Man and Hulk.

ELIMINAZIONE SCAMBIO

This screenshot shows the same AFSE app interface after the trade has been deleted. A green success message at the top of the "TRADES" screen reads "Trade deleted successfully". The rest of the screen is identical to the previous one, showing the trade proposal and the standard navigation controls.

ACCETTA SCAMBIO



Note extra

- Vista la natura lenta delle API marvel ho salvato id, nome e thumbnail di tutti gli eroi all'interno della collezione "Cards" su MongoDB. Questo torna utile quando devo mostrare tutte le carte ottenibili all'utente.
Negli altri casi, in cui le richieste ritornano 2/3 eroi, ho continuato ad utilizzare l'API Marvel, pur subendo dei rallentamenti.
- Carte con più di due copie non possono essere utilizzate per più scambi per una limitazione del codice, bisogna prima aspettare che una copia si liberi da uno scambio.