

Card Game

Panoramica

L'idea di questo progetto nasce dalla voglia di sviluppare una web app in grado di registrare dati relativi al gioco di carte Yu-Gi-Oh.

In particolare l'utente una volta registrato sarà in grado di conservare dati come:

- Mazzi in possesso
- Carte all'interno di ogni mazzo
- Risultati delle partite effettuate in precedenza con altri utenti
- Scambi di carte che si vogliono effettuare

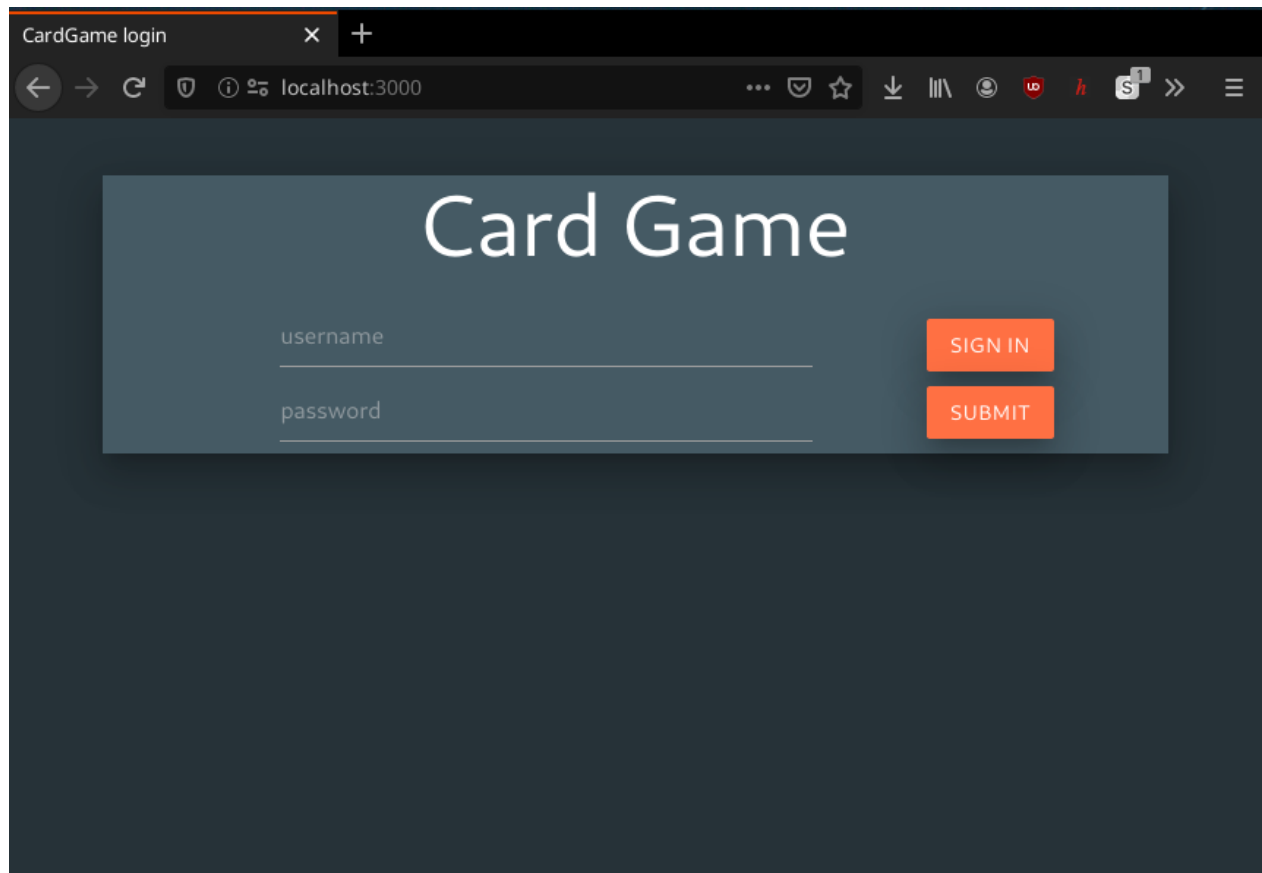
Obiettivi

L'obiettivo principale di card game è quello di fornire un'interfaccia user friendly, rendendo chiaro ai propri utenti le condizioni dei propri mazzi con l'aiuto di tabelle e grafici.

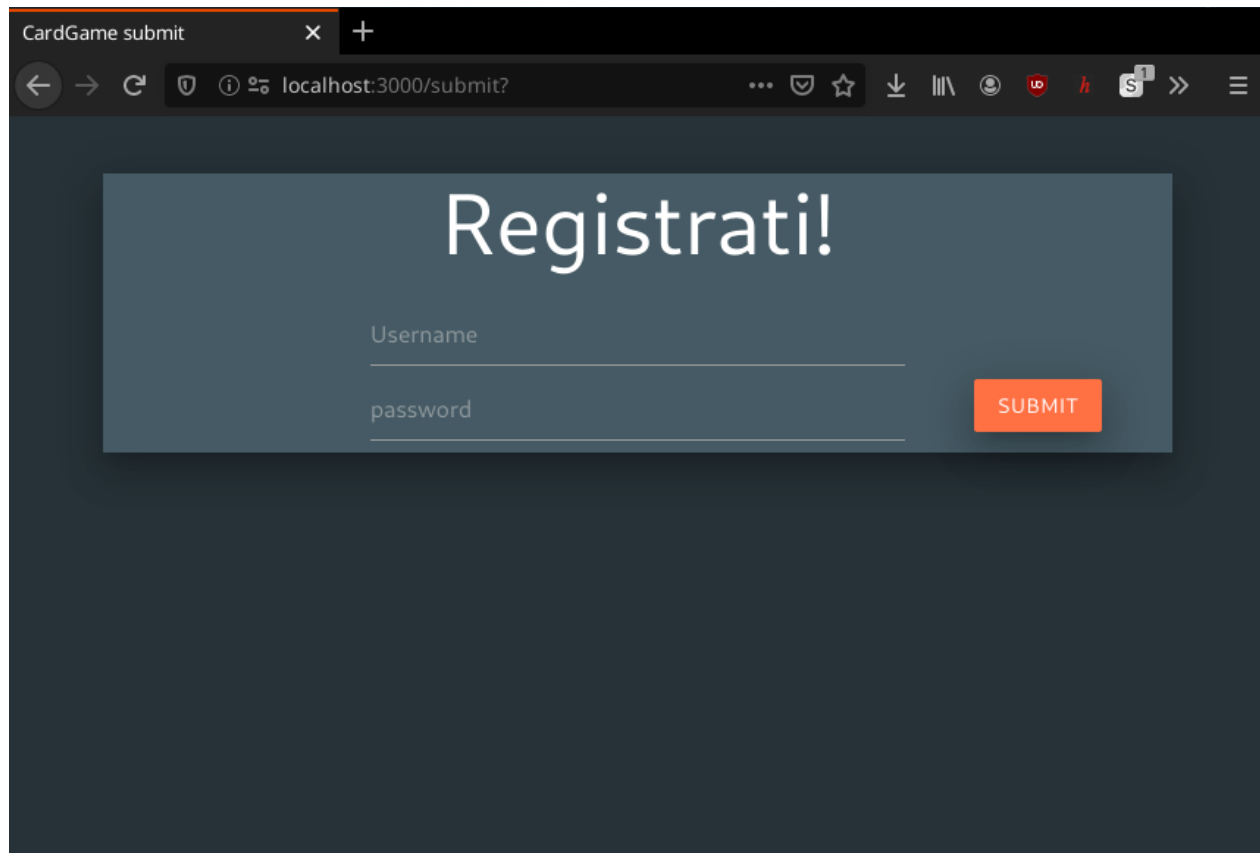
L'applicazione fornisce anche delle funzionalità utili durante le partite, come la possibilità di aggiornare i propri mazzi o punti vita in uso durante un match.

Login

L'applicazione fornisce un sistema di registrazione ed autenticazione per tenere traccia dei dati relativi ad ogni utente, se non si possiede un account è possibile registrarsi.



The screenshot shows a web browser window with the title 'CardGame login'. The address bar displays 'localhost:3000'. The main content area features a dark blue background with a central light blue box. Inside this box, the title 'Card Game' is prominently displayed at the top. Below the title, there are two input fields: 'username' and 'password'. To the right of these fields are two orange buttons: 'SIGN IN' and 'SUBMIT'. The browser's developer tools and various extension icons are visible in the top right corner of the address bar.



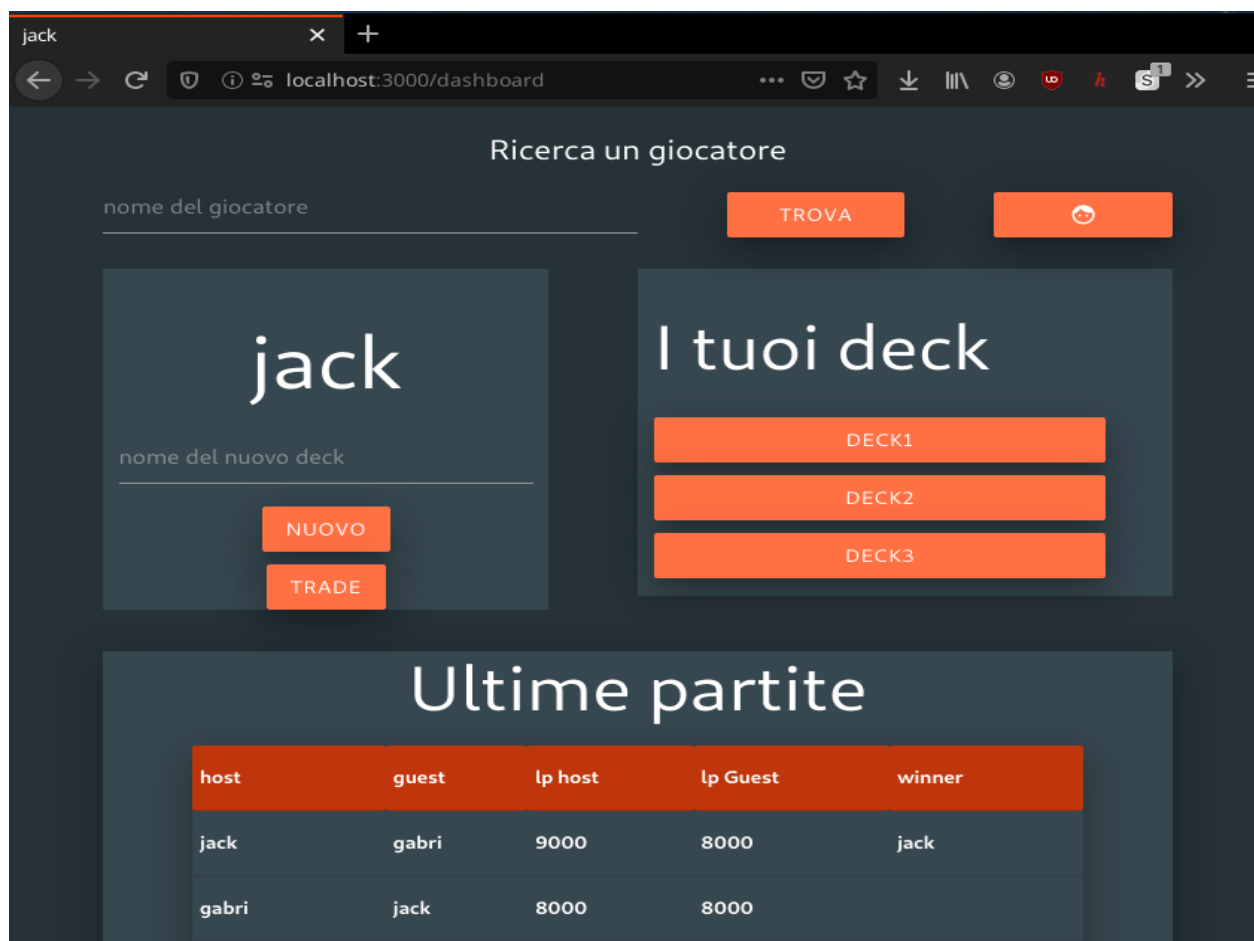
The screenshot shows a web browser window with the title 'CardGame submit'. The address bar displays 'localhost:3000/submit?'. The main content area features a dark blue background with a light blue rectangular form. The form has the heading 'Registrati!' in large white text. Below the heading are two input fields: 'Username' and 'password', both with light blue borders. To the right of the 'password' field is an orange button with the text 'SUBMIT' in white. The browser's developer tools are visible at the bottom of the window.

La registrazione avviene in maniera sicura in quanto lato server attraverso la libreria bcrypt viene criptata la password prima di essere salvata nel database.

Dashboard

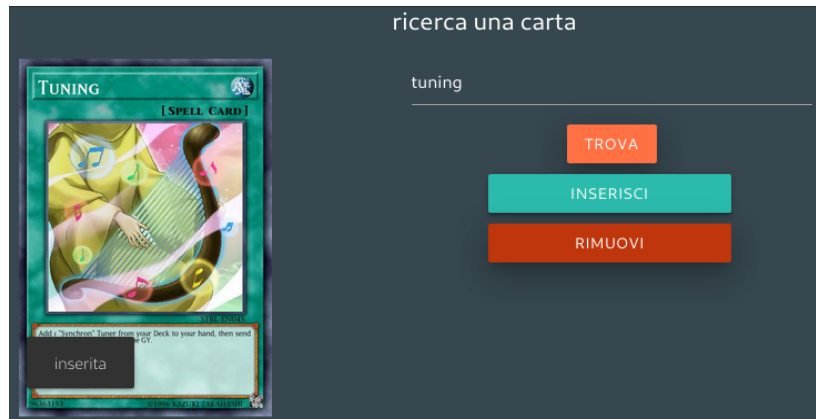
È la pagina principale all'interno della quale è possibile:

- Creare nuovi mazzi o visualizzare quelli già presenti
- Creare una nuova partita o partecipare ad una già esistente come guest
- Visitare il profilo di altri giocatori
- Visualizzare le ultime partite effettuate e le proprie statistiche



Gestione dei mazzi

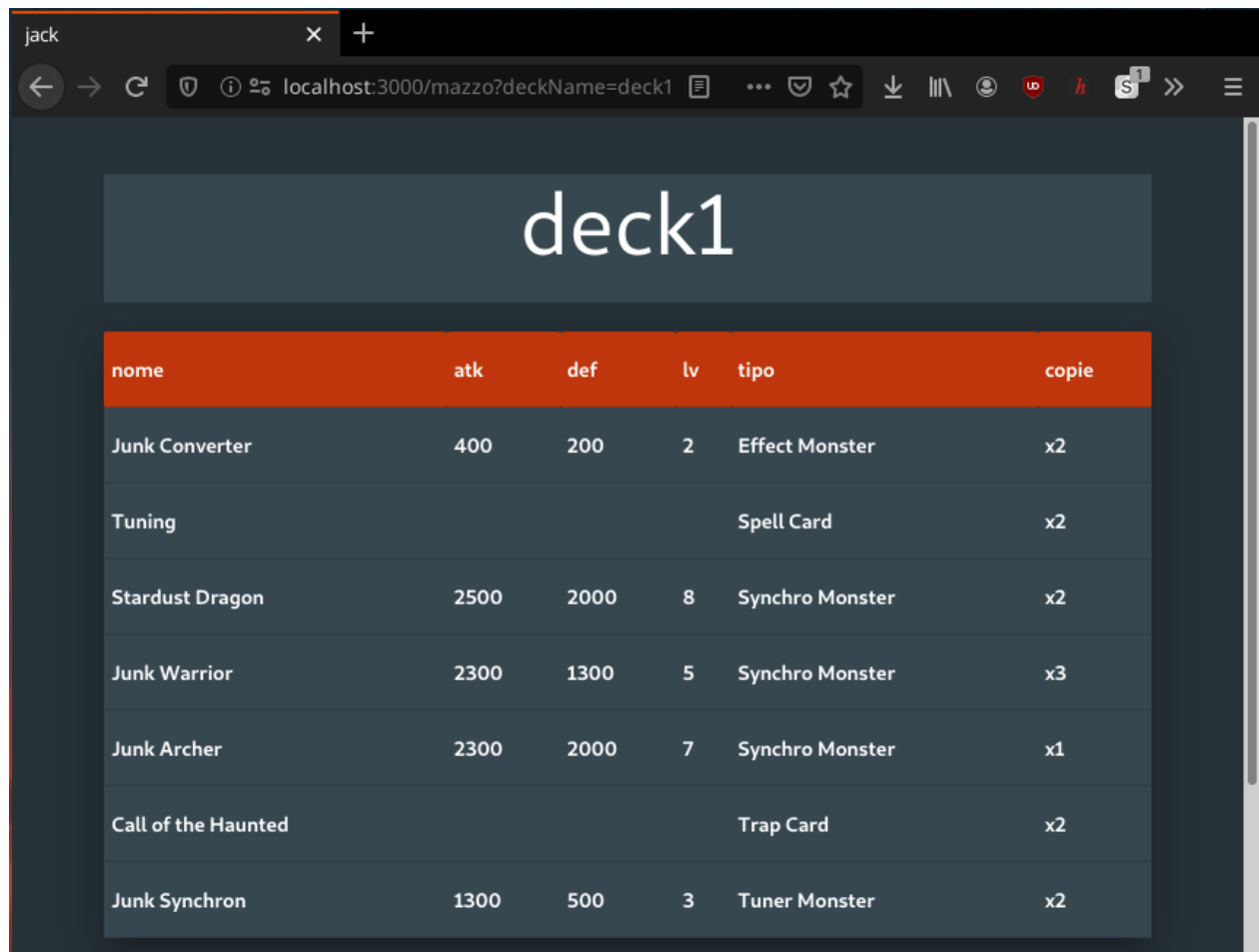
Ogni utente ha la possibilità di modificare i propri mazzi aggiungendo e/o rimuovendo carte, per farlo vi è una sezione dedicata alla ricerca di una carta.



Ogni carta come da regolamento non può essere inserita più di 3 volte nello stesso mazzo, il corretto inserimento e/o rimozione di ognuna viene comunicato con una notifica.



Nella stessa pagina è anche disponibile la lista delle carte che compongono il mazzo e due grafici che rappresentano la tipologia di carte all'interno ed il winrate relativo a quel mazzo.



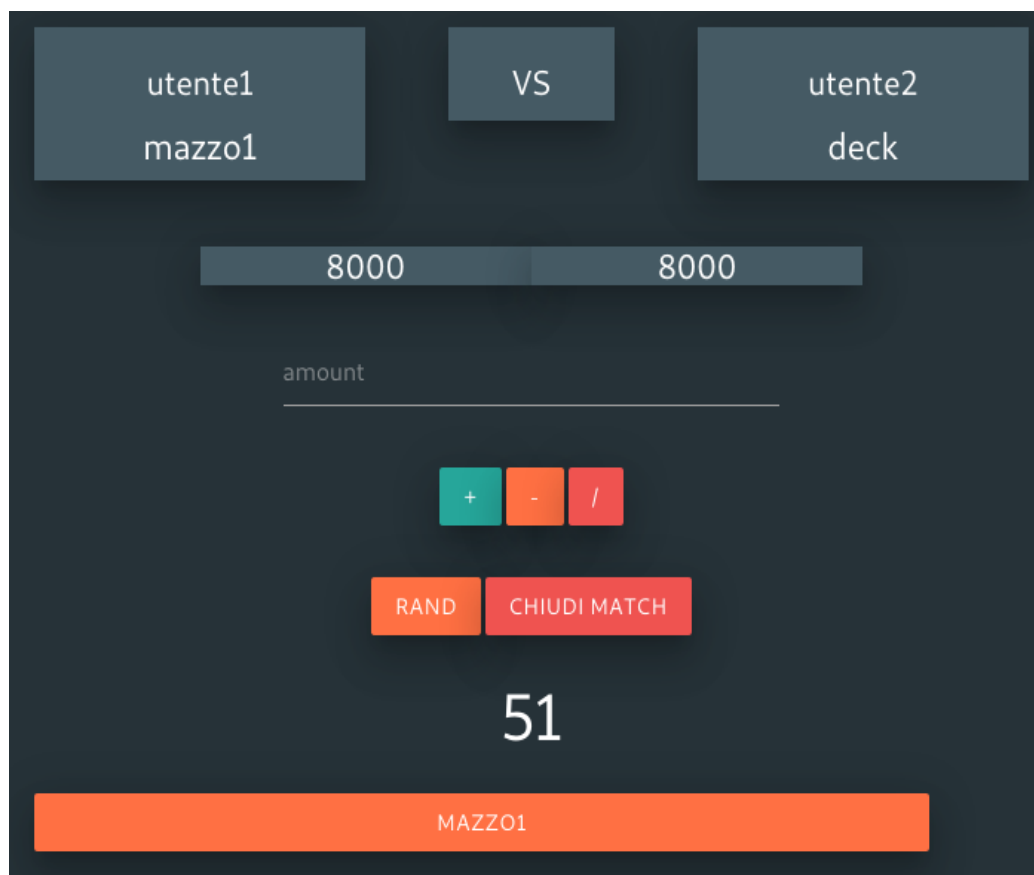
The screenshot shows a web browser window with the address bar displaying 'localhost:3000/mazzo?deckName=deck1'. The main content area has a dark background with the title 'deck1' in a large white font. Below the title is a table with a red header and dark gray rows. The table contains the following data:

nome	atk	def	lv	tipo	copie
Junk Converter	400	200	2	Effect Monster	x2
Tuning				Spell Card	x2
Stardust Dragon	2500	2000	8	Synchro Monster	x2
Junk Warrior	2300	1300	5	Synchro Monster	x3
Junk Archer	2300	2000	7	Synchro Monster	x1
Call of the Haunted				Trap Card	x2
Junk Synchron	1300	500	3	Tuner Monster	x2

All'interno della tabella sopra troviamo la lista delle carte di cui è composto il mazzo e le loro caratteristiche.

Partite

È possibile creare una nuova partita utilizzando il bottone apposito presente sulla dashboard, in questo modo si viene reindirizzati alla pagina del match, l'host comunicherà al guest l'id della partita in modo che possa unirsi anche lui. Una volta dentro entrambi c'è la possibilità di selezionare il mazzo da utilizzare, aumentare o diminuire i propri punti vita o chiudere la partita.



Caratteristiche

L'applicazione segue il pattern architetturale mvc grazie al quale è possibile dividere:

- Il metodo di accessione dati (fornito dal **model**)
- La visualizzazione dei dati (fornita dalla **view**)
- La traduzione dei comandi dell'utente in operazioni (fornita dal **controller**)

Seguendo questo pattern è possibile separare la logica applicativa (che sarà a carico di controller e model) e l'interfaccia utente (a carico delle view).

Per un'applicazione di questo tipo sarebbe necessario mantenere lo stato in modo che un utente loggato si riconosca, con l'obiettivo di rispettare i principi rest ho deciso di utilizzare il **jwt** (Json Web Token). Quando un utente effettua l'accesso viene generato un token che verrà poi utilizzato per verificare che esso sia autorizzato a compiere determinate operazioni.

Backend

Il backend si occupa della logica e della gestione dei dati all'interno del database (utenti, mazzi, carte ecc...), accettando richieste da parte dell'utente e rispondendo adeguatamente una volta che esse siano state elaborate.

Per la realizzazione del backend ho utilizzato node js (un runtime system per l'esecuzione di script javascript server side) ed il framework express (che fornisce delle funzioni avanzate per applicazioni web).

Per la ricerca delle carte ho utilizzato il servizio <https://db.ygoprodeck.com/api-guide/>

in particolare per ottenere tutte le informazioni utilizzo l'endpoint

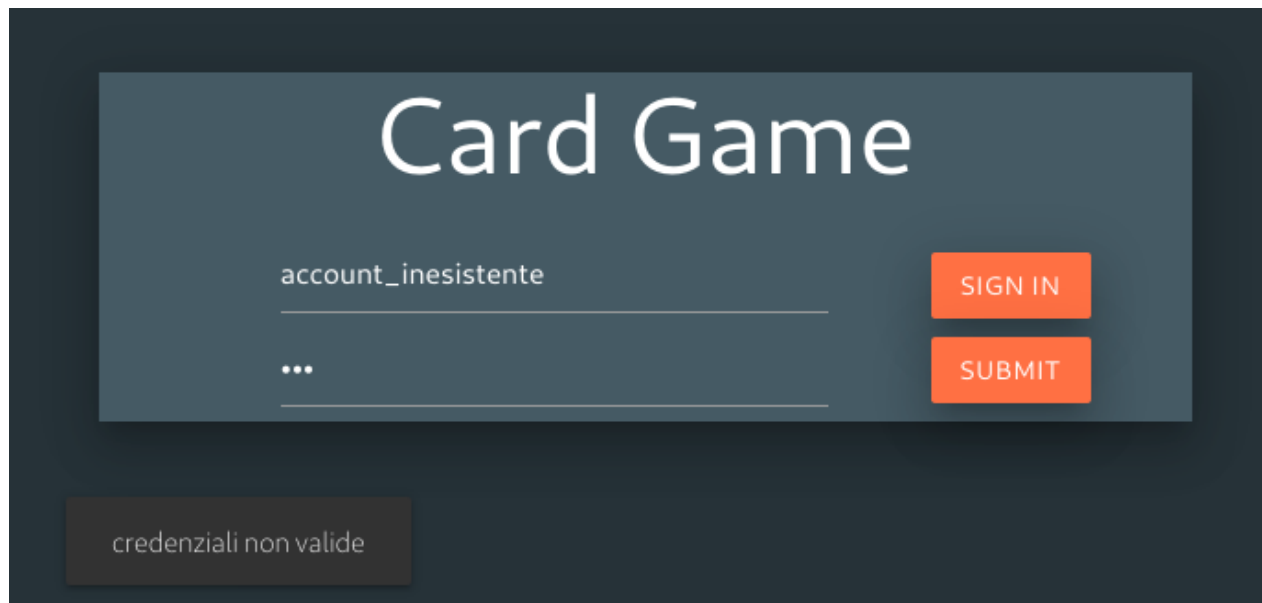
<https://db.ygoprodeck.com/api/v7/cardinfo.php> al quale aggiungo il nome della carta.

Frontend

Esso si occupa dell'interazione con l'utente presentando i dati all'interno di pagine html.

Per lo sviluppo del frontend ho utilizzato materialize (un framework che segue il design material di google) ed un file css per la definizione di alcune regole.

Il framework sopracitato da la possibilità di utilizzare molti strumenti che rendono l'interazione con l'applicazione molto piacevole, ho sfruttato le notifiche a schermo che si occupano di stampare messaggi per l'utente in base alla risposta ritornata dal server.



API REST

Card game fornisce una serie di endpoints quali:

- /user
- /deck
- /battle
- /trade

/user

Gestione degli account

get /:user/battle

Restituisce tutti i match di un utente.

Parametri(username: *string*)

- successo: (200)

```
{  
  "matches": [...]  
}
```

- fallimento (404)

```
"message": "nessuna partita  
trovata"
```

get /:user/deck

Restituisce tutti i mazzi di un utente.

Parametri(username: *string*)

- successo(200)

```
{
  "decks": [
    {
      "name": "deck1",
      "owner": "jack"
    },
    {
      "name": "deck2",
      "owner": "jack"
    },
    {
      "name": "deck3",
      "owner": "jack"
    }
  ]
}
```

- fallimento(404)

```
{"message": "impossibile trovare questo deck"}
```

se viene passato come parametro il nome del deck viene restituito il deck richiesto se esistente.

post

Crea un nuovo account.

Parametri(username: *string*, password: *string*)

- Successo (200)

```
{"user": username}
```

- Fallimento:

- Campo vuoto(400)

```
{"message": 'uno o più campi vuoti'}
```

- Username già preso(410)

```
{"message": 'username già preso'}
```

get

Restituisce un utente se registrato.

Parametri(username:*string*)

- Successo(200)

```
{
  "username": "jack",
  "win": 5,
  "lose": 5,
  "winrate": "50%",
  "games": [...],
  "used": [6,1,1],
  "decks": [...]}
}
```

- Fallimento(400)

```
{"message": 'utente non presente'}
```

/deck

Gestione dei mazzi:

post

Crea un nuovo mazzo e ritorna lo ritorna in caso di successo (necessario il jwt).

Parametri(deck:**string**) nome da dare al mazzo

- successo(200)

```
{
  "user": "username",
  "deck": "deckName"
}
```

- fallimento(409)

```
{
  "message": "nome non
disponibile"
}
```

put

Permette di aggiornare le carte di un mazzo (rimozione o aggiunta, necessario jwt).

Parametri(type: **int**, card:**object**, deck: **string**), se type è 0 allora la carta passata verrà aggiunta, se invece è 1 verrà rimossa (necessario il jwt).

- successo(200)

```
{  
  "message": "carta  
inserita/rimossa"  
}
```

- fallimento(400)

```
{  
  "message": "impossibile inserire/rimuovere  
questa carta"  
}
```

delete

Permette di rimuovere un mazzo (necessario il jwt).

Parametri(card: **object**, deck: **string**)

- successo(200)

```
{  
  "user" : "username",  
  "deck" : "deckName"  
}
```

- fallimento(400)

```
{"message": "l'utente non possiede questo mazzo"}
```

/battle

Gestione delle partite.

get

Restituisce tutti i match registrati.

Nessun parametro richiesto.

- successo(200)

```
{  
  "matches": [...]  
}
```

- fallimento(404)

```
{"message": "nessun match trovato"}
```

se viene passato come parametro l'id di un match allora viene ritornato solo quello identificato da esso.

post

Crea una nuova partita(necessario il jwt) restituendola in caso di successo.

Parametri(username: **string**)

- successo(200)

```
{
  "id": 10,
  "host": "jack",
  "guest": "prova"
}
```

- fallimento(400)

```
{"message": "impossibile creare la partita"}
```

put

Provvede vari aggiornamenti per il match (aumento e diminuzione dei punti vita, cambio del mazzo in uso, è necessario il jwt)

Parametri(id: **int**, type: **int**, amount: **int**,)

Il parametro type indica il tipo di update da effettuare al match (0: aumenta gli lp, 1: diminuisci gli lp, 2: dividi gli lp, 3: seleziona un nuovo mazzo, 4: entra il guest). Per i primi 3 casi è necessario anche il parametro amount che indica di quanto vanno aumentati/diminuiti/divisi gli lp.

- successo(200)

```
{"message": "punti vita aggiornati"} //caso 0, 1 o 2
{"message": "mazzo aggiornato"} // caso 3
{"match": {                                // caso 4
  "id": 10,
  "host": "jack",
  "guest": "prova",
  ...}
}
```

- fallimento(400)

```
{"message": "punti vita non aggiornati"} //caso 0, 1
```

```
o 2
{"message": "mazzo non aggiornato"} //caso 3
{"message": "errore"}
//caso 4
{"message": "tipo di aggiornamento non trovato"} //default
```

delete

Permette di terminare un match se si è l'host della partita(necessario il jwt).

Parametri(id: *int*)

- successo(200)

```
{
  "match":
  {
    "id": 10,
    "host": "jack",
    "guest": "prova",
    ...
  }
}
```

/trade

Gestione degli scambi.

get

Permette di ottenere tutti i trade.

Parametri(owner: **string**, card: **string**)

- successo(200)

```
{
  "trade":
  {
    "id": 1,
    "user": "prova",
    "card": "tuning",
    "message": "scambio"
  }
}
```

- fallimento(404)

```
{ "message": "nessun trade  
trovato" }
```

post

Permette di creare un nuovo trade(necessario il jwt).

Parametri(card: **string**, message: **string**)

- success(200)

```
{
  "trade":
  {
    "id": 1,
    "user": "prova",
    "card": "tuning",
    "message": "scambio"
  }
}
```

- fallimento(400)

```
{"message": "errore"}
```

delete

Permette di eliminare un trade(necessario il jwt).

Parametri(id: **int**)

- successo(200)

```
{
  "fieldCount": 0,
  "affectedRows": 1,
  "insertId": 0,
  "serverStatus": 34,
  "warningCount": 0,
  "message": '',
  "protocol41": true,
  "changedRows": 0
}
```

- fallimento

```
{"message": "errore"}
```