



Università degli Studi di Salerno  
Dipartimento di Informatica

---

Corso di Laurea Magistrale in Informatica

Basi di Dati 2  
**Yu-Gi-Oh! Catalog**

**Autori**

Simone D'Assisi

mat. 0522502038

Orlando Tomeo

mat. 0522502063

# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Obiettivi . . . . .	1
1.2	Analisi esplorativa del dataset . . . . .	2
1.3	Descrizione del Miniworld . . . . .	3
1.4	Contesto Applicativo . . . . .	3
1.4.1	Dominio di riferimento . . . . .	3
1.4.2	Utenti target . . . . .	3
1.4.3	Requisiti e aspettative del contesto . . . . .	4
<b>2</b>	<b>Soluzione Proposta</b>	<b>5</b>
2.1	Architettura generale . . . . .	5
2.2	Strumenti di sviluppo . . . . .	5
2.3	Modellazione dati e schema . . . . .	5
2.4	Funzionalità principali . . . . .	6
2.5	Proprietà BASE e teorema CAP . . . . .	7
<b>3</b>	<b>Sviluppo</b>	<b>8</b>
3.1	Backend . . . . .	8
3.1.1	Popolamento del Database . . . . .	8
3.1.2	Definizione delle query . . . . .	8
3.2	Frontend . . . . .	9
3.2.1	Pagina Home . . . . .	9
3.2.2	Catalogo . . . . .	10
3.2.3	Carta . . . . .	11
3.2.4	Login . . . . .	12
3.2.5	Signup . . . . .	12
3.2.6	Profilo . . . . .	13
3.2.7	Preferiti . . . . .	14
3.2.8	Navbar e Footer . . . . .	14

# 1 Introduzione

Il progetto nasce con l'intento di costruire un **catalogo digitale interattivo** delle carte da gioco *Yu-Gi-Oh!*, accessibile tramite un'interfaccia web moderna e user-friendly, pensata per appassionati, collezionisti e semplici curiosi. Il gioco di carte collezionabili *Yu-Gi-Oh!*, creato da Kazuki Takahashi e pubblicato da Konami, rappresenta una delle serie di Trading Card Game (*TCG*) più diffuse al mondo, con milioni di appassionati e un database in continua espansione. Ogni carta presenta un insieme eterogeneo di attributi, come il tipo (mostro, magia, trappola), l'attacco e la difesa, l'archetipo di appartenenza, eventuali effetti speciali e l'immagine grafica ufficiale. Queste caratteristiche, complesse e variabili, rendono la gestione strutturata dei dati un aspetto fondamentale per costruire un sistema informativo efficace.

## 1.1 Obiettivi

L'obiettivo principale del progetto è sviluppare un'applicazione web full-stack, capace di offrire una gestione avanzata delle carte da gioco Yu-Gi-Oh! tramite un database NoSQL. Gli obiettivi funzionali e tecnici sono:

- **Navigazione del catalogo carte:** offrire un'interfaccia fluida che consenta agli utenti di esplorare tutte le carte presenti nel database, con possibilità di ordinamento (es. per valore di attacco o difesa) e filtro (per attributi specifici come tipo, archetipo, livello, ecc.);
- **Accesso ai dettagli della singola carta:** fornire una scheda informativa dettagliata per ogni carta, comprensiva di immagine, statistiche, descrizione e archetipi;
- **Gestione utenti e autenticazione:** implementare un sistema di registrazione, login e gestione sicura dell'account, tramite hashing delle password e validazione degli input;
- **Personalizzazione tramite preferiti:** permettere agli utenti autenticati di aggiungere carte ai propri preferiti e costruire una collezione virtuale personale, persistente nel database;
- **Dashboard utente dinamica:** realizzare una pagina personale in cui l'utente possa visualizzare, aggiornare e gestire le proprie informazioni.

## 1.2 Analisi esplorativa del dataset

Le informazioni sulle carte utilizzate in questo progetto provengono dal dataset pubblico *Yu-Gi-Oh! Cards*, disponibile sulla piattaforma Kaggle<sup>1</sup>. Il dataset raccoglie i dati di 13.281 carte uniche, ognuna descritta attraverso un insieme di 29 attributi distinti, tra cui caratteristiche testuali (nome, descrizione, tipo), numeriche (attacco, difesa, livello), categoriche (archetipo, attributo, razza) e multimediali (link alle immagini). Una panoramica completa dei campi analizzati è riportata in Tabella 1.

Campo	Descrizione
<code>id</code>	Numero identificativo della carta nel dataset.
<code>name</code>	Nome della carta.
<code>type</code>	Tipo generale della carta (es. Monster, Spell, Trap...).
<code>desc</code>	Descrizione testuale o effetto della carta.
<code>atk</code>	Valore di attacco (solo per mostri).
<code>def</code>	Valore di difesa (solo per mostri).
<code>level</code>	Livello o rango della carta (solo per mostri).
<code>race</code>	Categoria o <i>Tipo</i> a cui appartiene la carta (es. Dragon, Warrior, Spellcaster...).
<code>attribute</code>	Attributo elementale della carta (es. FIRE, WATER, LIGHT...).
<code>scale</code>	Valore della Pendulum Scale (solo per Pendulum Monster).
<code>archetype</code>	Archetipo o serie tematica di appartenenza (opzionale).
<code>linkval</code>	Link Value della carta (solo per Link Monster).
<code>linkmarkers</code>	Direzioni dei Link Marker (es. Top, Bottom, Top-Left).
<code>image_url</code>	URL all'immagine principale della carta.
<code>image_url_small</code>	URL dell'immagine in formato ridotto.
<code>ban_tgc</code>	Stato della carta nella TCG Ban List (es. Banned, Limited...).
<code>ban_ocg</code>	Stato della carta nella OCG Ban List.
<code>ban_goat</code>	Stato della carta nel formato GOAT (formato vintage).
<code>staple</code>	Indica se la carta è considerata una "staple", ovvero universalmente utile in molti mazzi.
<code>views</code>	Numero totale di visualizzazioni online della carta.
<code>viewsweek</code>	Numero di visualizzazioni settimanali.
<code>upvotes</code>	Numero di voti positivi ricevuti dagli utenti.
<code>downvotes</code>	Numero di voti negativi ricevuti dagli utenti.
<code>formats</code>	Formati competitivi in cui la carta è legale o utilizzabile.
<code>treated_as</code>	Nome alternativo o identità con cui la carta viene trattata (es. trattata come "Red-Eyes B. Dragon").
<code>tcg_date</code>	Data di rilascio ufficiale nella regione TCG.
<code>ocg_date</code>	Data di rilascio nella regione OCG.
<code>konami_id</code>	Identificativo interno assegnato da Konami alla carta.
<code>has_effect</code>	Booleano che indica se la carta ha un effetto attivo.

Tabella 1: Struttura delle colonne presenti nel dataset Kaggle "Yu-Gi-Oh! Cards"

<sup>1</sup>Il dataset è reperibile al seguente indirizzo: <https://www.kaggle.com/datasets/ioexception/yugioh-cards>

### 1.3 Descrizione del Miniworld

Il sistema si basa su un dataset di carte del gioco di carte collezionabili *Yu-Gi-Oh!* e su una piattaforma utente che consente di interagire con tali carte attraverso funzionalità di autenticazione e gestione dei preferiti. Nel sistema proposto sono presenti due entità principali:

- **Carte:** ogni carta rappresenta un elemento unico del gioco *Yu-Gi-Oh!*, dotato di caratteristiche specifiche come nome, tipo, attributi, valori di attacco e difesa, descrizione, immagine e informazioni relative alle liste di ban ufficiali. Il dataset utilizzato contiene oltre 13.281 carte uniche, ciascuna descritta da un insieme eterogeneo di proprietà che consentono una classificazione dettagliata e precisa.
- **Utenti:** gli utenti sono gli attori principali del sistema, che possono registrarsi e accedere attraverso credenziali composte da email e password. Per ciascun utente vengono gestite quattro informazioni principali: il nome, un'email univoca, la password (salvata in forma cifrata mediante hashing sicuro) e una lista di carte preferite, rappresentata da riferimenti agli identificativi univoci delle carte nella collezione.

La relazione tra le entità è di tipo molti-a-molti, in quanto ogni utente può salvare più carte preferite, e ogni carta può essere preferita da più utenti. Non sono presenti ulteriori vincoli o relazioni complesse.

Il sistema è stato progettato per essere facilmente estendibile: in futuro si prevede la possibilità di introdurre funzionalità avanzate come la gestione di più collezioni personalizzate, filtri dinamici, statistiche sull'utilizzo delle carte, notifiche di aggiornamenti, e molto altro.

### 1.4 Contesto Applicativo

#### 1.4.1 Dominio di riferimento

Il progetto si inserisce nel contesto dei giochi di carte collezionabili, con un focus specifico sull'universo di *Yu-Gi-Oh!*, uno dei Trading Card Game (TCG) più diffusi a livello globale. In particolare, il sistema è pensato per supportare attività di consultazione, esplorazione e collezionismo personale delle carte. L'enorme varietà e la frequente introduzione di nuove carte rendono difficile per gli appassionati mantenere una visione chiara e aggiornata della collezione complessiva.

In questo scenario, un catalogo digitale strutturato e interattivo può semplificare l'accesso alle informazioni sulle carte, fornendo un'interfaccia intuitiva per navigare tra migliaia di elementi, filtrare in base a specifiche caratteristiche (es. tipo, attributo, livello) e salvare le carte di interesse. Il progetto si rivolge quindi a collezionisti, appassionati, e utenti occasionali che desiderano esplorare il database delle carte in modo efficiente, senza necessariamente essere coinvolti nell'aspetto competitivo del gioco.

#### 1.4.2 Utenti target

Il sistema è pensato per essere rivolto sia a collezionisti e appassionati, utenti esperti o di lunga data che desiderano esplorare l'intero database delle carte *Yu-Gi-Oh!*, consultarne i

dettagli e costruire una collezione virtuale personale, tenendo traccia delle carte preferite in modo ordinato, sia ai nuovi giocatori, utenti alle prime armi che utilizzano il sistema come strumento di supporto per conoscere meglio le carte. L'accesso alla piattaforma è strutturato per offrire funzionalità coerenti con il livello di coinvolgimento dell'utente:

- **Utenti non registrati:** possono navigare liberamente nel catalogo e visualizzare le caratteristiche dettagliate di ogni carta.
- **Utenti registrati:** hanno accesso a funzionalità aggiuntive, come la possibilità di salvare carte tra i preferiti, visualizzare una dashboard personale e gestire la propria collezione virtuale.

#### 1.4.3 Requisiti e aspettative del contesto

Il sistema è progettato per essere semplice e accessibile, con un'interfaccia intuitiva che consente agli utenti di navigare facilmente tra le diverse sezioni. Le funzionalità principali includono:

- Una **home** che descrive le principali funzionalità offerte dal sito tramite un carosello, guidando l'utente nella navigazione, e che mette in evidenza le carte più votate dagli utenti, offrendo un punto di ingresso immediato alle carte di maggior interesse.
- Una **sezione catalogo** che permette di sfogliare l'intero database di carte, con filtri per tipologia, attributo, livello e altri parametri rilevanti, facilitando la ricerca mirata.
- Una **pagina per la carta singola** che permette di visualizzarne i dettagli.
- Una **pagina profilo** riservata agli utenti registrati, dove possono gestire le proprie informazioni personali.
- Una **pagina preferiti**, dove gli utenti registrati possono gestire le proprie preferenze, visualizzare e modificare la lista di carte salvate.

Oltre alle funzionalità, si richiedono requisiti non funzionali quali:

- **Sicurezza** di base, con gestione delle password in modo sicuro tramite hashing e controlli di accesso per proteggere i dati personali degli utenti.
- **Usabilità** elevata, con interfaccia responsive e compatibilità con i principali browser e dispositivi mobili.
- **Scalabilità** e mantenibilità, per consentire future estensioni come statistiche di gioco, gestione di collezioni multiple o funzioni social.

Questi aspetti assicurano che il sistema sia non solo funzionale ma anche affidabile e pronto a evolversi con le esigenze degli utenti.

## 2 Soluzione Proposta

### 2.1 Architettura generale

Il sistema proposto adotta un'architettura client-server basata su un backend RESTful e un frontend single-page application (SPA). Il **backend** è sviluppato in *Python 3.12* con il framework Flask; espone API *REST* per gestire l'accesso e la manipolazione dei dati. Il database scelto è **MongoDB**, un sistema NoSQL che consente una gestione flessibile e scalabile dei dati. Il **frontend**, responsabile dell'interfaccia utente, della navigazione e dell'interazione con il backend tramite chiamate API è realizzato in *React*. La comunicazione tra frontend e backend avviene tramite richieste HTTP asincrone, realizzate con funzioni come fetch. Queste richieste seguono le convenzioni REST per implementare le operazioni CRUD (Create, Read, Update, Delete), permettendo al frontend di interagire con i dati in modo standardizzato e scalabile. Grazie all'approccio asincrono, l'interfaccia utente resta reattiva e può aggiornarsi dinamicamente quando arrivano i dati dal server.

### 2.2 Strumenti di sviluppo

**MongoDB** è stato scelto come database NoSQL per via della sua flessibilità nella gestione di dati semi-strutturati e per la capacità di scalare facilmente con l'aumento del volume di dati. Il modello documentale JSON-like si adatta bene al miniworld delle carte, le cui proprietà variano e possono contenere campi opzionali o complessi. Inoltre, MongoDB supporta il paradigma BASE (Basically Available, Soft state, Eventual consistency), che si sposa bene con le esigenze di un'applicazione di consultazione dove è possibile accettare una consistenza eventuale per migliorare la disponibilità e la scalabilità.

**Flask** è stato scelto come framework backend per la sua leggerezza e semplicità, che facilitano uno sviluppo rapido e modulare dell'API REST. Flask permette di creare un backend minimale ma estendibile, con gestione semplice delle rotte e facile integrazione con MongoDB tramite librerie come Flask-PyMongo. Questa scelta consente inoltre di mantenere il controllo diretto su ogni parte della logica server, senza complicazioni di framework più pesanti.

**React** è stato adottato per il frontend per la sua capacità di creare interfacce utente interattive e reattive tramite componenti riutilizzabili e gestione efficiente dello stato. L'approccio di React semplifica la costruzione di pagine dinamiche come il catalogo, la gestione dei preferiti e le pagine profilo. Inoltre, React facilita l'implementazione di chiamate asincrone alle API backend.

### 2.3 Modellazione dati e schema

Il sistema si basa principalmente su due entità fondamentali, modellate come due *collection* distinte nel database MongoDB:

- **Carte**: rappresentano gli oggetti principali del miniworld, con attributi variabili e complessi. Ogni documento nella collection **carte** contiene informazioni dettagliate quali:
  - **name**: nome della carta;

- `type`: tipo della carta (es. mostro, magia, trappola);
- `attribute`: attributo elementare (es. luce, oscurità);
- `level`: livello o rango della carta;
- `atk`: valore di attacco;
- `def`: valore di difesa;
- `description`: testo descrittivo con gli effetti o le regole della carta;
- `image_url`: URL dell’immagine associata;
- eventuali campi opzionali come `restriction` o `upvotes` per le valutazioni.

- **Utenti:** rappresentano gli utilizzatori del sistema con informazioni essenziali per l’autenticazione e la gestione della collezione personale. Ogni documento nella collection `users` contiene:
  - `nome`: nome dell’utente;
  - `email`: identificativo unico, usato per login e associazioni;
  - `password`: hash sicuro della password;
  - `preferiti`: array contenente gli `ObjectId` delle carte preferite, stabilendo così la relazione molti-a-molti tra utenti e carte.

La relazione tra utenti e carte è quindi rappresentata tramite l’array `preferiti` all’interno del documento utente, che mantiene i riferimenti alle carte salvate. Questa struttura permette di eseguire query efficienti per recuperare le carte preferite di un utente, anche grazie a operazioni di *JOIN* simulate lato backend tramite filtri `$in` in MongoDB.

## 2.4 Funzionalità principali

Il sistema offre un set di funzionalità essenziali per la gestione degli utenti, la consultazione delle carte e la personalizzazione della collezione personale, riassunte come segue:

- **Autenticazione e registrazione sicura:** gli utenti possono creare un account fornendo nome, email e password. La password viene memorizzata in forma hashata tramite la libreria `bcrypt`, garantendo la sicurezza dei dati sensibili. Il login consente di accedere alle funzionalità riservate agli utenti registrati.
- **Gestione CRUD degli utenti e preferiti:** gli utenti registrati possono aggiornare i propri dati personali, modificare la password e gestire la lista delle carte preferite, aggiungendo o rimuovendo elementi. Le operazioni di creazione, lettura, aggiornamento e cancellazione (CRUD) sono implementate nel backend con API REST dedicate.
- **Consultazione aperta delle carte:** tutti gli utenti, anche quelli non registrati, possono esplorare il catalogo completo delle carte, visualizzare i dettagli specifici di ciascuna carta e utilizzare filtri di ricerca avanzati per facilitare la navigazione.
- **Join tra collection:** per ottenere i dettagli completi delle carte salvate nei preferiti di un utente, il backend esegue una query che effettua una JOIN tra la collection `users` e la collection `carte`, utilizzando gli `ObjectId` memorizzati nell’array `preferiti`. Ciò permette di restituire un elenco dettagliato delle carte preferite in un’unica chiamata API.

## 2.5 Proprietà BASE e teorema CAP

Nel progetto si è scelto di adottare un modello NoSQL basato su MongoDB, che enfatizza le proprietà BASE, garantendo un bilanciamento tra disponibilità e consistenza adatto al dominio applicativo.

- **Basically Available:** il sistema assicura che le operazioni di lettura e scrittura siano sempre disponibili, garantendo risposte rapide anche in caso di carico elevato o parziale indisponibilità di nodi del database.
- **Soft state:** i dati possono temporaneamente trovarsi in uno stato non completamente consistente, in attesa che le modifiche si propaghino attraverso il sistema.
- **Eventual consistency:** il sistema si impegna a garantire che, dopo un certo intervallo di tempo, tutte le repliche dei dati convergano verso uno stato consistente, tollerando quindi una temporanea incoerenza tra i dati letti da client diversi.

Relativamente al teorema CAP (Consistency, Availability, Partition tolerance), il progetto pone l'accento su due dei tre vincoli:

- **Availability:** il sistema è progettato per rispondere sempre alle richieste degli utenti, garantendo l'accesso continuo alle funzionalità, anche in presenza di guasti o ritardi di rete.
- **Partition tolerance:** grazie all'architettura distribuita di MongoDB, il sistema è in grado di funzionare anche in caso di partizioni di rete, evitando interruzioni del servizio.

Per mantenere la coerenza tra le collection `users` e `carte`, in particolare nella gestione dei preferiti, si è optato per l'esecuzione di JOIN lato backend. Questo consente di evitare incoerenze dovute a richieste separate, migliorando la consistenza percepita dall'utente finale, pur rispettando le caratteristiche BASE e i compromessi indicati dal teorema CAP.

## 3 Sviluppo

### 3.1 Backend

Il backend dell'applicazione è sviluppato utilizzando **Flask**, un framework Python leggero e flessibile, abbinato a **MongoDB** come database NoSQL. La comunicazione con il database avviene tramite **Flask-PyMongo**, che facilita le operazioni CRUD con documenti JSON.

#### 3.1.1 Popolamento del Database

Il dataset contenente l'intera lista delle carte è stato reperito da Kaggle in formato CSV. Prima di procedere al popolamento del database MongoDB, è stata necessaria una fase preliminare di pulizia dei dati. In particolare, i valori `NaN` sono stati sostituiti con `None`, in modo che venissero correttamente convertiti in `null` durante la serializzazione in formato JSON. Per gestire questa fase di pulizia e l'importazione dei dati nel database, è stato sviluppato lo script `import_cards.py`, il quale deve essere eseguito una tantum prima dell'avvio del server Flask.

#### 3.1.2 Definizione delle query

La Tabella 2 riassume le principali query offerte dal backend Flask, progettato secondo l'architettura RESTful. Le operazioni sono suddivise in base al modello CRUD (Create, Read, Update, Delete), associato alle relative entità del sistema, cioè alle carte e agli utenti. Ogni endpoint consente l'interazione con una specifica risorsa, permettendo ad esempio di recuperare le informazioni su tutte le carte o su una specifica carta, gestire l'autenticazione degli utenti, aggiornare il profilo o mantenere una lista di carte preferite. Le query sono pensate per garantire un'interazione chiara, modulare ed estensibile tra il frontend e il backend dell'applicazione, permettendo al client di richiedere, modificare o eliminare dati in modo sicuro.

Metodo	Endpoint	Azione	Entità	Descrizione
GET	/carte	Read (R)	Carte	Recupera tutte le carte, con limite opzionale
GET	/carte/upvotes	Read (R)	Carte	Recupera carte ordinate per upvotes (top carte)
GET	/carta/{id}	Read (R)	Carte	Recupera una singola carta per ID
POST	/signup	Create (C)	Utente	Registra un nuovo utente
POST	/login	Read (R)	Utente	Effettua il login e verifica le credenziali
PUT	/utente/{email}	Update (U)	Utente	Aggiorna nome e/o password utente
GET	/preferiti/{email}	Read (R)	Utente	Recupera gli ID delle carte preferite di un utente
POST	/preferiti/{email}	Create (C)	Utente	Aggiunge una carta ai preferiti dell'utente
DELETE	/preferiti/{email}	Delete (D)	Utente	Rimuove una carta dai preferiti dell'utente
GET	/preferiti/dettagli/{email}	Read (R)	Carte (join)	Recupera i dettagli completi delle carte preferite

Tabella 2: Mappa delle operazioni CRUD del backend Yu-Gi-Oh!

## 3.2 Frontend

Il frontend dell'applicazione rappresenta l'interfaccia utente attraverso la quale l'utente finale può interagire con le funzionalità offerte dal sistema. Esso si occupa della visualizzazione e dell'interazione con i dati, fungendo da ponte tra l'utente e le API RESTful esposte dal backend. L'interfaccia è progettata per essere intuitiva, reattiva e accessibile, offrendo una navigazione fluida tra le varie sezioni: esplorazione delle carte, registrazione e login degli utenti, gestione del profilo e dei preferiti. La comunicazione con il backend avviene tramite chiamate HTTP asincrone, che permettono di recuperare e manipolare i dati in tempo reale, senza la necessità di ricaricare la pagina. Questo approccio, tipico delle web app moderne, migliora notevolmente l'esperienza utente.

### 3.2.1 Pagina Home

La pagina *Home* rappresenta la schermata principale dell'applicazione, offrendo un'introduzione visiva e funzionale agli utenti. Il design prevede un carosello dinamico di tre slide che presentano informazioni chiave e inviti all'azione, come l'esplorazione del catalogo delle carte o la registrazione per accedere alle funzionalità personalizzate. Ogni slide è composta da un titolo, una breve descrizione, un'immagine rappresentativa e, quando previsto, un link di navigazione verso altre pagine dell'applicazione (ad esempio la pagina del catalogo o del profilo utente).

Sotto il carosello, viene mostrata una selezione di carte “in evidenza”, ottenute tramite una chiamata API al backend che recupera le carte ordinate per popolarità (upvotes). Le carte sono visualizzate in una griglia flessibile, ciascuna con nome, immagine e attributi principali come attacco, difesa e livello, consentendo agli utenti di accedere rapidamente ai dettagli specifici di ogni carta tramite un collegamento dedicato.

La pagina gestisce inoltre la personalizzazione della navigazione in base allo stato di autenticazione dell'utente, modificando dinamicamente i link disponibili nel carosello (ad esempio, indirizzando al profilo se l'utente è loggato o alla pagina di registrazione altrimenti).



Figura 1: Homepage della web application.

### 3.2.2 Catalogo

La pagina *Catalogo* offre una panoramica completa delle carte disponibili, permettendo all’utente di esplorarle e filtrarle secondo diversi criteri. Le carte sono mostrate in un layout a griglia, ciascuna con nome, immagine, tipo, attributo, livello, attacco (ATK) e difesa (DEF), e ogni carta è un link che porta alla pagina di dettaglio corrispondente. La pagina implementa un sistema di filtri avanzato e una paginazione per migliorare l’esperienza di navigazione all’interno di un dataset molto ampio. Per gestire la visualizzazione di molte carte, il catalogo utilizza una paginazione con 10 carte per pagina. La paginazione è a gruppi di 5 pagine, con pulsanti per navigare avanti e indietro tra le pagine e la possibilità di inserire direttamente il numero di pagina desiderata.

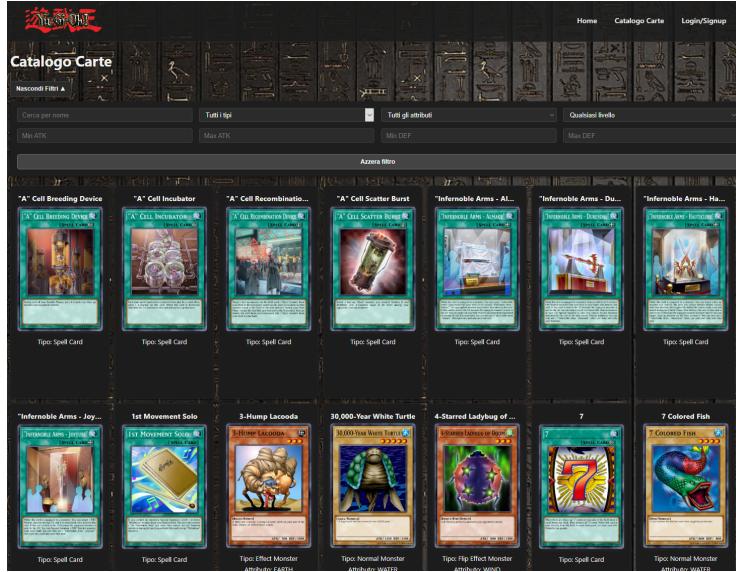


Figura 2: Pagina del Catalogo, in cui vengono mostrati anche i Filtri.

**Filtri disponibili** I filtri sono raggruppati in due righe e consentono di restringere la ricerca delle carte in base ai seguenti parametri:

- **Nome:** un campo di ricerca testuale che permette di cercare carte il cui nome contenga la stringa inserita (case insensitive).
- **Tipo:** un menu a tendina che permette di selezionare il tipo di carta (ad esempio *Mostro*, *Magia*, ecc.). Sono mostrati solo i tipi presenti nel dataset.
- **Attributo:** un menu a tendina per filtrare le carte in base all’attributo (ad esempio *Fuoco*, *Acqua*, ecc.), anch’essi dinamicamente derivati dal dataset.
- **Livello:** un menu a tendina che consente di selezionare il livello delle carte (numerico).
- **ATK:** due campi numerici che definiscono un intervallo di valori di attacco minimo e massimo.
- **DEF:** analogamente, due campi numerici che definiscono un intervallo di valori di difesa minimo e massimo.

È possibile combinare più filtri contemporaneamente per una ricerca più precisa. Un pulsante *Azzera filtro* consente di riportare tutti i filtri ai valori di default (cioè senza filtri attivi). Lo stato dei filtri e della pagina corrente è sincronizzato con la query string dell'URL, garantendo che la pagina sia linkabile e che i filtri possano essere condivisi o mantenuti al refresh della pagina.

### 3.2.3 Carta

La pagina *Carta* visualizza le informazioni dettagliate su una singola carta selezionata nel catalogo. All'inizializzazione, la pagina effettua una chiamata `fetch` all'endpoint backend per ottenere i dati completi della carta tramite il suo identificatore univoco (ID) estratto dai parametri della route. Durante il caricamento viene mostrato un messaggio *Caricamento....*. In caso di errore o ID non valido, viene visualizzato un messaggio di carta non trovata. La

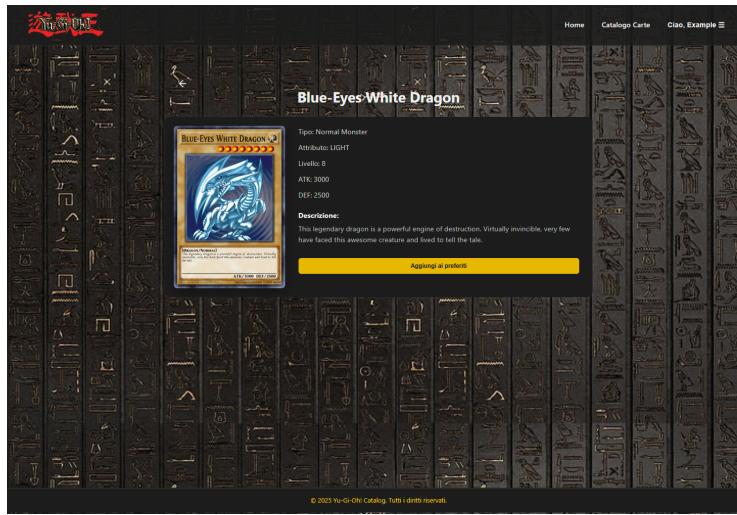


Figura 3: Pagina Carta, in cui vengono mostrati i dettagli delle singole carte.

pagina mostra:

- Nome della carta.
- Immagine con effetto di rotazione e ombra dinamica al passaggio del mouse (gestito da eventi `onMouseMove` e `onMouseLeave`).
- Tipo, attributo, livello, attacco (ATK) e difesa (DEF) con valori di fallback quando non disponibili.
- Descrizione testuale della carta.

Se l'utente è autenticato (email disponibile tramite prop `user`), la pagina consente di aggiungere o rimuovere la carta dai preferiti. Al caricamento della pagina, viene recuperata la lista dei preferiti dell'utente e verificato se la carta corrente è già tra questi. Inoltre, vengono gestiti stati di caricamento e possibili errori relativi ai preferiti. I pulsanti *Aggiungi ai preferiti* o *Rimuovi dai preferiti* permettono di modificare la lista, tramite chiamate POST e DELETE verso l'API. Se l'utente non è loggato, viene mostrato un messaggio che invita al login. Un pulsante con un'icona di freccia indietro, posizionato in alto, permette di tornare

alla pagina precedente nella cronologia di navigazione usando `navigate(-1)`. Questo aiuta l'utente a ritornare facilmente al catalogo o alla pagina da cui proveniva.

### 3.2.4 Login

La pagina di *Login* permette all'utente di autenticarsi inserendo email e password. Il componente React utilizza lo stato locale per gestire i valori di email, password, la visibilità della password e i messaggi di errore. Al submit del form, viene effettuata una richiesta POST all'endpoint `/login` inviando un JSON con email e password. Se la risposta è positiva, la funzione `onLogin` aggiorna lo stato globale dell'applicazione con i dati utente e si effettua il reindirizzamento alla home tramite React Router. In caso di errore, viene mostrato un messaggio di errore appropriato. La password può essere mostrata o nascosta tramite un pulsante toggle che ne modifica il tipo di input, migliorando l'usabilità. La pagina include inoltre un link alla pagina di registrazione per gli utenti non ancora iscritti.

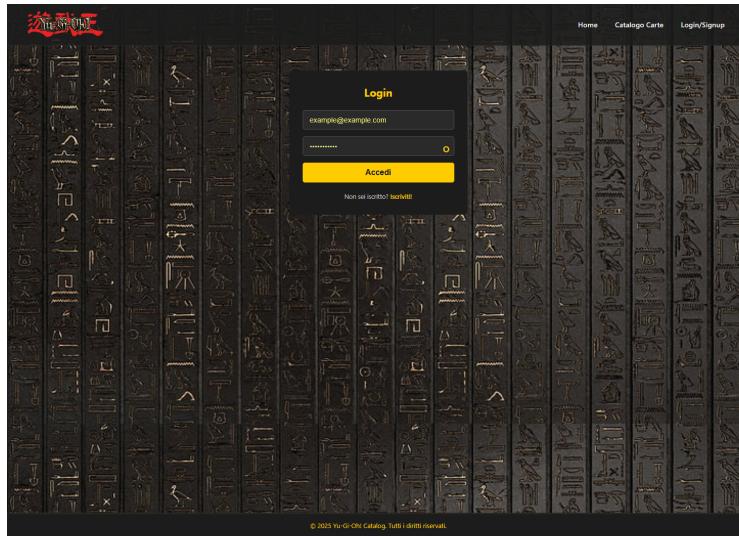


Figura 4: Pagina di Login, in cui viene mostrato il form di autenticazione.

### 3.2.5 Signup

La pagina di *Singup* consente agli utenti di creare un nuovo account inserendo nome, email, password e conferma password. Il componente React gestisce lo stato dei campi e i messaggi di errore tramite `useState`. La password deve rispettare una validazione minima: almeno 6 caratteri, con almeno una lettera e un numero, controllata tramite espressione regolare. Se le password non corrispondono, viene mostrato un messaggio di errore. Alla sottomissione del form, viene inviata una richiesta POST all'endpoint `/signup` con i dati utente in formato JSON. In caso di successo, il modulo si resetta, mostra un messaggio di conferma e, dopo un breve delay, l'utente viene reindirizzato alla pagina di login. Il componente include due campi password con pulsanti toggle per mostrare o nascondere la password. Infine, un link permette di passare rapidamente alla pagina di login per gli utenti già registrati.

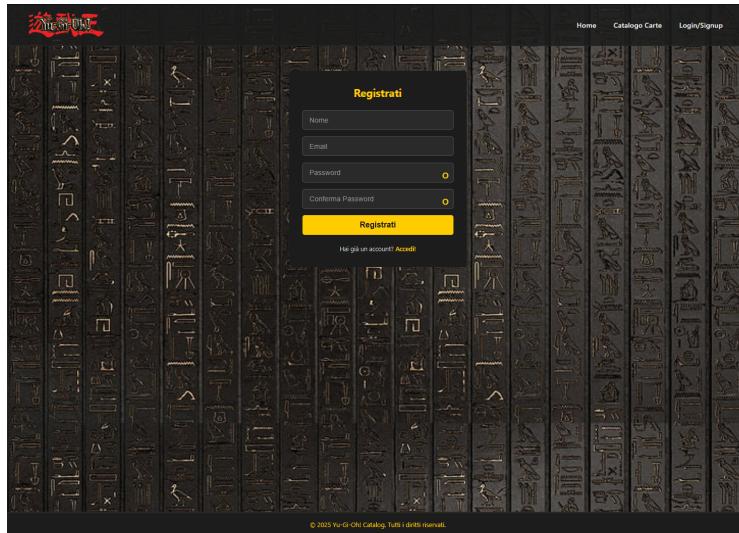


Figura 5: Pagina di Signup, in cui viene mostrato il form di registrazione.

### 3.2.6 Profilo

Il componente React *Profilo* permette all’utente autenticato di visualizzare e modificare i propri dati, inclusi nome e password. Il campo email è mostrato ma non modificabile. Il form gestisce lo stato locale dei campi tramite `useState`, con toggle per mostrare/nascondere ogni campo password per migliorare l’usabilità. La modifica della password richiede l’inserimento della vecchia password, verifica la corrispondenza tra nuova password e conferma, e la lunghezza minima di 6 caratteri. Al submit, i dati modificati sono inviati con una richiesta PUT all’endpoint `/utente/{email}`, aggiornando il profilo sul backend. In caso di successo, viene mostrato un messaggio di conferma, i campi password vengono resettati e il profilo aggiornato è passato al componente genitore tramite `onUpdate`. Eventuali errori di validazione o di rete sono mostrati all’utente.

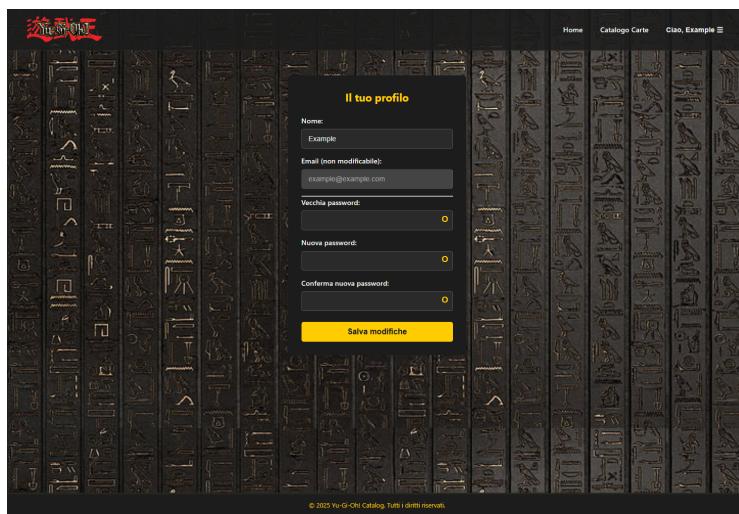


Figura 6: Pagina del Profilo, in cui vengono mostrati i form per modificare nome utente e password.

### 3.2.7 Preferiti

Il componente **Preferiti** consente all'utente autenticato di visualizzare l'elenco delle carte contrassegnate come preferite. Al caricamento del componente, tramite lo `useEffect`, viene inviata una richiesta GET all'endpoint `/preferiti/dettagli/{email}` per recuperare i dettagli completi delle carte preferite associate all'utente corrente. I dati vengono salvati nello stato locale `useState`. Se l'utente non è autenticato, viene mostrato un messaggio che invita al login. In caso di assenza di preferiti, il componente mostra un messaggio dedicato, mentre durante il caricamento è visibile un messaggio di attesa. Le carte preferite vengono visualizzate come card cliccabili (link alla pagina della singola carta) con nome, immagine, tipo, attributo, livello, attacco e difesa, se disponibili. La struttura delle card riutilizza lo stile definito per la pagina Catalogo.

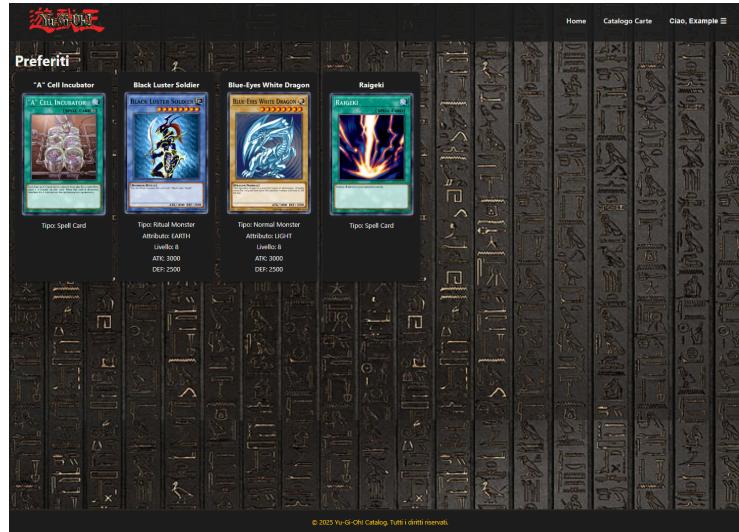


Figura 7: Pagina dei Preferiti, in cui vengono mostrate le carte aggiunte all'array `preferiti` dall'utente autenticato.

### 3.2.8 Navbar e Footer

Il componente **Navbar** rappresenta l'intestazione del sito e fornisce la navigazione principale. Include il logo cliccabile, i link a `Home` e `Catalogo Carte`, e un pulsante che varia in base allo stato dell'utente: se non autenticato, viene mostrato il link a `Login/Signup`, altrimenti un pulsante che apre un menu laterale (sidebar) con le voci `Profilo`, `Carte preferite` e `Logout`. Il logout viene gestito tramite la funzione `onLogout` e un redirect alla homepage. Il menu laterale può essere chiuso cliccando all'esterno o sul pulsante `×`.

Il componente **Footer** è invece fisso in fondo alla pagina e mostra un semplice messaggio con il nome del progetto e l'anno corrente, generato dinamicamente tramite `new Date().getFullYear()`.