

Relazione Progetto o web e mobile

Analisi dei requisiti

Il progetto Album delle Figurine dei personaggi di Hogwars è un'applicazione web che permette agli utenti di collezionare figurine elettroniche dei vari personaggi attraverso l'acquisto di pacchetti virtuali e facilita lo scambio di figurine tra utenti. Di seguito è presentata un'analisi dei requisiti funzionali e non funzionali.

Requisiti Funzionali

1. Gestione Utenti

- **Registrazione utente**
 - Raccolta dati personali: nome utente, email, password, mago preferito
 - Validazione delle informazioni inserite
 - Creazione dell'account utente
- **Autenticazione**
 - Login con credenziali
- **Gestione profilo**
 - Visualizzazione e modifica dei dati personali
 - Possibilità di eliminare il profilo

2. Sistema di Crediti

- **Acquisto crediti**
 - Simulazione di acquisto crediti virtuali
 - Visualizzazione del saldo attuale
- **Utilizzo crediti**
 - Spesa di crediti per l'acquisto di pacchetti (1 credito = 1 pacchetto)

3. Gestione Figurine e Pacchetti

- **Acquisto pacchetti**
 - Generazione di 5 figurine casuali per pacchetto
 - Visualizzazione delle figurine ottenute
- **Album personale**
 - Visualizzazione delle figurine possedute
 - Organizzazione delle figurine nell'album
 - Identificazione delle figurine doppie
- **Informazioni personaggi**
 - Visualizzazione dati di base: nome, descrizione, immagine
 - Visualizzazione dati avanzati: nomi alternativi, bacchetta, caratteristiche fisiche

4. Sistema di Scambio

- **Proposta di scambio**
 - Offerta di figurine doppie in cambio di figurine mancanti
 - Pubblicazione della proposta nella piattaforma
- **Gestione delle proposte**
 - Visualizzazione delle proposte disponibili
 - Accettazione delle proposte
 - Cancellazione delle proposte
 - Completamento dello scambio

5. Funzionalità Aggiuntive (almeno 3 da implementare)

- **Scambi complessi**
 - Possibilità di scambiare più figurine contemporaneamente (es. due per una)
- **Vendita figurine**

- Monetizzazione delle figurine in cambio di crediti
- **Utente amministratore**
 - Gestione di pacchetti speciali (es. 1.5 crediti per 9 figurine)
- **Controlli di integrità**
 - Verifica che non si scambino figurine già possedute
 - Verifica che non si offrano figurine doppie nella stessa proposta

Requisiti Non Funzionali

Interfaccia Utente

- **Usabilità:** interfaccia intuitiva e responsive
- **Separazione struttura/presentazione:** uso di HTML5 per la struttura e CSS3 per la presentazione
- **Interattività:** funzionalità dinamiche con JavaScript

Architettura

- **Front-end:** sviluppo con HTML5, CSS3 e JavaScript
 - **Back-end:** sistema basato su NodeJS
 - **Database:** archiviazione dati con MongoDB
 - **API:** esposizione delle funzionalità tramite API REST
 - **Documentazione API:** implementazione di Swagger per testing e documentazione
-

Funzionalità da sviluppare

1. Gestione Utenti

1.1 Registrazione

- Form di registrazione per raccogliere dati utente (username, email, password, personaggio preferito)
- Validazione dei dati inseriti
- Creazione del profilo utente nel database
- Conferma di registrazione avvenuta

1.2 Autenticazione

- Form di login con username/email e password
- Gestione delle sessioni utente
- Funzionalità di logout

1.3 Gestione Profilo

- Visualizzazione dei dati profilo
- Modifica dei dati personali
- Eliminazione account
- Dashboard personale con statistiche (figurine possedute, doppioni, mancanti)

2. Sistema di Crediti

2.1 Acquisto Crediti

- Interfaccia per la simulazione di acquisto crediti
- Aggiornamento del saldo utente

2.2 Gestione Saldo

- Visualizzazione del saldo corrente

3. Gestione Figurine

3.1 Pacchetti di Figurine

- Interfaccia per acquisto pacchetti
- Algoritmo di generazione figurine casuali
- Aggiornamento dell'album utente

3.2 Visualizzazione Album

- Griglia/lista di figurine possedute
- Filtri per figurine (possedute, doppie, mancanti)
- Ordinamento per vari criteri (quantità, alfabetico,)

3.3 Dettaglio Figurine

- Visualizzazione informazioni base (nome, descrizione, immagine)
- Visualizzazione informazioni estese (bacchetta, attore, vivo oppure no)
- Indicazione di possesso (singola, doppia, mancante)

3.4 Integrazione API Hogwarts

- Chiamate API per recupero dati dei magi
- Gestione delle risposte e conversione in formato utile

4. Sistema di Scambio

4.1 Creazione Proposte

- Interfaccia per selezionare figurine da offrire
- Interfaccia per selezionare figurine desiderate
- Pubblicazione della proposta di scambio

4.2 Gestione Proposte

- Bacheca delle proposte di scambio attive
- Filtri per trovare scambi consigliati
- Ricerca per personaggio specifico

4.3 Processo di Scambio

- Visualizzazione dettaglio proposta
- Funzionalità per accettare lo scambio
- Meccanismo di trasferimento figurine tra utenti

5. Funzionalità avanzate

Ho personalmente scelto di implementare questi 3 punti:

- **Vendita figurine**
- **Scambi complessi**
- **Controlli di integrità**

6. Funzionalità di Supporto

6.1 API Backend

- Definizione endpoint per tutte le funzionalità
- Implementazione logica di business
- Documentazione API con Swagger
- Gestione errori e risposte

6.2 Sicurezza

- Protezione delle credenziali utente
 - Validazione input/output
-

Progettazione della struttura

Struttura back-end

- (Token nell'header) = Si tratta di un parametro presente nell header (Authorization: Bearer Token)

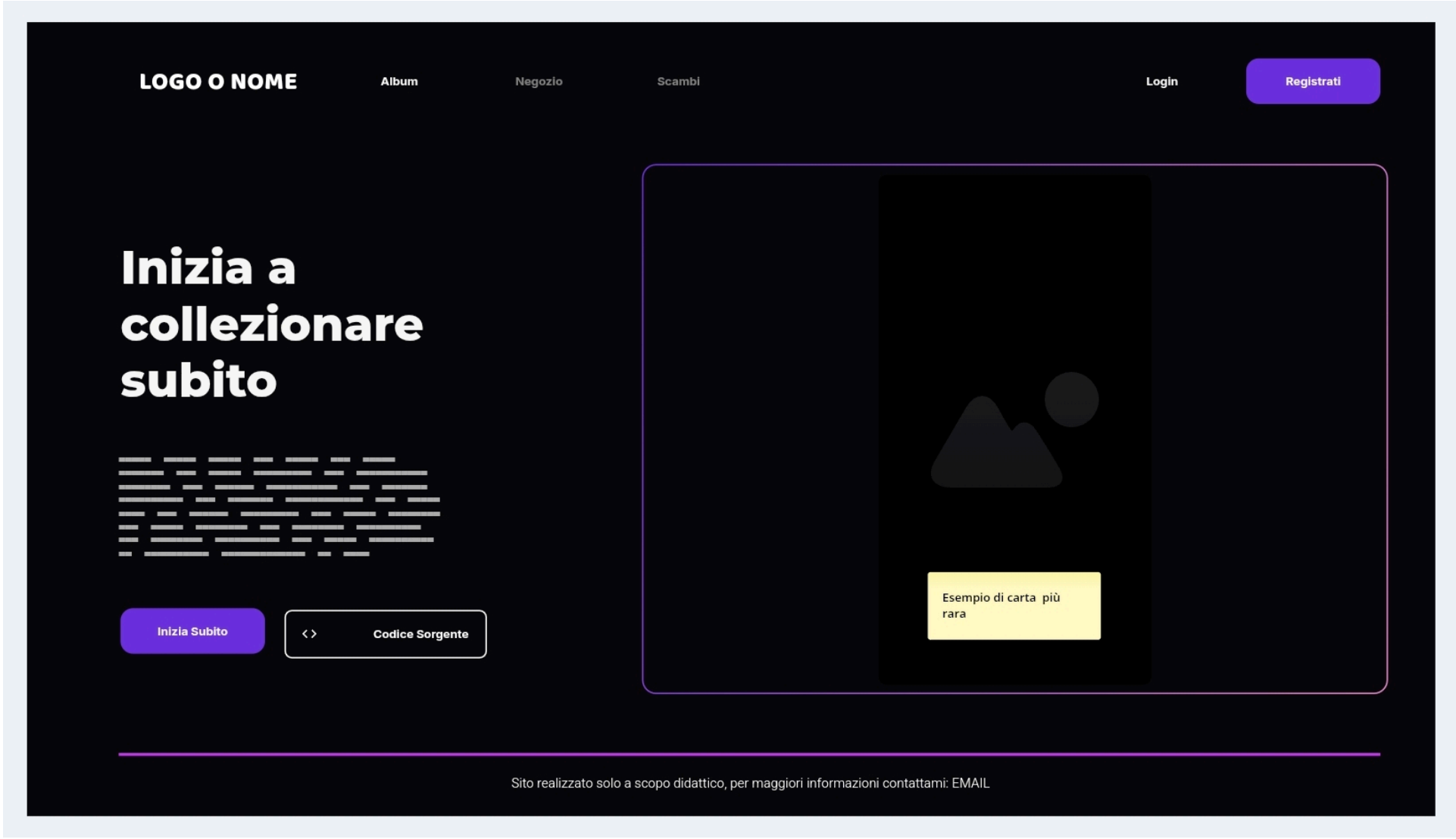
Endpoint	Tipo	Parametri	Risposta	Descrizione	Accesso senza login
/user/register	POST	username , email , favouriteWizard , password , confirmPassword	Messaggio di successo/errore	Crea un nuovo account utente.	Si
/user/login	POST	username (o email), password	Messaggio di successo e token JWT	Autentica un utente esistente e restituisce un token.	Si
/user/logout	POST	Nessuno (Token nell'header)	Messaggio di successo	Termina la sessione utente invalidando il token.	No
/user/token-status	GET	Nessuno (Token nell'header)	{ "valid": boolean }	Verifica se il token JWT fornito è ancora valido.	No
/user/info	GET	Nessuno (Token nell'header)	Oggetto con i dati dell'utente	Recupera i dati del profilo dell'utente autenticato.	No
/user/update	PUT	username , favouriteWizard	Messaggio di successo	Aggiorna i dati del profilo utente.	No
/user/delete	DELETE	Nessuno (Token nell'header)	Messaggio di successo	Elimina l'account dell'utente autenticato.	No
/user/cards/search	GET	Query params: searchQuery , sortBy , sortByAttributeName	Array di carte filtrate e ordinate	Cerca, filtra e ordina le carte possedute dall'utente.	No
/user/cards/missing	GET	Nessuno(Token nell'header)	Array di carte mancanti	Recupera la lista delle carte che l'utente non possiede.	No
/user/cards/double	GET	Nessuno(Token nell'header)	Array di carte doppie	Recupera la lista delle carte che l'utente possiede in più copie.	No
/user/cards/sell	POST	cards : Array di {id, quantity}	Messaggio e crediti guadagnati	Vende le carte specificate in cambio di crediti.	No
/user/cards/package/open	POST	quantity : Numero di pacchetti da aprire	Messaggio, nuove carte e crediti rimanenti	Apri uno o più pacchetti di carte pagando con i crediti.	No
/user/credits/buy	POST	amount : Quantità di crediti da acquistare	Messaggio e nuovo saldo crediti	Aggiunge crediti al saldo dell'utente.	No
/user/my-trades	GET	Nessuno (Token nell'header)	Array di scambi dell'utente	Recupera tutti gli scambi creati dall'utente.	No
/trade/create	POST	offeredCards , requestedCards , expireTime	Messaggio e oggetto dello scambio	Crea una nuova proposta di scambio.	No
/trade/all	GET	Nessuno (Token nell'header)	Array di tutti gli scambi aperti	Lista tutti gli scambi disponibili (esclusi quelli dell'utente).	No
/trade/search	GET	Query di ricerca con sortBy e searchQuesry (Token	Array degli scambi aperti che	Ricerca tra tutti gli scambi in base al	No

Endpoint	Tipo	Parametri	Risposta	Descrizione	Accesso senza login
		<i>nell'header)</i>	soddisfano la ricerca	nome della carta che si vole ottenere da uno scambio	
/trade/accept	POST	tradeId	Messaggio e oggetto dello scambio	Accetta una proposta di scambio esistente.	No
/trade/delete	DELETE	tradeId	Messaggio di successo	Annulla una proposta di scambio creata dall'utente.	No

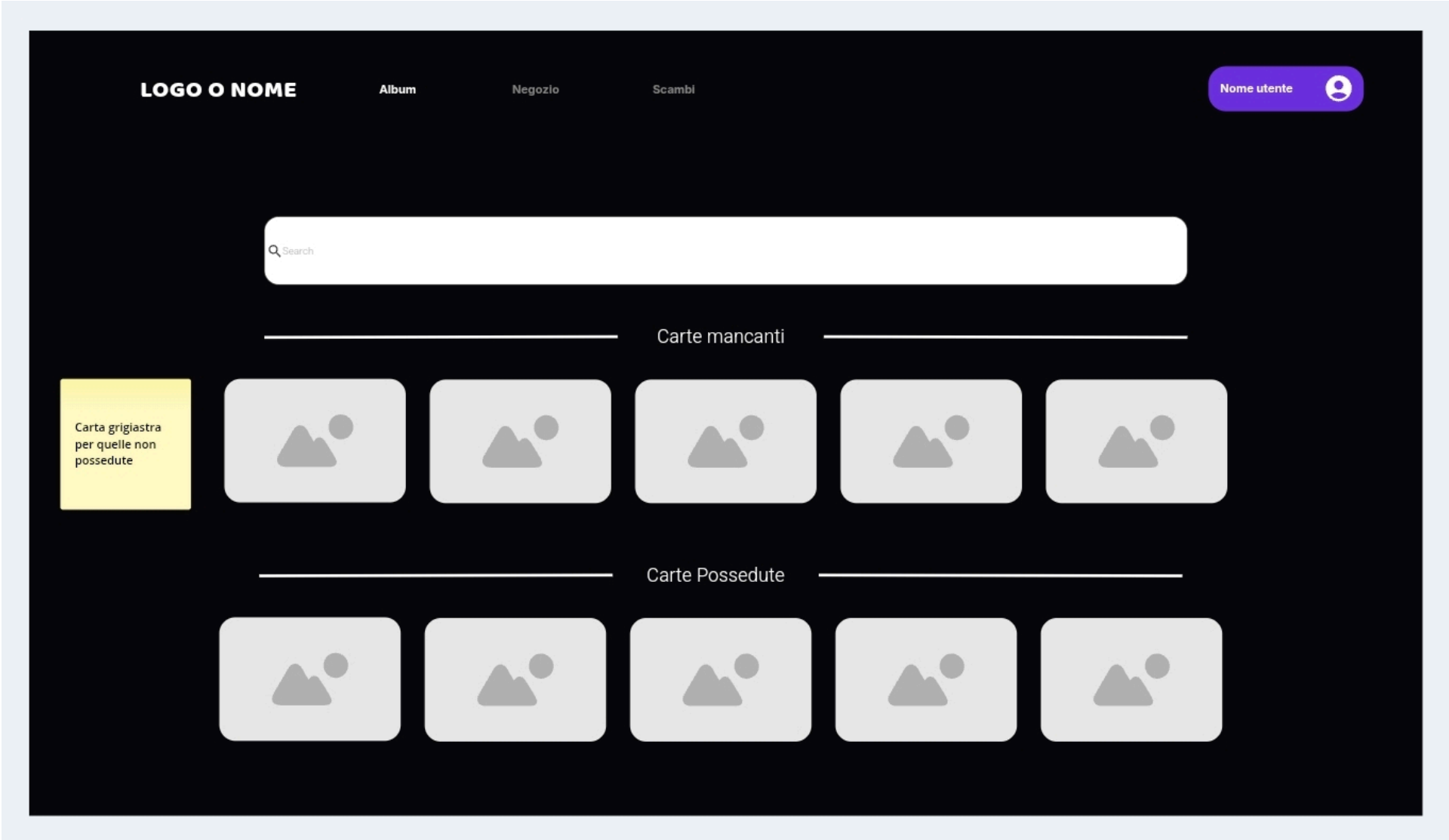
Struttura front-end

Questa è solo un mockup molto grezzo, la pagina web finale potrebbe essere differente

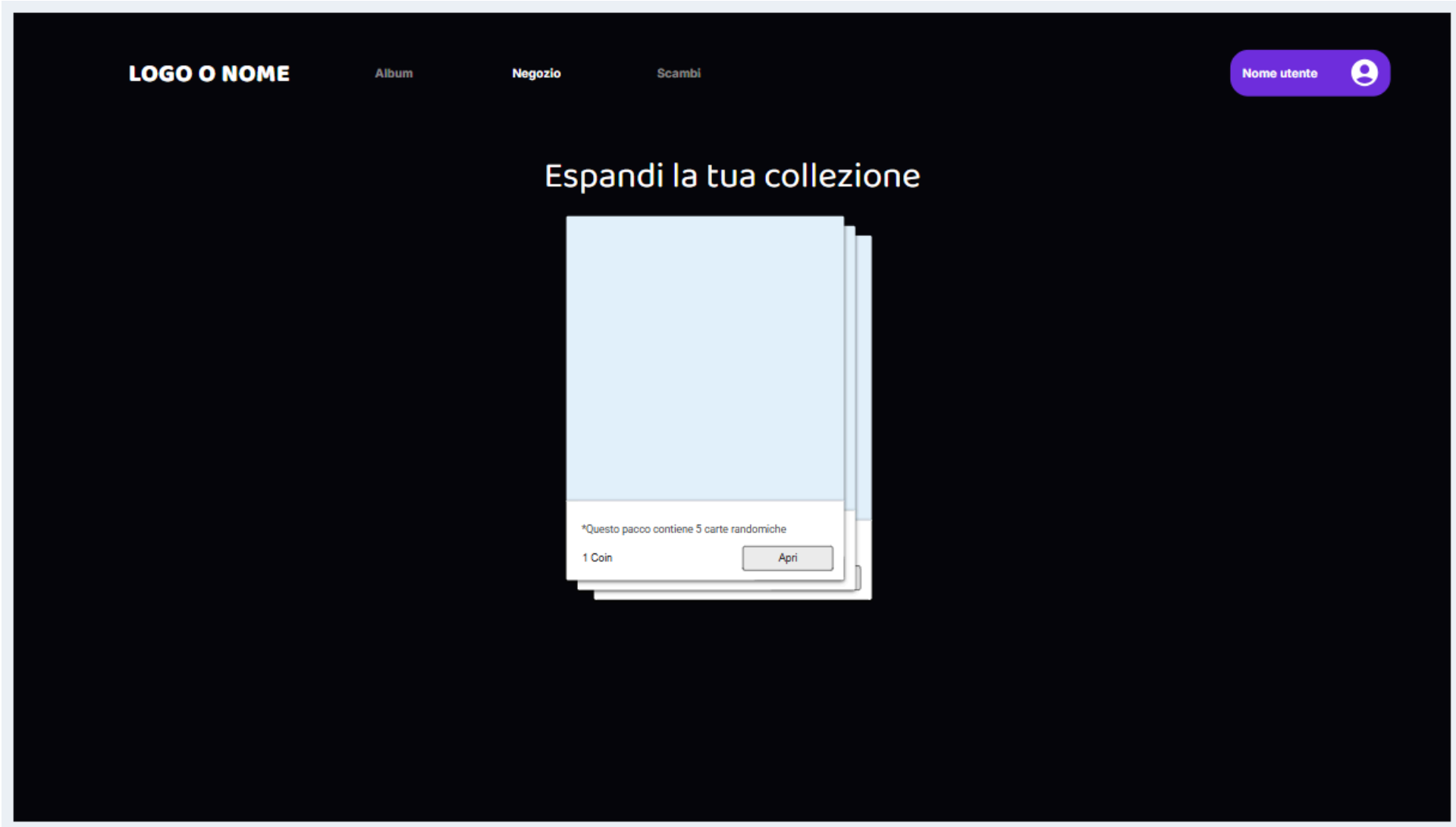
- Pagina web di presentazione:



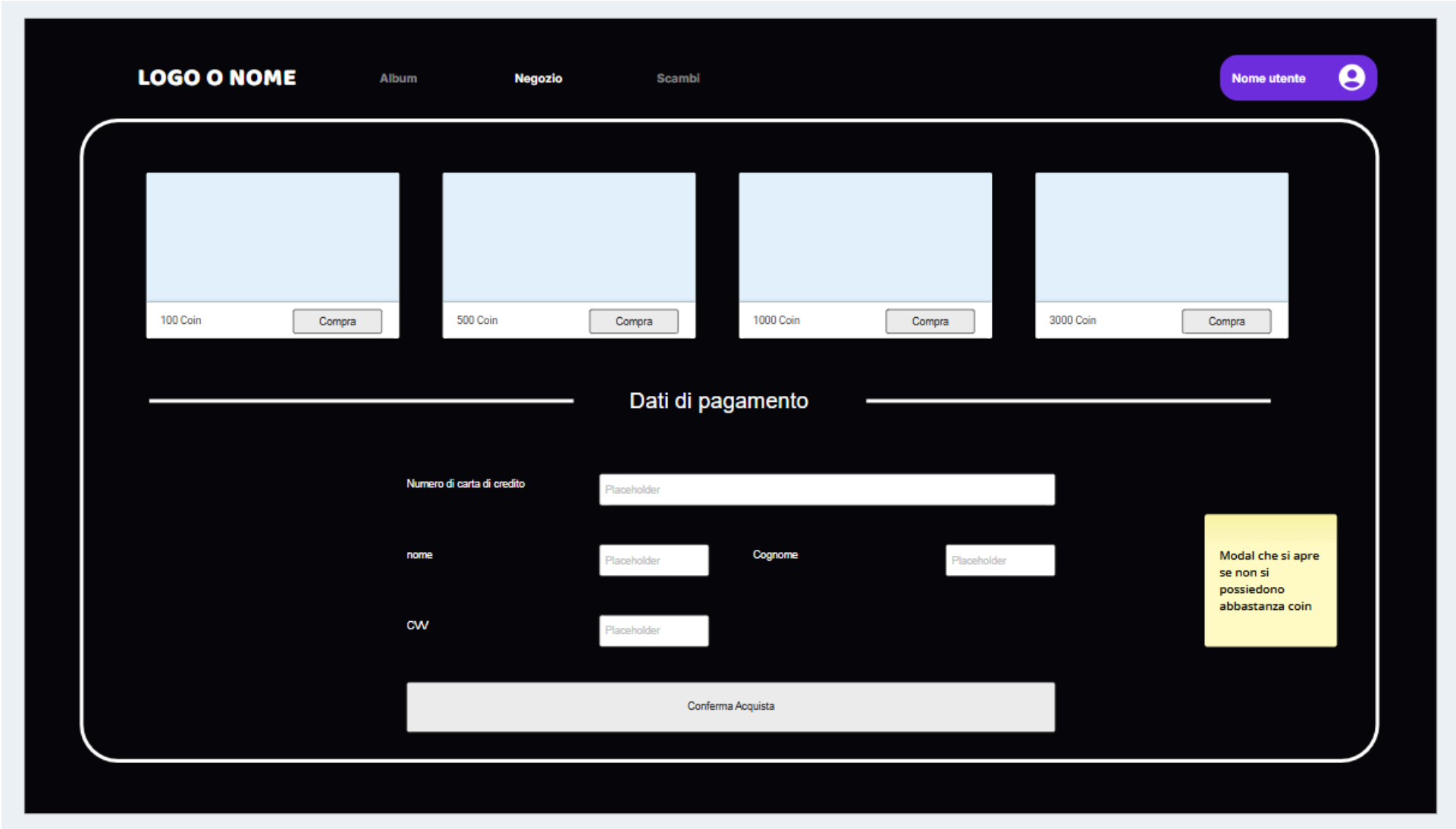
- Pagina album dopo che l'utente si e' registrato o loggato



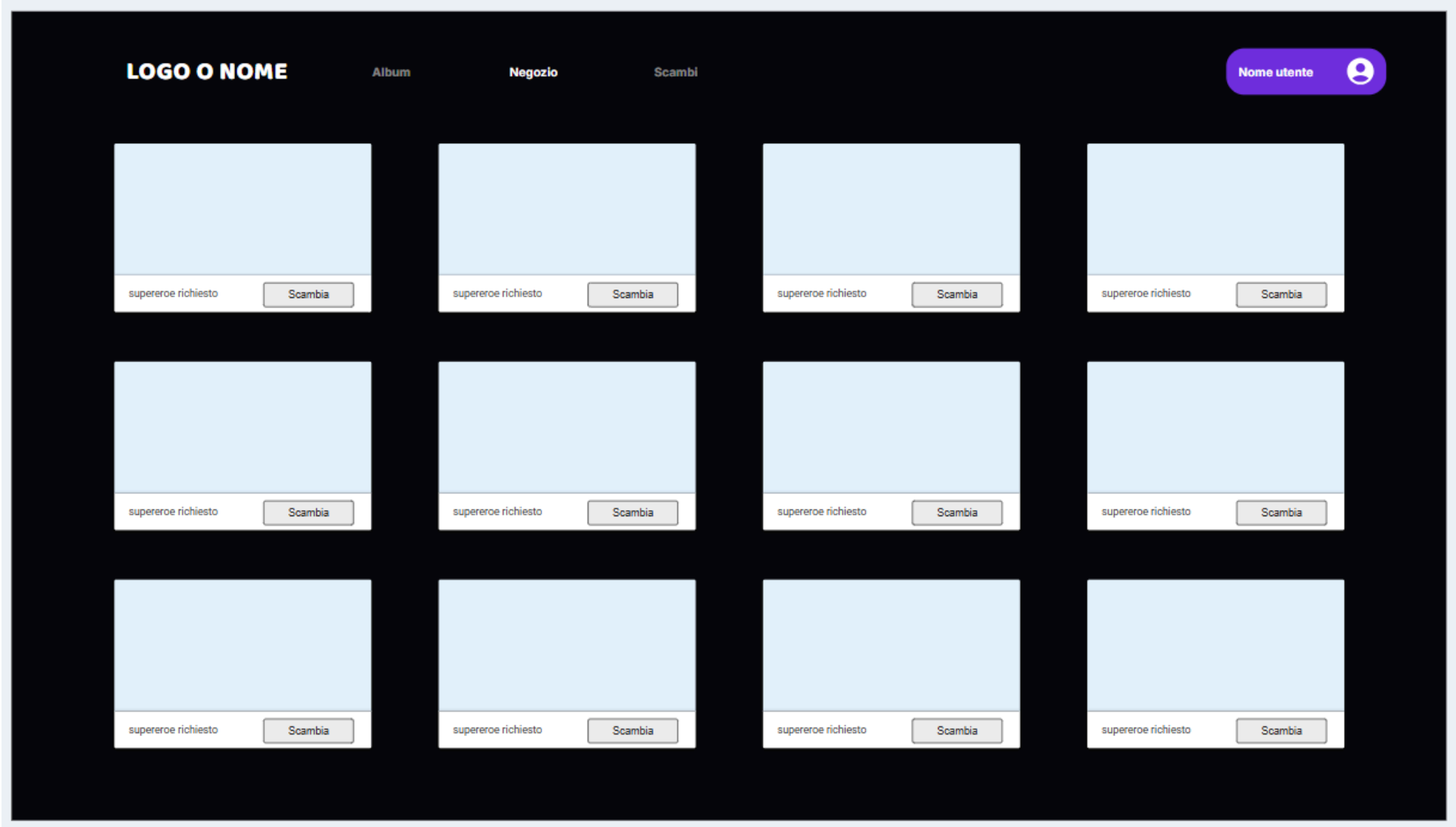
- pagina negozio



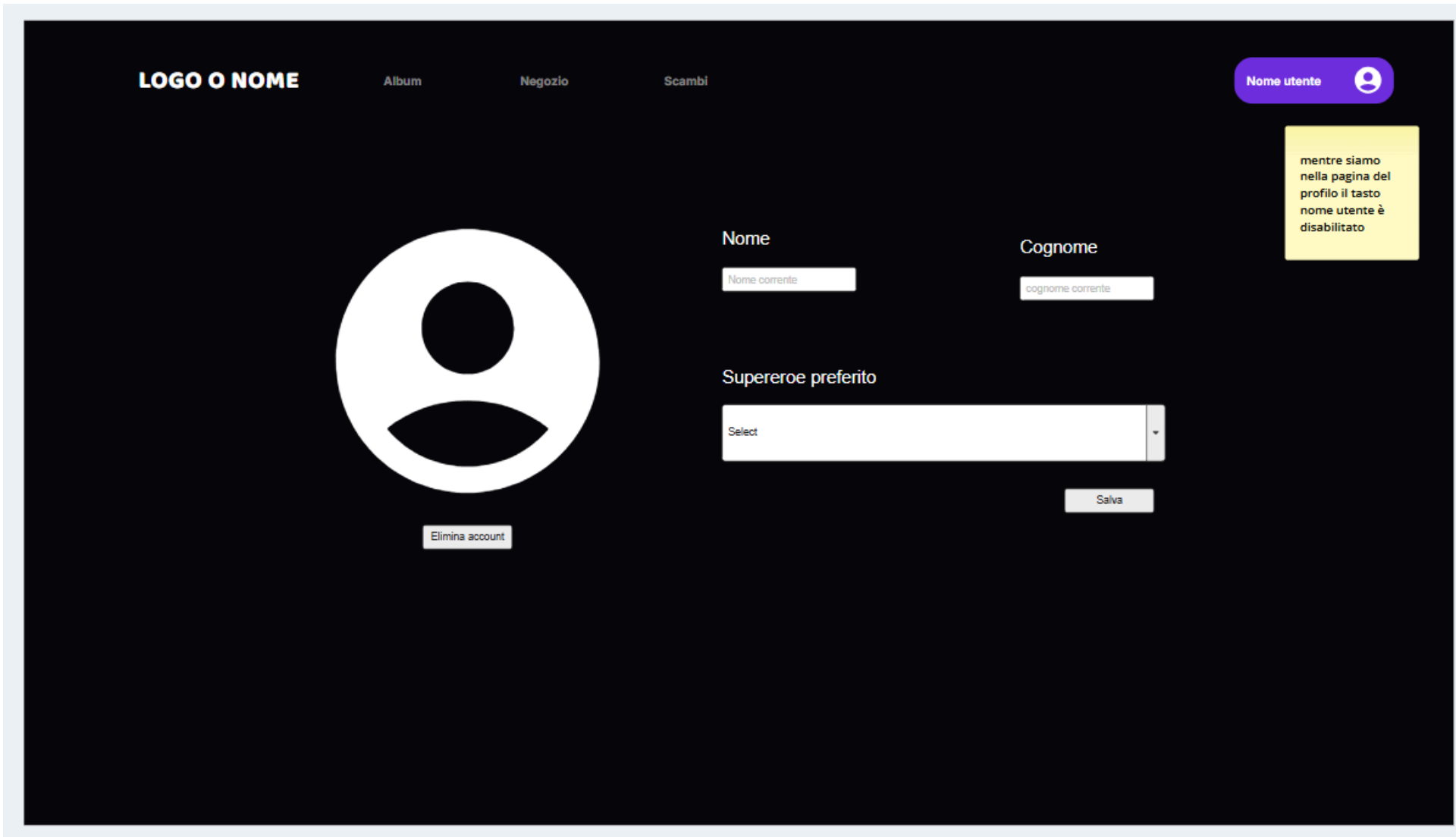
- pagina acquisto credito



- pagina scambio carte



- pagina utente



Implementazione

Tecnologie utilizzate

Front-end:

- Svelte (framework JavaScript)
- Tailwind CSS (framework CSS)
- Flowbite (libreria di componenti UI compatibile con Tailwind)

Render della pagina web:

- Approccio ibrido con sveltekit (server side rendering e client side)

Back-end:

- Node.js con Express (framework server)
- MongoDB (database NoSQL come da richiesta)
- Swagger (per la documentazione e testing delle API)
- JWT (JSON Web Token per l'autenticazione)

Strumenti di sviluppo:

- Git (controllo versione)

Perché le ho scelte?

Per realizzare questo progetto, ho scelto con le tecnologie più adatte, con l'obiettivo di costruire un'applicazione web moderna, performante e facile da mantenere, nel rispetto dei requisiti della traccia d'esame.

Front-end e Rendering:

- **Svelte e SvelteKit:** Ho optato per Svelte perché, a differenza di framework come React o Vue, non utilizza un Virtual DOM, ma compila il codice direttamente in JavaScript ottimizzato già in fase di build. Questo garantisce un'app più leggera e reattiva, ideale per un'interfaccia dinamica come quella di un album di figurine. SvelteKit, il framework ufficiale, mi offre la possibilità di sfruttare un rendering ibrido (SSR + CSR), migliorando sia l'esperienza utente con caricamenti rapidi che la SEO, nel caso l'app venisse pubblicata online.

- **Tailwind CSS e Flowbite:** Ho scelto Tailwind CSS per la sua praticità nello sviluppare interfacce personalizzate direttamente nell'HTML, senza dover scrivere CSS aggiuntivo. Questo approccio rende lo sviluppo più veloce e coerente. Insieme a Tailwind ho utilizzato Flowbite, una libreria di componenti UI basati su Tailwind, che mi ha permesso di concentrarmi sulla logica dell'app, senza perdere tempo a creare componenti da zero.

Back-end:

- **Node.js e Express:** Per uniformare il linguaggio tra front-end e back-end ho utilizzato JavaScript anche lato server, come richiesto dalla traccia. Node.js mi consente questa coerenza, mentre Express, essendo leggero e flessibile, è perfetto per costruire l'API dell'applicazione. Inoltre, grazie ai tantissimi pacchetti npm, ho avuto accesso a molti strumenti utili per la gestione della logica del progetto.
- **MongoDB:** I dati dell'applicazione (ad esempio, utenti con array di figurine o oggetti di scambio) si prestano bene a una struttura NoSQL. MongoDB, essendo un database a documenti, si adatta perfettamente a questa esigenza e alla traccia stessa. La sua flessibilità schema-less è un vantaggio per evolvere la struttura dati senza complicazioni.
- **JWT (JSON Web Token):** Per l'autenticazione ho usato i JWT, ideali in un'architettura stateless. Dopo il login, il client riceve un token che include in ogni richiesta successiva, permettendo al server di validarlo senza mantenere sessione, migliorando scalabilità e performance.
- **Swagger (OpenAPI):** Documentare l'API era fondamentale, specialmente in un contesto dove front-end e back-end sono separati. Con Swagger ho potuto generare una documentazione interattiva chiara e completa, utile sia in fase di sviluppo che per testare facilmente le rotte del back-end. Inoltre, era richiesto dalla traccia.

Strumenti di sviluppo:

- **Git:** Ho utilizzato Git per gestire il controllo di versione del progetto. Mi ha permesso di lavorare in modo ordinato, tracciare ogni modifica, tornare a versioni precedenti in caso di problemi, e gestire lo sviluppo in modo sicuro e collaborativo.

Scelte fatte durante il progetto

Gestione della sicurezza e autenticazione

1. Blacklist dei token

- Implementazione di una blacklist per i token JWT degli utenti che effettuano il logout o eliminano l'account
- Questo approccio garantisce che i token, anche se non ancora scaduti, non possano essere riutilizzati dopo il logout
- La blacklist viene periodicamente pulita dai token scaduti per ottimizzare le prestazioni

2. Validazione dei dati

- Implementazione di una rigorosa validazione dei dati sia lato client che server
- Utilizzo di middleware per la sanitizzazione degli input
- Controlli di autorizzazione su ogni endpoint per garantire l'accesso solo agli utenti autorizzati

Architettura del database

1. Schema delle collezioni

- *users*: profili, credenziali, collezione di carte di ogni utente, `id` dei trade proposti
- *trades*: proposte di scambio tra utenti con le carte offerte e quelle richieste incluso lo stato dello scambio e la scadenza
- *blacklist*: token invalidati

Ottimizzazione delle performance

1. Caching

- Implementazione di caching lato client per risorse statiche come le informazioni dell'utente e le carte che possiede, poi vengono aggiornate quando si fanno certe azioni come l'apertura di pacchetti, l'accettazione di scambi e la vendita delle carte
- Utilizzo di SvelteKit per gestire efficacemente il caching delle route

2. Lazy loading

- Caricamento progressivo delle carte e degli scambi

User Experience e User Interface

1. Feedback all'utente

- Messaggi di errore chiari e informativi che compaiono sotto forma di modali o di descrizione a campi di input
- Stato dei vari elementi chiaro come le carte già chieste in uno scambio ancora attivo, i pulsanti del login, registrazione, aggiornamento del profilo che mostrano se l'azione sia stata completata con successo o meno

2. Design Responsive

- Layout adattivo per diversi dispositivi

- Ottimizzazione dell'interfaccia per l'uso su dispositivi mobili
- Gestione efficiente dello spazio per visualizzare le carte

Modularità e manutenibilità

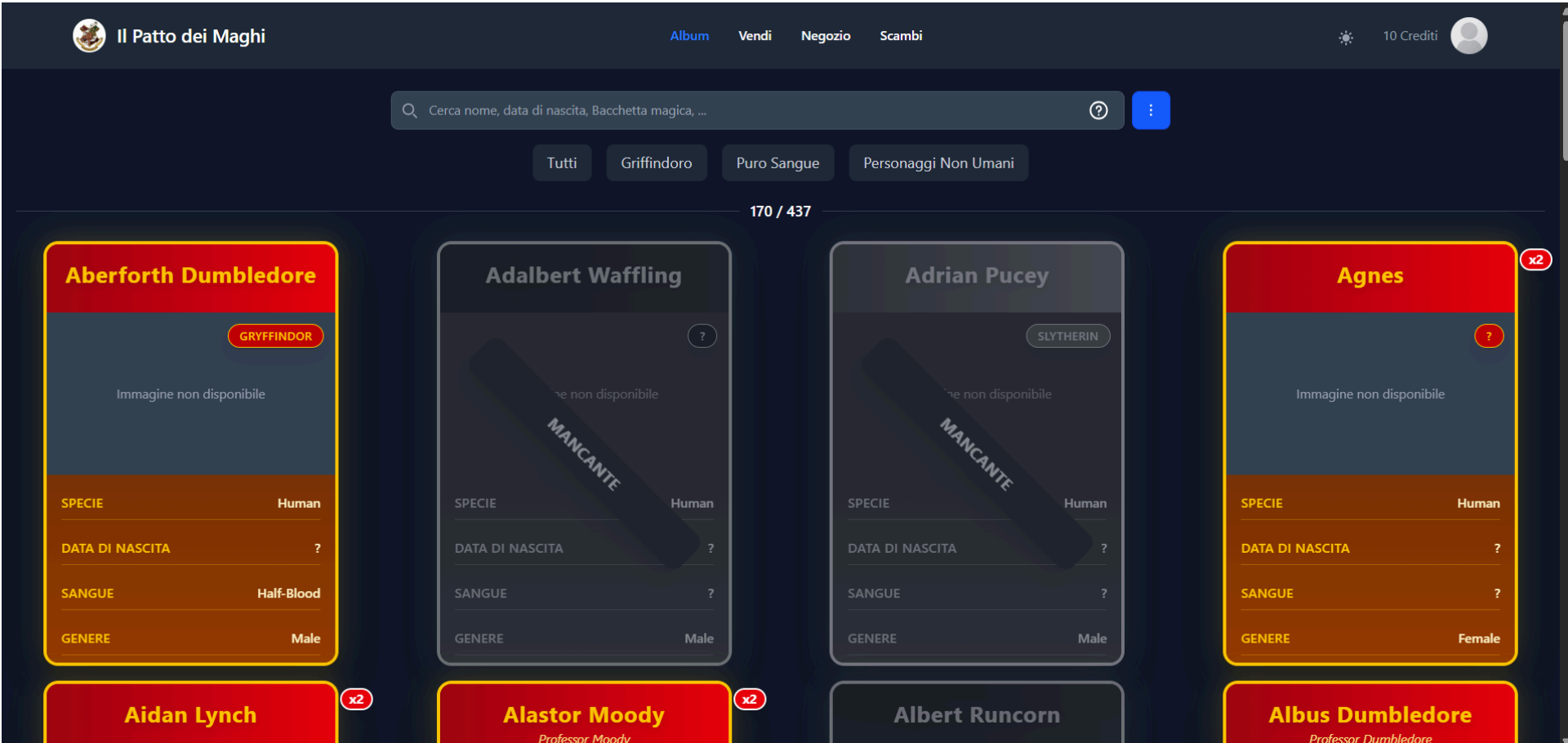
- 1. **Struttura del codice**
 - Organizzazione del codice in componenti riutilizzabili
 - Separazione chiara tra logica di business e presentazione

Risultato finale

- Pagina web di presentazione:



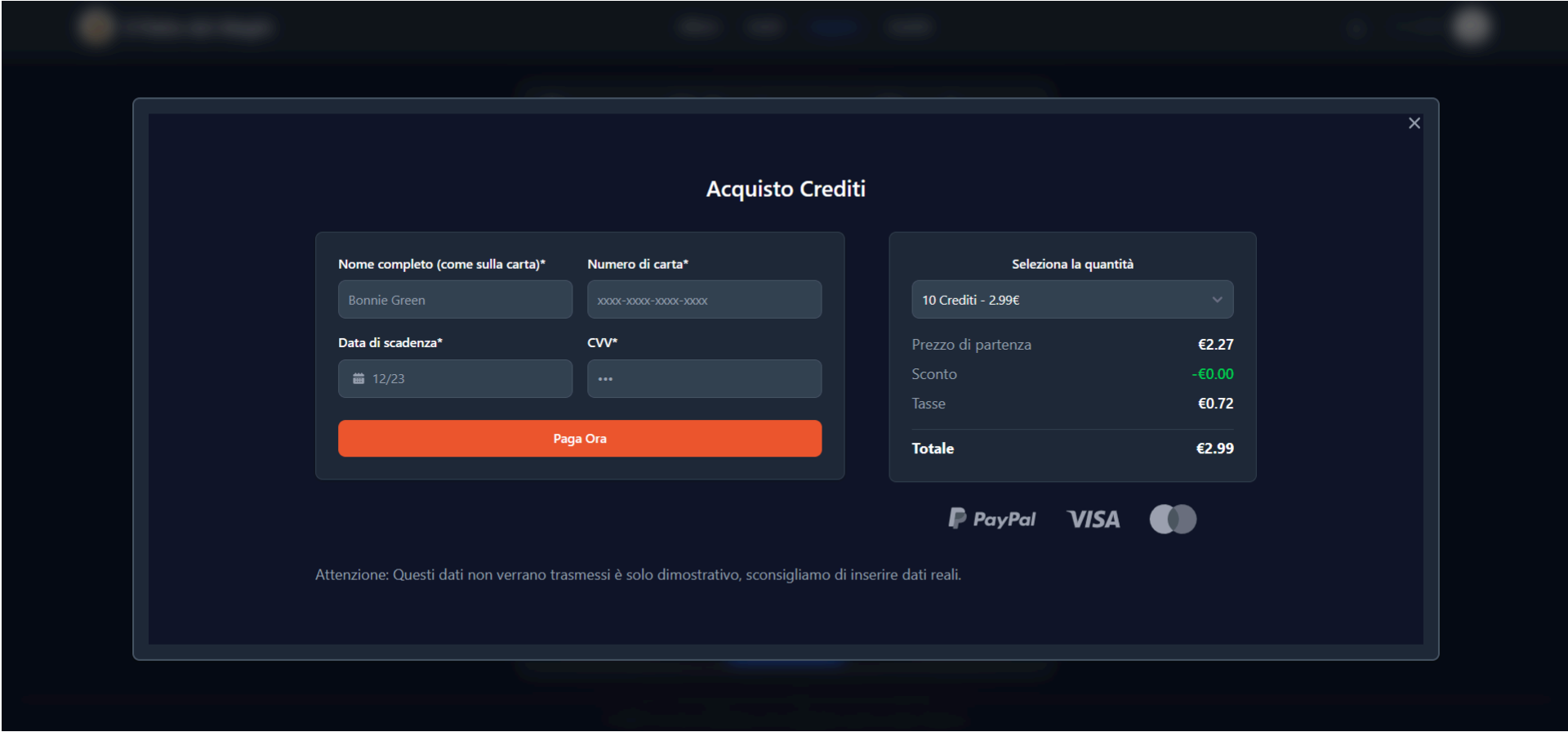
- Pagina album dopo che l'utente si e' registrato o loggato



- pagina negozio



- pagina acquisto credito



- pagina scambio carte

Il Patto dei Maghi

Album

Vendi

Negozio

Scambi

0 Crediti

Scambi Disponibili

Scopri migliaia di offerte di scambio da tutti i giocatori. Trova la carta perfetta per completare la tua collezione.

Cerca carte

Ordina per

Crea il tuo scambio

Nome carta che vuoi ricevere, nome utente...

Più recenti

+ Nuovo scambio

Filtri rapidi:

Tutti

Carte Mancanti

Più di 2 carte

Ddipi★ 0 (0 scambi)

Scade tra6 ore 39 minuti

OFFRE (2 Carte)2

CERCA (1 Carte)1

Draco Malfoy

Specie:Human

Sangue:Pure-Blood

Vivo

Lucius Malfoy

Specie:Human

Sangue:Pure-Blood


Vivo

Nuovo utente

URGENTE

Accetta scambio

- pagina profilo utente





Il Patto dei Maghi

Album

Vendi

Negozio

Scambi

 0 Crediti 

Profilo

Nome Utente

bot4328@gmail.com

Email*

bot4328@gmail.com

Mago Preferito

aaaa

Data di creazione del profilo*

20:43 il 07/06/2025

* campo non modificabile

Statistiche Carte

259

Carte totali

56

Doppioni

234

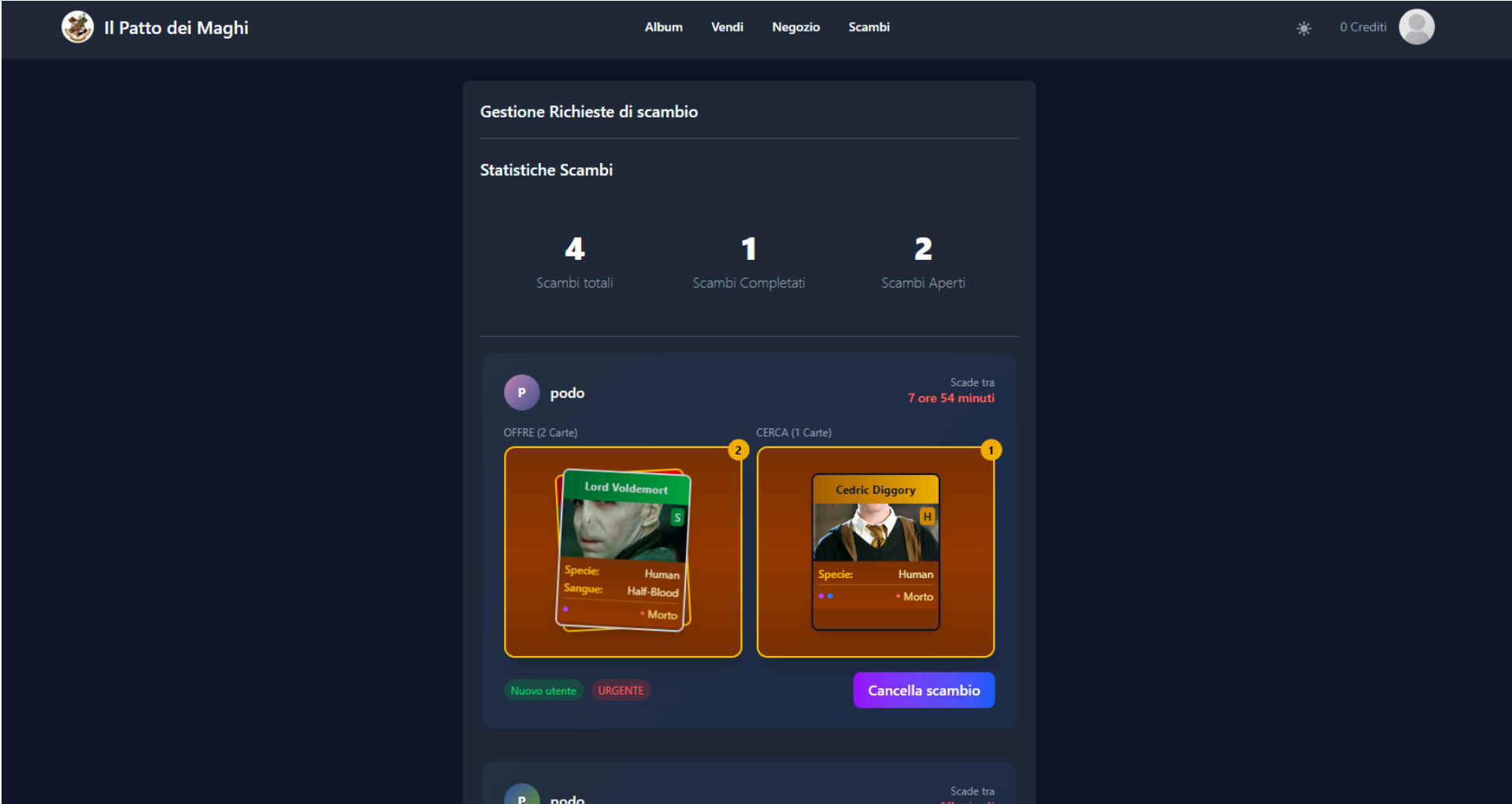
Mancanti

Aggiornato

Elimina Profilo

© 2025 Francesco Diploma, sito realizzato a puro scopo didattico

- pagina gestione scambi



[Demo video](#)

Ho incluso nella cartella `demo/` dei video che mostrano il funzionamento generale dei vari aspetti richiesti nella traccia d'esame.

[Credenziali di accesso](#)

Il progetto utilizza i file `.env` corretto per utilizzare mongoDB di Atlas, nel caso voglia utilizzare in locale è sufficiente de-commentare la sezione nell `.env` e commentare l'altra parte

Credenziali sito web utenti gia esistenti

```
1 username: "bot4328@gmail.com" (oppure "utente1")
2 password: "Ab123456"
```

```
1 username: "dipi@dipi.com" (oppure "podo12")
2 password: "Ab123456"
```

Credenziali per accerede ad mongoDB atlas

```
1 username: "tudisco2005@gmail.com"
2 password: "RFq9b8v2#t%runA"
```