Matricola: 62637A **Autore:** Donici Iustin Daniel

Marvel Universe: Album Figurine

1. Analisi

L'obiettivo è creare un'applicazione web per collezionare e gestire figurine digitali di supereroi, con due funzionalità chiave: l'acquisto di pacchetti "a sorpresa" di figurine e la possibilità di scambiare le figurine tra i giocatori. Poiché la specifica iniziale può risultare parziale e soggetta a interpretazioni, è fondamentale esporre soluzioni alternative, motivare le scelte implementative e documentare ogni fase. L'analisi si articola in tre step:

1. Raccogliere e definire i requisiti e le funzionalità;
2. Definire l'architettura e il layout delle pagine web;
3. Motivazioni e alternative valutate

1.1 Analisi dei requisiti funzionali

Requisito	Descrizione
Registrazione/Login	Creare account (username, email, password, supereroe preferito), autenticarsi e gestire profilo
Crediti	Acquistare crediti virtuali (1 credito = 1 pacchetto) e aggiornare il saldo utente
Pacchetti	Spendere 1 credito per un pacchetto da 5 figurine casuali (flag "doppione" se già possedute)
Album	Visualizzare griglia: celle piene (possedute) e placeholder (mancanti); filtro "doppioni"
Dettaglio eroe	Scheda con nome, immagine e descrizione; sezioni "Series", "Events", "Comics" caricate on-demand
Scambio base	Offrire un doppione per un supereroe, accettare proposte altrui con controlli di validità
Scambi avanzati	Offerte multicarta (più figurine offerte/richieste) e controlli su duplicati e integrità
Vendita	Mettere in vendita doppioni per crediti; acquisto trasferisce carta e aggiorna crediti
Ruolo amministratore	Creare/modificare pacchetti promozionali e consultare statistiche
Sicurezza/Performance	Password criptate, JWT, caching dei dati Marvel in client, indici MongoDB per query rapide

In particolare dobbiamo implementare le seguenti funzionalità...

- Registrazione e Login:
    - Form di registrazione con validazione client/server (username univoco, email valida, password robusta).
    - Login attraverso local storage
    - Area di modifica profilo: aggiornamento di email, password, supereroe preferito; pulsante per cancellare l'account.

- Visualizzazione dell'album:

    - Griglia che mostra tutte le figurine disponibili: quelle non ancora collezionate appaiono come placeholder, quelle già acquisite mostrano la copertina.
    - Filtri per mostrare solo i doppioni o le mancanti.
    - Cliccando su una figurina si apre una scheda di dettaglio che include nome, descrizione, immagine e, su richiesta, informazioni aggiuntive (series, events, comics).

- Gestione crediti e acquisto pacchetti:

    - Pagina "Acquista crediti" (simulazione): seleziona quanti crediti comprare, aggiornamento del saldo utente.
    - Pagina "Apri pacchetto": se l'utente ha almeno 1 credito, può comprarne uno, il sistema decrementa 1 credito e restituisce 5 figurine selezionate a caso.
    - Animazione di apertura del pacchetto e visualizzazione delle 5 carte estratte, con indicazione di quali sono doppioni.

- Spazio Scambi:

    - Form per creare una nuova offerta:
      Selezione dei propri doppioni (minimo 1, fino a X in base al livello di complessità)
      Scelta delle figurine desiderate (singolo o multiplo).
    - Sezione "Le tue offerte" con elenco delle proposte pubblicate, stato (aperta, accettata, ritirata) e pulsante per ritirare un'offerta non ancora accettata
    - Sezione "Offerte disponibili" con elenco filtrabile per figurina richiesta; bottone "Accetta" per ogni proposta

Al momento della conferma di accettazione, il server verifica:

- L'offerta è ancora aperta e non è stata ritirata/accettata.
- L'accettante possiede i crediti o le figurine necessarie (in caso di vendita o scambio a multipli).
- Le figurine in arrivo non sono già presenti nell'album dell'accettante.
- Non ci siano duplicati nelle stesse categorie di offerta.

In caso di esito positivo, avviene la transazione: spostamento delle proprietà `userId` dei documenti relativi alle figurine tra i due utenti, aggiornamento dello stato dell'offerta in "accepted".

Funzionalità aggiuntive:

1. Scambi Multi-figurina:
    - Consentire all'utente di offrire più di una figurina in cambio di una o più figurine desiderate.
    - Modifica del modulo di offerta per includere array di ID carte offerte e richieste.
    - Lato server, controllare che la lunghezza degli array rispetti le regole di gioco e che non si offra la stessa carta due volte.

2. Vendita di Doppioni per Crediti
    - Introduzione di una sezione "Mercato" parallela agli scambi:
      Ogni utente può mettere in vendita i propri doppioni specificando un prezzo in crediti.
    - Gli annunci mostrano l'immagine, il nome e il prezzo.
    - Al click su "Vendita", il sistema verifica il saldo crediti dell'acquirente, decrementa i suoi crediti, incrementa quelli del venditore, e assegna la figurina al compratore.
    - L'annuncio cambia stato in "sold" e non è più acquistabile.

3. Controlli di integrità avanzati
    - Lato server, prima di accettare uno scambio, controllare che la figurina richiesta non sia già posseduta dall'accettante.
    - Verifica che un'unica offerta non contenga due o più copie identiche della stessa carta.
    - Prevenzione di scambi circolari (opzionale): controllare che non si creino chain di offerte non terminate che potrebbero arrecare vantaggi illeciti.

1.2 Analisi Architettura pagine web

`index.html / Landing Page`

- Header con logo, indirizzamento nella pagina di Login/Registrazione e Trova il SuperEroe.
- Se utente non è autenticato: mostra link “Login” e “Registrati”;
  se autenticato: evidenzia saldo crediti e pulsanti per acquisto pacchetti e accesso album.
- Include un footer con contatti e link a documentazione o privacy policy.

`login.html` e `register.html`

- Form, con validazioni HTML5 (required, type=email, minlength per password).
- Link reciproci per passare dall’uno all’altro.
- JavaScript per eventuali controlli in tempo reale (conferma password, validità email).

`user_profile.html`

- Visualizza e imposta i dati anagrafici: email, nome utente, password, nome, cognome, data di nascita e supereroe preferito (modificabile).
- Form di modifica password (con conferma).
- Pulsante “Elimina Account” con conferma in modal.

`package.html`

- Pulsante “Apri Pacchetto (1 credito)”.
- Al click, chiamata a /api/pacchetto/compra: se esito OK, restituisce array di 5 ID supereroe.
- Con ogni ID, recupera i dati base (nome, thumbnail) dalla cache in localStorage o da una struttura JSON precaricata.
- Mostra un’animazione di apertura (CSS + JS) e una griglia con le 5 card:
  -Se la figurina è doppione, evidenziarla con badge “Doppione”.
  -Al click su ciascuna card, aprire un lightbox con dettaglio completo del supereroe.

`album.html`

- Griglia fissa che rappresenta tutte le figurine esistenti (es. n° totale di eroi Marvel recuperati).
- Ogni cella prevede:
  1. Se utente possiede la figurina: mostra immagine e consente il clic per maggiori info.
  2. Se non possiede: visualizza un placeholder neutro.
- Barra in alto per filtri:
  “Mostra Doppioni”, “Mostra Mancanti”, casella di ricerca testuale per nome supereroe.
- JavaScript per caricare le informazioni relative alle figurine possedute (chiamata a /api/stickers/:userId) e popolare la griglia.

`select_exchange.html`

1. Bottone “Nuovo scambio”, indirizzato verso `scambio.html`, apre un modal con:
    - Multi-select di doppioni posseduti
    - Input (testuale o dropdown) per eroe richiesto(valido sia singolo che multiplo).
    - Pulsante “Crea Scambio”.

2. Bottone “Vendi duplicati” indirizzato verso `VenditaFigurine.html` con:
    - Contenitore #pacchetto_figurine: area a griglia per le carte duplicate, da selezionare

3. Layout diviso in due sezioni:

    Sopra: “Scambi Disponibili”:
    Elenco paginato o scroll infinito delle offerte create da altri. Include filtro per supereroe richiesto. Ogni offerta mostra miniatura delle carte offerte e quelle richieste, con bottone “Accetta” (se valido).

    Sotto: “I tuoi scambi”
    elenco delle offerte create dall’utente, con stato e bottone “Elimina” per eliminare gli scambi conclusi.

`get_credits.html`

- Visualizza pacchetti di crediti disponibili: ad esempio “5 crediti a € 5”.
- Pulsante “Acquista” che modifica il valore dei crediti
- Al ritorno, aggiorna in tempo reale il saldo sulla nabar

`navbar.js` (Dashboard utente presente in tutte le pagine html)

- Mostra il saldo crediti attuale.
- Pulsanti principali: “Cerca SuperEroe”, “Vai al Mio Album”, “Vai agli Scambi”, “Apri Pacchetti”

1.3 Motivazioni e alternative valutate

- Database NoSQL

  MongoDB è stato scelto per la sua natura schemaless, che si adatta bene a memorizzare documenti eterogenei (figurine, offerte, vendite) senza dover definire relazioni rigide come in un database relazionale. In prospettiva, se il modello dati restasse stabile, si potrebbe migrare a SQL, ma per la fase iniziale MongoDB offre maggiore flessibilità.

- NodeJS + Express

  Permette di lavorare con JavaScript sia lato client che server, uniformando il linguaggio. Express è leggero e flessibile, ideale per costruire rapidamente servizi REST. In un'alternativa, si sarebbe potuto usare un framework più strutturato (ad esempio NestJS), ma Express semplifica la curva di apprendimento e soddisfa i requisiti minimi.

- Caching in Web Storage

  - Salvare in localStorage (o IndexedDB) la lista dei supereroi riduce le chiamate HTTP ridondanti e migliora l'esperienza utente (caricamenti più rapidi, selezione pacchetti istantanea).

  - In alternativa, si sarebbe potuto memorizzare questi dati esclusivamente sul server (cache lato server), ma il polling continuo di un file JSON statico risulta più efficiente per richieste ripetute dal client.

- JWT per autenticazione
  (soluzione ideale da implementare,nel relativo progetto la registrazione passa per local storage)

  - I JSON Web Token permettono di costruire API stateless, riducendo la necessità di sessioni memorizzate sul server.

  - Un'alternativa era usare sessioni con cookie, ma per un'app che in prospettiva potrebbe avere più microservizi o un front-end separato, JWT è più scalabile.

- Validazioni e sicurezza

  - Uso di express-validator per garantire che ogni endpoint riceva solo dati corretti (numero di crediti nel formato giusto, ID Marvel validi, password con criteri minimi).

  - Proteggere tutte le rotte sensibili con un middleware che decodifica il token e recupera le informazioni dell'utente (authMiddleware).

- Separazione Front-end/Back-end

  - Mantenere il front-end come insieme di file statici serviti dal back-end o da un server statico consente di organizzare meglio il codice.

  - Questa architettura facilita sostituzioni future (ad es. passare a un framework React/Angular senza stravolgere il back-end).

2. Flusso Esecutivo

Il flusso esecutivo e l'organizzazione dei dati è consultabile al seguente link:
https://excalidraw.com/#json=-pGpreGndzqdwPrwPakK7,--MOfehZ6eEgAYlZ_V6vhg

3. Progetto Complessivo

3.1 Manuale utilizzo

Configurazione .env
Il file .env è un semplice testo che contiene tutte le variabili di configurazione sensibili o soggette a cambiamento (chiavi API, credenziali del database, URL, porte, ecc.). Scopo principale:

- Separare le configurazioni dal codice: evita di inserire direttamente nel sorgente valori che cambiano fra ambienti (sviluppo/produzione) o che non dovrebbero essere pubblici (password, token).

- Facilitare l'avvio del progetto: ogni sviluppatore o server di produzione può avere il proprio .env con valori locali, senza modificare il codice. Per il relativo utilizzo cambiare le chiavi pubbliche e private insieme ai parametri MongoDB.

```
# ==========================
# Configurazione Server
# ==========================
HOST = 'localhost'          # indirizzo di binding del server
PORT = 999                  # porta di ascolto


# ==========================
# Parametri MongoDB
# ==========================

DB_USERNAME     = donicidaniel9                                     # nome utente DB
DB_PASSWORD     = hashed_password                                   # password DB
DB_CLUSTER      = cluster0.ogic9vu.mongodb.net                      # indirizzo cluster Atlas
DB_DBNAME       = Cluster0                                          # nome del database
DB_OPTIONS      = retryWrites=true&w=majority&appName=Cluster0      # opzioni di connessione


# ==========================
# Configurazione Marvel API
# ==========================
BASE_URL = https://gateway.marvel.com/v1/               # endpoint base
PUBLIC_KEY = b7a7d8b026e408ce896de511dbccf80b           # chiave pubblica
PRIVATE_KEY = a747a5cee9e7d501ce441cda0c662d510b95026e  # chiave privata
```

3.2 Testing Flusso dati

1. All'avvio, il back-end esegue una chiamata a `https://gateway.marvel.com/v1/public/characters` per recuperare la lista di tutti i supereroi (con i relativi ID, nome, thumbnail).

2. Tali dati vengono salvati in un file JSON statico su server (es. `data/eroi.json`).

3. Il front-end, al caricamento iniziale, recupera questo JSON e lo memorizza in `localStorage` (o IndexedDB) per velocizzare le ricerche e minimizzare le chiamate future.

4. Quando serve dettaglio avanzato (series, events, comics), il back-end agisce da proxy verso Marvel, aggiungendo chiavi private/pubbliche e gestendo limiti di richiesta.

Nel nostro sistema, il flusso dei dati avviene tramite chiamate a una serie di endpoint **RESTful** ciascun endpoint è un URL che rappresenta una risorsa—ad esempio "figurine", "pacchetto" o "album" e accetta i metodi HTTP standard (GET, POST, PUT, DELETE) per effettuare operazioni di lettura, creazione, aggiornamento o cancellazione su quella risorsa.
Il server verifica i dati (crediti, duplicati, permessi), aggiorna MongoDB di conseguenza e restituisce una risposta JSON. In questo modo ogni azione dell'utente (acquisto, scambio, vendita) passa tramite le route RESTful, mantenendo sincronizzati frontend e backend.

Autenticazione e utenti

- **GET /user** → recupera i dati o la pagina di gestione dell'utente loggato.
- **GET /login** → restituisce il form di login.
- **GET /register** → restituisce il form di registrazione.
- **GET /get-credits** → mostra l'interfaccia per acquistare crediti.
- **GET /print-credits/{username}** → restituisce il saldo crediti di uno specifico utente.
- **POST /edit-credits** → modifica (aggiunge o sottrae) crediti a un utente.
- **PUT /update-user** → aggiorna informazioni di profilo dell'utente autenticato.
- **DELETE /delete-user/{userid}** → elimina l'utente e tutte le sue risorse.

Figurine (Album)

- **GET /package** → pagina di richiesta pacchetto
- **POST /package** → genera e fornisce un pacchetto di 5 figurine
- **GET /card** → dettaglio di una figurina specifica
- **GET /album** → pagina generale degli album
- **GET /albums/{userid}** → elenco album di un dato utente
- **GET /albums_cards/{albumid}** → elenca tutte le figurine di un album
- **GET /albums_duplicated_cards/{albumid}** → elenca solo i doppioni di un album
- **POST /check_card_album** → verifica se una figurina è già nell'album di un utente
- **GET /character/{id}** → recupera i dettagli di un eroe dalle API Marvel
- **POST /create_album** → crea un nuovo album per l'utente
- **POST /characters** → ricerca/personaggi Marvel in base a una query
- **DELETE /sell_card/** → rimuove ("vende") una figurina specifica

---

Dettagli supereroe

GET /api/heroes/{idEroe}/details → esegue richiesta verso le API di Marvel e restituisce un oggetto con:

```
{
  "name": string,
  "description": string,
  "thumbnail": URL,
  "series": [ ... ],
  "events": [ ... ],
  "comics": [ ... ]
}
```

il server effettua caching di breve durata per ridurre le chiamate ripetute alla Marvel API).

Accetto scambi

POST /api/accept_exchange
→ riceve nel body:

```
{
  "idEroiOfferti": [ numero, ... ],
  "idEroiRichiesti": [ numero, ... ]
}
```

1. verifica che l'utente autenticato possieda effettivamente ciascun idEroe in idEroiOfferti come duplicato: true;

2. controlla che non ci siano ID duplicati in idEroiOfferti né in idEroiRichiesti;

3. crea un documento nella collezione "offerte" con campi proponenteId, idEroiOfferti, idEroiRichiesti, stato: "aperta", creataIl: now() (richiede header Authorization: Bearer <token>).

GET /api/accept_exchange?status=open
restituisce l'elenco di tutte le offerte con stato === "aperta", includendo informazioni su proponenteId, idEroiOfferti e idEroiRichiesti.

PUT /api/accept_exchange/exchangeId}/accept

1. verifica che l'offerta identificata da {idOfferta} esista e abbia stato === "aperta";

2. controlla che l'utente autenticato (accettanteId) non sia lo stesso proponenteId;

3. verifica che l'utente autenticato possieda ciascun eroe in idEroiRichiesti come duplicato: true;

4. controlla che nessun idEroe in idEroiOfferti sia già presente nell'album dell'accettante;

5. avvia una sessione MongoDB per la transazione che:

    - sposta i documenti "figurine" con idEroe ∈ idEroiOfferti dal proponente all'accettante (aggiorna userId);

    - sposta i documenti "figurine" con idEroe ∈ idEroiRichiesti dall'accettante al proponente;

    - aggiorna l'offerta con stato: "accettata", accettataIl: now(), accettanteId

DELETE /api/accept_exchange/{exchangeId}
permette al proponenteId dell'offerta {idOfferta} di ritirarla se stato === "aperta", impostando stato: "ritirata" o eliminando il documento

Vendita di figurine

POST /api/create_exchange
→ body JSON:

```
 {
  "idEroe": numero,
  "prezzoInCrediti": numero
}
```

1. verifica che l'utente autenticato possieda idEroe come duplicato: true e che non sia già in vendita (inVendita: false);

2. crea un documento nella collezione "vendite" con campi venditoreId, idEroe, prezzoInCrediti, stato: "disponibile", creataIl: now();

3. aggiorna il documento "figurina" corrispondente impostando inVendita: true

POST /api/create_exchange
restituisce l'elenco di tutti gli annunci con stato === "disponibile", indicando venditoreId, idEroe e prezzoInCrediti

PUT /api/create_exchange/exchangeId}/accept

1 .verifica che il saldo crediti dell'utente autenticato (acquirente) sia ≥ prezzoInCrediti;

2. decrementa i crediti dell'acquirente e incrementa quelli del venditore;

3. aggiorna la "figurina" associata a {exchangeId}, cambiando userId da venditoreId a acquirenteId e impostando inVendita: false;

4. aggiorna il documento "vendita" con stato: "venduta" e vendutaIl: now()

DELETE /api/delete-exchange/{exchangeId}
se l'utente autenticato è il venditoreId dell'annuncio {idVendita} e stato === "disponibile", imposta stato: "annullata" e aggiorna la "figurina" associata con inVendita: false

---

Creare un file swagger-output.json che descriva tutti questi percorsi in italiano, specificando:

- Metodo HTTP
- Percorso (es. /api/utenti/registrazione)
- Parametri di input (path, query, body)
- Struttura del JSON di risposta e possibili codici di errore (400, 401, 403, 404, 500)

Integrare quindi swagger.js UI su /api-docs per consentire il testing interattivo durante lo sviluppo.

Lo swagger sarà configurato nel modo seguente:

```
import swaggerAutogen from 'swagger-autogen';
import { config } from "../../../config/prefs.js";
const outputFile = './swagger-output.json';
const endpointsFiles = ['../../*.js', '../../../app.js'];

const doc = {
  "info": {
    "title": "Marvel Universe: Donici",
    "description": "Test degli endpoint Marvel Universe",
    "version": "1.0.0"
  },
  host: `${config.host}:${config.port}`,
  basePath: "/",
  schemes: ['http'],
  consumes: ['application/json'],
  produces: ['application/json'],
  tags: [
    {
     "name": "fetch",
     "description": "Endpoint di base."
    },
    {
      "name": "users",
      "description": "Endpoint per la gestione dei dati utente e delle operazioni correlate."
    },
    {
     "name": "auth",
     "description": "Endpoint relativi all'autenticazione e autorizzazione dell'utente."
    },
    {
     "name": "cards",
     "description": "Endpoint per la gestione delle carte dell'album."
    },
    {
      "name": "exchanges",
      "description": "Endpoint per gestire gli scambi."
    },
    {
    "name": "database",
    "description": "Endpoint per verificare la connessione al database."
  }
  ],
```

```
  definitions: {
    user: {
      _id: "ObjectId('686ba122e387eb703b5b5e3e')",
      name: "Iustin",
      username: "Fragola99",
      surname: "Donici",
      email: "donici@gmail.com",
      password: "807fe1d5176e0c7a958500eb58ef7f08",
      date: "2004-09-09",
      superhero: "1009652",
      credits: "75.2"
    },

    album: {
      _id: "ObjectId('686b9cff133bfdd6814f1e02')",
      user_Id: "686b98fc133bfdd6814f1dfc" ,
      name: "TopiGalattici"
    },

    cards: {
      _id: "ObjectId('686be7e5eefe0f7e839548bb')",
      user_Id: "ObjectId('686ba122e387eb703b5b5e3e')",
      album_Id: "686ba16fe387eb703b5b5e3f",
      card_Id: 1009610
    },

    exchanges: {
      _id: "ObjectId('686bef0116c4b01ee8851a84')",
      user_Id: "ObjectId('686ba122e387eb703b5b5e3e')",
      album_id: "686ba16fe387eb703b5b5e3f",
      requestedCard: "1009718"
    },

    exchanges_cards: {
      _id: "ObjectId('686bef0216c4b01ee8851a85')",
      exchange_id: "ObjectId('686bef0116c4b01ee8851a84')",
      card_Id: 1010366,
      user_Id: "ObjectId('686ba122e387eb703b5b5e3e')",
      album_id: "686ba16fe387eb703b5b5e3f"
    }
  }
};
  const swagger = swaggerAutogen(outputFile, endpointsFiles, doc)
```