Documentazione Progetto Web App: Amiibo Collection

Cyrene Abjelina 869001

Anno Accademico 2024/2025

Indice

1 Descrizione del progetto

"Amiibo Collection" è una web app progettata per consentire agli utenti di esplorare e gestire la propria collezione personale di amiibo. Il progetto segue il pattern architetturale Model-View-Controller MVC e utilizza un'architettura moderna basata su Node.js per il backend e React per il frontend, con un'interfaccia utente dinamica grazie all'uso di Vite come bundler.

L'autenticazione degli utenti è gestita tramite Auth0, che fornisce una soluzione sicura e affidabile per la registrazione e l'accesso. L'identificativo degli utenti e la loro collezione personale di amiibo vengono memorizzati in un database MongoDB, garantendo la persistenza delle informazioni.

L'applicazione offre diverse funzionalità:

- **Visualizzazione della lista completa degli amiibo**, con la possibilità di filtrare per tipo (Figure, Band, Yarn, Card) e/o per serie di giochi (Super Mario Bros., The Legend of Zelda, Animal Crossing, ecc.)
- **Pagina dettagliata di ogni amiibo** con le sue caratteristiche, tra cui identificativo, tipo, serie, e date di rilascio nei vari continenti (Giappone, USA, Europa, Australia)
- **Gestione della collezione personale**, dove gli utenti registrati possono salvare e rimuovere amiibo dalla propria collezione

Se un utente non autenticato tenta di salvare un amiibo, l'applicazione mostra una finestra pop-up invitandolo ad accedere o registrarsi, in modo da poter visualizzare la propria pagina personale.

2 Architettura del Progetto

L'applicazione è organizzata seguendo il pattern **Model-View-Controller (MVC**), con una chiara separazione tra backend e frontend per garantire una gestione modulare e scalabile del codice.

- **Backend** (Node.js + Express): gestisce la logica di business, le richieste ai servizi esterni (AmiiboAPI[1]), l'autenticazione degli utenti tramite Auth0 e il salvataggio dei dati degli amiibo in MongoDB

- **Frontend** (React + Vite): offre un'interfaccia utente dinamica e reattiva, con componenti modulari che permettono di esplorare la collezione di amiibo e gestire la

[1] da https://www.amiiboapi.com/

propria collezione personale. Le rotte sono definite con React Router, mentre lo stato globale degli amiibo salvati è gestito tramite un context dedicato SavedAmiiboProvider accessibile da tutte le view

- **Database** (MongoDB): per la persistenza dei dati relativi agli utenti e agli amiibo salvati con uno schema personalizzato per l'associazione tra utente e collezione

La struttura del progetto è divisa in due cartelle principali:

- `/backend` contenente la logica server-side
- `/frontend` contenente la logica client-side

L'integrazione tra backend e frontend avviene tramite richieste HTTP utilizzando axios, una libreria che semplifica tali richieste rispetto al metodo nativo fetch.

2.1 Backend (Node.js + Express)

Il backend è organizzato come segue:

- `server.js`, punto di ingresso dell'applicazione backend. Configura il server Express, applica i middleware principali (`cors`, `body-parser`), collega le rotte, stabilisce la connessione a MongoDB e avvia il server sulla porta 3000
- `controllers/` contiene la logica per le operazioni principali e suddivisa in:
  - `amiiboController.jsx` gestisce le richieste per il recupero degli amiibo (`fetchAllAmiibos`, `fetchLatestAmiibos`, `fetchAmiiboBySeries`, `fetchAmiiboBySeriesAndType`, `fetchAmiiboByType`, `fetchAmiiboByTail`)
  - `userController.jsx` gestisce le richieste per il recupero e la gestione della collezione personale degli amiibo (`getSavedAmiibos`, `saveAmiibo`, `removeAmiibo`)
- `models/` definisce gli schemi per MongoDB, tra cui:
  - `userModel.js` schema per l'utente con il proprio `auth0Id` e la lista degli amiibo salvati
- `middleware/` gestisce la logica tra richiesta e risposta
  - `authMiddleware.js` protegge la rotta di Profilo verificando la validità del token JWT fornito da Auth0, in caso affermativo viene aggiunto l'`auth0Id` dall'oggetto `req.user` per identificare l'utente
- `routes/` definisce le rotte disponibili per l'applicazione:
  - `amiiboRoutes.jsx` gestisce tutte le rotte relative agli amiibo (`/`, `/collection`, `/collection/:param`)
  - `userRoutes.jsx` gestisce le rotte relative agli utenti (`/profile`)

2.2 Frontend (React + Vite)

Il frontend è organizzato come segue:

- `App.jsx` include il context `SaveAmiiboProvider` per fornire globalmente la lista degli amiibo salvati a tutte le view dell'app se l'utente è autenticato, e definisce il layout generale `MainTemplate` e le rotte principali con `react-router-dom`
- `main.jsx` punto di ingresso del frontend, dove React viene montato sul DOM
- `src/` contiene il context, tutti i componenti, le view e le utility
    - `components/` componenti riutilizzabili per la costruzione dell'interfaccia utente tra cui:
        - `MainTemplate` definisce il layout comune a tutte le pagine
        - `AmiiboCard` visualizza le informazioni di un amiibo e permette di salvarlo o rimuoverlo dalla collezione personale
        - `AmiiboGrid, AmiiboTable` modalità di visualizzazione della collezione
        - `Header, Footer` layout del `MainTemplate`
        - `LoginButton, LogoutButton` gestisce l'autenticazione dell'utente con Auth0
        - `SaveAmiiboButton` gestisce il salvataggio/rimozione degli amiibo nella collezione personale
    - `context/` contiene Provider React per la gestione dello stato globale:
        - `SaveAmiiboProvider` gestisce lo stato degli amiibo salvati, rendendo accessibili i `savedTails` e le funzioni di salvataggio e rimozione a tutti i componenti
    - `views/` contiene le pagine principali dell'applicazione:
        - `Home` pagina iniziale con alcune anteprime degli amiibo, tra cui quelle più recenti e quelle di diverse serie, ad esempio di Animal Crossing
        - `AmiiboCollection` mostra l'elenco completo degli amiibo con opzioni di filtro per serie, per tipo e di modalità di visualizzazione
        - `AmiiboDetail` pagina dei dettagli di un amiibo specifico
        - `Info` pagina informativa sugli amiibo
        - `Profile` pagina personale dell'utente, che mostra gli amiibo salvati
    - `assets/` contiene:
        - `images/` immagini utilizzate nell'applicazione
        - `data/` con informazioni per la pagina `Info`
    - `utility/` funzioni di supporto

2.3 File di configurazione

- `.env` contiene le variabili d'ambiente per configurare l'applicazione, tra cui:

    - `AUTH0_DOMAIN`, `AUTH0_CLIENTID`, `AUTH0_AUDIENCE` per la gestione dell'autenticazione
    - `MOGODB_URI` per la connessione al database MongoDB
- `vite.config.js` configurazione di Vite per la build del progetto
- `package.json` (sia per frontend che backend) elenca le dipendenze necessarie al funzionamento del progetto

3 Navigazione

La navigazione dell'applicazione si basa su un menu principale presente nell'Header e una serie di link secondari nel Footer. Di seguito sono descritti i principali percorsi di navigazione:

Home (/)

La pagina principale mostra:

- Gli ultimi 9 amiibo aggiunti nel 2024, con un pulsante Explore More, che se viene cliccato dall'utente, viene reindirizzato alla sezione Collection (`/collection`)
- Una sezione con anteprime di alcune serie popolari come Super Mario Bros, The Legend of Zelda, Animal Crossing, ognuna con un pulsante Explore More, che se viene cliccato dall'utente, viene reindirizzato alla sezione Collection filtrato per la serie specificata (`/collection/:amiiboSeries`)

Collection (`/collection`)

La sezione principale dedicata all'esplorazione completa degli amiibo con la possibilità di filtrare i risultati per:

- Tipo (`/collection/:type`), ad esempio Figura, Carta, Yarn
- Serie (`/collection/:amiiboSeries`)
- Combinazione di tipo e serie (`/collection/:amiiboSeries&:type`)

Gli utenti possono scegliere tra una visualizzazione a griglia o tabella per sfogliare gli amiibo in base alle loro preferenze

Amiibo Detail (`/collection/:tail`)

Pagina dedicata a ciascun amiibo, identificato dal suo tail e si mostra il bottone `SaveAmiiboButton` per il salvataggio e rimozione e i dettagli dell'amiibo: type, amiiboSeries, release nei vari continenti (Giappone, USA, Europa, Australia).
Sotto la descrizione principale vengono presentati 6 amiibo correlati della stessa serie, con un pulsante Explore More per visualizzare l'intera serie

Info (`/info`)

Pagina informativa che contiene dettagli sugli amiibo

Profile (`/profile`)

Accessibile solo agli utenti registrati, mostra la lista degli amiibo salvati dall'utente. Infatti se l'utente è registrato, l'amiibo viene salvato nel profilo, altrimenti appare un popup che richiede il login o la registrazione. Gli utenti possono gestire la propria collezione rimuovendo gli amiibo dalla collezione personale cliccando nuovamente sul pulsante `SaveAmiiboButton` presente nella `AmiiboCard`

4 Note importanti

Identificativo unico (`tail`)

Ho deciso di considerare il parametro tail come identificativo unico degli amiibo per semplificare la gestione. L'AmiiboAPI fornisce infatti i parametri head e tail, che concatenati rappresentano l'ID univoco dell'amiibo. Tuttavia, per evitare lunghezze eccessive e garantire una maggiore leggibilità, soprattutto nella modalità di visualizzazione a tabella, ho scelto di utilizzare solo tail. Prima di prendere questa decisione, ho verificato che tail fosse effettivamente univoco per ogni amiibo.

Gestione delle rotte in `/collection`

La gestione delle richieste HTTP nella rotta /collection/:param è stata strutturata in modo da interpretare dinamicamente il parametro param. Il controllo avviene come segue:

- Se param contiene il carattere &, rappresenta la concatenazione di amiiboSeries e type
- Se param corrisponde a uno dei valori validi di type (Card, Band, Figure, Yarn), allora rappresenta type
- Se param è una stringa esadecimale, rappresenta il tail di un amiibo
- In tutti gli altri casi, param rappresenta una amiiboSeries.

Per garantire che la rotta sia sempre aggiornata rispetto ai filtri selezionati dall'utente, ho utilizzato le funzioni di updateURL, sfruttando URLSearchParams per modificare dinamicamente i parametri dell'URL senza ricaricare la pagina.

Gestione delle `amiiboSeries`

Per mantenere l'applicazione modulare e scalabile, la lista delle serie (amiiboSeries) filtrabili viene calcolata dinamicamente. A tal fine, ho definito la variabile allSeries, che estrae tutte le serie univoche dall'AmiiboAPI, rendendo il sistema sempre aggiornato rispetto alle novità.

Gestione globale dei salvataggi: `SavedAmiiboProvider`

Per sincronizzare in tempo reale lo stato degli amiibo salvati tra tutte le pagine, ho implementato tale context globale. Questo provider espone savedTails e due funzioni onSave

e onUnsave, che vengono utilizzate da tutta l'app per aggiornare lo stato globale. In questo modo, il pulsante di salvataggio riflette sempre correttamente lo stato attuale, senza dover gestire isSaved localmente in ogni componente.

5 Avvio del progetto

Per facilitare l'avvio del progetto, è stato configurato uno script che utilizza concurrently, una libreria che consente di eseguire più comandi contemporaneamente in un unico terminale. Ciò permette di avviare sia il server backend che il client frontend con un solo comando.

Nel file `package.json` del progetto, è stata aggiunta la seguente configurazione:

```
{
  "devDependencies": {
    "concurrently": "^9.1.2"
  },
  "scripts": {
    "start": "concurrently \"npm run server\" \"npm run client\"",
    "server": "cd backend && node server.js",
    "client": "cd frontend && npm run dev"
  }
}
```

Perciò il comando start consente di avviare contemporaneamente il server backend e il client frontend utilizzando `concurrently`, mentre il comando server per entrare nella directory backend, per poi eseguire il file server.js con Node.js avviando il backend Express, analogamente il comando client per entrare nella directory frontend ed eseguire npm run dev per avviare il server di sviluppo di Vite.

Per avviare il progetto è necessario:

1. Installare le dipendenze nella root del progetto con `npm install`, ciò installerà anche `concurrently`
2. Avviare l'intero progetto eseguendo il comando `npm start` nella root del progetto

Dopo l'avvio, il server backend sarà disponibile all'indirizzo configurato, http://localhost:3000, mentre il client frontend all'indirizzo http://localhost:5173.
Tale approccio semplifica lo sviluppo, permettendo di lavorare contemporaneamente su frontend e backend senza dover avviare i processi manualmente in più terminali.