Relazione progetto applicazioni web Nicholas Laurencich 163040

- **Sommario:** Il progetto sviluppato consiste in una applicazione di e-commerce dedicata alla vendita di videogiochi, console e accessori. L'applicativo prevede un frontend realizzato in HTML, CSS e JavaScript e un backend implementato in Node.js con il framework Express. I dati sono presi e salvati su un database SQLite. Le funzionalità implementate sono: la gestione del catalogo prodotti, la registrazione e autenticazione degli utenti, il carrello, wishlist, gestione ordini e profilo utente (modificabile). L'obiettivo del progetto è stato quello di realizzare un sistema completo e modulare, che coprisse le principali esigenze di un sito di vendita online, garantendo semplicità d'uso e coerenza tra interfaccia e logica applicativa. Il metodo seguito è stato iterativo: inizialmente ho definito il modello dei dati, successivamente, il backend REST con al contempo una parte di frontend per testare le funzionalità (fetch API). Infine ho applicato gli stili e migliorato l'usabilità. Il risultato è un'applicazione funzionante che consente di simulare l'acquisto completo: dalla navigazione dei prodotti alla registrazione, fino al checkout e alla consultazione degli ordini effettuati.

- **Modello dei dati:** L'applicazione web utilizza un database SQLite con una struttura semplice, composta principalmente da queste entità:
    - User: identificato da un id (intero, chiave primaria, assegnato dal sistema), una email (TEXT, unico con validazione per il formato), password (TEXT, con hash, e validazione che rispetti certi parametri come numero minimo di caratteri, maiuscole, minuscole, numeri e caratteri speciali), first_name (TEXT, se inserisci numeri da errore), last_name (TEXT) e address (TEXT). Tutti i campi sono obbligatori.
    - Product: id (integer, chiave primaria, assegnato dal sistema), title (TEXT), price (REAL), image (TEXT, link all'immagine nella cartella images), category (TEXT, categoria del prodotto: console, gioco, accessorio), description(TEXT, descrizione del prodotto).
    - Order: identificato da un id (intero, chiave primaria, assegnato dal sistema), userId (id di chi ha fatto l'ordine, INTEGER, unico), items(TEXT, oggetti ordinati), subtotal (REAL, prezzo dei soli prodotti), shipping_cost(REAL, prezzo spedizione), shipping_method(TEXT, metodo di spedizione), final_total(REAL, prezzo finale) e createdAt(TEXT, data dell'ordine).
    - Cards identificate da un id (intero, chiave primaria, assegnato dal sistema), userId (id di chi ha fatto l'ordine, INTEGER, unico), number (TEXT, tra i 13 e 19 numeri, non permette di inserire caratteri), holder (TEXT proprietario della carta) e expiry (TEXT, data scadenza, deve essere nel formato corretto MM/AA, se si inserisce una data antecedente alla registrazione da errore, come anche nel caso si inseriscano caratteri).
    - La wishlist e il carrello vengono salvati con cookie sul browser.

- **<u>Implementazione del back-end</u>**
  Il back-end dell'applicazione è realizzato con Node.js e Express.js, con un database relazionale SQLite per la persistenza dei dati. L'architettura segue un modello REST, in cui ogni risorsa (utenti, prodotti, ordini, carte) è accessibile tramite endpoint dedicati che ritornano dati in formato JSON.
    - **Struttura del server:** Il file principale è server.js, che avvia il server Express sulla porta configurata (di default 3000). Vengono integrati alcuni middleware fondamentali:
      - **morgan** per il logging delle richieste HTTP;
      - **express.json()** per la gestione del corpo delle richieste in formato JSON;
      - **cookie-parser** per i cookie;
      - **cors** per abilitare richieste cross-origin;
      - **express.static** per servire i file statici del frontend.
      - **static middleware** per servire i file frontend.

      Il database viene aperto all'avvio tramite sqlite3, collegandosi al file store.db. Per l'autenticazione è definito un middleware requireAuth, che verifica la presenza di un token nell'header X-Auth-Token e recupera i dati dell'utente dal database. In assenza di un token valido, viene restituito 401 Unauthorized. Il database è gestito tramite il modulo sqlite, aperto all'avvio e condiviso tra gli endpoint.

    - **Endpoint REST:** Gli endpoint sono divisi per risorse principali:
      - **Autenticazione e utenti**
        - POST /api/register: registrazione con validazione di tutti i campi, hashing sicuro della password (attraverso bcrypt) e salvataggio dell'utente.
        - POST /api/login: verifica delle credenziali e generazione di un "token" (in realtà l'ID utente, passato nell'header).
        - GET /api/me: recupero dei dati del profilo autenticato.
        - PUT /api/me: aggiornamento dei dati personali.
        - PUT /api/me/password: cambio password.
      - **Carte di pagamento**
        - GET /api/cards: elenco delle carte associate all'utente (solo ultime 4 cifre).
        - POST /api/cards: aggiunta di una nuova carta con validazione.
        - DELETE /api/cards/:id: eliminazione di una carta.
      - **Prodotti**
        - GET /api/products: ricerca e filtro dei prodotti con supporto a q, limit, offset, category.
        - GET /api/products/:id: dettaglio di un singolo prodotto.

        - **Ordini**
            - GET /api/orders: elenco degli ordini di un utente autenticato.
            - POST /api/orders: creazione di un ordine con validazione dei totali e dei limiti (max 20 articoli, max 5 per prodotto).
            - GET /api/orders/:id: dettaglio di un singolo ordine.
    - **Gestione degli status code:** gli endpoint rispettano le convenzioni REST:
        - 200 OK per risposte corrette;
        - 201 Created per risorse appena create (es. nuovi ordini o carte);
        - 204 No Content per operazioni senza contenuto di ritorno (es. eliminazione carte);
        - 400 Bad Request per input non validi o campi mancanti;
        - 401 Unauthorized per autenticazione mancante o token non valido;
        - 404 Not Found per risorse inesistenti (es. prodotto o ordine mancante);
        - 500 Internal Server Error per errori inattesi.

    - **Persistenza dei dati**
        - La persistenza è affidata al database SQLite, con tabelle per users, cards, products, e orders (definite in seed.js, file usato per popolare il database). La base dati viene inizializzata con alcune entità di esempio (prodotti, utenti, carte). Ogni tabella ha chiavi primarie auto-incrementali e vincoli di integrità (foreign keys tra utenti e carte/ordini).
        - Gli ordini vengono salvati con tutti i dettagli rilevanti: articoli (in JSON), subtotale, costi di spedizione, totale finale e timestamp di creazione.

    - **Validazione dei dati**
        - Il server implementa una validazione rigorosa lato backend, tramite espressioni regolari e controlli logici:
        - Email (emailRegex), nome e cognome (nameRegex), password (digest SHA256 lungo 64 caratteri), numero di carta (cardRegex), scadenza (expiryRegex con controllo data non scaduta), CVV e CAP.
        - In fase di ordine vengono verificati i prezzi dei prodotti direttamente dal database per prevenire manomissioni da parte del client.
        - Sono previsti limiti numerici (max 5 unità per prodotto).
    - Conclusione: Il back-end fornisce una struttura robusta e sicura per la gestione di utenti, prodotti, pagamenti e ordini. La combinazione di Express, SQLite e validazione rigorosa garantisce integrità dei dati e protezione da input non validi, mentre l'adozione di convenzioni REST rende il sistema facilmente integrabile con un frontend.

- **Implementazione del Front-End**
  L'architettura si basa sull'uso di più file JavaScript, ognuno dei quali gestisce una vista. Il progetto fa uso di funzionalità native del browser, come la Fetch API per la comunicazione con il backend e la manipolazione diretta del DOM (Document Object Model) per l'aggiornamento dinamico dell'interfaccia utente. L'intera applicazione è scritta in JavaScript, senza l'ausilio di librerie esterne.

    - **Pagine e Viste Implementate:** il progetto copre un'ampia gamma di funzionalità tipiche di un e-commerce, con ogni vista gestita da un file JavaScript dedicato. L'interfaccia utente viene renderizzata dinamicamente in base ai dati recuperati:
        - **Homepage (index.js):** La vista principale che mostra un elenco di prodotti, gestendo la paginazione, la ricerca per parola chiave e il filtraggio per categoria.
        - **Pagina Prodotto (product.js):** Una vista dedicata che carica e mostra i dettagli di un singolo prodotto, permettendo all'utente di aggiungerlo al carrello.
        - **Carrello (cart.js):** Gestisce il carrello dell'utente, visualizzando gli articoli aggiunti, permettendo la modifica delle quantità e la rimozione dei prodotti.
        - **Checkout (checkout.js):** Guida l'utente attraverso il processo di acquisto, calcolando subtotale, spese di spedizione e totale finale, e gestendo le opzioni di pagamento.
        - **Profilo (profile.js):** Consente all'utente di visualizzare e modificare le proprie informazioni personali e le carte di credito salvate. La password viene gestita con un hash SHA-256 prima di essere inviata.
        - **Ordini (orders.js e order-details.js):** Viste separate per visualizzare la cronologia degli ordini e i dettagli di un singolo ordine.
        - **Wishlist (wishlist.js):** Permette all'utente di gestire una lista di prodotti preferiti, potendo aggiungere o rimuovere articoli e trasferirli direttamente nel carrello.
        - **Autenticazione (login.js e register.js):** Pagine dedicate per l'accesso e la registrazione di nuovi utenti.
        - **Chi Siamo (about.js):** Una semplice vista informativa che presenta un componente interattivo a fisarmonica (accordion).
        - **App.js**: si occupa di gestire l'interazione tra l'utente e le api del backend (autenticazione, carrello, wishlist, navigazione e badge dinamici).

- **Organizzazione dei File:** L'organizzazione dei file riflette l'approccio modulare del progetto.
    - **Cartella js:** Ogni file js è contenuto in questa cartella. Index.js o cart.js, è responsabile della logica di una specifica vista. I file di autenticazione (login.js, register.js) e di gestione del profilo (profile.js) includono funzioni di validazione dei dati e logica per l'interazione con il backend. Il file app.js contiene tutte le funzioni globali e riutilizzabili, come apiFetch per le chiamate API e le funzioni per la gestione del carrello (getCart, setCart) e della wishlist (getWishlist, setWishlist).
    - **Cartella images:** contiene tutte le immagini dei prodotti in formato jpg.
    - **Cartella css:** Contiene l'intero stile dell'applicazione in formato css. Sfrutta le variabili CSS (definite in :root) per creare un tema scuro coerente e stili riutilizzabili per componenti comuni come card, btn-primary, e qty-input. Questo centralizza la gestione del design e facilita la modifica dell'aspetto dell'applicazione.
    - I file HTML sono contenuti nella cartella frontend.
- **Uso della Fetch API e Aggiornamento Dinamico del DOM**: La comunicazione con il backend è gestita in modo efficiente e centralizzato attraverso la Fetch API del browser. La funzione di utilità apiFetch() (definita in app.js) funge da wrapper per tutte le richieste, aggiungendo automaticamente il token di autenticazione per le chiamate che lo richiedono e gestendo gli errori in un unico punto. L'aggiornamento dell'interfaccia utente è completamente dinamico, sfruttando i dati ricevuti dalle risposte delle API.
    - **Costruzione del DOM:** Il codice JavaScript manipola direttamente il DOM per visualizzare i dati. Le viste come la homepage (index.js) e la wishlist (wishlist.js) generano il codice HTML necessario (come article e div) a partire da un array di prodotti e lo iniettano nell'elemento container (grid.innerHTML).
    - **Interattività:** Gli elementi interattivi, come i pulsanti "Aggiungi al carrello" o "Rimuovi", sono associati a event listener che, al clic, eseguono chiamate API e aggiornano lo stato del DOM senza ricaricare la pagina. Un esempio è il file cart.js, che aggiorna il totale del carrello in tempo reale quando l'utente cambia la quantità di un prodotto.
    - **Stato del Carrello e Wishlist:** I dati del carrello vengono salvati in localStorage, mentre quelli della wishlist vengono salvati nei cookie del browser. Questo permette di persistere lo stato del carrello e della lista dei desideri anche dopo che l'utente ha chiuso il browser.

- **Gestione dei Dati e Sicurezza:** L'applicazione gestisce diverse tipologie di dati sensibili e di stato:
    - **Autenticazione:** Il token di autenticazione dell'utente è salvato in localStorage per mantenere la sessione attiva.
    - **Crittografia:** Per la sicurezza delle credenziali, la password dell'utente non viene mai inviata in chiaro. Il front-end utilizza l'API nativa crypto.subtle.digest per calcolare un hash SHA-256 della password, che viene poi inviato al server.
    - **Stato dell'Applicazione:** Lo stato della paginazione (index.js) e altri dati temporanei sono gestiti in variabili JavaScript locali, garantendo che le informazioni persistano solo per il tempo necessario all'interno della vista corrente.
- **Librerie Esterne Adottate:** il frontend è stato realizzato in JavaScript, senza l'uso di alcuna libreria o framework esterno eccetto un link a w3 per il pulsante torna su. Mentre per il backend i middleware scritti sopra.

- **CONCLUSIONI:** le difficoltà maggiori le ho riscontrate nella realizzazione del backend, in particolare nel modo per far parlare il server con il client e rendere tutti i dati coerenti per far salvare correttamente le informazioni nel database. Ci sono molti punti di miglioramento, tra cui la possibilità di poter aggiungere più indirizzi nel proprio profilo (come si fa con le carte di pagamento), aggiungere altri metodi di pagamento, salvare il carrello non nella cache del browser ma sul database. Inoltre sarebbe stato bello realizzare tutta l'interfaccia per il gestore del e-commerce, il quale una volta autenticato, il server lo riconosce come gestore e gli si apre un interfaccia diversa, in cui può visualizzare ordini, quantità rimanenti di prodotti, inserire e rimuovere prodotti ecc. una cosa che ho scoperto mentre stavo sviluppando questa applicazione è la gestione del CVV per le carte di pagamento, il quale non viene salvato sul databse del sito ma attraverso un provider di tokenizzazione al gateway di pagamento. Ci sarà un token unico che rappresenterà la carta di pagamento. Un problema che non sono riuscito a risolvere è nella pagina di ringraziamento, ho certato di impedire di ritornare alla pagina del checkout, ma non ci sono riuscito. Premendo la freccia indietro del browser mi riporta alla pagina del checkout come se non avessi fatto l'ordine, il che è sbagliato.