Marco Bortolussi 152352

Chef Per Caso

1. Sommario

Questo progetto ha avuto come obiettivo lo sviluppo di un'applicazione web full-stack, "Chef Per Caso", un **ricettario** personale interattivo. L'applicazione è stata concepita per dimostrare la comprensione e l'applicazione dei principi dell'architettura RESTful, della gestione dei dati lato server e della costruzione di un'interfaccia utente dinamica e responsiva.

Il dominio applicativo scelto è quello di un ricettario, dove l'utente può esplorare una collezione di ricette culinarie. Le funzionalità principali realizzate includono:

- **Esplorazione Ricette:** Visualizzazione di un elenco di ricette con possibilità di navigazione e consultazione del dettaglio di ogni singolo piatto.
- **Ricerca Avanzata:** Funzionalità di *ricerca testuale* sulle ricette, combinata con un sistema di *paginazione* per gestire grandi set di risultati in modo efficiente.
- **Autenticazione Utente:** Un sistema di *login e registrazione* "light" che permette agli utenti di creare un account, accedere e gestire il proprio profilo.
- **Gestione Preferiti:** Gli utenti autenticati possono aggiungere e rimuovere ricette dalla propria lista di preferiti, personalizzando la loro esperienza.
- **Generazione Menu Casuale:** Una funzionalità divertente che suggerisce un menu completo (antipasto, primo, secondo, contorno, dolce) in base alle ricette disponibili.
- **Visualizzazione per Categoria:** Possibilità di filtrare e visualizzare le ricette per categoria, migliorando la navigazione del contenuto.

Le tecnologie utilizzate per lo sviluppo includono:

- **Backend**: Node.js con il framework Express.js per la gestione delle API REST. La persistenza dei dati (utenti, ricette, preferiti) è gestita tramite file JSON.
- **Frontend**: HTML5 per la struttura, CSS3 per lo stile (con un focus sul responsive design tramite Flexbox e Grid) e JavaScript per l'interattività e la comunicazione con il server tramite la Fetch API.

2. Modello dei Dati

Il modello dei dati dell'applicazione "Chef Per Caso" è stato progettato per supportare le funzionalità di un ricettario personale, gestendo tre entità principali: *Ricette*, *Utenti* e *Preferiti*. La persistenza dei dati è garantita attraverso la scrittura su file JSON dedicati.

2.1 Entità Principale: Ricette

L'entità Ricette rappresenta il *cuore* del ricettario. Ogni ricetta è un oggetto JSON con i seguenti attributi:

- **Id:** Numero Intero (Identificatore univoco della ricetta).

- **Categoria:** Stringa (Es. "Antipasto", "Primo", "Secondo", "Dolce"). Indica la tipologia della ricetta.
- **Nome:** Stringa (Nome della ricetta).
- **Descrizione**: Stringa (Descrizione dettagliata della ricetta).
- **Immagine**: Stringa (Percorso URL dell'immagine associata alla ricetta).
- **Ingredienti**: Array di Stringhe (Elenco degli ingredienti necessari).
- **Istruzioni**: Stringa (Passi dettagliati per la preparazione della ricetta).
- **Calorie**: Numero Intero (Valore calorico stimato della ricetta).
- **Tempo**: Numero Intero (Tempo di preparazione in minuti).

2.2 Entità Secondaria: Utenti (Account)

L'entità *Utenti* gestisce i dati degli account utente per l'autenticazione e la personalizzazione. Ogni utente è un oggetto JSON con i seguenti attributi:

- **Id**: Stringa (Identificatore univoco dell'utente, es. "user1", "user2"). Generato automaticamente dal server al momento della registrazione.
- **Username**: Stringa (Nome utente scelto dall'utente).
- **Password**: Stringa (Password dell'utente).

2.3 Entità Secondaria: Preferiti

L'entità *Preferiti* modella la relazione tra gli utenti e le ricette che essi hanno contrassegnato come preferite. È rappresentata come un oggetto JSON dove le chiavi sono gli id degli utenti e i valori sono array contenenti gli id delle ricette preferite da quell'utente.

- **UserId**: Stringa (Chiave dell'oggetto, corrispondente all'ID di un utente).
- **RicettaId**: Array di Numeri Interi (Valore associato alla chiave userId, contenente gli ID delle ricette preferite da quell'utente).

Relazione: Molti-a-Molti (un utente può avere molte ricette preferite, e una ricetta può essere preferita da molti utenti).

3. Implementazione del Back-end

Il back-end dell'applicazione "Chef Per Caso" è stato sviluppato utilizzando Node.js con il framework Express.js. Questa scelta ha permesso di creare un server robusto e scalabile, in grado di gestire le richieste HTTP in modo efficiente e di implementare un'API RESTful.

3.1 Architettura e Struttura del Server

Il server è organizzato in un singolo file server.js, ma segue principi modulari per la gestione delle diverse funzionalità:

- **Inizializzazione di Express**: Il server viene avviato e configurato per l'ascolto su una porta specifica.
- **Middleware**: Vengono utilizzati middleware per il parsing del corpo delle richieste JSON (*express.json()*) e per la gestione dei dati codificati in URL (*express.urlencoded({ extended: true })*). È stato incluso anche un middleware per servire file statici (immagini, CSS, JS del frontend) dalla directory public.

- **Gestione dei Dati**: I dati per utenti, ricette e preferiti sono caricati da file JSON (users.json, ricette.json, preferiti.json) all'avvio del server e mantenuti in memoria come array o oggetti JavaScript. Le modifiche ai dati (registrazione, aggiornamento profilo, gestione preferiti) vengono persistite scrivendo nuovamente sui rispettivi file JSON.
- **Funzioni di Utilità:** Sono presenti funzioni helper per la lettura e scrittura sincrona dei file JSON (saveFavorites, saveUsers), garantendo la persistenza dello stato dell'applicazione.

3.2 Endpoint RESTful

L'applicazione espone diversi endpoint RESTful per interagire con le risorse (utenti, ricette, preferiti) seguendo i principi HTTP standard (GET, POST, PUT, DELETE).

3.2.1 Gestione Utenti e Autenticazione

- *POST /register (Registrazione Utente):*
    - Funzionalità: Permette la creazione di un nuovo account utente.
    - Richiesta: Accetta un corpo JSON con username e password.
    - Validazione:
        - Verifica che username e password non siano vuoti.
        - Controlla l'unicità dell'username (case-insensitive) per evitare duplicati.
        - Genera un id univoco per il nuovo utente (es. "user1", "user2").
- *POST /login (Login Utente):*
    - Funzionalità: Autentica un utente esistente.
    - Richiesta: Accetta un corpo JSON con username e password.
    - Validazione: Verifica che le credenziali fornite corrispondano a un utente registrato.
- *PUT /api/users/:userId (Aggiornamento Profilo Utente):*
    - Funzionalità: Permette a un utente di aggiornare il proprio username o password.
    - Richiesta: Accetta un corpo JSON con username (opzionale), currentPassword (obbligatorio per la verifica) e newPassword (opzionale).
    - Validazione:
        - Richiede l'autenticazione tramite il middleware authenticateToken.
        - Verifica che l'utente autenticato sia lo stesso che sta cercando di modificare il profilo (req.userId === userId).
        - Controlla che currentPassword sia corretta.
        - Se fornito, il newPassword deve avere una lunghezza minima di 3 caratteri.
        - Se username viene modificato, verifica che il nuovo username non sia già in uso da un altro utente.
- *DELETE /api/users/:userId (Eliminazione Account Utente):*
    - Funzionalità: Permette a un utente di eliminare il proprio account.
    - Richiesta: Nessun corpo, l'ID è nell'URL.

- Validazione:
    - Richiede l'autenticazione tramite il middleware authenticateToken.
    - Verifica che l'utente autenticato sia lo stesso che sta cercando di eliminare l'account.
- Effetti Collaterali: Rimuove anche tutte le ricette preferite associate all'utente eliminato dal file preferiti.json.

3.2.2 Gestione Ricette

- *GET /api/ricette (Recupero Tutte le Ricette con Paginazione e Ricerca):*
    - Funzionalità: Restituisce un elenco di ricette, con supporto per ricerca testuale (query) e paginazione (limit, offset).
    - Richiesta: Parametri di query: query (stringa di ricerca), limit (numero di ricette per pagina) e offset (indice di partenza).
- *GET /api/ricette/:id (Recupero Singola Ricetta):*
    - Funzionalità: Restituisce i dettagli di una specifica ricetta.
    - Richiesta: L'ID della ricetta nell'URL.
    - Validazione: Verifica che l'ID sia un numero valido.

3.2.3 Gestione Preferiti

- *GET /api/favorites (Recupero Preferiti Utente):*
    - Funzionalità: Restituisce l'elenco delle ricette preferite per l'utente autenticato.
    - Richiesta: Richiede l'autenticazione tramite il middleware authenticateToken.
- *POST /api/favorites (Aggiungi Ricetta ai Preferiti):*
    - Funzionalità: Aggiunge una ricetta all'elenco dei preferiti dell'utente autenticato.
    - Richiesta: Accetta un corpo JSON con recipeId. Richiede l'autenticazione.
    - Validazione:
        - Verifica che recipeId sia fornito e che la ricetta esista.
        - Controlla che la ricetta non sia già presente nei preferiti dell'utente.
- *DELETE /api/favorites/:recipeId (Rimuovi Ricetta dai Preferiti):*
    - Funzionalità: Rimuove una ricetta dall'elenco dei preferiti dell'utente autenticato.
    - Richiesta: L'ID della ricetta nell'URL. Richiede l'autenticazione.
    - Validazione:
        - Verifica che recipeId sia fornito e che la ricetta sia effettivamente nei preferiti dell'utente.

3.3 Middleware di Autenticazione (authenticateToken)

Un middleware personalizzato, authenticateToken, è stato implementato per proteggere gli endpoint che richiedono un utente autenticato.

- Funzionamento:

1. Estrae il token di autenticazione dall'header Authorization .
2. Se il token è assente, restituisce un 401 Unauthorized.
3. Se il token è presente, verifica che corrisponda a un id utente valido nel sistema.
4. Se il token è valido, l'id dell'utente viene allegato all'oggetto req (es. req.userId), rendendolo disponibile per i successivi gestori di route.
5. Se il token non è valido, restituisce un 403 Forbidden.

3.4 Gestione degli Errori e Risposte JSON

Il back-end è configurato per inviare risposte JSON chiare e concise, accompagnate dagli appropriati codici di stato http, come 200, 401, 403, 404, 409, 500.

4. Implementazione del Front-end

L'interfaccia utente dell'applicazione "Chef Per Caso" è stata sviluppata utilizzando un approccio moderno con **HTML5** per la struttura, **CSS3** per lo stile e il design responsivo, e **JavaScript puro** per l'interattività e la comunicazione asincrona con il back-end.

4.1 Struttura delle Pagine e Organizzazione dei File

Il front-end è organizzato in una struttura di directory chiara e funzionale all'interno della cartella public, che è la directory principale servita direttamente dal server Express. Questa struttura comprende **sei cartelle dedicate a pagine specifiche** e una cartella common per gli asset condivisi.

- La cartella **common/** è cruciale per la modularità e l'efficienza. Contiene i file CSS (auth.css, message-box.css, confirm-modal.css) che definiscono gli stili condivisi tra tutte le pagine, come quelli per l'header, i modali di autenticazione e i messaggi di sistema. Al suo interno si trova anche auth.js, un file JavaScript comune a tutte le pagine, responsabile dell'inserimento dinamico dell'header (con il menu a tendina e le sezioni di login/profilo) e dei modali di autenticazione e messaggio in ogni pagina. Questo approccio elimina la duplicazione di codice HTML e JavaScript per elementi ricorrenti, centralizzando la loro gestione.
- Le **sei cartelle specifiche per pagina** (es. main/, piatti/, menu-casuale/, categoria/, preferiti/, search-results/) ospitano ciascuna il proprio file index.html che rappresenta la pagina principale. All'interno di queste cartelle si trovano anche i file CSS e JavaScript specifici per quella pagina (es. main/style.css, main/script.js), che vengono caricati solo quando la pagina corrispondente è visualizzata, ottimizzando il caricamento delle risorse.
- Infine, la cartella **img/** contiene tutte le risorse grafiche utilizzate nell'applicazione, come il logo e le immagini delle ricette.

Questa organizzazione garantisce che ogni componente sia facilmente localizzabile e che le modifiche a elementi comuni (come l'header o la logica di autenticazione) si riflettano in modo coerente su tutte le pagine che li includono, grazie all'inserimento dinamico via JavaScript.

4.2 Layout e Design Responsivo

Il layout dell'applicazione è stato progettato per essere intuitivo e ben organizzato, con un'enfasi sul **responsive design** per garantire una buona esperienza utente su dispositivi con schermi di diverse dimensioni (desktop, tablet, mobile).

- **Flexbox:** Ampiamente utilizzato per la disposizione degli elementi in una singola dimensione (righe o colonne), come la barra di navigazione, i controlli di paginazione e l'allineamento dei contenuti all'interno delle card. La proprietà flex-wrap: wrap; è stata impiegata per permettere agli elementi di andare a capo su nuove righe quando lo spazio orizzontale si riduce, adattandosi allo schermo.

- **CSS Grid:** Utilizzato per la creazione di layout bidimensionali più complessi, in particolare per la griglia delle ricette. La proprietà grid-template-columns: repeat(auto-fill, minmax(280px, 1fr)); è stata fondamentale per creare una griglia dinamica che adatta automaticamente il numero di colonne e la loro larghezza in base allo spazio disponibile, mantenendo una dimensione minima leggibile per ogni card.
- **Unità Relative:** L'uso di unità come rem, em, % e vw/vh ha contribuito alla fluidità del layout, assicurando che gli elementi si ridimensionino proporzionalmente alla dimensione dello schermo.
- **Stili Visivi:** Sono stati applicati stili moderni con angoli arrotondati, ombreggiature leggere e transizioni fluide per migliorare l'estetica e la user experience, rendendo l'interfaccia accattivante e facile da usare.

4.3 Interattività e Aggiornamento Dinamico del DOM

Il **JavaScript puro** è il motore dell'interattività lato client, gestendo le interazioni dell'utente e aggiornando dinamicamente il DOM in risposta alle richieste al server.

- **Fetch API:** Tutte le comunicazioni con il back-end avvengono tramite la Fetch API, consentendo richieste HTTP asincrone (GET, POST, PUT, DELETE) e la gestione delle risposte JSON. Questo approccio garantisce che l'interfaccia utente rimanga reattiva durante le operazioni di rete.
- **Aggiornamento del DOM:** Il DOM viene manipolato dinamicamente per riflettere lo stato corrente dell'applicazione. Esempi includono:
    - La pagina delle categorie che genera e visualizza dinamicamente tutte le ricette appartenenti alla categoria selezionata, caricando i dati dal back-end e costruendo gli elementi HTML.
    - La pagina dei risultati di ricerca che aggiorna la griglia delle ricette e i controlli di paginazione in base alla query e alla pagina corrente, senza richiedere un ricaricamento completo della pagina.
    - La gestione dei preferiti che aggiorna lo stato visivo dell'icona a cuore (piena/vuota) e la lista dei preferiti in tempo reale.
    - I modali di login/registrazione e modifica profilo che appaiono e scompaiono, con campi pre-compilati e messaggi di feedback che guidano l'utente.
- **Feedback Utente:** Vengono forniti feedback visivi chiari tramite modali personalizzati (showCustomMessage, showConfirmMessage) per operazioni completate con successo, messaggi informativi ed errori comprensibili, migliorando la chiarezza delle interazioni.

4.4 Navigazione e Autenticazione

- **Menu a Tendina (Dropdown Menu):** Un menu a tendina posizionato in alto a sinistra offre una navigazione intuitiva tra le diverse sezioni del ricettario (Home, I Miei Preferiti, e le varie categorie di ricette). Questo menu è parte dell'header dinamico inserito da common.js.
- **Accesso Globale a Login/Profilo:** I modali di login/registrazione e modifica profilo sono accessibili da **ogni pagina** dell'applicazione tramite pulsanti nell'header. Questo garantisce una user experience fluida, permettendo all'utente di autenticarsi o gestire il proprio account in qualsiasi momento senza interruzioni.
- **Stato di Autenticazione:** L'header aggiorna dinamicamente il testo del pulsante "Login" a "Logout" e mostra/nasconde il pulsante "Modifica Profilo" in base allo stato di autenticazione dell'utente (verificato tramite la presenza del authToken nel localStorage). La logica di autenticazione è centralizzata in common.js.

4.5 Validazione Lato Client

Un primo livello di validazione dei dati è implementato lato client per intercettare errori comuni prima che le richieste vengano inviate al server, migliorando la reattività dell'interfaccia e riducendo il carico sul back-end.

- **Form di Registrazione e Modifica Profilo:** Vengono eseguiti controlli JavaScript per:
    - Verificare che tutti i campi obbligatori siano compilati.
    - Assicurare che le password corrispondano.
    - Imporre una lunghezza minima per la password (almeno 3 caratteri).
    - Richiedere che la password contenga almeno un numero.
- **Feedback Immediato:** I messaggi di errore vengono visualizzati direttamente sotto i campi del form o tramite i modali di messaggio personalizzati, guidando l'utente nella correzione degli input in tempo reale.

5. Conclusioni

In conclusione, il progetto "Chef Per Caso" ha dimostrato la capacità di progettare e implementare un'applicazione web full-stack, affrontando le sfide tecniche e fornendo un prodotto funzionale e ben strutturato che rispetta i requisiti del corso e offre un'esperienza utente piacevole.

Nonostante le sfide, il progetto presenta diversi **punti di forza:**

- **Adesione ai Principi REST**: L'applicazione espone endpoint chiari e semanticamente significativi per le operazioni GET, POST, PUT e DELETE, rispettando l'architettura RESTful.
- **Architettura Modulare**: La separazione tra front-end e back-end, unita all'organizzazione dei file e all'uso di common.js per la gestione di elementi condivisi, rende il codice più leggibile, manutenibile e scalabile.
- **Interfaccia Utente Intuitiva e Responsiva**: L'ampio utilizzo di Flexbox e CSS Grid ha permesso di creare un layout che si adatta fluidamente a desktop, tablet e mobile, garantendo una buona user experience su qualsiasi dispositivo.
- **Interattività Dinamica**: L'uso estensivo di JavaScript puro e della Fetch API per l'aggiornamento dinamico del DOM riduce i ricaricamenti di pagina, offrendo un'esperienza utente fluida e reattiva.

5.1 Proposte di Miglioramento Future

Per le future evoluzioni, si potrebbero considerare i seguenti **miglioramenti**:

- **Funzionalità di Commenti e Valutazioni**: Permettere agli utenti di lasciare commenti e valutazioni sulle ricette per creare una community più interattiva.
- **Filtri e Ordinamento Avanzati**: Implementare opzioni di filtro e ordinamento più sofisticate (es. per tempo di preparazione, difficoltà, ingredienti specifici).
- **Notifiche Utente**: Aggiungere un sistema di notifica per avvisare gli utenti su nuove ricette, aggiornamenti o interazioni.