UNIVERSITÀ DEGLI STUDI DI UDINE

Dipartimento Politecnico di Ingegneria e Architettura

Corso di Laurea Magistrale in

Ingegneria Gestionale

Progetto del corso di Applicazioni Web

Applicazione web per la gestione del concessionario di automobili AutoTorino SPA

Professore:

Prof./Dott. Michael Soprano

Gruppo:

Giovanni Faccini 153762

Zuccolo Federico 147776

Anno Accademico 2024 – 2025

Indice

1. Sommario 3
2. Modello dei dati 3
   2.1. Entità principale: Auto (Car) 3
   2.2. Entità secondaria: user (utente) 4
   2.3. Entità secondaria: riparazione 4
3. Implementazione del Back-end 4
   3.1. Architettura del Server 4
   3.2. Endpoint REST e Operazioni CRUD 5
   3.3. Gestione degli Status Code http 5
   3.4. Validazione e Persistenza dei Dati 5
4. Implementazione del Front-end 6
   4.1. Viste principali 6
   4.2. Utilizzo della Fetch API 6
   4.3. Modal CRUD 6
   4.4. Client-side 6
   4.5. Stile 7
5. Conclusioni 7
   5.1. Funzionalità extra implementate 7
   5.2. Proposte di miglioramento 7

1. Sommario

L'obiettivo di questo progetto è lo sviluppo di un'applicazione web che segua i paradigmi REST.
Per raggiungere tale scopo, abbiamo realizzato un sistema gestionale per la concessionaria AutoTorino che permette la gestione completa del catalogo veicoli e delle riparazioni.

Il dominio applicativo scelto, quindi, riguarda la gestione di una concessionaria automobilistica. Il sistema gestisce tre entità principali: gli utenti (dipendenti della concessionaria), le automobili in vendita e le riparazioni. Ogni utente autenticato può visualizzare l'intero catalogo dei veicoli disponibili, può modificarlo inserendo o eliminando i veicoli che transitano per l'azienda e può gestire le riparazioni inserendo il tipo di riparazione da effettuare, il tempo richiesto e il costo.
L'applicazione è stata sviluppata utilizzando Node.js con Express.js per il backend e HTML5, CSS e Javascript per il front-end, seguendo il principio di separazione tra client e server. Per la persistenza dei dati abbiamo utilizzato il file system nativo di Node.js, memorizzando le informazioni in formato JSON.

Il gestionale offre operazioni CRUD complete (Create, Read, Update, Delete) sia sulle automobili che sulle riparazioni, permettendo agli utenti di creare nuove inserzioni, visualizzare il catalogo, modificare i dati esistenti ed eliminare veicoli o riparazioni. Oltre alle funzionalità base, abbiamo implementato un sistema di autenticazione che protegge le operazioni sensibili, una ricerca testuale che permette di trovare veicoli per marca o modello, filtri multipli combinabili (stato del veicolo, tipo di carburante, sede di disponibilità) e un sistema di paginazione.
Particolare attenzione è stata dedicata alla validazione dei dati, implementata su due livelli: lato client per fornire feedback immediati all'utente e lato server per garantire l'integrità dei dati memorizzati.

2. Modello dei dati

Il sistema si basa su tre entità principali che si trovano in un unico file JSON (data.json).

2.1. Entità principale: Auto (Car)

L'entità auto rappresenta un veicolo in vendita presso la concessionaria e contiene i seguenti attributi:

Attributo	Tipo	Validazione
Id	numero	univoco
Marca	stringa	Non vuoto
Modello	stringa	Non vuoto
Anno	numero	$2000 \leq$ anno ≤ 2025
Prezzo	numero	$0 <$ prezzo ≤ 500000
Chilometri	numero	chilometri ≥ 0
Carburante	stringa	"Benzina", "Diesel", "Elettrica", "Ibrida"
Stato	stringa	"Nuova", "Usata", "KM Zero"
Colore	stringa	Non vuoto
Disponibile	booleano	True/false
Immagine	stringa	URL valido o vuoto

Sede	stringa	'Domio', 'Rabuiese', 'Tavagnacco', 'Zoppola', 'Pordenone', 'Monza', 'Torino', 'Modena', 'Teramo', 'Roma'
descrizione	stringa	Min. 10 caratteri

2.2. Entità secondaria: user (utente)

L'entità secondaria utente rappresenta i dipendenti della concessionaria che, tramite il login, possono entrare all'interno del gestionale.

Attributo	Tipo	Validazione
Id	numero	Non vuoto
username	stringa	Non vuoto
password	stringa	Non vuoto

2.3. Entità secondaria: riparazione

L'entità secondaria riparazione rappresenta le possibili riparazioni che devono essere fatte sull'auto quando arriva nel concessionario.

Attributo	Tipo	Validazione
Id	numero	Univoco
carId	numero	Non vuoto, deve corrispondere a un'auto esistente.
tipoLavorazione	stringa	Non vuoto
statoLavorazione	stringa	"Da fare", "In corso", "Completato"
costo	numero	Costo > 0
oreRichieste	numero	Ore richieste per la lavorazione > 0

3. Implementazione del Back-end

3.1. Architettura del Server

Il server è implementato utilizzando Express.js sulla porta 3001. La sua struttura è modulare, con una netta e chiara separazione tra le diverse responsabilità.

Come **middleware di configurazione** sono state utilizzate le funzioni express.json() (per il parsing del body delle richieste) ed express.static() (per servire i file statici del frontend). Gli **endpoint** sono organizzati in tre gruppi logici: autenticazione, gestione auto e gestione riparazioni.

3.2. Endpoint REST e Operazioni CRUD

L'API segue i principi REST con una mappatura coerente tra verbi HTTP e operazioni:

- Autenticazione

  POST/api/login: Riceve username e password e verifica le credenziali confrontandole con l'array users nel database JSON. Se le credenziali sono valide, restituisce le informazioni dell'utente e un token (corrispondente all'ID utente). Il token viene poi utilizzato nelle successive richieste autenticate tramite l'header X-Auth-Token.

- Gestione Auto:

  GET/api/cars: recupera la lista delle auto con un supporto per la ricerca testuale (parametro q) e la paginazione (limit e offset). La ricerca filtra su marca e modello utilizzando IndexOf().

  GET/api/cars/:id: restituisce i dettagli di una singola auto identificata dall'ID numerico.

  POST/api/cars: crea una nuova auto. Richiede autenticazione (quindi l'header X-Auth-Token) e valida tutti i campi obbligatori: marca, modello, anno (il quale può spaziare tra 2000 e 2025), prezzo, chilometri, carburante, stato, colore, sede e descrizione. Genera automaticamente un nuovo ID incrementale.

  PUT/api/cars/:id: aggiorna un'auto esistente. Richiede autenticazione e permette l'aggiornamento parziale dei campi, mantenendo i valori esistenti per i campi non modificati.

  DELETE/api/cars/:id: elimina un'auto dal catalogo. Richiede autenticazione.

- Gestione Riparazioni:

  Gli endpoint per le riparazioni seguono la stessa struttura CRUD delle auto, con validazioni specifiche per i campi carId, tipoLavorazione, statoLavorazione (da_fare/in_corso/completato), costo (>0) e oreRichieste (>0).

3.3. Gestione degli Status Code http

Il server utilizza i seguenti status code per comunicare l'esito delle operazioni:

- **200 OK**: per operazioni GET e PUT completate con successo.
- **201 Created**: per operazioni POST che creano nuove auto/riparazioni.
- **204 No Content**: per operazioni DELETE completate con successo.
- **400 Bad Request**: quando i dati forniti non passano la validazione o mancano campi obbligatori.
- **401 Unauthorized**: per credenziali non valide o token mancante.
- **404 Not Found**: quando una risorsa richiesta non esiste.
- **500 Internal Server Error**: per errori generici del server.

3.4. Validazione e Persistenza dei Dati

La validazione dei dati avviene interamente sul backend per garantire l'integrità del database. Ogni endpoint di creazione e modifica, quando attivato, implementa controlli specifici, quali:

- **Validazione strutturale**: verifica la presenza di tutti i campi di compilazione obbligatoria utilizzando condizioni If esplicite.
- **Validazione di range**: controlla che i valori numerici rientrino in intervalli accettabili (anno tra 2000 e 2025, prezzo tra 0 e 500000, chilometri non negativi).
- **Validazione di enumerazione**: verifica che i valori categorici (stato della macchina, carburante...) corrispondano ad opzioni predefinite utilizzando array e cicli for per il confronto.
- **Validazione referenziale**: per le riparazioni, controlla che il carId faccia riferimento ad un'auto esistente nel database.

La persistenza è gestita tramite un file JSON (data.json) che viene letto all'inizio di ogni operazione e scritto al termine delle operazioni di modifica.

4. Implementazione del Front-end

Il front-end dell'applicazione è stato sviluppato utilizzando HTML, CSS e JavaScript senza l'ausilio di librerie esterne. I tre file principali costituenti il Front-end dell'applicazione sono: **index.html**, **style.css** e **script.js**.

4.1. Viste principali

- **Header fisso**: è presente in tutte le schermate, mostra il logo "AutoTorino Italia", il titolo dell'applicazione e, dopo il login, il nome dell'utente connesso con un pulsante di logout. L'header utilizza *position: sticky* per rimanere visibile durante lo scroll.
- **Vista Login**: presenta campi per username, password e gestione degli errori con messaggi visibili. Al login riuscito, il token viene salvato e l'interfaccia passa alla vista principale.
- **Vista Catalogo Auto**: presenta una barra di filtri con ricerca testuale e select per stato, carburante e sede. La paginazione permette di navigare tra i risultati.
- **Vista Riparazioni**: mostra l'elenco delle riparazioni in formato lista.

4.2. Utilizzo della Fetch API

Tutte le comunicazioni con il backend avvengono tramite la Fetch API. Le operazioni eseguite sono **GET requests** per recuperare liste e dettagli, **POST requests** per la creazione di nuove risorse, **PUT requests** per il loro aggiornamento e **DELETE requests** per la loro eliminazione.

4.3. Modal CRUD

Per le operazioni di creazione e modifica, vengono utilizzate finestre modali (overlay che coprono lo schermo) contenenti form completi. Ci sono due modal separate: una per le auto ed una per le riparazioni.

4.4. Client-side

Il file Javascript script.js, invece, contiene tutta la logica client-side ed è organizzato in sezioni funzionali:

- **Variabili globali**: mantengono lo stato dell'applicazione (token, username, pagina corrente, ecc.);
- **Funzioni di utilità**: per mostrare notifiche, formattare numeri, ecc.;

- **Gestione autenticazione**: funzioni login() e logout();
- **Gestione visualizzazione**: funzioni per mostrare/nascondere sezioni;
- **Operazioni CRUD Auto**: funzioni per caricare, creare, modificare ed eliminare auto;
- **Operazione CRUD Riparazioni**: stessa struttura del CRUD Auto;
- **Gestioni filtri e paginazione**: logica per ricerca e navigazione tra pagine

4.5. Stile

Infine, è presente il file CSS denominato style.css, il quale contiene tutti gli stili utilizzati per la creazione di titoli, contenitori, effetti ed altro. Esso, in particolare, implementa un design responsive con grid layout per la disposizione delle card ed un sistema di colori coerente basato su tonalità di blu.

5. Conclusioni

Il progetto "AutoTorino Italia" è un'applicazione web per la gestione di una concessionaria automobilistica che permette ai dipendenti di gestire il catalogo delle auto e le riparazioni in officina.

Durante lo sviluppo abbiamo affrontato alcune sfide:

- **Gestione del file JSON:** capire come salvare e leggere i dati correttamente usando `fs.promises` per evitare la perdita di modifiche.
- **Validazione manuale:** scrivere tutti i controlli di verifica senza dimenticare nulla e usando i codici HTTP corretti è stato laborioso.
- **Interfaccia senza framework:** creare le card in JavaScript puro richiedeva di scrivere HTML dentro stringhe.
- **Login e token:** far rimanere l'utente loggato è stato complesso. Dimenticavamo spesso di inviare il token nelle richieste causando errori, e dovevamo gestire correttamente la visualizzazione delle sezioni dopo login/logout.

I punti di forza sono l'interfaccia intuitiva con navigazione a tab, filtri accessibili e feedback immediati per ogni operazione.

5.1. Funzionalità extra implementate

Sono state introdotte alcune funzionalità extra, tra cui:

- **Sistema di filtri multipli**: oltre alla ricerca testuale, sono stati implementati filtri per stato del veicolo, carburante e sede che si applicano contemporaneamente.
- **Sistema di notifiche**: feedback visivo per ogni operazione completata o fallita.
- **Badge informativi**: indicatori visivi per stato auto e disponibilità.
- **Responsive design**: l'interfaccia si adatta a diverse dimensioni di schermo.

5.2. Proposte di miglioramento

Per un'eventuale evoluzione del progetto, si potrebbe:

- Permettere il caricamento diretto di foto delle auto invece di utilizzare solo URL;
- Aggiungere dashboard con grafici per analizzare vendite, riparazioni ed inventory;
- Implementare diversi livelli di permessi (ad esempio: amministratore delegato, operatore, visualizzatore, meccanico).