Corso di Programmazione Web e Mobile

A.A. 2021-2022

Gestionale Libri di Casa

◖ Massimo, Cossi, 923088 ◗

Autore: Massimo Cossi

Ultima modifica: 10 Febbraio 2022 - versione 1.0.0

Prima modifica: 04 Giugno 2021

Gestionale Libri di Casa

1. Introduzione

Ci sono case piene di librerie che contengono centinaia di libri, così tanti da non ricordarsi quanti e quali si hanno. L'idea di questo progetto è far sì che si tenga traccia di questi libri.

Questa applicazione Web consente di creare un elenco dei libri che si hanno in casa, inserendoli tramite una ricerca nell'API di Google Books mediante il titolo o l'autore del libro, oppure inserendo manualmente tutti i dati, compresi la trama e la copertina del libro; è inoltre possibile indicare se il libro è già stato letto o meno.

L'applicazione necessita di una registrazione e successivamente di un'autenticazione; i dati richiesti sono e-mail, username e password; questi dati e i dati dei libri vengono salvati su un database relazionale Mysql. Una volta salvati, i libri si possono modificare, cancellare e filtrare per titolo autore, tipologia e se sono stati letti o meno.

La parte front-end è stata fatta tramite Vue.cli e la parte backend tramite un server Node.js con il framework Express, per lo stile è stato usato il framework css Bulma.

1.1. Breve analisi dei requisiti

1.1.1. Destinatari

Capacità e possibilità tecniche.

I destinatari di questo progetto sono gli amanti dei libri, quindi potenzialmente chiunque. La web app consente di sapere sempre i libri in possesso dell'utente, sia quelli cartacei, che quelli in formato digitale (e-book e audiolibri). L'app non necessita di particolari conoscenze, in quanto presentando un'interfaccia semplice è utilizzabile da chiunque possieda un pc o uno smartphone e che abbia un minimo di conoscenza nell'impiego di una qualsiasi app. L'applicazione guida l'utente in maniera semplice nell'inserimento di tutti i dati necessari e nel caso in cui l'utente ometta qualche informazione fondamentale, ne viene richiesto l'inserimento prima di proseguire.

È un'app progettata per essere responsive, è accessibile sia mediante smartphone, che mediante pc, necessita unicamente di una connessione a internet; l'utente deve solamente aprire un qualsiasi browser e digitare il seguente indirizzo:

https://homes-book.herokuapp.com/login

Motivazione.

La motivazione alla base della Web Application è di intrattenimento. L'utente che si interfaccia all'app si troverà ad usarla in maniera meccanica ed automatica.

Basando l'analisi del pubblico a cui è rivolta l'applicazione web sul modello di ricerca dell'informazione, è possibile individuare come, nel caso di utilizzo del servizio web da un utente consapevole, le azioni saranno immediate in quanto l'utente che sceglie di utilizzare questo servizio saprà già di dover ricercare un libro e salvarlo nella sua libreria virtuale. Nel caso invece di un utente non consapevole, questo avrà comunque a disposizione un'interfaccia amichevole che lo porterà senza difficoltà ad eseguire le azioni e a capire il senso dell'applicazione. Entrambe le tipologie di utenti sono identificate come "attivi".

Essendo un applicativo con uno scopo ben preciso, che è quello dell'archiviazione dei propri libri, si presuppone che l'interesse da parte dell'utente nel compiere un'azione corretta sia alto, con un approccio di totale concentrazione sul compito da svolgere.

1.1.2. Modello di valore

Il servizio che dà valore all'applicazione è quello del collegamento all'API di Google Books che favorisce la velocità di ricerca e inserimento dei libri. L'utente, una volta scelto il libro di interesse, lo può velocemente cercare nella barra di ricerca offerta dall'applicazione, la quale rimanderà la visualizzazione di dieci libri, tra i quali l'utente potrà scegliere quello da lui pensato ed inserirlo velocemente nella sua libreria; a questo punto un libro inserito in libreria potrà essere facilmente modificato o eliminato.

1.1.3. Flusso dei dati

I contenuti principali di questa app sono i libri, questi si possono aggiungere tramite una ricerca per autore o titolo nella barra di ricerca, la quale farà una richiesta http GET all' API di Google Books, oppure sono inseribili manualmente tramite un form. Le caratteristiche che vengono indicate per ciascun libro sono:

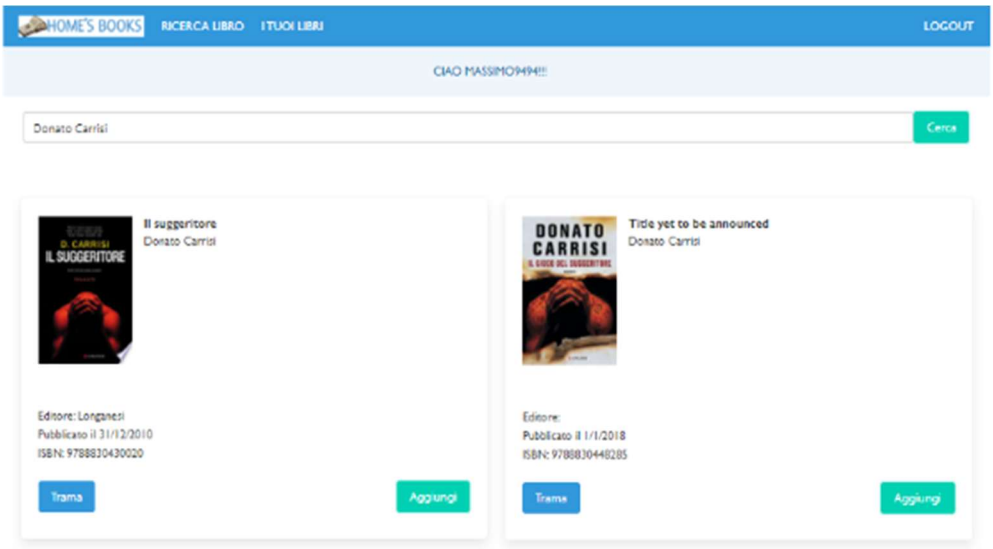
- Titolo (obbligatorio),
- Autore/i,
- Copertina,
- Editore,
- Data di pubblicazione,
- ISBN.

Queste caratteristiche sono già presenti in fase di ricerca, mentre una volta selezionato il libro di interesse, l'app chiederà di aggiungere ulteriori informazioni, entrambe obbligatorie:

- Tipologia (Cartaceo, Ebook o AudioLibro),
- Se è stato letto o meno.

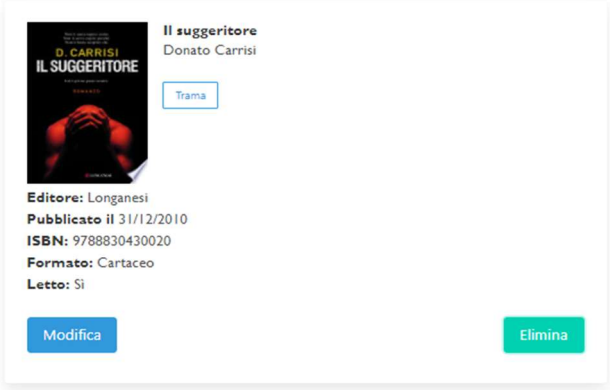
Ottenere i contenuti.

I contenuti possono essere ottenuti grazie alla barra di ricerca, tramite la quale l'app salva l'input in una variabile, dopodiché viene eseguita la richiesta GET concatenando la variabile con l'URL di Google Books, che restituisce un oggetto in formato JSON; l'app provvede a posizionare le istanze dell'oggetto nei vari campi citati sopra (titolo, autore, etc.) in una card di Bulma. Una volta scelto il libro, l'app richiede la tipologia e se il libro è stato letto o meno. Nella schermata qui riportata viene mostrato come viene effettuata la ricerca e come vengono formattati i risultati:

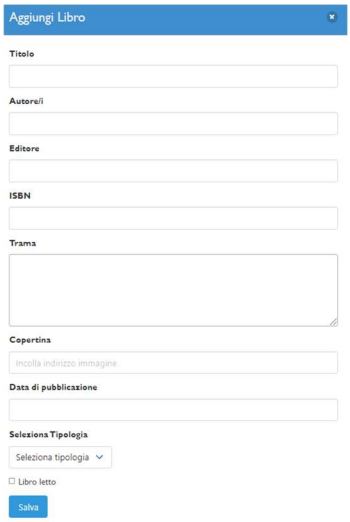


Richiesta effettuata dall'app una volta scelto il libro:

Il libro nella libreria personale viene salvato tramite una richiesta POST al server:



I contenuti possono anche essere inseriti tramite un form in una modale che viene attivata nella propria libreria con il bottone “Aggiungi”. Nella schermata si vede il form per l’aggiunta del libro:



I campi compilati nel form vengono mandati con una richiesta POST al server.

Archiviare e organizzare i contenuti

Sia nel caso di ricerca che di inserimento manuale, dopo la richiesta POST al server, quest’ultimo provvede a salvare i dati dei libri in un database relazionale MySQL.

Il server Node si serve di Sequelize per memorizzare i dati nel database, questo è un ORM (Object Relational Mapping Library), che crea dei modelli e provvede a mapparli nelle tabelle del database.

Modello creato per i libri e mappato nel database:

```
const Sequelize = require('sequelize');
const sequelize = require('../utils/database');


const Book = sequelize.define(
    'book',
    {
        id : {
            type: Sequelize.INTEGER,
            autoIncrement: true,
            allowNull : false,
            primaryKey: true
        },
        titolo: {
            type: Sequelize.STRING,
            allowNull : false,
        },
        autori: {
            type: Sequelize.STRING,
            allowNull : true,
        },
        trama: {
            type: Sequelize.TEXT,
            allowNull : true,
        },
```

id	titolo	autori	trama
205	L'ipotesi del male	Donato Carrisi	Mila Vasquez ingaggia una partita con le tenebr...
225	Io sono l'abisso	Donato Carrisi	(NULL)
265	Il tribunale delle anime	Donato Carrisi	Questa è la storia di un male antico ed eterno e...
335	Ragionevoli dubbi	Gianrico Carofiglio	L'attesissimo ritorno di Guido Guerrieri, l'avvoca...
405	La casa delle voci	Donato Carrisi	(NULL)
425	Il suggeritore	Donato Carrisi	«Il suggeritore è fantastico. Un concentrato puri...
445	Il suggeritore	Donato Carrisi	«Il suggeritore è fantastico. Un concentrato puri...

Pubblicare i contenuti

Tutti i contenuti dell’applicazione vengono recepiti attraverso l’API in formato JSON. I dati così ottenuti, opportunamente formattati, si prestano ottimamente per la pubblicazione nelle viste disponibili sulla piattaforma. Anche lo scambio di dati tra il frontend e il backend dell’applicazione avviene in formato JSON. Questa omogeneità dei contenuti riduce notevolmente il lavoro di sviluppo non essendo necessarie particolari conversioni.

1.1.4. Aspetti tecnologici

Gli aspetti tecnologici per questo progetto web si basano principalmente sull'utilizzo delle seguenti tecnologie:

- API (Google Books),
- Un database relazionale (MySQL),
- Vue.cli per l'interfaccia utente,
- Express per il server node.js,
- Bulma per lo stile dell'applicazione,
- Fetch API per la richiesta di risorse da parte dell'interfaccia utente.

Inoltre sono presenti altre librerie come:

- Sequelize, per l'inserimento e la richiesta dei dati da parte del server verso il database,
- Jsonwebtoken, per il processo di autenticazione,
- Helmet, per proteggere l'applicazione da alcune vulnerabilità web note configurando le intestazioni HTTP in modo appropriato.
- Express-validator, per la validazione delle richieste lato server.

Google Books API

L'utente ha la possibilità di inserire in uno spazio di ricerca il titolo o l'autore (o entrambi) dai quali è possibile ricavare un elenco dei libri di interesse.

Una volta effettuato l'invio, entra in gioco l'esecuzione della funzione asincrona getBooks(), la quale legge la variabile "query", la concatena con il link per Google Books ed inoltra la richiesta con la sintassi async/await. La funzione getBooks() è la seguente:

```
    async getBooks() {
      // sintassi async/await
      try {
        const result = await fetch('https://www.googleapis.com/books/v1/volumes?q='+this.query);
        const data = await result.json();
        this.books = data.items;
      }catch(error) {
        console.log('errore: ', error);
      }
    }
  }
}
```

La risposta viene fornita in formato JSON. A sua volta vengono creati i tag html necessari alla visualizzazione nel componente Book.vue.

Database MySQL

I libri scelti con la ricerca o inseriti manualmente con il form vengono salvati in una tabella in un database MySql, in questo vengono salvati anche i dati dell'utente che effettua la registrazione all'app. La tabella degli utenti è la seguente:

id	username	email	password	createdAt	updatedAt
5	Massimo9494	massimocossi94@gmail.com	$2a$12$9CPkcPFoIaUK1Dw72.dlX.t1f7Iz.GQHlX...	2022-01-30 09:43:47	2022-01-30 09:43:47

La tabella dei libri è la seguente:

id	titolo	autori	trama	thumbnail	dataPubblicazione
25	La donna dei fiori di carta	Donato Carrisi	Il monte Fumo è una cattedrale di ghiaccio, tea...	http://books.google.com/books/content?id=Abd...	2012-05-03T00:00:00+02:00

editore	isbn	tipologia	createdAt	updatedAt
Longanesi	9.788.830.434.271	Cartaceo	2021-07-03 16:48:43	2021-07-03 17:21:41

Con sequelize si crea una relazione “uno a molti” tra gli utenti e i libri, cioè un utente può avere più libri, un libro può essere solo di un utente e la tabella risultante è quella dei libri con l’aggiunta della colonna userId.

Vue.cli

VueJs è un framework Javascript open source per la creazione di interfacce utente. Si tratta di un framework progressivo, ovvero è pensato per essere adottabile in modo incrementale a differenza dei framework monolitici. Vue è usato principalmente per lo sviluppo front-end. Quest’ultimo offre un tool CLI (Command Line Intreface), che aiuta a creare velocemente un progetto; il pacchetto si chiama vue-cli ed è scaricabile facilmente da npm tramite il comando “npm i -g vue-cli”. Dopo averlo installato, basta lanciare il comando “vue init webpack <NOME_APP>” e verrà creata un’applicazione già funzionante.

Il framework utilizza il virtual DOM, cioè crea un DOM virtuale che viene sincronizzato con il DOM reale e modificando quello virtuale, le modifiche vengono automaticamente trasferite su quello reale.

Vue utilizza i componenti che sono blocchi di codice riutilizzabile all’interno della Web App. In Vue viene scritto tutto il codice in un solo file, questo contiene un template HTML separato dalla logica JavaScript, dando l’opportunità di usare HTML. Tutti i componenti hanno una loro istanza isolata; ciò significa che si possono trasmettere i dati di un componente padre internamente a un figlio, ma questi vanno passati come parametri tramite un’opzione offerta dal componete chiamata props.

Nei template dei componenti si utilizzano delle direttive come:

- v-for che dà la possibilità di renderizzare un elemento più volte, in base al modello dei dati che gli viene passato; si può usare questa direttiva per iterare un array e visualizzarne i dati. Nell’app viene usato per esempio per iterare l’array che contiene tutti i JSON dati in risposta dall’API.
- v-model che dà la possibilità di utilizzare il two-way data binding: tramite questa tecnica, ogni volta che l’input viene aggiornato, la variabile contenuta nel v-model di conseguenza si aggiorna. Nell’app viene usato per mettere la ricerca effettuata dall’utente in una variabile di nome “query”.
- v-if che permette di svolgere delle operazioni logiche e di confrontare delle variabili direttamente nel template HTML. Nell’app viene usato per mostrare o meno il pulsante “segna come letto” nell’elenco dei libri della libreria personale.
- v-bind che viene utilizzato per collegare un valore ad un attributo html; per questo si possono usare i “:” prima dell’attributo che si vuole collegare a una variabile. Nell’app viene usato per disabilitare il pulsante “Aggiungi Libro” se non viene scelta la tipologia o non si è detto se il libro è stato letto o meno.

Nella parte logica si usa JavaScirpt, il codice è suddiviso in più parti, ad esempio:

- Data, dove si mettono tutte le variabili,
- Methods, dove si mettono tutte le funzioni,
- Created, dove si mettono le funzioni che sono chiamate nel momento della creazione di quella pagina,
- Computed, per generare dei nuovi valori che in qualche modo sono delle trasformazioni delle proprietà base dell'oggetto "data" o che derivano da esso. Nell'applicazione è usato per il filtraggio dei libri.

Express

È stato utilizzato un server NodeJS che tramite libreria Express espone un API rest che riceve le richieste del client, le elabora, interroga il database quando necessario e risponde con un oggetto JSON.

Bulma

Bulma è framework CSS basato su Flexbox per la creazione di progetti responsive e mobile-first. Questo offre degli elementi di stile già fatti, utilizzabili direttamente nel codice HTML attraverso delle classi specifiche, queste reperibili direttamente dalla documentazione.

Fetch API con sintassi async/await

Nella parte frontend dell'applicazione, per richiedere e dare le risorse al server o all'API, è stata usata l'API fetch; questa è direttamente disponibile con il linguaggio JavaScript. "fetch()" ha una sintassi semplice e integrata nel modello ad oggetti di JavaScript. L'API prevede una gestione delle chiamate asincrone basata sulla restituzione di una promise a seguito di una richiesta; quando la richiesta viene completata, la promise viene risolta con l'oggetto Response, altrimenti, se la richiesta non riesce a causa di problemi, la promise viene rifiutata. async/await si adatta perfettamente a "fetch()" perché semplifica il lavoro con le promise. Si contrassegna la funzione con async e dentro a questa ci sarà "await fetch()" che avvia una richiesta http, la funzione asincrona viene sospesa fino al completamento della richiesta.

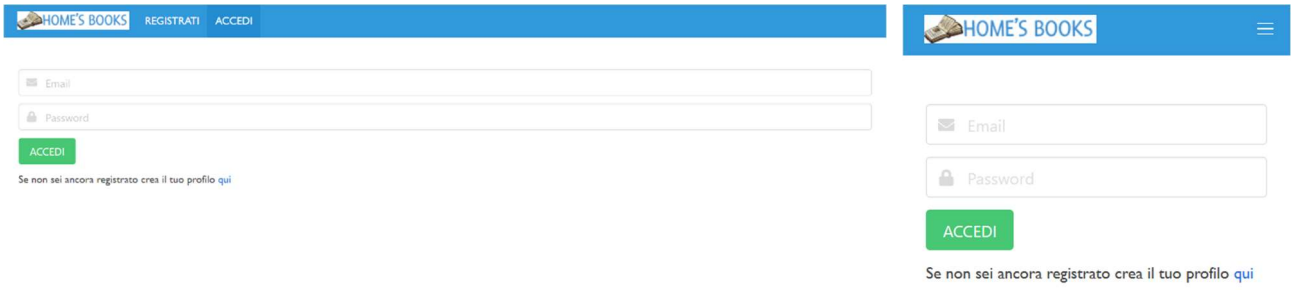
Sequelize

Per la creazione delle tabelle, per metterle in relazione fra di loro e per interrogarle, si è utilizzato Sequelize. Nell'applicazione sono stati creati due modelli, che sono i libri e gli utenti, questi sono stati messi in relazione fra loro.

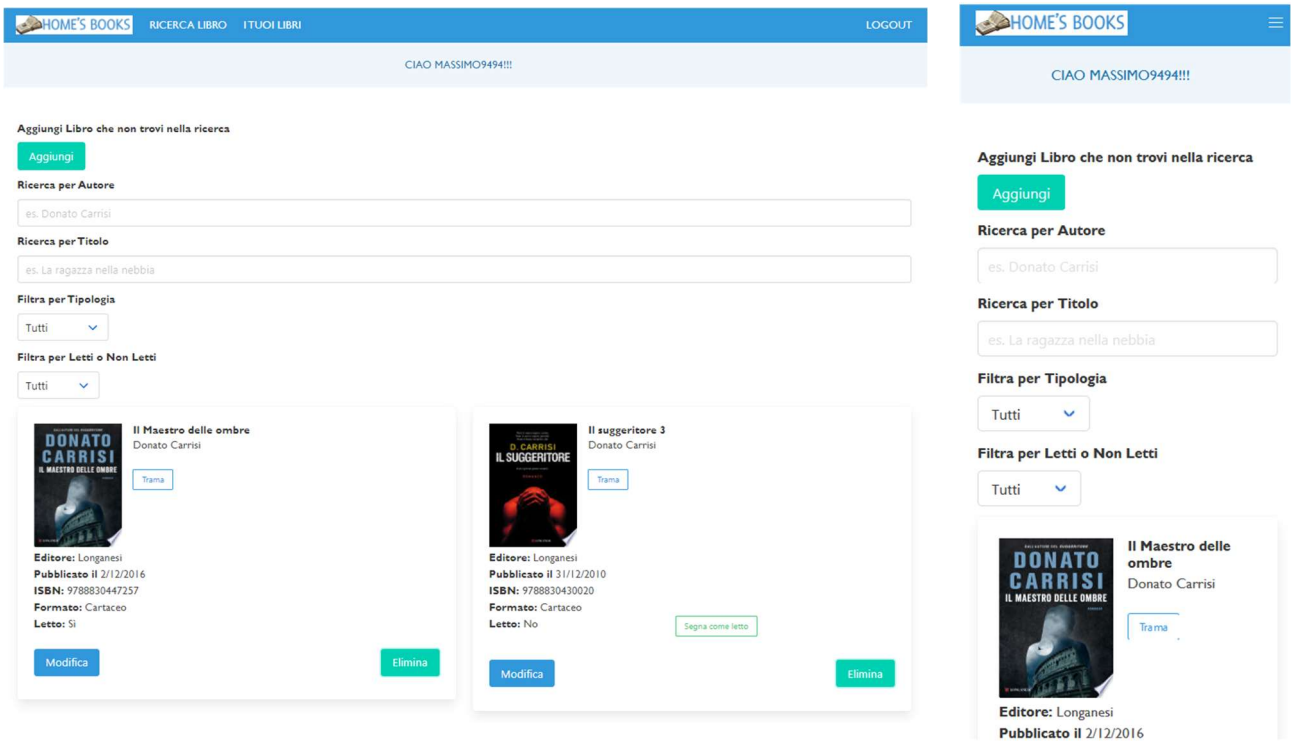
2. Interfacce

Un utente che si interfaccerà per la prima volta all'applicazione si dovrà registrare:

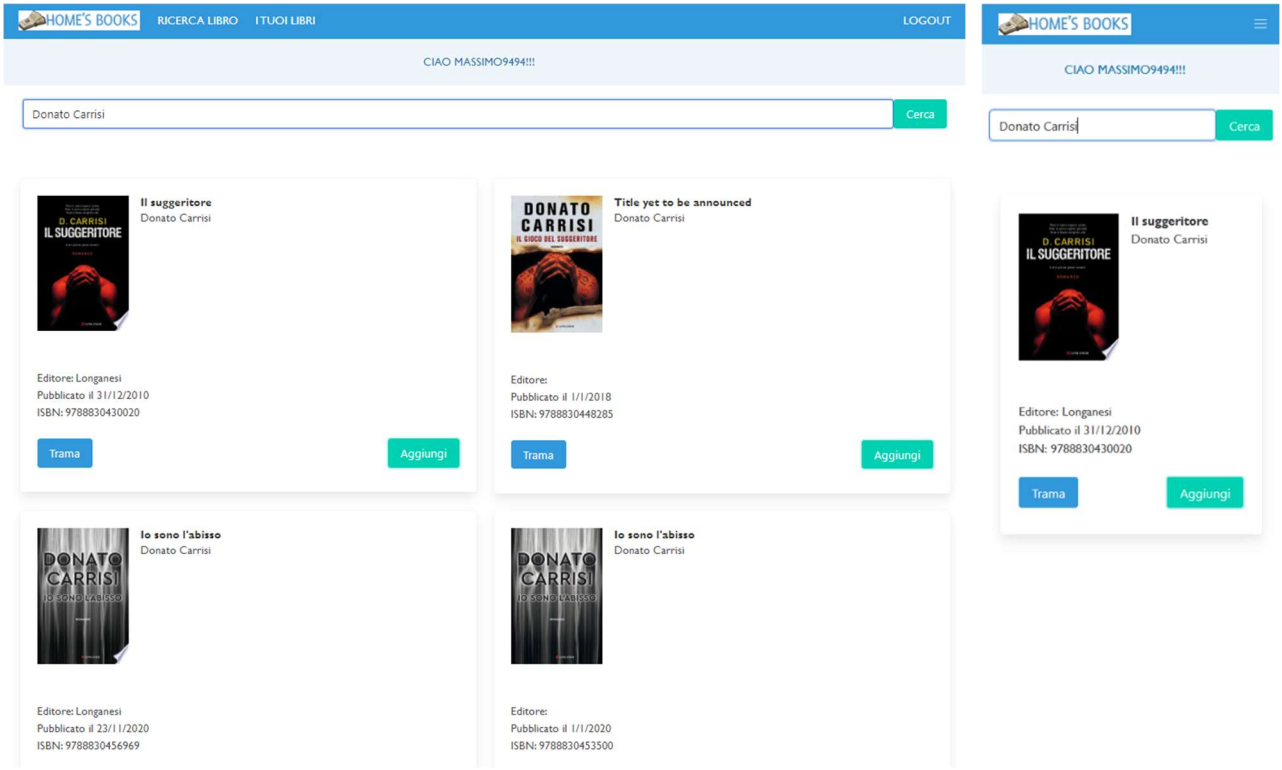
Una volta registrato dovrà eseguire il login:



Eseguito il login l'applicazione rimanderà l'utente alla pagina della propria libreria personale:



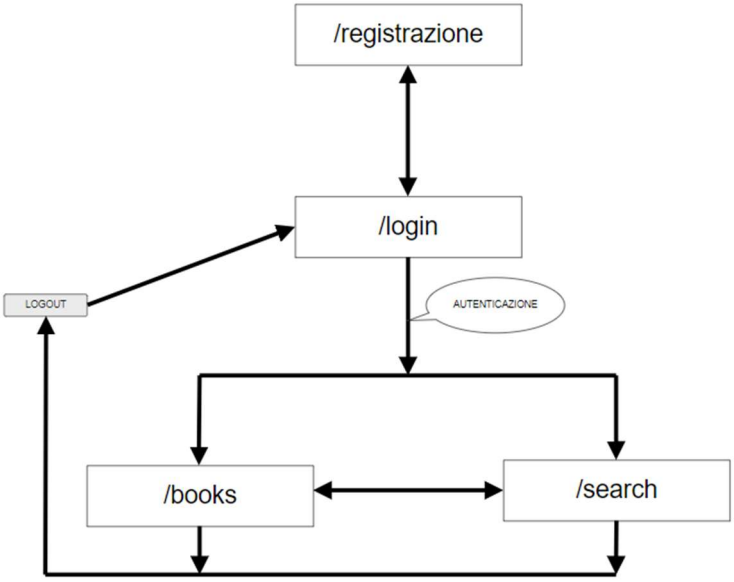
Se l'utente vorrà aggiungere un libro tramite la ricerca offerta dall'app, cliccherà "RICERCA LIBRO" nella barra di navigazione in alto e verrà rimandato alla ricerca:



Queste sono le principali interfacce dell'applicazione, nelle immagini a sinistra si ha la vista da PC desktop e in quelle a destra si ha la vista da mobile, questo dimostra come l'applicazione è usufruibile da qualsiasi dispositivo.

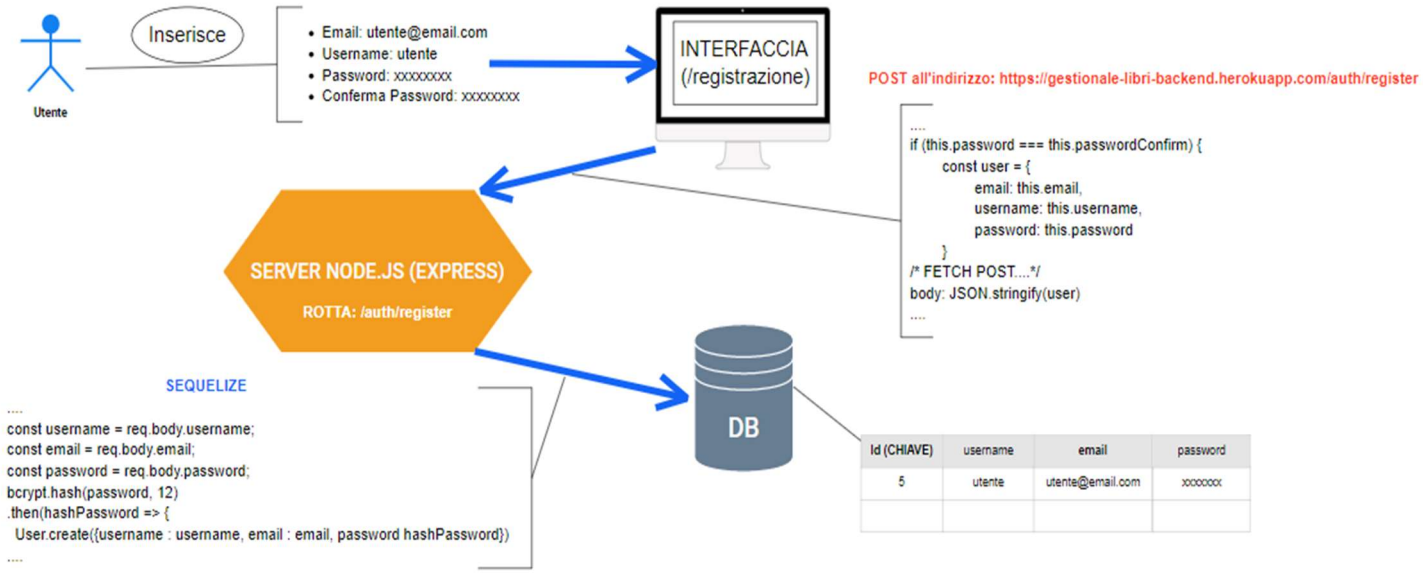
3. Architettura

3.1. Diagramma dell'ordine gerarchico delle risorse

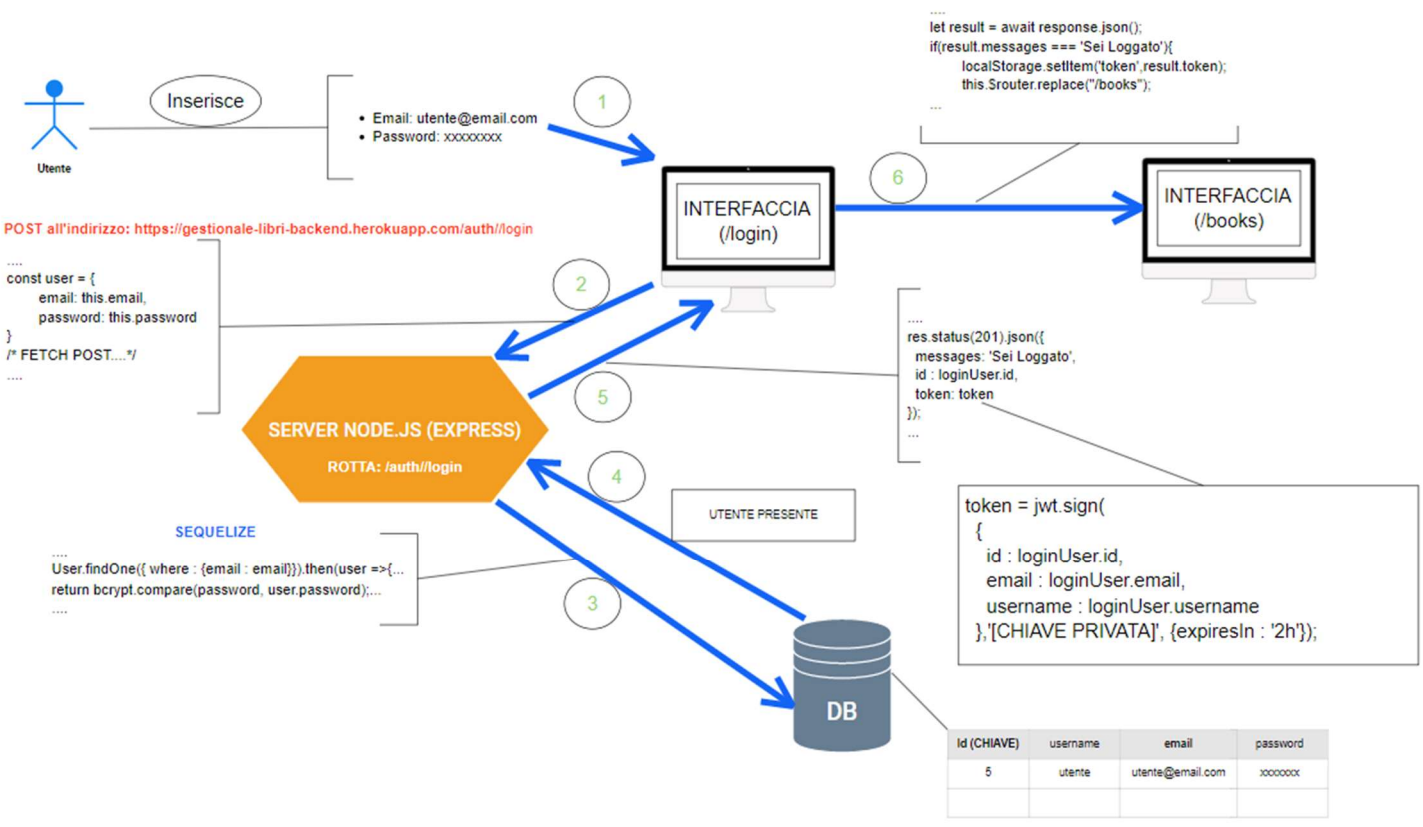


3.2. Descrizione delle risorse

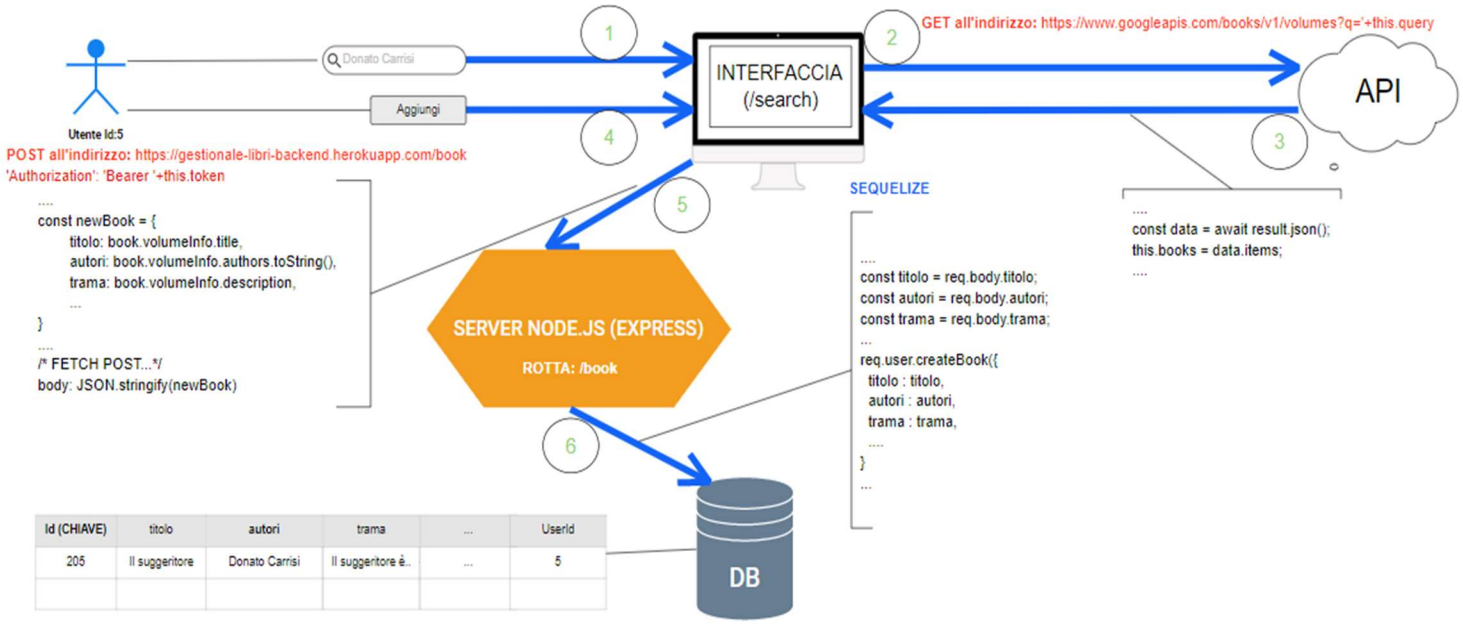
Fase di registrazione:



Dopo la registrazione l'utente procederà alla fase di login, dopodiché l'applicazione salverà un token nel local storage per tenere attiva la sessione:

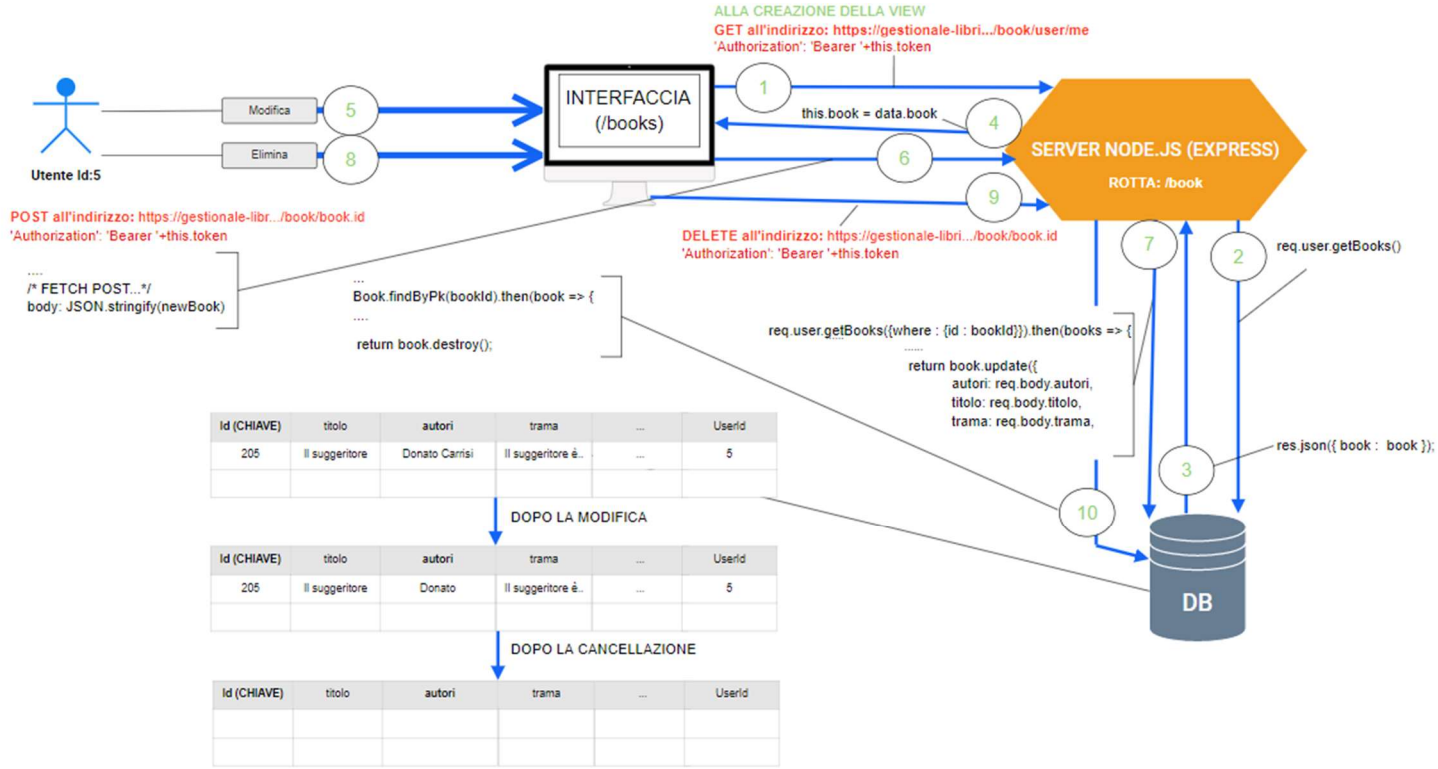


Fase di ricerca di un libro:



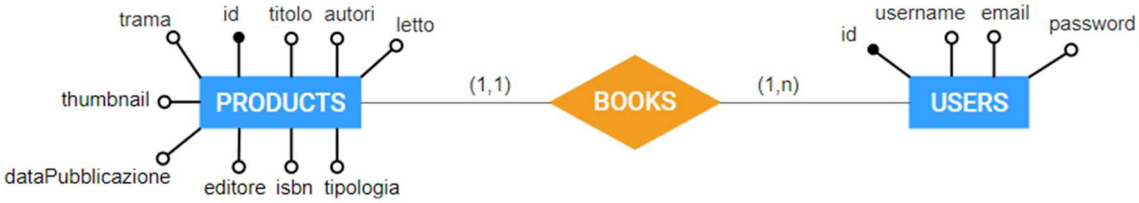
L'aggiunta di un libro in manuale ha lo stesso tipo di schema, l'unica differenza è che l'utente al posto che cliccare sul pulsante "Aggiungi" compila un form con tutti i dati e non ci sarà dunque la richiesta di GET all'API.

Infine si ha la libreria personale dove ci saranno i libri aggiunti dall'utente, in questa view si potrà aggiungere i libri manualmente, filtrarli, modificarli oppure eliminarli:



Dopo il login tutte le richieste dell'interfaccia al server passeranno da un middelware che controllerà l'HEADER con il campo "Authorization", l'eventuale presenza del parametro "Baerer" con il token di sessione, se questo non ci fosse il server restituirebbe uno stato 401 con un messaggio in formato JSON "Non Autorizzato".

3.3. Altri diagrammi



SCHEMA LOGICO

Relazione	Attributi
PRODUCTS	**Id**, titolo, trama, autori, letto, thumbnail, dataPubblicazione, editore, isbn, tipologia
USERS	**Id**, email, username, password
BOOKS	**Id**, titolo, trama, autori, letto, thumbnail, dataPubblicazione, editore, isbn, tipologia, userId

VINCOLI DI INTEGRITA' REFERENZIALE

Chiave esterna	Referenzia
BOOKS.userId	USERS.Id

4. Codice

4.1. HTML

Template HTML della libreria personale:

```
<template>
   <div>
      <div class="modal" :class="{ 'is-active': aggiungiModale }">
         <div class="modal-background"></div>
            <div class="modal-card">
               <header class="modal-card-head">
                  <p class="modal-card-title">Aggiungi Libro</p>
                  <button @click="aggiungiModale = false" class="delete"
                       aria-label="close"></button>
               </header>
               <section class="modal-card-body">
                  <div class="field">
                     <label class="label">Titolo</label>
                        <div class="control">
                           <input class="input" type="text" v-model="titolo">
                        </div>
                   </div>
                   <div class="field">
                      <label class="label">Autore/i</label>
                      <div class="control">
                         <input class="input" type="text" v-model="autori">
                      </div>
                    </div>
                    ......
                    <button class="button is-info" @click="createBook"
                        :disabled="tipologia==='Seleziona tipologia'">
                           Salva
                     </button>
               </section>
         </div>
      </div>
```

```
        <section class="section">
           <label class="label">Aggiungi Libro che non trovi nella ricerca</label>
           <button class="button is-primary"
                 @click="aggiungiModale = true">Aggiungi
           </button>

           <div class="field Ricerca">
                <label class="label">Ricerca per Autore</label>
                <div class="control">
                    <input class="input" type="text" v-model="ricercaAutore"
                         placeholder="es. Donato Carrisi">
                </div>
                ......
            </div>
            <div class="field">
                <label class="label">Filtra per Tipologia</label>
                <div class="control">
                    <div class="select">
                        <select  @change="onOptionChange">
                            <option value="All">Tutti</option>
                            <option
                                v-for="option in options"
                                :key="option.value"
                                :value="option.value"
                            >{{ option.label }}</option>
                        </select>
                    </div>
                </div>
            </div>
            ......
            <div class="modal" :class="{ 'is-active': confermaModal }">
                <div class="modal-background"></div>
                <div class="modal-card">
                    <section class="modal-card-body"
                      :class="{'conferma' : libroSuccesso,
                      'presente' : libroGiaPresente}">
                        <h1 class="presenzaLibro">{{ libroPresente }}</h1>
                        <button  @click="confermaModal = false"
                         class="button okbutton">
                            <p class="subtitle">OK</p>
                        </button>
                    </section>
                </div>
            </div>

            <div v-if="libriFiltrati.length > 0" class="columns is-multiline">
                <div class="column is-6" v-for="book in libriFiltrati" :key="book.id">
                    <my-book :book="book"
                       @deleteBook="onDeleteBook($event)"
                       @updateBook="onUpdateBook($event)"
                       @readBook="onReadBook($event)">
                    </my-book>
                </div>
            </div>
            <h2 class="title is-2" v-else>Nessun libro trovato.</h2>
        </section>
    </div>
</template>
```

4.2. CSS

Regole custom della libreria personale:

```
<style>
    .conferma{
        background: hsl(153, 53%, 53%);
        border-radius: 10px;
        display: flex;
        flex-flow: column;
        justify-content: space-around;
    }
    .presente{
        background: hsl(348, 86%, 61%);
        border-radius: 10px;
        display: flex;
        flex-flow: column;
        justify-content: space-around;
    }
    .presenzaLibro{
        font-size: 1.5rem;
    }
    .button{
        margin-bottom: 10px;
    }
    .Ricerca .control{
        margin-bottom: 10px;
    }
</style>
```

4.3. Google Books API

La chiamata getbooks è stata illustrata precedentemente, qui è come vengono predisposte nel componente Book.vue le informazioni prese dall'API:

```
<div class="card" v-if="book">
    <div class="card-content">
        <div class="media">
            <div class="media-left" v-if="book.volumeInfo.imageLinks">
                <figure class="image is-68x68">
                    <img :src="book.volumeInfo.imageLinks.thumbnail"
                       :alt="book.volumeInfo.title">
                </figure>
            </div>
            <div class="media-content">
                <p class="title is-6">{{ book.volumeInfo.title }}</p>
                <p class="subtitle is-6"
                  v-if="book.volumeInfo.authors">
                  {{getAuthors(book.volumeInfo.authors) }}</p>
            </div>
        </div>
        <div class="content">
            Editore: {{ book.volumeInfo.publisher }}<br/>
            Pubblicato il
            {{ new Date(book.volumeInfo.publishedDate).toLocaleDateString() }}<br/>
            ISBN: {{ getISBN(book.volumeInfo.industryIdentifiers) }}<br/>
        </div>
        <div class="cardButton">
            <button class="button is-primary is-info"
                @click="tramaModal = true">Trama</button>
            <button class="button is-primary is-focused"
                @click="aggiungiModal = true">Aggiungi</button> </div>
```

4.4. Fetch API JavaScript

Funzione asincrona createBook che viene chiamata nel momento che si clicca sul pulsante "Salva" nel form di aggiunta manuale di un libro:

```
async createBook() {
    let letto = document.getElementById('letto2');
    if (letto.checked)
        this.letto='Sì';
    else
        this.letto='No';
    const newBook = {
        titolo: this.titolo,
        autori: this.autori,
        ......
    };
    // Controllare se libro è già presente nella libreria
    let toAdd = true;
    const bookresponse = await fetch(BOOKS+'/user/me', {
        method: 'GET',
            headers: {
                'Authorization': 'Bearer '+this.token
            },
    });
    const bookresult = await bookresponse.json();
    for(let i=0; i<bookresult.book.length; i++ ){
        if (bookresult.book[i].titolo === newBook.titolo
             && bookresult.book[i].autore === newBook.autore
             &&  bookresult.book[i].tipologia === newBook.tipologia) {
            this.confermaModal = true;
            this.libroGiaPresente = true;
            this.libroSuccesso = false;
            this.libroPresente = 'Libro già presente in libreria'
            toAdd = false;
        }
    }
    if (toAdd) {
        const response = await fetch(BOOKS, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'Authorization': 'Bearer '+this.token
            },
            body: JSON.stringify(newBook)
        });
        let result = await response.json();
        if(result.messages !== 'Success Operation'){
            for(let i = 0; i<Object.keys(result.error).length; i++){
                if(result.error[i].msg == 'Titolo Maggiore di 3 caratteri'){
                    this.confermaModal = true;
                    this.libroGiaPresente = true;
                    this.libroSuccesso = false;
                    this.libroPresente = 'Titolo obbligatorio'
                }
                ......
            }
        }else{
            this.confermaModal = true;
             ......
        }
    }
}
```

4.5. Node.js (Express)

Nella cartella routes e nel file books.js:

```
router.post('/book',isAuth,
    [
        body('titolo').trim()
        .isLength({ min : 3})
        .withMessage('Titolo Maggiore di 3 caratteri')
        .exists().withMessage('Il Titolo è richiesto'),
        body('tipologia').trim().not().isEmpty()
        .withMessage('tipologia non vuota')
    ]
,bookController.createBook);
```

Nella cartella controllers e nel file books.js:

```
exports.createBook = (req, res, next) => {
    const errors = validationResult(req);
    if(!errors.isEmpty()){
        return res.status(422).json({
            message : 'Errore input',
            error : errors.array()
        });
    }
    const titolo = req.body.titolo;
    const autori = req.body.autori;
    const trama = req.body.trama;
    const thumbnail = req.body.thumbnail;
    const dataPubblicazione = req.body.dataPubblicazione;
    const editore = req.body.editore;
    const isbn = req.body.isbn;
    const tipologia = req.body.tipologia;
    const letto = req.body.letto;
    //INSERT NEL DATABASE
    req.user.createBook({
        titolo : titolo,
        autori : autori,
        trama : trama,
        thumbnail : thumbnail,
        dataPubblicazione : dataPubblicazione,
        editore : editore,
        isbn : isbn,
        tipologia : tipologia,
        letto: letto,
    }).then((book) => {
        res.status(201).json({
            messages: 'Success Operation',
            book : book
        });
    }).catch( err => {
        console.log('ERRORE CREAZIONE LIBRO',err);
    });
};
```

5. Conclusioni

Non avendo mai realizzato nessun progetto nell'ambito dello sviluppo web e non essendo io uno sviluppatore web di professione, nella stesura del progetto mi sono interfacciato con numerose sfide, le quali però hanno fatto affiorare in me passione e soddisfazioni. Creare questo progetto mi ha consentito di apprendere in modo approfondito le basi dello sviluppo web.

Il progetto è perfettamente funzionante, anche se potrebbero essere aggiunte delle migliorie, sia nella parte grafica che in quella funzionale, ad esempio potrebbero essere aggiunte le seguenti funzionalità:

- Visionare il profilo di altri utenti;
- Aggiungere "like" ai libri altrui;
- Chiedere l'amicizia ad altri utenti.

Quindi sarebbe bello creare una sorta di Social Network che avesse come tema quello dei libri.

La parte di autenticazione potrebbe essere migliorata mettendo la validazione dell'e-mail inviandone una di conferma, dalla quale l'utente dovrebbe seguire un link per confermare l'autenticità della propria e-mail, oppure introducendo l'autenticazione mediante Google per velocizzare ancora di più il processo.

Nel progetto ho scelto di dividere la parte frontend e quella backend per poter capire bene come svilupparle e studiarle; infatti, il server node con express potrebbe essere usato benissimo da qualsiasi altra applicazione frontend.

6. Nota bibliografica e sitografica

(1) W3School https://www.w3schools.com/

(2) Vue.cli https://italiancoders.it/vuejs-quickstart-con-vue-cli/

(3) Sequelize https://sequelize.org/v6/

(4) Node.js https://www.udemy.com/course/nodejs-corso-completo-server-rest-api-back-end/

(5) Vue.js https://vuejs.org/