Sviluppo Applicazione

Matteo Girardi, Vasile Donmovil

Gruppo T45

2023

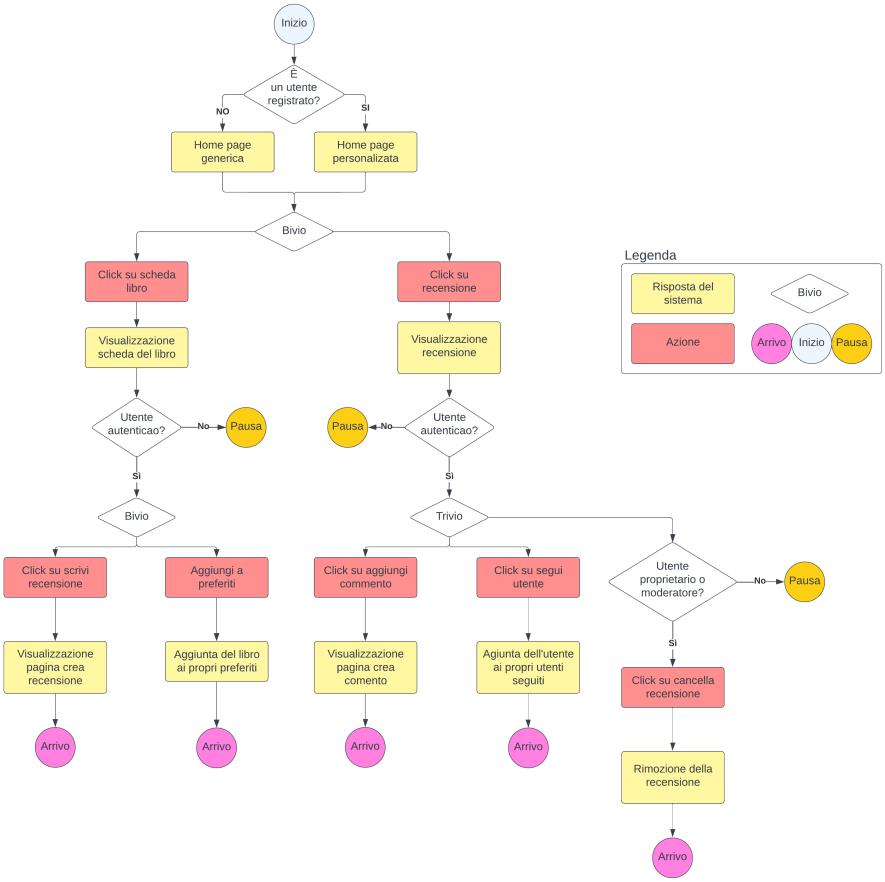
Contents

1 Scopo del documento

Nel documento corrente vengono riportate ulteriori e definitive informazioni riguardo allo sviluppo del sito. Nello specifico, presenta tutti gli artefatti necessari per il login, la ricerca e la creazione delle recensioni. In partenza viene analizzato lo User-flow legato ad un utente registrato, dopodiché vengono analizzate le API (tramite l'API Model) per la creazione, modifica e login di un profilo, le strutture dati, la visualizzazione, creazione e modifica di una recensione. Per ogni API che è stata utilizzata, vengono presentate descrizione delle funzionalità, documentazione e test utilizzati In ultima istanza vengono fornite informazioni del Git Repository e, infine, il deployment del servizio.

2 User-Flows

In questa sezione vengono riportati gli “user-flows” dell’utente registrato (che quindi può anche essere moderatore). Nella figura sottostante viene mostrato lo user-flow relativo alle azioni disponibili dall’homepage dagli utenti: visualiz- zazione, creazione recensione e aggiunta del libro ai preferiti; in aggiunta anche la rimozione di una recensione e/o commento se l’utente è l’autore o se ha il ruolo di moderatore. Lo schema utilizza la notazione “Pausa” quando l’azione non è disponibile e la notazione “Arrivo” quando viene raggiunta l’ultima azione disponibile per quel ramo. Viene, inoltre, presentata una legenda che descrive i simboli utilizzati.

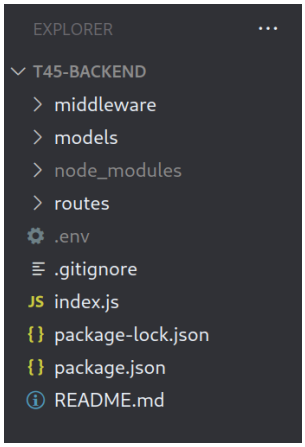


3 Application Implementation and Documentation

Nei precedenti documenti sono stati identificati tutti i requisiti funzionali e non funzionali dell'applicazione. Nella sezione precedente sono state descritte le features messe a disposizione di un Utente nel suo flusso applicativo. Nella seguente sezione vengono analizzati i software e i tools di sviluppo. L'applicazione è stata sviluppata utilizzando SvelteKit (front-end) e Express (back-end). Per la gestione e conservazione dei dati è stato utilizzato MongoDB, per l'autenticazione abbiamo scelto JWT (jsonwebtoken).

3.1 Project Structure

In questa sezione viene presentata la struttura del back-end, illustrata nell'immagine successiva.



Nella root del progetto si trova il file principale che esegue il server e la connessione al database, avvalendosi di vari file presenti nelle altre cartelle. Sempre nella root sono presenti i vari file di configurazione del server necessari alla corretta esecuzione del server.

La cartella principale contiene il file "index.js" che viene eseguito all'avvio del backend. La cartella principale contiene anche una serie di sotto-cartelle.

- **Routes** contiene le funzioni che elaborano le richieste costruendo le risposte da inviare al front-end. Sono anche specificati gli handler, che si occupano di gestire le chiamate ai vari endpoint del backend.
- **Middleware** contiene funzioni di supporto agli handlers in Routes.

- **Models** contiene le strutture dati utilizzate, ossia le "Schemas" di Mongoose. Utilizziamo gli schemas per interrogare il database e avere come risposta i dati.

3.2 Project Dependencies

Sono stati utilizzati e aggiunti al file **package.json** i seguenti moduli Node:

- bcrypt (versione 5.1.0)
- body-parser (versione 1.20.2)
- cors (versione 2.8.5)
- dotenv (versione 16.1.4)
- express (versione 4.18.2)
- jsonwebtoken (versione 9.0.0)
- mongoose (versione 7.2.4)
- nodemon (versione 2.0.22)

I moduli Node aggiuntivi utilizzati durante lo sviluppo sono:

- mocha (versione 10.2.0)
- supertest (versione 6.3.3)

3.3 Project Data

Per la gestione dati dell'applicazione viene usato il database MongoDB. Sono state definite diverse strutture principali, tra cui: Utente, Commento, Libro, Recensione.

Descrizione dello schemas Utente

Utente.js > ...
You, 17 hours ago | 1 author (You)

```
const mongoose = require("mongoose");

// Schema dell'Utente
const utenteSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
    unique: true,
  },
  password: {
    type: String,
    required: true,
  },
  libriPreferiti: [
    {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Libro",
    },
  ],
  follow: [
    {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Utente",
    },
  ],
});

// Schema dell'Utente Amministratore
const amministratoreSchema = new mongoose.Schema({
  ruolo: {
    type: String,
    default: "amministratore",
  },
});

// Schema dell'Utente Moderatore
const moderatoreSchema = new mongoose.Schema({
  ruolo: {
    type: String,
    default: "moderatore",
  },
});

// Modello dell'Utente
const Utente = mongoose.model("Utente", utenteSchema);

// Modello dell'Utente Amministratore
const Amministratore = Utente.discriminator(
  "Amministratore",
  amministratoreSchema
);

// Modello dell'Utente Moderatore
const Moderatore = Utente.discriminator("Moderatore", moderatoreSchema);

module.exports = { Utente, Amministratore, Moderatore };
```

Descrizione dello schemas Commento

```
You, 19 hours ago | 1 author (You)
const mongoose = require("mongoose");

const commentoSchema = new mongoose.Schema({
  testo: {
    type: String,
    required: true,
  },
  autore: {      You, 19 hours ago • FIrst commit …
    type: mongoose.Schema.Types.ObjectId,
    ref: "Utente",
    required: true,
  },
  recensione: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "Recensione",
    required: true,
  },
  dataCreazione: {
    type: Date,
    default: Date.now,
  },
});

const Commento = mongoose.model("Commento", commentoSchema);

module.exports = Commento;
```

Descrizione dello schemas Libro

```
const mongoose = require("mongoose");

const libroSchema = new mongoose.Schema({
  titolo: {
    type: String,
    required: true,
  },
  autore: {
    type: String,
    required: true,
  },
  annoPubblicazione: {
    type: Number,
    required: true,
  },
  generi: [
    {
      type: String,
    },
  ],
  recensioni: [
    {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Recensione",
    },
  ],
});

const Libro = mongoose.model("Libro", libroSchema);

module.exports = Libro;
```

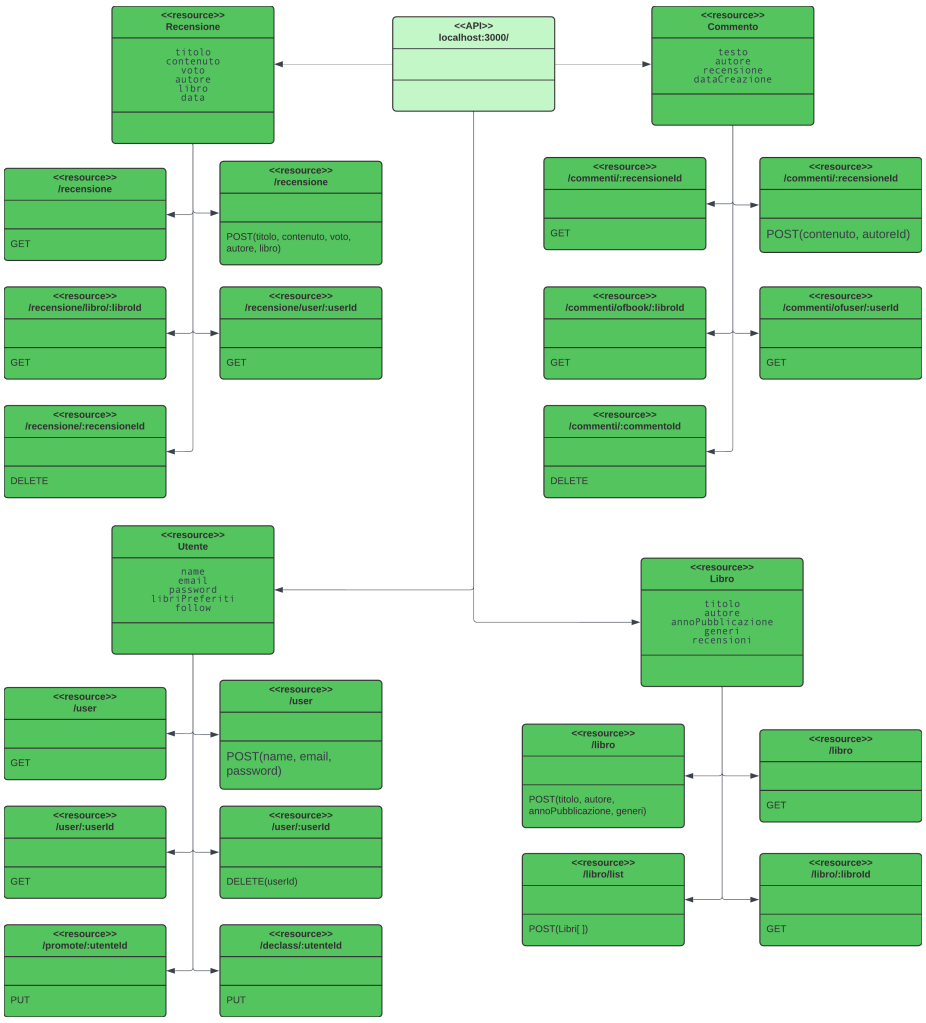
Descrizione dello schemas Recensione

```
const mongoose = require("mongoose");

const libroSchema = new mongoose.Schema({
  titolo: {
    type: String,
    required: true,
  },
  autore: {
    type: String,
    required: true,
  },
  annoPubblicazione: {
    type: Number,
    required: true,
  },
  generi: [
    {
      type: String,
    },
  ],
  recensioni: [
    {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Recensione",
    },
  ],
});

const Libro = mongoose.model("Libro", libroSchema);

module.exports = Libro;
```

3.4 Project APIs

3.4.1 Resources Extraction from the Class Diagram

Dal Class Diagram sono state estratte alcune risorse, rappresentate nella figura seguente.



Oltre alle classi nello schema sono presenti API per Follow, Preferiti, Feed, Auth. Verranno spiegati nel dettaglio in seguito.

Per **Utente** sono presenti dal lato back-end le risorse:

- GET /user: Ottiene tutti gli utenti
- GET /user/:userId: Ottiene un utente dato il suo ID
- POST /user: Crea un nuovo utente
- DELETE /user/:userId: Cancella un utente dato il suo ID
- PUT /user/promote/:utenteId/moderatore: Promuove un utente a moderatore
- PUT /user/declass/:utenteId/revocaModeratore: Revoca i privilegi di moderatore a un utente

Per **Commento** sono presenti dal lato back-end le risorse:

- GET /commenti/:recensioneId: Ottiene tutti i commenti di una recensione dato l'ID della recensione
- POST /commenti/:recensioneId: Pubblica un nuovo commento per una recensione dato l'ID della recensione
- GET /commenti/libro/:libroId: Ottiene tutti i commenti relativi a un libro dato l'ID del libro
- GET /commenti/user/:autoreId: Ottiene tutti i commenti di un autore dato l'ID dell'autore
- DELETE /commenti/:commentoId: Cancella un commento dato l'ID del commento (richiede autenticazione e privilegi di moderatore)

Per **Libro** sono presenti dal lato back-end le risorse:

- POST /libri: Aggiunge un nuovo libro
- POST /libri/list: Aggiunge un array di libri
- GET /libri: Ottiene tutti i libri
- GET /libri/:libroId: Ottiene un libro specifico dato l'ID del libro

Per **Recensione** sono presenti dal lato back-end le risorse:

- POST /recensioni: Pubblica una nuova recensione. Richiede un token di autenticazione (`verifyToken`) e accetta i parametri `titolo`, `contenuto`, `voto`, `autore` e `libro`. Restituisce la nuova recensione creata.
- GET /recensioni: Ottiene tutte le recensioni presenti nel database.
- GET /recensioni/user/:utenteId: Ottiene tutte le recensioni di un utente specificato dall'ID `:utenteId`.

- GET /recensioni/libro/:libroId: Ottiene tutte le recensioni di un libro specificato dall'ID `:libroId`.
- DELETE /recensioni/:recensioneId: Cancella una recensione specificata dall'ID `:recensioneId`.

Per **Auth** sono presenti dal lato back-end le risorse:

- POST /auth/login: Gestisce la richiesta di login dell'utente. Verifica se l'utente esiste nel database, verifica la password fornita e genera un token di autenticazione.
- POST /auth/logout: Gestisce la richiesta di logout dell'utente. Verifica il token di autenticazione nella richiesta e lo invalida nel server.

Per **Follow** sono presenti dal lato back-end le risorse:

- POST /follow/:userId: Aggiunge un utente alla lista degli utenti seguiti.
- DELETE /follow/:userId: Rimuove un utente dalla lista degli utenti seguiti.

Per **Preferiti** sono presenti dal lato back-end le risorse:

- POST /preferiti per aggiungere un libro ai preferiti dell'utente.
- DELETE /preferiti per rimuovere un libro dalla lista dei preferiti dell'utente.
- GET /preferiti per ottenere tutti i preferiti di un utente.

Per **Feed** sono presenti dal lato back-end le risorse:

- GET /feed/libri/:utenteId per ottenere il feed dei libri in base ai generi preferiti dell'utente.
- GET /feed/recensioni/:utenteId per ottenere il feed delle recensioni, considerando gli interessi dell'utente autenticato o mostrando un elenco generale per gli utenti non autenticati.

3.4.2 Resources Models

Identificate le risorse è stato costruito un diagramma delle API per identificare i request e response body e le varie tipologie di risposta che è possibile aspettarsi dalle API.

Diagramma per Utente

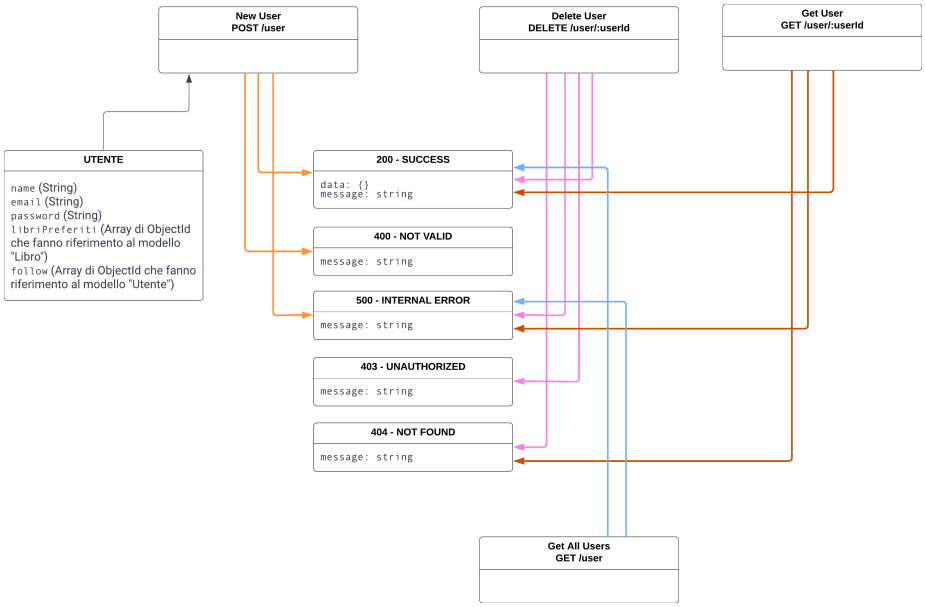


Diagramma per Recensione

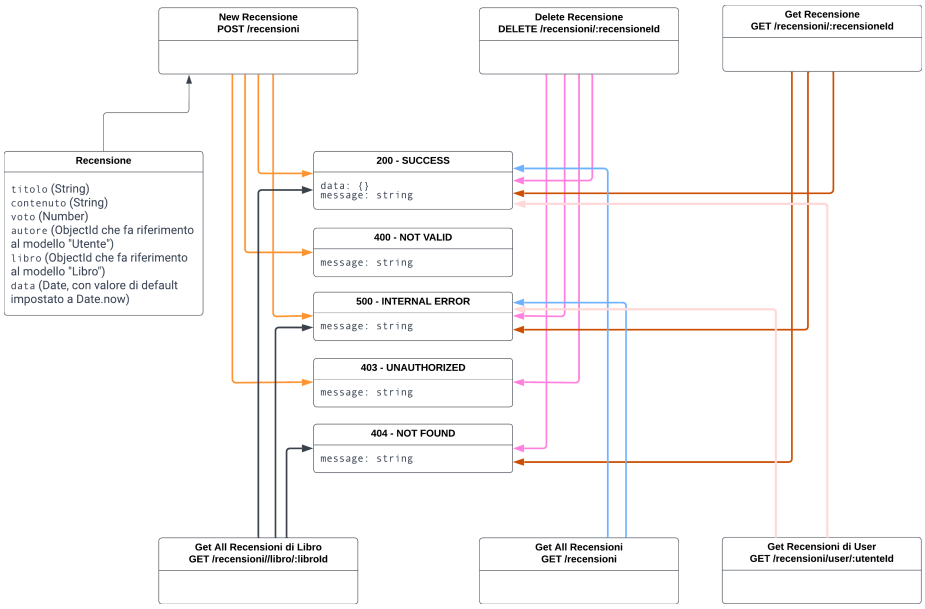


Diagramma per Commento

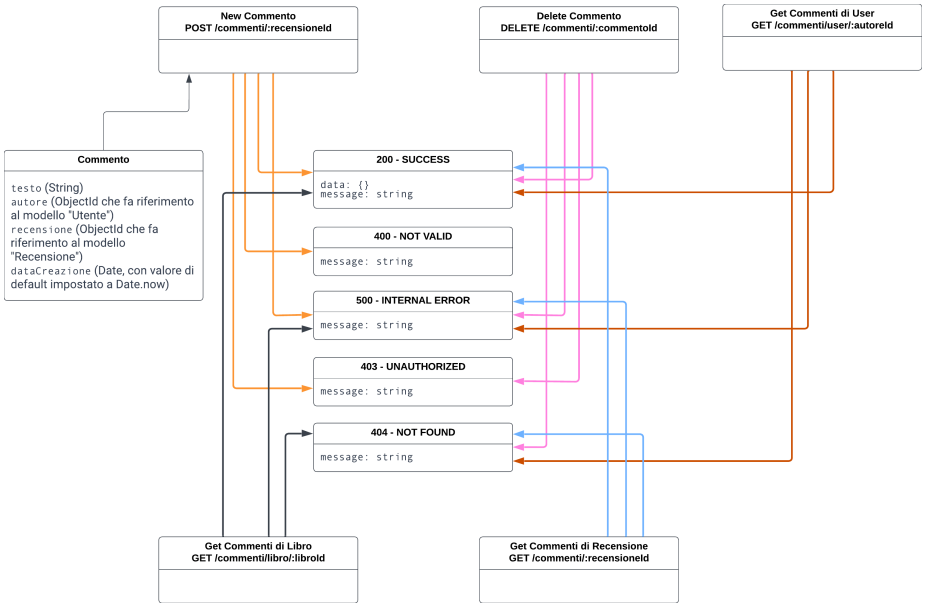
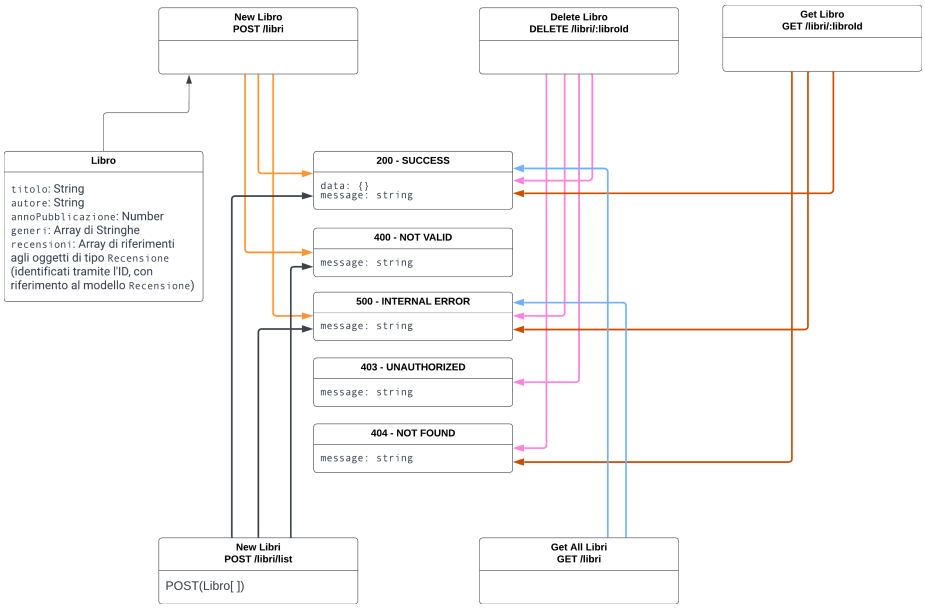


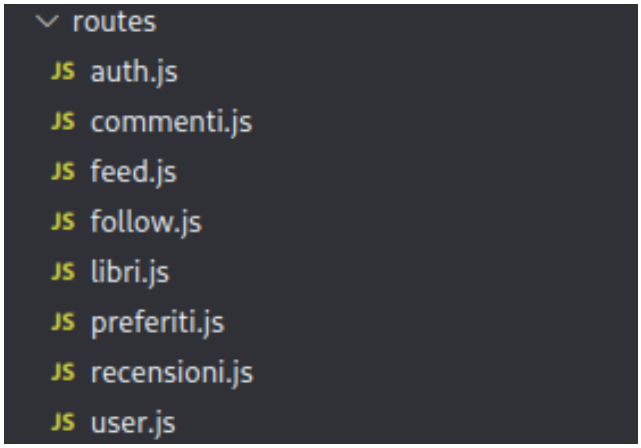
Diagramma per Libro



3.5 Aviluppo APIs

Le API vengono gestite dai router presenti nella cartella routes dell'app.

```
const express = require("express");
const app = express();
const mongoose = require("mongoose");
const bodyParser = require("body-parser");
const fs = require("fs");
const cors = require("cors");
require("dotenv/config");

app.use(bodyParser.json());

//routes
//import all routes
fs.readdirSync("./routes").forEach((file) => {
  if (file.endsWith(".js")) {
    const route = require(`./routes/${file}`);
    const routeName = file.split(".")[0];
    app.use(`/${routeName}`, route);
  }
});
```



3.5.1 Metodi della route /user

GET "/": Questo metodo gestisce la richiesta per ottenere tutti gli utenti nel database. Utilizzando il modello Utente, esegue una query per trovare tutti gli utenti e restituisce un array di utenti come risposta.

GET "/:userId": Questo metodo gestisce la richiesta per ottenere un utente specifico in base all'ID dell'utente. Utilizzando il modello Utente e l'ID fornito nella richiesta, esegue una query per trovare l'utente corrispondente. Se l'utente non viene trovato, viene restituito uno status code 404 con un messaggio di errore. Altrimenti, viene restituito l'utente come risposta.

POST "/": Questo metodo gestisce la richiesta per creare un nuovo utente. Estrae i dati necessari (nome, email, password) dalla richiesta e verifica se l'utente esiste già nel database in base all'indirizzo email fornito. Se l'utente esiste già, viene restituito uno status code 400 con un messaggio di errore. Altrimenti, viene creato un nuovo oggetto Utente utilizzando i dati forniti, la password viene criptata utilizzando bcrypt e viene salvato nel database. Viene restituito uno status code 200 con un messaggio di successo.

DELETE "/:userId": Questo metodo gestisce la richiesta per cancellare un utente specifico in base all'ID dell'utente. Prima di procedere alla cancellazione, viene verificato se l'utente che sta effettuando la richiesta è autorizzato a cancellare l'utente specificato. Se l'utente non è autorizzato, viene restituito uno status code 403 con un messaggio di errore. Se l'utente è autorizzato, viene eseguita una query per trovare e rimuovere l'utente dal database. Se l'utente non viene trovato, viene restituito uno status code 404 con un messaggio di errore. Altrimenti, viene restituito uno status code 200 con un messaggio di successo.

PUT "/promote/:utenteId": Questo metodo gestisce la richiesta per promuovere un utente a moderatore. Prima di procedere alla promozione, viene verificato se l'utente che effettua la richiesta è un amministratore. Se l'utente non è un amministratore, viene restituito uno status code 403 con un messaggio di errore. Se l'utente è un amministratore, viene eseguita una query per trovare l'utente specificato dall'ID fornito nella richiesta. Se l'utente non viene trovato, viene restituito uno status code 404 con un messaggio di errore. Altrimenti, il ruolo dell'utente viene aggiornato a "moderatore" e viene restituito uno status code 200 con un messaggio di successo insieme all'utente aggiornato.

PUT "/declass/:utenteId": Questo metodo gestisce la richiesta per revocare i privilegi di moderatore a un utente. Come nel metodo precedente, viene verificato se l'utente che effettua la richiesta è un amministratore. Se l'utente non è un amministratore, viene restituito uno status code 403 con un messaggio di errore. Se l'utente è un amministratore, viene eseguita una query per trovare l'utente specificato dall'ID fornito nella richiesta.

Se l'utente non viene trovato, viene restituito uno status code 404 con un messaggio di errore. Altrimenti, viene verificato se l'utente è un moderatore. Se l'utente non è un moderatore, viene restituito uno status code 400 con un messaggio di errore. Se l'utente è un moderatore, il ruolo dell'utente viene aggiornato a "utente" e viene restituito uno status code 200 con un messaggio di successo insieme all'utente aggiornato.

3.5.2 Metodi della route /recensioni

POST "/"

Pubblica una recensione. Verifica il token di autenticazione dell'utente. I dati della recensione (titolo, contenuto, voto, autore, libro) vengono estratti dalla richiesta. Viene creato un nuovo oggetto Recensione con i dati forniti e viene salvato nel database. Restituisce lo status code 201 con la nuova recensione creata come risposta.

GET "/"

Ottiene tutte le recensioni presenti nel database. Esegue una query per trovare tutte le recensioni e le restituisce come risposta.

GET "/user/:utenteId"

Ottiene tutte le recensioni effettuate da un utente specifico. L'ID dell'utente viene estratto dai parametri della richiesta. Esegue una query per trovare tutte le recensioni con l'autore corrispondente all'ID fornito e le restituisce come risposta.

GET "/libro/:libroId"

Ottiene tutte le recensioni relative a un libro specifico. L'ID del libro viene estratto dai parametri della richiesta. Esegue una query per trovare tutte le recensioni con il libro corrispondente all'ID fornito e le restituisce come risposta.

DELETE "/:recensioneId"

Cancella una recensione specifica in base all'ID della recensione fornito. Verifica il token di autenticazione dell'utente e il ruolo dell'utente (deve essere un moderatore). Esegue una query per trovare la recensione corrispondente all'ID fornito. Se la recensione non viene trovata, restituisce lo status code 404 con un messaggio di errore. Altrimenti, rimuove la recensione dal database e restituisce uno status code 200 con un messaggio di successo.

3.5.3 Metodi della route /preferiti

POST "/"

Aggiunge un libro alla lista dei preferiti dell'utente. I dati dell'utente (userId) e del libro (libroId) vengono estratti dalla richiesta. Esegue una query per trovare l'utente corrispondente all'ID fornito. Se l'utente non viene trovato, restituisce lo status code 404 con un messaggio di errore. Esegue una query per trovare il libro corrispondente all'ID fornito. Se il libro non viene trovato, restituisce lo status code 404 con un messaggio di errore. Verifica se il libro è già presente nella lista dei preferiti dell'utente. Se è presente, restituisce lo status code 400 con un messaggio di errore. Altrimenti, aggiunge il libro alla lista dei preferiti dell'utente e salva le modifiche all'utente nel database. Restituisce uno status code 200 con un messaggio di successo.

DELETE "/"

Rimuove un libro dalla lista dei preferiti dell'utente. I dati dell'utente (userId) e del libro (libroId) vengono estratti dai parametri della richiesta. Esegue una query per trovare l'utente corrispondente all'ID fornito. Se l'utente non viene trovato, restituisce lo status code 404 con un messaggio di errore. Verifica se il libro è presente nella lista dei preferiti dell'utente. Se non è presente, restituisce lo status code 400 con un messaggio di errore. Altrimenti, rimuove il libro dalla lista dei preferiti dell'utente e salva le modifiche all'utente nel database. Restituisce uno status code 200 con un messaggio di successo.

3.5.4 Metodi della route /libri

POST "/"

Aggiunge un nuovo libro al database. I dati del libro (titolo, autore, annoPubblicazione, generi) vengono estratti dalla richiesta. Crea un nuovo oggetto Libro con i dati forniti e lo salva nel database. Restituisce lo status code 201 con il nuovo libro creato come risposta.

POST "/list"

Aggiunge un array di libri al database. L'array di libri viene estratto dalla richiesta. Verifica che l'array di libri sia presente nella richiesta e che sia un array valido. Crea un nuovo oggetto Libro per ogni libro nell'array e li inserisce nel database utilizzando il metodo insertMany. Restituisce lo status code 201 con l'array dei libri inseriti come risposta.

GET "/"

Ottiene tutti i libri presenti nel database. Esegue una query per trovare tutti i documenti nella collezione "Libro" e restituisce i risultati come risposta.

GET "/:libroId"

Ottiene un libro specifico dal database in base all'ID fornito. L'ID del

libro viene estratto dai parametri della richiesta. Esegue una query per trovare il libro corrispondente all'ID fornito. Se il libro non viene trovato, restituisce lo status code 404 con un messaggio di errore. Altrimenti, restituisce il libro come risposta.

DELETE "/:libroId"

Cancella un libro dal database in base all'ID fornito. L'ID del libro viene estratto dai parametri della richiesta. Verifica se il libro esiste nel database. Se il libro non viene trovato, restituisce lo status code 404 con un messaggio di errore. Altrimenti, cancella il libro utilizzando il metodo findByIdAndDelete e cancella tutte le recensioni associate al libro utilizzando il metodo deleteMany della collezione "Recensione". Restituisce uno status code 200 con un messaggio di successo.

3.5.5 Metodi della route /follow

POST "/:userId"

Aggiunge un utente alla lista degli utenti seguiti. L'ID dell'utente corrente viene estratto dai parametri della richiesta e l'ID dell'utente da seguire viene estratto dal corpo della richiesta. Verifica se l'utente corrente esiste nel database. Se l'utente non viene trovato, restituisce lo status code 404 con un messaggio di errore. Verifica se l'utente da seguire è già presente nella lista degli utenti seguiti dell'utente corrente. Se l'utente è già presente, restituisce lo status code 400 con un messaggio di errore. Aggiunge l'ID dell'utente da seguire alla lista degli utenti seguiti dell'utente corrente e salva le modifiche. Restituisce uno status code 200 con un messaggio di successo.

GET "/:userId/preferiti"

Ottiene tutti i libri preferiti di un utente. L'ID dell'utente corrente viene estratto dai parametri della richiesta. Esegue una query per trovare l'utente corrispondente all'ID fornito e popola il campo "libriPreferiti". Se l'utente non viene trovato, restituisce lo status code 404 con un messaggio di errore. Restituisce i libri preferiti dell'utente come risposta.

DELETE "/:userId"

Rimuove un utente dalla lista degli utenti seguiti. L'ID dell'utente corrente viene estratto dai parametri della richiesta e l'ID dell'utente da rimuovere viene estratto dal corpo della richiesta. Verifica se l'utente corrente esiste nel database. Se l'utente non viene trovato, restituisce lo status code 404 con un messaggio di errore. Verifica se l'utente da rimuovere è presente nella lista degli utenti seguiti dell'utente corrente. Se l'utente non è presente, restituisce lo status code 400 con un messaggio di errore. Rimuove l'ID dell'utente da rimuovere dalla lista degli utenti seguiti dell'utente corrente e salva le modifiche. Restituisce uno status code 200 con un messaggio di successo.

3.5.6 Metodi della route /feed

GET "/libri/:utenteId"

Ottiene i libri consigliati per un utente. L'ID dell'utente viene estratto dai parametri della richiesta. Esegue una query per trovare l'utente corrispondente all'ID fornito. Se l'utente non viene trovato, restituisce lo status code 404 con un messaggio di errore. Ottiene i generi dei libri preferiti dell'utente. Calcola il numero minimo di generi che devono corrispondere per il feed. Esegue una query per trovare i libri nel feed che corrispondono ai generi preferiti dell'utente, limitando i risultati a 10 libri. Filtra ulteriormente i libri per garantire che abbiano almeno il 50% di generi in comune con i generi preferiti dell'utente. Restituisce i libri filtrati come risposta.

GET "/recensioni/:utenteId"

Ottiene le recensioni per un utente. L'ID dell'utente viene estratto dai parametri della richiesta. Determina se l'utente è autenticato o meno. Se l'utente è autenticato, esegue una query per trovare l'utente corrispondente all'ID fornito. Se l'utente non viene trovato, restituisce lo status code 404 con un messaggio di errore. Ottiene i generi dei libri preferiti dell'utente. Esegue una query per trovare le recensioni che hanno almeno il 50% dei generi compatibili con i generi preferiti dell'utente e che sono state scritte dagli autori seguiti dall'utente. Ordina le recensioni per data in ordine decrescente. Se l'utente non è autenticato, esegue una query per ottenere le ultime 10 recensioni ordinate per data in ordine decrescente. Restituisce le recensioni come risposta.

3.5.7 Metodi della route /commenti

GET "/:recensioneId"

Ottiene i commenti di una recensione. L'ID della recensione viene estratto dai parametri della richiesta. Esegue una query per trovare i commenti che sono associati alla recensione specificata. Restituisce i commenti come risposta.

POST "/:recensioneId"

Pubblica un commento. L'ID della recensione viene estratto dai parametri della richiesta. Il token di autenticazione dell'utente viene verificato utilizzando il middleware "verifyToken". L'ID dell'autore e il contenuto del commento vengono estratti dal corpo della richiesta. Verifica se la recensione esiste. Se la recensione non viene trovata, restituisce lo status code 404 con un messaggio di errore. Crea un nuovo commento utilizzando le informazioni fornite. Salva il commento nel database. Restituisce lo status code 200 con un messaggio di successo e il commento pubblicato come risposta.

GET "libro/:libroId"

Ottiene i commenti relativi a un libro. L'ID del libro viene estratto dai

parametri della richiesta. Esegue una query per trovare le recensioni che sono relative al libro specificato. Estrae gli ID delle recensioni. Esegue una query per trovare i commenti che sono relativi alle recensioni trovate. Restituisce i commenti come risposta.

GET "user/:autoreId"
Ottiene i commenti di un autore. L'ID dell'autore viene estratto dai parametri della richiesta. Esegue una query per trovare l'utente autore corrispondente all'ID fornito. Se l'utente autore non viene trovato, restituisce lo status code 404 con un messaggio di errore. Esegue una query per trovare le recensioni che sono scritte dall'utente autore. Estrae gli ID delle recensioni. Esegue una query per trovare i commenti che sono relativi alle recensioni dell'utente autore trovate. Restituisce i commenti come risposta.

DELETE "/:commentoId"
Cancella un commento. L'ID del commento viene estratto dai parametri della richiesta. Il token di autenticazione dell'utente viene verificato utilizzando il middleware "verifyToken". Viene verificato che l'utente abbia il ruolo di "moderatore" utilizzando il middleware "verifyRole". Esegue una query per trovare il commento da cancellare. Se il commento non viene trovato, restituisce lo status code 404 con un messaggio di errore. Cancella il commento dal database. Restituisce un messaggio di successo come risposta.

3.5.8 Metodi della route /auth

POST "/login"
Effettua il login dell'utente. L'endpoint richiede una richiesta POST con i campi "email" e "password" nel corpo della richiesta. Esegue una query per verificare se l'utente esiste nel database utilizzando l'email fornita. Se l'utente non viene trovato, restituisce lo status code 404 con un messaggio di errore. Verifica se la password fornita corrisponde alla password hash memorizzata nel database utilizzando la funzione bcrypt.compare. Se la password non è valida, restituisce lo status code 401 con un messaggio di errore. Genera un token di autenticazione utilizzando la funzione generateToken del middleware "auth". Restituisce il token come risposta.

POST "/logout"
Effettua il logout dell'utente. L'endpoint richiede una richiesta POST con un token di autenticazione valido nell'intestazione "Authorization" della richiesta. Verifica il token utilizzando il middleware "verifyToken". Ottiene il token dalla richiesta e invalida il token nel server. Restituisce uno status code 200 con un messaggio di successo come risposta.

4 Front-End Implementation

Il Front-End fornisce una visualizzazione grafica rispetto alle funzioni di:

- Creazione dell'Account
- Login
- Feed nella homepage
- Creazione di una nuova recensione
- Creazione di un nuovo commento
- Visualizzazione informazioni libro
- Follow di un utente
- Aggiunta Libro ai preferiti