Libibi

Antony Iembo,
Federico Filice

1.0 Panoramica

- “Libibi”, è nato con **l’idea** di realizzare un’interfaccia più semplice e intuitiva rispetto alla concorrenza.
- Lo **scopo** è dare la possibilità all’utente di tenere traccia e aggiungere recensioni di libri.
- Le **funzionalità** implementate permettono all’utente di inserire in liste separate, ottenere consigli sui prossimi libri da leggere e inserire recensioni.
- **Sistema ibrido** formato da un database locale e un API pubblica.

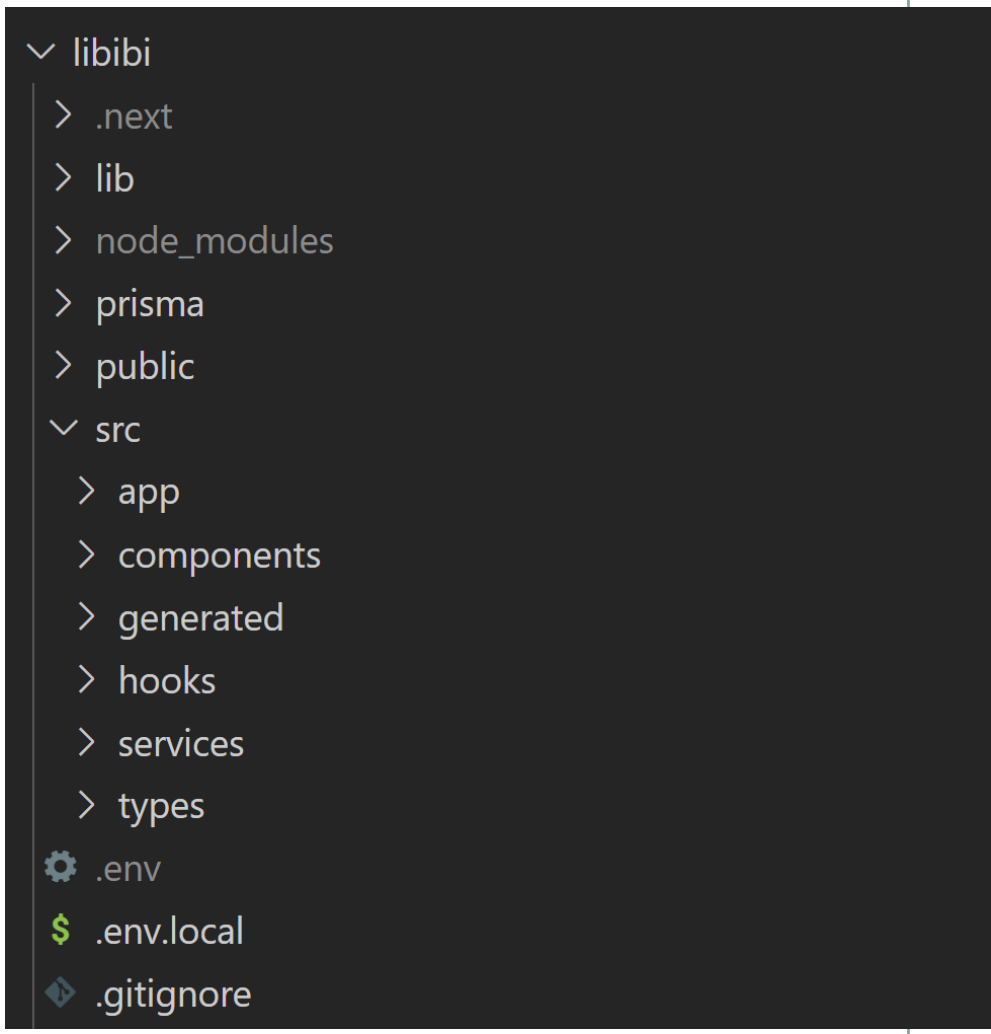
2.0 Specifiche tecniche

2.1 Teconologie usate

- **Framework:** NextJS
- **Database:** SQL Server
- **ORM:** Prisma
- **Linguaggi:** TypeScript, CSS, HTML
- **Applicazioni e Editor:** VS code, SSMS, GitHub
- **API esterne:** OpenLibrary, OpenRouter

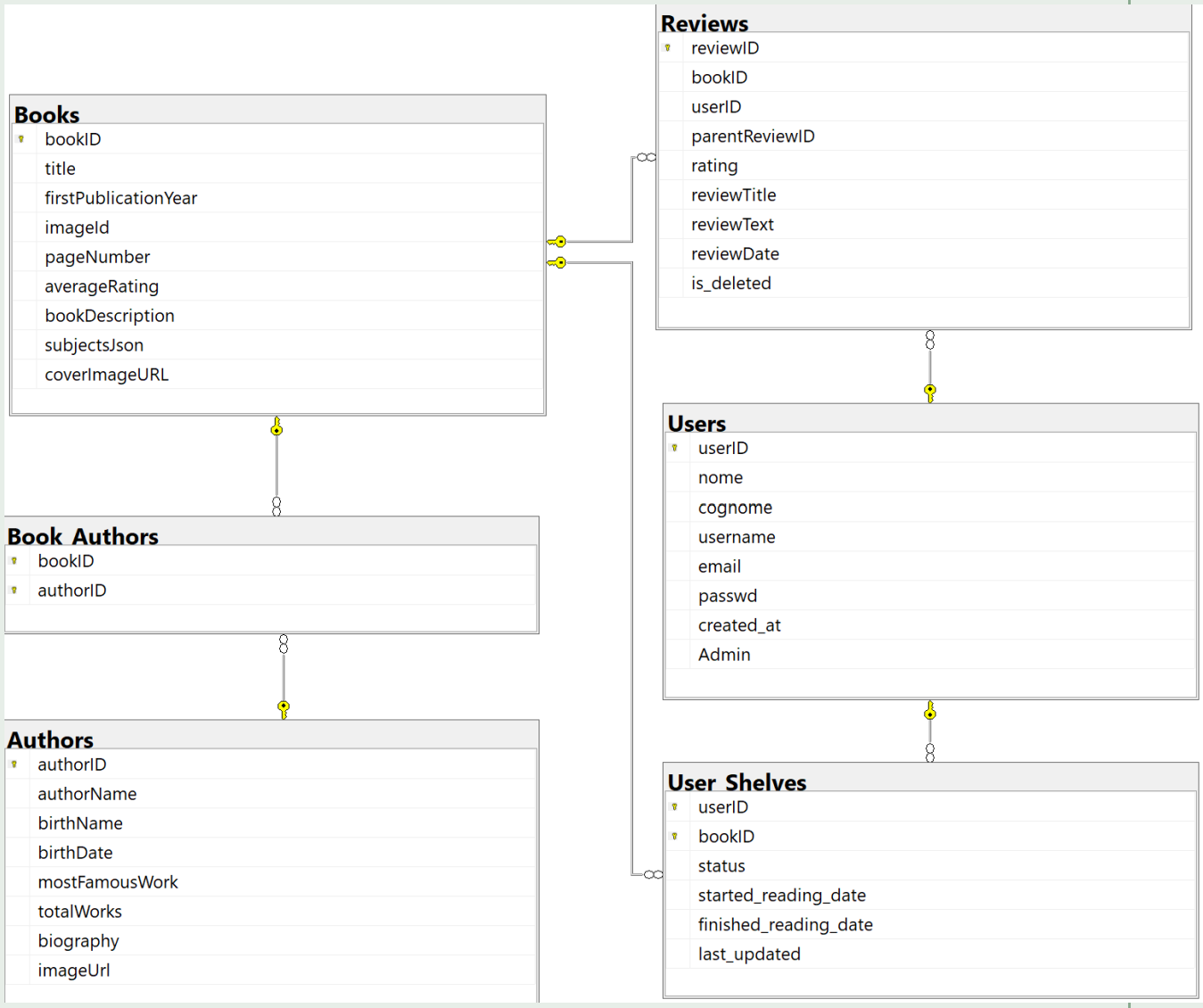
2.2 Struttura del progetto

- **lib:** script di connessione al database.
- **src:** contiene il codice sorgente della web app.
  - **app:** contiene la parte client.
  - **components:** contiene la parte server del sito.
  - **services:** servizi più complessi, autenticazione, script di ricerca e consigliati con API.
  - **types:** centralizza tutti i tipi di TypeScript, necessari a definire forma e struttura dei dati, ma anche per prevenire errori.
- **.env e .env.local:** file di configurazione. stringa di configurazione del database, chiave di accesso LLM.



2.3 Database

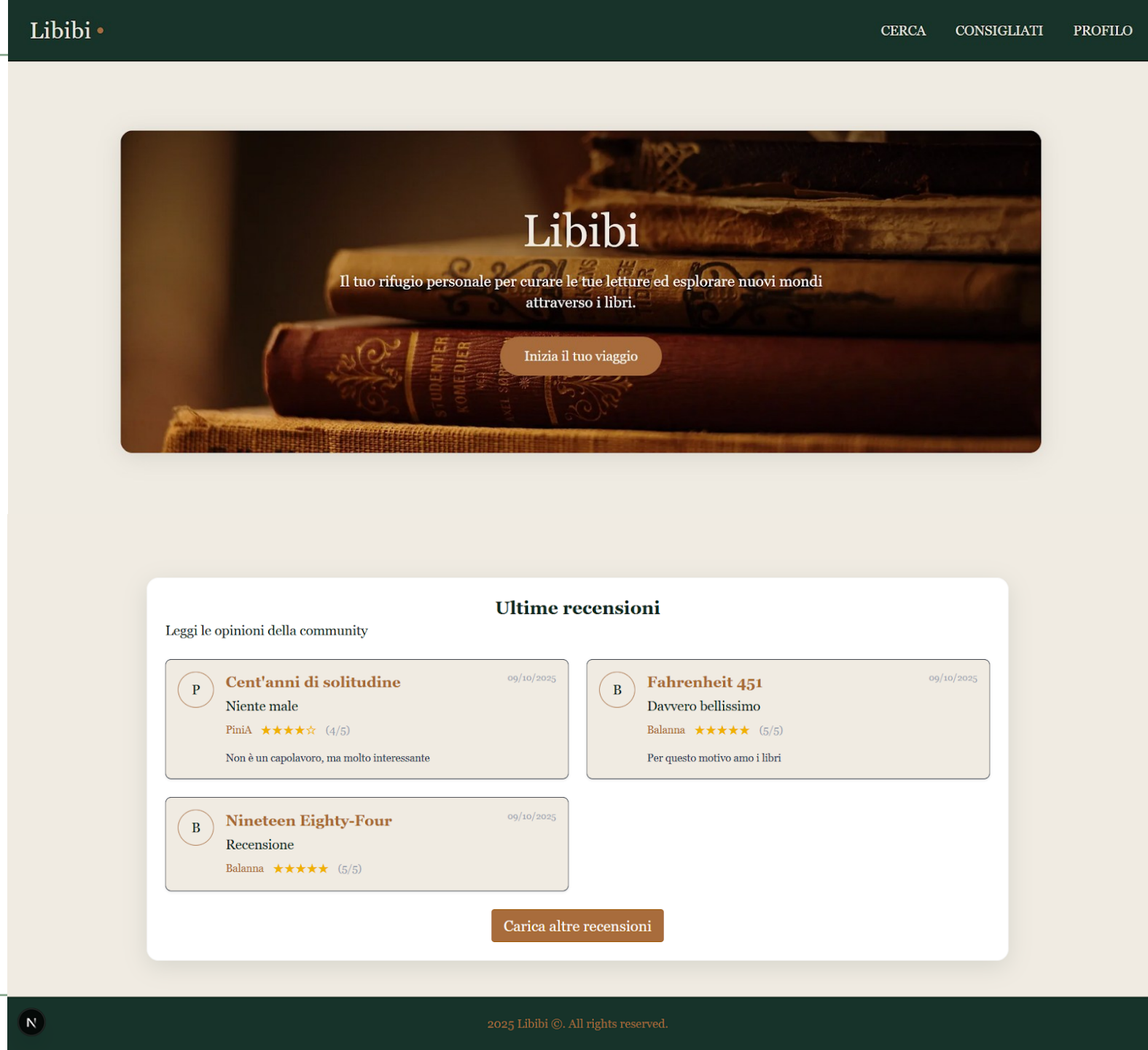
Il seguente grafico è la rappresentazione schematica del database.



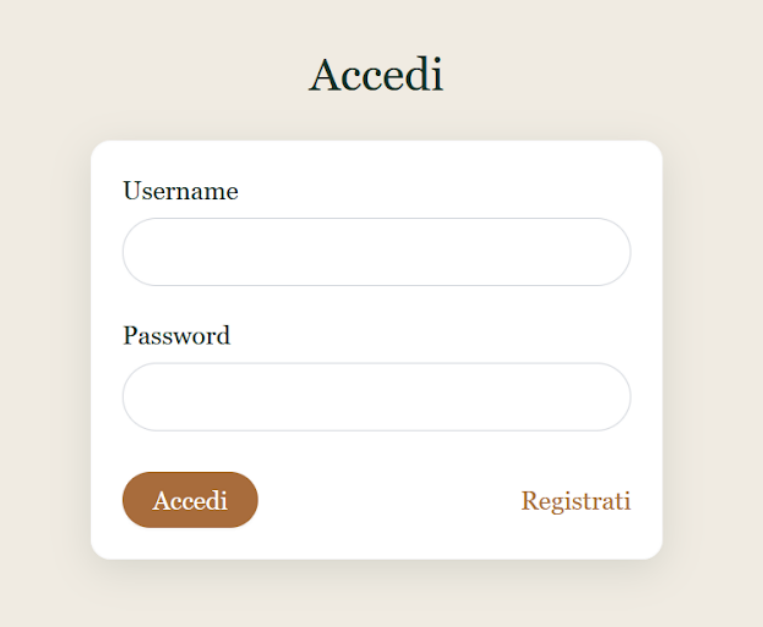
3.0 Struttura del sito

3.1 Homepage

- La **barra di navigazione** contiene il logo “Libibi”, e le tre pagine principali: Cerca, Consigliati, Profilo.
- Scorrendo verso il basso, possiamo trovare una sezione **“Ultime recensioni”**.
- è possibile leggere altre recensioni tramite il pulsante **“Carica altre recensioni”**.
- Con il pulsante **“Inizia il tuo viaggio”** possiamo accedere.



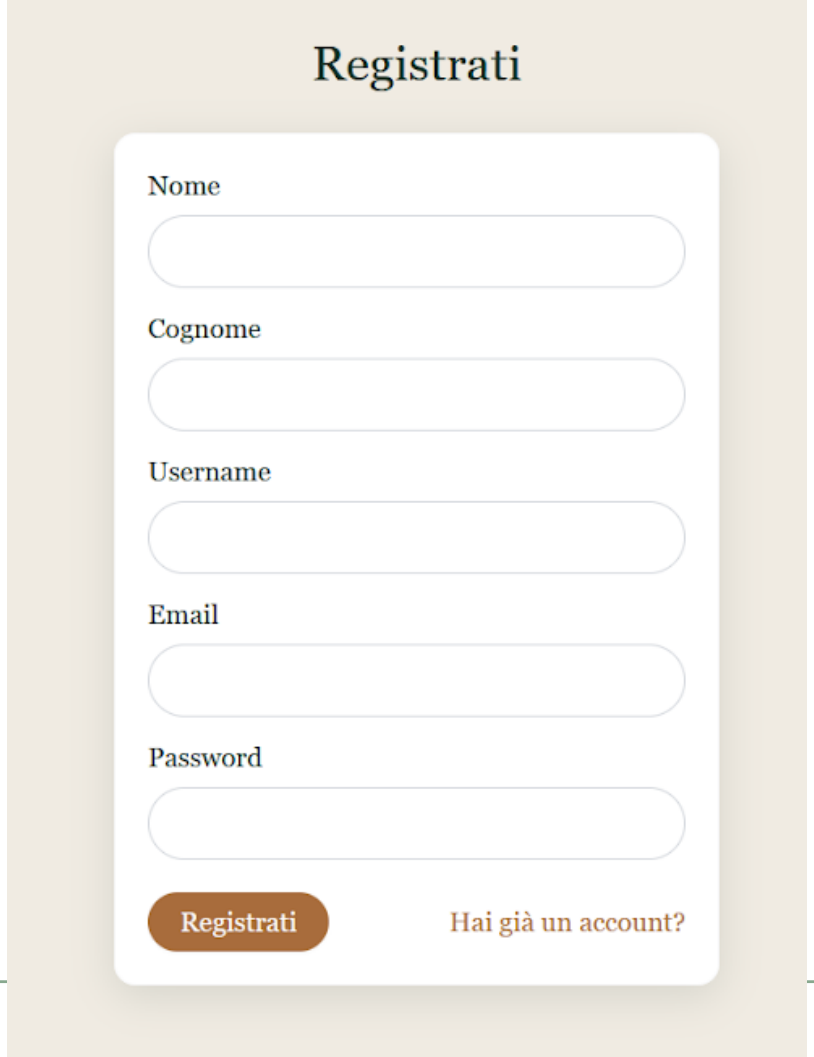
3.2 Log-in e Sign-in



- Nella pagina di **accesso** l'utente ha la possibilità di accedere oppure registrarsi.

- è possibile arrivare alla pagina di accesso quando si prova ad usare le funzionalità che richiedono l'accesso, ad esempio alla pagina "Consigliati" o "Profilo".

- L'utente si può **registrare** inserendo i suoi dati. Fatto ciò, l'utente viene reindirizzato alla pagina di accesso.



5.1 Autenticazione

L’autenticazione è così divisa:

- Abbiamo **users/PostUserData**, una POST che permette all’utente di potersi registrare inserendo le proprie generalità, qualora non lo avesse già fatto.

TypeScript

```
export async function POST(req: NextRequest) {
  try {
    const user = await req.json();
    if (!user) return NextResponse.json(
    { Errore: 'Oggetto utente nullo' }, { status: 400 });

    const pool = await connectToDatabase();

    // Controlla se Username o Email esistono già
    const checkRequest = pool.request();
    checkRequest.input('Username', sql.NVarChar(100), user.Username||'');
    checkRequest.input('Email', sql.NVarChar(255), user.Email || '');
    const existsResult = await checkRequest.query(
        "SELECT Username, Email FROM Users WHERE Username = "
        + "@Username OR Email = @Email");

// [...]
    // Inserimento nuovo utente
    const request = pool.request();
    request.input('Nome', sql.NVarChar(50), user.Nome || '');
// [...]
    request.input('Passwd', sql.NVarChar(255), user.Passwd || '');

    const insertQuery = `INSERT INTO Users (Nome, Cognome, Username,
Email, Passwd) VALUES (@Nome, @Cognome, @Username, @Email, @Passwd)`;
    await request.query(insertQuery);
// [...]
    // restituisce i dati inseriti
    return NextResponse.json(ret);
}
```

5.1 Autenticazione

L’autenticazione è così divisa:

- **/users/login**, valida le credenziali dell’utente leggendone username e password, impostando cookie con authenticated se esatte (creato da useAuth, il quale permette al cookie di rimanere attivo per 7 giorni).
- Se non validi mostra un errore di tipo 401.
- Se l’utente non esiste mostra un 404.
- Restituisce i dati utente da userService.ts.

TypeScript

```
export async function POST(req: NextRequest) {
  try {
    const body = await req.json();
    const username = (body.username ?? '') as string;
    const password = (body.password ?? '') as string;
    if (!username || !password) {
      return NextResponse.json({ Errore: 'Missing credentials' }, {
status: 400 });
    }

    // Valida le credenziali
    const valid = await validateUserAsync(username, password);
    if (!valid) {
      return NextResponse.json({ Errore: 'Unauthorized' }, { status: 401
});
    }
// [...]
    // Recupera i dati utente
    const user = await getByUsernameAsync(username);
    if (!user) return NextResponse.json({ Errore: 'User not found' }, {
status: 404 });

    // Restituisce i dati utente (senza password)
    const resp = {
      Id: user.userID,
      Nome: user.nome,
      Cognome: user.cognome,
      Username: user.username,
      Email: user.email,
      Admin: user.admin,
    };

    return NextResponse.json(resp);
}
```

5.1 Autenticazione

L’autenticazione è così divisa:

- **basicAuth**, middleware dell’autenticazione usato per proteggere le API, che permette la decodifica in base64 dell’username:password, mostrando i ruoli dell’utente se validi.

- Per l’autenticazione non abbiamo usato token JWT, ma un ibrido tra **cookie** (in localstorage) e **Basic Authentication**; con un sistema di auto-refresh che controlla che isLogged sia True ogni 5 secondi.

- Se l’utente non ha effettuato l’accesso, non potrà visualizzare alcune pagine specifiche e verrà reindirizzato alla pagina di Login.

TypeScript

```typescript
// l’header `Authorization`, controlla che sia di tipo “Basic” e
// decodifica la stringa Base64 per ottenere username e password.
export async function parseBasicAuthHeader(req: NextRequest) {
  const header = req.headers.get('authorization');
  if (!header || !header.toLowerCase().startsWith('basic ')) return null;

  const token = header.substring(6); // rimuove "Basic "
  const decoded = Buffer.from(token, 'base64').toString('utf8');
  const parts = decoded.split(':');
  if (parts.length !== 2) return null;

  return { username: parts[0], password: parts[1] };
}


// usa le credenziali decodificate per verificare l’utente nel database,
// controllare i ruoli associati e restituire i suoi dati
// se tutto è valido.
export async function requireBasicAuth(req: NextRequest) {
  const creds = await parseBasicAuthHeader(req);
  if (!creds) return null;

  const valid = await validateUserAsync(creds.username, creds.password);
  if (!valid) return null;

  const roles = await userRoles(creds.username);
  const user = await getByUsernameAsync(creds.username);

  return { username: creds.username, roles, user };
}
```

3.3 Ricerca

- Qualsiasi utente può cercare e visualizzare i libri presenti nel database, oppure avviene una ricerca automatica tramite l'API di OpenLibrary.
- Entrando su un titolo specifico, il libro e l'autore vengono registrati automaticamente nel database.

3.3 Ricerca

La ricerca **generica** per ottenere diversi risultati non salva le informazioni in database.

La ricerca **del libro specifico** ottiene tutti i dati del libro e dell'autore, prima cercando entrambi nel database, poi se non sono presenti, effettuando una richiesta all'API.

La ricerca **dell'autore specifico** ottiene tutti i dati dell'autore specifico.

3.3 Ricerca

TypeScript

```typescript
export async function GET(request: NextRequest) {
  const bookId = searchParams.get('bookId');

  // Validazione input, richiede ID libro
  if (!bookId) {
    return NextResponse.json({ errore: 'bookId non valido' },
    { status: 400 });
  }
//[...]

  // Ricerca nel database locale tramite query
  const result = await pool.request()
    .input('bookId', sql.NVarChar, bookId)
    .query(
    "SELECT b.bookID, b.title, b.firstPublicationYear, b.pageNumber,"
    + "b.averageRating, b.bookDescription, b.subjectsJson,"
    + "b.coverImageURL, a.authorID, a.authorName"
    + "FROM Books b"
    + "LEFT JOIN Book_Authors ba ON b.bookID = ba.bookID"
    + "LEFT JOIN Authors a ON ba.authorID = a.authorID"
    + "WHERE b.bookID = @bookId"
    );

  // Se vengono trovati risultati, termina esecuizone
  if (result.recordset.length > 0) {
    const responseData = transformDbToResponse(result.recordset);
    return NextResponse.json({ success: true, result: responseData });
  }
```

```typescript
  // Se non vengono trovati risultati, cerca su Openlibrary
  const bookDetail = await openLibraryService.getBookAsync(bookId);

  // Se non vengono trovati risultati ritorna errore 404
  if (!bookDetail) {
    return NextResponse.json({ Messaggio: 'Libro non trovato' },
    { status: 404 });
  }

  // Prima di terminare salva il risultato in database
  try {
    const bookForDB = transformOpenLibraryToDb(bookDetail);

    await pool.request()
        .input('id', sql.NVarChar, bookForDB.id)
        .input('title', sql.NVarChar, bookForDB.title)
//[...]
        .input('coverUrl', sql.NVarChar, bookForDB.coverUrl || null)
        .query(
        "INSERT INTO Books (bookID, title, firstPublicationYear,"
        + "pageNumber, averageRating,"
        + "bookDescription, subjectsJson, coverImageURL)"
        + "VALUES (@id, @title, @firstYear, @pages, @rating,"
        + "@description, @subjects, @coverUrl)");
    }
  }
// [...]

  // Restituisci i dettagli del libro ottenuti da OpenLibrary
  const response = transformOpenLibraryToResponse(bookDetail);
  return NextResponse.json(response);
}
```

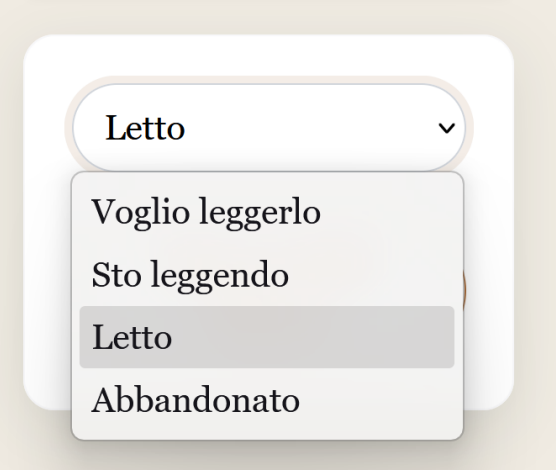
3.3 Ricerca


```typescript
export async function GET(request: NextRequest) {
  const bookId = searchParams.get('bookId');

  // Validazione input, richiede ID libro
  if (!bookId) {
    return NextResponse.json({ errore: 'bookId non valido' },
    { status: 400 });
  }
//[...]

  // Ricerca nel database locale tramite query
  const result = await pool.request()
    .input('bookId', sql.NVarChar, bookId)
    .query(
    "SELECT b.bookID, b.title, b.firstPublicationYear, b.pageNumber,"
    + "b.averageRating, b.bookDescription, b.subjectsJson,"
    + "b.coverImageURL, a.authorID, a.authorName"
    + "FROM Books b"
    + "LEFT JOIN Book_Authors ba ON b.bookID = ba.bookID"
    + "LEFT JOIN Authors a ON ba.authorID = a.authorID"
    + "WHERE b.bookID = @bookId"
    );

  // Se vengono trovati risultati, termina esecuizone
  if (result.recordset.length > 0) {
    const responseData = transformDbToResponse(result.recordset);
    return NextResponse.json({ success: true, result: responseData });
  }
```

```
  // Se non vengono trovati risultati, cerca su Openlibrary
  const bookDetail = await openLibraryService.getBookAsync(bookId);

  // Se non vengono trovati risultati ritorna errore 404
  if (!bookDetail) {
    return NextResponse.json({ Messaggio: 'Libro non trovato' },
    { status: 404 });
  }

  // Prima di terminare salva il risultato in database
  try {
    const bookForDB = transformOpenLibraryToDb(bookDetail);

    await pool.request()
        .input('id', sql.NVarChar, bookForDB.id)
        .input('title', sql.NVarChar, bookForDB.title)
//[...]
        .input('coverUrl', sql.NVarChar, bookForDB.coverUrl || null)
        .query(
        "INSERT INTO Books (bookID, title, firstPublicationYear,"
        + "pageNumber, averageRating,"
        + "bookDescription, subjectsJson, coverImageURL)"
        + "VALUES (@id, @title, @firstYear, @pages, @rating,"
        + "@description, @subjects, @coverUrl)");
    }
  }
// [...]

  // Restituisci i dettagli del libro ottenuti da OpenLibrary
  const response = transformOpenLibraryToResponse(bookDetail);
  return NextResponse.json(response);
}
```

3.4 Pagina libro

- L’utente può visualizzare le informazioni riguardanti il libro
- È possibile inserire il libro in scaffali specifici.



3.4 Pagina libro

- Nella pagina del libro è possibile scrivere una recensione scegliendo una valutazione, un titolo e una descrizione.
- Si possono vedere anche le recensioni di altri utenti.

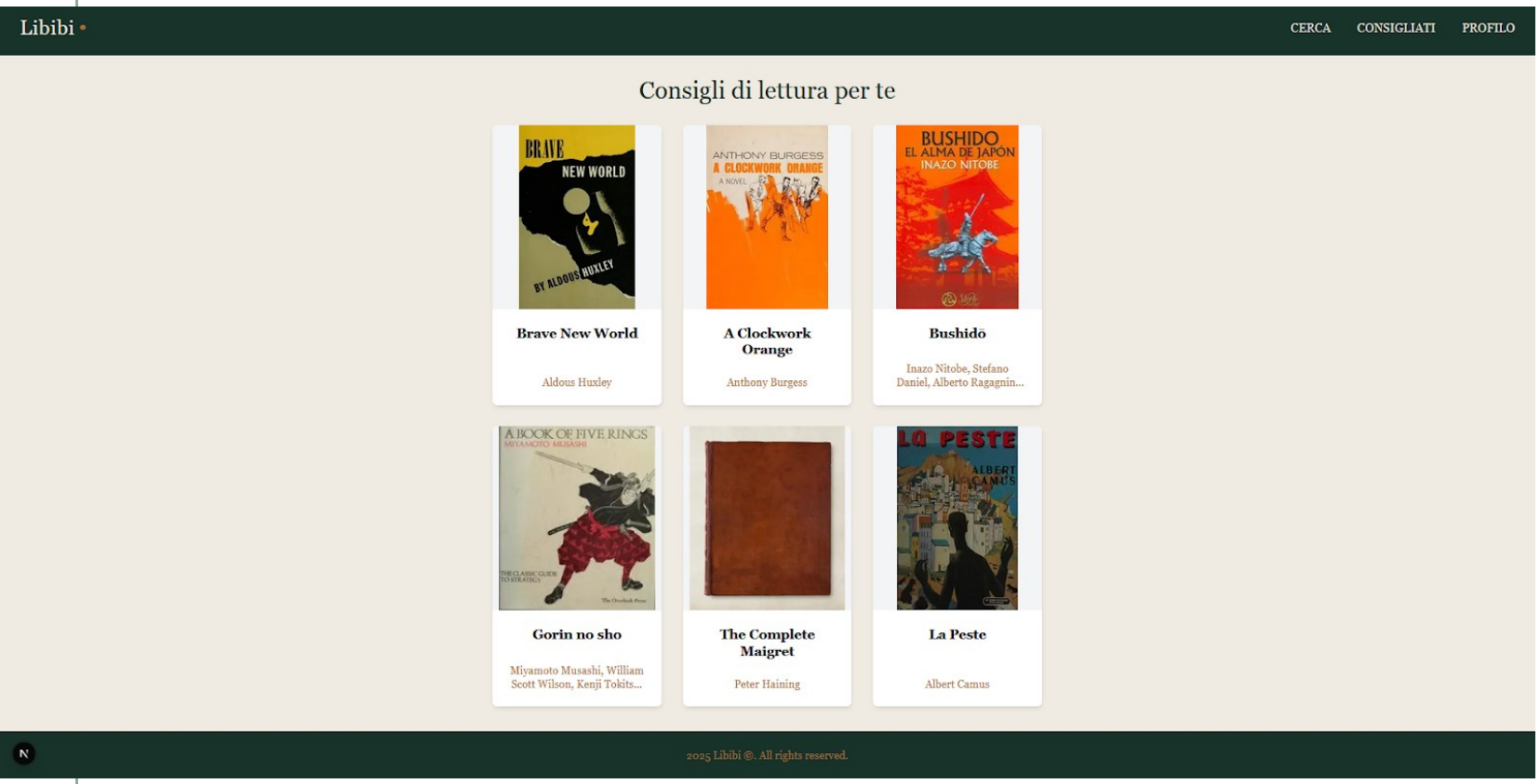
3.4 Pagina autore

- Premendo sul nome dell'autore è possibile leggerne la biografia e una breve descrizione delle sue opere più famose qualora le informazioni fossero disponibili dall'API.

3.5 Profilo utente

- Il profilo utente permette di visualizzare le **informazioni dell'utente** e una panoramica sulle **proprie liste** e le **recensioni scritte**.
- Rimuovendo dalla libreria un libro che si trova nello stato "Sto leggendo", esso verrà automaticamente spostato nello scaffale "Abbandonato".
- Tramite il pulsante di **Logout** l'utente può uscire dal proprio account.

3.6 I consigliati



- Contiene i libri consigliati per l'utente in base ai dati relativi alle sue letture.

3.6 Consigliati

```
Typ
const res = await fetch("https://openrouter.ai/api/v1/chat/completion
    method: "POST",
    headers: {
        "Authorization": "Bearer " + process.env.OPENROUTER_API_KEY,
        "Content-Type": "application/json",
    },
    body: JSON.stringify({
    model: "qwen/qwen-2.5-72b-instruct:free",
    messages: [
        { role: "system", content: systemPrompt },
        { role: "user", content: userPrompt }
    ],
    temperature: 0.7,
    top_p: 0.9,
    max_tokens: 300,
    }),
});
```

- Vengono scelti tramite il LLM Qwen2.5 - 72B instruct dall'API di OpenRouter.
- Il codice manda una **richiesta POST** all'API del sito che ospita il modello LLM da utilizzare.
- Inserisce la **chiave** necessaria per ottenere l'accesso.
- **Forma il prompt** da inviare inserendo prima il prompt di sistema e poi quello di user.
- Specifica i settaggi del modello:
    - **top_p** è la metrica che dice al modello di considerare una parte di token tra le più probabili.
    - **temperature** riguarda casualità nella scelta del token specifico nel range top_p prescelto.
    - **max_tokens** indica la lunghezza massima della risposta.

3.6 Consigliati

- Il **prompt** fornito a Qwen.
- **readerProfile** è formato dalla lista di libro e autore delle varie liste di lettura, organizzati in una lista e separati dai titoli come “VOGLIO LEGGERE:”.
- Ricevuta la risposta JSON, viene effettuato il **parsing** per estrarre stringhe “titolo, autore”.
- Viene usata la **ricerca per ottenere i libri**, prima in database e poi in OpenLibrary.

```
"Sei un bibliotecario esperto specializzato in raccomandazioni
personalizzate.\n"
+ "Analizza il profilo di lettura dell'utente e i suoi gusti per
suggerire libri perfetti per lui.\n\n"
+ "Considera:\n"
+ "- I libri che ha apprezzato di più (stelle alte)\n"
+ "- I generi e autori preferiti\n"
+ "- I libri abbandonati (da evitare generi/stili simili)\n"
+ "- Le preferenze che emergono dalle sue letture\n\n"
+ "Rispondi esclusivamente con un singolo JSON strutturato che contiene
tutti e 6 i libri.\n"
+ "Formato richiesto: {\"raccomandazioni\": [{\"titolo\": \"Nome del
libro\", \"autore\": \"Nome dell'autore\"}, {\"titolo\": \"Nome del
libro\", \"autore\": \"Nome dell'autore\"}, {\"titolo\": \"Nome del
libro\", \"autore\": \"Nome dell'autore\"}, {\"titolo\": \"Nome del
libro\", \"autore\": \"Nome dell'autore\"}, {\"titolo\": \"Nome del
libro\", \"autore\": \"Nome dell'autore\"}, {\"titolo\": \"Nome del
libro\", \"autore\": \"Nome dell'autore\"}]}";

 const userPrompt: string =
"Ecco il profilo di lettura dell'utente:\n\n"
```

4.0 Conclusioni

- **Il progetto “Libibi”** ha reso possibile realizzare un’applicazione web che permette di tenere traccia dei libri, di lasciare una recensione e anche di ottenere consigli tramite un algoritmo di intelligenza artificiale.
- **Competenze acquisite:**
  - sviluppo frontend e backend con NextJS (un framework basato su React, una delle librerie più usate per lo sviluppo web).
  - approfondimento delle conoscenze pregresse di SQL per la gestione di un database,
  - Implementazione di API pubbliche
- **In futuro:**
  - Potenziare l’aspetto social inserendo dei thread di discussione nella zona recensioni e aggiungendo una sezione per recensioni in formato video.
  - Ampliare le funzionalità di modifica del profilo utente.
  - Migliorare la ricerca accorpando diverse edizioni e lingue in un singolo meta-libro.