

CM3010 Databases & Advanced Data Techniques

Midterm Coursework Assignment

Report

1. Overview of Dataset

Dataset: <https://www.kaggle.com/datasets/farehasultan/bookstore?resource=download>

In the pursuit of comprehensive insights into the intricate dynamics of the book industry, this project centres around the exploration of a meticulously curated dataset derived from Kaggle. The dataset, sourced from bookdepository.com, offers detailed information about the various books, including their titles, authors, publications details and pricing.

The dataset comprises 14 columns and 537 rows of data, providing a comprehensive snapshot of the bookstore's inventory. Each column represents a specific attribute related to a book, including title, URL, image, format, year, price, author, rating, publisher, length, ISBN, city/country, categories, and quantity.

Dataset Columns	Functions
Title	Stores the titles of the books
URL	Contains the URL of the book on the Book Depository
Image	Stores the image of the URL of the book
Format	Indicates the format of the book
Year	Specifies the publication year of the book
Price	Represents the price of the book
Author	Contains the name of the author
Rating	Stores the rating assigned to the book
Publisher	Indicates the publisher company of the book
Length	Represents the length of the book in terms of the number of pages
ISBN	Stores the unique ISBN of the book
City/Country	Represents the location of the author
Categories	Specifies the category to which the book belongs
Quantity	Stores the quantity of available stocks of the book

1.1 Dataset Quality Assessment

1.1.1 Quality

The dataset demonstrates good quality, primarily originating from a reputable source, Kaggle. However, there are apparent challenges that need addressing during preprocessing, such as repetitive data and instances of multiple entries within a single column

The presence of repetitive data might impact data accuracy, and a cleanup strategy is required. There are also some instances of multiple entries in a single column, for example in “city/Country” and “Categories” can complicate relational modeling and demand structured handling.

1.1.2 Level of Detail

The dataset provides a rich level of detail, encompassing various aspects of each book, including title, author, format, pricing, and more. The detailed nature of the dataset allows for a comprehensive exploration of the book inventory. Such detailed data sets the stage for in-depth explorations and analyses.

1.1.3 Documentation

While the dataset lacks explicit documentation, the column headers offer clarity on the information presented. To enhance user understanding, a more detailed description of the dataset's structure and potential challenges would be beneficial, ensuring transparency and facilitating effective utilization.

1.1.4 Interrelation

The data demonstrates strong interrelation, particularly between columns like "Author," "Publisher," and "Categories," allowing for intricate modelling. The interrelated nature of the data supports the creation of a sophisticated database structure. Relationships between authors, publishers, and categories can be leveraged for complex queries and analyses.

1.1.5 Use

The dataset proves versatile, suitable for various purposes such as book catalogue management, sales analysis, and user recommendations. Its adaptability renders it valuable for both casual

exploration and in-depth analyses, positioning it as a valuable resource within the realm of book-related insights.

1.1.6 Discoverability

The dataset's discoverability through Kaggle enhances accessibility for interested users and researchers. Its availability on this platform ensures widespread access, fostering a collaborative environment for data exploration and analysis.

1.1.7 Terms of Use

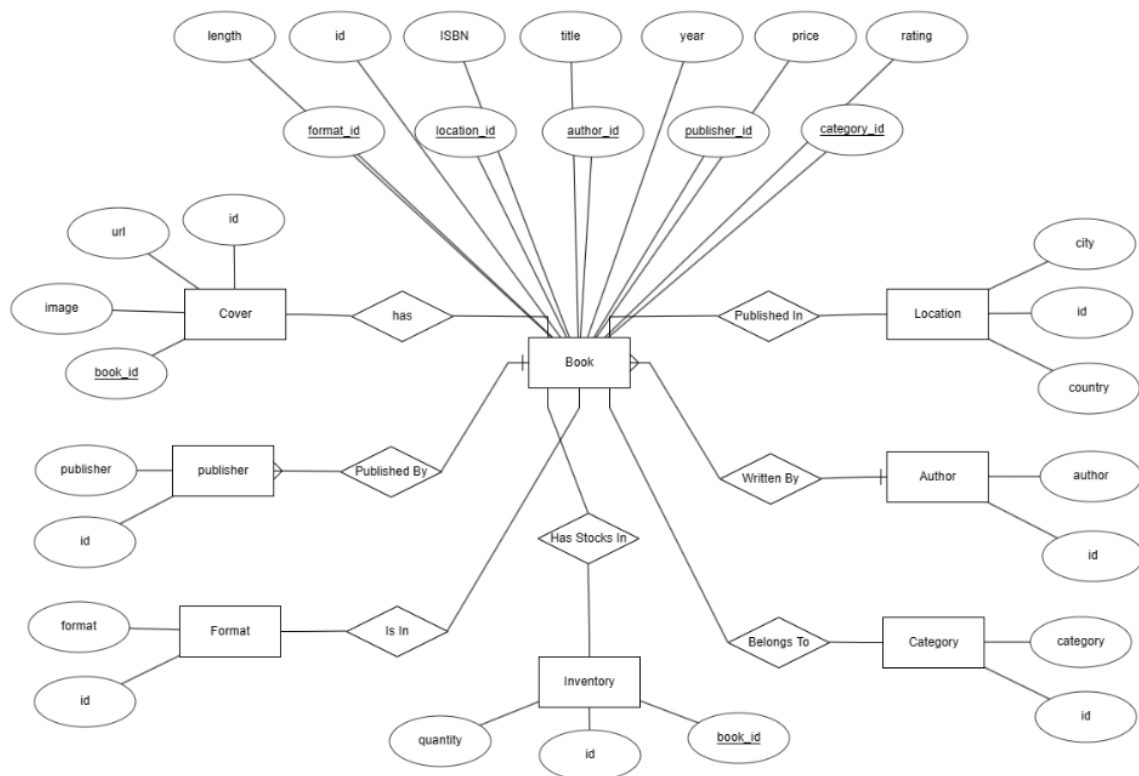
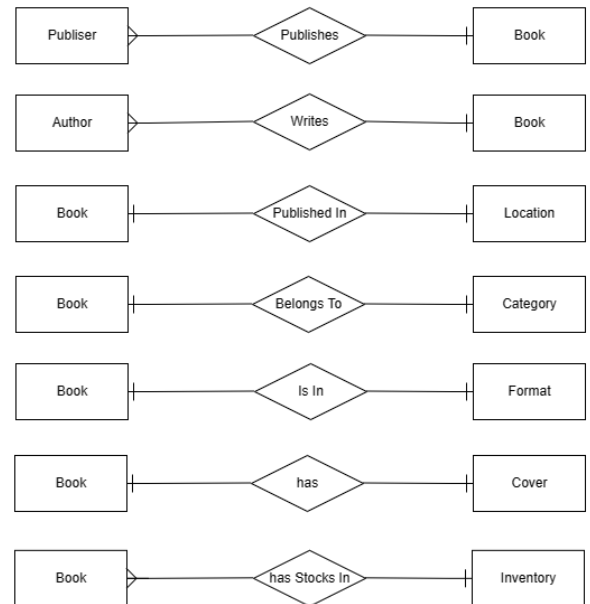
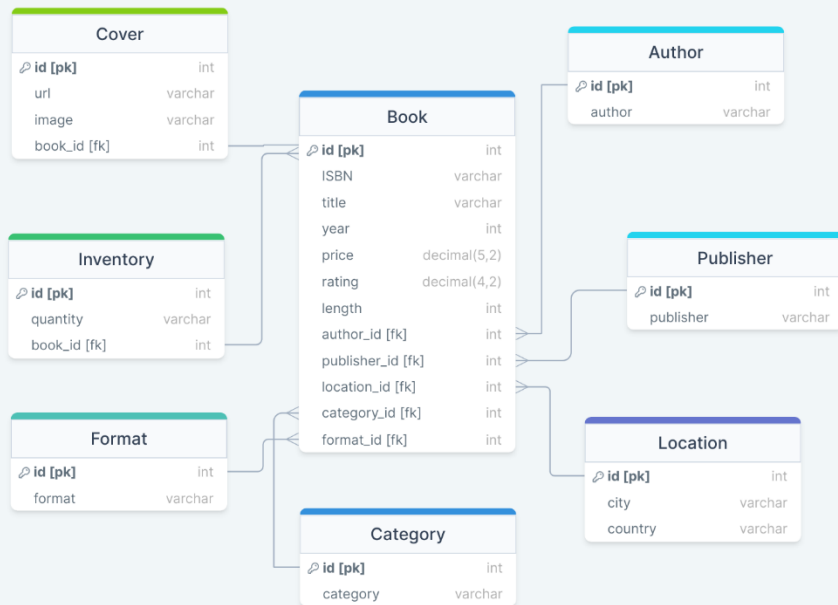
All books are hosted by bookdepository.com and the dataset's use aligns with fair use principles for academic purposes.

1.1.8 Interest in the Dataset

The dataset piques interest due to its potential to unveil intricate patterns within the book industry. A deep dive into the data could unravel insights into pricing dynamics, author popularity, and the nuanced interplay between book format, length, and user ratings.

1	Is there a way to filter the dataset to display information from a particular column, such as authors, publishers, or available formats?
2	How can I efficiently search for books categorized under a specific category in the dataset?
3	How can I locate books written by a particular author using the available dataset?
4	How can I identify books published by a specific publisher from the dataset?
5	How can I determine which books have both the highest and lowest quantities in stock?
6	Which books in the dataset have the highest ratings and how to identify them?

2. Modeling Data



Stage 3. Create the database

we detail the process of creating and populating a MySQL database for a CRUD bookstore web application. The implemented model encompasses tables for authors, publishers, locations, categories, formats, books, covers, and inventory. This document outlines the accuracy of the implementation, the sensibility of design choices, and critical reflections on data and implementation issues.

3.1 Creating a MySQL database and establishing a connection

Importing MySQL Module

This line imports the 'mysql' module, allowing the script to interact with MySQL databases using Node.js.

```
// Import the MySQL module
const mysql = require('mysql');
```

Database Connection Configuration

Here, a connection configuration object is created, specifying the host (localhost), username (root), and password for connecting to the MySQL server.

```
// Establish a database connection configuration
const db = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "Lavanya_lavanya02",
});
```

Connecting to MySQL Server

The script attempts to connect to the MySQL server using the provided configuration. If an error occurs during the connection, it will be thrown and logged. If successful, a message indicating a successful connection is logged to the console.

```
// Connect to the MySQL server
db.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
```

Creating the Database

This part of the script executes a query to create a database named "Books" if it doesn't already exist. The query is wrapped in an "IF NOT EXISTS" clause to avoid attempting to recreate the database if it has already been created. Any errors during this operation are thrown and logged. A message is also logged to indicate that the "Books" database has been created.

```
// Create the database if it doesn't already exist
db.query("CREATE DATABASE IF NOT EXISTS Books", function (err, result) {
  if (err) throw err;
  console.log("Books database created");
});
});
```

Finally, the script closes the database connection after the operations are completed.

3.2 Reading and Populating the Database

Once creating the SQL Database and establishing the connection, we will move on to reading and populating the database.

Data Conversion and Insertion

The script uses the 'csvtojson' library to convert data from a CSV file (book_dataset.csv) into JSON format.

```
// Import required modules for MySQL database connection and CSV-to-JSON conversion
const mysql = require('mysql');
const csvtojson = require('csvtojson');
```

Table Creation and Insertion

Tables are created for various entities such as author, publisher, location, category, format, book, cover, and inventory. Tables are created for various entities such as author, publisher, location, category, format, book, cover, and inventory. The **DROP TABLE** command is used to remove existing tables, ensuring a clean slate. Foreign key relationships are established between the book table and other related tables.

The converted JSON data is then iterated, and unique fields are stored in lists. The script then iterates over these lists to insert data into the respective tables (author, publisher, location, category, format, book, cover, inventory).

```
// This table stores the author's id, name, etc...
var authorTable = "CREATE TABLE IF NOT EXISTS author (" +
    "id INT UNSIGNED AUTO_INCREMENT NOT NULL," +
    "author VARCHAR(255) NOT NULL," +
    "PRIMARY KEY(id)";
```

Data Insertion and Processing

Some of the fields have repetitive data, and therefore only unique data are pushed into the lists.

The below code initializes various arrays to store unique values for authors, publishers, locations, categories, formats, and full data for all fields. It iterates through the CSV data, extracts values for each field, and populates the respective arrays. The code manages unique values to avoid redundant data in tables (e.g., author, publisher, category, and format).

```
// author, publisher, category, format tables contain repetitive data which results in a large
// dataset, so only push unique ones into the lists

if (author_list.includes(author) === false) author_list.push(author);
if (publisher_list.includes(publisher) === false) publisher_list.push(publisher);
if (category_list.includes(category) === false) category_list.push(category);
if (format_list.includes(format) === false) format_list.push(format);
```

The code below, iterates through the unique value arrays and inserts data into corresponding tables ('author', 'publisher', 'location', 'category', 'format'). This process populates the 'author' table with unique authors, assigning each one a unique identifier ('id') for future reference in the 'book' table.

```
// Iterate over the author_list to insert data into the 'author' table
for (var i = 0; i < author_list.length; i++) {
    // SQL insert command
    var sqlInsert = `INSERT INTO author VALUES (?, ?)`;

    // Data to be inserted. Since the loop starts at 0, increment i by 1 for the ID.
    var data = [i + 1, author_list[i]];

    // Inserting data of the current row into the database
    db.query(sqlInsert, data, (err, results, fields) => {
        if (err) {
            console.log("Unable to insert author item at row ", i + 1);
            return console.log(err);
        }
    });
}
```

Processing Location Data

In processing the location data, the location data in the CSV file contains both city and country information in a single field.

the 'city/country' field is split, and city and country data are processed separately. Special attention is given to cases where there is no location data or more than two elements in the split.

If there is no location data, null values are pushed. If there's one element, it is pushed into the country list with null in the city. For exactly two elements, both city and country are pushed normally.

```
for (var i = 0; i < location_list.length; i++) {  
    // Initialize empty string variable for saving city data later  
    var city_temp = "";  
  
    // Contains city and country data that needs to be saved into separate lists  
    var city_country_pair = location_list[i].split(", ");  
  
    // Handle different cases based on the length of city_country_pair  
    if (city_country_pair == "") {  
        city.push("NULL");  
        country.push("NULL");  
    } else if (city_country_pair.length == 1) {  
        city.push("NULL");  
        country.push(city_country_pair[0]);  
    } else if (city_country_pair.length == 2) {  
        city.push(city_country_pair[0]);  
        country.push(city_country_pair[1]);  
    }  
}
```

The city_temp variable is used to concatenate city data for cases with more than two elements. A loop handles this concatenation, and the last element becomes the country data.

```
// Iterate over the location_list to process and insert data into the 'location' table  
else if (city_country_pair.length > 2) {  
    for (var x = 0; x < city_country_pair.length - 1; x++) {  
        if (x == city_country_pair.length - 2) {  
            city_temp = city_temp + city_country_pair[x];  
        } else {  
            city_temp = city_temp + city_country_pair[x] + ", ";  
        }  
    }  
    city.push(city_temp);  
    country.push(city_country_pair[city_country_pair.length - 1]);  
}
```


Data Integrity and Foreign Key IDs

To maintain data integrity, foreign key IDs for fields like author, publisher, location, category, and format are retrieved correctly. The full field data (repetitive) is compared to the unique field data (non-repetitive) to obtain the correct IDs.

```
for (var a = 0; a < author_list.length; a++) {  
    if (author_full[i] == author_list[a]) {  
        author_id = a + 1;  
    }  
}  
  
for (var a = 0; a < publisher_list.length; a++) {  
    if (publisher_full[i] == publisher_list[a]) {  
        publisher_id = a + 1;  
    }  
}
```

Loops iterate through the populated lists, inserting data into corresponding tables. Foreign key IDs are correctly associated with the 'book' table to maintain relationships.

```
// Insert command  
var sqlInsert = `INSERT INTO book VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?)`;  
  
// Data to be inserted. Since the loop starts at 0, increment i by 1  
var data = [i + 1, ISBN_list[i], title_list[i], year_list[i], price_list[i], rating_list[i], length_list[i],  
    author_id, publisher_id, location_id, category_id, format_id];  
  
// Inserting data of the current row into the database  
db.query(sqlInsert, data, (err, results, fields) => {  
    if (err) {  
        console.log("Unable to insert book item at row ", i + 1);  
        return console.log(err);  
    }  
});  
}
```

After successful data insertion, a log message is printed, indicating that the database is correctly populated.

The code employs a systematic approach to convert CSV data into a well-structured MySQL database. It handles unique and repetitive data efficiently, ensuring data integrity and avoiding redundancy. Attention to detail is evident in the handling of location data and the retrieval of foreign key IDs.

Logging and Error Handling

My script provides appropriate log messages to indicate successful connections, database creation, and data insertion. Error handling is implemented, and error messages are logged, providing useful information in case of issues.

3.3 Enhanced User Experience and Optimization

Despite the successful normalization of the database, some challenges arise when trying to intuitively navigate relationships between tables. Specifically, understanding the author of a particular book requires cross-referencing the book table with the author table using the `author_id` foreign key. To address these challenges and enhance user experience, the web app provides comprehensive functionality and optimization.

Comprehensive Data Presentation

The web app is designed to offer users an all-encompassing view of the data, eliminating the need to navigate between tables to find relevant information. Users can access all fields related to books, authors, publishers, locations, categories, and formats at a glance, simplifying data exploration.

Sorting Rows

To facilitate easy data exploration, the web app allows users to sort rows based on a particular column in either ascending or descending order. This functionality is implemented in the `public/js/scripts.js` file, where a dynamic sorting function is defined. Users can click on column headers to trigger sorting, providing a seamless and intuitive experience.

Search Functionality

The web app addresses various user queries through search functionalities. Users can effortlessly search for specific book titles, books written by a particular author, books published by a specific publisher, or books within a particular category. The search mechanism is implemented using SQL queries in the `routes/main.js` file.

To Search for a Book Title: A search box is created in the index.ejs file. The entered keyword is sent via a POST request to the path /.

In main.js, SQL queries with inner joins are utilized to search for the keyword in the title field across related tables. The results are then returned to the index.ejs file for display.

Search for Books Written by a Particular Author: Similar to the book title search, the author_id foreign key is used to join the book and author tables.

Search for Books Published by a Particular Publisher: The publisher_id foreign key is employed to link the book and publisher tables.

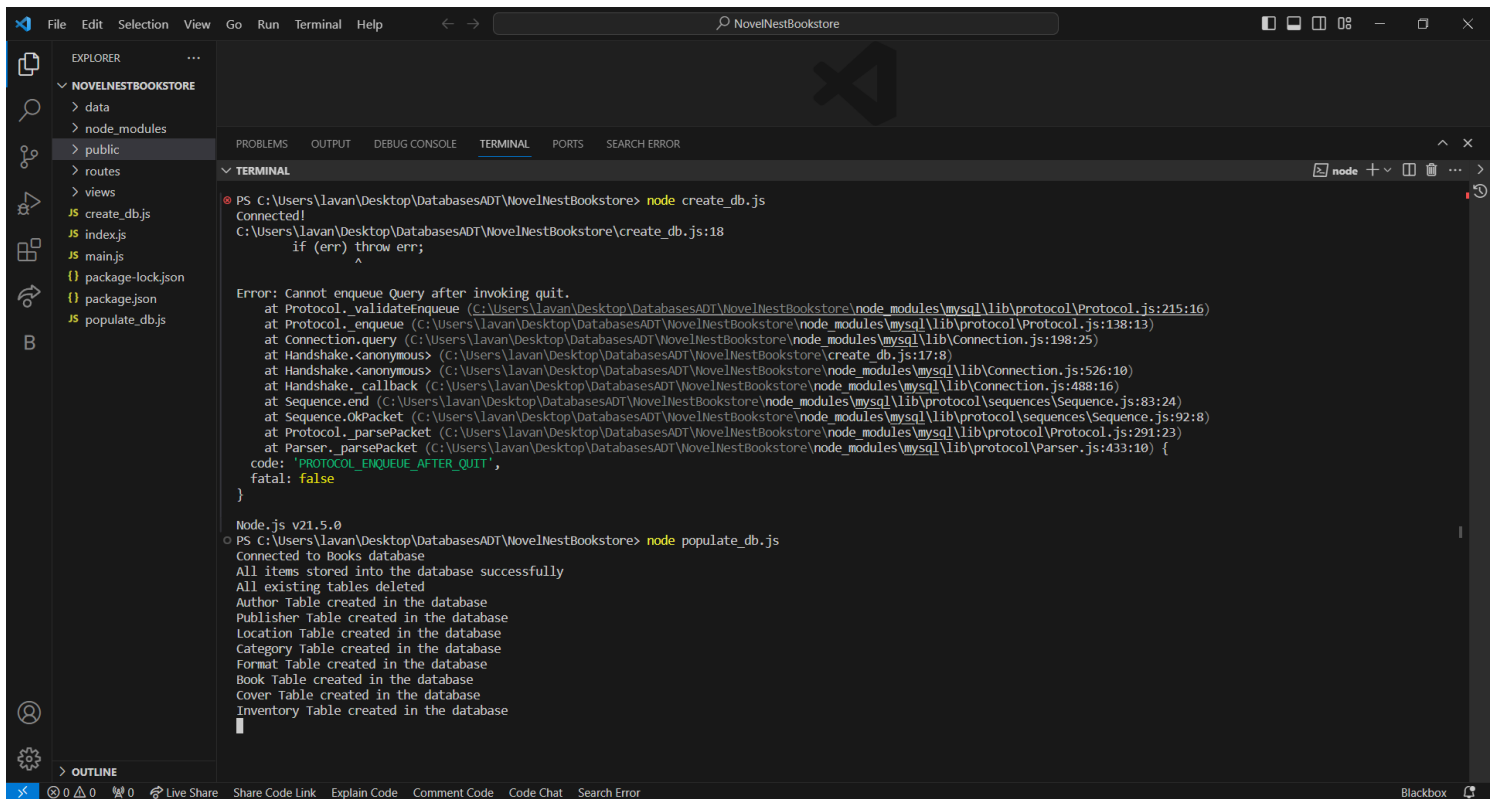
Search for Books by a Particular Category: The category_id foreign key is used for linking the book and category tables.

Individual Table Views

For users who prefer to focus on specific columns, each table in the database has a dedicated EJS file (author.ejs, publisher.ejs, etc.). These individual views provide a detailed look at the data within each table. Furthermore, search boxes are integrated into these views to enable targeted searches within specific tables.

In conclusion, the web app not only facilitates seamless navigation through the normalized database but also addresses the challenges associated with intuitively understanding relationships between tables. The implemented functionalities, including comprehensive data presentation, sorting, and search capabilities, contribute to an enhanced user experience. Moreover, the optimization achieved through normalization ensures efficient data storage and retrieval, making the web app a powerful tool for exploring and managing diverse information related to books.

3. Screenshots of the Web Application

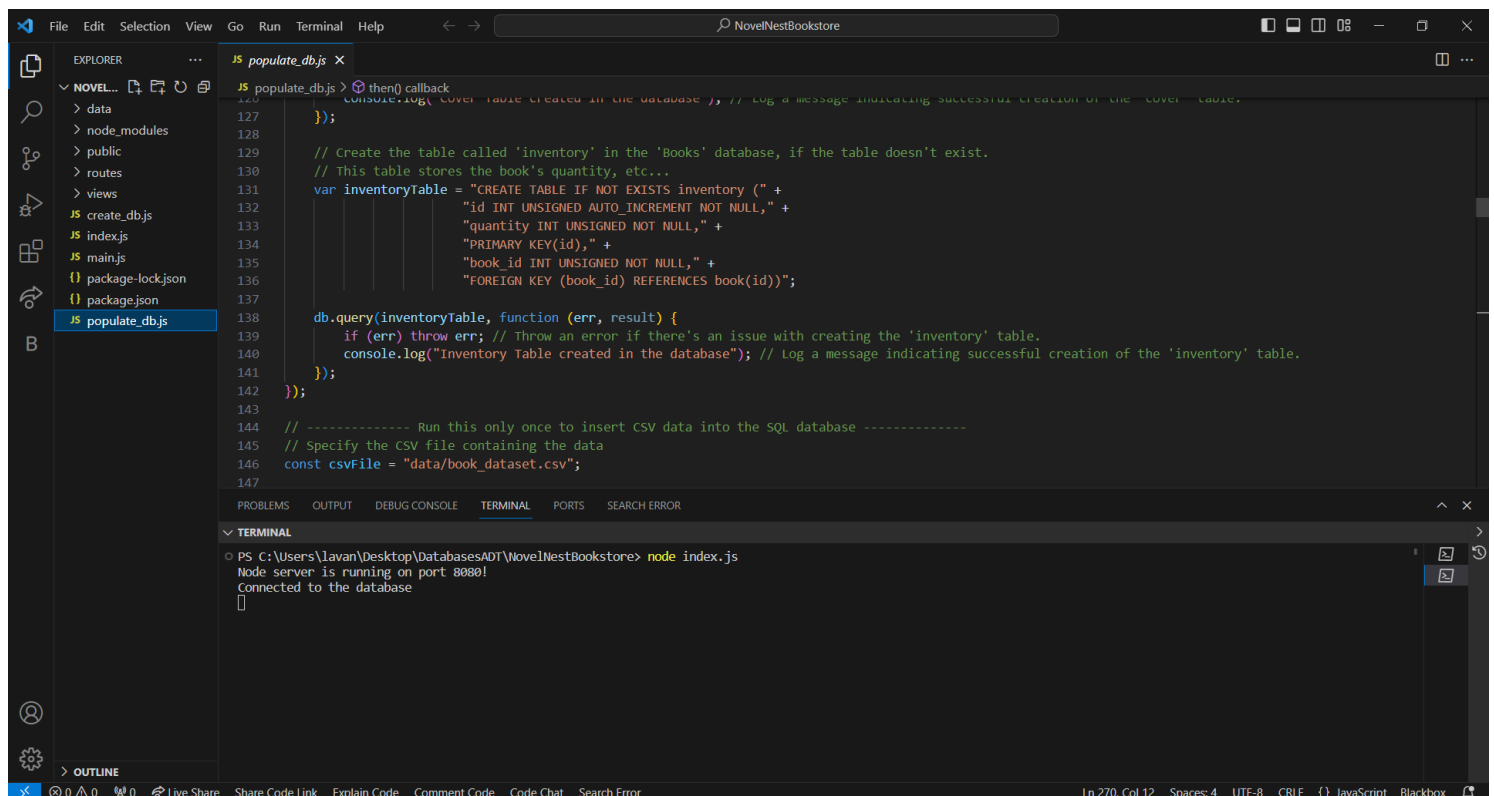


The screenshot shows the Visual Studio Code interface with the Explorer pane on the left displaying the file structure of the 'NOVELNESTBOOKSTORE' project. The file 'create_db.js' is selected. The Terminal pane at the bottom shows the execution of two commands: 'node create_db.js' and 'node populate_db.js'. The output of 'create_db.js' shows a connection to the database and a successful creation of the 'Books' database. The output of 'populate_db.js' shows the creation of several tables: Author, Publisher, Location, Category, Format, Book, Cover, and Inventory. The status bar at the bottom indicates the file is 'create_db.js' and the terminal is running 'node create_db.js'.

```
PS C:\Users\lavan\Desktop\DatabasesADT\NovelNestBookstore> node create_db.js
Connected!
C:\Users\lavan\Desktop\DatabasesADT\NovelNestBookstore\create_db.js:18
    if (err) throw err;
    ^
Error: Cannot enqueue Query after invoking quit.
    at Protocol.validateEnqueue (C:\Users\lavan\Desktop\DatabasesADT\NovelNestBookstore\node_modules\mysql\lib\protocol\Protocol.js:215:16)
    at Protocol.enqueue (C:\Users\lavan\Desktop\DatabasesADT\NovelNestBookstore\node_modules\mysql\lib\protocol\Protocol.js:138:13)
    at Connection.query (C:\Users\lavan\Desktop\DatabasesADT\NovelNestBookstore\node_modules\mysql\lib\connection.js:198:25)
    at Handshake.<anonymous> (C:\Users\lavan\Desktop\DatabasesADT\NovelNestBookstore\create_db.js:17:8)
    at Handshake._callback (C:\Users\lavan\Desktop\DatabasesADT\NovelNestBookstore\node_modules\mysql\lib\Connection.js:526:10)
    at Handshake._callback (C:\Users\lavan\Desktop\DatabasesADT\NovelNestBookstore\node_modules\mysql\lib\Connection.js:488:16)
    at Sequence.end (C:\Users\lavan\Desktop\DatabasesADT\NovelNestBookstore\node_modules\mysql\lib\protocol\sequences\Sequence.js:83:24)
    at Protocol.okPacket (C:\Users\lavan\Desktop\DatabasesADT\NovelNestBookstore\node_modules\mysql\lib\protocol\sequences\Sequence.js:92:8)
    at Protocol.parsePacket (C:\Users\lavan\Desktop\DatabasesADT\NovelNestBookstore\node_modules\mysql\lib\protocol\Protocol.js:291:23)
    at Parser.parsePacket (C:\Users\lavan\Desktop\DatabasesADT\NovelNestBookstore\node_modules\mysql\lib\protocol\Parser.js:433:10) {
  code: 'PROTOCOL_ENQUEUE_AFTER_QUIT',
  fatal: false
}

Node.js v21.5.0
PS C:\Users\lavan\Desktop\DatabasesADT\NovelNestBookstore> node populate_db.js
Connected to Books database
All items stored into the database successfully
All existing tables deleted
Author Table created in the database
Publisher Table created in the database
Location Table created in the database
Category Table created in the database
Format Table created in the database
Book Table created in the database
Cover Table created in the database
Inventory Table created in the database
```

Fig.1 Create Database: node create_db.js & Populate Database: node populate_db.js



The screenshot shows the Visual Studio Code interface with the Explorer pane on the left displaying the file structure of the 'NOVELNESTBOOKSTORE' project. The file 'populate_db.js' is selected. The Source Code pane shows the content of 'populate_db.js', which includes comments and SQL queries for creating the 'inventory' table and inserting data from a CSV file. The Terminal pane at the bottom shows the execution of 'node index.js', which outputs 'Node server is running on port 8080!' and 'Connected to the database'. The status bar at the bottom indicates the file is 'populate_db.js' and the terminal is running 'node index.js'.

```
JS populate_db.js
127 // Create the table called 'inventory' in the 'Books' database, if the table doesn't exist.
128 // This table stores the book's quantity, etc...
129 var inventoryTable = "CREATE TABLE IF NOT EXISTS inventory (" +
130   "id INT UNSIGNED AUTO_INCREMENT NOT NULL," +
131   "quantity INT UNSIGNED NOT NULL," +
132   "PRIMARY KEY(id)," +
133   "book_id INT UNSIGNED NOT NULL," +
134   "FOREIGN KEY (book_id) REFERENCES book(id))";
135
136 db.query(inventoryTable, function (err, result) {
137   if (err) throw err; // Throw an error if there's an issue with creating the 'inventory' table.
138   console.log("Inventory Table created in the database"); // Log a message indicating successful creation of the 'inventory' table.
139 });
140
141 // ----- Run this only once to insert CSV data into the SQL database -----
142 // Specify the CSV file containing the data
143 const csvFile = "data/book_dataset.csv";
144
145
146
147
PS C:\Users\lavan\Desktop\DatabasesADT\NovelNestBookstore> node index.js
Node server is running on port 8080!
Connected to the database
```

Fig. 2 Run the server: node index.js

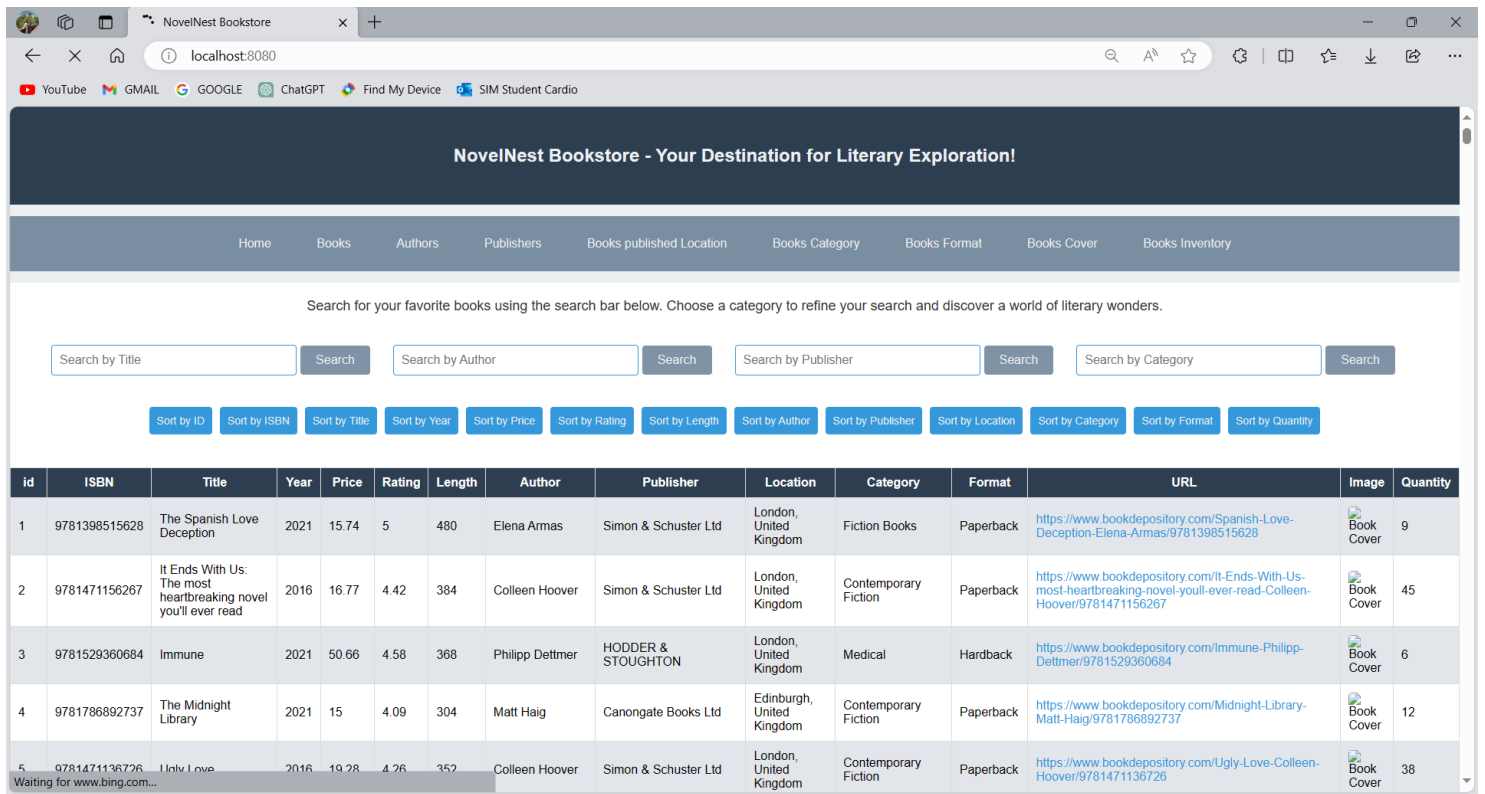


Fig. 3 Home Page

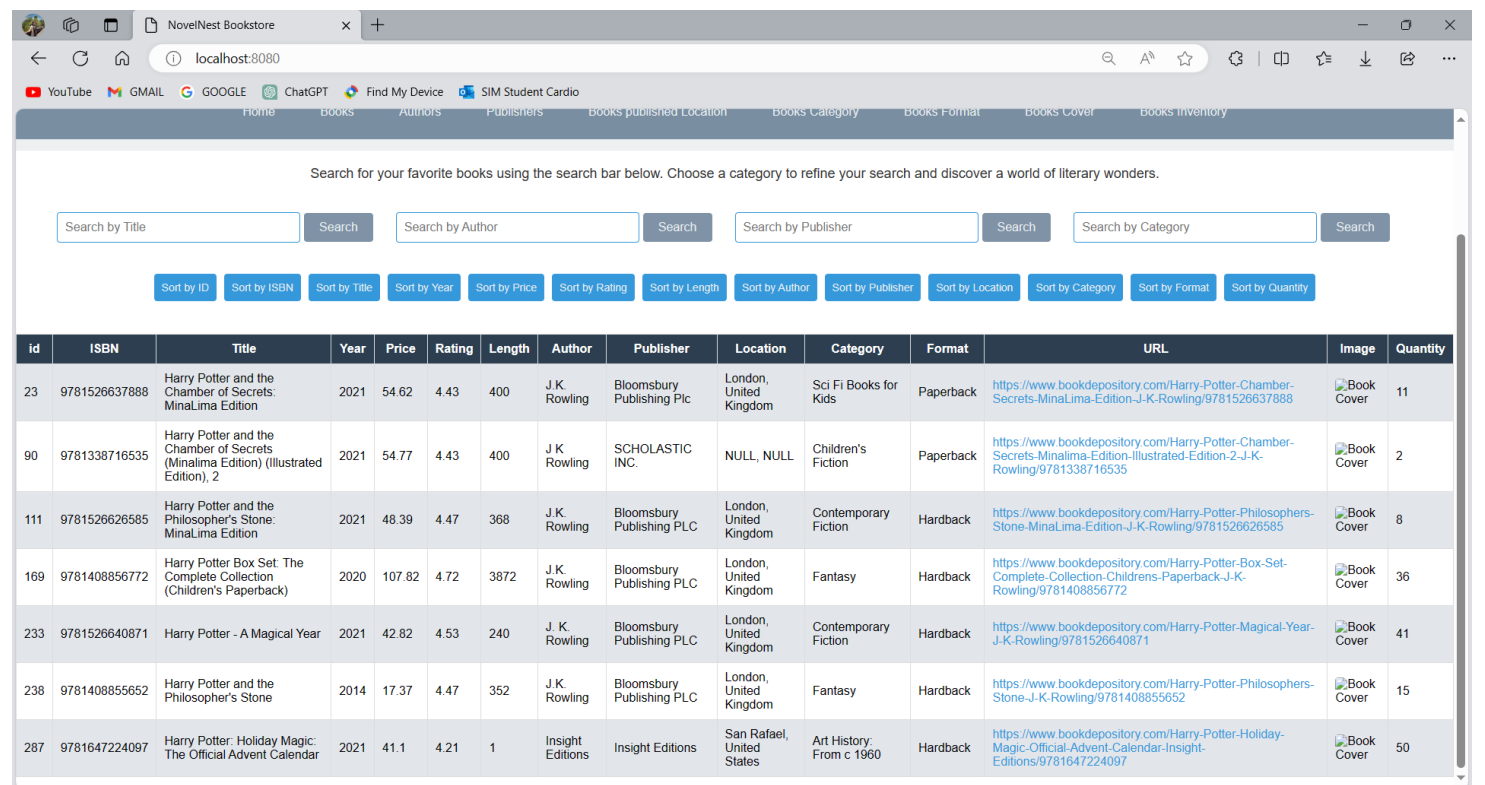


Fig. 4 Home Page – Sort by Price

Fig 4 shows that in home page when the sort by Price is clicked, it sorts the books by the price.

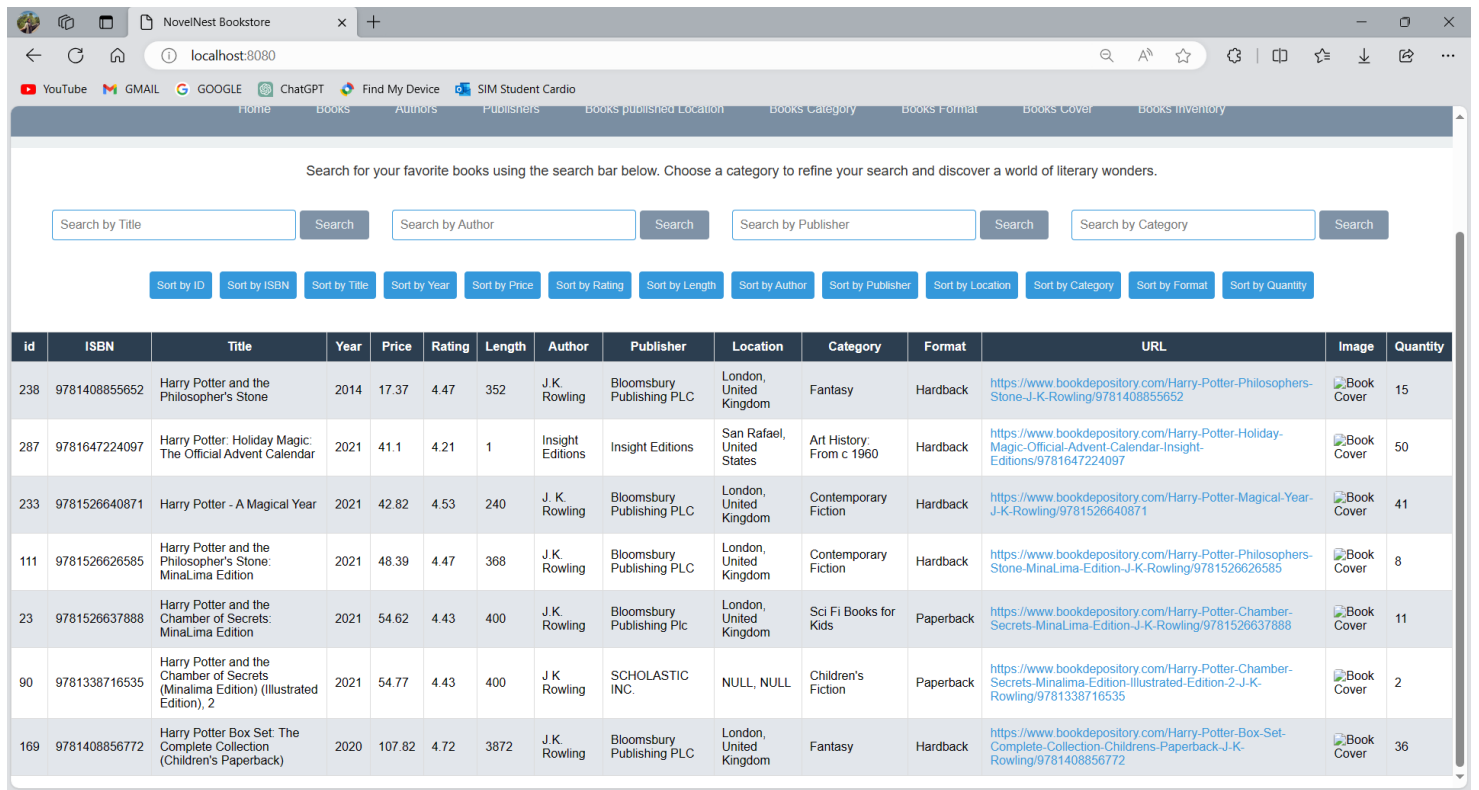


Fig. 5 Home Page – Search by Title

Fig 5 shows the home page when the search by book is implemented. Shows the list of 'Harry Potter' Books.

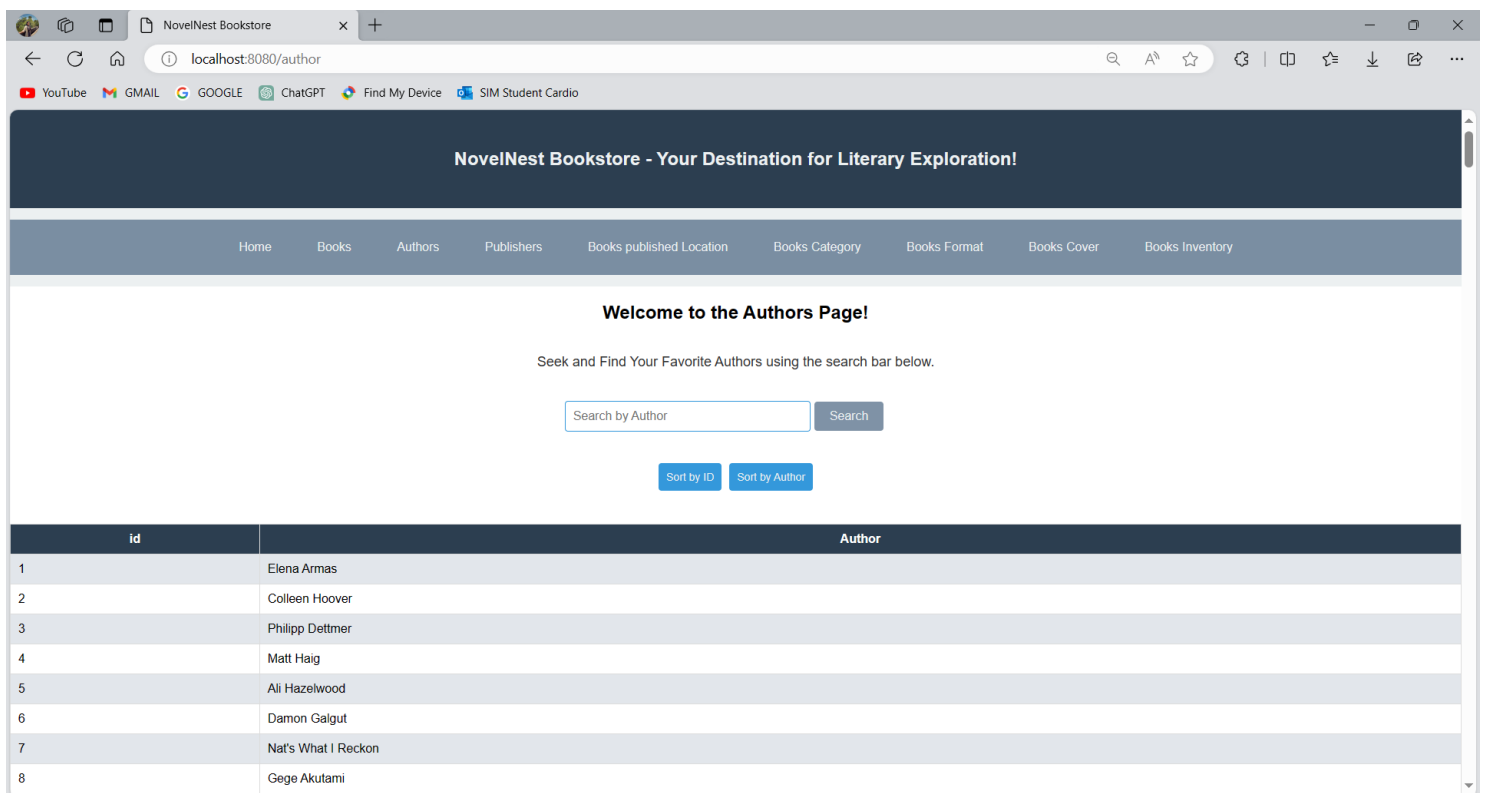


Fig. 6 Authors Page

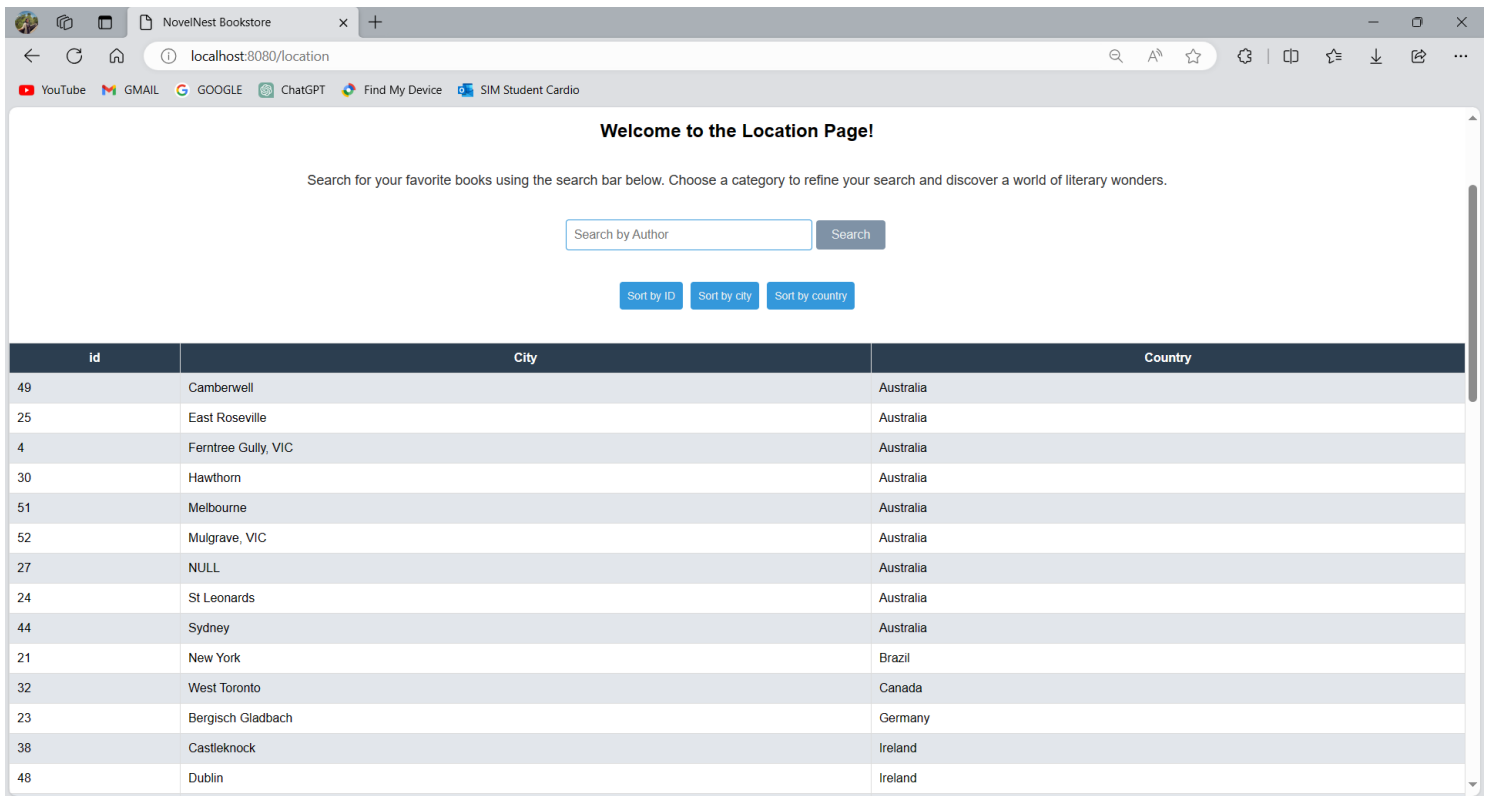


Fig. 7 Location Page – sorted by country

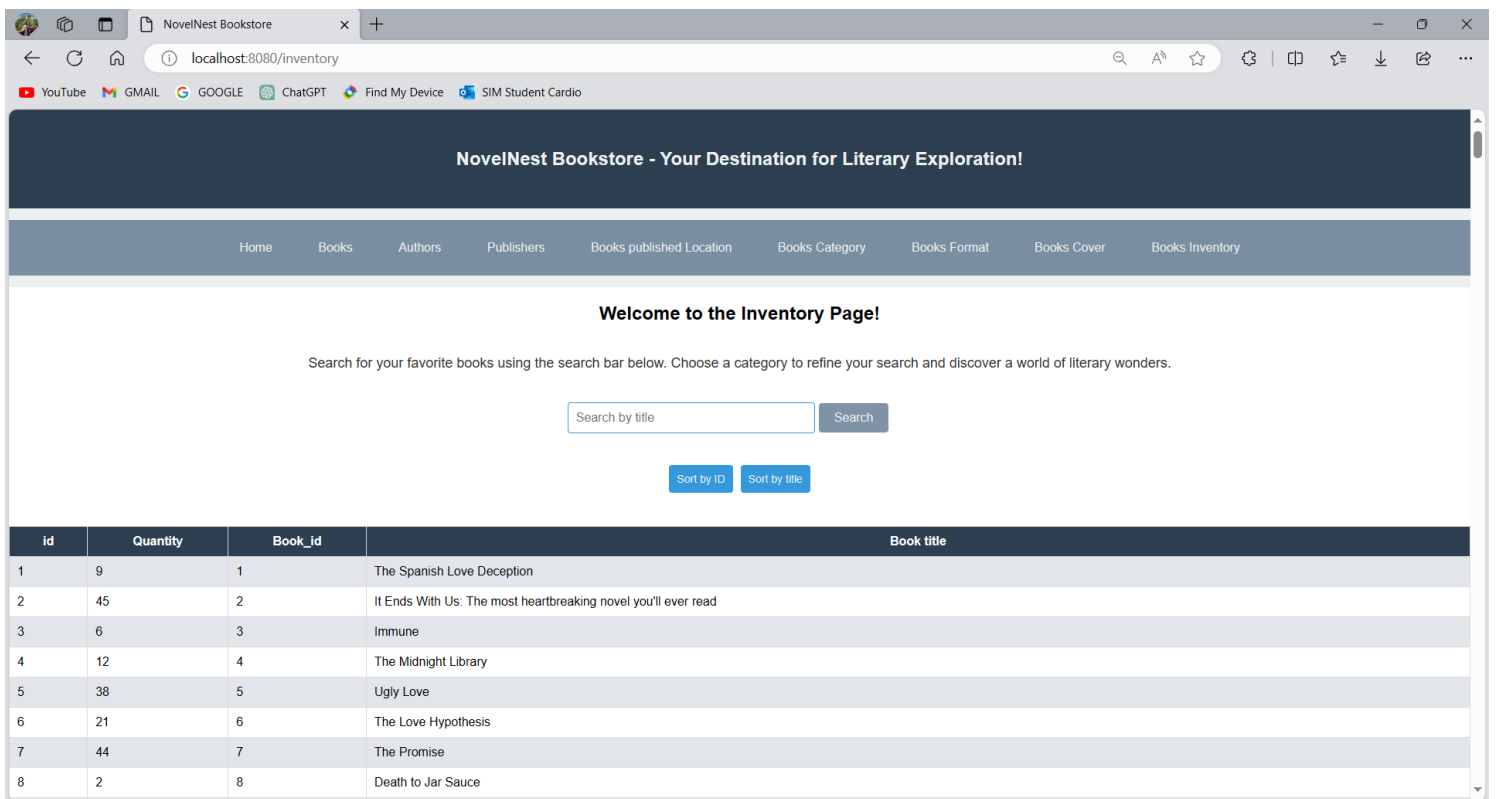


Fig. 8 Inventory Page