



Database Schema

1. books

Column Name	Data Type	Description
id	Integer	Book ID (Primary Key)
title	String	Book title
author_id	Integer	Foreign key referencing authors.id
genre_id	Integer	Foreign key referencing genres.id

2. authors

Column Name	Data Type	Description
id	Integer	Author ID (Primary Key)
name	String	Author's name

3. genres

Column Name	Data Type	Description
id	Integer	Genre ID (Primary Key)
name	String	Genre name

4. users

Column Name	Data Type	Description
id	Integer	User ID (Primary Key)
name	String	User name

email	String	User email
-------	--------	------------

Assignment Tasks

API Functionality (Required)

Implement REST API endpoints for each table with these operations:

Table	Endpoints
books	GET, POST, PUT, DELETE
authors	GET, POST, PUT, DELETE
genres	GET, POST, PUT, DELETE
users	GET, POST, PUT, DELETE

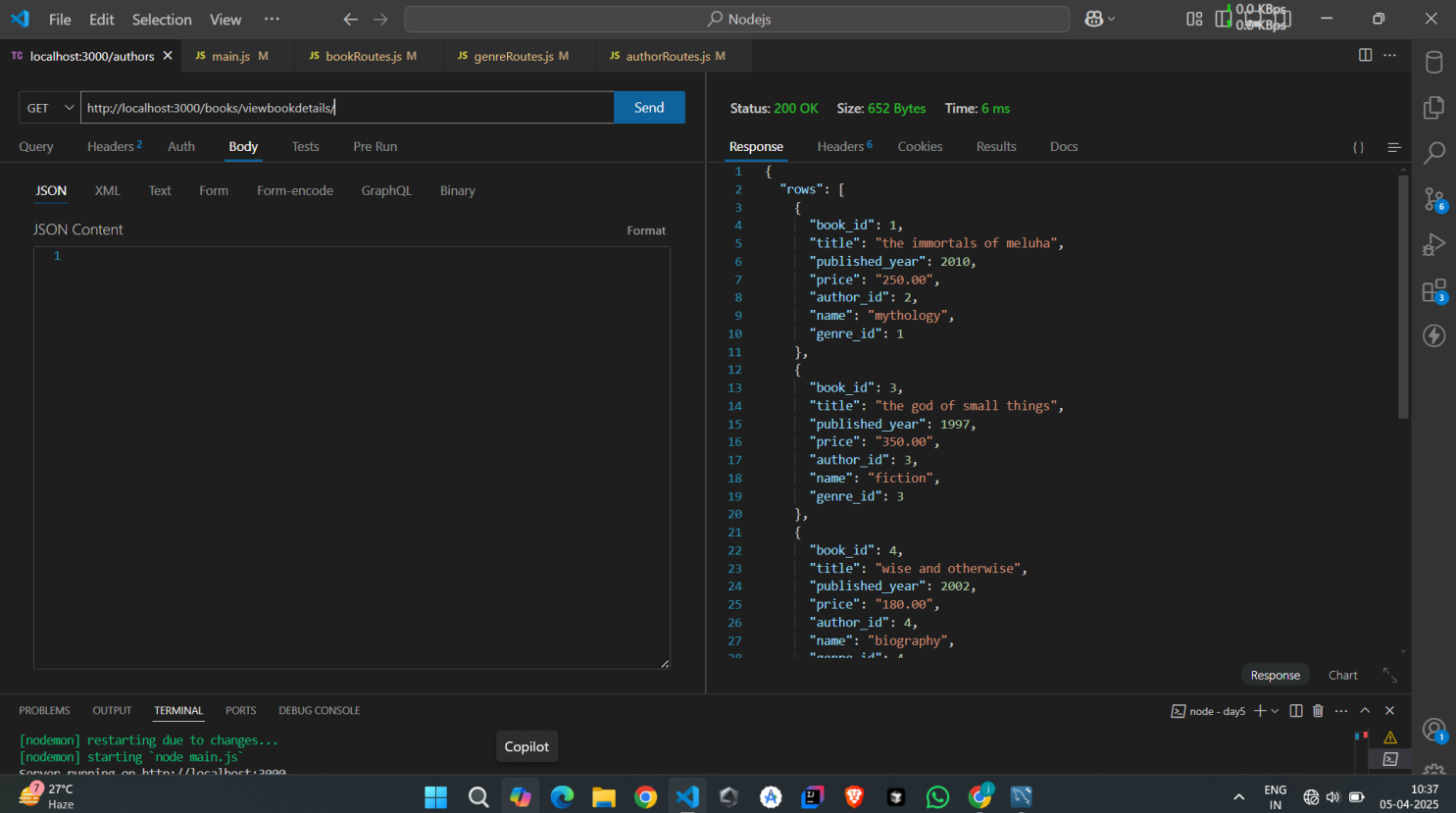
You Should Be Able To

- List all books **with author and genre names**.
 - Create new authors, genres, books, and users.
 - Update and delete records by ID.
-

[Github Link](#)

OUTPUTS

List all books by author and genre names:



The screenshot shows a REST client interface in VS Code. The request is a GET to `http://localhost:3000/books/viewbookdetails/`. The response is a JSON array of 3 books, each with fields: `book_id`, `title`, `published_year`, `price`, `author_id`, `name`, and `genre_id`.

```
1 {
2   "rows": [
3     {
4       "book_id": 1,
5       "title": "the immortals of meluha",
6       "published_year": 2010,
7       "price": "250.00",
8       "author_id": 2,
9       "name": "mythology",
10      "genre_id": 1
11    },
12    {
13      "book_id": 3,
14      "title": "the god of small things",
15      "published_year": 1997,
16      "price": "350.00",
17      "author_id": 3,
18      "name": "fiction",
19      "genre_id": 3
20    },
21    {
22      "book_id": 4,
23      "title": "wise and otherwise",
24      "published_year": 2002,
25      "price": "180.00",
26      "author_id": 4,
27      "name": "biography",
28      "genre_id": 4
29    }
30  ]
31 }
```

Sql query:

```
select
book_id,title,published_year,price,b.author_id,a.name,g.genre_id,g.n
ame from books b
join authors a on b.author_id=a.author_id
join genres g on b.genre_id=g.genre_id;
```

Create new authors, genres, books, and users.

The screenshot shows the VS Code interface with a REST client request to `http://localhost:3000/authors/` using the POST method. The request body is a JSON object: `{ "name": "myAuthor", "country": "india", "birth_year": 2002 }`. The response is a JSON object: `{ "id": 8 }`. The status is 201 Created, size is 8 Bytes, and time is 12 ms. The terminal shows the server running on `http://localhost:3000`.

```
POST http://localhost:3000/authors/

JSON Content
1 {
2   "name": "myAuthor",
3   "country": "india",
4   "birth_year": 2002
5 }
6 }
```

```
1 {
2   "id": 8
3 }
```

Status: 201 Created Size: 8 Bytes Time: 12 ms

Response Headers Cookies Results Docs

1 {
2 "id": 8
3 }

node - days + - - - - -
[nodemon] restarting due to changes...
[nodemon] starting 'node main.js'
Server running on http://localhost:3000

The screenshot shows the VS Code interface with a REST client request to `http://localhost:3000/books/` using the POST method. The request body is a JSON object: `{ "title": "the new book", "author_id": 2, "genre_id": 1, "published_year": 2010, "price": "250.00" }`. The response is a plain text message: `Book added successfully`. The status is 201 Created, size is 23 Bytes, and time is 13 ms. The terminal shows the server running on `http://localhost:3000`.

```
POST http://localhost:3000/books/

JSON Content
1 {
2   "title": "the new book",
3   "author_id": 2,
4   "genre_id": 1,
5   "published_year": 2010,
6   "price": "250.00"
7 }
8 }
```

```
1 Book added successfully
```

Status: 201 Created Size: 23 Bytes Time: 13 ms

Response Headers Cookies Results Docs

1 Book added successfully

node - days + - - - - -
[nodemon] restarting due to changes...
[nodemon] starting 'node main.js'
Server running on http://localhost:3000

The screenshot shows the VS Code interface with a REST client request to `http://localhost:3000/genres/`. The request is a POST with a JSON body:

```
1 {
2   "name": "Philosophy",
3   "description": "based on indian epics and deities"
4 }
```

The response is a 201 Created status with a JSON body:

```
1 {
2   "id": 7,
3   "name": "Philosophy"
4 }
```

The terminal at the bottom shows the server running on `http://localhost:3000`.

The screenshot shows the VS Code interface with a REST client request to `http://localhost:3000/users/`. The request is a POST with a JSON body:

```
1 {
2   "name": "new_USerrrr",
3   "email": "newUser@example.com",
4   "age": 24,
5   "created_at": "2024-02-29"
6 }
```

The response is a 201 Created status with a JSON body:

```
1 {
2   "message": "user added successfully"
3 }
```

The terminal at the bottom shows the server running on `http://localhost:3000`.

Update and delete records by ID.

PUT `http://localhost:3000/users/1` **Send**

Status: **200 OK** Size: **44 Bytes** Time: **12 ms**

Response

```
1 {
2   "message": "User info updated successfully"
3 }
```

JSON Content

```
1 {
2   "name": "myuser",
3   "email": "myuser@example.com",
4   "age": 24,
5   "created_at": "2024-02-29"
6 }
```

PROBLEMS OUTPUT **TERMINAL** PORTS DEBUG CONSOLE

```
sqlState: '23000',
sqlMessage: "Duplicate entry 'newUser@example.com' for key 'users.email'"
}
```

PUT `http://localhost:3000/books/4` **Send**

Status: **200 OK** Size: **25 Bytes** Time: **20 ms**

Response

```
1 Book updated successfully
```

JSON Content

```
1 {
2   "title": "wise and otherwiseeee",
3   "author_id": 3,
4   "genre_id": 3,
5   "published_year": 1997,
6   "price": "1234.00"
7 }
```

PROBLEMS OUTPUT **TERMINAL** PORTS DEBUG CONSOLE

The screenshot shows the Visual Studio Code interface with a REST client request and response. The request is a PUT to `http://localhost:3000/users/1` with a JSON body. The response is a 200 OK status with a JSON body indicating successful user info update.

Request:

```
PUT http://localhost:3000/users/1
```

Request Body (JSON):

```
{  "name": "jafarrrr",  "email": "jafar@example.com",  "age": 24,  "created_at": "2024-02-28"}
```

Response:

```
{  "message": "User info updated successfully"}
```

The status bar at the bottom shows the current file is `node - day5` and the terminal is active.

The screenshot shows the Visual Studio Code interface with a REST client request and response. The request is a DELETE method to `http://localhost:3000/users/1`. The response is a 200 OK status with a JSON body containing a message: `"message": "User Deleted with id 1"`. The interface includes a top menu bar, a file explorer, a search bar, and a bottom status bar.

File Edit Selection View ... Nodejs 0.0 KBps 0.0 KBps

localhost:3000/authors X JS main.js M JS bookRoutes.js M JS userRoutes.js M JS genreRoutes.js M JS authorRoutes.js M

DELETE http://localhost:3000/users/1 Send

Query Headers 2 Auth Body Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

1

Status: 200 OK Size: 36 Bytes Time: 12 ms

Response Headers 6 Cookies Results Docs

1 {
2 "message": "User Deleted with id 1"
3 }

Response Chart

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

node - day5

master* 0 0 0 Connect Go Live