Hands-On Exercise: Create a CRUD API with Express.js

You need to build a RESTful API using **Express.js** for managing a list of books. The API will allow users to perform **CRUD** (Create, Read, Update, Delete) operations on a book collection using **JSON** data.

Requirements:

1. **Project Setup**
    - Initialize a new Node.js project.
    - Install the express package.
2. **Data Structure**
    - Create an array named books to store book objects. Each book should have the following properties:
        - id (integer)
        - title (string)
        - author (string)
        - year (integer)
        - genres (array of strings)
3. **API Endpoints**
    - Implement the following endpoints:

a. GET /api/books

- **Functionality**: Return the entire list of books.
- **Expected Response**:

```
[
 { "id": 1, "title": "The Alchemist", "author": "Paulo Coelho", "year": 1988, "genres": ["Adventure", "Fantasy"] },
 { "id": 2, "title": "The Great Gatsby", "author": "F. Scott Fitzgerald", "year": 1925, "genres": ["Classic", "Fiction"] }
]
```

b. GET /api/books/:id

- **Functionality**: Return a single book based on its ID.
- **Expected Response** (if found):

  { "id": 1, "title": "The Alchemist", "author": "Paulo Coelho", "year": 1988, "genres": ["Adventure", "Fantasy"] }

- **Error Response** (if not found):

  { "message": "Book not found" }

c. POST /api/books

- **Functionality**: Add a new book to the list.
- **Request Body Example**:

  { "title": "1984", "author": "George Orwell", "year": 1949, "genres": ["Dystopian", "Science Fiction"] }

- **Expected Response** (with new book ID):

  { "id": 3, "title": "1984", "author": "George Orwell", "year": 1949, "genres": ["Dystopian", "Science Fiction"] }

d. PUT /api/books/:id

- **Functionality**: Update an existing book's details based on its ID.
- **Request Body Example**:

  { "title": "1984", "author": "George Orwell", "year": 1949, "genres": ["Political Fiction"] }

- **Expected Response**:

  { "id": 3, "title": "1984", "author": "George Orwell", "year": 1949, "genres": ["Political Fiction"] }

- **Error Response** (if not found):

{ "message": "Book not found" }

e. DELETE /api/books/:id

- **Functionality**: Remove a book from the list based on its ID.
- **Expected Response**:

  { "message": "Book deleted" }

- **Error Response** (if not found):

  { "message": "Book not found" }

4. **Testing**

   - Test all API endpoints using **Postman** or **curl** to ensure they work as expected.

5. **Bonus Task**:

   - Add a search endpoint **GET /api/books/search?author=** that returns books by a specific author (case-insensitive).
   - Example Request: **/api/books/search?author=orwell**
   - Expected Response:

     [
     { "id": 3, "title": "1984", "author": "George Orwell", "year": 1949, "genres": ["Dystopian", "Science Fiction"] }
     ]

Deliverables:

- A working Express.js server with the described CRUD operations.
- JSON responses for all endpoints.
- Test results for each API endpoint (using Postman or curl).

Complete this hands-on exercise to solidify your understanding of creating and handling CRUD operations in a REST API using Express.js!