Task 1:

Book Struct:

```go
type Book struct {
    ID            int     `json:"id"`
    Title         string  `json:"title" binding:"required"`
    Author        string  `json:"author" binding:"required"`
    ISBN          string  `json:"isbn"`
    Price         float64 `json:"price" binding:"required,gt=0"`
    Stock         int     `json:"stock" binding:"gte=0"`
    PublishedYear int     `json:"published_year"`
    Description   string  `json:"description"`
    CreatedAt     string  `json:"created_at"`
}
```

```go
var db *sql.DB

func initDB() error {
    var err error
    db, err = sql.Open("sqlite3", "./bookstore.db")
    if err != nil {
        return err
    }

    createTableSQL := `
    CREATE TABLE IF NOT EXISTS books (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        title TEXT NOT NULL,
        author TEXT NOT NULL,
        isbn TEXT UNIQUE,
        price REAL NOT NULL CHECK(price > 0),
        stock INTEGER DEFAULT 0,
        published_year INTEGER,
        description TEXT,
        created_at DATETIME DEFAULT CURRENT_TIMESTAMP
    );`
    _, err = db.Exec(createTableSQL)
    return err
}
```

- Calls sql.Open("sqlite3", "./bookstore.db") to open the DB connection and assigns it to the package-level db variable.

- Builds a CREATE TABLE IF NOT EXISTS books (...) SQL string defining columns, constraints, and defaults.
- Executes the CREATE TABLE statement via db.Exec(createTableSQL) to create the table when absent.
- Returns any error encountered (either from opening the DB or executing the CREATE TABLE).

```go
func seedData() {
    var count int
    db.QueryRow("SELECT COUNT(*) FROM books").Scan(&count)
    if count > 0 {
        return
    }

    sampleBooks := []Book{
        {Title: "The Go Programming Language", Author: "Alan Donovan", ISBN: "978-0134190440", Price: 39.99, Stock: 15, PublishedYear:
        {Title: "Clean Code", Author: "Robert C. Martin", ISBN: "978-0132350884", Price: 29.99, Stock: 20, PublishedYear: 2008, Descri
        {Title: "Design Patterns", Author: "Erich Gamma", ISBN: "978-0201633610", Price: 49.99, Stock: 10, PublishedYear: 1994, Descri
        {Title: "The Pragmatic Programmer", Author: "Andrew Hunt", ISBN: "978-0201616224", Price: 35.99, Stock: 12, PublishedYear: 199
        {Title: "Code Complete", Author: "Steve McConnell", ISBN: "978-0735619678", Price: 38.99, Stock: 8, PublishedYear: 2004, Descr
    }

    for _, b := range sampleBooks {
        _, err := db.Exec(`INSERT INTO books
        (title, author, isbn, price, stock, published_year, description) VALUES (?, ?, ?, ?, ?, ?, ?)`,
            b.Title, b.Author, b.ISBN, b.Price, b.Stock, b.PublishedYear, b.Description)
        if err != nil {
            log.Println("Failed to seed book:", b.Title, err)
        }
    }
}
```

- Runs SELECT COUNT(*) FROM books and scans it into count.
- If count > 0, it returns early (no seeding).
- Otherwise, constructs a slice of Book values (sampleBooks) with classic programming book entries.
- Iterates the slice and inserts each book using db.Exec(...) with parameterized ? placeholders.
- Logs a message (via log.Println) if any individual insert fails (but continues seeding the rest).

```go
// GET /books
func getBooks(c *gin.Context) {
    rows, err := db.Query("SELECT id, title, author, isbn, price, stock, published_year, description, created_at FROM books")
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        return
    }
    defer rows.Close()

    books := []Book{}
    for rows.Next() {
        var b Book
        err := rows.Scan(&b.ID, &b.Title, &b.Author, &b.ISBN, &b.Price, &b.Stock, &b.PublishedYear, &b.Description, &b.CreatedAt)
        if err != nil {
            c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
            return
        }
        books = append(books, b)
    }

    c.JSON(http.StatusOK, gin.H{
        "books": books,
        "count": len(books),
    })
}
```

- Executes SELECT id, title, author, isbn, price, stock, published_year, description, created_at FROM books.
- Iterates rows.Next(), scanning each row into a Book struct and appending to books slice.
- If any DB error occurs during query or scan, responds with 500 Internal Server Error and the error message.
- If successful, responds with 200 OK and a JSON object

```go
// GET /books/:id
func getBook(c *gin.Context) {
    id := c.Param("id")
    var b Book
    err := db.QueryRow(`SELECT id, title, author, isbn, price, stock, published_year, description, created_at
    FROM books WHERE id = ?`, id).Scan(
        &b.ID, &b.Title, &b.Author, &b.ISBN, &b.Price, &b.Stock, &b.PublishedYear, &b.Description, &b.CreatedAt,
    )
    if err != nil {
        if err == sql.ErrNoRows {
            c.JSON(http.StatusNotFound, gin.H{"error": "Book not found"})
        } else {
            c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        }
        return
    }
    c.JSON(http.StatusOK, b)
}
```

- Reads id from URL param: c.Param("id").
- Executes db.QueryRow(... WHERE id = ?).Scan(...) to fetch that book.
- If Scan returns sql.ErrNoRows, responds 404 Not Found with {"error": "Book not found"}.
- If another DB error occurs, responds 500 Internal Server Error with the error text.

- If successful, responds 200 OK with the Book JSON.

```go
// POST /books
func createBook(c *gin.Context) {
    var b Book
    if err := c.ShouldBindJSON(&b); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid request data", "details": err.Error()})
        return
    }

    result, err := db.Exec(`INSERT INTO books
    (title, author, isbn, price, stock, published_year, description) VALUES (?, ?, ?, ?, ?, ?, ?)`,
        b.Title, b.Author, b.ISBN, b.Price, b.Stock, b.PublishedYear, b.Description)
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        return
    }

    id, err := result.LastInsertId()
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        return
    }
    b.ID = int(id)
    err = db.QueryRow(`SELECT created_at FROM books WHERE id = ?`, b.ID).Scan(&b.CreatedAt)
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        return
    }
    c.JSON(http.StatusCreated, b)
}
```

- Declares var b Book.
- Uses c.ShouldBindJSON(&b) — Gin binds the incoming JSON into b and enforces binding rules (e.g., title/author required, price > 0).
- If binding fails, responds 400 Bad Request with {"error":"Invalid request data","details":...}.
- Runs INSERT INTO books (...) VALUES (?, ?, ?, ?, ?, ?, ?) with parameters from b.
- If insert fails (e.g., unique ISBN constraint), responds 500 Internal Server Error with the error.
- Retrieves the last insert id via result.LastInsertId() and assigns it to b.ID.
- Sets err = db.QueryRow(` SELECT created_at FROM books WHERE id = ?` , b.ID).Scan(&b.CreatedAt) to reflect creation.
- Responds 201 Created with the created b JSON.

```go
// PUT /books/:id
func updateBook(c *gin.Context) {
    id := c.Param("id")
    var b Book
    if err := c.ShouldBindJSON(&b); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid request data", "details": err.Error()})
        return
    }

    res, err := db.Exec(`UPDATE books SET title=?, author=?, isbn=?, price=?, stock=?, published_year=?, description=? WHERE id=?`,
        b.Title, b.Author, b.ISBN, b.Price, b.Stock, b.PublishedYear, b.Description, id)
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        return
    }

    rowsAffected, _ := res.RowsAffected()
    if rowsAffected == 0 {
        c.JSON(http.StatusNotFound, gin.H{"error": "Book not found"})
        return
    }

    b.ID = atoi(id)
    c.JSON(http.StatusOK, b)
}
```

- Reads id from c.Param("id").
- Binds incoming JSON to var b Book using ShouldBindJSON. On bind error, responds 400 Bad Request.
- Executes an UPDATE books SET … WHERE id = ? with values from b.
- Checks res.RowsAffected(); if 0 then no row matched → responds 404 Not Found.
- Sets b.ID = atoi(id) to reflect the updated record's id (so the response object includes the id).
- Responds 200 OK with the b JSON.

```go
// DELETE /books/:id
func deleteBook(c *gin.Context) {
    id := c.Param("id")
    res, err := db.Exec("DELETE FROM books WHERE id=?", id)
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        return
    }
    rowsAffected, _ := res.RowsAffected()
    if rowsAffected == 0 {
        c.JSON(http.StatusNotFound, gin.H{"error": "Book not found"})
        return
    }
    c.JSON(http.StatusOK, gin.H{"message": "Book deleted successfully"})
}
```

- Reads id from c.Param("id").
- Executes DELETE FROM books WHERE id = ?.

- Calls res.RowsAffected() to see how many rows were deleted.
- If 0, returns 404 Not Found ({"error":"Book not found"}).
- Otherwise returns 200 OK and {"message":"Book deleted successfully"}.

```go
// helper
func atoi(s string) int {
    var i int
    fmt.Sscan(s, &i)
    return i
}
```

Declares var i int and uses fmt.Sscan(s, &i) to parse the string into i. Then, returns i. If parsing fails, i will be 0 (no explicit error handling).

```go
func main() {
    if err := initDB(); err != nil {
        log.Fatal("Failed to initialize database:", err)
    }
    defer db.Close()

    seedData()

    router := gin.Default()
    router.GET("/books", getBooks)
    router.GET("/books/:id", getBook)
    router.POST("/books", createBook)
    router.PUT("/books/:id", updateBook)
    router.DELETE("/books/:id", deleteBook)

    fmt.Println("🚀 Server starting on :8080")
    router.Run(":8080")
}
```

1. Calls seedData() — inserts sample books if table empty.
2. Registers routes and handler functions:
    - GET /books → getBooks
    - GET /books/:id → getBook
    - POST /books → createBook
    - PUT /books/:id → updateBook

- DELETE /books/:id → deleteBook

3. Starts the HTTP server on port 8080 with router.Run(":8080").

Output:

```
PS C:\Users\VUTHANHHUNG\Desktop\Netcen Pro\VuThanhNhan - ITITIU21267 - Lab5\Task1> go run .\main.go
[GIN-debug] [WARNING] Creating an Engine instance with the Logger and Recovery middleware already attached.

[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
 - using env:   export GIN_MODE=release
 - using code:  gin.SetMode(gin.ReleaseMode)

[GIN-debug] GET    /books                    --> main.getBooks (3 handlers)
[GIN-debug] GET    /books/:id                --> main.getBook (3 handlers)
[GIN-debug] POST   /books                    --> main.createBook (3 handlers)
[GIN-debug] PUT    /books/:id                --> main.updateBook (3 handlers)
[GIN-debug] DELETE /books/:id                --> main.deleteBook (3 handlers)
🚀 Server starting on :8080
```

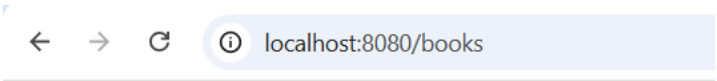
Rows: 5

	id	title	author	isbn	price	stock	publishe...	description
1	1	The Go Programming Language	Alan Donovan	978-0134190440	39.99	15	2015	Complete guide to Go progr
2	2	Clean Code	Robert C. Martin	978-0132350884	29.99	20	2008	A Handbook of Agile Softwa
3	3	Design Patterns	Erich Gamma	978-0201633610	49.99	10	1994	Elements of Reusable Object
4	4	The Pragmatic Programmer	Andrew Hunt	978-0201616224	35.99	12	1999	From Journeyman to Master
5	5	Code Complete	Steve McConnell	978-0735619678	38.99	8	2004	A Practical Handbook of Soft
6								

Get all books

curl http://localhost:8080/books

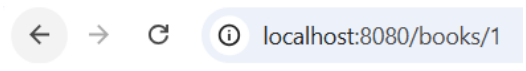
```
PS C:\Users\VUTHANHHUNG\Desktop\Netcen Pro\VuThanhNhan - ITITIU21267 - Lab5\Task1> curl http://localhost:8080/books


StatusCode        : 200
StatusDescription : OK
Content           : {"books":[{"id":1,"title":"The Go Programming Language","author":"Alan
                    Donovan","isbn":"978-0134190440","price":39.99,"stock":15,"published_year":2015,"description":"Complete guide to Go
                    programming",...
RawContent        : HTTP/1.1 200 OK
                    Content-Length: 1143
                    Content-Type: application/json; charset=utf-8
                    Date: Thu, 13 Nov 2025 15:05:09 GMT

                    {"books":[{"id":1,"title":"The Go Programming Language","author":"Alan Dono...
Forms             : {}
Headers           : {[Content-Length, 1143], [Content-Type, application/json; charset=utf-8], [Date, Thu, 13 Nov 2025 15:05:09 GMT]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 1143
```



Tạo bản in đẹp


```
{
  "books": [
    {
      "id": 1,
      "title": "The Go Programming Language",
      "author": "Alan Donovan",
      "isbn": "978-0134190440",
      "price": 39.99,
      "stock": 15,
      "published_year": 2015,
      "description": "Complete guide to Go programming",
      "created_at": "2025-11-13T15:03:20Z"
    },
    {
      "id": 2,
      "title": "Clean Code",
      "author": "Robert C. Martin",
      "isbn": "978-0132350884",
      "price": 29.99,
      "stock": 20,
      "published_year": 2008,
      "description": "A Handbook of Agile Software Craftsmanship",
      "created_at": "2025-11-13T15:03:20Z"
    },
    {
      "id": 3,
      "title": "Design Patterns",
      "author": "Erich Gamma",
      "isbn": "978-0201633610",
      "price": 49.99,
      "stock": 10,
      "published_year": 1994,
      "description": "Elements of Reusable Object-Oriented Software",
      "created_at": "2025-11-13T15:03:20Z"
    },
    {
      "id": 4,
      "title": "The Pragmatic Programmer",
      "author": "Andrew Hunt",
      "isbn": "978-0201616224",
      "price": 35.99,
      "stock": 12,
      "published_year": 1999,
      "description": "From Journeyman to Master",
      "created_at": "2025-11-13T15:03:20Z"
    },
    {
      "id": 5,
      "title": "Code Complete",
      "author": "Steve McConnell",
      "isbn": "978-0735619678",
      "price": 38.99,
      "stock": 8,
      "published_year": 2004,
      "description": "A Practical Handbook of Software Construction",
      "created_at": "2025-11-13T15:03:20Z"
    }
  ],
```


Get book by ID curl http://localhost:8080/books/1



Tạo bản in đẹp

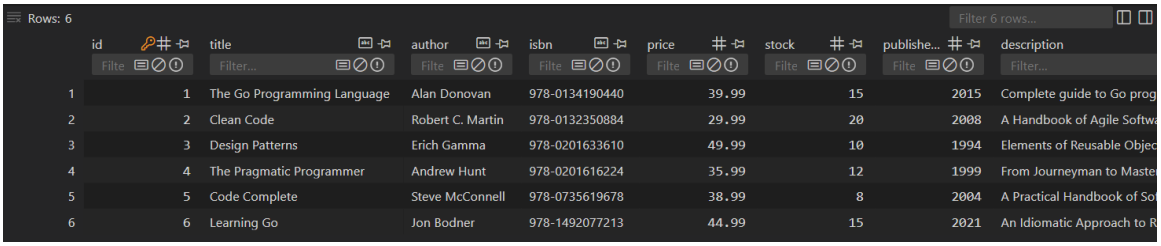

```
{
  "id": 1,
  "title": "The Go Programming Language",
  "author": "Alan Donovan",
  "isbn": "978-0134190440",
  "price": 39.99,
  "stock": 15,
  "published_year": 2015,
  "description": "Complete guide to Go programming",
  "created_at": "2025-11-13T15:03:20Z"
}
```

```
PS C:\Users\VUTHANHHUNG\Desktop\Netcen Pro\VuThanhNhan - ITITIU21267 - Lab5\Task1> curl http://localhost:8080/books/1


StatusCode        : 200
StatusDescription : OK
Content           : {"id":1,"title":"The Go Programming Language","author":"Alan
                    Donovan","isbn":"978-0134190440","price":39.99,"stock":15,"published_year":2015,"description":"Complete guide to Go
                    programming","created_a...
RawContent        : HTTP/1.1 200 OK
                    Content-Length: 226
                    Content-Type: application/json; charset=utf-8
                    Date: Thu, 13 Nov 2025 15:05:37 GMT

                    {"id":1,"title":"The Go Programming Language","author":"Alan Donovan","isbn"...
Forms             : {}
Headers           : {[Content-Length, 226], [Content-Type, application/json; charset=utf-8], [Date, Thu, 13 Nov 2025 15:05:37 GMT]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 226
```

Create new book

curl -X POST http://localhost:8080/books \ -H "Content-Type: application/json" \ -d '{ "title": "Learning Go", "author": "Jon Bodner", "isbn": "978-1492077213", "price": 44.99, "stock": 15, "published_year": 2021, "description": "An Idiomatic Approach to Real-World Go Programming" }'

```
VUTHANHHUNG@LAPTOP-N7HIB7T0 MINGW64 ~/Desktop/Netcen Pro/VuThanhNhan - ITITIU21267 - Lab5
$ curl -X POST http://localhost:8080/books \  -H "Content-Type: application/json" \  -d '{    "title": "Learning Go",    "author": "Jon Bodner",
"isbn": "978-1492077213",    "price": 44.99,    "stock": 15,    "published_year": 2021,    "description": "An Idiomatic Approach to Real-World Go Pr
ogramming"  }'
{"id":6,"title":"Learning Go","author":"Jon Bodner","isbn":"978-1492077213","price":44.99,"stock":15,"published_year":2021,"description":"An Idiomat
ic Approach to Real-World Go Programming","created_at":"2025-11-13 22:14:18"}curl: (3) URL rejected: Malformed input to a URL function
curl: (3) URL rejected: Malformed input to a URL function
```



Rows: 6

Filter 6 rows...

	id	title	author	isbn	price	stock	publishe...	description
1	1	The Go Programming Language	Alan Donovan	978-0134190440	39.99	15	2015	Complete guide to Go prog
2	2	Clean Code	Robert C. Martin	978-0132350884	29.99	20	2008	A Handbook of Agile Softwa
3	3	Design Patterns	Erich Gamma	978-0201633610	49.99	10	1994	Elements of Reusable Objec
4	4	The Pragmatic Programmer	Andrew Hunt	978-0201616224	35.99	12	1999	From Journeyman to Maste
5	5	Code Complete	Steve McConnell	978-0735619678	38.99	8	2004	A Practical Handbook of So
6	6	Learning Go	Jon Bodner	978-1492077213	44.99	15	2021	An Idiomatic Approach to R

Update book

curl -X PUT http://localhost:8080/books/1 \ -H "Content-Type: application/json" \ -d '{ "title": "The Go Programming Language (2nd Edition)", "author": "Alan Donovan", "isbn": "978-0134190440", "price": 39.99, "stock": 20, "published_year": 2015, "description": "Complete guide to Go programming" }'

```
VUTHANHHUNG@LAPTOP-N7HIB7T0 MINGW64 ~/Desktop/Netcen Pro/VuThanhNhan - ITITIU21267 - Lab5
$ curl -X PUT http://localhost:8080/books/1 \
  -H "Content-Type: application/json" \
  -d '{
    "title": "The Go Programming Language (2nd Edition)",
    "author": "Alan Donovan",
    "isbn": "978-0134190440",
    "price": 39.99,
    "stock": 20,
    "published_year": 2015,
    "description": "Complete guide to Go programming"
  }'
{"id":1,"title":"The Go Programming Language (2nd Edition)","author":"Alan Donovan","isbn":"978-0134190440","price":39.99,"stock":20,"published_year
":2015,"description":"Complete guide to Go programming","created_at":""}
```

Delete book

curl -X DELETE http://localhost:8080/books/1

Rows: 5

	id	title	author	isbn	price	stock
1	2	Clean Code	Robert C. Martin	978-0132350884	29.99	20
2	3	Design Patterns	Erich Gamma	978-0201633610	49.99	10
3	4	The Pragmatic Programmer	Andrew Hunt	978-0201616224	35.99	12
4	5	Code Complete	Steve McConnell	978-0735619678	38.99	8
5	6	Learning Go	Jon Bodner	978-1492077213	44.99	15
6						

```
VUTHANHHUNG@LAPTOP-N7HIB7T0 MINGW64 ~/Desktop/Netcen Pro/VuThanhNhan - ITITIU21267 - Lab5
$ curl -X DELETE http://localhost:8080/books/1
{"message":"Book deleted successfully"}
```

Task 2:

Only the following functions were modified from Task 1:

1. getBooks(c *gin.Context)
2. searchBooks(c *gin.Context)
3. filterBooks(c *gin.Context)

All CRUD functions (getBook, createBook, updateBook, deleteBook) remain identical to Task 1.

Database initialization, seeding, and server setup (main()) are unchanged.

```go
// GET /books with optional filters and sorting
func getBooks(c *gin.Context) {
    sortBy := c.DefaultQuery("sort", "id")
    validSorts := map[string]string{
        "price_asc":  "price ASC",
        "price_desc": "price DESC",
        "title":      "title ASC",
        "year_desc":  "published_year DESC",
        "id":         "id ASC",
    }

    orderBy, ok := validSorts[sortBy]
    if !ok {
        orderBy = "id ASC"
    }

    query := "SELECT id, title, author, isbn, price, stock, published_year, description, created_at FROM books WHERE 1=1"
    var args []interface{}

    if minPrice := c.Query("min_price"); minPrice != "" {
        query += " AND price >= ?"
        args = append(args, minPrice)
    }
    if maxPrice := c.Query("max_price"); maxPrice != "" {
        query += " AND price <= ?"
        args = append(args, maxPrice)
    }
    if author := c.Query("author"); author != "" {
        query += " AND LOWER(author) LIKE LOWER(?)"
        args = append(args, "%"+author+"%")
    }
    if year := c.Query("year"); year != "" {
        query += " AND published_year = ?"
        args = append(args, year)
    }

    query += " ORDER BY " + orderBy

    rows, err := db.Query(query, args...)
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        return
    }
    defer rows.Close()

    var books []Book
    for rows.Next() {
        var b Book
        if err := rows.Scan(&b.ID, &b.Title, &b.Author, &b.ISBN, &b.Price,
            &b.Stock, &b.PublishedYear, &b.Description, &b.CreatedAt); err != nil {
            c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
            return
        }
        books = append(books, b)
    }

    c.JSON(http.StatusOK, gin.H{"books": books, "count": len(books)})
}
```

1. Reads sort query param; defaults to "id".
2. Maps allowed sort values to SQL ORDER BY clauses.
3. Builds a base SQL query: SELECT ... FROM books WHERE 1=1.
4. Checks optional query parameters and adds them to SQL:
    - min_price → AND price >= ?
    - max_price → AND price <= ?
    - author → AND LOWER(author) LIKE LOWER(?)

- year → AND published_year = ?

5. Appends ORDER BY clause.
6. Executes query and scans rows into Book structs.
7. Returns JSON with books and count.

```go
// GET /books/search?q=query
func searchBooks(c *gin.Context) {
    query := c.Query("q")
    if query == "" {
        c.JSON(http.StatusBadRequest, gin.H{"error": "Search query required"})
        return
    }

    searchPattern := "%" + query + "%"
    rows, err := db.Query(`
        SELECT id, title, author, isbn, price, stock, published_year, description, created_at
        FROM books
        WHERE LOWER(title) LIKE LOWER(?) OR LOWER(author) LIKE LOWER(?)`,
        searchPattern, searchPattern)
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        return
    }
    defer rows.Close()

    var books []Book
    for rows.Next() {
        var b Book
        if err := rows.Scan(&b.ID, &b.Title, &b.Author,
            &b.ISBN, &b.Price, &b.Stock, &b.PublishedYear, &b.Description, &b.CreatedAt); err != nil {
            c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
            return
        }
        books = append(books, b)
    }

    c.JSON(http.StatusOK, gin.H{
        "books": books,
        "count": len(books),
        "query": query,
    })
}
```

- Reads q from query parameter.
- Returns 400 if q is empty.
- Prepares search pattern: %query%.
- Executes SQL: WHERE LOWER(title) LIKE LOWER(?) OR LOWER(author) LIKE LOWER(?).
- Iterates over rows and scans them into Book structs.
- Returns JSON with books, count, and the original query.

```go
// GET /books/filter?min_price=X&max_price=Y&author=Z&year=YYYY
func filterBooks(c *gin.Context) {
    query := "SELECT id, title, author, isbn, price, stock, published_year, description, created_at FROM books WHERE 1=1"
    var args []interface{}

    if minPrice := c.Query("min_price"); minPrice != "" {
        query += " AND price >= ?"
        args = append(args, minPrice)
    }
    if maxPrice := c.Query("max_price"); maxPrice != "" {
        query += " AND price <= ?"
        args = append(args, maxPrice)
    }
    if author := c.Query("author"); author != "" {
        query += " AND LOWER(author) LIKE LOWER(?)"
        args = append(args, "%"+author+"%")
    }
    if year := c.Query("year"); year != "" {
        query += " AND published_year = ?"
        args = append(args, year)
    }

    rows, err := db.Query(query, args...)
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        return
    }
    defer rows.Close()
```

```go
    var books []Book
    for rows.Next() {
        var b Book
        if err := rows.Scan(&b.ID, &b.Title, &b.Author, &b.ISBN, &b.Price, &b.Stock,
            &b.PublishedYear, &b.Description, &b.CreatedAt); err != nil {
            c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
            return
        }
        books = append(books, b)
    }

    c.JSON(http.StatusOK, gin.H{
        "books":   books,
        "count":   len(books),
        "filters": c.Request.URL.Query(),
    })
}

// Helper: string to int
func atoi(s string) int {
    var i int
    fmt.Sscanf(s, "%d", &i)
    return i
}
```

1. Builds base query: SELECT … FROM books WHERE 1=1.
2. Reads query params and appends SQL conditions if present:
    - min_price → AND price >= ?
    - max_price → AND price <= ?
    - author → AND LOWER(author) LIKE LOWER(?)
    - year → AND published_year = ?
3. Executes query with the collected arguments.
4. Scans results into Book structs.

5. Returns JSON with books, count, and filters applied.

```go
func main() {
    if err := initDB(); err != nil {
        log.Fatal("Failed to initialize database:", err)
    }
    defer db.Close()

    seedData()

    router := gin.Default()

    // CRUD routes
    router.GET("/books", getBooks)
    router.GET("/books/:id", getBook)
    router.POST("/books", createBook)
    router.PUT("/books/:id", updateBook)
    router.DELETE("/books/:id", deleteBook)

    // Advanced queries
    router.GET("/books/search", searchBooks)
    router.GET("/books/filter", filterBooks)

    fmt.Println("🚀 Server running on :8080")
    router.Run(":8080")
}
```

1. Calls seedData() — inserts sample books if table is empty.
2. Creates a Gin router with router := gin.Default().
3. Registers CRUD routes:
    - GET /books → getBooks
    - GET /books/:id → getBook
    - POST /books → createBook
    - PUT /books/:id → updateBook
    - DELETE /books/:id → deleteBook
4. Registers Advanced Query routes:
    - GET /books/search → searchBooks
    - GET /books/filter → filterBooks

5. Starts the HTTP server on port 8080 via router.Run(":8080").

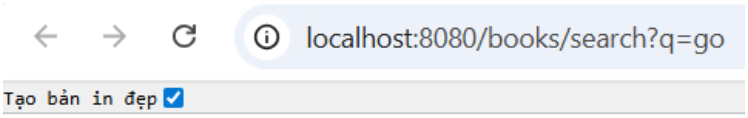
Output:

```
PS C:\Users\VUTHANHHUNG\Desktop\Netcen Pro\VuThanhNhan - ITITIU21267 - Lab5\Task2> go run .\main.go
[GIN-debug] [WARNING] Creating an Engine instance with the Logger and Recovery middleware already attached.

[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
 - using env:   export GIN_MODE=release
 - using code:  gin.SetMode(gin.ReleaseMode)

[GIN-debug] GET    /books                    --> main.getBooks (3 handlers)
[GIN-debug] GET    /books/:id                --> main.getBook (3 handlers)
[GIN-debug] POST   /books                    --> main.createBook (3 handlers)
[GIN-debug] PUT    /books/:id                --> main.updateBook (3 handlers)
[GIN-debug] DELETE /books/:id                --> main.deleteBook (3 handlers)
[GIN-debug] GET    /books/search             --> main.searchBooks (3 handlers)
[GIN-debug] GET    /books/filter             --> main.filterBooks (3 handlers)
🚀 Server running on :8080
[GIN-debug] [WARNING] You trusted all proxies, this is NOT safe. We recommend you to set a value.
Please check https://github.com/gin-gonic/gin/blob/master/docs/doc.md#dont-trust-all-proxies for details.
[GIN-debug] Listening and serving HTTP on :8080
```

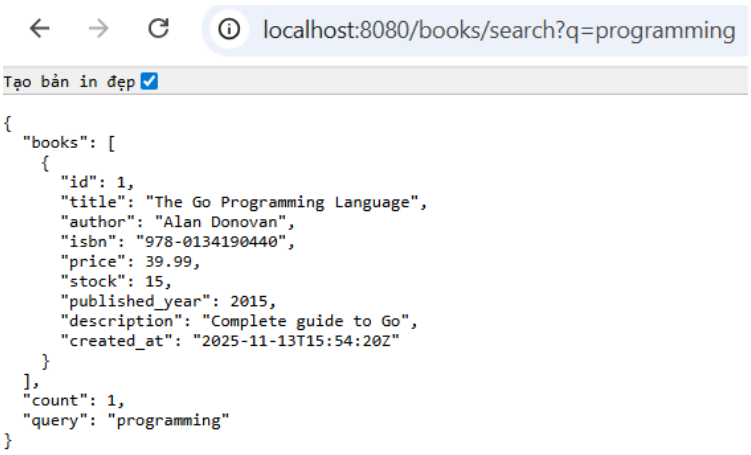
Search books

curl "http://localhost:8080/books/search?q=go"

curl "http://localhost:8080/books/search?q=programming"



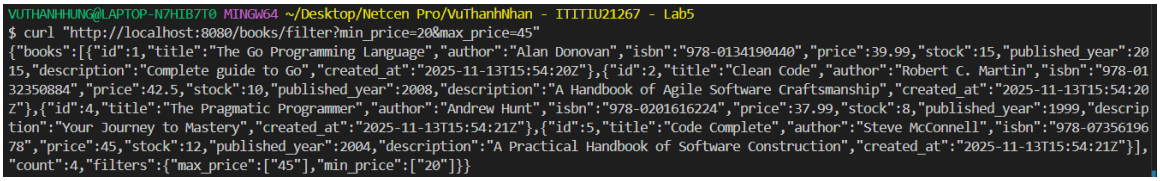

```
{
  "books": [
    {
      "id": 1,
      "title": "The Go Programming Language",
      "author": "Alan Donovan",
      "isbn": "978-0134190440",
      "price": 39.99,
      "stock": 15,
      "published_year": 2015,
      "description": "Complete guide to Go",
      "created_at": "2025-11-13T15:54:20Z"
    }
  ],
  "count": 1,
  "query": "go"
}
```

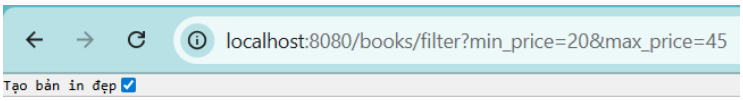



```
{
  "books": [
    {
      "id": 1,
      "title": "The Go Programming Language",
      "author": "Alan Donovan",
      "isbn": "978-0134190440",
      "price": 39.99,
      "stock": 15,
      "published_year": 2015,
      "description": "Complete guide to Go",
      "created_at": "2025-11-13T15:54:20Z"
    }
  ],
  "count": 1,
  "query": "programming"
}
```


Filter by price range

curl "http://localhost:8080/books/filter?min_price=20&max_price=45"

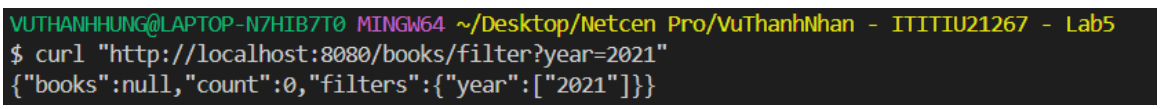


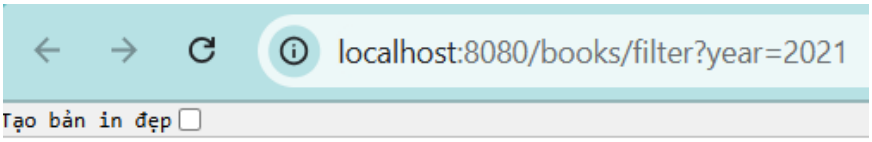



```
{
  "books": [
    {
      "id": 1,
      "title": "The Go Programming Language",
      "author": "Alan Donovan",
      "isbn": "978-0134190440",
      "price": 39.99,
      "stock": 15,
      "published_year": 2015,
      "description": "Complete guide to Go",
      "created_at": "2025-11-13T15:54:20Z"
    },
    {
      "id": 2,
      "title": "Clean Code",
      "author": "Robert C. Martin",
      "isbn": "978-0132350884",
      "price": 42.5,
      "stock": 10,
      "published_year": 2008,
      "description": "A Handbook of Agile Software Craftsmanship",
      "created_at": "2025-11-13T15:54:20Z"
    },
```


Filter by year

curl "http://localhost:8080/books/filter?year=2021"

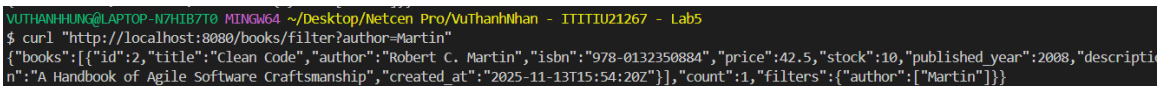


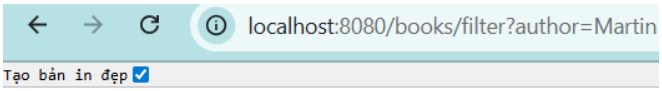


{"books":null,"count":0,"filters":{"year":["2021"]}}

Filter by author

curl "http://localhost:8080/books/filter?author=Martin"

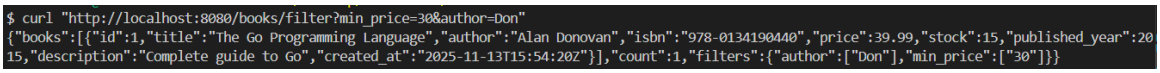


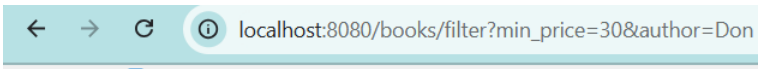


{
  "books": [
    {
      "id": 2,
      "title": "Clean Code",
      "author": "Robert C. Martin",
      "isbn": "978-0132350884",
      "price": 42.5,
      "stock": 10,
      "published_year": 2008,
      "description": "A Handbook of Agile Software Craftsmanship",
      "created_at": "2025-11-13T15:54:20Z"
    }
  ],
  "count": 1,
  "filters": {
    "author": [
      "Martin"
    ]
  }
}

Combine filters

curl "http://localhost:8080/books/filter?min_price=30&author=Don"






```
{
  "books": [
    {
      "id": 1,
      "title": "The Go Programming Language",
      "author": "Alan Donovan",
      "isbn": "978-0134190440",
      "price": 39.99,
      "stock": 15,
      "published_year": 2015,
      "description": "Complete guide to Go",
      "created_at": "2025-11-13T15:54:20Z"
    }
  ],
  "count": 1,
  "filters": {
    "author": [
      "Don"
    ],
    "min_price": [
      "30"
    ]
  }
}
```


Sort books

curl "http://localhost:8080/books?sort=price_asc"

curl "http://localhost:8080/books?sort=title"

curl "http://localhost:8080/books?sort=year_desc"