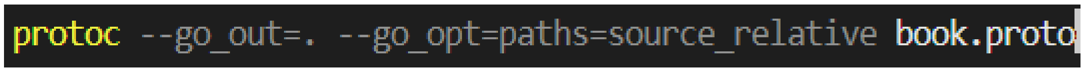
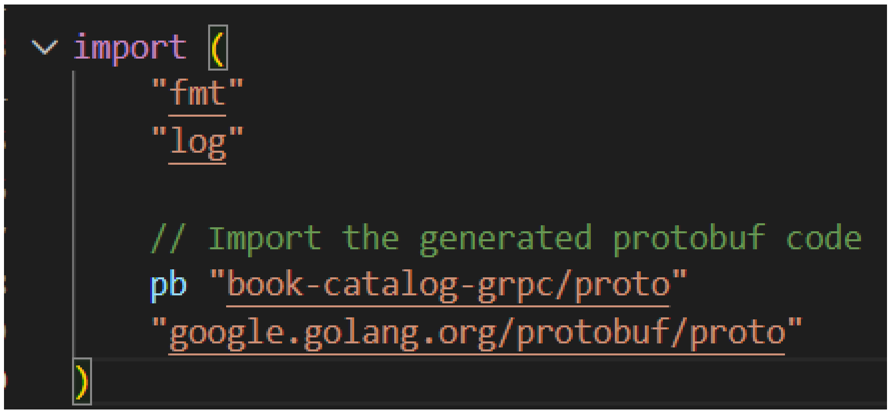
Task 1:



```
protoc --go_out=. --go_opt=paths=source_relative book.proto
```

- protoc: Protocol Buffers compiler.
- --go_out=.: Generates Go structs/messages for the proto definitions in the current directory.



```
import (
    "fmt"
    "log"

    // Import the generated protobuf code
    pb "book-catalog-grpc/proto"
    "google.golang.org/protobuf/proto"
)
```

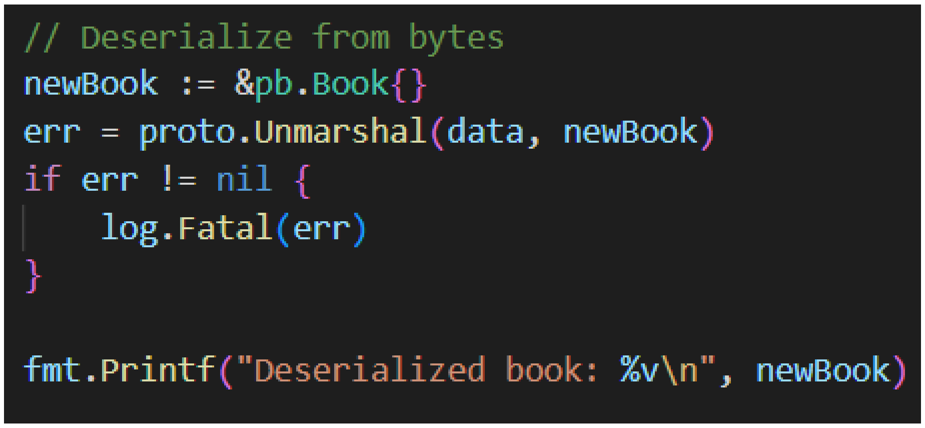
- pb "book-catalog-grpc/proto": Imports generated protobuf code with alias pb (protocol buffer)

```go
func main() {
    // Create a Book instance
    book := &pb.Book{
        Id:            1,
        Title:         "The Go Programming Language",
        Author:        "Alan Donovan",
        Isbn:          "978-0134190440",
        Price:         39.99,
        Stock:         15,
        PublishedYear: 2015,
    }

    fmt.Printf("Book: %v\n", book)

    // Create DetailedBook with category and tags
    detailedBook := &pb.DetailedBook{
        Book:        book,
        Category:    pb.BookCategory_NONFICTION,
        Description: "A comprehensive introduction to Go programming.",
        Tags:        []string{"programming", "go", "technical"},
        Rating:      4.5,
    }

    fmt.Printf("\nDetailed Book: %v\n", detailedBook)
    fmt.Printf("Category: %s\n", detailedBook.Category)
    fmt.Printf("Tags: %v\n", detailedBook.Tags)
```

- Initializes a Book struct with fields like Id, Title, Author, ISBN, Price, Stock, and PublishedYear, then prints the struct using %v.
- Composes a DetailedBook containing a nested Book.
- Adds Category (enum), Description, Tags (string slice), and Rating.
- Prints the full struct, category, and tags.

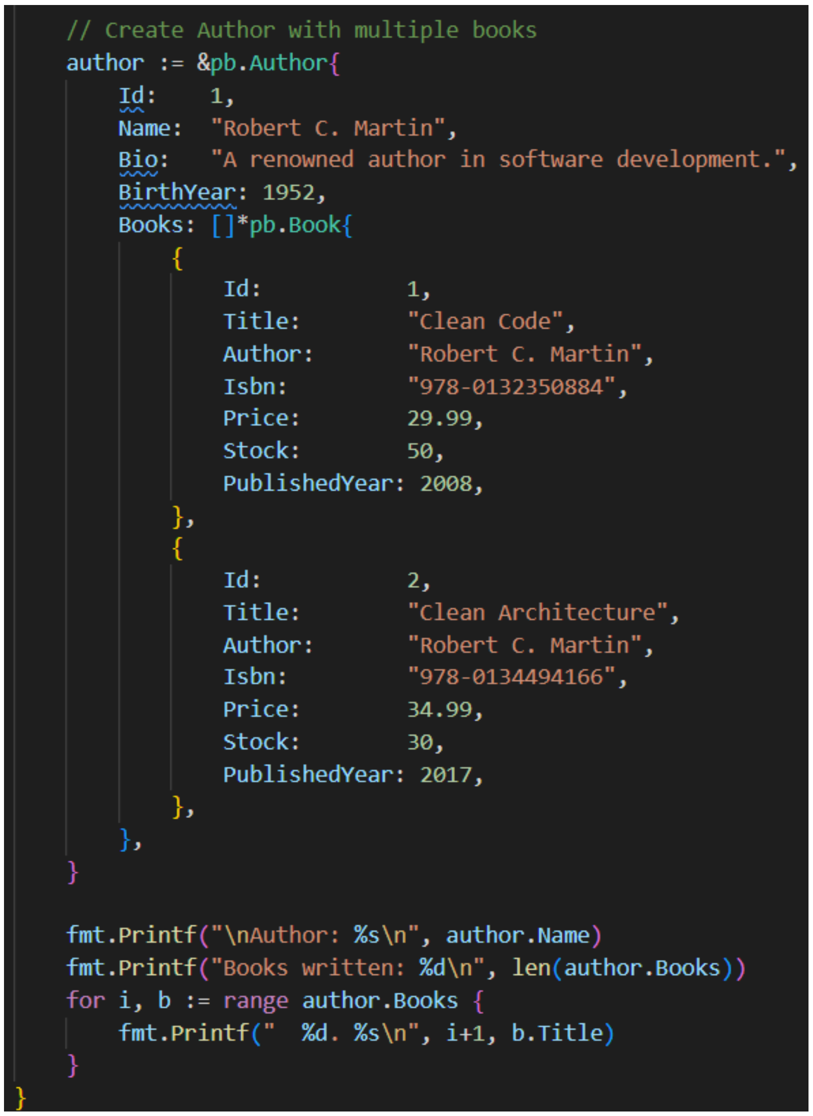
```go
// Serialize to bytes
data, err := proto.Marshal(book)
if err != nil {
    log.Fatal(err)
}

fmt.Printf("\nSerialized size: %d bytes\n", len(data))
```

- proto.Marshal converts Book to binary bytes; prints serialized size.



```go
// Deserialize from bytes
newBook := &pb.Book{}
err = proto.Unmarshal(data, newBook)
if err != nil {
    log.Fatal(err)
}

fmt.Printf("Deserialized book: %v\n", newBook)
```

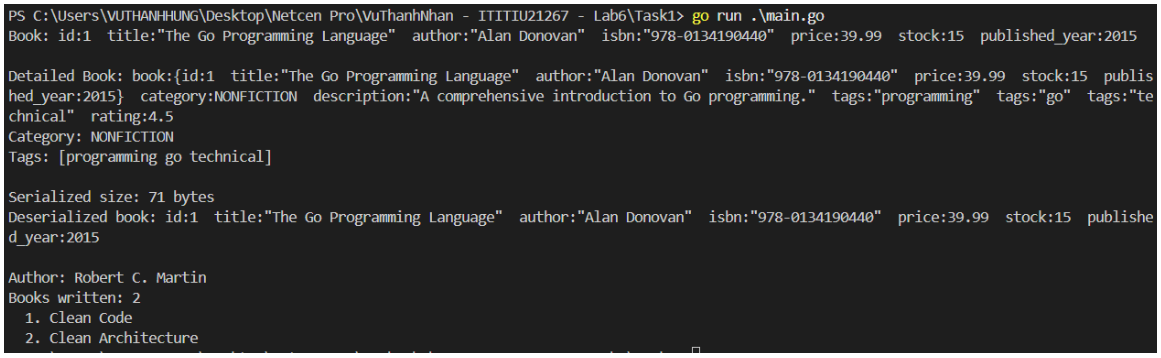
- proto.Unmarshal reconstructs a new Book from bytes; prints deserialized struct.



```go
    // Create Author with multiple books
    author := &pb.Author{
        Id:     1,
        Name:   "Robert C. Martin",
        Bio:    "A renowned author in software development.",
        BirthYear: 1952,
        Books: []*pb.Book{
            {
                Id:            1,
                Title:         "Clean Code",
                Author:        "Robert C. Martin",
                Isbn:          "978-0132350884",
                Price:         29.99,
                Stock:         50,
                PublishedYear: 2008,
            },
            {
                Id:            2,
                Title:         "Clean Architecture",
                Author:        "Robert C. Martin",
                Isbn:          "978-0134494166",
                Price:         34.99,
                Stock:         30,
                PublishedYear: 2017,
            },
        },
    }

    fmt.Printf("\nAuthor: %s\n", author.Name)
    fmt.Printf("Books written: %d\n", len(author.Books))
    for i, b := range author.Books {
        fmt.Printf("  %d. %s\n", i+1, b.Title)
    }
}
```

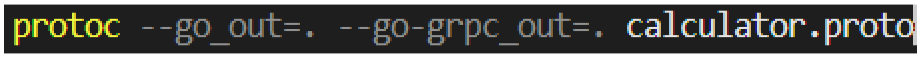
- Initializes an Author struct with Id, Name, Bio, BirthYear.
- Assigns a slice of Book pointers to demonstrate a one-to-many relationship.
- Prints author name, number of books, and a numbered list of book titles.

Output:



```
PS C:\Users\VUTHANHHUNG\Desktop\Netcen Pro\VuThanhNhan - ITITIU21267 - Lab6\Task1> go run .\main.go
Book: id:1  title:"The Go Programming Language"  author:"Alan Donovan"  isbn:"978-0134190440"  price:39.99  stock:15  published_year:2015

Detailed Book: book:{id:1  title:"The Go Programming Language"  author:"Alan Donovan"  isbn:"978-0134190440"  price:39.99  stock:15  publis
hed_year:2015}  category:NONFICTION  description:"A comprehensive introduction to Go programming."  tags:"programming"  tags:"go"  tags:"te
chnical"  rating:4.5
Category: NONFICTION
Tags: [programming go technical]

Serialized size: 71 bytes
Deserialized book: id:1  title:"The Go Programming Language"  author:"Alan Donovan"  isbn:"978-0134190440"  price:39.99  stock:15  publishe
d_year:2015

Author: Robert C. Martin
Books written: 2
  1. Clean Code
  2. Clean Architecture
```

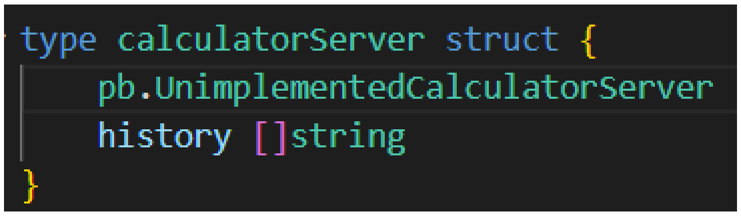
Task 2:



```
protoc --go_out=. --go-grpc_out=. calculator.proto
```

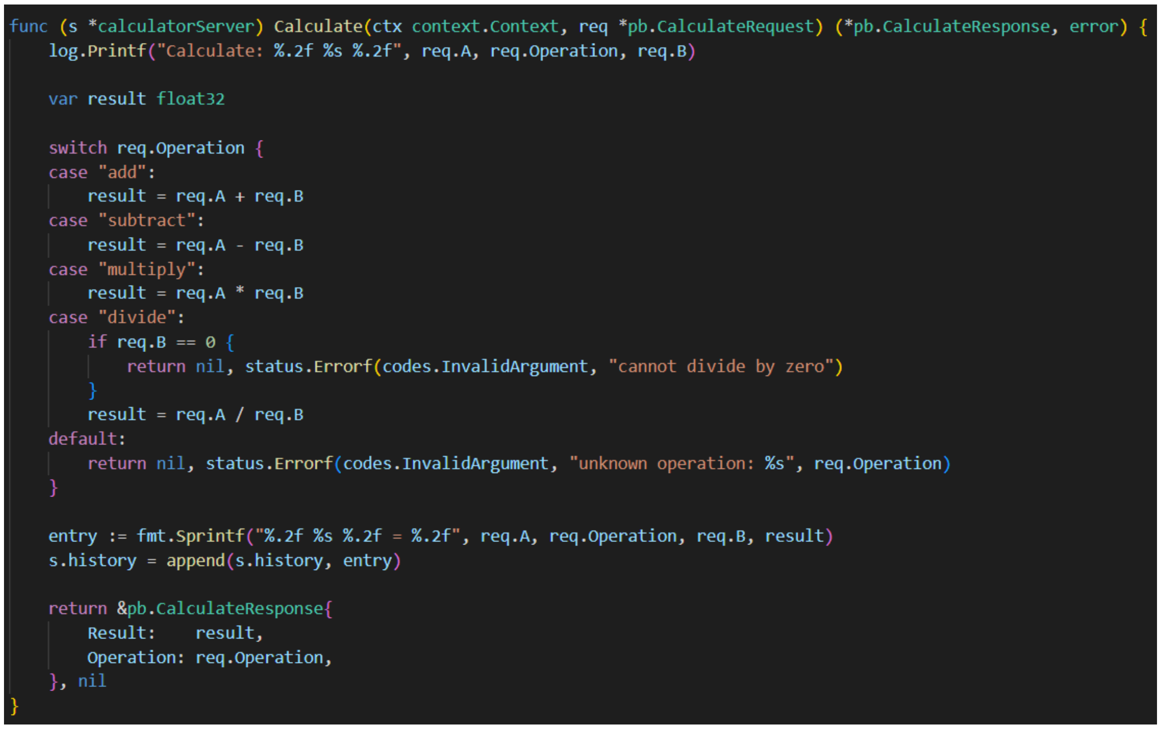
- protoc: Protocol Buffers compiler.
- --go_out=.: Generates Go structs/messages for the proto definitions in the current directory.
- --go-grpc_out=.: Generates Go gRPC service code (interfaces and stubs) in the current directory.

Server:



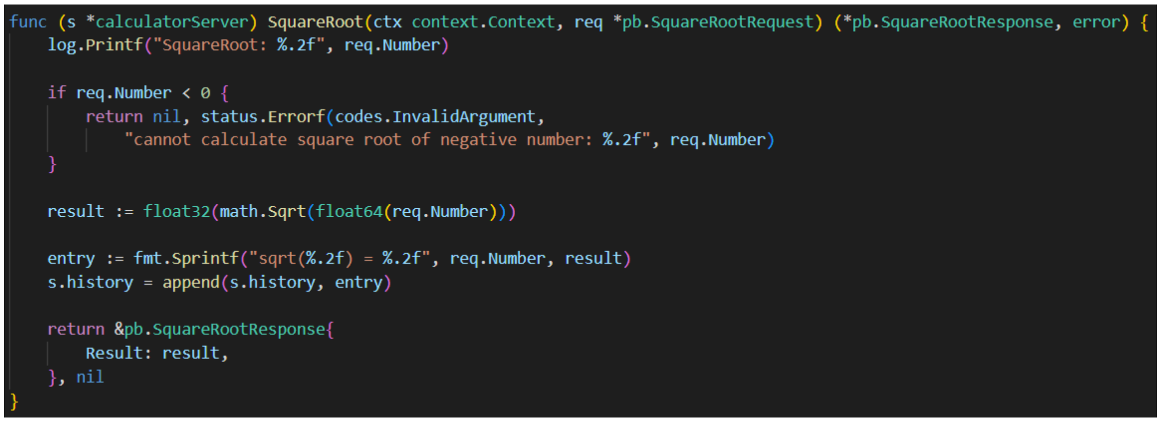
```
type calculatorServer struct {
	pb.UnimplementedCalculatorServer
	history []string
}
```

- Defines the server struct that implements the Calculator service
- pb.UnimplementedCalculatorServer: Embedding this ensures forward compatibility (if new methods are added to the proto, the code won't break)
- history []string: A slice to store calculation history in memory



```go
func (s *calculatorServer) Calculate(ctx context.Context, req *pb.CalculateRequest) (*pb.CalculateResponse, error) {
    log.Printf("Calculate: %.2f %s %.2f", req.A, req.Operation, req.B)

    var result float32

    switch req.Operation {
    case "add":
        result = req.A + req.B
    case "subtract":
        result = req.A - req.B
    case "multiply":
        result = req.A * req.B
    case "divide":
        if req.B == 0 {
            return nil, status.Errorf(codes.InvalidArgument, "cannot divide by zero")
        }
        result = req.A / req.B
    default:
        return nil, status.Errorf(codes.InvalidArgument, "unknown operation: %s", req.Operation)
    }

    entry := fmt.Sprintf("%.2f %s %.2f = %.2f", req.A, req.Operation, req.B, result)
    s.history = append(s.history, entry)

    return &pb.CalculateResponse{
        Result:    result,
        Operation: req.Operation,
    }, nil
}
```

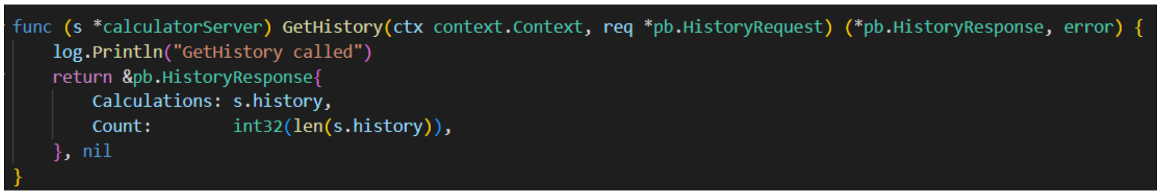
- Takes context.Context (required for all gRPC methods) and a pointer to the request; returns a pointer to the response and an error
- Logs the incoming request with 2 decimal places formatting
- Switch statement handles different operations based on the operation field
- Creates a formatted string of the calculation then appends this entry to the server's history slice
- Returns the response with the calculation result and operation



```go
func (s *calculatorServer) SquareRoot(ctx context.Context, req *pb.SquareRootRequest) (*pb.SquareRootResponse, error) {
    log.Printf("SquareRoot: %.2f", req.Number)

    if req.Number < 0 {
        return nil, status.Errorf(codes.InvalidArgument,
            "cannot calculate square root of negative number: %.2f", req.Number)
    }

    result := float32(math.Sqrt(float64(req.Number)))

    entry := fmt.Sprintf("sqrt(%.2f) = %.2f", req.Number, result)
    s.history = append(s.history, entry)

    return &pb.SquareRootResponse{
        Result: result,
    }, nil
}
```

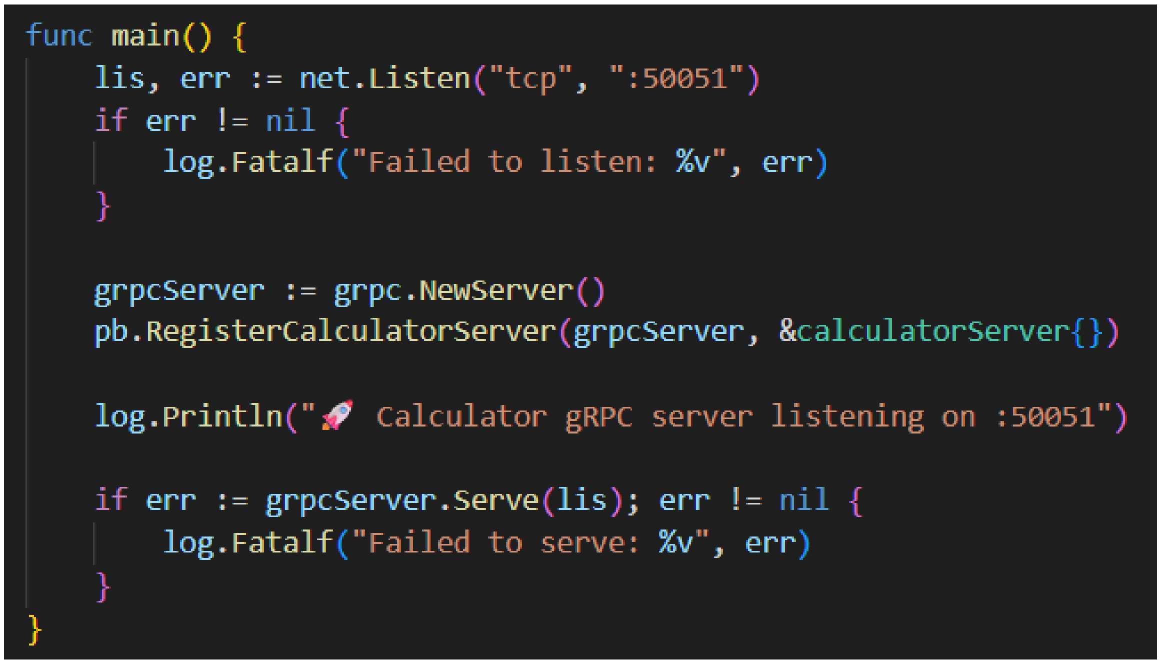
- Logs the incoming square root request
- Validates input: negative numbers don't have real square roots
- math.Sqrt() requires float64, so the number is cast to float64 for the calculation and the result is converted back to float32.

- Creates history entry and appends to history
- Returns the result wrapped in the response message



```go
func (s *calculatorServer) GetHistory(ctx context.Context, req *pb.HistoryRequest) (*pb.HistoryResponse, error) {
    log.Println("GetHistory called")
    return &pb.HistoryResponse{
        Calculations: s.history,
        Count:        int32(len(s.history)),
    }, nil
}
```

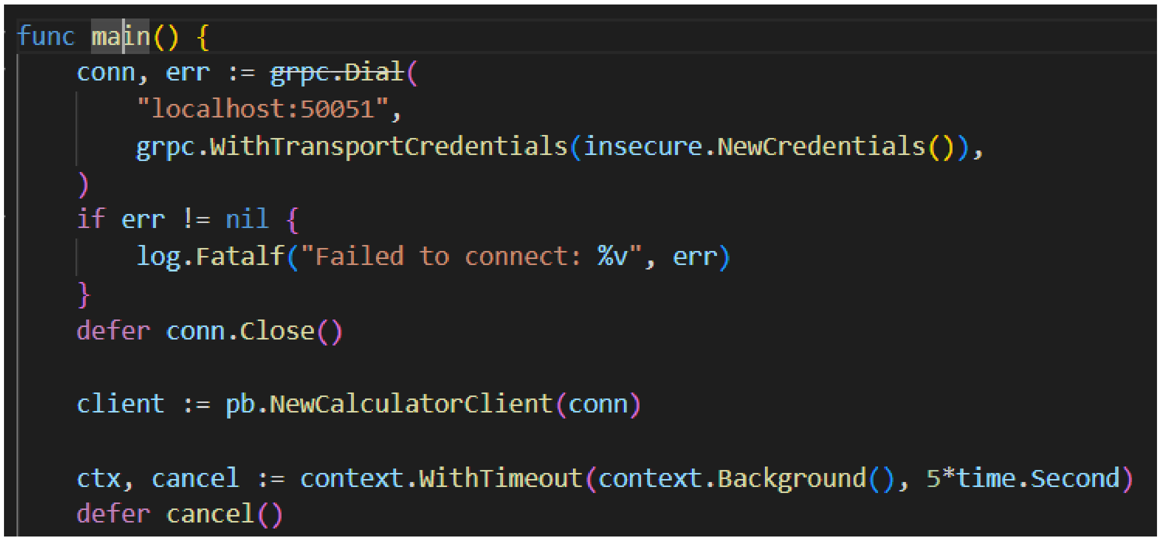
- Simple logging when history is requested and returning all stored calculations and the count.



```go
func main() {
    lis, err := net.Listen("tcp", ":50051")
    if err != nil {
        log.Fatalf("Failed to listen: %v", err)
    }

    grpcServer := grpc.NewServer()
    pb.RegisterCalculatorServer(grpcServer, &calculatorServer{})

    log.Println("🚀 Calculator gRPC server listening on :50051")

    if err := grpcServer.Serve(lis); err != nil {
        log.Fatalf("Failed to serve: %v", err)
    }
}
```

- Starts the gRPC server by creating a TCP listener on port 50051
- Initializing a new gRPC server, registering the calculatorServer implementation
- Logging that the server is running, and finally calling Serve to begin handling incoming gRPC requests.

Client:



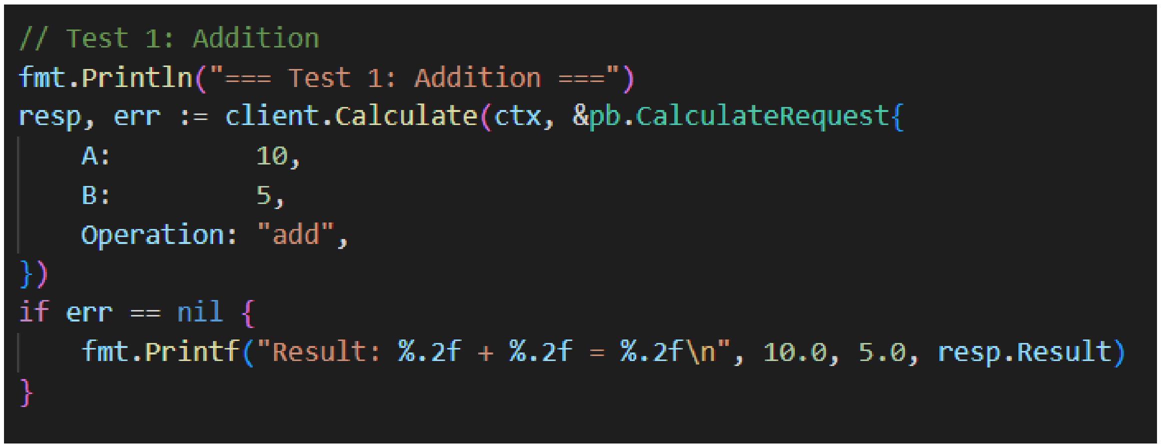
```go
func main() {
    conn, err := ~~grpc.Dial~~(
        "localhost:50051",
        grpc.WithTransportCredentials(insecure.NewCredentials()),
    )
    if err != nil {
        log.Fatalf("Failed to connect: %v", err)
    }
    defer conn.Close()

    client := pb.NewCalculatorClient(conn)

    ctx, cancel := context.WithTimeout(context.Background(), 5*time.Second)
    defer cancel()
```

Establish Connection

- grpc.Dial("localhost:50051", grpc.WithTransportCredentials(insecure.NewCredentials())): connects to the server over an insecure channel.
- Exits with fatal error if connection fails.
- defer conn.Close(): ensures the connection closes when the program ends.

Create Client & Context

- pb.NewCalculatorClient(conn): generates a Calculator service client stub.
- context.WithTimeout(..., 5*time.Second): creates a context with a 5-second RPC timeout.
- defer cancel(): cleans up the context when finished.



```go
// Test 1: Addition
fmt.Println("=== Test 1: Addition ===")
resp, err := client.Calculate(ctx, &pb.CalculateRequest{
    A:         10,
    B:         5,
    Operation: "add",
})
if err == nil {
    fmt.Printf("Result: %.2f + %.2f = %.2f\n", 10.0, 5.0, resp.Result)
}
```

- Prints header, sends Calculate request {A:10, B:5, Operation:"add"}, and prints the formatted result if no error.

```go
// Test 2: Division
fmt.Println("\n=== Test 2: Division ===")
resp, err = client.Calculate(ctx, &pb.CalculateRequest{
    A:         20,
    B:         4,
    Operation: "divide",
})
if err == nil {
    fmt.Printf("Result: %.2f / %.2f = %.2f\n", 20.0, 4.0, resp.Result)
}
```

- Runs a valid division test (20 / 4), prints result, and uses the same context as previous calls.

```go
// Test 3: Division by Zero
fmt.Println("\n=== Test 3: Division by Zero ===")
_, err = client.Calculate(ctx, &pb.CalculateRequest{
    A:         10,
    B:         0,
    Operation: "divide",
})
if err != nil {
    st, _ := status.FromError(err)
    fmt.Printf("Expected error: %s\n", st.Message())
}
```

- Sends a divide-by-zero request, expects an error, converts it to a gRPC status, and prints the server-returned message.

```go
// Test 4: Square Root
fmt.Println("\n=== Test 4: Square Root ===")
sqrtResp, err := client.SquareRoot(ctx, &pb.SquareRootRequest{Number: 16})
if err == nil {
    fmt.Printf("Result: sqrt(16.00) = %.2f\n", sqrtResp.Result)
}
```

- Calls SquareRoot with 16, prints the result (expected 4.00) when successful.

```go
// Test 5: Negative sqrt
fmt.Println("\n=== Test 5: Negative Square Root ===")
_, err = client.SquareRoot(ctx, &pb.SquareRootRequest{Number: -4})
if err != nil {
    st, _ := status.FromError(err)
    fmt.Printf("Expected error: %s\n", st.Message())
}
```

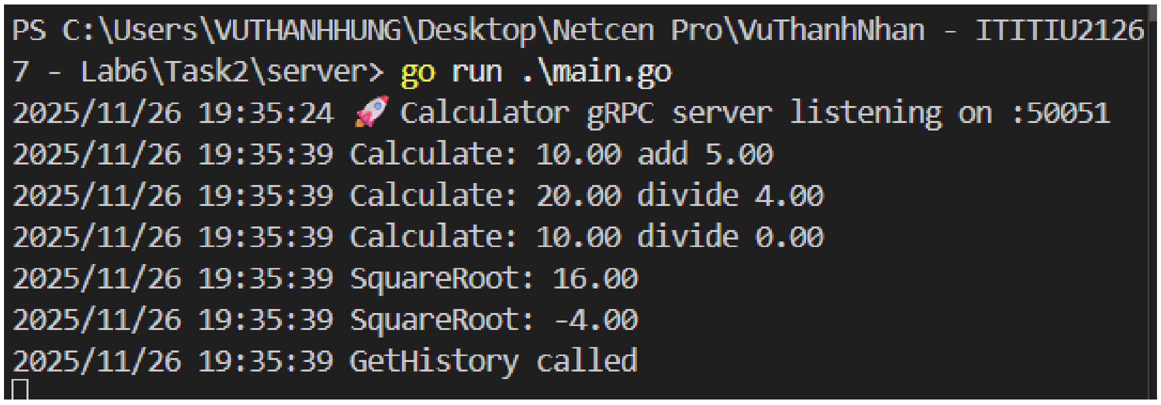
- Sends a negative number to SquareRoot, receives a gRPC error, extracts and prints the error message.

```go
// Test 6: Get history
fmt.Println("\n=== Test 6: History ===")
hist, _ := client.GetHistory(ctx, &pb.HistoryRequest{})
fmt.Printf("Calculations: %d\n", hist.Count)
for i, h := range hist.Calculations {
    fmt.Printf("%d. %s\n", i+1, h)
}
```

- Calls GetHistory, prints the total count, and iterates through all stored calculation entries for display.

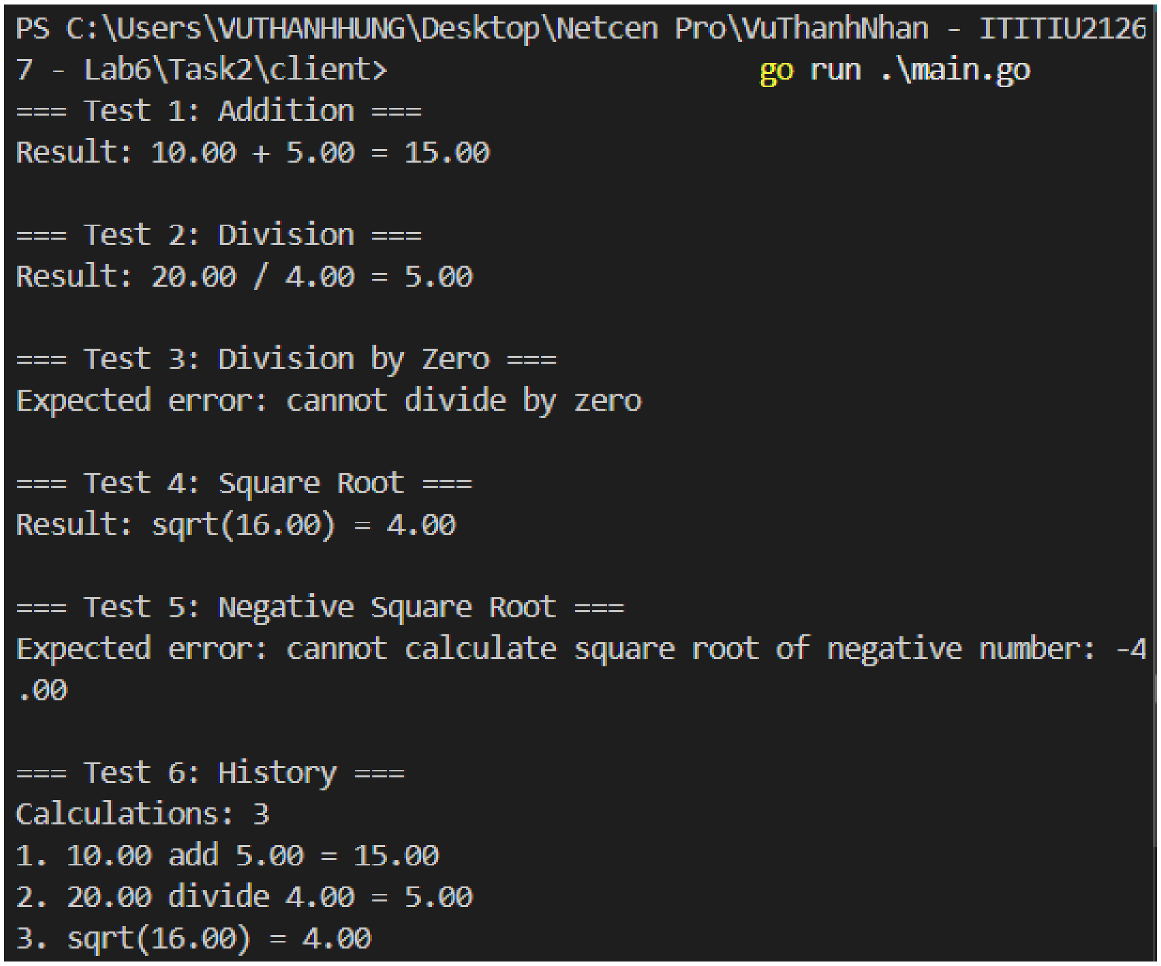
Output:

Server:



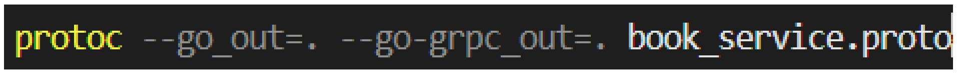
```
PS C:\Users\VUTHANHHUNG\Desktop\Netcen Pro\VuThanhNhan - ITITIU2126
7 - Lab6\Task2\server> go run .\main.go
2025/11/26 19:35:24 🚀 Calculator gRPC server listening on :50051
2025/11/26 19:35:39 Calculate: 10.00 add 5.00
2025/11/26 19:35:39 Calculate: 20.00 divide 4.00
2025/11/26 19:35:39 Calculate: 10.00 divide 0.00
2025/11/26 19:35:39 SquareRoot: 16.00
2025/11/26 19:35:39 SquareRoot: -4.00
2025/11/26 19:35:39 GetHistory called
```

Client:



```
PS C:\Users\VUTHANHHUNG\Desktop\Netcen Pro\VuThanhNhan - ITITIU2126
7 - Lab6\Task2\client>                          go run .\main.go
=== Test 1: Addition ===
Result: 10.00 + 5.00 = 15.00

=== Test 2: Division ===
Result: 20.00 / 4.00 = 5.00

=== Test 3: Division by Zero ===
Expected error: cannot divide by zero

=== Test 4: Square Root ===
Result: sqrt(16.00) = 4.00

=== Test 5: Negative Square Root ===
Expected error: cannot calculate square root of negative number: -4
.00

=== Test 6: History ===
Calculations: 3
1. 10.00 add 5.00 = 15.00
2. 20.00 divide 4.00 = 5.00
3. sqrt(16.00) = 4.00
```

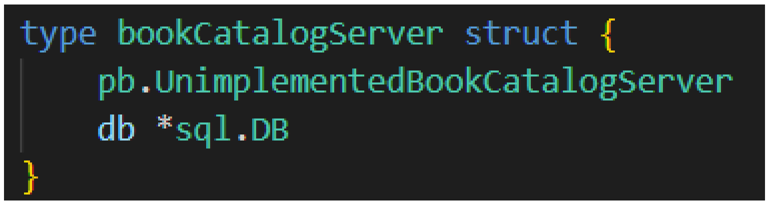
Task 3:



```
protoc --go_out=. --go-grpc_out=. book_service.proto
```

- protoc: Protocol Buffers compiler.
- --go_out=.: Generates Go structs/messages for the proto definitions in the current directory.
- --go-grpc_out=.: Generates Go gRPC service code (interfaces and stubs) in the current directory.

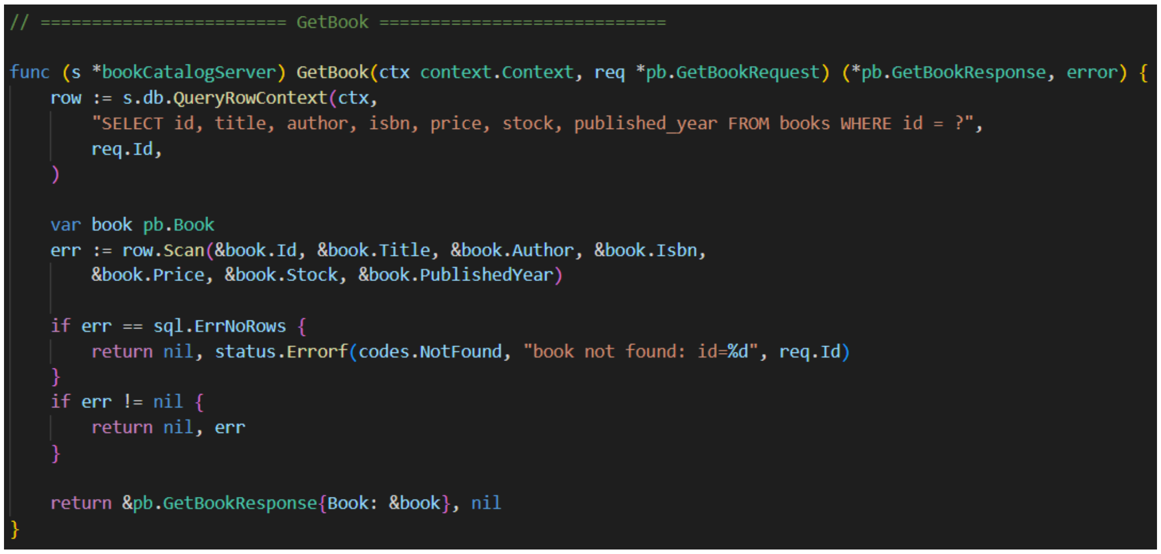
Server:



```go
type bookCatalogServer struct {
	pb.UnimplementedBookCatalogServer
	db *sql.DB
}
```

bookCatalogServer defines the gRPC server for the BookCatalog service.

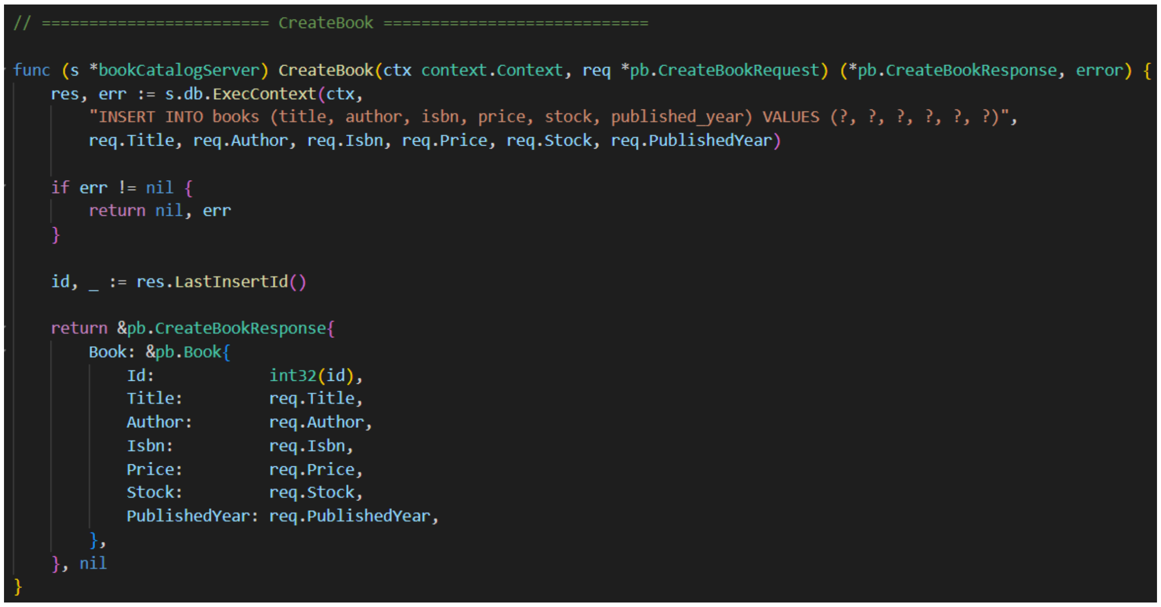
- Embeds pb.UnimplementedBookCatalogServer for forward compatibility.
- Holds a *sql.DB connection shared by all RPC methods, enabling database CRUD operations.



```go
// ======================= GetBook ==========================

func (s *bookCatalogServer) GetBook(ctx context.Context, req *pb.GetBookRequest) (*pb.GetBookResponse, error) {
	row := s.db.QueryRowContext(ctx,
		"SELECT id, title, author, isbn, price, stock, published_year FROM books WHERE id = ?",
		req.Id,
	)

	var book pb.Book
	err := row.Scan(&book.Id, &book.Title, &book.Author, &book.Isbn,
		&book.Price, &book.Stock, &book.PublishedYear)

	if err == sql.ErrNoRows {
		return nil, status.Errorf(codes.NotFound, "book not found: id=%d", req.Id)
	}
	if err != nil {
		return nil, err
	}

	return &pb.GetBookResponse{Book: &book}, nil
}
```

GetBook retrieves a single book by its ID.

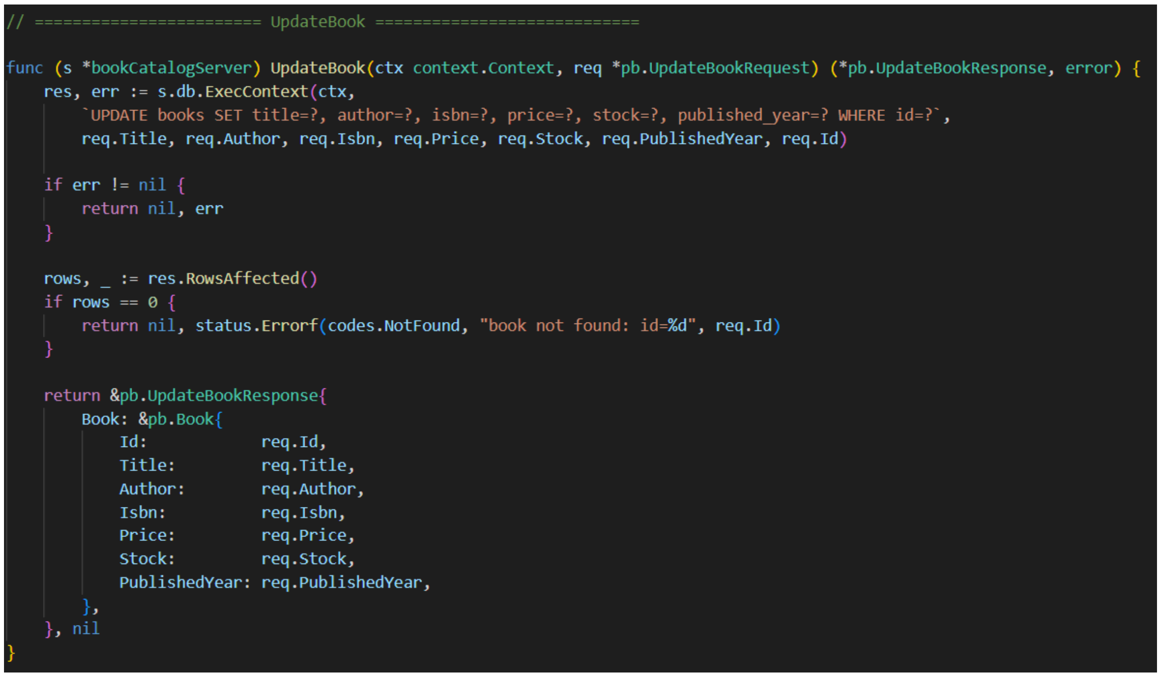
- Executes a parameterized SQL query using QueryRowContext to safely fetch one row (ctx enables cancellation/timeout; ? prevents SQL injection).
- Scans the result into a pb.Book struct, matching the SELECT column order.
- If no row exists, returns a gRPC NotFound error; any other scan/query error is returned as-is.
- On success, wraps the populated Book in GetBookResponse and returns it.



```go
// ======================= CreateBook ===========================

func (s *bookCatalogServer) CreateBook(ctx context.Context, req *pb.CreateBookRequest) (*pb.CreateBookResponse, error) {
    res, err := s.db.ExecContext(ctx,
        "INSERT INTO books (title, author, isbn, price, stock, published_year) VALUES (?, ?, ?, ?, ?, ?)",
        req.Title, req.Author, req.Isbn, req.Price, req.Stock, req.PublishedYear)

    if err != nil {
        return nil, err
    }

    id, _ := res.LastInsertId()

    return &pb.CreateBookResponse{
        Book: &pb.Book{
            Id:            int32(id),
            Title:         req.Title,
            Author:        req.Author,
            Isbn:          req.Isbn,
            Price:         req.Price,
            Stock:         req.Stock,
            PublishedYear: req.PublishedYear,
        },
    }, nil
}
```

CreateBook inserts a new book into the database.

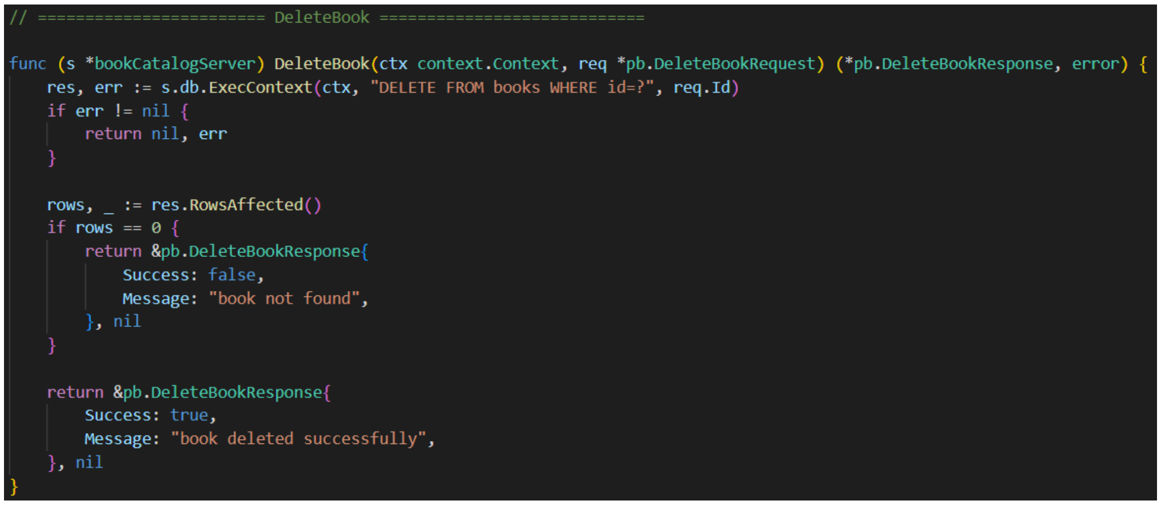
- Uses ExecContext to run an INSERT statement (no returned rows).
- Fills the six ? placeholders with the request fields; id is omitted because it's auto-generated.
- Returns an error immediately if the insert fails.
- Retrieves the generated ID using LastInsertId and converts it to int32.
- Builds and returns a CreateBookResponse containing the newly created Book, including its assigned ID.



```go
// ======================== UpdateBook ============================

func (s *bookCatalogServer) UpdateBook(ctx context.Context, req *pb.UpdateBookRequest) (*pb.UpdateBookResponse, error) {
    res, err := s.db.ExecContext(ctx,
        `UPDATE books SET title=?, author=?, isbn=?, price=?, stock=?, published_year=? WHERE id=?`,
        req.Title, req.Author, req.Isbn, req.Price, req.Stock, req.PublishedYear, req.Id)

    if err != nil {
        return nil, err
    }

    rows, _ := res.RowsAffected()
    if rows == 0 {
        return nil, status.Errorf(codes.NotFound, "book not found: id=%d", req.Id)
    }

    return &pb.UpdateBookResponse{
        Book: &pb.Book{
            Id:            req.Id,
            Title:         req.Title,
            Author:        req.Author,
            Isbn:          req.Isbn,
            Price:         req.Price,
            Stock:         req.Stock,
            PublishedYear: req.PublishedYear,
        },
    }, nil
}
```

UpdateBook modifies an existing book by ID.

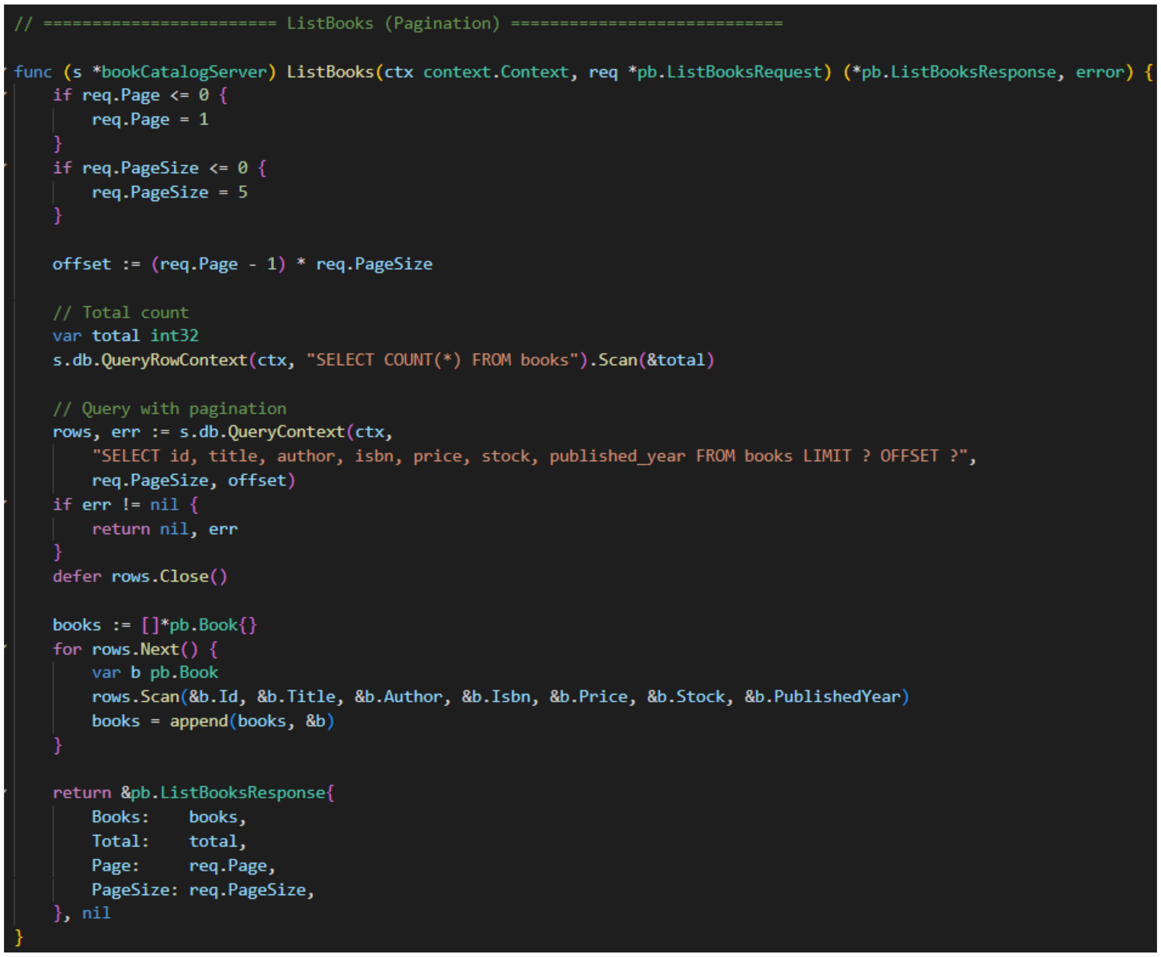
- Executes a parameterized UPDATE statement with seven placeholders (six fields + id).
- Returns any database errors immediately.
- Checks RowsAffected(); if zero, returns gRPC NotFound (book doesn’t exist).
- On success, returns UpdateBookResponse containing the updated book, echoing the request values.



```go
// ======================== DeleteBook ============================

func (s *bookCatalogServer) DeleteBook(ctx context.Context, req *pb.DeleteBookRequest) (*pb.DeleteBookResponse, error) {
    res, err := s.db.ExecContext(ctx, "DELETE FROM books WHERE id=?", req.Id)
    if err != nil {
        return nil, err
    }

    rows, _ := res.RowsAffected()
    if rows == 0 {
        return &pb.DeleteBookResponse{
            Success: false,
            Message: "book not found",
        }, nil
    }

    return &pb.DeleteBookResponse{
        Success: true,
        Message: "book deleted successfully",
    }, nil
}
```

DeleteBook removes a book by ID.

- Executes a parameterized DELETE query and handles any database errors.
- Checks RowsAffected(); if zero, returns a response with Success: false and a “book not found” message (not an error).
- If a row was deleted, returns Success: true with a confirmation message.

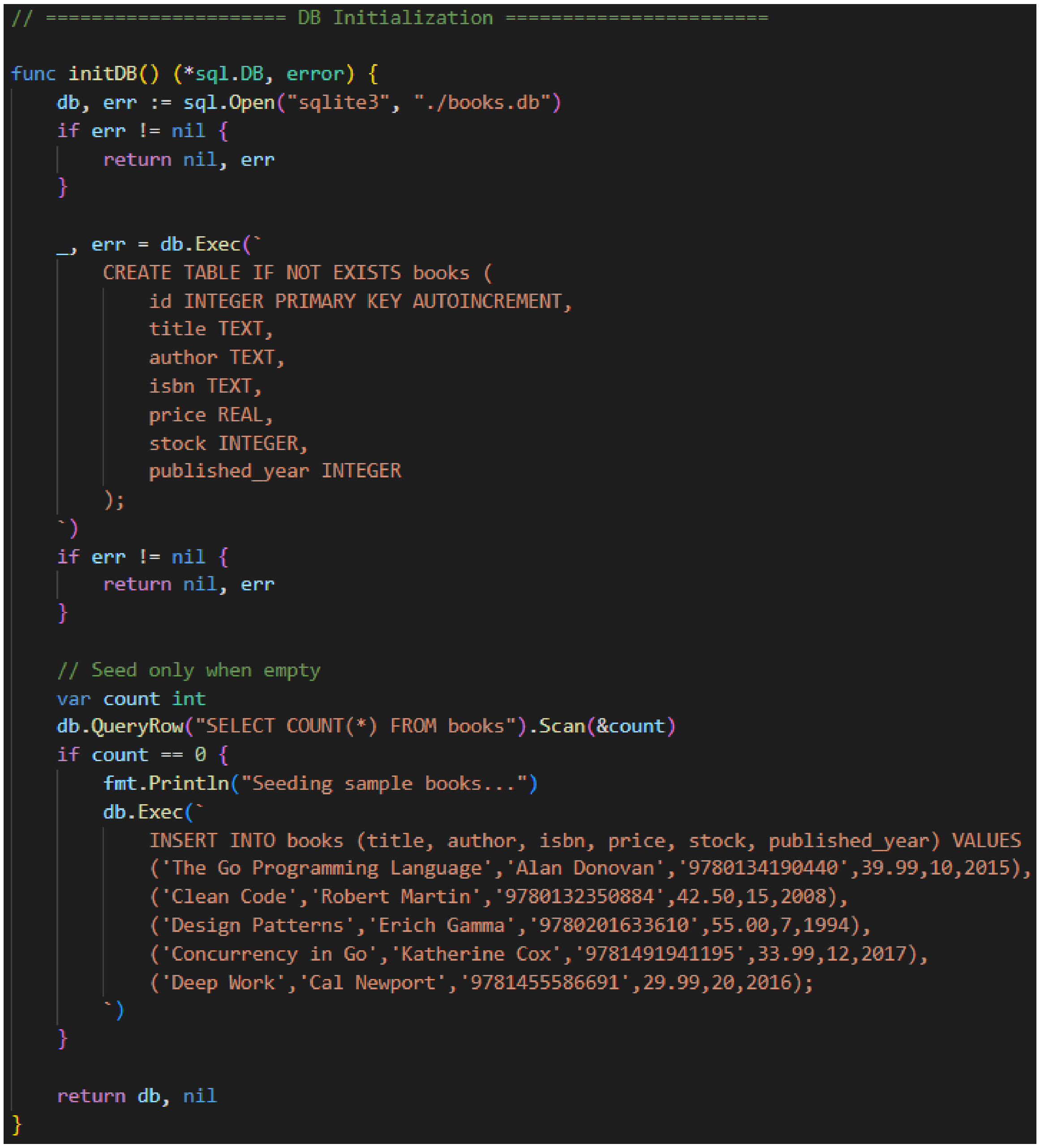


```go
// ======================== ListBooks (Pagination) ============================

func (s *bookCatalogServer) ListBooks(ctx context.Context, req *pb.ListBooksRequest) (*pb.ListBooksResponse, error) {
    if req.Page <= 0 {
        req.Page = 1
    }
    if req.PageSize <= 0 {
        req.PageSize = 5
    }

    offset := (req.Page - 1) * req.PageSize

    // Total count
    var total int32
    s.db.QueryRowContext(ctx, "SELECT COUNT(*) FROM books").Scan(&total)

    // Query with pagination
    rows, err := s.db.QueryContext(ctx,
        "SELECT id, title, author, isbn, price, stock, published_year FROM books LIMIT ? OFFSET ?",
        req.PageSize, offset)
    if err != nil {
        return nil, err
    }
    defer rows.Close()

    books := []*pb.Book{}
    for rows.Next() {
        var b pb.Book
        rows.Scan(&b.Id, &b.Title, &b.Author, &b.Isbn, &b.Price, &b.Stock, &b.PublishedYear)
        books = append(books, &b)
    }

    return &pb.ListBooksResponse{
        Books:    books,
        Total:    total,
        Page:     req.Page,
        PageSize: req.PageSize,
    }, nil
}
```

ListBooks returns a paginated list of books.

- Validates Page and PageSize, defaulting to 1 and 5 if invalid.
- Calculates offset = (Page-1) * PageSize to skip rows.
- Queries total book count for pagination info.
- Executes a parameterized SELECT with LIMIT and OFFSET to fetch the current page.

- Iterates over rows, scanning each book into a pb.Book and appending to a slice.
- Returns ListBooksResponse containing the current page's books, total count, page number, and page size.

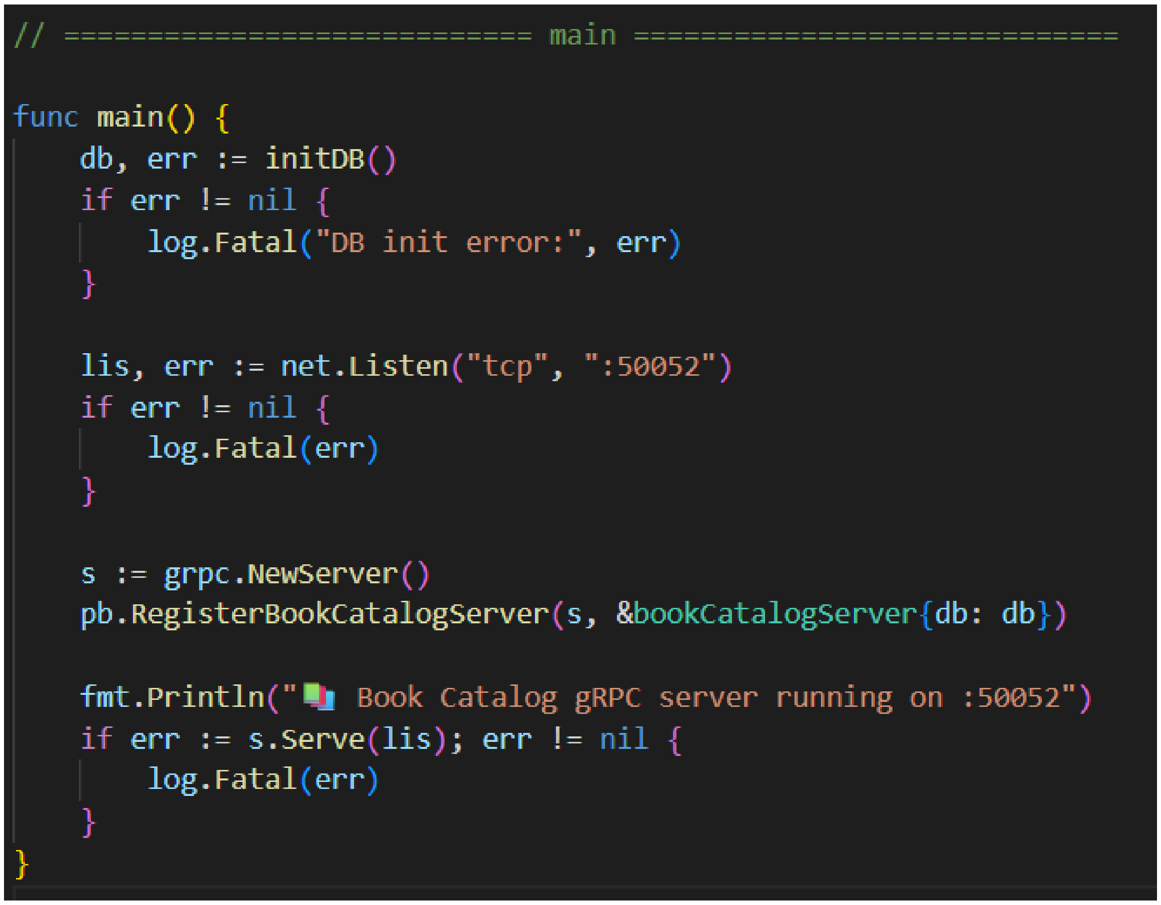


```go
// ==================== DB Initialization ======================

func initDB() (*sql.DB, error) {
    db, err := sql.Open("sqlite3", "./books.db")
    if err != nil {
        return nil, err
    }

    _, err = db.Exec(`
        CREATE TABLE IF NOT EXISTS books (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            title TEXT,
            author TEXT,
            isbn TEXT,
            price REAL,
            stock INTEGER,
            published_year INTEGER
        );
    `)
    if err != nil {
        return nil, err
    }

    // Seed only when empty
    var count int
    db.QueryRow("SELECT COUNT(*) FROM books").Scan(&count)
    if count == 0 {
        fmt.Println("Seeding sample books...")
        db.Exec(`
            INSERT INTO books (title, author, isbn, price, stock, published_year) VALUES
            ('The Go Programming Language','Alan Donovan','9780134190440',39.99,10,2015),
            ('Clean Code','Robert Martin','9780132350884',42.50,15,2008),
            ('Design Patterns','Erich Gamma','9780201633610',55.00,7,1994),
            ('Concurrency in Go','Katherine Cox','9781491941195',33.99,12,2017),
            ('Deep Work','Cal Newport','9781455586691',29.99,20,2016);
        `)
    }

    return db, nil
}
```

initDB initializes the SQLite database and returns a *sql.DB connection.

- Opens the SQLite database file (books.db), creating it if needed.

- Creates the books table if it doesn’t exist, defining id as auto-increment primary key and other book fields.
- Checks if the table is empty; if so, seeds it with five sample books.
- Returns the database connection for use by the server.

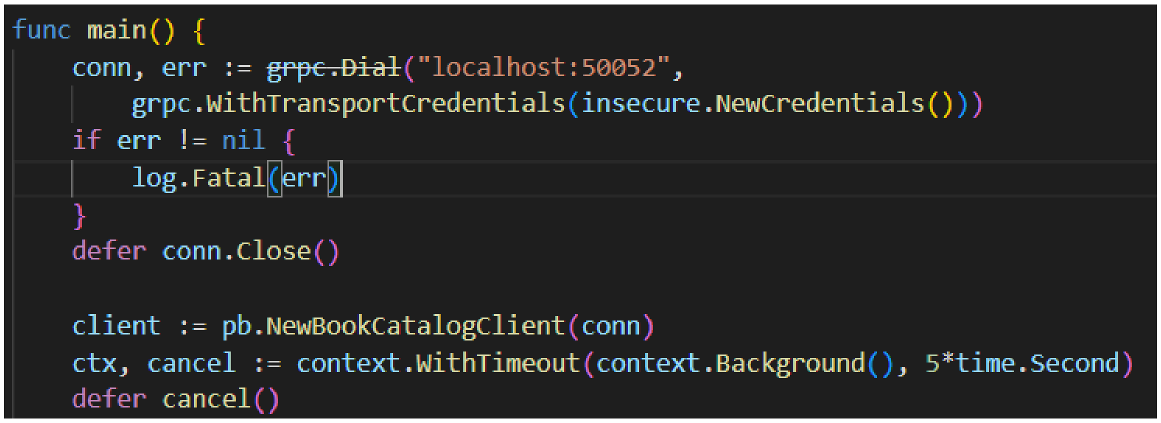


```go
// =========================== main =============================

func main() {
    db, err := initDB()
    if err != nil {
        log.Fatal("DB init error:", err)
    }

    lis, err := net.Listen("tcp", ":50052")
    if err != nil {
        log.Fatal(err)
    }

    s := grpc.NewServer()
    pb.RegisterBookCatalogServer(s, &bookCatalogServer{db: db})

    fmt.Println("📚 Book Catalog gRPC server running on :50052")
    if err := s.Serve(lis); err != nil {
        log.Fatal(err)
    }
}
```

main starts the Book Catalog gRPC server.

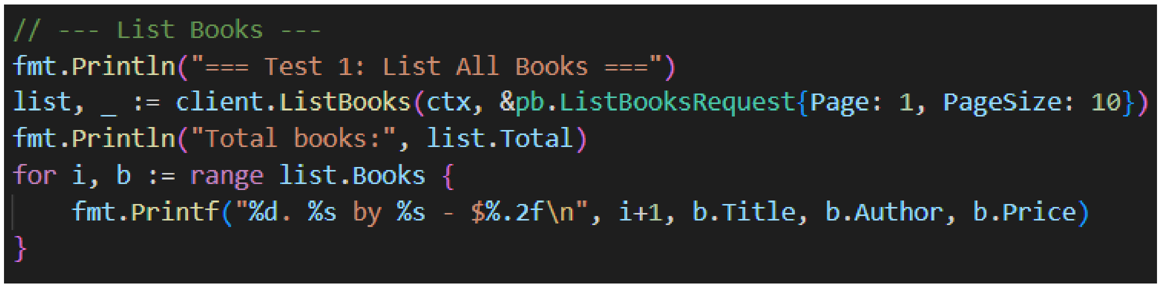
- Initializes the database using initDB, logging a fatal error if it fails.
- Creates a TCP listener on port 50052 for incoming gRPC connections.
- Instantiates a new gRPC server and registers bookCatalogServer with the database connection.
- Prints a startup message and begins serving, blocking indefinitely until the server stops or an error occurs.

Client:



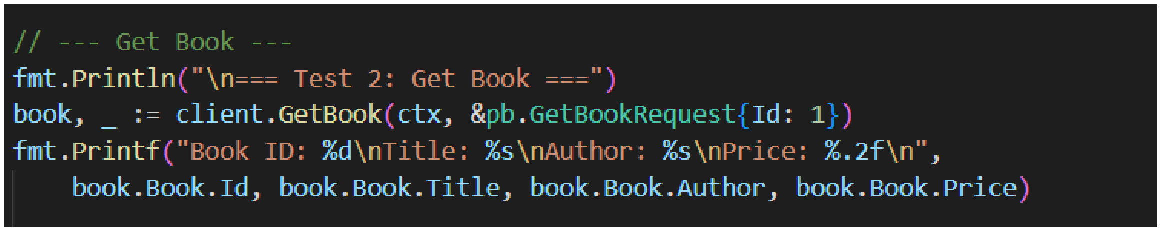
```go
func main() {
    conn, err := ~~grpc.Dial~~("localhost:50052",
        grpc.WithTransportCredentials(insecure.NewCredentials()))
    if err != nil {
        log.Fatal(err)
    }
    defer conn.Close()

    client := pb.NewBookCatalogClient(conn)
    ctx, cancel := context.WithTimeout(context.Background(), 5*time.Second)
    defer cancel()
```

- Connects to localhost:50052 using an insecure channel.
- Creates a BookCatalog client stub with a 5-second timeout context.



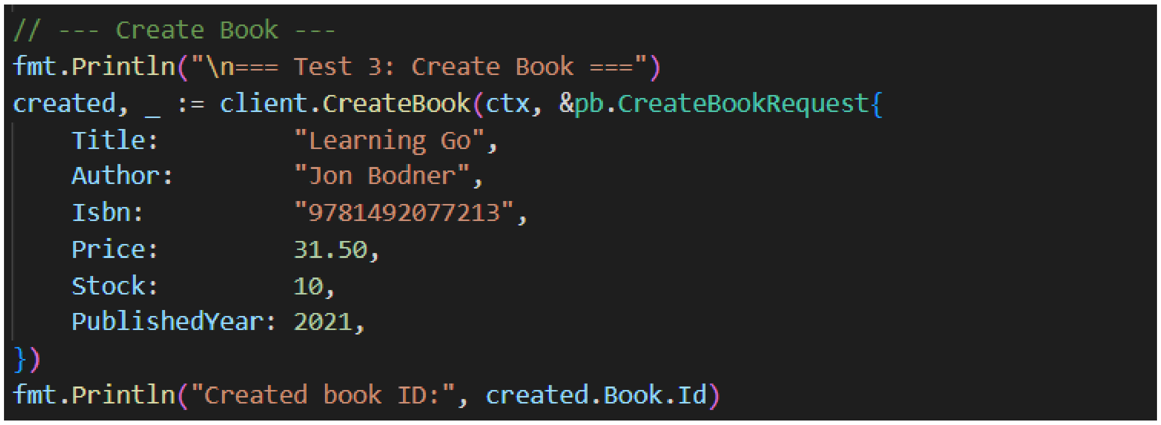
```go
// --- List Books ---
fmt.Println("=== Test 1: List All Books ===")
list, _ := client.ListBooks(ctx, &pb.ListBooksRequest{Page: 1, PageSize: 10})
fmt.Println("Total books:", list.Total)
for i, b := range list.Books {
    fmt.Printf("%d. %s by %s - $%.2f\n", i+1, b.Title, b.Author, b.Price)
}
```

- Calls ListBooks (page 1, 10 items) and prints total books and a numbered list of title, author, and price.



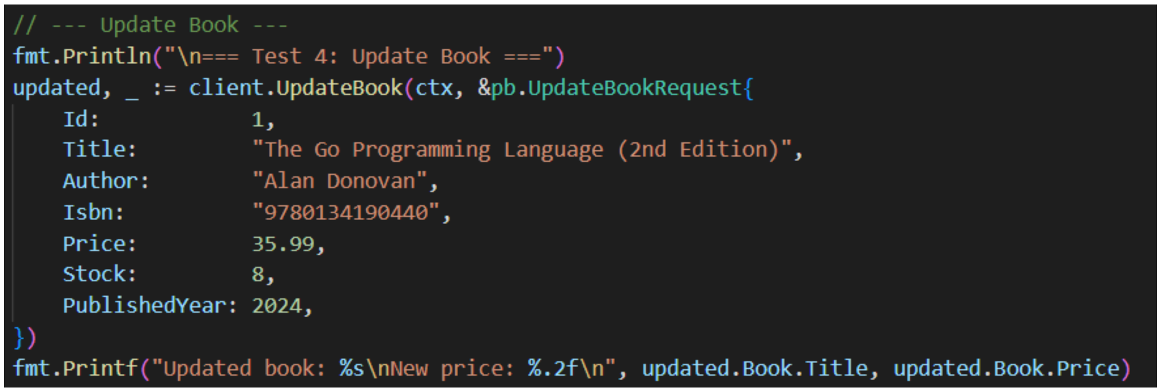
```go
// --- Get Book ---
fmt.Println("\n=== Test 2: Get Book ===")
book, _ := client.GetBook(ctx, &pb.GetBookRequest{Id: 1})
fmt.Printf("Book ID: %d\nTitle: %s\nAuthor: %s\nPrice: %.2f\n",
    book.Book.Id, book.Book.Title, book.Book.Author, book.Book.Price)
```

- Calls GetBook for ID=1 and prints detailed info (ID, title, author, price).



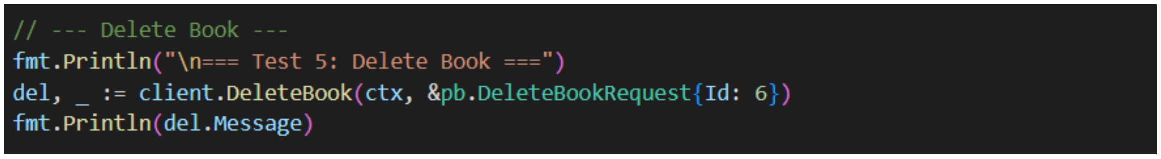
```go
// --- Create Book ---
fmt.Println("\n=== Test 3: Create Book ===")
created, _ := client.CreateBook(ctx, &pb.CreateBookRequest{
    Title:         "Learning Go",
    Author:        "Jon Bodner",
    Isbn:          "9781492077213",
    Price:         31.50,
    Stock:         10,
    PublishedYear: 2021,
})
fmt.Println("Created book ID:", created.Book.Id)
```

- Calls CreateBook with book details; prints the server-generated ID.



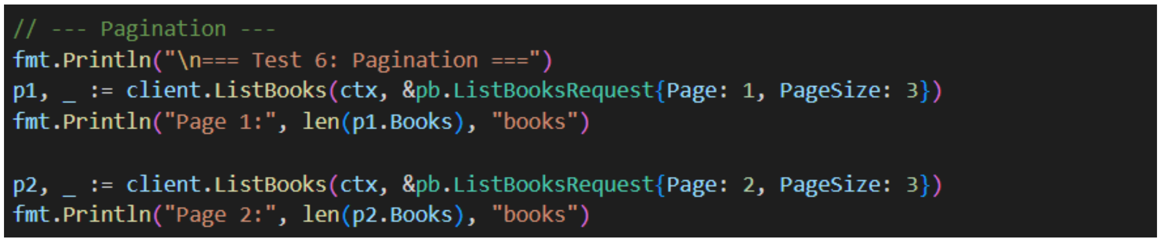
```go
// --- Update Book ---
fmt.Println("\n=== Test 4: Update Book ===")
updated, _ := client.UpdateBook(ctx, &pb.UpdateBookRequest{
    Id:            1,
    Title:         "The Go Programming Language (2nd Edition)",
    Author:        "Alan Donovan",
    Isbn:          "9780134190440",
    Price:         35.99,
    Stock:         8,
    PublishedYear: 2024,
})
fmt.Printf("Updated book: %s\nNew price: %.2f\n", updated.Book.Title, updated.Book.Price)
```

- Calls UpdateBook for ID=1 with new values; prints updated title and price.



```go
// --- Delete Book ---
fmt.Println("\n=== Test 5: Delete Book ===")
del, _ := client.DeleteBook(ctx, &pb.DeleteBookRequest{Id: 6})
fmt.Println(del.Message)
```

- Calls DeleteBook for ID=6; prints the server message (deleted successfully or not found).

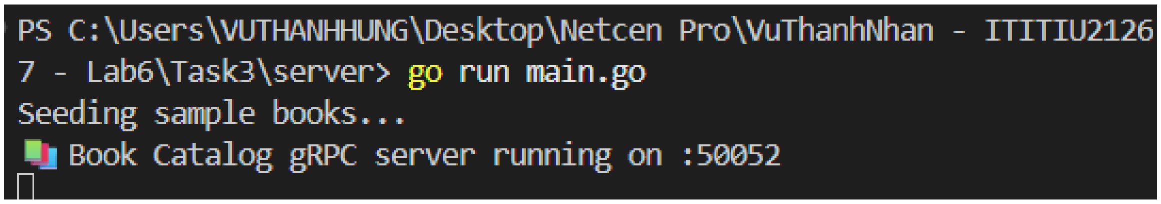


```go
// --- Pagination ---
fmt.Println("\n=== Test 6: Pagination ===")
p1, _ := client.ListBooks(ctx, &pb.ListBooksRequest{Page: 1, PageSize: 3})
fmt.Println("Page 1:", len(p1.Books), "books")

p2, _ := client.ListBooks(ctx, &pb.ListBooksRequest{Page: 2, PageSize: 3})
fmt.Println("Page 2:", len(p2.Books), "books")
```

- Calls ListBooks with page size 3 for pages 1 and 2; prints number of books returned to demonstrate pagination.

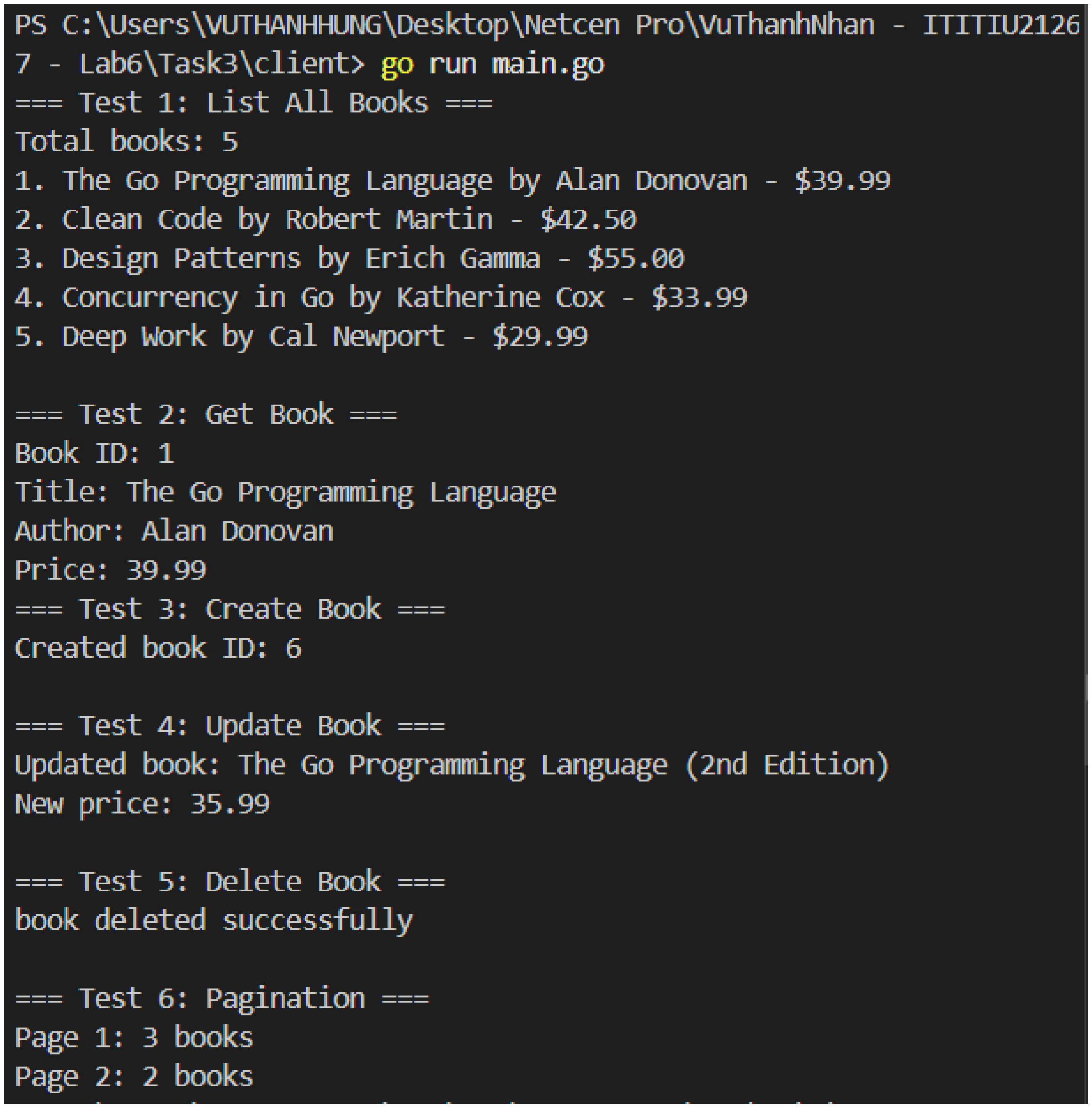
Output:

Server:

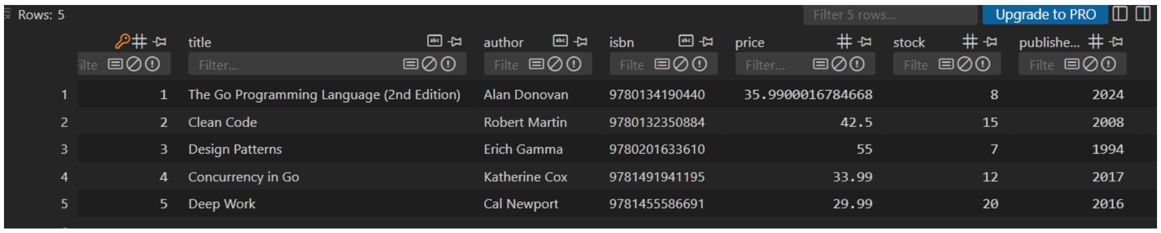


```
PS C:\Users\VUTHANHHUNG\Desktop\Netcen Pro\VuThanhNhan - ITITIU2126
7 - Lab6\Task3\server> go run main.go
Seeding sample books...
📚 Book Catalog gRPC server running on :50052
```

Client:



```
PS C:\Users\VUTHANHHUNG\Desktop\Netcen Pro\VuThanhNhan - ITITIU2126
7 - Lab6\Task3\client> go run main.go
=== Test 1: List All Books ===
Total books: 5
1. The Go Programming Language by Alan Donovan - $39.99
2. Clean Code by Robert Martin - $42.50
3. Design Patterns by Erich Gamma - $55.00
4. Concurrency in Go by Katherine Cox - $33.99
5. Deep Work by Cal Newport - $29.99

=== Test 2: Get Book ===
Book ID: 1
Title: The Go Programming Language
Author: Alan Donovan
Price: 39.99
=== Test 3: Create Book ===
Created book ID: 6

=== Test 4: Update Book ===
Updated book: The Go Programming Language (2nd Edition)
New price: 35.99

=== Test 5: Delete Book ===
book deleted successfully

=== Test 6: Pagination ===
Page 1: 3 books
Page 2: 2 books
```



Rows: 5

		title	author	isbn	price	stock	publishe...
1	1	The Go Programming Language (2nd Edition)	Alan Donovan	9780134190440	35.9900016784668	8	2024
2	2	Clean Code	Robert Martin	9780132350884	42.5	15	2008
3	3	Design Patterns	Erich Gamma	9780201633610	55	7	1994
4	4	Concurrency in Go	Katherine Cox	9781491941195	33.99	12	2017
5	5	Deep Work	Cal Newport	9781455586691	29.99	20	2016