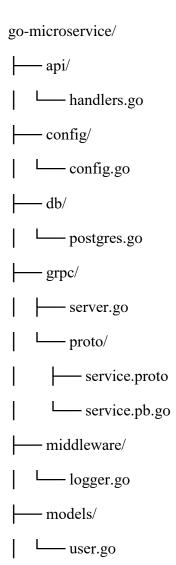
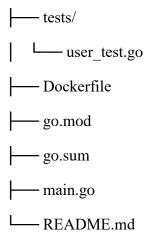
High-Performance Concurrent Golang Microservice with REST & gRPC Support

Keywords: Golang concurrency, gRPC microservice, PostgreSQL integration, RESTful API in Go, middleware chaining, unit testing in Go, GoFiber, Dockerized Go app, Go channels, Go routines, Go context, structured logging in Go.

Folder Structure





go.mod

module github.com/yourname/go-microservice

```
require (
github.com/gofiber/fiber/v2 v2.50.1
github.com/jackc/pgx/v5 v5.5.2
google.golang.org/grpc v1.64.0
github.com/joho/godotenv v1.5.1
)
```

main.go

```
package main
import (
"context"
```

```
"log"
       "os"
       "os/signal"
       "syscall"
       "time"
       "github.com/gofiber/fiber/v2"
       "github.com/joho/godotenv"
       "go-microservice/config"
       "go-microservice/api"
       "go-microservice/middleware"
       "go-microservice/db"
       "go-microservice/grpc"
)
func main() {
       // Load environment variables
       if err := godotenv.Load(); err != nil {
              log.Println("Warning: .env file not found.")
       }
       // Initialize PostgreSQL
       db.InitPostgres()
```

```
// Set up Fiber app
app := fiber.New()
app.Use(middleware.LoggerMiddleware)
api.SetupRoutes(app)
// Start REST API Server
go func() {
       log.Println("□ REST API running on port 8080")
       if err := app.Listen(":8080"); err != nil {
              log.Fatalf("Failed to start REST server: %v", err)
       }
}()
// Start gRPC Server
go grpc.StartGRPCServer()
// Graceful Shutdown
quit := make(chan os.Signal, 1)
signal.Notify(quit, syscall.SIGINT, syscall.SIGTERM)
<-quit
log.Println("□ Shutting down...")
ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
defer cancel()
```

```
_ = app.Shutdown()
log.Println("Application shutdown complete.")
}
```

api/handlers.go

```
import (
          "github.com/gofiber/fiber/v2"
           "go-microservice/models"
)

func SetupRoutes(app *fiber.App) {
            api := app.Group("/api/v1")

            api.Post("/users", models.CreateUserHandler)
            api.Get("/users/:id", models.GetUserHandler)
            api.Get("/users", models.ListUsersHandler)
}
```

models/user.go

```
package models
import (
"context"
```

```
"fmt"
       "strconv"
       "time"
       "github.com/gofiber/fiber/v2"
       "go-microservice/db"
)
type User struct {
       ID
              int
                     `json:"id"`
                string `json:"name"`
       Name
               string `json:"email"`
       Email
       CreatedAt time.Time `json:"created_at"`
}
func CreateUserHandler(c *fiber.Ctx) error {
       u := new(User)
       if err := c.BodyParser(u); err != nil {
              return c.Status(fiber.StatusBadRequest).JSON(fiber.Map{"error": "invalid
body"})
       }
       err := db.DB.QueryRow(context.Background(),
              "INSERT INTO users(name, email, created_at) VALUES($1, $2, $3)
RETURNING id",
              u.Name, u.Email, time.Now()).Scan(&u.ID)
       if err != nil {
```

```
return c.Status(500).JSON(fiber.Map{"error": "insert failed"})
       }
       return c.Status(201).JSON(u)
}
func GetUserHandler(c *fiber.Ctx) error {
       id, err := strconv.Atoi(c.Params("id"))
       if err != nil {
              return c.Status(400).JSON(fiber.Map{"error": "invalid ID"})
       }
       var user User
       err = db.DB.QueryRow(context.Background(),
              "SELECT id, name, email, created_at FROM users WHERE id=$1", id).
              Scan(&user.ID, &user.Name, &user.Email, &user.CreatedAt)
       if err != nil {
              return c.Status(404).JSON(fiber.Map{"error": "user not found"})
       }
       return c.JSON(user)
}
func ListUsersHandler(c *fiber.Ctx) error {
       rows, err := db.DB.Query(context.Background(), "SELECT id, name, email, created_at
FROM users")
```

```
if err != nil {
    return c.Status(500).JSON(fiber.Map{"error": "query error"})
}

defer rows.Close()

var users []User
for rows.Next() {
    var u User
    rows.Scan(&u.ID, &u.Name, &u.Email, &u.CreatedAt)
    users = append(users, u)
}

return c.JSON(users)
}
```

db/postgres.go

```
package db

import (
     "context"
     "log"
     "os"
     "time"

     "github.com/jackc/pgx/v5/pgxpool"
)
```

```
var DB *pgxpool.Pool

func InitPostgres() {
        connStr := os.Getenv("DATABASE_URL")
        var err error

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

        DB, err = pgxpool.New(ctx, connStr)
        if err != nil {
                  log.Fatalf("Unable to connect to PostgreSQL: %v", err)
        }
        log.Println("□ PostgreSQL connection established")
}
```

middleware/logger.go

package middleware

)

```
import (
    "github.com/gofiber/fiber/v2"
    "log"
    "time"
```

```
func LoggerMiddleware(c *fiber.Ctx) error {
    start := time.Now()
    err := c.Next()
    duration := time.Since(start)
    log.Printf("[%s] %s - %s (%dms)", c.Method(), c.Path(), c.IP(), duration.Milliseconds())
    return err
}
```

grpc/server.go

```
package grpc

import (
        "log"
        "net"

        "google.golang.org/grpc"
        pb "go-microservice/grpc/proto"
)

type UserService struct {
        pb.UnimplementedUserServiceServer
}
```

```
func (s *UserService) Ping(ctx context.Context, req *pb.PingRequest) (*pb.PingResponse, error)
       return &pb.PingResponse{Message: "pong"}, nil
}
func StartGRPCServer() {
       listener, err := net.Listen("tcp", ":50051")
       if err != nil {
              log.Fatalf("Failed to listen on port 50051: %v", err)
       }
       s := grpc.NewServer()
       pb.RegisterUserServiceServer(s, &UserService{})
       log.Println("□ gRPC server running on port 50051")
       if err := s.Serve(listener); err != nil {
              log.Fatalf("gRPC server failed: %v", err)
       }
}
```

grpc/proto/service.proto

```
syntax = "proto3";
package proto;
```

```
service UserService {
  rpc Ping(PingRequest) returns (PingResponse);
}
message PingRequest { }
message PingResponse {
  string message = 1;
}
```

tests/user_test.go

```
import (
    "bytes"
    "encoding/json"
    "net/http"
    "net/http/httptest"
    "testing"

    "github.com/gofiber/fiber/v2"
    "go-microservice/api"
)

func TestCreateUser(t *testing.T) {
```

Dockerfile

```
FROM golang:1.21-alpine
```

```
WORKDIR /app
```

```
COPY go.mod ./
```

COPY go.sum ./

RUN go mod download

COPY..

RUN go build -o main .

EXPOSE 8080

EXPOSE 50051

CMD ["./main"]