Go Developer Test

Faraz Ahmad
imfaraz101@gmail.com
+971553578044

Answer 1:

```
package main

import (
"context"
"database/sql"
"fmt"
"math/rand"
"time"

"github.com/gin-gonic/gin" //gin
"github.com/jackc/pgx/v5" // pgx PostgreSQL driver
"github.com/jackc/pgx/v5/pgtype"
"golangTest/golangTest"
"net/http"
)

// User structure
type User struct {
ID int `json:"id"`
Name string `json:"name"`
PhoneNumber string `json:"phone_number"`
OTP string `json:"otp"`
OTPExpiration string `json:"otp_expiration"`
}

var ctx = context.Background()
var db golangTest.DBTX

func main() {
var err error
// Connecting to the database
db, err := pgx.Connect(ctx, "user=faraz dbname=test sslmode=verify-full")
if err != nil {

}
defer db.Close(ctx)

router := gin.Default()

// Creating routes using gin
router.POST("/api/users", createUser)
router.POST("/api/users/generateotp", generateOTP)
router.POST("/api/users/verifyotp", verifyOTP)
```

```go
router.Run(":8080")
}

func createUser(c *gin.Context) {
var newUser User
// Parsing the JSON body into the newUser struct
if err := c.BindJSON(&newUser); err != nil {
c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
return
}

// Checking if the phone number already exists in the database
var exists bool
queries := golangTest.New(db)
exists, err := queries.CheckPhoneExistence(ctx, newUser.PhoneNumber)
if err != nil {
c.JSON(http.StatusInternalServerError, gin.H{"error": "Database error"})
return
}

if exists {
c.JSON(http.StatusBadRequest, gin.H{"error": "Phone number already in use"})
return
}

// Inserting the new user into the database
err = queries.CreateUser(ctx, golangTest.CreateUserParams{
Name: newUser.Name,
PhoneNumber: newUser.PhoneNumber,
})
if err != nil {
c.JSON(http.StatusInternalServerError, gin.H{"error": "Unable to create user"})
return
}

c.JSON(http.StatusCreated, gin.H{"message": "User created successfully"})
}

func generateOTP(c *gin.Context) {
var User User
// Parsing the JSON body to get the user's phone number
if err := c.BindJSON(&User); err != nil {
c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
return
}

// Checking if the phone number exists in the database
var exists bool
queries := golangTest.New(db)
exists, err := queries.CheckPhoneExistence(ctx, User.PhoneNumber)
if err != nil {
```

```
c.JSON(http.StatusInternalServerError, gin.H{"error": "Database error"})
return
}

if !exists {
c.JSON(http.StatusNotFound, gin.H{"error": "User not found"})
return
}

// Generating a random 4-digit OTP
otp := fmt.Sprintf("%04d", rand.Intn(10000))

// Setting OTP expiration time (1 minute from now)
expirationTime := time.Now().Add(time.Minute)

// Updating user's OTP in the database
// _, err = queries.UpdateUserOTP("UPDATE users SET otp = $1, otp_expiration_time = $2 WHERE
phone_number = $3", otp, expirationTime, User.PhoneNumber)
err = queries.UpdateUserOTP(ctx, golangTest.UpdateUserOTPParams{
PhoneNumber: User.PhoneNumber,
Otp: pgtype.Text{String: otp},
OtpExpirationTime: pgtype.Timestamp{Time: expirationTime},
})
if err != nil {
c.JSON(http.StatusInternalServerError, gin.H{"error": "Unable to generate OTP"})
return
}

c.JSON(http.StatusOK, gin.H{"message": "OTP generated successfully"})
}

func verifyOTP(c *gin.Context) {
var User User

// Parsing the JSON body to get the phone number and OTP
if err := c.BindJSON(&User); err != nil {
c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
return
}

// Retrieving the OTP and its expiration time from the database
var storedOTP, otpExpiration string
queries := golangTest.New(db)
otpRow, err := queries.GetOTP(ctx, User.PhoneNumber)
if err != nil {
if err == sql.ErrNoRows {
// User with the given phone number not found
c.JSON(http.StatusNotFound, gin.H{"error": "User not found"})
} else {
// Database error
c.JSON(http.StatusInternalServerError, gin.H{"error": "Database error"})
}
```

```go
return
}
otpRow.Otp.Scan(&storedOTP)
otpRow.OtpExpirationTime.Scan(&otpExpiration)

// Checking if the OTP is correct
if storedOTP != User.OTP {
c.JSON(http.StatusUnauthorized, gin.H{"error": "Incorrect OTP"})
return
}

// Checking if the OTP has expired
expirationTime, _ := time.Parse(time.RFC3339, otpExpiration)
if time.Now().After(expirationTime) {
c.JSON(http.StatusUnauthorized, gin.H{"error": "OTP has expired"})
return
}

// OTP is correct and not expired
c.JSON(http.StatusOK, gin.H{"message": "OTP verified successfully"})
}
```

Question 2:

```go
package main

import (
"container/heap"
"fmt"
)

func main() {
fmt.Println(rearrangeString("aab")) // Example 1
fmt.Println(rearrangeString("aaab")) // Example 2
}

type CharFrequency struct {
char rune
count int
}

type MaxHeap []CharFrequency

func (h MaxHeap) Len() int { return len(h) }
func (h MaxHeap) Less(i, j int) bool { return h[i].count > h[j].count }
func (h MaxHeap) Swap(i, j int) { h[i], h[j] = h[j], h[i] }

func (h *MaxHeap) Push(x interface{}) {
*h = append(*h, x.(CharFrequency))
}

func (h *MaxHeap) Pop() interface{} {
```

```
old := *h
n := len(old)
x := old[n-1]
*h = old[0 : n-1]
return x
}

func rearrangeString(s string) string {
// Frequency map
frequencyMap := make(map[rune]int)
for _, ch := range s {
frequencyMap[ch]++
}

// Create and populate the max heap
maxHeap := &MaxHeap{}
heap.Init(maxHeap)
for ch, count := range frequencyMap {
heap.Push(maxHeap, CharFrequency{ch, count})
}

var result []rune
var prev CharFrequency

for maxHeap.Len() > 0 {
current := heap.Pop(maxHeap).(CharFrequency)
result = append(result, current.char)
current.count--

if prev.count > 0 {
heap.Push(maxHeap, prev)
}

prev = current

// If only one type of character is left and it's frequency is more than 1, it's not
possible to rearrange
if maxHeap.Len() == 1 && (*maxHeap)[0].count > 1 {
return ""
}
}

return string(result)
}
```

Question 3:

```
package main

import (
"context"
```

```go
"fmt"
"log"
"os"

"github.com/jackc/pgx/v5" // pgx PostgreSQL driver
)

type Seat struct {
ID int
Student string
}

func main() {
// Database connection string
postgresUrl := "postgres://faraz:A1s2d3f4.@localhost:5432/Seat"
db, err := pgx.Connect(context.Background(), os.Getenv(postgresUrl))
if err != nil {
log.Fatalf("Unable to connect to database: %v\n", err)
}
defer db.Close(context.Background())

// Execute the query
rows, err := db.Query(context.Background(), "SELECT CASE WHEN MOD(id, 2) = 0 THEN id - 1
WHEN id = (SELECT MAX(id) FROM Seat) AND MOD(id, 2) = 1 THEN id ELSE id + 1 END AS id,
student FROM Seat ORDER BY id")
if err != nil {
log.Fatalf("Query failed: %v\n", err)
}
defer rows.Close()

// Iterate through the result set
for rows.Next() {
var seat Seat
if err := rows.Scan(&seat.ID, &seat.Student); err != nil {
log.Fatalf("Query scan failed: %v\n", err)
}
fmt.Printf("ID: %d, Student: %s\n", seat.ID, seat.Student)
}

// Check for errors from iterating over rows
if err := rows.Err(); err != nil {
log.Fatalf("Error during rows iteration: %v\n", err)
}
}
```

Question 4:

```go
package main

import (
"fmt"
"math/rand"
```

```go
"sync"
"time"
)

const BufferSize = 10

var (
buffer = make([]byte, BufferSize)
rwMutex = sync.RWMutex{}
)

func main() {
var M, N int
// Example: M = 8, N = 2
M, N = 8, 2
startRoutines(M, N)

// M, N = 8, 8
// startRoutines(M, N)

// M, N = 8, 16
// startRoutines(M, N)

// M, N = 2, 8
// startRoutines(M, N)

select {} // Keep the main goroutine running
}

func startRoutines(M, N int) {
for i := 0; i < M; i++ {
go readBuffer(i)
}
for i := 0; i < N; i++ {
go writeBuffer(i)
}
}

func readBuffer(id int) {
for {
rwMutex.RLock() // Acquire the read lock
fmt.Printf("Reader %d: Reading data: %v\n", id, buffer)
rwMutex.RUnlock() // Release the read lock

time.Sleep(time.Duration(rand.Intn(1000)) * time.Millisecond)
}
}

func writeBuffer(id int) {
for {
rwMutex.Lock() // Acquire the write lock
byteToWrite := byte(rand.Intn(256))
buffer[rand.Intn(BufferSize)] = byteToWrite
fmt.Printf("Writer %d: Writing data: %d\n", id, byteToWrite)
rwMutex.Unlock() // Release the write lock

time.Sleep(time.Duration(rand.Intn(1000)) * time.Millisecond)
}
}
```