```go
package main

import (
	"context"
	"fmt"
	"log"
	"net"
	"net/rpc"
	"os"
	"time"

	"go.mongodb.org/mongo-driver/bson"
	"go.mongodb.org/mongo-driver/mongo"
	"go.mongodb.org/mongo-driver/mongo/options"

	"Lab5/models"
)

const (
	database   = "bank"
	collection = "accounts"
)

type Server struct {
	client    *mongo.Client
	accountID string
	prepared  map[string]int // Lưu trạng thái các giao dịch đang chờ commit
}

// Phase 1: Prepare - Kiểm tra số dư và "giữ" tiền trước khi commit
func (s *Server) PrepareTransaction(req models.TransactionRequest, res *models.TransactionResponse) error {
	log.Printf("[Server %s] Preparing transaction: %s → %s, Amount: %d", s.accountID, req.From, req.To, req.Amount)

	ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
	defer cancel()

	coll := s.client.Database(database).Collection(collection)

	// Kiểm tra số dư tài khoản nguồn
	if req.From == s.accountID {
		var sender struct {
			Balance int `bson:"balance"`
		}
		err := coll.FindOne(ctx, bson.M{"_id": req.From}).Decode(&sender)
		if err != nil {
			res.Success = false
			res.Message = "Source account not found"
```

```
48                          log.Printf("[Server %s] ERROR: Source account not
found!", s.accountID)
49                          return err
50                  }
51                  log.Printf("[Server %s] Current balance of %s: %d",
s.accountID, req.From, sender.Balance)
52
53                  if sender.Balance < req.Amount {
54                          res.Success = false
55                          res.Message = "Insufficient balance"
56                          log.Printf("[Server %s] ERROR: Insufficient balance
in %s", s.accountID, req.From)
57                          return fmt.Errorf("insufficient balance")
58                  }
59          }
60
61          // Đánh dấu giao dịch đã được chuẩn bị nhưng chưa commit
62          s.prepared[req.From] = req.Amount
63          log.Printf("[Server %s] Transaction prepared: %s → %s, Amount: %d",
s.accountID, req.From, req.To, req.Amount)
64
65          res.Success = true
66          res.Message = "Transaction prepared successfully"
67          return nil
68  }
69
70  // Phase 2: Commit - Thực hiện giao dịch nếu tất cả các bên đều chuẩn bị xong
71  func (s *Server) CommitTransaction(req models.TransactionRequest, res
*models.TransactionResponse) error {
72          log.Printf("[Server %s] Committing transaction: %s → %s, Amount: %d",
s.accountID, req.From, req.To, req.Amount)
73
74          ctx, cancel := context.WithTimeout(context.Background(),
10*time.Second)
75          defer cancel()
76
77          coll := s.client.Database(database).Collection(collection)
78
79          // Thực hiện giao dịch
80          _, err := coll.UpdateOne(ctx, bson.M{"_id": req.From}, bson.M{"$inc":
bson.M{"balance": -req.Amount}})
81          if err ≠ nil {
82                  res.Success = false
83                  res.Message = "Failed to debit source account"
84                  log.Printf("[Server %s] ERROR: Failed to debit %s",
s.accountID, req.From)
85                  return err
86          }
87
88          _, err = coll.UpdateOne(ctx, bson.M{"_id": req.To}, bson.M{"$inc":
bson.M{"balance": req.Amount}})
89          if err ≠ nil {
90                  // Nếu cộng tiền vào tài khoản đích thất bại, rollback ngay
91                  log.Printf("[Server %s] ERROR: Failed to credit %s, rolling
back...", s.accountID, req.To)
```

```
92                  s.RollbackTransaction(req, res)
93                  return err
94          }
95
96          // Xóa trạng thái chuẩn bị
97          delete(s.prepared, req.From)
98
99          res.Success = true
100         res.Message = "Transaction committed successfully"
101         log.Printf("[Server %s] SUCCESS: Transaction committed!",
    s.accountID)
102         return nil
103 }
104
105 // Phase 2 (Fail Case): Rollback nếu Commit thất bại
106 func (s *Server) RollbackTransaction(req models.TransactionRequest, res
    *models.TransactionResponse) error {
107         log.Printf("[Server %s] Rolling back transaction: %s → %s, Amount:
    %d", s.accountID, req.From, req.To, req.Amount)
108
109         ctx, cancel := context.WithTimeout(context.Background(),
    10*time.Second)
110         defer cancel()
111
112         coll := s.client.Database(database).Collection(collection)
113
114         // Kiểm tra xem giao dịch có trong danh sách `prepared` không
115         if amount, exists := s.prepared[req.From]; exists {
116                 // Hoàn trả số tiền đã trừ trước đó
117                 _, err := coll.UpdateOne(ctx, bson.M{"_id": req.From},
    bson.M{"$inc": bson.M{"balance": amount}})
118                 if err ≠ nil {
119                         res.Success = false
120                         res.Message = "Rollback failed"
121                         log.Printf("[Server %s] ERROR: Rollback failed for
    %s", s.accountID, req.From)
122                         return err
123                 }
124
125                 // Xóa khỏi danh sách `prepared`
126                 delete(s.prepared, req.From)
127                 log.Printf("[Server %s] Rollback successful!", s.accountID)
128
129                 res.Success = true
130                 res.Message = "Transaction rolled back successfully"
131         } else {
132                 log.Printf("[Server %s] WARNING: No prepared transaction
    found to rollback!", s.accountID)
133                 res.Success = false
134                 res.Message = "No prepared transaction found"
135         }
136         return nil
137 }
138
139 func startServer(mongoURI, port, accountID string) {
```

```
140         listener, err := net.Listen("tcp", ":"+port)
141         if err ≠ nil {
142                 log.Fatalf("Failed to listen on port %s: %v", port, err)
143         }
144
145         server := &Server{accountID: accountID, prepared:
    make(map[string]int)}
146         server.client, err = mongo.Connect(context.Background(),
    options.Client().ApplyURI(mongoURI))
147         if err ≠ nil {
148                 log.Fatalf("Failed to connect to MongoDB: %v", err)
149         }
150         defer server.client.Disconnect(context.Background())
151
152         rpcServer := rpc.NewServer()
153         rpcServer.Register(server)
154
155         log.Printf("[Server %s] Running on port %s, connected to MongoDB at
    %s", accountID, port, mongoURI)
156         for {
157                 conn, err := listener.Accept()
158                 if err ≠ nil {
159                         log.Println("[Server] Connection error:", err)
160                         continue
161                 }
162                 go rpcServer.ServeConn(conn)
163         }
164 }
165
166 func main() {
167         if len(os.Args) < 4 {
168                 log.Fatalf("Usage: go run server.go <MongoDB_URI> <Port>
    <AccountID>")
169         }
170
171         mongoURI := os.Args[1]
172         port := os.Args[2]
173         accountID := os.Args[3]
174
175         startServer(mongoURI, port, accountID)
176 }
177
```