APPENDIX

1. gRPC CODE

a. CLIENT CODE

IMPORT CODE

```
package main

import (
"context"
"encoding/json"
"github.com/labstack/echo"
"google.golang.org/grpc"
"log"
"net/http"
"os"
ts "project/transactions-service-grpc/proto"
"project/transactions-service-
grpc/transaksiServer/transaksiRepository"
"time"
)
```

TRX SERVICE

```
const (address = "localhost:50051")

func Trx(transaksi transaksiRepository.RequestTransaksi) bool {
     conn, err := grpc.Dial(address, grpc.WithInsecure(),
grpc.WithBlock())

if err != nil {
log.Fatal("Cannot Connected:", err.Error())
return false
}
defer conn.Close()

c := ts.NewInquiryTransactionClient(conn)
var responseTxt []string
requestInput := ts.RequestTransactions{
     DocNo:                transaksi.DocNo,
     PaymentMethodId:      transaksi.PaymentMethodId,
     TotalPrice:           float32(transaksi.TotalPrice),
     InitialStatusTrx:     int32(transaksi.InitialStatusTrx),
          StatusTrx:            int32(transaksi.StatusTrx),
          ResponseDesc:         transaksi.ResponseDesc,
```

```
            StatusSettlement:     transaksi.StatusSettlement,
            CorporateId:          transaksi.CorporateId,
            PaymentVendorId:      transaksi.PaymentMethodId,
            TimeTransactionStart: transaksi.TimeTransactionStart,
            MerchantKey:          transaksi.MerchantKey,
            DeviceId:             transaksi.DeviceId,
            TotalQuantity:        transaksi.TotalQuantity,
            UuidCard:             transaksi.UuidCard,
            CardNumber:           transaksi.CardNumber,
            CreatedAt:            transaksi.CreatedAt,
            TimeStart:            time.Now().Format("2006-01-02
15:04:05.000000 MST"),
}

responseInquiry, err :=
c.InquiryTransaction(context.Background(), &requestInput)
if err != nil {
log.Fatal("InquiryTransaction Failed: ", err)
return false
}

Log.Println(responseInquiry)
responseMarshall, _ := json.Marshal(responseInquiry)
responseTxt = append(responseTxt, string(responseMarshall))

var file, errorTxt =
os.OpenFile("/Users/hanserhanto/Documents/Project/go/src/grpc-
skripsi/grpc/transaksiClient/responseTxt.txt", os.O_RDWR, 0644)
if errorTxt != nil {
log.Fatal("Open File Txt Error:", errorTxt)
return false
}

defer file.Close()
responseTxtMarshall, _ := json.Marshal(responseTxt)
_, errorWrite := file.Write(responseTxtMarshall)
if errorWrite != nil {
log.Fatal("Write File Txt Error:", errorWrite)
return false
}
errorSync := file.Sync()
if errorSync != nil {
log.Fatal("Sync File Txt Error:", errorSync)
return false
}
return true
}
```

MAIN

```
func main() {
e := echo.New()
e.POST("/grpc/skripsi/trx-grpc", func(c echo.Context) error {
u := new(transaksiRepository.RequestTransaksi)
if err := c.Bind(u); err != nil {
return c.JSON(http.StatusOK, err.Error())
}
result := Trx(*u)
return c.JSON(http.StatusOK, result)
})
e.Logger.Fatal(e.Start(":" + "50053"))
}
```

b. SERVER CODE

CONNECT TO DB

```
func setupConnection() (*sql.DB, error) {
var connection = fmt.Sprintf("user=%s password=%s dbname=%s
host=%s port=%s sslmode=%s",
"hanserhanto", "MKPmobile123@", "grpc_server", "localhost",
"5432", "disable")
fmt.Println("Connection Info:", "postgres", connection)

db, err := sql.Open("postgres", connection)
if err != nil {
return db, errors.New("Connection closed: Failed Connect
Database")
}

db.SetMaxOpenConns(10)
db.SetMaxIdleConns(10)
db.SetConnMaxLifetime(10 * time.Second)
return db, nil

}
```

TRANSACTION STRUCT

```
type RequestInquiryTransaksi struct {
DocNo                string   `json:"docNo"`
PaymentMethodId      int64    `json:"paymentMethodId"`
TotalPrice           float32  `json:"totalPrice"`
InitialStatusTrx     int32    `json:"initialStatusTrx"`
StatusTrx            int32    `json:"statusTrx"`
ResponseDesc         string   `json:"responseDesc"`
StatusSettlement     bool     `json:"statusSettlement"`
CorporateId          int64    `json:"corporateId"`
PaymentVendorId      int64    `json:"paymentVendorId"`
TimeTransactionStart string   `json:"timeTransactionStart"`
MerchantKey          string   `json:"merchantKey"`
DeviceId             string   `json:"deviceId"`
TotalQuantity        int64    `json:"totalQuantity"`
UUIDCard             string   `json:"uuidCard"`
CardNumber           string   `json:"cardNumber"`
CreatedAt            string   `json:"createdAt"`
CpuUsage             int64    `json:"cpuUsage"`
MemoriUsage          int64    `json:"memoriUsage"`
Ms                   int64    `json:"ms"`
}
```

```
type RequestTransaksi struct {
DocNo                 string  `json:"docNo"`
PaymentMethodId       int64   `json:"paymentMethodId"`
TotalPrice            float64 `json:"totalPrice"`
InitialStatusTrx      int64   `json:"initialStatusTrx"`
StatusTrx             int64   `json:"statusTrx"`
ResponseDesc          string  `json:"responseDesc"`
ResponseDatetime      string  `json:"responseDatetime"`
StatusSettlement      bool    `json:"statusSettlement"`
CorporateId           int64   `json:"corporateId"`
PaymentVendorId       int64   `json:"paymentVendorId"`
TimeTransactionStart string  `json:"timeTransactionStart"`
MerchantKey           string  `json:"merchantKey"`
DeviceId              string  `json:"deviceId"`
TotalQuantity         int64   `json:"totalQuantity"`
UuidCard              string  `json:"uuidCard"`
CardNumber            string  `json:"cardNumber"`
CreatedAt             string  `json:"createdAt"`
}
```

TRANSACTION REPOSITORY

```
func (ctx trxRepository) InsertInquiryTransaksi(input
transaksiRepository.RequestInquiryTransaksi) (*int64, error) {
var id int64
query := `INSERT INTO transactions(
doc_no, paymentmethod_id, total_price, initial_status_trx,
status_trx,

response_desc, response_datetime, status_settlement,
corporate_id, paymentvendor_id,

time_transaction_start, time_transaction_end, merchant_key,
device_id, total_quantity,

uuid_card, card_number, created_at, cpu_usage, memori_usage, ms
) VALUES (
$1, $2, $3, $4, $5,
$6, $7, $8, $9, $10,
$11, $12, $13, $14, $15,
$16, $17, $18, $19, $20, $21
)RETURNING id`
```

```
err := ctx.RepoDB.DB.QueryRowContext(context.Background(),
query, input.DocNo, input.PaymentMethodId, input.TotalPrice,
input.InitialStatusTrx, input.StatusTrx,
input.ResponseDesc, "", input.StatusSettlement,
input.CorporateId, input.PaymentVendorId,
input.TimeTransactionStart, "", input.MerchantKey,
input.DeviceId, input.TotalQuantity,
input.UUIDCard, input.CardNumber, input.CreatedAt,
input.CpuUsage, input.MemoriUsage, input.Ms).Scan(&id)

if err != nil {
return nil, err
}
return &id, nil
}

func (ctx trxRepository) UpdateMs(ms, id int64) error {

query := `UPDATE transactions set
ms = $1 WHERE id= $2`
_, err := ctx.RepoDB.DB.ExecContext(context.Background(), query,
ms, id)
if err != nil {
return err
}

return nil
}
```

TRANSACTIONS SERVICE

```
func (svc transactionService) InquiryTransaction(ctx
context.Context, input *ts.RequestTransactions)
(*ts.ResponseInquiryTransaction, error) {
_, _ = metadata.FromIncomingContext(ctx)
var result ts.ResponseInquiryTransaction

memory, _ := mem.VirtualMemory()
cpu, _ := cpu.Percent(time.Second, false)
memoryUsage := int(math.Ceil(memory.UsedPercent))
cpuUsage := int(math.Ceil(cpu[0]))

requestInquiry := transaksiRepository.RequestInquiryTransaksi{
     DocNo:                input.DocNo,
     PaymentMethodId:      input.PaymentMethodId,
     TotalPrice:           input.TotalPrice,
     InitialStatusTrx:     input.InitialStatusTrx,
```

```
        StatusTrx:              input.StatusTrx,
        ResponseDesc:           input.ResponseDesc,
        StatusSettlement:       input.StatusSettlement,
        CorporateId:            input.CorporateId,
        PaymentVendorId:        input.PaymentVendorId,
        TimeTransactionStart:   input.TimeTransactionStart,
        MerchantKey:            input.MerchantKey,
        DeviceId:               input.DeviceId,
        TotalQuantity:          input.TotalQuantity,
        UUIDCard:               input.UuidCard,
        CardNumber:             input.CardNumber,
        CreatedAt:              input.CreatedAt,
        CpuUsage:               int64(cpuUsage),
        MemoriUsage:            int64(memoryUsage),
}

id, err :=
svc.Service.TrxRepo.InsertInquiryTransaksi(requestInquiry)
if err != nil {
result.Result = false
result.Success = false
result.ResponseDatetime = time.Now().Format("2006-01-02
15:04:05")
result.StatusCode = "204"
result.Msg = err.Error()
result.TimeEnd = 0
return &result, err
}

timeCreate, _ := time.Parse("2006-01-02 15:04:05.000000 MST",
input.TimeStart)
ms := time.Since(timeCreate).Milliseconds()
result.TimeEnd = ms

err = svc.Service.TrxRepo.UpdateMs(ms, *id)
if err != nil {
result.Result = false
result.Success = false
result.ResponseDatetime = time.Now().Format("2006-01-02
15:04:05")
result.StatusCode = "204"
result.Msg = err.Error()
result.TimeEnd = 0
return &result, err
}

result.Result = true
```

```
result.Success = true
result.ResponseDatetime = time.Now().Format("2006-01-02
15:04:05")
result.StatusCode = "200"
result.Msg = ""
result.TimeEnd = ms

return &result, nil
}
```

MAIN

```
const (port = ":50051")

func main() {
lis, err := net.Listen("tcp", port)
if err != nil {
log.Fatal("failed to listen")
}

config.OpenConnection()
defer config.CloseConnectionDB()

db := config.GetDB()

repo := transaksiRepository.NewRepository(db,
context.Background())
services := app.SetupApp(db, repo)

s := grpc.NewServer()
trxSvc := trxService.NewTransactionService(services)
proto.RegisterInquiryTransactionServer(s, trxSvc)
log.Println("server listening at", lis.Addr())

if err := s.Serve(lis); err != nil {
log.Fatal("failed to serve:", err)
}
}
```

C. PROTO BUFFER

```
syntax = "proto3";
package transaksi;

message RequestTransactions{
string    docNo = 1;
int64     paymentMethodId = 2;
float     totalPrice = 3;
int32     initialStatusTrx = 4;
int32     statusTrx = 5;
string    responseDesc = 6;
string    responseDatetime = 7;
bool      statusSettlement = 8;
int64     corporateId = 9;
int64     paymentVendorId = 10;
string    timeTransactionStart = 11;
string    merchantKey = 12;
string    deviceId = 13;
int64     totalQuantity = 14;
string    uuidCard = 15;
string    cardNumber = 16;
string    createdAt = 17;
string    timeStart = 18;
}

message ResponseInquiryTransaction{
string    statusCode = 1;
bool      success = 2;
string    responseDatetime = 3;
bool      result = 4;
string    msg = 5;
int64     timeEnd = 6;
}

service InquiryTransaction{
rpc InquiryTransaction(RequestTransactions) returns
(ResponseInquiryTransaction);
}

option go_package = "/;proto";
```

2. REST API

a. CLIENT CODE

IMPORT CODE

```
package main

import (
"bytes"
"crypto/tls"
"encoding/json"
"github.com/labstack/echo"
"io/ioutil"
"log"
"net/http"
"os"
"project/transactions-service-
restapi/transaksiServer/transaksiRepository"
"time"
)
```

TRANSACTIONS SERVICE

```
type RequestTransaksi struct {
DocNo                string   `json:"docNo"`
PaymentMethodId      int64    `json:"paymentMethodId"`
TotalPrice           float32  `json:"totalPrice"`
InitialStatusTrx     int32    `json:"initialStatusTrx"`
StatusTrx            int32    `json:"statusTrx"`
ResponseDesc         string   `json:"responseDesc"`
StatusSettlement     bool     `json:"statusSettlement"`
CorporateId          int64    `json:"corporateId"`
PaymentVendorId      int64    `json:"paymentVendorId"`
TimeTransactionStart string   `json:"timeTransactionStart"`
MerchantKey          string   `json:"merchantKey"`
DeviceId             string   `json:"deviceId"`
TotalQuantity        int64    `json:"totalQuantity"`
UUIDCard             string   `json:"uuidCard"`
CardNumber           string   `json:"cardNumber"`
CreatedAt            string   `json:"createdAt"`
TimeStart            string   `json:"timeStart"`
}

type ResponseTransaksi struct {
StatusCode       string `json:"statusCode"`
Succes           bool   `json:"success"`
```

```
ResponseDatetime string `json:"responseDatetime"`
Result           bool   `json:"result"`
Msg              string `json:"msg"`
TimeEnd          int64  `json:"timeEnd"`
}

func Trx(transaksi transaksiRepository.RequestTrx) bool {
var responseTxt []string
requestInput := RequestTransaksi{
DocNo:                transaksi.DocNo,
PaymentMethodId:      transaksi.PaymentMethodId,
TotalPrice:           float32(transaksi.TotalPrice),
InitialStatusTrx:     int32(transaksi.InitialStatusTrx),
StatusTrx:            int32(transaksi.StatusTrx),
ResponseDesc:         transaksi.ResponseDesc,
StatusSettlement:     transaksi.StatusSettlement,
CorporateId:          transaksi.CorporateId,
PaymentVendorId:      transaksi.PaymentVendorId,
TimeTransactionStart: transaksi.TimeTransactionStart,
MerchantKey:          transaksi.MerchantKey,
DeviceId:             transaksi.DeviceId,
TotalQuantity:        transaksi.TotalQuantity,
UUIDCard:             transaksi.UuidCard,
CardNumber:           transaksi.CardNumber,
CreatedAt:            transaksi.CreatedAt,
TimeStart:            time.Now().Format("2006-01-02
15:04:05.000000 MST"),
}

bodyRequest, err := json.Marshal(requestInput)
if err != nil {
log.Fatal("error to Marshals - ", err.Error())
return false
}

httpreq, err := http.NewRequest("POST",
"http://localhost:50051/restapi/skripsi/transactions",
bytes.NewBuffer(bodyRequest))
if err != nil {
log.Fatal("error to newrequest - ", err.Error())
return false
}
defer httpreq.Body.Close()
httpreq.Close = true
httpreq.Header.Add("Content-Type", "application/json")
httpreq.Header.Set("Connection", "close")
```

```
tr := &http.Transport{
TLSClientConfig: &tls.Config{InsecureSkipVerify: true},
}

client := &http.Client{Transport: tr}
resp, err := client.Do(httpreq)
if err != nil {
log.Fatal("error client do - ", err.Error())
return false
}

defer resp.Body.Close()
body, err := ioutil.ReadAll(resp.Body)
if err != nil {
log.Fatal("error read response body - ", err.Error())
return false
}

responseRequest := ResponseTransaksi{}
err = json.Unmarshal(body, &responseRequest)

log.Println(responseRequest)
responseMarshall, _ := json.Marshal(responseRequest)
responseTxt = append(responseTxt, string(responseMarshall))

var file, errorTxt =
os.OpenFile("/Users/hanserhanto/Documents/Project/go/src/restapi
-skripsi/restapi/transaksiClient/responseTxt.txt", os.O_RDWR,
0644)
if errorTxt != nil {
log.Fatal("Open File Txt Error:", errorTxt)
return false
}

defer file.Close()
responseTxtMarshall, _ := json.Marshal(responseTxt)
_, errorWrite := file.Write(responseTxtMarshall)
if errorWrite != nil {
log.Fatal("Write File Txt Error:", errorWrite)
return false
}
errorSync := file.Sync()
if errorSync != nil {
log.Fatal("Sync File Txt Error:", errorSync)
return false
}
return true}
```

MAIN

```
func main() {

e := echo.New()

e.POST("/restapi/skripsi/trx", func(c echo.Context) error {
u := new(transaksiRepository.RequestTrx)
if err := c.Bind(u); err != nil {
return c.JSON(http.StatusOK, err.Error())
}
result := Trx(*u)
return c.JSON(http.StatusOK, result)
})

e.Logger.Fatal(e.Start(":" + "50052"))}
```

b. SERVER CODE

CONNECT TO DB

```
func setupConnection() (*sql.DB, error) {
var connection = fmt.Sprintf("user=%s password=%s dbname=%s
host=%s port=%s sslmode=%s",
"hanserhanto", "MKPmobile123@", "restapi_server", "localhost",
"5432", "disable")
fmt.Println("Connection Info:", "postgres", connection)

db, err := sql.Open("postgres", connection)
if err != nil {
return db, errors.New("Connection closed: Failed Connect
Database")
}

db.SetMaxOpenConns(10)
db.SetMaxIdleConns(10)
db.SetConnMaxLifetime(10 * time.Second)
return db, nil
}
```

TRANSACTION STRUCT

```
type RequestInquiryTransaksi struct {
DocNo                string  `json:"docNo"`
PaymentMethodId      int64   `json:"paymentMethodId"`
TotalPrice           float32 `json:"totalPrice"`
InitialStatusTrx     int32   `json:"initialStatusTrx"`
StatusTrx            int32   `json:"statusTrx"`
ResponseDesc         string  `json:"responseDesc"`
StatusSettlement     bool    `json:"statusSettlement"`
CorporateId          int64   `json:"corporateId"`
PaymentVendorId      int64   `json:"paymentVendorId"`
TimeTransactionStart string  `json:"timeTransactionStart"`
MerchantKey          string  `json:"merchantKey"`
DeviceId             string  `json:"deviceId"`
TotalQuantity        int64   `json:"totalQuantity"`
UUIDCard             string  `json:"uuidCard"`
CardNumber           string  `json:"cardNumber"`
CreatedAt            string  `json:"createdAt"`
CpuUsage             int64   `json:"cpuUsage"`
MemoriUsage          int64   `json:"memoriUsage"`
Ms                   int64   `json:"ms"`
}
```

```
type RequestTransaksi struct {
DocNo                 string  `json:"docNo"`
PaymentMethodId       int64   `json:"paymentMethodId"`
TotalPrice            float32 `json:"totalPrice"`
InitialStatusTrx      int32   `json:"initialStatusTrx"`
StatusTrx             int32   `json:"statusTrx"`
ResponseDesc          string  `json:"responseDesc"`
StatusSettlement      bool    `json:"statusSettlement"`
CorporateId           int64   `json:"corporateId"`
PaymentVendorId       int64   `json:"paymentVendorId"`
TimeTransactionStart string  `json:"timeTransactionStart"`
MerchantKey           string  `json:"merchantKey"`
DeviceId              string  `json:"deviceId"`
TotalQuantity         int64   `json:"totalQuantity"`
UUIDCard              string  `json:"uuidCard"`
CardNumber            string  `json:"cardNumber"`
CreatedAt             string  `json:"createdAt"`
TimeStart             string  `json:"timeStart"`
}

type RequestTrx struct {
DocNo                 string  `json:"docNo"`
PaymentMethodId       int64   `json:"paymentMethodId"`
TotalPrice            float64 `json:"totalPrice"`
InitialStatusTrx      int64   `json:"initialStatusTrx"`
StatusTrx             int64   `json:"statusTrx"`
ResponseDesc          string  `json:"responseDesc"`
ResponseDatetime      string  `json:"responseDatetime"`
StatusSettlement      bool    `json:"statusSettlement"`
CorporateId           int64   `json:"corporateId"`
PaymentVendorId       int64   `json:"paymentVendorId"`
TimeTransactionStart string  `json:"timeTransactionStart"`
MerchantKey           string  `json:"merchantKey"`
DeviceId              string  `json:"deviceId"`
TotalQuantity         int64   `json:"totalQuantity"`
UuidCard              string  `json:"uuidCard"`
CardNumber            string  `json:"cardNumber"`
CreatedAt             string  `json:"createdAt"`
}
```

TRANSACTIONS REPOSITORY

```
func (ctx trxRepository) InsertInquiryTransaksi(input
transaksiRepository.RequestInquiryTransaksi) (*int64, error) {

var id int64

query := `INSERT INTO transactions(doc_no, paymentmethod_id,
total_price, initial_status_trx, status_trx, response_desc,
response_datetime, status_settlement, corporate_id,
paymentvendor_id, time_transaction_start, time_transaction_end,
merchant_key, device_id, total_quantity, uuid_card, card_number,
created_at, cpu_usage, memori_usage, ms ) VALUES (

$1, $2, $3, $4, $5,

$6, $7, $8, $9, $10,

$11, $12, $13, $14, $15,

$16, $17, $18, $19, $20, $21

)

RETURNING id`

err := ctx.RepoDB.DB.QueryRowContext(context.Background(),
query, input.DocNo, input.PaymentMethodId, input.TotalPrice,
input.InitialStatusTrx, input.StatusTrx,


input.ResponseDesc, "", input.StatusSettlement,
input.CorporateId, input.PaymentVendorId,


input.TimeTransactionStart, "", input.MerchantKey,
input.DeviceId, input.TotalQuantity,
```

```
input.UUIDCard, input.CardNumber, input.CreatedAt,
input.CpuUsage, input.MemoriUsage, input.Ms).Scan(&id)
if err != nil {
return nil, err
}
return &id, nil
}
func (ctx trxRepository) UpdateMs(ms, id int64) error {
query := `UPDATE transactions set
ms = $1 WHERE id= $2`
_, err := ctx.RepoDB.DB.ExecContext(context.Background(), query,
ms, id)
if err != nil {
return err
}
return nil
}
```

TRANSACTIONS SERVICE

```
func (svc transactionService) InquiryTransaction(input
transaksiRepository.RequestTransaksi)
*transaksiRepository.ResponseTransaksi {

var result transaksiRepository.ResponseTransaksi

memory, _ := mem.VirtualMemory()

cpu, _ := cpu.Percent(time.Second, false)

memoryUsage := int(math.Ceil(memory.UsedPercent))

cpuUsage := int(math.Ceil(cpu[0]))


requestInquiry := transaksiRepository.RequestInquiryTransaksi{
      DocNo:                input.DocNo,
      PaymentMethodId:      input.PaymentMethodId,
      TotalPrice:           input.TotalPrice,
      InitialStatusTrx:     input.InitialStatusTrx,
      StatusTrx:            input.StatusTrx,
      ResponseDesc:         input.ResponseDesc,
      StatusSettlement:     input.StatusSettlement,
      CorporateId:          input.CorporateId,
      PaymentVendorId:      input.PaymentVendorId,
      TimeTransactionStart: input.TimeTransactionStart,
      MerchantKey:          input.MerchantKey,
      DeviceId:             input.DeviceId,
      TotalQuantity:        input.TotalQuantity,
      UUIDCard:             input.UUIDCard,
```

```
        CardNumber:             input.CardNumber,
        CreatedAt:              input.CreatedAt,
        CpuUsage:               int64(cpuUsage),
        MemoriUsage:            int64(memoryUsage),
        }

id, err :=
svc.Service.TrxRepo.InsertInquiryTransaksi(requestInquiry)
if err != nil {
result.Result = false
result.Succes = false
result.ResponseDatetime = time.Now().Format("2006-01-02
15:04:05")
result.StatusCode = "204"
result.Msg = err.Error()
result.TimeEnd = 0
return &result
}

timeCreate, _ := time.Parse("2006-01-02 15:04:05.000000 MST",
input.TimeStart)
ms := time.Since(timeCreate).Milliseconds()
result.TimeEnd = ms

err = svc.Service.TrxRepo.UpdateMs(ms, *id)
```

```
if err != nil {
result.Result = false
result.Succes = false
result.ResponseDatetime = time.Now().Format("2006-01-02
15:04:05")
result.StatusCode = "204"
result.Msg = err.Error()
result.TimeEnd = 0
return &result
}

result.Result = true
result.Succes = true
result.ResponseDatetime = time.Now().Format("2006-01-02
15:04:05")
result.StatusCode = "200"
result.Msg = ""
result.TimeEnd = ms
return &result
}
```

MAIN

```
func main() {
e := echo.New()
config.OpenConnection()
defer config.CloseConnectionDB()
```

```
db := config.GetDB()
repo := transaksiRepository.NewRepository(db,
context.Background())
services := app.SetupApp(db, repo)

trxSvc := trxService.NewTransactionService(services)

e.POST("/restapi/skripsi/transactions", func(c echo.Context)
error {
u := new(transaksiRepository.RequestTransaksi)
if err := c.Bind(u); err != nil {
return c.JSON(http.StatusOK, err.Error())
}
result := trxSvc.InquiryTransaction(*u)
return c.JSON(http.StatusOK, result)
})
e.Logger.Fatal(e.Start(":" + "50051"))
}
```

PAPER NAME

17.K1.0028.docx

WORD COUNT

7046 Words

CHARACTER COUNT

34857 Characters

PAGE COUNT

28 Pages

FILE SIZE

168.0KB

SUBMISSION DATE

Jul 21, 2022 10:36 AM GMT+7

REPORT DATE

Jul 21, 2022 10:36 AM GMT+7

1% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 0% Internet database
- Crossref database
- 1% Submitted Works database
- 0% Publications database
- Crossref Posted Content database