**TUGAS BESAR**

**ALGORITMA PEMROGRAMAN**

***E-MONEY***

**KELAS S1-IT-23-002**

**HESMI ARIA YANTI, S.Kom., M.Kom**



**Dibuat Oleh:**

**1. Muhammad Faiz Satrio (103062300086)**

**2. Stiefanny Dwi Chandra (103062300102)**

**PROGRAM STUDI S1 TEKNOLOGI INFORMASI**

**FAKULTAS INFORMATIKA**

**UNIVERSITAS TELKOM**

**JAKARTA**

**2024**

**DAFTAR ISI**

1. Deskripsi Aplikas*i E-Money*.........................................................................................1
2. Tujuan dan Manfaat Aplikasi *E-Money.*.......................................................................1
3. Alur/Tahapan Aplikasi *E-Money*....................................................................................
4. *Script.*..............................................................................................................................
5. Kesimpulan dan Saran....................................................................................................

DAFTAR PUSTAKA.....................................................................................................

1. **Deskripsi Aplikasi *(E-Money)***

*E-money* atau uang elektronik adalah alat pembayaran yang digunakan untuk melakukan transaksi keuangan secara elektronik. Fungsinya adalah sebagai pengganti uang tunai dalam melakukan berbagai pembayaran seperti membeli barang atau jasa, membayar tagihan, dan transfer dana. Aplikasi ini sangat memudahkan masyarakat untuk melakukan berbagai transaksi non-tunai secara digital tanpa harus membawa uang kas. Aplikasi ini dibuat menggunakan bahasa pemrograman *Go Languange.* Beberapa jenis aplikasi *e-money* yang populer di Indonesia antara lain:

1. GoPay (dari Gojek)

2. OVO

3. Dana

4. LinkAja

5. *ShopeePay*, dll

Alat pembayarannya menggunakan uang elektronik yang memenuhi unsur:

1. Diterbitkan atas nilai uang yang disetor terlebih dahulu.

2. Nilai uang disimpan secara elektronik dalam suatu media seperti server atau chip.

3. Digunakan untuk melakukan transaksi pembayaran dengan cara elektronik.

4. Bersifat pengganti uang tunai.

1. **Tujuan dan Manfaat Aplikasi *E-Money***

Menurut laporan dari Asosiasi Penyelenggara Jasa Internet Indonesia (APJII), tujuan dari aplikasi *e-money* adalah:

1. Mendukung pertumbuhan ekonomi digital karena semakin banyaknya transaksi yang dilakukan secara digital, aplikasi *e-money* memfasilitasi perputaran uang dalam ekosistem ekonomi digital yang terus berkembang pesat.

2. Meningkatkan inklusi keuangan karena aplikasi ini mempermudah masyarakat yang belum memiliki rekening bank untuk melakukan transaksi non-tunai, sehingga mendorong inklusi keuangan yang lebih luas. Contoh: membayar tol, transportasi, dll dilakukan dengan cepat dan efisien tanpa harus menyiapkan uang tunai pas.

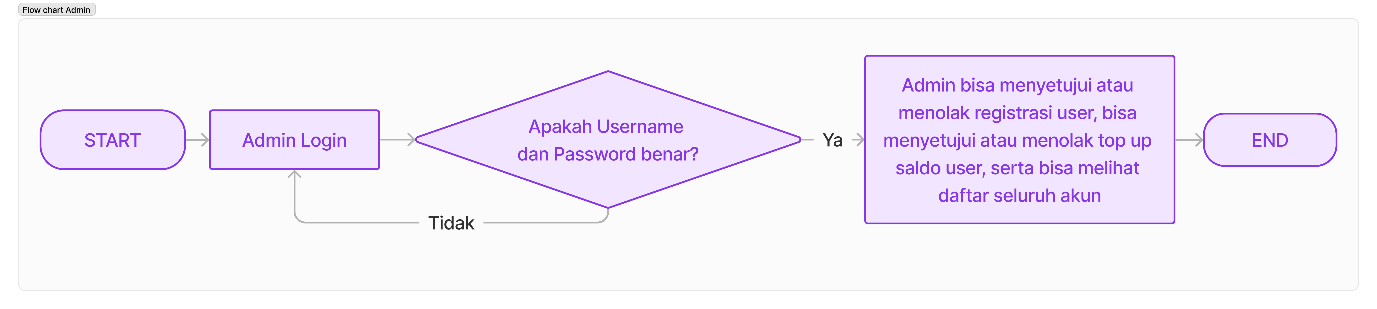
3. Memberikan pengalaman berbelanja yang *seamless* sebab e*-money* memungkinkan pembayaran yang cepat, praktis, dan tanpa perlu membawa uang tunai, sehingga meningkatkan pengalaman berbelanja secara digital.

4. Menghemat biaya pemerintah dengan mengurangi biaya pengelolaan uang tunai yang cukup besar bagi pemerintah dan lembaga keuangan.

5. Mendukung transaksi non-tunai yang lebih aman dibandingkan uang tunai karena mengurangi risiko kehilangan atau pencurian uang tunai.

6. Mempermudah masyarakat dalam melakukan transaksi pembayaran dengan cepat, dan praktis.

1. **Alur/Tahapan Aplikasi *E-Money***



1. Alur Untuk Admin
   1. Admin melakukan login dengan kredensial yang telah tersimpan
   2. Jika kredensial sesuai maka akan diarahkan ke menu Admin dengan list :
      * Approve/Reject Registration

Admin dapat menyetujui atau menerima Registrasi yang dilakukan oleh User

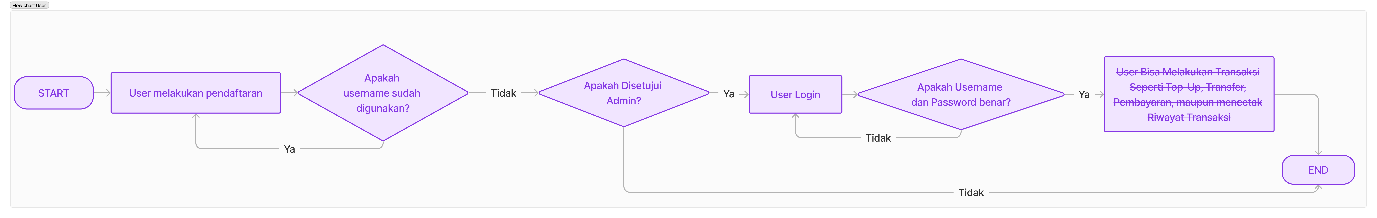
* + - Print Account List

Admin dapat melihat list akun yang terdaftar dalam aplikasi e-money

* + - Approve/Reject Top Up Requests

Admin dapat menyetujui atau menolak permintaan top up saldo dari User

* + - Logout



1. Alur Untuk User
   1. User melakukan registrasi terlebih dahulu
   2. User menunggu registrasi disetujui oleh Admin
   3. Apabila registrasi disetujui, user dapat melakukan login
   4. Jika kredensial sesuai maka akan diarahkan ke menu User dengan list :
      * Check Balance

User dapat mengecek sisa saldo

* + - Transfer Money

User dapat melakukan Transfer kepada akun lain yang terdaftar

* + - Make Payment

User dapat melakukan Pembayaran

* + - Print Transaction History

User dapat melihat histori transaksi

* + - Top Up Balance

User dapat melakukan Top Up Saldo

* + - Logout

1. ***Script***

package *main*

import (

"bufio"

"encoding/json"

"fmt"

"log"

"os"

"strconv"

"sync"

"time"

)

type *Account* struct {

ID *string*

Password *string*

Balance *float64*

Approved *bool*

Transactions []*Transaction*

}

type *Transaction* struct {

ID *int*

AccountID *string*

Type *string*

Amount *float64*

Date *string*

Details *string*

}

type *Registration* struct {

ID *string*

Password *string*

}

type *TopUpRequest* struct {

ID *int*

AccountID *string*

Amount *float64*

Date *string*

Approved *bool*

}

var (

accounts = make(map[*string*]\**Account*)

registrations = make(map[*string*]*Registration*)

topUpRequests = []*TopUpRequest*{}

mu *sync*.*Mutex*

currentUser \**Account*

nextTopUpID = 1

)

const accountsFilePath = "accounts.json"

func main() {

loadAccounts()

defer saveAccounts()

scanner := bufio.NewScanner(os.Stdin)

fmt.Println("Welcome to the e-money system!")

if isEmptyAccountsFile() {

fmt.Println("No accounts found. Please register as admin or register a new account.")

for {

fmt.Println("1. Admin Login")

fmt.Println("2. Register Account")

fmt.Println("3. Exit")

if !scanner.Scan() {

break

}

option := scanner.Text()

switch option {

case "1":

if loginadm(scanner) {

adminMenu(scanner)

}

case "2":

registerAccount(scanner)

case "3":

fmt.Println("Goodbye!")

return

default:

fmt.Println("Invalid option. Please try again.")

}

}

} else {

fmt.Println("Choose an option:")

fmt.Println("1. User Login")

fmt.Println("2. Admin Login")

fmt.Println("3. Register Account")

fmt.Println("4. Exit")

for {

if !scanner.Scan() {

break

}

option := scanner.Text()

switch option {

case "1":

if loginusr(scanner) {

userMenu(scanner)

}

case "2":

if loginadm(scanner) {

adminMenu(scanner)

}

case "3":

registerAccount(scanner)

case "4":

fmt.Println("Goodbye!")

return

default:

fmt.Println("Invalid option. Please try again.")

}

}

}

}

func isEmptyAccountsFile() *bool* {

data, err := os.ReadFile(accountsFilePath)

if err != nil {

if !os.IsNotExist(err) {

log.Println("Failed to read accounts file:", err)

}

return true

}

return len(data) == 0

}

func registerAccount(*scanner* \**bufio*.*Scanner*) {

fmt.Print("Enter account ID: ")

if !scanner.Scan() {

return

}

id := scanner.Text()

fmt.Print("Enter password: ")

if !scanner.Scan() {

return

}

password := scanner.Text()

mu.Lock()

defer mu.Unlock()

if \_, exists := accounts[id]; exists {

if \_, exists := registrations[id]; exists {

fmt.Println("Account already exists.")

return

}

}

registrations[id] = *Registration*{ID: id, Password: password}

fmt.Println("Registration submitted for approval.")

}

func loginadm(*scanner* \**bufio*.*Scanner*) *bool* {

fmt.Print("Enter admin ID: ")

if !scanner.Scan() {

return false

}

id := scanner.Text()

fmt.Print("Enter password: ")

if !scanner.Scan() {

return false

}

password := scanner.Text()

mu.Lock()

defer mu.Unlock()

if id == "admin" && password == "admin" {

currentUser = &*Account*{ID: "admin", Password: "admin"}

return true

}

fmt.Println("Invalid admin credentials.")

return false

}

func loginusr(*scanner* \**bufio*.*Scanner*) *bool* {

fmt.Print("Enter account ID: ")

if !scanner.Scan() {

return false

}

id := scanner.Text()

fmt.Print("Enter password: ")

if !scanner.Scan() {

return false

}

password := scanner.Text()

mu.Lock()

defer mu.Unlock()

account, exists := accounts[id]

if !exists || account.Password != password || !account.Approved {

fmt.Println("Invalid credentials or account not approved.")

return false

}

currentUser = account

return true

}

func userMenu(*scanner* \**bufio*.*Scanner*) {

for {

fmt.Println("1. Check Balance")

fmt.Println("2. Transfer Money")

fmt.Println("3. Make Payment")

fmt.Println("4. Print Transaction History")

fmt.Println("5. Top Up Balance")

fmt.Println("6. Logout")

if !scanner.Scan() {

break

}

option := scanner.Text()

switch option {

case "1":

checkBalance()

case "2":

transferMoney(scanner)

case "3":

makePayment(scanner)

case "4":

printTransactionHistory()

case "5":

topUpBalance(scanner)

case "6":

currentUser = nil

return

default:

fmt.Println("Invalid option. Please try again.")

}

}

}

func adminMenu(*scanner* \**bufio*.*Scanner*) {

for {

fmt.Println("1. Approve/Reject Registration")

fmt.Println("2. Print Account List")

fmt.Println("3. Approve/Reject Top Up Requests")

fmt.Println("4. Logout")

if !scanner.Scan() {

break

}

option := scanner.Text()

switch option {

case "1":

handleRegistrations(scanner)

case "2":

printAccountList()

case "3":

handleTopUpRequests(scanner)

case "4":

currentUser = nil

return

default:

fmt.Println("Invalid option. Please try again.")

}

}

}

func handleRegistrations(*scanner* \**bufio*.*Scanner*) {

mu.Lock()

defer mu.Unlock()

for id, reg := range registrations {

fmt.Printf("Approve account %s? (y/n): ", id)

if !scanner.Scan() {

return

}

if scanner.Text() == "y" {

accounts[id] = &*Account*{ID: id, Password: reg.Password, Balance: 0, Approved: true}

fmt.Println("Account approved.")

} else {

fmt.Println("Account rejected.")

}

delete(registrations, id)

}

}

func printAccountList() {

mu.Lock()

defer mu.Unlock()

fmt.Println("Account List:")

for \_, account := range accounts {

fmt.Printf("ID: %s, Balance: %.2f, Approved: %v\n", account.ID, account.Balance, account.Approved)

}

}

func checkBalance() {

mu.Lock()

defer mu.Unlock()

fmt.Printf("Balance for account %s: %.2f\n", currentUser.ID, currentUser.Balance)

}

func transferMoney(*scanner* \**bufio*.*Scanner*) {

fmt.Print("Enter recipient account ID: ")

if !scanner.Scan() {

return

}

recipientID := scanner.Text()

fmt.Print("Enter amount to transfer: ")

if !scanner.Scan() {

return

}

amountStr := scanner.Text()

amount, err := strconv.ParseFloat(amountStr, 64)

if err != nil || amount <= 0 {

fmt.Println("Invalid amount.")

return

}

mu.Lock()

defer mu.Unlock()

recipient, exists := accounts[recipientID]

if !exists {

fmt.Println("Recipient account not found.")

return

}

if currentUser.Balance < amount {

fmt.Println("Insufficient funds.")

return

}

currentUser.Balance -= amount

recipient.Balance += amount

transaction := *Transaction*{

ID: len(currentUser.Transactions) + 1,

AccountID: currentUser.ID,

Type: "Transfer",

Amount: amount,

Date: time.Now().Format(time.RFC3339),

Details: fmt.Sprintf("Transferred to %s", recipientID),

}

currentUser.Transactions = append(currentUser.Transactions, transaction)

recipient.Transactions = append(recipient.Transactions, *Transaction*{

ID: len(recipient.Transactions) + 1,

AccountID: recipient.ID,

Type: "Transfer",

Amount: amount,

Date: time.Now().Format(time.RFC3339),

Details: fmt.Sprintf("Received from %s", currentUser.ID),

})

fmt.Println("Transfer successful.")

}

func makePayment(*scanner* \**bufio*.*Scanner*) {

fmt.Print("Enter payment type (e.g., food, phone, electricity, BPJS): ")

if !scanner.Scan() {

return

}

paymentType := scanner.Text()

fmt.Print("Enter amount to pay: ")

if !scanner.Scan() {

return

}

amountStr := scanner.Text()

amount, err := strconv.ParseFloat(amountStr, 64)

if err != nil || amount <= 0 {

fmt.Println("Invalid amount.")

return

}

mu.Lock()

defer mu.Unlock()

if currentUser.Balance < amount {

fmt.Println("Insufficient funds.")

return

}

currentUser.Balance -= amount

transaction := *Transaction*{

ID: len(currentUser.Transactions) + 1,

AccountID: currentUser.ID,

Type: "Payment",

Amount: amount,

Date: time.Now().Format(time.RFC3339),

Details: paymentType,

}

currentUser.Transactions = append(currentUser.Transactions, transaction)

fmt.Println("Payment successful.")

}

func printTransactionHistory() {

mu.Lock()

defer mu.Unlock()

fmt.Printf("Transaction history for account %s:\n", currentUser.ID)

for \_, transaction := range currentUser.Transactions {

fmt.Printf("ID: %d, Type: %s, Amount: %.2f, Date: %s, Details: %s\n",

transaction.ID, transaction.Type, transaction.Amount, transaction.Date, transaction.Details)

}

}

func saveAccounts() {

mu.Lock()

defer mu.Unlock()

data, err := json.MarshalIndent(accounts, "", " ")

if err != nil {

log.Println("Failed to marshal accounts:", err)

return

}

err = os.WriteFile(accountsFilePath, data, 0644)

if err != nil {

log.Println("Failed to save accounts:", err)

}

}

func loadAccounts() {

mu.Lock()

defer mu.Unlock()

data, err := os.ReadFile(accountsFilePath)

if err != nil {

if !os.IsNotExist(err) {

log.Println("Failed to read accounts file:", err)

}

return

}

err = json.Unmarshal(data, &accounts)

if err != nil {

log.Println("Failed to unmarshal accounts:", err)

}

}

func topUpBalance(*scanner* \**bufio*.*Scanner*) {

fmt.Print("Enter amount to top up: ")

if !scanner.Scan() {

return

}

amountStr := scanner.Text()

amount, err := strconv.ParseFloat(amountStr, 64)

if err != nil || amount <= 0 {

fmt.Println("Invalid amount.")

return

}

mu.Lock()

defer mu.Unlock()

topUpRequests = append(topUpRequests, *TopUpRequest*{

ID: nextTopUpID,

AccountID: currentUser.ID,

Amount: amount,

Date: time.Now().Format(time.RFC3339),

Approved: false,

})

nextTopUpID++

fmt.Println("Top up request submitted.")

}

func handleTopUpRequests(*scanner* \**bufio*.*Scanner*) {

mu.Lock()

defer mu.Unlock()

for i, request := range topUpRequests {

if !request.Approved {

fmt.Printf("Approve top up request %d for account %s of amount %.2f? (y/n): ", request.ID, request.AccountID, request.Amount)

if !scanner.Scan() {

return

}

if scanner.Text() == "y" {

account := accounts[request.AccountID]

account.Balance += request.Amount

topUpRequests[i].Approved = true

fmt.Println("Top up approved.")

} else {

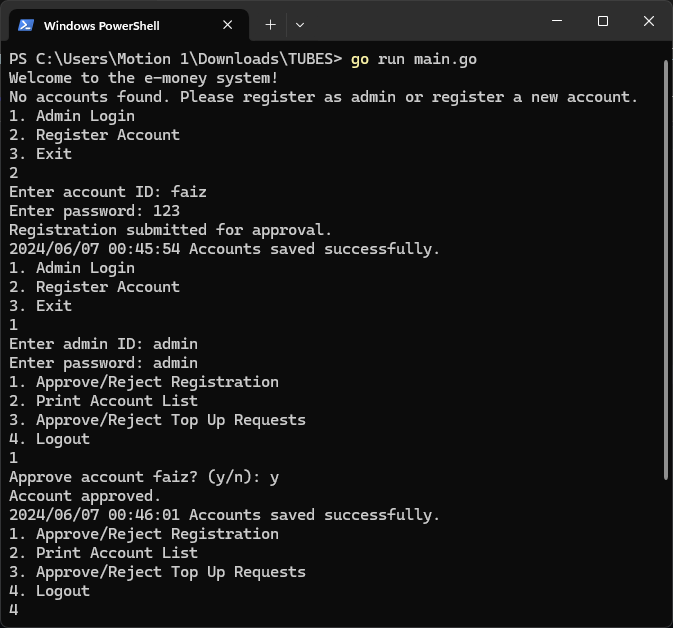
fmt.Println("Top up rejected.")

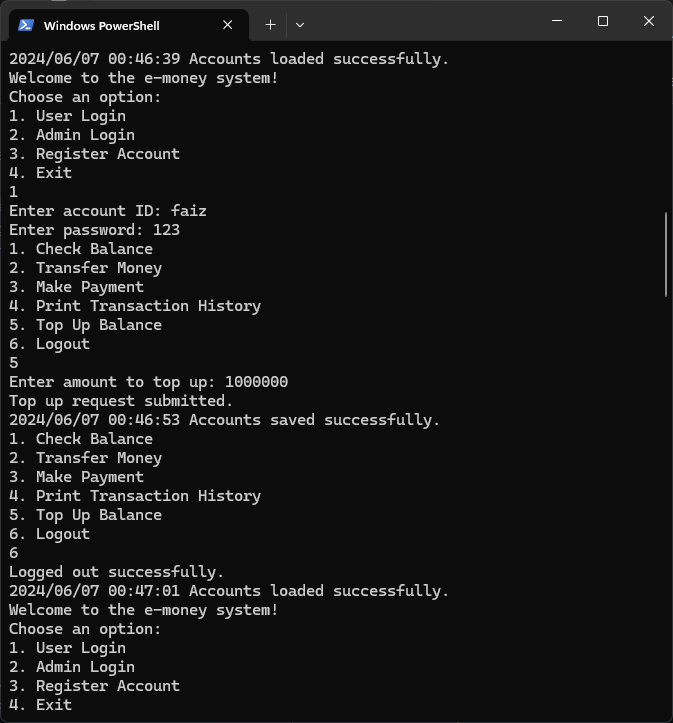
}

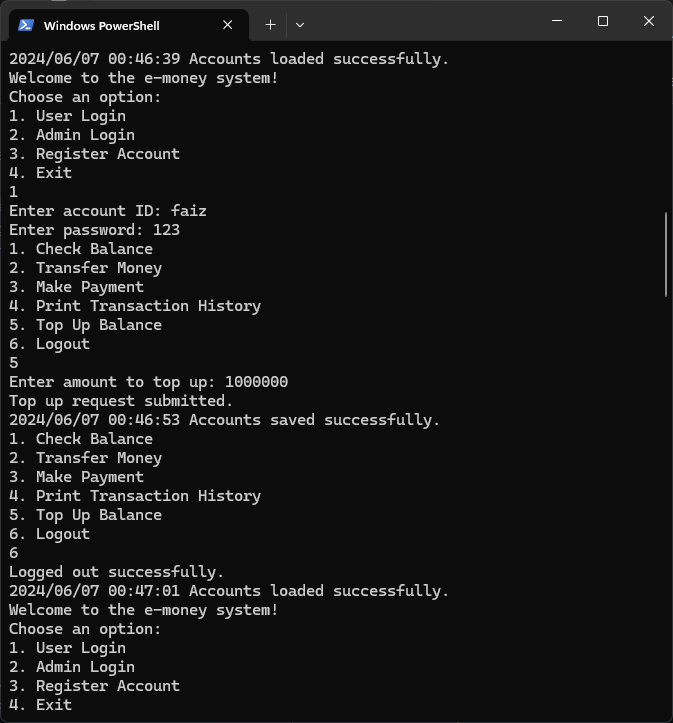
}

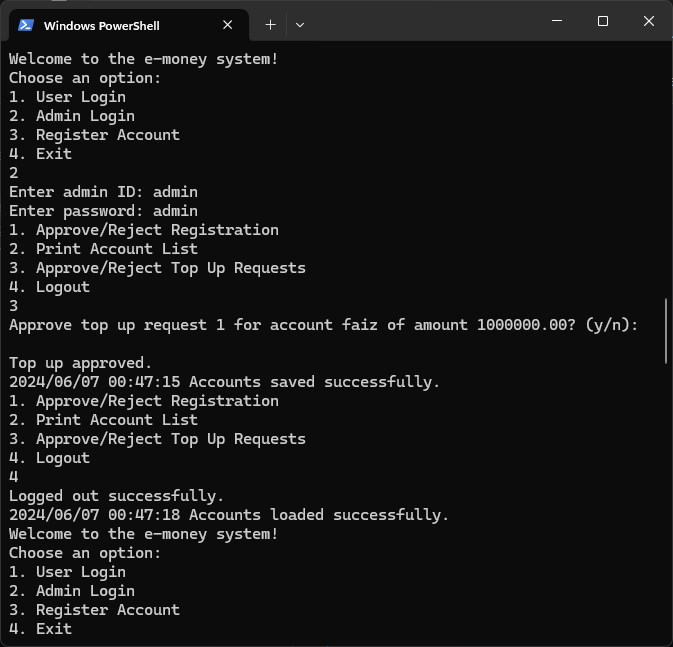
}

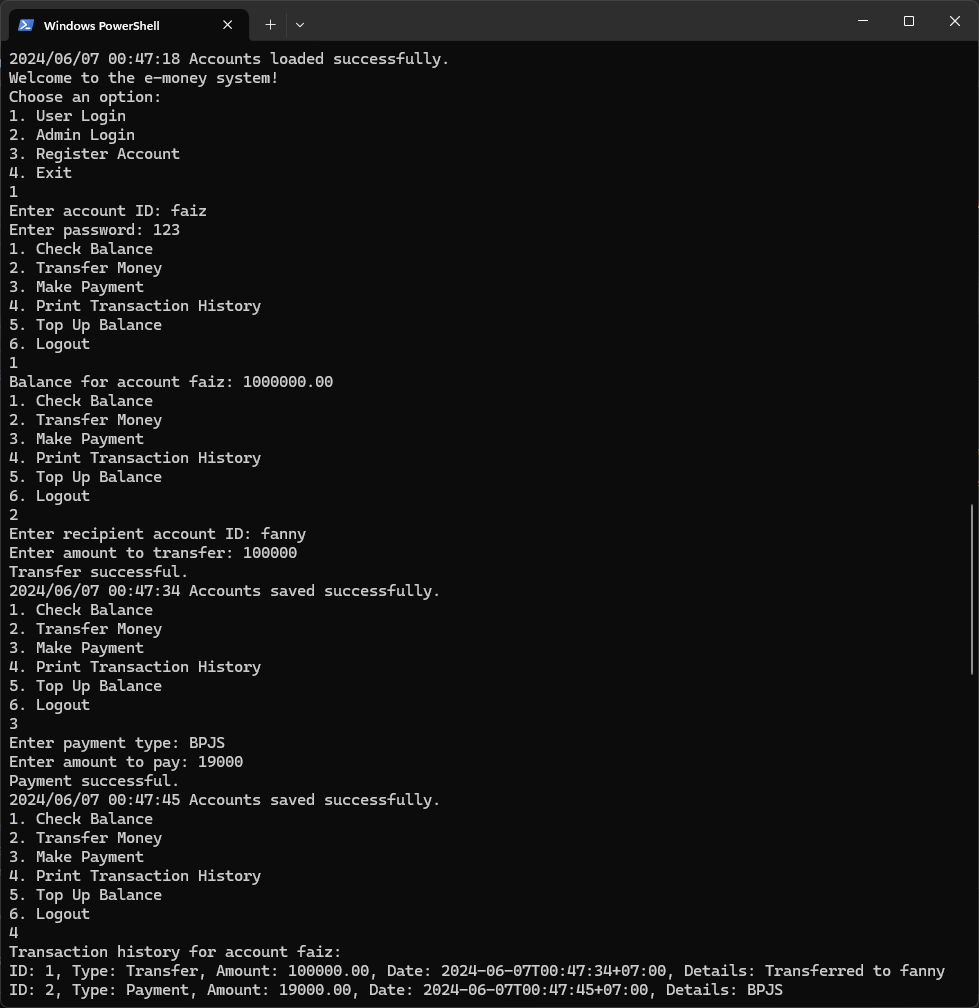
}

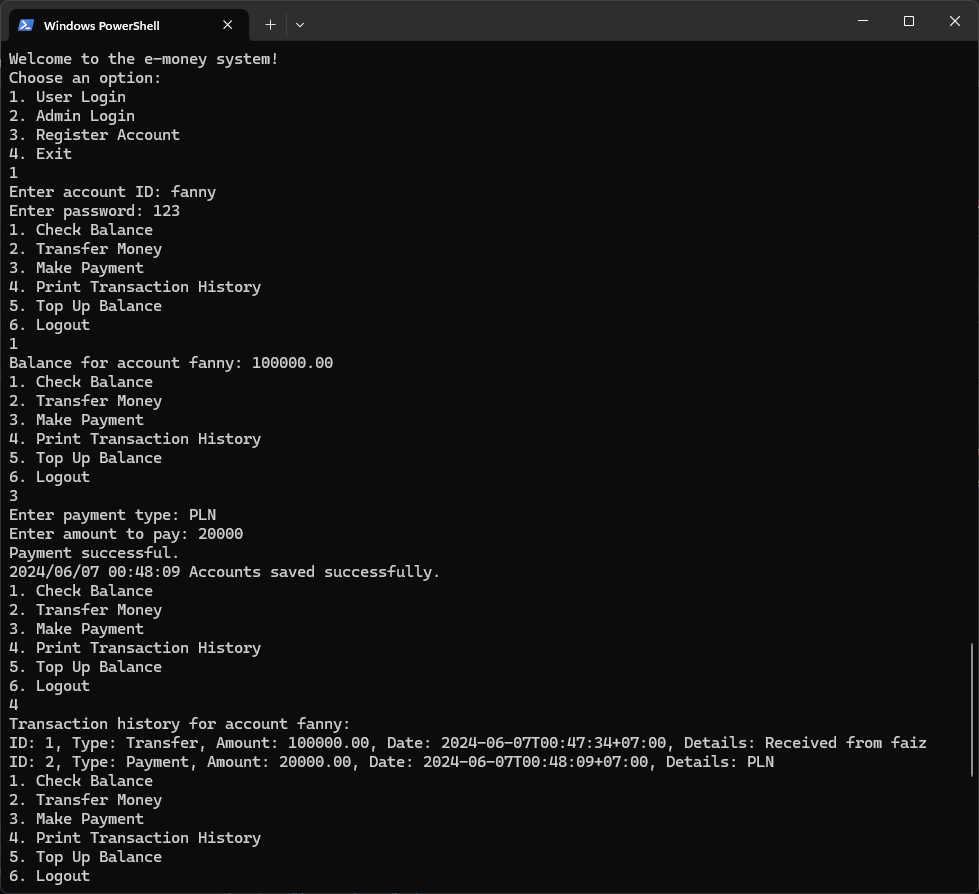


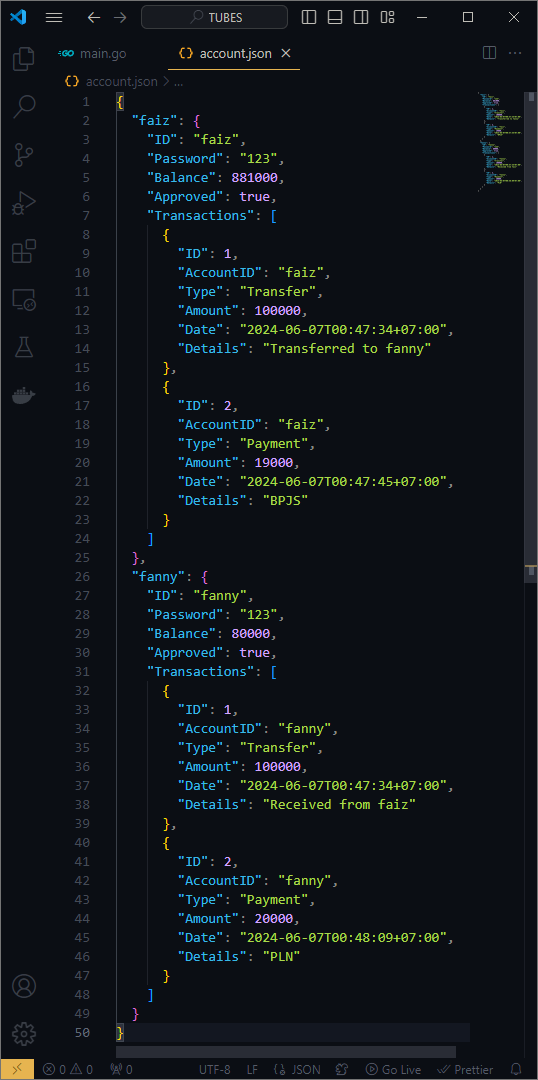












1. **Kesimpulan dan Saran**
   * **Kesimpulan**

Sebuah aplikasi e-money sederhana yang ditulis dalam bahasa Go (Golang) ini menyediakan fungsi untuk mengelola akun pengguna, melakukan transaksi, dan mengelola permintaan top-up. Berikut adalah penjelasan mendetail tentang kode dan fungsinya:

* Struktur Data

1. Account : Mewakili akun pengguna dengan atribut seperti ID, Password, Balance, Approved (status persetujuan), dan Transactions (riwayat transaksi).

2. Transaction : Mewakili transaksi dengan atribut seperti ID, AccountID, Type, Amount, Date, dan Details.

3. Registration : Mewakili data registrasi akun yang perlu disetujui oleh admin.

4. TopUpRequest : Mewakili permintaan top-up saldo dengan atribut seperti ID, AccountID, Amount, Date, dan Approved.

* Fungsi Utama

1. . Main:

- Fungsi utama yang memulai aplikasi.

- Memuat akun dari file JSON.

- Menyediakan menu untuk login sebagai admin, pengguna, atau registrasi akun baru.

1. IsEmptyAccountsFile: Memeriksa apakah file akun kosong atau tidak ada.
2. RegisterAccount: Menyimpan data registrasi akun baru untuk persetujuan admin.
3. Loginadm: Memverifikasi kredensial admin untuk login.
4. Loginusr: Memverifikasi kredensial pengguna untuk login dan memeriksa status persetujuan akun.
5. UserMenu: Menampilkan menu pengguna dengan opsi untuk memeriksa saldo, transfer uang, melakukan pembayaran, mencetak riwayat transaksi, dan top-up saldo.
6. AdminMenu: Menampilkan menu admin dengan opsi untuk menyetujui atau menolak registrasi, mencetak daftar akun, menyetujui atau menolak permintaan top-up.

* Fungsi Pendukung

1. HandleRegistrations: Admin dapat menyetujui atau menolak registrasi akun baru.
2. PrintAccountList: Mencetak daftar akun yang ada beserta saldo dan status persetujuannya.
3. CheckBalance: Memeriksa dan menampilkan saldo akun pengguna yang sedang login.
4. TransferMoney: Mengelola transfer uang dari akun pengguna ke akun penerima.
5. MakePayment: Mengelola pembayaran untuk berbagai layanan (misal: makanan, telepon, listrik).
6. PrintTransactionHistory: Mencetak riwayat transaksi akun pengguna.
7. SaveAccounts: Menyimpan data akun ke file JSON.
8. LoadAccount: Memuat data akun dari file JSON.
9. TopUpBalance: Mengirimkan permintaan top-up saldo untuk disetujui oleh admin.
10. HandleTopUpRequests: Admin dapat menyetujui atau menolak permintaan top-up saldo.
    * **Saran**

1. Penggunaan Konstanta untuk Admin Credentials

Untuk meningkatkan keamanan dan fleksibilitas, sebaiknya admin ID dan password disimpan sebagai konstanta atau dalam konfigurasi eksternal.

2. Enkripsi Password

Gunakan enkripsi untuk menyimpan password agar lebih aman. Anda bisa menggunakan paket `bcrypt` dari Go.

3. Validasi Input

Tambahkan validasi input pada saat pendaftaran dan login untuk memastikan ID dan password memenuhi kriteria tertentu (misalnya, panjang minimal).

4. Peningkatan Logika Penguncian (Mutex)

Gunakan read-write mutex (`sync.RWMutex`) untuk meningkatkan performa aplikasi saat melakukan operasi baca yang banyak.

5. Struktur Data dan File Penyimpanan

Pisahkan file penyimpanan akun dan transaksi agar lebih terstruktur. Anda bisa menggunakan direktori khusus untuk menyimpan data.

6. Unit Testing

Tambahkan unit test untuk memastikan bahwa fungsi-fungsi penting bekerja dengan benar. Gunakan paket `testing` dari Go.

7. Fitur Keamanan

- Two-Factor Authentication (2FA): Tambahkan lapisan keamanan tambahan dengan menerapkan 2FA saat pengguna login atau melakukan transaksi penting.

- Pemantauan Aktivitas: Implementasikan sistem pemantauan aktivitas untuk mendeteksi dan memberi tahu pengguna tentang aktivitas yang mencurigakan pada akun mereka.

- Pembatasan Login: Batasi jumlah percobaan login untuk mencegah serangan brute force.

8. Fitur Akun dan Transaksi

- Histori Transaksi: Buat histori transaksi yang lebih detail dengan opsi untuk menyaring berdasarkan tanggal, tempat, jenis transaksi, atau jumlah.

- Pemberitahuan Real-Time: Kirim pemberitahuan real-time melalui email atau pesan teks untuk setiap transaksi yang dilakukan.

9. Fitur Analisis Pengeluaran: Berikan fitur analisis pengeluaran yang membantu pengguna melihat pola pengeluaran mereka dan membuat anggaran.

**DAFTAR PUSTAKA**

www.kemenkeu.go.id

https://apjii.or.id/survei

<https://money.kompas.com/read/2021/09/20/163000926/apa-itu-uang-elektronik-atau-e-money-ini-pengertian-dan-manfaatnya>

<https://www.bi.go.id/id/edukasi-perlindungan-konsumen/edukasi/produk-dan-jasa-sp/uang-elektronik/Pages/default.aspx>