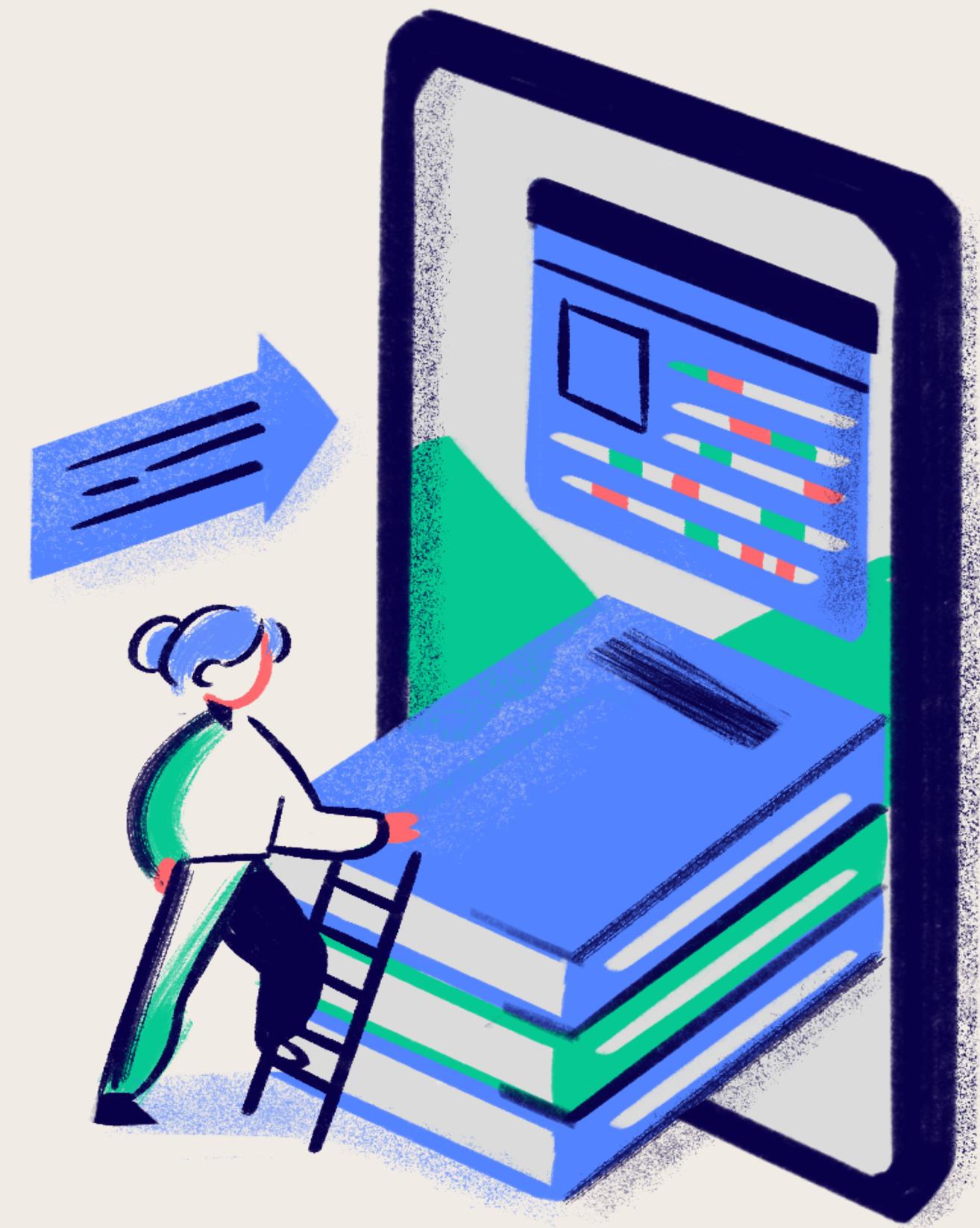


PEMROGRAMAN JARINGAN

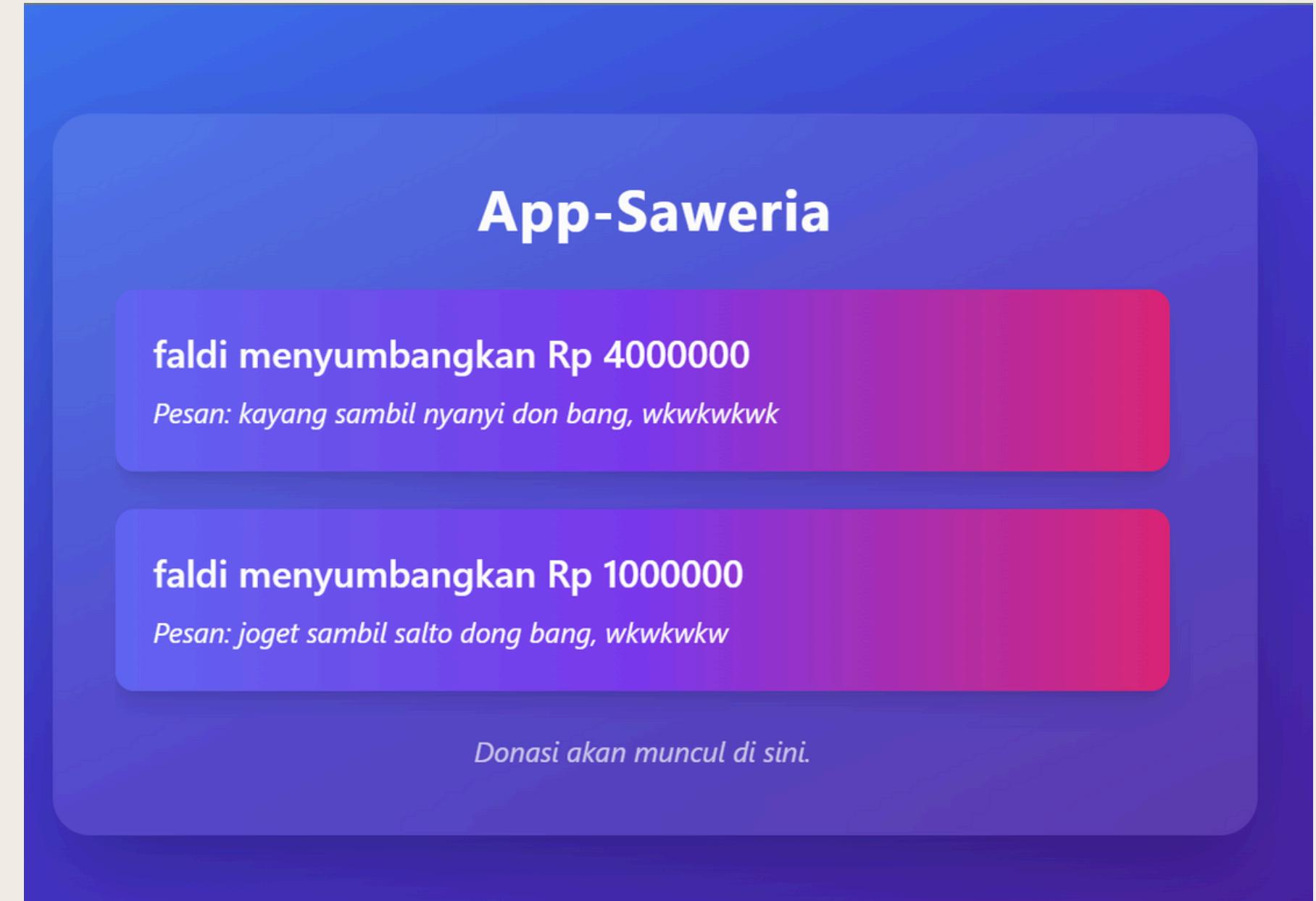
APP-SAWERIA SEDERHANA

YOHANES J PALIS
213400009



APP-SAWERIA?

Aplikasi Saweria sederhana adalah platform atau aplikasi yang memungkinkan orang untuk melakukan donasi secara online kepada content creator, kreator, atau penyelenggara acara. Pada aplikasi saweria kita pengguna dapat menjadi donatur,penerima donasi.



3 PROTOKOL YANG DIGUNAKAN

01. TCP

02. UDP

03. WEBSOCKET

TCP (TRANSMISSION CONTROL PROTOCOL)

Tujuan: Komunikasi berbasis koneksi yang andal, menjamin pengiriman data dalam urutan yang benar.

Fungsi:

- Menerima donasi melalui koneksi TCP.
- Memproses pesan seperti "TOP_UP" untuk memperbarui saldo pengguna.
- Mengirimkan donasi ke WebSocket untuk distribusi ke klien.

```
1 // Fungsi untuk menangani koneksi TCP
2 func handleTCP() {
3     listener, _ := net.Listen("tcp", ":8081") // Menunggu koneksi TCP pada port 8081
4     defer listener.Close() // Menutup listener saat selesai
5     for {
6         conn, _ := listener.Accept() // Menerima koneksi dari klien
7         go func(c net.Conn) {
8             defer c.Close() // Menutup koneksi TCP setelah selesai
9
10        var donation Donation
11        reader := bufio.NewReader(c) // Membaca data dari koneksi TCP
12        line, err := reader.ReadString('\n') // Membaca input baris per baris
13        if err != nil {
14            log.Println("Error reading from TCP client:", err)
15            return
16        }
17
18        // Memisahkan input berdasarkan spasi
19        parts := strings.Fields(line)
20        if len(parts) < 3 {
21            log.Println("Invalid input format:", line)
22            return
23        }
24
25        // Memproses pengirim dan jumlah donasi
26        donation.From = parts[0]
27        donation.Amount, err = strconv.ParseFloat(parts[1], 64) // Mengonversi jumlah donasi dari string ke float
28        if err != nil {
29            log.Println("Invalid amount format:", parts[1])
30            return
31        }
32
33        // Menggabungkan sisa bagian sebagai pesan
34        donation.Message = strings.Join(parts[2:], " ")
35
36        // Jika pesan adalah "TOP_UP", top-up saldo pengguna
37        if donation.Message == "TOP_UP" {
38            balances[donation.From] += donation.Amount
39        } else {
40            balances[donation.From] -= donation.Amount
41            broadcast <- donation // Mengirim donasi ke channel broadcast untuk diteruskan ke WebSocket
42        }
43    }(conn)
44 }
45 }
```

UDP (USER DATAGRAM PROTOCOL)

Tujuan: Komunikasi yang lebih cepat, tanpa jaminan pengiriman data. Digunakan untuk aplikasi yang membutuhkan kecepatan dan tidak bergantung pada keandalan pengiriman.

Fungsi:

- Digunakan untuk Mengecek saldo.
- Mengirimkan saldo pengguna dengan perintah CHECK_BALANCE.

```
1 // Fungsi untuk menangani koneksi UDP
2 func handleUDP() {
3     addr, _ := net.ResolveUDPAddr("udp", ":8082") // Menentukan alamat UDP
4     conn, _ := net.ListenUDP("udp", addr)           // Menunggu koneksi UDP pada port 8082
5     defer conn.Close()                            // Menutup koneksi UDP setelah selesai
6     for {
7         buffer := make([]byte, 1024)             // Menyediakan buffer untuk menerima data UDP
8         n, remoteAddr, _ := conn.ReadFromUDP(buffer) // Membaca data dari UDP
9         message := string(buffer[:n])            // Mengonversi data menjadi string
10
11        // Memisahkan perintah dengan spasi
12        parts := strings.Fields(message)
13        if len(parts) < 2 {
14            continue
15        }
16        username := parts[0]
17        command := parts[1]
18
19        // Perintah untuk memeriksa saldo
20        if command == "CHECK_BALANCE" {
21            balance := balances[username]
22            response := fmt.Sprintf("Saldo anda saat ini: %.2f", balance)
23            conn.WriteToUDP([]byte(response), remoteAddr) // Mengirimkan saldo ke klien UDP
24        }
25    }
26 }
```

WEBSOCKET

Tujuan: Komunikasi dua arah secara real-time antara server dan klien.

Fungsi:

- Menerima donasi dari klien WebSocket.
- Menyebarluaskan donasi ke semua klien yang terhubung secara langsung.
- Menangani koneksi yang terputus dan menghapus klien yang tidak aktif.

```
1 // Fungsi untuk menangani broadcast donasi ke semua klien WebSocket
2 func handleWebSocket() {
3     for {
4         donation := <-broadcast // Menunggu donasi yang diterima
5         clientMutex.Lock()
6         // Mengirimkan donasi ke semua klien WebSocket
7         for client := range clients {
8             err := client.WriteJSON(donation) // Mengirim data donasi ke WebSocket client
9             if err != nil {
10                 log.Printf("Error broadcasting to client: %v", err)
11                 client.Close() // Menutup koneksi WebSocket jika terjadi kesalahan
12                 delete(clients, client) // Menghapus klien yang terputus
13             }
14         }
15         clientMutex.Unlock()
16     }
17 }
```

```
1 // Fungsi untuk menangani koneksi WebSocket
2 func wsHandler(w http.ResponseWriter, r *http.Request) {
3     upgrader.CheckOrigin = func(r *http.Request) bool { return true } // Memungkinkan koneksi dari origin manapun
4     conn, err := upgrader.Upgrade(w, r, nil) // Upgrade koneksi HTTP menjadi WebSocket
5     if err != nil {
6         log.Println("WebSocket upgrade failed:", err)
7         return
8     }
9     clientMutex.Lock()
10    clients[conn] = true // Menambahkan koneksi WebSocket yang baru ke dalam map clients
11    clientMutex.Unlock()
12
13    // Mendengarkan dan menerima donation dari klien WebSocket
14    for {
15        var donation Donation
16        err := conn.ReadJSON(&donation) // Membaca data JSON dari WebSocket
17        if err != nil {
18            log.Printf("Client disconnected: %v", err)
19            clientMutex.Lock()
20            delete(clients, conn) // Menghapus koneksi yang terputus dari map clients
21            clientMutex.Unlock()
22            conn.Close() // Menutup koneksi WebSocket
23            break
24        }
25        broadcast <- donation // Mengirim donation yang diterima ke channel broadcast
26    }
27 }
28 }
```

4 FILE UTAMA

SERVER.GO

CLIENT.GO

CLIENT1.GO

INDEX.HTML

SERVER.GO

Berfungsi untuk mengelola donasi, saldo pengguna, dan untuk menyebarkan informasi donasi ke klien yang terhubung menggunakan 3 protokol utama yaitu , TCP,UDP,WEBSOCKET

Struktur Donasi

Fungsi HandelTCP

Variabel Global

Fungsi HandelUDP

Fungi Main

Fungsi HandleWebsoccet

Fungsi WsHandler

CLIENT.GO

Bertujuan untuk memungkinkan pengguna masuk, mengecek saldo, melakukan donasi, dan menambahkan saldo (top-up) melalui koneksi ke server. Server akan mengelola saldo dan menyimpan data transaksi donasi.

Main()

donate(reader *bufio.Reader)

login(reader *bufio.Reader)

topUpSaldo(reader *bufio.Reader)

mainMenu(reader *bufio.Reader)

checkBalance()

CLIENT1.GO

Fungsi utama client1.go adalah untuk bertindak sebagai client yang menerima data donasi dalam format JSON dari server WebSocket, lalu mem-parsing dan menampilkannya ke dalam log sebagai informasi yang mudah dibaca.

Menghubungkan ke Server WebSocket

Struktur Data Donation

Menerima Data Donasi

Penutupan Koneksi

Menampilkan Informasi Donasi

INDEX.HTML

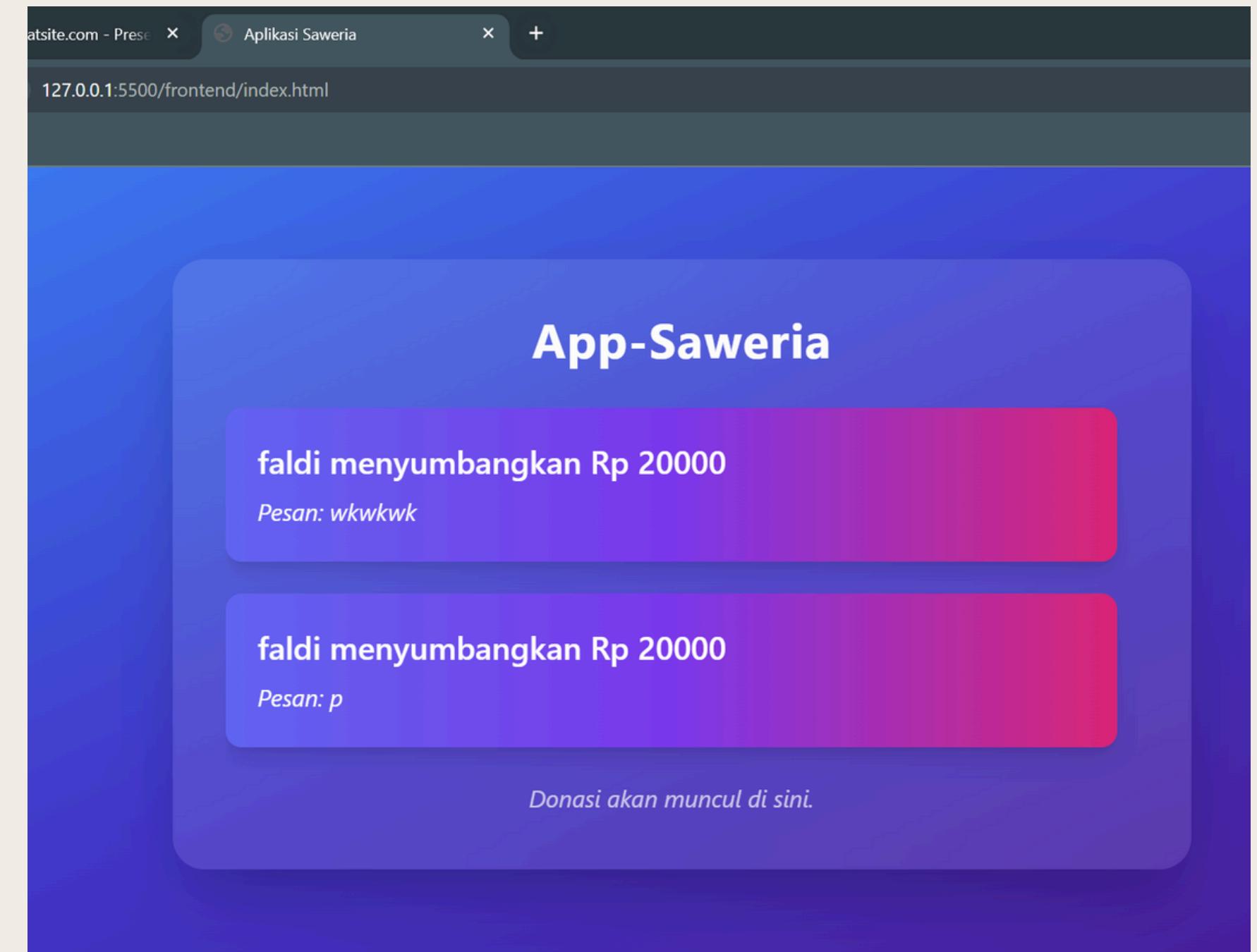
File index.html adalah antarmuka web yang menampilkan daftar donasi secara real-time menggunakan data yang diterima dari server WebSocket.

- Koneksi ke Server WebSocket
- Menerima dan Menampilkan Data Donasi
- Membuat Elemen untuk Menampilkan Donasi
- Menambahkan Donasi ke Daftar
- Antarmuka dan Gaya Tampilan Utama

HASIL IMPLEMENTASI

```
1 package main
2
3 import (
4     "fmt"
5     "log"
6     "net/http"
7     "strings"
8     "time"
9 )
10
11 type User struct {
12     Name string
13     Balance float64
14 }
15
16 var users = map[string]*User{
17     "lou": {Name: "Lou", Balance: 0.0},
18     "faldi": {Name: "Faldi", Balance: 0.0},
19 }
20
21 func main() {
22     http.HandleFunc("/", handler)
23     log.Println("Server started")
24     log.Fatal(http.ListenAndServe(":8080", nil))
25 }
26
27 func handler(w http.ResponseWriter, r *http.Request) {
28     if r.Method == "GET" {
29         w.Write([]byte("GET"))
30     } else if r.Method == "POST" {
31         var user User
32         err := json.NewDecoder(r.Body).Decode(&user)
33         if err != nil {
34             log.Println("Error decoding JSON:", err)
35             http.Error(w, "Bad Request", http.StatusBadRequest)
36             return
37         }
38         if _, ok := users[user.Name]; !ok {
39             log.Println("User not found:", user.Name)
40             http.Error(w, "User not found", http.StatusNotFound)
41             return
42         }
43         if user.Balance <= 0 {
44             log.Println("Insufficient balance for", user.Name)
45             http.Error(w, "Insufficient balance", http.StatusBadRequest)
46             return
47         }
48         user.Balance -= 1000
49         users[user.Name] = &user
50         log.Println("Donation received from", user.Name)
51         log.Println("New balance for", user.Name, ":", user.Balance)
52         w.Write([]byte("POST"))
53     }
54 }
55
56 func ListenAndServe() {
57     log.Println("Starting server on port 8080")
58     http.HandleFunc("/", handler)
59     log.Fatal(http.ListenAndServe(":8080", nil))
60 }
61
62 func handler(w http.ResponseWriter, r *http.Request) {
63     if r.Method == "GET" {
64         w.Write([]byte("GET"))
65     } else if r.Method == "POST" {
66         var user User
67         err := json.NewDecoder(r.Body).Decode(&user)
68         if err != nil {
69             log.Println("Error decoding JSON:", err)
70             http.Error(w, "Bad Request", http.StatusBadRequest)
71             return
72         }
73         if _, ok := users[user.Name]; !ok {
74             log.Println("User not found:", user.Name)
75             http.Error(w, "User not found", http.StatusNotFound)
76             return
77         }
78         if user.Balance <= 0 {
79             log.Println("Insufficient balance for", user.Name)
80             http.Error(w, "Insufficient balance", http.StatusBadRequest)
81             return
82         }
83         user.Balance += 1000
84         users[user.Name] = &user
85         log.Println("Donation received from", user.Name)
86         log.Println("New balance for", user.Name, ":", user.Balance)
87         w.Write([]byte("POST"))
88     }
89 }
90
91 func handler(w http.ResponseWriter, r *http.Request) {
92     if r.Method == "GET" {
93         w.Write([]byte("GET"))
94     } else if r.Method == "POST" {
95         var user User
96         err := json.NewDecoder(r.Body).Decode(&user)
97         if err != nil {
98             log.Println("Error decoding JSON:", err)
99             http.Error(w, "Bad Request", http.StatusBadRequest)
100            return
101        }
102        if _, ok := users[user.Name]; !ok {
103            log.Println("User not found:", user.Name)
104            http.Error(w, "User not found", http.StatusNotFound)
105            return
106        }
107        if user.Balance <= 0 {
108            log.Println("Insufficient balance for", user.Name)
109            http.Error(w, "Insufficient balance", http.StatusBadRequest)
110            return
111        }
112        user.Balance += 1000
113        users[user.Name] = &user
114        log.Println("Donation received from", user.Name)
115        log.Println("New balance for", user.Name, ":", user.Balance)
116        w.Write([]byte("POST"))
117    }
118 }
```

```
1 package main
2
3 import (
4     "fmt"
5     "log"
6     "net/http"
7     "strings"
8     "time"
9 )
10
11 type User struct {
12     Name string
13     Balance float64
14 }
15
16 var users = map[string]*User{
17     "lou": {Name: "Lou", Balance: 0.0},
18     "faldi": {Name: "Faldi", Balance: 0.0},
19 }
20
21 func main() {
22     http.HandleFunc("/", handler)
23     log.Println("Server started")
24     log.Fatal(http.ListenAndServe(":8080", nil))
25 }
26
27 func handler(w http.ResponseWriter, r *http.Request) {
28     if r.Method == "GET" {
29         w.Write([]byte("GET"))
30     } else if r.Method == "POST" {
31         var user User
32         err := json.NewDecoder(r.Body).Decode(&user)
33         if err != nil {
34             log.Println("Error decoding JSON:", err)
35             http.Error(w, "Bad Request", http.StatusBadRequest)
36             return
37         }
38         if _, ok := users[user.Name]; !ok {
39             log.Println("User not found:", user.Name)
40             http.Error(w, "User not found", http.StatusNotFound)
41             return
42         }
43         if user.Balance <= 0 {
44             log.Println("Insufficient balance for", user.Name)
45             http.Error(w, "Insufficient balance", http.StatusBadRequest)
46             return
47         }
48         user.Balance -= 1000
49         users[user.Name] = &user
50         log.Println("Donation received from", user.Name)
51         log.Println("New balance for", user.Name, ":", user.Balance)
52         w.Write([]byte("POST"))
53     }
54 }
55
56 func ListenAndServe() {
57     log.Println("Starting server on port 8080")
58     http.HandleFunc("/", handler)
59     log.Fatal(http.ListenAndServe(":8080", nil))
60 }
61
62 func handler(w http.ResponseWriter, r *http.Request) {
63     if r.Method == "GET" {
64         w.Write([]byte("GET"))
65     } else if r.Method == "POST" {
66         var user User
67         err := json.NewDecoder(r.Body).Decode(&user)
68         if err != nil {
69             log.Println("Error decoding JSON:", err)
70             http.Error(w, "Bad Request", http.StatusBadRequest)
71             return
72         }
73         if _, ok := users[user.Name]; !ok {
74             log.Println("User not found:", user.Name)
75             http.Error(w, "User not found", http.StatusNotFound)
76             return
77         }
78         if user.Balance <= 0 {
79             log.Println("Insufficient balance for", user.Name)
80             http.Error(w, "Insufficient balance", http.StatusBadRequest)
81             return
82         }
83         user.Balance += 1000
84         users[user.Name] = &user
85         log.Println("Donation received from", user.Name)
86         log.Println("New balance for", user.Name, ":", user.Balance)
87         w.Write([]byte("POST"))
88     }
89 }
90
91 func handler(w http.ResponseWriter, r *http.Request) {
92     if r.Method == "GET" {
93         w.Write([]byte("GET"))
94     } else if r.Method == "POST" {
95         var user User
96         err := json.NewDecoder(r.Body).Decode(&user)
97         if err != nil {
98             log.Println("Error decoding JSON:", err)
99             http.Error(w, "Bad Request", http.StatusBadRequest)
100            return
101        }
102        if _, ok := users[user.Name]; !ok {
103            log.Println("User not found:", user.Name)
104            http.Error(w, "User not found", http.StatusNotFound)
105            return
106        }
107        if user.Balance <= 0 {
108            log.Println("Insufficient balance for", user.Name)
109            http.Error(w, "Insufficient balance", http.StatusBadRequest)
110            return
111        }
112        user.Balance += 1000
113        users[user.Name] = &user
114        log.Println("Donation received from", user.Name)
115        log.Println("New balance for", user.Name, ":", user.Balance)
116        w.Write([]byte("POST"))
117    }
118 }
```



**THANK
YOU VERY
MUCH!**

