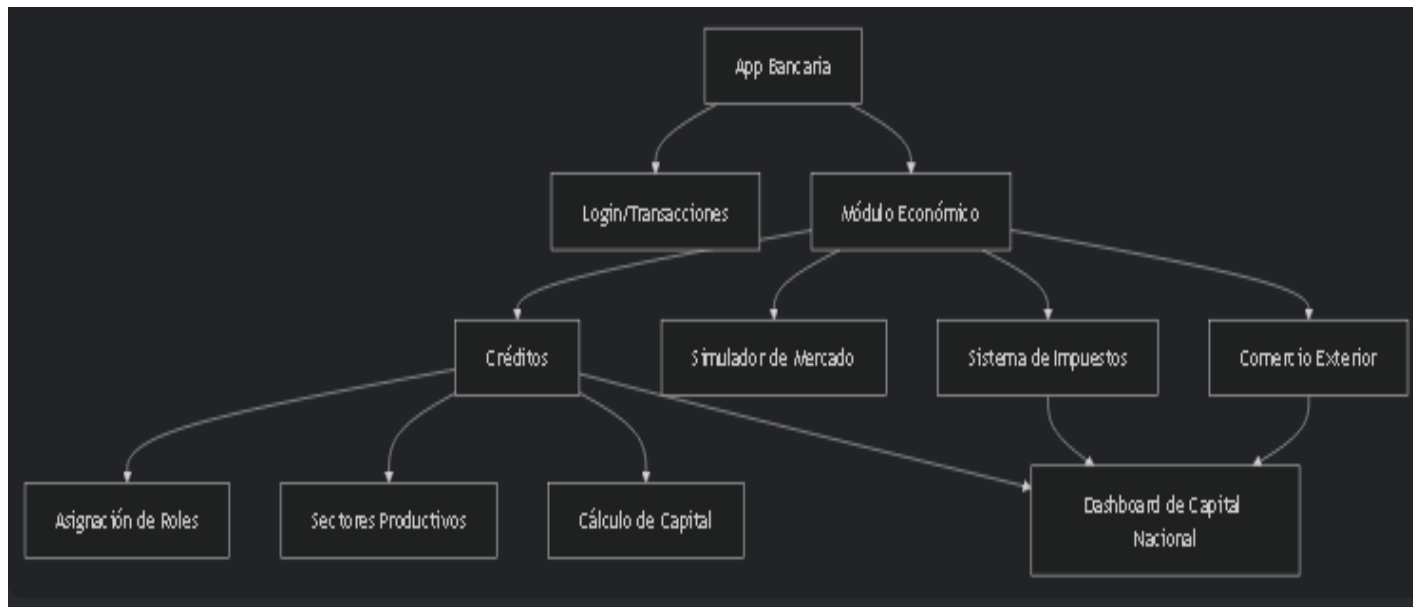


Informe sobre el app **bancario**

Jhoan Bernal

30 de enero de 2025

Diagrama de Arquitectura




implementado

implementado:

- Sistema de login/registro de usuarios
- Operaciones bancarias básicas (depósitos, transferencias)cobro de iva por cada transacción
- Generación de códigos únicos para transacciones
- Interfaz de usuario con saldo y historial
- seccion de creditos con cobro automatico de iva semanal

base de dato

Modelo básico para una base de datos bancaria en MySQL, que incluye las tablas de Clientes, Cuentas y Transacciones.) 

```
-- phpMyAdmin SQL Dump
-- version 5.2.1
-- https://www.phpmyadmin.net/
--
-- Servidor: 127.0.0.1
-- Tiempo de generación: 28-03-2025 a las 19:56:48
-- Versión del servidor: 10.4.32-MariaDB
-- Versión de PHP: 8.2.12

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
START TRANSACTION;
SET time_zone = "+00:00";

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT
*/;

/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS
*/;
```

```
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION
*/;

/*!40101 SET NAMES utf8mb4 */;

--

-- Base de datos: `banco`

--

CREATE DATABASE IF NOT EXISTS `banco` DEFAULT CHARACTER SET
latin1 COLLATE latin1_swedish_ci;

USE `banco`;

-----

--

-- Estructura de tabla para la tabla `administradores`

--

CREATE TABLE `administradores` (

  `id` int(11) NOT NULL,

  `nombre` varchar(100) NOT NULL,

  `apellido` varchar(100) NOT NULL,
```

```

`email` varchar(100) NOT NULL,

`password` varchar(255) NOT NULL,

`fecha_creacion` timestamp NOT NULL DEFAULT
current_timestamp()

) ENGINE=InnoDB DEFAULT CHARSET=latin1
COLLATE=latin1_swedish_ci;

--

-- Volcado de datos para la tabla `administradores`

--

INSERT INTO `administradores` (`id`, `nombre`, `apellido`,
`email`, `password`, `fecha_creacion`) VALUES

(1, 'admin', 'admin', 'admin@example.com',
'$2a$10$zfZzXwgmh4vk3rROMZLO2eRJXJfEbGcimsn0bMGyrY4SrDIKuiine',
'2025-03-11 23:27:55');

-----

--

-- Estructura de tabla para la tabla `clientes`

--

```

```

CREATE TABLE `clientes` (

  `id` INT NOT NULL AUTO_INCREMENT,

  `nombre` VARCHAR(100) NOT NULL,

  `apellido` VARCHAR(100) NOT NULL,

  `fecha_nacimiento` DATE NOT NULL,

  `email` VARCHAR(100) NOT NULL,

  `telefono` VARCHAR(15) DEFAULT NULL,

  `password` VARCHAR(60) NOT NULL,

  `codigo_cuenta` VARCHAR(255) DEFAULT NULL,

  PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1
COLLATE=latin1_swedish_ci;

--

-- Volcado de datos para la tabla `clientes`

--

INSERT INTO `clientes` (`id`, `nombre`, `apellido`,
`fecha_nacimiento`, `email`, `telefono`, `password`,
`codigo_cuenta`) VALUES

(6, 'jhoan', 'bernal', '2025-03-19', 'jhoan@gmail.com', NULL,
'$2a$10$aTt7gpg10HazikeMHdqvL.9ke5vdaxfPWUqau.IrIVQ5pmH0YDAKm',
'CUENTA-535624'),

```

```
(7, 'alber', 'erez', '2025-03-20', 'jose@gmail.com', NULL,
'$2a$10$mR4lQI0weHecmzW48KcwCupJl.95EMnMMtamuVdHZZMQG6lwgdnXC',
'CUENTA-565728');
```

```
-- -----
```

```
--
```

```
-- Estructura de tabla para la tabla `cuentas`
```

```
--
```

```
CREATE TABLE `cuentas` (
  `id` int(11) NOT NULL,
  `id_cliente` int(11) NOT NULL,
  `tipo_cuenta` varchar(50) DEFAULT NULL,
  `saldo` float DEFAULT 0,
  `fecha_creacion` timestamp NOT NULL DEFAULT
current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=latin1
COLLATE=latin1_swedish_ci;
```

```
--
```

```
-- Volcado de datos para la tabla `cuentas`
```

```
--  
  
INSERT INTO `cuentas` (`id`, `id_cliente`, `tipo_cuenta`,  
`saldo`, `fecha_creacion`) VALUES  
  
(1, 6, 'Cuenta de Ahorros', 496, '2025-03-12 00:03:21'),  
(2, 7, 'Cuenta de Ahorros', 24, '2025-03-12 23:51:46');
```

```
--  
  
-- Estructura de tabla para la tabla `historial_recargas`  
  
--
```

```
CREATE TABLE `historial_recargas` (  
  `id` int(11) NOT NULL,  
  `id_admin` int(11) NOT NULL,  
  `id_cliente` int(11) NOT NULL,  
  `monto` decimal(10,2) NOT NULL,  
  `fecha_recarga` timestamp NOT NULL DEFAULT  
current_timestamp(),  
  `codigo_cuenta` varchar(255) NOT NULL
```



```
) ENGINE=InnoDB DEFAULT CHARSET=latin1
COLLATE=latin1_swedish_ci;

--

-- Volcado de datos para la tabla `historial_recargas`
--

INSERT INTO `historial_recargas` (`id`, `id_admin`,
`id_cliente`, `monto`, `fecha_recarga`, `codigo_cuenta`) VALUES
(1, 1, 6, 2333.00, '2025-03-12 18:20:27', 'CUENTA-535624'),
(2, 1, 6, 500.00, '2025-03-13 20:00:14', 'CUENTA-535624');

-----

--

-- Estructura de tabla para la tabla `transacciones`
--

CREATE TABLE `transacciones` (
  `id` int(11) NOT NULL,
  `id_cuenta` int(11) NOT NULL,
```

```

`tipo_transaccion` enum('Deposito','Retiro') NOT NULL,

`monto` decimal(10,2) NOT NULL,

`fecha_transaccion` timestamp NOT NULL DEFAULT
current_timestamp()

) ENGINE=InnoDB DEFAULT CHARSET=latin1
COLLATE=latin1_swedish_ci;

--

-- Crear tabla para historial de transacciones entre usuarios
CREATE TABLE historial_transacciones_usuarios (

    id INT NOT NULL AUTO_INCREMENT,

    id_emisor INT NOT NULL,

    id_receptor INT NOT NULL,

    monto DECIMAL(10,2) NOT NULL,

    fecha_transaccion TIMESTAMP NOT NULL DEFAULT
current_timestamp(),

    descripcion VARCHAR(255) NOT NULL,

    codigo_cuenta_emisor VARCHAR(255) NOT NULL,

    codigo_cuenta_receptor VARCHAR(255) NOT NULL,

    FOREIGN KEY (id_emisor) REFERENCES Clientes(id),

    FOREIGN KEY (id_receptor) REFERENCES Clientes(id),

    PRIMARY KEY (id)

```

```
) ;

--

-- Índices para tablas volcadas

--

--

-- Indices de la tabla `administradores`

--

ALTER TABLE `administradores`

  ADD PRIMARY KEY (`id`),

  ADD UNIQUE KEY `email` (`email`);

--

-- Indices de la tabla `clientes`

--

ALTER TABLE `clientes`

  ADD PRIMARY KEY (`id`),

  ADD UNIQUE KEY `email` (`email`);

--
```

```
-- Indices de la tabla `cuentas`  
--  
ALTER TABLE `cuentas`  
  ADD PRIMARY KEY (`id`),  
  ADD KEY `id_cliente` (`id_cliente`);  
  
--  
-- Indices de la tabla `historial_recargas`  
--  
ALTER TABLE `historial_recargas`  
  ADD PRIMARY KEY (`id`),  
  ADD KEY `id_admin` (`id_admin`),  
  ADD KEY `id_cliente` (`id_cliente`);  
  
--  
-- Indices de la tabla `transacciones`  
--  
ALTER TABLE `transacciones`  
  ADD PRIMARY KEY (`id`),  
  ADD KEY `id_cuenta` (`id_cuenta`);
```

```
--  
  
-- AUTO_INCREMENT de las tablas volcadas  
  
--  
  
--  
  
-- AUTO_INCREMENT de la tabla `administradores`  
  
--  
  
ALTER TABLE `administradores`  
  
    MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=2;  
  
--  
  
-- AUTO_INCREMENT de la tabla `clientes`  
  
--  
  
ALTER TABLE `clientes`  
  
    MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=8;  
  
--  
  
-- AUTO_INCREMENT de la tabla `cuentas`  
  
--  
  
ALTER TABLE `cuentas`  
  
    MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=3;
```

```
--  
  
-- AUTO_INCREMENT de la tabla `historial_recargas`  
  
--  
  
ALTER TABLE `historial_recargas`  
  
    MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=3;  
  
--  
  
-- AUTO_INCREMENT de la tabla `transacciones`  
  
--  
  
ALTER TABLE `transacciones`  
  
    MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;  
  
--  
  
-- Restricciones para tablas volcadas  
  
--  
  
--  
  
-- Filtros para la tabla `cuentas`  
  
--  
  
ALTER TABLE `cuentas`
```

```
ADD CONSTRAINT `cuentas_ibfk_1` FOREIGN KEY (`id_cliente`)
REFERENCES `clientes` (`id`);
```

```
--
```

```
-- Filtros para la tabla `historial_recargas`
```

```
--
```

```
ALTER TABLE `historial_recargas`
```

```
ADD CONSTRAINT `historial_recargas_ibfk_1` FOREIGN KEY
(`id_admin`) REFERENCES `administradores` (`id`),
```

```
ADD CONSTRAINT `historial_recargas_ibfk_2` FOREIGN KEY
(`id_cliente`) REFERENCES `clientes` (`id`);
```

```
--
```

```
-- Filtros para la tabla `transacciones`
```

```
--
```

```
ALTER TABLE `transacciones`
```

```
ADD CONSTRAINT `transacciones_ibfk_1` FOREIGN KEY
(`id_cuenta`) REFERENCES `cuentas` (`id`);
```

```
--
```

```
-- Añadir clave primaria a la tabla `clientes`
```

```
--
```

```
ALTER TABLE `clientes` ADD PRIMARY KEY (id);

COMMIT;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS
*/;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

Detalles de las Tablas:

1. Clientes:

id: Identificador único para cada cliente (clave primaria).

nombre: Nombre del cliente.

apellido: Apellido del cliente.

fecha_nacimiento: Fecha de nacimiento del cliente.

email: Correo electrónico del cliente (debe ser único).


telefono: Número de teléfono del cliente (opcional).

Password: Agregado para almacenar la contraseña

2. Cuentas:

id: Identificador único para cada cuenta (clave primaria).

id_cliente: Identificador del cliente que posee la cuenta (clave foránea que referencia a Clientes).



tipo_cuenta: Tipo de cuenta (Ahorros o Corriente).

saldo: Saldo actual de la cuenta.

fecha_creacion: Fecha en que se creó la cuenta.

3. Transacciones:

id: Identificador único para cada transacción (clave primaria).

id_cuenta: Identificador de la cuenta asociada a la transacción (clave foránea que referencia a Cuentas).

tipo_transaccion: Tipo de transacción (Depósito o Retiro).

monto: Monto de la transacción.

fecha_transaccion: Fecha en que se realizó la transacción.

Este modelo básico te permitirá gestionar clientes, cuentas y transacciones en una base de datos bancaria.

Para desarrollar un proyecto de base de datos bancaria en Go (Golang), necesitarás utilizar un paquete para interactuar con MySQL, como `database/sql` junto con un driver como `go-sql-driver/mysql`. A continuación, te muestro un ejemplo básico de cómo podrías estructurar tu proyecto en Go, incluyendo la conexión a la base de datos y las operaciones básicas para las tablas que definimos anteriormente.

db.go Driver Conexión a la Base de Datos

```
go mod init banco
```

Descarga el driver de MySQL ejecutando el siguiente comando:

```
go get github.com/go-sql-driver/mysql
```

Estructura del proyecto

/banco

```
|— db
|  └─ db.go
|— models
|  └─ models.go
|— doc
|  └─ notas.txt
└─ go.mod
└─ go.sum
└─ main.go
```

db.go Conexión a la Base de Datos

```
package db

import (
```

```
"database/sql"

"log"

_ "github.com/go-sql-driver/mysql"
)

var db *sql.DB

func InitDB() {

    var err error

    // Cambia "root:@tcp(127.0.0.1:3306)/Banco" por
tus credenciales

    db, err = sql.Open("mysql",
"root:@tcp(127.0.0.1:3306)/Banco")

    if err != nil {

        log.Fatal(err)

    }

    if err = db.Ping(); err != nil {
```

```
        log.Fatal(err)

    }

}

func CloseDB() {

    if db != nil {

        db.Close()

    }

}
```

model.go

```
package model

import (

    "time"

)

type Cliente struct {

    ID          int          `db:"id"`
```

```
Nombre          string    `db:"nombre"`
Apellido        string    `db:"apellido"`
FechaNacimiento time.Time `db:"fecha_nacimiento"`
Email           string    `db:"email"`
Telefono        string
Password        string    `db:"password"`
}

type Cuenta struct {
    ID            int
    IDCliente     int
    TipoCuenta    string
    Saldo         float64
    FechaCreacion time.Time
}

type Transaccion struct {
    ID            int
    IDCuenta      int
    TipoTransaccion string
    Monto         float64
}
```

```
    FechaTransaccion time.Time
}
```

main.go

```
package main

import (
    "banco/db"
    "banco/routes"
)

func main() {
    db.InitDB()
    defer db.CloseDB()

    r := routes.SetupRouter()
    r.Run(":8080") // Inicia el servidor en el puerto 8080
}
```

```
}
```

creando Admin

1. Modificaciones en models.go

En el archivo models.go, se definió el modelo Administrador para almacenar la información de los administradores en la base de datos. Aquí está el modelo que se agregó:

```
type Administrador struct {  
    ID          int    `db:"id"`           // ID del administrador  
    Nombre      string `db:"nombre"`       // Nombre del administrador  
    Apellido    string `db:"apellido"`     // Apellido del administrador  
    Email       string `db:"email"`        // Email del administrador  
    Password    string `db:"password"`     // Contraseña hasheada del administrador  
}
```

2. Modificaciones en routes.go

En el archivo routes.go, se realizaron las siguientes modificaciones:

a. Función para mostrar la página de registro de administradores

Se creó la función showRegisterAdminPage para mostrar la plantilla de registro de administradores:

```
func showRegisterAdminPage(c *gin.Context) {  
    c.HTML(http.StatusOK, "registerAdmin.html", nil)
```

```
}
```

b. Ruta para mostrar la página de registro de administradores

Se agregó la ruta para acceder a la página de registro de administradores:

```
r.GET("/registerAdmin", authMiddleware(),
showRegisterAdminPage) // Asegúrate de que esta línea esté
presente
```

c. Función para registrar un administrador

Se implementó la función registerAdmin para manejar el registro de nuevos administradores:

```
func registerAdmin(c *gin.Context) {

    var json struct {

        Nombre    string `form:"nombre" binding:"required"`

        Apellido  string `form:"apellido" binding:"required"`

        Email    string `form:"email" binding:"required,email"`

        Password string `form:"password"
binding:"required,min=8"`

    }

}
```



```

    if err := c.ShouldBind(&json); err != nil {

        c.JSON(http.StatusBadRequest, gin.H{"error": "Datos
inválidos: " + err.Error()})

        return

    }

    // Hashear la contraseña

    hashedPassword, err :=
bcrypt.GenerateFromPassword([]byte(json.Password),
bcrypt.DefaultCost)

    if err != nil {

        c.JSON(http.StatusInternalServerError, gin.H{"error":
"Error al hashear la contraseña"})

        return

    }

    // Insertar el nuevo administrador en la base de datos

    query := "INSERT INTO Administradores (nombre, apellido,
email, password) VALUES (?, ?, ?, ?)"

    _, err = db.DB.Exec(query, json.Nombre, json.Apellido,
json.Email, hashedPassword)

    if err != nil {

```

```

        fmt.Printf("Error al registrar el administrador: %v\n",
err) // Imprimir el error en el servidor

        c.JSON(http.StatusInternalServerError, gin.H{"error":
"Error al registrar el administrador"})

        return

    }

    // Establecer la sesión para el nuevo administrador

    session := sessions.Default(c)

    session.Set("user_id", 1) // Asumiendo que el ID del nuevo
administrador es 1

    session.Save()

    c.Redirect(http.StatusSeeOther, "/admin") // Redirigir a la
página de administración
}

```

d. Ruta para registrar un administrador

Se agregó la ruta para manejar el registro de administradores:

```

r.POST("/registerAdmin", authMiddleware(), registerAdmin) //
Asegúrate de que esta línea esté presente

```

3. Creación de la plantilla registerAdmin.html

Se creó la plantilla registerAdmin.html para permitir que los administradores ingresen sus datos. Aquí está el contenido de la plantilla:

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width,
initial-scale=1.0">

    <title>Registrar Administrador</title>

    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/boot
strap.min.css" rel="stylesheet">

</head>

<body>

<div class="container mt-4 mb-4 mx-auto" style="max-width:
800px;">

    <h2>Registrar Administrador</h2>

    <form action="/registerAdmin" method="POST">

        <div class="mb-3">

            <label for="nombre"
class="form-label">Nombre</label>
```

```
        <input type="text" class="form-control" id="nombre"
name="nombre" required>

    </div>

    <div class="mb-3">

        <label for="apellido"
class="form-label">Apellido</label>

        <input type="text" class="form-control"
id="apellido" name="apellido" required>

    </div>

    <div class="mb-3">

        <label for="email" class="form-label">Email</label>

        <input type="email" class="form-control" id="email"
name="email" required>

    </div>

    <div class="mb-3">

        <label for="password"
class="form-label">Contraseña</label>

        <input type="password" class="form-control"
id="password" name="password" required>

    </div>

    <button type="submit" class="btn
btn-primary">Registrar</button>

</form>
```

```
</div>

<!-- Bootstrap JS y Popper.js -->

<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd
/popper.min.js"></script>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootst
rap.min.js"></script>

</body>

</html>
```

4. Creación de la plantilla admin.html

Se creó la plantilla admin.html para permitir que el administrador recargue el saldo de los clientes. Aquí está el contenido de la plantilla:

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width,
initial-scale=1.0">

    <title>Administración</title>

    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/boot
strap.min.css" rel="stylesheet">
```

```
</head>

<body>

<div class="container mt-4 mb-4 mx-auto" style="max-width:
800px;">

    <h2>Recargar Saldo de Clientes</h2>

    <form action="/recargar" method="POST">

        <div class="mb-3">

            <label for="codigoCuenta" class="form-label">Código
de Cuenta</label>

            <input type="text" class="form-control"
id="codigoCuenta" name="codigoCuenta" required>

        </div>

        <div class="mb-3">

            <label for="monto" class="form-label">Monto a
Recargar</label>

            <input type="number" class="form-control" id="monto"
name="monto" required>

        </div>

        <button type="submit" class="btn
btn-primary">Recargar</button>

    </form>

</div>
```

```
<!-- Bootstrap JS y Popper.js -->

<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd
/popper.min.js"></script>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootst
rap.min.js"></script>

</body>

</html>
```

Resumen

Modificaciones en models.go: Se definió el modelo Administrador.

Modificaciones en routes.go:

Se implementaron las funciones showRegisterAdminPage y registerAdmin.

Se configuraron las rutas para mostrar la página de registro y para registrar un nuevo administrador.

Creación de la plantilla registerAdmin.html: Se creó la plantilla para registrar administradores.

Creación de la plantilla admin.html: Se creó la plantilla para que el administrador recargue el saldo de los clientes.

Con esta documentación, deberías tener un registro claro de cómo se implementaron las funcionalidades relacionadas con el registro y la administración de administradores.

creando recargas o depósito de saldo a los clientes siendo admin

Vamos a resumir los pasos que tomamos en los archivos routes.go, admin.html y banco.sql para que la funcionalidad de recarga de saldo funcione correctamente. Luego, también te proporcionaré una guía sobre cómo implementar la alerta de mensaje y el historial de recargas.

Resumen de los pasos realizados

1. Modificaciones en routes.go

Definición de la ruta para recargar saldo:

Se agregó la ruta POST /recargar que permite a los administradores recargar el saldo de los clientes.

```
r.POST("/recargar", authMiddleware(), recargarSaldo) //
```

Añadir esta línea

Implementación de la función recargarSaldo:

Esta función maneja la lógica para recargar el saldo de un cliente. Incluye:

Verificación de la sesión del administrador.

Obtención del codigoCuenta y el monto desde el formulario.

Consulta a la base de datos para obtener el id_cliente usando el codigo_cuenta.

Actualización del saldo en la tabla Cuentas.

Manejo de errores y redirección al panel de administración.

```
func recargarSaldo(c *gin.Context) {  
  
    session := sessions.Default(c)
```



```
userID := session.Get("user_id")

// Verificar si el usuario es un administrador
if userID == nil || userID != 1 {
    c.Redirect(http.StatusFound, "/login")
    return
}

codigoCuenta := c.PostForm("codigoCuenta")
monto, err := strconv.ParseFloat(c.PostForm("monto"), 64)
if err != nil {
    c.JSON(http.StatusBadRequest, gin.H{"error": "Monto inválido"})
    return
}

// Obtener el ID del cliente usando el codigo_cuenta
var idCliente int

err = db.DB.QueryRow("SELECT id FROM Clientes WHERE codigo_cuenta = ?", codigoCuenta).Scan(&idCliente)

if err != nil {
```

```

        fmt.Printf("Error al obtener el ID del cliente: %v\n",
err) // Imprimir el error en el servidor

        c.JSON(http.StatusBadRequest, gin.H{"error": "Código de
cuenta no encontrado"})

        return

    }

    // Actualizar el saldo en la base de datos

    result, err := db.DB.Exec("UPDATE Cuentas SET saldo = saldo
+ ? WHERE id_cliente = ?", monto, idCliente)

    if err != nil {

        fmt.Printf("Error al recargar el saldo: %v\n", err) //
Imprimir el error en el servidor

        c.JSON(http.StatusInternalServerError, gin.H{"error":
"Error al recargar el saldo"})

        return

    }

    // Verificar si se actualizó alguna fila

    rowsAffected, err := result.RowsAffected()

    if err != nil {

        fmt.Printf("Error al obtener filas afectadas: %v\n",
err)

```

```

        c.JSON(http.StatusInternalServerError, gin.H{"error":
"Error al verificar la actualización"})

        return

    }

    if rowsAffected == 0 {

        c.JSON(http.StatusBadRequest, gin.H{"error": "Código de
cuenta no encontrado"})

        return

    }

    c.Redirect(http.StatusFound, "/admin") // Redirigir a la
página de administración
}

```

2. Modificaciones en admin.html

Formulario para recargar saldo:

Se creó un formulario en la plantilla admin.html que permite a los administradores ingresar el codigoCuenta y el monto a recargar.

```

<!DOCTYPE html>

<html lang="en">

<head>

```

```
<meta charset="UTF-8">

<meta name="viewport" content="width=device-width,
initial-scale=1.0">

<title>Administración</title>

<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/boot
strap.min.css" rel="stylesheet">

</head>

<body>

<div class="container mt-4 mb-4 mx-auto" style="max-width:
800px;">

    <h2>Recargar Saldo de Clientes</h2>

    <form action="/recargar" method="POST">

        <div class="mb-3">

            <label for="codigoCuenta" class="form-label">Código
de Cuenta</label>

            <input type="text" class="form-control"
id="codigoCuenta" name="codigoCuenta" required>

        </div>

        <div class="mb-3">

            <label for="monto" class="form-label">Monto a
Recargar</label>
```

```

        <input type="number" class="form-control" id="monto"
name="monto" required>

    </div>

    <button type="submit" class="btn
btn-primary">Recargar</button>

</form>

</div>

<!-- Bootstrap JS y Popper.js -->

<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd
/popper.min.js"></script>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootst
rap.min.js"></script>

</body>

</html>

```

3. Modificaciones en banco.sql

Estructura de la base de datos:

Asegurarse de que las tablas Clientes y Cuentas estén correctamente definidas para almacenar la información necesaria, como el codigo_cuenta y el saldo.

```

CREATE TABLE `administradores` (

    `id` int(11) NOT NULL AUTO_INCREMENT,

    `nombre` varchar(100) NOT NULL,

```

```
`apellido` varchar(100) NOT NULL,  
`email` varchar(100) NOT NULL,  
`password` varchar(255) NOT NULL,  
`fecha_creacion` timestamp NOT NULL DEFAULT  
current_timestamp(),  
  
PRIMARY KEY (`id`),  
  
UNIQUE KEY `email` (`email`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1  
COLLATE=latin1_swedish_ci;
```

Implementación de la alerta de mensaje y el historial de recargas

1. Alerta de mensaje al recargar

Para mostrar una alerta cuando un administrador recarga el saldo de un cliente, puedes usar un mensaje flash en la sesión.

1. Modificar la función recargarSaldo

Asegúrate de que estás configurando el mensaje en la sesión después de una recarga exitosa. Aquí tienes un ejemplo de cómo hacerlo

```
func recargarSaldo(c *gin.Context) {  
  
    session := sessions.Default(c)  
  
    userID := session.Get("user_id")
```

```
// Verificar si el usuario es un administrador

if userID == nil || userID != 1 {

    c.Redirect(http.StatusFound, "/login")

    return

}

codigoCuenta := c.PostForm("codigoCuenta")

monto, err := strconv.ParseFloat(c.PostForm("monto"), 64)

if err != nil {

    c.JSON(http.StatusBadRequest, gin.H{"error": "Monto inválido"})

    return

}

// Obtener el ID del cliente usando el codigo_cuenta

var idCliente int

err = db.DB.QueryRow("SELECT id FROM Clientes WHERE codigo_cuenta = ?", codigoCuenta).Scan(&idCliente)

if err != nil {

    fmt.Printf("Error al obtener el ID del cliente: %v\n", err) // Imprimir el error en el servidor
```

```
        c.JSON(http.StatusBadRequest, gin.H{"error": "Código de
cuenta no encontrado"})

        return
    }

    // Actualizar el saldo en la base de datos

    result, err := db.DB.Exec("UPDATE Cuentas SET saldo = saldo
+ ? WHERE id_cliente = ?", monto, idCliente)

    if err != nil {

        fmt.Printf("Error al recargar el saldo: %v\n", err) //
Imprimir el error en el servidor

        c.JSON(http.StatusInternalServerError, gin.H{"error":
"Error al recargar el saldo"})

        return
    }

    // Verificar si se actualizó alguna fila

    rowsAffected, err := result.RowsAffected()

    if err != nil {

        fmt.Printf("Error al obtener filas afectadas: %v\n",
err)

        c.JSON(http.StatusInternalServerError, gin.H{"error":
"Error al verificar la actualización"})
    }
}
```



```
        return

    }

    if rowsAffected == 0 {

        c.JSON(http.StatusBadRequest, gin.H{"error": "Código de
cuenta no encontrado"})

        return

    }

    // Insertar en el historial de recargas

    _, err = db.DB.Exec("INSERT INTO historial_recargas
(id_admin, id_cliente, monto, codigo_cuenta) VALUES (?, ?, ?,
?)", userID, idCliente, monto, codigoCuenta)

    if err != nil {

        fmt.Printf("Error al registrar la recarga en el
historial: %v\n", err)

    }

    // Establecer el mensaje de éxito en la sesión

    session.Set("mensaje", "Recarga realizada con éxito.")

    session.Save()
```

```
c.Redirect(http.StatusFound, "/admin") // Redirigir a la
página de administración
}
```

```
func showAdminPage(c *gin.Context) {

    session := sessions.Default(c)

    userID := session.Get("user_id")

    // Verificar si el usuario es un administrador

    if userID == nil || userID != 1 { // Asegúrate de que el ID
1 sea el administrador

        c.Redirect(http.StatusFound, "/login")

        return

    }

    // Obtener el mensaje de la sesión

    mensaje := session.Get("mensaje")

    session.Delete("mensaje") // Eliminar el mensaje después de
mostrarlo

    c.HTML(http.StatusOK, "admin.html", gin.H{
```

```

        "mensaje": mensaje, // Pasar el mensaje a la plantilla
    }) // Cargar la plantilla de administración
}

```

Mostrar el mensaje en admin.html:

Agrega el siguiente código al inicio del archivo admin.html para mostrar el mensaje:

```

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width,
initial-scale=1.0">

    <title>Administración</title>

    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/boot
strap.min.css" rel="stylesheet">

</head>

<body>

{{ if .ShowNav }}

    {{ template "nav" . }} <!-- Incluir la plantilla de
navegación -->

{{ end }}

```

```

<div class="container mt-4 mb-4 mx-auto" style="max-width:
800px;">

    <h2>Recargar Saldo de Clientes</h2>

    {{ if .mensaje }}

    <div class="alert alert-success alert-dismissible fade show"
role="alert" id="alertMessage">

        {{ .mensaje }}

        <button type="button" class="btn-close"
data-bs-dismiss="alert" aria-label="Close"></button>

    </div>

    {{ end }}

    <form action="/recargar" method="POST">

        <div class="mb-3">

            <label for="codigoCuenta" class="form-label">Código
de Cuenta</label>

            <input type="text" class="form-control"
id="codigoCuenta" name="codigoCuenta" required>

        </div>

        <div class="mb-3">

            <label for="monto" class="form-label">Monto a
Recargar</label>

```

```
        <input type="number" class="form-control" id="monto"
name="monto" required>

    </div>

    <button type="submit" class="btn
btn-primary">Recargar</button>

</form>

</div>

<!-- Bootstrap JS y Popper.js -->

<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd
/popper.min.js"></script>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootst
rap.min.js"></script>

<script>

    // Función para ocultar el mensaje de alerta después de 5
segundos

    window.onload = function() {

        const alertMessage =
document.getElementById('alertMessage');

        if (alertMessage) {

            setTimeout(function() {
```

```
        const bsAlert = new
bootstrap.Alert(alertMessage);

        bsAlert.close();

        }, 5000); // 5000 milisegundos = 5 segundos
    }

};

</script>

</body>

</html>
```

2. Historial de recargas

Para implementar un historial de recargas, sigue estos pasos:

Crear una nueva tabla en la base de datos para almacenar el historial de recargas:

```
CREATE TABLE `historial_recargas` (

  `id` int(11) NOT NULL,

  `id_admin` int(11) NOT NULL,

  `id_cliente` int(11) NOT NULL,

  `monto` decimal(10,2) NOT NULL,

  `fecha_recarga` timestamp NOT NULL DEFAULT
current_timestamp(),

  `codigo_cuenta` varchar(255) NOT NULL
```

```
) ENGINE=InnoDB DEFAULT CHARSET=latin1
COLLATE=latin1_swedish_ci;
```

2) Crear una nueva función para mostrar el historial de recargas en la página de administración:

```
func showHistorialRecargas(c *gin.Context) {

    session := sessions.Default(c)

    userID := session.Get("user_id")

    // Verificar si el usuario es un administrador

    if userID == nil || userID != 1 {

        c.Redirect(http.StatusFound, "/login")

        return

    }

    // Modificar la consulta para obtener nombres y apellidos,
ordenado por fecha descendente

    rows, err := db.DB.Query(`

        SELECT

            a.nombre as admin_nombre,

            a.apellido as admin_apellido,

            c.nombre as cliente_nombre,

            c.apellido as cliente_apellido,
```

```

        h.monto,

        DATE_FORMAT(h.fecha_recarga, '%d/%m/%Y %H:%i') as
fecha_recarga,

        h.codigo_cuenta

FROM historial_recargas h

JOIN Administradores a ON h.id_admin = a.id

JOIN Clientes c ON h.id_cliente = c.id

ORDER BY h.fecha_recarga DESC`)

if err != nil {

    c.JSON(http.StatusInternalServerError, gin.H{"error":
"Error al obtener el historial"})

    return

}

defer rows.Close()

var historial []struct {

    AdminNombre      string

    AdminApellido    string

    ClienteNombre    string

    ClienteApellido  string

```



```
    Monto          float64

    FechaRecarga   string

    CodigoCuenta   string
}

for rows.Next() {

    var recarga struct {

        AdminNombre      string

        AdminApellido     string

        ClienteNombre     string

        ClienteApellido  string

        Monto             float64

        FechaRecarga      string

        CodigoCuenta      string

    }

    if err := rows.Scan(

        &recarga.AdminNombre,

        &recarga.AdminApellido,

        &recarga.ClienteNombre,

        &recarga.ClienteApellido,

        &recarga.Monto,
```

```

        &recarga.FechaRecarga,
        &recarga.CodigoCuenta); err != nil {
            fmt.Printf("Error al escanear el historial: %v\n",
err)

            continue
        }

        historial = append(historial, recarga)
    }

    c.HTML(http.StatusOK, "base.html", gin.H{
        "historial": historial,
        "ShowNav":    true,
        "IsAdmin":    true,
        "Title":       "Historial de Recargas",
        "Content":     "historial.html",
    })
}

```

Los cambios principales son:

Agregamos ORDER BY h.fecha_recarga DESC al final de la consulta SQL

Usamos DATE_FORMAT para formatear la fecha en un formato más legible

El DESC asegura que las recargas más recientes aparezcan primero

Esto hará que:

La recarga más reciente aparezca en la parte superior de la tabla

Las fechas se muestren en formato día/mes/año hora:minuto

El orden sea cronológicamente inverso (de más nuevo a más antiguo)

Agregar una nueva ruta para el historial en routes.go:

```
r.GET("/historial", authMiddleware(), showHistorialRecargas)
// Añadir esta línea
```

Crear una nueva plantilla historial.html para mostrar el historial de recargas:

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width,
initial-scale=1.0">

    <title>Historial de Recargas</title>

    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/boot
strap.min.css" rel="stylesheet">

</head>

<body>

{{ if .ShowNav }}
```

```

    {{ template "nav" . }} <!-- Incluir la plantilla de
navegación -->

{{ end }}

{{ define "content" }}

<div class="container mt-4 mb-4 mx-auto" style="max-width:
800px;">

    <h2>Historial de Recargas</h2>

    <table class="table">

        <thead>

            <tr>

                <th>Administrador</th>

                <th>Cliente</th>

                <th>Monto</th>

                <th>Fecha de Recarga</th>

                <th>Código de Cuenta</th>

            </tr>

        </thead>

        <tbody>

            {{ range .historial }}

                <tr>

```

```

        <td>{{ .AdminNombre }} {{ .AdminApellido }}</td>

        <td>{{ .ClienteNombre }} {{ .ClienteApellido
    }}</td>

        <td>{{ .Monto }} Bs</td>

        <td>{{ .FechaRecarga }}</td>

        <td>{{ .CodigoCuenta }}</td>

    </tr>

    {{ end }}

</tbody>

</table>

</div>

{{ end }}

<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd
/popper.min.js"></script>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootst
rap.min.js"></script>

</body>

</html>

```

Resumen Final

Implementar la alerta de mensaje al recargar saldo.

Crear un historial de recargas en la base de datos y mostrarlo en una nueva página de administración.

Con estos pasos, deberías poder implementar la funcionalidad de alerta y el historial de recargas.

creando login y registro

/banco

```
|— cmd
|   |— main.go
|— db
|   |— db.go
|— models
|   |— models.go
|— routes
|   |— routes.go
|— @templates
|   |— login.html
|   |— register.html
|— go.mod
|— go.sum
```

1. Configurar las Rutas en routes/routes.go

Actualiza tu archivo routes.go para incluir las rutas para el inicio de sesión y el registro, así como para renderizar las plantillas HTML.

```
package routes

import (
    "banco/db" // Asegúrate de importar tu paquete de base de
    datos
    "fmt"
    "net/http"
    "time"

    "github.com/gin-gonic/gin"
    "github.com/gin-gonic/gin/binding"
    "golang.org/x/crypto/bcrypt"
)

func SetupRouter() *gin.Engine {
    r := gin.Default()

    // Cargar las plantillas HTML
```

```
    r.LoadHTMLGlob("@templates/*") // Asegúrate de que esta ruta
sea correcta

    r.GET("/", func(c *gin.Context) {

        c.JSON(http.StatusOK, gin.H{"message": "Bienvenido a la
aplicación bancaria"})

    })

    r.GET("/login", showLoginPage)

    r.POST("/login", login)

    r.GET("/register", showRegisterPage)

    r.POST("/register", register)

    return r
}

func showLoginPage(c *gin.Context) {

    c.HTML(http.StatusOK, "login.html", nil)

}

func showRegisterPage(c *gin.Context) {
```



```

        c.HTML(http.StatusOK, "register.html", nil)
    }

func login(c *gin.Context) {

    var json struct {

        Email    string `json:"email"`

        Password string `json:"password"`

    }

    if err := c.ShouldBindJSON(&json); err != nil {

        c.JSON(http.StatusBadRequest, gin.H{"error": "Datos
inválidos"})

        return

    }

}

func register(c *gin.Context) {

    var json struct {

        Nombre    string `form:"nombre"
binding:"required"`

```

```

        Apellido      string      `form:"apellido"
binding:"required"`

        Email         string      `form:"email"
binding:"required,email"`

        Password      string      `form:"password"
binding:"required,min=8"`

        FechaNacimiento time.Time `form:"fecha_nacimiento"
binding:"required" time_format:"2006-01-02"`
    }

    if err := c.ShouldBindWith(&json, binding.Form); err != nil
{
        c.JSON(http.StatusBadRequest, gin.H{"error": "Datos
inválidos: " + err.Error()})

        return
    }

    // Convertir el string de fecha a time.Time

    fecha, err := time.Parse("2006-01-02",
c.PostForm("fecha_nacimiento"))

    if err != nil {

        c.JSON(http.StatusBadRequest, gin.H{"error": "Formato de
fecha inválido. Use YYYY-MM-DD"})

        return
    }

```

```

    }

    json.FechaNacimiento = fecha

    // Insertar el nuevo cliente en la base de datos

    query := "INSERT INTO Clientes (nombre, apellido, email,
password, fecha_nacimiento) VALUES (?, ?, ?, ?, ?)"

    hashedPassword, err := bcrypt.GenerateFromPassword(
        []byte(json.Password),
        bcrypt.DefaultCost, // Produce hash de 60 caracteres
    )

    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error":
"Error al registrar el usuario"})
        return
    }

    _, err = db.DB.Exec(query, json.Nombre, json.Apellido,
json.Email, hashedPassword, json.FechaNacimiento)

    if err != nil {
        fmt.Printf("ERROR SQL: %v\nQUERY: %s\nPARAMS: %s, %s,
%s, [hashed], %s\n",

```

```
        err, query, json.Nombre, json.Apellido, json.Email,
json.FechaNacimiento)

        c.JSON(http.StatusInternalServerError, gin.H{
            "error": "Error en el servidor. Por favor intenta
más tarde",
        })

        return
    }

    c.JSON(http.StatusOK, gin.H{"message": "Registro exitoso"})
}
```

creando alerta del registro con exito redirección a login

```
// Establecer el mensaje de éxito en la sesión

session := sessions.Default(c)
```

```
session.Set("mensaje", "¡Registro exitoso! Por favor inicia sesión")

session.Save()

// Redirigir a la página de login

c.Redirect(http.StatusSeeOther, "/")
}
```

Después de un registro exitoso, establezca el mensaje en la sesión

```
// Establecer el mensaje de éxito en la sesión

session := sessions.Default(c)

session.Set("mensaje", "¡Registro exitoso! Por favor inicia sesión")

session.Save()

// Redirigir a la página de login

c.Redirect(http.StatusSeeOther, "/")
```

Luego redirija a la página de login

```
// Redirigir a la página de login

c.Redirect(http.StatusSeeOther, "/")
```

Ahora el flujo será:

Usuario se registra exitosamente

Se establece el mensaje de éxito en la sesión

Se redirige a la página de login

La página de login muestra el mensaje de la sesión

El mensaje se limpia después de mostrarlo

Esto debería resolver el problema y ahora deberías ver el mensaje "¡Registro exitoso! Por favor inicia sesión" después de registrarte.

creando las plantillas

2. Crear las Plantillas HTML

Asegúrate de que tus archivos HTML en la carpeta @templates estén configurados correctamente. Aquí tienes ejemplos básicos para login.html y register.html.

@templates/login.html

```
<!DOCTYPE html>

<html lang="es">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width,
initial-scale=1.0">

    <title>Iniciar Sesión</title>

</head>

<body>
```

```
<h1>Iniciar Sesión</h1>

<form action="/login" method="post">

    <label for="email">Email:</label>

    <input type="email" id="email" name="email"
required>

    <br>

    <label for="password">Contraseña:</label>

    <input type="password" id="password"
name="password" required>

    <br>

    <button type="submit">Iniciar Sesión</button>

</form>

<p>¿No tienes una cuenta? <a
href="/register">Regístrate aquí</a></p>

</body>

</html>
```

@templates/register.html

```
<!DOCTYPE html>
```

```
<html lang="es">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width,
initial-scale=1.0">

    <title>Registro</title>

</head>

<body>

    <h1>Registro</h1>

    <form action="/register" method="post">

        <label for="nombre">Nombre:</label>

        <input type="text" id="nombre" name="nombre" required>

        <br>

        <label for="apellido">Apellido:</label>

        <input type="text" id="apellido" name="apellido"
required>

        <br>

        <label for="email">Email:</label>

        <input type="email" id="email" name="email" required>

        <br>

        <label for="password">Contraseña:</label>
```



```
<input
    type="password"
    id="password"
    name="password"
    required
    minlength="8"
    pattern=".*\d.*"
>

<small>Mínimo 8 caracteres con al menos un
número</small>

<br>

<label for="fecha_nacimiento">Fecha de
Nacimiento:</label>

<input
    type="date"
    id="fecha_nacimiento"
    name="fecha_nacimiento"
    required
    pattern="\d{4}-\d{2}-\d{2}"
>

<br>
```

```
<button type="submit">Registrarse</button>

</form>

<p>¿Ya tienes una cuenta? <a href="/login">Inicia sesión
aquí</a></p>

</body>

</html>
```

3. Actualizar cmd/main.go

Asegúrate de que tu archivo main.go esté configurado para usar las rutas que acabas de definir.

```
package main

import (

    "banco/db"

    "banco/routes"

)

func main() {

    db.InitDB()

    defer db.CloseDB()

    r := routes.SetupRouter()
```

```
r.Run(":8080") // Inicia el servidor en el puerto 8080
}
```

guardando los registros en la db

Para guardar los registros en la base de datos, necesitas realizar algunos cambios en tu función register. A continuación, te muestro cómo puedes hacerlo:

Pasos para Guardar Registros en la Base de Datos

Conectar a la Base de Datos: Asegúrate de que ya tienes la conexión a la base de datos establecida en tu archivo db.go.

Modificar la Función register: Agrega la lógica para insertar los datos del nuevo cliente en la base de datos.

Código Modificado

Aquí tienes un ejemplo de cómo podría verse la función register después de realizar los cambios necesarios:

```
package routes

import (
    "banco/db" // Asegúrate de importar tu paquete de base de
    datos
    "fmt"
    "net/http"
    "time"
```

```
    "github.com/gin-gonic/gin"
    "github.com/gin-gonic/gin/binding"
    "golang.org/x/crypto/bcrypt"
)

func SetupRouter() *gin.Engine {
    r := gin.Default()

    // Cargar las plantillas HTML
    r.LoadHTMLGlob("@templates/*") // Asegúrate de que esta ruta
    sea correcta

    r.GET("/", func(c *gin.Context) {
        c.JSON(http.StatusOK, gin.H{"message": "Bienvenido a la
    aplicación bancaria"})
    })

    r.GET("/login", showLoginPage)
    r.POST("/login", login)
    r.GET("/register", showRegisterPage)
```

```
r.POST("/register", register)

return r
}

func showLoginPage(c *gin.Context) {
    c.HTML(http.StatusOK, "login.html", nil)
}

func showRegisterPage(c *gin.Context) {
    c.HTML(http.StatusOK, "register.html", nil)
}

func login(c *gin.Context) {
    var json struct {
        Email    string `json:"email"`
        Password string `json:"password"`
    }

    if err := c.ShouldBindJSON(&json); err != nil {
```

```

        c.JSON(http.StatusBadRequest, gin.H{"error": "Datos
inválidos"})

        return

    }

}

func register(c *gin.Context) {

    var json struct {

        Nombre          string    `form:"nombre"
binding:"required"`

        Apellido         string    `form:"apellido"
binding:"required"`

        Email            string    `form:"email"
binding:"required,email"`

        Password         string    `form:"password"
binding:"required,min=8"`

        FechaNacimiento time.Time `form:"fecha_nacimiento"
binding:"required" time_format:"2006-01-02"`

    }

    if err := c.ShouldBindWith(&json, binding.Form); err != nil
{

```

```
c.JSON(http.StatusBadRequest, gin.H{"error": "Datos
inválidos: " + err.Error()})

return

}

// Convertir el string de fecha a time.Time

fecha, err := time.Parse("2006-01-02",
c.PostForm("fecha_nacimiento"))

if err != nil {

    c.JSON(http.StatusBadRequest, gin.H{"error": "Formato de
fecha inválido. Use YYYY-MM-DD"})

    return

}

json.FechaNacimiento = fecha

// Insertar el nuevo cliente en la base de datos

query := "INSERT INTO Clientes (nombre, apellido, email,
password, fecha_nacimiento) VALUES (?, ?, ?, ?, ?)"

hashedPassword, err := bcrypt.GenerateFromPassword(

    []byte(json.Password),

    bcrypt.DefaultCost, // Produce hash de 60 caracteres
```

```

    )

    if err != nil {

        c.JSON(http.StatusInternalServerError, gin.H{"error":
"Error al registrar el usuario"})

        return

    }

    _, err = db.DB.Exec(query, json.Nombre, json.Apellido,
json.Email, hashedPassword, json.FechaNacimiento)

    if err != nil {

        fmt.Printf("ERROR SQL: %v\nQUERY: %s\nPARAMS: %s, %s,
%s, [hashed], %s\n",
            err, query, json.Nombre, json.Apellido, json.Email,
json.FechaNacimiento)

        c.JSON(http.StatusInternalServerError, gin.H{

            "error": "Error en el servidor. Por favor intenta
más tarde",

        })

        return

    }

    c.JSON(http.StatusOK, gin.H{"message": "Registro exitoso"})

```



```
}
```

seguridad contrasena

Consideraciones de Seguridad

Hash de Contraseña: Es recomendable que no almacenes las contraseñas en texto plano. Considera usar una función de hash (como bcrypt) para almacenar las contraseñas de manera segura.

Ejemplo de Hash de Contraseña

Si decides implementar el hash de la contraseña, puedes hacerlo de la siguiente manera:

- 1) Modificar db.go: Cambia la declaración de la variable db para que sea exportada. Esto se hace cambiando el nombre de la variable a DB (con mayúscula).

```
package db

import (
    "database/sql"
    "log"

    _ "github.com/go-sql-driver/mysql"
```

```
)

var DB *sql.DB

func InitDB() {

    var err error

    // Cambia "root:@tcp(127.0.0.1:3306)/Banco" por
    tus credenciales

    DB, err = sql.Open("mysql",
"root:@tcp(127.0.0.1:3306)/Banco")

    if err != nil {

        log.Fatal(err)

    }

    if err = DB.Ping(); err != nil {

        log.Fatal(err)

    }

}
```

```
func CloseDB() {  
  
    if DB != nil {  
  
        DB.Close()  
  
    }  
  
}
```

2) Modificar routes.go: Cambia la llamada a db.Exec por db.DB.Exec para usar la variable exportada.

```
package routes  
  
import (  
  
    "banco/db" // Asegúrate de importar tu paquete de base de  
datos  
  
    "fmt"  
  
    "net/http"  
  
    "time"  
  
    "github.com/gin-gonic/gin"  
  
    "github.com/gin-gonic/gin/binding"  
  
    "golang.org/x/crypto/bcrypt"
```

```
)

func SetupRouter() *gin.Engine {

    r := gin.Default()

    // Cargar las plantillas HTML

    r.LoadHTMLGlob("@templates/*") // Asegúrate de que esta ruta
sea correcta

    r.GET("/", func(c *gin.Context) {

        c.JSON(http.StatusOK, gin.H{"message": "Bienvenido a la
aplicación bancaria"})

    })

    r.GET("/login", showLoginPage)

    r.POST("/login", login)

    r.GET("/register", showRegisterPage)

    r.POST("/register", register)

    return r

}
```

```
func showLoginPage(c *gin.Context) {  
    c.HTML(http.StatusOK, "login.html", nil)  
}  
  
func showRegisterPage(c *gin.Context) {  
    c.HTML(http.StatusOK, "register.html", nil)  
}  
  
func login(c *gin.Context) {  
    var json struct {  
        Email    string `json:"email"`  
        Password string `json:"password"`  
    }  
  
    if err := c.ShouldBindJSON(&json); err != nil {  
        c.JSON(http.StatusBadRequest, gin.H{"error": "Datos  
inválidos"})  
        return  
    }  
}
```

```

}

func register(c *gin.Context) {
    var json struct {
        Nombre      string    `form:"nombre"
binding:"required"`
        Apellido    string    `form:"apellido"
binding:"required"`
        Email       string    `form:"email"
binding:"required,email"`
        Password    string    `form:"password"
binding:"required,min=8"`
        FechaNacimiento time.Time `form:"fecha_nacimiento"
binding:"required" time_format:"2006-01-02"`
    }

    if err := c.ShouldBindWith(&json, binding.Form); err != nil
    {
        c.JSON(http.StatusBadRequest, gin.H{"error": "Datos
inválidos: " + err.Error()})

        return
    }
}

```

```
// Convertir el string de fecha a time.Time

fecha, err := time.Parse("2006-01-02",
c.PostForm("fecha_nacimiento"))

if err != nil {

    c.JSON(http.StatusBadRequest, gin.H{"error": "Formato de
fecha inválido. Use YYYY-MM-DD"})

    return

}

json.FechaNacimiento = fecha

// Insertar el nuevo cliente en la base de datos

query := "INSERT INTO Clientes (nombre, apellido, email,
password, fecha_nacimiento) VALUES (?, ?, ?, ?, ?)"

hashedPassword, err := bcrypt.GenerateFromPassword(

    []byte(json.Password),

    bcrypt.DefaultCost, // Produce hash de 60 caracteres

)

if err != nil {

    c.JSON(http.StatusInternalServerError, gin.H{"error":
"Error al registrar el usuario"})

    return
}
```

```

    }

    _, err = db.DB.Exec(query, json.Nombre, json.Apellido,
json.Email, hashedPassword, json.FechaNacimiento)

    if err != nil {

        fmt.Printf("ERROR SQL: %v\nQUERY: %s\nPARAMS: %s, %s,
%s, [hashed], %s\n",

            err, query, json.Nombre, json.Apellido, json.Email,
json.FechaNacimiento)

        c.JSON(http.StatusInternalServerError, gin.H{

            "error": "Error en el servidor. Por favor intenta
más tarde",

            })

        return

    }

    c.JSON(http.StatusOK, gin.H{"message": "Registro exitoso"})
}

```

Resumen

Cambia la variable db a DB en db.go para que sea exportada.

Usa db.DB.Exec en routes.go para ejecutar la consulta.

Después de realizar estos cambios, intenta ejecutar tu aplicación nuevamente.

se creó la función de guardado de registro en la db falta el redireccionamiento a login y la función de login

avance mutación 2 fase de la estructura del app

banco/

- |— @templates/
 - | |— cuenta.html
 - | |— login.html
 - | |— register.html
- |— db/
 - | |— db.go
- |— doc/
 - | |— notas.txt
- |— model/
 - | |— models.go
- |— routes/
 - | |— routes.go
- |— test/
 - | |— test.go
- |— go.mod
- |— go.sum
- |— main.go

actualizando dependencias

¡Entendido! Para descargar todas las dependencias de tu proyecto Go, sigue estos pasos:

```
# Descarga todas las dependencias
go mod tidy
```

creando redireccionamiento de registro a login

Ahora, al completar el registro exitosamente, el usuario será redirigido a la página de login con un mensaje de confirmación. Aquí están los cambios necesarios:

1. Actualizar el registro (routes.go):

```
// ... código existente ...

if err != nil {

    // Manejo de errores

}

// Redireccionar con parámetro de éxito

c.Redirect(http.StatusSeeOther, "/login?registro=exitoso")

return
```

2. Actualizar la plantilla de login (login.html):

```
<!DOCTYPE html>

<html lang="es">

<head>

    <!-- ... meta tags ... -->

</head>
```

```
<body>

    {{if .mensaje}}

    <div class="alert alert-success">

        {{.mensaje}}

    </div>

    {{end}}

    <h1>Iniciar Sesión</h1>

    <!-- ... formulario ... -->

</body>

</html>
```

3. Actualizar el handler de login (routes.go):

```
func showLoginPage(c *gin.Context) {

    mensaje := ""

    if c.Query("registro") == "exitoso" {

        mensaje = "¡Registro exitoso! Por favor inicia sesión"

    }

}
```

```
c.HTML(http.StatusOK, "login.html", gin.H{
    "mensaje": mensaje,
})
}
```

Estos cambios harán que:

Después del registro exitoso, el usuario sea redirigido a /login

Se muestre un mensaje verde de confirmación

El mensaje desaparece al recargar la página

sistema de autenticación

1. Middleware de autenticación (routes.go):

```
// Añadir al inicio del archivo

import (
    "github.com/gin-contrib/sessions"
    "github.com/gin-contrib/sessions/cookie"
)

func SetupRouter() *gin.Engine {
    r := gin.Default()

    // Configurar sesiones seguras
```

```
store :=
cookie.NewStore([]byte("secreto-muy-seguro-32-caracteres"))

store.Options(sessions.Options{

    MaxAge:    86400 * 7, // 7 días

    HttpOnly: true,

    Secure:    true, // Solo HTTPS

    SameSite: http.SameSiteStrictMode,

})

r.Use(sessions.Sessions("banco-session", store))

// ... resto de configuraciones ...

}
```

2. Protección de rutas (routes.go):

```
func authMiddleware() gin.HandlerFunc {

    return func(c *gin.Context) {

        session := sessions.Default(c)

        userID := session.Get("user_id")

        if userID == nil {
```

```

        c.Redirect(http.StatusFound, "/login")

        c.Abort()

        return

    }

    c.Next()

}

}

// En SetupRouter()
r.GET("/cuenta", authMiddleware(), func(c *gin.Context) {

    c.HTML(http.StatusOK, "cuenta.html", nil)

}))

```

3. Sistema de login seguro (routes.go):

```

func login(c *gin.Context) {

    var json struct {

        Email    string `form:"email" binding:"required,email"`

        Password string `form:"password"
binding:"required,min=8"`

    }

```

```

        if err := c.ShouldBind(&json); err != nil {

            c.HTML(http.StatusBadRequest, "login.html",
gin.H{"error": "Credenciales inválidas"})

            return

        }

        var cliente model.Cliente

        err := db.DB.QueryRow("SELECT id, password FROM Clientes
WHERE email = ?", json.Email).Scan(&cliente.ID,
&cliente.Password)

        if err != nil ||
bcrypt.CompareHashAndPassword([]byte(cliente.Password),
[]byte(json.Password)) != nil {

            c.HTML(http.StatusUnauthorized, "login.html",
gin.H{"error": "Credenciales incorrectas"})

            return

        }

        session := sessions.Default(c)

        session.Set("user_id", cliente.ID)

        session.Save()

```

```
c.Redirect(http.StatusFound, "/cuenta")
}
```

4. Logout seguro (routes.go):

```
r.GET("/logout", func(c *gin.Context) {
    session := sessions.Default(c)
    session.Clear()
    session.Options(sessions.Options{MaxAge: -1})
    session.Save()
    c.Redirect(http.StatusFound, "/login")
}))
```

CREANDO PLANTILLAS DE NAV

Paso 1: Configurar la carga de plantillas

Asegúrate de que estás cargando las plantillas correctamente en tu aplicación Gin. Aquí te muestro cómo hacerlo:

```
func SetupRouter() *gin.Engine {
    r := gin.Default()
```



```
// Configurar sesiones seguras

store :=
cookie.NewStore([]byte("secreto-muy-seguro-32-caracteres"))

store.Options(sessions.Options{

    MaxAge: 86400 * 7, // 7 días

    HttpOnly: true,

    Secure: true, // Solo HTTPS

    SameSite: http.SameSiteStrictMode,

})

r.Use(sessions.Sessions("banco-session", store))


// Cargar las plantillas HTML

r.LoadHTMLGlob("@templates/*") // Asegúrate de que esta ruta
sea correcta


r.GET("/", func(c *gin.Context) {

    c.JSON(http.StatusOK, gin.H{"message": "Bienvenido a la
aplicación bancaria"})

})


r.GET("/login", showLoginPage)
```



```
r.POST("/login", login)

r.GET("/register", showRegisterPage)

r.POST("/register", register)


r.GET("/cuenta", authMiddleware(), func(c *gin.Context) {

    c.HTML(http.StatusOK, "cuenta.html", nil)

})


r.GET("/logout", func(c *gin.Context) {

    session := sessions.Default(c)

    session.Clear()

    session.Options(sessions.Options{MaxAge: -1})

    session.Save()

    c.Redirect(http.StatusFound, "/login")

})


return r

}
```

Paso 2: Crear una plantilla base

Creas una plantilla base que incluya el menú de navegación y otras partes comunes de tu sitio. Puedes usar la directiva `{{ template }}` para incluir otras plantillas dentro de esta.

Asegúrate de que la plantilla nav.html solo se incluya en cuenta.html y no en login.html. Esto se puede controlar desde la plantilla base base.html.

base.html

Asegúrate de que la plantilla nav.html solo se incluya cuando sea necesario. Si estás usando base.html como plantilla base para todas las páginas, necesitas condicionar la inclusión de nav.html.

base.html

```
<!DOCTYPE html>

<html lang="es">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width,
initial-scale=1.0">

    <title>{{ .Title }}</title>

    <!-- Bootstrap CSS -->

    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/boot
strap.min.css" rel="stylesheet">

</head>

<body>

    <!-- Incluir la plantilla de navegación solo si está
definida -->

    {{ if .ShowNav }}

        {{ template "nav" . }}
```

```

    {{ end }}

    <div class="container">

        {{ block "content" . }}{{ end }}

    </div>

    <!-- Bootstrap JS y Popper.js -->

    <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd
/popper.min.js"></script>

    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootst
rap.min.js"></script>

</body>

</html>

```

Paso 3: Usar la plantilla base en otras plantillas

Ahora puedes usar la plantilla base en otras plantillas, como cuenta.html.

```

{{ define "content" }}

<h1>Bienvenido</h1>

{{ end }}

```

Paso 4: Renderizar las plantillas

Cuando renderices una página, asegúrate de pasar el nombre de la plantilla base y los datos necesarios, Supongamos que tienes una plantilla base llamada base.html que incluye el contenido de otras plantillas. Aquí está cómo podrías renderizar la página de inicio de sesión:

```
func showCuentaPage(c *gin.Context) {  
  
    c.HTML(http.StatusOK, "base.html", gin.H{  
  
        "Title":    "Cuenta",  
  
        "ShowNav": true, // Mostrar la navegación  
  
    })  
  
}
```

CREANDO EL NAV EN CUENTA.HTML

Modificar nav.html para Centrar los Elementos

Puedes utilizar las clases de Bootstrap para centrar los elementos dentro del contenedor de la barra de navegación. Aquí tienes un ejemplo de cómo hacerlo:

```
{{ define "nav" }}  
  
<nav class="navbar navbar-expand-lg navbar-light bg-light mt-4  
mb-4 mx-4">  
  
    <div class="container-fluid">  
  
        <a class="navbar-brand" href="#">MiApp</a>
```

```

        <button class="navbar-toggler" type="button"
data-bs-toggle="offcanvas" data-bs-target="#offcanvasNavbar"
aria-controls="offcanvasNavbar">

            <span class="navbar-toggler-icon"></span>

        </button>

        <div class="offcanvas offcanvas-start" tabindex="-1"
id="offcanvasNavbar" aria-labelledby="offcanvasNavbarLabel">

            <div class="offcanvas-header">

                <h5 class="offcanvas-title"
id="offcanvasNavbarLabel">MiApp</h5>

                <button type="button" class="btn-close"
data-bs-dismiss="offcanvas" aria-label="Close"></button>

            </div>

            <div class="offcanvas-body">

                <ul class="navbar-nav justify-content-end
flex-grow-1 pe-3">

                    <li class="nav-item">

                        <a class="nav-link active"
aria-current="page" href="#">Inicio</a>

                    </li>

                    <li class="nav-item">

                        <a class="nav-link" href="#">Usuario</a>

                    </li>

```

```

        <li class="nav-item">
            <a class="nav-link" href="#">Cuenta</a>
        </li>

        <li class="nav-item">
            <a class="nav-link"
href="#">Transacción</a>
        </li>
    </ul>

    <ul class="navbar-nav ms-auto">
        <li class="nav-item">
            <a class="nav-link"
href="/logout">Cerrar Sesión</a>
        </li>
    </ul>
</div>
</div>
</div>
</nav>
{{ end }}

```

Explicación

offcanvas-start: Esta clase hace que el menú offcanvas se deslice desde la izquierda.

navbar-toggler: El botón que activa el menú offcanvas en dispositivos móviles.

offcanvas-body: Contiene los elementos del menú que se mostrarán en el offcanvas.

CREANDO EL BOTÓN DE CIERRE DE SESION EN EL NAV EN CUENTA.HTML

Para agregar un enlace de "Cerrar Sesión" en la esquina derecha de la barra de navegación, puedes utilizar las utilidades de Bootstrap para alinear elementos a la derecha. Aquí te muestro cómo hacerlo:

```
{{ define "nav" }}  
  
<nav class="navbar navbar-expand-lg navbar-light bg-light mt-4  
mb-4 mx-4">  
  
    <div class="container-fluid">  
  
        <a class="navbar-brand" href="#">MiApp</a>  
  
        <button class="navbar-toggler" type="button"  
data-bs-toggle="offcanvas" data-bs-target="#offcanvasNavbar"  
aria-controls="offcanvasNavbar">  
  
            <span class="navbar-toggler-icon"></span>  
  
        </button>  
  
        <div class="offcanvas offcanvas-start" tabindex="-1"  
id="offcanvasNavbar" aria-labelledby="offcanvasNavbarLabel">  
  
            <div class="offcanvas-header">  
  
                <h5 class="offcanvas-title"  
id="offcanvasNavbarLabel">MiApp</h5>  
  
                <button type="button" class="btn-close"  
data-bs-dismiss="offcanvas" aria-label="Close"></button>
```



```
</div>

<div class="offcanvas-body">

    <ul class="navbar-nav justify-content-end
flex-grow-1 pe-3">

        <li class="nav-item">

            <a class="nav-link active"
aria-current="page" href="#">Inicio</a>

        </li>

        <li class="nav-item">

            <a class="nav-link" href="#">Usuario</a>

        </li>

        <li class="nav-item">

            <a class="nav-link" href="#">Cuenta</a>

        </li>

        <li class="nav-item">

            <a class="nav-link"
href="#">Transacción</a>

        </li>

    </ul>

    <ul class="navbar-nav ms-auto">

        <li class="nav-item">
```

```

                <a class="nav-link"
href="/logout">Cerrar Sesión</a>

            </li>

        </ul>

    </div>

</div>

</div>

</div>
</nav>
{{ end }}

```

Explicación

ms-auto: Esta clase se aplica al segundo ul que contiene el enlace de "Cerrar Sesión". Utiliza márgenes automáticos a la izquierda para empujar el contenido hacia la derecha.

Enlace de "Cerrar Sesión": Asegúrate de que el enlace /logout esté correctamente configurado en tus rutas para manejar la acción de cerrar sesión.

Verificar el Resultado

Después de realizar estos cambios, revisa el resultado en tu navegador. El enlace de "Cerrar Sesión" debería aparecer en la esquina derecha de la barra de navegación.

Configuración de la Ruta de Cerrar Sesión

Asegúrate de que la ruta /logout esté definida en tu aplicación para manejar el cierre de sesión. Aquí tienes un ejemplo de cómo podría verse en tu archivo routes.go:

```

r.GET("/logout", func(c *gin.Context) {

    session := sessions.Default(c)

```

```
session.Clear()

session.Options(sessions.Options{MaxAge: -1})

session.Save()

c.Redirect(http.StatusFound, "/login")

})
```

Resumen

Condiciona la inclusión de nav.html: Usa una variable como ShowNav para controlar si la navegación debe mostrarse.

Configura las vistas: Asegúrate de que las vistas pasen el valor correcto para ShowNav dependiendo de si la navegación debe mostrarse o no.

Con estos cambios, la plantilla de navegación solo se mostrará en las páginas donde ShowNav esté configurado como true.

creando cuerpo de la página cuenta.html

Para crear un conjunto de grid en cuenta.html que muestre el saldo, el nombre del cliente y un apartado para el historial de transacciones, puedes utilizar las utilidades de Bootstrap para crear un diseño de cuadrícula. Aquí te muestro cómo hacerlo:

Paso 1: Modificar cuenta.html

Vamos a definir el contenido de cuenta.html utilizando Bootstrap para crear un diseño de cuadrícula que incluya las secciones que necesitas.

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width,
initial-scale=1.0">

    <title>Credenciales del Usuario</title>

    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/boot
strap.min.css" rel="stylesheet">

    <link href="/assets/css/style-persnalizado.css"
rel="stylesheet">

</head>

<body>

{{ template "nav" . }} <!-- Incluir la plantilla de navegación
-->
```

```

<div class="container mt-4 mb-4 mx-auto" style="max-width:
800px;">

    <div class="row">

        <!-- Sección de Información del Cliente -->

        <div class="col-12">

            <div class="card mb-4">

                <div class="card-header ">

                    <h4 class="black">Información del
Cliente</h4>

                </div>

                <div class="card-body ">

                    <h5 class="card-title black">Nombre del
Cliente</h5>

                    <div class="profile-header">

                        <div class="row align-items-center">

                            <div class="col-auto profile-image">

                            </div>

                            <div class="col ml-md-n2
profile-user-info">

```

```

                                <h4 class="user-name mb-3">{{
.Nombre }} {{ .Apellido }}</h4>

                                </div>

                                </div>

                                </div>

                                <h5 class="card-title black">Saldo</h5>

                                <div class="profile-header">

                                    <div class="row align-items-center">

                                        <div class="col-auto profile-image">

                                        </div>

                                        <div class="col ml-md-n2
profile-user-info">

                                            <h4 class="card-text black">{{
.Saldo }}</h4>

                                        </div>

                                    </div>

                                </div>

                                </div>

                                <div class="card-footer">

```

```

        <form action="/abrirCuenta" method="POST">

            <button type="submit" class="btn
btn-primary black">Abrir Cuenta</button>

        </form>

    </div>

    {{ end }}

</div>

</div>

<!-- Sección de Historial de Transacciones -->

<div class="col-12">

    <div class="card">

        <div class="card-header ">

            <h4 class="black">Historial de
Transacciones</h4>

        </div>

        <div class="card-body ">

            <table class="table">

                <thead>

                    <tr>

                        <th class="black">Fecha</th>

```

```

        <th
class="black">Descripción</th>

        <th class="black">Tipo</th>

        <th class="black">Usuario</th>

        <th class="black">Código
Cuenta</th>

        <th class="black">Monto</th>

    </tr>

</thead>

<tbody class="black">

    {{ if .Transacciones }}

        {{ range .Transacciones }}

            <tr>

                <td class="black">{{ .Fecha
}}</td>

                <td class="black">{{
.Descripcion }}</td>

                <td class="black">{{
.TipoOperacion }}</td>

                <td class="black">{{
.NombreOtro }} {{ .ApellidoOtro }}</td>

                <td class="black">{{
.CodigoCuenta }}</td>

```



```

        <td class="black">{{ .Monto
}} Bs</td>

    </tr>

    {{ end }}

    {{ else }}

    <tr>

        <td colspan="6"
class="text-center black">No hay transacciones disponibles.</td>

    </tr>

    {{ end }}

</tbody>

</table>

</div>

</div>

</div>

</div>

</div>

<!-- Bootstrap JS y Popper.js -->

<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd
/popper.min.js"></script>

```

```
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.min.js"></script>

</body>

</html>
```

Explicación

col-12: Utiliza col-12 para que cada sección ocupe todo el ancho disponible del contenedor, asegurando que estén una debajo de la otra.

mx-auto: Esta clase se utiliza para centrar el contenedor horizontalmente en la página.

style="max-width: 800px;": Limita el ancho máximo del contenedor para que no se extienda demasiado en pantallas grandes, manteniendo el contenido centrado y legible.

Paso 2: Pasar Datos desde la Vista

Asegúrate de que la vista showCuentaPage pase los datos necesarios al renderizar la plantilla cuenta.html. Aquí tienes un ejemplo de cómo podrías hacerlo:

```
func showCuentaPage(c *gin.Context) {

    // Supongamos que obtienes estos datos de la base de datos

    clienteNombre := "Juan Pérez"

    saldo := 1500.00

    transacciones := []struct {

        Fecha      string

        Descripcion string

        Monto       float64

    }{
```

```

        {"2023-10-01", "Depósito", 500.00},

        {"2023-10-05", "Pago de Servicios", -100.00},

        {"2023-10-10", "Transferencia", -200.00},

    }

    c.HTML(http.StatusOK, "base.html", gin.H{

        "Title":          "Cuenta",

        "ShowNav":        true,

        "ClienteNombre":  clienteNombre,

        "Saldo":          saldo,

        "Transacciones": transacciones,

    })

}

```

Explicación

Grid de Bootstrap: Utilizamos las clases de Bootstrap para crear un diseño de cuadrícula (row y col-md-*) que divide la página en secciones.

Tarjetas de Bootstrap: Usamos tarjetas (card) para mostrar la información del cliente y el historial de transacciones.

Tabla de Transacciones: La tabla muestra el historial de transacciones, iterando sobre los datos pasados desde la vista.

Explicación

mt-4 y mb-4: Estas clases aplican un margen superior e inferior de 1.5rem (aproximadamente 24px) al contenedor.

mx-4: Esta clase aplica un margen lateral de 1.5rem (aproximadamente 24px) al contenedor, centrando el contenido horizontalmente dentro de la página.

separando los botones de nav en la página cuenta.html

```
{{ define "nav" }}  
  
<nav class="navbar navbar-expand-lg navbar-light mt-4 mb-4"  
style="max-width: 780px; margin: auto;">  
  
    <div class="container-fluid">  
  
        <a class="navbar-brand" href="#" style="font-size: 1rem;  
background-color: #007bff; color: white; padding: 5px 10px;  
border-radius: 5px;">MiApp</a> <!-- Aplicar color de fondo y  
estilo -->  
  
        <button class="navbar-toggler" type="button"  
data-bs-toggle="offcanvas" data-bs-target="#offcanvasNavbar"  
aria-controls="offcanvasNavbar">  
  
            <span class="navbar-toggler-icon"></span>  
  
        </button>  
  
        <div class="offcanvas offcanvas-start" tabindex="-1"  
id="offcanvasNavbar" aria-labelledby="offcanvasNavbarLabel">  
  
            <div class="offcanvas-header">
```

```

        <h5 class="offcanvas-title"
id="offcanvasNavbarLabel" style="font-size: 1rem;
background-color: #007bff; color: white; padding: 5px 10px;
border-radius: 5px;">MiApp</h5> <!-- Aplicar color de fondo y
estilo -->

        <button type="button" class="btn-close"
data-bs-dismiss="offcanvas" aria-label="Close"></button>

    </div>

    <div class="offcanvas-body">

        <ul class="navbar-nav justify-content-end
flex-grow-1 pe-3">

            <li class="nav-item" style="margin-right:
15px;">

                <a class="nav-link" href="#"
style="background-color: #007bff; color: white; padding: 5px
10px; border-radius: 5px;">Inicio</a>

            </li>

            <li class="nav-item" style="margin-right:
15px;">

                <a class="nav-link" href="#"
style="background-color: #007bff; color: white; padding: 5px
10px; border-radius: 5px;">Usuario</a>

            </li>

            <li class="nav-item" style="margin-right:
15px;">

```

```

        <a class="nav-link" href="#"
style="background-color: #007bff; color: white; padding: 5px
10px; border-radius: 5px;">Cuenta</a>

    </li>

    <li class="nav-item" style="margin-right:
15px;">

        <a class="nav-link" href="#"
style="background-color: #007bff; color: white; padding: 5px
10px; border-radius: 5px;">Transacción</a>

    </li>

</ul>

<ul class="navbar-nav ms-auto">

    <li class="nav-item">

        <a class="nav-link" href="/logout"
style="background-color: #007bff; color: white; padding: 5px
10px; border-radius: 5px;">Cerrar Sesión</a>

    </li>

</ul>

</div>

</div>

</div>

</nav>

{{ end }}

```

CREANDO BOTÓN PARA ABRIR CUENTA EN EL CUENTA.HTML

Para implementar la funcionalidad de permitir que cada cliente registre una cuenta y que esta cuenta esté sincronizada con el código de cuenta que el administrador puede recargar, debes seguir varios pasos. A continuación, te detallo cómo hacerlo:

1. Modificar la tabla Cuentas

Asegúrate de que la tabla Cuentas tenga los campos necesarios para almacenar el ID del cliente y el saldo. Aquí tienes un ejemplo de cómo podría verse la estructura de la tabla:

```
CREATE TABLE Cuentas (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    id_cliente INT NOT NULL,  
    tipo_cuenta VARCHAR(50),  
    saldo FLOAT DEFAULT 0,  
    fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    FOREIGN KEY (id_cliente) REFERENCES Clientes(id)  
);
```

2. Crear la función para abrir una cuenta

Agrega una nueva función en tu archivo routes.go que permita a los clientes abrir una cuenta. Esta función debe insertar un nuevo registro en la tabla Cuentas con el ID del cliente.

Aquí tienes un ejemplo de cómo podría verse la función:

```
func abrirCuenta(c *gin.Context) {  
  
    session := sessions.Default(c)  
  
    userID := session.Get("user_id")  
  
    if userID == nil {  
  
        c.Redirect(http.StatusFound, "/login")  
  
        return  
  
    }  
  
    // Insertar una nueva cuenta para el cliente  
  
    query := "INSERT INTO Cuentas (id_cliente, tipo_cuenta, saldo) VALUES (?, ?, ?)"  
  
    _, err := db.DB.Exec(query, userID, "Cuenta de Ahorros", 0)  
    // Puedes cambiar el tipo de cuenta según sea necesario  
  
    if err != nil {  
  
        c.JSON(http.StatusInternalServerError, gin.H{"error":  
"Error al abrir la cuenta"})  
  
        return  
  
    }  
}
```



```
c.Redirect(http.StatusFound, "/cuenta") // Redirigir a la
página de cuenta
}
```

3. Agregar la ruta para abrir una cuenta

Asegúrate de agregar una ruta en tu archivo routes.go para manejar la solicitud de abrir una cuenta:

```
r.POST("/abrirCuenta", authMiddleware(), abrirCuenta) //
Asegúrate de que esta línea esté presente
```

4. Modificar la plantilla de cuenta

En la plantilla donde los clientes inician sesión, agrega un botón o un formulario que permita abrir una cuenta. Aquí tienes un ejemplo de cómo podría verse:

```
<form action="/abrirCuenta" method="POST">
    <button type="submit">Abrir Cuenta</button>
</form>
```

5. Sincronización del saldo

Cuando el administrador recargue el saldo de una cuenta, asegúrate de que la lógica de recarga esté correctamente implementada. La función recargarSaldo que ya tienes debería actualizar el saldo de la cuenta correspondiente utilizando el código de cuenta.

Aquí tienes un ejemplo de cómo podría verse la lógica de recarga:

```
func recargarSaldo(c *gin.Context) {
    session := sessions.Default(c)
    userID := session.Get("user_id")
```

```

// Verificar si el usuario es un administrador

if userID == nil || userID != 1 {

    c.Redirect(http.StatusFound, "/login")

    return

}

codigoCuenta := c.PostForm("codigoCuenta")

monto, err := strconv.ParseFloat(c.PostForm("monto"), 64)

if err != nil {

    c.JSON(http.StatusBadRequest, gin.H{"error": "Monto
inválido"})

    return

}

// Actualizar el saldo en la base de datos

_, err = db.DB.Exec("UPDATE Cuentas SET saldo = saldo + ?
WHERE codigo_cuenta = ?", monto, codigoCuenta)

if err != nil {

    c.JSON(http.StatusInternalServerError, gin.H{"error":
"Error al recargar el saldo"})

    return

```

```
}

    c.Redirect(http.StatusFound, "/admin") // Redirigir a la
página de administración
}
```

Resumen

Modificar la tabla Cuentas: Asegúrate de que la tabla tenga los campos necesarios.

Crear la función abrirCuenta: Permite a los clientes abrir una cuenta.

Agregar la ruta para abrir una cuenta: Asegúrate de que la ruta esté configurada.

Modificar la plantilla de cuenta: Agrega un botón para abrir una cuenta.

Sincronización del saldo: Asegúrate de que la lógica de recarga de saldo esté correctamente implementada.

Con estos pasos, deberías poder implementar la funcionalidad para que los clientes abran cuentas y que estas estén sincronizadas con el saldo que el administrador puede recargar.

CREANDO LA PÁGINA DE USUARIO.HTML

Para incluir la plantilla de navegación (nav.html) en tu plantilla de usuario (usuario.html), puedes utilizar el método de plantillas de Go. A continuación, te muestro cómo hacerlo.

Actualización en usuario.html

Debes incluir la plantilla de navegación en tu archivo usuario.html. Aquí está el código actualizado:

```
<!DOCTYPE html>
```

```

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width,
initial-scale=1.0">

    <title>Document</title>

    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/boot
strap.min.css" rel="stylesheet">

</head>

<body>

{{ template "nav" . }} <!-- Incluir la plantilla de navegación
-->

<div class="container mt-4 mb-4 mx-auto" style="max-width:
800px;">

    <h2>Credenciales del Usuario</h2>

    <div class="card mb-4">

        <div class="card-header">Información del Usuario</div>

        <div class="card-body">

            <h5 class="card-title">Nombre: {{ .Nombre }}</h5>

            <h5 class="card-title">Apellido: {{ .Apellido
}}</h5>

            <h5 class="card-title">Email: {{ .Email }}</h5>

```

```

        <h5 class="card-title">Teléfono: {{ .Telefono
    }}</h5>

        <h5 class="card-title">Fecha de Nacimiento: {{
    .FechaNacimiento.Format "2006-01-02" }}</h5>

    </div>

</div>

</div>

<!-- Bootstrap JS y Popper.js -->

<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd
/popper.min.js"></script>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootst
rap.min.js"></script>

</body>

</html>

```

Explicación de la actualización

Se ha añadido la línea `{{ template "nav" . }}` al inicio del archivo `usuario.html`. Esto incluye la plantilla de navegación en la página de usuario, permitiendo que se muestre de la misma manera que en la plantilla base.

Con esta modificación, la navegación se verá igual en ambas plantillas (`base.html` y `usuario.html`).

2) Para crear la URL que redirija a la página `usuario.html` cuando se presione el botón correspondiente en la navegación, necesitas asegurarte de que el botón en `nav.html` tenga el enlace correcto. A continuación, te muestro cómo hacerlo.

en nav.html

Debes modificar el enlace del botón "Usuario" en tu archivo nav.html para que apunte a la ruta que renderiza la plantilla usuario.html. Aquí está el código actualizado:

```
<li class="nav-item" style="margin-right: 15px;">
    <a class="nav-link" href="/usuario" style="background-color:
#007bff; color: white; padding: 5px 10px; border-radius: 5px;">Usuario</a> <!-- Enlace
actualizado -->
</li>
```

Explicación de la actualización

Se ha actualizado el enlace del botón "Usuario" para que apunte a /usuario. Esto significa que cuando el usuario haga clic en este botón, será redirigido a la ruta que renderiza la plantilla usuario.html.

3) Configuración de la ruta en routes.go

Asegúrate de que en tu archivo routes.go tengas una ruta que maneje la solicitud a /usuario y que renderice la plantilla usuario.html. Aquí hay un ejemplo de cómo podrías hacerlo:

```
// ... existing code ...
r.GET("/usuario", authMiddleware(), showUsuarioPage) // Añadir esta línea
// ... existing code ...
```

4) Y luego, define la función showUsuarioPage para renderizar la plantilla:

```
func showUsuarioPage(c *gin.Context) {

    // Aquí deberías obtener los datos del usuario desde la base
    de datos

    // y pasarlos a la plantilla. Por ejemplo:

    usuario := model.Cliente{ // Asegúrate de que este modelo
    exista

        Nombre:          "Juan",

        Apellido:         "Pérez",

        Email:            "juan.perez@example.com",

        Telefono:         "123456789",

        FechaNacimiento: time.Now().AddDate(-30, 0, 0), //
    Ejemplo de fecha
    }

    c.HTML(http.StatusOK, "usuario.html", usuario)
}
```

obtener los campos Nombre, Apellido, Email y Telefono y mostrarlo en USUARIO.HTML

Si solo deseas obtener los campos Nombre, Apellido, Email y Telefono del cliente en la función showUsuarioPage, puedes simplificar tanto la consulta SQL como el modelo que estás utilizando. Aquí te muestro cómo hacerlo.

```
func showUsuarioPage(c *gin.Context) {

    session := sessions.Default(c)

    userID := session.Get("user_id")

    if userID == nil {

        c.Redirect(http.StatusFound, "/login")

        return
    }

    var cliente struct {

        Nombre    string

        Apellido  string

        Email     string

        Telefono  sql.NullString // Mantener como sql.NullString
para manejar NULL

    }

    // Obtener los datos del cliente desde la base de datos

    err := db.DB.QueryRow("SELECT nombre, apellido, email,
telefono FROM Clientes WHERE id = ?",
userID).Scan(&cliente.Nombre, &cliente.Apellido, &cliente.Email,
&cliente.Telefono)
```



```
if err != nil {  
    fmt.Printf("Error al ejecutar la consulta: %v\n", err)  
    // Log del error  
    c.JSON(http.StatusInternalServerError, gin.H{"error":  
"Error al obtener los datos del cliente"})  
    return  
}  
  
// Pasar los datos a la plantilla  
c.HTML(http.StatusOK, "usuario.html", gin.H{  
    "ShowNav": true,  
    "Nombre": cliente.Nombre,  
    "Apellido": cliente.Apellido,  
    "Email": cliente.Email,  
    "Telefono": cliente.Telefono, // Pasar el campo Telefono  
})  
}
```

2. Actualizar la plantilla usuario.html

Asegúrate de que tu plantilla usuario.html esté configurada para mostrar solo los campos que has seleccionado:

```

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width,
initial-scale=1.0">

    <title>Credenciales del Usuario</title>

    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/boot
strap.min.css" rel="stylesheet">

</head>

<body>

{{ template "nav" . }} <!-- Incluir la plantilla de navegación
-->

<div class="container mt-4 mb-4 mx-auto" style="max-width:
800px;">

    <h2>Credenciales del Usuario</h2>

    <div class="card mb-4">

        <div class="card-header">Información del Usuario</div>

        <div class="card-body">

            <h5 class="card-title">Nombre: {{ .Nombre }}</h5>

            <h5 class="card-title">Apellido: {{ .Apellido
}}</h5>

```

```

        <h5 class="card-title">Email: {{ .Email }}</h5>

        <h5 class="card-title">Teléfono: {{ if
        .Telefono.Valid }}{{ .Telefono.String }}{{ else }}No
        disponible{{ end }}</h5>

    </div>

</div>

</div>

<!-- Bootstrap JS y Popper.js -->

<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd
/popper.min.js"></script>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootst
rap.min.js"></script>

</body>

</html>

```

Resumen

Consulta SQL: Modifica la consulta para que solo seleccione nombre, apellido, email y telefono.

Modelo: Usa una estructura anónima para almacenar solo los campos que necesitas.

Plantilla: Asegúrate de que la plantilla usuario.html muestre solo los campos seleccionados.

Con estas modificaciones, deberías poder obtener y mostrar solo los campos deseados del cliente.

Creando vistas de los botones

Para asegurarte de que la barra de navegación (nav.html) muestre diferentes botones según si el usuario está logueado como administrador o como cliente, puedes utilizar condiciones en tu plantilla. A continuación, te muestro cómo puedes modificar tu archivo nav.html para lograr esto.

```
{{ define "nav" }}

<nav class="navbar navbar-expand-lg navbar-light mt-4 mb-4"
style="max-width: 780px; margin: auto;">

    <div class="container-fluid">

        <a class="navbar-brand" href="#" style="font-size: 1rem;
background-color: #007bff; color: white; padding: 5px 10px;
border-radius: 5px;">MiApp</a> <!-- Aplicar color de fondo y
estilo -->

        <button class="navbar-toggler" type="button"
data-bs-toggle="offcanvas" data-bs-target="#offcanvasNavbar"
aria-controls="offcanvasNavbar">

            <span class="navbar-toggler-icon"></span>

        </button>

        <div class="offcanvas offcanvas-start" tabindex="-1"
id="offcanvasNavbar" aria-labelledby="offcanvasNavbarLabel">

            <div class="offcanvas-header">

                <h5 class="offcanvas-title"
id="offcanvasNavbarLabel" style="font-size: 1rem;
background-color: #007bff; color: white; padding: 5px 10px;
border-radius: 5px;">MiApp</h5> <!-- Aplicar color de fondo y
estilo -->
```

```

        <button type="button" class="btn-close"
data-bs-dismiss="offcanvas" aria-label="Close"></button>

    </div>

    <div class="offcanvas-body">

        <ul class="navbar-nav justify-content-end
flex-grow-1 pe-3">

            {{ if .IsAdmin }} <!-- Mostrar botones solo
si es admin -->

                <li class="nav-item" style="margin-right:
15px;">

                    <a class="nav-link" href="/admin"
style="background-color: #007bff; color: white; padding: 5px
10px; border-radius: 5px;">Administración</a>

                </li>

                <li class="nav-item" style="margin-right:
15px;">

                    <a class="nav-link" href="/historial"
style="background-color: #007bff; color: white; padding: 5px
10px; border-radius: 5px;">Historial de Recargas</a>

                </li>

            {{ end }}

            {{ if .IsUser }} <!-- Mostrar botones solo
si es usuario -->

                <li class="nav-item" style="margin-right:
15px;">

```

```

        <a class="nav-link" href="/usuario"
style="background-color: #007bff; color: white; padding: 5px
10px; border-radius: 5px;">Usuario</a> <!-- Enlace actualizado
-->

    </li>

    <li class="nav-item" style="margin-right:
15px;">

        <a class="nav-link" href="/cuenta"
style="background-color: #007bff; color: white; padding: 5px
10px; border-radius: 5px;">Cuenta</a>

    </li>

    <li class="nav-item" style="margin-right:
15px;">

        <a class="nav-link" href="/transaccion"
style="background-color: #007bff; color: white; padding: 5px
10px; border-radius: 5px;">Transacción</a>

    </li>

    {{ end }}

</ul>

<ul class="navbar-nav ms-auto">

    <li class="nav-item">

        <a class="nav-link" href="/logout"
style="background-color: #007bff; color: white; padding: 5px
10px; border-radius: 5px;">Cerrar Sesión</a>

    </li>

```

```
        </ul>

    </div>

</div>

</div>

</nav>

{{ end }}
```

1. Explicación de los cambios

Condiciones para mostrar botones:

Se utiliza `{{ if .IsAdmin }}` para mostrar los botones de administración y historial solo si el usuario es un administrador.

Se utiliza `{{ if .IsUser }}` para mostrar los botones de cuenta y transacción solo si el usuario es un cliente.

2. Modificar las funciones para pasar el contexto adecuado

Asegúrate de que las funciones que muestran las páginas establezcan las variables `IsAdmin` y `IsUser` en el contexto que se pasa a la plantilla.

Por ejemplo, en la función `showAdminPage`:

```
func showAdminPage(c *gin.Context) {

    session := sessions.Default(c)

    userID := session.Get("user_id")

    // Verificar si el usuario es un administrador
```

```

    if userID == nil || userID != 1 { // Asegúrate de que el ID
1 sea el administrador

        c.Redirect(http.StatusFound, "/login")

        return
    }

    // Obtener el mensaje de la sesión

    mensaje := session.Get("mensaje")

    session.Delete("mensaje") // Eliminar el mensaje después de
mostrarlo

    c.HTML(http.StatusOK, "admin.html", gin.H{

        "mensaje": mensaje, // Pasar el mensaje a la plantilla

        "ShowNav": true,    // Mostrar la barra de navegación

        "IsAdmin": true,    // Indicar que es un administrador

    }) // Cargar la plantilla de administración
}

func registerAdmin(c *gin.Context) {

    var json struct {

        Nombre    string `form:"nombre" binding:"required"`

```



```
    Apellido string `form:"apellido" binding:"required"`

    Email     string `form:"email" binding:"required,email"`

    Password string `form:"password"
binding:"required,min=8"`
}

if err := c.ShouldBind(&json); err != nil {

    c.JSON(http.StatusBadRequest, gin.H{"error": "Datos
inválidos: " + err.Error()})

    return

}

// Hashear la contraseña

hashedPassword, err :=
bcrypt.GenerateFromPassword([]byte(json.Password),
bcrypt.DefaultCost)

if err != nil {

    c.JSON(http.StatusInternalServerError, gin.H{"error":
"Error al hashear la contraseña"})

    return

}

// Insertar el nuevo administrador en la base de datos
```

```

    query := "INSERT INTO Administradores (nombre, apellido,
email, password) VALUES (?, ?, ?, ?)"

    _, err = db.DB.Exec(query, json.Nombre, json.Apellido,
json.Email, hashedPassword)

    if err != nil {

        fmt.Printf("Error al registrar el administrador: %v\n",
err) // Imprimir el error en el servidor

        c.JSON(http.StatusInternalServerError, gin.H{"error":
"Error al registrar el administrador"})

        return

    }

    // Establecer la sesión para el nuevo administrador

    session := sessions.Default(c)

    session.Set("user_id", 1) // Asumiendo que el ID del nuevo
administrador es 1

    session.Save()

    c.Redirect(http.StatusSeeOther, "/admin") // Redirigir a la
página de administración
}

```

Y en la función que muestra la página de usuario cuenta y transacciones:

```

func showUsuarioPage(c *gin.Context) {

```

```
session := sessions.Default(c)

userID := session.Get("user_id")

if userID == nil {

    c.Redirect(http.StatusFound, "/login")

    return

}

var cliente struct {

    Nombre    string

    Apellido  string

    Email     string

    Telefono  sql.NullString // Mantener como sql.NullString
para manejar NULL

}

err := db.DB.QueryRow("SELECT nombre, apellido, email,
telefono FROM Clientes WHERE id = ?",
userID).Scan(&cliente.Nombre, &cliente.Apellido, &cliente.Email,
&cliente.Telefono)

if err != nil {
```

```

        fmt.Printf("Error al ejecutar la consulta: %v\n", err)
// Log del error

        c.JSON(http.StatusInternalServerError, gin.H{"error":
"Error al obtener los datos del cliente"})

        return

    }

    c.HTML(http.StatusOK, "usuario.html", gin.H{

        "ShowNav": true,

        "IsUser": true, // Indicar que es un usuario

    })

}

```

cuenta

```

func showCuentaPage(c *gin.Context) {

    // Supongamos que obtienes estos datos de la base de datos

    clienteNombre := "Juan Pérez"

    saldo := 1500.00

    transacciones := []struct {

        Fecha      string

        Descripcion string

        Monto       float64
    }
}

```

```

    }{

        {"2023-10-01", "Depósito", 500.00},

        {"2023-10-05", "Pago de Servicios", -100.00},

        {"2023-10-10", "Transferencia", -200.00},

    }

    c.HTML(http.StatusOK, "base.html", gin.H{

        "Title":          "Cuenta",

        "ShowNav":        true,

        "IsUser":         true, // Indicar que es un usuario

        "ClienteNombre":  clienteNombre,

        "Saldo":           saldo,

        "Transacciones":  transacciones,

    })

}

```

transacciones

```

func showTransaccionPage(c *gin.Context) {

    session := sessions.Default(c)

    userID := session.Get("user_id")

    if userID == nil {

```

```
c.Redirect(http.StatusFound, "/login")

return

}

var saldo float64

// Obtener el saldo del cliente desde la base de datos

err := db.DB.QueryRow("SELECT saldo FROM Cuentas WHERE
IDCliente = ?", userID).Scan(&saldo)

if err != nil {

    fmt.Printf("Error al obtener el saldo: %v\n", err)

    c.JSON(http.StatusInternalServerError, gin.H{"error":
"Error al obtener el saldo"})

    return

}


c.HTML(http.StatusOK, "transaccion.html", gin.H{

    "Saldo": saldo,

    "IsUser": true, // Indicar que es un usuario

})

}
```



Modifica nav.html para incluir condiciones que muestren diferentes botones según si el usuario es un administrador o un cliente.

Asegúrate de que las funciones que muestran las páginas establezcan las variables IsAdmin y IsUser en el contexto que se pasa a la plantilla.

Con estos cambios, la barra de navegación debería mostrar los botones correctos según el tipo de usuario que ha iniciado sesión.

Creando sistema de transacciones bancarias

Para formatear el saldo de manera que sea legible y se ajuste al formato monetario venezolano (Bolívares), puedes utilizar la función `fmt.Sprintf` en Go para dar formato a los números. A continuación, te muestro cómo puedes hacerlo en la función `showTransaccionPage` y en la plantilla `transaccion.html`.

```
func showTransaccionPage(c *gin.Context) {

    session := sessions.Default(c)

    userID := session.Get("user_id")

    if userID == nil {

        c.Redirect(http.StatusFound, "/login")

        return

    }

    var saldo float64

    // Obtener el saldo del cliente desde la base de datos

    err := db.DB.QueryRow("SELECT saldo FROM cuentas WHERE
id_cliente = ?", userID).Scan(&saldo)
```

```

    if err != nil {

        fmt.Printf("Error al obtener el saldo: %v\n", err)

        c.JSON(http.StatusInternalServerError, gin.H{"error":
"Error al obtener el saldo"})

        return

    }

    // Formatear el saldo a un formato legible en bolívares

    saldoFormateado := fmt.Sprintf("%.2f Bs", saldo) // Formato
a dos decimales

    c.HTML(http.StatusOK, "transaccion.html", gin.H{

        "Saldo":  saldoFormateado, // Pasar el saldo formateado
a la plantilla

        "IsUser": true, // Indicar que es un usuario

    })

}

```

2. Modificar la plantilla transaccion.html

Asegúrate de que la plantilla transaccion.html esté configurada para mostrar el saldo correctamente. Aquí tienes un ejemplo de cómo debería verse:


```

<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/boot
strap.min.css" rel="stylesheet">

    <link href="/assets/css/style-persnalizado.css"
rel="stylesheet">

</head>

<body>

{{ template "nav" . }} <!-- Incluir la plantilla de navegación
-->

<div class="container mt-4 mb-4 mx-auto" style="max-width:
800px;">

    <div class="card mb-4">

        <h2 class="black">Transacciones</h2>

        {{ if .mensaje }} <!-- Mostrar el mensaje de alerta -->

            <div class="alert {{ if eq .mensaje "Código de cuenta
destino no encontrado" }}alert-danger{{ else }}alert-success{{
end }}" role="alert" id="alertMessage">

                {{ .mensaje }}

            </div>

        {{ end }}

```

```

<div class="card-header">Saldo Actual</div>

<div class="profile-header">

    <div class="row align-items-center">

        <div class="col-auto profile-image">

        </div>

        <div class="col ml-md-n2 profile-user-info">

            <h4 class="card-text black">{{ .Saldo
}}</h4>

        </div>

    </div>

</div>

</div>

</div>

<div class="card mb-4">

    <div class="card-header">Transferir Dinero</div>

    <div class="card-body">

        <form action="/transferir" method="POST">

            <div class="mb-3">

                <label for="codigoCuenta"
class="form-label">Código de Cuenta Destino</label>

```

```
        <input type="text" class="form-control"
id="codigoCuenta" name="codigoCuenta" required>

    </div>

    <div class="mb-3">

        <label for="monto" class="form-label">Monto
a Transferir</label>

        <input type="number" class="form-control"
id="monto" name="monto" required>

    </div>

    <button type="submit" class="btn
btn-primary">Transferir</button>

    </form>

</div>

</div>

</div>

<!-- Bootstrap JS y Popper.js -->

<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd
/popper.min.js"></script>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootst
rap.min.js"></script>

</body>

</html>
```

la funcionalidad de transferencia de dinero

resumen detallado de los pasos que se siguieron para implementar la funcionalidad de transferencia de dinero en tu aplicación, incluyendo las modificaciones realizadas en routes.go y transaccion.html.

Pasos para Implementar la Funcionalidad de Transferencia

1. Definir la Ruta de Transferencia en routes.go

Se agregó una nueva ruta para manejar las solicitudes de transferencia. Esto permite que la aplicación reconozca la URL /transferir y ejecute la función correspondiente.

```
r.POST("/transferir", authMiddleware(), transferir) // Ruta para manejar la transferencia
```

2. Implementar la Función de Transferencia en routes.go

Se creó la función transferir que contiene la lógica para realizar la transferencia de dinero. Los pasos dentro de esta función son:

Verificar Autenticación: Se asegura de que el usuario esté autenticado.

```
if userID == nil {  
  
    c.Redirect(http.StatusFound, "/login")  
  
    return  
  
}
```

Obtener Datos del Formulario: Se obtienen el código de cuenta destino y el monto a transferir desde el formulario enviado.

```
// Obtener el código de cuenta y el monto a transferir
```

```
codigoCuentaDestino := c.PostForm("codigoCuenta")

monto, err := strconv.ParseFloat(c.PostForm("monto"), 64)
```

Verificar Existencia de la Cuenta Destino: Se consulta la base de datos para verificar que la cuenta destino existe y se obtienen los datos del destinatario.

```
err = db.DB.QueryRow("SELECT id, nombre, apellido FROM Clientes
WHERE codigo_cuenta = ?",
codigoCuentaDestino).Scan(&idClienteDestino, &nombreDestino,
&apellidoDestino)
```

Obtener y Verificar el Saldo del Usuario: Se consulta el saldo del usuario que está realizando la transferencia y se verifica si es suficiente.

```
err = db.DB.QueryRow("SELECT saldo FROM Cuentas WHERE id_cliente
= ?", userID).Scan(&saldo)
```

Actualizar Saldos: Se actualizan los saldos de las cuentas del remitente y del destinatario en la base de datos.

```
_, err = db.DB.Exec("UPDATE Cuentas SET saldo = saldo - ? WHERE
id_cliente = ?", monto, userID)

_, err = db.DB.Exec("UPDATE Cuentas SET saldo = saldo + ? WHERE
id_cliente = ?", monto, idClienteDestino)
```

Manejo de Errores: En caso de errores, se establece un mensaje en la sesión y se redirige al usuario a la página de transacciones.

```
session.Set("mensaje", "Saldo insuficiente para realizar la
transferencia")

session.Save()

c.Redirect(http.StatusFound, "/transaccion")
```

Mensaje de Éxito: Si la transferencia se realiza correctamente, se establece un mensaje de éxito en la sesión.

```
session.Set("mensaje", fmt.Sprintf("Transferencia de %.2f Bs a %s %s (Código: %s) realizada con éxito.", monto, nombreDestino, apellidoDestino, codigoCuentaDestino))
```

3. Mostrar el Mensaje de Alerta en transaccion.html

En la plantilla transaccion.html, se agregó un bloque condicional para mostrar el mensaje de alerta si existe. Esto permite que el usuario vea el resultado de la transferencia (ya sea un error o un éxito).

```
{{ if .mensaje }} <!-- Mostrar el mensaje de alerta -->

<div class="alert alert-success" role="alert"
id="alertMessage">

    {{ .mensaje }}

</div>

{{ end }}
```

4. Ocultar el Mensaje de Alerta Automáticamente

Se implementó un script en transaccion.html que oculta el mensaje de alerta después de 5 segundos. Esto mejora la experiencia del usuario al no dejar mensajes visibles indefinidamente.

```
// Ocultar el mensaje de alerta después de 2 segundos

window.onload = function() {
```

```

        const alertMessage =
document.getElementById('alertMessage');

        if (alertMessage) {

            setTimeout(function() {

                alertMessage.style.display = 'none'; // Ocultar
el mensaje

                }, 2000); // 2000 milisegundos = 2 segundos

            }

        };

```

Resumen Final

Con estos pasos, se logró implementar la funcionalidad de transferencia de dinero en la aplicación, asegurando que los usuarios puedan realizar transferencias de manera efectiva y recibir retroalimentación a través de mensajes de alerta.

creando confirmación de transferencia con un modal

1. Corrección de la Consulta SQL:**

- ****Problema:**** La consulta original buscaba `codigo_cuenta` en la tabla equivocada.
- ****Solución:**** Modificar el endpoint `/obtener-destinatario` para consultar directamente en `Clientes`.
- ****Código Modificado:****

```

err := db.DB.QueryRow(`

SELECT nombre, apellido

FROM Clientes

WHERE codigo_cuenta = ?`, codigo).Scan(&nombre, &apellido)

```

2. Reestructuración del Flujo del Modal:**

Problema:* El modal se mostraba antes de cargar los datos.

****Solución:**** Separar el botón del evento de submit y usar JavaScript para controlar el flujo.

```
` ``html

<!-- Botón modificado -->

<button type="button" class="btn btn-primary"
id="btnTransferir">Transferir</button>

` ``
```

****3. Implementación de Lógica Asíncrona:****

- ****Problema:**** Los datos no se actualizaban en tiempo real.

- ****Solución:**** Usar `fetch` para obtener datos del servidor antes de mostrar el modal.

- ****Código JavaScript:****

```
document.getElementById('btnTransferir').addEventListener('click', function() {

    // Validar campos

    // Obtener datos

    fetch(`/obtener-destinatario?codigo=${codigoCuenta}`)

        .then(response => response.json())

        .then(data => {

            // Actualizar UI

            new

bootstrap.Modal(document.getElementById('confirmModal')).show();
```



```
    });  
  });
```

****4. Formateo de Datos:****

- ****Problema:**** El monto mostraba formato incorrecto.
- ****Solución:**** Usar `toFixed(2)` para mostrar 2 decimales.
- ****Implementación:****

```javascript

```
document.getElementById('transferAmount').textContent =
`${parseFloat(monto).toFixed(2)} Bs`;
```

#### **\*\*5. Manejo de Errores:\*\***

- **\*\*Problema:\*\*** No se detectaban cuentas inválidas.
- **\*\*Solución:\*\*** Agregar bloque `catch` y mostrar estados de error.
- **\*\*Código:\*\***

```javascript

```
.catch(error => {  
    document.getElementById('recipientName').textContent =  
    "Error";  
    document.getElementById('recipientSurname').textContent =  
    "Error";  
});
```

****6. Limpieza de Estado:****

- ****Problema:**** Datos persistentes entre transferencias.
- ****Solución:**** Resetear valores al cerrar el modal.

- ****Implementación:****

```javascript

```
document.getElementById('confirmModal').addEventListener('hidden.bs.modal', function () {

 document.getElementById('transferAmount').textContent =
 "Cargando...";

 // ... (resetear otros campos)

});
```

**\*\*Verificación Final:\*\***

1. El usuario ingresa código de cuenta y monto
2. Al hacer clic en "Transferir":
  - Valida campos vacíos
  - Muestra el monto formateado
  - Consulta el servidor para obtener nombre/apellido
  - Muestra el modal solo cuando tiene todos los datos
3. El modal muestra información precisa y actualizada
4. La confirmación ejecuta el formulario correctamente

## mostrar el nombre y apellido en la página de cuenta.html y el historial

la función `showCuentaPage` para mostrar los datos reales del usuario logueado. Necesitamos obtener el ID del usuario de la sesión y consultar tanto la tabla `Clientes` como `Cuentas`.

Primero, modifica la función `showCuentaPage` en `routes.go`:

```
func showCuentaPage(c *gin.Context) {

 session := sessions.Default(c)

 userID := session.Get("user_id")

 if userID == nil {

 c.Redirect(http.StatusFound, "/login")

 return

 }

 // Estructura para almacenar los datos del cliente
 var cliente struct {

 Nombre string

 Apellido string

 Saldo float64

 }
```

```
// Obtener nombre, apellido y saldo del cliente

err := db.DB.QueryRow(`

 SELECT

 c.nombre,

 c.apellido,

 COALESCE(cu.saldo, 0) as saldo

 FROM Clientes c

 LEFT JOIN Cuentas cu ON c.id = cu.id_cliente

 WHERE c.id = ?`, userID).Scan(&cliente.Nombre,
&cliente.Apellido, &cliente.Saldo)

if err != nil {

 fmt.Printf("Error al obtener datos del cliente: %v\n",
err)

 c.JSON(http.StatusInternalServerError, gin.H{"error":
"Error al obtener datos del cliente"})

 return

}

// Formatear el saldo con el símbolo de Bolívars

saldoFormateado := fmt.Sprintf("%.2f Bs", cliente.Saldo)
```

```

 c.HTML(http.StatusOK, "base.html", gin.H{ // Cambiado a
base.html

 "Title": "Cuenta",

 "ShowNav": true,

 "IsUser": true,

 "Nombre": cliente.Nombre,

 "Apellido": cliente.Apellido,

 "Saldo": saldoFormateado,

 "Content": "cuenta.html", // Indicar qué template de
contenido usar

 })
}

```

## historial de transacciones entre usuarios

Te ayudaré a crear la tabla y las funciones necesarias para el historial de transacciones entre usuarios.

Primero, vamos a crear la tabla en el archivo banco.sql:

```

-- Crear tabla para historial de transacciones entre usuarios

CREATE TABLE historial_transacciones_usuarios (

 id INT NOT NULL AUTO_INCREMENT,

 id_emisor INT NOT NULL,

```

```

 id_receptor INT NOT NULL,

 monto DECIMAL(10,2) NOT NULL,

 fecha_transaccion TIMESTAMP NOT NULL DEFAULT
current_timestamp(),

 descripcion VARCHAR(255) NOT NULL,

 codigo_cuenta_emisor VARCHAR(255) NOT NULL,

 codigo_cuenta_receptor VARCHAR(255) NOT NULL,

 FOREIGN KEY (id_emisor) REFERENCES Clientes(id),

 FOREIGN KEY (id_receptor) REFERENCES Clientes(id),

 PRIMARY KEY (id)

);

```

Ahora, vamos a modificar la función transferir en routes.go para que registre la transacción en el historial: **routes.go**

```

 // Después de realizar la transferencia exitosamente,
 registrar en el historial

 descripcion := fmt.Sprintf("Transferencia a %s %s",
nombreDestino, apellidoDestino)

 // Obtener el código de cuenta del emisor

 var codigoCuentaEmisor string

```

```

 err = db.DB.QueryRow("SELECT codigo_cuenta FROM Clientes
WHERE id = ?", userID).Scan(&codigoCuentaEmisor)

 if err != nil {

 fmt.Printf("Error al obtener código de cuenta emisor:
%v\n", err)

 }

 // Registrar la transacción en el historial

 _, err = db.DB.Exec(`

 INSERT INTO historial_transacciones_usuarios

 (id_emisor, id_receptor, monto, descripcion,
codigo_cuenta_emisor, codigo_cuenta_receptor)

 VALUES (?, ?, ?, ?, ?, ?)`,

 userID, idClienteDestino, monto, descripcion,
codigoCuentaEmisor, codigoCuentaDestino)

 if err != nil {

 fmt.Printf("Error al registrar en historial: %v\n", err)

 }

```

Crear la función para obtener el historial de transacciones: routes.go

```
// Obtener el historial de transacciones

transacciones, err := historialtransaccioneusuarios(userID)

if err != nil {

 fmt.Printf("Error al obtener historial: %v\n", err)

}
```

```
"Apellido": cliente.Apellido,

"Saldo": saldoFormateado,

"Content": "cuenta.html", // Indicar qué template de contenido usar

"Transacciones": transacciones,

})
```

y

```
func historialtransaccioneusuarios(userID interface{}) ([]struct {

 Fecha string

 Descripcion string

 Monto float64

 NombreOtro string

 ApellidoOtro string

})
```



```

 CodigoCuenta string

 TipoOperacion string
 }, error) {

 var transacciones []struct {

 Fecha string

 Descripcion string

 Monto float64

 NombreOtro string

 ApellidoOtro string

 CodigoCuenta string

 TipoOperacion string

 }

 // Consulta para obtener tanto las transacciones enviadas
 como recibidas

 rows, err := db.DB.Query(`

 SELECT

 DATE_FORMAT(h.fecha_transaccion, '%d/%m/%Y %H:%i')
as fecha,

 h.descripcion,

 h.monto,

```

```

CASE

 WHEN h.id_emisor = ? THEN c_receptor.nombre

 ELSE c_emisor.nombre

END as nombre_otro,

CASE

 WHEN h.id_emisor = ? THEN c_receptor.apellido

 ELSE c_emisor.apellido

END as apellido_otro,

CASE

 WHEN h.id_emisor = ? THEN
h.codigo_cuenta_receptor

 ELSE h.codigo_cuenta_emisor

END as codigo_cuenta,

CASE

 WHEN h.id_emisor = ? THEN 'Enviado'

 ELSE 'Recibido'

END as tipo_operacion

FROM historial_transacciones_usuarios h

JOIN Clientes c_emisor ON h.id_emisor = c_emisor.id

JOIN Clientes c_receptor ON h.id_receptor =
c_receptor.id

```

```
WHERE h.id_emisor = ? OR h.id_receptor = ?

ORDER BY h.fecha_transaccion DESC`,

userID, userID, userID, userID, userID, userID)

if err != nil {

 return nil, err

}

defer rows.Close()

for rows.Next() {

 var t struct {

 Fecha string

 Descripcion string

 Monto float64

 NombreOtro string

 ApellidoOtro string

 CodigoCuenta string

 TipoOperacion string

 }

 err := rows.Scan(&t.Fecha, &t.Descripcion, &t.Monto,

&t.NombreOtro,
```

```

 &t.ApellidoOtro, &t.CodigoCuenta, &t.TipoOperacion)

 if err != nil {

 return nil, err

 }

 transacciones = append(transacciones, t)

}

return transacciones, nil

}

```

Modificar la función showCuentaPage para mostrar el historial: **routes.go**

```

 fmt.Printf("Error al obtener historial: %v\n", err)

}

c.HTML(http.StatusOK, "cuenta.html", gin.H{

 "Title": "Cuenta",

 "ShowNav": true,

 "IsUser": true,

 "Nombre": cliente.Nombre,

 "Apellido": cliente.Apellido,

 "Saldo": saldoFormateado,

 "Content": "cuenta.html",

```

```

 "Transacciones": transacciones,

 "TieneCuenta": tieneCuenta, // Pasar el estado de la
cuenta a la plantilla

 })
}

```

Actualizar la tabla en cuenta.html para mostrar la información adicional:

```

 <tr>

 <th>Fecha</th>

 <th>Descripción</th>

 <th>Tipo</th>

 <th>Usuario</th>

 <th>Código Cuenta</th>

 <th>Monto</th>

 </tr>

 </thead>

 <tbody>

 {{ if .Transacciones }}

 {{ range .Transacciones }}

 <tr>

 <td>{{ .Fecha }}</td>

 <td>{{ .Descripcion }}</td>

 <td>{{ .TipoOperacion }}</td>

 <td>{{ .NombreOtro }} {{ .ApellidoOtro }}</td>

 <td>{{ .CodigoCuenta }}</td>

 <td>{{ .Monto }} Bs</td>

 </tr>

```

```

 </tr>

 {{ end }}

 {{ else }}

 <tr>

 <td colspan="6" class="text-center">No hay
transacciones disponibles.</td>

 </tr>

 {{ end }}

</tbody>

</table>

</div>

</div>

</div>

</div>

{{ end }}

```

Estos cambios permitirán:

Almacenar todas las transacciones entre usuarios

Mostrar un historial detallado en la página de cuenta

Diferenciar entre transacciones enviadas y recibidas

Mostrar información detallada de cada transacción incluyendo la fecha, descripción, monto y datos del otro usuario involucrado

La tabla mostrará la fecha, descripción, tipo de operación (enviado/recibido), información del otro usuario, código de cuenta y monto de cada transacción.

botón "Abrir Cuenta" solo aparezca cuando el usuario no tenga una cuenta creada.

```
// Verificar si el usuario tiene una cuenta

var tieneCuenta bool

err := db.DB.QueryRow("SELECT EXISTS(SELECT 1 FROM Cuentas WHERE id_cliente = ?)",
userID).Scan(&tieneCuenta)

if err != nil {

 fmt.Printf("Error al verificar cuenta: %v\n", err)

}
```

```
err = db.DB.QueryRow(`
```

```
"TieneCuenta": tieneCuenta, // Pasar el estado de la cuenta a la plantilla
```

Luego, modificamos el template cuenta.html para mostrar el botón condicionalmente:**cuenta.html**


```
{{ if not .TieneCuenta }}

{{ end }}
```

Los cambios realizados son:

En showCuentaPage:

Agregamos una consulta para verificar si el usuario tiene una cuenta



Usamos EXISTS para una verificación eficiente

Pasamos el estado de la cuenta (TieneCuenta) a la plantilla

En cuenta.html:

Envolvemos el div del botón en una condición {{ if not .TieneCuenta }}

El botón solo se mostrará si el usuario no tiene una cuenta

Ahora:

Si el usuario no tiene cuenta, verá el botón "Abrir Cuenta"

Si el usuario ya tiene una cuenta, el botón no se mostrará

La verificación se hace de manera eficiente usando EXISTS en la consulta SQL

## STYLOS AL LOGIN.HTML

```
<!DOCTYPE html>

<html lang="es">

<head>

 <meta charset="UTF-8">

 <meta name="viewport" content="width=device-width,
initial-scale=1.0">

 <title>Iniciar Sesión - Banco</title>

 <!-- Bootstrap 5 CSS -->
```



```

 <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/boot
strap.min.css" rel="stylesheet">

 <!-- Font Awesome -->

 <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/
css/all.min.css">

</head>

<body class="bg-light">

 <div class="container-fluid min-vh-100 d-flex
align-items-center justify-content-center py-5">

 <div class="row w-100 justify-content-center mx-0">

 <div class="col-12 col-lg-10">

 <div class="card shadow-lg">

 <div class="row g-0">

 <!-- Lado izquierdo -->

 <div class="col-md-6 bg-primary
text-white p-5 d-flex align-items-center
justify-content-center">

 <div class="text-center">

 <i class="fas fa-university
fa-4x mb-3"></i>

 <h2 class="mb-3">Banco
Digital</h2>

```

```

 <p class="lead">Tu banco de
confianza</p>

 </div>

</div>

<!-- Lado derecho -->

<div class="col-md-6 p-5">

 <div class="mx-auto"
style="max-width: 400px;">

 <h1 class="h3 mb-2">Iniciar
Sesión</h1>

 <p class="text-muted
mb-4">Accede a tu cuenta bancaria</p>

 {{if .mensaje}}

 <div class="alert alert-success
alert-dismissible fade show" role="alert">

 {{.mensaje}}

 <button type="button"
class="btn-close" data-bs-dismiss="alert"
aria-label="Close"></button>

 </div>

 {{end}}

```

```
<form action="/login"
method="post">

 <div class="mb-3">

 <label for="email"
class="form-label">Email</label>

 <input type="email"
class="form-control form-control-lg" id="email" name="email"
required>

 </div>

 <div class="mb-3">

 <label for="password"
class="form-label">Contraseña</label>

 <input type="password"
class="form-control form-control-lg" id="password"
name="password" required>

 </div>

 <div class="d-grid">

 <button class="btn
btn-primary btn-lg" type="submit">Iniciar Sesión</button>

 </div>

</form>

<div class="text-center my-3">
```

```

 <a href="#"
class="text-decoration-none">¿Olvidaste tu contraseña?

 </div>

 <div class="position-relative
my-4">

 <hr>

 <span
class="position-absolute top-50 start-50 translate-middle
bg-white px-3 text-muted">o

 </div>

 <div class="text-center mb-4">

 <p class="text-muted
mb-3">Iniciar sesión con</p>

 <div class="d-flex
justify-content-center gap-3">

 <a href="#" class="btn
btn-dark rounded-circle p-3">

 <i class="fab
fa-facebook-f"></i>

 <a href="#" class="btn
btn-danger rounded-circle p-3">

```



```
</body>

</html>
```

Estos cambios aseguran que:

El contenedor tenga espacio suficiente arriba y abajo (48px)

No se pegue a los bordes del navegador

Mantenga una apariencia limpia y profesional

Sea responsive en todos los tamaños de pantalla

## creando stylos de registro.html

```
<!DOCTYPE html>

<html lang="es">

<head>

 <meta charset="UTF-8">

 <meta name="viewport" content="width=device-width,
initial-scale=1.0">

 <title>Registro - Banco</title>

 <!-- Bootstrap 5 CSS -->

 <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/boot
strap.min.css" rel="stylesheet">

 <!-- Font Awesome -->
```

```

 <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/
css/all.min.css">

</head>

<body class="bg-light">

 <div class="container-fluid min-vh-100 d-flex
align-items-center justify-content-center py-5">

 <div class="row w-100 justify-content-center mx-0">

 <div class="col-12 col-lg-10">

 <div class="card shadow-lg">

 <div class="row g-0">

 <!-- Lado izquierdo -->

 <div class="col-md-6 bg-primary
text-white p-5 d-flex align-items-center
justify-content-center">

 <div class="text-center">

 <i class="fas fa-university
fa-4x mb-3"></i>

 <h2 class="mb-3">Banco
Digital</h2>

 <p class="lead">Tu banco de
confianza</p>

 </div>

 </div>

 </div>

 </div>

 </div>

 </div>

```

```

 <!-- Lado derecho -->

 <div class="col-md-6 p-5">

 <div class="mx-auto"
style="max-width: 400px;">

 <h1 class="h3
mb-2">Registro</h1>

 <p class="text-muted mb-4">Crea
tu cuenta bancaria</p>

 <form action="/register"
method="post">

 <div class="mb-3">

 <label for="nombre"
class="form-label">Nombre</label>

 <input type="text"
class="form-control form-control-lg" id="nombre" name="nombre"
required>

 </div>

 <div class="mb-3">

 <label for="apellido"
class="form-label">Apellido</label>

```



```

 <input type="text"
class="form-control form-control-lg" id="apellido"
name="apellido" required>

 </div>

 <div class="mb-3">

 <label for="email"
class="form-label">Email</label>

 <input type="email"
class="form-control form-control-lg" id="email" name="email"
required>

 </div>

 <div class="mb-3">

 <label for="password"
class="form-label">Contraseña</label>

 <input type="password"
class="form-control form-control-lg" id="password"
name="password" required

 minlength="8"
pattern=".*\d.*">

 <div
class="form-text">Mínimo 8 caracteres con al menos un
número</div>

 </div>

 <div class="mb-3">

```

```

 <label
for="fecha_nacimiento" class="form-label">Fecha de
Nacimiento</label>

 <input type="date"
class="form-control form-control-lg" id="fecha_nacimiento"

name="fecha_nacimiento" required pattern="\d{4}-\d{2}-\d{2}">

 </div>

 <div class="d-grid">

 <button class="btn
btn-primary btn-lg" type="submit">Registrarse</button>

 </div>

</form>

<div class="position-relative
my-4">

 <hr>

 <span
class="position-absolute top-50 start-50 translate-middle
bg-white px-3 text-muted">o

</div>

<div class="text-center mb-4">

```

```

 <p class="text-muted
mb-3">Registrarse con</p>

 <div class="d-flex
justify-content-center gap-3">

 <a href="#" class="btn
btn-dark rounded-circle p-3">

 <i class="fab
fa-facebook-f"></i>

 <a href="#" class="btn
btn-danger rounded-circle p-3">

 <i class="fab
fa-google"></i>

 </div>

 </div>

 <div class="text-center">

 <p class="mb-0">¿Ya tienes
una cuenta? Inicia
sesión aquí</p>

 </div>

</div>
</div>

```

```

 </div>

 </div>

</div>

</div>

</div>

<!-- Bootstrap 5 JS Bundle -->

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootst
rap.bundle.min.js"></script>

</body>

</html>

```

## fondo de las páginas html banco en el body

Para evitar que el fondo de la imagen se vea afectado cuando se despliega y oculta el contenido del botón "Detalles", es importante asegurarse de que el contenedor que contiene la información del usuario no altere el flujo del diseño.

Podemos hacer esto utilizando propiedades CSS que mantengan el fondo intacto y eviten que el contenido afecte el diseño general. Aquí tienes el código actualizado:

```

body {

 margin: 0; /* Elimina los márgenes por defecto */

 padding: 0; /* Elimina el padding por defecto */

```

```
 min-height: 100vh; /* Ocupa al menos el 100% del viewport
height */

 background-image: url('../img/OIP.jpg');

 background-size: cover;

 background-position: center;

 background-repeat: no-repeat; /* Evita que se repita la
imagen */

 background-attachment: fixed; /* Opcional: fija la imagen al
desplazarse */

 color: white;
}

body * {

 color: white; /* Aplica color blanco a todos los elementos
dentro del body */

}

h5 {

 color: black; /* Aplica color blanco a todos los elementos
dentro del body */

}

.card {

 background-color: rgba(255, 255, 255, 0.9);

}
```

```
.card-header {
 background-color: rgba(0, 0, 0, 0.1);
 color: black;
}

.card-body h5 {
 color: black;
}

.nav-tabs .nav-link {
 color: white !important;
}

.nav-tabs .nav-link.active {
 background-color: rgba(255, 255, 255, 0.2) !important;
 color: white !important;
 border-color: transparent;
}

.profile-header {
```

```
background-color: rgba(0, 0, 0, 0.5);

padding: 20px;

border-radius: 10px;

margin-bottom: 20px;
}

.user-name {

 color: white !important;
}

.text-muted {

 color: rgba(255, 255, 255, 0.8) !important;
}
```

agregamos los link de los stylos en los archivos html

```
<link href="/assets/css/style-persnalizado.css"
rel="stylesheet">
```

## creando interacción de los botones de detalles

```
<!DOCTYPE html>

<html lang="en">

<head>

 <meta charset="UTF-8">

 <meta name="viewport" content="width=device-width,
initial-scale=1.0">

 <title>Credenciales del Usuario</title>

 <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/boot
strap.min.css" rel="stylesheet">

 <link href="/assets/css/style-persnalizado.css"
rel="stylesheet">

</head>

<body>

 {{ template "nav" . }} <!-- Incluir la plantilla de
navegación -->

 <div class="container mt-4 mb-4 mx-auto" style="max-width:
800px;">
```



```

<div class="row">

 <div class="col-md-12">

 <div class="profile-header">

 <div class="row align-items-center">

 <div class="col-auto profile-image">

 </div>

 <div class="col ml-md-n2
profile-user-info">

 <h4 class="user-name mb-3">{{
.Nombre }} {{ .Apellido }}</h4>

 </div>

 </div>

 </div>

 <div class="profile-menu">

 <ul class="nav nav-tabs nav-tabs-solid">

 <li class="nav-item">

 <a class="nav-link active black"
id="toggle-details">Detalles


```

```


 </div>

 <div class="tab-content profile-tab-cont"
id="user-info" style="display: none;">

 <div class="tab-pane fade show active"
id="per_details_tab">

 <div class="card mb-4">

 <div class="card-header
black">Información del Usuario</div>

 <div class="card-body">

 <h5 class="card-title
black">Nombre: {{ .Nombre }}</h5>

 <h5 class="card-title
black">Apellido: {{ .Apellido }}</h5>

 <h5 class="card-title
black">Email: {{ .Email }}</h5>

 <h5 class="card-title
black">Teléfono: {{ if .Telefono.Valid }}{{ .Telefono.String
}}{{ else }}No disponible{{ end }}</h5>

 </div>

 </div>

 </div>

 </div>

</div>

```

```
 </div>

 </div>

 <!-- Bootstrap JS y Popper.js -->

 <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd
/popper.min.js"></script>

 <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootst
rap.min.js"></script>

 <script>

document.getElementById('toggle-details').addEventListener('click', function() {

 var userInfo = document.getElementById('user-info');

 if (userInfo.style.display === 'none') {

 userInfo.style.display = 'block'; // Muestra la
información

 } else {

 userInfo.style.display = 'none'; // Oculta la
información

 }

});

 </script>

</body>
```

```
</html>
```

Cambios Realizados:

Botón de Detalles: Se agregó un id al enlace de "Detalles" para poder referenciarlo en JavaScript.

Contenido Oculto: El contenido de la información del usuario está inicialmente oculto (style="display: none;").

JavaScript: Se agregó un script que alterna la visibilidad del contenido de información del usuario al hacer clic en el botón de "Detalles".

## agregando icono al nav.html

```




```

## agregando estilos de nav resolución pc y resolución movil

cambiar el color del ícono del botón hamburguesa (el que se usa para abrir el menú en dispositivos móviles) a color blanco. Para esto, necesitamos modificar la clase del navbar y agregar una clase específica para el ícono.

```
<nav class="navbar navbar-expand-lg navbar-dark mt-4 mb-4"
style="max-width: 780px; margin: auto;">
```



He cambiado la clase navbar-light por navbar-dark. Este cambio hará que el ícono del botón hamburguesa cambie automáticamente a color blanco. Bootstrap maneja esto internamente cuando usamos navbar-dark.

El cambio es sutil pero efectivo:

La clase navbar-dark le indica a Bootstrap que use colores claros para los elementos de navegación

Esto afecta automáticamente al ícono del botón hamburguesa (navbar-toggler-icon), cambiándolo a blanco

## voy a crear efectos a los botones de nav

Voy a crear una transición suave para los botones del menú de navegación, manteniendo el color blanco de las letras y agregando un efecto hover atractivo. Primero, verificaré si ya existe un archivo de estilos personalizado y luego agregaré los nuevos estilos.

```
.nav-link.navmobile {

 color: white !important;

 transition: all 0.3s ease;

 padding: 8px 15px;

 position: relative;

}

.nav-link.navmobile:hover {

 background-color: rgba(11, 159, 204, 0.76);

 transform: translateY(-2px);

 box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

}
```

```
}

.nav-link.navmobile::after {

 content: '';

 position: absolute;

 bottom: 0;

 left: 0;

 width: 0;

 height: 2px;

 background-color: white;

 transition: width 0.3s ease;

}

.nav-link.navmobile:hover::after {

 width: 100%;

}
```

He agregado estilos que incluyen:

Color de texto blanco fijo con `color: white !important`

Efectos de transición suaves:

Al pasar el mouse, el botón se eleva ligeramente (`transform: translateY(-2px)`)

Se agrega un sutil sombreado (`box-shadow`)

Aparece una línea blanca debajo del texto que se expande desde la izquierda



Un fondo semi-transparente al hacer hover

Transiciones suaves de 0.3 segundos para todos los efectos

Los botones ahora tendrán:

Color blanco permanente

Animación al pasar el mouse

Efecto de elevación

Línea animada debajo del texto

Fondo semi-transparente al hover

## color a el menu de nav en la resolución movil

```
.offcanvas {
 background-color: hsla(214, 85.60%, 53.70%, 0.61)
!important;
}

.offcanvas .navbar-nav .nav-link {
 color: white !important;
}
```

He realizado los siguientes cambios:

Agregué la clase .offcanvas con el color de fondo específico que solicitaste: hsla(214, 85.60%, 53.70%, 0.61)

Aseguré que los enlaces dentro del menú lateral mantengan el color blanco para mejor visibilidad

Usé !important para asegurar que estos estilos tengan prioridad sobre los estilos por defecto de Bootstrap

Ahora cuando abras el menú hamburguesa, verás el panel lateral con el mismo color azul semi-transparente que especificaste.

## creando efectos del botón detalles del html

```
/* efectos al boton detalle */

#toggle-details {

 transition: all 0.3s ease;

 position: relative;

 overflow: hidden;

 padding: 8px 20px;

 border-radius: 5px;

}

#toggle-details::before {

 content: '';

 position: absolute;

 top: 0;

 left: -100%;
```



```
width: 100%;

height: 100%;

background: linear-gradient(90deg, transparent, rgba(4, 47,
165, 0.2), transparent);

transition: 0.5s;
}

#toggle-details:hover {

 transform: translateY(-3px);

 box-shadow: 0 5px 15px rgba(9, 48, 223, 0.66);

 background-color: hsla(214, 93.90%, 45.10%, 0.78);


 color: white !important;
}

#toggle-details:hover::before {

 left: 100%;
}

#toggle-details:active {

 transform: translateY(0px);
}
```



cuando selecciono ícono del botón hamburguesa (el que se usa para abrir el menú en dispositivos móviles) me sale un menú a la izquierda pero es de color blanco el fondo cambialo a este color el hsla(214, 85.60%, 53.70%, 0.61)

Voy a modificar el color de fondo del menú lateral (offcanvas) para que coincida con el color que especificaste.

He realizado los siguientes cambios:

Agregué la clase .offcanvas con el color de fondo específico que solicitaste: hsla(214, 85.60%, 53.70%, 0.61)

Aseguré que los enlaces dentro del menú lateral mantengan el color blanco para mejor visibilidad

Usé !important para asegurar que estos estilos tengan prioridad sobre los estilos por defecto de Bootstrap

Ahora cuando abras el menú hamburguesa, verás el panel lateral con el mismo color azul semi-transparente que especificaste.

¿Necesitas algún otro ajuste en el menú lateral?

For the code present, we get this error:

```
...
```

```
ERROR: Invalid Token. Expected expression end "}}" but found stringStart instead.
```

```
...
```

How can I resolve this? If you propose a fix, please make it concise.

El error ocurre porque hay un comentario HTML dentro de una expresión de Go template. Voy a remover el comentario HTML que está causando el problema.

He eliminado el comentario HTML `<!-- Incluir la plantilla de navegación -->` que estaba causando el error. Los comentarios HTML no deben estar dentro de las expresiones de Go template `{{ }}`. Esto debería resolver el error de sintaxis.

creale un efecto a este boton



Voy a crear un efecto atractivo para el botón "Detalles" usando una combinación de transiciones y efectos hover.

He agregado varios efectos al botón "Detalles":

Efecto de elevación: Al pasar el mouse, el botón se eleva ligeramente

Efecto de brillo: Un destello suave que se mueve de izquierda a derecha al hover

Cambio de color: Cambia al color azul semi-transparente que estás usando en la aplicación

Efecto al click: El botón regresa a su posición original cuando se hace clic

Transiciones suaves: Todas las animaciones tienen transiciones suaves

Los efectos incluyen:

Elevación con sombra

Animación de brillo

Cambio de color al azul que usas en la app

Texto en blanco al hover

Efecto de presión al hacer clic

## mostrar un mensaje en la página de administración similar al que se muestra en admin.html

Para mostrar un mensaje en la página de administración similar al que se muestra en admin.html, necesitas modificar la función que maneja la recarga de saldo en routes.go. En lugar de devolver un JSON, puedes redirigir al usuario a la página de administración y pasar el mensaje a la plantilla.

Aquí te muestro cómo hacerlo:

Modificar la función recargarSaldo para que redirija a la página de administración y pase el mensaje.

Asegurarte de que el mensaje se muestre en la plantilla admin.html.

```
func recargarSaldo(c *gin.Context) {

 session := sessions.Default(c)

 userID := session.Get("user_id")

 // Verificar si el usuario es un administrador
 if userID == nil || userID != 1 {

 c.Redirect(http.StatusFound, "/login")

 return
 }

 codigoCuenta := c.PostForm("codigoCuenta")
 monto, err := strconv.ParseFloat(c.PostForm("monto"), 64)

 if err != nil {

 session.Set("mensaje", "Monto inválido")

 session.Save()

 c.Redirect(http.StatusFound, "/admin") // Redirigir a la
página de administración

 return
 }
}
```

```

 }

 // Obtener el ID del cliente usando el codigo_cuenta

 var idCliente int

 err = db.DB.QueryRow("SELECT id FROM Clientes WHERE
codigo_cuenta = ?", codigoCuenta).Scan(&idCliente)

 if err != nil {

 fmt.Printf("Error al obtener el ID del cliente: %v\n",
err) // Imprimir el error en el servidor

 session.Set("mensaje", "Código de cuenta no encontrado")

 session.Save()

 c.Redirect(http.StatusFound, "/admin") // Redirigir a la
página de administración

 return

 }

 // Actualizar el saldo en la base de datos

 result, err := db.DB.Exec("UPDATE Cuentas SET saldo = saldo
+ ? WHERE id_cliente = ?", monto, idCliente)

 if err != nil {

 fmt.Printf("Error al recargar el saldo: %v\n", err) //
Imprimir el error en el servidor

 session.Set("mensaje", "Error al recargar el saldo")

```

```

 session.Save()

 c.Redirect(http.StatusFound, "/admin") // Redirigir a la
página de administración

 return
 }

 // Verificar si se actualizó alguna fila
 rowsAffected, err := result.RowsAffected()

 if err != nil {

 fmt.Printf("Error al obtener filas afectadas: %v\n",
err)

 session.Set("mensaje", "Error al verificar la
actualización")

 session.Save()

 c.Redirect(http.StatusFound, "/admin") // Redirigir a la
página de administración

 return
 }

 if rowsAffected == 0 {

 session.Set("mensaje", "Código de cuenta no encontrado")

 session.Save()
 }

```

```
 c.Redirect(http.StatusFound, "/admin") // Redirigir a la
página de administración

 return

 }

 // Insertar en el historial de recargas

 _, err = db.DB.Exec("INSERT INTO historial_recargas
(id_admin, id_cliente, monto, codigo_cuenta) VALUES (?, ?, ?,
?)", userID, idCliente, monto, codigoCuenta)

 if err != nil {

 fmt.Printf("Error al registrar la recarga en el
historial: %v\n", err)

 }

 // Establecer el mensaje de éxito en la sesión

 session.Set("mensaje", "Recarga realizada con éxito.")

 session.Save()

 c.Redirect(http.StatusFound, "/admin") // Redirigir a la
página de administración
}
```

## Resumen

Eliminar el mensaje de la sesión: Después de mostrar el mensaje en admin.html, se elimina de la sesión para que no aparezca al refrescar la página.

Asegúrate de que el mensaje se establezca correctamente: Esto ya está configurado en la función recargarSaldo.

Con estos cambios, el mensaje solo se mostrará una vez y no persistirá al refrescar la página.

Para resolver este problema, debes mover la lógica de eliminación del mensaje de la sesión a la función que maneja la solicitud en routes.go, específicamente en la función showAdminPage. Aquí es donde se debe eliminar el mensaje de la sesión después de que se ha mostrado.

```
func showAdminPage(c *gin.Context) {

 session := sessions.Default(c)

 userID := session.Get("user_id")

 // Verificar si el usuario es un administrador
 if userID == nil || userID != 1 { // Asegúrate
de que el ID 1 sea el administrador

 c.Redirect(http.StatusFound, "/login")

 return

 }
}
```



```

 // Obtener el mensaje de la sesión
 mensaje := session.Get("mensaje")

 session.Delete("mensaje") // Eliminar el
mensaje después de mostrarlo

 c.HTML(http.StatusOK, "admin.html", gin.H{
 "mensaje": mensaje, // Pasar el mensaje a
la plantilla

 "ShowNav": true, // Mostrar la barra de
navegación

 "IsAdmin": true, // Indicar que es un
administrador

 }) // Cargar la plantilla de administración
}

```

En tu plantilla admin.html, asegúrate de que el bloque que muestra el mensaje se mantenga como está:

```

{{ if .mensaje }}

<div class="alert {{ if eq .mensaje "Código de cuenta no
encontrado" }}alert-danger{{ else }}alert-success{{ end }}
alert-dismissible fade show" role="alert" id="alertMessage">

```

```
 {{ .mensaje }}

 <button type="button" class="btn-close"
data-bs-dismiss="alert" aria-label="Close"></button>

</div>

{{ end }}
```

## Resumen

Eliminar el mensaje de la sesión: Esto se hace en la función `showAdminPage` después de obtener el mensaje.

No intentes manipular la sesión directamente en la plantilla: Las plantillas de Go no pueden ejecutar código Go, solo pueden mostrar datos.

Con estos cambios, deberías poder evitar el error y mostrar el mensaje correctamente sin que persista al refrescar la página

```
USUARIOS ||--o{ CRÉDITOS : "Solicita"

USUARIOS {

 string id PK

 string nombre

 string rol

 string sector

}
```

```
CRÉDITOS {

 string id PK

 string usuario_id FK

 string sector_id FK

 float monto_aprobado

 float tasa_interes

 string proposito -- "Agricultura, Industria, etc."
}

SECTORES {

 string id PK

 string nombre

 float riesgo -- "0.1 (bajo) a 0.9 (alto)"

 float tasa_generacion_capital
}
```



## conclusión

Un sistema bancario completo desarrollado en Go utilizando el framework Gin para el manejo de rutas y MySQL como base de datos. Implementa operaciones bancarias esenciales con seguridad mejorada.

### Características Principales

#### Sistema de Autenticación

- Registro de nuevos usuarios con validación de campos
- Inicio de sesión seguro con sesiones persistentes
- Almacenamiento seguro de contraseñas usando bcrypt
- Gestión de sesiones con cookies cifradas

#### Operaciones Bancarias

- Depósitos: Acreditación de fondos a cuentas
- Consultas de Saldo: Visualización de saldo actual
- Historial de Transacciones: Registro detallado de movimientos
- Transferencias: Envío de fondos entre cuentas con validación



## Módulo de Depósitos

- Interfaz administrativa para gestionar depósitos
- Validación de efectivo físico
- Actualización instantánea de saldos

## Sistema de Transferencias

- Transferencias entre cuentas con verificación de saldo
- Generación de códigos únicos por transacción
- Validación en tiempo real de códigos de seguridad
- Historial completo de operaciones

## Tecnologías Utilizadas

### Backend

- Lenguaje Principal: Go (Golang) 1.18+
- Framework Web: Gin (para enrutamiento y middleware)
- Gestión de Sesiones: Gin Sessions con almacenamiento en cookies
- Base de Datos: MySQL
- Driver de DB: [github.com/go-sql-driver/mysql](https://github.com/go-sql-driver/mysql)
- Hashing de Contraseñas: Bcrypt ([golang.org/x/crypto/bcrypt](https://golang.org/x/crypto/bcrypt))
- Manejo de Formularios: Gin Binding

### Frontend

- Plantillas HTML: Gin HTML rendering
- Estilos: CSS (puede extenderse con Bootstrap u otros frameworks)

### Funcionalidades Técnicas

- Generación de códigos únicos para transferencias

- Validación de datos de entrada en formularios
- Manejo de errores y respuestas HTTP adecuadas
- Conexión segura a base de datos MySQL

## Instalación y Configuración

### Requisitos Previos

- Go 1.18+
- MySQL 5.7+
- Git

Clonar el repositorio:

```
git clone https://github.com/jhoan28310576/App-Bancario.git
cd App-Bancario/banco
```

Instalar dependencias:

```
go mod download
```

Configurar conexión a DB (en db/[db.go](#)):

```
const (

 host = "localhost"

 port = 5432
```

```
user = "tu_usuario"

password = "tu_contraseña"

dbname = "app_bancario"

)
```

Iniciar el servidor:

```
go run main.go
```