POLI BANCO

# INSTRUCTIVO TÉCNICO
# CAJERO AUTOMÁTICO

## CONTENIDO

POLI
BANCO

# PRESENTACIÓN

Este manual técnico documenta el desarrollo del sistema de cajero automático "Polibanco", un proyecto que ofrece una simulación realista de las operaciones fundamentales de un cajero automático. A lo largo del documento, se exploran las herramientas empleadas en su creación, se detalla el diseño del sistema y se proporcionan instrucciones claras para su uso y futura modificación.

POLI
BANCO

# RESUMEN

El sistema de cajero automático "Polibanco" es una aplicación desarrollada en C que utiliza Zinjai para su interfaz gráfica y GitHub para la gestión de código, permitiendo a los usuarios realizar operaciones bancarias básicas como retiros, depósitos y transferencias. La elección de C como lenguaje de programación proporciona eficiencia y flexibilidad, mientras que Zinjai facilita la creación de una interfaz amigable.

GitHub, por otro lado, juega un papel crucial al permitir el control de versiones, el trabajo colaborativo y el almacenamiento seguro del código. Esto facilita el desarrollo conjunto, el seguimiento de cambios, la reversión a versiones anteriores y la organización del proyecto en ramas y etiquetas.

POLI
BANCO

## OBJETIVO

Crear una aplicación de cajero automático (ATM) que permita realizar operaciones básicas de gestión de cuentas bancarias, como consultar saldo, retirar dinero, depositar dinero, transferir dinero y ver el historial de transacciones. El sistema debe incluir validaciones y manejo de errores para cada operación.

## FINALIDAD DEL MANUAL

Este manual tiene como finalidad proporcionar una guía completa para comprender el funcionamiento del sistema "Polibanco". Está dirigido a desarrolladores, testers y cualquier persona interesada en aprender sobre el desarrollo de aplicaciones de cajeros automáticos.

# ASPECTOS TÉCNICOS

## 1.1. HERRAMIENTAS UTILIZADAS PARA EL DESARROLLO

### 1.1.1. Zinjai

Zinjai es un entorno de desarrollo integrado (IDE) que se utilizó para escribir, compilar y depurar el código fuente del sistema "Polibanco". Ofrece una interfaz intuitiva y herramientas que facilitan el desarrollo en C.

### 1.1.2. GitHub

GitHub es una plataforma de desarrollo colaborativo basada en la nube que utiliza el sistema de control de versiones Git. Se utilizó para almacenar el código fuente del proyecto, facilitar el trabajo en equipo, el seguimiento de cambios y el control de versiones.

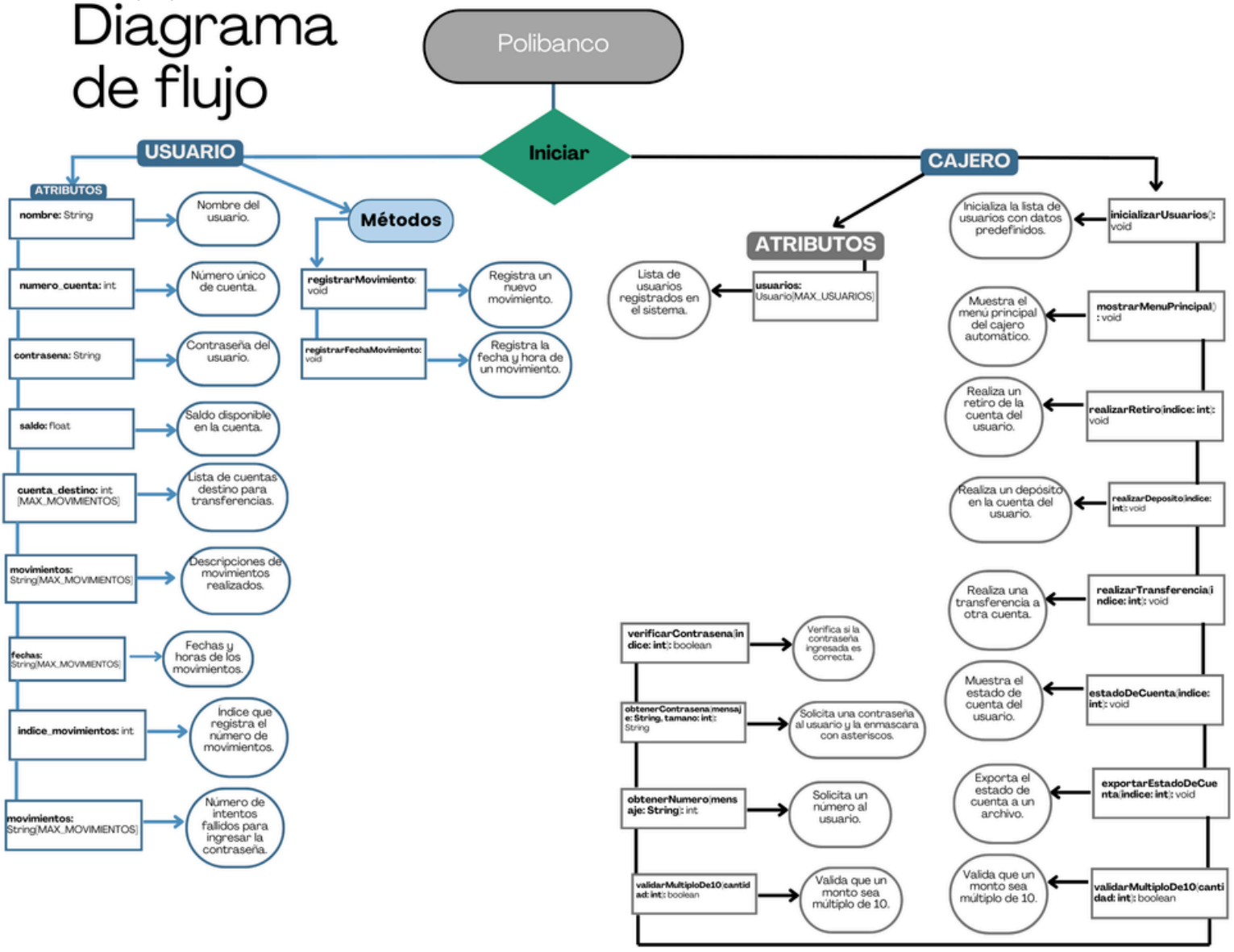
### 1.1.3. Lenguaje C

El sistema "Polibanco" se desarrolló utilizando el lenguaje de programación C. C es un lenguaje de propósito general, eficiente y flexible, ideal para este tipo de aplicaciones que requieren un control preciso de los recursos del sistema.

POLI BANCO

# DIAGRAMAS DE MODELAMIENTO

## Diagrama de flujo




El diagrama de flujo ilustra el funcionamiento de "Polibanco", un sistema de cajero automático. En esencia, el usuario interactúa con el cajero para realizar operaciones bancarias básicas. Introduce su contraseña y selecciona la operación deseada, como un retiro, depósito o transferencia. El cajero, a su vez, valida la contraseña del usuario y, si es correcta, procede a procesar la operación.

# DIAGRAMAS DE CASO DE USO

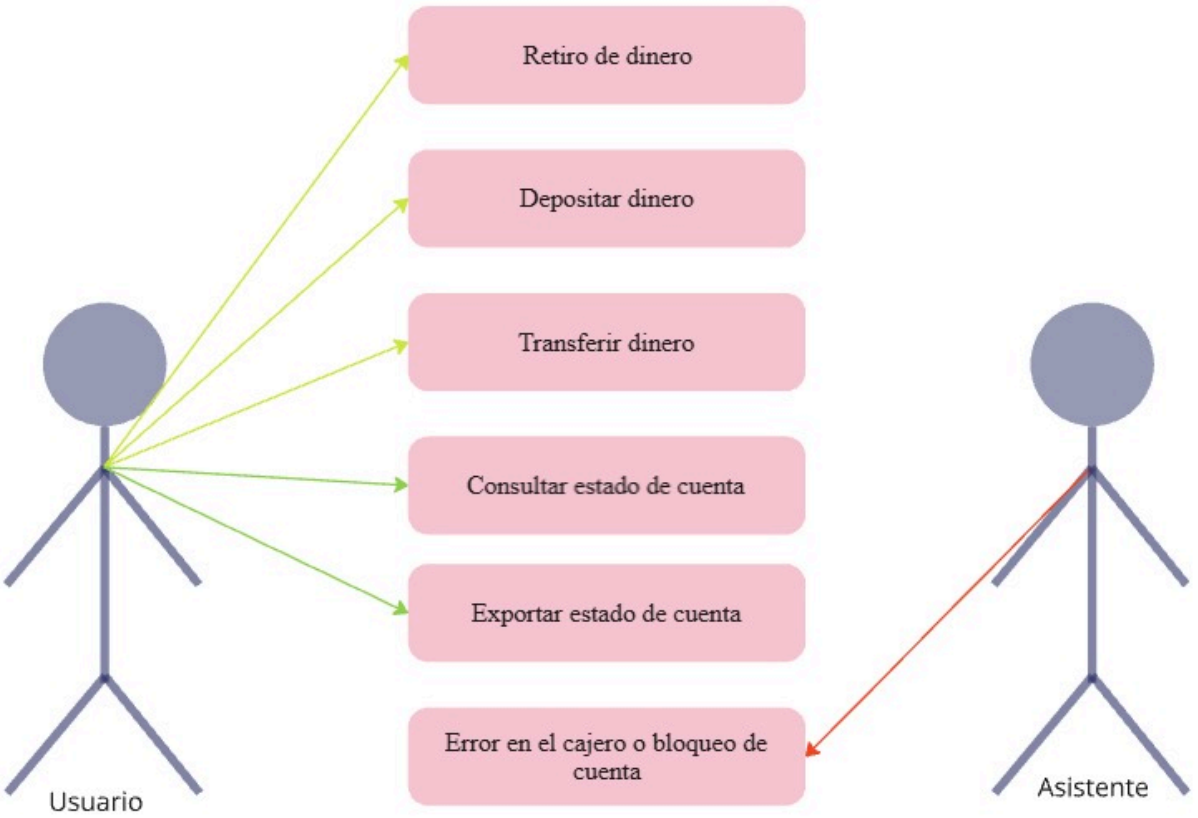



Figura. Casos de Uso

# DICCIONARIO DE DATOS

| Nombre del Campo | Tipo de Dato | Descripción |
|---|---|---|
| nombre | String | Nombre del usuario |
| numero_cuenta | int | Número único de cuenta |
| contrasena | String | Contraseña del usuario |
| saldo | float | Saldo disponible en la cuenta |
| cuenta_destino | int [MAX_MOVIMIENTOS] | Lista de cuentas destino para transferencias |
| movimientos | String [MAX_MOVIMIENTOS] | Descripciones de movimientos realizados |
| fecha | String [MAX_MOVIMIENTOS] | Fechas y horas de los movimientos |
| indice_movimientos | int | Índice que registra el número de movimientos |
| intentos_fallidos | int | Número de intentos fallidos para ingresar la contraseña |

# DESARROLLO DEL SISTEMA

## 3.1. MODIFICACIÓN LOCAL

Para modificar el código fuente del sistema "Polibanco", se debe utilizar Zinjai o cualquier otro editor de texto que soporte C. Se recomienda descargar el código desde el repositorio de GitHub y realizar las modificaciones localmente.

## 3.2. DESCRIPCIÓN CÓDIGO FUENTE

**Función:** ***int obtenerNumero(const char *mensaje)***
**Descripción:** Solicita un número entero al usuario, valida que la entrada sea correcta y devuelve el valor introducido.
**Proceso:**

- Muestra el mensaje especificado como parámetro.
- Espera y valida la entrada del usuario.
- Verifica que sea un número entero válido.
- Rechaza caracteres no válidos con mensajes de error y solicita la entrada nuevamente.
- Devuelve el número ingresado.

**Consideraciones:**

- Asegurarse de que la entrada sea estrictamente un número entero.
- Maneja errores de entrada como caracteres alfabéticos o especiales.

## DESARROLLO DEL SISTEMA

**Función:** ***void obtenerTexto(const char *mensaje, char *texto, int tamano)***
**Descripción:** Permite capturar una cadena de texto del usuario, asegurándose de que no supere el tamaño máximo permitido.
**Proceso:**

- Lee la entrada del usuario y almacena la cadena en el puntero texto.
- Verifica que la longitud no exceda el tamaño especificado.

**Consideraciones:**

- Evitar desbordamientos de búfer mediante la limitación de caracteres.
- Garantizar que la cadena sea segura y libre de caracteres no deseados.

**Función:** ***void obtenerContrasena(const char *mensaje, char *contrasena, int tamano)***
**Descripción:** Solicita una contraseña al usuario y la almacena en el puntero contrasena. Enmascara la entrada para mantener la privacidad.
**Proceso:**

- Lee los caracteres introducidos sin mostrarlos directamente en pantalla, reemplazándolos por asteriscos.
- Permite borrar caracteres con la tecla Backspace.

**Consideraciones:**

- Limitar la longitud de la contraseña al tamaño máximo permitido.
- Proteger contra ataques que intenten interceptar la entrada de la contraseña.

## DESARROLLO DEL SISTEMA

**Función: *void mostrarMenuPrincipal()***
**Descripción:** Presenta al usuario el menú principal con las opciones disponibles.
Proceso:

- Muestra un encabezado del sistema.
- Lista las opciones numeradas.

**Consideraciones:**

- Asegurarse de que las opciones sean claras y fáciles de entender.
- Mantener el formato de salida organizado y limpio.

**Función: *void realizarRetiro(int indice)***
**Descripción:** Permite al usuario retirar una cantidad específica de su saldo, con validaciones de límites y múltiplos.
**Proceso:**

- Presenta opciones de montos predeterminados y una opción para ingresar un monto personalizado.
- Valida el monto ingresado
- Debe ser un múltiplo de 10.
- Debe ser mayor a 0 y menor o igual a $400.

**Consideraciones:**

- Manejar errores como saldos insuficientes y montos no válidos.
- Limitar el monto máximo a $400 por operación.

# DESARROLLO DEL SISTEMA

**Función: *void realizarDeposito(int indice)***
**Descripción:** Permite al usuario depositar una cantidad específica en su cuenta, con validaciones de límites y múltiplos.
**Proceso:**

- Presenta opciones de montos predeterminados y una opción para ingresar un monto personalizado.
- Valida el monto ingresado:
- Debe ser un múltiplo de 10.
- Debe ser mayor a 0 y menor o igual a $1000.

**Consideraciones:**

- Garantizar que los montos sean válidos y seguros.
- Manejar límites superiores para evitar depósitos excesivos.

**Función: *void realizarTransferencia(int indice)***
Descripción: Facilita la transferencia de fondos entre usuarios, con validaciones de cuentas y límites de operación.
Proceso:

- Solicita el número de cuenta destino y valida que exista.
- Verifica que no sea la misma cuenta del remitente.
- Solicita el monto a transferir y valida:
- Que sea mayor a 0 y menor o igual al saldo disponible.
- Que no exceda el límite de $400.

Consideraciones:

- Manejar errores como cuentas inexistentes o transferencias a la misma cuenta.
- Limitar las transferencias a $400 por operación.

## DESARROLLO DEL SISTEMA

**Función:** ***void exportarEstadoDeCuenta(int indice)***
**Descripción:** Genera un archivo de texto con el estado de cuenta del usuario, incluyendo datos financieros y movimientos recientes.
**Proceso:**

- Abre o crea un archivo "EstadoCuenta.txt" en modo escritura.

**Consideraciones:**

- Verificar permisos de escritura en el sistema.
- Manejar errores como fallos al crear o cerrar el archivo.

**Función:** ***int validarMultiploDe10(int cantidad)***
**Descripción:** Verifica si un número es múltiplo de 10, devolviendo 1 para verdadero y 0 para falso.
**Proceso:**

- · Realiza la operación módulo (cantidad % 10).

**Consideraciones:**

- Ideal para validar montos en operaciones financieras.

**Función:** ***int verificarContrasena(int indice)***
**Descripción:** Valida la contraseña del usuario para otorgar acceso al sistema.
**Proceso:**

- Solicita la contraseña enmascarada del usuario.
- Compara la contraseña ingresada con la almacenada.

**Consideraciones:**

- Proteger la entrada de la contraseña mediante enmascaramiento.
- Implementar un bloqueo efectivo después de múltiples intentos fallidos.

## DESARROLLO DEL SISTEMA

**Función:** ***void registrarMovimiento(Usuario *usuario, const char *descripcion)***

**Descripción:** Registra un nuevo movimiento financiero en la cuenta del usuario especificado, almacenando una descripción del movimiento en el historial de la cuenta.

**Proceso:**

- ·Calcula el índice adecuado para almacenar el movimiento utilizando el operador módulo con el máximo de movimientos permitidos.
- ·Guarda la descripción en el historial de movimientos del usuario.

**Consideraciones:**

- ·Implementar una estructura circular para el historial de transacciones para gestionar eficientemente el límite de movimientos y validar que la longitud de la descripción no supera el límite permitido.

**Función:** ***void registrarFechaMovimiento(Usuario *usuario)***

**Descripción:** Registra la fecha y hora de la transacción más reciente en el historial de la cuenta del usuario utilizando la estructura de tiempo del sistema.

**Proceso:**

- ·Recupera la fecha y hora actuales utilizando la función time.
- ·Formatea la fecha en una cadena legible utilizando strftime.
- ·Almacena la fecha formateada en la entrada apropiada del historial.

**Consideraciones:**

- ·Asegura un formato de fecha consistente (por ejemplo, DD/MM/AAAA HH:MM:SS).
- ·Maneje errores potenciales al recuperar la hora del sistema.

# DESARROLLO DEL SISTEMA

**Función:** ***void estadoDeCuenta(indice)***
**Descripción:** Muestra un resumen detallado de la cuenta del usuario identificado por el índice dado, incluyendo la información financiera actual y el historial de transacciones recientes.
**Proceso:**

- Recupera los datos de la cuenta:
- Nombre del titular de la cuenta.
- Número de cuenta.
- Saldo disponible.
- Historial de transacciones con fechas.
- Presenta la información en un formato claro, legible y organizado.

**Consideraciones:**

- Garantizar el orden cronológico de las transacciones.
- Validar que el histórico está libre de registros duplicados o inconsistentes.

# CONTROL DE CÓDIGO GITHUB

### 3.2.1. Creación de un repositorio

Se creó un repositorio en GitHub para almacenar el código fuente del sistema "Polibanco". Esto permite llevar un control de las versiones del código y facilita el trabajo colaborativo.

### 3.2.2. Ramificaciones (Branching)

Se utilizaron ramas (branches) en Git para desarrollar nuevas funcionalidades o corregir errores de forma independiente, sin afectar la rama principal del proyecto. En este proyecto se crearon las siguientes ramas:

- corrección-registrar-movimiento: Para corregir un error en la impresión de fechas en el historial de movimientos.
- mejora-estadocuenta: Para mejorar la información mostrada en el estado de cuenta.
- mejora-transferencia: Para agregar un límite a la cantidad máxima de transferencia.
- control-contrasena: Para implementar un control de intentos fallidos de contraseña.
- mejora-retiro: Para establecer un límite en la cantidad de retiro.
- mejora-deposito: Para limitar la cantidad máxima de depósito.
- exportar-archivo: Para mejorar la función de exportar el estado de cuenta a un archivo.

### 3.2.3. Confirmaciones (Commits)

Cada cambio realizado en el código se registró como una confirmación (commit) en Git con un mensaje descriptivo. Esto permite llevar un historial de las modificaciones y facilita la reversión a versiones anteriores si es necesario.

## CONTROL DE CÓDIGO GITHUB

### 3.2.4. Fusiones (Merges)

Una vez que los cambios en una rama se probaron y aprobaron, se fusionaron (merge) con la rama principal del proyecto (main).

### 3.2.5 Resolución de Conflictos

Durante el proceso de fusión, se puede presentar un conflicto si dos ramas modifican la misma línea de código. En este proyecto se presentó un conflicto al fusionar la rama mejora-transferencia con main. El conflicto se resolvió manualmente, combinando los cambios de ambas ramas y realizando pruebas para asegurar el correcto funcionamiento.

## REQUERIMIENTOS DE SOFTWARE

### 4.1. REQUISITOS MÍNIMOS

- **Sistema operativo:** Windows
- **IDE:** Zinjai (u otro compilador de C)
- **Librerías:** stdio.h, stdlib.h, string.h, conio.h, time.h

POLI BANCO

# CONTÁCTANOS:

**Contacto para Atención al Cliente**
Para resolver cualquier duda, reporte o asistencia, Comuníquese con nosotros a través de los siguientes canales:

**Teléfono celular de Atención al Cliente:**

- 0999715370
- 0978972319
- 0963468464
- 0969064848
- 0984898217
- 0988572554

Horario: Lunes a domingo, 24/7

**Correo Electrónico de Soporte:**

- byron.salazar01@epn.edu.ec
- luis.pacheco03@epn.edu.ec
- joel.champutiz@epn.edu.ec

Respuesta en un máximo de 24 horas hábiles.