



# **Actividad | 3 |**

## **Programa 2 (Parte 2)**

### **Desarrollo de Aplicaciones Móviles IV**

## **Ingeniería en Desarrollo de Software**



academi**ag**lobal

**TUTOR: Marco Antonio Rodríguez**

**ALUMNO: Yanira Lizbeth Lopez Navarro**

**FECHA: 09/08/2024**

# Índice

Introducción .....	3
Descripción .....	4
Justificación .....	5
Desarrollo: .....	6
Conclusión .....	15
Referencias .....	16

## Introducción

En el mundo actual, donde la digitalización de los servicios es cada vez más importante, el Banco Mexicano busca mejorar la experiencia de sus clientes mediante el desarrollo de una aplicación de banca en línea. Esta aplicación, creada en el lenguaje de programación Swift, permitirá a los usuarios realizar operaciones financieras esenciales como depósitos, retiros y consultas de saldo, además de proporcionar una opción segura para cerrar sesión. La elección de Swift se debe a su eficiencia, seguridad y capacidad para crear aplicaciones de alto rendimiento en dispositivos iOS, lo que asegura una experiencia de usuario óptima.

La actividad a desarrollar se enfoca en completar la funcionalidad de las opciones de "Saldo" y "Salir" en la aplicación. Estas opciones son cruciales para proporcionar una experiencia bancaria completa y satisfactoria. La opción "Saldo" permitirá a los usuarios consultar su saldo actual y decidir si desean realizar otras operaciones. La opción "Salir" asegurará que los usuarios puedan cerrar su sesión de manera segura, protegiendo su información financiera.

En esta actividad, se detallará cómo implementar estas funcionalidades de manera efectiva, asegurando que la aplicación no solo cumpla con las expectativas de los usuarios, sino que también fortalezca la relación entre el Banco Mexicano y sus clientes mediante la provisión de un servicio bancario moderno, accesible y seguro.

## Descripción

El Banco Mexicano necesita desarrollar una aplicación de banca en línea que permita a los usuarios realizar transacciones esenciales como depósitos, retiros y consultas de saldo, así como cerrar sesión. La aplicación debe ser desarrollada en Swift, un lenguaje de programación moderno y eficiente, especialmente adecuado para dispositivos iOS. En la actividad previa, se creó la interfaz del menú principal y se implementaron las opciones de “Depósito” y “Retiro”. Ahora, se debe completar el desarrollo agregando funcionalidad a las opciones de “Saldo” y “Salir”.

Para la opción de “Saldo”, el programa debe mostrar el saldo actual del usuario y preguntar si desea realizar otra operación. Esto garantiza que los usuarios tengan acceso rápido y claro a la información de sus cuentas, facilitando una gestión financiera más eficiente. Para la opción de “Salir”, el programa debe informar al usuario que la sesión se ha cerrado, proporcionando una experiencia de cierre segura y clara.

La implementación de estas funcionalidades es crucial para ofrecer una experiencia de usuario completa y satisfactoria. Permitir a los usuarios consultar su saldo en cualquier momento y cerrar sesión de manera segura son aspectos fundamentales de cualquier aplicación bancaria, ya que contribuyen a la transparencia y la seguridad de las transacciones. En resumen, completar estas funcionalidades no solo mejora la aplicación en términos de usabilidad y seguridad, sino que también refuerza la confianza de los usuarios en la plataforma bancaria del Banco Mexicano.

## Justificación

Implementar una aplicación de banca en línea utilizando Swift es una decisión estratégica y beneficiosa tanto para el Banco Mexicano como para sus clientes. Swift es un lenguaje de programación moderno y eficiente, diseñado específicamente para el desarrollo de aplicaciones en el ecosistema de Apple. Su capacidad para crear aplicaciones rápidas, seguras y de alto rendimiento lo convierte en la elección ideal para una aplicación bancaria, donde la seguridad y la eficiencia son primordiales.

La solución propuesta permite a los clientes realizar operaciones esenciales como depósitos, retiros y consultas de saldo de manera sencilla y segura. Al ofrecer estas funcionalidades directamente desde una aplicación móvil, el Banco Mexicano responde a las necesidades de conveniencia y accesibilidad de sus clientes, quienes valoran poder gestionar sus finanzas en cualquier momento y desde cualquier lugar. Esta accesibilidad es especialmente relevante en el contexto actual, donde la digitalización de servicios es cada vez más demandada.

Además, la implementación de una opción para cerrar sesión de manera segura refuerza la confianza de los usuarios en la aplicación, asegurándoles que su información financiera está protegida. En conjunto, estas características no solo mejoran la experiencia del usuario, sino que también fortalecen la relación entre el banco y sus clientes, promoviendo la lealtad y el uso continuo de los servicios bancarios en línea. Esta solución representa una inversión en la innovación y competitividad del Banco Mexicano, alineándose con las expectativas modernas de los consumidores y asegurando una gestión financiera eficiente y segura.

## Desarrollo: Codificación

El código comienza importando la biblioteca Foundation, que proporciona las herramientas básicas necesarias para desarrollar en Swift. Luego, se define la clase BankAccount, que representa una cuenta bancaria. Dentro de esta clase, se declara una propiedad balance de tipo Double, la cual se encargará de almacenar el saldo de la cuenta. El inicializador init(balance: Double) permite crear una nueva instancia de BankAccount asignando un valor inicial al saldo. La línea self.balance = balance asegura que el valor del parámetro balance se asigne a la propiedad balance de la instancia actual de la clase. Esto significa que cada vez que se crea una nueva cuenta bancaria, se le puede especificar un saldo inicial, que se almacenará en la propiedad balance.

```
import Foundation
```

```
class BankAccount {  
    var balance: Double  
  
    init(balance: Double) {  
        self.balance = balance  
    }  
}
```

La función deposit(amount: Double) dentro de la clase BankAccount permite agregar una cantidad específica al saldo de la cuenta bancaria. Cuando se llama a esta función con un valor amount de tipo Double, el saldo actual de la cuenta (balance) se incrementa sumando la cantidad especificada. Después de actualizar el saldo, la función imprime un mensaje que confirma que el depósito se ha realizado con éxito y muestra el nuevo saldo de la cuenta. Esta función es esencial para gestionar los ingresos en la cuenta, permitiendo a los usuarios agregar fondos y recibir una confirmación inmediata de la operación realizada.

```
func deposit(amount: Double) {  
    balance += amount  
    print ("Depósito de \(amount) completado. Balance actual: \(balance)")  
}
```

La función `withdraw(amount: Double) -> Bool` en la clase `BankAccount` gestiona la retirada de dinero de la cuenta. Al invocar esta función con un valor `amount`, primero verifica si la cantidad solicitada es menor o igual al saldo actual (`balance`). Si esta condición se cumple, el saldo se reduce en la cantidad especificada, se imprime un mensaje confirmando que el retiro fue exitoso, junto con el saldo actualizado, y la función devuelve `true` indicando que la transacción fue exitosa. Si la cantidad solicitada excede el saldo disponible, se imprime un mensaje de "Fondos insuficientes" y la función devuelve `false`, señalando que la operación no pudo completarse debido a la falta de fondos. Esta función es fundamental para garantizar que los usuarios no retiren más dinero del que tienen disponible en su cuenta.

```
func withdraw(amount: Double) -> Bool {
  if amount <= balance {
    balance -= amount
    print ("Retiro de \$(amount) completado. Balance actual: \$(balance)")
    return true
  } else {
    print ("Fondos insuficientes. Balance actual: \$(balance)")
    return false
  }
}
```

La función `checkBalance() -> Double` en la clase `BankAccount` devuelve el saldo actual de la cuenta bancaria. Al ser llamada, simplemente retorna el valor almacenado en la propiedad `balance`, lo que permite a los usuarios consultar el saldo disponible en su cuenta. Esta función es útil para verificar el estado financiero de la cuenta en cualquier momento.

Posteriormente, se define la clase `OnlineBanking`, que está diseñada para gestionar las interacciones con una cuenta bancaria. Dentro de esta clase, se declara una propiedad constante `account` de tipo `BankAccount`, que representa la cuenta bancaria con la que se trabajará. El inicializador `init(account: BankAccount)` toma una instancia de `BankAccount` como argumento y la asigna a la propiedad `account`, estableciendo así la conexión entre la instancia de `OnlineBanking` y una cuenta bancaria específica. Esta configuración permite que la clase `OnlineBanking` realice operaciones, como depósitos y retiros, en la cuenta bancaria asociada.

```
func checkBalance() -> Double {
  return balance
}
}
```

```

class OnlineBanking {
    let account: BankAccount

    init(account: BankAccount) {
        self.account = account
    }
}

```

La función `showMenu()` en la clase `OnlineBanking` presenta un menú interactivo al usuario, permitiéndole elegir entre diferentes opciones para gestionar su cuenta bancaria. Inicialmente, se establecen dos variables: `shouldExit`, que controla si el programa debe terminar, y `hasLoggedInBefore`, que rastrea si el usuario ha realizado una operación de retiro anteriormente. La función entra en un bucle `while` que continúa ejecutándose hasta que `shouldExit` se establece en `true`.

Dentro del bucle, se imprime un menú con opciones para realizar depósitos, retiros, consultar el saldo o salir. Luego, se lee la entrada del usuario y se convierte en un número entero. Dependiendo de la opción seleccionada, se llaman a diferentes funciones: `handleDeposit()` para realizar depósitos, `handleWithdrawal(hasLoggedInBefore: hasLoggedInBefore)` para retiros, `account.checkBalance()` para consultar el saldo, y se imprime el saldo disponible seguido de una solicitud para continuar con otra operación mediante `askToContinue()`. Si la opción es 4, el bucle se termina, se imprime un mensaje de cierre de sesión y el programa finaliza. Si la opción no es válida o la entrada no puede ser convertida a un número, se muestra un mensaje de error y el bucle se repite.

```

func showMenu() {
    var shouldExit = false
    var hasLoggedInBefore = false

    while !shouldExit {
        print("""
        Bienvenido a la Banca en Línea
        1. Depósito
        2. Retiro
        3. Saldo
        4. Salir

```



Elige el número de la opción:

""")

```
if let choice = readLine(), let option = Int(choice) {
    switch option {
    case 1:
        handleDeposit()
    case 2:
        handleWithdrawal(hasLoggedInBefore: hasLoggedInBefore)
        hasLoggedInBefore = true
    case 3:
        let balance = account.checkBalance()
        print("Tienes un saldo de: \(balance) pesos")
        askToContinue()
    case 4:
        shouldExit = true
        print("Cerrando sesión de cuenta, vuelva pronto...")
    default:
        print("Opción inválida. Por favor, intente nuevamente.")
    }
} else {
    print("Entrada inválida. Por favor, intente nuevamente.")
}
}
```

La función `handleDeposit()` gestiona el proceso de depósito de dinero en la cuenta bancaria. Primero, se establece un bucle `while` con la variable `continueDepositing` inicializada en `true`. Dentro del bucle, el usuario es solicitado para ingresar una cantidad a depositar. Si la entrada es válida y puede convertirse en un número decimal (`Double`), la función `account.deposit(amount: amount)` se llama para añadir el monto al saldo. Si la entrada no es válida, se muestra un mensaje de error y el bucle continúa. Luego, se pregunta al usuario si desea realizar otro depósito. Dependiendo de la respuesta (en minúsculas), el bucle puede continuar o finalizar. Finalmente, se llama a `askToContinue()` para preguntar al usuario si desea realizar otra operación.

```

private func handleDeposit() {
    var continueDepositing = true

    while continueDepositing {
        print("Ingresa la cantidad a Depositar:")
        if let amountStr = readLine(), let amount = Double(amountStr) {
            account.deposit(amount: amount)
        } else {
            print("Entrada inválida.")
            continue
        }

        print("¿Deseas realizar otro depósito? s/S n/N")
        if let choice = readLine()?.lowercased() {
            if choice == "n" {
                continueDepositing = false
            } else if choice != "s" {
                print("Entrada inválida.")
            }
        }
    }

    askToContinue()
}

```

La función `handleWithdrawal(hasLoggedInBefore: Bool)` maneja los retiros de dinero de la cuenta bancaria. Primero, verifica si el usuario no ha realizado un retiro previamente y si el saldo de la cuenta es cero; en ese caso, muestra un mensaje de "No cuentas con saldo" y llama a `askToContinue()` antes de terminar la función. Si el saldo es suficiente, el bucle `while` continúa solicitando la cantidad a retirar. Si la entrada es válida y se puede convertir en `Double`, se intenta realizar el retiro con `account.withdraw(amount: amount)`. Si el retiro es exitoso, se pregunta al usuario si desea realizar otro retiro; si la respuesta es "n", el bucle se interrumpe. Si el retiro no es exitoso o la entrada es inválida, el bucle se termina y se llama a `askToContinue()`.

```

private func handleWithdrawal(hasLoggedInBefore: Bool) {
    if !hasLoggedInBefore && account.balance == 0 {
        print("No cuentas con saldo")
        askToContinue()
        return
    }

    var continueWithdrawing = true

    while continueWithdrawing {
        print("Ingresa cantidad a retirar:")
        if let amountStr = readLine(), let amount = Double(amountStr) {
            let success = account.withdraw(amount: amount)
            if success {
                print("¿Deseas realizar otro retiro? s/S n/N")
                if let choice = readLine()?.lowercased() {
                    if choice == "n" {
                        continue withdrawing = false
                    } else if choice != "s" {
                        print("Entrada inválida.")
                    }
                }
            } else {
                continueWithdrawing = false
            }
        } else {
            print("Entrada inválida.")
        }
    }

    askToContinue()
}

```

La función `askToContinue()` pregunta al usuario si desea continuar con otra operación. Si el usuario responde "n", se imprime un mensaje de cierre de sesión y el programa se detiene con `exit(0)`. Si la respuesta no es válida, se muestra un mensaje de error y se vuelve a llamar a `askToContinue()`.

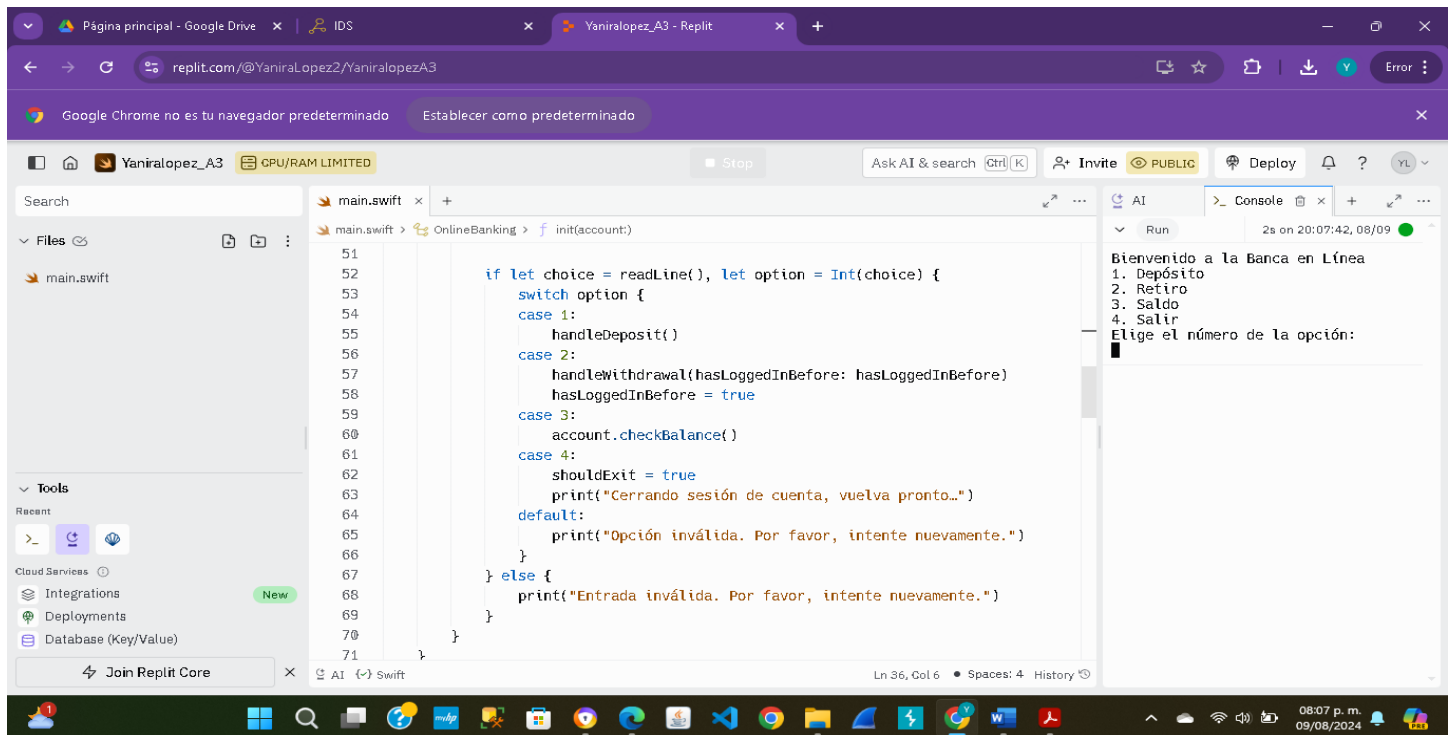
```
private func askToContinue() {
    print("¿Desea continuar con otra operación? s/S n/N")
    if let choice = readLine()?.lowercased() {
        if choice == "n" {
            print("Cerrando sesión de cuenta, vuelva pronto...")
            exit(0)
        } else if choice != "s" {
            print("Entrada inválida.")
            askToContinue()
        }
    }
}
```

Finalmente, fuera de las funciones, se crea una instancia de `BankAccount` con un saldo inicial de 0.0 y una instancia de `OnlineBanking` vinculada a esta cuenta. Luego, se llama a `showMenu()` para iniciar el menú interactivo.

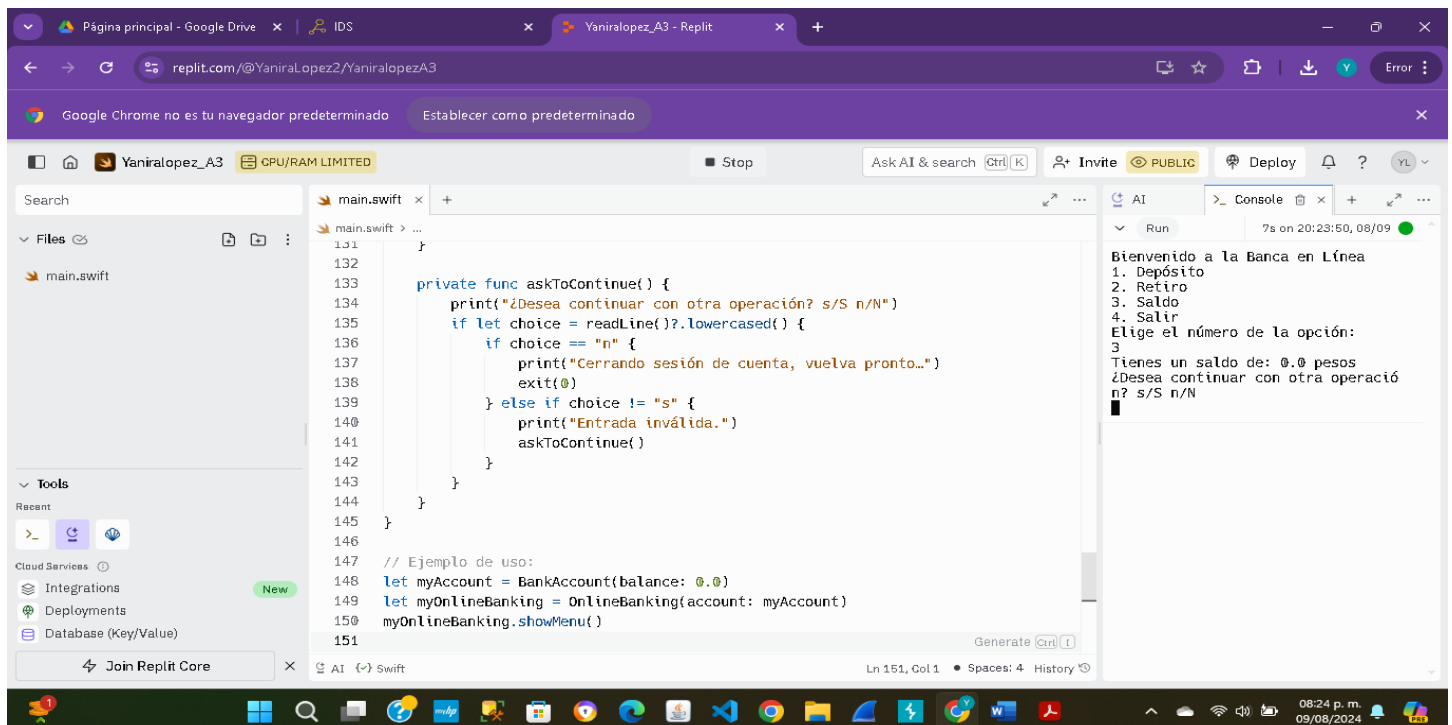
```
let myAccount = BankAccount(balance: 0.0)
let myOnlineBanking = OnlineBanking(account: myAccount)
myOnlineBanking.showMenu()
```

## Prueba del Programa

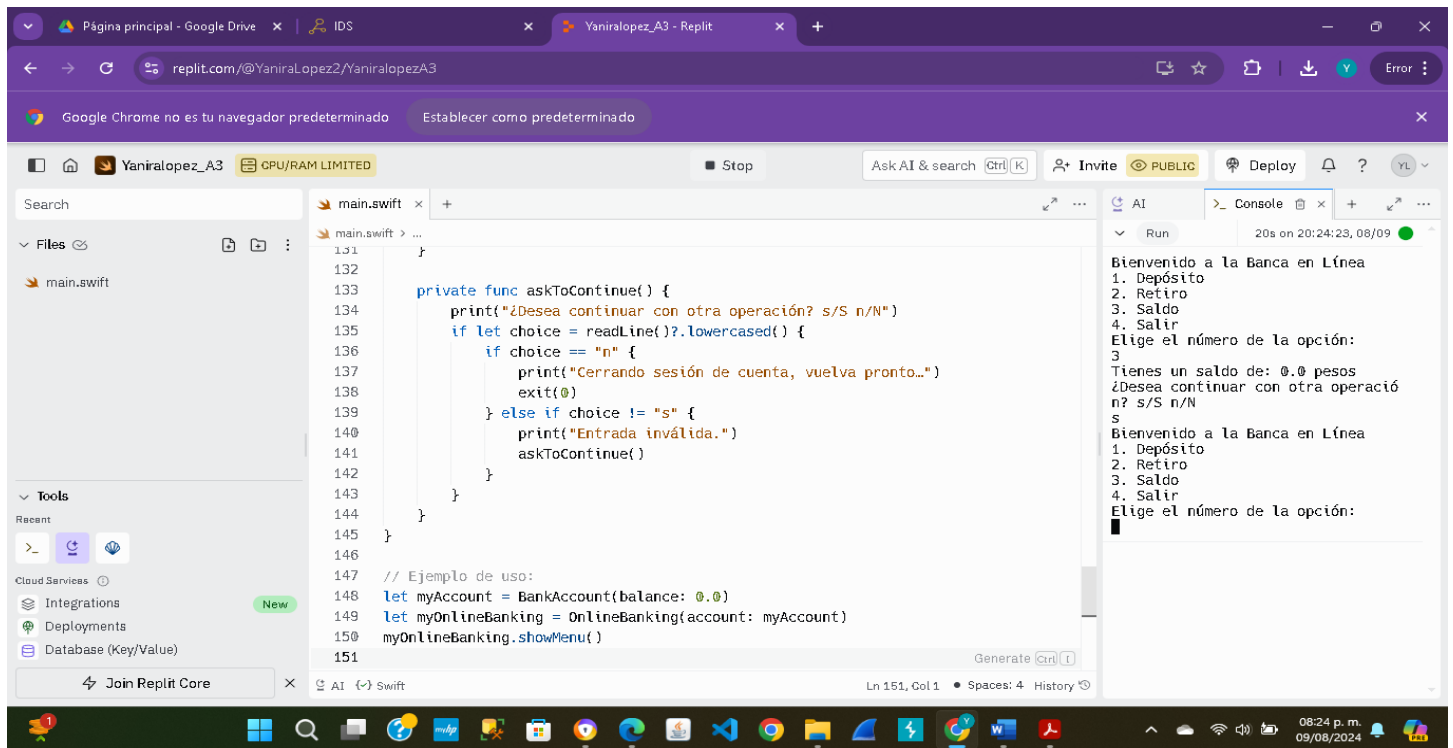
En la siguiente imagen, podemos observar que se abre el entorno de trabajo junto con el programa que se realizó en la Actividad 2. He cambiado el nombre del archivo para facilitar su identificación.



Al seleccionar la “Opción: 3 Saldo”, se mostrará el mensaje "Tienes un saldo de: 0 pesos". Luego, se preguntará si deseas continuar con otra operación, donde deberás elegir entre "s/S" para seguir o "n/N" para salir.



A continuación, podemos observar que en caso de teclear la letra “S”, deberá mostrar de nuevo el menú inicial.

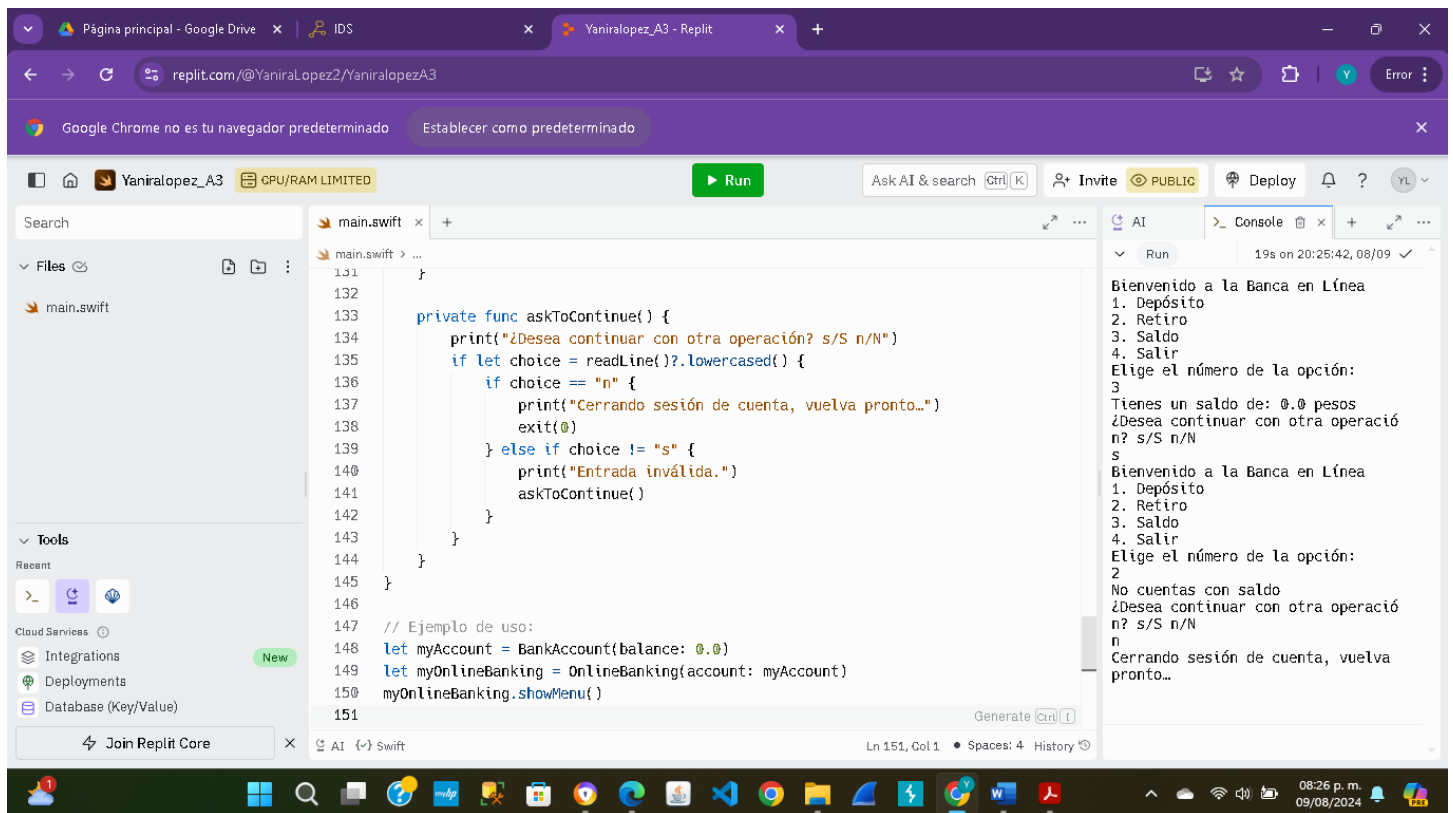


```
131 }
132
133 private func askToContinue() {
134     print("¿Desea continuar con otra operación? s/S n/N")
135     if let choice = readLine()?.lowercased() {
136         if choice == "n" {
137             print("Cerrando sesión de cuenta, vuelva pronto...")
138             exit(0)
139         } else if choice != "s" {
140             print("Entrada inválida.")
141             askToContinue()
142         }
143     }
144 }
145
146 // Ejemplo de uso:
147 let myAccount = BankAccount(balance: 0.0)
148 let myOnlineBanking = OnlineBanking(account: myAccount)
149 myOnlineBanking.showMenu()
150
151
```

Console Output:

```
Bienvenido a la Banca en Línea
1. Depósito
2. Retiro
3. Saldo
4. Salir
Elige el número de la opción:
3
Tienes un saldo de: 0.0 pesos
¿Desea continuar con otra operación? s/S n/N
s
Bienvenido a la Banca en Línea
1. Depósito
2. Retiro
3. Saldo
4. Salir
Elige el número de la opción:
```

La imagen nos muestra que en caso de teclear la letra “N”, deberá mostrar la siguiente pantalla: en la cual se cierra la sesión y nos invita a volver pronto



```
131 }
132
133 private func askToContinue() {
134     print("¿Desea continuar con otra operación? s/S n/N")
135     if let choice = readLine()?.lowercased() {
136         if choice == "n" {
137             print("Cerrando sesión de cuenta, vuelva pronto...")
138             exit(0)
139         } else if choice != "s" {
140             print("Entrada inválida.")
141             askToContinue()
142         }
143     }
144 }
145
146 // Ejemplo de uso:
147 let myAccount = BankAccount(balance: 0.0)
148 let myOnlineBanking = OnlineBanking(account: myAccount)
149 myOnlineBanking.showMenu()
150
151
```

Console Output:

```
Bienvenido a la Banca en Línea
1. Depósito
2. Retiro
3. Saldo
4. Salir
Elige el número de la opción:
3
Tienes un saldo de: 0.0 pesos
¿Desea continuar con otra operación? s/S n/N
n
Cerrando sesión de cuenta, vuelva pronto...
```

## Conclusión

La implementación de una aplicación de banca en línea con las funcionalidades de “Saldo” y “Salir” es de gran importancia tanto en el ámbito profesional del desarrollo de software como en la vida cotidiana de los usuarios. Para los desarrolladores, completar estas opciones en la aplicación representa un avance significativo en la creación de una herramienta bancaria integral, que no solo cumple con las funciones básicas esperadas por los clientes, sino que también ofrece una experiencia de usuario fluida y segura. Utilizar Swift para desarrollar esta aplicación asegura que el producto final sea eficiente, seguro y compatible con los dispositivos iOS, lo cual es esencial para mantener altos estándares de calidad y rendimiento.

Para los usuarios del Banco Mexicano, estas funcionalidades mejoran considerablemente la gestión de sus finanzas diarias. La capacidad de consultar el saldo en tiempo real ofrece una visión clara de sus recursos financieros, facilitando la toma de decisiones informadas. La opción de cerrar sesión de manera segura asegura la protección de su información personal, reforzando la confianza en la plataforma.

En resumen, el desarrollo de esta aplicación no solo optimiza la eficiencia y la seguridad en las transacciones bancarias, sino que también responde a las necesidades modernas de conveniencia y accesibilidad. Esto no solo fortalece la competitividad del banco, sino que también contribuye a una mejor calidad de vida para los clientes al permitirles manejar sus finanzas de manera más eficaz y segura.

## Referencias

Ingeniería en desarrollo de software. Universidad México Internacional. Recuperado el día 30 de julio de 2024  
<https://umi.edu.mx/coppel/IDS/mod/scorm/player.php>

*Video conferencing, web conferencing, webinars, screen sharing.* (s. f.). Zoom. <https://academiaglobal-mx.zoom.us/rec/share/nTWXaqNCZBAFPLiWG6IWqlgPgKFWr3FSO4ylTCDqhIUKNhrw3RIxyIQm9Njig.GIMns9WCVCdtdW1Q>

*Video conferencing, web conferencing, webinars, screen sharing.* (s. f.-b). Zoom. [https://academiaglobal-mx.zoom.us/rec/play/pSGwnoUITgl\\_mawFlrgrJB\\_yYAgpygo\\_3p5O21Rh1eapK2-ipQGT4kgOdCrZnNh3-7uGzPQaNgDwgcDE.9oer0iuZGwvAuWhB?canPlayFromShare=true&from=share\\_recording\\_detail&continueMode=true&componentName=rec-play&originRequestUrl=https%3A%2F%2Facademiaglobal-mx.zoom.us%2Frec%2Fshare%2FU1sVMAB-my2VQZPAEqgo98RrNPwHMZR5kTs11J5CQrb\\_aS-M3AMh4DIdJeu9YCrQ.RsaqulHdKjy79QD](https://academiaglobal-mx.zoom.us/rec/play/pSGwnoUITgl_mawFlrgrJB_yYAgpygo_3p5O21Rh1eapK2-ipQGT4kgOdCrZnNh3-7uGzPQaNgDwgcDE.9oer0iuZGwvAuWhB?canPlayFromShare=true&from=share_recording_detail&continueMode=true&componentName=rec-play&originRequestUrl=https%3A%2F%2Facademiaglobal-mx.zoom.us%2Frec%2Fshare%2FU1sVMAB-my2VQZPAEqgo98RrNPwHMZR5kTs11J5CQrb_aS-M3AMh4DIdJeu9YCrQ.RsaqulHdKjy79QD)

*Video conferencing, web conferencing, webinars, screen sharing.* (s. f.-c). Zoom. [https://academiaglobal-mx.zoom.us/rec/play/LCkDK2-FjxZgox4O5W3W338G0d-15ayxwcIvkkz9wf3mdHpdsO4lIHM8kmaZ9k-dd7ixQA7XxEXIEES1.zr6NC6eApkbPON77?canPlayFromShare=true&from=share\\_recording\\_detail&continueMode=true&componentName=rec-play&originRequestUrl=https%3A%2F%2Facademiaglobal-mx.zoom.us%2Frec%2Fshare%2FK5z-irUQv05suwwfSrGmyShbS9vJLV-xL9YmHpg4iGDMMAvUXTMNj8IziS9W-aH\\_OR\\_3fp8bRbjXNIjS](https://academiaglobal-mx.zoom.us/rec/play/LCkDK2-FjxZgox4O5W3W338G0d-15ayxwcIvkkz9wf3mdHpdsO4lIHM8kmaZ9k-dd7ixQA7XxEXIEES1.zr6NC6eApkbPON77?canPlayFromShare=true&from=share_recording_detail&continueMode=true&componentName=rec-play&originRequestUrl=https%3A%2F%2Facademiaglobal-mx.zoom.us%2Frec%2Fshare%2FK5z-irUQv05suwwfSrGmyShbS9vJLV-xL9YmHpg4iGDMMAvUXTMNj8IziS9W-aH_OR_3fp8bRbjXNIjS)