



FACULTAD DE

INGENIERÍA

CARRERA DE INGENIERÍA DE SISTEMAS
COMPUTACIONALES

GUÍA DOCUMENTADA
INFORME PROYECTO

Autores:

Ramos Alva, Rodrigo Raul Alonso (45% *participación*)
Santillana Olivares, Sebastian Rodolfo (10% *participación*)
Portocarrero Huaman, Kiara Lidia (45% *participación*)

Curso:

Técnicas de Programación Orientada a Objetos

Docente del Curso:
Jose Carlos Anicama Silva

Chorrillos – Perú
2025-1

INSTRUCCIONES

- Desarrollar guía de primeros pasos con Git.
 - Creación de repositorio local, clonar un repositorio de GitHub a la máquina local.
 - Navegación básica por el repositorio.
 - Primer commit. Realizar cambios en el código,
 - Utilizar comandos git add y git commit.
 - Visualizar historial de commits.

DESARROLLO N°1

1. Documenta la guía paso a paso con Git (10 puntos).

GUÍA DE PRIMEROS PASOS CON GIT

1. Documenta la guía paso a paso con Git (10 puntos)

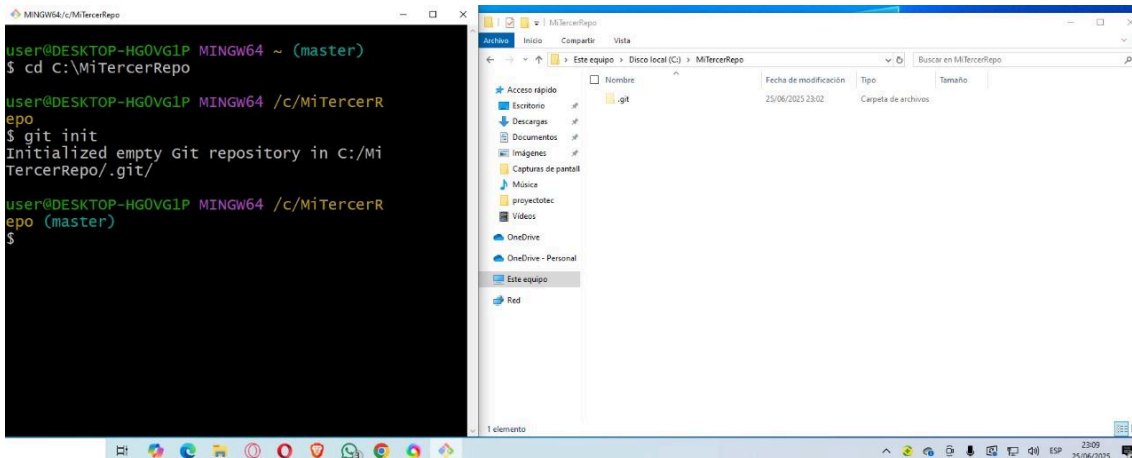
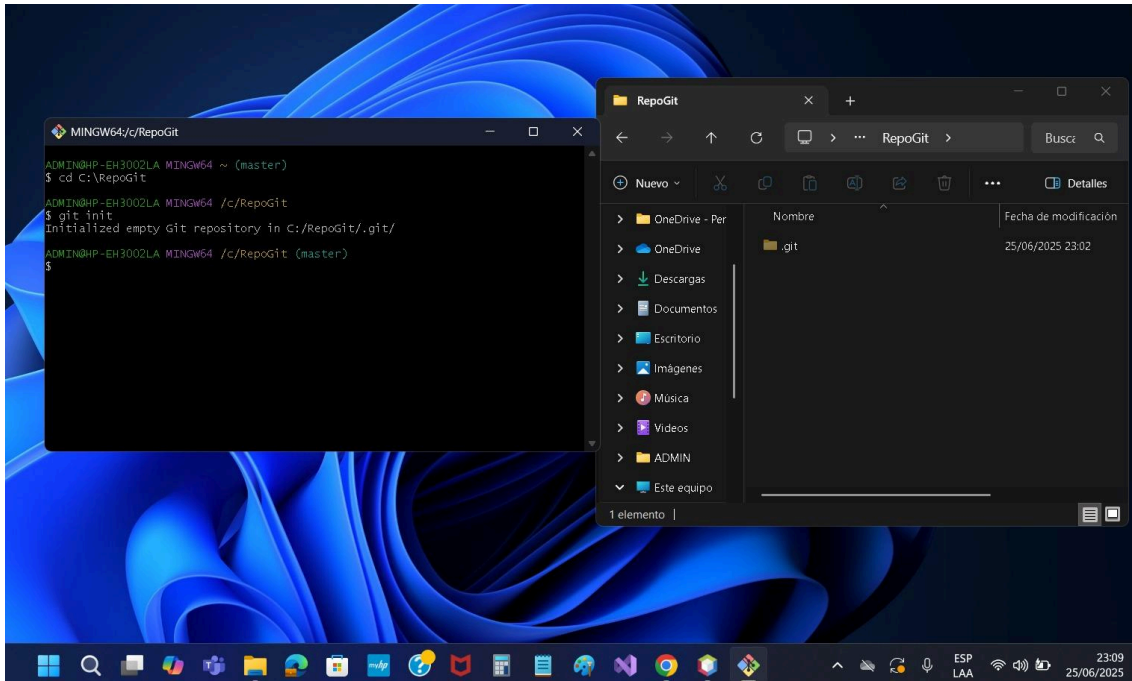
GIT: Software de Control de Versiones de manera local

1: Crear un repositorio local

Para crear un repositorio local en tu máquina:

git init

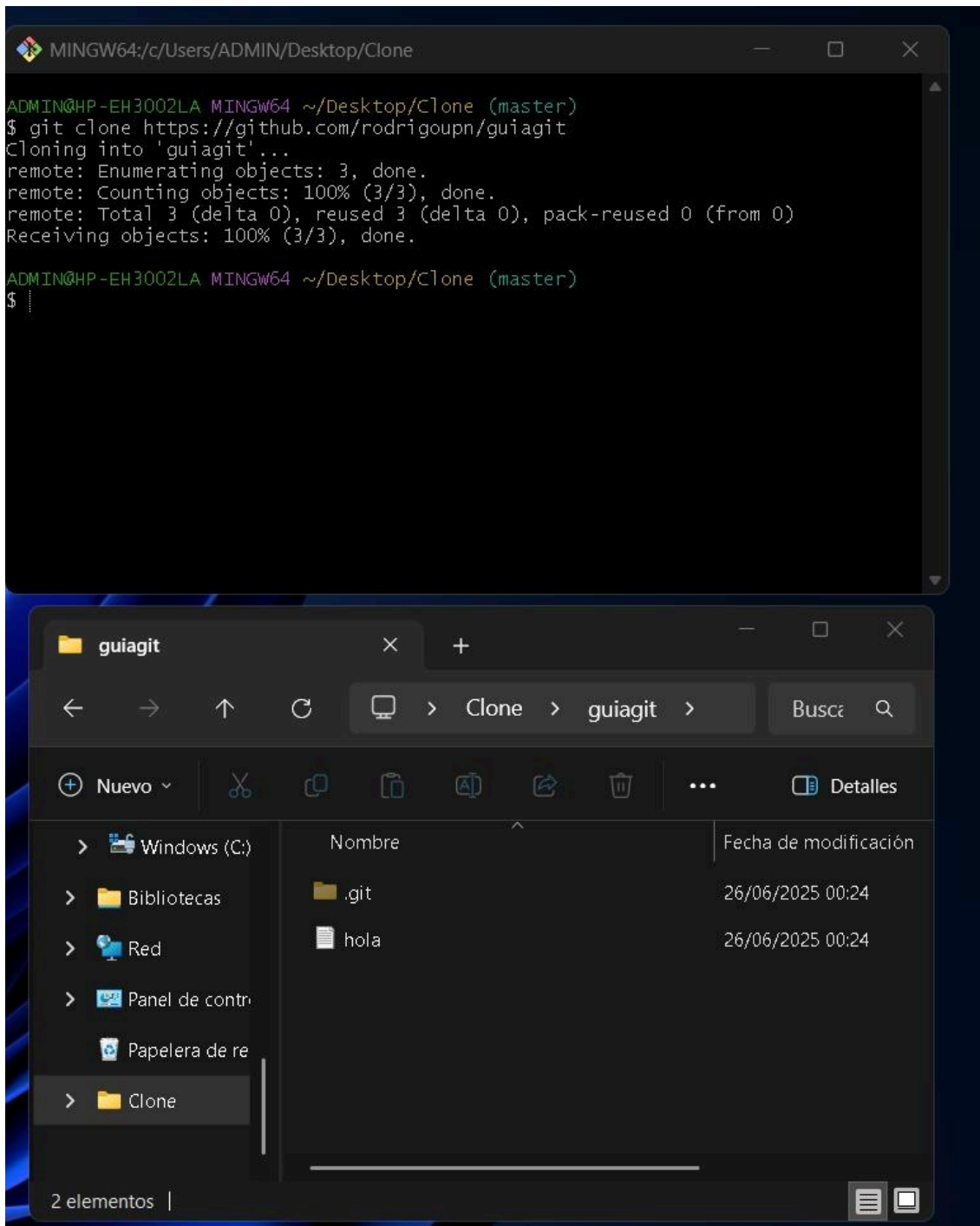
Esto crea un nuevo repositorio Git en el directorio que nosotros elijamos



2: Clonar un repositorio desde GitHub

Para clonar un repositorio existente desde GitHub:

git clone https://github.com/rodrigoupn/guiagit



Esto descargará todos los archivos del repositorio y su historial en una carpeta nueva.

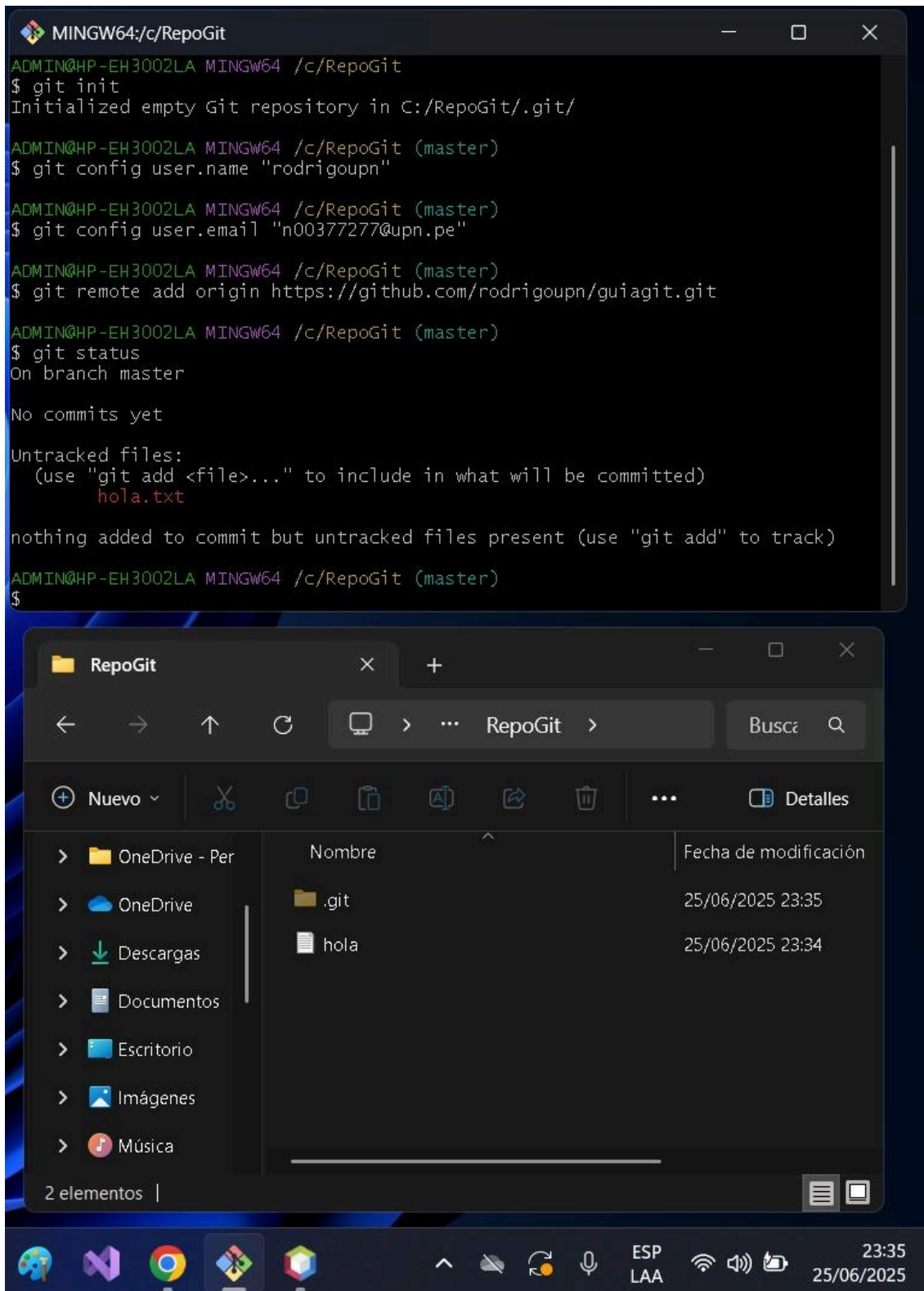
3: Navegación básica por el repositorio

Una vez clonado o creado, puedes navegar y obtener información básica con los siguientes comandos:

cd repositorio

git status # Ver estado de archivos

git log # Ver historial de commits



The screenshot displays a Windows desktop environment. At the top, the UPN (Universidad Privada del Norte) logo is visible. Below it, the commands for navigating to the repository and checking its status and history are listed. The main part of the image shows two overlapping windows. The top window is a terminal titled 'MINGW64:/c/RepoGit' showing the execution of the following commands: `git init`, `git config user.name "rodrigoupn"`, `git config user.email "n00377277@upn.pe"`, `git remote add origin https://github.com/rodrigoupn/guiagit.git`, `git status`, and `git log`. The output of these commands indicates that a new repository has been initialized, configuration has been set, and a remote origin has been added. The bottom window is a File Explorer titled 'RepoGit', showing the contents of the C:/RepoGit directory. It lists two items: a folder named '.git' and a file named 'hola.txt', both with modification dates of 25/06/2025. The taskbar at the bottom shows various application icons and the system clock indicating 23:35 on 25/06/2025.

```
ADMIN@HP-EH3002LA MINGW64 /c/RepoGit
$ git init
Initialized empty Git repository in C:/RepoGit/.git/

ADMIN@HP-EH3002LA MINGW64 /c/RepoGit (master)
$ git config user.name "rodrigoupn"

ADMIN@HP-EH3002LA MINGW64 /c/RepoGit (master)
$ git config user.email "n00377277@upn.pe"

ADMIN@HP-EH3002LA MINGW64 /c/RepoGit (master)
$ git remote add origin https://github.com/rodrigoupn/guiagit.git

ADMIN@HP-EH3002LA MINGW64 /c/RepoGit (master)
$ git status
On branch master

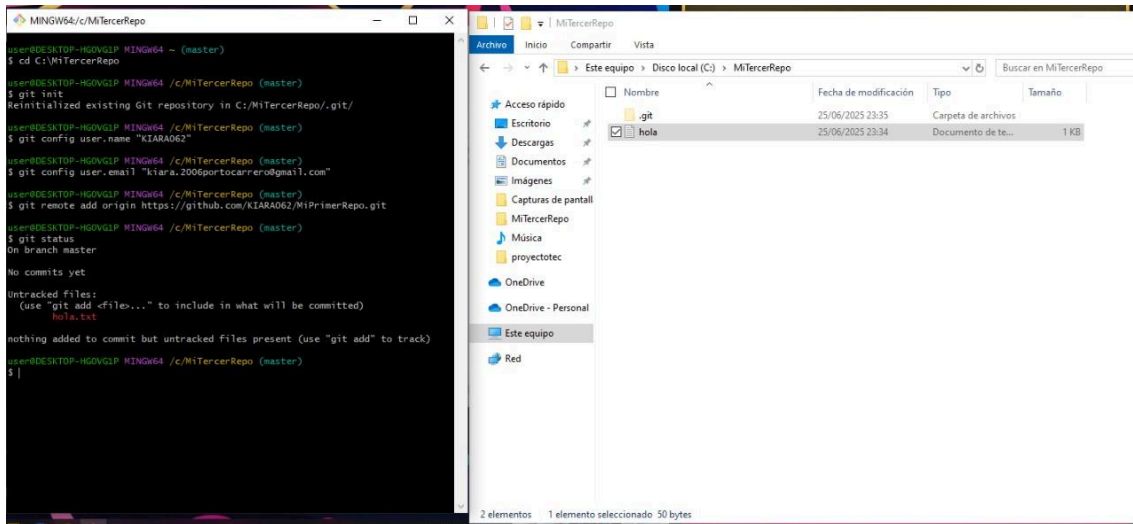
No commits yet

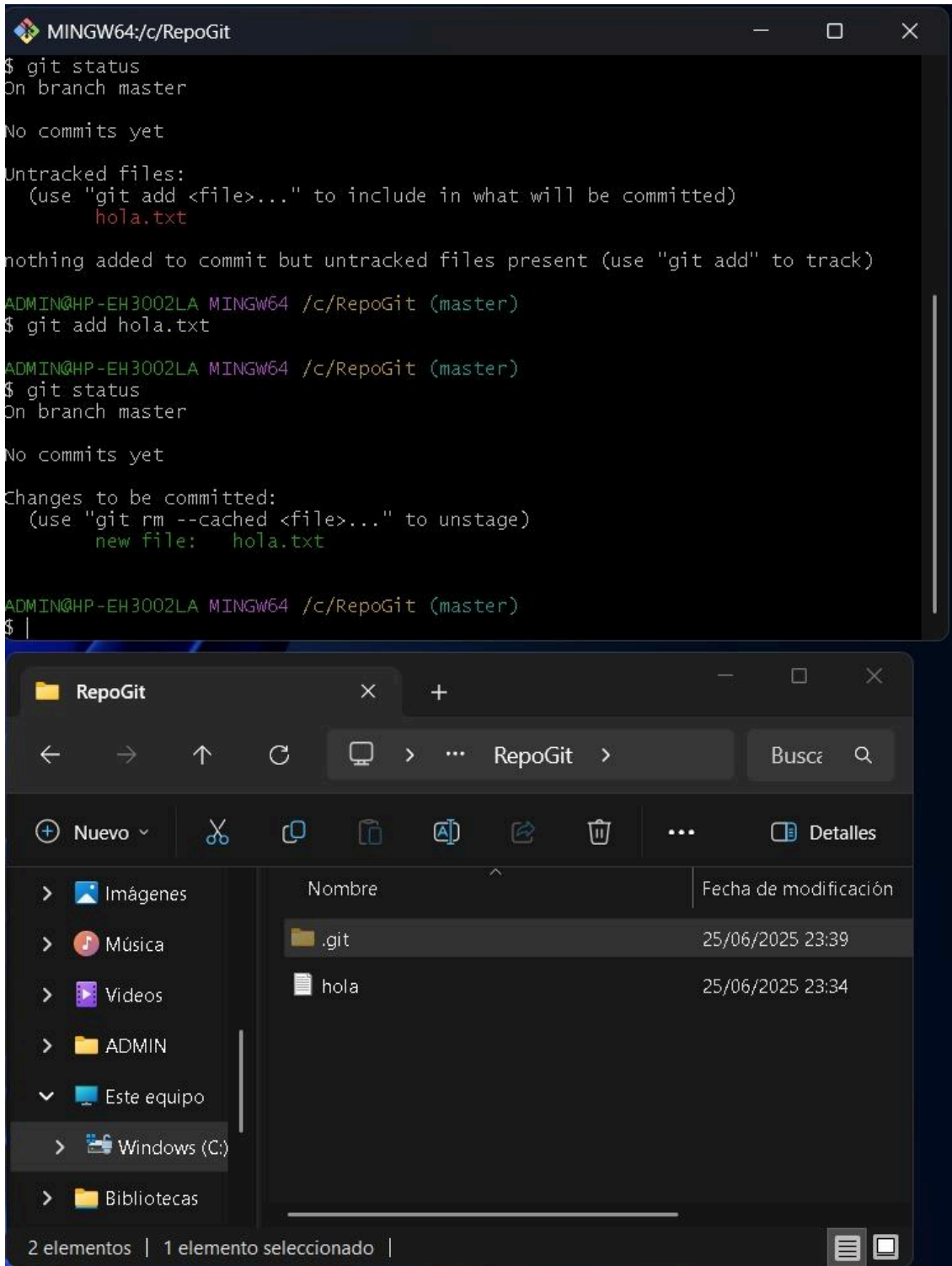
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        hola.txt

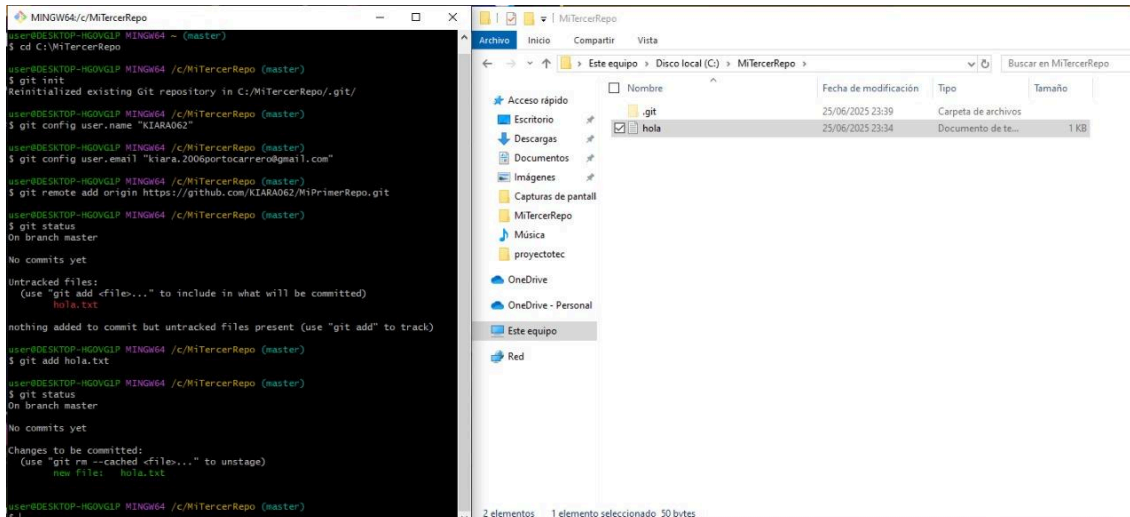
nothing added to commit but untracked files present (use "git add" to track)

ADMIN@HP-EH3002LA MINGW64 /c/RepoGit (master)
$
```

Nombre	Fecha de modificación
.git	25/06/2025 23:35
hola	25/06/2025 23:34







4: Primer Commit (primer envío de cambios)

1. Agrega los cambios al área de preparación:

```
git add README.md
```

2. Haz el commit:

```
git commit -m "Primer commit: Añadido README"
```

5: Usar git add y git commit

1. Agregar cambios:

```
git add main.py
```

2. Confirmar cambios:

```
git commit -m "Añadido script principal main.py"
```

6: Visualizar historial de commits

```
git log
```

Salida esperada:


```
commit sa8sd8ad8...
```

```
Autor: Tu Nombre <tucorreo@correo.com>
```

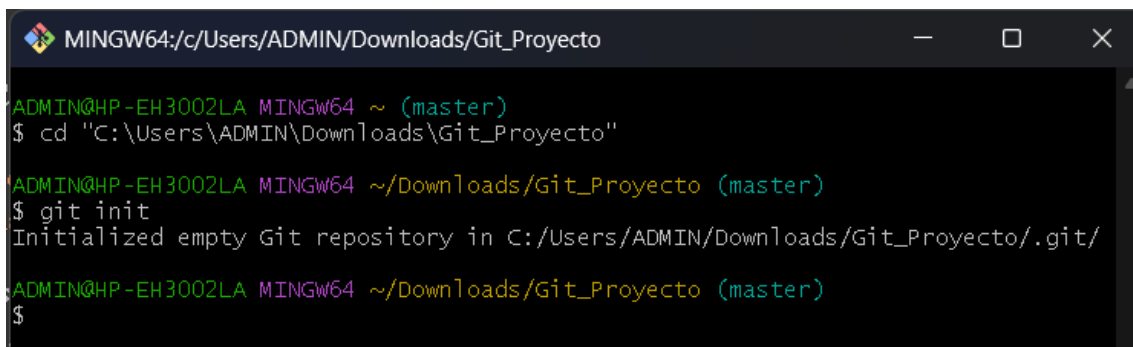
```
Date: Thu Jun 19 14:22:11 2025
```

```
Añadido script principal main.py
```

2. Aplica los pasos documentados a la práctica de un Proyecto, capture las principales acciones de Git (10 puntos).

Proyecto de ejemplo: “Calculadora Básica en Python”

1: Crear repositorio local



```
MINGW64:/c/Users/ADMIN/Downloads/Git_Proyecto
ADMIN@HP-EH3002LA MINGW64 ~ (master)
$ cd "C:\Users\ADMIN\Downloads\Git_Proyecto"

ADMIN@HP-EH3002LA MINGW64 ~/Downloads/Git_Proyecto (master)
$ git init
Initialized empty Git repository in C:/Users/ADMIN/Downloads/Git_Proyecto/.git/

ADMIN@HP-EH3002LA MINGW64 ~/Downloads/Git_Proyecto (master)
$
```

2: Crear archivo principal

```
echo "def sumar(a, b): return a + b" > calculadora.py
```

3: Agregar y hacer primer commit

```
git add calculadora.py
git commit -m "Primer commit: función sumar"
```

4: Editar archivo, agregar más funciones

```
echo "def restar(a, b): return a - b" >> calculadora.py

git add calculadora.py
git commit -m "Añadida función restar"
```

5: Ver historial de commits

```
git log --oneline
```

6: Clonar repositorio desde GitHub

```
git clone https://github.com/tuusuario/CalculadoraPython.git  
cd CalculadoraPython
```

INSTRUCCIONES

- Desarrollar guía de primeros pasos con Git.
 - Manejo de branches y merges.
 - Crear y cambiar entre ramas (branches) utilizando comandos git Branch y git checkout.
 - Fusión de branches. Realizar cambios en una rama y fusionarlos con la rama principal.
 - Uso de git merge para combinar ramas.
 - Resolución de conflictos.

DESARROLLO N°2

1. Complete la guía paso a paso con Git (10 puntos).

GUÍA DE PRIMEROS PASOS CON GIT: Manejo de Branches y Merges

1: Crear una nueva rama (branch)

Para crear una nueva rama:

git branch nueva-rama

Esto crea una rama llamada nueva-rama.

2: Cambiar entre ramas (checkout)

Para cambiarnos a la nueva rama:

git checkout nueva-rama

También se puede crear y cambiar en un solo paso con:

git checkout -b nueva-rama

3: Realizar cambios en una rama

Editar o crea un archivo en la rama:

echo "def multiplicar(a, b): return a * b" >> calculadora.py

Guardar el cambio y luego:

git add calculadora.py

git commit -m "Añadida función multiplicar"

4: Cambiar a la rama principal (main)

git checkout main

5: Fusionar ramas con git merge

Una vez en la rama main, ejecuta:

git merge nueva-rama

Esto combinará los cambios de nueva-rama en main.

2. Aplica los pasos documentados a la práctica de un Proyecto, capture las principales acciones de Git (10 puntos).

Proyecto: Calculadora Python

1: Crear una rama para nuevas funciones

```
git checkout -b funciones-avanzadas
```

2: Agregar nueva función

```
echo "def dividir(a, b): return a / b if b != 0 else 'Error'" >>  
calculadora.py
```

```
git add calculadora.py  
git commit -m "Añadida función dividir"
```

3: Volver a la rama principal

```
git checkout main
```

4: Fusionar ramas

```
git merge funciones-avanzadas
```

INSTRUCCIONES

- Colaboración en GitHub.

- Invitar a compañeros a colaborar en el repositorio.
- Clonar un repositorio compartido.
- Pull request: realizar cambios en una rama y crear un Pull Request en GitHub.
- Evidenciar revisión de código.

DESARROLLO N°3

1. Complete la guía paso a paso con Git (10 puntos).

1. Invitar colaboradores

En GitHub: ir a la opción Settings > Collaborators del repositorio

Invitar a mis compañeros por su nombre de usuario GitHub

2. Clonar el repositorio remoto

Mi compañero debe ejecutar:

```
git clone https://github.com/usuario/simulador-cajero.git
```

```
cd simulador-cajero
```

3. Crear una rama para trabajar

Utilizamos el comando **git checkout -b autenticación-usuario**

4. Subir cambios a la rama remota

```
git add .
```

```
git commit -m "Commit confirmado"
```

```
git push origin autenticación-usuario
```

5. Crear un Pull Request

Ir a GitHub y seleccionar la rama

Hacer clic en “**Compare & Pull Request**”

(Escribir una descripción clara de los cambios)

- Asignar a un compañero como revisor

6. Revisión de código y fusión

El revisor puede comentar y aprobar cambios

Una vez revisado se puede hacer merge con la rama master o main

2. Aplica los pasos documentados a la práctica de un Proyecto, capture las principales acciones de Git (10 puntos).

Aplicación práctica del flujo Git en el proyecto del simulador de cajero automático

1. Inicialización del repositorio local

```
cd simulador-cajero  
git init
```

- Se inicia el proyecto localmente para comenzar el versionado.

2. Creación de archivos base del simulador

- Creamos las carpetas

3. Primer commit

```
git add .  
git commit -m "Estructura de carpetas base del cajero automático"
```

- Se guarda el estado inicial del proyecto con su estructura modular.

4. Vinculación con el repositorio remoto en GitHub

```
git remote add origin https://github.com/KIA/cajero-educativo.git  
git push -u origin master
```

- Se conecta el repositorio local con GitHub y se sube el primer commit.

5. Creación de una rama para implementar autenticación

```
git checkout -b autenticacion-pin
```

- Se crea una nueva rama para trabajar sin afectar la rama principal.

6. Cambios y registro en la rama

```
git add login.cpp
```

```
git commit -m "Implementación de verificación de PIN de 4 dígitos"
```

```
git push origin autenticacion-pin
```

7. Pull Request para revisión del equipo

- Desde GitHub buscar el “Comparar y hacer Pull Request” → "Agregado el módulo de autenticación PIN"
- Se asigna un revisor del equipo.

8. Revisión y merge

- El revisor comenta que falta validación por número de intentos.
- Se actualiza login.cpp, se hace un nuevo commit, y se aprueba el Pull Request para fusionar con master.

9. Desarrollo de transacciones: nueva rama

```
git checkout -b transacciones-basicas
```

- Se agregan funciones para retiro, depósito, y verificación de saldo.

10. Registro y colaboración

```
git add transacciones.cpp
```

```
git commit -m "Funciones de retiro y depósito con actualización de saldo"
```

```
git push origin transacciones-basicas
```

- Se documentan cambios, se colabora y se suben ramas temáticas para revisión.

INSTRUCCIONES

· Mejores prácticas y comandos avanzados.

- Convención de nombres, mensajes de commit claros, importancia de branches.
- Estrategias de ramificación (Git Flow, por ejemplo)
- Comandos avanzados: git stash, git revert y git Cherry-pick

DESARROLLO N°4

1. Complete la guía paso a paso con Git (10 puntos).

Guía paso a paso con Git (Simulador de Cajero Automático)

1. Convención de nombres y mensajes de commit

Nombres de ramas (snake_case o kebab-case):

- autenticacion-usuario
- retirar-dinero-logica
- interfaz-grafica

Buenas prácticas para commits:

- Escribir en presente: **Agrega validación de PIN**
- Ser específicos: **Corrige bug en retiro cuando saldo es cero**
- Evitar mensajes como **arreglo, commit final, o cambio**

1

2. Uso de ramas (Branching)

git checkout -b interfaz-grafica

3. Estrategia de ramificación – Git Flow (Simplificada)

Rama	Propósito
<code>main</code>	Versión estable de producción
<code>develop</code>	Integración del trabajo en curso
<code>feature/*</code>	Desarrollo de nuevas funcionalidades
<code>hotfix/*</code>	Correcciones urgentes a producción

```
git checkout develop
git checkout -b feature/retiro-monto
```

4. Subir los cambios

```
git push origin feature/retiro-monto
```

5. Crear un Pull Request

Desde GitHub:

- Comparar rama con `develop`.
- Agregar título claro.
- Describir qué se hizo y por qué.
- Asignar revisores.

6. Comandos avanzados útiles

`git stash` Guarda tus cambios temporalmente si necesitas cambiar de rama:

```
git stash
```

```
git checkout otra-rama
```

git stash pop

git revert Deshace un commit sin borrar el historial:

git revert 4f3e0d2

git cherry-pick Aplica un commit específico de otra rama:

git cherry-pick 7a1bc9e

7. Fusión de ramas y limpieza

git checkout develop

git merge feature/retiro-monto

git branch -d feature/retiro-monto

2. Aplica los pasos documentados a la práctica de un Proyecto, capture las principales acciones de Git (10 puntos).

Aplicación de Git al Proyecto: Simulador de Cajero Automático

1. Inicialización del repositorio

mkdir simulador-cajero

cd simulador-cajero

git init

> Se creó el repositorio local donde se alojará el código del simulador.

2. Archivos iniciales del proyecto

touch login.cpp transacciones.cpp interfaz.cpp README.md

3. Primer commit

git add .

git commit -m "Estructura base del simulador: login, interfaz y transacciones"

4. Vinculación con el repositorio remoto

git remote add origin https://github.com/KIA/simulador-cajero.git

git push -u origin main

5. Creación de rama para desarrollo de autenticación

git checkout -b feature/autenticacion-pin

6. Modificación del código y commit temático

git add login.cpp

git commit -m "Implementa validación de PIN con 3 intentos máximos"

7. Subir rama al repositorio remoto

git push origin feature/autenticacion-pin

8. Creación de Pull Request y revisión colaborativa

> Desde GitHub, se creó un Pull Request describiendo la nueva funcionalidad. > Un compañero revisó el código, hizo sugerencias, se mejoró y aprobó.

9. Fusión de la rama en **main**

git checkout main

git merge feature/autenticacion-pin

10. Uso de comando avanzado **git stash para cambiar de tarea**

git stash

git checkout develop

git stash pop

INSTRUCCIONES

En grupos de Proyecto:

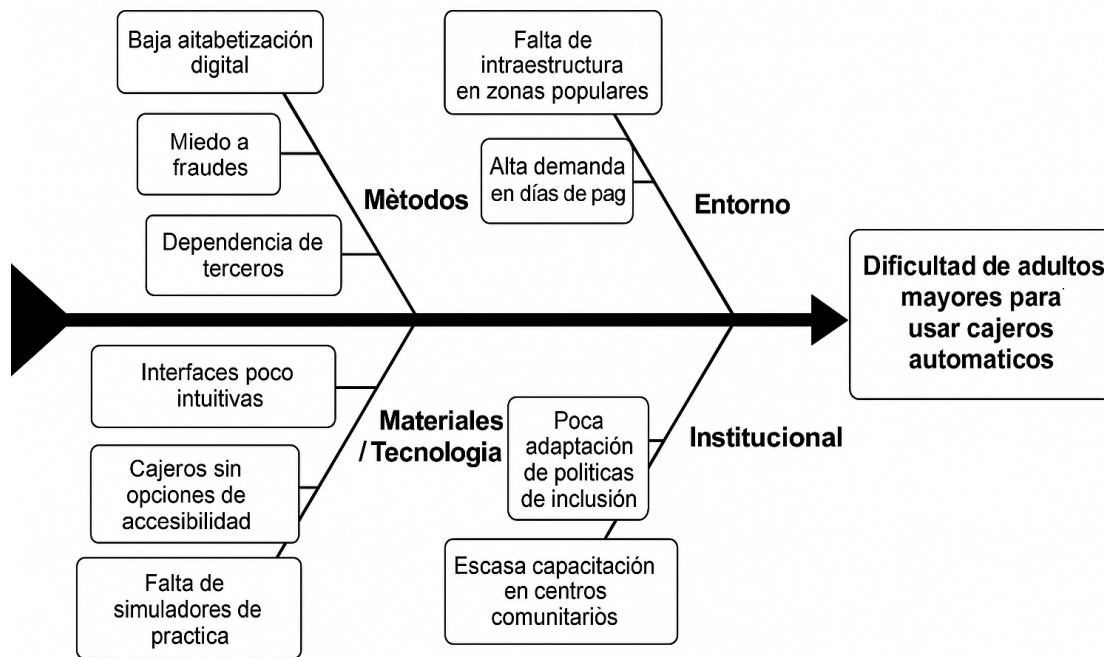
Identificar y formular el problema (diagrama de Ishikawa u otro), documentar antecedentes adecuadamente (Cita APA).

DESARROLLO N°6

1. Identificación del problema, mediante la redacción de la realidad problemática (5 puntos).

En el distrito de Chorrillos, Lima, muchos adultos mayores y personas con poca experiencia digital enfrentan serias dificultades para utilizar cajeros automáticos, lo que limita su acceso a servicios financieros básicos como retirar dinero, consultar saldos o realizar transferencias. Esta situación se agrava debido a la falta de educación tecnológica en este grupo etario, así como por el temor a cometer errores o ser víctimas de fraudes. El problema afecta especialmente a personas mayores de 60 años, pero también a usuarios con bajo nivel de alfabetización digital, quienes dependen frecuentemente de terceros para realizar sus transacciones bancarias. Esta dependencia no solo vulnera su autonomía económica, sino que también puede exponerlos a situaciones de abuso o pérdida de privacidad. Además, muchas veces se generan largas colas y demoras en agencias bancarias porque estos usuarios prefieren ser atendidos por personal en lugar de usar los cajeros automáticos. La falta de iniciativas tecnológicas inclusivas contribuye a que este problema persista. Este fenómeno ocurre de manera constante, especialmente durante los días de pago de pensiones o bonos, en zonas como Chorrillos donde hay una alta concentración de adultos mayores. Desarrollar un simulador de cajero automático permitirá que estas personas puedan practicar en un entorno seguro y libre de riesgos, facilitando su aprendizaje y familiarización con los servicios bancarios digitales.

2. Formulación del problema mediante el diagrama Ishikawa o similar (5 puntos).



3. Redactar antecedentes internacionales, nacionales y locales. (5 puntos).

Antecedentes Internacionales

El problema de la inclusión digital en adultos mayores es una preocupación global. Según la Organización Mundial de la Salud (OMS), el acceso a la tecnología debe formar parte del envejecimiento saludable, sin embargo, muchos adultos mayores carecen de habilidades básicas para interactuar con herramientas digitales (WHO, 2021). Un estudio en España identificó que el 43% de adultos mayores tienen dificultades para realizar operaciones bancarias digitales, lo que incrementa su exclusión financiera (Rodríguez et al., 2020).

Antecedentes Nacionales (Perú)

En el Perú, el Banco Central de Reserva (BCRP) reportó en 2023 que el 30% de adultos mayores no utiliza servicios bancarios digitales por desconocimiento o miedo a cometer errores. Además, el Ministerio de Desarrollo e Inclusión Social (MIDIS) ha implementado programas para mejorar la inclusión financiera, pero estos no siempre están adaptados a los adultos mayores o personas con baja alfabetización digital (MIDIS, 2023). Según un informe del INEI (2022), solo el 18% de personas mayores de 60 años en Lima acceden frecuentemente a servicios digitales, lo que refleja una brecha importante.

Antecedentes Locales (Chorrillos, Lima)

Chorrillos es uno de los distritos con mayor población adulta mayor en Lima Metropolitana. Muchos de estos residentes enfrentan barreras tecnológicas, agravadas por la falta de infraestructura digital y programas de capacitación accesibles en centros

comunitarios o clubes de adultos mayores. En una encuesta local realizada por la Municipalidad de Chorrillos en 2022, se encontró que el 62% de adultos mayores no saben usar correctamente un cajero automático y dependen de terceros para realizar transacciones básicas. Esto aumenta su vulnerabilidad y disminuye su independencia financiera.

4. Aplicar la citación APA 7 en la información. (5 puntos).

INEI. (2022). Encuesta Nacional de Hogares sobre Condiciones de Vida y Pobreza 2022. Instituto Nacional de Estadística e Informática.
<https://www.inei.gob.pe/>

MIDIS. (2023). Informe sobre inclusión financiera en adultos mayores. Ministerio de Desarrollo e Inclusión Social del Perú. <https://www.gob.pe/midis>

Rodríguez, M., López, J., & García, F. (2020). Exclusión financiera en adultos mayores: barreras y soluciones digitales. *Revista Española de Gerontología*, 38(2), 77-89. <https://doi.org/10.1016/j.reg.2020.04.003>

World Health Organization. (2021). Digital inclusion for healthy ageing: Guidance and toolkit. <https://www.who.int/publications/i/item/9789240039230>

INSTRUCCIONES

En grupos de Proyecto:

- Identificar restricciones realistas que afectan al proyecto y propone alternativas de solución en base a ellas.
- Establecer objetivos del proyecto.
- Alcance de la solución y objetivos.

DESARROLLO N°7

1. Identificar restricciones del proyecto (5 puntos).

Restricciones técnicas:

- Limitado acceso a dispositivos tecnológicos (tabletas, PCs) por parte de los usuarios objetivo.
- Algunos adultos mayores pueden tener discapacidades visuales, auditivas o motrices que dificultan la interacción con el simulador.

Restricciones presupuestales:

- Recursos financieros limitados para el desarrollo completo del simulador y su implementación en centros comunitarios o bibliotecas.

Restricciones humanas:

- Escasez de personal capacitado en tecnología educativa para adultos mayores.
- Posible resistencia al cambio por parte del público objetivo.

Restricciones temporales:

- Tiempo limitado para desarrollar, probar y validar el simulador antes de su implementación piloto.

Restricciones logísticas:

- Dificultades para implementar el simulador en zonas con limitada conectividad a internet o falta de infraestructura básica.

2. Propuesta de alternativas de solución en el proyecto realistas. (5 puntos).

Simulador digital de cajero automático (opción principal):

- Software educativo interactivo que simula el uso de un cajero automático con interfaz amigable y accesible (letras grandes, instrucciones por voz).
- Instalado en centros comunitarios, clubes de adultos mayores, o incluso accesible desde celulares o tablets.

Capacitaciones presenciales con maquetas físicas:

- Uso de modelos físicos de cajeros automáticos con pantallas simuladas para enseñanza práctica en talleres comunitarios.

Videos tutoriales y guías impresas ilustradas:

- Materiales simples, visuales y adaptados al público objetivo que expliquen paso a paso cómo usar un cajero automático.

Aplicación móvil de práctica:

- App descargable desde celulares que simula la experiencia bancaria en modo seguro y sin riesgo de perder dinero.

3. Establecer objetivos del proyecto. (5 puntos).

Objetivo general:

Desarrollar e implementar un simulador educativo de cajero automático para capacitar a adultos mayores y personas con baja alfabetización digital del distrito de Chorrillos, promoviendo su inclusión financiera y autonomía.

Objetivos específicos:

1. Diseñar un simulador interactivo que reproduce las funciones básicas de un cajero automático (retiro, consulta de saldo, transferencia).
2. Validar la facilidad de uso del simulador con un grupo piloto de adultos mayores en Chorrillos.
3. Implementar el simulador en espacios accesibles como centros comunitarios y clubes de adultos mayores.
4. Reducir el nivel de dependencia tecnológica en adultos mayores al permitirles practicar en un entorno seguro.
5. Evaluar el impacto del simulador mediante encuestas antes y después de su uso.

4. Establecer el alcance del proyecto. (5 puntos).

Alcance del proyecto:

- **Ámbito geográfico:** El proyecto se desarrollará y aplicará inicialmente en el distrito de Chorrillos, Lima.
- **Usuarios objetivo:** Adultos mayores (60 años o más) y personas con baja alfabetización digital.

- **Fases del proyecto:**

- Diagnóstico y recolección de necesidades del usuario.
- Diseño y desarrollo del simulador digital.
- Validación piloto con un grupo reducido.

- Capacitación a facilitadores comunitarios.
- Implementación en centros seleccionados de Chorrillos.
- Monitoreo y evaluación de resultados.
- **Exclusiones:**
 - El proyecto **no abarca** el desarrollo de software para cajeros reales ni operaciones financieras reales.
 - No incluye la capacitación a personal bancario ni modificaciones a cajeros existentes.
 - El proyecto **no garantiza** el acceso individual a dispositivos móviles por parte de cada usuario.

INSTRUCCIONES

En grupos de Proyecto:

- Documentar al menos 40 requerimientos funcionales.

- Elaborar historias de usuario.

DESARROLLO N°8

1. Estructurar el diseño detallado de la historia de usuario (5 puntos).

ID	Historia de Usuario	Criterios de Aceptación
HU01	Como cliente, quiero ingresar mi DNI de 8 dígitos, para poder acceder de manera segura.	<ul style="list-style-type: none"> - El sistema solo permite avanzar si el DNI tiene exactamente 8 dígitos numéricos. - Se muestra un mensaje de error si el formato es inválido. - Se limita a 3 intentos.
HU02	Como cliente, quiero ingresar un PIN de 4 dígitos al retirar dinero, para proteger mi cuenta.	<ul style="list-style-type: none"> - Solo se permite el retiro si el PIN es correcto. - Si el PIN es incorrecto, se muestra un mensaje. - El retiro no se ejecuta sin validación de PIN.
HU03	Como cliente, quiero consultar mi saldo, para saber cuánto dinero tengo disponible.	<ul style="list-style-type: none"> - Se muestra el saldo actual del cliente en pantalla. - No se modifica el saldo. - Puede realizarse en cualquier momento tras ingresar al sistema.

HU04	Como cliente, quiero ingresar un monto a depositar, para aumentar mi saldo.	<ul style="list-style-type: none"> - El sistema acepta montos positivos numéricos. - Se actualiza correctamente el saldo del cliente. - Se notifica al usuario del nuevo saldo.
HU05	Como cliente, quiero retirar dinero solo si tengo saldo suficiente, para evitar sobregiros.	<ul style="list-style-type: none"> - El sistema verifica el saldo antes de retirar. - Si no hay fondos suficientes, se muestra un mensaje y no se procesa. - El saldo se actualiza correctamente si procede.
HU06	Como cliente, quiero recibir un recibo al realizar una operación, para tener evidencia.	<ul style="list-style-type: none"> - Después de una operación, se genera un archivo <code>.txt</code> con los detalles. - El recibo incluye nombre, operación, monto, fecha y saldo actual.
HU07	Como cliente, quiero decidir si deseo generar un recibo o no, para ahorrar papel.	<ul style="list-style-type: none"> - El sistema pregunta al usuario si desea el recibo. - Si elige "No", no se genera ningún archivo. - Si elige "Sí", se genera normalmente.
HU08	Como cliente, quiero que el recibo se abra automáticamente en el Bloc de notas, para revisar fácilmente los detalles.	<ul style="list-style-type: none"> - El archivo generado se abre automáticamente con <code>notepad</code> en Windows. - El contenido del recibo es legible y bien estructurado.
HU09	Como cliente, quiero que cada recibo tenga un número único de operación, para identificar o reportar la transacción fácilmente.	<ul style="list-style-type: none"> - Cada operación incluye un número único de 10 dígitos en el recibo. - Este número no se repite. - Está claramente visible en el recibo.

HU10	Como cliente, quiero ver el número de operación en una ventana emergente, para anotarlo rápidamente si lo necesito	<ul style="list-style-type: none"> - Después de cada operación, si se genera un recibo, se muestra el número en un JOptionPane. - El cliente puede anotar o copiar el número.
HU11	Como cliente, quiero una interfaz clara con botones, para usar el sistema fácilmente.	<ul style="list-style-type: none"> - La GUI incluye botones para consultar, depositar, retirar y salir. - Cada botón funciona y llama a la operación correspondiente.
HU12	Como cliente, quiero recibir mensajes claros si ingreso datos inválidos, para saber que corregir	<ul style="list-style-type: none"> - Si se ingresa un monto no numérico o negativo, se muestra un error. - Si el DNI o PIN tienen formato incorrecto, se notifica. - No se realizan operaciones erróneas.
HU13	Como sistema, quiero limitar intentos de ingreso de DNI, para evitar accesos no autorizados.	<ul style="list-style-type: none"> - Se permiten máximo 3 intentos fallidos de DNI. - Luego de eso, el sistema bloquea o finaliza la sesión. - Cada intento muestra feedback.
HU14	Como cliente, quiero que el sistema valide que los montos sean válidos, para evitar errores.	<ul style="list-style-type: none"> - El sistema valida que el monto sea numérico y mayor que cero. - Se muestra error si el monto es inválido. - Solo se ejecuta la operación si el monto es válido.
HU15	Como cliente, quiero recibir confirmación de que la operación fue exitosa para sentirme seguro de que se completó correctamente	<ul style="list-style-type: none"> - Cada operación muestra un mensaje de éxito en pantalla. - Si el recibo se genera, se indica el número de operación. - En caso de error, se informa con detalle.

2. Ejecutar y elaborar el recojo de información de la empresa, de tal manera que se elaboren las historias de usuario reales. (5 puntos).

Hallazgo (Recojo de Información)	Historia(s) de Usuario Derivada(s)
Muchos jóvenes no saben cómo usar un cajero, y cometen errores con montos.	HU03: Como cliente, quiero ver mi saldo antes de operar, para no equivocarme.
Los adultos mayores olvidan su PIN o lo ingresan mal repetidamente.	HU05: Como cliente, quiero tener 3 intentos para ingresar mi DNI correctamente.
Usuarios desean saber si su transacción fue exitosa y quieren una constancia.	HU08, HU10: Como cliente, quiero generar un recibo/consulta, para guardar un respaldo.
Las interfaces deben ser claras, sin saturación visual, con colores amigables.	HU15: Para ofrecer una buena experiencia, quiero una interfaz amigable y clara.
Usuarios solicitan confirmación de operaciones con número de transacción.	HU12: Para asegurar mi transacción, quiero que aparezca un número de operación.
Algunos usuarios se equivocan al retirar y no comprenden bien el saldo restante.	HU06: Como cliente, quiero saber mi nuevo saldo tras retirar.
Jóvenes desean practicar operaciones para familiarizarse antes de usar un cajero real.	HU01, HU02, HU13: Como cliente nuevo, quiero practicar con un cajero simulado.
Adultos mayores no saben si deben ingresar monto antes o después de elegir operación.	HU09, HU11: Para no confundirme, quiero instrucciones claras al depositar o retirar.
Algunos clientes prefieren no imprimir recibo para no gastar papel.	HU08: Como cliente, quiero elegir si deseo recibo.
Personal de capacitación quiere que el sistema permita verificar acciones comunes.	HU14: Como encargado, quiero que el sistema permita múltiples simulaciones.
Algunos usuarios olvidan cerrar la sesión.	HU07: Como cliente, quiero tener un botón de salida claro y visible.
Usuarios deben validar su identidad para evitar suplantaciones.	HU04: Como cliente, quiero validar mi identidad con DNI.

3. Establecer y documentar los requerimientos funcionales. (8 puntos).

ID

Requerimientos Funcionales

RF01

El sistema debe permitir el acceso de un cliente mediante la validación de un DNI numérico de exactamente 8 dígitos.

- RF02** El sistema debe permitir consultar el saldo actual del cliente una vez que ha iniciado sesión correctamente.
- RF03** El sistema debe permitir depositar un monto positivo en la cuenta del cliente y actualizar el saldo en tiempo real.
- RF04** El sistema debe permitir realizar un retiro de dinero solo si el cliente valida correctamente su PIN de 4 dígitos.
- RF05** El sistema debe verificar que el cliente tenga fondos suficientes antes de permitir una operación de retiro.
- RF06** El sistema debe mostrar un menú gráfico que incluya las opciones: consultar saldo, depositar, retirar y salir del sistema.
- RF07** El sistema debe generar un archivo de recibo en formato .txt que incluya: nombre del cliente, DNI, tipo de operación, monto y saldo.
- RF08** El sistema debe mostrar al cliente una ventana para confirmar si desea generar un recibo antes de crear el archivo.
- RF09** El recibo generado debe abrirse automáticamente en el Bloc de notas para que el cliente lo visualice sin buscarlo manualmente.
- RF10** Cada recibo generado debe contener un número de operación único y aleatorio, el cual también se debe mostrar en una ventana emergente.
- RF11** El sistema debe permitir al cliente cambiar su PIN de 4 dígitos una vez validado el actual.
- RF12** El sistema debe registrar un historial de operaciones del cliente en un archivo.

- RF13** El sistema debe mostrar el historial de operaciones realizadas en la sesión actual.
- RF14** El sistema debe permitir cerrar sesión manualmente desde un botón en la interfaz.
- RF15** El sistema debe mostrar el nombre completo del cliente después de ingresar el DNI.
- RF16** El sistema debe permitir registrar nuevos clientes desde una interfaz de administrador.
- RF17** El sistema debe permitir eliminar una cuenta de cliente (solo con credenciales de administrador).
- RF18** El sistema debe encriptar el PIN del cliente para protegerlo en el almacenamiento local.
- RF19** El sistema debe emitir alertas si se detectan más de 3 intentos fallidos de ingreso.
- RF20** El sistema debe ofrecer la opción de impresión física del recibo si se encuentra una impresora conectada.
- RF21** El sistema debe calcular y mostrar el saldo promedio mensual del cliente.
- RF22** El sistema debe permitir simular un préstamo y calcular cuotas mensuales con interés.
- RF23** El sistema debe permitir transferencias simuladas entre cuentas de clientes registrados.

- RF24** El sistema debe mostrar el logo de la institución (BCP) en cada pantalla o ventana.
- RF25** El sistema debe registrar la fecha y hora exacta de cada transacción.
- RF26** El sistema debe validar que el monto ingresado tenga máximo dos decimales.
- RF27** El sistema debe bloquear el acceso durante 10 segundos tras 3 intentos fallidos.
- RF28** El sistema debe tener una interfaz de ayuda accesible desde el menú principal.
- RF29** El sistema debe mostrar un resumen financiero con ingresos (depósitos) y egresos (retiros).
- RF30** El sistema debe permitir al cliente actualizar su nombre en caso de error (con validación de PIN).
- RF31** El sistema debe enviar un correo electrónico simulado con el recibo, si el cliente registra uno.
- RF32** El sistema debe mostrar una barra de progreso al ejecutar operaciones largas.
- RF33** El sistema debe usar colores institucionales del BCP para su interfaz.
- RF34** El sistema debe emitir sonidos o alertas sonoras ante errores o confirmaciones (si el sistema lo permite).

- RF35** El sistema debe permitir al cliente configurar un límite diario de retiro simulado.
- RF36** El sistema debe mostrar mensajes de bienvenida personalizados usando el nombre del cliente.
- RF37** El sistema debe permitir configurar la moneda de operación (Soles o Dólares).
- RF38** El sistema debe ofrecer accesibilidad para personas con visión reducida (ej: letras grandes).
- RF39** El sistema debe contar con modo claro/oscuro según preferencia del usuario.
- RF40** El sistema debe registrar automáticamente todas las operaciones en un archivo log de auditoría para revisión técnica.

4. Establecer y documentar los requerimientos no funcionales. (2 puntos)

ID	Requerimiento No Funcional
RNF01	El sistema debe responder a cualquier acción del usuario en un tiempo no mayor a 2 segundos.
RNF02	La interfaz gráfica debe ser intuitiva y fácil de usar, incluso para usuarios sin conocimientos técnicos.
RNF03	Los datos sensibles como el PIN deben ser enmascarados visualmente y protegidos contra acceso no autorizado.

- RNF04** El sistema debe ser compatible con Windows 10 o superior, y funcionar correctamente en resoluciones desde 1024x768.
- RNF05** El sistema debe ser modular y fácilmente mantenible, permitiendo agregar nuevas funciones sin alterar el núcleo.

INSTRUCCIONES

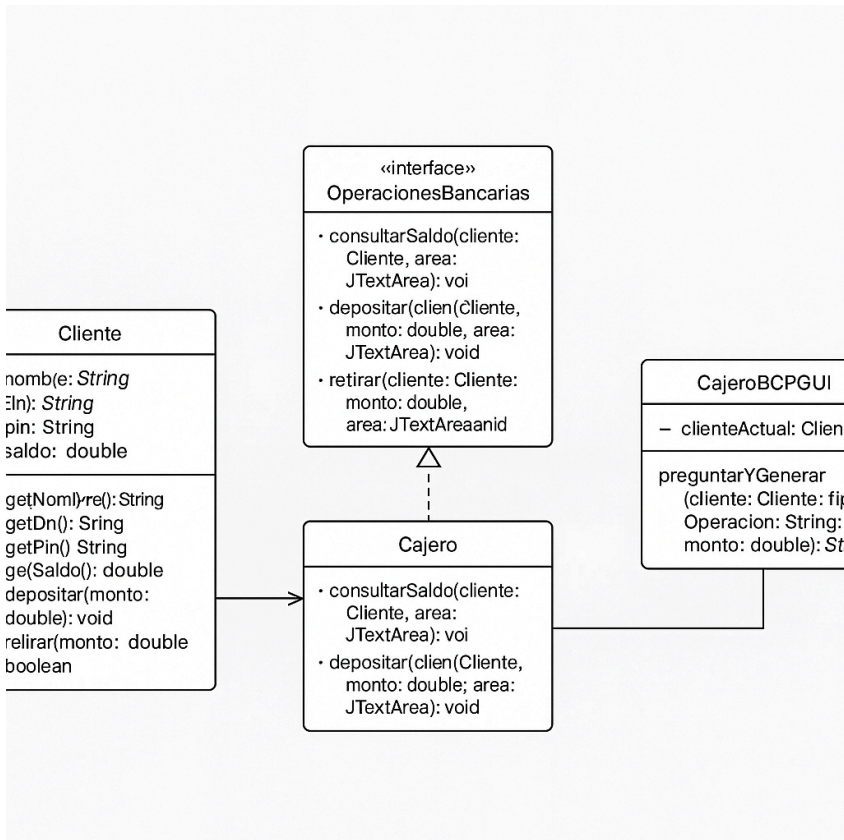
En grupos de Proyecto:

- Elaborar diagrama de clases.

DESARROLLO N°9

1. Diseñar los diagramas de clases del proyecto (5 puntos).

El siguiente es el diagrama de clases del proyecto, con las clases principales que conforman el simulador de cajero automático:



2. Relacionar los diagramas de clase creados con la programación implementada del proyecto. (5 puntos).

A continuación se muestra cómo se refleja el diagrama en el código Java:

1. Clase **Cliente**

En el código:

Clase que modela a un cliente del banco con:

- Atributos: **nombre**, **dni**, **pin**, **saldo**
- Métodos: **getNombre()**, **getDni()**, **getPin()**, **getSaldo()**, **depositar()**, **retirar()**

En el diagrama de clases:

Representado como una clase concreta con los mismos atributos y métodos. Cumple con encapsulamiento y operaciones necesarias.

Relación:

Clase fundamental usada en todo el flujo del programa. Instancias se crean en el **main** y se pasan a la interfaz.

2. Interfaz OperacionesBancarias

En el código:

Define 3 métodos:

- consultarSaldo()
- depositar()
- retirar()

En el diagrama de clases:

Representado correctamente como una interfaz (<<interface>>) que luego es implementada por la clase Cajero.

Relación:

Promueve la separación de responsabilidades y permite una implementación modular.

3. Clase Cajero

En el código:

Implementa la interfaz OperacionesBancarias.

Define la lógica para:

- consultar saldo
- depositar dinero
- retirar dinero (validando fondos)

En el diagrama de clases:

Se muestra implementando la interfaz con los métodos sobrescritos y relacionados con la clase Cliente.

Relación:

Encapsula la lógica de negocio del cajero. Es utilizada por la clase de interfaz gráfica para ejecutar acciones.

4. Clase CajeroBCPGUI

En el código:

Clase principal con JFrame que representa la interfaz de usuario con botones,

imágenes, validación de DNI, PIN y mensajes.

Contiene **outputArea**, botones (**JButton**) y lógica de interacción.

En el diagrama de clases:

Representada como una clase con el atributo **clienteActual** y uso del método estático **preguntarYGenerar()** de la clase **Recibo** (interna).

Relación:

Controla la interacción con el usuario final. Llama a métodos de **Cajero** según la opción del usuario.

5. Clase Interna **Recibo**

En el código:

Clase estática declarada dentro de **CajeroBCPGUI**.

Genera un recibo en archivo **.txt** con datos del cliente y abre automáticamente el Bloc de Notas.

En el diagrama:

No está explícitamente representada.

Se puede considerar agregarla en una versión extendida del diagrama.

Relación:

Está vinculada con la lógica de generación de recibos y uso de **Runtime** para abrir Notepad.

Relación entre clases:

- **CajeroBCPGUI** usa a **Cliente** y a **Cajero**.
- **Cajero** implementa **OperacionesBancarias**.
- **Recibo** es parte interna de **CajeroBCPGUI** y accede a **Cliente**.

4. Control de estado de requerimientos funcionales implementados en relación con las historias de usuario. (5 puntos).

A continuación, se presenta el estado actual de implementación de los requerimientos funcionales, según las historias de usuario trabajadas

ID	Requerimiento Funcional	Relacionado a HU	Estado de Implementación	Observaciones
RF01	Validar acceso del cliente con DNI de 8 dígitos.	HU01	✓ Implementado	Controla el formato y permite 3 intentos antes de bloquear.
RF02	Permitir la consulta del saldo actual del cliente.	HU03	✓ Implementado	Mostrado correctamente desde la interfaz con botón.
RF03	Depositar un monto positivo y actualizar el saldo.	HU04	✓ Implementado	Valida números positivos, actualiza saldo y muestra resultado.
RF04	Retirar dinero solo con validación de PIN de 4 dígitos.	HU02	✓ Implementado	Solicita PIN y valida antes de ejecutar la transacción.
RF05	Validar que el cliente tenga saldo suficiente antes del retiro.	HU05	✓ Implementado	Previene sobregiros, muestra mensaje si no hay fondos.
RF06	Mostrar una interfaz gráfica con botones para las operaciones.	HU11	✓ Implementado	Uso de JFrame y JButton, diseño claro.
RF07	Generar un recibo en .txt con los datos de la operación bancaria.	HU06, HU09	✓ Implementado	Recibo incluye nombre, operación, monto, fecha, saldo y número único.
RF08	Preguntar al usuario si desea generar un recibo antes de crearlo.	HU07	✓ Implementado	Diálogo de confirmación JOptionPane funcional.
RF09	Abrir el recibo automáticamente en el Bloc de notas tras una operación.	HU08	✓ Implementado	Usa Runtime.getRuntime().exec("notepad archivo.txt").
RF10	Mostrar número de operación único en una ventana emergente tras la operación.	HU10, HU15	✓ Implementado	Número generado aleatoriamente y mostrado al cliente correctamente.

INSTRUCCIONES

En grupos de Proyecto:

- Documentar criterios de aceptación de cada requerimiento.
- Implementar haciendo uso de Java los requerimientos declarados y personalizados en historias de usuario (15 requerimientos)

DESARROLLO N° 10

1. Crear una tabla de control de los requerimientos y los criterios de aceptación (5 puntos).

ID	Requerimiento Funcional	Criterios de Aceptación (CA)	Estado de Implementación
RF01	Validar acceso del cliente con DNI de 8 dígitos.	<ul style="list-style-type: none"> - El DNI debe contener exactamente 8 dígitos numéricos. - Se debe permitir hasta 3 intentos antes de bloquear el acceso. 	✓ Implementado
RF02	Consultar el saldo actual del cliente.	<ul style="list-style-type: none"> - El usuario debe poder visualizar su saldo en la interfaz. - El saldo no debe modificarse tras esta acción. 	✓ Implementado
RF03	Depositar un monto positivo y actualizar el saldo.	<ul style="list-style-type: none"> - Solo se permite ingresar montos numéricos mayores a cero. - El nuevo saldo debe actualizarse y mostrarse correctamente. 	✓ Implementado
RF04	Retirar dinero solo con validación de PIN de 4 dígitos.	<ul style="list-style-type: none"> - El PIN debe ser exactamente de 4 dígitos. - Si el PIN es incorrecto, el retiro no se realiza. 	✓ Implementado
RF05	Validar que el cliente tenga saldo suficiente antes del retiro.	<ul style="list-style-type: none"> - El sistema debe comprobar si el saldo es suficiente. - Si no hay fondos, se debe mostrar un mensaje de error sin afectar el saldo. 	✓ Implementado
RF06	Mostrar una interfaz gráfica con botones para las operaciones.	<ul style="list-style-type: none"> - El sistema debe tener botones visibles y funcionales para cada operación. - La navegación entre pantallas debe ser clara e intuitiva. 	✓ Implementado
RF07	Generar un recibo .txt con los datos de la operación bancaria.	<ul style="list-style-type: none"> - El recibo debe incluir nombre, tipo de operación, monto, saldo, DNI, fecha y número de operación. - Debe guardarse correctamente como archivo .txt. 	✓ Implementado
RF08	Preguntar al usuario si desea generar un recibo antes de crearlo.	<ul style="list-style-type: none"> - Se debe mostrar un cuadro de diálogo para confirmar. - Si el usuario responde "No", no se debe generar ningún archivo. 	✓ Implementado
RF09	Abrir el recibo automáticamente en el Bloc de notas tras una operación.	<ul style="list-style-type: none"> - El sistema debe abrir Notepad con el recibo generado si el usuario eligió generar uno. - El contenido debe mostrarse sin errores. 	✓ Implementado
RF10	Mostrar número de operación único en una ventana emergente tras la operación.	<ul style="list-style-type: none"> - El número debe ser único (generado aleatoriamente). - Debe mostrarse en una ventana emergente y registrarse en el recibo si este se genera. 	✓ Implementado

2. Relación implementación Java y requerimientos realizados. (15 puntos).

HU	Resumen de Historia de Usuario	Requerimiento Funcional (RF)	Implementación en el Código (Clase / Método / Sección)
HU01	Como cliente, quiero ingresar mi DNI para acceder al sistema.	RF01	main() en CajeroBCPGUI => Validación del DNI (8 dígitos), búsqueda en arreglo, control de intentos.
HU02	Como cliente, deseo validar mi identidad con PIN para retirar dinero.	RF04	retirarBtn.addActionListener => JOptionPane.showInputDialog solicita PIN; validación con clienteActual.getPin()
HU03	Como cliente, quiero ver mi saldo.	RF02	consultarBtn.addActionListener => cajero.consultarSaldo() y se muestra en outputArea
HU04	Como cliente, deseo depositar dinero para aumentar mi saldo.	RF03	depositarBtn.addActionListener => cajero.depositar() + validación del monto
HU05	Como cliente, no quiero que me permitan retirar si no tengo saldo suficiente.	RF05	Método retirar(double monto) en clase Cliente verifica si el monto es menor al saldo
HU06	Como cliente, quiero que me generen un recibo con los datos de la operación.	RF07	Clase interna Recibo => método preguntarYGenerar(...) genera .txt con nombre, saldo, fecha, operación
HU07	Como cliente, quiero elegir si deseo o no un recibo.	RF08	preguntarYGenerar() => uso de JOptionPane.showConfirmDialog antes de crear el recibo
HU08	Para poder visualizar el comprobante, quiero que se abra en el Bloc de notas.	RF09	Dentro de preguntarYGenerar(...) -> Runtime.getRuntime().exec("notepad " + fileName);
HU09	Para validar mi operación, quiero ver el recibo con mi saldo y nombre.	RF07	En el contenido del archivo generado (writer.write()) => se muestra nombre, saldo, operación, etc.
HU10	Para control interno, deseo que cada operación tenga un número único.	RF10	Generación de número de operación aleatorio con Random().nextLong() en preguntarYGenerar()
HU11	Para una mejor experiencia, quiero una interfaz gráfica con botones e imágenes.	RF06	Diseño GUI en CajeroBCPGUI => usamos el JFrame, JButton, JTextArea, JLabel
HU12	Para saber con quién interactúo, quiero ver el logo del banco en todo momento.	RF06	Carga de imagen /logo_bcp.png con ImageIcon, mostrada en JLabel logo
HU13	Como cliente, quiero saber si el DNI ingresado es incorrecto.	RF01	Dentro del while de login => se muestra JOptionPane con error si el DNI no es válido
HU14	Como cliente, deseo que se limiten mis intentos si ingreso datos mal.	RF01	Variable intentos controla hasta 3 intentos fallidos, luego muestra mensaje de acceso denegado
HU15	Para verificar mi transacción, quiero ver el número de operación al terminar.	RF10	JOptionPane.showMessageDialog muestra nroOperacion al cliente luego de depósito o retiro (si generé recibo)