Diplomado en Tecnologías de Información

Curso: Programación IV TI-151

Sistema de Tarjetas

Alcance 1

Integrantes:

Nestor Leiva Villalobos

Rodrigo Rene Elías Ramírez

Profesora:

Milena Mata Sojo

Cartago, 11 febrero 2026

Tabla de contenido

Introducción....3
Problema por Resolver....3
Diagrama de Base de Datos....4
Diagrama de Casos de Uso....6
Diagrama de Clases....7
Capitulo 2: Implementación de Base de Datos....8
Script MySQL....8
Script SQLServer....12
Capitulo 3: Autorizador de Transacciones....15
AUT 1 & 2....19
AUT 3....22
AUT 4....24
AUT 5....25
Capitulo 4: Componentes del Core Bancario....27
Capitulo 5: Simulador de Transacciones....32
Conclusiones....34
LOGROS ALCANZADOS....35
Recomendaciones....36
A. Arquitectura y Escalabilidad....36
B. Seguridad y Cumplimiento....36
C. Monitoreo y Observabilidad....36
D. Calidad del Software....36
E. Base de Datos....36
COMPETENCIAS CONSOLIDADAS....37
Bibliografía....38

Introducción

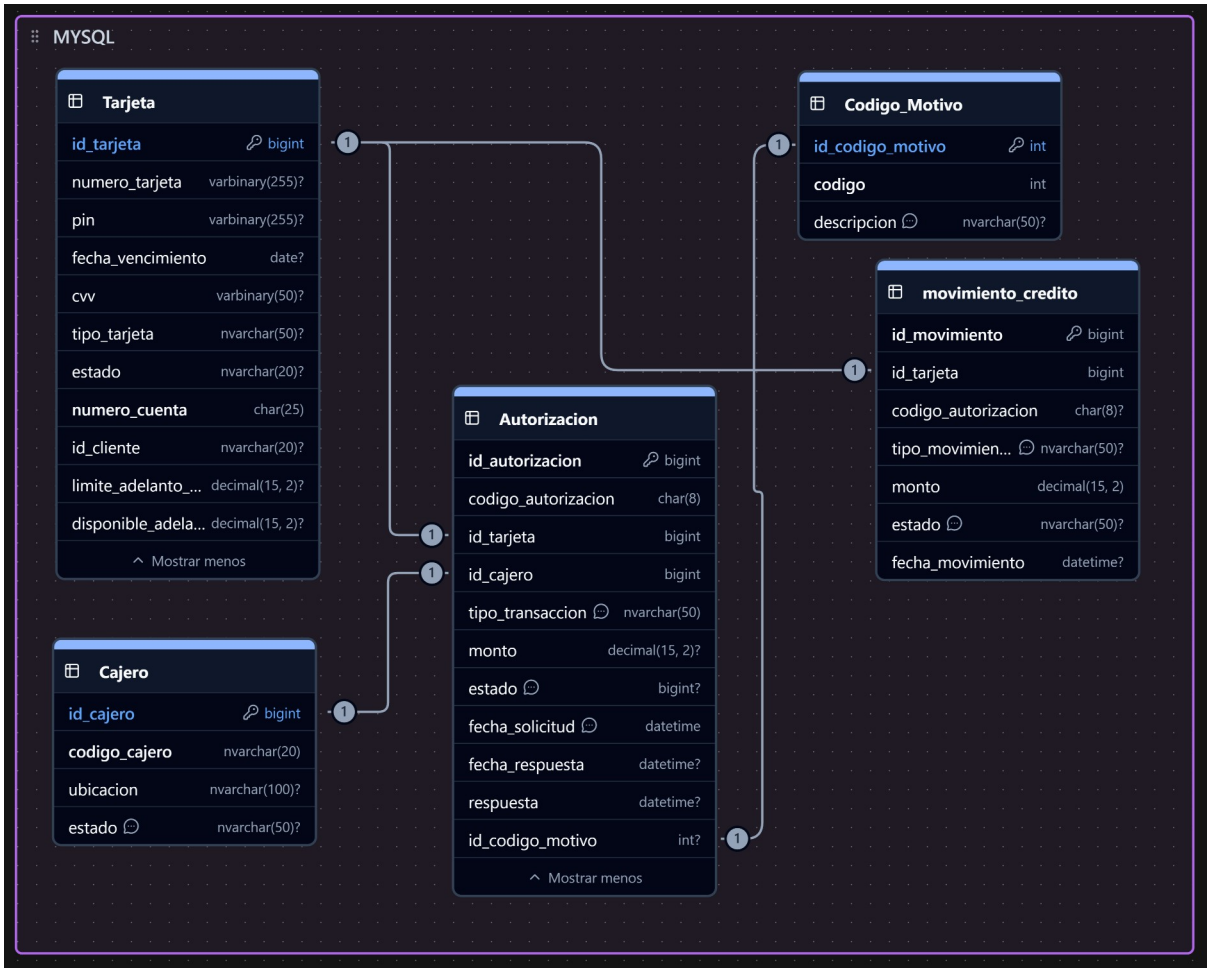
Problema por Resolver

En la actualidad, el sector bancario enfrenta el desafío constante de ofrecer servicios más rápidos, seguros y eficientes para satisfacer las demandas de sus clientes. El Banco ABC ha identificado la necesidad de modernizar su sistema de procesamiento de transacciones con tarjetas de débito y crédito, debido a que su infraestructura actual presenta limitaciones en velocidad, integración y escalabilidad. Estas deficiencias dificultan la administración oportuna de los movimientos financieros y pueden afectar la experiencia del usuario final, especialmente ante el aumento del volumen de operaciones electrónicas.

El nuevo sistema busca optimizar la gestión de las transacciones, garantizando que cada operación —ya sea retiro, depósito, consulta o transferencia— se procese de forma segura y confiable. Además, el proyecto pretende mejorar la interconexión entre los distintos módulos del sistema, asegurando la integridad de los datos y una comunicación efectiva entre las plataformas internas del banco y las redes externas de pago. Con esta renovación tecnológica, el Banco ABC podrá ofrecer un servicio más competitivo, minimizando tiempos de espera y reduciendo los riesgos de errores operativos o fraudes.

Este documento presenta la propuesta de diseño y desarrollo del nuevo sistema, detallando sus componentes, requerimientos y beneficios esperados. De esta manera, se busca sentar las bases para una infraestructura moderna que respalde el crecimiento futuro del banco y mejore la atención a sus clientes.

Diagrama de Base de Datos



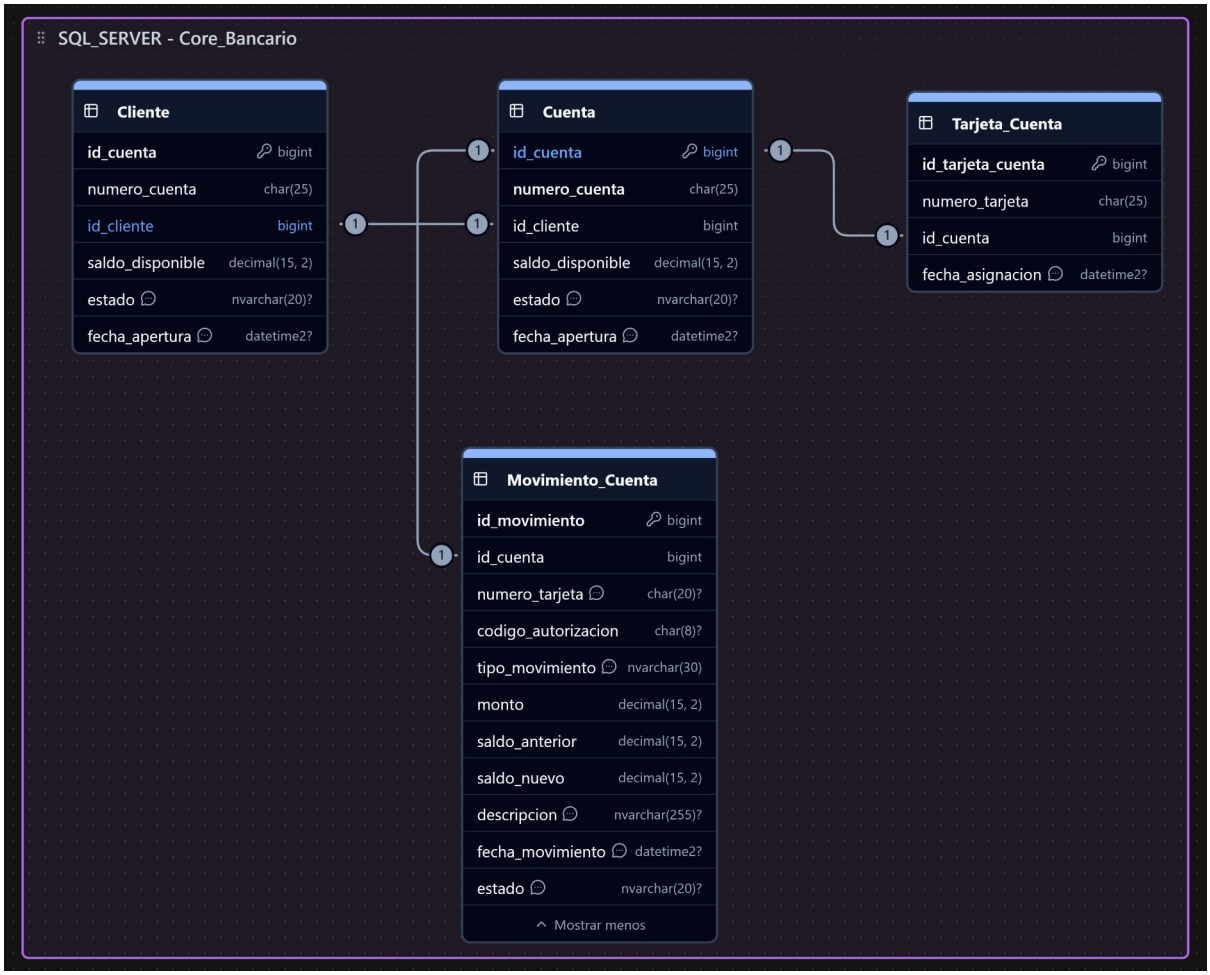

SQL_SERVER - Core_Bancario
Cliente
id_cuenta bigint
numero_cuenta char(25)
id_cliente bigint
saldo_disponible decimal(15, 2)
estado nvarchar(20)?
fecha_apertura datetime2?
Cuenta
id_cuenta bigint
numero_cuenta char(25)
id_cliente bigint
saldo_disponible decimal(15, 2)
estado nvarchar(20)?
fecha_apertura datetime2?
Tarjeta_Cuenta
id_tarjeta_cuenta bigint
numero_tarjeta char(25)
id_cuenta bigint
fecha_asignacion datetime2?
Movimiento_Cuenta
id_movimiento bigint
id_cuenta bigint
numero_tarjeta char(20)?
codigo_autorizacion char(8)?
tipo_movimiento nvarchar(30)
monto decimal(15, 2)
saldo_anterior decimal(15, 2)
saldo_nuevo decimal(15, 2)
descripcion nvarchar(255)?
fecha_movimiento datetime2?
estado nvarchar(20)?
Mostrar menos

Diagrama de Casos de Uso

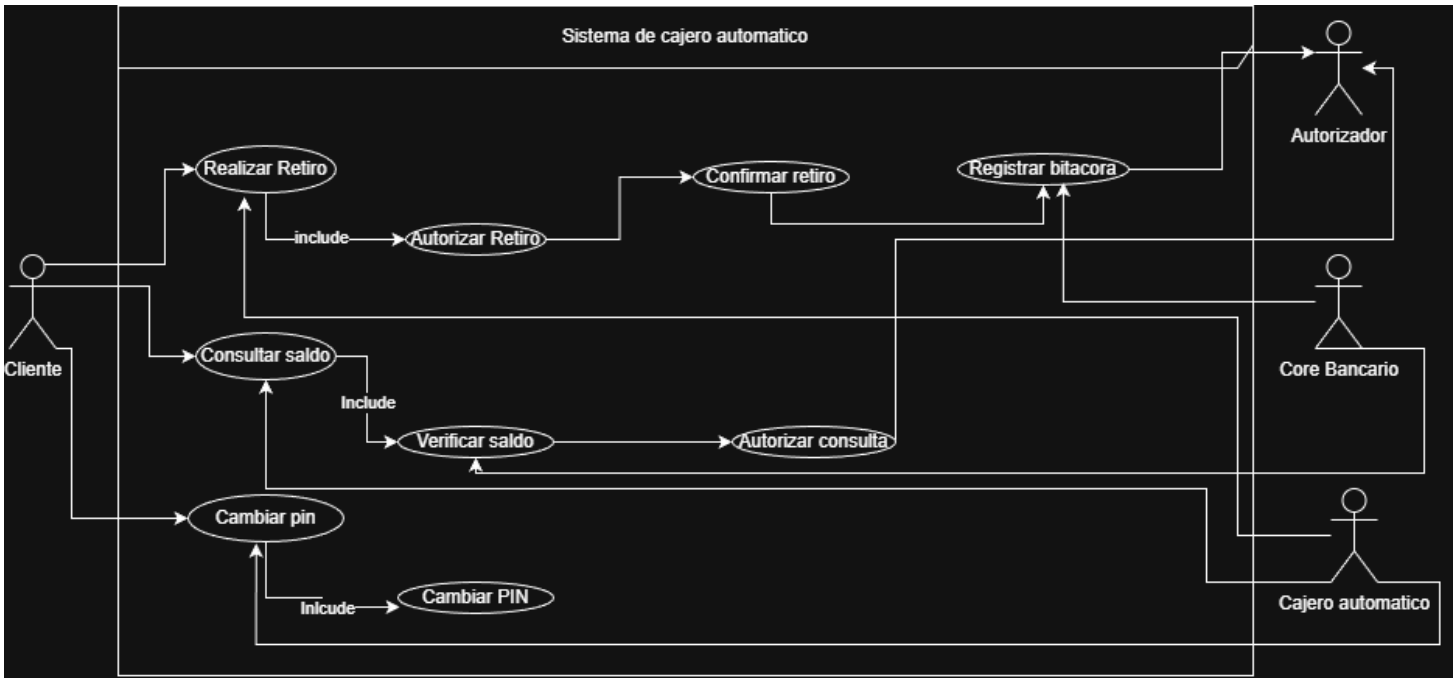
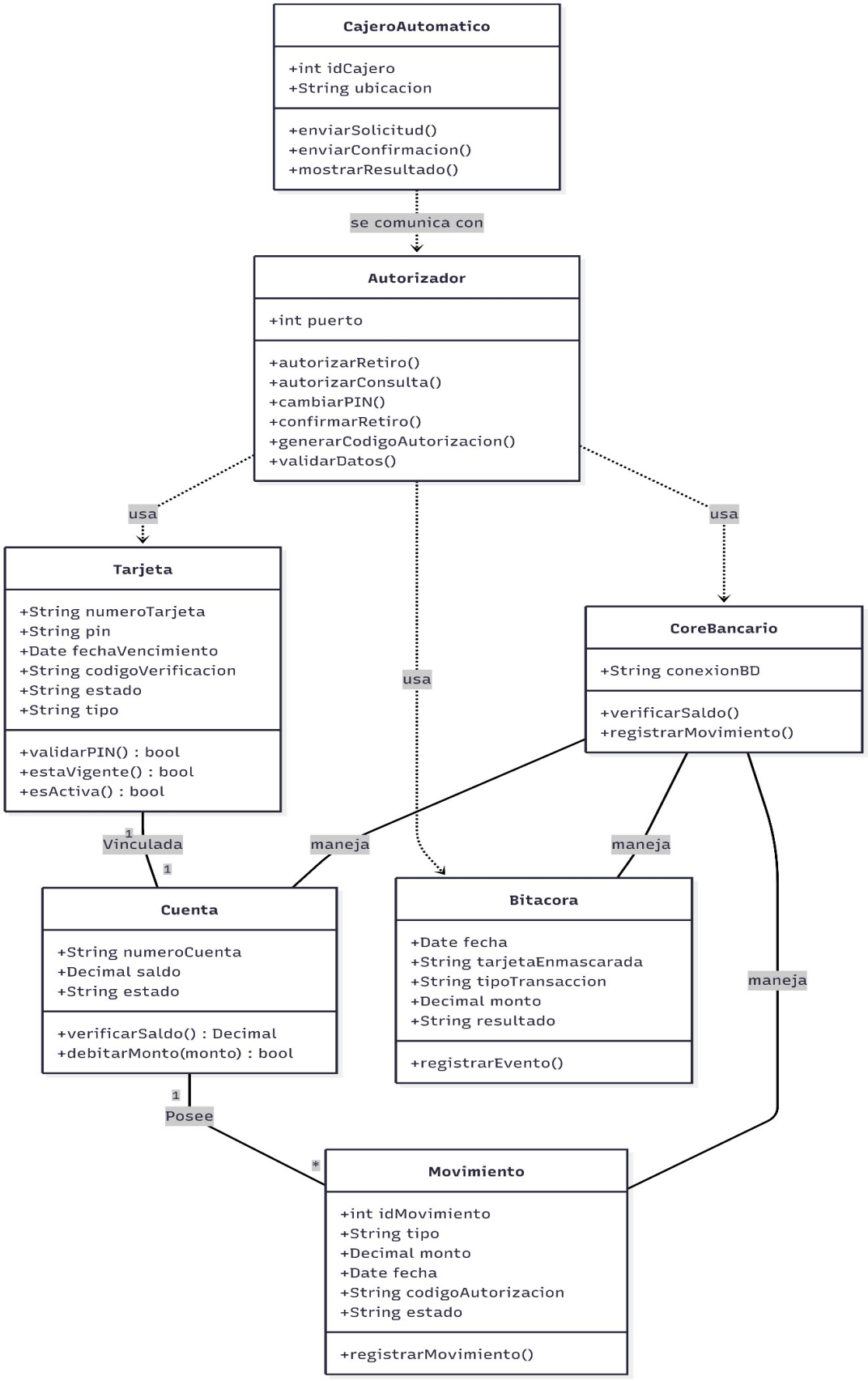


Diagrama de Clases



Capitulo 2: Implementación de Base de Datos

Script MySQL

```
--
================================================================
=====
--              SISTEMA DE TARJETAS DE CRÉDITO / DÉBITO CON AUDITORÍA
--              Proyecto Programación IV - Colegio Universitario de Cartago
--
================================================================
=====

DROP DATABASE IF EXISTS sistema_tarjeta;
CREATE DATABASE sistema_tarjeta CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci;
USE sistema_tarjeta;

-- ============================================
-- TABLAS PRINCIPALES
-- ============================================

CREATE TABLE cajero (
    id_cajero BIGINT PRIMARY KEY AUTO_INCREMENT,
    codigo_cajero VARCHAR(20) UNIQUE NOT NULL,
    ubicacion VARCHAR(100),
    estado VARCHAR(20) DEFAULT 'ACTIVO' CHECK (estado IN ('ACTIVO',
'INACTIVO'))
);

CREATE TABLE codigo_motivo (
    id_codigo_motivo INT PRIMARY KEY AUTO_INCREMENT,
    codigo INT UNIQUE NOT NULL CHECK (codigo BETWEEN 1 AND 5),
    descripcion VARCHAR(100) NOT NULL
);

CREATE TABLE tipo_transaccion (
    id_tipo_transaccion INT PRIMARY KEY AUTO_INCREMENT,
    codigo_tipo VARCHAR(30) UNIQUE NOT NULL,
    descripcion VARCHAR(100) NOT NULL
);

CREATE TABLE cuenta (
    id_cuenta BIGINT PRIMARY KEY AUTO_INCREMENT,
    numero_cuenta VARCHAR(23) NOT NULL UNIQUE,  -- Formato XXXX-XXXX-
XXXX-XX
    tipo_cuenta VARCHAR(10) NOT NULL CHECK (tipo_cuenta IN ('DEBITO',
'CREDITO')),
```

```
    saldo_disponible DECIMAL(15,2) NOT NULL DEFAULT 0 CHECK
(saldo_disponible >= 0),
    saldo_total DECIMAL(15,2) NOT NULL DEFAULT 0 CHECK (saldo_total >= 0),
    limite_credito DECIMAL(15,2) NOT NULL DEFAULT 0 CHECK (limite_credito >=
0),
    disponible_credito DECIMAL(15,2) NOT NULL DEFAULT 0 CHECK
(disponible_credito >= 0),
    estado VARCHAR(20) DEFAULT 'ACTIVA' CHECK (estado IN ('ACTIVA',
'INACTIVA', 'CERRADA'))
);

CREATE TABLE tarjeta (
    id_tarjeta BIGINT PRIMARY KEY AUTO_INCREMENT,
    numero_tarjeta VARCHAR(20) NOT NULL,
    pin VARCHAR(4) NOT NULL,
    fecha_vencimiento DATE NOT NULL,
    cvv VARCHAR(3) NOT NULL,
    tipo_tarjeta VARCHAR(10) NOT NULL CHECK (tipo_tarjeta IN ('DEBITO',
'CREDITO')),
    estado VARCHAR(20) DEFAULT 'ACTIVA' CHECK (estado IN ('ACTIVA',
'INACTIVA', 'VENCIDA')),
    id_cuenta BIGINT NOT NULL,
    identificacion_cliente VARCHAR(20),
    limite_adelanto_efectivo DECIMAL(15,2) DEFAULT 0 CHECK
(limite_adelanto_efectivo >= 0),
    disponible_adelanto_efectivo DECIMAL(15,2) DEFAULT 0 CHECK
(disponible_adelanto_efectivo >= 0),
    UNIQUE KEY uk_numero_tarjeta (numero_tarjeta),
    CONSTRAINT fk_tarjeta_cuenta FOREIGN KEY (id_cuenta) REFERENCES
cuenta(id_cuenta)
);

CREATE TABLE autorizacion (
    id_autorizacion BIGINT PRIMARY KEY AUTO_INCREMENT,
    codigo_autorizacion CHAR(8) NOT NULL UNIQUE,
    id_tarjeta BIGINT NOT NULL,
    id_cajero BIGINT NOT NULL,
    id_tipo_transaccion INT NOT NULL,
    monto DECIMAL(15,2),
    estado VARCHAR(20) DEFAULT 'PENDIENTE' CHECK (estado IN
('PENDIENTE', 'APROBADA', 'RECHAZADA', 'CONFIRMADA')),
    fecha_solicitud DATETIME DEFAULT CURRENT_TIMESTAMP,
    fecha_respuesta DATETIME NULL,
    respuesta VARCHAR(10),
    id_codigo_motivo INT NULL,
    CONSTRAINT fk_autorizacion_tarjeta FOREIGN KEY (id_tarjeta) REFERENCES
tarjeta(id_tarjeta),
    CONSTRAINT fk_autorizacion_cajero FOREIGN KEY (id_cajero) REFERENCES
cajero(id_cajero),
    CONSTRAINT fk_autorizacion_tipo FOREIGN KEY (id_tipo_transaccion)
```

```
REFERENCES tipo_transaccion(id_tipo_transaccion),
   CONSTRAINT fk_autorizacion_motivo FOREIGN KEY (id_codigo_motivo)
REFERENCES codigo_motivo(id_codigo_motivo)
);

CREATE TABLE movimiento (
   id_movimiento BIGINT PRIMARY KEY AUTO_INCREMENT,
   id_tarjeta BIGINT NOT NULL,
   codigo_autorizacion CHAR(8) NOT NULL,
   tipo_movimiento VARCHAR(50) NOT NULL,
   monto DECIMAL(15,2) NOT NULL,
   estado VARCHAR(20) DEFAULT 'PENDIENTE' CHECK (estado IN
('PENDIENTE', 'APROBADO', 'RECHAZADO')),
   fecha_movimiento DATETIME DEFAULT CURRENT_TIMESTAMP,
   CONSTRAINT fk_mov_credito_tarjeta FOREIGN KEY (id_tarjeta) REFERENCES
tarjeta(id_tarjeta),
   CONSTRAINT fk_mov_credito_aut FOREIGN KEY (codigo_autorizacion)
REFERENCES autorizacion(codigo_autorizacion)
);

--  TABLA AUDITORÍA CORREGIDA (sin DEFAULT USER())
CREATE TABLE auditoria (
   id_auditoria BIGINT PRIMARY KEY AUTO_INCREMENT,
   tabla_afectada VARCHAR(50) NOT NULL,
   accion VARCHAR(20) NOT NULL,
   id_registro_afectado BIGINT,
   numero_referencia VARCHAR(50),
   campo_modificado VARCHAR(50),
   valor_anterior TEXT,
   valor_nuevo TEXT,
   usuario VARCHAR(100) DEFAULT NULL,        -- CORREGIDO: NULL en lugar de
USER()
   ip_address VARCHAR(45) DEFAULT NULL,
   fecha_auditoria DATETIME DEFAULT CURRENT_TIMESTAMP,
   INDEX idx_fecha_accion (fecha_auditoria, accion),
   INDEX idx_numero_ref (numero_referencia),
   INDEX idx_tabla_fecha (tabla_afectada, fecha_auditoria)
);

-- TRIGGERS (capturan USER() dinámicamente)
DELIMITER $$
CREATE TRIGGER tr_tarjeta_update_auditoria
AFTER UPDATE ON tarjeta
FOR EACH ROW
BEGIN
   IF OLD.pin != NEW.pin THEN
      INSERT INTO auditoria (
         tabla_afectada, accion, numero_referencia,
         campo_modificado, valor_anterior, valor_nuevo,
         id_registro_afectado, usuario
```

```
        ) VALUES (
            'tarjeta', 'UPDATE', NEW.numero_tarjeta, 'pin',
            OLD.pin, NEW.pin, NEW.id_tarjeta, USER()
        );
    END IF;

    IF OLD.estado != NEW.estado THEN
        INSERT INTO auditoria (
            tabla_afectada, accion, numero_referencia,
            campo_modificado, valor_anterior, valor_nuevo, usuario
        ) VALUES (
            'tarjeta', 'UPDATE', NEW.numero_tarjeta, 'estado',
            OLD.estado, NEW.estado, USER()
        );
    END IF;
END$$

CREATE TRIGGER tr_cuenta_update_auditoria
AFTER UPDATE ON cuenta
FOR EACH ROW
BEGIN
    IF OLD.saldo_disponible != NEW.saldo_disponible OR
       OLD.saldo_total != NEW.saldo_total OR
       OLD.disponible_credito != NEW.disponible_credito THEN
        INSERT INTO auditoria (
            tabla_afectada, accion, numero_referencia, campo_modificado,
            valor_anterior, valor_nuevo, id_registro_afectado, usuario
        ) VALUES (
            'cuenta', 'UPDATE', NEW.numero_cuenta, 'saldos',
            CONCAT('disp:', OLD.saldo_disponible, '|total:', OLD.saldo_total, '|cred:',
OLD.disponible_credito),
            CONCAT('disp:', NEW.saldo_disponible, '|total:', NEW.saldo_total, '|cred:',
NEW.disponible_credito),
            NEW.id_cuenta, USER()
        );
    END IF;
END$$
DELIMITER ;

-- ÍNDICES
CREATE INDEX idx_cajero_codigo ON cajero(codigo_cajero);
CREATE INDEX idx_autorizacion_codigo ON autorizacion(codigo_autorizacion);
CREATE INDEX idx_autorizacion_fecha ON autorizacion(fecha_solicitud);
CREATE INDEX idx_cuenta_tipo ON cuenta(tipo_cuenta, estado);
CREATE INDEX idx_tarjeta_tipo ON tarjeta(tipo_tarjeta, estado);

--
================================================================
=====
--                              ALTER TABLES
```

```
--
==========================================================================
=====

-- ALTER TABLE tarjeta
--       MODIFY numero_tarjeta VARBINARY (32), MODIFY pin VARBINARY (32),
MODIFY cvv VARBINARY (32);
```

Script SQLServer

```
-- ====================================================

-- CORE BANCARIO - SQL SERVER

 -- ====================================================

USE master; GO

-- creo la Base de Datos si No Existe

IF NOT EXISTS ( SELECT name FROM SYS.DATABASES WHERE NAME =
'core_bancario') BEGIN CREATE DATABASE core_bancario; PRINT '+ BASE DE
DATOS CORE_BANCARIO CREADA'; END ELSE PRINT ' ++ BASE DE DATOS
YA EXISTE'; GO

USE core_bancario; GO

-- ====================================================

-- TABLAS

-- ====================================================

-- TABLA CLIENTE

IF NOT EXISTS (SELECT * FROM SYSOBJECTS WHERE name = 'cliente' AND
xtype = 'U') BEGIN CREATE TABLE cliente ( id_cliente BIGINT PRIMARY KEY
IDENTITY(1,1), nombre NVARCHAR (100) NOT NULL, identificacion CHAR (20)
UNIQUE NOT NULL, estado NVARCHAR(10) DEFAULT 'ACTIVO' CHECK (estado
IN ('ACTIVO','INACTIVO')), fecha_registro DATETIME2 DEFAULT SYSDATETIME()
); PRINT '+ TABLA CLIENTE CREADA'; END GO



-- TABLA CUENTA
```

IF NOT EXISTS (SELECT * FROM SYSOBJECTS WHERE name = 'cuenta' AND xtype = 'U') BEGIN CREATE TABLE cuenta (id_cuenta BIGINT PRIMARY KEY IDENTITY (1,1), numero_cuenta CHAR(25) UNIQUE NOT NULL, id_cliente BIGINT NOT NULL, saldo_disponible DECIMAL(15,2) DEFAULT 0 NOT NULL, estado NVARCHAR(10) DEFAULT 'ACTIVA' CHECK (estado IN ('ACTIVA', 'INACTIVA', 'BLOQUEADA')) NOT NULL, fecha_apertura DATETIME2 DEFAULT SYSDATETIME()); PRINT '+ TABLA CUENTA CREADA'; END GO

-- TABLA TARJETA_CUENTA

IF NOT EXISTS (SELECT * FROM SYSOBJECTS WHERE name = 'tarjeta_cuenta' AND xtype = 'U') BEGIN CREATE TABLE tarjeta_cuenta (id_tarjeta_cuenta BIGINT PRIMARY KEY IDENTITY (1,1), numero_tarjeta CHAR(25) NOT NULL, id_cuenta BIGINT NOT NULL, fecha_asignacion DATETIME2 DEFAULT SYSDATETIME(), estado NVARCHAR(10) DEFAULT 'ACTIVA' CHECK (estado IN ('ACTIVA', 'INACTIVA', 'VENCIDA'))); PRINT '+ TABLA TARJETA_CUENTA CREADA'; END GO

-- TABLA MOVIMIENTO_CUENTA

IF NOT EXISTS (SELECT * FROM SYSOBJECTS WHERE name = 'movimiento_tarjeta' AND xtype = 'U') BEGIN CREATE TABLE movimiento_tarjeta (id_movimiento BIGINT PRIMARY KEY IDENTITY(1,1), id_cuenta BIGINT NOT NULL, numero_tarjeta CHAR(20), codigo_autorizacion CHAR(8), tipo_movimiento NVARCHAR(30) NOT NULL, monto DECIMAL(15,2) NOT NULL, saldo_anterior DECIMAL(15,2) NOT NULL, saldo_nuevo DECIMAL(15,2) NOT NULL, descripcion NVARCHAR(255), fecha_movimiento DATETIME2 DEFAULT SYSDATETIME(), estado NVARCHAR(20) DEFAULT 'PROCESADO' CHECK (estado IN ('PENDIENTE','PROCESADO','RECHAZADO'))); PRINT '+ TABLA MOVIMIENTO_TARJETA CREADA'; END GO

-- ==

-- TABLAS

-- ==

USE core_bancario;

-- FK CUENTA -> CLIENTE IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE name = 'FK_cuenta_cliente') BEGIN ALTER TABLE cuenta ADD CONSTRAINT FK_cuenta_cliente FOREIGN KEY (id_cliente) REFERENCES cliente(id_cliente); PRINT '+ FK CUENTA_CLIENTE CREADA'; END GO

-- KF TARJETA -> CUENTA IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE name = 'FK_tarjeta_cuenta') BEGIN ALTER TABLE tarjeta_cuenta ADD

CONSTRAINT FK_tarjeta_cuenta FOREIGN KEY (id_cuenta) REFERENCES cuenta(id_cuenta); PRINT '+ FK TARJETA_CUENTA CREADA'; END GO

-- FK MOVIMIENTO_TARJETA -> CUENTA IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE name = 'FK_movimiento_cuenta') BEGIN ALTER TABLE movimiento_tarjeta ADD CONSTRAINT FK_movimiento_cuenta FOREIGN KEY (id_cuenta) REFERENCES cuenta(id_cuenta); PRINT '+ FK MOVIMIENTO_CUENTA CREADA'; END GO

-- ==========

-- INDICES

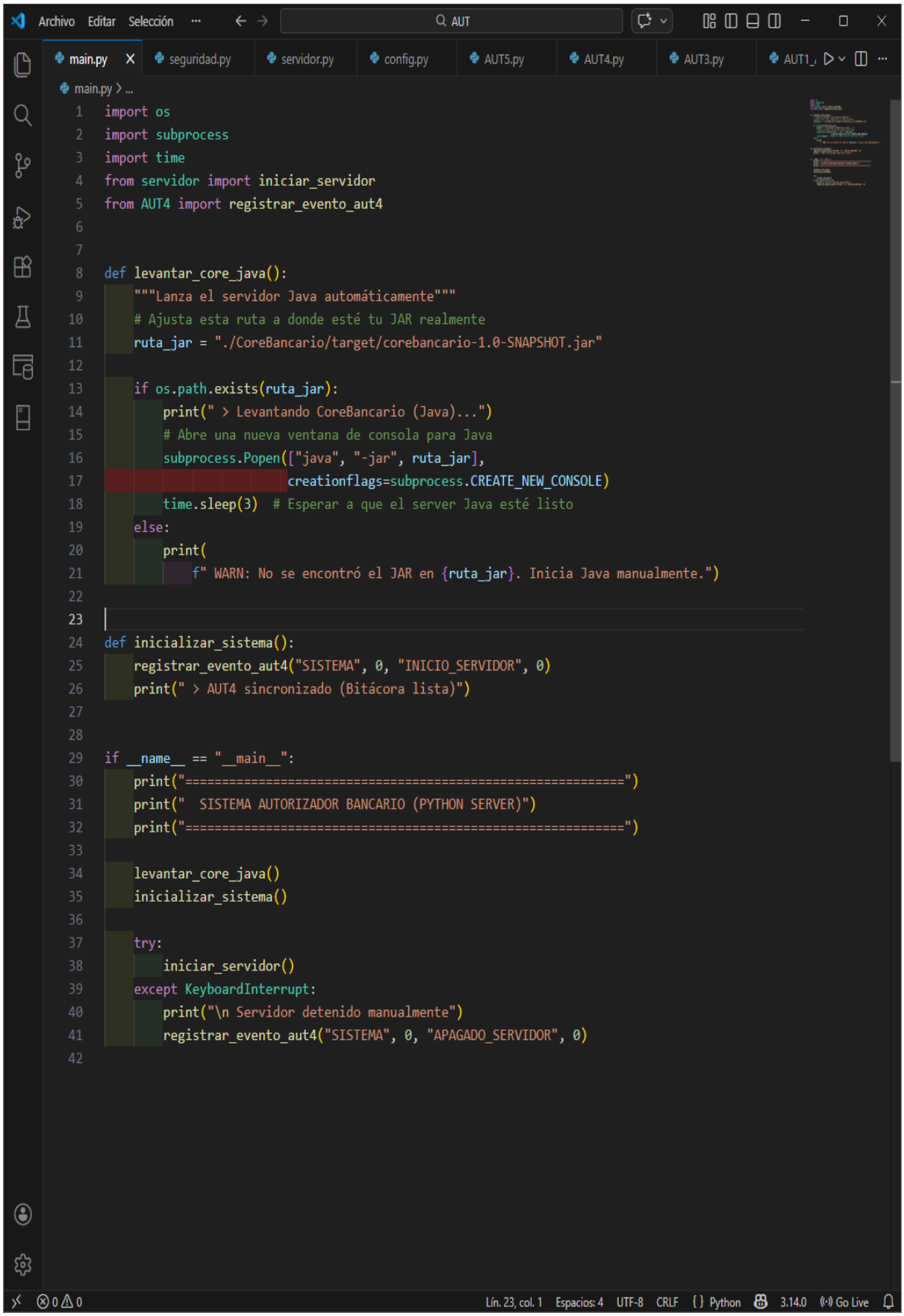
-- ==========

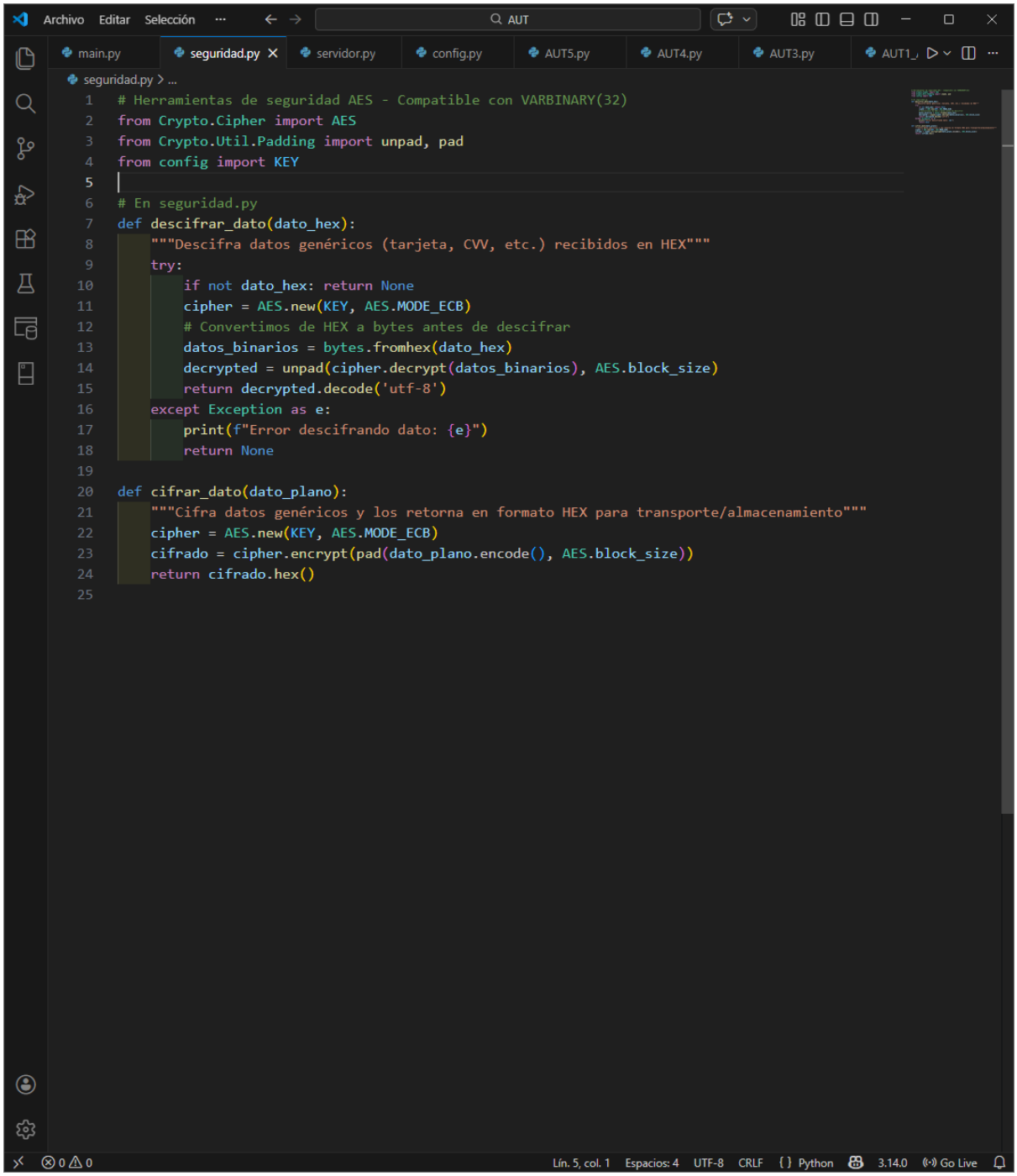
IF NOT EXISTS (SELECT * FROM sys.indexes WHERE name = 'IX_cuenta_numero') CREATE INDEX IX_cuenta_numero ON cuenta(numero_cuenta);

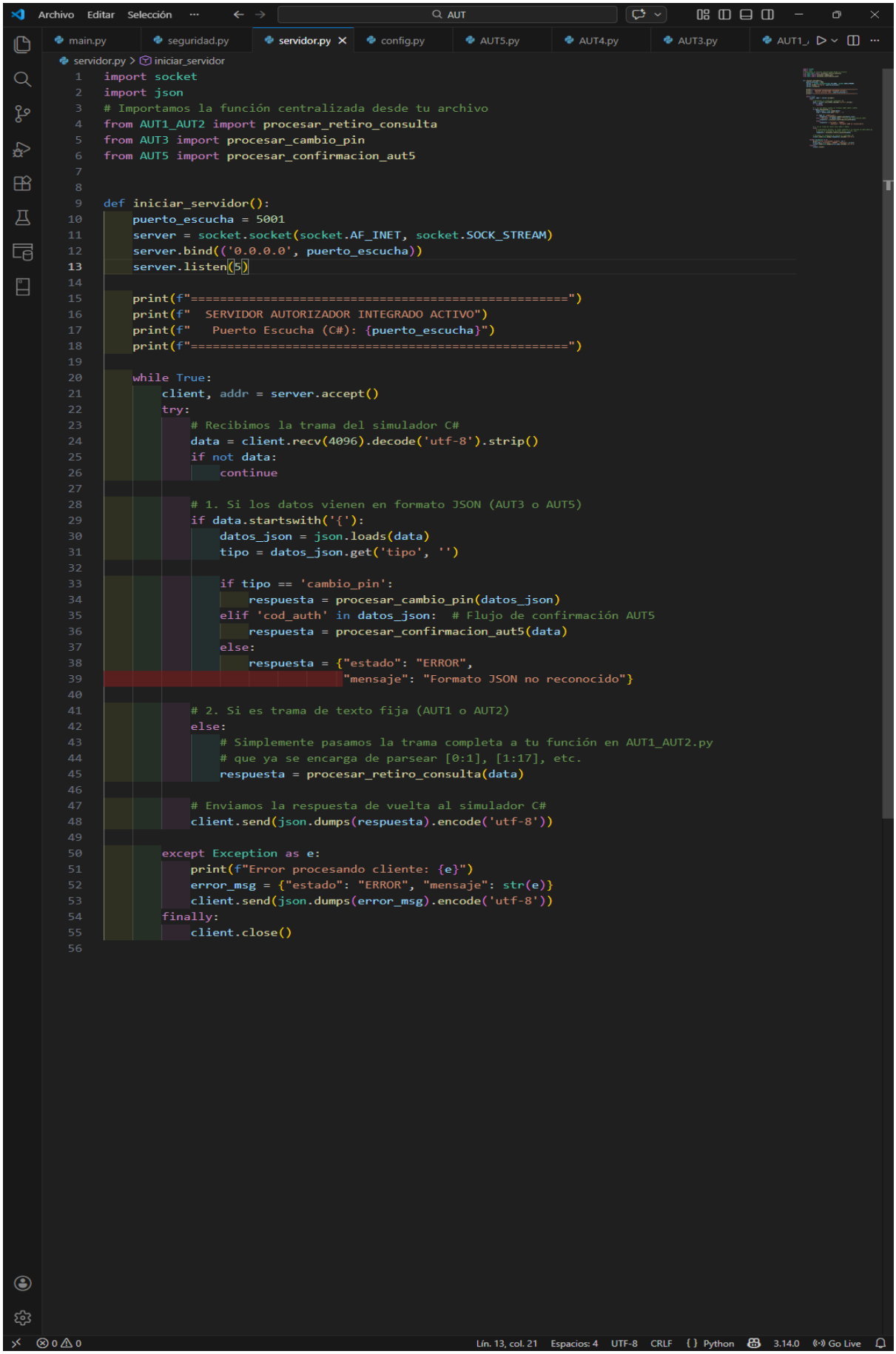
IF NOT EXISTS (SELECT * FROM sys.indexes WHERE name = 'IX_tarjeta_numero') CREATE INDEX IX_tarjeta_numero ON tarjeta_cuenta(numero_tarjeta);

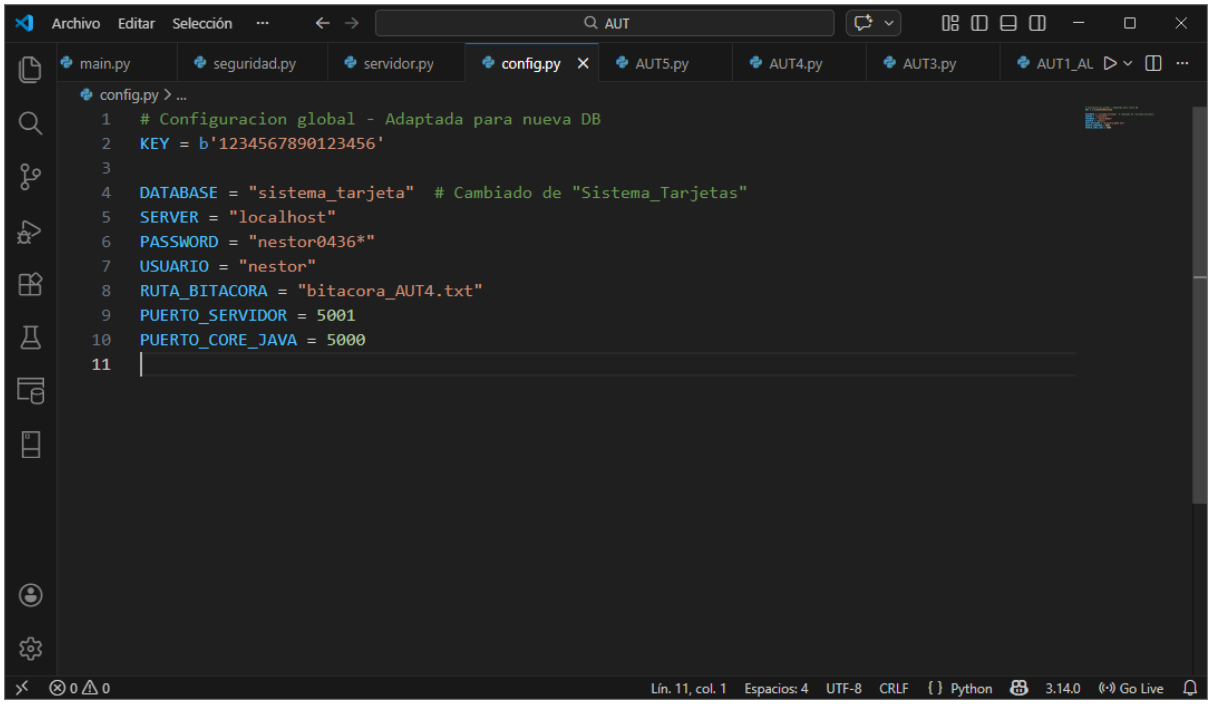
IF NOT EXISTS (SELECT * FROM sys.indexes WHERE name = 'IX_movimiento_fecha') CREATE INDEX IX_movimiento_fecha ON movimiento_tarjeta(fecha_movimiento); GO

Capitulo 3: Autorizador de Transacciones



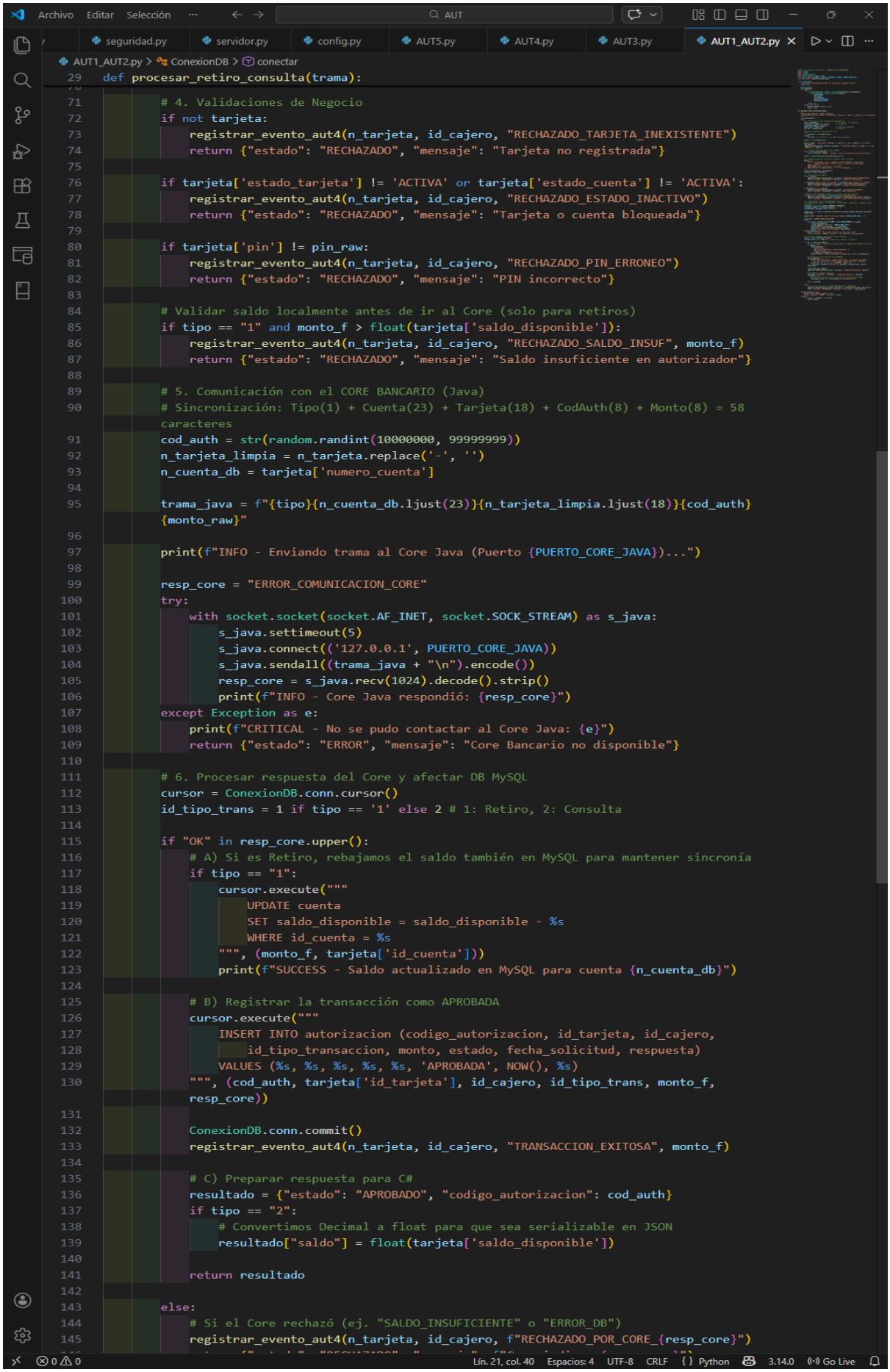

Archivo Editar Selección
AUT
main.py
seguridad.py
servidor.py
config.py
AUT5.py
AUT4.py
AUT3.py
seguridad.py > ...
# Herramientas de seguridad AES - Compatible con VARBINARY(32)
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad, pad
from config import KEY
# En seguridad.py
def descifrar_dato(dato_hex):
"""Descifra datos genéricos (tarjeta, CVV, etc.) recibidos en HEX"""
try:
if not dato_hex: return None
cipher = AES.new(KEY, AES.MODE_ECB)
# Convertimos de HEX a bytes antes de descifrar
datos_binarios = bytes.fromhex(dato_hex)
decrypted = unpad(cipher.decrypt(datos_binarios), AES.block_size)
return decrypted.decode('utf-8')
except Exception as e:
print(f"Error descifrando dato: {e}")
return None
def cifrar_dato(dato_plano):
"""Cifra datos genéricos y los retorna en formato HEX para transporte/almacenamiento"""
cipher = AES.new(KEY, AES.MODE_ECB)
cifrado = cipher.encrypt(pad(dato_plano.encode(), AES.block_size))
return cifrado.hex()
Lín. 5, col. 1
Espacios: 4
UTF-8
CRLF
Python
3.14.0
Go Live

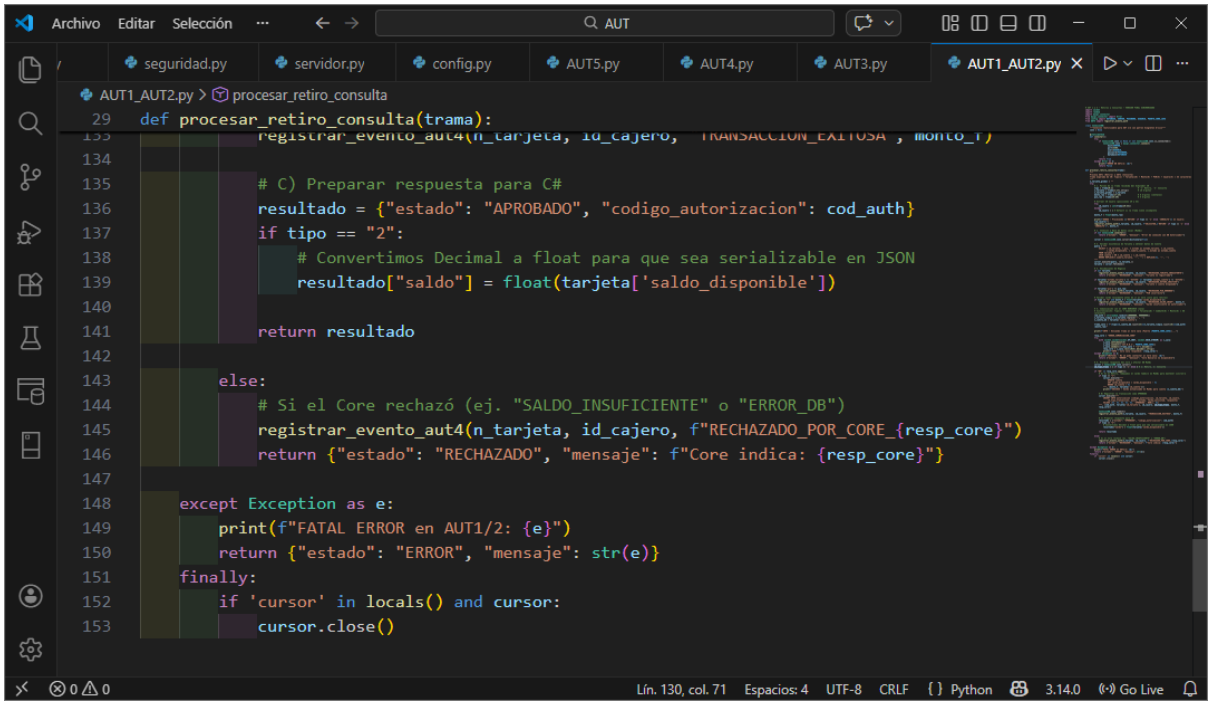

Archivo Editar Selección
AUT
main.py
seguridad.py
servidor.py
config.py
AUT5.py
AUT4.py
AUT3.py
servidor.py > iniciar_servidor
import socket
import json
# Importamos la función centralizada desde tu archivo
from AUT1_AUT2 import procesar_retiro_consulta
from AUT3 import procesar_cambio_pin
from AUT5 import procesar_confirmacion_aut5
def iniciar_servidor():
puerto_escucha = 5001
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(('0.0.0.0', puerto_escucha))
server.listen(5)
print(f"==========")
print(f" SERVIDOR AUTORIZADOR INTEGRADO ACTIVO")
print(f" Puerto Escucha (C#): {puerto_escucha}")
print(f"==========")
while True:
client, addr = server.accept()
try:
# Recibimos la trama del simulador C#
data = client.recv(4096).decode('utf-8').strip()
if not data:
continue
# 1. Si los datos vienen en formato JSON (AUT3 o AUT5)
if data.startswith('{'):
datos_json = json.loads(data)
tipo = datos_json.get('tipo', '')
if tipo == 'cambio_pin':
respuesta = procesar_cambio_pin(datos_json)
elif 'cod_auth' in datos_json: # Flujo de confirmación AUT5
respuesta = procesar_confirmacion_aut5(data)
else:
respuesta = {"estado": "ERROR",
"mensaje": "Formato JSON no reconocido"}
# 2. Si es trama de texto fija (AUT1 o AUT2)
else:
# Simplemente pasamos la trama completa a tu función en AUT1_AUT2.py
# que ya se encarga de parsear [0:1], [1:17], etc.
respuesta = procesar_retiro_consulta(data)
# Enviamos la respuesta de vuelta al simulador C#
client.send(json.dumps(respuesta).encode('utf-8'))
except Exception as e:
print(f"Error procesando cliente: {e}")
error_msg = {"estado": "ERROR", "mensaje": str(e)}
client.send(json.dumps(error_msg).encode('utf-8'))
finally:
client.close()
Lín. 13, col. 21 Espacios: 4 UTF-8 CRLF Python 3.14.0 Go Live


Archivo Editar Selección
AUT
main.py
seguridad.py
servidor.py
config.py
AUT5.py
AUT4.py
AUT3.py
AUT1_AL
config.py > ...
1 # Configuracion global - Adaptada para nueva DB
2 KEY = b'1234567890123456'
3
4 DATABASE = "sistema_tarjeta" # Cambiado de "Sistema_Tarjetas"
5 SERVER = "localhost"
6 PASSWORD = "nestor0436*"
7 USUARIO = "nestor"
8 RUTA_BITACORA = "bitacora_AUT4.txt"
9 PUERTO_SERVIDOR = 5001
10 PUERTO_CORE_JAVA = 5000
11
0 0
Lín. 11, col. 1
Espacios: 4
UTF-8
CRLF
Python
3.14.0
Go Live

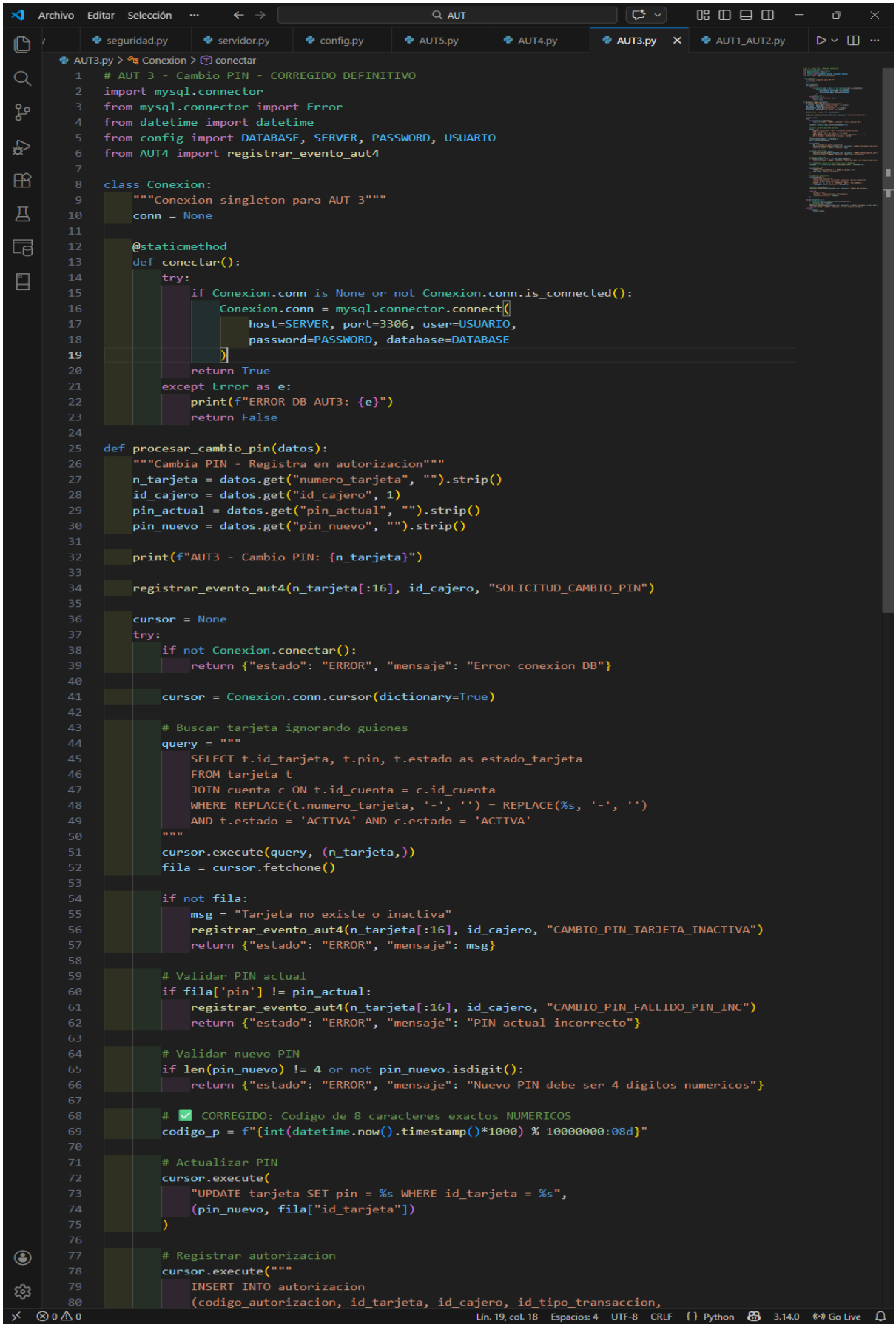
AUT 1 & 2

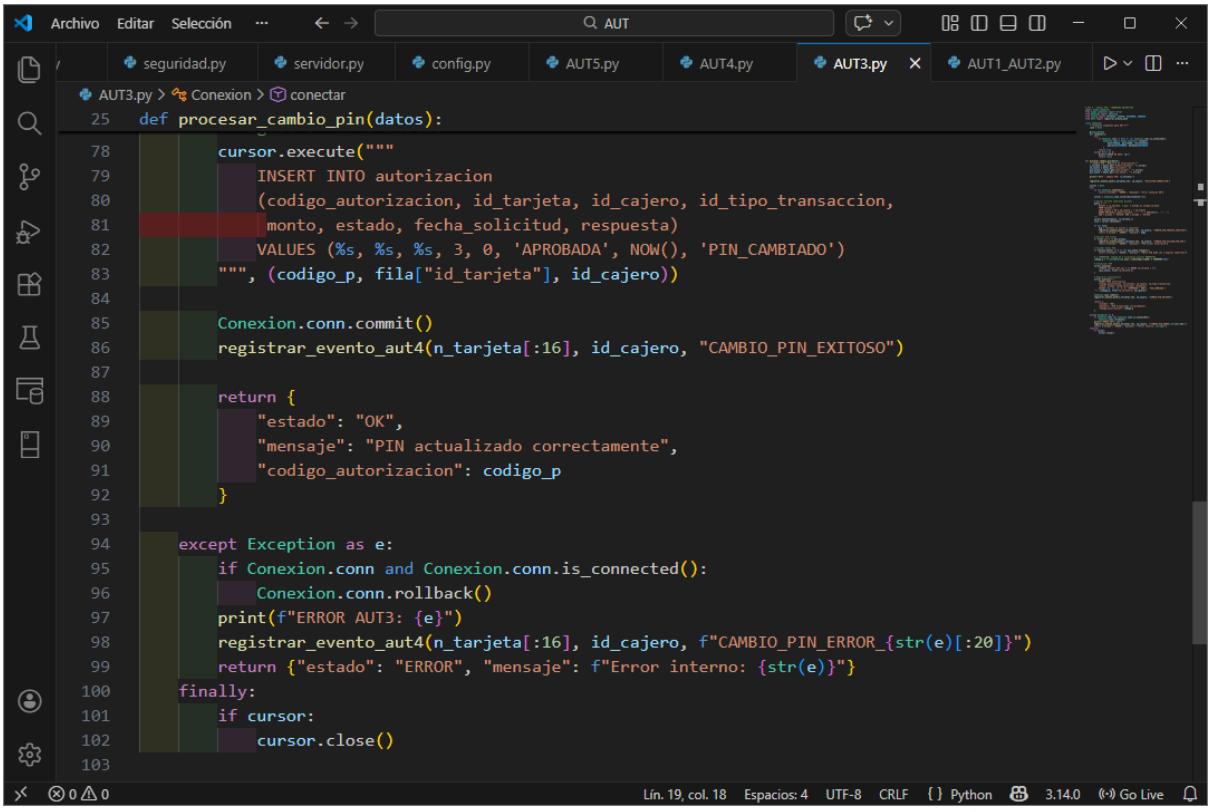

Archivo Editar Selección
AUT
seguridad.py
servidor.py
config.py
AUT5.py
AUT4.py
AUT3.py
AUT1_AUT2.py
AUT1_AUT2.py > ConexionDB > conectar
# AUT 1 y 2 - Retiros y Consultas - VERSION FINAL SINCRONIZADA
import socket
import random
import mysql.connector
from mysql.connector import Error
from config import DATABASE, SERVER, PASSWORD, USUARIO, PUERTO_CORE_JAVA
from AUT4 import registrar_evento_aut4
class ConexionDB:
"""Conexion reutilizable para AUT 1/2 con patron Singleton básico"""
conn = None
@staticmethod
def conectar():
try:
if ConexionDB.conn is None or not ConexionDB.conn.is_connected():
ConexionDB.conn = mysql.connector.connect(
host=SERVER,
port=3306,
user=USUARIO,
password=PASSWORD,
database=DATABASE
)
return True
except Error as e:
print(f"ERROR DB AUT1/2: {e}")
return False
def procesar_retiro_consulta(trama):
"""
Procesa AUT1 (Retiro) y AUT2 (Consulta)
Trama esperada de C#: Tipo(1) + Tarjeta(16) + Monto(8) + PIN(4) + Cajero(4) = 33 caracteres
"""
n_tarjeta_global = ""
try:
# 1. Parseo de la trama recibida del Simulador C#
tipo = trama[0:1] # '1' Retiro, '2' Consulta
n_tarjeta = trama[1:17].strip() # 16 dígitos
n_tarjeta_global = n_tarjeta
monto_raw = trama[17:25] # 8 dígitos (centavos)
pin_raw = trama[25:29] # 4 dígitos
# Extraer ID Cajero (posiciones 29 a 33)
try:
id_cajero = int(trama[29:33])
except:
id_cajero = 1 # Default si la trama viene incompleta
monto_f = float(monto_raw)
print(f"DEBUG - Procesando {('RETIRO' if tipo == '1' else 'CONSULTA')} en Cajero: {id_cajero}")
registrar_evento_aut4(n_tarjeta, id_cajero, f"SOLICITUD_{'RETIRO' if tipo == '1' else 'CONSULTA'}", monto_f)
# 2. Conexion a Base de Datos Local (MySQL)
if not ConexionDB.conectar():
return {"estado": "ERROR", "mensaje": "Error de conexión con DB Autorizador"}
cursor = ConexionDB.conn.cursor(dictionary=True)
# 3. Validar existencia de Tarjeta y obtener datos de Cuenta
query = """
SELECT t.id_tarjeta, t.pin, t.estado as estado_tarjeta, t.id_cuenta,
c.saldo_disponible, c.numero_cuenta, c.estado as estado_cuenta
FROM tarjeta t
JOIN cuenta c ON t.id_cuenta = c.id_cuenta
WHERE REPLACE(t.numero_tarjeta, '-', '') = REPLACE(%s, '-', '')
"""
cursor.execute(query, (n_tarjeta,))
tarjeta = cursor.fetchone()
# 4. Validaciones de Negocio
if not tarjeta:
registrar_evento_aut4(n_tarjeta, id_cajero, "RECHAZADO_TARJETA_INEXISTENTE")
return {"estado": "RECHAZADO", "mensaje": "Tarjeta no registrada"}
if tarjeta['estado_tarjeta'] != 'ACTIVA' or tarjeta['estado_cuenta'] != 'ACTIVA':
registrar_evento_aut4(n_tarjeta, id_cajero, "RECHAZADO_ESTADO_INACTIVO")
return {"estado": "RECHAZADO", "mensaje": "Tarjeta o cuenta bloqueada"}
Lín. 21, col. 40 Espacios: 4 UTF-8 CRLF { } Python 3.14.0 Go Live


Archivo Editar Selección
AUT
seguridad.py
servidor.py
config.py
AUT5.py
AUT4.py
AUT3.py
AUT1_AUT2.py
AUT1_AUT2.py > ConexionDB > conectar
def procesar_retiro_consulta(trama):
# 4. Validaciones de Negocio
if not tarjeta:
registrar_evento_aut4(n_tarjeta, id_cajero, "RECHAZADO_TARJETA_INEXISTENTE")
return {"estado": "RECHAZADO", "mensaje": "Tarjeta no registrada"}
if tarjeta['estado_tarjeta'] != 'ACTIVA' or tarjeta['estado_cuenta'] != 'ACTIVA':
registrar_evento_aut4(n_tarjeta, id_cajero, "RECHAZADO_ESTADO_INACTIVO")
return {"estado": "RECHAZADO", "mensaje": "Tarjeta o cuenta bloqueada"}
if tarjeta['pin'] != pin_raw:
registrar_evento_aut4(n_tarjeta, id_cajero, "RECHAZADO_PIN_ERRONEO")
return {"estado": "RECHAZADO", "mensaje": "PIN incorrecto"}
# Validar saldo localmente antes de ir al Core (solo para retiros)
if tipo == "1" and monto_f > float(tarjeta['saldo_disponible']):
registrar_evento_aut4(n_tarjeta, id_cajero, "RECHAZADO_SALDO_INSUF", monto_f)
return {"estado": "RECHAZADO", "mensaje": "Saldo insuficiente en autorizador"}
5. Comunicación con el CORE BANCARIO (Java)
Sincronización: Tipo(1) + Cuenta(23) + Tarjeta(18) + CodAuth(8) + Monto(8) = 58
caracteres
cod_auth = str(random.randint(10000000, 99999999))
n_tarjeta_limpia = n_tarjeta.replace('-', '')
n_cuenta_db = tarjeta['numero_cuenta']
trama_java = f"{tipo}{n_cuenta_db.ljust(23)}{n_tarjeta_limpia.ljust(18)}{cod_auth}
{monto_raw}"
print(f"INFO - Enviando trama al Core Java (Puerto {PUERTO_CORE_JAVA})...")
resp_core = "ERROR_COMUNICACION_CORE"
try:
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s_java:
s_java.settimeout(5)
s_java.connect(('127.0.0.1', PUERTO_CORE_JAVA))
s_java.sendall((trama_java + "\n").encode())
resp_core = s_java.recv(1024).decode().strip()
print(f"INFO - Core Java respondió: {resp_core}")
except Exception as e:
print(f"CRITICAL - No se pudo contactar al Core Java: {e}")
return {"estado": "ERROR", "mensaje": "Core Bancario no disponible"}
# 6. Procesar respuesta del Core y afectar DB MySQL
cursor = ConexionDB.conn.cursor()
id_tipo_trans = 1 if tipo == '1' else 2 # 1: Retiro, 2: Consulta
if "OK" in resp_core.upper():
# A) Si es Retiro, rebajamos el saldo también en MySQL para mantener sincronía
if tipo == "1":
cursor.execute("""
UPDATE cuenta
SET saldo_disponible = saldo_disponible - %s
WHERE id_cuenta = %s
""", (monto_f, tarjeta['id_cuenta']))
print(f"SUCCESS - Saldo actualizado en MySQL para cuenta {n_cuenta_db}")
# B) Registrar la transacción como APROBADA
cursor.execute("""
INSERT INTO autorizacion (codigo_autorizacion, id_tarjeta, id_cajero,
id_tipo_transaccion, monto, estado, fecha_solicitud, respuesta)
VALUES (%s, %s, %s, %s, %s, 'APROBADA', NOW(), %s)
""", (cod_auth, tarjeta['id_tarjeta'], id_cajero, id_tipo_trans, monto_f,
resp_core))
ConexionDB.conn.commit()
registrar_evento_aut4(n_tarjeta, id_cajero, "TRANSACCION_EXITOSA", monto_f)
# C) Preparar respuesta para C#
resultado = {"estado": "APROBADO", "codigo_autorizacion": cod_auth}
if tipo == "2":
# Convertimos Decimal a float para que sea serializable en JSON
resultado["saldo"] = float(tarjeta['saldo_disponible'])
return resultado
else:
# Si el Core rechazó (ej. "SALDO_INSUFICIENTE" o "ERROR_DB")
registrar_evento_aut4(n_tarjeta, id_cajero, f"RECHAZADO_POR_CORE_{resp_core}")
Lín. 21, col. 40 Espacios: 4 UTF-8 CRLF Python 3.14.0 Go Live

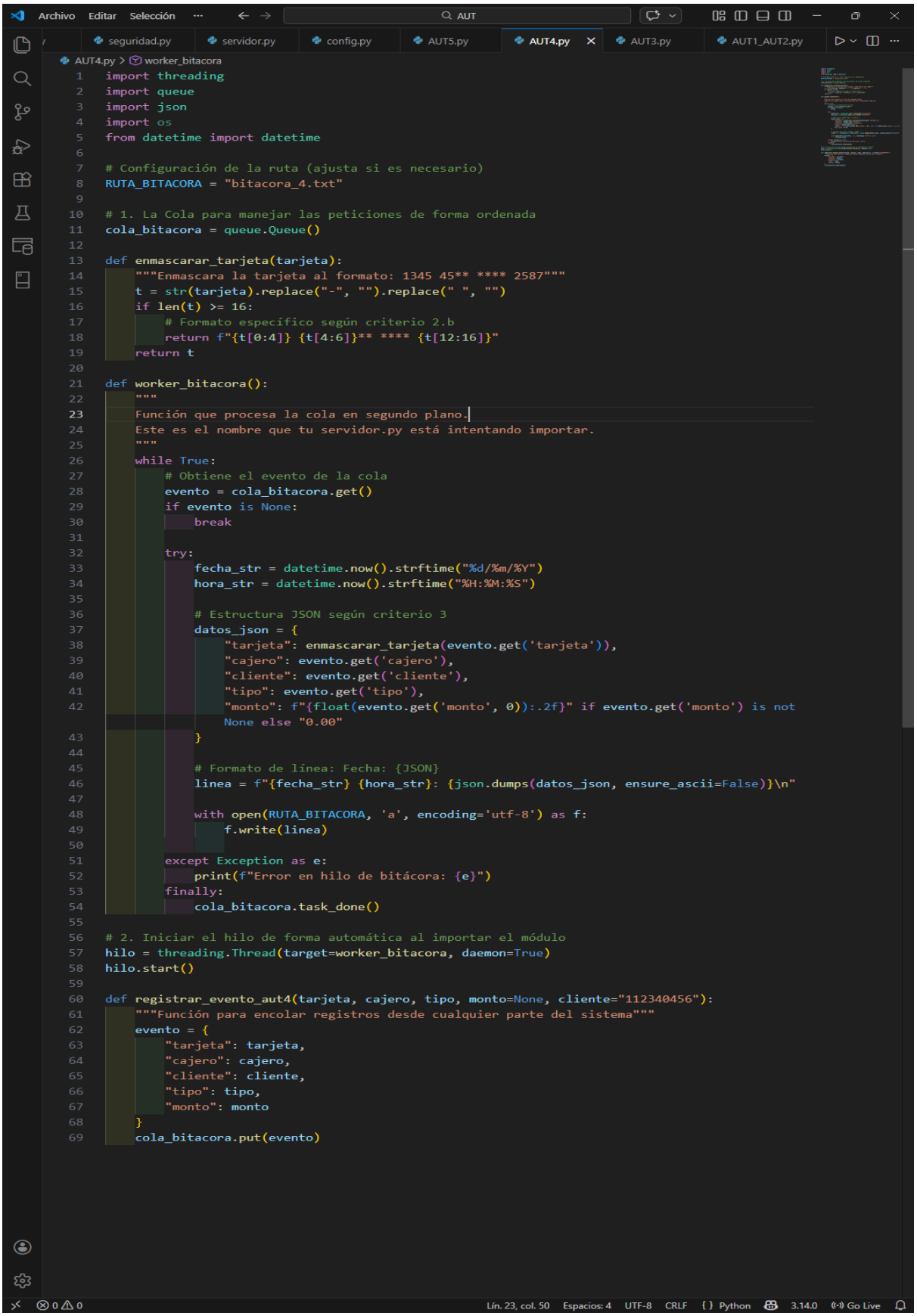

Archivo Editar Selección
AUT
seguridad.py
servidor.py
config.py
AUT5.py
AUT4.py
AUT3.py
AUT1_AUT2.py
AUT1_AUT2.py > procesar_retiro_consulta
def procesar_retiro_consulta(trama):
registrar_evento_aut4(n_tarjeta, id_cajero, "TRANSACCION_EXITOSA", monto_f)
# C) Preparar respuesta para C#
resultado = {"estado": "APROBADO", "codigo_autorizacion": cod_auth}
if tipo == "2":
# Convertimos Decimal a float para que sea serializable en JSON
resultado["saldo"] = float(tarjeta['saldo_disponible'])
return resultado
else:
# Si el Core rechazó (ej. "SALDO_INSUFICIENTE" o "ERROR_DB")
registrar_evento_aut4(n_tarjeta, id_cajero, f"RECHAZADO_POR_CORE_{resp_core}")
return {"estado": "RECHAZADO", "mensaje": f"Core indica: {resp_core}"}
except Exception as e:
print(f"FATAL ERROR en AUT1/2: {e}")
return {"estado": "ERROR", "mensaje": str(e)}
finally:
if 'cursor' in locals() and cursor:
cursor.close()
Lín. 130, col. 71
Espacios: 4
UTF-8
CRLF
Python
3.14.0
Go Live

AUT 3

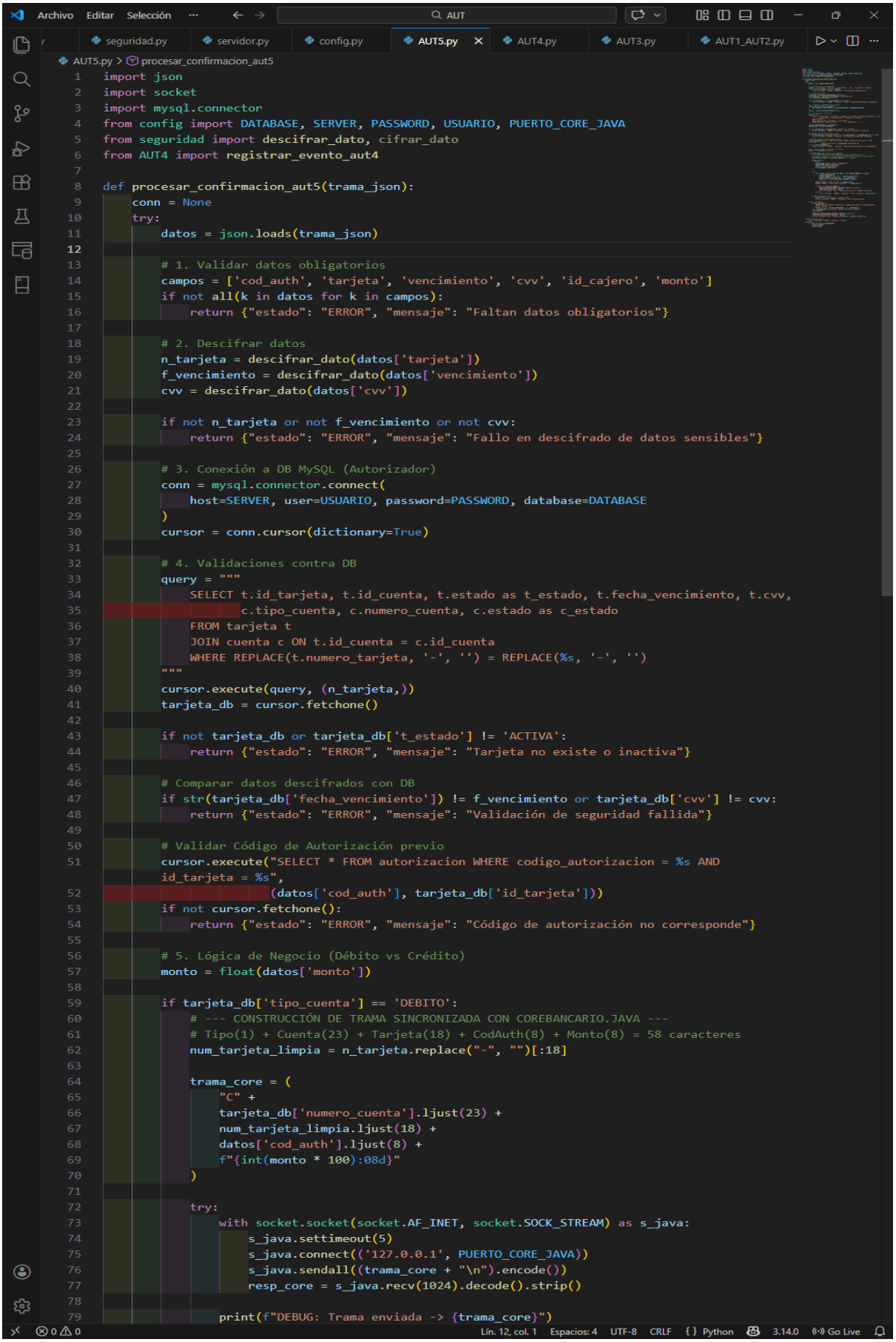

Archivo Editar Selección
AUT
seguridad.py
servidor.py
config.py
AUT5.py
AUT4.py
AUT3.py
AUT1_AUT2.py
AUT3.py > Conexion > conectar
# AUT 3 - Cambio PIN - CORREGIDO DEFINITIVO
import mysql.connector
from mysql.connector import Error
from datetime import datetime
from config import DATABASE, SERVER, PASSWORD, USUARIO
from AUT4 import registrar_evento_aut4
class Conexion:
"""Conexion singleton para AUT 3"""
conn = None
@staticmethod
def conectar():
try:
if Conexion.conn is None or not Conexion.conn.is_connected():
Conexion.conn = mysql.connector.connect(
host=SERVER, port=3306, user=USUARIO,
password=PASSWORD, database=DATABASE
)
return True
except Error as e:
print(f"ERROR DB AUT3: {e}")
return False
def procesar_cambio_pin(datos):
"""Cambia PIN - Registra en autorizacion"""
n_tarjeta = datos.get("numero_tarjeta", "").strip()
id_cajero = datos.get("id_cajero", 1)
pin_actual = datos.get("pin_actual", "").strip()
pin_nuevo = datos.get("pin_nuevo", "").strip()
print(f"AUT3 - Cambio PIN: {n_tarjeta}")
registrar_evento_aut4(n_tarjeta[:16], id_cajero, "SOLICITUD_CAMBIO_PIN")
cursor = None
try:
if not Conexion.conectar():
return {"estado": "ERROR", "mensaje": "Error conexion DB"}
cursor = Conexion.conn.cursor(dictionary=True)
# Buscar tarjeta ignorando guiones
query = """
SELECT t.id_tarjeta, t.pin, t.estado as estado_tarjeta
FROM tarjeta t
JOIN cuenta c ON t.id_cuenta = c.id_cuenta
WHERE REPLACE(t.numero_tarjeta, '-', '') = REPLACE(%s, '-', '')
AND t.estado = 'ACTIVA' AND c.estado = 'ACTIVA'
"""
cursor.execute(query, (n_tarjeta,))
fila = cursor.fetchone()
if not fila:
msg = "Tarjeta no existe o inactiva"
registrar_evento_aut4(n_tarjeta[:16], id_cajero, "CAMBIO_PIN_TARJETA_INACTIVA")
return {"estado": "ERROR", "mensaje": msg}
# Validar PIN actual
if fila['pin'] != pin_actual:
registrar_evento_aut4(n_tarjeta[:16], id_cajero, "CAMBIO_PIN_FALLIDO_PIN_INC")
return {"estado": "ERROR", "mensaje": "PIN actual incorrecto"}
# Validar nuevo PIN
if len(pin_nuevo) != 4 or not pin_nuevo.isdigit():
return {"estado": "ERROR", "mensaje": "Nuevo PIN debe ser 4 digitos numericos"}
# ✅ CORREGIDO: Codigo de 8 caracteres exactos NUMERICOS
codigo_p = f"{int(datetime.now().timestamp()*1000) % 10000000:08d}"
# Actualizar PIN
cursor.execute(
"UPDATE tarjeta SET pin = %s WHERE id_tarjeta = %s",
(pin_nuevo, fila["id_tarjeta"])
)
Registrar autorizacion
cursor.execute("""
INSERT INTO autorizacion
(codigo_autorizacion, id_tarjeta, id_cajero, id_tipo_transaccion,
Lín. 19, col. 18 Espacios: 4 UTF-8 CRLF Python 3.14.0 Go Live

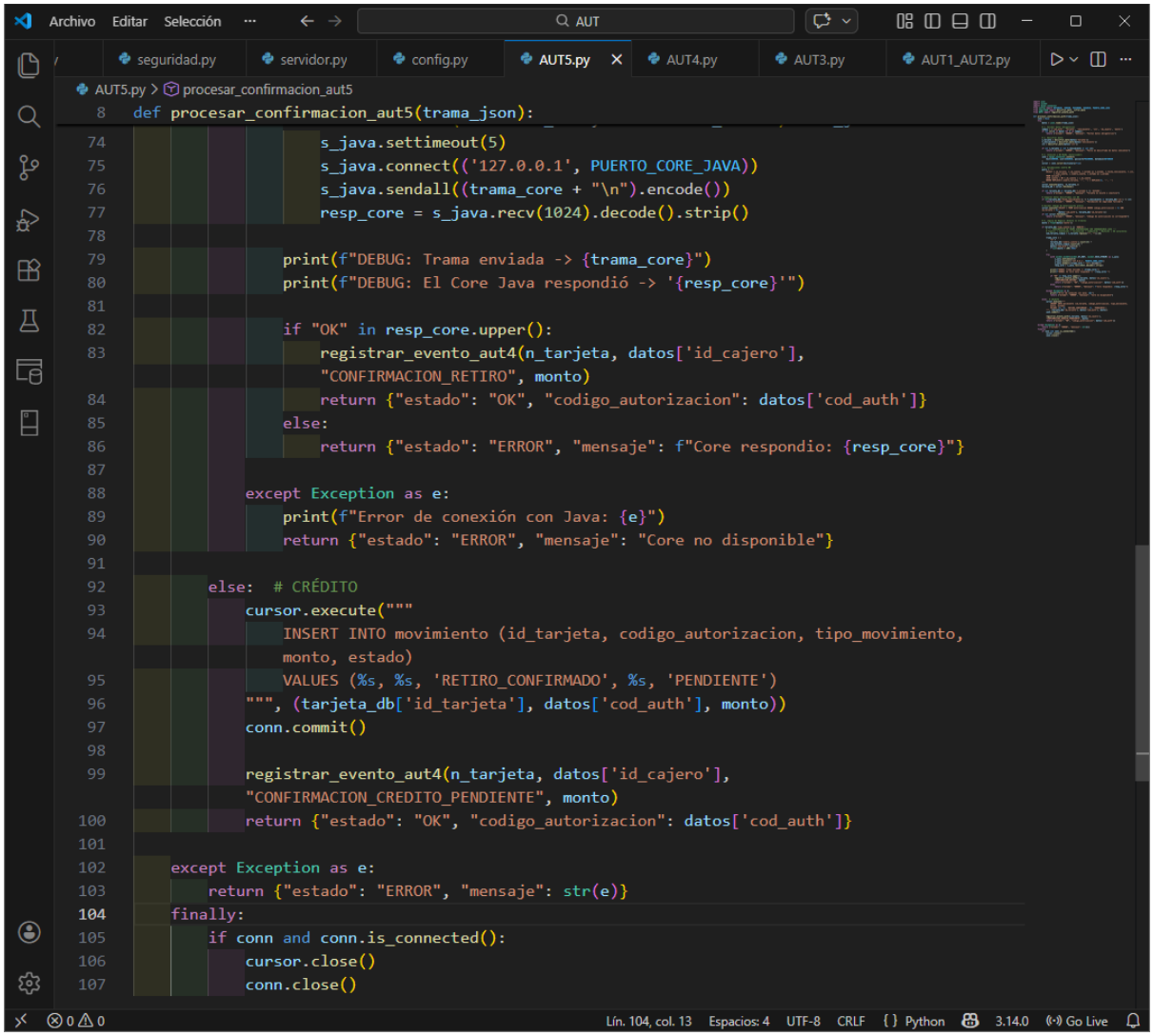

Archivo Editar Selección
AUT
seguridad.py
servidor.py
config.py
AUT5.py
AUT4.py
AUT3.py
AUT1_AUT2.py
AUT3.py > Conexion > conectar
def procesar_cambio_pin(datos):
cursor.execute("""
INSERT INTO autorizacion
(codigo_autorizacion, id_tarjeta, id_cajero, id_tipo_transaccion,
monto, estado, fecha_solicitud, respuesta)
VALUES (%s, %s, %s, 3, 0, 'APROBADA', NOW(), 'PIN_CAMBIADO')
""", (codigo_p, fila["id_tarjeta"], id_cajero))
Conexion.conn.commit()
registrar_evento_aut4(n_tarjeta[:16], id_cajero, "CAMBIO_PIN_EXITOSO")
return {
"estado": "OK",
"mensaje": "PIN actualizado correctamente",
"codigo_autorizacion": codigo_p
}
except Exception as e:
if Conexion.conn and Conexion.conn.is_connected():
Conexion.conn.rollback()
print(f"ERROR AUT3: {e}")
registrar_evento_aut4(n_tarjeta[:16], id_cajero, f"CAMBIO_PIN_ERROR_{str(e)[:20]}")
return {"estado": "ERROR", "mensaje": f"Error interno: {str(e)}"}
finally:
if cursor:
cursor.close()
Lín. 19, col. 18
Espacios: 4
UTF-8
CRLF
Python
3.14.0
Go Live

AUT 4

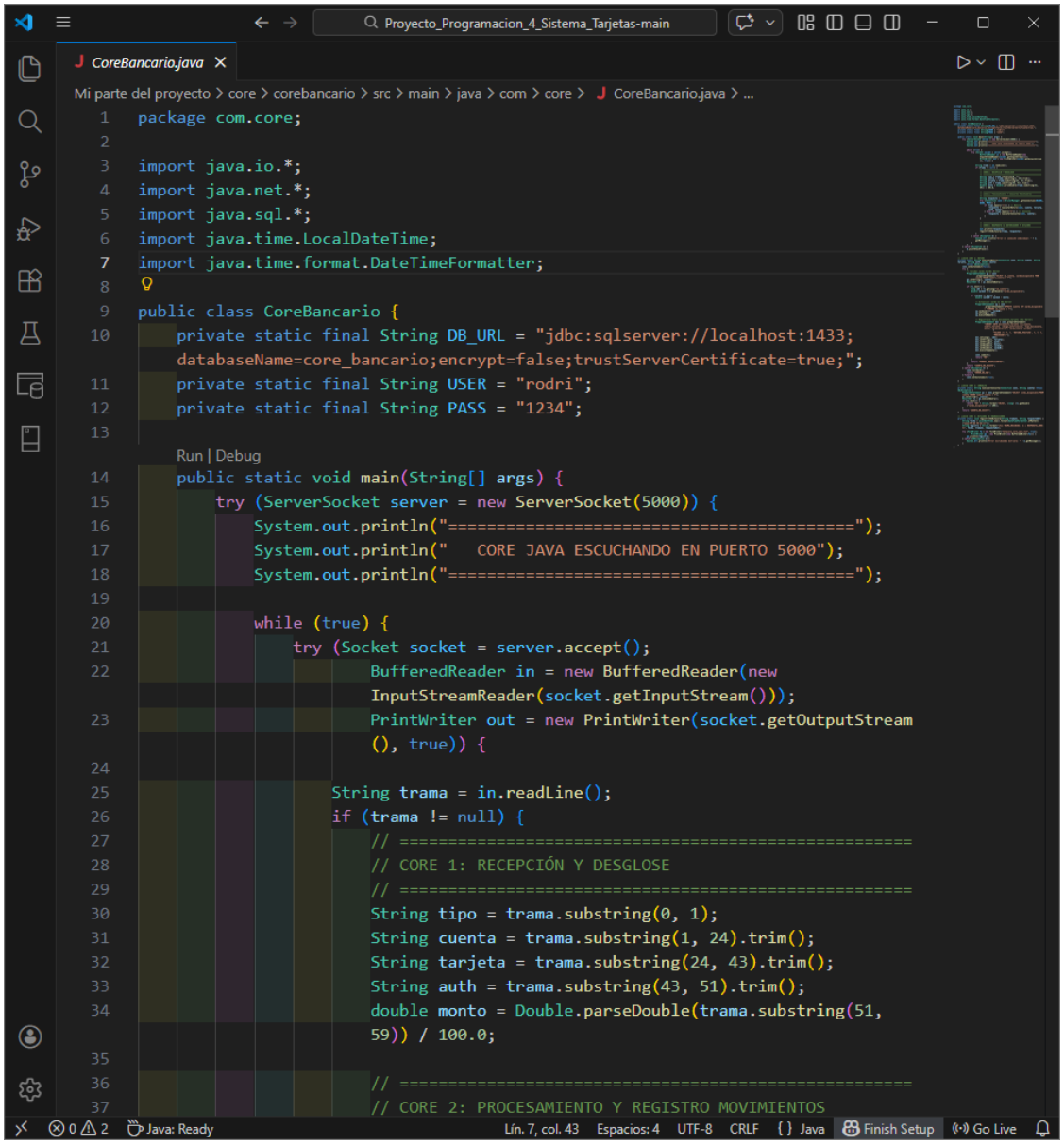

Archivo Editar Selección
AUT
seguridad.py
servidor.py
config.py
AUT5.py
AUT4.py
AUT3.py
AUT1_AUT2.py
AUT4.py > worker_bitacora
import threading
import queue
import json
import os
from datetime import datetime
# Configuración de la ruta (ajusta si es necesario)
RUTA_BITACORA = "bitacora_4.txt"
# 1. La Cola para manejar las peticiones de forma ordenada
cola_bitacora = queue.Queue()
def enmascarar_tarjeta(tarjeta):
"""Enmascara la tarjeta al formato: 1345 45** **** 2587"""
t = str(tarjeta).replace("-", "").replace(" ", "")
if len(t) >= 16:
# Formato específico según criterio 2.b
return f"{t[0:4]} {t[4:6]}** **** {t[12:16]}"
return t
def worker_bitacora():
"""
Función que procesa la cola en segundo plano.
Este es el nombre que tu servidor.py está intentando importar.
"""
while True:
# Obtiene el evento de la cola
evento = cola_bitacora.get()
if evento is None:
break
try:
fecha_str = datetime.now().strftime("%d/%m/%Y")
hora_str = datetime.now().strftime("%H:%M:%S")
# Estructura JSON según criterio 3
datos_json = {
"tarjeta": enmascarar_tarjeta(evento.get('tarjeta')),
"cajero": evento.get('cajero'),
"cliente": evento.get('cliente'),
"tipo": evento.get('tipo'),
"monto": f"{float(evento.get('monto', 0)):.2f}" if evento.get('monto') is not None else "0.00"
}
# Formato de línea: Fecha: {JSON}
linea = f"{fecha_str} {hora_str}: {json.dumps(datos_json, ensure_ascii=False)}\n"
with open(RUTA_BITACORA, 'a', encoding='utf-8') as f:
f.write(linea)
except Exception as e:
print(f"Error en hilo de bitácora: {e}")
finally:
cola_bitacora.task_done()
# 2. Iniciar el hilo de forma automática al importar el módulo
hilo = threading.Thread(target=worker_bitacora, daemon=True)
hilo.start()
def registrar_evento_aut4(tarjeta, cajero, tipo, monto=None, cliente="112340456"):
"""Función para encolar registros desde cualquier parte del sistema"""
evento = {
"tarjeta": tarjeta,
"cajero": cajero,
"cliente": cliente,
"tipo": tipo,
"monto": monto
}
cola_bitacora.put(evento)
Lín. 23, col. 50 Espacios: 4 UTF-8 CRLF Python 3.14.0 Go Live

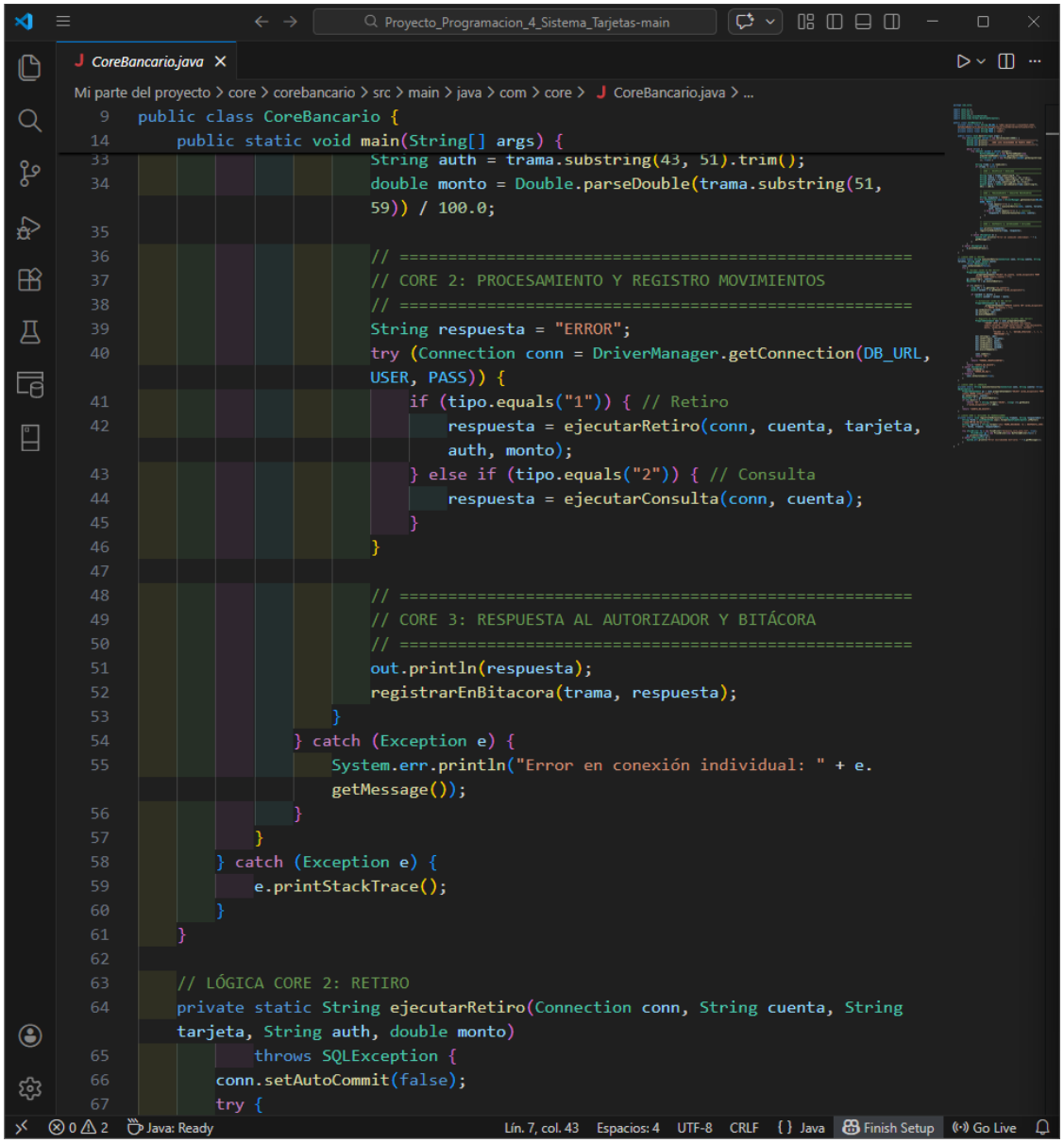
AUT 5

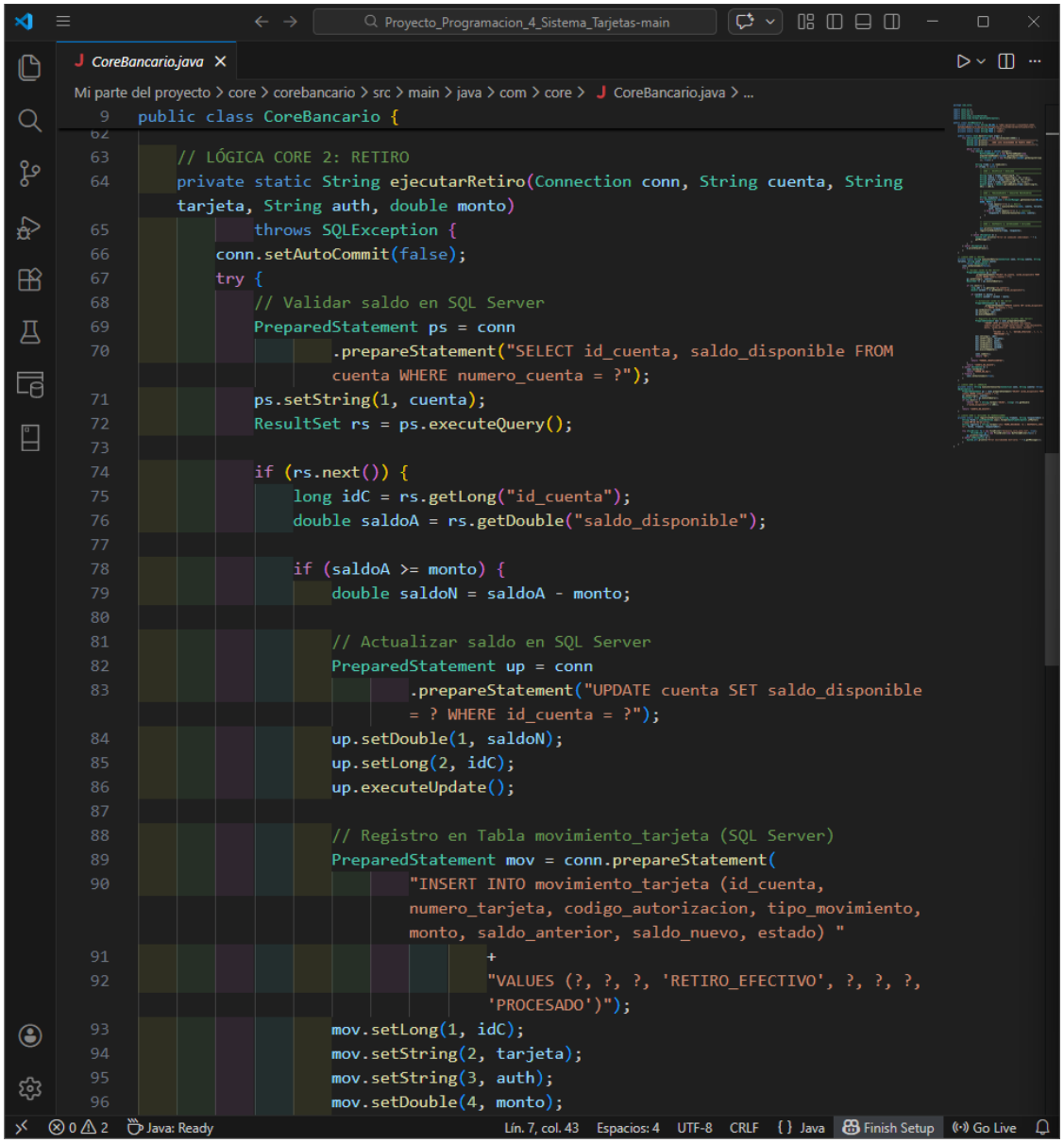

Archivo Editar Selección
AUT
seguridad.py
servidor.py
config.py
AUT5.py
AUT4.py
AUT3.py
AUT1_AUT2.py
AUT5.py > procesar_confirmacion_aut5
import json
import socket
import mysql.connector
from config import DATABASE, SERVER, PASSWORD, USUARIO, PUERTO_CORE_JAVA
from seguridad import descifrar_dato, cifrar_dato
from AUT4 import registrar_evento_aut4
def procesar_confirmacion_aut5(trama_json):
conn = None
try:
datos = json.loads(trama_json)
# 1. Validar datos obligatorios
campos = ['cod_auth', 'tarjeta', 'vencimiento', 'cvv', 'id_cajero', 'monto']
if not all(k in datos for k in campos):
return {"estado": "ERROR", "mensaje": "Faltan datos obligatorios"}
# 2. Descifrar datos
n_tarjeta = descifrar_dato(datos['tarjeta'])
f_vencimiento = descifrar_dato(datos['vencimiento'])
cvv = descifrar_dato(datos['cvv'])
if not n_tarjeta or not f_vencimiento or not cvv:
return {"estado": "ERROR", "mensaje": "Fallo en descifrado de datos sensibles"}
# 3. Conexión a DB MySQL (Autorizador)
conn = mysql.connector.connect(
host=SERVER, user=USUARIO, password=PASSWORD, database=DATABASE
)
cursor = conn.cursor(dictionary=True)
# 4. Validaciones contra DB
query = """
SELECT t.id_tarjeta, t.id_cuenta, t.estado as t_estado, t.fecha_vencimiento, t.cvv,
c.tipo_cuenta, c.numero_cuenta, c.estado as c_estado
FROM tarjeta t
JOIN cuenta c ON t.id_cuenta = c.id_cuenta
WHERE REPLACE(t.numero_tarjeta, '-', '') = REPLACE(%s, '-', '')
"""
cursor.execute(query, (n_tarjeta,))
tarjeta_db = cursor.fetchone()
if not tarjeta_db or tarjeta_db['t_estado'] != 'ACTIVA':
return {"estado": "ERROR", "mensaje": "Tarjeta no existe o inactiva"}
# Comparar datos descifrados con DB
if str(tarjeta_db['fecha_vencimiento']) != f_vencimiento or tarjeta_db['cvv'] != cvv:
return {"estado": "ERROR", "mensaje": "Validación de seguridad fallida"}
Validar Código de Autorización previo
cursor.execute("SELECT * FROM autorizacion WHERE codigo_autorizacion = %s AND
id_tarjeta = %s",
(datos['cod_auth'], tarjeta_db['id_tarjeta']))
if not cursor.fetchone():
return {"estado": "ERROR", "mensaje": "Código de autorización no corresponde"}
# 5. Lógica de Negocio (Débito vs Crédito)
monto = float(datos['monto'])
if tarjeta_db['tipo_cuenta'] == 'DEBITO':
# --- CONSTRUCCIÓN DE TRAMA SINCRONIZADA CON COREBANCARIO.JAVA ---
# Tipo(1) + Cuenta(23) + Tarjeta(18) + CodAuth(8) + Monto(8) = 58 caracteres
num_tarjeta_limpia = n_tarjeta.replace("-", "")[:18]
trama_core = (
"C" +
tarjeta_db['numero_cuenta'].ljust(23) +
num_tarjeta_limpia.ljust(18) +
datos['cod_auth'].ljust(8) +
f"{int(monto * 100):08d}"
)
try:
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s_java:
s_java.settimeout(5)
s_java.connect(('127.0.0.1', PUERTO_CORE_JAVA))
s_java.sendall((trama_core + "\n").encode())
resp_core = s_java.recv(1024).decode().strip()
print(f"DEBUG: Trama enviada -> {trama_core}")
Lín. 12, col. 1 Espacios: 4 UTF-8 CRLF { } Python 3.14.0 Go Live

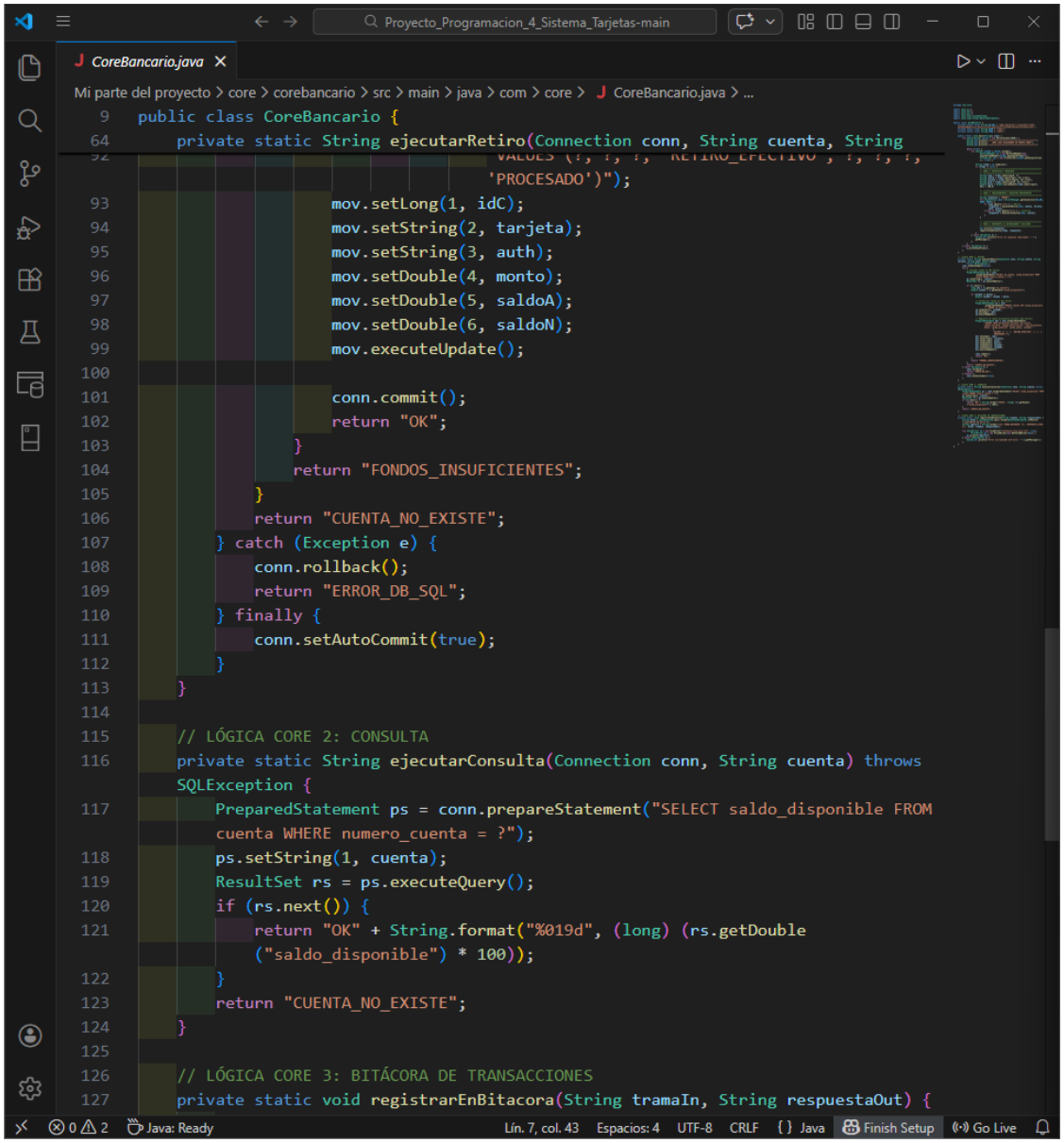

Archivo Editar Selección
AUT
seguridad.py
servidor.py
config.py
AUT5.py
AUT4.py
AUT3.py
AUT1_AUT2.py
AUT5.py > procesar_confirmacion_aut5
def procesar_confirmacion_aut5(trama_json):
s_java.settimeout(5)
s_java.connect(('127.0.0.1', PUERTO_CORE_JAVA))
s_java.sendall((trama_core + "\n").encode())
resp_core = s_java.recv(1024).decode().strip()
print(f"DEBUG: Trama enviada -> {trama_core}")
print(f"DEBUG: El Core Java respondió -> '{resp_core}'")
if "OK" in resp_core.upper():
registrar_evento_aut4(n_tarjeta, datos['id_cajero'],
"CONFIRMACION_RETIRO", monto)
return {"estado": "OK", "codigo_autorizacion": datos['cod_auth']}
else:
return {"estado": "ERROR", "mensaje": f"Core respondio: {resp_core}"}
except Exception as e:
print(f"Error de conexión con Java: {e}")
return {"estado": "ERROR", "mensaje": "Core no disponible"}
else: # CRÉDITO
cursor.execute("""
INSERT INTO movimiento (id_tarjeta, codigo_autorizacion, tipo_movimiento,
monto, estado)
VALUES (%s, %s, 'RETIRO_CONFIRMADO', %s, 'PENDIENTE')
""", (tarjeta_db['id_tarjeta'], datos['cod_auth'], monto))
conn.commit()
registrar_evento_aut4(n_tarjeta, datos['id_cajero'],
"CONFIRMACION_CREDITO_PENDIENTE", monto)
return {"estado": "OK", "codigo_autorizacion": datos['cod_auth']}
except Exception as e:
return {"estado": "ERROR", "mensaje": str(e)}
finally:
if conn and conn.is_connected():
cursor.close()
conn.close()
Lín. 104, col. 13 Espacios: 4 UTF-8 CRLF Python 3.14.0 Go Live

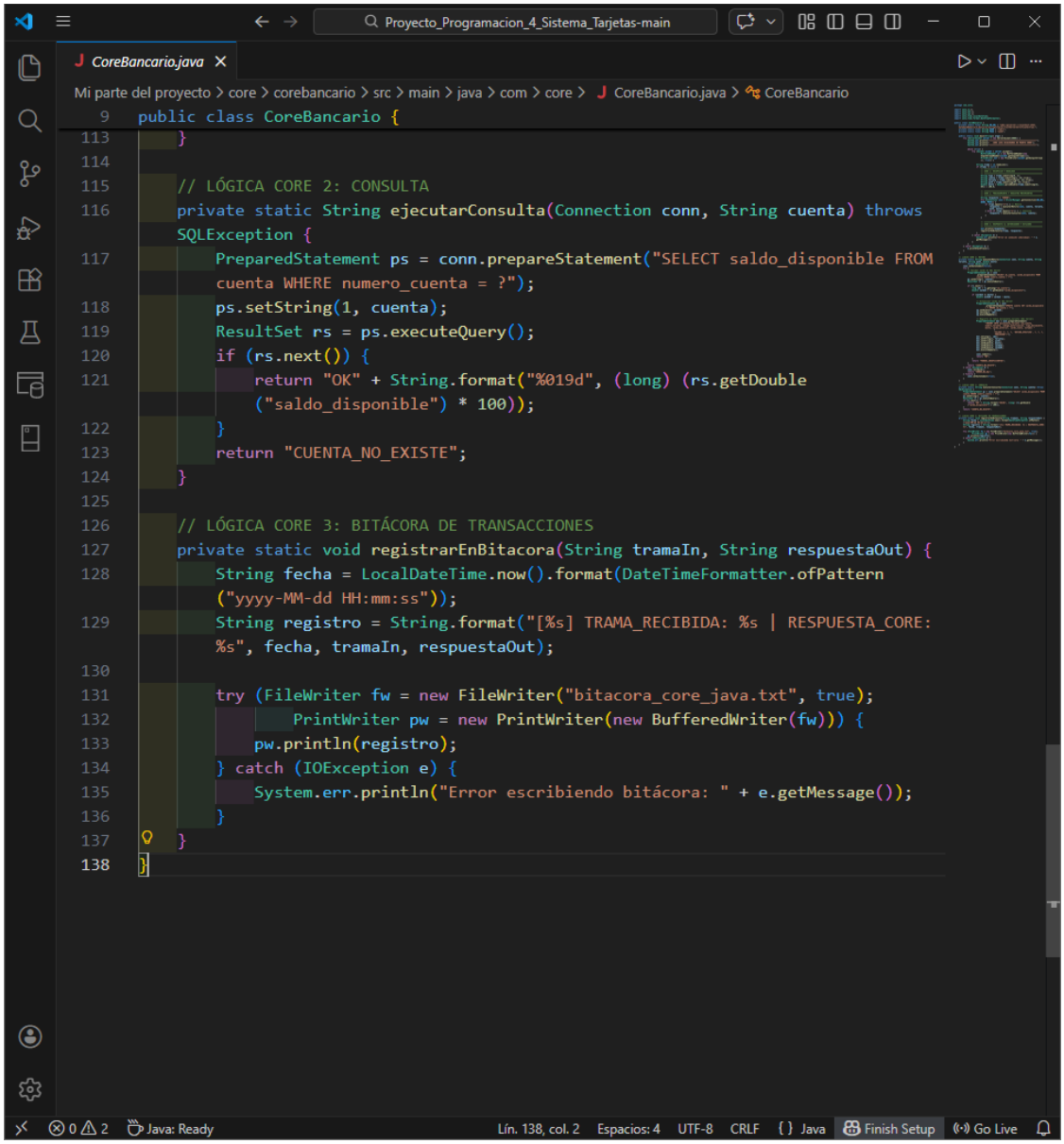
Capitulo 4: Componentes del Core Bancario



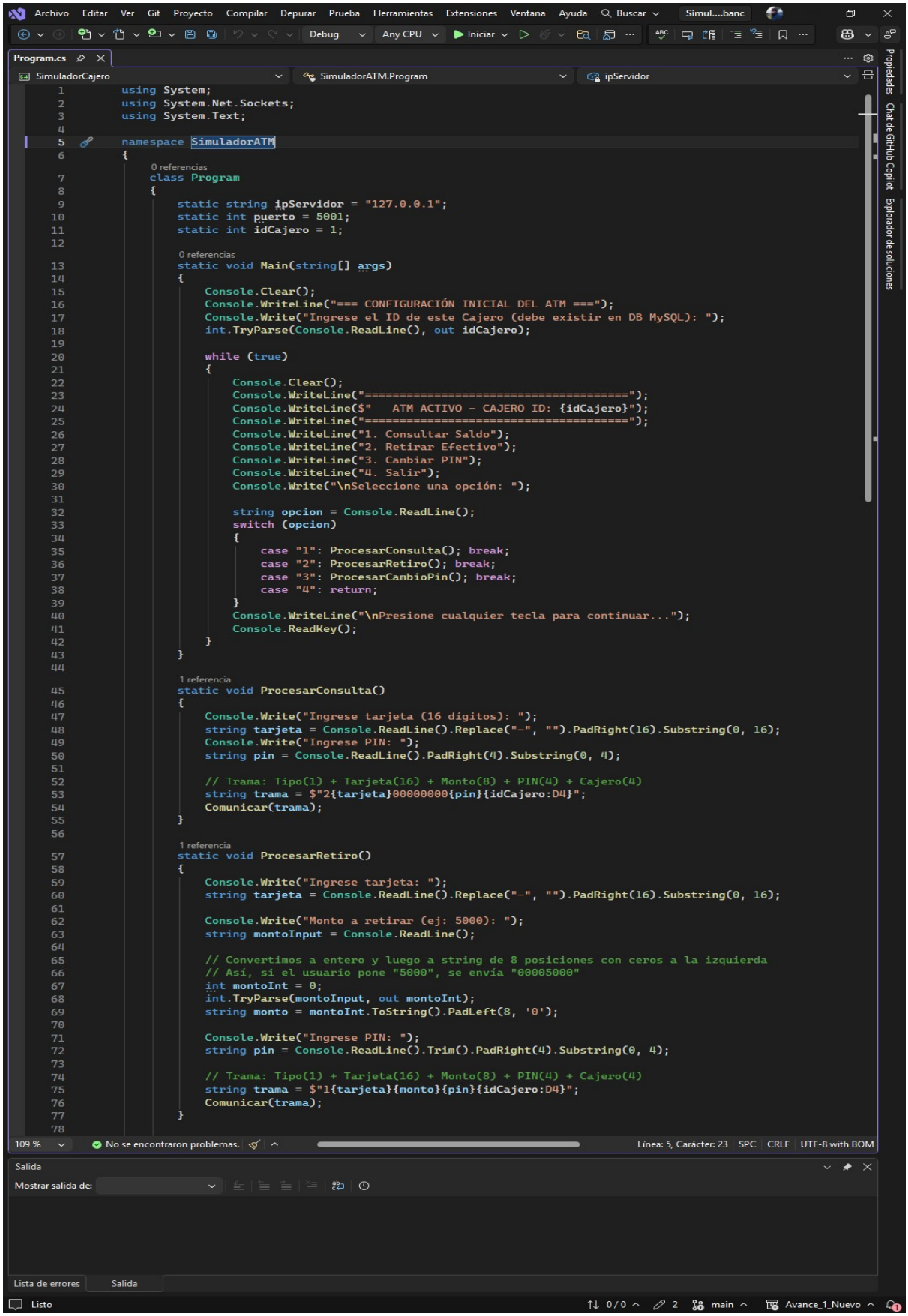

Proyecto_Programacion_4_Sistema_Tarjetas-main
CoreBancario.java
Mi parte del proyecto > core > corebancario > src > main > java > com > core > CoreBancario.java > ...
public class CoreBancario {
public static void main(String[] args) {
String auth = trama.substring(43, 51).trim();
double monto = Double.parseDouble(trama.substring(51,
59)) / 100.0;
// ==
// CORE 2: PROCESAMIENTO Y REGISTRO MOVIMIENTOS
// ==
String respuesta = "ERROR";
try (Connection conn = DriverManager.getConnection(DB_URL,
USER, PASS)) {
if (tipo.equals("1")) { // Retiro
respuesta = ejecutarRetiro(conn, cuenta, tarjeta,
auth, monto);
} else if (tipo.equals("2")) { // Consulta
respuesta = ejecutarConsulta(conn, cuenta);
}
}
// ==
// CORE 3: RESPUESTA AL AUTORIZADOR Y BITÁCORA
// ==
out.println(respuesta);
registrarEnBitacora(trama, respuesta);
}
} catch (Exception e) {
System.err.println("Error en conexión individual: " + e.
getMessage());
}
}
} catch (Exception e) {
e.printStackTrace();
}
}
// LÓGICA CORE 2: RETIRO
private static String ejecutarRetiro(Connection conn, String cuenta, String
tarjeta, String auth, double monto)
throws SQLException {
conn.setAutoCommit(false);
try {
0 2
Java: Ready
Lín. 7, col. 43
Espacios: 4
UTF-8
CRLF
Java
Finish Setup
Go Live


Proyecto_Programacion_4_Sistema_Tarjetas-main
CoreBancario.java
Mi parte del proyecto > core > corebancario > src > main > java > com > core > CoreBancario.java > ...
public class CoreBancario {
// LÓGICA CORE 2: RETIRO
private static String ejecutarRetiro(Connection conn, String cuenta, String tarjeta, String auth, double monto)
throws SQLException {
conn.setAutoCommit(false);
try {
// Validar saldo en SQL Server
PreparedStatement ps = conn
.prepareStatement("SELECT id_cuenta, saldo_disponible FROM cuenta WHERE numero_cuenta = ?");
ps.setString(1, cuenta);
ResultSet rs = ps.executeQuery();
if (rs.next()) {
long idC = rs.getLong("id_cuenta");
double saldoA = rs.getDouble("saldo_disponible");
if (saldoA >= monto) {
double saldoN = saldoA - monto;
// Actualizar saldo en SQL Server
PreparedStatement up = conn
.prepareStatement("UPDATE cuenta SET saldo_disponible = ? WHERE id_cuenta = ?");
up.setDouble(1, saldoN);
up.setLong(2, idC);
up.executeUpdate();
// Registro en Tabla movimiento_tarjeta (SQL Server)
PreparedStatement mov = conn.prepareStatement(
"INSERT INTO movimiento_tarjeta (id_cuenta, numero_tarjeta, codigo_autorizacion, tipo_movimiento, monto, saldo_anterior, saldo_nuevo, estado) "
+
"VALUES (?, ?, ?, 'RETIRO_EFECTIVO', ?, ?, ?, 'PROCESADO')");
mov.setLong(1, idC);
mov.setString(2, tarjeta);
mov.setString(3, auth);
mov.setDouble(4, monto);
Java: Ready
Lín. 7, col. 43
Espacios: 4
UTF-8
CRLF
Java
Finish Setup
Go Live

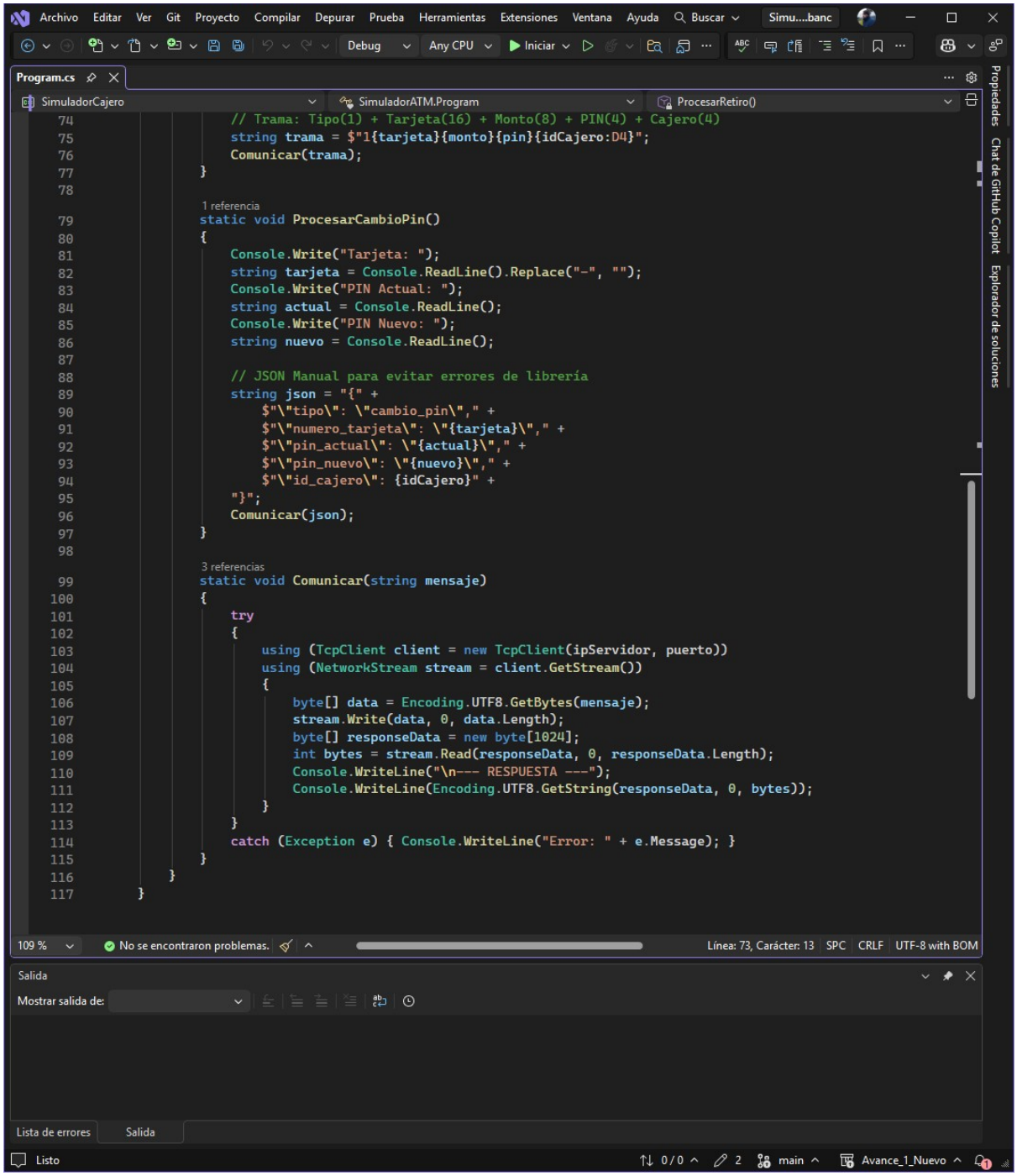

Proyecto_Programacion_4_Sistema_Tarjetas-main
CoreBancario.java
Mi parte del proyecto > core > corebancario > src > main > java > com > core > CoreBancario.java > ...
public class CoreBancario {
private static String ejecutarRetiro(Connection conn, String cuenta, String
'PROCESADO')");
mov.setLong(1, idC);
mov.setString(2, tarjeta);
mov.setString(3, auth);
mov.setDouble(4, monto);
mov.setDouble(5, saldoA);
mov.setDouble(6, saldoN);
mov.executeUpdate();
conn.commit();
return "OK";
return "FONDOS_INSUFICIENTES";
return "CUENTA_NO_EXISTE";
} catch (Exception e) {
conn.rollback();
return "ERROR_DB_SQL";
} finally {
conn.setAutoCommit(true);
// LÓGICA CORE 2: CONSULTA
private static String ejecutarConsulta(Connection conn, String cuenta) throws
SQLException {
PreparedStatement ps = conn.prepareStatement("SELECT saldo_disponible FROM
cuenta WHERE numero_cuenta = ?");
ps.setString(1, cuenta);
ResultSet rs = ps.executeQuery();
if (rs.next()) {
return "OK" + String.format("%019d", (long) (rs.getDouble
("saldo_disponible") * 100));
return "CUENTA_NO_EXISTE";
// LÓGICA CORE 3: BITÁCORA DE TRANSACCIONES
private static void registrarEnBitacora(String tramaIn, String respuestaOut) {
Java: Ready
Lín. 7, col. 43
Espacios: 4
UTF-8
CRLF
Java
Finish Setup
Go Live


Proyecto_Programacion_4_Sistema_Tarjetas-main
CoreBancario.java
Mi parte del proyecto > core > corebancario > src > main > java > com > core > CoreBancario.java > CoreBancario
public class CoreBancario {
}
// LÓGICA CORE 2: CONSULTA
private static String ejecutarConsulta(Connection conn, String cuenta) throws SQLException {
PreparedStatement ps = conn.prepareStatement("SELECT saldo_disponible FROM cuenta WHERE numero_cuenta = ?");
ps.setString(1, cuenta);
ResultSet rs = ps.executeQuery();
if (rs.next()) {
return "OK" + String.format("%019d", (long) (rs.getDouble("saldo_disponible") * 100));
}
return "CUENTA_NO_EXISTE";
}
// LÓGICA CORE 3: BITÁCORA DE TRANSACCIONES
private static void registrarEnBitacora(String tramaIn, String respuestaOut) {
String fecha = LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"));
String registro = String.format("[%s] TRAMA_RECIBIDA: %s | RESPUESTA_CORE: %s", fecha, tramaIn, respuestaOut);
try (FileWriter fw = new FileWriter("bitacora_core_java.txt", true);
PrintWriter pw = new PrintWriter(new BufferedWriter(fw))) {
pw.println(registro);
} catch (IOException e) {
System.err.println("Error escribiendo bitácora: " + e.getMessage());
}
}
}
Java: Ready
Lín. 138, col. 2
Espacios: 4
UTF-8
CRLF
Java
Finish Setup
Go Live

Capitulo 5: Simulador de Transacciones



```
using System;
using System.Net.Sockets;
using System.Text;

namespace SimuladorATM
{
    0 referencias
    class Program
    {
        static string ipServidor = "127.0.0.1";
        static int puerto = 5001;
        static int idCajero = 1;

        0 referencias
        static void Main(string[] args)
        {
            Console.Clear();
            Console.WriteLine("=== CONFIGURACIÓN INICIAL DEL ATM ===");
            Console.Write("Ingrese el ID de este Cajero (debe existir en DB MySQL): ");
            int.TryParse(Console.ReadLine(), out idCajero);

            while (true)
            {
                Console.Clear();
                Console.WriteLine("======================================");
                Console.WriteLine($"   ATM ACTIVO - CAJERO ID: {idCajero}");
                Console.WriteLine("======================================");
                Console.WriteLine("1. Consultar Saldo");
                Console.WriteLine("2. Retirar Efectivo");
                Console.WriteLine("3. Cambiar PIN");
                Console.WriteLine("4. Salir");
                Console.Write("\nSeleccione una opción: ");

                string opcion = Console.ReadLine();
                switch (opcion)
                {
                    case "1": ProcesarConsulta(); break;
                    case "2": ProcesarRetiro(); break;
                    case "3": ProcesarCambioPin(); break;
                    case "4": return;
                }
                Console.WriteLine("\nPresione cualquier tecla para continuar...");
                Console.ReadKey();
            }
        }

        1 referencia
        static void ProcesarConsulta()
        {
            Console.Write("Ingrese tarjeta (16 dígitos): ");
            string tarjeta = Console.ReadLine().Replace("-", "").PadRight(16).Substring(0, 16);
            Console.Write("Ingrese PIN: ");
            string pin = Console.ReadLine().PadRight(4).Substring(0, 4);

            // Trama: Tipo(1) + Tarjeta(16) + Monto(8) + PIN(4) + Cajero(4)
            string trama = $"2{tarjeta}00000000{pin}{idCajero:D4}";
            Comunicar(trama);
        }

        1 referencia
        static void ProcesarRetiro()
        {
            Console.Write("Ingrese tarjeta: ");
            string tarjeta = Console.ReadLine().Replace("-", "").PadRight(16).Substring(0, 16);

            Console.Write("Monto a retirar (ej: 5000): ");
            string montoInput = Console.ReadLine();

            // Convertimos a entero y luego a string de 8 posiciones con ceros a la izquierda
            // Así, si el usuario pone "5000", se envía "00005000"
            int montoInt = 0;
            int.TryParse(montoInput, out montoInt);
            string monto = montoInt.ToString().PadLeft(8, '0');

            Console.Write("Ingrese PIN: ");
            string pin = Console.ReadLine().Trim().PadRight(4).Substring(0, 4);

            // Trama: Tipo(1) + Tarjeta(16) + Monto(8) + PIN(4) + Cajero(4)
            string trama = $"1{tarjeta}{monto}{pin}{idCajero:D4}";
            Comunicar(trama);
        }
```


Archivo Editar Ver Git Proyecto Compilar Depurar Prueba Herramientas Extensiones Ventana Ayuda Buscar
Debug
Any CPU
Iniciar
Program.cs
SimuladorCajero
SimuladorATM.Program
ProcesarRetiro()
// Trama: Tipo(1) + Tarjeta(16) + Monto(8) + PIN(4) + Cajero(4)
string trama = $"1{tarjeta}{monto}{pin}{idCajero:D4}";
Comunicar(trama);
1 referencia
static void ProcesarCambioPin()
Console.Write("Tarjeta: ");
string tarjeta = Console.ReadLine().Replace("-", "");
Console.Write("PIN Actual: ");
string actual = Console.ReadLine();
Console.Write("PIN Nuevo: ");
string nuevo = Console.ReadLine();
// JSON Manual para evitar errores de librería
string json = "{" +
$"\"tipo\": \"cambio_pin\"," +
$"\"numero_tarjeta\": \"{tarjeta}\"," +
$"\"pin_actual\": \"{actual}\"," +
$"\"pin_nuevo\": \"{nuevo}\"," +
$"\"id_cajero\": {idCajero}" +
"}";
Comunicar(json);
3 referencias
static void Comunicar(string mensaje)
try
using (TcpClient client = new TcpClient(ipServidor, puerto))
using (NetworkStream stream = client.GetStream())
byte[] data = Encoding.UTF8.GetBytes(mensaje);
stream.Write(data, 0, data.Length);
byte[] responseData = new byte[1024];
int bytes = stream.Read(responseData, 0, responseData.Length);
Console.WriteLine("\n--- RESPUESTA ---");
Console.WriteLine(Encoding.UTF8.GetString(responseData, 0, bytes));
catch (Exception e) { Console.WriteLine("Error: " + e.Message); }
109 %
No se encontraron problemas.
Línea: 73, Carácter: 13
SPC
CRLF
UTF-8 with BOM
Salida
Mostrar salida de:
Lista de errores
Salida
Listo
0 / 0
2
main
Avance_1_Nuevo
Propiedades
Chat de GitHub Copilot
Explorador de soluciones

Conclusiones

El desarrollo del Avance 1 del Sistema de Tarjetas representa un hito importante en la comprensión y aplicación práctica de los conceptos de programación distribuida, comunicación cliente-servidor, gestión de concurrencia y persistencia de datos en entornos reales. A lo largo de este primer avance, el grupo ha establecido las bases técnicas de un sistema bancario distribuido que integra múltiples componentes críticos de una arquitectura transaccional moderna.

La exitosa implementación del flujo autorizador muestra la habilidad del equipo para diseñar y ejecutar sistemas que manejan comunicación asíncrona, validación de seguridad y auditoría en tiempo real. La adopción de un modelo de bitácora en segundo plano mediante hilos independientes refleja un entendimiento profundo de los principios de no bloqueo y escalabilidad, que son clave en sistemas de alta disponibilidad como los que exige el sector financiero.

Además, el desarrollo del Core Bancario en Java con SQL Server muestra la competencia técnica del grupo en la integración de bases de datos relacionales empresariales y el manejo de transacciones. La correcta implementación del formato de trama fijo y el uso de PreparedStatements con commit/rollback aseguran la integridad referencial y la consistencia de datos, que son esenciales en cualquier sistema de gestión monetaria.

El proyecto ha permitido al equipo fortalecer sus conocimientos en arquitectura de microservicios, donde cada componente, opera de forma independiente pero coordinada, comunicándose mediante protocolos estandarizados. Esta separación de responsabilidades facilita el mantenimiento, la escalabilidad horizontal y la futura adición de nuevos servicios.

LOGROS ALCANZADOS

Infraestructura distribuida robusta: Se estableció una arquitectura cliente-servidor multi-hilo capaz de manejar múltiples transacciones concurrentes sin pérdida de datos.

Sistema de auditoría completo: La implementación de bitácoras en tiempo real con encolamiento de mensajes asegura la trazabilidad total de operaciones y cumple con los estándares regulatorios bancarios.

Integración multiplataforma: La interoperabilidad entre Python, Java & SQL Server, demuestra habilidad en el manejo de heterogeneidad tecnológica.

Seguridad transaccional: Validaciones estrictas de PIN, tarjeta, fondos disponibles y códigos de autorización protegen la integridad del sistema.

Persistencia empresarial: El diseño de esquemas relacionales normalizados con índices optimizados asegura rendimiento y escalabilidad en SQL Server.

Recomendaciones

A. Arquitectura y Escalabilidad

Implementar balanceadores de carga para distribuir horizontalmente transacciones entre múltiples instancias del autorizador.

Usar colas de mensajes distribuidas en lugar de colas en memoria para asegurar la persistencia de eventos críticos.

Utilizar contenedores con Docker para despliegues consistentes en entornos de desarrollo, pruebas y producción.

B. Seguridad y Cumplimiento

Usar algoritmos criptográficos robustos (AES-256) para cifrar datos sensibles en tránsito y reposo.

Implementar autenticación mutua entre componentes del sistema.

Gestionar la rotación automática de claves y auditorías de accesos a datos sensibles.

C. Monitoreo y Observabilidad

Crear alertas en tiempo real para fallos críticos (conexiones a BD, saldos negativos).

D. Calidad del Software

Mantener una cobertura de pruebas unitarias mayor al 85% para lógica crítica (validaciones, transacciones).

Realizar pruebas de integración automatizadas para validar la comunicación entre componentes.

Ejecutar pruebas de carga simulando más de 1000 transacciones concurrentes.

E. Base de Datos

Implementar replicación de lectura en SQL Server para separar cargas.

Realizar backups automatizados con una estrategia 3-2-1 (tres copias, dos medios, una fuera del sitio).

F. Experiencia del Desarrollador

Estandarizar convenciones de nombres, formatos y documentación.

COMPETENCIAS CONSOLIDADAS

El proyecto ha permitido al equipo dominar:

- Arquitectura de sistemas distribuidos
- Programación concurrente y paralela
- Integración empresarial multiplataforma
- Persistencia de datos ACID
- Gestión de transacciones financieras
- Auditoría y trazabilidad
- Optimización de rendimiento

PERSPECTIVAS

Este Avance 1 coloca al equipo en una buena posición para enfrentar desarrollos críticos en el sector financiero. La experiencia adquirida es aplicable a proyectos reales y sistemas de pagos.

Recomendación final: Mantener el rigor técnico demostrado, priorizar la calidad sobre la velocidad de desarrollo y documentar cada decisión arquitectónica para facilitar la incorporación de nuevos miembros y futuras auditorías.

Bibliografía

NestorLeiva. (s. f.). GitHub - NestorLeiva/Proyecto_Programacion_4_Sistema_Tarjetas: Avance 1. GitHub. https://github.com/NestorLeiva/Proyecto_Programacion_4_Sistema_Tarjetas

ChartDB – Database schema diagrams visualizer. (2025, 11 agosto).

https://chartdb.io/ *Google Gemini*. (s. f.). Gemini.

https://gemini.google.com/app?hl=es

IA. (s. f.). Perplexity. https://www.perplexity.ai/

Sockets con Python. (s. f.). YouTube. https://www.youtube.com/playlist?list=PLoTnWByJggZUXdGv6aqRVkbR6qtVecZBG