**📋 Análisis Completo del Proyecto**

**🗂️ ARCHIVOS - Clasificación por Utilidad**

**✅ ARCHIVOS CRÍTICOS (Mantener)**

| **Archivo** | **Función** | **Estado** |
| --- | --- | --- |
| main.py | Punto de entrada FastAPI | ✅ Crítico |
| chat.py (endpoints) | API principal de chat | ✅ Crítico |
| flow\_manager.py | Gestor de flujo principal | ✅ Crítico |
| session.py | Configuración SQLAlchemy | ✅ Crítico |
| Modelos (user.py, conversation.py, message.py) | Modelos SQLAlchemy | ✅ Críticos |
| base\_conocimiento.yaml | Configuración de flujo | ✅ Crítico |

**⚠️ MACHINE LEARNING - CONFUSIÓN IDENTIFICADA**

**PROBLEMA**: Tienes 3 archivos ML diferentes:

| **Archivo** | **Complejidad** | **Líneas** | **Recomendación** |
| --- | --- | --- | --- |
| ml.py | Simple | ~400 | ✅ **MANTENER** (Sklearn básico) |
| ml\_service.py | Extrema | ~1000+ | ❌ **ELIMINAR** (Demasiado complejo) |
| ml\_service\_adaptado.py | Media | ~600 | ✅ **MANTENER** (Versión práctica) |

**DECISIÓN RECOMENDADA**:

* **USAR**: ml\_service\_adaptado.py como engine principal
* **MANTENER**: ml.py como clasificador base
* **ELIMINAR**: ml\_service.py (innecesariamente complejo)

**❌ ARCHIVOS INNECESARIOS (Eliminar)**

| **Archivo** | **Razón** | **Acción** |
| --- | --- | --- |
| base\_class.py | Duplicado de base.py | 🗑️ Eliminar |
| models.py | No usado, conflicto con models/ | 🗑️ Eliminar |
| crud\_user.py | Obsoleto, reemplazado por servicios | 🗑️ Eliminar |
| crud\_conversation.py | Obsoleto, reemplazado por servicios | 🗑️ Eliminar |

**🔧 ARCHIVOS ÚTILES (Mantener)**

| **Categoría** | **Archivos** | **Estado** |
| --- | --- | --- |
| **Servicios** | state\_manager.py, variable\_service.py, log\_service.py | ✅ Útiles |
| **Configuración** | config.py, auth.py, logging.py | ✅ Necesarios |
| **Admin** | admin\_config.py, migracion\_yaml\_sql.py | ✅ Importantes |
| **Schemas** | Todos en schemas/ | ✅ Necesarios |
| **Monitoreo** | monitoring\_system.py | ✅ Útil |

**🗄️ INVENTARIO COMPLETO DE TABLAS**

**📊 TABLAS PRINCIPALES (Core)**

sql

*-- Usuarios y Conversaciones*

users *-- Usuarios del sistema*

conversations *-- Conversaciones activas*

messages *-- Mensajes de conversación*

**⚙️ TABLAS DE CONFIGURACIÓN**

sql

*-- Sistema de Estados*

Estados\_Conversacion *-- Estados del flujo (validar\_documento, etc.)*

Opciones\_Estado *-- Botones/opciones por estado*

Variables\_Sistema *-- Variables del template ({{nombre\_cliente}})*

Configuracion\_Global *-- Configuración general del sistema*

**🤖 TABLAS MACHINE LEARNING**

sql

*-- Intenciones y Patrones*

Intenciones *-- Intenciones ML (CONSULTA\_DEUDA, PAGO, etc.)*

Patrones\_Intencion *-- Patrones por intención ("cuanto debo", etc.)*

Sinonimos *-- Sinónimos para NLP ("deuda" = "obligación")*

modelos\_ml *-- Registro de modelos entrenados*

datos\_entrenamiento *-- Datos para entrenar clasificadores*

predicciones\_ml *-- Log de predicciones ML*

**💰 TABLAS DE NEGOCIO (Críticas)**

sql

*-- Datos de Clientes (LA MÁS IMPORTANTE)*

ConsolidadoCampañasNatalia *-- Clientes, deudas, ofertas, etc.*

planes\_pago *-- Planes de pago registrados*

**📈 TABLAS DE MONITOREO**

sql

*-- Métricas y Performance*

metricas\_conversacion *-- Métricas por conversación*

performance\_metrics *-- Métricas de rendimiento*

ml\_cache *-- Cache de predicciones ML*

**🚨 PROBLEMAS CRÍTICOS IDENTIFICADOS**

**1. Confusión ML**

* 3 archivos ML diferentes
* Imports circulares
* No está claro cuál usar

**2. Archivos Duplicados**

* base.py vs base\_class.py
* CRUDs obsoletos vs servicios actuales

**3. Imports Rotos**

python

*# En flow\_manager.py - PROBLEMA*

from app.machine\_learning.ml\_service\_adaptado import MLConversationEngineAdaptado

*# Pero el archivo se llama ml\_service\_adaptado.py, no está en carpeta machine\_learning/*

**📋 PLAN DE LIMPIEZA RECOMENDADO**

**PASO 1: Eliminar Archivos Innecesarios**

bash

*# Eliminar duplicados y obsoletos*

rm app/db/base\_class.py

rm app/models.py

rm app/crud/crud\_user.py

rm app/crud/crud\_conversation.py

rm app/machine\_learning/ml\_service.py *# El complejo*

**PASO 2: Reorganizar ML**

bash

*# Mantener solo estos 2 archivos ML:*

app/machine\_learning/ml.py *# Clasificador sklearn*

app/machine\_learning/ml\_service\_adaptado.py *# Engine principal*

**PASO 3: Corregir Imports**

* Actualizar imports en flow\_manager.py
* Verificar imports en chat.py
* Asegurar rutas correctas

**🎯 ESTRUCTURA FINAL RECOMENDADA**

app/

├── api/endpoints/

│ ├── chat.py # ✅ API principal

│ └── admin\_config.py # ✅ Admin

├── core/

│ ├── config.py # ✅ Config

│ └── logging.py # ✅ Logging

├── db/

│ ├── base.py # ✅ Base SQLAlchemy

│ └── session.py # ✅ Sesión DB

├── models/

│ ├── user.py # ✅ Modelo usuario

│ ├── conversation.py # ✅ Modelo conversación

│ └── message.py # ✅ Modelo mensaje

├── schemas/

│ └── [todos los schemas] # ✅ Schemas Pydantic

├── services/

│ ├── flow\_manager.py # ✅ Gestor principal

│ ├── state\_manager.py # ✅ Estados

│ └── [otros servicios] # ✅ Servicios

├── machine\_learning/

│ ├── ml.py # ✅ Clasificador sklearn

│ └── ml\_service\_adaptado.py # ✅ Engine ML

└── main.py # ✅ Entry point

**⚡ PRÓXIMOS PASOS**

1. **¿Confirmas la limpieza de archivos?**
2. **¿Procedemos con la corrección de imports?**
3. **¿Revisamos la estructura de tablas en detalle?**

¿Qué decides hacer primero?

**🤖 Sistema de Negociación de Deudas - Chatbot Inteligente**

Un sistema avanzado de chatbot para negociación de deudas con integración de Machine Learning, OpenAI y gestión de flujos conversacionales configurable.

**📋 Tabla de Contenidos**

* [Características](#-caracter%C3%ADsticas)
* [Arquitectura](#%EF%B8%8F-arquitectura)
* [Requisitos](#-requisitos)
* [Instalación](#-instalaci%C3%B3n)
* [Configuración](#%EF%B8%8F-configuraci%C3%B3n)
* [Estructura de Base de Datos](#-estructura-de-base-de-datos)
* [API Endpoints](#-api-endpoints)
* [Flujo de Conversación](#-flujo-de-conversaci%C3%B3n)
* [Integración ML](#-integraci%C3%B3n-ml)
* [Variables del Sistema](#-variables-del-sistema)
* [Despliegue](#-despliegue)
* [Troubleshooting](#-troubleshooting)

**🚀 Características**

**Core Features**

* **🎯 Detección Automática de Cédulas**: Reconocimiento inteligente de documentos de identidad
* **🧠 Machine Learning**: Clasificación automática de intenciones del usuario
* **🤖 Integración OpenAI**: Humanización de respuestas y generación empática
* **📊 Gestión de Estados**: Flujo conversacional configurable y persistente
* **💰 Cálculo Dinámico**: Ofertas y planes de pago personalizados
* **📋 Contexto Persistente**: Mantenimiento de información del cliente entre sesiones

**Funcionalidades Avanzadas**

* **🔄 Cache Inteligente**: Sistema de cache para contextos y configuraciones
* **🛡️ Recuperación de Emergencia**: Fallbacks automáticos ante errores
* **📈 Logging Completo**: Registro detallado de todas las interacciones
* **🎨 UI Dinámica**: Botones contextuales basados en el estado de la conversación

**🏗️ Arquitectura**

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐

│ Frontend │ │ FastAPI │ │ SQL Server │

│ (Chat UI) │◄──►│ Backend │◄──►│ Database │

└─────────────────┘ └─────────────────┘ └─────────────────┘

│

▼

┌─────────────────┐

│ Servicios │

│ ┌───────────┐ │

│ │ ML │ │

│ │ Service │ │

│ └───────────┘ │

│ ┌───────────┐ │

│ │ OpenAI │ │

│ │ Service │ │

│ └───────────┘ │

│ ┌───────────┐ │

│ │Variables │ │

│ │ Service │ │

│ └───────────┘ │

└─────────────────┘

**📦 Requisitos**

**Software**

* **Python**: 3.9+
* **SQL Server**: 2017+
* **Node.js**: 16+ (para frontend)

**Dependencias Python**

txt

fastapi>=0.104.0

uvicorn[standard]>=0.24.0

sqlalchemy>=2.0.0

pyodbc>=4.0.39

python-multipart>=0.0.6

python-dotenv>=1.0.0

pydantic>=2.5.0

scikit-learn>=1.3.0

joblib>=1.3.2

openai>=1.3.0

PyYAML>=6.0.1

pandas>=2.1.0

numpy>=1.24.0

**🔧 Instalación**

**1. Clonar Repositorio**

bash

git clone <repository-url>

cd chatbot-negociacion

**2. Crear Entorno Virtual**

bash

python -m venv venv

source venv/bin/activate *# Linux/Mac*

*# venv\Scripts\activate # Windows*

**3. Instalar Dependencias**

bash

pip install -r requirements.txt

**4. Configurar Base de Datos**

sql

*-- Ejecutar scripts de creación de tablas (ver sección de BD)*

**⚙️ Configuración**

**Variables de Entorno (.env)**

env

# Base de Datos

DATABASE\_URL=mssql+pyodbc://user:password@server/database?driver=ODBC+Driver+17+for+SQL+Server

# OpenAI (Opcional)

OPENAI\_API\_KEY=sk-your-api-key-here

# Configuración del Servidor

HOST=0.0.0.0

PORT=8000

DEBUG=True

# Machine Learning

ML\_MODEL\_PATH=models/intention\_classifier\_FIXED\_20250605\_100645.joblib

ML\_CONFIDENCE\_THRESHOLD=0.7

# Cache

CACHE\_TTL=3600

CONTEXT\_CACHE\_SIZE=1000

**🗄️ Estructura de Base de Datos**

**Tablas Principales**

**1. ConsolidadoCampañasNatalia (Datos de Clientes)**

sql

CREATE TABLE ConsolidadoCampañasNatalia (

Id INT IDENTITY(1,1) PRIMARY KEY,

Nombre\_del\_cliente NVARCHAR(200),

Cedula NVARCHAR(20),

Telefono NVARCHAR(20),

Email NVARCHAR(100),

Saldo\_total DECIMAL(18,2),

Capital DECIMAL(18,2),

Intereses DECIMAL(18,2),

Oferta\_1 DECIMAL(18,2),

Oferta\_2 DECIMAL(18,2),

Oferta\_3 DECIMAL(18,2),

Oferta\_4 DECIMAL(18,2),

banco NVARCHAR(100),

Producto NVARCHAR(100),

NumerodeObligacion NVARCHAR(50),

Campaña NVARCHAR(100),

Hasta\_3\_cuotas DECIMAL(18,2),

Hasta\_6\_cuotas DECIMAL(18,2),

Hasta\_12\_cuotas DECIMAL(18,2),

Hasta\_18\_cuotas DECIMAL(18,2),

created\_at DATETIME DEFAULT GETDATE(),

updated\_at DATETIME DEFAULT GETDATE()

);

**2. conversations (Gestión de Conversaciones)**

sql

CREATE TABLE conversations (

id INT IDENTITY(1,1) PRIMARY KEY,

user\_id INT NOT NULL,

current\_state NVARCHAR(50) DEFAULT 'validar\_documento',

context\_data NVARCHAR(MAX),

is\_active BIT DEFAULT 1,

created\_at DATETIME DEFAULT GETDATE(),

updated\_at DATETIME DEFAULT GETDATE(),

ended\_at DATETIME NULL,

FOREIGN KEY (user\_id) REFERENCES users(id)

);

**3. messages (Log de Mensajes)**

sql

CREATE TABLE messages (

id INT IDENTITY(1,1) PRIMARY KEY,

conversation\_id INT NOT NULL,

sender\_type NVARCHAR(10) CHECK (sender\_type IN ('user', 'system')),

text\_content NVARCHAR(MAX),

button\_selected NVARCHAR(100),

previous\_state NVARCHAR(50),

next\_state NVARCHAR(50),

timestamp DATETIME DEFAULT GETDATE(),

metadata NVARCHAR(MAX),

FOREIGN KEY (conversation\_id) REFERENCES conversations(id)

);

**4. users (Usuarios del Sistema)**

sql

CREATE TABLE users (

id INT IDENTITY(1,1) PRIMARY KEY,

email NVARCHAR(100) UNIQUE NOT NULL,

full\_name NVARCHAR(200),

hashed\_password NVARCHAR(255),

is\_active BIT DEFAULT 1,

created\_at DATETIME DEFAULT GETDATE(),

updated\_at DATETIME DEFAULT GETDATE()

);

**5. Estados\_Conversacion (Configuración de Estados)**

sql

CREATE TABLE Estados\_Conversacion (

id INT IDENTITY(1,1) PRIMARY KEY,

nombre NVARCHAR(50) UNIQUE NOT NULL,

mensaje\_template NVARCHAR(MAX),

accion NVARCHAR(100),

condicion NVARCHAR(100),

estado\_siguiente\_true NVARCHAR(50),

estado\_siguiente\_false NVARCHAR(50),

estado\_siguiente\_default NVARCHAR(50),

activo BIT DEFAULT 1,

created\_at DATETIME DEFAULT GETDATE()

);

**6. Opciones\_Estado (Botones por Estado)**

sql

CREATE TABLE Opciones\_Estado (

id INT IDENTITY(1,1) PRIMARY KEY,

opcion\_id NVARCHAR(50) NOT NULL,

estado\_nombre NVARCHAR(50) NOT NULL,

texto\_boton NVARCHAR(100) NOT NULL,

estado\_destino NVARCHAR(50),

orden\_visualizacion INT DEFAULT 1,

activo BIT DEFAULT 1,

FOREIGN KEY (estado\_nombre) REFERENCES Estados\_Conversacion(nombre)

);

**Índices Recomendados**

sql

*-- Índices para optimización*

CREATE INDEX IDX\_conversations\_user\_active ON conversations(user\_id, is\_active);

CREATE INDEX IDX\_messages\_conversation\_timestamp ON messages(conversation\_id, timestamp);

CREATE INDEX IDX\_consolidado\_cedula ON ConsolidadoCampañasNatalia(Cedula);

CREATE INDEX IDX\_estados\_nombre ON Estados\_Conversacion(nombre);

**🔌 API Endpoints**

**Chat Principal**

* **POST** /api/v1/chat/message - Procesar mensaje del usuario
* **GET** /api/v1/chat/historial/{conversation\_id} - Obtener historial

**Testing y Diagnóstico**

* **POST** /api/v1/chat/test-cedula - Probar búsqueda de cédulas
* **POST** /api/v1/chat/test-variables - Probar resolución de variables
* **POST** /api/v1/chat/test-openai - Verificar integración OpenAI
* **GET** /api/v1/chat/test - Health check del sistema

**Ejemplo de Uso**

bash

*# Enviar mensaje*

curl -X POST "http://localhost:8000/api/v1/chat/message" \

-H "Content-Type: application/json" \

-d '{

"user\_id": 1,

"message": "1077321055",

"conversation\_id": 1

}'

**🔄 Flujo de Conversación**

**Estados del Sistema**

1. **validar\_documento** - Solicitar y validar cédula
2. **informar\_deuda** - Mostrar información del cliente
3. **evaluar\_intencion\_pago** - Evaluar interés en pagar
4. **proponer\_planes\_pago** - Presentar opciones de pago
5. **seleccionar\_plan** - Confirmar selección del plan
6. **generar\_acuerdo** - Generar acuerdo final
7. **cliente\_no\_encontrado** - Manejar cliente no encontrado
8. **gestionar\_objecion** - Manejar objeciones

**Transiciones Automáticas**

python

*# Ejemplo de lógica de transición*

if cedula\_detectada and cliente\_encontrado:

nuevo\_estado = "informar\_deuda"

elif mensaje\_contiene("si", "acepto", "quiero"):

nuevo\_estado = "proponer\_planes\_pago"

elif ml\_intencion == "SOLICITUD\_PLAN":

nuevo\_estado = "proponer\_planes\_pago"

**🧠 Integración ML**

**Modelo de Clasificación**

* **Archivo**: models/intention\_classifier\_FIXED\_20250605\_100645.joblib
* **Intenciones Detectadas**:
  + SALUDO
  + INTENCION\_PAGO
  + SOLICITUD\_PLAN
  + CONFIRMACION
  + RECHAZO
  + CONSULTA\_DEUDA

**Uso del ML**

python

*# El sistema automáticamente:*

1. Analiza cada mensaje del usuario

2. Clasifica la intención con nivel de confianza

3. Toma decisiones basadas en la confianza (>0.7)

4. Aplica fallbacks si la confianza es baja

**🔧 Variables del Sistema**

**Variables Principales**

* {{nombre\_cliente}} - Nombre del cliente
* {{saldo\_total}} - Saldo total de la deuda
* {{banco}} - Entidad financiera
* {{oferta\_1}}, {{oferta\_2}} - Ofertas de pago
* {{hasta\_3\_cuotas}}, {{hasta\_6\_cuotas}} - Planes de cuotas
* {{pago\_flexible}} - Pago con descuento del 60%
* {{ahorro\_maximo}} - Ahorro máximo posible

**Variables Calculadas**

python

*# Cálculos automáticos*

pago\_flexible = saldo\_total \* 0.4 *# 60% descuento*

ahorro\_maximo = saldo\_total - oferta\_1

cuota\_mensual = plan\_total / numero\_cuotas

**🚀 Despliegue**

**Desarrollo Local**

bash

*# Iniciar servidor*

python main.py

*# O usando uvicorn directamente*

uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload

**Producción**

bash

*# Usando gunicorn*

gunicorn app.main:app -w 4 -k uvicorn.workers.UvicornWorker --bind 0.0.0.0:8000

*# O usando Docker*

docker build -t chatbot-negociacion .

docker run -p 8000:8000 chatbot-negociacion

**Variables de Producción**

env

DEBUG=False

DATABASE\_URL=mssql+pyodbc://prod\_user:prod\_pass@prod\_server/prod\_db

OPENAI\_API\_KEY=sk-prod-key

ML\_CONFIDENCE\_THRESHOLD=0.8

**🔍 Troubleshooting**

**Problemas Comunes**

**1. Variables No Se Resuelven**

python

*# Verificar que existe el método en VariableService*

def \_resolver\_variable\_individual\_corregida(self, variable: str, datos: Dict[str, Any]) -> str:

*# Implementación del método*

**2. Cliente No Encontrado**

sql

*-- Verificar datos en la tabla principal*

SELECT TOP 5 \* FROM ConsolidadoCampañasNatalia WHERE Cedula = '1077321055';

**3. Error de Conexión OpenAI**

bash

*# Verificar API key*

curl -H "Authorization: Bearer $OPENAI\_API\_KEY" https://api.openai.com/v1/models

**4. Contexto Se Pierde**

python

*# Verificar persistencia en BD*

SELECT context\_data FROM conversations WHERE id = 1;

**Logs de Debug**

python

*# Activar logs detallados*

import logging

logging.basicConfig(level=logging.DEBUG)

**📊 Monitoreo**

**Métricas Importantes**

* **Tasa de Resolución**: % de conversaciones completadas
* **Tiempo de Respuesta**: Latencia promedio del sistema
* **Precisión ML**: Accuracy del clasificador de intenciones
* **Tasa de Conversión**: % de usuarios que aceptan planes

**Endpoints de Salud**

* GET /api/v1/chat/test - Estado general del sistema
* GET /health - Health check básico
* GET /metrics - Métricas de rendimiento

**📞 Soporte**

Para soporte técnico o consultas:

* **Email**: [soporte@systemgroup.com](mailto:soporte@systemgroup.com)
* **Documentación**: [Wiki del Proyecto]
* **Issues**: [GitHub Issues]

**Versión**: 2.1.0  
**Última Actualización**: Junio 2025  
**Mantenido por**: Systemgroup Tech Team