**🏗️ Arquitectura Detallada y Sistema de Machine Learning**

**🎯 Arquitectura del Sistema**

**📊 Patrón de Capas (Layered Architecture)**

graph TD

A[🌐 API Layer] --> B[🛠️ Services Layer]

B --> C[🗄️ Models Layer]

C --> D[💾 Database Layer]

E[🧠 ML Layer] --> B

F[📊 Monitoring Layer] --> B

G[🔧 Core Layer] --> B

**🏗️ Componentes Principales**

**1. 🌐 API Layer (/api)**

# Responsabilidades:

- Exponer endpoints REST

- Validación de entrada (Pydantic schemas)

- Manejo de autenticación

- Serialización de respuestas

- Rate limiting y middleware

# Componentes clave:

api/endpoints/chat.py # Endpoint principal de chat

api/endpoints/admin\_config.py # Administración del sistema

api/deps.py # Dependencias inyectables

**2. 🛠️ Services Layer (/services)**

# Responsabilidades:

- Lógica de negocio central

- Orquestación de flujos

- Integración entre componentes

- Procesamiento de datos

# Componentes especializados:

services/flow\_manager.py # 🔄 Coordinador principal

services/state\_manager.py # 📊 Gestión de estados

services/variable\_service.py # 🔧 Variables dinámicas

services/nlp\_service.py # 🧠 Procesamiento NLP

services/conversation\_service.py # 💬 Lógica de conversación

**3. 🗄️ Models Layer (/models)**

# Responsabilidades:

- Definición de entidades

- Mapeo objeto-relacional (ORM)

- Validaciones de datos

- Relaciones entre entidades

models/conversation.py # 💬 Conversaciones

models/message.py # 📨 Mensajes

models/user.py # 👤 Usuarios

**4. 💾 Database Layer (/db)**

# Responsabilidades:

- Configuración de conexiones

- Gestión de sesiones

- Modelos base

- Migraciones

db/session.py # 🔌 Pool de conexiones

db/base.py # 🏗️ Clase base para modelos

**5. 🧠 ML Layer (/machine\_learning)**

# Responsabilidades:

- Clasificación de intenciones

- Procesamiento de lenguaje natural

- Entrenamiento de modelos

- Predicción en tiempo real

machine\_learning/ml.py # 🤖 Clasificador principal

machine\_learning/ml\_service\_adaptado.py # 🔧 Servicio adaptado

machine\_learning/train/ # 📚 Scripts de entrenamiento

**🔄 Flujo de Datos Detallado**

**📱 1. Recepción de Mensaje**

# chat.py - Endpoint principal

@router.post("/message")

async def process\_chat\_message(request: ChatRequest):

# 1. Validación de entrada (Pydantic)

# 2. Obtener/crear conversación

# 3. Delegar a flow\_manager

**🎯 2. Procesamiento Inteligente**

# flow\_manager.py - Coordinador central

def procesar\_mensaje(message, conversation\_id, context):

# 1. Análisis ML de intención

ml\_result = ml\_service.classify\_intention(message)

# 2. Detección de patrones (cédulas, números)

patterns = detect\_patterns(message)

# 3. Determinación de acción

action = determine\_action(ml\_result, patterns, context)

# 4. Ejecución de lógica específica

response = execute\_action(action, context)

# 5. Resolución de variables

final\_response = variable\_service.resolver\_variables(response)

return final\_response

**🗄️ 3. Gestión de Estado**

# state\_manager.py - Estados de conversación

def update\_conversation\_state(conversation\_id, new\_state, context\_data):

# 1. Obtener conversación actual

# 2. Actualizar estado

# 3. Serializar contexto a JSON

# 4. Persistir en base de datos

**🔧 4. Variables Dinámicas**

# variable\_service.py - Resolución de variables

def resolver\_variables(text, context):

# 1. Extraer variables {{variable}}

# 2. Buscar valores en contexto y BD

# 3. Formatear según tipo de dato

# 4. Reemplazar en texto final

**🧠 Machine Learning en Detalle**

**🎯 Tipo de ML: Clasificación de Texto Supervisada**

**📚 Enfoque: Natural Language Processing (NLP)**

# Clasificación multiclase para intenciones de chat

Problema: Texto libre → Categoría de intención

Método: Aprendizaje supervisado

Algoritmo: Probablemente Naive Bayes o SVM

**🔧 Pipeline de Machine Learning**

**1. 📝 Preprocessing de Texto**

# ml\_service\_adaptado.py

def preprocess\_text(text):

# 1. Normalización (minúsculas, acentos)

# 2. Tokenización (dividir en palabras)

# 3. Eliminación de stopwords

# 4. Stemming/Lemmatización

# 5. Vectorización (TF-IDF o Bag of Words)

return processed\_features

**2. 🎯 Clasificación de Intenciones**

# Clases de intención entrenadas:

INTENCIONES = {

"SALUDO": ["hola", "buenos días", "buenas tardes"],

"CONSULTA\_SALDO": ["deuda", "debo", "saldo", "cuanto"],

"SOLICITUD\_DESCUENTO": ["descuento", "rebaja", "oferta"],

"ACEPTACION\_OFERTA": ["acepto", "está bien", "de acuerdo"],

"NEGACION": ["no", "rechazar", "no acepto"],

"DESPEDIDA": ["gracias", "adiós", "hasta luego"]

}

# Proceso de clasificación:

def classify\_intention(text):

# 1. Preprocesar texto

features = preprocess\_text(text)

# 2. Aplicar modelo entrenado

prediction = model.predict(features)

confidence = model.predict\_proba(features)

# 3. Retornar intención con confianza

return {

"intencion": prediction,

"confianza": confidence.max()

}

**🔬 Algoritmos ML Implementados**

**📊 Naive Bayes (Probable implementación)**

# Ventajas para clasificación de texto:

✅ Excelente para texto corto (mensajes de chat)

✅ Maneja bien vocabulario limitado

✅ Rápido entrenamiento e inferencia

✅ Funciona bien con datasets pequeños

✅ Probabilidades interpretables

# Características técnicas:

- Asume independencia entre palabras

- Calcula P(intención|palabras)

- Utiliza teorema de Bayes

- Maneja múltiples clases naturalmente

**⚙️ Support Vector Machine (Alternativa posible)**

# Ventajas para NLP:

✅ Excelente con espacios de alta dimensión

✅ Efectivo con TF-IDF features

✅ Margen de separación robusto

✅ Kernel trick para patrones complejos

# Implementación típica:

from sklearn.svm import SVC

from sklearn.feature\_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()

classifier = SVC(kernel='linear', probability=True)

**📈 Feature Engineering**

**🔤 Extracción de Características**

# 1. Features Textuales

- Bag of Words (BoW)

- TF-IDF (Term Frequency-Inverse Document Frequency)

- N-gramas (unigramas, bigramas)

- Longitud del mensaje

- Presencia de números/patrones

# 2. Features Contextuales

- Estado actual de conversación

- Historial de intenciones

- Tiempo en la conversación

- Palabras clave específicas del dominio

# 3. Features de Dominio (Negociación)

- Patrones de cédulas (números de 8-10 dígitos)

- Términos financieros

- Expresiones de cantidad monetaria

- Indicadores de aceptación/rechazo

**🎓 Entrenamiento del Modelo**

**📚 Dataset de Entrenamiento**

# train/train\_intention\_classifier.py

training\_data = [

("hola como estas", "SALUDO"),

("cuanto debo", "CONSULTA\_SALDO"),

("mi cedula es 93388915", "CONSULTA\_SALDO"),

("quiero un descuento", "SOLICITUD\_DESCUENTO"),

("acepto la oferta", "ACEPTACION\_OFERTA"),

("no me interesa", "NEGACION"),

("gracias adios", "DESPEDIDA")

]

# Proceso de entrenamiento:

def train\_model():

# 1. Cargar y limpiar datos

X, y = load\_training\_data()

# 2. Dividir train/test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y)

# 3. Vectorizar texto

vectorizer = TfidfVectorizer()

X\_train\_vec = vectorizer.fit\_transform(X\_train)

# 4. Entrenar clasificador

classifier = MultinomialNB()

classifier.fit(X\_train\_vec, y\_train)

# 5. Evaluar modelo

accuracy = classifier.score(X\_test\_vec, y\_test)

return classifier, vectorizer, accuracy

**⚡ Inferencia en Tiempo Real**

# ml.py - Clasificador en producción

class IntentionClassifier:

def \_\_init\_\_(self):

self.model = joblib.load('trained\_model.pkl')

self.vectorizer = joblib.load('vectorizer.pkl')

def predict(self, text):

# 1. Preprocesar

clean\_text = self.preprocess(text)

# 2. Vectorizar

features = self.vectorizer.transform([clean\_text])

# 3. Predecir

prediction = self.model.predict(features)[0]

confidence = self.model.predict\_proba(features).max()

return {

"intencion": prediction,

"confianza": float(confidence)

}

**🔍 Detección de Patrones**

**🆔 Detección de Cédulas**

# nlp\_service.py

def detectar\_cedula(text):

# Regex para números de 8-10 dígitos

pattern = r'\b\d{8,10}\b'

matches = re.findall(pattern, text)

if matches:

return {

"cedula\_detectada": matches[0],

"patron\_encontrado": True

}

return {"patron\_encontrado": False}

**💰 Detección de Montos**

def detectar\_montos(text):

# Patrones monetarios

patterns = [

r'\$[\d,]+', # $123,456

r'\d+\s\*pesos', # 1000 pesos

r'\d+\s\*mil', # 500 mil

r'\d+\s\*millones' # 2 millones

]

for pattern in patterns:

if re.search(pattern, text, re.IGNORECASE):

return True

return False

**📊 Métricas y Evaluación**

**🎯 Métricas del Modelo**

# Métricas de clasificación

metrics = {

"accuracy": 0.92, # Precisión global

"precision": { # Precisión por clase

"SALUDO": 0.95,

"CONSULTA\_SALDO": 0.88,

"SOLICITUD\_DESCUENTO": 0.85

},

"recall": { # Recall por clase

"SALUDO": 0.93,

"CONSULTA\_SALDO": 0.90,

"SOLICITUD\_DESCUENTO": 0.82

},

"f1\_score": 0.89 # F1 promedio

}

**⏱️ Métricas de Rendimiento**

performance\_metrics = {

"prediction\_time": "< 50ms", # Tiempo de predicción

"model\_size": "2.5MB", # Tamaño del modelo

"memory\_usage": "15MB", # Memoria en runtime

"throughput": "500 predictions/sec" # Capacidad

}

**🔄 Reentrenamiento y Mejora Continua**

**📈 Auto-entrenamiento**

# auto\_train.py

def auto\_retrain():

# 1. Recopilar nuevos datos etiquetados

new\_data = collect\_labeled\_conversations()

# 2. Combinar con dataset existente

combined\_data = merge\_datasets(existing\_data, new\_data)

# 3. Reentrenar modelo

new\_model = train\_model(combined\_data)

# 4. Evaluar mejora

if new\_model.accuracy > current\_model.accuracy:

deploy\_model(new\_model)

**🧹 Limpieza de Datos**

# train/limpieza\_data.py

def clean\_training\_data():

# 1. Eliminar duplicados

# 2. Corregir etiquetas inconsistentes

# 3. Balancear clases

# 4. Validar calidad de datos

# 5. Normalizar formato

**🚀 Optimizaciones Futuras**

**🤖 Modelos Avanzados**

# Posibles mejoras:

1. 🧠 BERT/RoBERTa para mejor comprensión contextual

2. 🔄 LSTM/GRU para secuencias conversacionales

3. 🎯 Transformer personalizado para dominio específico

4. 📊 Ensemble methods combinando múltiples modelos

5. 🔧 Active Learning para etiquetado inteligente

**📈 Funcionalidades Futuras**

# Capacidades en desarrollo:

- Análisis de sentimiento (positivo/negativo)

- Detección de urgencia en mensajes

- Extracción de entidades nombradas (fechas, montos)

- Generación de respuestas con GPT

- Clasificación de personalidad del cliente

- Predicción de probabilidad de pago

**🎯 Integración ML con Arquitectura**

**🔄 Flujo ML-Business Logic**

# Cómo ML se integra en el flujo principal:

1. 📨 Mensaje llega a chat.py

2. 🧠 flow\_manager.py invoca ML

3. 🎯 ml\_service clasifica intención

4. 📊 Resultado ML + contexto → acción

5. 🔧 variable\_service personaliza respuesta

6. 💬 Respuesta contextualizada al usuario

**⚡ Ventajas de la Arquitectura ML**

* **🚀 Latencia baja**: < 50ms por predicción
* **🔧 Modular**: ML independiente de lógica de negocio
* **📈 Escalable**: Fácil cambiar/mejorar modelos
* **🧪 Testeable**: Métricas y evaluación automatizada
* **🔄 Adaptable**: Reentrenamiento automático

Esta arquitectura combina **NLP clásico** con **lógica de negocio robusta** para crear un sistema de chat inteligente específicamente optimizado para **negociación de deudas**.