Reporte de avance

Requerimiento solicitado:

Crear un chatbot que sea capaz de satisfacer la siguiente lógica.

- Bienvenida y menú de opciones
    - Consulta de saldo
        - Captura de numero de tarjeta
            - Validación de vigencia
                - Vigente
                    - devolución de información de saldo
                - Cancelada
                    - Notificación
    - Cancelación de tarjetas
        - Captura de numero de tarjeta
            - validación de vigencia
                - Vigente
                    - Confirmación de movimiento
                        - Si
                            - Folio
                        - No
                            - Menú inicial
                - Cancelada
                    - Notificación de status
    - Información
        - 5 escenarios libre

Objetivo: desarrollar un chatbot con inteligencia artificial en Python que sea capaz de identificar la intención del usuario y responderle de acuerdo con lo solicitado por el usuario

Procedimiento: para realizar esta tarea se planteo 3 partes fundamentales que debe tener el chatbot

- El api para que se pueda consumir el servicio
- La lógica del despliegue de mensajes de acuerdo con lo que el chatbot identifico como la intención del usuario
- El modelo de inteligencia artificial que se utilizara para entrenar al chatbot, esto con lleva lo siguiente
    - creación de datos
    - análisis de datos

- preprocesamiento de esos datos
- el entrenamiento de esos datos
- el análisis de los resultados del entrenamiento
- toma de decisiones para mejorar el modelo

este proceso se repite hasta tener un modelo lo bastante robusto para poder identificar la intención del cliente con el chatbot

Desarrollo hasta el momento:

- El api ya fue creado y se puede consumir (el servicio se aloja en mi computadora)

```python
if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=5000)
```

```
INFO:     Started server process [20796]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:5000 (Press CTRL+C to quit)
INFO:     189.188.186.156:0 - "POST /message HTTP/1.1" 200 OK
INFO:     207.248.247.20:0 - "POST /message HTTP/1.1" 200 OK
INFO:     91.168.238.3:0 - "POST /message HTTP/1.1" 200 OK
```

- La lógica del despliegue de los mensajes no esta terminada, hasta el momento cubre 5 de los 7 escenarios solicitados

Desarrollados:

- Consultar saldo
- Cancelar tarjeta
- Información sobre como consultar saldo
- Información sobre como cancelar tarjeta
- Información sobre prestamos

```
def handle_data_request(message, intent):
```

```
else:
    if intent == "consultar_saldo":
        collected_data['intent'] = intent
        response_message = "Para la consulta de saldo requerimos nos proporcione los 16 dígitos de su tarjeta."
        waiting_for_digits = True

    elif intent == "cancelar_tarjeta":
        collected_data['intent'] = intent
        response_message = "Para cancelar su tarjeta requerimos nos proporcione los 16 dígitos de su tarjeta."
        waiting_for_digits = True

    elif intent == "informe_saldo":
        response_message = 'Para consultar tu saldo escribe /dame mi saldo/, ten a la mano los datos de tu tarjeta para poder consultar tu saldo

    elif intent == "informe_tarjeta":
        response_message = 'Para poder cancelar tu tarjeta escribe /cancelar tarjeta/, ten a la mano los datos de tu tarjeta para poder consulta

    elif intent == "info_prestamos":
        collected_data['intent'] = intent
        response_message = 'Claro, manejamos los siguientes tipos de préstamos: Pyme, Personal, Escolar, Medico. Sobre qué tipo de préstamo dese
        waiting_for_loan_type = True

    else:
        response_message = f"La intención detectada es {intent}."
```

En el desarrollo de la IA para el chatbot ya se desarrollo el modelo pero se siguen implementando mejoras, ya que algunas intenciones las clasifica mal o se confunde entre intenciones, a continuación una tabla de la precisión de cada una de las intenciones hasta el momento

	precision	recall	f1-score	support
consultar_saldo	0.79	0.92	0.85	12
cancelar_tarjeta	0.90	0.92	0.91	60
informe_tarjeta	0.91	0.91	0.91	55
informe_saldo	1.00	0.79	0.88	14
info_prestamos	1.00	1.00	1.00	18
accuracy			0.91	159
macro avg	0.92	0.91	0.91	159
weighted avg	0.92	0.91	0.91	159

F1-score es la media general entre mas alta mejor pero no sobre pasar el 0.95 por que eso indica un mal entrenamiento del modelo

Se esta trabajando en un mejor preprocesamiento, en un aumento de datos y en una mejor selección de parámetros para el modelo.

```python
# Función de preprocesamiento
def preprocess(text):
    # Eliminar acentos
    nfkd_form = unicodedata.normalize('NFKD', text)
    text = "".join([c for c in nfkd_form if not unicodedata.combining(c)])
    # Convertir a minúsculas
    text = text.lower()
    # Quitar signos de puntuación
    text = text.translate(str.maketrans('', '', punctuation))
    # Tokenizar y lematizar
    doc = nlp(text)
    tokens = [token.lemma_ for token in doc if token.lemma_ not in stop_words and token.lemma_ not in punctuation]
    return ' '.join(tokens)

# Datos de entrenamiento
data = [...

# Aumentar datos mediante técnicas de aumento de datos
def augment_data(text, num_variations):
    words = text.split()
    augmented_texts = []
    for _ in range(num_variations):
        random.shuffle(words)
```

```python
def augment_data(text, num_variations):
    words = text.split()
    augmented_texts = []
    for _ in range(num_variations):
        random.shuffle(words)
        augmented_texts.append(' '.join(words))
    return augmented_texts

# Aplicar aumento de datos a clases con menor rendimiento
augmented_data = []
for text, label in data:
    augmented_data.append((text, label))
    if label in ["cancelar_tarjeta", "informe_tarjeta"]:
        augmented_texts = augment_data(text, 3)  # Generar 3 variaciones
        for aug_text in augmented_texts:
            augmented_data.append((aug_text, label))

# Separar textos y etiquetas
texts, labels = zip(*augmented_data)

# Aplicar preprocesamiento
texts = [preprocess(text) for text in texts]

# Crear el vectorizador y el modelo SVM
vectorizer = TfidfVectorizer()

# Convertir etiquetas a índices
label_map = {"consultar_saldo": 0, "cancelar_tarjeta": 1, "informe_tarjeta": 2, "informe_saldo": 3, "info_prestamos": 4}
y = [label_map[label] for label in labels]

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(texts, y, test_size=0.2, random_state=42)
```

```python
X_train, X_test, y_train, y_test = train_test_split(texts, y, test_size=0.2, random_state=42

# Crear pipeline de vectorizador y clasificador con SVM
pipeline = make_pipeline(vectorizer, SVC(probability=True, class_weight='balanced'))

# Parámetros para la búsqueda en cuadrícula (Grid Search)
param_grid = {
    'svc__C': [0.1, 1, 10],
    'svc__kernel': ['linear', 'rbf']
}

# Búsqueda en cuadrícula con validación cruzada
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy')

# Entrenar el modelo
grid_search.fit(X_train, y_train)

# Evaluar el modelo
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# Imprimir la matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Imprimir el reporte de clasificación
class_report = classification_report(y_test, y_pred, target_names=label_map.keys())
print("\nClassification Report:")
print(class_report)

# Guardar el vectorizador y el modelo entrenado
with open('tfidf_vectorizer.pkl', 'wb') as f:
    pickle.dump(vectorizer, f)
```

Conclusión:

Ya se tiene mas de la mitad del proyecto desarrollado al modelo le falta principalmente mas datos agregar las otras 2 opciones de consulta y generar y entrenar el modelo con los otros 2 tipos de consulta o intención