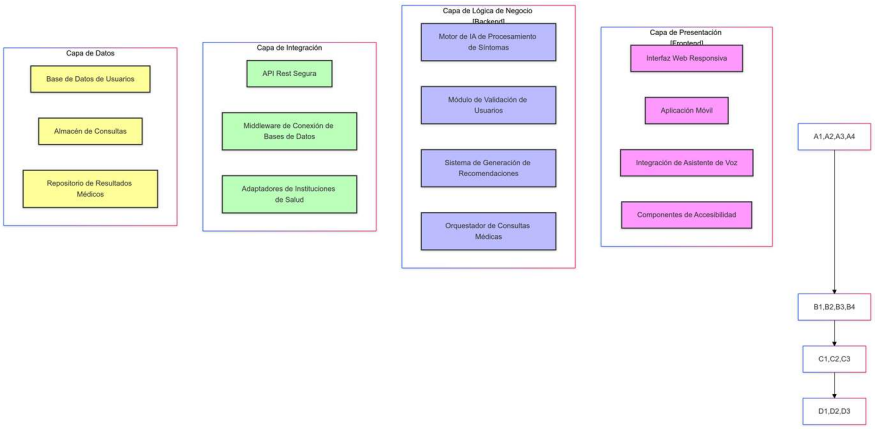
DISEÑO DE BX

Durante el desarrollo de Baymax, el equipo se enfocó en crear una arquitectura robusta, segura y flexible. Nuestro objetivo principal era construir un sistema que no solo fuera técnicamente sólido, sino que también garantizara la privacidad y la experiencia del usuario.

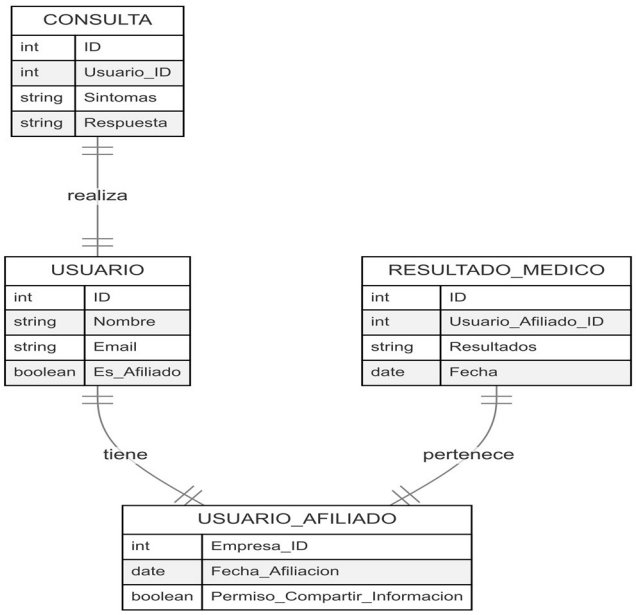
Arquitectura de Capas:



La arquitectura por capas nos permitió modularizar cada componente, facilitando futuras mejoras y mantenimiento. El modelo de datos se diseñó pensando en la flexibilidad, permitiendo adaptarse a diferentes escenarios médicos.

La seguridad fue un pilar fundamental. Implementamos múltiples capas de protección, desde la encriptación de datos hasta tokens de acceso granulares. La integración con sistemas externos se planificó para ser dinámica y segura.

Modelo de Datos



IMPLEMENTACION DE UN METODO DE SEGURIDAD PARA BX

Durante la implementación de Baymax, desarrollamos un módulo denominado SeguridadBaymax utilizando Python como lenguaje de programación. Este módulo fue diseñado para garantizar la protección de la información sensible y la autenticación segura dentro del sistema. Detallamos aquí su estructura y funcionamiento:

1. Bibliotecas Importadas:
    - *cryptography.fernet*: La utilizamos para implementar cifrado simétrico. Este tipo de cifrado aseguraba la protección de los datos médicos y personales de los usuarios, permitiendo encriptar y desencriptar información de forma segura con una clave secreta.
    - *authlib*: Sirvió para integrar OAuth, un estándar abierto que permitía delegar la autenticación de usuarios de manera segura a través de terceros.
    - *flask_jwt_extended*: Facilitó la gestión de tokens JWT (JSON Web Tokens) para autenticar usuarios y manejar permisos dentro de la aplicación.
2. Clase SeguridadBaymax:
   Diseñamos esta clase como un contenedor para las funcionalidades relacionadas con la seguridad. Incluía generación de claves de cifrado, gestión de tokens y validación de consentimiento.
    - Inicialización:
      Al inicializar la clase (__init__), generamos una clave de cifrado única usando *Fernet.generate_key()* y configuramos el sistema de OAuth y JWT. Esto aseguraba que el sistema estuviera listo para proteger datos y autenticar usuarios desde el inicio.
    - Método encriptar_dato_sensible:
      Creamos este método para proteger información sensible, como datos médicos. Se utilizó el cifrado simétrico con la suite de *Fernet*. Al recibir un dato en texto, el método lo convertía en formato encriptado (ciphertext), asegurando que solo pudiera ser desencriptado con la clave secreta generada previamente.
    - Método generar_token_acceso:
      Este método gestionaba la creación de tokens JWT, asignando un nivel de permiso granular a cada usuario según su rol y necesidades. Incluimos *claims* personalizados como el ID del usuario, nivel de acceso y tipos de consulta permitidos. Esto ayudó a implementar un control detallado sobre qué funcionalidades podía acceder cada usuario dentro del sistema.
    - Método validar_consentimiento:
      Implementamos este método para verificar que los usuarios hubieran otorgado consentimiento informado antes de acceder a las funcionalidades de Baymax. Validaba que los usuarios aceptaran términos y condiciones, confirmaran la política de privacidad y cumplieran con la mayoría de edad. Este enfoque garantizaba conformidad con normativas éticas y legales.
3. Instancia de Configuración:
   Al final del código, creamos una instancia de la clase SeguridadBaymax, que centralizó las configuraciones de seguridad del sistema. Esto facilitó su integración con otras partes del proyecto.

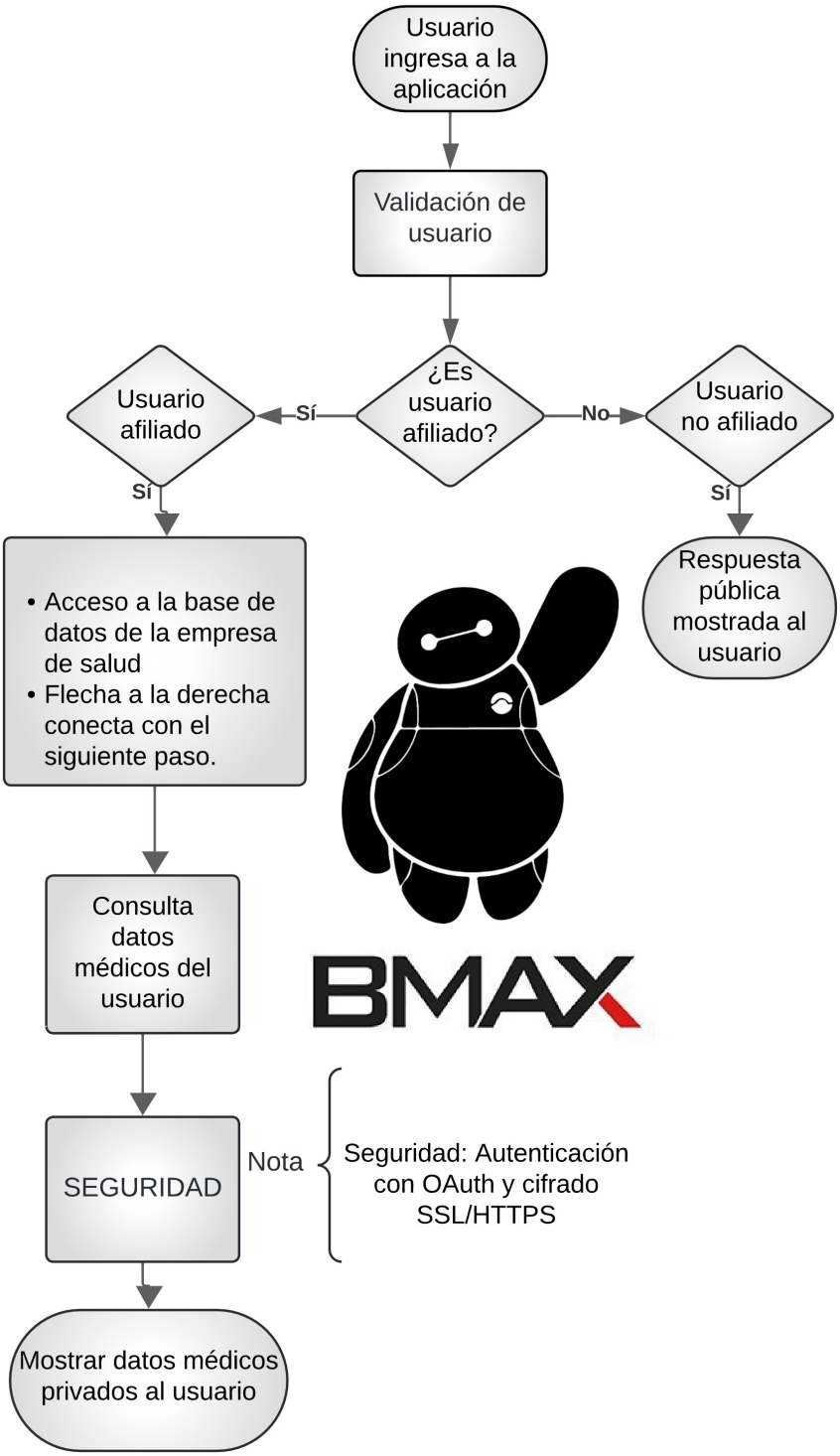
```python
from cryptography.fernet import Fernet
from authlib.integrations.flask_client import OAuth
from flask_jwt_extended import JWTManager, create_access_token

class SeguridadBaymax:
    def __init__(self):
        # Generación de claves de encriptación
        self.key = Fernet.generate_key()
        self.cipher_suite = Fernet(self.key)

        # Configuración de OAuth
        self.oauth = OAuth()
        self.jwt_manager = JWTManager()

    def encriptar_dato_sensible(self, dato):
        """
        Método para encriptar información médica sensible
        Implementamos cifrado simétrico para protección de extremo a extremo
        return self.cipher_suite.encrypt(dato.encode())

    def generar_token_acceso(self, usuario, nivel_permiso):
        """
        Generación de tokens con niveles de acceso granulares
        Permitiendo control fino de permisos
        """
        claims = {
            "usuario_id": usuario.id,
            "nivel_acceso": nivel_permiso,
            "tipo_consulta": ["informativa", "medica"]
        }
        return create_access_token(identity=usuario.id, additional_claims=claims)

    def validar_consentimiento(self, usuario):
        """
        Verificación de consentimiento informado
        Implementamos validación en múltiples niveles
```

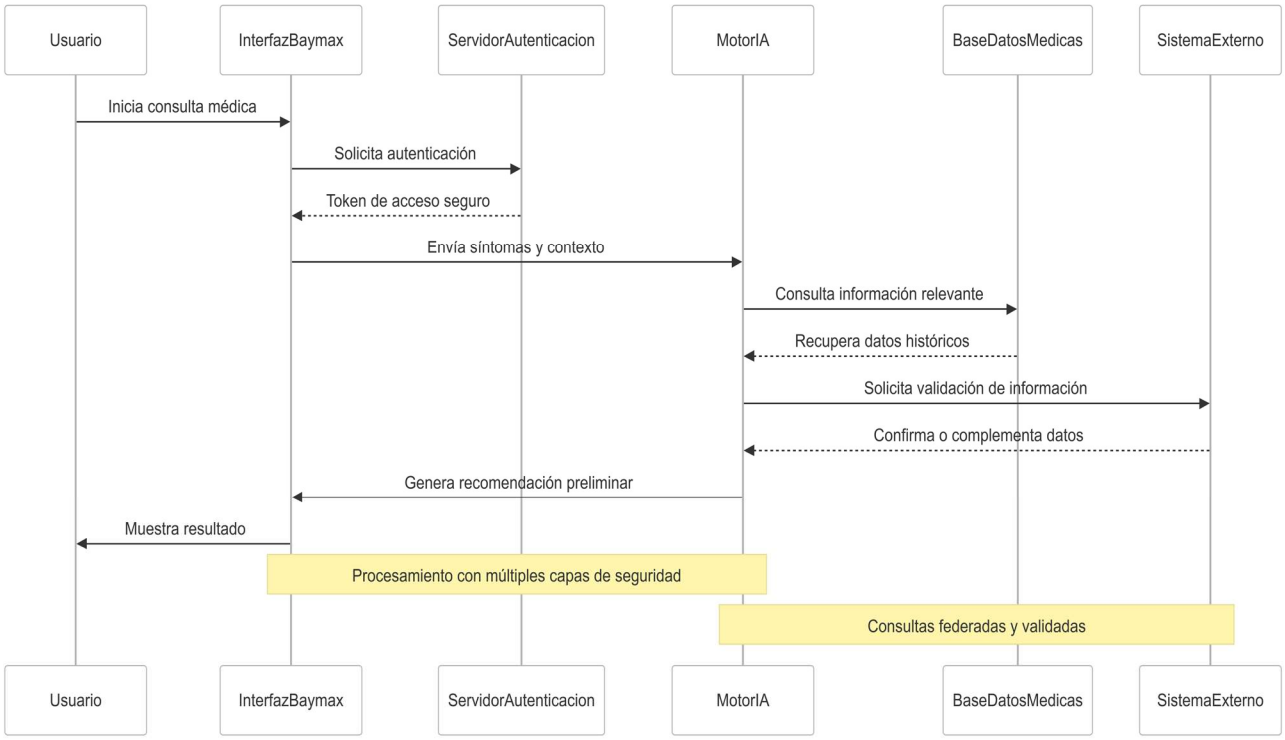
Modelo direccionamiento

Entidad	Atributo	Descripción	Tipo de Dato
Usuario	ID	Identificador único del usuario	Integer
	Nombre	Nombre completo del usuario	String
	Email	Correo electrónico del usuario	String
	Es_Afiliado	Indica si el usuario está afiliado a una empresa de salud	Boolean
Usuario_Afiliado	Empresa_ID	Identificador de la empresa de salud a la que está afiliado	Integer (FK)
	Fecha_Afiliación	Fecha de afiliación del usuario	Date
	Permiso_Compartir_I nformación	Indica si el usuario permite compartir sus datos médicos	Boolean
Empresa_Salud	ID	Identificador único de la empresa	Integer
	Nombre	Nombre de la empresa de salud	String
	Base_Datos	URL o dirección de la base de datos de la empresa	String
	Normativas_Privacida d	Normativas de privacidad que rigen el uso de los datos	String
Consulta	ID	Identificador único de la consulta	Integer
	Usuario_ID	Identificador del usuario que realizó la consulta	Integer (FK)

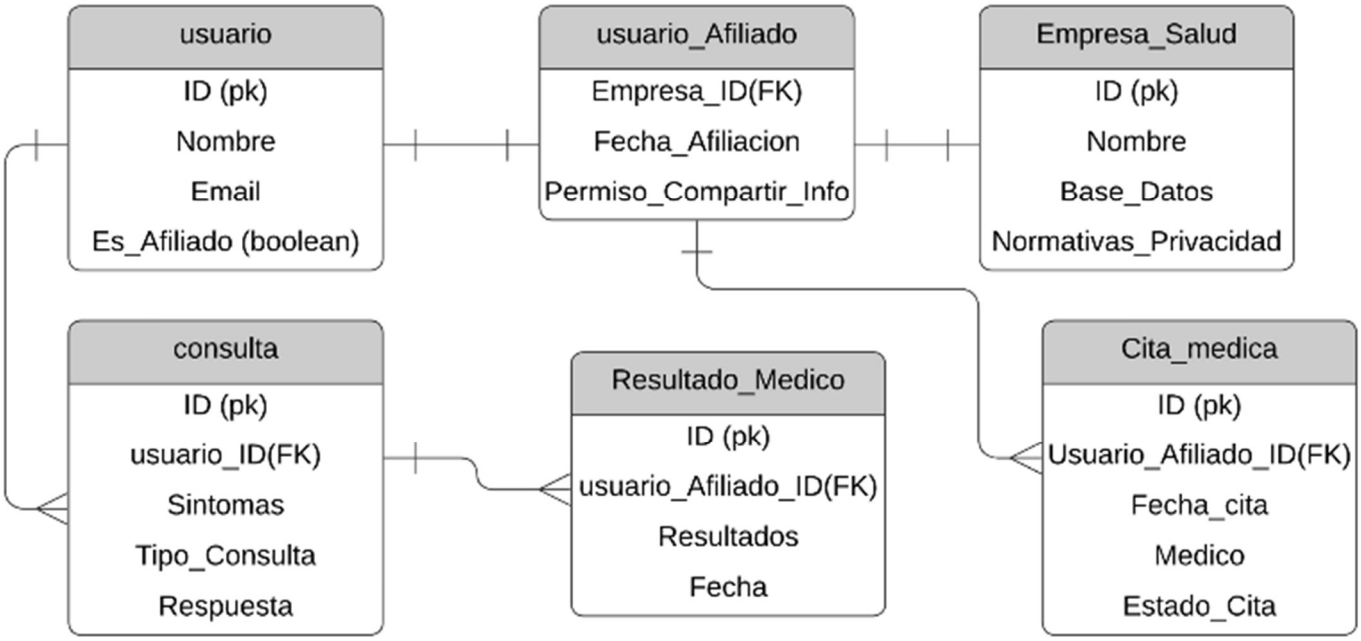
	Sintomas	Detalles sobre los síntomas ingresados por el usuario	String
Entidad	Atributo	Descripción	Tipo de Dato
	Tipo_Consulta	Tipo de consulta: informativa o detallada	Enum (informativa, detallada)
	Respuesta	Respuesta generada por Baymax	String
Resultado_Médico	ID	Identificador del resultado médico	Integer
	Usuario_Afiliado_ID	Identificador del usuario afiliado al que pertenece el resultado	Integer (FK)
	Resultados	Detalles del resultado médico	String
	Fecha	Fecha del resultado médico	Date
Cita_Médica	ID	Identificador único de la cita médica	Integer
	Usuario_Afiliado_ID	Identificador del usuario afiliado que programó la cita	Integer (FK)
	Fecha_Cita	Fecha de la cita médica	Date
	Médico	Nombre del médico asignado a la cita	String
	Estado_Cita	Estado de la cita (confirmada, pendiente, cancelada)	Enum (confirmada, pendiente, cancelada)

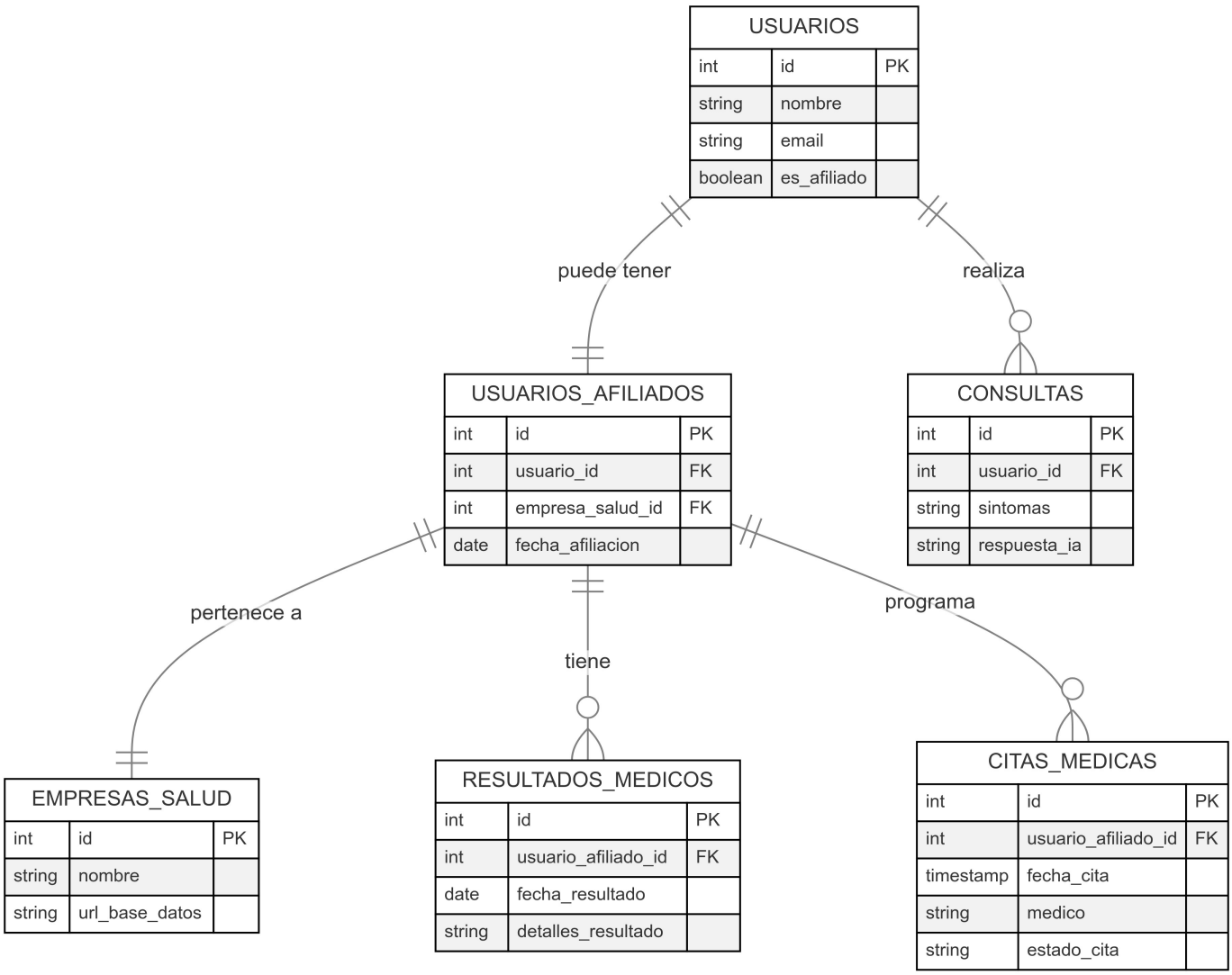
DIAGRAMA DE FLUJO
BMAX





MODELO DE ENTIDAD RELACIÓN DEL PROYECTO





1. **Diseño Centrado en Privacidad**
    - Separamos claramente datos de usuarios generales y afiliados
    - Uso de campos booleanos para control de permisos
    - Flexibilidad con tipos de datos como JSONB para resultados médicos
2. **Consideraciones Técnicas**
    - Uso de PostgreSQL por su robustez en manejo de datos sensibles
    - Índices para optimizar consultas
    - Relaciones definidas con restricciones de integridad referencial
3. **Flexibilidad y Escalabilidad**
    - Estructura que permite manejar diferentes tipos de usuarios
    - Campos genéricos que permiten adaptación a diferentes escenarios médicos

Aspectos Destacados:

- Control granular de permisos
- Almacenamiento seguro de información
- Cumplimiento de normativas de privacidad
- Modelo extensible para futuras necesidades