



PROYECTO SGE

2ª EVALUACIÓN

CFGS Desarrollo de Aplicaciones
Multiplataforma
Informática y Comunicaciones

Citas médicas API

Año: < 2025>

Fecha de presentación: (fecha de presentación)

Nombre y Apellidos: Adrián Alonso Pérez
Email: adrianalonso200@gmail.com

Contenido

1	Introducción.....	4
2	Estado del arte	4
2.1	Arquitectura de microservicios.....	4
2.2	API	4
2.3	Estructura de una API	4
3	Descripción general del proyecto	5
3.1	Objetivos.....	5
3.2	Entorno de trabajo.....	5
4	Documentación técnica	6
4.1	Análisis del sistema	6
4.1.1	Usuario	6
4.1.2	Medico	8
4.1.3	Cita	10
4.2	Diseño de la base de datos	11
4.3	Implementación.....	13
4.3.1	Exceptions.py	13
4.3.2	Models.py	14
4.3.3	Schemas.py	15
4.3.4	Security.py	15
4.3.5	Utiles.py	16
4.3.6	Database.py	16
4.3.7	CitasMedicas.db.....	17
4.3.8	Cita.py	18
4.3.9	Médico.py	19
4.3.10	Usuario.py	22

4.4	Pruebas	23
4.5	Despliegue de la aplicación.....	25
5	Manuales	26
5.1	Manual de usuario:	26
5.2	Manual de instalación.....	29
6	Conclusiones y posibles ampliaciones	29
7	Bibliografía	30

1 Introducción

El proyecto trata de la creación de un api para gestionar citas médicas, utilizando FastApi para su creación desarrollándolo con Python, utilizando SQLite como base de datos y con seguridad de JWT.

2 Estado del arte

2.1 Arquitectura de microservicios

Es un desarrollo de software, en el que una aplicación se divide en varios servicios pequeños independientes. Cada microservicio hace una funcionalidad específica y se comunica con otros microservicios como por APIs.

2.2 API

Conjunto de reglas y definiciones que permite que se comuniquen entre aplicaciones. Definen los métodos y las estructuras de datos que las aplicaciones pueden usar para interactuar.

2.3 Estructura de una API

El protocolo más normal suele ser el http que comunica al cliente y servidor.

Métodos http, se dividen en 4; Get (obtiene datos), Post (Crea nuevos datos), Put (Edita un dato existente) y Delete (Elimina un dato).

Partes de una URL, se divide en; Protocolo (es la seguridad "https:"), Dominio (El servidor donde se encuentra la API "citasmedicas.com"), Recurso (Sería como un subapartado del servidor "/usuarios/") y Parametro (En caso de que fuera necesario se identifica "1" en este caso por su id).
<https://citasmedicas.com/usuarios/1>

Formas de crear una API; Por flask (FrameWork ligero y flexible, fácil de aprender y expandir) o FastAPI la cual es la utilizada en este proyecto ya que tiene un frameWork moderno, rápido y optimizado para APIS con Python (lenguaje utilizado para hacerla api)

3 Descripción general del proyecto

3.1 Objetivos

El objetivo principal sería saber qué, y cómo funciona una API, para que se pueda obtener información de ella y utilizarla en un proyecto.

3.2 Entorno de trabajo

- Visual Studio Code: Como IDE.
- Python: Es el lenguaje utilizado para hacer la API.
- FastAPI: Es el frameWork para desarrollar la API.
- SQLite: Utilizado para la base de datos.
- PostMan: Para realizar las pruebas con autenticación.
- Swagger: Para la documentación y para más pruebas.

4 Documentación técnica

4.1 Análisis del sistema

4.1.1 Usuario

Obtener usuarios:

```
@router.get("/")
def getUsuarios(db: Session = Depends(get_db)):
    """
    Obtiene todos los usuarios
    """
    data = db.query(models.Usuario).all()
    resultado = [
        {
            "id": item.id,
            "nombre": item.nombre,
            "email": item.email,
            "contraseña": item.contrasena,
            "rol": item.rol,
        }
        for item in data
    ]
    return resultado
```

Obtiene todos los usuarios de la base de datos y devuelve su información con la estructura indicada.

Crear usuario:

```
@router.post("/registrar")
def registerUsuario(usuario: Usuario, db: Session = Depends(get_db)):
    """
    Crear un usuario con los atributos indicados
    """
    hashed_password = hashearContrasena(usuario.contrasena)
    newUser = models.Usuario(
        nombre=usuario.nombre,
        email=usuario.email,
        contrasena=hashed_password,
        rol=usuario.rol,
    )
    db.add(newUser)
    db.commit()
    db.refresh(newUser)
    return {"Respuesta": "Usuario creado"}
```

Crea un usuario por los atributos recibidos y manda un mensaje de usuario creado.

Login:

```
@router.post("/login")
def loginUser(login_data: Login, db: Session = Depends(get_db)):
    """
    Inicia sesion con un usuario existente, este te devolvera un token el cual tienes
    que poner en la autenticación
    """
    usuario = db.query(models.Usuario).filter(models.Usuario.email == login_data.email).first()

    if usuario and comprobarContrasena(login_data.contrasena, usuario.contrasena):
        token = crear_jwt(usuario.email)
        return {"access_token": token, "token_type": "bearer"}

    raise HTTPException(status_code=401, detail="Credenciales incorrectas")
```

Esto es la autenticación JWT el cual si el usuario y la contraseña existen te devuelve un token, el cual se pone en el postman.

4.1.2 Medico

Obtener médicos:

```
@router.get("/")
def getMedicos(db:Session=Depends(get_db)):
    """
    Obtener todos los medicos
    """
    data = db.query(models.Medico).all()

    resultado = [
        {
            "id": item.id,
            "nombre ": item.nombre,
            "especialidad": item.especialidad,
            "horario": item.horario,
        }
        for item in data
    ]
    return resultado
```

Obtiene todos los médicos de la base de datos y devuelve su información con la estructura indicada.

Buscar por especialidad:

```
@router.get("/buscar")
def getMedicosPorEspecialidad(
    especialidad: str = Query(..., description="Especialidad a buscar", min_length=1),
    db: Session = Depends(get_db)
):
    """
    Filtra los medicos por la especialidad indicada. (cardiologia)
    """
    data = db.query(models.Medico).filter(
        func.trim(func.lower(models.Medico.especialidad)) == func.lower(func.trim(especialidad))
    ).all()

    resultado = [
        (variable) item: Medico
        {
            "id": item.id,
            "nombre": item.nombre,
            "especialidad": item.especialidad,
            "horario": item.horario,
        }
        for item in data
    ]
    return resultado
```

Obtiene todos los médicos de la base de datos filtrado por la especialidad indicada y devuelve su información con la estructura indicada.

Crear medico:


```
@router.post("/crearMedico")
def crearMedico(medico: Medico, db: Session = Depends(get_db)):
    """
    Crea un medico por los atributos indicados
    """
    newMedico = models.Medico(
        nombre=medico.nombre,
        especialidad=medico.especialidad,
        horario=medico.horario,
    )
    db.add(newMedico)
    db.commit()
    db.refresh(newMedico)
    return {"Respuesta": "Medico creado"}
```

Crea un medico con los atributos recibidos y envía un mensaje de médico creado.

Actualizar médico:

```
@router.put("/actualizar/{id}", summary="Actualizar medico")
def actualizarMedico(id: int, medico: schemas.Medico, db: Session = Depends(get_db), current_user: str = Depends(get_current_user)):
    """
    Actualiza el medico indicado por el id, con los atributos recibidos.
    Necesita autenticación JWT
    """
    medico_db = db.query(models.Medico).filter(models.Medico.id == id).first()
    if medico_db is None:
        raise NotFoundException(detail="No existe ese medico")

    medico_db.nombre = medico.nombre
    medico_db.especialidad = medico.especialidad
    medico_db.horario = medico.horario

    db.commit()
    db.refresh(medico_db)

    return {"mensaje": "Medico actualizado"}
```

Actualiza el médico por el id indicado y los atributos recibidos, envía un mensaje de médico actualizado. Esta implementado la seguridad JWT.

Eliminar médico:

```
@router.delete("/eliminar/{id}", summary="Eliminar un medico")
def eliminarMedico(id: int, db: Session = Depends(get_db), current_user: str = Depends(get_current_user)):
    """
    Elimina el medico indicado por el id.
    Necesita autenticación JWT
    """
    medico = db.query(models.Medico).filter(models.Medico.id == id).first()
    if medico is None:
        raise NotFoundException(detail="No existe ese medico")

    db.delete(medico)
    db.commit()
    return {"mensaje": "Medico eliminado"}
```

Elimina el médico especificado por el id y envía un mensaje de médico eliminado. Esta implementado la seguridad JWT.

4.1.3 Cita

Obtener citas:

```
@router.get("/")
def getCitas(db: Session = Depends(get_db)):
    """
    Obtiene todas las citas.
    """
    data = db.query(models.Cita).all()

    resultado = [
        {
            "id": item.id,
            "fecha": item.fecha,
            "estado": item.estado,
            "usuario_id": item.usuario_id,
            "medico_id": item.medico_id,
        }
        for item in data
    ]
    return resultado
```

Obtiene todas las citas de la base de datos y devuelve su información con la estructura indicada.

Obtener citas por fecha:

```
@router.get("/fecha/{fecha}", summary="Obtener citas por fecha")
def getCitasPorFecha(fecha: date, db: Session = Depends(get_db)):
    """
    Obtiene todas las citas sobre la fecha indicada.
    Ej (2025-02-21)
    """
    data = db.query(models.Cita).filter(models.Cita.fecha == fecha).all()

    if not data:
        raise NotFoundException(detail="No se encontraron citas para esta fecha")

    resultado = [
        {
            "id": item.id,
            "fecha": item.fecha,
            "estado": item.estado,
            "usuario_id": item.usuario_id,
            "medico_id": item.medico_id,
        }
        for item in data
    ]
    return resultado
```

Obtiene todas las citas filtrada por la fecha que reciba.

Crear cita:

```
@router.post("/crearCita")
def crearCita(cita: Cita, db: Session = Depends(get_db)):
    """
    Crea una cita con los atributos recibidos.
    """
    newCita = models.Cita(
        fecha=cita.fecha,
        estado=cita.estado,
        usuario_id=cita.usuario_id,
        medico_id=cita.medico_id,
    )
    db.add(newCita)
    db.commit()
    db.refresh(newCita)
    return {"Respuesta": "Cita creada"}
```

Crea todas las citas con los atributos recibidos y muestra un mensaje de cita creada.

Actualizar cita:

```
@router.put("/actualizar/{id}", summary="Actualizar cita")
def actualizarCita(id: int, cita: schemas.Cita, db: Session = Depends(get_db), current_user: str = Depends(get_current_user)):
    """
    Actualiza la cita indicada por el id, con los atributos recibidos.
    Necesita autenticación JWT
    """
    cita_db = db.query(models.Cita).filter(models.Cita.id == id).first()
    if cita_db is None:
        raise NotFoundException(detail="No existe esa cita")

    cita_db.fecha = cita.fecha
    cita_db.estado = cita.estado
    cita_db.usuario_id = cita.usuario_id
    cita_db.medico_id = cita.medico_id

    db.commit()
    db.refresh(cita_db)

    return {"mensaje": "Cita actualizada"}
```

Actualiza la cita por el id y con los atributos recibidos, envía un mensaje de cita actualizada. Tiene seguridad JWT.

Eliminar cita:

```
@router.delete("/eliminar/{id}", summary="Eliminar una cita")
def eliminarCita(id: int, db: Session = Depends(get_db), current_user: str = Depends(get_current_user)):
    """
    Elimina la cita indicada por el id.
    Necesita autenticación JWT
    """
    cita = db.query(models.Cita).filter(models.Cita.id == id).first()
    if cita is None:
        raise NotFoundException(detail="No existe la cita")
    db.delete(cita)
    db.commit()
    return {"mensaje": "Cita eliminada"}
```

Elimina la cita por el id recibido y envía un mensaje de cita eliminada. Tiene seguridad JWT.

4.2 Diseño de la base de datos

La base de datos consta de 3 tablas.

Tabla usuario: Tiene un id auto incrementable, nombre, email que es único, contraseña y rol (paciente, administrador...).

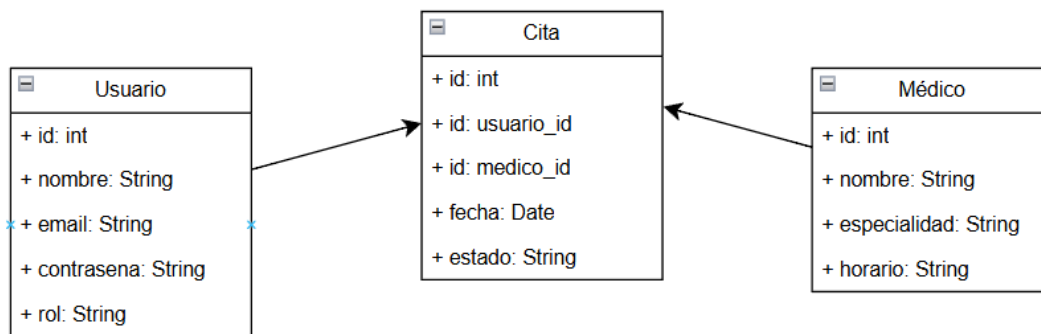
Tabla médico: Tiene un id auto incrementable, nombre, especialidad (cardiología, dermatología...) y horario (mañana, tarde, noche)

Tabla cita: Tiene un id auto incrementable, fecha, estado (pendiente, completado...), usuario_id que sería la relación con usuario y medico_id que es la relación con médico.

Usuario con Cita sería One2Many.

Médico con Cita sería One2Many.

Cita con Médico y con Usuario es Many2One



4.3 Implementación

4.3.1 Exceptions.py

```
from fastapi import HTTPException

class ForbiddenException(HTTPException):
    def __init__(self, detail: str = "Acceso denegado"):
        super().__init__(status_code=403, detail=detail)

class BadRequestException(HTTPException):
    def __init__(self, detail: str = "Solicitud incorrecta"):
        super().__init__(status_code=400, detail=detail)

class InternalServerErrorException(HTTPException):
    def __init__(self, detail="Error interno del servidor"):
        super().__init__(status_code=500, detail=detail)

class NotFoundException(HTTPException):
    def __init__(self, detail: str = "No se encontro"):
        super().__init__(status_code=404, detail=detail)

class UnauthorizedException(HTTPException):
    def __init__(self, detail: str = "Sin autorización"):
        super().__init__(status_code=401, detail=detail)
```

Esta clase tiene todas las dependencias definidas.

4.3.2 Models.py

```
from sqlalchemy import *
from sqlalchemy.orm import relationship
from app.db.database import Base

class Medico(Base):
    __tablename__ = "medicos"
    __table_args__ = {"extend_existing": True}
    id = Column(Integer, primary_key=True, autoincrement=True)
    nombre = Column(String, nullable=False)
    especialidad = Column(String, nullable=False)
    horario = Column(String)

class Cita(Base):
    __tablename__ = "citas"
    __table_args__ = {"extend_existing": True}
    id = Column(Integer, primary_key=True, autoincrement=True)
    usuario_id = Column(Integer, ForeignKey("usuarios.id"))
    medico_id = Column(Integer, ForeignKey("medicos.id"))
    fecha = Column(Date, nullable=False)
    estado = Column(String, default="pendiente")
    usuario = relationship("Usuario")
    medico = relationship("Medico")

class Usuario(Base):
    __tablename__ = "usuarios"
    __table_args__ = {"extend_existing": True}
    id = Column(Integer, primary_key=True, autoincrement=True)
    nombre = Column(String, nullable=False)
    email = Column(String, unique=True, nullable=False)
    contrasena = Column(String, nullable=False)
    rol = Column(String, default="paciente", nullable=False)
```

Aquí se encuentran todas las clases y sus atributos.

```
from datetime import date, datetime
from pydantic import BaseModel

class Usuario(BaseModel):
    nombre: str
    email: str
    contrasena: str
    rol: str = "paciente"

class Login(BaseModel):
    email: str
    contrasena: str

    class Config:
        orm_mode = True

class Medico(BaseModel):
    nombre: str
    especialidad: str
    horario: str
    class Config:
        orm_mode = True

class Cita(BaseModel):
    usuario_id: int
    medico_id: int
    fecha: date
    estado: str = "pendiente"

    class Config:
        orm_mode = True
```

4.3.3 Schemas.py

Esta clase sirve para estructurar como quiero que esten las clases para ver cómo se muestran.

```
from jose import JWTError, jwt
from fastapi import HTTPException, Depends
from datetime import datetime, timedelta
from fastapi.security import OAuth2PasswordBearer

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="usuario/login")

SECRET_KEY = "tu_clave_secreta"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 30

def crear_jwt(email: str):
    expiration = datetime.utcnow() + timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    to_encode = {"sub": email, "exp": expiration}
    encoded_jwt = jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
    return encoded_jwt

def verificar_jwt(token: str) -> str:
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        email: str = payload.get("sub")
        if email is None:
            raise JWTError("Email no encontrado en el token")
        return email
    except JWTError:
        raise HTTPException(status_code=401, detail="Token no válido")

def get_current_user(token: str = Depends(oauth2_scheme)):
    email = verificar_jwt(token)
    return email
```

4.3.4 Security.py

Clase seguridad para el JWT, crear el token comprobarlo.

4.3.5 Utiles.py

```
import hashlib

#Hashear la contraseña
def hashearContrasena(contrasena: str) -> str:
    return hashlib.sha256(contrasena.encode()).hexdigest()

#Comprobar la contraseña hasheada
def comprobarContrasena(contrasena: str, conHasheada: str) -> bool:
    return hashearContrasena(contrasena) == conHasheada
```

Esta clase sirve para encriptar la contraseña y para verificar la contraseña.

4.3.6 Database.py

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, declarative_base

SQLALCHEMY_DATABASE_URL = "sqlite:///./citasmedicas.db"

engine = create_engine(SQLALCHEMY_DATABASE_URL, connect_args={"check_same_thread": False})

SessionLocal = sessionmaker(bind=engine, autocommit=False, autoflush=False)

Base = declarative_base()

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

Clase para la base de datos, la cual utiliza sqlite y se genera automáticamente el archivo .db con las tablas creadas

4.3.7 CitasMedicas.db

```

1 SQLite format 3
2
3
4
5 id INTEGER NOT NULL,
6 usuario_id INTEGER,
7 medico_id INTEGER,
8 fecha DATE NOT NULL,
9 estado VARCHAR,
10 PRIMARY KEY (id),
11 FOREIGN KEY(usuario_id) REFERENCES usuarios (id),
12 FOREIGN KEY(medico_id) REFERENCES medicos (id)
13 )
14 id INTEGER NOT NULL,
15 nombre VARCHAR NOT NULL,
16 especialidad VARCHAR NOT NULL,
17 horario VARCHAR,
18 PRIMARY KEY (id)
19 )
20 CREATE TABLE usuarios (
21 id INTEGER NOT NULL,
22 nombre VARCHAR NOT NULL,
23 email VARCHAR NOT NULL,
24 contrasena VARCHAR NOT NULL,
25 rol VARCHAR NOT NULL,
26 PRIMARY KEY (id),
27 UNIQUE (email)
28 )
29
30
31
32
33
34

```

Se autogenera y por eso hay tantas cosas raras.

4.3.8 Cita.py

```
from fastapi import APIRouter, Depends
from sqlalchemy.orm import Session
from app import schemas
from app.schemas import Cita
from app.db.database import get_db
from app import models
from app.exceptions import NotFoundException
from app.security import get_current_user

router = APIRouter(
    prefix="/cita",
    tags=["cita"]
)

@router.get("/")
def getCitas(db: Session = Depends(get_db)):
    """
    Obtiene todas las citas.
    """
    data = db.query(models.Cita).all()

    resultado = [
        {
            "id": item.id,
            "fecha ": item.fecha,
            "estado": item.estado,
            "usuario_id": item.usuario_id,
            "medico_id": item.medico_id,
        }
        for item in data
    ]
    return resultado

@router.post("/crearCita")
def crearCita(cita: Cita, db: Session = Depends(get_db)):
    """
    Crea una cita con los atributos recibidos.
    """
    newCita = models.Cita(
        fecha=cita.fecha,
        estado=cita.estado,
        usuario_id=cita.usuario_id,
        medico_id=cita.medico_id,
    )
    db.add(newCita)
    db.commit()
    db.refresh(newCita)
    return {"Respuesta": "Cita creada"}

@router.get("/fecha/{fecha}", summary="Obtener citas por fecha")
def getCitasPorFecha(fecha: date, db: Session = Depends(get_db)):
    """
    Obtiene todas las citas sobre la fecha indicada.
    Ej (2025-02-21)
    """
    data = db.query(models.Cita).filter(models.Cita.fecha == fecha).all()

    if not data:
        raise NotFoundException(detail="No se encontraron citas para esta fecha")

    resultado = [
        {
            "id": item.id,
            "fecha": item.fecha,
            "estado": item.estado,
            "usuario_id": item.usuario_id,
            "medico_id": item.medico_id,
        }
        for item in data
    ]
    return resultado
```

Funciones getCitas, y crearCita, citas por fecha e imports.

```
@router.put("/actualizar/{id}", summary="Actualizar cita")
def actualizarCita(id: int, cita: schemas.Cita, db: Session = Depends(get_db), current_user: str = Depends(get_current_user)):
    """
    Actualiza la cita indicada por el id, con los atributos recibidos.
    Necesita autenticación JWT
    """
    cita_db = db.query(models.Cita).filter(models.Cita.id == id).first()
    if cita_db is None:
        raise NotFoundException(detail="No existe esa cita")

    cita_db.fecha = cita.fecha
    cita_db.estado = cita.estado
    cita_db.usuario_id = cita.usuario_id
    cita_db.medico_id = cita.medico_id

    db.commit()
    db.refresh(cita_db)

    return {"mensaje": "Cita actualizada"}

@router.delete("/eliminar/{id}", summary="Eliminar una cita")
def eliminarCita(id: int, db: Session = Depends(get_db), current_user: str = Depends(get_current_user)):
    """
    Elimina la cita indicada por el id.
    Necesita autenticación JWT
    """
    cita = db.query(models.Cita).filter(models.Cita.id == id).first()
    if cita is None:
        raise NotFoundException(detail="No existe la cita")
    db.delete(cita)
    db.commit()
    return {"mensaje": "Cita eliminada"}
```

Actualizar y eliminar cita.

4.3.9 Médico.py

```
from fastapi import APIRouter, Depends, Query
from sqlalchemy import func
from sqlalchemy.orm import Session
from app import schemas
from app.schemas import Medico
from app.db.database import get_db
from app import models
from app.exceptions import NotFoundException
from app.security import get_current_user

router = APIRouter(
    prefix="/medico",
    tags=["Medico"]
)

@router.get("/")
def getMedicos(db: Session = Depends(get_db)):
    """
    Obtener todos los medicos
    """
    data = db.query(models.Medico).all()

    resultado = [
        {
            "id": item.id,
            "nombre": item.nombre,
            "especialidad": item.especialidad,
            "horario": item.horario,
        }
        for item in data
    ]
    return resultado
```

Get medicos e imports.

```
@router.get("/buscar")
def getMedicosPorEspecialidad(
    especialidad: str = Query(..., description="Especialidad a buscar", min_length=1),
    db: Session = Depends(get_db)
):
    """
    Filtra los medicos por la especialidad indicada. (cardiologia)
    """
    data = db.query(models.Medico).filter(
        func.trim(func.lower(models.Medico.especialidad)) == func.lower(func.trim(especialidad))
    ).all()

    resultado = [
        {
            "id": item.id,
            "nombre": item.nombre,
            "especialidad": item.especialidad,
            "horario": item.horario,
        }
        for item in data
    ]
    return resultado

@router.post("/crearMedico")
def crearMedico(medico: Medico, db: Session = Depends(get_db)):
    """
    Crea un medico por los atributos indicados
    """
    newMedico = models.Medico(
        nombre=medico.nombre,
        especialidad=medico.especialidad,
        horario=medico.horario,
    )
    db.add(newMedico)
    db.commit()
    db.refresh(newMedico)
    return {"Respuesta": "Medico creado"}
```

Buscar por especialidad y crear médico.

```
@router.put("/actualizar/{id}", summary="Actualizar medico")
def actualizarMedico(id: int, medico: schemas.Medico, db: Session = Depends(get_db), current_user: str = Depends(get_current_user)):
    """
    Actualiza el medico indicado por el id, con los atributos recibidos.
    Necesita autenticación JWT
    """
    medico_db = db.query(models.Medico).filter(models.Medico.id == id).first()
    if medico_db is None:
        raise NotFoundException(detail="No existe ese medico")

    medico_db.nombre = medico.nombre
    medico_db.especialidad = medico.especialidad
    medico_db.horario = medico.horario

    db.commit()
    db.refresh(medico_db)

    return {"mensaje": "Medico actualizado"}

@router.delete("/eliminar/{id}", summary="Eliminar un medico")
def eliminarMedico(id: int, db: Session = Depends(get_db), current_user: str = Depends(get_current_user)):
    """
    Elimina el medico indicado por el id.
    Necesita autenticación JWT
    """
    medico = db.query(models.Medico).filter(models.Medico.id == id).first()
    if medico is None:
        raise NotFoundException(detail="No existe ese medico")

    db.delete(medico)
    db.commit()
    return {"mensaje": "Medico eliminado"}
```

Actualizar y eliminar médico.

4.3.10 Usuario.py

```
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from app.schemas import Login, Usuario
from app.db.database import get_db
from app import models
from app.utiles import hashearContrasena, comprobarContrasena
from app.security import crear_jwt

router = APIRouter(
    prefix="/usuario",
    tags=["Usuario"]
)

@router.get("/")
def getUsuarios(db: Session = Depends(get_db)):
    """
    Obtiene todos los usuarios
    """
    data = db.query(models.Usuario).all()
    resultado = [
        {
            "id": item.id,
            "nombre": item.nombre,
            "email": item.email,
            "contraseña": item.contrasena,
            "rol": item.rol,
        }
        for item in data
    ]
    return resultado

@router.post("/registrar")
def registerUsuario(usuario: Usuario, db: Session = Depends(get_db)):
    """
    Crear un usuario con los atributos indicados
    """
    hashed_password = hashearContrasena(usuario.contrasena)
    newUser = models.Usuario(
        nombre=usuario.nombre,
        email=usuario.email,
        contrasena=hashed_password,
        rol=usuario.rol,
    )
    db.add(newUser)
    db.commit()
    db.refresh(newUser)
    return {"Respuesta": "Usuario creado"}
```

Imports, obtener usuarios y crear usuario.

```
@router.post("/login")
def loginUser(login_data: Login, db: Session = Depends(get_db)):
    """
    Inicia sesion con un usuario existente, este te devolvera un token el cual tienes
    que poner en la autentificación
    """
    usuario = db.query(models.Usuario).filter(models.Usuario.email == login_data.email).first()

    if usuario and comprobarContrasena(login_data.contrasena, usuario.contrasena):
        token = crear_jwt(usuario.email)
        return {"access_token": token, "token_type": "bearer"}

    raise HTTPException(status_code=401, detail="Credenciales incorrectas")
```

Login para el token.

4.4 Pruebas

Crear un usuario y obtener el token. Vamos a Swagger. Rellenamos el body con los datos que queramos. Le damos a ejecutar y nos da una respuesta 200.

POST

/usuario/registrar

Registerusuario

^

Crear un usuario con los atributos indicados

Parameters

Cancel

Reset

No parameters

Request body

required

application/json

▼

```
{
  "nombre": "Adrian",
  "email": "adrian2@gmail.com",
  "contrasena": "123",
  "rol": "paciente"
}
```

Execute

Request URL

http://127.0.0.1:8000/usuario/registrar

Server response

Code	Details
200	<div><div>Response body</div><div><pre>{ "Respuesta": "Usuario creado" }</pre></div><div><div>Download</div></div></div> <div><div>Response headers</div><div><pre>content-length: 30 content-type: application/json date: Tue, 11 Feb 2025 10:52:57 GMT server: uvicorn</pre></div></div>

Responses

Code	Description	Links
------	-------------	-------

Ahora hago un get para comprobar que se haya creado el usuario: (El tercero). Como se ve la contraseña esta encriptada.

200

Response body

```
[
  {
    "id": 1,
    "nombre": "adrian",
    "email": "adrian@gmail.com",
    "contraseña": "a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3",
    "rol": "string"
  },
  {
    "id": 2,
    "nombre": "adrian",
    "email": "a",
    "contraseña": "ca978112ca1bbdcafac231b39a23dc4da786eff8147c4e72b9807785afee48bb",
    "rol": "string"
  },
  {
    "id": 3,
    "nombre": "Adrian",
    "email": "adrian2@gmail.com",
    "contraseña": "a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3",
    "rol": "paciente"
  }
]
```

Response headers

```
content-length: 439
content-type: application/json
date: Tue, 11 Feb 2025 10:53:37 GMT
server: uvicorn
```

Responses

Code	Description	Links
------	-------------	-------

La segunda prueba seria obtener el token: (Rellenamos con el mail y contrasena)

POST /usuario/login Loginuser

Inicia sesion con un usuario existente, este te devolvera un token el cual tienes que poner en la autentificación

Parameters Cancel Reset

No parameters

Request body required application/json

```
{
  "email": "adrian2@gmail.com",
  "contrasena": "123"
}
```

Execute

Code

Details

200

Response body

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhZHZpYW4yQGdtYWlsLmVubSIzImV4cCI6MTczOTI3MzEyOX0.89nmeCVPbTI34-u2P4_09pk9kX7VokY1unniMmA7CP0",
  "token_type": "bearer"
}
```

Download

Response headers

```
content-length: 181
content-type: application/json
date: Tue, 11 Feb 2025 10:55:29 GMT
server: uvicorn
```

Responses

Code

Description

Links

Y como podemos ver devuelve el token.

4.5 Despliegue de la aplicación

Yo lo desplégue en local utilizando uvicorn con uvicorn:app.

```
PS D:\FastAPI\proyecto_citas_medicas> uvicorn main:app
C:\Users\adrian.aloper.4\AppData\Local\Programs\Python\Python312\Lib\site-packages\pydantic\_internal\_c
fig.py:345: UserWarning: Valid config keys have changed in V2:
* 'orm_mode' has been renamed to 'from_attributes'
INFO:      Started server process [21168]
INFO:      Started server process [21168]
INFO:      Waiting for application startup.
INFO:      Application startup complete.
```

5 Manuales

5.1 Manual de usuario:

The screenshot shows the Swagger UI for the 'API de citas medicas' (version 0.1.0, OAS 3.1). The interface is in Spanish and lists the following endpoints:

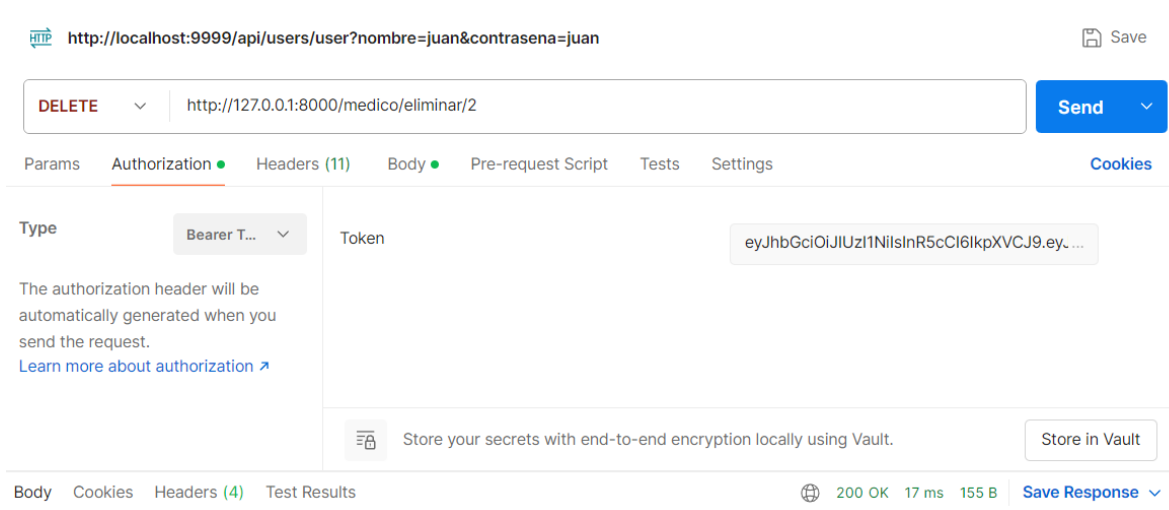
- Usuario**
 - GET /usuario/ Getusuarios
 - POST /usuario/regarstrar Registerusuario
 - POST /usuario/login Loginuser
- Cita**
 - GET /cita/ Getcitas
 - POST /cita/crearCita Crearcita
 - PUT /cita/actualizar/{id} Actualizar cita
 - DELETE /cita/eliminar/{id} Eliminar una cita
- Medico**
 - GET /medico/ Getmedicos
 - GET /medico/buscar Getmedicosporespecialidad
 - POST /medico/crearMedico Crearmedico
 - PUT /medico/actualizar/{id} Actualizar medico
 - DELETE /medico/eliminar/{id} Eliminar un medico

Endpoints marked with a lock icon (e.g., PUT, DELETE) indicate they are protected and require authentication.

Esto sería lo primero que se ve al ir al swagger. Hay apartados que ya se podrían utilizar, pero los que tienen un candado habría que loggarse y obtener el token.

Como ya indiqué como se hace para obtener el token ahora voy a enseñar como introducir el token para que funcione.

Abrimos postman, en authorization elegimos el tipo de Bearer Token y ya podremos acceder a las funciones con JWT.



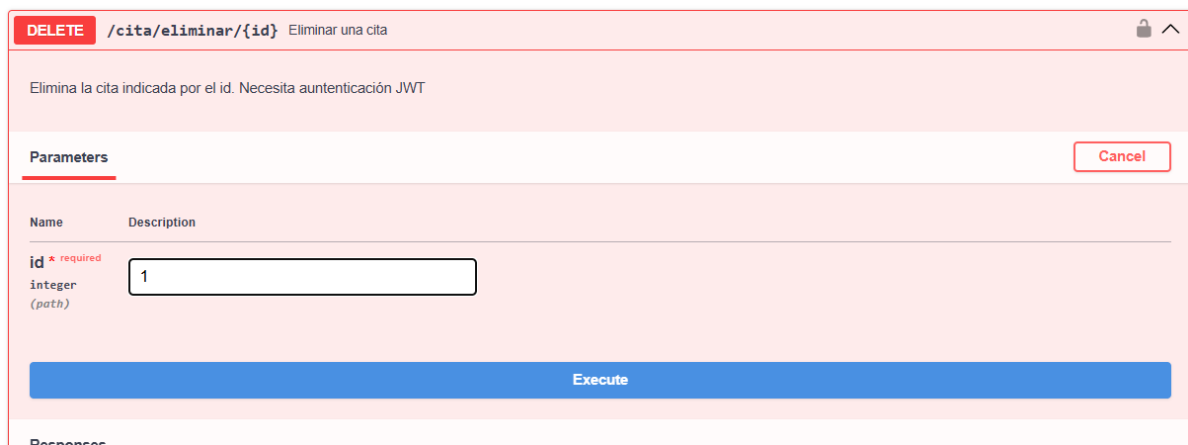
The screenshot shows the Postman interface with the URL `http://localhost:9999/api/users/user?nombre=juan&contrasena=juan`. The **Authorization** tab is selected, and the **Type** is set to **Bearer Token**. The **Token** field contains the JWT token `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ...`. Below the token field, there is a checkbox for **Store your secrets with end-to-end encryption locally using Vault** and a **Store in Vault** button. The bottom status bar shows **200 OK**, **17 ms**, and **155 B**.

Ahora voy a hacer un ejemplo, voy a eliminar una cita. Primero voy a hacer un get para ver las citas.



The screenshot shows the **Response body** of a GET request, which is a JSON array of two appointment objects. The first object has `"id": 1`, `"fecha": "2025-02-11"`, `"estado": "pendiente"`, `"usuario_id": 1`, and `"medico_id": 1`. The second object has `"id": 2`, `"fecha": "2025-02-21"`, `"estado": "pendiente"`, `"usuario_id": 1`, and `"medico_id": 2`. The **Response headers** section shows `content-length: 163`, `content-type: application/json`, `date: Tue, 11 Feb 2025 11:07:52 GMT`, and `server: uvicorn`.

Ahora vamos a eliminar cita y elegimos el id 1 y le damos a execute.



The screenshot shows the **DELETE** request configuration in Postman. The URL is `/cita/eliminar/{id}` with the description **Eliminar una cita**. The **Parameters** section shows a required integer parameter `id` with the value `1`. The **Execute** button is visible at the bottom.

Tiene que dar un error ya que aún no tenemos la autenticación, pero lo que queremos es la url para ponerla en postman:

Curl

```
curl -X 'DELETE' \
  'http://127.0.0.1:8000/cita/eliminar/1' \
  -H 'accept: application/json'
```

Request URL

<http://127.0.0.1:8000/cita/eliminar/1>

Server response

Code	Details
401 <small>Undocumented</small>	<p>Error: Unauthorized</p> <p>Response body</p> <pre>{ "detail": "Not authenticated" }</pre> <p>Response headers</p> <pre>content-length: 30 content-type: application/json date: Tue, 11 Feb 2025 11:09:16 GMT server: uvicorn www-authenticate: Bearer</pre>

Responses

Code	Description	Links
------	-------------	-------

Ahora en el postman, con el token puesto (dura 30min cada token) y la url, le damos a enviar y nos sale que la cita fue eliminada:

HTTP <http://localhost:9999/api/users/user?nombre=juan&contrasena=juan> Save

DELETE <http://127.0.0.1:8000/cita/eliminar/1> Send

Params Authorization Headers (11) Body Pre-request Script Tests Settings Cookies

Type Bearer T... Token eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ...

The authorization header will be automatically generated when you send the request.
[Learn more about authorization](#)

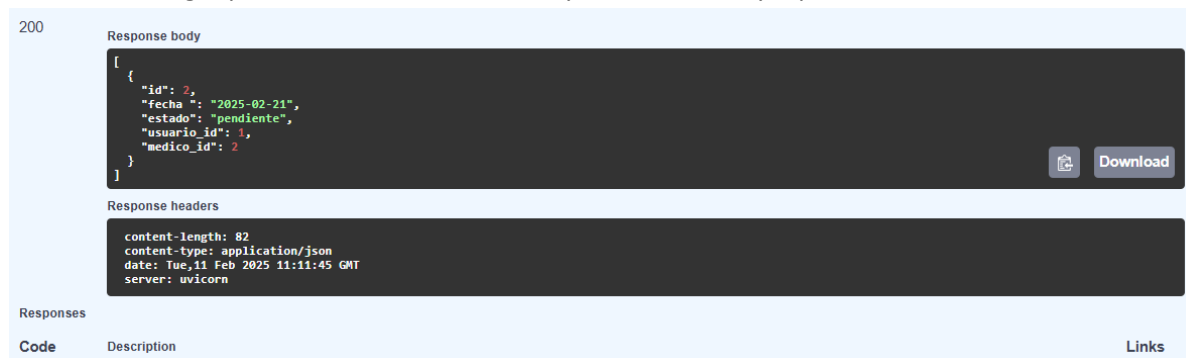
Store your secrets with end-to-end encryption locally using Vault. Store in Vault

Body Cookies Headers (4) Test Results 200 OK 22 ms 153 B Save Response

Pretty Raw Preview Visualize JSON

```
1
2  "mensaje": "Cita eliminada"
3
```

Hacemos otro get para obtener todas las citas y no nos saldría ya que la hemos eliminado.



200

Response body

```
{
  "id": 2,
  "fecha": "2025-02-21",
  "estado": "pendiente",
  "usuario_id": 1,
  "medico_id": 2
}
```

Response headers

```
content-length: 82
content-type: application/json
date: Tue, 11 Feb 2025 11:11:45 GMT
server: uvicorn
```

Responses

Code	Description	Links
------	-------------	-------

5.2 Manual de instalación

Para desplegar la aplicación habría que:

Descargar el proyecto, abrirlo en visual y hacer un **pip install -r .\requirements.txt** para importar las dependencias (hay algunas que no están).

Una vez instalado las dependencias para ejecutar la API hay que ejecutar **uvicorn main:app**.

Al ejecutarlo saldrá una url la cual habría que poner la url/docs para entrar a Swagger.

6 Conclusiones y posibles ampliaciones

En cuanto las dificultades, son el tiempo. Yo personalmente tuve el problema que te comenté del portátil y que no pude hacerlo en la torre ya que me están haciendo reformas por moho.

En cuestión del aprendizaje, sería una buena experiencia, ya que al no tener nada de información me ha permitido investigar y comprender más las APIs.

Una ampliación sería hacer la aplicación móvil u otra en cuestión el usuario sea paciente o administrador tenga acceso a más funciones.

7 Bibliografía

- [FastAPI](#)
- [Pydantic](#)
- Pdf tema 10.
- ChatGpt