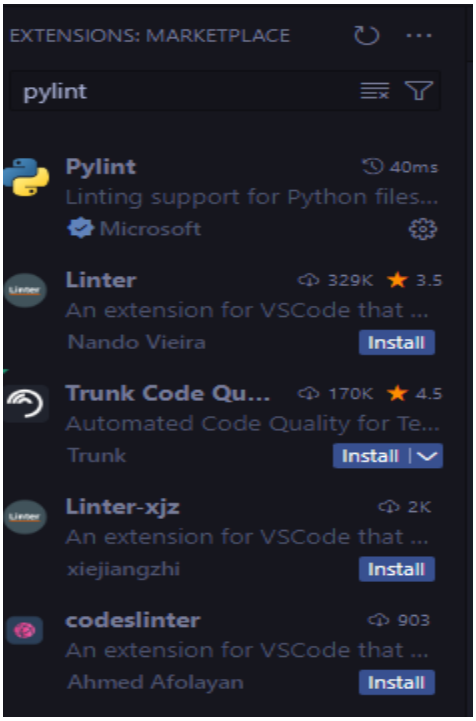
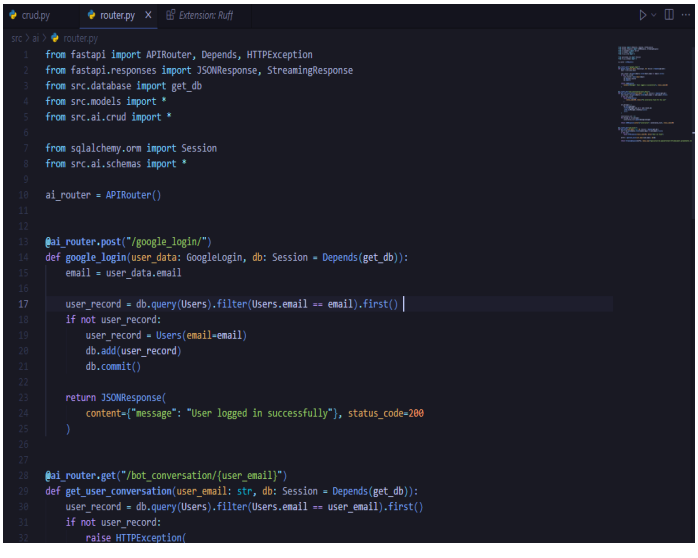
Linter seleccionado: Elegimos Pylint como linter para este proyecto debido a su rigor, flexibilidad y profundidad en el análisis estático del código. A diferencia de otras herramientas, Pylint no solo detecta errores y problemas de estilo, sino que también evalúa la complejidad del código, ayudando a mantener buenas prácticas y facilitar la mantenibilidad. Su amplio conjunto de reglas y capacidad de personalización permiten adaptarlo a los estándares específicos del proyecto. Además, su integración con pyproject.toml y otras configuraciones lo hace una opción robusta para equipos que buscan un código limpio, bien estructurado y alineado con PEP 8.

Implementación linter seleccionado:

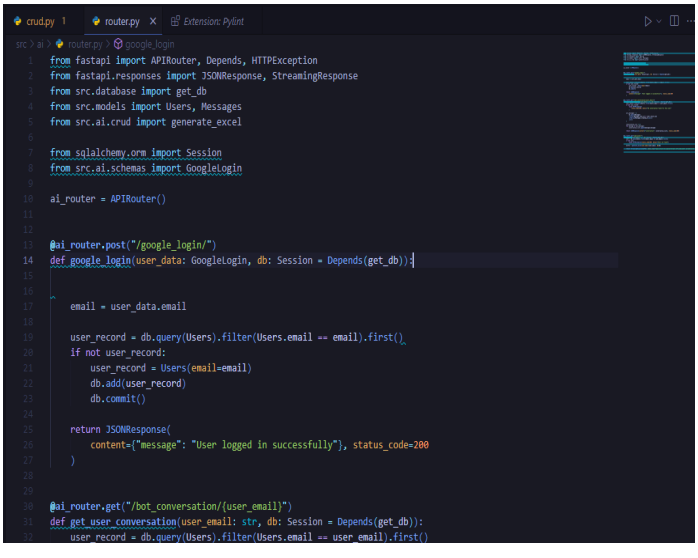
Ya que el linter se encuentra en las extensiones de VScode, bastará con buscar su nombre y posterior a ello le daremos en el botón de instalar.



Una vez instalada la extensión, se nos mostrarán automáticamente los errores en el editor.



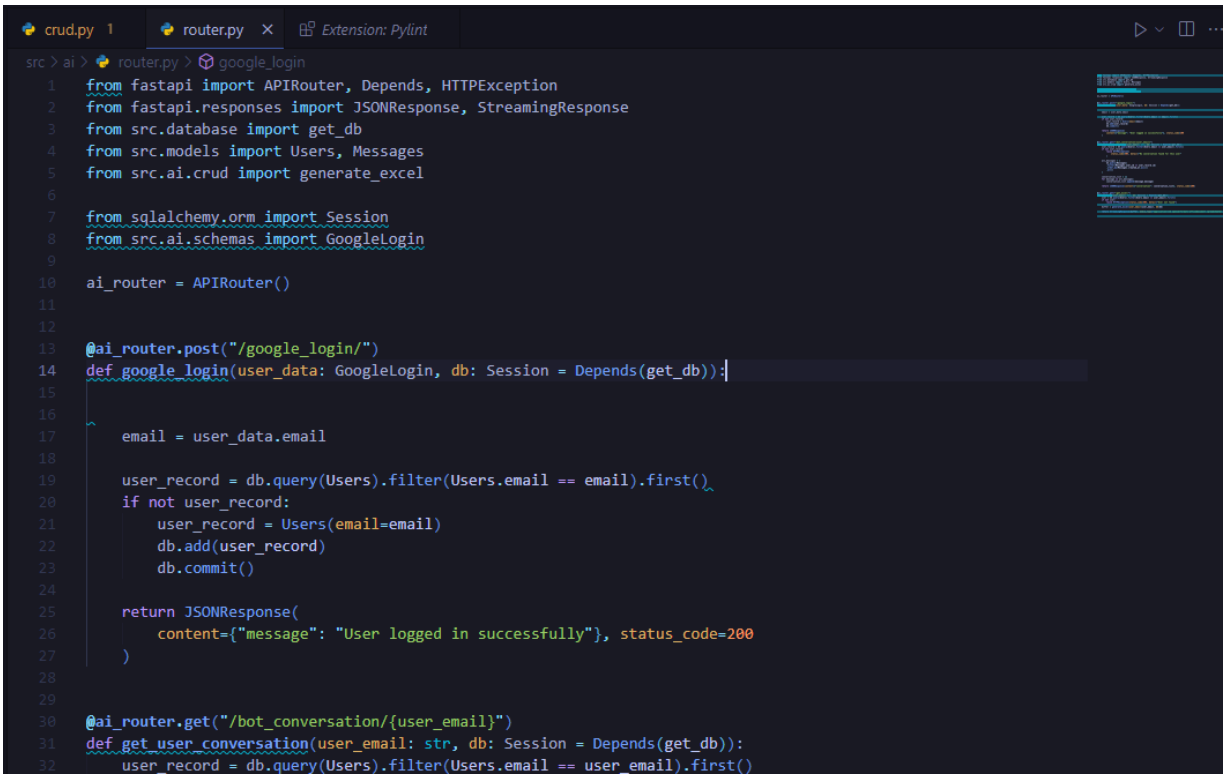
```python
from fastapi import APIRouter, Depends, HTTPException
from fastapi.responses import JSONResponse, StreamingResponse
from src.database import get_db
from src.models import *
from src.ai.crud import *

from sqlalchemy.orm import Session
from src.ai.schemas import *

ai_router = APIRouter()


@ai_router.post("/google_login/")
def google_login(user_data: GoogleLogin, db: Session = Depends(get_db)):
    email = user_data.email

    user_record = db.query(Users).filter(Users.email == email).first()
    if not user_record:
        user_record = Users(email=email)
        db.add(user_record)
        db.commit()

    return JSONResponse(
        content={"message": "User logged in successfully"}, status_code=200
    )


@ai_router.get("/bot_conversation/{user_email}")
def get_user_conversation(user_email: str, db: Session = Depends(get_db)):
    user_record = db.query(Users).filter(Users.email == user_email).first()
    if not user_record:
        raise HTTPException(
```



```python
from fastapi import APIRouter, Depends, HTTPException
from fastapi.responses import JSONResponse, StreamingResponse
from src.database import get_db
from src.models import Users, Messages
from src.ai.crud import generate_excel

from sqlalchemy.orm import Session
from src.ai.schemas import GoogleLogin

ai_router = APIRouter()


@ai_router.post("/google_login/")
def google_login(user_data: GoogleLogin, db: Session = Depends(get_db)):

    email = user_data.email

    user_record = db.query(Users).filter(Users.email == email).first()
    if not user_record:
        user_record = Users(email=email)
        db.add(user_record)
        db.commit()

    return JSONResponse(
        content={"message": "User logged in successfully"}, status_code=200
    )


@ai_router.get("/bot_conversation/{user_email}")
def get_user_conversation(user_email: str, db: Session = Depends(get_db)):
    user_record = db.query(Users).filter(Users.email == user_email).first()
```

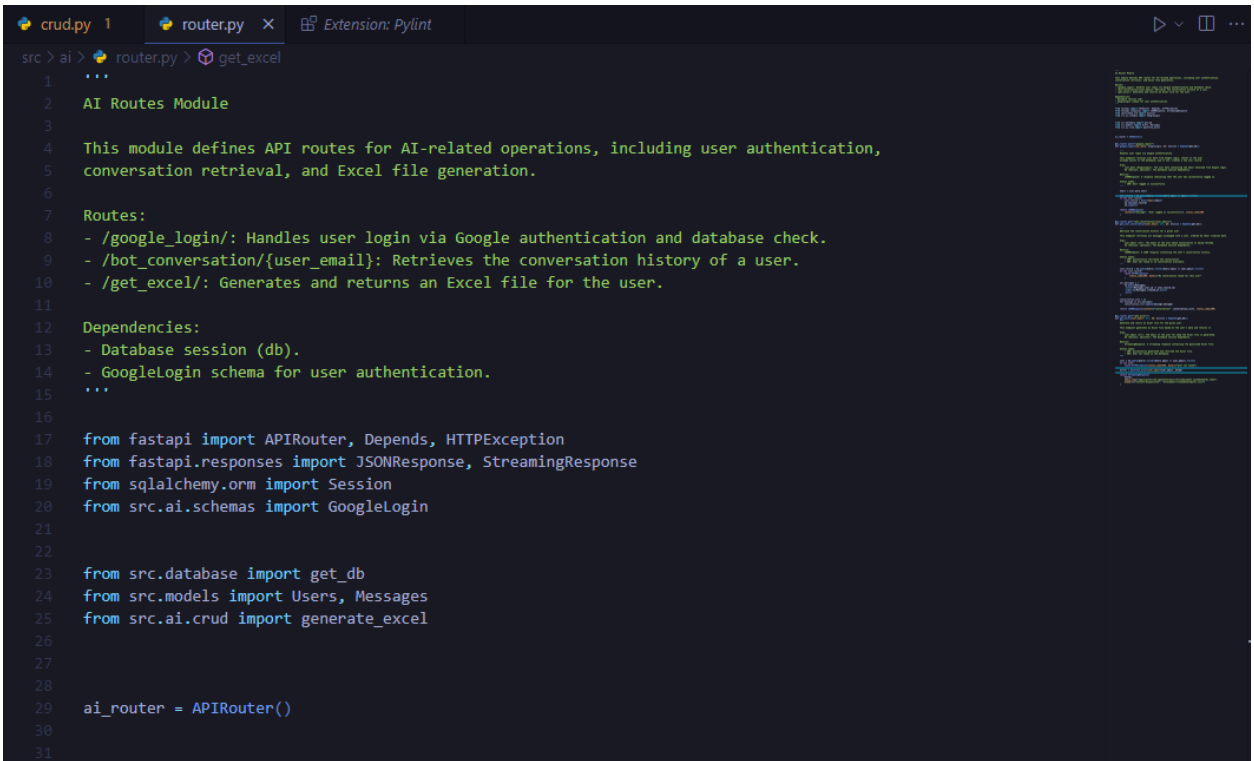
Código antes y después de la implementación:

Antes:

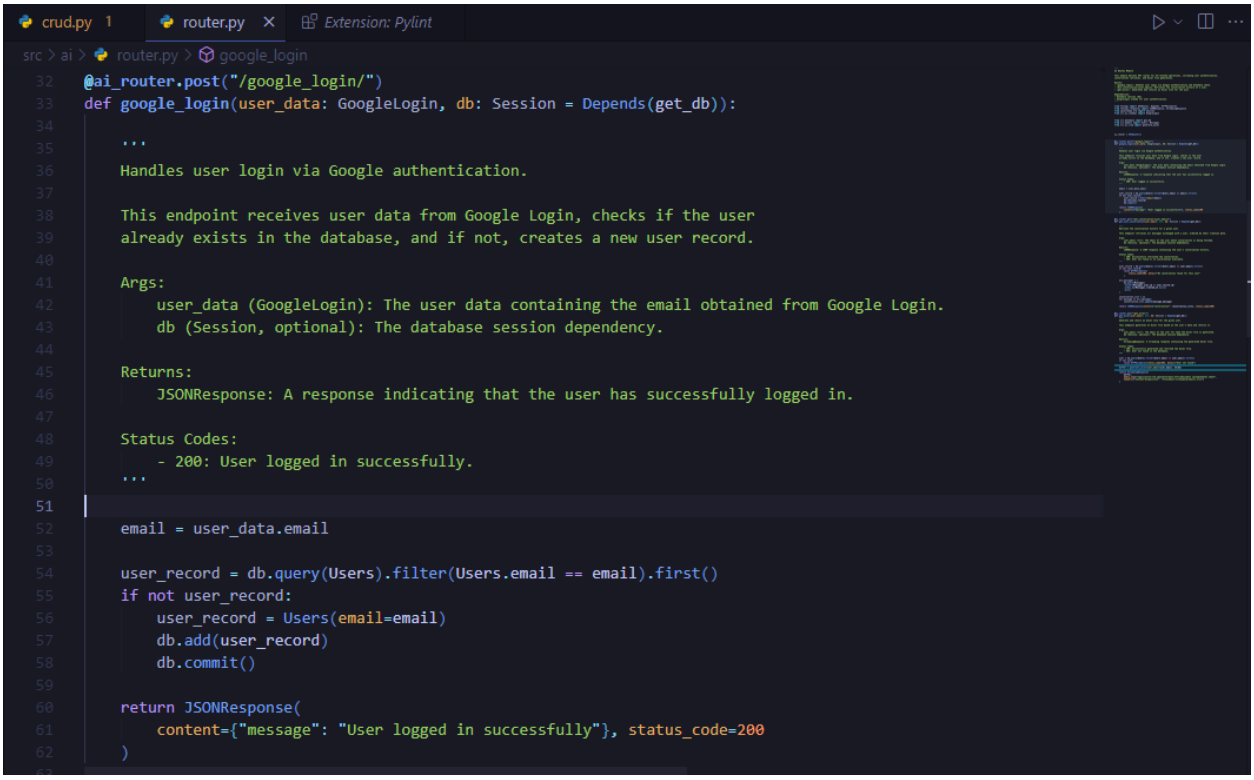


```python
from fastapi import APIRouter, Depends, HTTPException
from fastapi.responses import JSONResponse, StreamingResponse
from src.database import get_db
from src.models import Users, Messages
from src.ai.crud import generate_excel

from sqlalchemy.orm import Session
from src.ai.schemas import GoogleLogin

ai_router = APIRouter()


@ai_router.post("/google_login/")
def google_login(user_data: GoogleLogin, db: Session = Depends(get_db)):


    email = user_data.email

    user_record = db.query(Users).filter(Users.email == email).first()
    if not user_record:
        user_record = Users(email=email)
        db.add(user_record)
        db.commit()

    return JSONResponse(
        content={"message": "User logged in successfully"}, status_code=200
    )


@ai_router.get("/bot_conversation/{user_email}")
def get_user_conversation(user_email: str, db: Session = Depends(get_db)):
    user_record = db.query(Users).filter(Users.email == user_email).first()
```

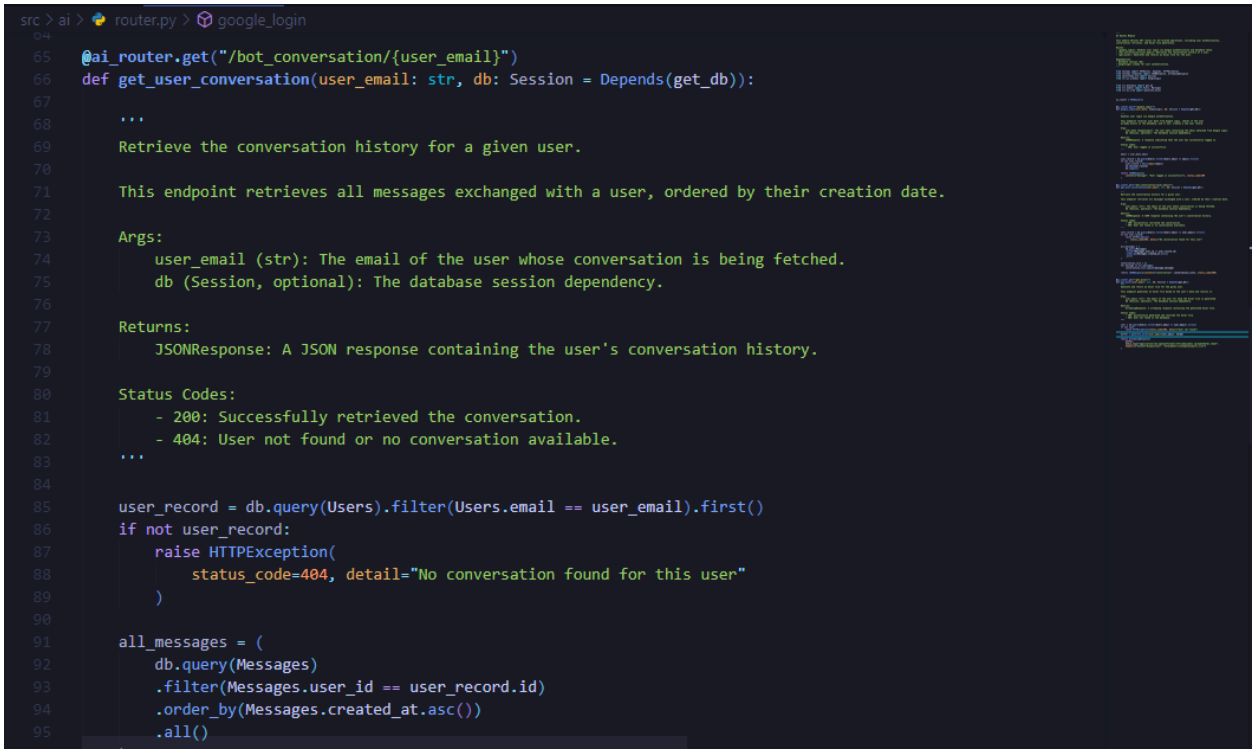
Después:



```python
'''
AI Routes Module

This module defines API routes for AI-related operations, including user authentication,
conversation retrieval, and Excel file generation.

Routes:
- /google_login/: Handles user login via Google authentication and database check.
- /bot_conversation/{user_email}: Retrieves the conversation history of a user.
- /get_excel/: Generates and returns an Excel file for the user.

Dependencies:
- Database session (db).
- GoogleLogin schema for user authentication.
'''

from fastapi import APIRouter, Depends, HTTPException
from fastapi.responses import JSONResponse, StreamingResponse
from sqlalchemy.orm import Session
from src.ai.schemas import GoogleLogin


from src.database import get_db
from src.models import Users, Messages
from src.ai.crud import generate_excel



ai_router = APIRouter()
```



```python
@ai_router.post("/google_login/")
def google_login(user_data: GoogleLogin, db: Session = Depends(get_db)):

    '''
    Handles user login via Google authentication.

    This endpoint receives user data from Google Login, checks if the user
    already exists in the database, and if not, creates a new user record.

    Args:
        user_data (GoogleLogin): The user data containing the email obtained from Google Login.
        db (Session, optional): The database session dependency.

    Returns:
        JSONResponse: A response indicating that the user has successfully logged in.

    Status Codes:
        - 200: User logged in successfully.
    '''

    email = user_data.email

    user_record = db.query(Users).filter(Users.email == email).first()
    if not user_record:
        user_record = Users(email=email)
        db.add(user_record)
        db.commit()

    return JSONResponse(
        content={"message": "User logged in successfully"}, status_code=200
    )
```



```python
@ai_router.get("/bot_conversation/{user_email}")
def get_user_conversation(user_email: str, db: Session = Depends(get_db)):

    '''
    Retrieve the conversation history for a given user.

    This endpoint retrieves all messages exchanged with a user, ordered by their creation date.

    Args:
        user_email (str): The email of the user whose conversation is being fetched.
        db (Session, optional): The database session dependency.

    Returns:
        JSONResponse: A JSON response containing the user's conversation history.

    Status Codes:
        - 200: Successfully retrieved the conversation.
        - 404: User not found or no conversation available.
    '''

    user_record = db.query(Users).filter(Users.email == user_email).first()
    if not user_record:
        raise HTTPException(
            status_code=404, detail="No conversation found for this user"
        )

    all_messages = (
        db.query(Messages)
        .filter(Messages.user_id == user_record.id)
        .order_by(Messages.created_at.asc())
        .all()
```

```python
@ai_router.get("/get_excel/")
def get_excel(user_email: str, db: Session = Depends(get_db)):
    """
    Generate and return an Excel file for the given user.

    This endpoint generates an Excel file based on the user's data and returns it.

    Args:
        user_email (str): The email of the user for whom the Excel file is generated.
        db (Session, optional): The database session dependency.

    Returns:
        StreamingResponse: A streaming response containing the generated Excel file.

    Status Codes:
        - 200: Successfully generated and returned the Excel file.
        - 404: User not found in the database.
    """

    user = db.query(Users).filter(Users.email == user_email).first()
    if not user:
        raise HTTPException(status_code=404, detail="User not found")
    buffer = generate_excel(user_email=user_email, db=db)
    return StreamingResponse(
        buffer,
        media_type="application/vnd.openxmlformats-officedocument.spreadsheetml.sheet",
        headers={"Content-Disposition": "attachment;filename=products.xlsx"}
    )
```

Las principales mejoras indicadas por el linter fueron el orden de las importaciones, el importar únicamente las funciones y modelos que se usan en el código, el colocar una breve descripción del módulo en formato docstring al inicio de todo el archivo, el agregar docstrings informativos en cada ruta para mejor entendimiento de estas y el no tener más de 180 caracteres por línea, respetando los lineamientos estipulados por PEP8.

Narración por cada integrante

Fabian espitia sotelo

El código de mis compañeros es bastante fácil de leer y sigue una estructura bastante limpia. Yo me encargo principalmente de la parte del backend y, aunque no trabajó directamente con el frontend, la integración de estos dos está quedando bien. Los comentarios están presentes y son bastante útiles, lo cual hace que entender el código no sea una tarea complicada. Además, todo parece alinearse con los estándares que definimos al principio, así que no hay sorpresas. Si

tuviera que modificar algo, no dudaría mucho, ya que el código está bien segmentado y es fácil de seguir. No creo que haya ningún lío, siempre que sigamos la misma estructura."

Jose Julian Pira Naranjo

Yo no soy tan experto en algunas partes del proyecto como algunos de mis compañeros, pero he estado trabajando más en la integración y en hacer que todo funcione bien a nivel general. Cuando revisé el código, me pareció fácil de seguir en cuanto a los componentes principales, aunque algunas veces me costó entender el propósito de ciertas funciones, pero creo que es por que no estoy tan familiarizado con algunos detalles. En general, sí me parece que cumple con los estándares, aunque tal vez algunos nombres de variables podrían ser más claros. Si tuviera que cambiar algo, no sé si tendría problemas, pero podría tomarme algo de tiempo porque no conozco completamente todos los aspectos de cada módulo.

Javier Esteban Martinez Giron

Al principio, me sentí un poco perdido con el código de mis compañeros. A veces es difícil seguir el flujo porque hay muchas cosas con las que no estoy familiarizado como lo eran cosas de SQLAlchemy que no manejo bien, pero estoy aprendiendo. Creo que mi compañero que se encargó del frontend hizo un muy buen trabajo en organizar su código, aunque me cuesta un poco seguir los detalles técnicos. Sin embargo, el frontend con Next.js y NextAuth me pareció más intuitivo.Creo que en general el código es ordenado, aunque hay algunas funciones largas que podrían ser más fáciles de leer.

Sergio Nicolas Rey Mora

He tenido la oportunidad de revisar todo el código del equipo y, en general, está bastante bien organizado. Mi objetivo es asegurar me de que todo siga un patrón coherente y de que todos estemos en la misma página, y creo que hemos logrado esto. El código de mis compañeros es limpio, bien estructurado y fácil de leer en su mayoría.En backend, FastAPI es una buena elección por su velocidad y compatibilidad con OpenAPI. En frontend, NextAuth facilita la autenticación. Con respecto a modificar el código, no creo que haya complicaciones. Si todos seguimos los principios que hemos acordado, no deberíamos tener problemas en el futuro. Creo que cumple los estándares que designamos y que, con algunos ajustes menores, será fácil de mantener y escalar.