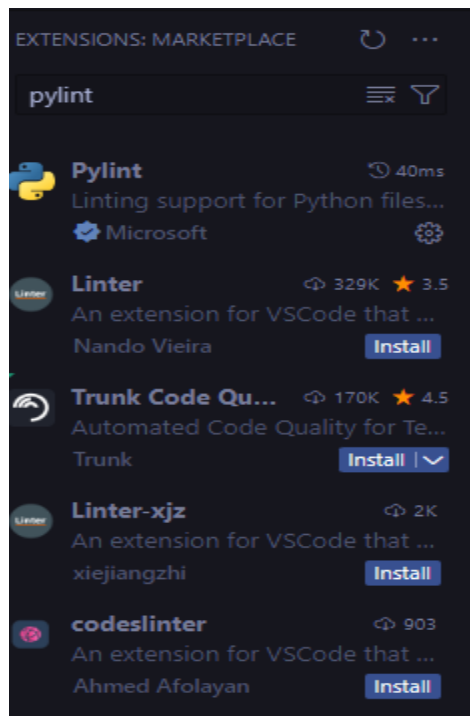


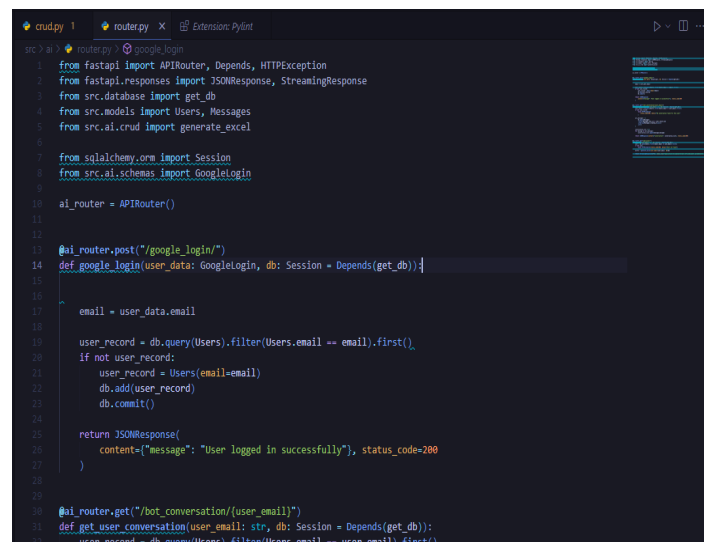
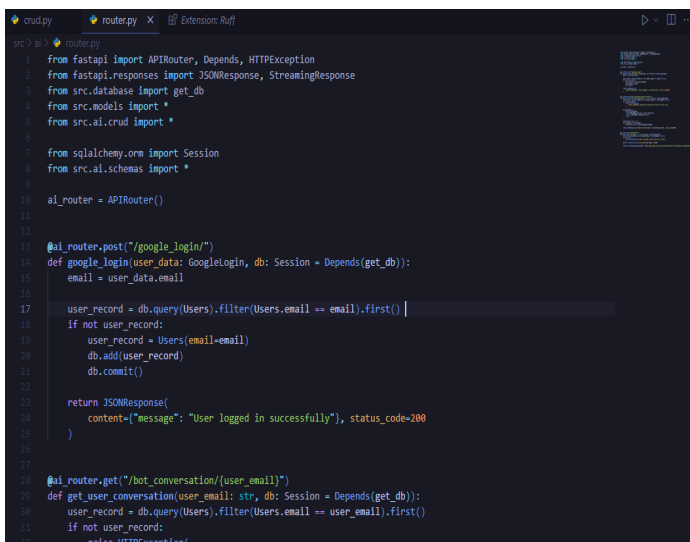
Linter seleccionado: Elegimos Pylint como linter para este proyecto debido a su rigor, flexibilidad y profundidad en el análisis estático del código. A diferencia de otras herramientas, Pylint no solo detecta errores y problemas de estilo, sino que también evalúa la complejidad del código, ayudando a 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.



Código antes y después de la implementación:

Antes:

```
crud.py 1 router.py x Extension: Pylint
src > ai > router.py > google_login
1 from fastapi import APIRouter, Depends, HTTPException
2 from fastapi.responses import JSONResponse, StreamingResponse
3 from src.database import get_db
4 from src.models import Users, Messages
5 from src.ai.crud import generate_excel
6
7 from sqlalchemy.orm import Session
8 from src.ai.schemas import GoogleLogin
9
10 ai_router = APIRouter()
11
12
13 @ai_router.post("/google_login/")
14 def google_login(user_data: GoogleLogin, db: Session = Depends(get_db)):
15
16     email = user_data.email
17
18     user_record = db.query(Users).filter(Users.email == email).first()
19     if not user_record:
20         user_record = Users(email=email)
21         db.add(user_record)
22         db.commit()
23
24     return JSONResponse(
25         content={"message": "User logged in successfully"}, status_code=200
26     )
27
28
29
30 @ai_router.get("/bot_conversation/{user_email}")
31 def get_user_conversation(user_email: str, db: Session = Depends(get_db)):
32     user_record = db.query(Users).filter(Users.email == user_email).first()
```

Después:

```
crud.py 1 router.py x Extension: Pylint
src > ai > router.py > get_excel
1 ...
2 AI Routes Module
3
4 This module defines API routes for AI-related operations, including user authentication,
5 conversation retrieval, and Excel file generation.
6
7 Routes:
8 - /google_login/: Handles user login via Google authentication and database check.
9 - /bot_conversation/{user_email}: Retrieves the conversation history of a user.
10 - /get_excel/: Generates and returns an Excel file for the user.
11
12 Dependencies:
13 - Database session (db).
14 - GoogleLogin schema for user authentication.
15 ...
16
17 from fastapi import APIRouter, Depends, HTTPException
18 from fastapi.responses import JSONResponse, StreamingResponse
19 from sqlalchemy.orm import Session
20 from src.ai.schemas import GoogleLogin
21
22
23 from src.database import get_db
24 from src.models import Users, Messages
25 from src.ai.crud import generate_excel
26
27
28
29 ai_router = APIRouter()
30
31
```

```
crud.py 1 router.py x Extension: Pylint
src > ai > router.py > google_login
32 @ai_router.post("/google_login/")
33 def google_login(user_data: GoogleLogin, db: Session = Depends(get_db)):
34
35     ...
36     Handles user login via Google authentication.
37
38     This endpoint receives user data from Google Login, checks if the user
39     already exists in the database, and if not, creates a new user record.
40
41     Args:
42         user_data (GoogleLogin): The user data containing the email obtained from Google Login.
43         db (Session, optional): The database session dependency.
44
45     Returns:
46         JSONResponse: A response indicating that the user has successfully logged in.
47
48     Status Codes:
49         - 200: User logged in successfully.
50     ...
51
52     email = user_data.email
53
54     user_record = db.query(Users).filter(Users.email == email).first()
55     if not user_record:
56         user_record = Users(email=email)
57         db.add(user_record)
58         db.commit()
59
60     return JSONResponse(
61         content={"message": "User logged in successfully"}, status_code=200
62     )
```

```
src > ai > router.py > google_login
65 @ai_router.get("/bot_conversation/{user_email}")
66 def get_user_conversation(user_email: str, db: Session = Depends(get_db)):
67
68     ...
69     Retrieve the conversation history for a given user.
70
71     This endpoint retrieves all messages exchanged with a user, ordered by their creation date.
72
73     Args:
74         user_email (str): The email of the user whose conversation is being fetched.
75         db (Session, optional): The database session dependency.
76
77     Returns:
78         JSONResponse: A JSON response containing the user's conversation history.
79
80     Status Codes:
81         - 200: Successfully retrieved the conversation.
82         - 404: User not found or no conversation available.
83     ...
84
85     user_record = db.query(Users).filter(Users.email == user_email).first()
86     if not user_record:
87         raise HTTPException(
88             status_code=404, detail="No conversation found for this user"
89         )
90
91     all_messages = (
92         db.query(Messages)
93         .filter(Messages.user_id == user_record.id)
94         .order_by(Messages.created_at.asc())
95         .all()
```

```

@api_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.