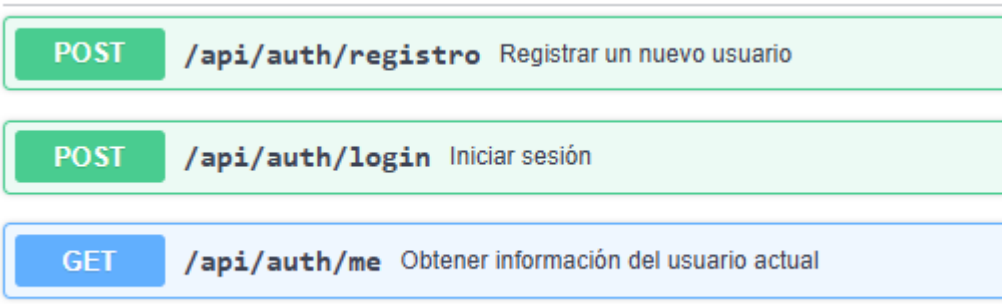
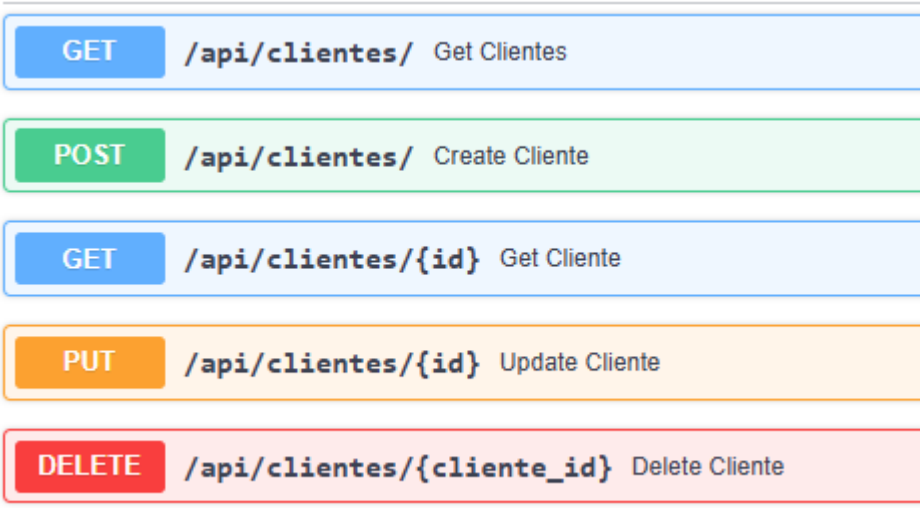
Desarrollo del Backend, Arquitectura Base y Modelo C4 del Sistema Multiplataforma.
Producto esperado:
Un repositorio remoto (GitHub o GitLab) correctamente estructurado y documentado que contenga:

1. **Backend funcional, con Endpoints REST implementados (usuarios, autenticación y entidades base).**

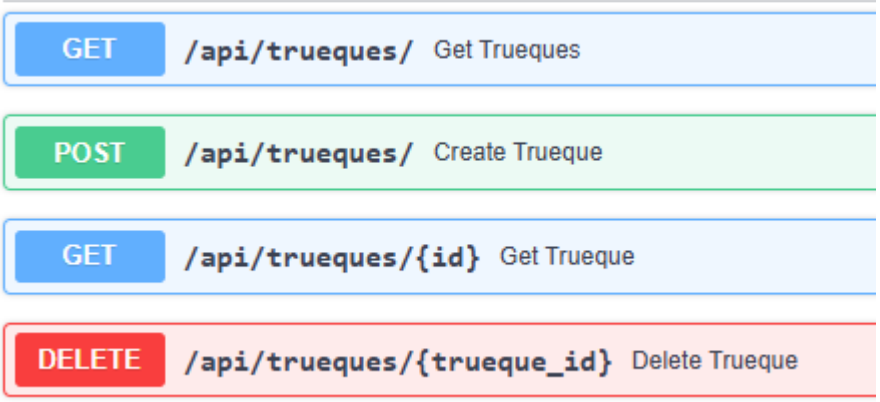
- **Autenticación**



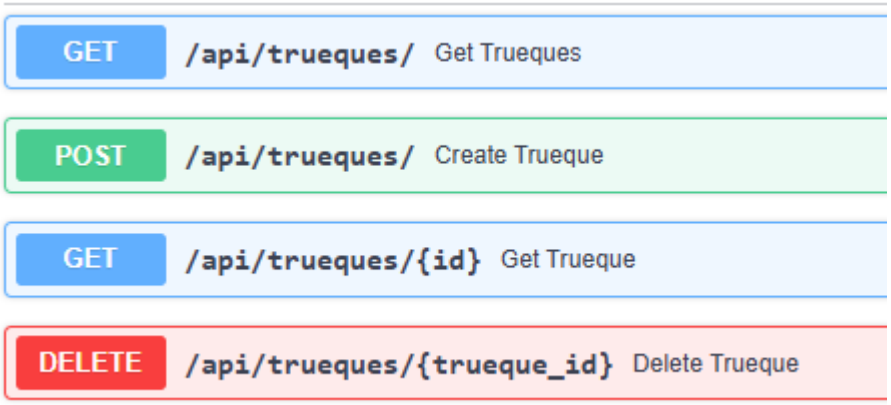
- **Clientes**



- **Trueques**



- **Tipo identificación**

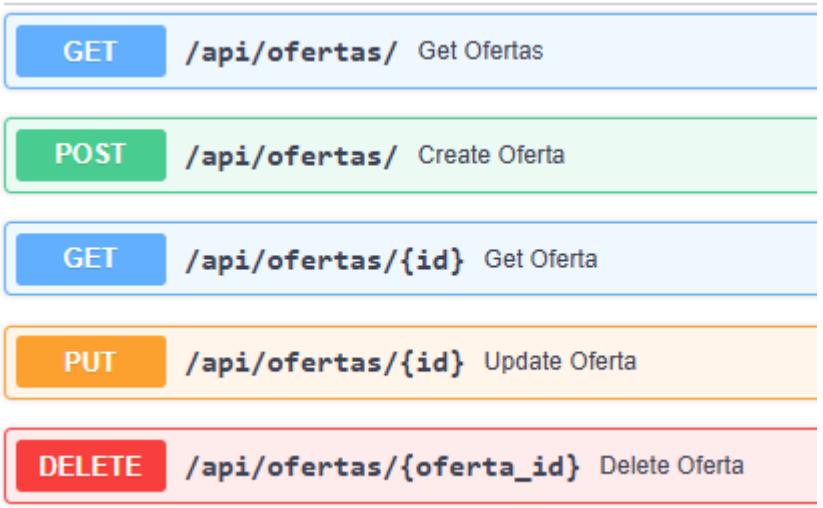


- **Personas**

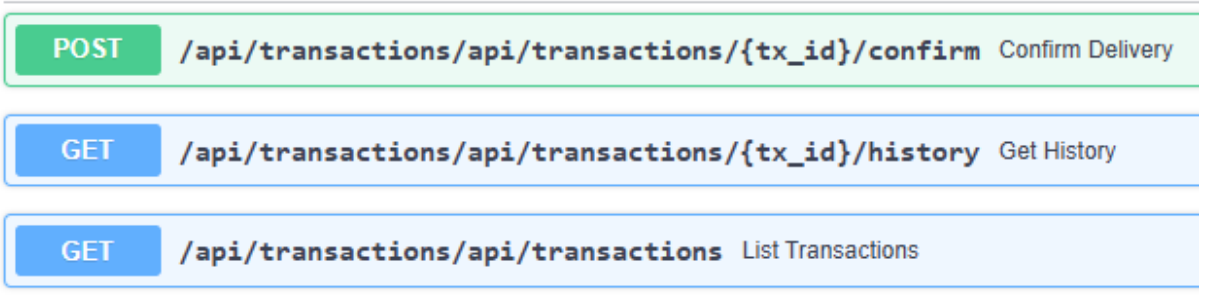
- **Categorias**

- **Articulos**

- **Ofertas**



- **Transacciones**

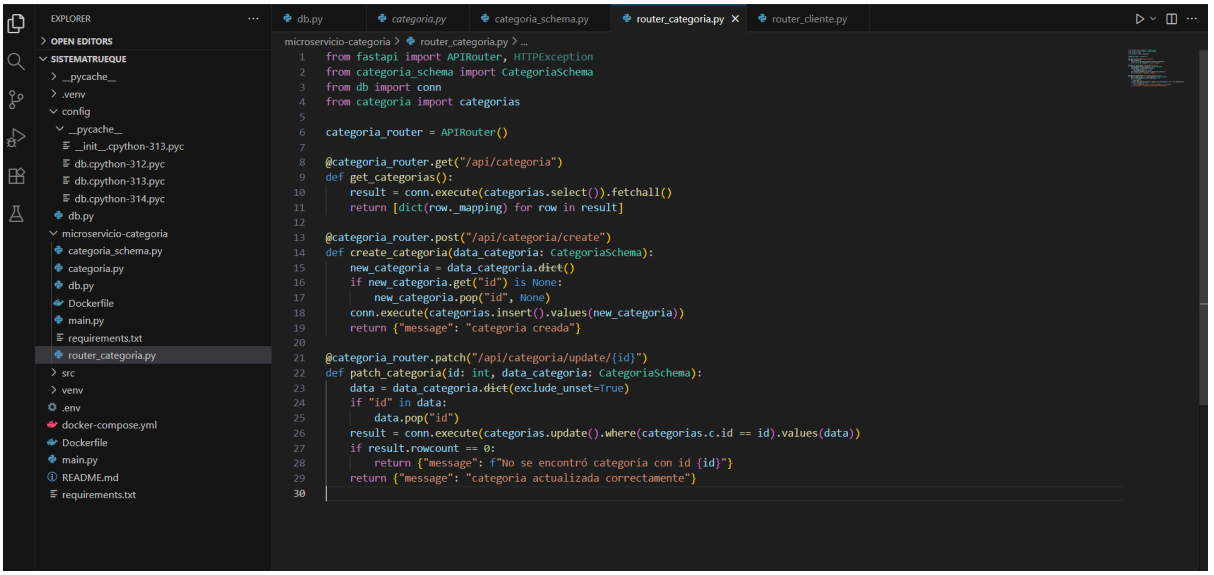


- **Configuración del entorno, dependencias y estructura modular del proyecto.**

- **Dependencias**

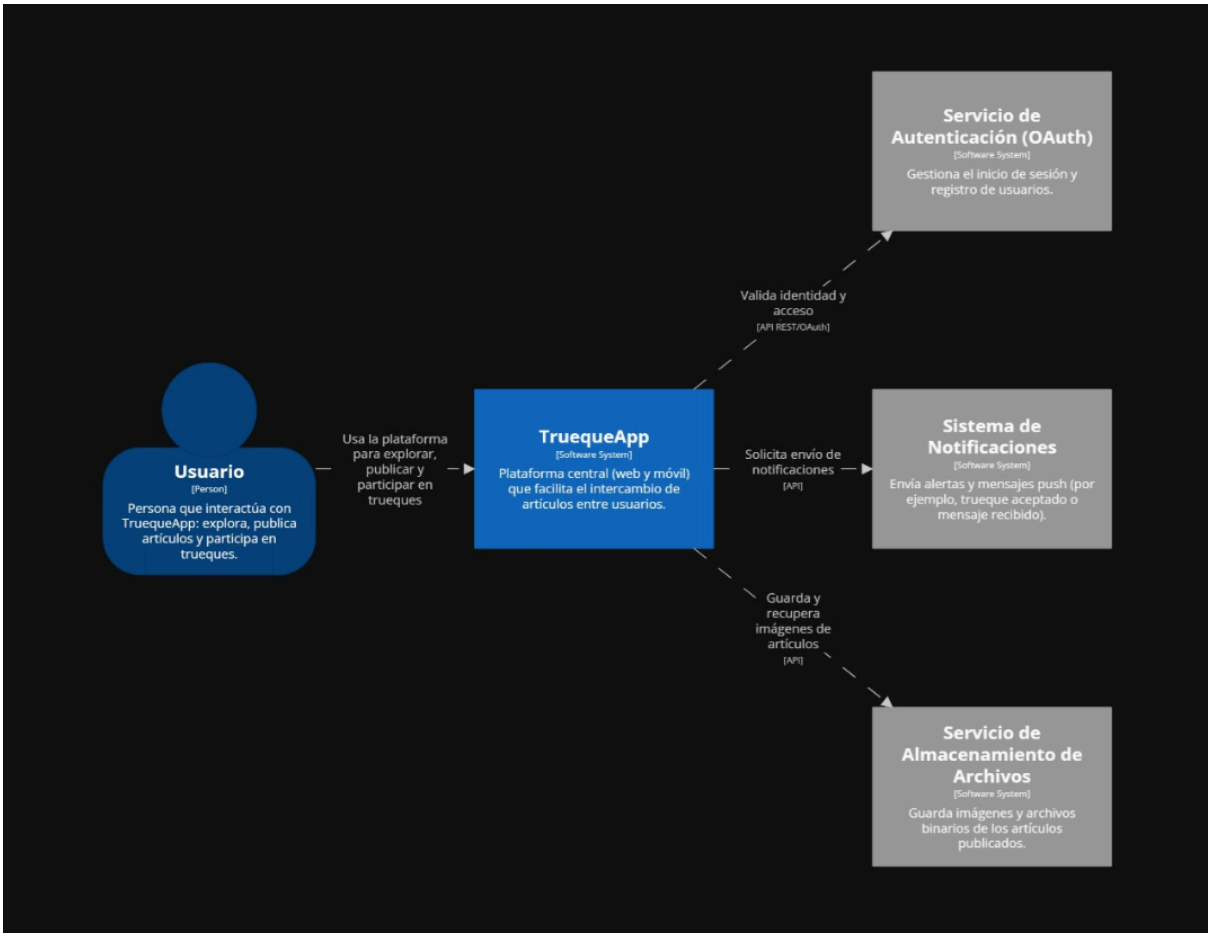
```
annotated-types==0.7.0
anyio==4.11.0
bcrypt==5.0.0
certifi==2025.10.5
cffi==2.0.0
click==8.3.0
colorama==0.4.6
cryptography==46.0.3
dnspython==2.8.0
email-validator==2.3.0
fastapi==0.119.0
fastapi-cli==0.0.13
fastapi-cloud-cli==0.3.1
greenlet==3.2.4
h11==0.16.0
httpcore==1.0.9
httptools==0.7.1
httpx==0.28.1
idna==3.11
Jinja2==3.1.6
mariadb==1.1.14
markdown-it-py==4.0.0
MarkupSafe==3.0.3
mdurl==0.1.2
packaging==25.0
passlib==1.7.4
pycparser==2.23
pydantic==2.12.2
PyJWT==2.8.0
python-jose[cryptography]==3.3.0
pydantic_core==2.41.4
Pygments==2.19.2
PyMySQL==1.1.2
python-dotenv==1.1.1
python-multipart==0.0.20
PyYAML==6.0.3
rich==14.2.0
rich-toolkit==0.15.1
rignore==0.7.1
sentry-sdk==2.42.0
```

- **Integración inicial de un microservicio o módulo independiente.**

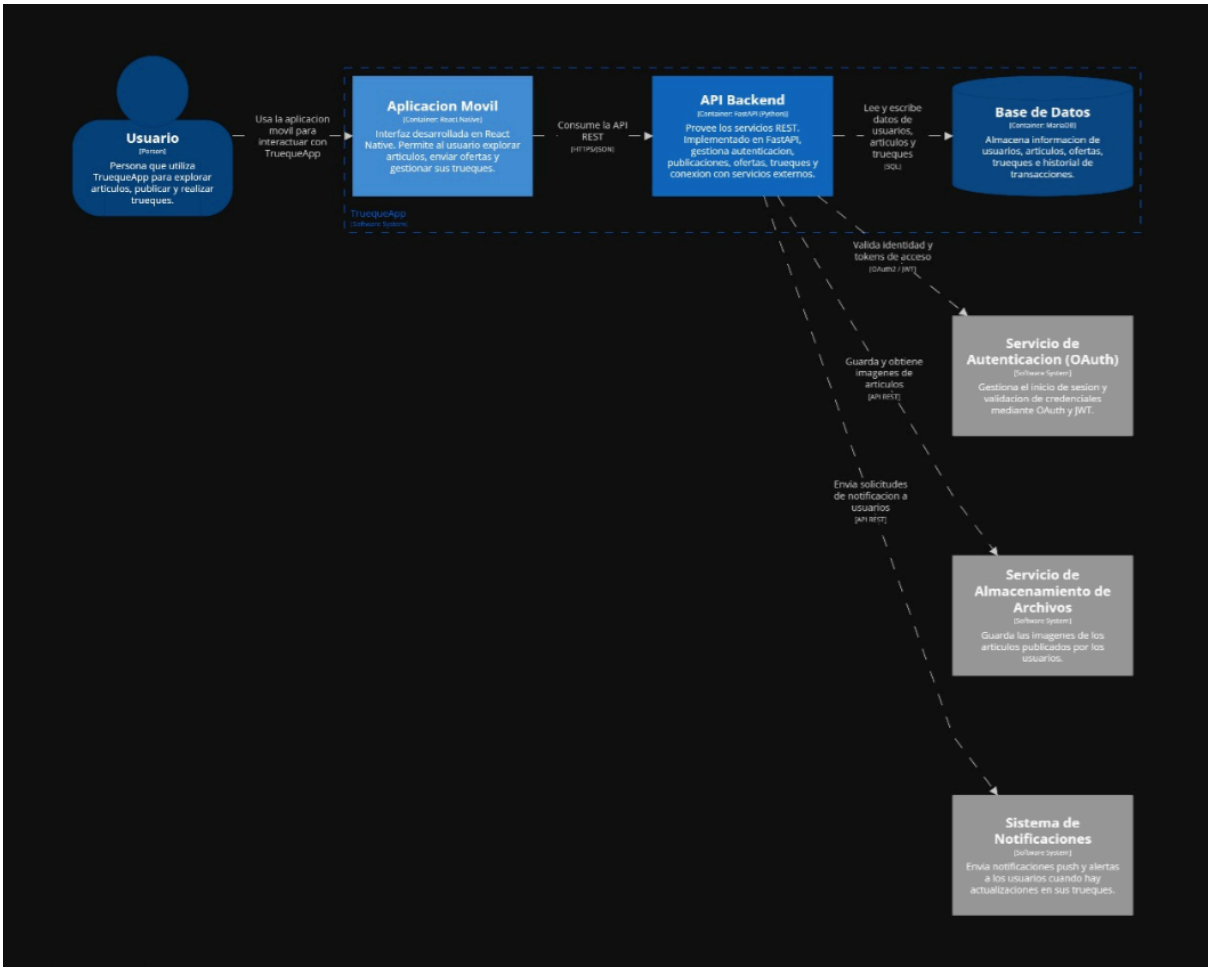


2. **Modelo C4 – Arquitectura del sistema:**

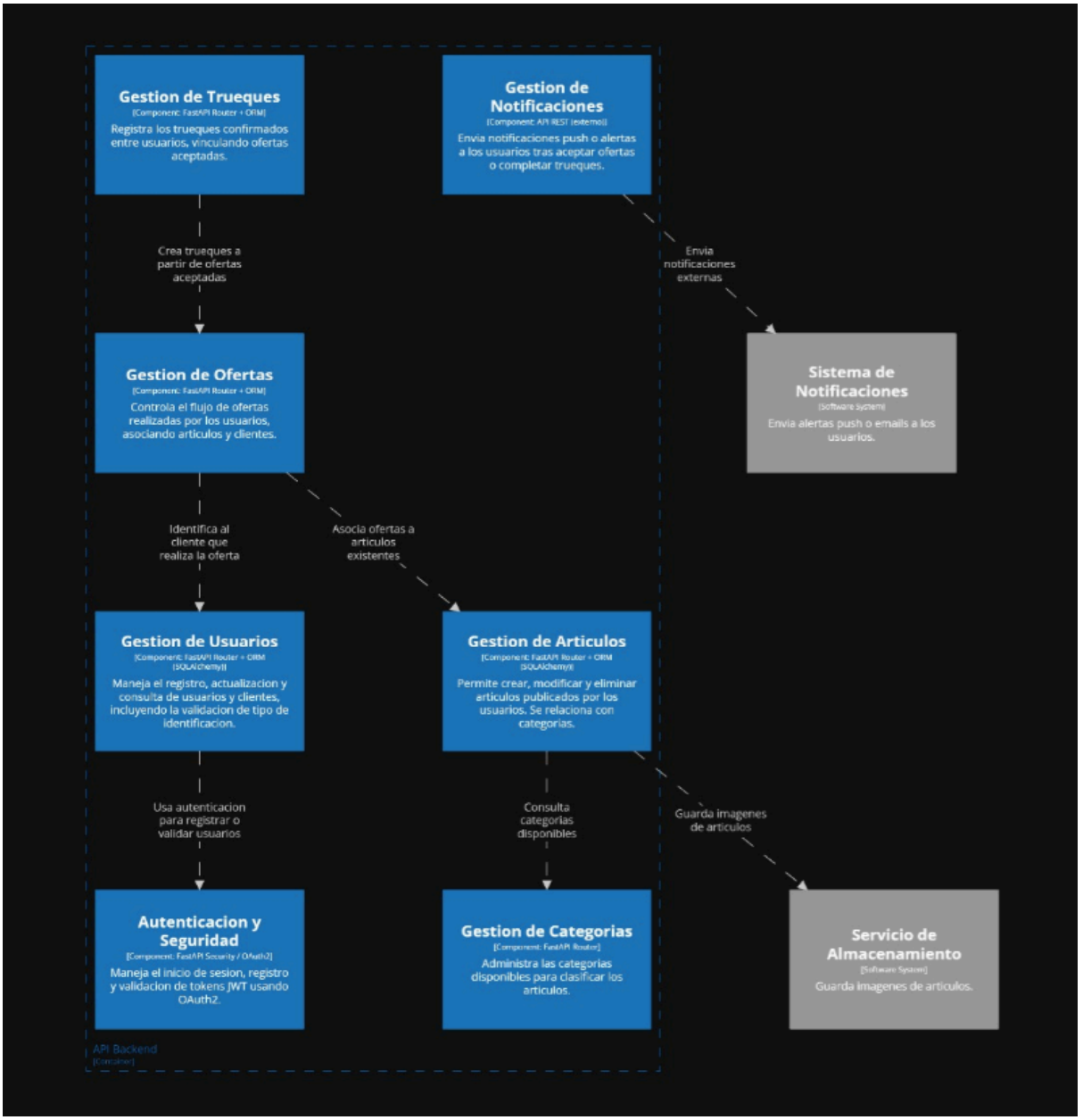
- **Nivel 1: Context Diagram – visión general del sistema y actores externos.**



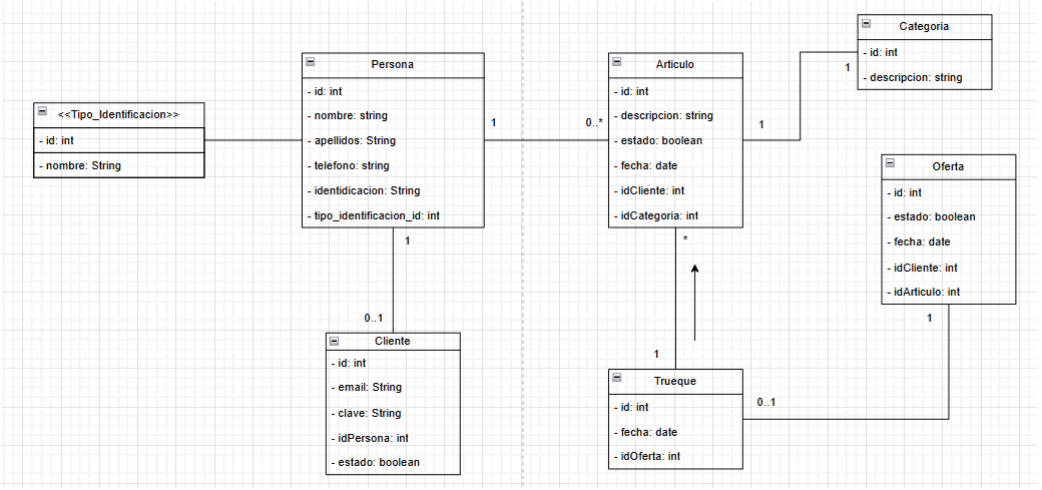
- **Nivel 2: Container Diagram – relación entre backend, frontend y base de datos.**



- **Nivel 3: Component Diagram – estructura interna del backend (controladores, servicios, repositorios).**



- **Nivel 4: Code Diagram – diagramas de clase**



3. **Documentación técnica:**

- **Descripción de arquitectura y dependencias.**

TruequeApp es una plataforma web y móvil diseñada para permitir el intercambio (trueque) de artículos entre usuarios registrados.

Su arquitectura está basada en el modelo multicapa y modular, siguiendo el estilo arquitectónico de microservicios ligeros con separación de responsabilidades clara entre frontend, backend, base de datos y servicios auxiliares.

El sistema se compone de varios contenedores (módulos) independientes que interactúan entre sí mediante API REST y eventos asíncronos (notificaciones y mensajería).

Esta estructura favorece la escalabilidad, mantenibilidad y seguridad, permitiendo desplegar o actualizar cada módulo sin afectar al resto del sistema.

Componentes Principales

- **Frontend Web (WebApp)**

Tecnología: HTML5, CSS3, JavaScript (Vanilla JS)

Función: Es la capa de presentación del sistema. Proporciona una interfaz gráfica intuitiva, amigable y responsiva, accesible desde distintos dispositivos (computadoras, tabletas y teléfonos móviles).

Permite a los usuarios realizar las siguientes acciones:

Crear cuentas y autenticarse.

Publicar, editar o eliminar artículos.

Buscar artículos mediante filtros.

Gestionar propuestas de trueque.

Acceder a mensajería y notificaciones.

Dependencias:

Se comunica con el Backend (API REST) mediante peticiones HTTP/HTTPS (fetch o XMLHttpRequest).

Utiliza Bootstrap o CSS Grid/Flexbox para el diseño adaptable (responsivo).

Puede integrar JavaScript modular para separar lógicas (por ejemplo: auth.js, articles.js, trades.js).

- **Aplicación Móvil**

Tecnología: React Native.

Función: Permite a los usuarios interactuar desde dispositivos móviles con las mismas funcionalidades que la versión web.

Dependencias:

Consume el Backend/API Central.

Se comunica con el Servicio de Notificaciones para recibir alertas y mensajes en tiempo real.

- **Backend / API Central**

Tecnología:FastAPI.

Función: Es el núcleo lógico del sistema, encargado de procesar las solicitudes del frontend, aplicar las reglas de negocio y coordinar la comunicación entre los módulos internos.

Responsabilidades:

Autenticación (JWT / OAuth2).

Gestión de usuarios, artículos y trueques.

Envío de correos electrónicos de verificación.

Integración con servicios externos.

Dependencias internas:

Base de Datos: para persistir toda la información.

Servicio de Almacenamiento: para gestionar imágenes de los artículos.

Servicio de Búsqueda: para indexar artículos y ejecutar búsquedas rápidas.

Servicio de Notificaciones: para comunicación en tiempo real.

Sistema de Correo: para enviar correos de confirmación y notificación.

- **Base de Datos**

Tecnología: MariaDB

Función: Almacenar de forma estructurada toda la información del sistema, como usuarios, artículos, mensajes y trueques.

Dependencias:

Accedida únicamente por el Backend.

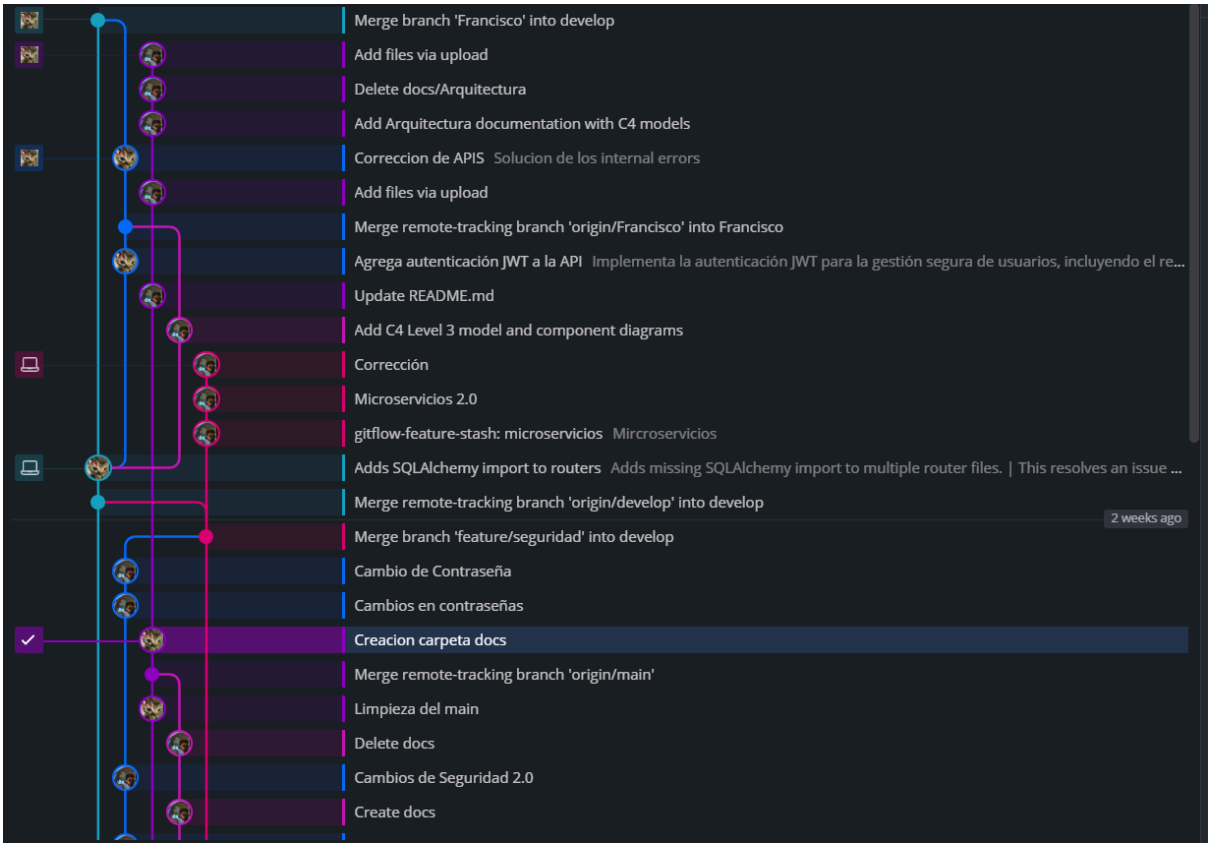
4. **Informe de seguridad:**

- **Evidencia de aplicación de principios OWASP Top 10, autenticación JWT u OAuth2 y cifrado de contraseñas.**

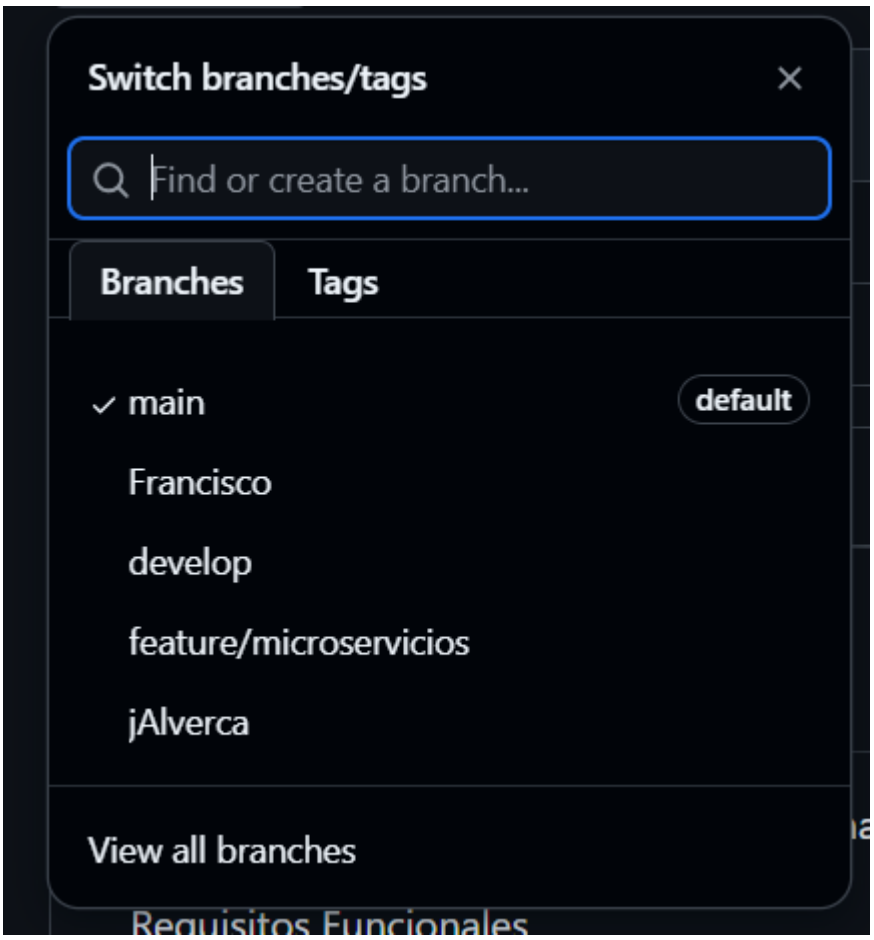
```python
# CONFIGURACIÓN DE HASHEO DE CONTRASEÑAS
# bcrypt es un algoritmo de hasheo muy seguro y lento
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

def hash_password(password: str) -> str:
    return pwd_context.hash(password)

def verify_password(plain_password: str, hashed_password: str) -> bool:
    return pwd_context.verify(plain_password, hashed_password)

# FUNCIONES PARA MANEJO DE JWT

def create_access_token(data: dict, expires_delta: Optional[timedelta] = None) -> str:
    # Copiamos la data para no modificar el original
    to_encode = data.copy()

    # Calculamos cuándo expira el token
    if expires_delta:
        expire = datetime.~~utcnow~~() + expires_delta
    else:
        expire = datetime.~~utcnow~~() + timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)

    # Agregamos la fecha de expiración al token
    to_encode.update({"exp": expire})

    # Creamos el token firmado con nuestra clave secreta
    encoded_jwt = jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)

    return encoded_jwt

def verify_token(token: str) -> Optional[dict]:
    try:
        # Intentamos decodificar el token
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        return payload
    except JWTError:
        # Si falla (token inválido, expirado, o modificado), retornamos None
        return None
```

```python
# CONFIGURACIÓN DEL ESQUEMA DE SEGURIDAD
# HTTPBearer es el esquema que espera un token en el header:

security = HTTPBearer()

def get_current_user(credentials: HTTPAuthorizationCredentials = Depends(security)) -> dict:
    # Extraer el token del objeto credentials
    token = credentials.credentials

    # Verificar el token usando nuestra función de auth_utils
    payload = verify_token(token)

    # Si el token es inválido, verify_token retorna None
    if payload is None:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Token inválido o expirado. Por favor, inicia sesión nuevamente.",
            headers={"WWW-Authenticate": "Bearer"},  # Estándar HTTP para autenticación
        )

    # Extraer el username del payload del token
    # "sub" es el campo estándar para el identificador del usuario
    username: Optional[str] = payload.get("sub")
    if username is None:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Token no contiene información válida del usuario.",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return payload


def get_optional_user(credentials: Optional[HTTPAuthorizationCredentials] = Depends(security)) -> Optional[dict]:
    if credentials is None:
        return None
    token = credentials.credentials
    payload = verify_token(token)
    return payload if payload else None
```

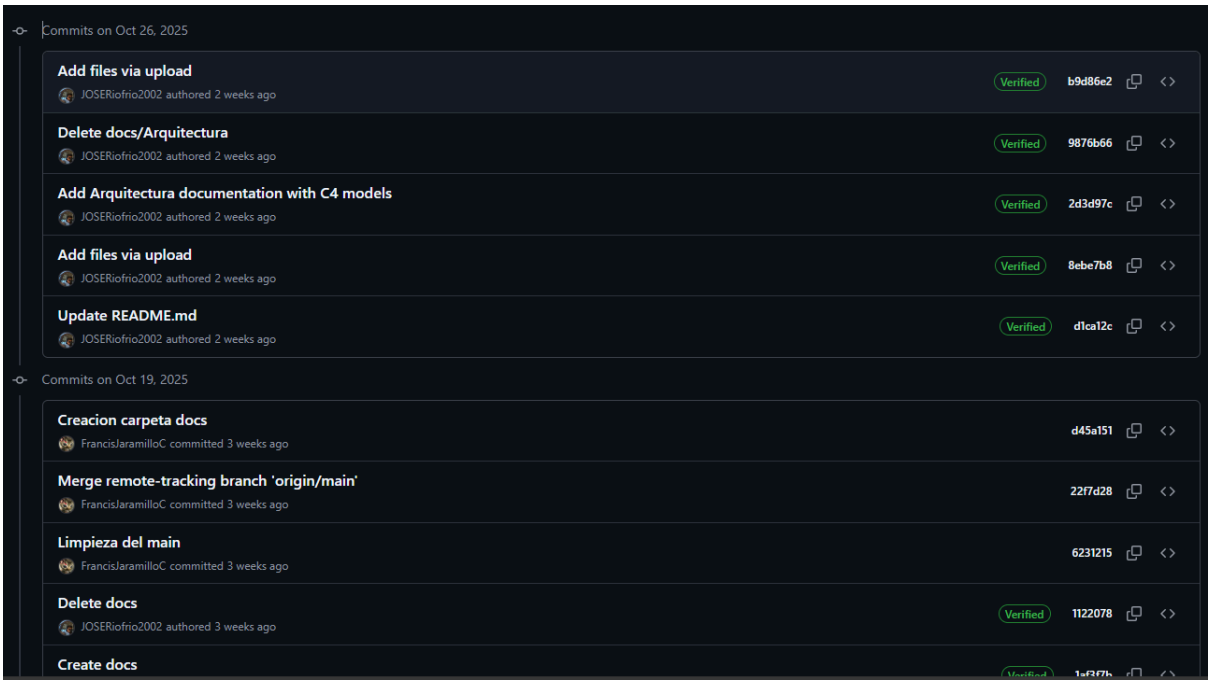
5. Control de versiones con GitKraken y flujo GitFlow:

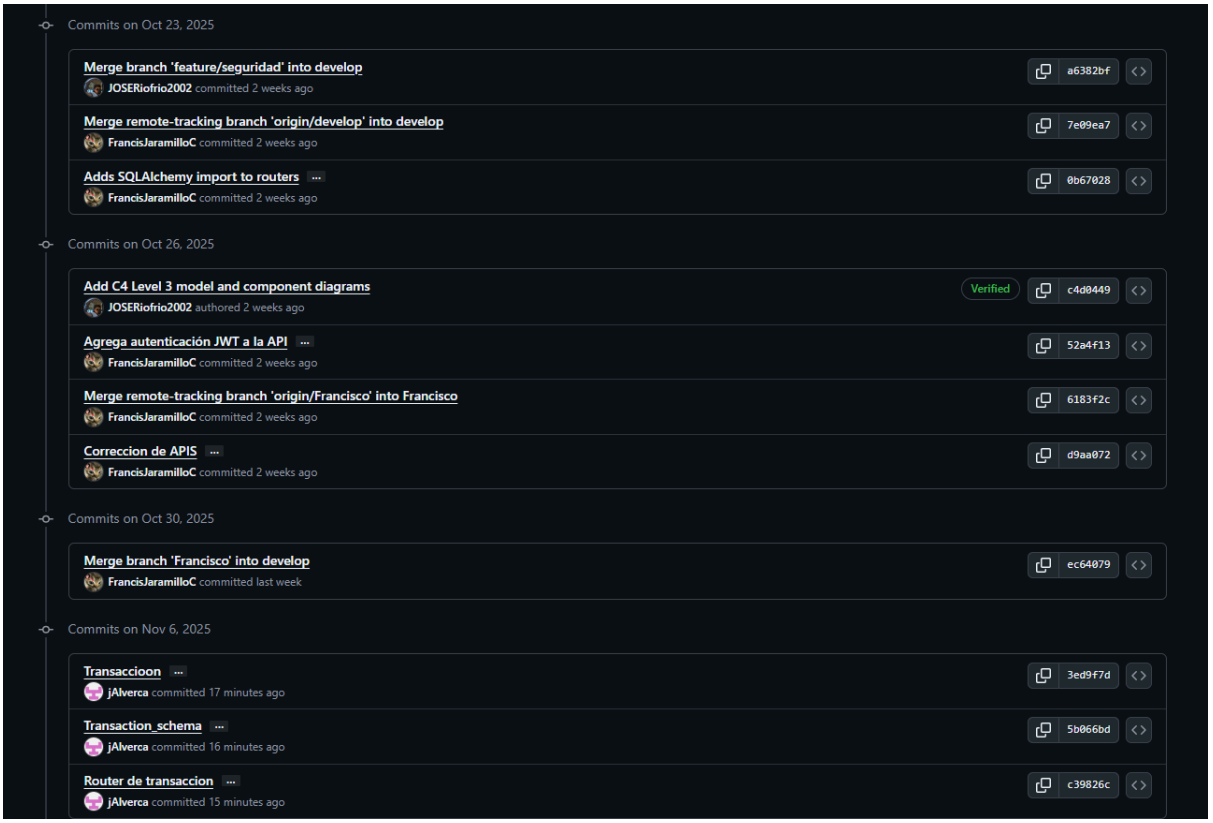


- **Ramas principales**



- **Commits descriptivos y merges ordenados.**




Commits on Oct 23, 2025
Merge branch 'feature/seguridad' into develop
JOSERiofrio2002 committed 2 weeks ago
a6382bf
Merge remote-tracking branch 'origin/develop' into develop
FrancisJaramilloC committed 2 weeks ago
7e09ea7
Adds SQLAlchemy import to routers
FrancisJaramilloC committed 2 weeks ago
0b67028
Commits on Oct 26, 2025
Add C4 Level 3 model and component diagrams
JOSERiofrio2002 authored 2 weeks ago
Verified
c4d0449
Agrega autenticación JWT a la API
FrancisJaramilloC committed 2 weeks ago
52a4f13
Merge remote-tracking branch 'origin/Francisco' into Francisco
FrancisJaramilloC committed 2 weeks ago
6183f2c
Correccion de APIS
FrancisJaramilloC committed 2 weeks ago
d9aa072
Commits on Oct 30, 2025
Merge branch 'Francisco' into develop
FrancisJaramilloC committed last week
ec64079
Commits on Nov 6, 2025
Transaccioon
jAlverca committed 17 minutes ago
3ed9f7d
Transaction_schema
jAlverca committed 16 minutes ago
5b066bd
Router de transaccion
jAlverca committed 15 minutes ago
c39826c