

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

POST	<code>/api/auth/registro</code>	Registrar un nuevo usuario
POST	<code>/api/auth/login</code>	Iniciar sesión
GET	<code>/api/auth/me</code>	Obtener información del usuario actual

- Clientes

GET	<code>/api/clientes/</code>	Get Clientes
POST	<code>/api/clientes/</code>	Create Cliente
GET	<code>/api/clientes/{id}</code>	Get Cliente
PUT	<code>/api/clientes/{id}</code>	Update Cliente
DELETE	<code>/api/clientes/{cliente_id}</code>	Delete Cliente

- Trueques

GET	<code>/api/trueques/</code>	Get Trueques
POST	<code>/api/trueques/</code>	Create Trueque
GET	<code>/api/trueques/{id}</code>	Get Trueque
DELETE	<code>/api/trueques/{trueque_id}</code>	Delete Trueque

- Tipo identificación

GET /api/trueques/ Get Trueques

POST /api/trueques/ Create Trueque

GET /api/trueques/{id} Get Trueque

DELETE /api/trueques/{trueque_id} Delete Trueque

- Personas

GET /api/personas/ Get Personas

POST /api/personas/ Create Persona

GET /api/personas/{id} Get Persona

PUT /api/personas/{id} Update Persona

DELETE /api/personas/{persona_id} Delete Persona

- Categorias

GET /api/categorias/ Get Categorias

POST /api/categorias/ Create Categoria

PATCH /api/categorias/{id} Patch Categoria

DELETE /api/categorias/{categoria_id} Delete Categoria

- Articulos

GET	<code>/api/articulos/</code> Get Articulos
POST	<code>/api/articulos/</code> Create Articulo
GET	<code>/api/articulos/{id}</code> Get Articulo
DELETE	<code>/api/articulos/{articulo_id}</code> Delete Articulo
DELETE	<code>/api/articulos/secure-delete/{articulo_id}</code> Secure Delete Articulo

- Ofertas

GET	<code>/api/ofertas/</code> Get Ofertas
POST	<code>/api/ofertas/</code> Create Oferta
GET	<code>/api/ofertas/{id}</code> Get Oferta
PUT	<code>/api/ofertas/{id}</code> Update Oferta
DELETE	<code>/api/ofertas/{oferta_id}</code> Delete Oferta

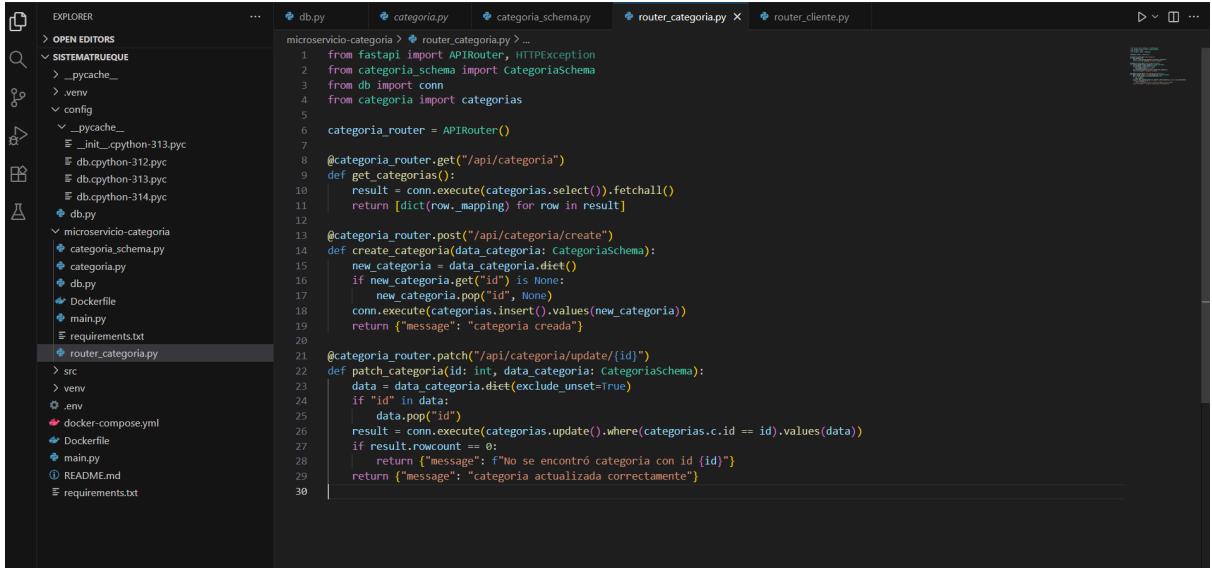
- Transacciones

POST	<code>/api/transactions/api/transactions/{tx_id}/confirm</code> Confirm Delivery
GET	<code>/api/transactions/api/transactions/{tx_id}/history</code> Get History
GET	<code>/api/transactions/api/transactions</code> List Transactions

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

```
1 annotated-types==0.7.0
2 anyio==4.11.0
3 bcrypt==5.0.0
4 certifi==2025.10.5
5 cffi==2.0.0
6 click==8.3.0
7 colorama==0.4.6
8 cryptography==46.0.3
9 dnspython==2.8.0
10 email-validator==2.3.0
11 fastapi==0.119.0
12 fastapi-cli==0.0.13
13 fastapi-cloud-cli==0.3.1
14 greenlet==3.2.4
15 h11==0.16.0
16 httpcore==1.0.9
17 httptools==0.7.1
18 httpx==0.28.1
19 idna==3.11
20 Jinja2==3.1.6
21 mariadb==1.1.14
22 markdown-it-py==4.0.0
23 MarkupSafe==3.0.3
24 mdurl==0.1.2
25 packaging==25.0
26 passlib==1.7.4
27 pycparser==2.23
28 pydantic==2.12.2
29 PyJWT==2.8.0
30 python-jose[cryptography]==3.3.0
31 pydantic_core==2.41.4
32 Pygments==2.19.2
33 PyMySQL==1.1.2
34 python-dotenv==1.1.1
35 python-multipart==0.0.20
36 PyYAML==6.0.3
37 rich==14.2.0
38 rich-toolkit==0.15.1
39 rignore==0.7.1
40 sentry-sdk==2.42.0
```

- Integración inicial de un microservicio o módulo independiente.



```

EXPLORER
OPEN EDITORS
SISTEMATRUEQUE
  > __pycache__
  > .venv
  > config
    > __pycache__
      __init__.cpython-313.pyc
      db.cpython-312.pyc
      db.cpython-313.pyc
      db.cpython-314.pyc
    db.py
    microsservicio-categoría
      categoria_schema.py
      categoria.py
      db.py
      Dockerfile
      main.py
      docker-compose.yml
      Dockerfile
      requirements.txt
      README.md
      requirements.txt

ROUTER
  db.py
  categoria.py
  categoria_schema.py
  router_categoria.py X
  router_cliente.py

ROUTER SOURCE
  router_categoria.py
  router_cliente.py

ROUTER TEST
  router_categoria.py
  router_cliente.py

ROUTER VCS
  router_categoria.py
  router_cliente.py

ROUTER WORKSPACE
  router_categoria.py
  router_cliente.py

```

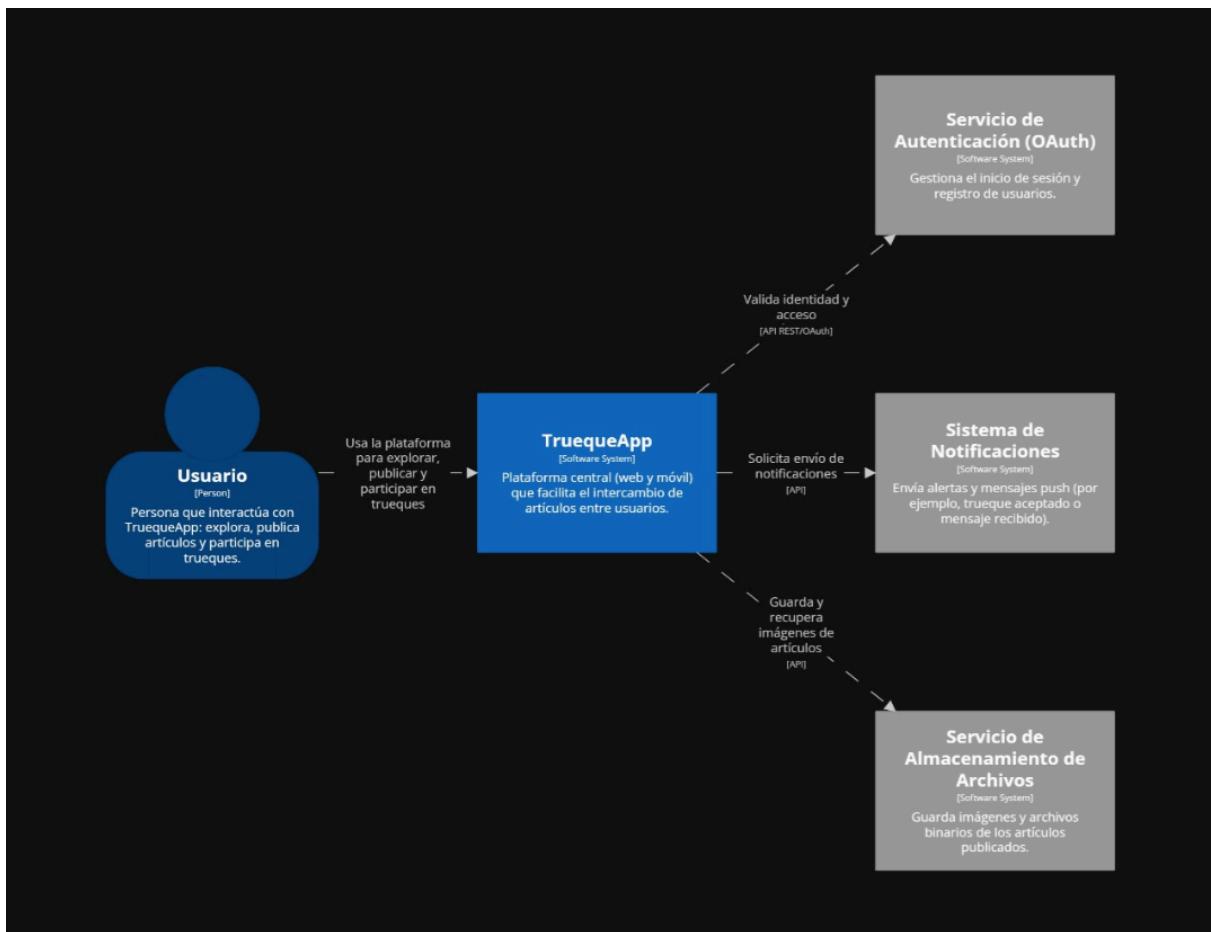
```

1  from fastapi import APIRouter, HTTPException
2  from categoria_schema import CategoriaSchema
3  from db import conn
4  from categoria import categorias
5
6  categoria_router = APIRouter()
7
8  @categoria_router.get("/api/categoría")
9  def get_categorías():
10    result = conn.execute(categorias.select())
11    return [dict(row._mapping) for row in result]
12
13 @categoria_router.post("/api/categoría/create")
14 def create_categoria(data_categoria: CategoriaSchema):
15   new_categoria = data_categoria.dict()
16   if new_categoria.get("id") is None:
17     new_categoria.pop("id", None)
18   conn.execute(categorias.insert().values(new_categoria))
19   return {"message": "Categoría creada"}
20
21 @categoria_router.patch("/api/categoría/update/{id}")
22 def patch_categoria(id: int, data_categoria: CategoriaSchema):
23   data = data_categoria.dict(exclude_unset=True)
24   if "id" in data:
25     data.pop("id")
26   result = conn.execute(categorias.update().where(categorias.c.id == id).values(data))
27   if result.rowcount == 0:
28     return {"message": f"No se encontró categoría con id {id}"}
29   return {"message": "Categoría actualizada correctamente"}
30

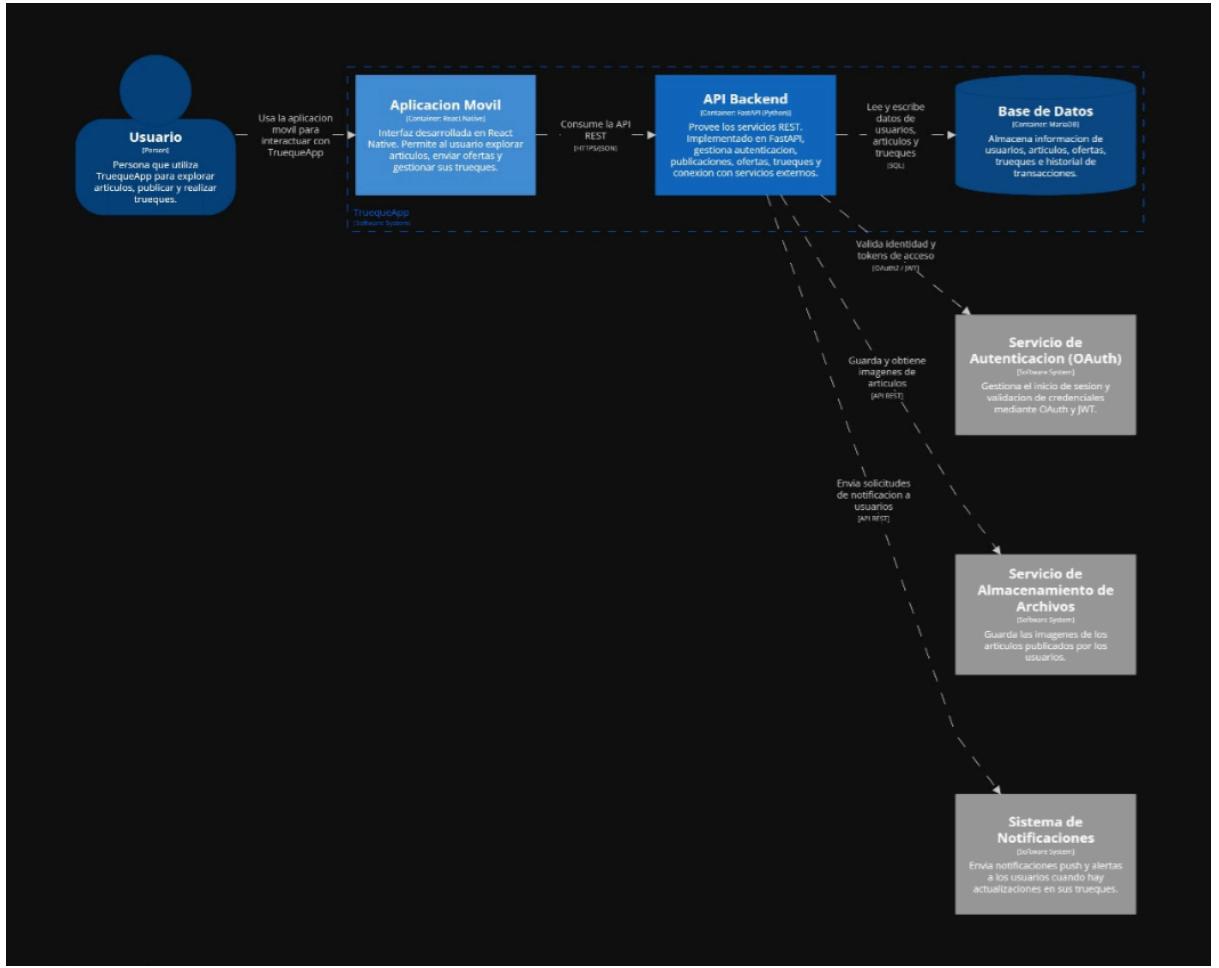
```

2. Modelo C4 – Arquitectura del sistema:

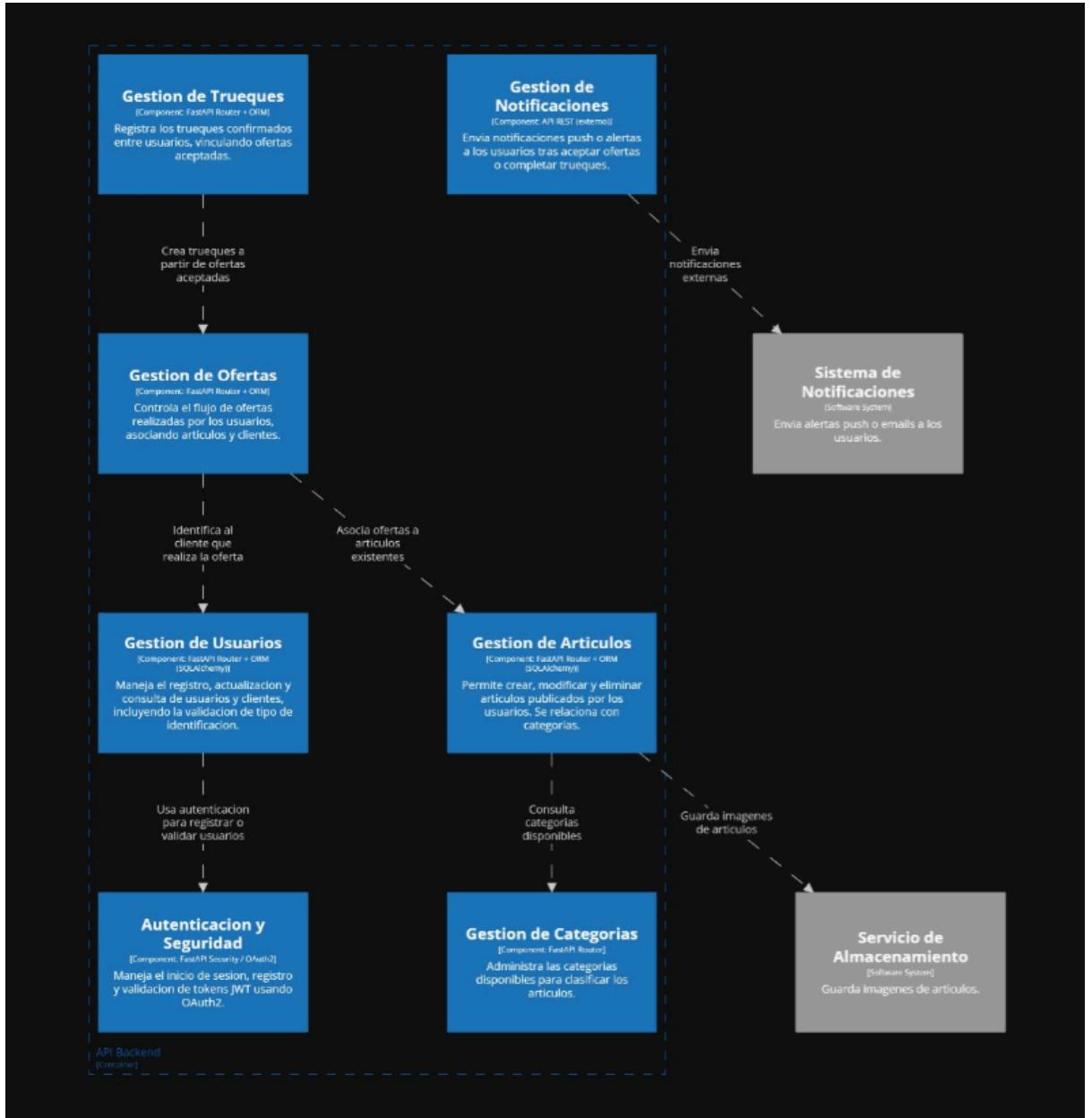
- Nivel 1: Context Diagram – visión general del sistema y actores externos.



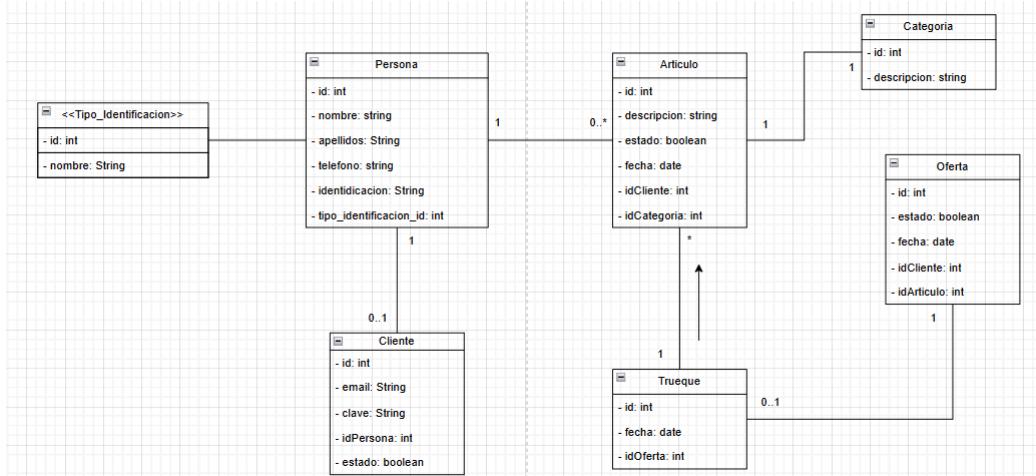
- Nivel 2: Container Diagram – relación entre backend, frontend y base de datos.



- 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 responsive, 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:

Consumo 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.**

```
# 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: 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

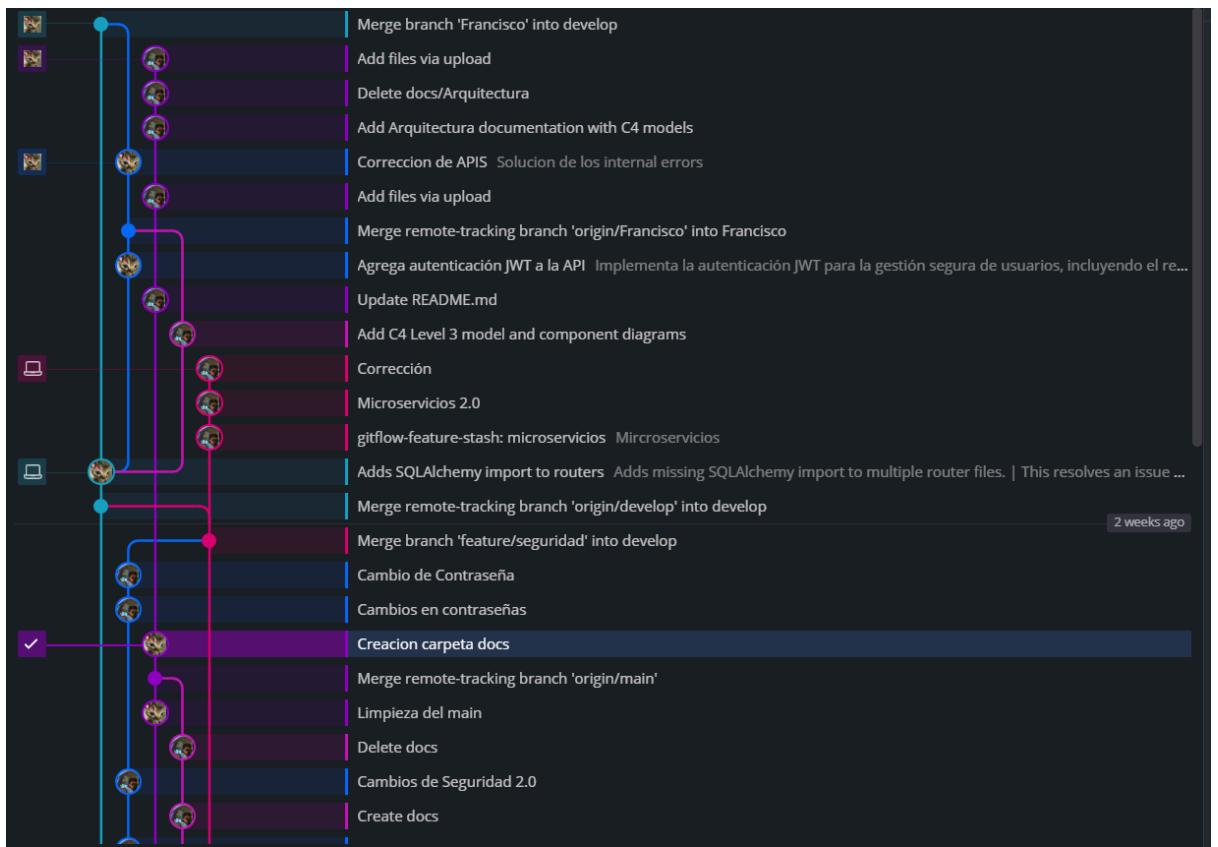
def verify_token(token: str) -> Optional[dict]:
    try:
        # Intentamos decodificar el token
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        return payload
    except JWTError:
        # Si falla (token inválido, expirado, o modificado), retornamos None
        return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

```

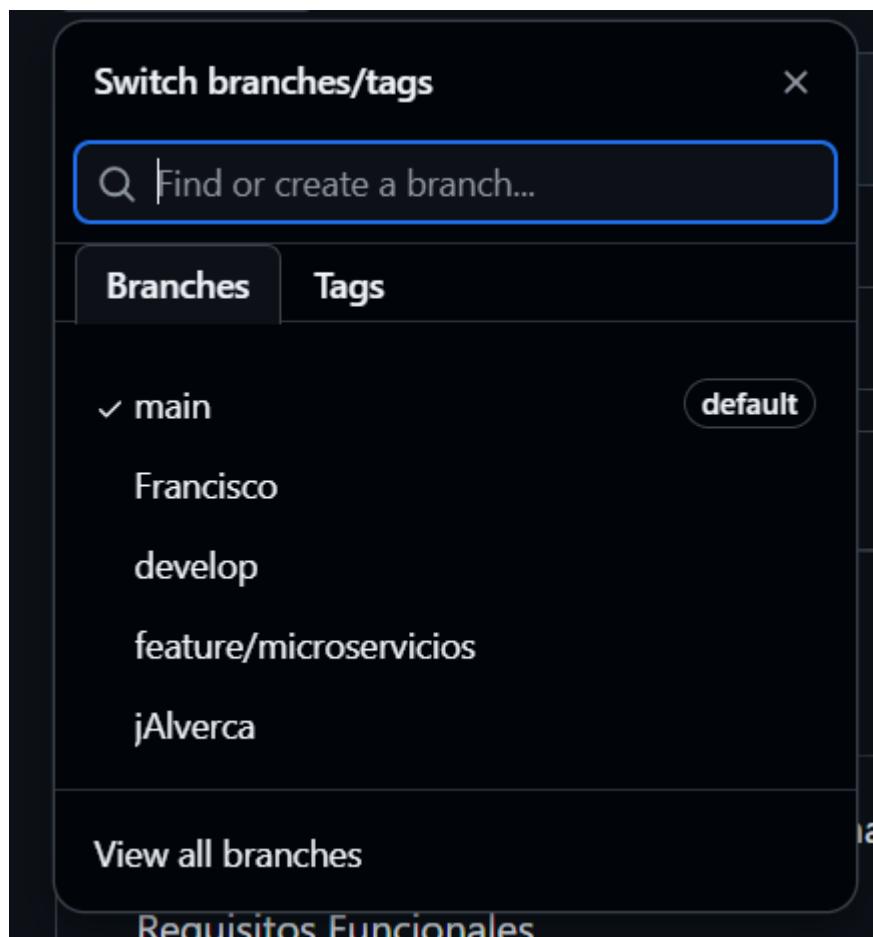
6  # CONFIGURACIÓN DEL ESQUEMA DE SEGURIDAD
7  # HTTPBearer es el esquema que espera un token en el header:
8
9  security = HTTPBearer()
10
11 def get_current_user(credentials: HTTPAuthorizationCredentials = Depends(security)) -> dict:
12     # Extraer el token del objeto credentials
13     token = credentials.credentials
14
15     # Verificar el token usando nuestra función de auth_utils
16     payload = verify_token(token)
17
18     # Si el token es inválido, verify_token retorna None
19     if payload is None:
20         raise HTTPException(
21             status_code=status.HTTP_401_UNAUTHORIZED,
22             detail="Token inválido o expirado. Por favor, inicia sesión nuevamente.",
23             headers={"WWW-Authenticate": "Bearer"}, # Estándar HTTP para autenticación
24         )
25
26     # Extraer el username del payload del token
27     # "sub" es el campo estándar para el identificador del usuario
28     username: Optional[str] = payload.get("sub")
29     if username is None:
30         raise HTTPException(
31             status_code=status.HTTP_401_UNAUTHORIZED,
32             detail="Token no contiene información válida del usuario.",
33             headers={"WWW-Authenticate": "Bearer"},
34         )
35     return payload
36
37
38 def get_optional_user(credentials: Optional[HTTPAuthorizationCredentials] = Depends(security)) -> Optional[dict]:
39     if credentials is None:
40         return None
41     token = credentials.credentials
42     payload = verify_token(token)
43     return payload if payload else None

```

5. Control de versiones con GitKraken y flujo GitFlow:



- Ramas principales



- Commits descriptivos y merges ordenados.

-o-	Commits on Oct 26, 2025
	Add files via upload JOSERlofrio2002 authored 2 weeks ago Verified b9d86e2
	Delete docs/Arquitectura JOSERlofrio2002 authored 2 weeks ago Verified 9876b66
	Add Arquitectura documentation with C4 models JOSERlofrio2002 authored 2 weeks ago Verified 2d3d97c
	Add files via upload JOSERlofrio2002 authored 2 weeks ago Verified 8be7b8
	Update README.md JOSERlofrio2002 authored 2 weeks ago Verified d1ca12c
-o-	Commits on Oct 19, 2025
	Creacion carpeta docs FrancisJaramilloC committed 3 weeks ago d45a151
	Merge remote-tracking branch 'origin/main' FrancisJaramilloC committed 3 weeks ago 22f7d28
	Limpieza del main FrancisJaramilloC committed 3 weeks ago 6231215
	Delete docs JOSERlofrio2002 authored 3 weeks ago Verified 1122078
	Create docs FrancisJaramilloC committed 3 weeks ago 1a1277b

-o	Commits on Oct 23, 2025
	<p>Merge branch 'feature/seguridad' into develop</p> <p>JOSERiofrio2002 committed 2 weeks ago</p> <p>Merge remote-tracking branch 'origin/develop' into develop</p> <p>FrancisJaramilloC committed 2 weeks ago</p> <p>Adds SQLAlchemy import to routers ...</p> <p>FrancisJaramilloC committed 2 weeks ago</p>
-o	Commits on Oct 26, 2025
	<p>Add C4 Level 3 model and component diagrams</p> <p>JOSERiofrio2002 authored 2 weeks ago</p> <p>Agrega autenticación JWT a la API ...</p> <p>FrancisJaramilloC committed 2 weeks ago</p> <p>Merge remote-tracking branch 'origin/Francisco' into Francisco</p> <p>FrancisJaramilloC committed 2 weeks ago</p> <p>Corrección de APIs ...</p> <p>FrancisJaramilloC committed 2 weeks ago</p>
-o	Commits on Oct 30, 2025
	<p>Merge branch 'Francisco' into develop</p> <p>FrancisJaramilloC committed last week</p>
-o	Commits on Nov 6, 2025
	<p>Transaccion ...</p> <p>jAlverca committed 17 minutes ago</p> <p>Transaction schema ...</p> <p>jAlverca committed 16 minutes ago</p> <p>Router de transaccion ...</p> <p>jAlverca committed 15 minutes ago</p>