Bitácora Digital

Kevin Felipe Moreno Ramirez

Fecha de inicio (planteamiento):

27/09/2025

Contenido

Introducción

El presente proyecto consiste en el desarrollo de una **aplicación web de tipo blog** que integre **frontend, backend y base de datos**, con el fin de ofrecer un sistema completo de publicación y gestión de contenidos.
La plataforma permitirá a los usuarios **registrarse**, **iniciar sesión** y **gestionar sus propias publicaciones**, así **como visualizar los posts realizados** por otros miembros de la comunidad. Además, contará con un sistema de **roles** de usuario que definirá los permisos y responsabilidades de cada perfil, garantizando así un manejo adecuado de la información.

Este proyecto busca no solo consolidar los conocimientos fundamentales en programación backend, sino también aplicar principios de diseño web, persistencia de datos y seguridad, de manera que se construya una aplicación funcional, escalable y segura.

Objetivo general

Desarrollar una aplicación de blog que permita a los usuarios autenticarse, crear y gestionar publicaciones, interactuar con los contenidos de otros usuarios y mantener un control adecuado mediante roles y permisos, integrando frontend, backend y base de datos en un sistema completo.

Objetivos específicos

- **Implementar** un sistema de registro e inicio de sesión con validación de credenciales y seguridad en el manejo de contraseñas.
- **Establecer** un control de roles de usuario (administrador, autor, lector) que determine los permisos de acceso y gestión de contenido.
- **Diseñar e implementar** operaciones CRUD (crear, leer, actualizar y eliminar) para la administración de publicaciones.
- **Desarrollar** una interfaz de usuario amigable que facilite la interacción con el sistema.
- **Integrar** una base de datos relacional para el almacenamiento y persistencia de usuarios y publicaciones.
- **Garantizar** prácticas de seguridad y escalabilidad, permitiendo la futura extensión del sistema con nuevas funcionalidades como comentarios, buscador de publicaciones o categorización de contenido.

Alcance del Proyecto

Alcance funcional

El sistema de blog permitirá:

- **Registro e inicio de sesión** de usuarios con validación de credenciales.
- **Gestión de roles:**
  - **Administrador:** podrá gestionar usuarios y moderar publicaciones.
  - **Autor:** podrá crear, editar y eliminar sus propias publicaciones.
  - **Lector:** podrá visualizar publicaciones de los demás usuarios.
- **Gestión de publicaciones (CRUD):** cada usuario podrá crear, modificar y eliminar únicamente sus propios posts.
- **Visualización de contenido:** cualquier usuario autenticado podrá acceder al listado de publicaciones y ver los posts de otros miembros.

Alcance no funcional

- **Seguridad:** manejo de contraseñas encriptadas y validación de sesiones.
- **Escalabilidad:** la arquitectura permitirá agregar funcionalidades adicionales (ej. comentarios, likes, categorías).
- **Interfaz amigable y responsive**, accesible desde distintos dispositivos.
- **Persistencia de datos** mediante una base de datos relacional optimizada.
- **Disponibilidad:** sistema funcional para múltiples usuarios concurrentes.

Exclusiones

- No se incluirá integración con redes sociales.
- No se implementará carga de archivos multimedia en esta primera versión.
- No se desarrollarán notificaciones en tiempo real.

Justificación

El desarrollo de este proyecto surge como una iniciativa de **aprendizaje autónomo**, con el propósito de poner en práctica los conocimientos adquiridos en programación, bases de datos y desarrollo web. La elección de un blog como caso de estudio responde a su utilidad como ejercicio integral, ya que permite aplicar conceptos fundamentales como autenticación de usuarios, manejo de roles, operaciones CRUD y diseño de interfaces.

Asimismo, la motivación principal detrás de este trabajo es el **interés personal por el aprendizaje continuo** y el gusto por fortalecer las competencias técnicas a través de proyectos prácticos. Al abordar la construcción de una aplicación completa (frontend, backend y base de datos), se busca no solo afianzar conocimientos teóricos, sino también desarrollar habilidades aplicables en entornos reales de desarrollo de software.

En este sentido, el proyecto no se limita a un requerimiento académico, sino que constituye una oportunidad de **exploración y experimentación**, fomentando la autonomía, la creatividad y la capacidad de resolución de problemas en el ámbito tecnológico.

Marco Teórico

Blog

Un blog es una aplicación web que permite a los usuarios crear y publicar contenidos en formato de entradas o publicaciones, generalmente organizadas de manera cronológica. Este tipo de plataforma resulta ideal para fines académicos y prácticos, ya que integra funcionalidades de autenticación de usuarios, control de permisos y gestión de datos, lo que lo convierte en un caso de uso representativo para el aprendizaje de desarrollo de software.

CRUD (Create, Read, Update, Delete)

El modelo CRUD hace referencia a las cuatro operaciones básicas que permiten la gestión de datos en un sistema: crear, leer, actualizar y eliminar. Estas operaciones son fundamentales en el desarrollo de aplicaciones web, ya que permiten la interacción entre los usuarios y la información almacenada en la base de datos.

Roles de usuario

Los roles representan diferentes niveles de acceso y permisos dentro de una aplicación. En un sistema de blog, se suelen implementar roles como:

- **Administrador**: encargado de gestionar usuarios y moderar publicaciones.
- **Autor**: puede crear, editar y eliminar sus propios posts.
- **Lector**: accede únicamente a la visualización de publicaciones.

Este mecanismo garantiza un control adecuado sobre la información y los recursos disponibles en el sistema.

Backend y FastAPI

El backend se encarga de la lógica del sistema, el manejo de solicitudes y la interacción con la base de datos. Para este proyecto se empleará **FastAPI**, un framework de Python que permite construir APIs de manera rápida, eficiente y con alto rendimiento, gracias al uso de tipado estático y validación automática de datos.

Frontend con Reflex

En lugar de desarrollar interfaces con HTML y CSS tradicionales, se utilizará **Reflex**, un framework de Python que permite construir aplicaciones web completas con código puramente en Python. Reflex abstrae la necesidad de trabajar directamente con tecnologías de frontend como React o JavaScript, facilitando la creación de interfaces modernas y reactivas mediante componentes declarados en Python.

Base de datos relacional

La persistencia de la información se llevará a cabo en una base de datos relacional (ej. PostgreSQL o MySQL). Estas bases de datos permiten organizar la información en tablas con relaciones bien definidas, lo que resulta adecuado para representar usuarios, roles y publicaciones dentro de un sistema de blog.

Seguridad en aplicaciones web

La seguridad es un aspecto crítico en cualquier sistema. Para este proyecto se implementará el hashing de **contraseñas** y mecanismos de autenticación (como JWT) que garanticen la confidencialidad y protección de los datos de los usuarios.

Requerimientos del Proyecto

Requerimientos funcionales

El sistema deberá cumplir con las siguientes funcionalidades principales:

1. **Gestión de usuarios**
   - **Registro** de nuevos usuarios con validación de datos.
   - **Inicio de sesión** con credenciales seguras.
   - **Manejo de contraseñas** mediante hashing.
2. **Control de roles**
   - **Administrador:** podrá gestionar usuarios (activar, desactivar, eliminar) y moderar publicaciones.
   - **Autor:** podrá crear, editar y eliminar únicamente sus publicaciones.
   - **Lector:** podrá visualizar publicaciones de otros usuarios sin opción de modificación.
3. **Gestión de publicaciones (CRUD)**
   - **Crear** nuevas publicaciones con título, contenido y fecha de creación.
   - **Listar** todas las publicaciones existentes.
   - **Visualizar** publicaciones individuales.
   - **Editar** publicaciones propias.
   - **Eliminar** publicaciones propias.
4. **Interacción de usuarios**
   - **Visualización** de los posts de todos los usuarios autenticados.

Requerimientos no funcionales

Además de las funcionalidades, el sistema deberá cumplir con criterios de calidad:

1. **Seguridad**
   - Hashing de contraseñas y validación segura de sesiones.
   - Control de accesos basado en roles.
2. **Escalabilidad**
   - La arquitectura deberá permitir agregar nuevas funcionalidades (ej. comentarios, categorías, “likes”) sin necesidad de modificar el núcleo del sistema.
3. **Usabilidad**
   - Interfaz construida en **Reflex (Python)**, intuitiva y de fácil navegación.
   - Diseño responsive para acceso desde distintos dispositivos.
4. **Persistencia de datos**
   - Uso de una base de datos relacional para garantizar integridad y consistencia.
5. **Disponibilidad**
   - El sistema debe soportar múltiples usuarios concurrentes sin pérdida de datos.

Arquitectura del Sistema

La arquitectura del sistema Bitácora Digital se basa en una estructura de tres capas principales: **frontend, backend y base de datos**. Cada capa cumple un rol específico y se comunica con las demás a través de interfaces bien definidas.

1. **Frontend** (Interfaz de Usuario)
    - Desarrollado con **Reflex (Python)**, el cual permite construir aplicaciones web completas desde Python, sin necesidad de trabajar directamente con HTML, CSS o JavaScript.
    - Se encarga de presentar las vistas al usuario y de capturar sus interacciones (registro, inicio de sesión, creación y edición de publicaciones).
    - La comunicación con el backend se realiza mediante solicitudes HTTP (API REST).

2. **Backend** (Lógica del Sistema)
    - Implementado con FastAPI, un framework moderno de Python para el desarrollo de APIs rápidas y seguras.
    - Se encarga de procesar las solicitudes del frontend, aplicar las reglas de negocio, gestionar la autenticación y validar permisos de acuerdo a los roles de usuario.
    - Proporciona los endpoints necesarios para operaciones CRUD sobre usuarios y publicaciones.

3. **Base de Datos** (Persistencia de Datos)
    - Se utilizará un gestor de base de datos relacional (ej. PostgreSQL o MySQL).
    - Contendrá las tablas necesarias para almacenar usuarios, roles y publicaciones.
    - Garantiza la integridad, consistencia y persistencia de la información.

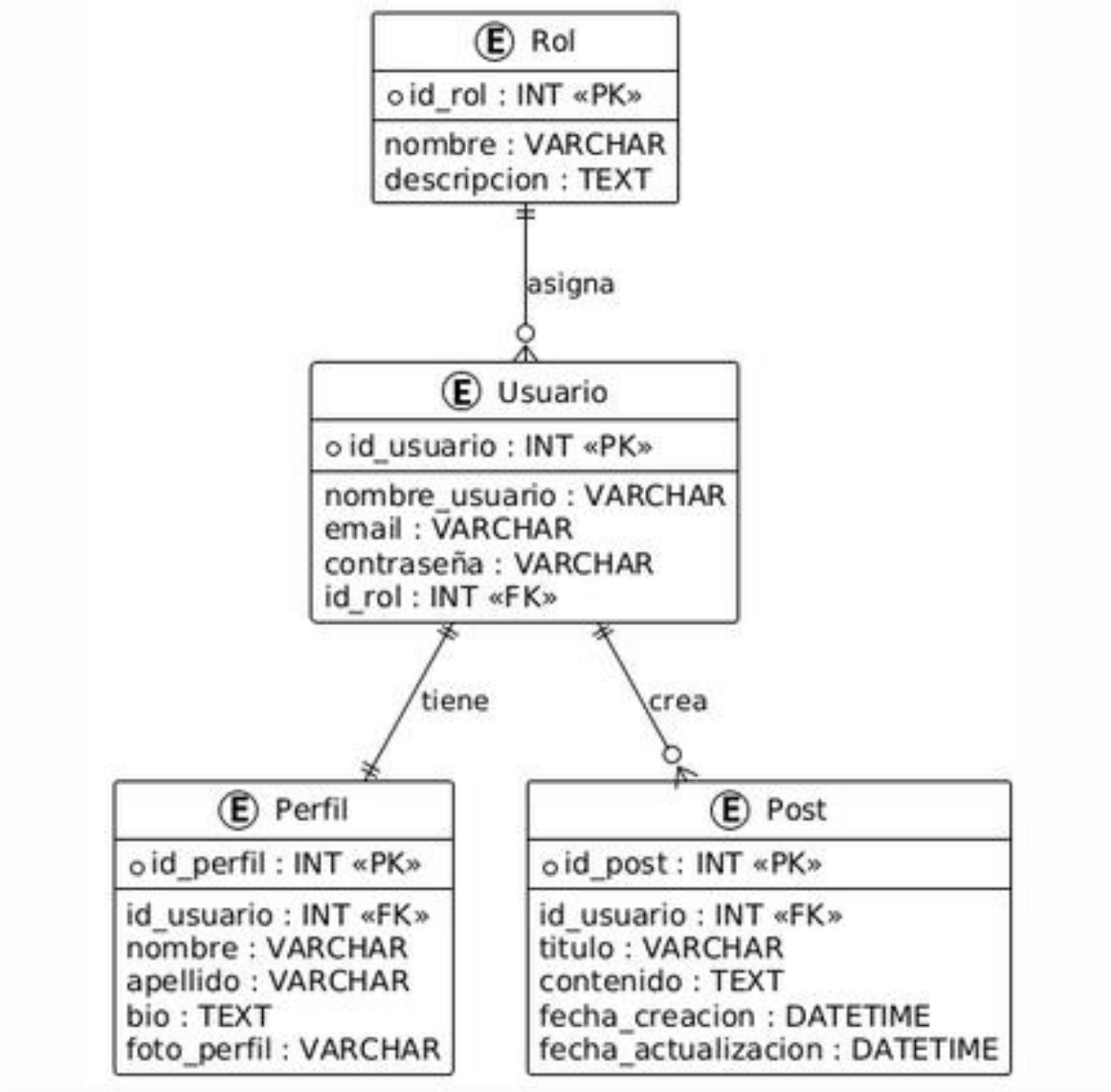
Comunicación entre componentes

- El **usuario** interactúa con la aplicación a través de la interfaz construida en **Reflex.**
- **Reflex** envía solicitudes al **backend** en **FastAPI**, que procesa los datos y aplica las validaciones correspondientes.
- El **backend** se comunica con la **base de datos** relacional para realizar operaciones **CRUD** y devolver las respuestas al **frontend**.
- Finalmente, el **frontend** muestra al **usuario** los resultados de sus acciones.


Frontend (Reflex)
Componentes UI / UX
Estados (Reactivos)
Router Navegación
HTTP / Rest API
Backend (FastAPI)
Routers (End Points)
Autenticacion (JWT, Auth)
Validación (Pydantic)
Controladores
Servicios
SQL / ORM
Base De Datos PostgreSQL
Usuarios (Users)
Publicaciones (Posts)
Roles (roles)

Modelado De Datos

Modelo Entidad Relación



Entidades y atributos

Rol

- **id_rol:** INT, PK → Identificador único del rol.
- **nombre:** VARCHAR → Nombre del rol (ej. administrador, autor, lector).
- **descripcion:** TEXT → Descripción detallada del rol y sus permisos.

Usuario

- **id_usuario:** INT, PK → Identificador único del usuario.
- nombre_usuario: VARCHAR → Nombre único para el inicio de sesión.
- **email:** VARCHAR → Correo electrónico del usuario.
- **contraseña:** VARCHAR → Contraseña cifrada para la autenticación.
- **id_rol:** INT, FK → Relación con la entidad Rol.

Perfil

- **id_perfil:** INT, PK → Identificador único del perfil.
- **id_usuario:** INT, FK → Relación con la entidad Usuario.
- **nombre:** VARCHAR → Nombre real del usuario.
- **apellido:** VARCHAR → Apellido del usuario.
- **bio:** TEXT → Descripción o biografía del usuario.
- **foto_perfil:** VARCHAR → Ruta o URL de la foto de perfil.

Post

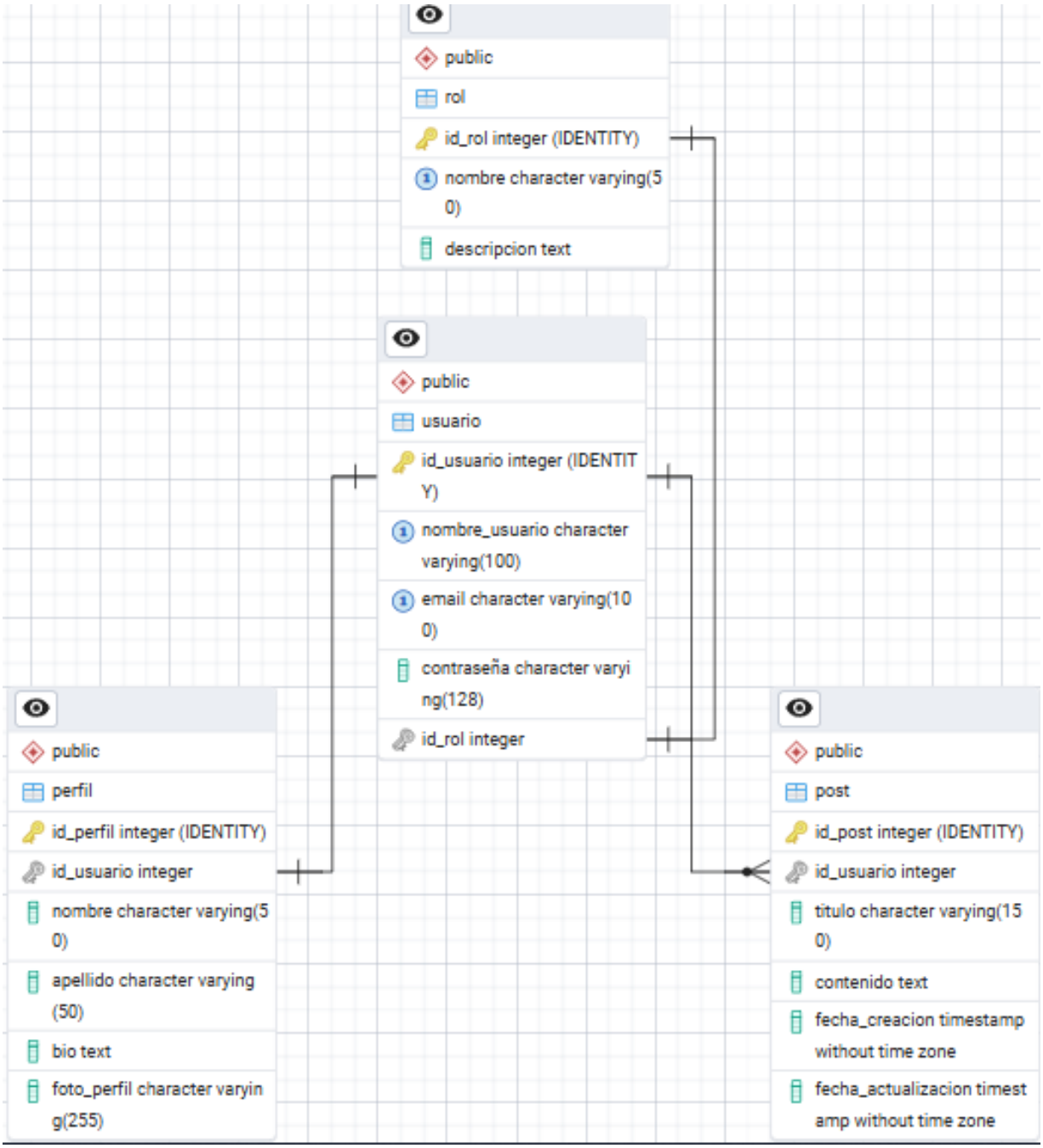
- **id_post:** INT, PK → Identificador único del post.
- **id_usuario:** INT, FK → Relación con la entidad Usuario.
- **titulo:** VARCHAR → Título del post.
- **contenido:** TEXT → Contenido principal del post.
- **fecha_creacion:** DATETIME → Fecha y hora en la que se creó el post.
- **fecha_actualizacion:** DATETIME → Fecha y hora de la última modificación.

Relaciones

1. **Rol – Usuario**
   - Relación 1:N (un rol puede asignarse a muchos usuarios).
   - Un usuario pertenece a un único rol.
2. **Usuario – Perfil**
   - Relación 1:1 (cada usuario tiene un único perfil asociado).
   - El perfil almacena información personal adicional del usuario.
3. **Usuario – Post**
   - Relación 1:N (un usuario puede crear muchos posts).

Cada post pertenece a un único usuario.

Modelo Físico de datos



Tablas y campos

1. **Tabla: rol**
    - **id_rol:** INTEGER IDENTITY, PK → Identificador único del rol.
    - **nombre:** VARCHAR(50) → Nombre del rol (ej. Administrador, Autor, Lector).
    - **descripcion:** TEXT → Breve explicación del rol y permisos asociados.

- **Relaciones:**
    - Un rol puede ser asignado a muchos usuarios (1:N).

2. **Tabla: usuario**
    - **id_usuario:** INTEGER IDENTITY, PK → Identificador único del usuario.
    - **nombre_usuario:** VARCHAR(100) → Nombre único para inicio de sesión.
    - **email:** VARCHAR(100) → Dirección de correo electrónico.
    - **contraseña:** VARCHAR(128) → Contraseña en formato hash.
    - **id_rol:** INTEGER, FK → Relación con la tabla rol.

- **Relaciones:**
    - Cada usuario pertenece a un único rol (N:1).
    - Cada usuario tiene un perfil asociado (1:1).
    - Cada usuario puede crear múltiples posts (1:N).

3. **Tabla: perfil**
    - **id_perfil:** INTEGER IDENTITY, PK → Identificador único del perfil.
    - **id_usuario:** INTEGER, FK → Relación con la tabla usuario.
    - **nombre:** VARCHAR(50) → Nombre personal del usuario.
    - **apellido:** VARCHAR(50) → Apellido personal del usuario.
    - **bio:** TEXT → Descripción o biografía.
    - **foto_perfil:** VARCHAR(255) → URL o ruta de la foto de perfil.

- **Relaciones:**
    - Cada perfil pertenece a un único usuario (1:1).

4. **Tabla: post**
    - **id_post:** INTEGER IDENTITY, PK → Identificador único del post.
    - **id_usuario:** INTEGER, FK → Relación con la tabla usuario.
    - **titulo:** VARCHAR(150) → Título del post.
    - **contenido:** TEXT → Texto del post.
    - **fecha_creacion:** TIMESTAMP → Fecha y hora de creación.
    - **fecha_actualizacion:** TIMESTAMP → Fecha y hora de la última modificación.

- **Relaciones:**
    - Cada post pertenece a un único usuario (N:1).

Diccionario de datos

1. Tabla: usuario

Contiene la información principal de los usuarios registrados en el sistema.

Campo	Tipo de Dato	Longitud	Nulo	PK	FK	Descripción
id_usuario	integer	-	No	Sí	No	Identificador único del usuario.
nombre_usuario	character varying	100	No	No	No	Nombre de usuario para autenticación y visualización.
email	character varying	100	No	No	No	Correo electrónico único del usuario.
contraseña	character varying	128	Sí	No	No	Contraseña encriptada para acceso.
id_rol	integer	-	No	No	Sí	Relación con la tabla **rol** (rol asignado al usuario).

2. Tabla: rol

Define los roles que puede tener un usuario en el sistema.

Campo	Tipo de Dato	Longitud	Nulo	PK	FK	Descripción
id_rol	integer	-	No	Sí	No	Identificador único del rol.
nombre	character varying	50	No	No	No	Nombre del rol (ej: administrador, usuario).
descripcion	text	-	Sí	No	No	Descripción del rol y sus permisos.

3. Tabla: perfil

Contiene información extendida del usuario.

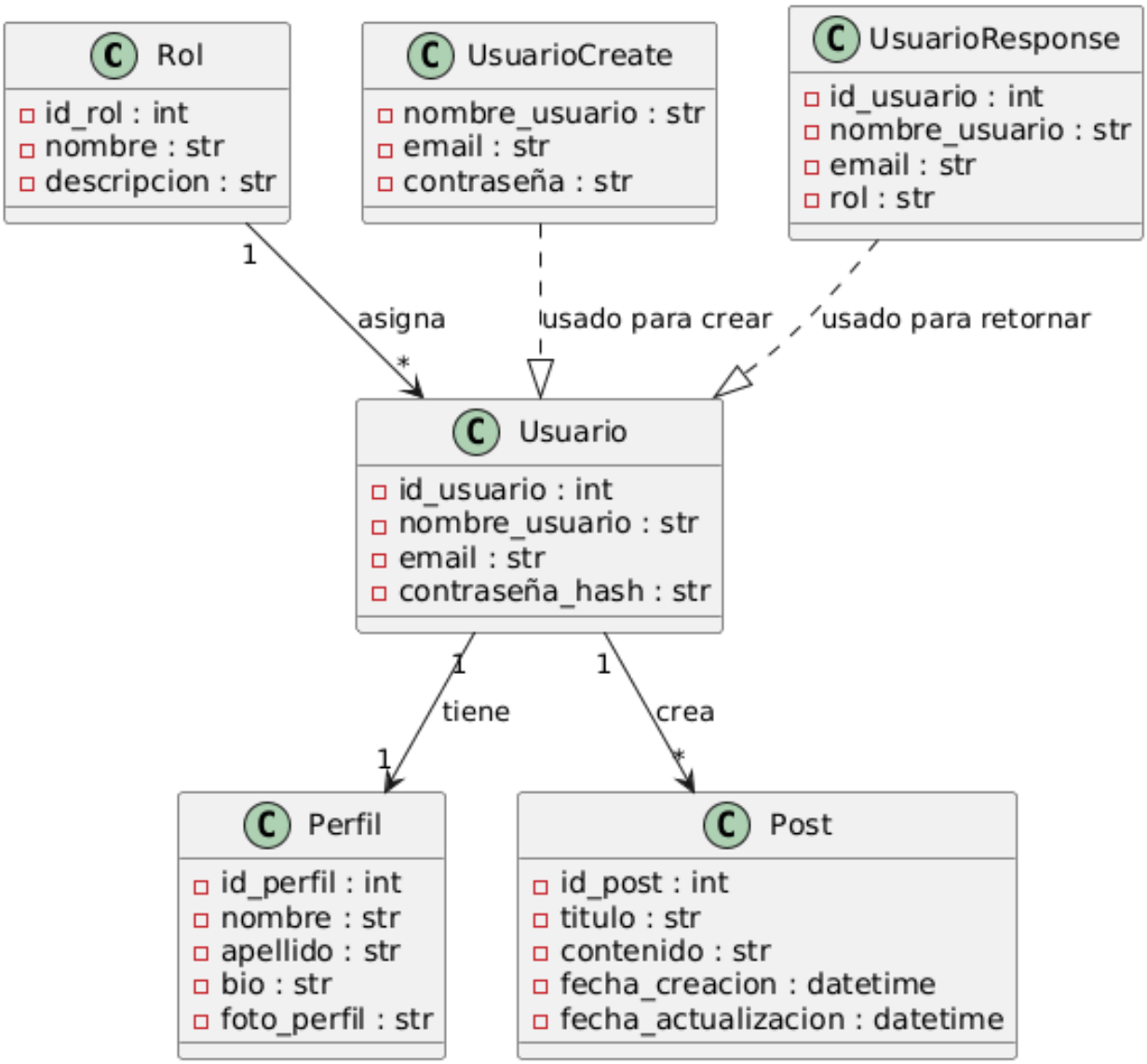
Campo	Tipo de Dato	Longitud	Nulo	PK	FK	Descripción
id_perfil	integer	-	No	Sí	No	Identificador único del perfil.
id_usuario	integer	-	No	No	Sí	Relación con la tabla **usuario**.
nombre	character varying	50	No	No	No	Nombre real del usuario.
apellido	character varying	50	Sí	No	No	Apellido del usuario.
bio	text	-	Sí	No	No	Descripción breve o biografía del usuario.
foto_perfil	character varying	255	Sí	No	No	Ruta o URL de la foto de perfil.

4. Tabla: post

Almacena las publicaciones realizadas por los usuarios.

Campo	Tipo de Dato	Longitud	Nulo	PK	FK	Descripción
id_post	integer	-	No	Sí	No	Identificador único del post.
id_usuario	integer	-	No	No	Sí	Relación con la tabla **usuario** (autor del post).
titulo	character varying	150	No	No	No	Título de la publicación.
contenido	text	-	No	No	No	Texto principal del post.
fecha_creacion	timestamp without tz	-	No	No	No	Fecha y hora en que se creó el post.
fecha_actualiza	timestamp without tz	-	Sí	No	No	Última fecha de actualización del post.

Diagrama de clases



1. **Rol**

- **Atributos:**
  - **id_rol :** int → Identificador único del rol.
  - **nombre :** str → Nombre del rol (ej. admin, usuario).
  - **descripcion :** str → Breve descripción del rol.
- **Relaciones:**
  - Un Rol puede estar asignado a múltiples Usuarios (1 a muchos).

2. **Usuario**

- **Atributos:**
  - **id_usuario :** int → Identificador único del usuario.
  - **nombre_usuario :** str → Nombre de usuario para login.
  - **email** : str → Correo electrónico único.
  - **contraseña_hash :** str → Contraseña almacenada de forma segura (encriptada).
- **Relaciones:**
  - Un Usuario pertenece a un Rol.
  - Un Usuario tiene asociado un único Perfil (1 a 1).
  - Un Usuario puede crear múltiples Posts (1 a muchos).

3. **UsuarioCreate**

- **Función:** Clase de soporte usada en los endpoints de creación de usuario.
- **Atributos:**
  - **nombre_usuario :** str
  - **email :** str
  - **contraseña :** str (solo para registro, nunca se retorna).
- **Relación:** Se utiliza para instanciar un nuevo Usuario.

4. **UsuarioResponse**

- **Función:** Clase usada para devolver información de usuarios a través de la API.
- **Atributos:**
  - **id_usuario :** int
  - **nombre_usuario :** str
  - **email :** str
  - **rol :** str (para mostrar el rol asociado).
- **Relación:** Representa un Usuario sin exponer información sensible como la contraseña.

5. **Perfil**

- **Atributos:**
  - **id_perfil :** int → Identificador único.
  - **nombre :** str → Nombre del usuario real.
  - **apellido :** str → Apellido del usuario.
  - **bio :** str → Descripción personal.
  - **foto_perfil :** str → URL o ruta de la foto de perfil.
- **Relación:** Cada Usuario tiene un Perfil (1 a 1).

6. **Post**

- **Atributos:**
  - **id_post :** int → Identificador único del post.
  - **titulo :** str → Título del post.
  - **contenido :** str → Contenido del post.
  - **fecha_creacion :** datetime → Fecha en que se creó el post.
  - **fecha_actualizacion :** datetime → Última fecha de edición.
- **Relación:** Un Usuario puede crear múltiples Posts (1 a muchos).