Documentación Técnica Sistema de Infracciones

Supuestos

1. **Relaciones entre Entidades:**
    - **Persona y Vehículo:** Se asume que cada vehículo está asociado a una única persona.
    - **Oficiales:** Se asume que cada oficial tiene un identificador único y su nombre para autenticación.
    - **Integridad Referencial:** Se asegura que al eliminar una persona, todos sus vehículos relacionados se eliminan también para mantener la integridad de los datos.
2. **API de Cargar Infracciones:**
    - Se asume que cada llamada a la API incluye un token de autenticación válido en el encabezado.
    - Se verifica la existencia del vehículo antes de registrar una infracción.
    - Se manejan adecuadamente los códigos de respuesta HTTP: 200 para éxito, 404 para no encontrado y 500 para errores internos del servidor.
3. **Generación de Informes:**
    - Se asume que el parámetro de correo electrónico es válido y corresponde a una persona registrada.
    - No se requiere autenticación para este endpoint, ya que solo proporciona datos específicos asociados al correo proporcionado.

Claves de Usuarios Administrativos de la BD

Las credenciales para la administración de la base de datos deben configurarse en las variables de entorno del proyecto. Estas pueden estar definidas en un archivo .env o directamente en la configuración del entorno de despliegue. Un ejemplo de configuración podría ser:

DATABASE_URL=postgresql://admin_user:<password>@localhost:5432/n5_db

Idioma de código fuente y endpoints

El código de la aplicación está en inglés para alinearse con las mejores prácticas y estándares de la industria del desarrollo de software, donde el inglés es el idioma predominante. Esto facilita la colaboración internacional y el mantenimiento del código por parte de desarrolladores de diferentes partes del mundo.

Sin embargo, los endpoints de la API están en español para reflejar el contexto específico del challenge, asegurando que la funcionalidad sea fácilmente comprensible para los usuarios finales que probablemente sean hispanohablantes, como los oficiales de policía que utilizarán la aplicación.

Sistema de Autenticación Basado en Tokens

Esta aplicación utiliza un sistema de autenticación basado en tokens para asegurar sus endpoints. El sistema de autenticación se basa en JSON Web Tokens (JWT) para verificar la identidad de los usuarios (oficiales). El badge_number de un oficial se utiliza como identidad dentro del token JWT.

Código del Sistema de Autenticación

Decorador de Autenticación

El archivo app/auth/decorators.py contiene un decorador officer_required que asegura que solo los oficiales autenticados puedan acceder a ciertos endpoints.

Este decorador verifica el token JWT y adjunta el objeto officer al contexto de la solicitud.

Generación de Tokens en Flask-Admin

El archivo app/admin/__init__.py contiene la configuración de Flask-Admin y la lógica para generar tokens JWT para los oficiales:

Generación de un Nuevo Token desde Flask-Admin

Para generar un nuevo token desde Flask-Admin, sigue estos pasos:

- Abre Flask-Admin y navega a la vista de oficiales.
- Al crear o editar un oficial, el token JWT se generará automáticamente utilizando el badge_number del oficial.
- Puedes ver en la lista de oficiales, los tokens de cada oficial necesarios para autenticación del API.

Docker, Docker-Compose y Docker Hub

Para cumplir con el requerimiento de crear una imagen Docker que permita ejecutar todos los componentes de la solución (servidor de aplicaciones, base de datos, etc.) y subirla a un repositorio público de Docker, se han implementado las siguientes estrategias:

Solución

Se crearon dos archivos `docker-compose.yml`, uno para desarrollo (`docker-compose.yml`) y otro para producción (`docker-compose.prod.yml`). Este enfoque permite mantener el código fuente separado del entorno de datos, facilitando las actualizaciones y el desarrollo local.

Ventajas

- **Flexibilidad en el desarrollo:** El archivo `docker-compose.yml` para desarrollo monta el código fuente local en el contenedor, permitiendo a los desarrolladores modificar el código y ver los cambios en tiempo real sin necesidad de reconstruir la imagen Docker.
- **Facilidad de ejecución en producción:** El archivo `docker-compose.prod.yml` para producción utiliza imágenes Docker precargadas con la aplicación y la base de datos, garantizando que la solución se ejecute de manera consistente y sin configuraciones adicionales.
- **Separación de preocupaciones:** Mantener archivos separados para desarrollo y producción asegura que los entornos no se mezclen, permitiendo un ciclo de desarrollo limpio y una configuración de producción robusta.
-

Ejecución en Producción desde Docker Hub (se puede probar localmente)

Para ejecutar la aplicación en un entorno de producción utilizando imágenes Docker desde Docker Hub, sigue estos pasos:

1. Asegúrate de tener Docker y Docker Compose instalados.

Ejecuta los siguientes comandos para iniciar los servicios usando el archivo de configuración de producción:

```
docker-compose -f docker-compose.prod.yml up
```

Esto descargará las imágenes necesarias desde Docker Hub y levantará tanto la aplicación web como la base de datos con los datos precargados, permitiendo una ejecución inmediata y consistente de la solución en un entorno de producción.

Empaquetado en un Virtual Env

El requerimiento de empaquetar la solución en un entorno virtual (Virtual Env) no se abordó explícitamente debido a que se implementó una solución basada en contenedores Docker. A continuación, se explica el enfoque y sus ventajas:

Razones para no usar un Virtualenv tradicional

1. **Contenedores Docker como alternativa:**
    - **Aislamiento y consistencia:** Los contenedores Docker proporcionan un aislamiento completo del entorno de ejecución, asegurando que la aplicación se ejecute de la misma manera en cualquier máquina, independientemente de las dependencias del sistema operativo o de las versiones de las bibliotecas.
    - **Reproducibilidad:** Docker facilita la creación de entornos reproducibles, eliminando los problemas de "funciona en mi máquina" que a menudo se encuentran con entornos virtuales tradicionales.
    - **Facilidad de uso:** Con Docker Compose, es sencillo levantar toda la pila de servicios (aplicación web, base de datos, etc.) con un solo comando, lo que simplifica el desarrollo y la implementación.
2. **Infraestructura para producción:**
    - **Preparación para Producción:** El uso de Docker es una práctica estándar en entornos de producción modernos. Docker Compose permite definir, ejecutar y escalar servicios de manera eficiente.
    - **Despliegue Simplificado:** Al subir las imágenes Docker a un repositorio público, cualquier persona puede descargar y ejecutar la aplicación completa con datos precargados sin configuraciones adicionales.

Arquitectura AWS Propuesta

Para el despliegue de la aplicación en un entorno de producción, se recomienda la siguiente arquitectura en AWS:

1. **Amazon ECS (Elastic Container Service):**
    - Utilizado para desplegar y orquestar los contenedores Docker de la aplicación.
    - Facilita la escalabilidad y gestión de las cargas de trabajo.
2. **Amazon RDS (Relational Database Service):**
    - Para gestionar la base de datos PostgreSQL de forma escalable y segura.
    - Proporciona alta disponibilidad y recuperación ante desastres.
3. **Amazon S3 (Simple Storage Service):**
    - Para almacenamiento de respaldos y archivos estáticos.
    - Garantiza durabilidad y disponibilidad de los datos.

4. **Amazon CloudWatch:**
    - Para monitoreo y logging de la aplicación.
    - Permite rastrear métricas y generar alarmas para la administración proactiva del sistema.

Justificación

- **ECS y RDS** permiten escalar fácilmente los recursos según la demanda, asegurando alta disponibilidad y desempeño.
- **S3** proporciona almacenamiento seguro y redundante para respaldos críticos.
- **CloudWatch** asegura un monitoreo constante y detallado, permitiendo una administración eficiente del sistema.