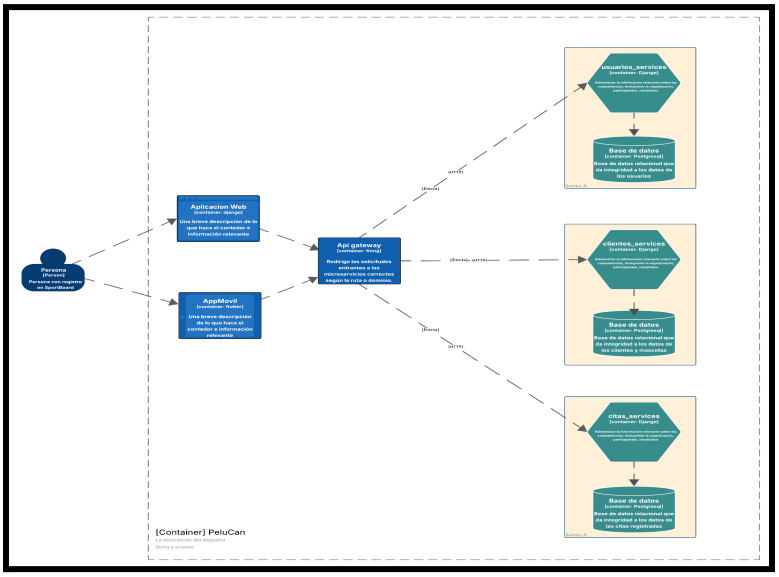
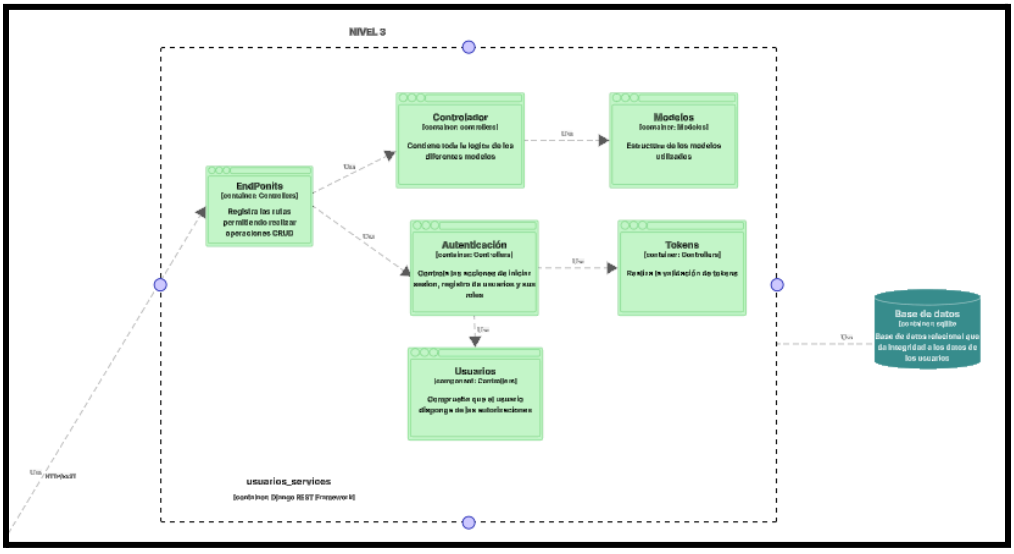
1. Resumen del microservicio creado

El sistema de gestión de citas para peluquería canina fue diseñado bajo una arquitectura de microservicios contenerizada con Docker, garantizando escalabilidad, mantenibilidad y despliegue eficiente.

Cada microservicio funciona de manera independiente, comunicándose a través de Kong API Gateway, que actúa como intermediario central para las solicitudes REST.

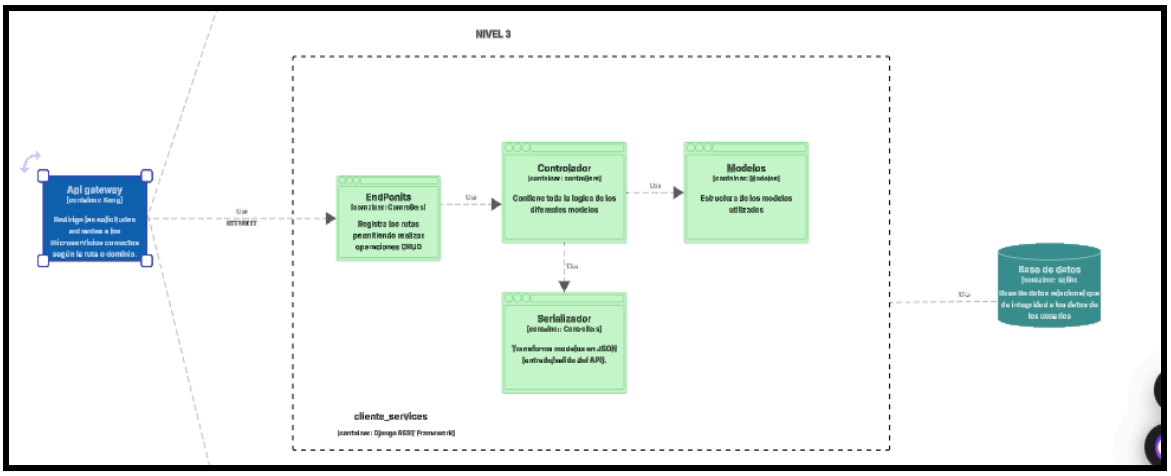


Nivel 2: Diagrama de Contenedores del sistema PelusaAPP
El sistema está compuesto por los siguientes servicios:
- **UsuarioService**: Gestiona autenticación, roles y credenciales, aparte administra la información del peluquero.



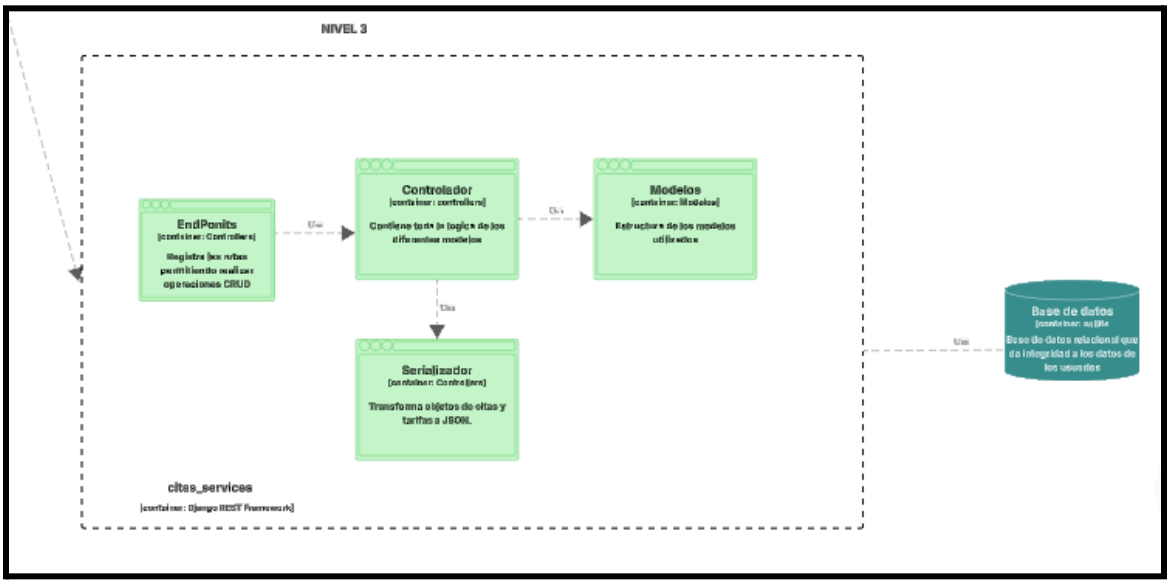
Nivel 3: Diagrama de Componentes (usuarioService)

- **ClienteService**: Administra la información del cliente y sus mascotas.



Nivel 3: Diagrama de Componentes (ClienteService)

- **CitaService**: Controla la gestión de citas, horarios y asignación de peluqueros.



Nivel 3: Diagrama de Componentes (citasService)

2. Integración mediante APIs

Los servicios se comunican mediante API REST utilizando peticiones HTTP en formato JSON. La autenticación se maneja mediante JWT (JSON Web Tokens), emitidos por el UsuarioService y validados por los otros servicios a través de Kong, actuando como API Gateway, centralizando el tráfico entre los microservicios.

Ejemplo de integración:

- POST /api/usuarios/register/

```
{
  "username": "maria",
  "email": "maria@example.com",
  "password": "12345",
  "rol": "cliente"
}
```

- POST /api/clientes/

```
{
  "user_id": 1,
  "telefono": "0998877665",
  "direccion": "Calle Los Álamos 23"
}
```

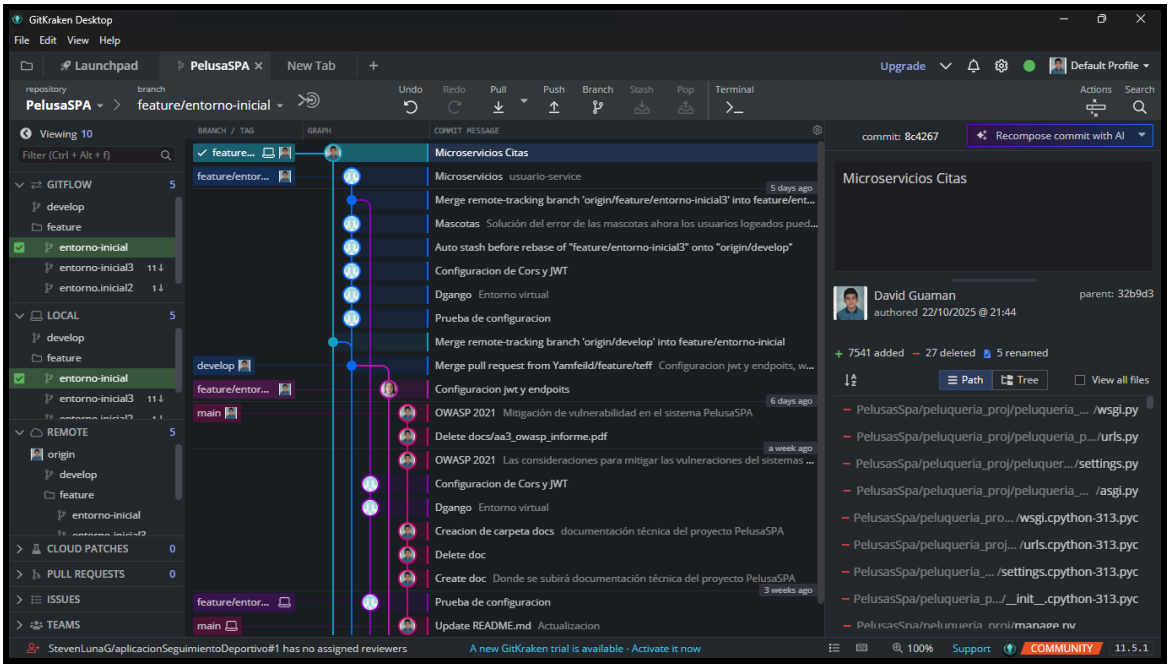
- POST /api/citas/

```
{
  "cliente_id": 1,
  "mascota_id": 2,
  "fecha": "2025-11-01",
  "hora": "09:30",
  "servicio": "Corte de pelo",
  "estado": "pendiente"
}
```

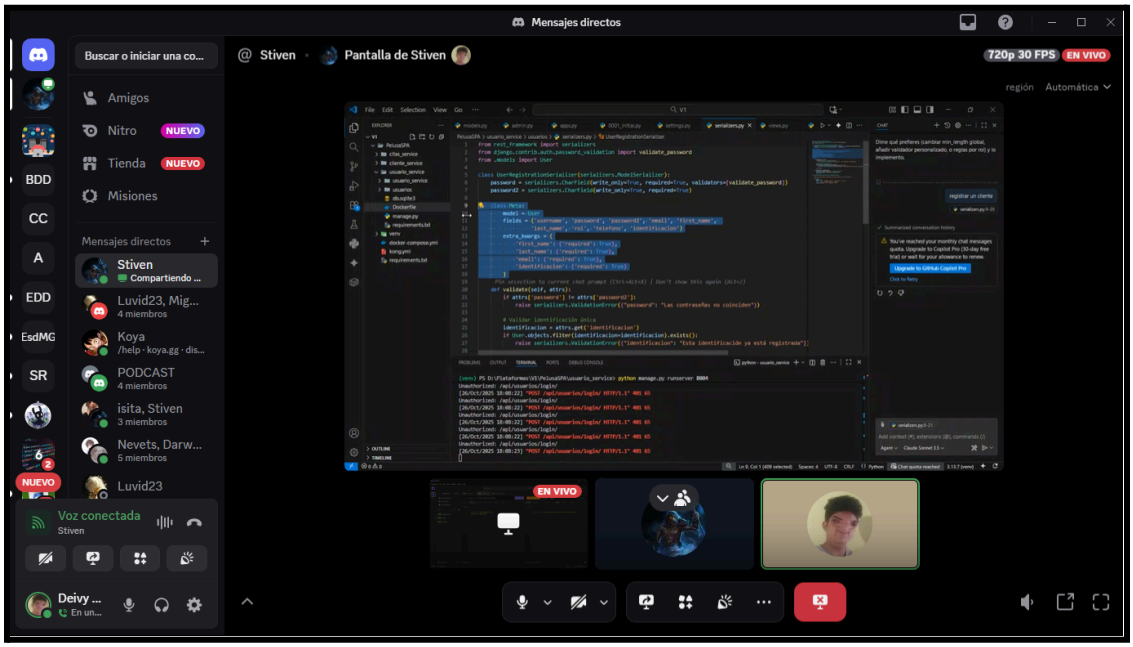
3. Herramientas colaborativas utilizadas

Se emplearon las siguientes herramientas:

- **GitHub/Git Kraken**: Control de versiones y repositorio del código fuente.



- **Discord**: Comunicación y coordinación del equipo.



Repositorio Git:https://github.com/Yamfeild/PelusaSPA.git

4. Buenas prácticas y aprendizajes

Buenas prácticas aplicadas:

- Desacoplamiento por dominio: Cada microservicio tiene su propia base de datos y lógica independiente, facilitando el mantenimiento.

- Documentación de API con Insomnia: Ayuda a tener una visualización

complementaria para ver el funcionamiento de los endpoints

Reflexión personal:
La arquitectura de microservicios permitió dividir el proyecto, aumentar la productividad y facilitar el mantenimiento, el reto principal fueron la autenticación centralizada y la comunicación entre servicios.

5. Preguntas de reflexión

1. ¿Qué ventajas observas en la integración mediante APIs REST respecto a un monolito tradicional?
Permite una mayor escalabilidad, ya que cada servicio puede desplegarse y escalarse de forma independiente cada microservicio puede desarrollarse con diferentes lenguajes o frameworks y facilita el trabajo colaborativo entre equipos, ya que pueden desarrollar, probar y mantener servicios separados sin afectar al resto.

2. ¿Cómo aportan las herramientas colaborativas a la gestión técnica de los microservicios?
Facilita la trazabilidad de los cambios, mejoró la comunicación entre equipos y permiten un control más eficiente en el desarrollo, también la coordinación en entornos distribuidos integrando tareas, documentación y control de versiones.

3. ¿Qué riesgos existen al distribuir un sistema en varios microservicios y cómo pueden mitigarse?
Los principales riesgos son la latencia en la comunicación y la complejidad en la gestión de dependencias, se mitigan aplicando patrones como API Gateway, autenticación centralizada y monitoreo continuo.

6. Bibliografía

- Newman, S. (2021). Building Microservices. O'Reilly Media.
- Richardson, C. (2022). Microservices Patterns. Manning Publications.
- OWASP Foundation (2023). OWASP Secure Microservices Design.
- Docker Inc. Docker Documentation.
- GitHub Docs. Collaborative Development Workflows.