ISFT N°151 – Tecnicatura Superior en Análisis de Sistemas

Materia: Prácticas Profesionalizantes III

Autores: Valentina Tiberio y Martina Lapenta

Docente: Matías Gastón Santiago

Año: 2025

🐾 RescatePET - Guía para ejecutar el proyecto

---

1. Requisitos previos

Antes de ejecutar el proyecto, asegurate de tener instaladas las siguientes herramientas:

- **Python 3.11+**
- **Flask** (framework principal del backend)
- **SQLite** (base de datos por defecto)
- **Visual Studio Code** o cualquier editor de texto
- **Git** y cuenta en **GitHub** (para control de versiones)

---

2. Instalación

Clonar el repositorio desde GitHub:

```
git clone https://github.com/ValentinaTiberio/isft151-pp3.git
```

Entrar al directorio del proyecto:

```
cd isft151-pp3/fp
```

Instalar las dependencias necesarias:

```
pip install flask flask_sqlalchemy flask_cors jwt
```

Crear la base de datos:

Ejecutar el script SQL ubicado en

`/doc/rescatepet_db.sql`
Este archivo genera las tablas principales: usuarios, animales, adopciones y hogares de tránsito.

---

3. Cómo ejecutar el proyecto

-Desde la carpeta /fp, iniciar el servidor con:

```
python -m backend.main
```

El backend levanta desde `fp/backend/main.py`

-El sistema se ejecutará en:

```
http://127.0.0.1:5000
```

El frontend se carga automáticamente al acceder a esa dirección desde el navegador.

---

4. Endpoints principales

Endpoint	Método	Descripción
`/api/users/register`	POST	Registrar nuevo usuario
`/api/users/login`	POST	Iniciar sesión y obtener token JWT
`/api/animals/`	GET / POST	Listar o registrar animales
`/api/adopciones/`	GET / POST	Listar o crear solicitudes de adopción
`/api/adopciones/usuario/<id>`	GET	Listar solicitudes de adopción por usuario
`/api/adopciones/<id>`	PUT	Actualizar estado de solicitud (Aprobada/Rechazada)
`/api/transitos/`	GET / POST	Listar o asignar hogares de tránsito
`/api/transitos/usuario/<id>`	GET	Listar solicitudes de tránsito por usuario
`/api/transitos/<id>`	PUT	Actualizar estado de solicitud de tránsito
`/api/reportes/`	GET	Consultar estadísticas de rescates y adopciones

Documentación detallada de la WebAPI

Módulo: UserService

Este módulo contiene los procedimientos remotos encargados de la **gestión de usuarios**, incluyendo registro, autenticación y listado.

Cada endpoint expone una función del backend que procesa la información, opera sobre la base de datos y devuelve una respuesta JSON al frontend.

1. Registrar Usuario
Procedimiento remoto: createUser
Endpoint: /api/users/register
Método HTTP: POST
Formato Serialización: JSON
Cabecera entrada: –
Cabecera salida: –
Estructura de datos IN:

```
{
  "username": "martina",
  "email": "martina@mail.com",
  "password": "1234"
}
```

Estructura de datos OUT:

```
{
  "id": 3,
  "username": "martina",
  "email": "martina@mail.com"
}
```

Estructura de datos ERR:

```
{
  "type": "error",
  "description": "El email ya está registrado"
}
```

2. Login de usuario
Procedimiento remoto: authenticateUser
Endpoint: /api/users/login
Método HTTP: POST
Formato: JSON
Cabecera entrada: –

Cabecera salida: –
Estructura de datos IN:

```
{
  "email": "martina@mail.com",
  "password": "1234"
}
```

Estructura de datos OUT:

```
{
  "token": "jwt-generated-token",
  "user": {
    "id": 3,
    "username": "martina",
    "email": "martina@mail.com"
  }
}
```

Estructura de datos ERR:

```
{
  "type": "error",
  "description": "Credenciales inválidas"
}
```

3. Obtener todos los usuarios

Endpoint: /api/users/
Método HTTP: GET
Formato Serialización: JSON
Cabecera entrada: -
Cabecera salida: -
Estructura de datos IN:-

Estructura de datos OUT:

```
[
  {
    "id": 1,
    "username": "admin",
    "email": "admin@example.com"
  },
  {
    "id": 2,
    "username": "martina",
    "email": "martina@gmail.com"
  }
```

]

Estructura de datos ERR:
{
  "type": "error",
  "description": "No se pudieron obtener los usuarios."
}

Módulo: AnimalService

Gestiona animales: creación, listado y entrega de imágenes.

4. Registrar animal

Procedimiento remoto: createAnimal
Endpoint: /api/animals/
Método: POST
Cabecera entrada: –
Cabecera salida: –

Formato: multipart/form-data
Estructura de datos IN:
nombre = "Coco"
especie = "Perro"
estado = "En adopción"
foto = (archivo imagen jpg/png)

Estructura de datos OUT:
{
  "id": 8,
  "nombre": "Coco",
  "especie": "Perro",
  "estado": "En adopción",
  "foto": "coco_7281.jpg"
}

Estructura de datos ERR:
{
  "type": "error",
  "description": "Faltan datos obligatorios"
}

5. Listar animales
Endpoint: /api/animals/
Método: GET

Cabecera entrada: –
Cabecera salida: –

Formato: multipart/form-data
Estructura de datos IN:-

Estructura de datos OUT:

```
[
  {
    "id": 1,
    "nombre": "Luna",
    "especie": "Gato",
    "estado": "Encontrado",
    "foto": "luna.jpg"
  }
]
```

Estructura de datos ERR:

```
{
  "type": "error",
  "description": "No se pudieron obtener los animales"
}
```

MÓDULO: AdopcionService

6. Crear solicitud de adopción
Endpoint: /api/adopciones/
Método: POST
Cabecera entrada: –
Cabecera salida: –
Estructura de datos IN:

```
{
  "user_id": 3,
  "animal_id": 8
}
```

Estructura de datos OUT:

```
{
  "id": 20,
  "user_id": 3,
  "animal_id": 8,
  "estado": "Pendiente",
  "fecha_solicitud": "2025-11-08 15:22"
```

}

Estructura de datos ERR:

```
{
  "type": "error",
  "description": "Ya solicitaste adoptar este animal"
}
```


7. Listar solicitudes
Endpoint: /api/adopciones/
Método: GET
Cabecera entrada: –
Cabecera salida: –
Estructura de datos IN:
Estructura de datos OUT:

```
[
  {
    "id": 20,
    "animal_id": 8,
    "animal_nombre": "Coco",
    "usuario_nombre": "martina",
    "estado": "Pendiente",
    "fecha_solicitud": "2025-11-08 15:22"
  }
]
```

Estructura de datos ERR:-

8. Actualizar estado de adopción
Endpoint: /api/adopciones/{id}
Método: PUT
Cabecera entrada: –
Cabecera salida: –
Estructura de datos IN:

```
{
 "estado": "Aprobada"
}
```

Estructura de datos OUT:

```
{
  "id": 20,
  "estado": "Aprobada"
}
```


Estructura de datos ERR:

```
{
  "type": "error",
```

```
  "description": "Solicitud no encontrada"
}
```

MÓDULO: TransitoService

9. Crear solicitud de tránsito

Endpoint: /api/transitos/
Método: POST
Cabecera entrada: –
Cabecera salida: –
Estructura de datos IN:

```
{
  "user_id": 3,
  "animal_id": 8,
  "direccion": "Calle Siempreviva 742",
  "duracion": 7
}
```

Estructura de datos OUT:

```
{
  "mensaje": "Tránsito creado correctamente",
  "id": 12
}
```

Estructura de datos ERR:

```
{
  "type": "error",
  "description": "El usuario ya pidió tránsito para este animal"
}
```

10. Listar todos los tránsitos
Endpoint: /api/transitos/
Método: GET
Cabecera entrada: –
Cabecera salida: –
Estructura de datos IN:-

Estructura de datos OUT:

```
[
  {
    "id": 12,
    "animal_id": 8,
    "animal_nombre": "Coco",
    "usuario_nombre": "martina",
    "direccion": "Calle Siempreviva 742",
    "duracion_dias": 7,
```

```
    "estado": "Pendiente"
  }
]
```

Estructura de datos ERR:-

11. Actualizar estado del tránsito

Endpoint: /api/transitos/{id}
Método: PUT
Cabecera entrada: –
Cabecera salida: –
Estructura de datos IN:

```
{
  "estado": "Aprobado"
}
```

Estructura de datos OUT:

```
{
  "mensaje": "Estado actualizado a Aprobado"
}
```

Estructura de datos ERR:

```
{
  "type": "error",
  "description": "Transito no encontrado"
}
```

MÓDULO: ReportesService

12.Obtener reportes
Endpoint: /api/reportes/
Método: GET
Cabecera entrada: –
Cabecera salida: –
Estructura de datos IN:-

Estructura de datos OUT:

```
{
  "animales_totales": 25,
  "adopciones_pendientes": 4,
  "transitos_activos": 2
}
```

Estructura de datos ERR:

```
{
  "type": "error",
  "description": "No se pudieron generar los reportes"
}
```

5. Estructura del proyecto

```
fp/
 ├── backend/
 │    ├── app/
 │    │    ├── models/
 │    │    ├── routes/
 │    │    ├── services/
 │    ├── instance/
 │    ├── uploads/
 │    ├── main.py
 │
 ├── doc/
 │    ├── api_doc.pdf
 │    ├── rescatepet_db.sql
 │
 ├── frontend/
 │    ├── components/
 │    ├── admin_reportes.html
 │    ├── admin_solicitudes.html
 │    ├── admin_transitos.html
 │    ├── animales.html
 │    ├── index.html
 │    ├── inicio.html
 │    ├── registro.html
 │    ├── solicitudes.html
 │
 ├── release/
 ├── srs/
 ├── media/
```

---

6. Notas finales

- El sistema permite registrar usuarios, animales, adopciones y hogares de tránsito.
- Los datos se almacenan en una base de datos SQLite.
- Las imágenes se guardan en el servidor backend dentro de la carpeta `/uploads`.
- El token JWT es requerido para acceder a ciertas rutas protegidas.
- El frontend está desarrollado con HTML, CSS y JavaScript, con un diseño simple y adaptable.
- El módulo de reportes muestra los totales de animales registrados, adopciones y tránsitos aprobados.
- Se agregó panel de administración, donde se pueden aprobar o rechazar solicitudes de adopción y tránsito.
- Los usuarios pueden ver el estado actualizado de sus solicitudes desde “Mis Solicitudes”.
- El frontend incluye mensajes dinámicos y actualizaciones automáticas de la galería de animales.

---

7. Conclusión

El sistema **RescatePET** promueve la adopción responsable y la colaboración entre usuarios, ofreciendo una gestión completa de animales, adopciones y hogares de tránsito. Su arquitectura modular (Flask + SQLite + frontend HTML/JS) facilita el mantenimiento y la ampliación de funcionalidades futuras, como notificaciones por correo y paneles analíticos más avanzados.