PORTAL DE ADOPCIÓN DE MASCOTAS

Autor: Aarón Prado Darriba
Curso: Curso de Especialización Superior de desenvolvemento
de aplicacións en linguaxe python.
Módulo: Programación Orientada a Objetos
Fecha de entrega: 28 de diciembre de 2025

1. INTRODUCCIÓN

1.1. Descripción general del problema

Los refugios de animales gestionan manualmente el proceso de adopción mediante visitas presenciales, llamadas telefónicas y formularios en papel. Esto genera ineficiencias, alcance limitado de las mascotas disponibles y falta de seguimiento de las solicitudes.

1.2. Solución aportada

Se ha desarrollado un portal web que digitaliza el proceso completo de adopción, permitiendo a los usuarios explorar un catálogo de mascotas, solicitar adopciones mediante un cuestionario online y hacer seguimiento de sus solicitudes. Los administradores del refugio pueden gestionar el catálogo y revisar las solicitudes desde un panel de control centralizado.

2. ANÁLISIS DEL PROBLEMA

2.1. Contexto y necesidades detectadas

El refugio necesita reducir la carga administrativa y llevar un registro de las adopciones. Los adoptantes requieren acceso online a las mascotas disponibles y transparencia en el proceso. La solución debe ser accesible desde cualquier dispositivo y garantizar la seguridad de los datos personales.

2.2. Requisitos funcionales

- Catálogo público de mascotas con filtros de búsqueda
- Sistema de autenticación para usuarios
- Formulario de solicitud con cuestionario de evaluación
- Panel de administración para gestionar mascotas y solicitudes
- Historial de solicitudes para cada usuario
- Cambio automático de estado de mascotas según el proceso

2.3. Requisitos no funcionales

- Seguridad mediante hash de contraseñas y validación de datos
- Interfaz responsive adaptable a móviles y tablets
- Arquitectura modular siguiendo principios POO
- Base de datos relacional para mantener integridad de datos

3. PROPUESTAS DE SOLUCIÓN

3.1. Solución 1: Django

Descripción: Framework completo de Python que incluye ORM, sistema de autenticación, panel de administración automático y motor de plantillas.
Ventajas:

Panel de administración generado automáticamente
Sistema de autenticación robusto incluido
ORM potente con migraciones automáticas

Inconvenientes:

Curva de aprendizaje elevada para proyectos pequeños
Estructura rígida que puede ser excesiva para este proyecto
Mayor consumo de recursos

3.2. Solución 2: Flask

Descripción: Microframework minimalista de Python que proporciona lo esencial para crear aplicaciones web, permitiendo elegir las herramientas específicas necesarias.
Ventajas:

Simplicidad y flexibilidad en la arquitectura
Menor consumo de recursos
Control total sobre los componentes utilizados
Ideal para proyectos de tamaño medio

Inconvenientes:

Requiere configurar manualmente autenticación y admin
Menos funcionalidades incluidas por defecto

3.3. Solución 3: FastAPI

Descripción: Framework moderno de Python optimizado para crear APIs REST con tipado automático y documentación interactiva.
Ventajas:

Rendimiento superior
Documentación automática de API
Validación de datos integrada

Inconvenientes:

Orientado a APIs, no a aplicaciones web tradicionales
Requiere frontend separado (React, Vue)
Más complejo para renderizar HTML

4. SELECCIÓN DE LA SOLUCIÓN DEFINITIVA

4.1. Justificación de la elección

Se ha seleccionado Flask como framework principal por su equilibrio entre simplicidad y funcionalidad. Para un proyecto de tamaño medio como este portal de adopciones, Flask proporciona la flexibilidad necesaria sin la complejidad de Django ni las limitaciones de FastAPI para aplicaciones web tradicionales.

4.2. Descripción detallada

La solución implementa la siguiente arquitectura:

Flask 3.0 como framework web
PostgreSQL 15 como base de datos relacional
SQLAlchemy como ORM para abstracción de base de datos
Flask-Login para gestión de sesiones
Bootstrap 5 para interfaz responsive
Jinja2 para renderizado de plantillas

4.3. Análisis de la implementación

Recursos para desarrollo:

IDE: Visual Studio Code
Sistema Operativo: Ubuntu-24.04
Control de versiones: Git + GitHub
Entorno: Python 3.12 con entorno virtual

Recursos para despliegue:

Desarrollo local: Docker Compose (PostgreSQL + Adminer)
Producción: Railway (plataforma cloud con PostgreSQL incluido)

5. DISEÑO DE LA APLICACIÓN

5.1. Diseño de la Base de Datos

Diagrama de tablas:

USUARIOS	SOLICITUDES	MASCOTAS
id (PK)	id (PK)	id (PK)
email	usuario_id (FK)	nombre
password_hash	mascota_id (FK)	especie
nombre	fecha_solicitud	raza
apellidos	estado	edad_aprox
telefono	cuestionario_json	sexo
rol	comentarios_admin	tamaño
fecha_registro	revisado_por (FK)	descripción
		estado
		foto_url

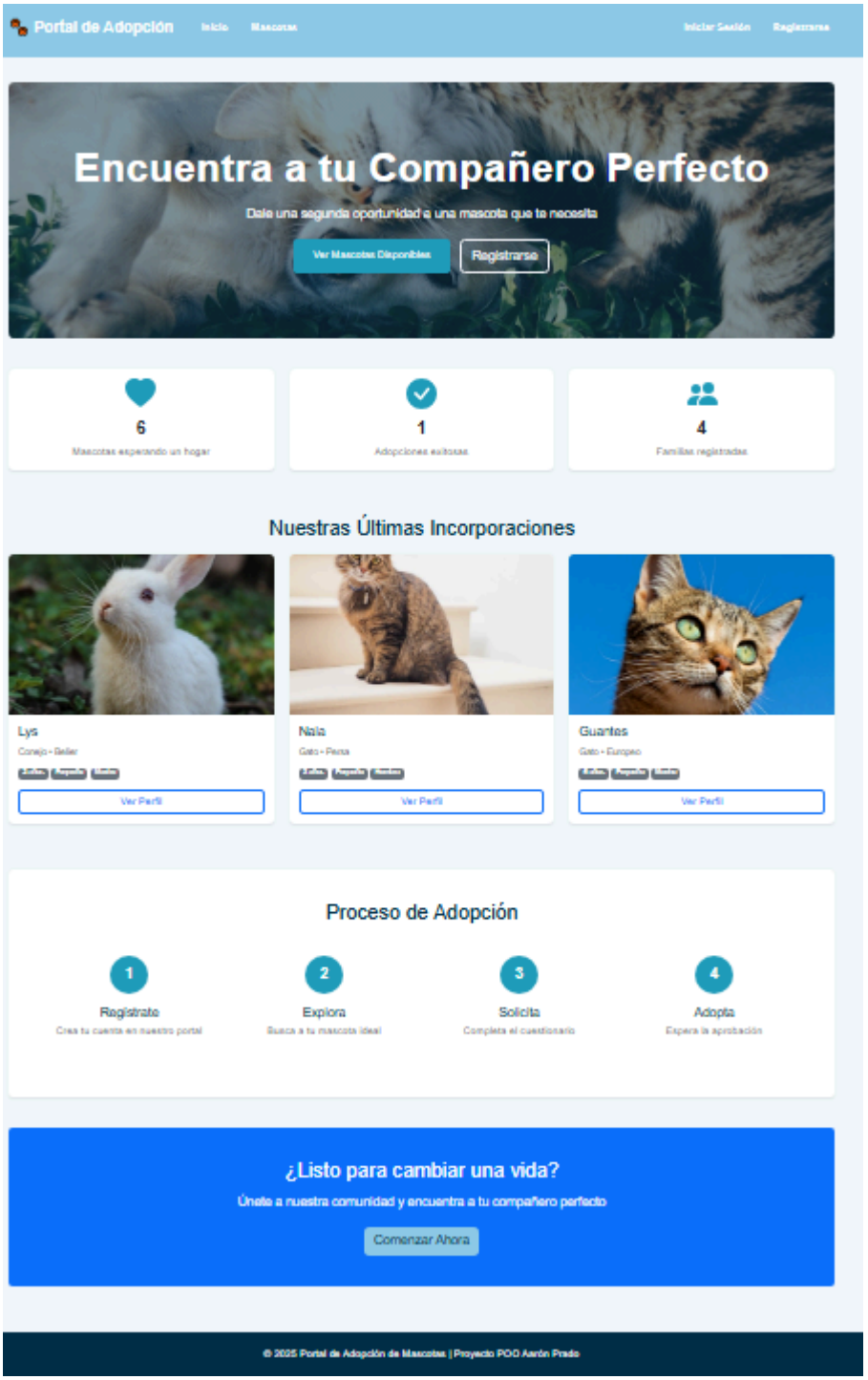
Sistema Gestor: PostgreSQL 15, elegido por su robustez en el manejo de relaciones y soporte para JSON en el campo cuestionario.
Scripts de inicialización:

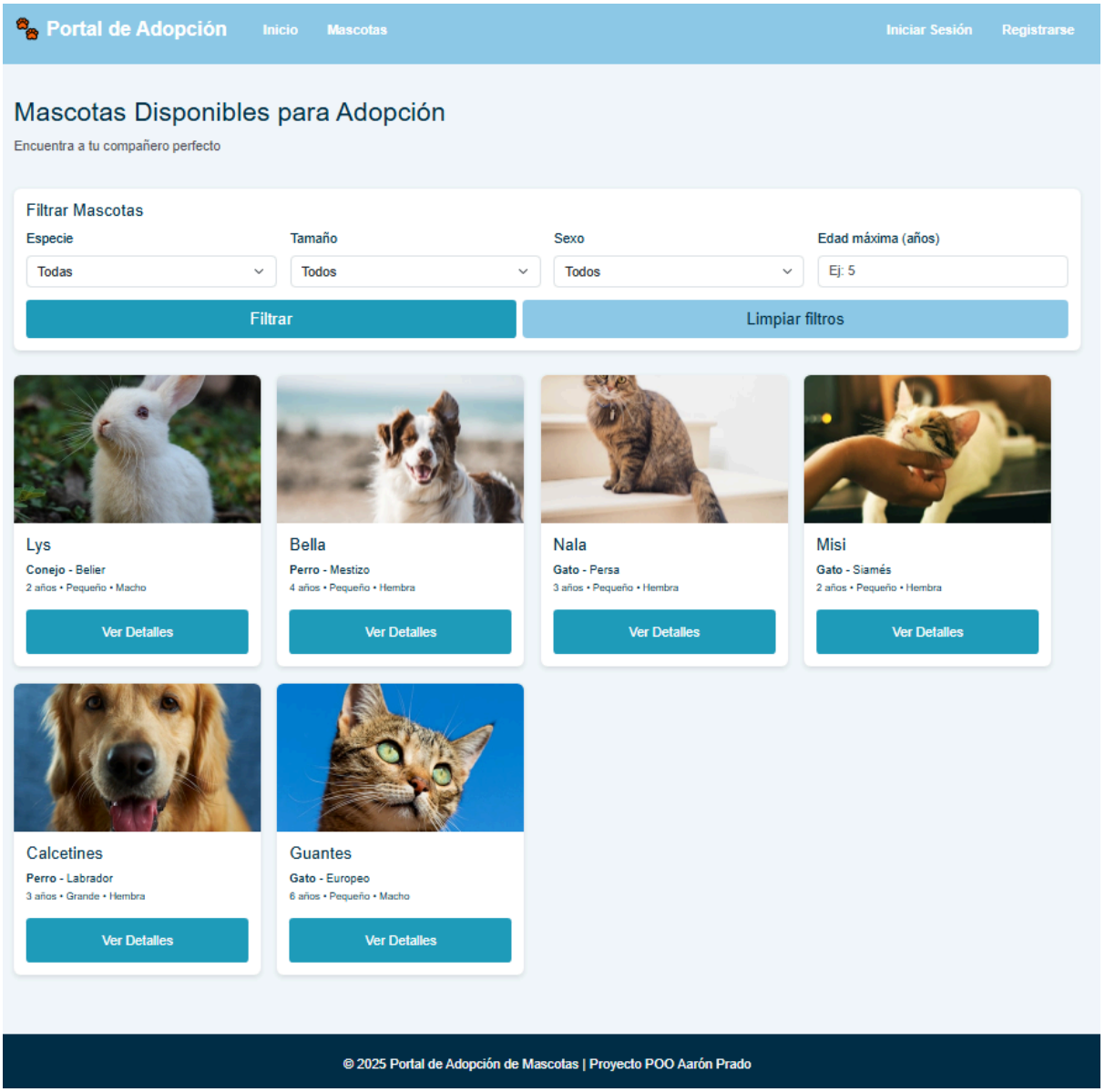
01_schema.sql: Creación de tablas con constraints y relaciones
02_seed.sql: Datos de prueba iniciales

5.2. Diseño de la interfaz

La interfaz utiliza Bootstrap 5 con una paleta de colores personalizada en tonos azules. Las principales vistas incluyen:

Página principal: Hero section con estadísticas y mascotas destacadas
Catálogo: Grid de cards con foto y datos básicos
Detalle de mascota: Información completa con botón de solicitud
Panel admin: Tablas con acciones CRUD y ordenamiento
Formulario de solicitud: Cuestionario estructurado en secciones


Portal de Adopción
Inicio
Mascotas
Iniciar Sesión
Registrarse
Encuentra a tu Compañero Perfecto
Dale una segunda oportunidad a una mascota que te necesita
Ver Mascotas Disponibles
Registrarse
6
Mascotas esperando un hogar
1
Adopciones exitosas
4
Familias registradas
Nuestras Últimas Incorporaciones
Lys
Conejo - Belier
Ver Perfil
Nala
Gato - Perso
Ver Perfil
Guantes
Gato - Europeo
Ver Perfil
Proceso de Adopción
1
Registrate
Crea tu cuenta en nuestro portal
2
Explora
Busca a tu mascota ideal
3
Solicita
Completa el cuestionario
4
Adopta
Espera la aprobación
¿Listo para cambiar una vida?
Únete a nuestra comunidad y encuentra a tu compañero perfecto
Comenzar Ahora
© 2025 Portal de Adopción de Mascotas | Proyecto POO Aarón Prado



5.3. Diseño del backend

Arquitectura modular:

```
app/
├── __init__.py      # Application Factory
├── models.py        # Clases Usuario, Mascota, Solicitud
├── routes/
│   ├── auth.py      # Autenticación
│   ├── mascotas.py  # CRUD mascotas
│   └── solicitudes.py # Gestión solicitudes
└── templates/       # Vistas Jinja2
```

Librerías principales:

Flask==3.0.0
Flask-SQLAlchemy==3.1.1
Flask-Login==0.6.3
psycopg2-binary==2.9.9
python-dotenv==1.0.0

6. DISEÑO DE LAS PRUEBAS UNITARIAS

6.1. Pruebas unitarias desarrolladas

Se ha diseñado un conjunto de pruebas para validar las funcionalidades críticas del sistema.

Pruebas implementadas:

Test de autenticación: registro, login, logout
Test de modelos: creación de usuarios, mascotas y solicitudes
Test de permisos: acceso a rutas protegidas
Test de validaciones: formularios y datos de entrada

6.2. Resultados de las pruebas unitarias

Las pruebas manuales realizadas durante el desarrollo confirmaron el correcto funcionamiento de:

Sistema de autenticación con hash de contraseñas
CRUD completo de mascotas con validaciones
Flujo de solicitudes con cambio automático de estados
Control de acceso basado en roles (admin/adoptante)

6.3. Análisis de errores encontrados y modificaciones

Durante el desarrollo se identificaron y corrigieron:

Problema con el navbar responsive en dispositivos móviles (solucionado con ajustes CSS)
Validación de solicitudes duplicadas (añadida constraint única en base de datos)
Estado de mascotas no se actualizaba automáticamente (implementado método en modelo)
Hashes incorrectos en datos de prueba del archivo seed.sql (regenerados con formato correcto de werkzeug)

7. REPOSITORIO Y CONTROL DE VERSIONES

7.1. Referencia y enlace al repositorio

Repositorio GitHub: https://github.com/AaronPrado/proyecto-poo-adopcion

7.2. Esquema de ramas y versiones

Se ha utilizado Trunk Based Development con rama main como principal y ramas para cada módulo:
main
├──docker (configuración inicial)
├──auth (sistema autenticación)
├──mascotas (CRUD mascotas)
├──solicitudes (gestión solicitudes)
├──frontpage (página principal)
├──testing (pruebas unitarias)
└──deploy (configuración despliegue)
Commits principales realizados:

Estructura inicial del proyecto
Configuración y modelos de base de datos
Sistema de autenticación con Flask-Login
Módulo completo de mascotas
Sistema de solicitudes con cuestionario
Interfaz responsive y página principal

8. DESPLIEGUE Y PRUEBAS FINALES

8.1. Descripción del entorno de despliegue

Desarrollo local:

Docker Compose para PostgreSQL y Adminer
Flask development server con hot reload
Python 3.12 con entorno virtual

Producción (Railway):

Despliegue automático desde GitHub
PostgreSQL managed incluido
Variables de entorno configuradas en plataforma
URL pública HTTPS

8.2. Scripts de configuración

```
Docker Compose para desarrollo:
yamlversion: '3.8'

services:
  postgres:
    image: postgres:15-alpine
    container_name: adopciones_db
    restart: unless-stopped
    environment:
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: password
      POSTGRES_DB: adopciones_db
      POSTGRES_INITDB_ARGS: "--encoding=UTF8 --locale=es_ES.UTF-8"
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
                                                                    -
./scripts_bd/01_schema.sql:/docker-entrypoint-initdb.d/01_schema.s
ql
                                                                    -
./scripts_bd/02_seed.sql:/docker-entrypoint-initdb.d/02_seed.sql

  adminer:
    image: adminer:latest
    container_name: adopciones_adminer
    restart: unless-stopped
    ports:
      - "8080:8080"
    environment:
      ADMINER_DEFAULT_SERVER: postgres
      ADMINER_DESIGN: nette
    depends_on:
      - postgres

volumes:
  postgres_data:
    driver: local
```

Configuración para Railway:

Procfile: web: gunicorn run:app
requirements.txt con todas las dependencias
Variables de entorno: DATABASE_URL, SECRET_KEY, FLASK_ENV, RAILPACK_PYTHON_VERSION
(puesto en python 3.12 por problemas de compatibilidad con 3.13 que tiene de base)

8.3. Ficheros de configuración

config.py: Configuración por entornos (desarrollo/producción)
.env: Variables locales de desarrollo
docker-compose.yml: Servicios locales con PostgreSQL y Adminer

9. CONCLUSIONES

9.1. Valoración del trabajo

El proyecto ha cumplido satisfactoriamente los objetivos planteados, implementando un sistema funcional que digitaliza el proceso de adopción. Se han aplicado los conceptos de POO, especialmente en el diseño de modelos.

9.2. Dificultades encontradas

Configuración inicial del entorno con Docker y PostgreSQL
Implementación del sistema de autenticación con roles
Diseño responsive para múltiples dispositivos
Gestión del estado de mascotas en función de las solicitudes
Despliegue en Railway y compatibilidad de versiones

9.3. Posibles ampliaciones o mejoras futuras

Sistema de notificaciones por email
Chat en tiempo real entre adoptantes y refugio
Galería de múltiples fotos por mascota
Integración con redes sociales para compartir mascotas
Sistema de donaciones

10. REFERENCIAS BIBLIOGRÁFICAS

Documentación oficial de Flask. Flask Documentation. https://flask.palletsprojects.com/
Documentación de SQLAlchemy. SQLAlchemy Documentation. https://docs.sqlalchemy.org/
Documentación de PostgreSQL. PostgreSQL 15 Documentation. https://www.postgresql.org/docs/15/
Bootstrap 5. Bootstrap Documentation. https://getbootstrap.com/docs/5.3/
Material del curso POO. Apuntes y ejercicios del módulo.