# **DOCUMENTACIÓN DE DISEÑO DEL SOFTWARE - PET SERVICES**

## **1. Introducción**

El diseño del software **Pet Services** establece la estructura y organización de los componentes que conforman el sistema, asegurando una arquitectura robusta, escalable y fácil de mantener. Este documento detalla las **interfaces gráficas**, el **modelo arquitectónico**, los **patrones de diseño** utilizados y el **modelo de datos**, con el propósito de ofrecer una visión clara del desarrollo y funcionamiento del software.

## **2. Interfaces Gráficas**

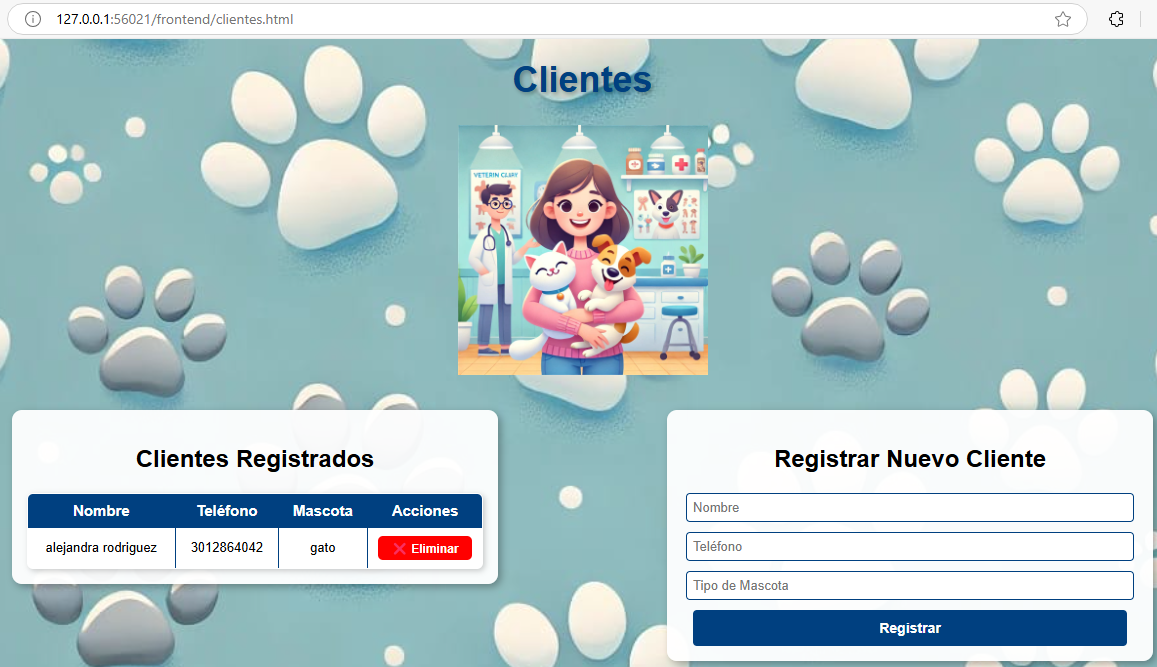
### **2.1 Diseño de las Pantallas**

Las interfaces gráficas fueron diseñadas para ser intuitivas y amigables con el usuario. Se estructuran de la siguiente manera:

* **Pantalla Principal:**
  + Título llamativo en color púrpura.
  + Frase de bienvenida en color fucsia y con fuente resaltada.
  + Imagen animada representativa de mascotas.
  + Botones de acceso rápido a los módulos principales.



* **Módulo de Clientes:**
  + Tabla de clientes registrados (Nombre, Teléfono, Mascota).
  + Formulario para registrar nuevos clientes.
  + Botón para eliminar clientes.
  + Imagen representativa centrada entre las tablas.



* **Módulo de Servicios:**
  + Tabla de servicios existentes con imágenes representativas.
  + Formulario para registrar un nuevo servicio.
  + Botón de eliminación de servicios.

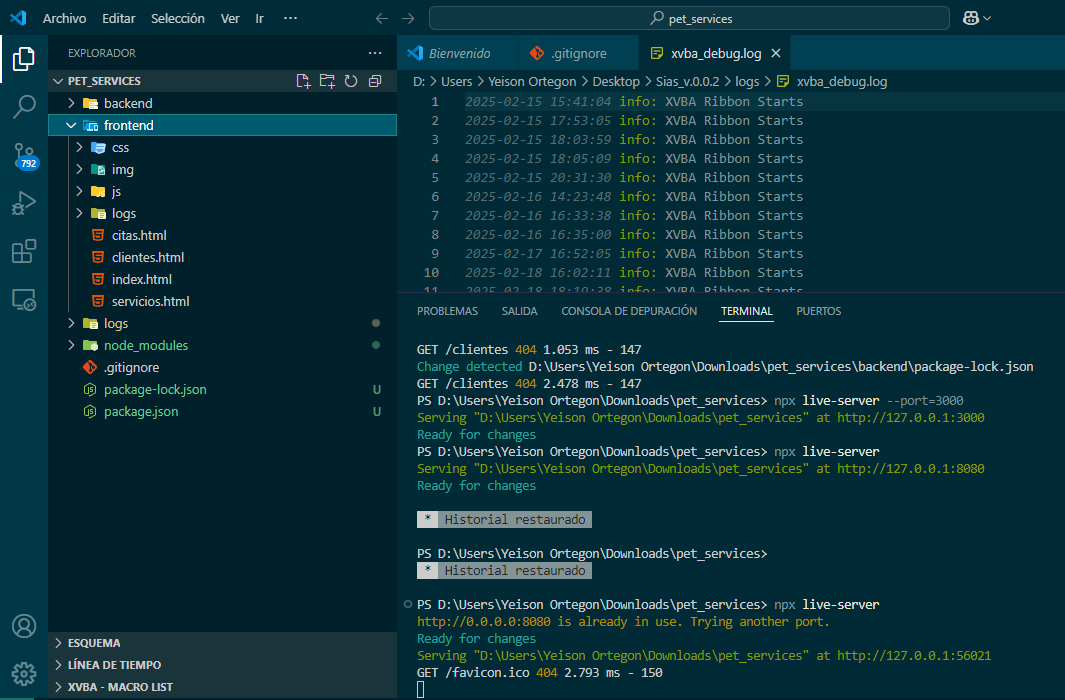


* **Módulo de Citas:**
  + Tabla de citas registradas con información detallada.
  + Formulario para agendar nuevas citas.
  + Botón de cancelación de citas.



### **2.2 Tecnologías Utilizadas en el Frontend**

* **HTML5, CSS3 y JavaScript:** Para la construcción y diseño responsivo de las interfaces.
* **Bootstrap:** Para mejorar la apariencia de los formularios y tablas.
* **Animaciones CSS:** Para hacer la interfaz más atractiva y dinámica.



## **3. Modelo Arquitectónico**

### **3.1 Arquitectura del Software**

El sistema sigue una **arquitectura Cliente-Servidor**, basada en **Node.js** para el backend y **MySQL** como gestor de base de datos. La estructura se divide en los siguientes módulos:

* **Frontend (Cliente):**
  + Interfaz de usuario en **HTML, CSS y JavaScript**.
  + Envío de peticiones HTTP a través de **Fetch API**.
* **Backend (Servidor):**
  + Desarrollado con **Node.js** y el framework **Express.js**.
  + Implementación de API REST para gestionar clientes, servicios y citas.
  + Validación y autenticación de datos.
* **Base de Datos:**
  + Servidor **MySQL** gestionado con **phpMyAdmin**.
  + Tablas organizadas con relaciones definidas entre clientes, servicios y citas.

### **3.2 Modelo de Comunicación**

El cliente envía solicitudes HTTP al servidor, el cual responde con datos en formato JSON. Se sigue el esquema **MVC (Modelo-Vista-Controlador),** separando la lógica del negocio de la presentación de datos.

## **4. Patrones de Diseño**

Para la implementación del software, se emplearon los siguientes **patrones de diseño:**

### **4.1 Patrón MVC (Modelo-Vista-Controlador)**

* **Modelo:** Gestiona la lógica de la base de datos y estructuras de datos.
* **Vista:** Representa la interfaz gráfica y el diseño de las páginas.
* **Controlador:** Procesa las solicitudes del usuario y coordina la interacción entre la Vista y el Modelo.

### **4.2 Patrón Singleton**

Se aplicó en la conexión con la base de datos para garantizar que solo exista una instancia activa durante la ejecución del software.

### **4.3 Patrón Factory**

Se utilizó para la creación de objetos en el backend, facilitando la gestión y reusabilidad del código.

## **5. Modelo de Datos**

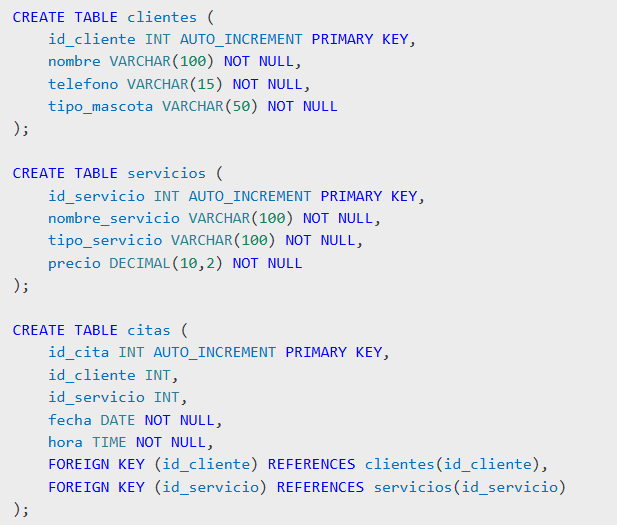
El **modelo entidad-relación (MER)** define la estructura de la base de datos con las siguientes entidades:

### **5.1 Estructura de las Tablas**

* **Clientes:** id\_cliente, nombre, telefono, tipo\_mascota.
* **Servicios:** id\_servicio, nombre\_servicio, tipo\_servicio, precio.
* **Citas:** id\_cita, id\_cliente, id\_servicio, fecha, hora.

### **5.2 Relaciones entre Tablas**

* Un **cliente** puede agendar múltiples **citas**.
* Un **servicio** puede ser agendado en varias **citas**.
* Cada **cita** está asociada a un **cliente** y un **servicio**.



## **6. Conclusiones**

El diseño del software **Pet Services** ha sido estructurado siguiendo buenas prácticas de desarrollo, utilizando patrones de diseño adecuados, una arquitectura cliente-servidor eficiente y una base de datos relacional bien organizada. Gracias al modularidad del sistema, es posible ampliar funcionalidades en el futuro, asegurando la escalabilidad y mantenibilidad del software.