

## Documentación Técnica del Proyecto

### 1. Tecnologías Utilizadas

El desarrollo del proyecto se realizará utilizando un conjunto de tecnologías consolidadas que garantizan compatibilidad, eficiencia y facilidad de mantenimiento:

#### 1.1 Frontend

- **HTML5**: Lenguaje de marcado estándar para estructurar contenido en la web.
- **Bootstrap**: Framework CSS que facilita el diseño responsive y componentes visuales reutilizables.
- **jQuery**: Librería JavaScript que simplifica la manipulación del DOM, eventos, peticiones AJAX y compatibilidad entre navegadores.

#### 1.2 Backend

- **PHP**: Lenguaje de programación del lado del servidor. Utilizado para la lógica del negocio, control de flujo de datos, acceso a base de datos y generación de vistas dinámicas.

#### 1.3 Base de Datos

- **MySQL**: Sistema de gestión de bases de datos relacional. Escalable, confiable y ampliamente utilizado en conjunto con PHP.
- La conexión se realizará mediante el archivo `conexion.php` ubicado en la carpeta `/database`.

1. Se crea la base de datos "PetBook" con codificación UTF8MB4 para soportar caracteres especiales.

2. Se definen las tablas principales:

- Roles: Tipos de usuario (administrador, usuario, etc.).
- Usuarios: Información personal y credenciales de acceso.
- Especies: Tipos de animales (perro, gato, etc.).
- Razas: Razas asociadas a cada especie.
- Mascotas: Mascotas registradas por los usuarios.
- Publicaciones: Avisos sobre mascotas (perdidas, encontradas, adoptadas).
- Comentarios: (comentado) Para interacción entre usuarios en publicaciones.

3. Se incluye un sistema de localización jerárquico:

- Paises, Provincias, Partidos, Localidades, Calles.

4. Todas las tablas incluyen claves primarias y foráneas para mantener la integridad referencial.

5. Se establecen restricciones de unicidad y relaciones entre entidades.
6. El diseño permite registrar usuarios, asociarles mascotas, publicar avisos y gestionar la ubicación detallada de eventos o domicilios.

Este esquema es la base para una red social orientada a la comunidad de mascotas, facilitando la búsqueda, adopción y recuperación de animales.

---

## 2. Convenciones de Nomenclatura

El objetivo de las convenciones es mantener un código limpio, legible y coherente entre todos los integrantes del equipo. Se utilizarán las siguientes pautas:

### 2.1 Formato

- **Clases (PHP):** PascalCase → UsuarioController.php

```
class Usuario
{
    public function __construct()
    {
        // Logica
    }

    class ConsultaUsuario
    {
        public function __construct()
        {
            // Logica
        }
    }
}
```

- **Variables (PHP / JS):** camelCase → \$nombreUsuario, let formularioValido

```
<?php
$nombreUsuario;
$fechaNacimiento;
$esActivo;
?>

<script>
let nombreUsuario = $('#inputNombre').val();
const fechaActual = new Date();
</script>
```

- **Funciones y Métodos:** camelCase → validarFormulario()

```

public function guardarNombre(string $nuevoNombre): void
{
    $this->nombre = $nuevoNombre;
}

public function obtenerDatos(): array
{
    return [
        'nombre' => $this->nombre,
        'email' => $this->email
    ];
}

```

- **Constantes:** SCREAMING\_SNAKE\_CASE → MAX\_INTENTOS

```

<?php

define('TIEMPO_EXPIRACION_SESION', 3600);

const ESTADO_ACTIVO = 'activo';

?>

```

- **Tablas en la Base de Datos:** PascalCase, singular → Usuario
- **Campos de Tablas:** snake\_case → fecha\_registro

## 2.2 Archivos

- **PHP:** NombreClase.php, funcionesValidacion.php
- **JavaScript:** validacionFormulario.js, cargaDatos.js
- **HTML:** formulario-registro.html, vistaPublicaciones.html

## 2.3 Reglas Generales

- **Representatividad semántica:** Los nombres deben reflejar claramente su contenido o propósito.

```

<?php

//Ejemplo correcto
function eliminarPublicacionUsuario($publicacionId) { ... }

//Ejemplo incorrecto
function eliminarPerdido($id) { ... } // ¿Qué se elimina? ¿De dónde?

?>

```

- **Pluralidad semántica**

Usar nombres en plural si manejan múltiples entidades: obtenerUsuarios()  
En singular, si representan uno solo: obtenerUsuario(\$usuarioid)

- **Uno:**

```
php
```

```
function obtenerUsuario($usuarioId): Usuario { ... }
```

- **Muchos:**

```
php
```

```
function obtenerUsuarios(): array { ... }
```

- **Evitar redundancia:** No repetir innecesariamente el nombre de la entidad dentro de sus métodos.

```
<?php

//Incorrecto
class ControladorUsuario {
|   public function listarUsuarios() { ... }
}

//Correcto
class ControladorUsuario {
|   public function listar() { ... }
}

//también correcto si el contexto lo exige

class ControladorGrupo {
|   public function listarUsuarios() { ... }
}

?>
```

- **Encapsulamiento de lógica compleja:**

Extraer fragmentos complejos en funciones auxiliares con nombres claros y específicos.

```

<?php

// Sin encapsulamiento
class Factura
{
    public function calcularTotal(float $subtotal): float
    {
        $impuesto = $subtotal * 0.21;
        $descuento = $subtotal > 1000 ? $subtotal * 0.1 : 0;
        return $subtotal + $impuesto - $descuento;
    }
}

// Con encapsulamiento de lógica
class Factura
{
    public function calcularTotal(float $subtotal): float
    {
        return $subtotal + $this->calcularImpuesto($subtotal) - $this->calcularDescuento($subtotal);
    }

    private function calcularImpuesto(float $monto): float
    {
        return $monto * 0.21;
    }

    private function calcularDescuento(float $monto): float
    {
        return $monto > 1000 ? $monto * 0.1 : 0;
    }
}

?>

```

- **Alineamiento visual:**

Alinear las asignaciones para facilitar el escaneo.

```

<?php

// Incorrecto
$nombre = 'Ana';
$apellido = 'Pérez';
$correo = 'ana@mail.com';

// Correcto

$nombre      = 'Ana';
$apellido   = 'Pérez';
$correo     = 'ana@mail.com';

?>
<html>
    <input type="text" id="nombre"    name="nombre">
    <input type="text" id="apellido"  name="apellido">
</html>

```

```
// ✗ Desordenado
$datos = [
    "nombre" => "Ana",
    "apellidoLargo" => "Fernández",
    "edad" => 30
];

// ✓ Alineado
$datos = [
    "nombre"      => "Ana",
    "apellidoLargo" => "Fernández",
    "edad"        => 30
];
```

- **Evitar abreviaturas:** Usar nombres completos y descriptivos para mejorar la comprensión del código.

```
<?php

// Incorrecto
$usrNom;
$pubExt;
$fnac;

// Correcto
$nombreUsuario;
$publicacionExtraviada;
$fechaNacimiento;

?>
```

- **Comentarios útiles:**

Comentar código de manera eficiente implica escribir anotaciones que realmente aporten valor y claridad al lector del código.

No comentar obviedades.

Comentar algoritmos, excepciones y decisiones no evidentes.

Usar estilo de documentación si es posible.

```
// Función con comentario útil: suma dos números y devuelve el resultado.
/**
 * Suma dos números.
 *
 * @param int $a Primer número.
 * @param int $b Segundo número.
 * @return int Resultado de la suma.
 */
function sumar($a, $b) {
    return $a + $b;
}

// Comentario menos útil: solo repite el nombre de la función.
/**
 * Esta función resta dos números.
 */
function restar($a, $b) {
    return $a - $b;
}

// Comentario útil: explica el propósito y el tipo de retorno.
/**
 * Verifica si un usuario está activo.
 *
 * @param bool $activo Estado del usuario.
 * @return string Mensaje indicando el estado.
 */
function estadoUsuario($activo) {
    return $activo ? "Usuario activo" : "Usuario inactivo";
}

// Comentario menos útil: no aporta información relevante.
/**
 * Hace algo con el usuario.
 */
function procesarUsuario($usuario) {
    // Procesamiento ficticio
    return strtoupper($usuario);
}
```

---

### 3. Estructura del Proyecto

La estructura de carpetas sigue un esquema claro de separación de responsabilidades:

```

/PetBook
├── /app
│   ├── /Controllers      # Lógica del servidor (PHP)
│   ├── /Models           # Clases que representan entidades y acceden a la BD
│   ├── /Views            # Vistas HTML principales
│   └── /Helpers          # Funciones utilitarias y validaciones

├── /public
│   ├── /css              # Hojas de estilo
│   ├── /js               # Scripts JS/jQuery
│   ├── /img              # Imágenes estáticas
│   └── index.php         # Punto de entrada

├── /database
│   ├── /migrations       # Scripts para crear/modificar la BD
│   └── conexion.php      # Conexión a base de datos

├── /routes              # Archivo o carpeta de enrutamiento (si aplica)

├── /tests               # Pruebas manuales o automáticas

└── README.md            # Guía del proyecto
└── .gitignore           # Ignorar archivos temporales/cache

```

### 3.1 Separación de Capas

- **Modelo:** Entidades y lógica de acceso a datos (UsuarioModel.php)
- **Vista:** Archivos HTML que se muestran al usuario (registro.html)
- **Controlador:** Orquesta el flujo entre modelo y vista (UsuarioController.php)

### 3.2 Organización de Archivos

- Scripts JS divididos por funcionalidad: ajaxUsuarios.js, validacionesFormulario.js
- Hojas de estilo separadas: main.css (global), formulario.css (específica)
- Evitar mezclar lógica PHP con HTML salvo en includes (header.php, footer.php)

### 3.3 Rutas y Configuración

- Definición clara de URLs y controladores correspondientes.
- Configuración separada en archivos como config.php o .env.
- Nunca almacenar contraseñas o claves sensibles en el código fuente.

## 4. Estilo de Código

### 4.1 Indentación y Tabulación

- Se usarán **4 espacios** por nivel.
- No se mezclarán espacios y tabulaciones.

```
//Correcto
if ($usuario->esActivo()) {
    enviarNotificacion($usuario);
}

//Incorrecto
if ($usuario->esActivo())
    enviarNotificacion($usuario);
```

#### 4.2 Longitud de Línea

- Máximo recomendado: **100 caracteres** por línea.

#### 4.3 Llaves

- Siempre deben usarse llaves {} incluso si el bloque tiene una sola línea.

```
//Correcto
if ($esAdmin) {
    mostrarPanelAdmin();
}

// Incorrecto
if ($esAdmin) mostrarPanelAdmin(); // ¡peligroso!
```

#### 4.4 Espaciado y Saltos de Línea

- Línea en blanco entre bloques lógicos.
- Espacios entre operadores, argumentos, etc.

```
$precioConDescuento = $producto->precio - $producto->descuento;

if ($precioConDescuento < 100) {
    mostrarOfertaEspecial();
}
```

#### 4.5 Encabezado de Archivos (opcional)

php

CopiarEditar

```
/*
 * Autor: Juan Manuel Yarza
 * Fecha: Julio 2025
 * Descripción: Controlador de usuarios
 */
```

#### 4.6 Código Muerto

- Se eliminarán bloques comentados que no se vayan a reutilizar.

#### **4.7 Idioma Estándar**

- Todo el código estará en **español neutro**, incluidos nombres de variables, funciones y comentarios.
- 

### **5. Herramientas de Desarrollo**

#### **5.1 Control de Versiones**

- Se utilizará **Git** para el control de versiones.
- El repositorio se alojará en **GitHub** para colaboración, respaldo y revisión.

#### **5.2 Editor de Código**

- Se utilizará **Visual Studio Code (VS Code)** para desarrollar tanto el frontend como el backend y crear la base de datos.

#### **5.3 Servidor Local**

- Se utilizará **XAMPP** como servidor web local para correr Apache, gestionar MySQL y simular el entorno de producción.