****

**UNIVERSIDAD PRIVADA DE TACNA**

**FACULTAD DE INGENIERÍA**

**Escuela Profesional de Ingeniería de Sistemas**

**Proyecto *PetGuard - Gestión y Cuidado Inteligente de Mascotas***

Curso: Programación III

Docente: Ing. Juan Manuel Choque Flores

Integrantes:

**Rosas Chambilla, Jefferson (2021072618)**

**Challo Coaquera Alexander wilson (2017057578)**

**Rodriguez Cardenas, Patrick Elvis (2022075751)**

**Tacna – Perú**

**2025 - I**

| CONTROL DE VERSIONES | | | | | |
| --- | --- | --- | --- | --- | --- |
| Versión | Hecha por | Revisada por | Aprobada por | Fecha | Motivo |
| 1.0 | SNMY | JRC/ACC/PRC | ERRM | 27/05/2025 | Versión Original |

Sistema *PetGuard - Gestión y Cuidado Inteligente de Mascotas*

Documento de Arquitectura de Software

Versión *1.0*

| CONTROL DE VERSIONES | | | | | |
| --- | --- | --- | --- | --- | --- |
| Versión | Hecha por | Revisada por | Aprobada por | Fecha | Motivo |
| 1.0 | MPV | ELV | ARV | 10/10/2020 | Versión Original |

ÍNDICE GENERAL

[**1. Introducción 6**](#_heading=h.o8749zdyp88y)

[1.1. Propósito 6](#_heading=h.od9flueqnhtg)

[1.2. Alcance 6](#_heading=h.uduflyz8hmzm)

[1.3. Definición, siglas y abreviaturas 8](#_heading=h.w3749z82b2tr)

[1.4. Referencias 8](#_heading=h.hgypoqoewo4r)

[1.5. Visión General 8](#_heading=h.p5njoji3gabw)

[1.5.1. Objetivo de esta Visión General 8](#_heading=h.s839kwj2gci5)

[**2. Representación Arquitectónica 9**](#_heading=h.ekfjkd5e8ao8)

[2.1. Escenarios 9](#_heading=h.ttbvczb9l7w1)

[2.1.1. Registro de Mascota por un Usuario Ciudadano 9](#_heading=h.ivw011aetr6p)

[2.1.2. Fallo en la conexión a la base de datos 9](#_heading=h.2io5ism7121y)

[2.1.3. Escalabilidad por aumento de usuarios 9](#_heading=h.y13zh2n6eh3e)

[2.2. Vista Lógica 10](#_heading=h.1xtpfd6fbs4t)

[2.3. Vista del Proceso 12](#_heading=h.gne7b6xhmcpl)

[2.3.1. Descripción General 12](#_heading=h.i6sjj8tbwuya)

[2.3.2. Proceso: Registro de Mascota 12](#_heading=h.skkzh5u33vuc)

[2.3.3. Proceso: Autenticación de Usuario 12](#_heading=h.2ip19misuyfg)

[2.3.4. Proceso: Generación de Reportes 13](#_heading=h.j31u2pddwveu)

[2.4. Vista del desarrollo 13](#_heading=h.co9szjwftm9l)

[2.5. Vista Física 14](#_heading=h.qwduydea0rfq)

[2.5.1. Despliegue Básico en Entorno Local 14](#_heading=h.et0zh79qwfq)

[2.5.2. Componentes físicos 14](#_heading=h.8voelesguby)

[2.5.3. Consideraciones Físicas para el Despliegue en Producción 14](#_heading=h.k9iqyhcxetw1)

[**3. Objetivos y Limitaciones Arquitectónicas 15**](#_heading=h.is2ccw5q4k0)

[3.1. Disponibilidad 15](#_heading=h.3u5jer5fzdko)

[3.2. Seguridad 15](#_heading=h.7erxjxbejeu7)

[3.3. Adaptabilidad 16](#_heading=h.xnbbg8hrnjw0)

[3.4. Rendimiento 16](#_heading=h.9xdau06594u3)

[**4. Análisis de Requerimientos 16**](#_heading=h.6o9b9odihxs5)

[4.1. Requerimientos Funcionales 16](#_heading=h.l548hjtr1cxq)

[4.2. Requerimientos No Funcionales 17](#_heading=h.lh1683giuohb)

[4.2.1. Rendimiento: 17](#_heading=h.k6mh4noa13q7)

[4.2.2. Seguridad: 17](#_heading=h.5g86kss1l7sr)

[4.2.4. Escalabilidad: 18](#_heading=h.ctjau9cce6o2)

[4.2.5. Mantenibilidad: 18](#_heading=h.u5muwj2f225f)

[4.2.6. Compatibilidad: 18](#_heading=h.uz322b6ko8y)

[4.2.7. Disponibilidad: 18](#_heading=h.lh3i8d1owakh)

[**5. Vistas de Casos de Uso 18**](#_heading=h.hzhs3gkbdmjs)

[**6. Vista Lógica 20**](#_heading=h.1s8oaml1ewrp)

[6.1. Diagrama Contextual 21](#_heading=h.80ait4xwn2vj)

[**7. Vista de Procesos 23**](#_heading=h.workvo3xjovo)

[7.1. Diagrama de Proceso Actual 23](#_heading=h.kfipfttgk5h5)

[7.2. Diagrama de Proceso Propuesto 23](#_heading=h.olfnq3ajt1ji)

[**8. Vista de Despliegue 24**](#_heading=h.jhpz6evtk56j)

[8.1. Diagrama de Contenedor 24](#_heading=h.go3xdocise96)

[**9. Vista de Implementación 25**](#_heading=h.tmm00ha8cmrb)

[9.1. Diagrama de Componentes 25](#_heading=h.fd8bw2p3cshd)

[**10. Vista de Datos 26**](#_heading=h.jglgsdiuua6k)

[10.1. Diagrama Entidad Relación 26](#_heading=h.6w491bvhfb6j)

[**11. Atributos de Calidad 28**](#_heading=h.mtkviafu0s9r)

[11.1. Escenario de Seguridad 28](#_heading=h.30pf3nqqbqt1)

[11.2. Escenario de Usabilidad 29](#_heading=h.f8vz4lk4tnxs)

[11.3. Escenario de Adaptabilidad 29](#_heading=h.mcnbzkar5xyp)

[11.4. Escenario de Disponibilidad 30](#_heading=h.x3j8rz11w7ag)

[11.5. Otro Escenario 30](#_heading=h.61j20fqk5e5b)

# **Introducción**

## Propósito

El propósito del presente Documento de Arquitectura de Software (SAD) es describir de manera estructurada y detallada la arquitectura del sistema web de registro de información de mascotas, proporcionando una visión técnica integral que sirva como guía para el desarrollo, mantenimiento y evolución del sistema.

Este documento tiene como objetivos principales:

* Establecer los componentes principales del sistema, sus responsabilidades y cómo interactúan entre sí.
* Definir las decisiones arquitectónicas clave, incluyendo patrones, estilos y tecnologías utilizadas (Java, JSP, Servlets, HeidiSQL, etc.).
* Servir como referencia técnica para el equipo de desarrollo, pruebas, despliegue y mantenimiento del sistema.
* Facilitar la comprensión del diseño del sistema a partes interesadas no técnicas, como clientes, usuarios y directivos.
* Garantizar que la arquitectura propuesta cumpla con los requisitos funcionales y no funcionales del sistema, tales como rendimiento, seguridad, escalabilidad y usabilidad.

El SAD busca asegurar que el desarrollo del sistema se realice de forma ordenada, coherente y alineada con los objetivos del proyecto, minimizando riesgos técnicos y promoviendo la reutilización, mantenibilidad y calidad del software.

## Alcance

El presente Documento de Arquitectura de Software (SAD) abarca la definición, diseño y justificación de la arquitectura del sistema web de registro de información de mascotas, desarrollado en Java mediante JSP, Servlets y controlado por una base de datos gestionada con HeidiSQL (MySQL/MariaDB).

Este documento establece los lineamientos técnicos y estructurales necesarios para asegurar que el sistema sea funcional, escalable, seguro y mantenible a lo largo del tiempo. El SAD está orientado a satisfacer tanto los requisitos funcionales (como el registro de mascotas, historial médico, control de vacunas, etc.) como los requisitos no funcionales (seguridad, usabilidad, rendimiento, etc.) del proyecto.

El documento incluye:

* Descripción general del sistema y su propósito.
* Identificación de los módulos principales (autenticación, gestión de mascotas, historial médico, usuarios, etc.).
* Vista lógica de la arquitectura, que detalla la estructura modular y la interacción entre componentes (MVC).
* Vista de implementación, que describe cómo se organizan los archivos en el proyecto NetBeans.
* Vista de despliegue, con información sobre la ejecución del sistema en servidores (Apache Tomcat) y acceso desde navegadores.
* Decisiones arquitectónicas clave: uso de patrones, frameworks, herramientas y tecnologías adoptadas.
* Análisis de riesgos técnicos y estrategias para mitigarlos.
* Requisitos de hardware y software necesarios para el correcto funcionamiento del sistema.
* Criterios de calidad aplicados: seguridad, rendimiento, escalabilidad, mantenibilidad y disponibilidad.

## Definición, siglas y abreviaturas

* MVC: Es un patrón de software que separa la lógica de una aplicación entre 3 componentes interconectados: Modelo, Vista y Controlador.
* PetGuard: Nombre del Sistema Web que planteamos en este documento.
* UML: Es un lenguaje de modelado visual estándar que se utiliza para representar los componentes del sistema de forma gráfica.

## Referencias

* Documento de Factibilidad
* Documento de Especificación de Requerimientos Funcionales - SRS
* Documento de Arquitectura de Software - SAD

## Visión General

Este documento de Arquitectura de Software (SAD) proporciona una descripción estructurada de la arquitectura del Sistema Web PetGuard, cuyo objetivo es facilitar la gestión centralizada de datos relacionados con mascotas domésticas, sus propietarios y su historial médico, a través de una plataforma accesible, segura y eficiente.

La arquitectura descrita en este documento se basa en tecnologías Java, utilizando JSP y Servlets para el desarrollo del lado del servidor, y HeidiSQL (MySQL/MariaDB) como sistema de gestión de base de datos. El proyecto se desarrolla en Apache NetBeans 23, y se ejecuta en un servidor Apache Tomcat.

### Objetivo de esta Visión General

* Establecer una comprensión compartida entre desarrolladores, diseñadores, partes interesadas y personal técnico sobre cómo está estructurado el sistema y por qué se tomaron ciertas decisiones arquitectónicas.
* Proporcionar una guía técnica sólida para el desarrollo, implementación y mantenimiento del sistema.
* Alinear la solución técnica con los objetivos funcionales y no funcionales del proyecto.

# **Representación Arquitectónica**

## Escenarios

### Registro de Mascota por un Usuario Ciudadano

Estímulo: Un usuario, dueño de la mascota, accede al sistema y completa el formulario de registro de mascota.

Fuente: Usuario autenticado.

Entorno: Normal operación del sistema desde un navegador web.

Artefacto afectado: Módulo de Registro de Mascotas, Sistema Web, base de datos.

Respuesta: El sistema valida los datos ingresados y registra la mascota en la base de datos en menos de 2 segundos.

### Fallo en la conexión a la base de datos

Estímulo: El sistema pierde conexión con la base de datos MySQL.

Fuente: Problema en el servidor de base de datos o red.

Entorno: Producción.

Artefacto afectado: Sistema Web, backend, servidor de base de datos.

Respuesta: Se lanza un mensaje de error amigable al usuario y se registra el evento para análisis posterior.

### Escalabilidad por aumento de usuarios

Estímulo: El sistema pasa de 100 a 1,000 usuarios concurrentes durante una campaña de adopción masiva.

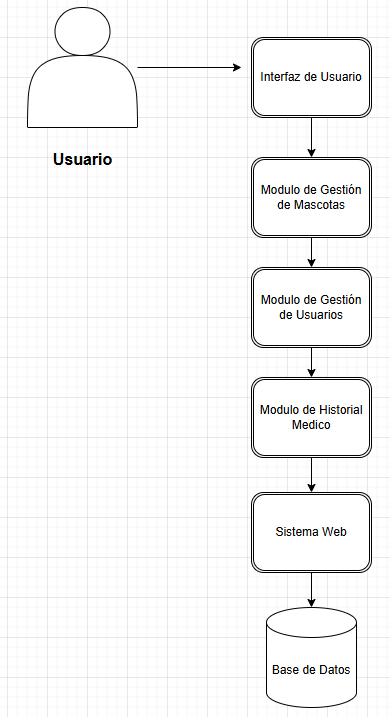
Fuente: Plataforma de adopciones y ciudadanía activa.

Entorno: Evento programado de alta demanda.

Artefacto afectado: Backend, frontend, base de datos, infraestructura.

Respuesta: El sistema balancea carga y mantiene el rendimiento con uso de caché y escalado horizontal.

## Vista Lógica

****

## Vista del Proceso

### Descripción General

El sistema PetGuard se basa en procesos web asincrónicos donde las peticiones HTTP del usuario son gestionadas por controladores (servlets), quienes se comunican con la capa de datos y retornan respuestas a las vistas (JSP). A continuación, se describen los procesos principales:

### Proceso: Registro de Mascota

Actor: Usuario  
 Descripción: Flujo que permite al usuario registrar una nueva mascota en el sistema.

Flujo del proceso:

1. El usuario accede al formulario registrarMascota.jsp.
2. El usuario completa los campos y adjunta una imagen.
3. El formulario se envía al ControladorMascota con acción Registrar.
4. El controlador valida los datos y crea un objeto Mascota.
5. El objeto se envía al MascotaDAO, el cual inserta los datos en la base de datos.
6. El sistema responde con un mensaje de éxito y redirige a listarMascotas.jsp.

### Proceso: Autenticación de Usuario

Actor: Usuario o Administrador  
 Descripción: Flujo de acceso seguro al sistema.

Flujo del proceso:

1. El usuario accede a login.jsp e ingresa su DNI y contraseña.
2. Se envía la solicitud al ControladorUsuario con la acción Login.
3. Se consulta la base de datos mediante UsuarioDAO.
4. Si las credenciales son correctas, se redirige al panel correspondiente (dashboardUsuario.jsp o dashboardAdmin.jsp).
5. Si las credenciales son inválidas, se muestra un mensaje de error.

### Proceso: Generación de Reportes

Actor: Administrador  
 Descripción: Flujo que permite obtener reportes estadísticos de las mascotas registradas.

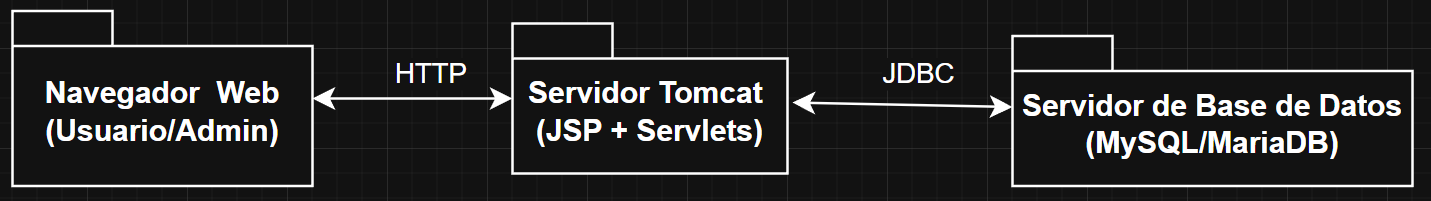
Flujo del proceso:

1. El administrador accede a reporteMascotas.jsp.
2. Selecciona filtros (usuario, tipo de animal, estado).
3. Se envía la solicitud al ControladorAdmin con acción GenerarReporte.
4. Se consulta la base de datos y se agrupan los datos.
5. Se genera un archivo visual (tabla o PDF) con los resultados.

## Vista del desarrollo

## Vista Física

### Despliegue Básico en Entorno Local

****

### Componentes físicos

* Cliente (Frontend):  
  + Navegador web moderno (Chrome, Firefox, Edge).
  + Acceso mediante URL del sistema (localhost:8080/PetGuard o IP externa en despliegue productivo).
* Servidor de Aplicaciones:  
  + Apache Tomcat 10+
  + Contiene los archivos .jsp, servlets .class, y configuraciones del proyecto.
* Servidor de Base de Datos:  
  + MySQL o MariaDB
  + Gestionado localmente con HeidiSQL o remotamente en servidores compatibles.

### Consideraciones Físicas para el Despliegue en Producción

* Escalabilidad horizontal: Puede migrarse a contenedores (Docker), usar balanceadores de carga o plataformas en la nube (ej. Oracle Cloud, Render).
* Seguridad: Implementación de HTTPS mediante certificados SSL.
* Backups automáticos: Se recomienda automatizar respaldos de la base de datos en horarios no operativos.
* Red interna o acceso externo: Posibilidad de habilitar acceso desde redes públicas a través de configuración de firewall o túneles seguros.

# **Objetivos y Limitaciones Arquitectónicas**

## Disponibilidad

**Objetivo:** Garantizar que el sistema PetGuard esté disponible para los usuarios finales de forma continua, especialmente en horarios operativos. Esto implica:

* Soporte para ejecución en entornos locales (intranet).
* Posibilidad de migración a un entorno en la nube pública o privada.
* Mecanismos de respaldo y recuperación ante fallos.

**Limitación:** Actualmente no se contempla infraestructura de alta disponibilidad. El sistema depende de un único servidor local (Tomcat + MySQL), por lo tanto, una caída del servidor implica inactividad total del sistema.

## Seguridad

**Objetivo:** Proteger la integridad, confidencialidad y disponibilidad de los datos almacenados en el sistema. Las medidas incluyen:

* Cifrado de contraseñas.
* Validaciones de entrada para evitar inyecciones SQL.
* Gestión de usuarios con roles diferenciados (usuario, administrador).
* Control de sesiones y cierre automático por inactividad.

**Limitación:** No se integra aún un protocolo HTTPS para comunicaciones seguras. La gestión de roles es básica y no se ha implementado una capa de auditoría o control de logs de seguridad.

## Adaptabilidad

**Objetivo:** Diseñar la arquitectura del sistema para que sea fácilmente adaptable a nuevas funcionalidades o entornos. Ejemplos:

* Arquitectura basada en MVC que permite sustituir componentes con bajo impacto.
* Modularidad para añadir nuevos módulos como notificaciones, seguimiento por GPS, o app móvil.

**Limitación:** La interfaz aún no está completamente adaptada a móviles (sin diseño responsive). Tampoco se han implementado mecanismos automáticos de internacionalización (i18n).

## Rendimiento

**Objetivo:** Garantizar un rendimiento aceptable bajo carga normal de trabajo. Específicamente:

* Tiempos de respuesta menores a 2 segundos por operación.
* Uso eficiente de consultas SQL y recursos del servidor.

**Limitación:** El rendimiento puede degradarse con más de 100 usuarios concurrentes debido a la falta de balanceo de carga, optimización de consultas o caché en capas intermedias.

# **Análisis de Requerimientos**

## Requerimientos Funcionales

Los requerimientos funcionales definen lo que el sistema PetGuard debe hacer desde el punto de vista del usuario final. A continuación, se detallan los principales:

1. Registro de usuarios: El sistema permitirá registrar nuevos usuarios ciudadanos y administrativos.
2. Autenticación de usuarios: El sistema debe validar las credenciales ingresadas (DNI y clave) y redirigir según el tipo de usuario.
3. Gestión de mascotas: Los usuarios podrán registrar, modificar, eliminar o consultar la información de sus mascotas.
4. Historial médico: El sistema permitirá ingresar y consultar registros médicos asociados a cada mascota (vacunas, desparasitaciones, controles).
5. Control de vacunas: Alertas de vacunas pendientes por fecha programada.
6. Gestión de usuarios administrativos: El administrador podrá gestionar el listado de usuarios (habilitar, deshabilitar).
7. Reportes: El sistema podrá generar reportes básicos sobre la cantidad de mascotas registradas por usuario y por tipo.
8. Carga de imágenes: Se permitirá subir una imagen por mascota.
9. Asignación de código QR o código de identificación: Cada mascota tendrá un código único para facilitar su identificación.

## Requerimientos No Funcionales

Los requerimientos no funcionales son condiciones de calidad o restricciones técnicas que debe cumplir el sistema:

### Rendimiento:

* + El tiempo de respuesta para consultas o registros no debe superar los 2 segundos bajo condiciones normales.

### Seguridad:

* + Las contraseñas se almacenarán cifradas.
  + El sistema deberá tener validación de entrada y prevención de inyecciones SQL.  
    1. **Usabilidad:**
  + La interfaz será sencilla, con formularios claros y mensajes de validación amigables.
  + Compatible con dispositivos móviles a través del navegador.

### **Escalabilidad:**

* + La arquitectura permitirá añadir nuevos módulos o funciones sin afectar los existentes.

### Mantenibilidad:

* + Separación por capas (MVC) para facilitar el mantenimiento y las futuras mejoras.

### Compatibilidad:

* + Funciona en navegadores modernos (Chrome, Firefox, Edge).
  + Compatible con servidores Tomcat y bases de datos MySQL/MariaDB.

### Disponibilidad:

* + El sistema estará disponible para uso local en intranet o en la nube según configuración posterior.

# **Vistas de Casos de Uso**

| Actor | Descripción |
| --- | --- |
| Usuario | Persona que accede al sistema para registrar y gestionar información de mascotas. |
| Administrador | Encargado de supervisar el sistema, gestionar usuarios y generar reportes. |
| Asistente | Representa los procesos automáticos y validaciones internas del sistema. |

| **ID** | **Caso de Uso** | **Actor Principal** | **Descripción breve** |
| --- | --- | --- | --- |
| CU01 | Registrarse | Usuario | Crea una cuenta nueva en el sistema. |
| CU02 | Iniciar sesión | Usuario | Accede al sistema validando sus credenciales. |
| CU03 | Recuperar contraseña | Usuario | Permite recuperar el acceso mediante validación de correo o DNI. |
| CU04 | Registrar mascota | Usuario | Registra una nueva mascota con datos básicos y foto. |
| CU05 | Consultar mascotas | Usuario | Lista las mascotas registradas por el usuario. |
| CU06 | Editar datos de mascota | Usuario | Modifica información de mascotas ya registradas. |
| CU07 | Eliminar mascota | Usuario | Permite eliminar una mascota del registro. |
| CU08 | Registrar historial médico | Usuario | Agrega un nuevo evento médico (vacuna, consulta, etc.) a una mascota. |
| CU09 | Consultar historial médico | Usuario | Visualiza los controles médicos por mascota. |
| CU10 | Descargar carnet de vacunación | Usuario | Descarga un PDF con el historial de vacunación de una mascota. |
| CU11 | Generar reporte de mascotas | Administrador | Genera reportes consolidados de mascotas por usuario, especie o estado. |
| CU12 | Gestionar usuarios | Administrador | Activa, desactiva y edita los datos de los usuarios. |
| CU13 | Registrar asistente veterinario | Administrador | Permite crear cuentas de usuarios con rol veterinario para consultas especializadas. |
| CU14 | Asignar mascota a veterinario | Administrador | Relaciona mascotas a personal asistente para seguimientos clínicos. |
| CU15 | Enviar alertas de vacunación | Asistente | Genera notificaciones automáticas por fechas de vacunas próximas. |
| CU16 | Validar datos ingresados | Asistente | Verifica integridad de datos al momento del registro. |
| CU17 | Notificar cambio de estado de mascota | Asistente | Informa al usuario si su mascota ha sido modificada por un veterinario. |
| CU18 | Generar estadísticas por especie | Administrador | Visualiza estadísticas globales por tipo de animal. |
| CU19 | Registrar adopción | Usuario | Registra el proceso de adopción de una mascota sin dueño. |
| CU20 | Consultar mascotas adoptables | Usuario | Visualiza una lista de mascotas disponibles para adopción. |

# **Vista Lógica**

El sistema **PetGuard** ha sido diseñado utilizando el patrón arquitectónico **Modelo-Vista-Controlador (MVC)**, el cual permite una separación clara entre la presentación, la lógica de negocio y el acceso a los datos. Esta estructura facilita la mantenibilidad, escalabilidad y reutilización del código fuente.

🔷 Modelo

Representa la capa encargada de la lógica de negocio y la estructura de datos. Aquí se incluyen las clases entidad y las interfaces de acceso a base de datos.

* **Clases entidad**:  
  + - Usuario.java
    - Mascota.java
    - HistorialMedico.java
    - Administrador.java
* **Interfaces CRUD**:  
  + - CRUDUsuario.java
    - CRUDMascota.java
    - CRUDHistorial.java  
       Definen métodos como insertar(), listar(), modificar(), eliminar().
* **Clases DAO (Data Access Object)**:  
  + - UsuarioDAO.java
    - MascotaDAO.java
    - HistorialDAO.java  
       Estas implementan las interfaces CRUD y realizan la conexión con la base de datos.
* **Conexión a base de datos**:  
  + clsConexion.java: centraliza los parámetros de conexión y utiliza JDBC para interactuar con MySQL/MariaDB.

🔷 Vista

Representa la interfaz de usuario, desarrollada con **JSP (JavaServer Pages)**, donde se muestran formularios, tablas, reportes y mensajes al usuario. Estas vistas se comunican con los controladores para enviar o recibir información.

Ejemplos:

* index.jsp – Inicio
* login.jsp – Inicio de sesión
* registrarMascota.jsp
* listarMascotas.jsp
* reporteMascotas.jsp

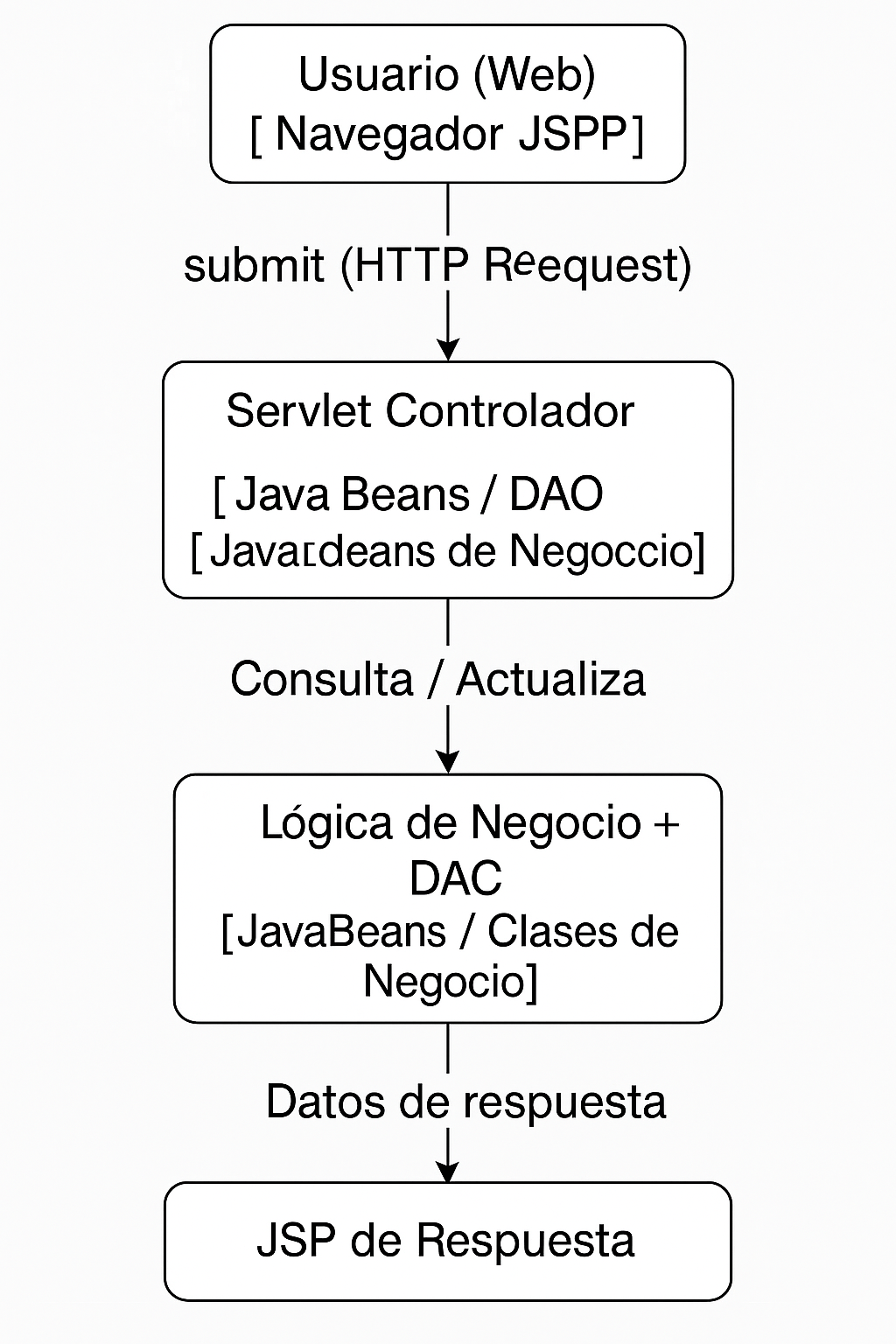
🔷 Controlador

Son los Servlets que coordinan la interacción entre la vista y el modelo. Procesan las solicitudes del usuario, invocan la lógica necesaria y redirigen la respuesta adecuada a la vista.

Ejemplos:

* ControladorUsuario.java – Maneja login, registro y sesiones.
* ControladorMascota.java – Gestiona el registro, edición y consulta de mascotas.
* ControladorHistorial.java – Procesa las acciones sobre el historial médico.
* ControladorAdmin.java – Generación de reportes y acciones administrativas.

## Diagrama Contextual



El usuario accede a registrarMascota.jsp.

Al enviar el formulario, se llama a ControladorMascota?action=registrar.

El servlet recoge los datos, instancia un objeto Mascota, y lo pasa al MascotaDAO.

MascotaDAO inserta el registro en la base de datos.

El servlet redirige al usuario a listarMascotas.jsp mostrando el resultado.

# **Vista de Procesos**

## Diagrama de Proceso Actual

Actualmente, el sistema **PetGuard** se encuentra en desarrollo, por lo tanto, **no existe un sistema anterior formal** que automatice el registro y cuidado de mascotas. Las acciones como el control de vacunas, historial médico o registro de mascotas podrían hacerse en **formato físico o de forma dispersa** (papel, hojas de cálculo, etc.).

**Limitaciones del proceso actual (manual o informal):**

* Pérdida de información por manejo físico.
* Falta de alertas automáticas.
* Inaccesibilidad remota a los registros.
* Baja eficiencia en gestión de historial médico.

## Diagrama de Proceso Propuesto

El nuevo sistema PetGuard automatiza estos procesos de la siguiente manera:

Proceso 1: Registro de Mascota

Actor: Usuario

Flujo:

Usuario accede a registrarMascota.jsp.

Completa los datos del formulario y adjunta imagen.

El formulario se envía al ControladorMascota.

El controlador valida y pasa los datos a MascotaDAO.

Se inserta el registro en la base de datos.

Se redirige a listarMascotas.jsp con mensaje de éxito.

Proceso 2: Inicio de Sesión

Actor: Usuario o Administrador

Flujo:

El usuario accede a login.jsp.

Ingresa DNI y contraseña.

El ControladorUsuario verifica con UsuarioDAO.

Si es válido, se redirige al dashboard respectivo (dashboardUsuario.jsp o dashboardAdmin.jsp).

Si no, se muestra mensaje de error.

Proceso 3: Generación de Reportes

Actor: Administrador

Flujo:

Admin accede a reporteMascotas.jsp.

Selecciona filtros como tipo de mascota o usuario.

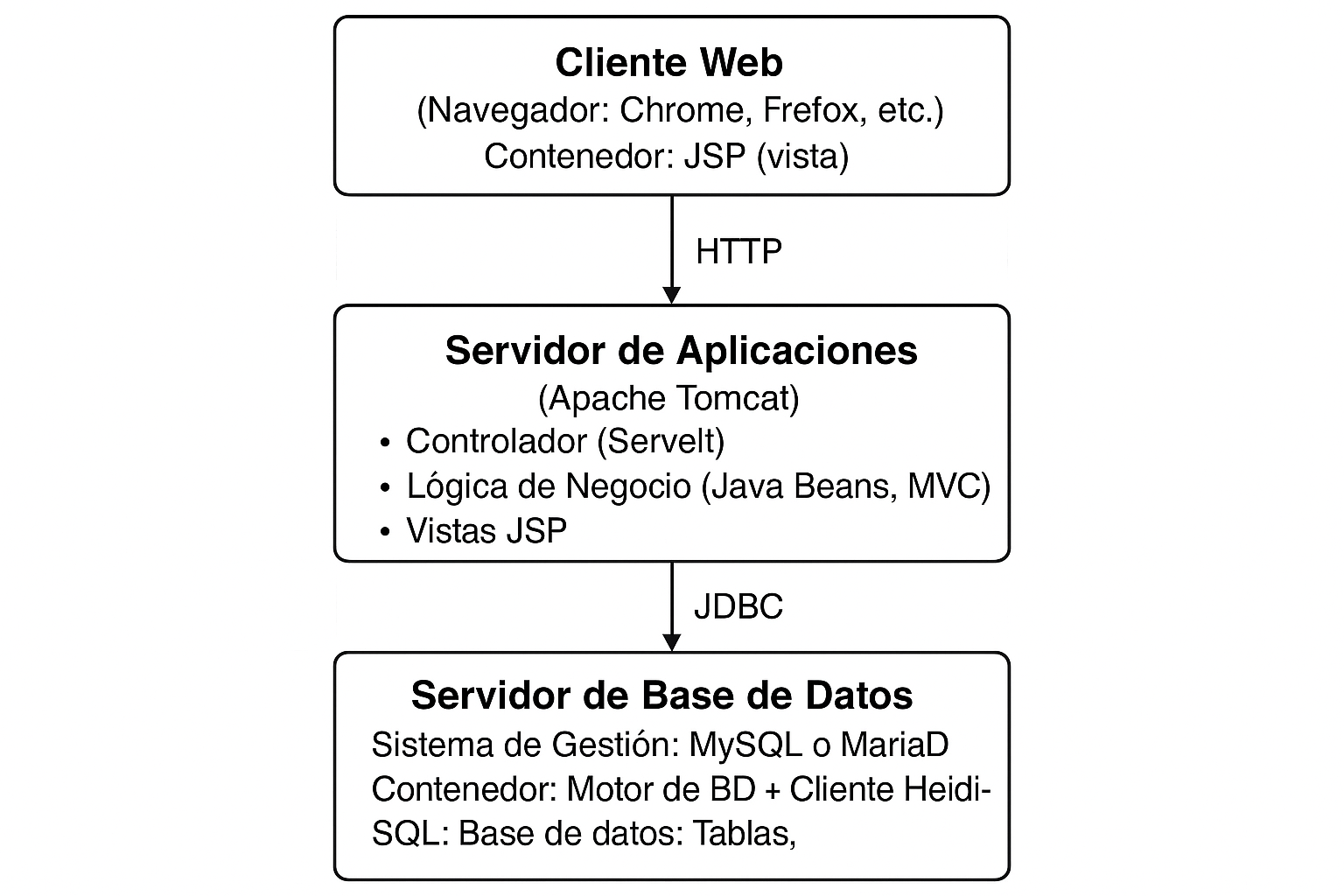
El ControladorAdmin consulta y agrupa datos desde BD.

Se muestra el reporte en tabla o archivo PDF.

# **Vista de Despliegue**

## Diagrama de Contenedor

El sistema se despliega típicamente en una arquitectura de **tres capas**, organizada de la siguiente forma:



Componentes Físicos

Cliente (Frontend)

Navegador web moderno (Chrome, Firefox, Edge).

Accede al sistema mediante URL como http://localhost:8080/PetGuard.

Servidor de Aplicaciones

Apache Tomcat 10 o superior.

Hospeda los archivos .jsp, Servlets, y clases .class.

Maneja peticiones y redirecciona vistas.

Servidor de Base de Datos

MySQL o MariaDB.

Gestionado con HeidiSQL localmente o remotamente.

Almacena datos de usuarios, mascotas, historial médico, etc.

Consideraciones para Producción

Escalabilidad horizontal: Se podrá migrar el sistema a la nube con contenedores (Docker) y usar balanceadores de carga.

Seguridad: Configuración de HTTPS con certificados SSL.

Backups automáticos: Respaldos periódicos de la base de datos usando cron + mysqldump.

Acceso externo: Exposición mediante IP pública, DNS o túneles seguros como No-IP o Ngrok.

# **Vista de Implementación**

## Diagrama de Componentes

Vista (JSP)

Encargada de mostrar información al usuario:

index.jsp – Página principal del sistema.

login.jsp – Formulario de inicio de sesión.

registrarMascota.jsp – Registro de mascotas.

listarMascotas.jsp – Consulta general.

dashboardAdmin.jsp – Panel de control para administrador.

reporteMascotas.jsp – Reporte general por filtros.

Controladores (Servlets)

Gestionan la lógica de navegación y coordinación entre vistas y modelo:

ControladorUsuario.java – Login, gestión de sesiones.

ControladorMascota.java – Registro, edición, listado y eliminación de mascotas.

ControladorHistorial.java – Registro y consulta de historial médico.

ControladorAdmin.java – Generación de reportes, administración de usuarios.

Modelo

Contiene la lógica de negocio y acceso a datos:

Entidades:

Usuario.java, Mascota.java, HistorialMedico.java, Administrador.java

Interfaces CRUD:

CRUDUsuario.java, CRUDMascota.java, CRUDHistorial.java

DAOs:

UsuarioDAO.java, MascotaDAO.java, HistorialDAO.java

Se encargan de insertar, listar, actualizar y eliminar en la BD.

Utilitarios

clsConexion.java: Maneja la conexión a la base de datos usando JDBC.

web.xml: Define los servlets, rutas y parámetros de configuración de la aplicación.

# **Vista de Datos**

## Diagrama Entidad Relación

Entidades Principales

Usuario

dni (PK)

nombre

apellido

correo

clave

rol (usuario, administrador, veterinario)

estado

Mascota

idMascota (PK)

nombre

especie

raza

sexo

fechaNacimiento

foto

codigoQR

dniUsuario (FK) → Relación con Usuario

HistorialMedico

idHistorial (PK)

idMascota (FK)

fecha

tipoControl (vacuna, consulta, desparasitación, etc.)

descripcion

Adopcion

idAdopcion (PK)

idMascota (FK)

fechaAdopcion

adoptadoPor (FK a Usuario)

estado (adoptada / disponible)

AlertaVacuna

idAlerta (PK)

idMascota (FK)

fechaProgramada

tipoVacuna

estado (pendiente / enviada)

Relaciones

Un Usuario puede tener muchas Mascotas (1:N).

Una Mascota puede tener muchos HistorialMedico (1:N).

Una Mascota puede estar relacionada a una Adopción (0 o 1).

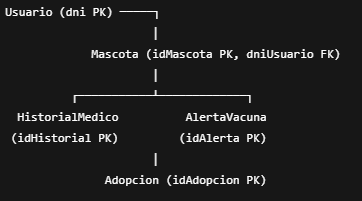
Una Mascota puede tener múltiples Alertas de Vacunación (1:N).

Observaciones Técnicas

Todas las claves primarias son enteros autoincrementales excepto dni, que actúa como PK única para usuarios.

Las imágenes se almacenan como rutas relativas o nombres de archivo.

Se utiliza codificación UTF-8 para soportar acentos y caracteres especiales.



# **Atributos de Calidad**

## Escenario de Seguridad

Nombre: Protección de datos del usuario

Estímulo: Un atacante intenta acceder directamente a la base de datos con inyección SQL.

Fuente: Usuario malicioso.

Entorno: Navegador web mediante formulario.

Artefacto afectado: Formulario de login o registro de mascota.

Respuesta:

El sistema filtra y valida los datos de entrada.

No ejecuta comandos maliciosos en la BD.

Se muestra un mensaje genérico de error sin exponer detalles técnicos.

Medida de respuesta:

100% de intentos de inyección son rechazados en pruebas.

Contraseñas almacenadas con algoritmo hash (ej. SHA-256).

## Escenario de Usabilidad

Nombre: Facilidad de uso en la interfaz

Estímulo: Un usuario nuevo ingresa al sistema por primera vez.

Fuente: Usuario final.

Entorno: Navegador web (PC o móvil).

Artefacto afectado: Interfaz JSP de registro, login, panel principal.

Respuesta:

Los formularios muestran etiquetas y ejemplos claros.

Se indican errores en campos obligatorios.

Los botones tienen nombres intuitivos.

Medida de respuesta:

Tiempo promedio para registrar una mascota: < 2 minutos.

95% de usuarios completan su tarea sin asistencia externa.

## Escenario de Adaptabilidad

Nombre: Inclusión de nuevos módulos

Estímulo: Se requiere implementar un módulo de seguimiento por GPS.

Fuente: Requerimiento funcional nuevo.

Entorno: Desarrollo evolutivo.

Artefacto afectado: Código del sistema.

Respuesta:

Se implementa sin afectar las funcionalidades existentes.

Se reutilizan estructuras existentes de usuarios y mascotas.

Medida de respuesta:

Tiempo estimado de integración: < 3 días.

Impacto en el resto del sistema: < 10% del código modificado.

## Escenario de Disponibilidad

Nombre: Fallo del servidor en horario operativo

Estímulo: El servidor Tomcat deja de funcionar por corte eléctrico.

Fuente: Evento no planificado.

Entorno: Producción local o nube.

Artefacto afectado: Servidor de aplicaciones y base de datos.

Respuesta:

Se emite una alerta.

El sistema puede restaurarse desde backup manual.

Medida de respuesta:

Tiempo de recuperación manual: < 15 minutos con backup local.

Impacto: solo datos en curso no guardados.

## Otro Escenario

Nombre: Campaña de registro masivo de mascotas

Estímulo: 500 usuarios acceden simultáneamente.

Fuente: Evento masivo (adopción nacional).

Entorno: Producción.

Artefacto afectado: Servidor, base de datos.

Respuesta:

El sistema mantiene funcionalidad mínima.

Las operaciones críticas se priorizan (registro, login).

Medida de respuesta:

Tiempo medio de respuesta: ≤ 5 s bajo carga.

Caída controlada de servicios secundarios como reportes.