**1. Introducción General**

**Project Pets** es una aplicación de tipo **full stack** destinada a la **gestión integral** de una clínica veterinaria. Proporciona:

* Un **sistema de roles** para **Administrador**, **Veterinario** y **Cliente**.
* Un **backend** desarrollado en **Java 21** con **Spring Boot 3.4.x**.
* Un **frontend** con **React** y **TypeScript**, construyendo una interfaz modular y responsiva.
* Una **base de datos** MySQL conteniendo los datos de usuarios, mascotas, servicios, inventario y más.
* **Contenedores Docker** orquestados con **Docker Compose** para facilitar el despliegue (por ejemplo, en AWS).

El objetivo es **automatizar** y **organizar** las tareas habituales en una veterinaria, tales como:

1. **Registrar usuarios** (roles distintos).
2. **Administrar mascotas** y su información médica.
3. **Solicitar, programar y actualizar** servicios veterinarios.
4. **Controlar el inventario** (productos, stock).
5. **Generar y consultar** historias clínicas por servicio.

**2. Arquitectura del Backend (Ejemplo de Organización)**

El **backend** sigue un **patrón** basado en las **capas** típicas de Spring Boot:

1. **Capa de Controladores (Controller)**
   * Implementan los **endpoints REST**:
     + Por ejemplo, /api/auth/login para autenticar, /api/admin/usuarios para gestionar usuarios, /api/vet/servicios para los veterinarios, etc.
   * Cada **controlador** se encarga de agrupar la funcionalidad relativa a una entidad o a una parte del dominio (usuarios, mascotas, servicios, inventario...).
   * **EJEMPLO**: El **AuthController** maneja las rutas relacionadas con login, registro y recuperación de contraseña.
2. **Capa de Servicios (Service)**
   * Contiene la **lógica de negocio**.
   * **EJEMPLO**: El **ServicioService** define métodos como “crearServicioCliente” (verifica la mascota, asocia fecha y hora, inicializa estado PENDIENTE) y “actualizarEstadoServicio” (verifica si el veterinario tiene permiso, si está permitido pasar a estado FINALIZADO o CANCELADO, etc.).
3. **Capa de Repositorios (Repository)**
   * Extiende de **Spring Data JPA**, permitiendo **operaciones CRUD** en la base de datos sin excesivo código manual.
   * **EJEMPLO**: El **MascotaRepository** gestiona la tabla “mascota” y ofrece métodos como “findAll()” o “findById()”.
4. **Entidades (Entity)**
   * Mapean las tablas de la base de datos (usuario, rol, mascota, servicio, etc.).
   * **EJEMPLO**: La entidad “Usuario” con campos como “idUsuario”, “correo”, “clave” (encriptada), “rol”, etc.
5. **DTO (Data Transfer Objects)**
   * Diseñados para **transportar datos** entre el backend y el frontend, evitando exponer directamente las entidades.
   * **EJEMPLO**: “UsuarioDTO” con campos de nombre, apellido, correo, y “ServicioDTO” con campos de fecha, hora, estado, referencias a idMascota e idVet.
6. **Configuración y Seguridad**
   * Se emplea **Spring Security** con **JWT**:
     + Un **filtro** se ejecuta antes que los controladores, validando el token JWT en la cabecera.
     + Se definen rutas abiertas (/api/auth/\*\*) y rutas restringidas (por ejemplo, /api/admin/\*\* solo para rol ADMINISTRADOR).
   * El **JWT** se genera al iniciar sesión, incluyendo claims como “rol” y “idUsuario”, con tiempo de expiración.

**Ejemplo de Flujo** en backend:

* Un **cliente** solicita POST /api/cliente/servicios con su token JWT.
* El **filtro de seguridad** comprueba que el token sea válido y que el rol sea CLIENTE.
* El **controlador** (ServicioClienteController) recibe la solicitud, valida datos y llama al **ServicioService**.
* El **ServicioService** llama al **ServicioRepository** para insertar el nuevo registro en la tabla “servicio”, inicializando el estado en “PENDIENTE”.
* Devuelve un **ServicioDTO** con la información recién creada.

**3. Arquitectura del Frontend (Ejemplo de Organización)**

El **frontend** se compone de **React** + **TypeScript**, estructurado de la siguiente manera:

1. **Páginas (Pages)**
   * Se agrupan en directorios, p. ej. pages/DashboardAdmin, pages/GestionUsuarios, pages/Login.
   * Cada **página** representa una sección de la aplicación (registro de usuarios, pantalla de inicio, panel de administración, etc.).
   * **EJEMPLO**: Una página “GestionInventarioAdmin” muestra una tabla, un buscador y un formulario modal para crear/editar productos.
2. **Componentes (Components)**
   * Elementos reutilizables de la interfaz: menús laterales, modales, spinners de carga, alertas, etc.
   * **EJEMPLO**: “MenuAdmin” contiene botones o enlaces a funcionalidades específicas de administradores (usuarios, inventario, servicios, etc.).
3. **Servicios (Services)**
   * Aquí se configuran las llamadas a la **API del backend** (usando librerías HTTP como Axios).
   * **EJEMPLO**: Un archivo usuarioService con funciones “listarUsuariosPaginado”, “crearUsuario”, “eliminarUsuario”, las cuales internamente hacen peticiones GET, POST o DELETE a las rutas del backend.
4. **Contextos y Estado Global**
   * Se define un **AlertContext** para mostrar mensajes emergentes (errores, éxitos) en cualquier parte de la interfaz.
   * Opcionalmente, un **LoadingService** para el spinner global, incrementando un contador cuando se inicia una petición y decrementándolo al terminar, de modo que la UI muestre la animación de carga.
5. **Rutas (React Router)**
   * Se configura un “BrowserRouter” con diversas rutas: “/login”, “/admin”, “/cliente”, “/veterinario”, “/restablecer-clave”...
   * Un **componente privado** (PrivateRoute) puede restringir acceso solo si existe un token JWT y el rol coincide con la ruta.

**Ejemplo de Flujo** en frontend:

* El usuario accede al **Login** e introduce sus credenciales.
* El componente **Login** llama a la función “login(correo, clave)” en authService, que envía la petición al backend.
* Si la respuesta es satisfactoria, el token JWT y el rol se guardan en localStorage.
* El sistema redirige a la **pantalla principal** de acuerdo con el rol (ej. /admin si es un administrador).
* Posteriormente, cada acción (crear usuario, listar mascotas, etc.) invoca servicios de Axios con el token en la cabecera.

**4. Base de Datos y Scripts de Inicio**

1. **MySQL**
   * Se crea la base de datos dbprojectpets.
   * Contiene tablas clave: usuario, rol, mascota, cliente, servicio, historiaclinica, inventario, tiposervicio, etc.
2. **Scripts de Inicialización**
   * Al arrancar el contenedor de MySQL, se ejecutan scripts .sql que **crean las tablas** y **insertan datos** iniciales.
   * Ejemplo:
     + Se definen los roles ADMINISTRADOR, VETERINARIO y CLIENTE.
     + Se crean usuarios de prueba, uno por cada rol, con contraseñas cifradas.
     + Se cargan especies y razas (perros, gatos, aves, conejos...) para que el sistema cuente con un catálogo amplio desde el principio.
     + Se insertan registros en “tiposervicio” (consulta básica, vacunación anual, etc.) y en “inventario” (productos de ejemplo).

Esto permite **usar** la aplicación de inmediato sin necesidad de agregar manualmente la información básica.

**5. Autenticación y Seguridad con Spring Security y JWT**

1. **Login**
   * El usuario envía correo y contraseña a /api/auth/login.
   * El backend valida las credenciales y **genera** un **token JWT**, incluyendo el rol y el idUsuario.
2. **JWT**
   * El token se envía al frontend y se **almacena** en localStorage o sessionStorage.
   * Cada **petición subsecuente** incluye Authorization: Bearer <token>.
3. **Spring Security**
   * Define en la configuración qué rutas son libres (registro, login) y cuáles exigen rol ADMINISTRADOR, VETERINARIO o CLIENTE.
   * Un **filtro** intercepta la petición, valida el token y ajusta el contexto de seguridad para que los controladores sepan quién es el usuario.

**Beneficio**: Se obtiene un **sistema sin estado** (stateless). El backend no mantiene sesiones en memoria, sino que confía en la firma del token JWT.

**6. Envío de Correos (Recuperación de Contraseña)**

* **Objetivo**: Permitir a los usuarios que, si olvidan su contraseña, la restablezcan sin asistencia del administrador.
* **Flujo**:
  1. El usuario solicita restablecer clave, proporcionando su correo.
  2. El backend genera un token temporal (por ejemplo, un UUID) y lo asocia al correo en un registro de memoria o tabla auxiliar.
  3. Se envía un correo con un **enlace** del tipo http://<frontend-url>/restablecer-clave?token=....
  4. El usuario hace clic, ingresa la nueva contraseña y el backend valida el token para actualizar la BD.
* **Tecnología de Correo**:
  1. Se usa un servicio de JavaMailSender (o similar) configurado con un servidor SMTP o un proveedor externo.
  2. El backend compone el asunto y contenido del mensaje, añadiendo el enlace con el token.

**7. Contenedores Docker y Docker Compose**

1. **Docker**
   * El backend se empaqueta en una imagen de Java: se compila con Maven y se ejecuta en un entorno de OpenJDK.
   * El frontend se builda con Node y luego se sirve en una imagen de Nginx, exponiendo el puerto 80.
   * MySQL corre en su imagen oficial, recibiendo volúmenes persistentes y scripts de inicio.
2. **Docker Compose**
   * El archivo docker-compose.yml declara tres servicios: db (MySQL), backend (Spring Boot) y frontend (Nginx).
   * Define las **variables de entorno** (conexión a la base de datos, etc.).
   * Permite levantar todo el **ecosistema** con un único comando, docker-compose up -d.
3. **Despliegue en AWS**
   * En una instancia EC2 (Linux), se instala Docker y Docker Compose.
   * Se clona o transfiere el repositorio con los archivos del proyecto.
   * Al ejecutar docker-compose up -d, se construyen las imágenes (si no están disponibles) y se arranca la aplicación en cuestión de minutos.

**8. Flujos Clave y Ejemplos de Uso**

1. **Registro y Login**
   * Un cliente se registra con nombre, correo y contraseña, recibiendo por defecto el rol CLIENTE.
   * Inicia sesión en la ruta de login, el backend valida y genera JWT.
2. **Gestión de Usuarios (Admin)**
   * El administrador ingresa en /admin/usuarios.
   * Ve una **lista paginada** de usuarios. Puede crear nuevos (seleccionando el rol), editar o eliminarlos.
   * El sistema valida si el correo es único y encripta la contraseña en la BD.
3. **Mascotas y Clientes**
   * Un cliente (rol CLIENTE) en su Dashboard va a “Mis Mascotas”.
   * Puede **crear** una mascota, eligiendo especie y raza de catálogos precargados (p. ej., “Perro” - “Labrador Retriever”).
   * La mascota queda asociada a su idUsuario, quedando visible sólo para él y para roles con permisos especiales (ADMIN).
4. **Servicios Veterinarios**
   * El cliente crea un nuevo servicio para su mascota, escogiendo el **tipo de servicio** (consulta, vacunación) y una **fecha/hora**.
   * El estado del servicio inicia en PENDIENTE.
   * Un veterinario accede a “Servicios Vet”, ve los servicios pendientes y puede aceptarlos, poniéndolos EN\_CURSO.
   * Al finalizar, actualiza el estado a “FINALIZADO” y añade notas en la historia clínica.
5. **Inventario**
   * El administrador ingresa a “Gestión de Inventario” y ve la **lista de productos** con stock disponible.
   * Puede crear nuevos productos, actualizar cantidades, etc.
   * El veterinario generalmente sólo visualiza el inventario para confirmar existencias antes de proceder con un servicio.

**9. Conclusión**

1. **Modularidad**:
   * Separación backend–frontend, roles bien delimitados, y rutas con autorizaciones distintas.
   * Posibilita que un equipo se encargue del backend en Java y otro del frontend en React.
2. **Seguridad**:
   * Con Spring Security + JWT se implementa un sistema ágil y escalable, sin almacenar sesiones en el servidor.
3. **Experiencia de Usuario**:
   * El frontend presenta menús específicos según el rol:
     + Administrador maneja usuarios, inventario, tipos de servicio y servicios globales.
     + Veterinario controla servicios asignados, historia clínica e inventario en modo lectura.
     + Cliente maneja sus mascotas, agenda servicios y revisa historiales.
4. **Contenedores y Despliegue**:
   * Docker Compose garantiza reproducibilidad en cualquier entorno.
   * Escalable fácilmente en la nube, con posibilidad de añadir balanceadores y más réplicas del backend.