DESARROLLO DE APLICACION JAVA

Proyecto Ayres de Etcheverry

Profesor: Silva Miguel

Repositorio:
https://github.com/DanielCardozo25/Proyecto-Guarderia-Vehiculos

González Elías
Chuca Nahuel
Marcos Apaza
Cardozo Daniel
Programación II - 2601 -2025

DESCRIPCION GENERAL

Tema seleccionado: Ayres de Etcheverry

Objetivos del sistema es gestionar la información de una guardería de motorhomes, casas rodantes de arrastre, caravanas y trailers, referida a sus instalaciones, empleados, socios y vehículos que guarda

Actores y sus roles

Administrador: Es el encargado de Administrar el Sistema, realizar el CRUD de todos los actores, asigna tareas a los empleados y asigna vehículos a los garajes.

Empleado: Se encargan de realizar las tareas asignadas y avisar de que fueron realizadas.

Socio: Son los clientes que pueden ver sus vehículos en que sección se encuentran, asignar garajes si es que son propietarios. Y añadir servicios, prestados por la empresa para sus vehículos.

CAPAS DEL SISTEMA

1. Capa de Presentación

- Propósito: Manejar las peticiones HTTP del cliente (navegador/interfaz), gestionar el flujo de la aplicación y devolver la vista o el dato.

- Componentes Clave:
  - Controladores (@Controller o @RestController en Spring): Son la entrada para las interacciones de los actores (Administrador, Empleado, Socio).
  - Motor de Plantillas (Thymeleaf): Se usa para renderizar las vistas (HTML) que el usuario final ve.

- Flujo: Recibe una petición, la valida mínimamente, determina qué lógica de negocio se necesita y llama a la Capa de Servicio. También gestiona la diferenciación de funcionalidades basada en los 3 actores requeridos.

CAPAS DEL SISTEMA

2. Capa de Servicios, negocio.

- **Propósito: Contener la lógica de negocio, específicar las reglas del sistema (listar socios por antigüedad, asignar vehículo verificando dimensiones).**

- **Componentes:**
  - **Servicios (@Service en Spring): Clases que contienen los métodos que implementan la lógica del negocio.**
  - **Principios SOLID: Aplicar los principios como la Responsabilidad Única (S) y la Inversión de Dependencias (D).**

- **Flujo: Recibe las peticiones del Controlador, aplica la lógica necesaria, utiliza la Capa de Repositorio para obtener o modificar datos, y devuelve el resultado al Controlador.**

CAPAS DEL SISTEMA

3. Capa de Persistencia y repositorio

- **Propósito: Abstraer el acceso a la base de datos relacional (MySQL) y gestionar las operaciones CRUD completas (ABM).**

- **Componentes:**
  - **Repositorios (@Repository en Spring Data JPA): Interfaces que interactúan con el objeto o entidad.**
  - **Frameworks: Spring Data JPA e Hibernate.**

- **Flujo: Recibe las peticiones de manipulación de datos de la Capa de Servicio y se comunica directamente con la Base de Datos Relacional diseñada con el Modelo Entidad-Relación. Devuelve las entidades (Modelo) resultantes.**

CAPAS DEL SISTEMA

4. Capa de Modelo - Entidades

- **Propósito: Representar la estructura de los datos persistentes del sistema, mapeando las clases de Java a las tablas de la base de datos (según el Modelo Entidad-Relación).**

- **Componentes**

:

  - **Entidades (@Entity de JPA): Administrador, Empleado Socio, Vehiculo, Garage y Zona.**

  - **Frameworks: Hibernate se encarga de la traducción entre estas clases y las filas de la BD.**

  - **Principios de POO: Implementa la Abstracción, Encapsulamiento y sirve de base para la Herencia y el Polimorfismo.**

ESTRUCTURA DEL PROYECTO

1. **Pilares de la POO**

1.1 **Herencia y Abstracción**

Para modelar la jerarquía de los usuarios del sistema, se ha implementado una jerarquía de herencia:

Persona (Administrador, socio y empleado): Base abstracta para manejar los roles de acceso y credenciales del sistema, clave para el flujo en la Capa de Control.

1.2 **Polimorfismo**

Se aplica en la gestión de las interacciones de los usuarios y en la persistencia:

- Gestión de Actores: Las clases Administrador, Socio y Empleado implementan una interfaz común permitiendo que la capa de Control trate a los tres actores de manera uniforme al gestionar el login y la validación de permisos, pero ejecutando funcionalidades distintas según el tipo real del objeto.
- Servicios: Los servicios exponen métodos polimórficos manejando diferentes tipos de vehículos o diferentes tipos de servicios de mantenimiento.

ESTRUCTURA DEL PROYECTO

1.3 Encapsulamiento

El encapsulamiento se aplica rigurosamente en la Capa de Modelo. Todos los atributos de las entidades (Socio, Vehículo, Garaje, etc.) se declaran como private, y su acceso se gestiona exclusivamente a través de métodos public getters y setters.

2. Componentes Clave por Dominio

Las siguientes clases representan los elementos centrales del negocio y se encuentran en la capa de Modelo, mapeadas directamente a la Base de Datos Relacional:

Componente	Atributos Clave	Relaciones con otras Clases
Socio	DNI, Nombre, Fecha de Ingreso	1-N con Vehiculo. 1-N con Garage.
Vehiculo	Matrícula, Tipo, Dimensiones (ancho, prof.)	N-1 con Socio. 1-1 con Garage.
Garage	Nro. Garage, Lectura Luz, Mantenimiento	N-1 con Zona. 1-1 con Vehiculo.
Zona	Letra, Tipo de Vehículos, Capacidad	1-N con Garage. N-N con Empleado.
Empleado	Código, Especialidad	N-N con Zona.

TECNOLOGIA Y FRAMEWORKS

El desarrollo del proyecto está basado en la plataforma Java (JDK) con una serie de herramientas y frameworks que permitieron crear un proyecto robusto y escalable.

1. Spring Boot (Arquitectura y Despliegue)
Es el framework principal que se utiliza para configurar, construir y desplegar la aplicación. Su objetivo es simplificar la configuración inicial y el desarrollo de aplicaciones basadas en el ecosistema Spring permitiendo los siguientes aspectos:

- Configuración automática del Servidos Web con la Base de Datos.
- Arquitectura de capas (Controlador, Servicio, Repositorio) mediante la Inversión de Control (IoC) y la Inyección de Dependencias (DI).
- Spring MVC permite el desarrollo de la Capa de Presentación (.controller) para gestionar las peticiones HTTP y el uso de la aplicación
- Servidor Embebido Incluye el servidor web (Tomcat) directamente en el ejecutable.

TECNOLOGIA Y FRAMEWORKS

2. Spring Data JPA y Hibernate (Persistencia)

La gestión de la persistencia completa de datos (ABM) se realiza a través de la integración de estos dos frameworks clave, cumpliendo con el requisito de usar una base de datos relacional.

2.1 Hibernate (Implementa JPA)

Hibernate es una implementación de Java Persistence API (JPA). Es un Mapeo de Objetos-Relacional (ORM) entre los objetos Java y las tablas de la base de datos.

- Objeto-Relacional: Permite que las Entidades (.model) (ej. Socio.java) se relacionen directamente con las tablas de la base de datos, eliminando la necesidad de escribir la mayor parte del código SQL manual.
- Manejo de Relaciones: Traduce las relaciones de POO (Herencia, 1-N, N-N) a comandos SQL y claves foráneas en la Base de Datos.
- Base de Datos Relacional: El uso de Hibernate utiliza el Modelo Entidad-Relación formal creado en MySQL, cumpliendo con el requisito del proyecto.

TECNOLOGIA Y FRAMEWORKS

2.2 Spring Data JPA (Simplificación de Repositorios)

Simplifica el uso de Hibernate en la Capa de Repositorio (.repository).

- ABM sin Código: Permite definir interfaces de repositorio (ej. SocioRepository) que, al extender JpaRepository, heredan automáticamente métodos esenciales para las operaciones CRUD (save, findById, findAll, delete).

- Consultas Automáticas: Permite definir consultas complejas simplemente nombrando los métodos de las interfaces (ej. findByDni(String dni) o findByOrderByFechaIngresoAsc()), delegando la implementación SQL a Spring Data JPA y Hibernate.

TECNOLOGIA Y FRAMEWORKS

3. Base de Datos Relacional (MySQL)

MySQLes nuestro sistema de gestión de base de datos relacional (SGBDR) para almacenar la información del proyecto de manera estructurada y persistente, Alineándose con el Modelo Entidad-Relación diseñado.

- Almacenamiento Persistente: Garantiza que los datos de Socios, Vehículos, Garajes, Zonas y Empleados persistan entre las ejecuciones del sistema.

- Integridad de Datos: Aprovecha las restricciones relacionales para mantener la coherencia (ej. asegurar que un Vehículo siempre esté asociado a un Socio existente).

CLASES PRINCIPALES

1. Clases de punto de entrada (MVC)

1.1 Clases Entidades (Modelo): Representan los objetos y las tablas de la base de datos. Contienen atributos, getters, setters y validaciones básicas

1.2 Clases Repositorio (Modelo): Contienen métodos para realizar operaciones CRUD y se relaciona con la Base de Datos.

1.3 Clases Servicio (Modelo): Coordina la aplicación y el repositorio. El Controlador solo debe llamar a estas clases.

1.4 Clases de Controlador (Controlador): Valida los parametros de entrada, recibe las peticiones de HTTP y llama a los servicios necesarios. Selecciona las vistas y devuelve una respuesta.

1.5 Clases Vista son aquellas que tienen Archivos HTML o Js, se encarga de la parte visual que vera en el usuario en la pagina Web.

CODIGO RELEVANTE

ENTIDAD

```
public abstract class Persona {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private Integer dni;
    private String nombre;
    private String direccion;
    private String telefono;
    private String contrasenia;

    public Persona() {
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public Integer getDni() {
        return dni;
```

CODIGO RELEVANTE

REPOSITORIO

```java
import com.proyectofinal.guarderia.guarderia_web.modelos.Administrador;
import org.springframework.data.repository.CrudRepository;

public interface AdministradorRepository extends CrudRepository<Administrador, Integer> {


    Administrador findByDniAndContrasenia(Integer dni, String contrasenia);
}
```

SERVICIO

```java
@Service
public class AdministradorService {

    @Autowired
    private AdministradorRepository administradorRepository;

    public Administrador login(Integer dni, String contrasenia) {
        return administradorRepository.findByDniAndContrasenia(dni, contrasenia);
    }

    public Administrador guardar(Administrador administrador) {
        return administradorRepository.save(administrador);
    }

    public Administrador buscarPorId(Integer id) {
        return administradorRepository.findById(id).orElse(null);
    }

    public void eliminar(Integer id) {
        administradorRepository.deleteById(id);
    }
```

CODIGO RELEVANTE

CONTROLADOR

```java
@Controller
@RequestMapping("/administrador")
public class ControladorAdministrador {

    @Autowired
    private AdministradorService administradorService;

    // ------------------------------------------------------
    // HOME
    // ------------------------------------------------------

    @GetMapping("/")
    public String paginaPrincipal() {
        return "index";
    }

    // ------------------------------------------------------
    // LOGIN
    // ------------------------------------------------------

    @GetMapping("/login")
    public String mostrarLogin(Model model) {
        model.addAttribute("administrador", new Administrador());
        return "login-administrador";
    }
```

VISTA

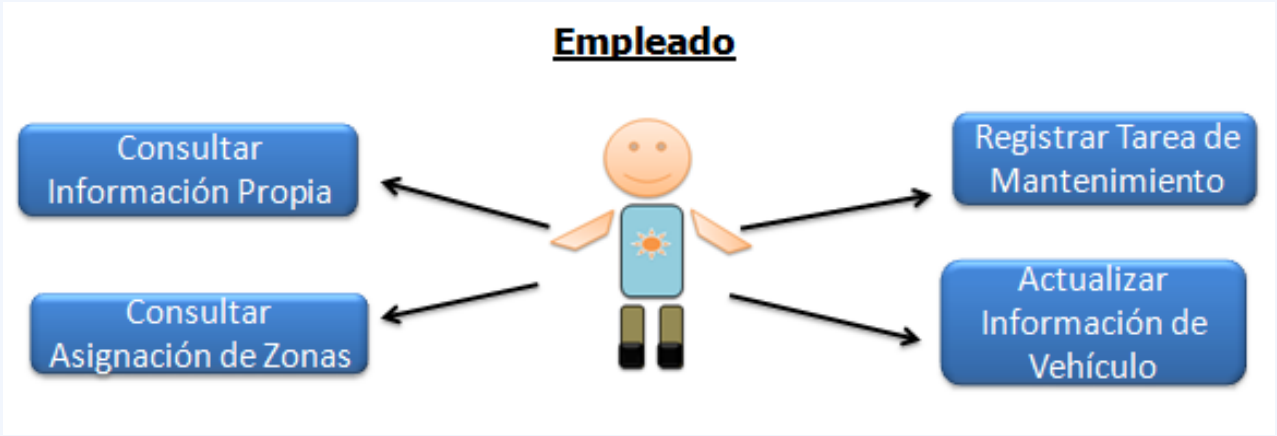
```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Lista de Administradores</title>
      <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.8/dist/css/bootstrap.min.css" rel="stylesheet" i

</head>
<body class="d-flex flex-column align-items-center min-vh-100 py-4 bg-body-tertiary">
    <div class="container py-3">
        <header>
            <nav class="navbar navbar-expand-md navbar-primary fixed-top bg-primary">
                <div class="container-fluid">
                    <ul class="nav nav-pills">
                        <li class="nav-item">
                        <a class="nav-link active" aria-current="page" th:href="@{/administrador/logout}">Cer
                        </li>
                    </ul>
                </div>
            </nav>
            <br>
            <br>
            <br>
            <div class="pricing-header p-3 pb-md-4 mx-auto text-center">
                <h1 class="display-4 fw-normal text-body-emphasis">Lista de Administradores</h1>
            </div>
```

DIAGRAMA CASOS USO

Puede crear nuevos Administradores, gestionar (CRUD) a los socios, empleados, vehículos, mantenimientos de los empleados y consultar por el estado general de la guardería

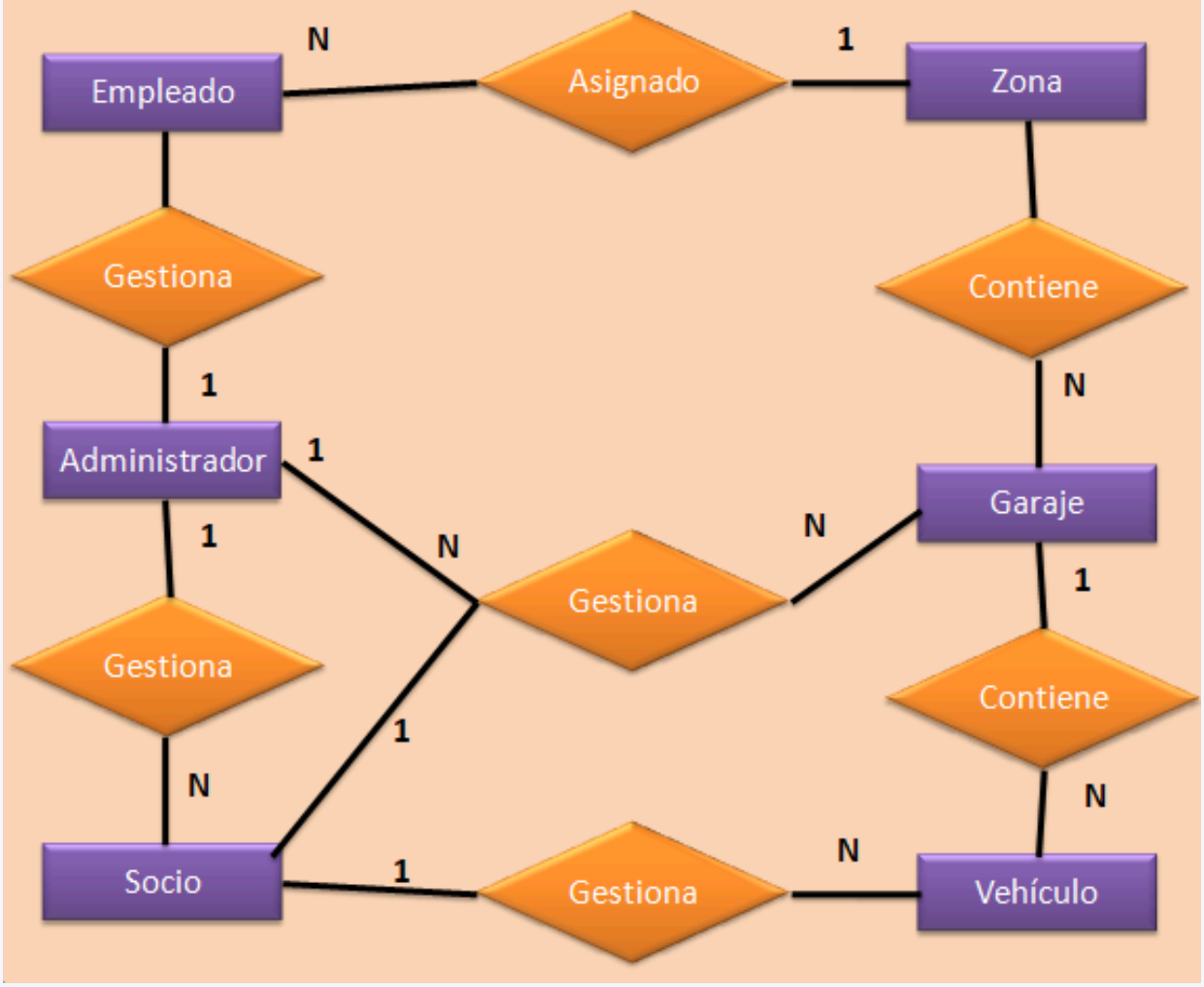
DIAGRAMA CASOS USO



Puede consultar sus propios datos, consultar que zonas y tareas tiene asignadas y actualizar el estado del vehículo (Si se realizo lo solicitado o no)

Puede consultar sus propios datos, Solicitar servicios a sus vehículos, consultar sobre donde está ubicado su garaje y actualizar sus datos y agregar o dar de baja vehículos

DIAGRAMA ENTIDAD RELACION



Administrador: Datos administrador, FK con Empleado, Socio y Garajes.
Empleado: Datos Empleados, FK con Administrador, Zona y Vehículos.
Socio: Datos Socio, FK con Administrado, Garajes y Vehículos.
Zona: Segmenta las zonas en base a las dimensiones de los garajes.
Garajes: Es donde se colocan vehículos dependiendo de las características dadas en las zonas
Vehículos: Datos del mismo con FK con su Garaje y Socio.

MANUAL DE USUARIO

El Sistema esta divido en 3 Secciones Administrador, Empleado o Socio.

1. **Administrador**

1.1 Para ingresar el Administrador debe ingresar su DNI y contraseña.
1.2 Gestionar Administradores: Ver los creados y editarlos.
1.3 Crear Administrador: Ingresando los datos requeridos.
1.4 Gestionar Empleados: Ver los creados, crear nuevos y editarlos.
1.5 Crear Empleados: Ingresando los datos requeridos.
1.6 Gestionar Socios: Ver los creados, crear nuevos y editarlos.
1.7 Crear Socio: Ingresando los datos requeridos.
1.8 Gestionar Vehículos: Ver los creados, crear nuevos y editarlos.
1.9 Crear Vehículos: Ingresando los datos requeridos.
1.10 Gestionar Garaje: Ver los creados, crear nuevos y editarlos.
1.11 Crear Garaje: Ingresando los datos requeridos.

MANUAL DE USUARIO

1.12 Gestionar Zonas: Ver los creados, crear nuevos y editarlos.
1.13 Crear Zonas: Ingresando los datos requeridos.
1.14 Gestionar Mantenimientos: Ver los creados, crear nuevos y editarlos.
1.15 Crear Mantenimientos: Ingresando los datos requeridos.
1.16 Gestionar Especialidades: Ver los creados, crear nuevos y editarlos.
1.17 Crear Especialidades: Ingresando los datos requeridos.
1.18 Gestionar Renta de Garaje: Ver los creados, crear nuevos y editarlos.
1.19 Crear Renta de Garaje: Ingresando los datos requeridos.
1.20 Gestionar Orden de Trabajo: Ver los creados, crear nuevos y editarlos.

MANUAL DE USUARIO

2. **Empleado**

2.1 Ver mis Datos: Datos del mismo.
2.2 Ver Ordenes de trabajo asignada: Sus propias tareas asignadas, con el ID de asignación, el vehículo (matricula) y el mantenimiento a realizar.
2.3 Consultar Vehículos Registrados: Autos designados con todas sus características.
2.4 Consultar tipos de mantenimientos: Tipos de Mantenimientos que realiza el mismo empleado.

3. **Socio**

3.1 Gestionar mis garajes rentados y vehículos asignados.
3.2 Gestionar mis vehículos.
3.3 Gestionar mis Mantenimientos Contratadas.