Universidad Autónoma de Chihuahua

Facultad de ingeniería

Ingeniería en Ciencias de la Computación

Fundamentos de bases de datos

Manual Tecnico PetCare

Asesor: José Saúl De Lira Miramontes

Autores:

Rebeca Portillo Saenzpardo 368094

Manuel Martínez Martínez 368064

Pablo Gael Torres 368073

Fecha: 28/05/2025

Tecnologías usadas

Backend (Servidor):

Node.js: Entorno de ejecución para JavaScript en el servidor.

Express.js: Framework para crear el servidor web y las rutas API.

oracledb: Driver para conectar Node.js con Oracle Database.

CORS: Middleware para permitir peticiones entre frontend y backend.

body-parser: Middleware para procesar datos JSON y formularios.

Base de datos:

Oracle Database

Sistema de gestión de bases de datos relacional (usando tablas como CITA, DUENO, MASCOTA, etc.).

Frontend (Cliente):

HTML: Estructura de las páginas web.

CSS: Estilos personalizados (archivo estilos.css).

JavaScript: Lógica del lado del cliente para interacción y consumo de la API.

Estructura del proyecto

```
petcare-system/
├── client/                    # Frontend ( HTML/CSS/JS)
│   ├── css/
│   │   └── estilos.css
│   ├── js/
│   │   ├── duenos.js          # Lógica de dueños
│   │   ├── mascotas.js        # Lógica de mascotas
│   │   ├── citas.js           # Lógica de citas
│   │   ├── consultas.js       # Lógica de consultas
│   │   ├── tratamientos.js    # Lógica de tratamientos
│   │   ├── veterinarios.js    # Lógica de veterinarios
│   │   └── main.js            # Punto de entrada del frontend
│   └── index.html
├── server/                    # Backend (Node.js)
│   ├── config/
│   │   └── database.js        # Configuración de Oracle
│   ├── controllers/           # Logica para manejar las solicitudes HTTP
│   ├── models/                # Modelos de base de datos
│   ├── routes/                # Rutas API
│   ├── app.js                 # Aplicación principal
│   └── server.js              # Servidor
├── package.json
└── README.md
```

Explicación del código

server/server.js – Punto de entrada del backend

Este archivo es el núcleo principal del backend en el sistema PetCare. Se encarga de iniciar el servidor web con Express.js, configurar los middlewares, conectar con la base de datos Oracle y registrar todas las rutas de la API.

Funcionalidades clave:

1. Inicialización del servidor

Se importan los módulos necesarios y se crea la instancia de Express:

```
const express = require('express');

const cors = require('cors');

const bodyParser = require('body-parser');

const path = require('path');

const db = require('./config/database');
```

2. Middlewares esenciales

Se configuran middlewares para permitir solicitudes entre dominios, y para procesar datos JSON y formularios:

```
app.use(cors()); // Permite peticiones desde el frontend

app.use(bodyParser.json()); // Parsea application/json

app.use(bodyParser.urlencoded({ extended: true })); // Parsea form-data
```

3. Archivos estáticos

Se sirve el frontend directamente desde la carpeta client/:

```
app.use(express.static(path.join(__dirname, '../client')));
```

4. Rutas API

Aquí se importan y registran todas las rutas del sistema (dueños, mascotas, citas, etc.), cada una apuntando a su módulo respectivo en la carpeta routes/:

```
const duenosRoutes = require('./routes/duenos');

const mascotasRoutes = require('./routes/mascotas');

const veterinariosRoutes = require('./routes/veterinarios');

const consultasRoutes = require('./routes/consultas');

const citasRoutes = require('./routes/citas');

const tratamientosRoutes = require('./routes/tratamientos');
```

5. Manejo global de errores

Cualquier error que no se haya capturado previamente se maneja aquí para evitar caídas y dar una respuesta controlada al cliente:

```
app.use((err, req, res, next) => {

  console.error('Error global:', err);

  res.status(500).json({ error: err.message });

});
```

6. Inicialización del servidor y la base de datos

Antes de levantar el servidor, se establece la conexión con Oracle usando el archivo database.js. Si todo sale bien, se arranca el servidor en el puerto 3000 (o el definido por entorno):

```
db.initialize()

  .then(() => {

    const port = process.env.PORT || 3000;

    app.listen(port, () => {

      console.log(`Servidor listo en http://localhost:${port}`);

    });

  })  .catch(err => console.error('Error inicializando DB:', err));
```

server/config/database.js – Conexión a Oracle Database

Este archivo es el encargado de establecer la conexión entre el backend Node.js y la base de datos Oracle. Centraliza la lógica para conectarse, ejecutar consultas y cerrar conexiones. Usamos el módulo oficial `oracledb`.

Funcionalidades clave:

1. Importación del módulo oracledb

```
const oracledb = require('oracledb');
```

Este módulo permite a Node.js comunicarse con bases de datos Oracle mediante conexiones directas y SQL.

2. Configuración de la conexión

```
const dbConfig = {

  user: 'petcare',

  password: 'petcare123',

  connectString: 'localhost:1521/xe' // Cambia según tu configuración

};
```

3. Pool de conexiones

```
let pool;
```

Se declara una variable para guardar el pool de conexiones. Este pool permite que múltiples partes del sistema usen la base de datos sin tener que abrir y cerrar conexiones manualmente cada vez.

4. Inicialización del pool

```
async function initialize() {

  pool = await oracledb.createPool(dbConfig);

  console.log('Conexión a Oracle establecida');}
```

Esta función se llama al iniciar la aplicación `server.js`. Crea el pool y deja lista la base de datos para ejecutar consultas.

5. Cierre del pool

```
async function close() {
  if (pool) {
    await pool.close();
  }
}
```

Se encarga de cerrar el pool de conexiones cuando se apaga el servidor. Evita que queden conexiones colgadas.

6. Ejecución de consultas SQL

```
async function simpleExecute(statement, binds = [], opts = {}) {
  let conn;
  try {
    conn = await pool.getConnection();
    const result = await conn.execute(statement, binds, opts);
    return result;
  } catch (err) {
    console.error('Error en simpleExecute:', err);
    throw err;
  } finally {
    if (conn) {
      try {
        await conn.close();
      } catch (err) {
        console.error('Error al cerrar conexión:', err);
      }
    }
```

Esta función hace todo lo necesario para ejecutar una consulta:

7. Exportación del módulo

```
module.exports = {
  initialize,
  close,
  simpleExecute
};
```

Esto permite usar las funciones desde otros archivos del backend (

routes/, controllers/ y models/ – Lógica del Backend REST API

La arquitectura del backend de PetCare sigue el patrón **MVC (Modelo- Vista -controlador)** adaptado para una API REST. Cada uno de estos directorios cumple una función específica:

`controllers/` – Lógica de negocio

Los controladores hacen lo siguiente:

1. Reciben la solicitud (req).
2. Validan los datos si es necesario.
3. Llaman al modelo correspondiente para interactuar con la base de datos.
4. Devuelven una respuesta al cliente (res).

`controllers/mascotas.js`:

```
const Mascota = require('../models/mascota');
const db = require('../config/database');

exports.createMascota = async (req, res) => {
  if (!req.body || !req.body.nombre || !req.body.especie ||
!req.body.id_dueno) {
    return res.status(400).json({
```

```
      success: false,
      message: 'Datos incompletos: nombre, especie y dueño son
requeridos'
    });
  }
```

La estructura se repite para `dueños`, `veterinarios`, `citas`, etc., cada uno con sus funciones CRUD.

`routes/` – Rutas de la API

Aquí se definen las URLs del servidor backend y a qué función del controlador deben llamar.

En `routes/mascotas.js` se definen rutas como:

```
router.post('/', mascotasController.createMascota);
router.get('/', mascotasController.getAllMascotas);
router.get('/duenos', mascotasController.getDuenosForSelect);
```

Estas rutas permiten que el cliente (frontend) pueda hacer solicitudes como:

- `GET /api/mascotas` → Ver todas las mascotas
- `POST /api/mascotas` → Crear una nueva mascota

Cada archivo de rutas importa su respectivo controlador y lo asocia a los endpoints.

`models/` – Acceso a la base de datos

Los modelos encapsulan el acceso a la base de datos Oracle, utilizando de database.js.

Ejemplo con `models/mascotas.js`:

```
const db = require('../config/database');
const oracledb = require('oracledb');
```

```
class Mascota {

  static async create(nombre, especie, raza, edad, id_dueno) {

    const sql = `

      INSERT INTO "MASCOTA" (NOMBRE, ESPECIE, RAZA, EDAD, ID_DUEÑO)

      VALUES (:nombre, :especie, :raza, :edad, :id_dueno)

      RETURNING ID_MASCOTA INTO :id

    `;

    const binds = {

      nombre,

      especie,

      raza,

      edad,

      id_dueno,

      id: { type: oracledb.NUMBER, dir: oracledb.BIND_OUT }

    };
```

Aquí es donde se define la consulta SQL específica que se ejecutará, y normalmente se devuelve el resultado directamente al controlador.

Frontend

El sistema **PetCare** está diseñado como una aplicación web modular enfocada en la gestión integral de una clínica veterinaria. La arquitectura se basa en una interfaz construida con HTML, CSS y JavaScript, y se apoya en una base de datos relacional (por ejemplo, Oracle SQL) para el almacenamiento persistente. A continuación, se describen los módulos principales desde una perspectiva técnica:

1. Módulo de Dueños

Este módulo gestiona los registros de los propietarios de las mascotas. Cada dueño se representa como una entidad en la base de datos, con atributos como `id_dueno`, `nombre`, `apellido`, `direccion`, `telefono` y `correo`. Se implementan operaciones CRUD (Create, Read, Update, Delete) mediante

formularios en la interfaz web que se comunican con el backend para actualizar la base de datos. La integridad referencial se garantiza mediante claves foráneas que vinculan a los dueños con sus respectivas mascotas.

2. Módulo de Mascotas

El módulo de mascotas registra entidades que contienen información como `id_mascota`, `nombre`, `especie`, `raza`, `sexo`, `fecha_nacimiento`, y una clave foránea `id_dueno` que establece la relación con su propietario. Este módulo permite navegar entre mascotas asociadas a un dueño, y consultar historiales médicos o actividades vinculadas (como citas, tratamientos y consultas). Técnicamente, permite realizar joins entre tablas para recuperar información relacional.

3. Módulo de Citas

Este módulo administra las solicitudes de atención médica programadas. En términos de base de datos, maneja una entidad `cita` con campos como `id_cita`, `id_mascota`, `id_veterinario`, `fecha_hora`, `motivo`, y `estado`. Se manejan validaciones de fechas, conflictos de horario y restricciones de integridad para evitar asignaciones duplicadas. La interfaz permite seleccionar una mascota y programar una cita con un veterinario específico.

4. Módulo de Consultas

Las consultas representan visitas clínicas efectivas realizadas a las mascotas. Cada registro incluye `id_consulta`, `id_cita` (clave foránea), `diagnostico`, `observaciones`, y `fecha_consulta`. Este módulo permite la documentación de las atenciones médicas y se vincula con los módulos de Citas, Mascotas y Tratamientos. Técnicamente, las consultas se integran con vistas o reportes médicos generados a partir de múltiples relaciones entre tablas.

5. Módulo de Tratamientos

Aquí se registran los tratamientos prescritos tras una consulta médica. La tabla `tratamiento` incluye campos como `id_tratamiento`, `id_consulta`, `medicamento`, `dosis`, `frecuencia`, `duracion`, y `observaciones`. Se mantiene un historial de tratamientos por mascota, que puede ser consultado desde el módulo correspondiente. Este módulo permite generar reportes cronológicos y validar si un tratamiento está en curso o finalizado.

6. Módulo de Veterinarios

Este módulo centraliza la información del personal médico. Se estructura mediante una tabla `veterinario` con campos como `id_veterinario`, `nombre`, `apellido`, `especialidad`, `telefono`, `correo`, y `horario`. Su propósito es facilitar la asignación de citas y consultas a profesionales específicos. Este módulo también puede incluir autenticación o permisos de usuario en versiones más avanzadas del sistema.

Interacción entre módulos

Todos los módulos están relacionados mediante claves primarias y foráneas, formando una estructura de base de datos normalizada. La interfaz web se estructura con secciones específicas enlazadas mediante anclas en el HTML (`#dueños`, `#mascotas`, etc.), y cada módulo incluye botones de navegación y formularios de interacción. La lógica de control puede implementarse con JavaScript o integrarse con un backend en PHP, Node.js, u otra tecnología, dependiendo del alcance del sistema.

Hoja de estilos CSS

1. Variables globales `:root`

Defines una paleta de colores reutilizable:

```
:root {
  --primary: #00b894;
  --secondary: #0984e3;
  --dark: #2d3436;
  --light: #f5f6fa;
  --danger: #d63031;
  --warning: #fdcb6e;
```

Estas variables se usan para mantener la consistencia visual y facilitar cambios temáticos.

2. Reset general y estilos base

```
margin: 0;

  padding: 0;

  box-sizing: border-box;

  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

}
```

Esto reinicia márgenes y paddings, establece una caja de contorno más predecible y define la fuente predeterminada.

3. Estructura del layout principal

```
body {

  display: flex;

  min-height: 100vh;

  background-color: #f1f2f6;

}
```

Hace que el body actúe como un contenedor flex horizontal, para permitir un sidebar fijo a la izquierda y contenido a la derecha.

4. Menú lateral (sidebar)

- .sidebar: Contenedor lateral con fondo blanco, sombra y padding.
- .logo: Contiene el nombre/logo del sistema.
- .nav-menu y .nav-item: Lista de navegación interactiva, con estilos para hover y activo.

5. Contenido principal

- .main-content: Área donde se cargan dinámicamente las secciones.
- .header: Encabezado superior con título y usuario.

- .user-avatar: Círculo con iniciales o ícono del usuario conectado.

6. Tarjetas y estadísticas

```
.card {
  background: white;
  border-radius: 10px;
  box-shadow: 0 4px 6px rgba(0,0,0,0.1);
  padding: 1.5rem;
  margin-bottom: 2rem;
}
```

Componentes visuales para mostrar datos (como KPIs, totales o info relevante). .card-stats usa degradados e interactividad (hover con transform).

7. Formularios

- .form-group: Contenedor de campos individuales.
- .form-control: Inputs estilizados con bordes, padding y focus visual.
- .btn, .btn-primary: Botones personalizados, con hover dinámico.

8. Tablas

Estructura estándar:

- .table, .table th, .table td: Estilo limpio y responsivo.
- .badge, .badge-success, .badge-warning: Pequeños indicadores visuales (estatus).

9. Secciones dinámicas

```
.content-section {
  display: none;
```

```
}

.content-section.active {

  display: block;

  animation: fadeIn 0.5s;

}
```

Permite cargar módulos específicos al hacer clic, sin recargar la página. Usa animación fadeIn.

10. Misiones y Visión

Diseño en dos columnas con tarjetas visuales:

- .mission-vision-grid, .mission-card, .vision-card: Uso de grid y estilos diferenciales por sección.

11. Testimonios (Carrusel)

Sistema simple para cambiar testimonios:

- .testimonial: Oculto por defecto, se activa con .active.
- .testimonial-btn: Botones laterales con navegación (prev/next).

12. Switch (Interruptor de modo oscuro)

```
nput:checked + .slider {

  background-color: var(--primary);

}

input:checked + .slider:before {

  transform: translateX(26px);

}
```

Diseño moderno tipo iOS para un interruptor de tema claro/oscuro. La palanca cambia color y posición con transform.

Cómo instalar y correr la app

Para poder utilizar la aplicación, es necesario contar con los siguientes requisitos:

- Tener la base de datos configurada en Oracle.
- Instalar Node.js en tu sistema.
- Instalar las siguientes dependencias de Node.js:
- Express
- body-parser
- cors
- oracledb
- nodemon (solo para desarrollo)

Se instalan con los siguientes comandos:

npm install express body-parser cors oracledb

npm install --save-dev nodemon

Cómo ejecutar la aplicación

- Abre una terminal (CMD).
- Navega a la carpeta del proyecto: petcare-system.
- Ejecuta el siguiente comando para iniciar el servidor:

  node server/server.js

- Abre tu navegador y accede a la URL

  http://localhost:3000

Carga del Script SQL en Oracle

Requisitos previos

Tener instalado Oracle Database (en nuestro caso, Oracle 21c XE). Y a Oracle SQL Developer.

Pasos para cargar el script

Abrir Oracle SQL Developer, hacer una nueva conexión con el nombre de PetCare y la contraseña petcare123, ejecutar el script petcare.sql por partes una tabla a la vez

Script

```
CREATE TABLE Dueno (
        id_dueno NUMBER PRIMARY KEY,
        nombre VARCHAR2(100),
        telefono VARCHAR2(20),
        correo VARCHAR2(100)
);
CREATE SEQUENCE dueno_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;
CREATE OR REPLACE TRIGGER dueno_trigger
BEFORE INSERT ON Dueno
FOR EACH ROW
BEGIN
        SELECT dueno_seq.NEXTVAL
```

```
        INTO :NEW.id_dueno
        FROM dual;
END;
/
CREATE TABLE Mascota (
        id_mascota NUMBER PRIMARY KEY,
        nombre VARCHAR2(100),
        especie VARCHAR2(50),
        raza VARCHAR2(50),
        edad NUMBER,
        id_dueño NUMBER,
        FOREIGN KEY (id_dueño) REFERENCES Dueño(id_dueño)
);
CREATE SEQUENCE mascota_seq START WITH 1 INCREMENT BY 1;
CREATE OR REPLACE TRIGGER mascota_trigger
BEFORE INSERT ON Mascota
FOR EACH ROW
BEGIN
        SELECT mascota_seq.NEXTVAL
        INTO :NEW.id_mascota
        FROM dual;
END;
/
CREATE TABLE Veterinario (
        id_veterinario NUMBER PRIMARY KEY,
```

```
    nombre VARCHAR2(100),
    especialidad VARCHAR2(100)
);
CREATE SEQUENCE veterinario_seq START WITH 1 INCREMENT BY 1;
CREATE OR REPLACE TRIGGER veterinario_trigger
BEFORE INSERT ON Veterinario
FOR EACH ROW
BEGIN
    SELECT veterinario_seq.NEXTVAL
    INTO :NEW.id_veterinario
    FROM dual;
END;
/
CREATE TABLE Cita (
    id_cita NUMBER PRIMARY KEY,
    id_mascota NUMBER,
  id_veterinario NUMBER,
    fecha DATE,
    hora VARCHAR2(10),
    motivo VARCHAR2(200),
    FOREIGN KEY (id_mascota) REFERENCES Mascota(id_mascota),
    FOREIGN KEY (id_veterinario) REFERENCES Veterinario(id_veterinario)
);
CREATE SEQUENCE cita_seq START WITH 1 INCREMENT BY 1;
CREATE OR REPLACE TRIGGER cita_trigger
```

```
BEFORE INSERT ON Cita
FOR EACH ROW
BEGIN
    SELECT cita_seq.NEXTVAL
    INTO :NEW.id_cita
    FROM dual;
END;
/
CREATE TABLE Consulta (
    id_consulta NUMBER PRIMARY KEY,
    id_mascota NUMBER,
    fecha DATE,
    diagnostico VARCHAR2(200),
    observaciones VARCHAR2(300),
    FOREIGN KEY (id_mascota) REFERENCES Mascota(id_mascota)
);
CREATE SEQUENCE consulta_seq START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE TRIGGER consulta_trigger
BEFORE INSERT ON Consulta
FOR EACH ROW
BEGIN
    SELECT consulta_seq.NEXTVAL
    INTO :NEW.id_consulta
    FROM dual;
```

```
END;
/
CREATE TABLE Tratamiento (
        id_tratamiento NUMBER PRIMARY KEY,
        id_consulta NUMBER,
        medicamento VARCHAR2(100),
        dosis VARCHAR2(100),
        duracion VARCHAR2(50),
        FOREIGN KEY (id_consulta) REFERENCES Consulta(id_consulta)
);
CREATE SEQUENCE tratamiento_seq START WITH 1 INCREMENT BY 1;
CREATE OR REPLACE TRIGGER tratamiento_trigger
BEFORE INSERT ON Tratamiento
FOR EACH ROW
BEGIN
        SELECT tratamiento_seq.NEXTVAL
        INTO :NEW.id_tratamiento
        FROM dual;
END;
/
```

Resumen de las tablas y relaciones de tu base de datos:

1. Dueno

id_dueno (PK)

nombre

telefono

correo

2. Mascota

id_mascota (PK)

nombre

especie

raza

edad

id_dueño (FK → Dueno)

3. Veterinario

id_veterinario (PK)

nombre

especialidad

4. Cita

id_cita (PK)

id_mascota (FK → Mascota)

id_veterinario (FK → Veterinario)

fecha

hora

motivo

5. Consulta

id_consulta (PK)

id_mascota (FK → Mascota)

fecha

diagnostico

observaciones

6. Tratamiento

id_tratamiento (PK)

id_consulta (FK → Consulta)

medicamento

dosis

duracion

Esquema ERDPlus