

# MANUAL TÉCNICO

## Avícola El Manantial – Sistema de Gestión

Pereira, Risaralda

Universidad Tecnológica de Pereira

Juan Pablo Sánchez Zapata

### Contenido

OBJETIVOS .....	3
Objetivo general .....	3
Objetivos específicos .....	3
INTRODUCCIÓN .....	4
1. DESCRIPCIÓN DE ELEMENTOS. ....	5
1.1. Elementos físicos. ....	5
1.1.1. Servidor Local:.....	5
1.1.2. Estaciones de Trabajo:.....	5
1.1.3. Red Local (LAN): .....	5
1.1.4. Infraestructura de Respaldo: .....	5
1.2. Elementos lógicos. ....	6
1.2.1. Frameworks y herramientas utilizadas.....	6
1.2.2. Scripts para la base de datos .....	6
1.2.3. Estructura del proyecto.....	7
2. DESPLIEGUE DE LA SOLUCIÓN .....	11
2.1. Configuración del entorno .....	11
2.2. Creación de la base de datos.....	12
2.3. Configuración del backend.....	13
2.3.1. Ejecución del backend.....	15
2.4. Configuración del frontend.....	16
2.5. Flujo de inicialización.....	20
3. SCRIPTS PARA LA GESTIÓN DE LA BASE DE DATOS .....	21
3.1. Creación de la base de datos.....	21
3.2. Creación de las tablas .....	21
3.3. Inserción de datos.....	23
3.4. Procedimientos .....	28
3.4.1. Gestión de inventario.....	28
3.4.2. Gestión de ventas .....	30

3.4.3.	Gestión de proveedores y pedidos.....	31
3.4.4.	Gestión de compras.....	32
3.4.5.	Gestión de pagos .....	33
3.5.	Vistas y disparadores.....	34
4.	ANEXOS.....	37

# OBJETIVOS

## Objetivo general

- Crear un aplicativo web de fácil uso que permita a la empresa “Avícola El Manantial” automatizar y optimizar el manejo de su inventario, el registro de ventas, la relación con los proveedores y pedidos, y el control de pagos.

## Objetivos específicos

- Garantizar que la información esté correctamente almacenada y disponible.
- Facilitar el acceso al inventario de productos.
- Registrar y actualizar automáticamente el stock de productos.
- Garantizar el manejo de funcionalidades exclusivas solo al administrador para salvaguardar la integridad de los datos.
- Registrar ventas y compras actualizando automática el stock de productos.
- Diseñar una interfaz sencilla, clara e intuitiva que facilite su uso a cualquier usuario final.

# INTRODUCCIÓN

El presente documento constituye el **Manual Técnico** del proyecto "*Sistema de Gestión Avícola El Manantial*", una solución diseñada para optimizar y automatizar los procesos operativos de una empresa avícola dedicada a la comercialización de huevos y productos de la canasta familiar. Este sistema integra herramientas modernas de desarrollo web, bases de datos relacionales y técnicas de programación para garantizar una gestión eficiente de inventarios, ventas, pedidos, pagos y control de stock.

El objetivo principal del manual es proporcionar una guía detallada que permita comprender la infraestructura física y lógica del sistema, así como el paso a paso necesario para su despliegue, mantenimiento y uso correcto. Además, se describe la estructura y funcionalidad de los scripts utilizados para la gestión de la base de datos, incluyendo la creación de tablas, procedimientos almacenados y triggers.

Este manual está dirigido a administradores del sistema, desarrolladores o cualquier personal técnico que requiera implementar, personalizar o mantener la solución. Se espera que, al seguir las instrucciones contenidas en este documento, se facilite la correcta instalación y configuración del sistema, garantizando su óptimo funcionamiento.

# 1. DESCRIPCIÓN DE ELEMENTOS.

A continuación, se presentan los elementos físicos y lógicos utilizados para la construcción de este sistema y su estructura como tal.

## 1.1. Elementos físicos.

El sistema de gestión "*Avícola El Manantial*" ha sido diseñado para funcionar sobre una infraestructura de hardware básica y accesible, que garantiza el correcto desempeño de las operaciones. Los elementos físicos implementados en el proyecto son los siguientes:

### 1.1.1. Servidor Local:

El servidor local es la pieza central de la infraestructura física, configurado para ejecutar el backend del sistema y gestionar la base de datos.

Es un equipo de cómputo con las siguientes especificaciones mínimas:

- a. **Procesador:** Intel Core i5 o equivalente.
- b. **Memoria RAM:** 8 GB.
- c. **Almacenamiento:** 256 GB SSD (o HDD con suficiente capacidad para los datos generados).
- d. **Sistema Operativo:** Windows 11.

### 1.1.2. Estaciones de Trabajo:

Los usuarios, tanto administradores como empleados, interactúan con el sistema a través de navegadores web en equipos configurados como estaciones de trabajo.

Requisitos mínimos:

- a. **Navegador Web:** Google Chrome, Mozilla Firefox o Microsoft Edge (versión actualizada).
- b. **Resolución de Pantalla:** 1280x720 o superior.
- c. **Conexión a Red Local:** Conectividad al servidor mediante LAN o Wi-Fi estable.

### 1.1.3. Red Local (LAN):

La red local garantiza la comunicación eficiente entre las estaciones de trabajo y el servidor. Se utilizó una red Ethernet o Wi-Fi segura para la interacción entre los dispositivos.

### 1.1.4. Infraestructura de Respaldo:

Se sugiere la implementación de dispositivos de almacenamiento externo (discos duros o servidores NAS) para realizar copias de seguridad de la base de datos de forma periódica, garantizando la protección de los datos críticos.

## 1.2. Elementos lógicos.

Las herramientas utilizadas para el despliegue de esta aplicación son conocidas y sencillas. En este apartado solo se explicarán cuáles fueron las herramientas usadas y los elementos de la base de datos como también la estructura del proyecto.

### 1.2.1. Frameworks y herramientas utilizadas

En este apartado se presentan las herramientas utilizadas tanto para el backend, como para el frontend y la estructura de la base de datos.

- a. **Backend:** Node.js con Express
- b. **Base de datos:** MySQL, phpMyAdmin
- c. **Servidor:** XAMPP, Postman
- d. **Frontend:** HTML5, CSS3, JavaScript y Font Awesome
- e. **Dependencias:** MySQL2 y cors para realizar los endpoints correctamente.

### 1.2.2. Scripts para la base de datos

Se manejan en total 12 tablas y distintos procedimientos. Cabe recalcar que para esta versión del proyecto no todos los procedimientos se tuvieron en cuenta al igual que los disparadores o triggers.

El aplicativo permite el manejo de Proveedores, Productos, Pedidos, Detalles de Pedidos, Compras y Detalles de Compras, Clientes, Usuarios, Ventas y Detalles de Ventas como también manejo de pagos.

**Aclaración:** La tabla faltante, que es la tabla de descuento, no se tuvo en cuenta por el momento para esta versión del proyecto

Cada tabla fue inicializada con valores por defecto para poder comprobar su funcionamiento a la hora de hacer la conexión del backend con la base de datos.

**Procedimientos:** En cuanto a los procedimientos en general se usaron procedimientos para agregar productos, agregar ventas, agregar compras y actualizaciones en la base de datos como actualizar el stock de un producto cuando este sea vendido o cuando se efectúe la compra de un pedido que fue completado.

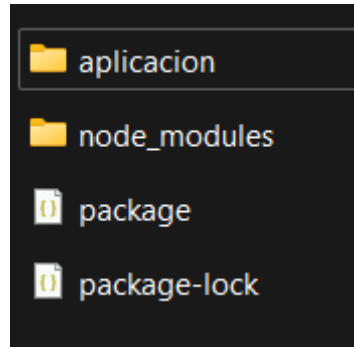
**Triggers:** Para esta versión del proyecto, los disparadores se crearon, sin embargo, no se usaron en el despliegue de la aplicación, en su defecto, las alertas necesarias como por ejemplo alertar por la necesidad de reabastecer un producto o actualizar el stock de un producto se manejan con los endpoints en el servidor con simples consultas.

**Vistas:** Se crearon, efectivamente algunos scripts para las vistas, es decir, tablas que el usuario quisiera ver con facilidad, para esta versión, por restricciones en límite de tiempo, en vez de usar las vistas simplemente las consultas lo hacen posible.

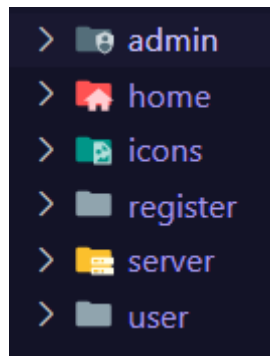
Cada uno de estos scripts los puede encontrar en un archivo con extensión .sql. Tomando todo el documento y montándolo en un motor de bases de datos como phpMyAdmin o un servidor de MariaDB o MySQL, la base de datos se creará sin problemas con cada una de las tablas, procedimientos, triggers y datos.

### 1.2.3. Estructura del proyecto

Todo el proyecto en realidad está contenido sobre una carpeta llamada *aplicación*. Como primera carpeta, esta está acompañada de la carpeta de los paquetes de node.js que fueron instalados con las dependencias.

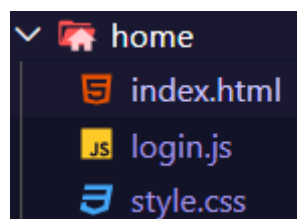


Dentro de la carpeta *aplicación* se manejan diversas funcionalidades así:



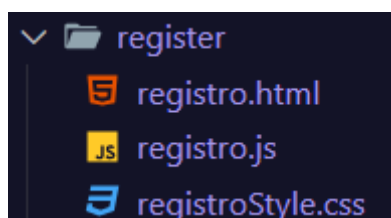
#### Carpeta /home

La carpeta *home*, es el punto de entrada de la aplicación. Sin esta carpeta no hay a donde llegar, de manera que, al ser simplemente un inicio de sesión, lo primero que puede hacer la aplicación puede llevar según el rol, al HTML del administrador, o al vendedor según cada caso. También, a través de */home* se podría llegar al formulario de registro.



#### Carpeta /register

En esta carpeta, al igual que en casi todas, se tiene un HTML con el formulario de registro al cual se le da estilo con su respectivo archivo en CSS. Lo que hace *login.js* es ser la parte cliente del proceso cliente-servidor con el servidor creado en la carpeta *server*.



## Carpeta /server

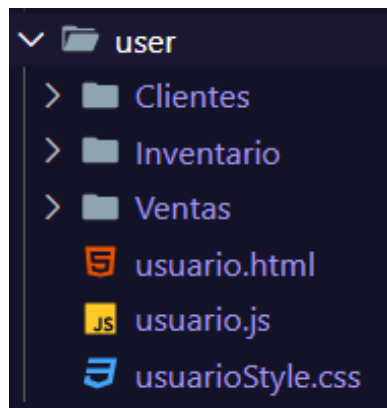
Esta carpeta podría cumplir el papel de ser el cerebro en backend de toda esta aplicación. En la carpeta server tenemos el acceso a la base de datos creada en phpMyAdmin y también el endpoint con node.js y express para realizar las consultas solicitadas en cada parte de esta aplicación.



La aplicación cuenta con dos posibles usuarios. Uno de ellos es el usuario “vendedor” que tiene distintos permisos al usuario “administrador”. Para manejar cada uno de ellos se tienen carpetas distintas:

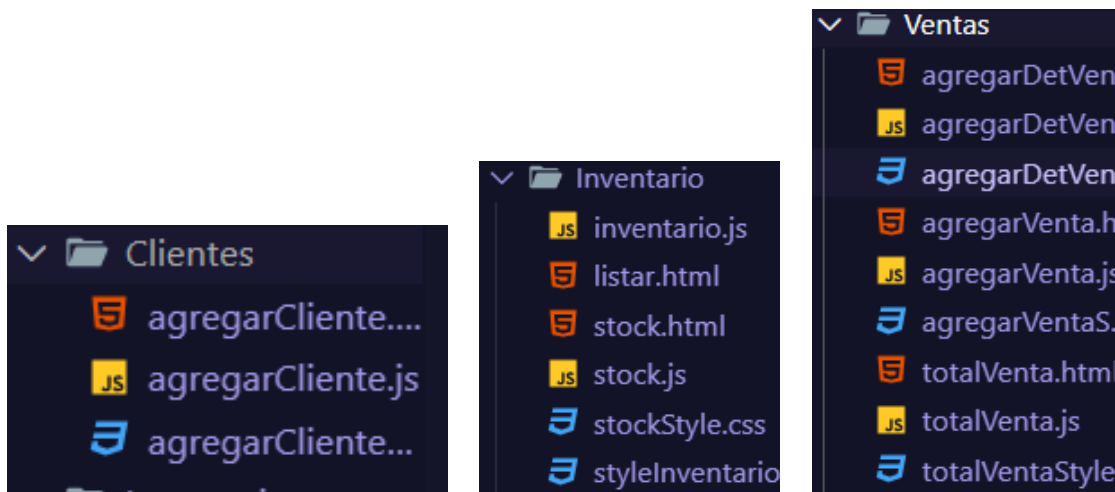
## Carpeta /user

La carpeta /user contiene un archivo HTML con su respectivo CSS y archivo de cliente-servidor para linkear algunos botones de diferentes funcionalidades que se permiten para un usuario que en este caso “user” es para representar el usuario con rol “vendedor”. Se tiene control sobre clientes en la parte de agregar un cliente, de igual manera para el inventario, se puede consultar el stock de un producto como también se puede ver el inventario de productos. El usuario vendedor debe poder registrar una venta y un detalle de venta conociendo también el total de cada venta que realice.



Dentro de cada carpeta se encuentran de manera respectiva los tres archivos esenciales para poder cumplir con el funcionamiento y el llamado al endpoint que se tiene en el archivo server.js.

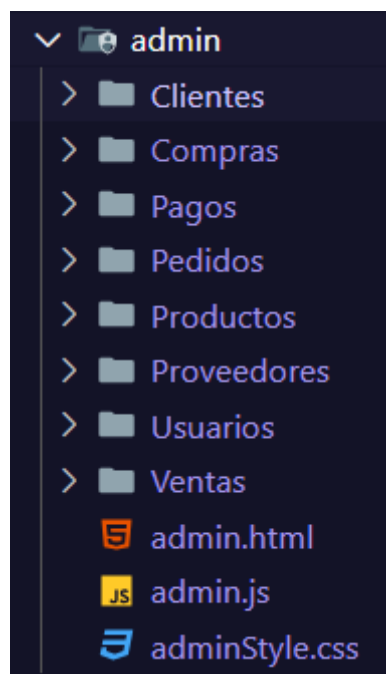




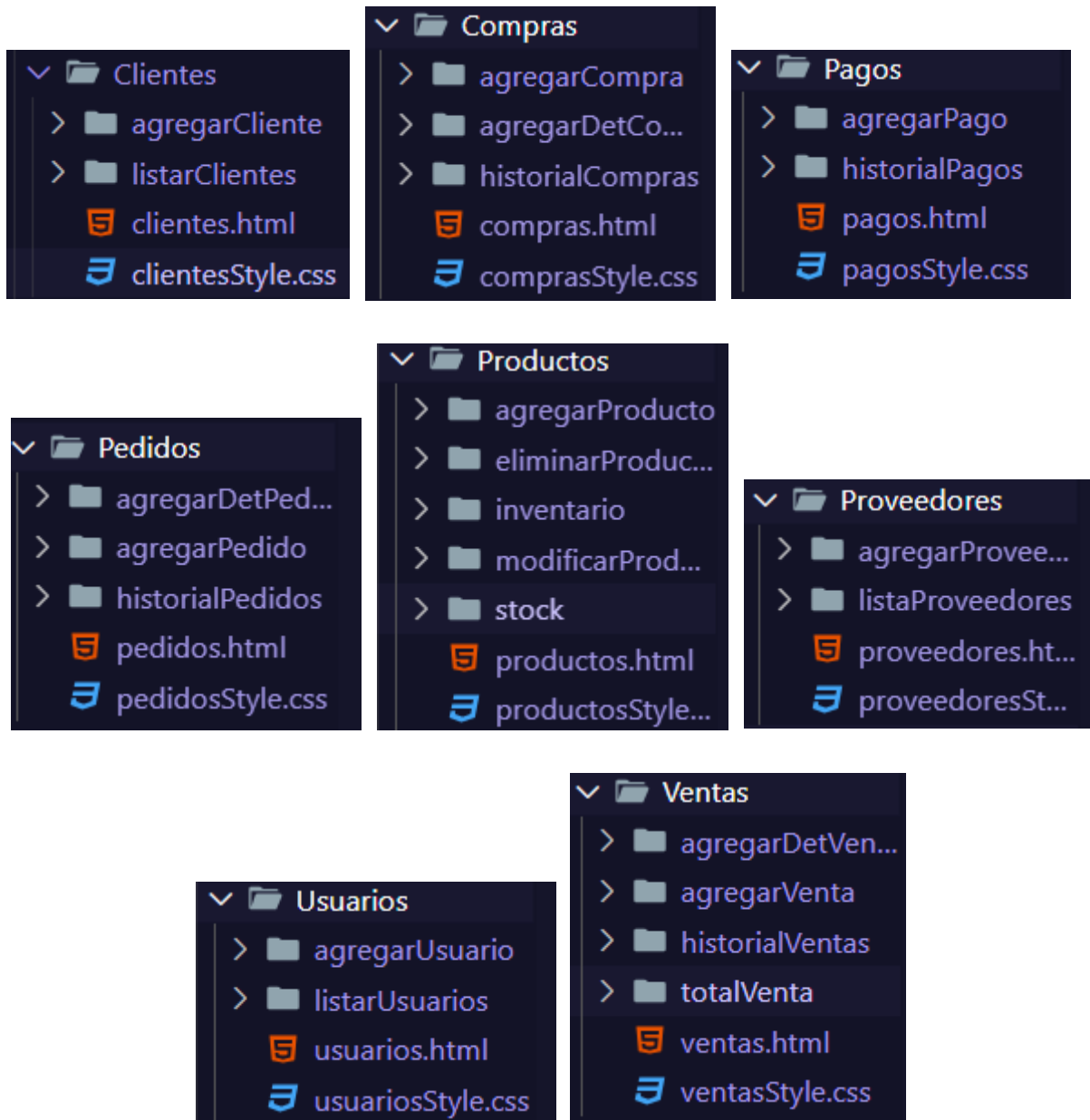
Nótese que cada HTML tiene su respectivo código de estilo y código de cliente-servidor. Al ser más pocas funcionalidades no es necesario crear subcarpetas para gestionar más funcionalidades, lo cual es el caso contrario de la carpeta admin.

### Carpeta /admin

Claro está que el usuario administrador tiene más privilegios en la aplicación, como lo es tener control, sobre todo, control sobre los productos, sobre las ventas, compras, pagos, proveedores, pedidos y demás. De esta manera, el control sobre cada una de estas entidades definidas como tablas en la base de datos tiene también diferentes funcionalidades. Así que la estructura principal de la carpeta /admin es:



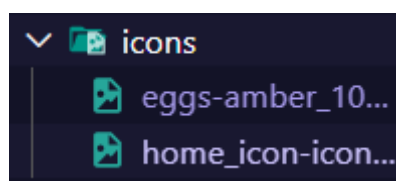
Como se evidencia, el administrador tiene más control sobre diferentes entidades que el usuario que solo es vendedor. En este caso cada carpeta (Clientes, Compras, Pagos, etc.) contiene dentro de sí algunas carpetas más ya que son varias las funcionalidades que se tienen con cada entidad, mejor se agrupan para que sea más fácil y ordenado el acceso:



Dentro de cada carpeta para controlar una entidad de la base de datos, se encuentran las carpetas respectivas para cada acción que se tiene. Cabe aclarar que cada archivo .js para cada funcionalidad debe estar conectado a través de un método (get o post) con los endpoint creados en el servidor.

### Carpeta /icons

Esta carpeta es sencilla ya que solo contiene los .ico de iconos para poner a la pestaña de la aplicación y un símbolo de casa para poner en páginas que están mas adelantadas de la principal.



## 2. DESPLIEGUE DE LA SOLUCIÓN

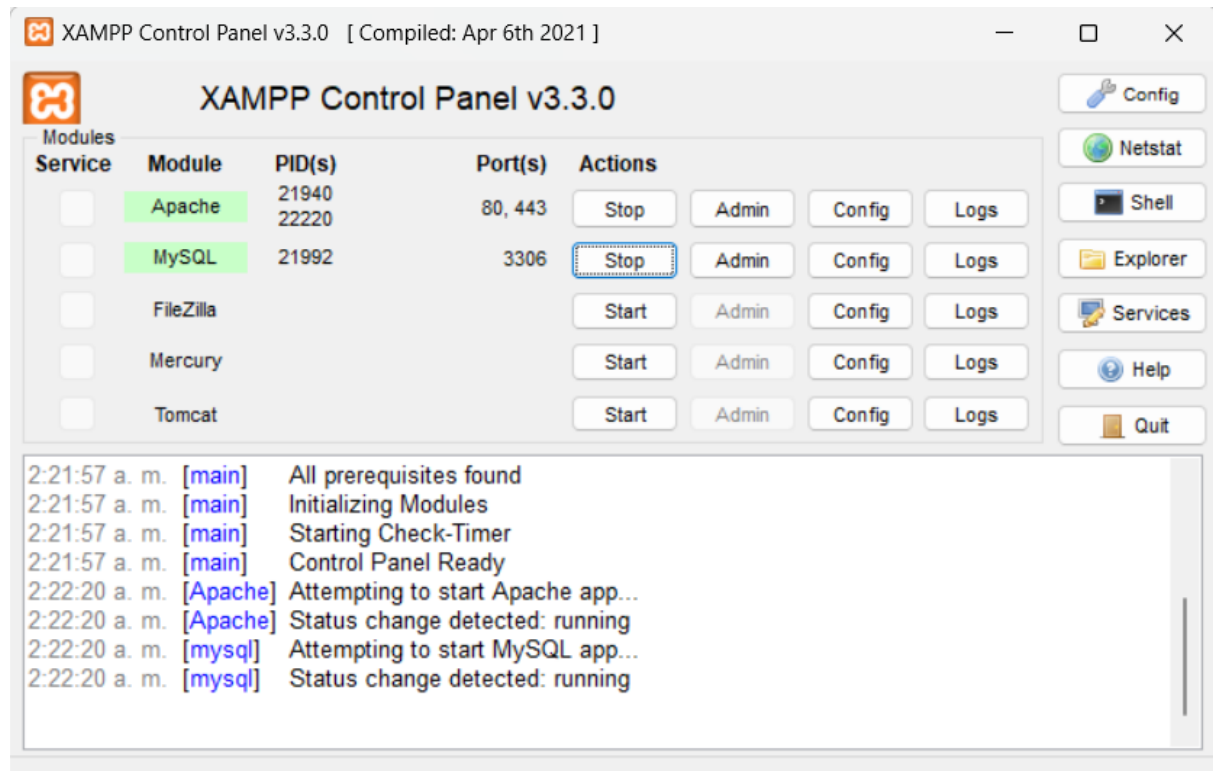
A través de este apartado se busca plantear de manera técnica la solución al problema planteado asumiendo la propuesta escrita en el documento de los requerimientos del proyecto:

*“Se propone desarrollar un sistema web de gestión de inventario y control de ventas basado en una base de datos relacional que permita a la empresa automatizar y optimizar el manejo de su inventario, el registro de ventas, la relación con los proveedores y pedidos y el control de pagos. Con esto, se garantiza que la información esté correctamente almacenada y disponible. Gracias a que se podrán registrar todas las ventas (sin importar el método de pago), habrá un seguimiento detallado de las transacciones diarias y Avícola El Manantial podrá tomar decisiones más informadas sobre su inventario, mejorar la relación con sus proveedores y controlar mejor sus flujos de efectivo, asegurando así un mayor control sobre las utilidades.” (Propuesta de solución)*

### 2.1. Configuración del entorno

#### INSTALACIÓN DE XAMPP

Primero, es necesario contar con un servidor local con el cual acceder a la base de datos. Para esto simplemente instale XAMPP según el sistema operativo de su máquina. (Fuente: <https://www.apachefriends.org/es/index.html>)



Asegúrese de iniciar el módulo de Apache y en este caso de MySQL. Esto lo hacemos con el fin de proceder a la creación de la base de datos con el motor de bases de datos: phpMyAdmin. Luego de iniciar los módulos, el botón de Admin de la línea de MySQL haga click para ser redirigido a phpMyAdmin.

**Nota:** Si llega a olvidar iniciar uno de los dos módulos, jamás entrara a phpMyAdmin.

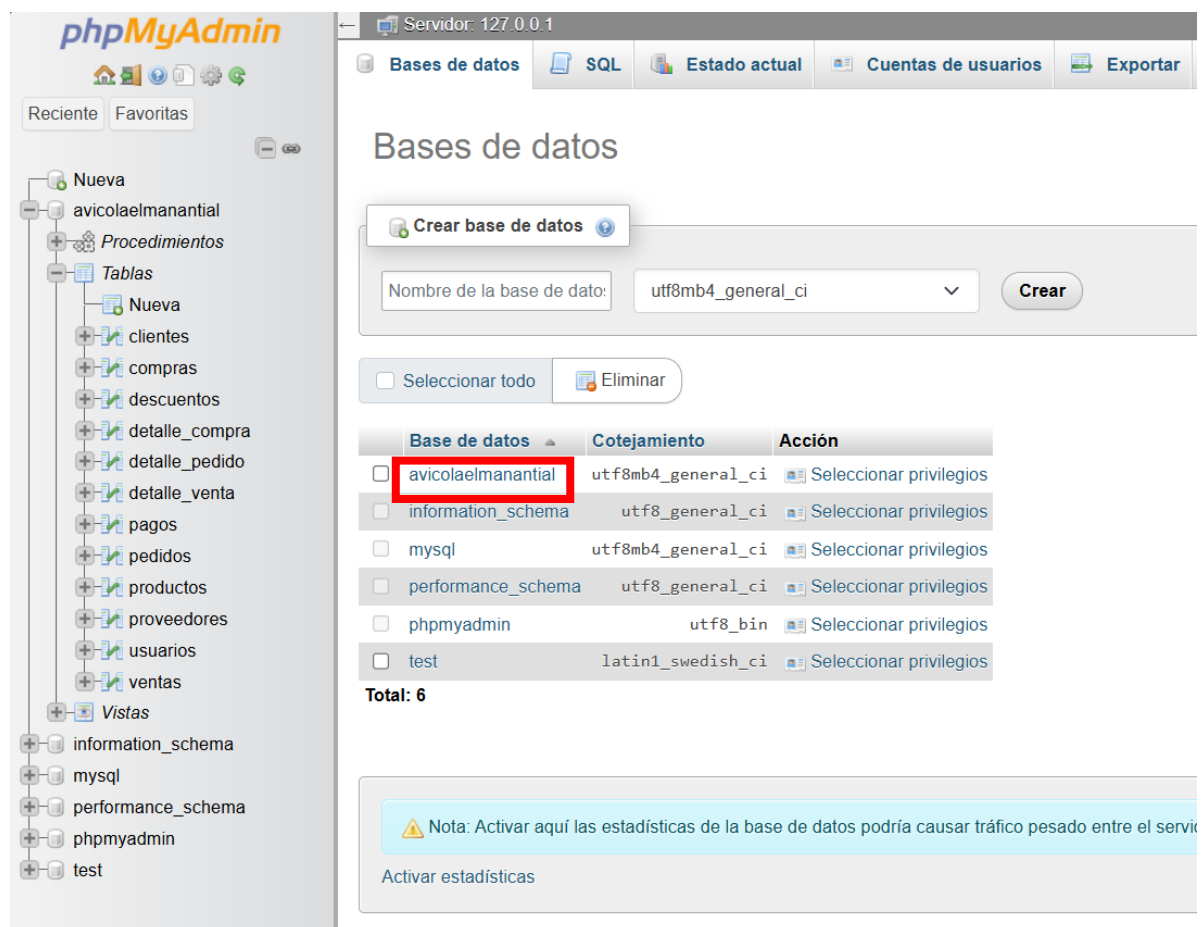
## 2.2. Creación de la base de datos

Al abrir phpMyAdmin, se puede proceder con la creación de la base de datos, sus tablas, la inserción de datos y los respectivos procedimientos y triggers. Aunque el motor permite crear de manera manual e interactiva la base de datos, en los scripts está todo el proceso con las consultas para crear la base de datos que en este caso se llamará “*avicolaelmanantial*”

```
-- Consulta para crear la base de datos en PHPMyAdmin
CREATE DATABASE IF NOT EXISTS AvicolaElManantial;

USE AvicolaElManantial;
```

Luego, ejecute todos los scripts, pero sin los disparadores. Para esta versión del proyecto, los disparadores aún no se manejan, sino que las alertas que se deben generar y actualizaciones automáticas se hacen con condiciones que, al cumplirse, ejecutan procesos y consultas. Básicamente, su phpMyAdmin debería verse así, (si no tiene más bases de datos creadas):



The screenshot displays the phpMyAdmin web interface. On the left, a sidebar shows a tree view of the database structure, with 'avicolaelmanantial' selected under the 'Nueva' (New) section. The main content area is titled 'Bases de datos' (Databases) and shows a list of existing databases. The 'avicolaelmanantial' database is highlighted with a red rectangular box. Below the list, there is a 'Total: 6' indicator and a note about activating statistics.

Base de datos	Cotejamiento	Acción
<input type="checkbox"/> avicolaelmanantial	utf8mb4_general_ci	<a href="#">Seleccionar privilegios</a>
<input type="checkbox"/> information_schema	utf8_general_ci	<a href="#">Seleccionar privilegios</a>
<input type="checkbox"/> mysql	utf8mb4_general_ci	<a href="#">Seleccionar privilegios</a>
<input type="checkbox"/> performance_schema	utf8_general_ci	<a href="#">Seleccionar privilegios</a>
<input type="checkbox"/> phpmyadmin	utf8_bin	<a href="#">Seleccionar privilegios</a>
<input type="checkbox"/> test	latin1_swedish_ci	<a href="#">Seleccionar privilegios</a>

Total: 6

Nota: Activar aquí las estadísticas de la base de datos podría causar tráfico pesado entre el servicio.

[Activar estadísticas](#)

Ya se ha creado la base de datos.

## 2.3. Configuración del backend

Para comenzar a realizar el código para el frontend y el backend es necesario comenzar a configurar el entorno con las herramientas que se usarán para acceder a la base de datos. Dado que para este proyecto se trabajó con JavaScript, es necesario tener un framework que sea un endpoint con la base de datos. Así que lo primero es instalar node.js.

(Fuente: <https://nodejs.org/en/>)

Para instalar el framework anterior solo es necesario abrir el ejecutable y dar next.

### Verificar que esté instalado Node JS

Debe abrir el terminal de su sistema y ejecutar el comando `node -v`

```
C:\Users\usuario>node -v
v22.11.0
```

Luego de verificar que node fue instalado correctamente, para poder ejecutar el proyecto necesita instalar los módulos y las dependencias necesarias. Para esto, abra su terminal y vaya a la dirección de la carpeta del proyecto, es necesario que lo haga en su terminal.

Escriba el comando: 'npm init -y'

```
C:\Users\usuario\Documents\UTP-Juan\SEMESTRE VI\DB\Prueba>npm init -y|
```

Esto generará un json al cual se deben agregar las dependencias:

```
C:\Users\usuario\Documents\UTP-Juan\SEMESTRE VI\DB\Prueba>npm init -y
Wrote to C:\Users\usuario\Documents\UTP-Juan\SEMESTRE VI\DB\Prueba\package.json:

{
  "name": "prueba",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

Se procede a instalar las librerías necesarias que, en este caso para empezar, sería mysql2:  
'npm install mysql2'

```
C:\Users\usuario\Documents\UTP-Juan\SEMESTRE VI\DB\Prueba>npm install mysql2

added 13 packages, and audited 14 packages in 2s

1 package is looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Posterior a esto, se instala express: 'npm install mysql2'

```
C:\Users\usuario\Documents\UTP-Juan\SEMESTRE VI\DB\Prueba>npm install express

added 65 packages, and audited 79 packages in 3s

14 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Y finalmente, se instala Cors para no tener errores a la hora de conectar la base de datos por culpa de medidas de seguridad del navegador: 'npm install cors'

```
C:\Users\usuario\Documents\UTP-Juan\SEMESTRE VI\DB\Prueba>npm install cors

added 2 packages, and audited 81 packages in 2s

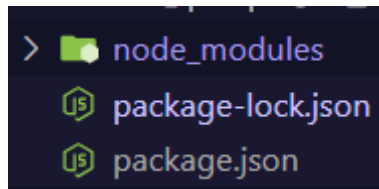
14 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

El paquete de json quedaría con las dependencias así:

```
{
  "name": "prueba",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "cors": "^2.8.5",
    "express": "^4.21.1",
    "mysql2": "^3.11.4"
  }
}
```

Se crea una carpeta que guarda las instalaciones y los paquetes de json con las dependencias:



### 2.3.1. Ejecución del backend

Luego de tener instalado node con sus respectivas dependencias, es necesario crear los archivos que permitirán a la aplicación conectarse con la base de datos. Para este caso serían dos archivos. Uno (database.js) es el encargado de conectarse con la base de datos que tenemos en phpMyAdmin. Y el otro (server.js) es el que tiene la función de ejecutar consultas SQL a través de node a la base de datos y hacer get o post escuchando en un puerto específico.

#### Conexión con la base de datos

```
const mysql = require('mysql2');

const connection =
mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: '',
  database: 'avicolaelmanantial'
});

connection.connect((err) => {
  if (err) {
    console.log('Error connecting
to DB', err);
    return;
  }
  console.log('Connected to DB');
})

module.exports = connection;
```

El usuario es root, y no hay una contraseña ya que el servidor con el que estamos trabajando es con XAMPP.

Posteriormente, el archivo server será el más extenso, ya que contiene los endpoints para cada función que la aplicación tendrá. Se usa get para obtener información de la base de datos sin alterar nada, y post para hacer modificaciones o agregar datos.

Lo principal que debe ir en el archivo server es:

```
const express = require('express');
const cors = require('cors');
const connection =
require('./database');

const app = express();
const port = 4000;

app.use(cors());
app.use(express.json());
app.use(express.urlencoded({ extended:
true }));
```

Con esto, a la hora de querer iniciar la aplicación, siempre será necesario ejecutar el servidor. Para esto, dentro de la carpeta /server del proyecto, ejecute el comando: 'node server.js'

```
C:\Users\usuario\Documents\UTP-Juan\SEMESTRE VI\DB\ProyectoFinal\despliegue\aplicacion\server>node server.js
Server running on port 4000
Connected to DB
|
```

La conexión con la base de datos es exitosa y está escuchando en el puerto 4000 los métodos post o get que se hagan.

## 2.4. Configuración del frontend

Como ya se ha visto anteriormente, se tiene una jerarquía de carpetas, y en cada carpeta, por lo menos hay un archivo HTML y un archivo de estilo CSS. Para esta interfaz muy sencilla, se reutilizó código en muchas funciones, ya que muchas funciones eran formularios y otras simplemente tablas.



```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="shortcut icon" href="../../icons/eggs-amber_109464.ico">
  <link rel="stylesheet" type="text/css" href="registroStyle.css">
  <title>Registro</title>
</head>
<body>

  <div class="header">
    <h1>Registro de usuario</h1>
  </div>

  <button class="btn-volver" onclick="location.href='../home/index.html'">Volver</button>

  <div class="formulario">
    <h1>Registro de usuario</h1>
    <form id="registro-form" method="post">
      <div class="username">
        <input type="text" id="dni" name="dni" required>
        <label>DNI de usuario</label>
      </div>
      <div class="username">
        <input type="text" id="primer-nombre" name="primer-nombre" required>
        <label>Primer nombre</label>
      </div>
      <div class="username">
        <input type="text" id="segundo-nombre" name="segundo-nombre" required>
        <label>Segundo nombre</label>
      </div>
      <div class="username">
        <input type="text" id="primer-apellido" name="primer-apellido" required>
        <label>Primer apellido</label>
      </div>
      <div class="username">
        <input type="text" id="segundo-apellido" name="segundo-apellido" required>
        <label>Segundo apellido</label>
      </div>
      <div class="username">
        <input type="text" id="telefono" name="telefono" required>
        <label>Telefono</label>
      </div>
      <div class="username">
        <input type="password" id="password" name="password" required>
        <label>Contraseña</label>
      </div>
      <input type="submit" value="Registrar">
    </form>
  </div>

  <script src="registro.js"></script>
</body>
</html>

```

El anterior es un ejemplo de un formulario, este caso, el formulario de registro. El estilo que se le da a este formulario es muy similar en cada HTML que tiene un formulario.

```

body {
  margin: 0;
  padding: 0;
  font-family: Arial, sans-serif;
  background: linear-gradient(150deg, #f3f192, #f37506);
  height: 140vh;
}

.header {
  position: absolute;
  top: 10%;
  left: 50%;
  transform: translate(-50%, 0);
  width: 400px;
  padding: 10px 0;
  background: #f37506;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
  margin-left: auto;
  border-radius: 15px;
}

.header h1 {
  color: #fff;
  text-align: center;
}

```

```

.fa-solid{
  color: #fb6000;
  position: absolute;
  top: 0px;
  right: 100px;
  font-size: 40px;
  cursor: pointer;
}

.fa-solid:hover{
  color: #fc1808;
}

.btn-volver {
  position: absolute;
  top: 10px;
  right: 10px;
  background: #ed8701;
  border: 1px solid;
  font-size: 20px;
  color: #ffffff;
  cursor: pointer;
  font-weight: bold;
  border-radius: 8px;
}

.btn-volver:hover {
  color: #f34906;
}

```

```

.formulario{
  position: absolute;
  top: 80%;
  left: 50%;
  transform: translate(-50%, -50%);
  width: 400px;
  background: #fff;
  border-radius: 15px;
}

.formulario h1{
  text-align: center;
  padding: 0 0 20px 0;
  border-bottom: 1px solid silver;
}

.formulario form{
  padding: 0 20px 20px 20px;
}

form .username{
  position: relative;
  border-bottom: 2px solid #adadad;
  margin: 30px 0;
}

```

```

.username input{
  width: 100%;
  padding: 0 5px;
  height: 40px;
  font-size: 16px;
  border: none;
  background: none;
  outline: none;
  position: relative;
}

.username label{
  position: absolute;
  top: 50%;
  left: 5px;
  color: #adadad;
  transform: translateY(-50%);
  font-size: 16px;
  pointer-events: none;
  transition: .5s;
}

.username span::before{
  content: '';
  position: absolute;
  top: 40px;
  left: 0;
  width: 100%;
  height: 2px;
  background: #e1951a;
  transition: .5s;
}

```

```

.username input:focus ~ label,
.username input:valid ~ label{
  top: -5px;
  color: #e1951a;
}

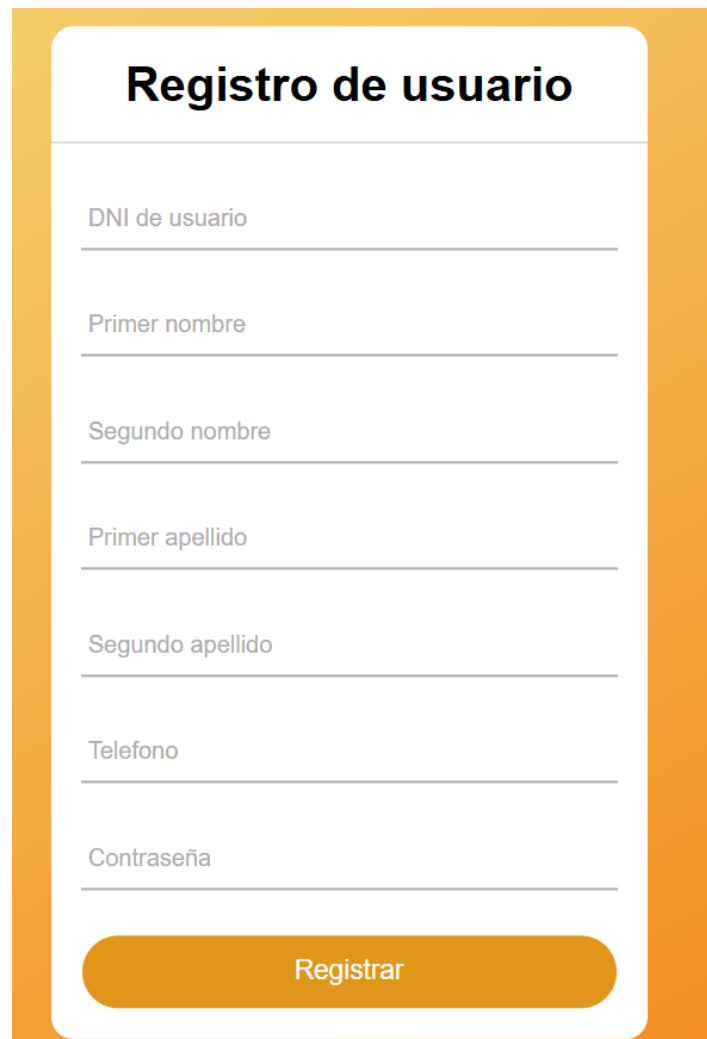
.username input:focus ~ span::before,
.username input:focus ~ span::before{
  width: 100%;
}

input[type="submit"]{
  width: 100%;
  height: 50px;
  background: #e1951a;
  border: 1px solid;
  border-radius: 25px;
  font-size: 18px;
  color: #fff;
  cursor: pointer;
  outline: none;
}

input[type="submit"]:hover{
  border-color: #f37506;
  background: #ed8701;
  color: #fff;
}

```

Para obtener algo así:



The image shows a user registration form titled "Registro de usuario". It is set against a white background with a light orange border. The form contains the following elements:

- Title:** "Registro de usuario" in bold black text.
- Fields:** Seven input fields with light gray placeholder text:
  - DNI de usuario
  - Primer nombre
  - Segundo nombre
  - Primer apellido
  - Segundo apellido
  - Telefono
  - Contraseña
- Button:** A rounded orange button with the text "Registrar" in white.

Notará que, en muchas partes de registro de datos en la aplicación, se usará el mismo diseño.

### Iniciar la aplicación

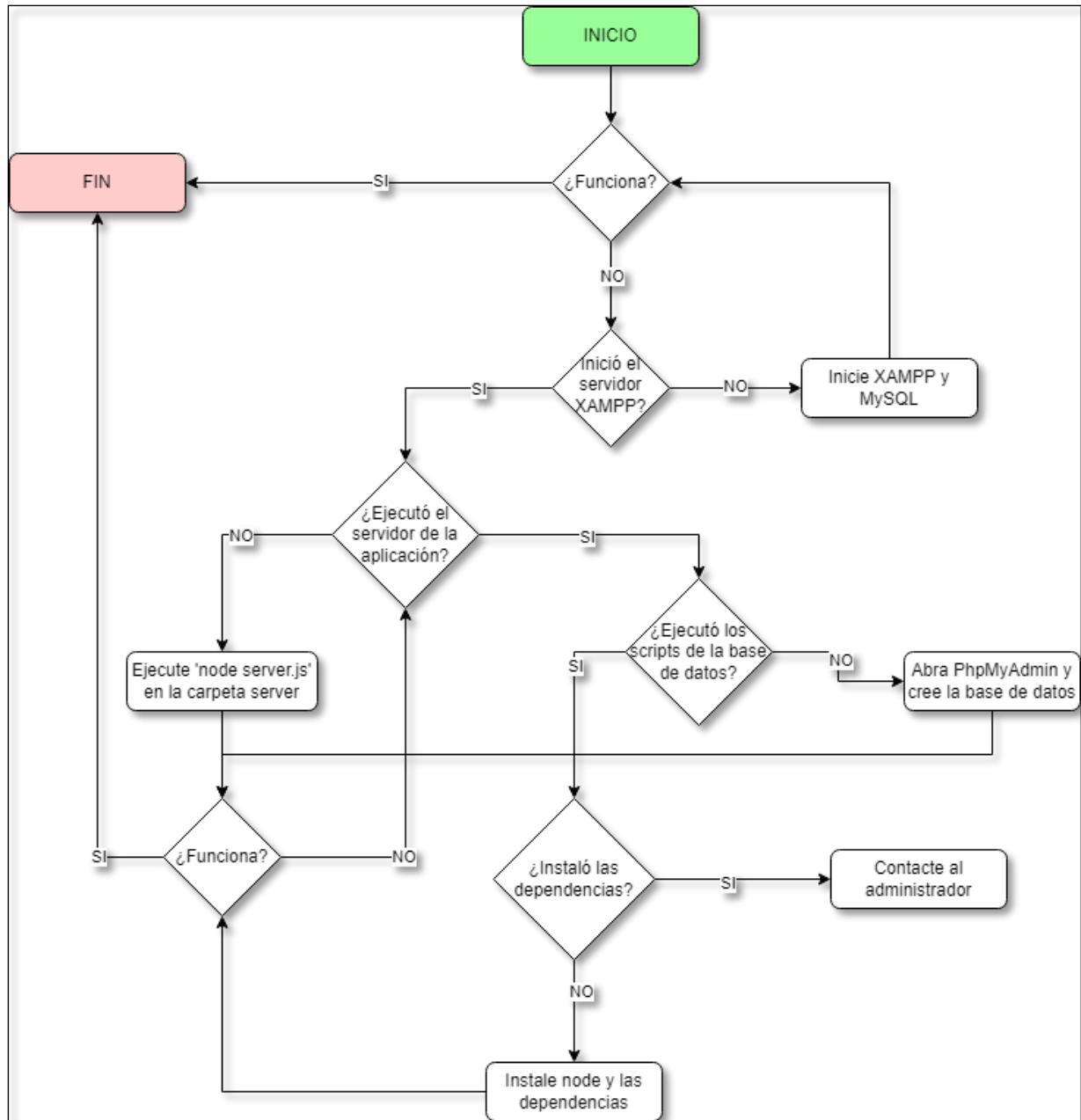
Esta versión del proyecto se presenta en localhost, por lo tanto basta con ir a la carpeta /home del proyecto y abrir el archivo index.html con un navegador, recuerde que para hacer esto debe tener el servidor XAMPP activo y también el servidor server.js con los endpoints para que la conexión con la base de datos sea efectiva.

Si maneja Visual Studio Code, puede abrirlo con el live server (extensión de visual). Así, cualquier cambio que se realice en el frontend se ira modificando en vivo.

**Nota:** Si se realiza algún cambio en el servidor es necesario reiniciarlo para que los cambios se efectúen.

**Así, se inicia la aplicación y puede empezar a rumear cada funcionalidad.**

## 2.5. Flujo de inicialización



## 3. SCRIPTS PARA LA GESTIÓN DE LA BASE DE DATOS

En este apartado se profundiza en los scripts usados para la creación de la base de datos y sus componentes adicionales.

### 3.1. Creación de la base de datos

```
CREATE DATABASE IF NOT EXISTS AvicolaElManantial;  
USE AvicolaElManantial;
```

En esta parte simplemente se crea un base de datos llamada AvicolaElManantial.

### 3.2. Creación de las tablas

Lo que se hace en estos scripts es crear cada una de las tablas necesarias en el proyecto, simplemente se les dan los nombres a las columnas con sus respectivos tipos, las llaves primarias y también foráneas en los casos en que las tablas son referenciadas por otra debido a las relaciones que hay entre ellas. El script de la creación de tablas se presenta a continuación:

*-- Tabla de Proveedores*

```
CREATE TABLE Proveedores (  
    proveedor_id INT AUTO_INCREMENT PRIMARY KEY ,  
    nombre_proveedor VARCHAR(100),  
    telefono VARCHAR(15),  
    direccion VARCHAR(150),  
    productos_proveedor TEXT  
);
```

*-- Tabla de Productos*

```
CREATE TABLE Productos (  
    producto_id INT AUTO_INCREMENT PRIMARY KEY,  
    nombre_producto VARCHAR(100),  
    tipo_producto VARCHAR(50),  
    descripcion TEXT,  
    precio_venta DECIMAL(10, 2),  
    stock_actual INT,  
    stock_minimo INT,  
    unidad_medida VARCHAR(20)  
);
```

*-- Tabla de Pedidos*

```
CREATE TABLE Pedidos (  
    pedido_id INT AUTO_INCREMENT PRIMARY KEY,  
    proveedor_id INT,  
    fecha_pedido DATETIME,  
    estado_pedido VARCHAR(20),  
    FOREIGN KEY (proveedor_id) REFERENCES Proveedores(proveedor_id)  
);
```

*-- Tabla de Detalle\_Pedido*

```
CREATE TABLE Detalle_Pedido (  
    id_detalleP INT AUTO_INCREMENT PRIMARY KEY,  
    producto_id INT,  
    pedido_id INT,  
    cantidad_solicitada INT,
```

```

    cantidad_recibida INT,
    FOREIGN KEY (producto_id) REFERENCES Productos(producto_id),
    FOREIGN KEY (pedido_id) REFERENCES Pedidos(pedido_id)
);

```

*-- Tabla de Compras*

```

CREATE TABLE Compras (
    compra_id INT AUTO_INCREMENT PRIMARY KEY,
    pedido_id INT,
    fecha_compra DATE,
    monto_total DECIMAL(10, 2),
    UNIQUE (pedido_id), -- Para garantizar la relacion 1 a 1
    FOREIGN KEY (pedido_id) REFERENCES Pedidos(pedido_id)
);

```

*-- Tabla de Detalle\_Compra*

```

CREATE TABLE Detalle_Compra (
    id_detalleC INT AUTO_INCREMENT PRIMARY KEY,
    producto_id INT,
    compra_id INT,
    cantidad_comprada INT,
    precio_unitario DECIMAL(10, 2),
    subtotal DECIMAL(10, 2),
    FOREIGN KEY (producto_id) REFERENCES Productos(producto_id),
    FOREIGN KEY (compra_id) REFERENCES Compras(compra_id)
);

```

*-- Tabla de Clientes*

```

CREATE TABLE Clientes (
    DNI_cliente VARCHAR(10) PRIMARY KEY,
    nombre_cliente VARCHAR(100),
    tipo_cliente VARCHAR(20),
    total_compras INT
);

```

*-- Tabla de Descuentos*

```

CREATE TABLE Descuentos (
    descuento_id INT AUTO_INCREMENT PRIMARY KEY,
    DNI_cliente VARCHAR(10),
    porcentaje_dcto DECIMAL(5, 2),
    descripcion VARCHAR(255),
    FOREIGN KEY (DNI_cliente) REFERENCES Clientes(DNI_cliente)
);

```

*-- Tabla de Usuarios*

```

CREATE TABLE Usuarios (
    DNI_usuario VARCHAR(10) PRIMARY KEY,
    primer_nombre VARCHAR(50),
    segundo_nombre VARCHAR(50),
    primer_apellido VARCHAR(50),
    segundo_apellido VARCHAR(50),
    telefono VARCHAR(15),
    rol VARCHAR(20),
    password VARCHAR(255)
);

```

*-- Tabla de Ventas*

```

CREATE TABLE Ventas (
    venta_id INT AUTO_INCREMENT PRIMARY KEY,
    DNI_usuario VARCHAR(10),
    DNI_cliente VARCHAR(10),
    fecha_venta DATETIME,
    total_venta DECIMAL(10, 2),
    FOREIGN KEY (DNI_usuario) REFERENCES Usuarios(DNI_usuario),
    FOREIGN KEY (DNI_cliente) REFERENCES Clientes(DNI_cliente)
);

-- Tabla de Detalle_Venta
CREATE TABLE Detalle_Venta (
    id_detalleV INT AUTO_INCREMENT PRIMARY KEY,
    producto_id INT,
    venta_id INT,
    cantidad_vendida INT,
    precio_unitario DECIMAL(10, 2),
    subtotal DECIMAL(10, 2),
    FOREIGN KEY (producto_id) REFERENCES Productos(producto_id),
    FOREIGN KEY (venta_id) REFERENCES Ventas(venta_id)
);

-- Tabla de Pagos
CREATE TABLE Pagos (
    pago_id INT AUTO_INCREMENT PRIMARY KEY,
    venta_id INT,
    monto_pagado DECIMAL(10, 2),
    fecha_pago DATE,
    metodo_pago VARCHAR(20),
    estado VARCHAR(20),
    saldo_pendiente DECIMAL(10, 2),
    UNIQUE (venta_id), -- Para garantizar la relacion 1 a 1
    FOREIGN KEY (venta_id) REFERENCES Ventas(venta_id)
);

```

### 3.3. Inserción de datos

Es muy importante inicializar la base de datos con algunos valores para cada tabla ya que esto permite que a la hora de probar la aplicación se conozca si la conexión con la base de datos sí está siendo exitosa o no. Así que con los siguientes scripts simplemente se hace inserción de datos a cada una de las tablas creadas en el apartado anterior:

```

-- INSERCIÓN DE DATOS ESTÁTICOS Y DE EJEMPLO PARA PRUEBAS

-- Proveedores
INSERT INTO Proveedores (nombre_proveedor, telefono, direccion,
productos_proveedor) VALUES
('granja villa ines', '3145893245', 'galeria la 40', 'huevos'),
('distribuidora del rio', '3154768534', 'avenida del rio', 'huevos, azucar'),
('distribuidora alvarez', '3125467238', 'calle 10', 'panela, arroz, vasos
desechables'),
('tropper', '3154789654', 'calle 20', 'mirringo, ringo, sal'),
('colanta', '3203451278', 'calle 30', 'leche'),
('alqueria', '3126785498', 'calle 32', 'leche, crema de leche'),
('aceite ideal', '3458761234', 'calle 34', 'aceite, electrolit, atun'),
('escobas y traperos la 9a', '3145872390', 'cra 9', 'escobas, trapeadores,
recogedores'),

```

```
('el globo', '3126785463', 'cra 15', 'lentejas, frijoles, espaguetis,
mantequilla, chocolate, cafe, papel higienico, fruco, candelas, panelada,
maizena, esponjas'),
('el granjero', '3125647389', 'calle 35', 'microfibra, fibra'),
('el confite', '3153785654', 'calle 54', 'bocadillos, jabon');
```

```
-- Productos
```

```
INSERT INTO Productos (nombre_producto, tipo_producto, descripcion, precio_venta,
stock_actual, stock_minimo, unidad_medida) VALUES
('huevo super blanco', 'alimento', 'huevos de gallina por unidad', 700.00, 877,
600, 'unidad'),
('panal de huevo super blanco', 'alimento', 'huevos de gallina por 30 unidades',
18000.00, 29, 20, 'panal'),
('huevo aa blanco', 'alimento', 'huevos de gallina por unidad', 600.00, 3302,
600, 'unidad'),
('panal de huevo aa blanco', 'alimento', 'huevos de gallina por 30 unidades',
15000.00, 110, 20, 'panal'),
('huevo a blanco', 'alimento', 'huevos de gallina por unidad', 500.00, 1017, 600,
'unidad'),
('panal de huevo a blanco', 'alimento', 'huevos de gallina por 30 unidades',
14000.00, 33, 20, 'panal'),
('huevo b blanco', 'alimento', 'huevos de gallina por unidad', 500.00, 120.00,
300, 'unidad'),
('panal de huevo b blanco', 'alimento', 'huevos de gallina por 30 unidades',
13000.00, 4, 10, 'panal'),
('huevo b rojo', 'alimento', 'huevos de gallina por unidad', 500.00, 375, 300,
'unidad'),
('panal de huevo b rojo', 'alimento', 'huevos de gallina por 30 unidades',
13000.00, 12, 10, 'panal'),
('azucar', 'alimento', 'azucar blanca', 2800.00, 22, 10, 'libras'),
('panela trebol cono', 'alimento', 'panela de cana', 6600.00, 42, 15, 'atao'),
('panela trebol kilera', 'alimento', 'panela de cana', 6300.00, 20, 15, 'kilo'),
('panela supia', 'alimento', 'panela de cana', 5500.00, 24, 15, 'kilo'),
('arroz roa', 'alimento', 'arroz blanco', 2400.00, 140, 25, 'libra'),
('paquete vasos desechables', 'desechables', 'vasos desechables', 2000.00, 2, 1,
'paquete'),
('mirringo', 'alimento para mascotas', 'concentrado para perros', 5600.00, 4, 2,
'paquete'),
('ringo', 'alimento para mascotas', 'concentrado para gatos', 5600.00, 8, 4,
'paquete'),
('sal', 'alimento', 'sal de mesa', 1300.00, 6, 5, 'libras'),
('leche montefrio', 'alimento', 'leche entera coalanta', 3800, 131, 54, 'bolsa'),
('leche entera alqueria', 'alimento', 'leche entera alqueria', 6000.00, 5, 10,
'bolsa'),
('leche alqueria economica', 'alimento', 'leche entera alqueria', 4800.00, 2, 10,
'bolsa'),
('leche alqueria personal', 'alimento', 'leche entera alqueria', 1000.00, 13, 5,
'bolsa'),
('crema de leche 125 ml', 'alimento', 'crema de leche', 3200.00, 2, 4, 'unidad'),
('crema de leche 180 ml', 'alimento', 'crema de leche', 5400.00, 4, 4, 'unidad'),
('aceite 250 ml', 'alimento', 'aceite de cocina', 2000.00, 23, 10, 'botella'),
('aceite 500 ml', 'alimento', 'aceite de cocina', 3800.00, 22, 10, 'botella'),
('aceite 1000 ml', 'alimento', 'aceite de cocina', 7400.00, 22, 10, 'botella'),
('aceite 2000 ml', 'alimento', 'aceite de cocina', 15300.00, 12, 5, 'botella'),
('aceite 3000 ml', 'alimento', 'aceite de cocina', 23000.00, 12, 4, 'botella'),
('electrolit', 'bebida', 'bebida hidratante', 7800.00, 2, 5, 'botella'),
('atun', 'alimento', 'atun enlatado', 3700.00, 40, 15, 'unidad'),
('escoba', 'limpieza', 'escobas', 7200.00, 3, 3, 'unidad'),
('trapeador xl', 'limpieza', 'trapeadores', 8700.00, 1, 3, 'unidad'),
('trapeador de microfibra', 'limpieza', 'trapeadores', 11400.00, 3, 3, 'unidad'),
('recogedor', 'limpieza', 'recogedores de plastico', 4800.00, 4, 3, 'unidad'),
('lenteja', 'alimento', 'lentejas', 4000.00, 3, 5, 'libra'),
('frijol', 'alimento', 'frijoles', 5800.00, 5, 5, 'libra'),
```



```
( 'espaguetis', 'alimento', 'espaguetis', 2000.00, 2, 5, 'paquete'),
( 'mantequilla la buena', 'alimento', 'mantequilla', 3000, 8, 5, 'unidad'),
( 'chocolate luker', 'alimento', 'chocolate', 1000.00, 27, 10, 'pastilla'),
( 'nescafe 10g', 'alimento', 'cafe', 1600.00, 10, 5, 'sobre'),
( 'papel higienico', 'aseo', 'papel higienico', 3000, 0, 10, 'rollo'),
( 'fruco', 'alimento', 'fruco', 3200.00, 8, 5, 'sobre'),
( 'candela tokai', 'alimento', 'candelas', 1500.00, 28, 10, 'unidad'),
( 'panelada', 'alimento', 'panelada', 1500.00, 5, 10, 'caja'),
( 'maizena', 'alimento', 'maizena', 4300.00, 4, 5, 'caja'),
( 'esponja', 'aseo', 'esponjas', 600.00, 2, 10, 'unidad'),
( 'fibra', 'herramienta', 'fibra para amarrar o sujetar', 18000.00, 3, 3,
'rollo'),
( 'bocadillo', 'alimento', 'bocadillos', 700.00, 53, 15, 'unidad'),
( 'jabon protex', 'aseo', 'jabon', 3800.00, 4, 5, 'unidades'),
( 'maggi', 'alimento', 'sazonador', 600.00, 64, 20, 'cubo');
```

-- Pedidos (Lo actualiza el usuario cuando realice un pedido real) (pendiente, completado, cancelado)

```
INSERT INTO Pedidos (proveedor_id, fecha_pedido, estado_pedido) VALUES
(1, '2024-11-07 08:30:21', 'pendiente'),
(2, '2024-11-07 08:31:21', 'pendiente'),
(3, '2024-11-07 08:32:21', 'pendiente'),
(4, '2024-11-07 08:33:21', 'pendiente'),
(5, '2024-11-07 08:34:21', 'pendiente'),
(6, '2024-11-07 08:35:21', 'pendiente'),
(7, '2024-11-07 08:36:21', 'pendiente'),
(8, '2024-11-07 08:37:21', 'pendiente'),
(9, '2024-11-07 08:38:21', 'pendiente'),
(10, '2024-11-07 08:39:21', 'pendiente'),
(11, '2024-11-07 08:40:21', 'pendiente');
```

-- Detalle\_Pedido (Lo actualiza el usuario cuando realice un pedido real)

```
INSERT INTO Detalle_Pedido (producto_id, pedido_id, cantidad_solicitada,
cantidad_recibida) VALUES
(1, 1, 100, 0),
(2, 1, 10, 0),
(3, 2, 100, 0),
(4, 2, 10, 0),
(5, 3, 100, 0),
(6, 3, 10, 0),
(7, 4, 100, 0),
(8, 4, 10, 0),
(9, 5, 100, 0),
(10, 5, 10, 0),
(11, 6, 100, 0),
(12, 6, 10, 0),
(13, 7, 100, 0),
(14, 7, 10, 0),
(15, 8, 100, 0),
(16, 8, 10, 0),
(17, 9, 100, 0),
(18, 9, 10, 0),
(19, 10, 100, 0),
(20, 10, 10, 0),
(21, 11, 100, 0),
(22, 11, 10, 0);
```

-- Compras (Lo actualiza el usuario cuando realice una compra real)

```
INSERT INTO Compras (pedido_id, fecha_compra, monto_total) VALUES
(1, '2024-11-07', 0),
(2, '2024-11-07', 0),
(3, '2024-11-07', 0),
(4, '2024-11-07', 0),
```

```
(5, '2024-11-07', 0),
(6, '2024-11-07', 0),
(7, '2024-11-07', 0),
(8, '2024-11-07', 0),
(9, '2024-11-07', 0),
(10, '2024-11-07', 0),
(11, '2024-11-07', 0);
```

-- *Detalle\_Compra (Lo actualiza el usuario cuando realice una compra real) El subtotal es cero ya que aun no se han registrado las cantidades compradas ya que los pedidos estan pendientes*

**INSERT INTO Detalle\_Compra** (producto\_id, compra\_id, cantidad\_comprada, precio\_unitario, subtotal) **VALUES**

```
(1, 1, 0, 530.00, 0),
(2, 1, 0, 15900.00, 0),
(3, 2, 0, 440.00, 0),
(4, 2, 0, 13200.00, 0),
(5, 3, 0, 400.00, 0),
(6, 3, 0, 12000.00, 0),
(7, 4, 0, 370.00, 0),
(8, 4, 0, 11100.00, 0),
(9, 5, 0, 350.00, 0),
(10, 5, 0, 10500.00, 0),
(11, 6, 0, 2300.00, 0),
(12, 6, 0, 5500.00, 0),
(13, 7, 0, 5250.00, 0),
(14, 7, 0, 4600.00, 0),
(15, 8, 0, 2000.00, 0),
(16, 8, 0, 2000.00, 0),
(17, 9, 0, 4700.00, 0),
(18, 9, 0, 4700.00, 0),
(19, 10, 0, 1070.00, 0),
(20, 10, 0, 3170.00, 0),
(21, 11, 0, 5000.00, 0),
(22, 11, 0, 4000.00, 0);
```

-- *Clientes (El usuario debe registrar los clientes)*

**INSERT INTO Clientes** (DNI\_cliente, nombre\_cliente, tipo\_cliente, total\_compras) **VALUES**

```
('1055357609', 'juan perez', 'particular', 0),
('1088394820', 'maria rodriguez', 'regular', 0),
('1085323883', 'juan carlos', 'nuevo', 0),
('9876543210', 'tienda juanchito', 'particular', 0),
('1234567891', 'samuel rios', 'particular', 0),
('9876543211', 'kamila', 'regular', 0),
('1234567892', 'la coste', 'nuevo', 0),
('9876543212', 'colibri', 'particular', 0),
('1234567893', 'lina gallego', 'regular', 0),
('9876543213', 'saul baron', 'regular', 0);
```

-- *Descuentos (El usuario debe registrar los descuentos)*

**INSERT INTO Descuentos** (DNI\_cliente, porcentaje\_dcto, descripcion) **VALUES**

```
('1055357609', 0.00, 'descuento por ser cliente particular'),
('1088394820', 0.10, 'descuento por ser cliente regular'),
('1085323883', 0.15, 'descuento por ser cliente nuevo'),
('9876543210', 0.00, 'descuento por ser cliente particular'),
('1234567891', 0.00, 'descuento por ser cliente particular'),
('9876543211', 0.10, 'descuento por ser cliente regular'),
('1234567892', 0.15, 'descuento por ser cliente nuevo'),
('9876543212', 0.00, 'descuento por ser cliente particular'),
('1234567893', 0.10, 'descuento por ser cliente regular'),
('9876543213', 0.10, 'descuento por ser cliente regular');
```

-- Usuarios

```
INSERT INTO Usuarios (DNI_usuario, primer_nombre, segundo_nombre,
primer_apellido, segundo_apellido, telefono, rol, password) VALUES
('1234567890', 'janneth', '', 'zapata', 'mejia', '3145893245', 'administrador',
'123456'),
('1234567891', 'oswaldo', '', 'sanchez', 'moreno', '3154768534', 'administrador',
'654321'),
('1234567892', 'juan', 'pablo', 'sanchez', 'zapata', '3125467238', 'vendedor',
'987654');
```

-- Ventas (Lo actualiza el usuario cuando realice una venta real)

-- El total de venta se actualiza automaticamente con la suma de los subtotales de los productos vendidos

```
INSERT INTO Ventas (DNI_usuario, DNI_cliente, fecha_venta, total_venta) VALUES
('1234567890', '1055357609', '2024-11-07 08:30:21', 0),
('1234567891', '1088394820', '2024-11-07 08:31:21', 0),
('1234567892', '1085323883', '2024-11-07 08:32:21', 0),
('1234567890', '9876543210', '2024-11-07 08:33:21', 0),
('1234567891', '1234567891', '2024-11-07 08:34:21', 0),
('1234567892', '9876543211', '2024-11-07 08:35:21', 0),
('1234567890', '1234567892', '2024-11-07 08:36:21', 0),
('1234567891', '9876543212', '2024-11-07 08:37:21', 0),
('1234567892', '1234567893', '2024-11-07 08:38:21', 0),
('1234567890', '9876543213', '2024-11-07 08:39:21', 0);
```

-- Detalle\_Venta El precio unitario se extrae del precio de venta en la tabla de productos. Por el momento estos datos son ficticios. El usuario debe actualizarlos cuando realice una venta real.

-- El subtotal se generará automaticamente con la cantidad vendida y el precio unitario

```
INSERT INTO Detalle_Venta (producto_id, venta_id, cantidad_vendida,
precio_unitario, subtotal) VALUES
(1, 1, 100, 700.00, 0),
(2, 1, 10, 18000.00, 0),
(3, 2, 100, 600.00, 0),
(4, 2, 10, 15000.00, 0),
(5, 3, 100, 500.00, 0),
(6, 3, 10, 14000.00, 0),
(7, 4, 100, 500.00, 0),
(8, 4, 10, 13000.00, 0),
(9, 5, 100, 500.00, 0),
(10, 5, 10, 13000.00, 0),
(11, 6, 100, 2800.00, 0);
```

-- Pagos (Lo actualiza el usuario cuando realice un pago real)

```
INSERT INTO Pagos (venta_id, monto_pagado, fecha_pago, metodo_pago, estado,
saldo_pendiente) VALUES
(1, 0, '2024-11-07', 'efectivo', 'pendiente', 0),
(2, 0, '2024-11-07', 'efectivo', 'pendiente', 0),
(3, 0, '2024-11-07', 'efectivo', 'pendiente', 0),
(4, 0, '2024-11-07', 'efectivo', 'pendiente', 0),
(5, 0, '2024-11-07', 'efectivo', 'pendiente', 0),
(6, 0, '2024-11-07', 'efectivo', 'pendiente', 0),
(7, 0, '2024-11-07', 'efectivo', 'pendiente', 0),
(8, 0, '2024-11-07', 'efectivo', 'pendiente', 0),
(9, 0, '2024-11-07', 'efectivo', 'pendiente', 0),
(10, 0, '2024-11-07', 'efectivo', 'pendiente', 0);
```

### 3.4. Procedimientos

A continuación, se presentan cada uno de los procedimientos usados en la base de datos, los cuales fueron útiles a la hora de realizar los endpoints de cliente-servidor.

**Nota:** Cada procedimiento está debidamente comentado con la explicación de lo que hace.

Los procedimientos son valga la redundancia, procesos que actúan como funciones pero sin devolver ningún valor, se pueden hacer cálculos e inserción de datos. También selección o cualquier tipo de consulta. Para los procedimientos usados en esta base de datos, fue muy importante tener procedimientos que insertaran datos pero que de manera automática hiciera un cálculo importante. Ayuda bastante.

#### 3.4.1. Gestión de inventario

Con estos procedimientos se hacen cálculos de Stock, se automatiza el proceso para agregar un producto, se actualizan las disponibilidades de productos luego de una venta o de recibir un pedido y también se pueden obtener datos de un producto específicos.

```
-- PROCEDIMIENTOS
```

```
-- GESTION DE INVENTARIO
```

```
DELIMITER //
```

```
-- Procedimiento para agregar un producto nuevo al inventario
```

```
CREATE PROCEDURE AgregarProducto(  
    IN nombre_producto VARCHAR(100),  
    IN tipo_producto VARCHAR(50),  
    IN descripcion TEXT,  
    IN precio_venta DECIMAL(10, 2),  
    IN stock_actual INT,  
    IN stock_minimo INT,  
    IN unidad_medida VARCHAR(20)  
)  
BEGIN  
    INSERT INTO Productos (nombre_producto, tipo_producto, descripcion,  
precio_venta, stock_actual, stock_minimo, unidad_medida)  
    VALUES (nombre_producto, tipo_producto, descripcion, precio_venta,  
stock_actual, stock_minimo, unidad_medida);  
END //
```

```
-- Procedimiento para actualizar el nombre de un producto
```

```
CREATE PROCEDURE ActualizarNombreProducto(  
    IN producto_id INT,  
    IN nuevo_nombre VARCHAR(100)  
)  
BEGIN  
    UPDATE Productos SET nombre_producto = nuevo_nombre WHERE producto_id =  
producto_id;  
END //
```

```
-- Procedimiento para visualizar el stock actual de un producto
```

```
CREATE PROCEDURE VerStockActual(  
    IN producto_id INT  
)  
BEGIN
```

```

    SELECT stock_actual FROM Productos WHERE producto_id = producto_id;
END //

-- Procedimiento para visualizar el stock minimo de un producto
CREATE PROCEDURE VerStockMinimo(
    IN producto_id INT
)
BEGIN
    SELECT stock_minimo FROM Productos WHERE producto_id = producto_id;
END //

-- Procedimiento para modificar el stock minimo de un producto
CREATE PROCEDURE ActualizarStockMinimo(
    IN producto_id INT,
    IN nuevo_stock_minimo INT
)
BEGIN
    UPDATE Productos SET stock_minimo = nuevo_stock_minimo WHERE producto_id =
producto_id;
END //

-- Procedimiento para actualizar el precio de venta de un producto
CREATE PROCEDURE ActualizarPrecioVenta(
    IN producto_id INT,
    IN precio_venta DECIMAL(10, 2)
)
BEGIN
    UPDATE Productos SET precio_venta = precio_venta WHERE producto_id =
producto_id;
END //

-- Procedimiento para visualizar el precio de venta de un producto
CREATE PROCEDURE VerPrecioVenta(
    IN producto_id INT
)
BEGIN
    SELECT precio_venta FROM Productos WHERE producto_id = producto_id;
END //

-- Procedimiento para actualizar el precio de compra de un producto
CREATE PROCEDURE ActualizarPrecioCompra(
    IN producto_id INT,
    IN precio_compra_unitario DECIMAL(10, 2)
)
BEGIN
    UPDATE Detalle_Compra SET precio_unitario = precio_compra_unitario WHERE
producto_id = producto_id;
END //

-- Procedimiento para visualizar el precio de compra de un producto
CREATE PROCEDURE VerPrecioCompra(
    IN producto_id INT
)
BEGIN
    SELECT precio_unitario FROM Detalle_Compra WHERE producto_id = producto_id;
END //

-- Procedimiento para actualizar el stock actual de un producto automaticamente
despues de una venta
CREATE PROCEDURE ActualizarStock(
    IN producto_id INT,
    IN cantidad_vendida INT
)

```

```

BEGIN
    UPDATE Productos SET stock_actual = stock_actual - cantidad_vendida WHERE
    producto_id = producto_id;
END //

```

### 3.4.2. Gestión de ventas

Con estos procedimientos se puede registrar una venta, calcular el total de una venta, o calcular el total de una venta, lo cual será muy importante dado que una venta puede tener varios detalles ya que un cliente puede llevar más de un producto.

```

-- GESTION DE VENTAS

-- Procedimiento para registrar una venta
CREATE PROCEDURE RegistrarVenta(
    IN DNI_usuario VARCHAR(10),
    IN DNI_cliente VARCHAR(10),
    IN fecha_venta DATETIME,
    IN total_venta DECIMAL(10, 2)
)
BEGIN
    INSERT INTO Ventas (DNI_usuario, DNI_cliente, fecha_venta, total_venta)
    VALUES (DNI_usuario, DNI_cliente, fecha_venta, total_venta);
END //

-- Procedimiento para agregar un detalle de venta
CREATE PROCEDURE AgregarDetalleVenta(
    IN producto_id INT,
    IN venta_id INT,
    IN cantidad_vendida INT,
    IN precio_unitario DECIMAL(10, 2)
)
BEGIN
    DECLARE subtotal DECIMAL(10, 2);
    SET subtotal = cantidad_vendida * precio_unitario;
    INSERT INTO Detalle_Venta (producto_id, venta_id, cantidad_vendida,
    precio_unitario, subtotal)
    VALUES (producto_id, venta_id, cantidad_vendida, precio_unitario, subtotal);
END //

-- Procedimiento para calcular el total de una venta
CREATE PROCEDURE CalcularTotalVenta(
    IN venta_id INT
)
BEGIN
    DECLARE total DECIMAL(10, 2);
    SET total = (SELECT SUM(subtotal) FROM Detalle_Venta WHERE venta_id =
    venta_id);
    UPDATE Ventas SET total_venta = total WHERE venta_id = venta_id;
END //

-- Procedimiento para aplicar un descuento
CREATE PROCEDURE AplicarDescuento(
    IN DNI_cliente VARCHAR(10),
    IN venta_id INT
)
BEGIN
    DECLARE descuento DECIMAL(5, 2);
    DECLARE total DECIMAL(10, 2);

```

```

-- Se debe buscar el porcentaje de descuento del cliente
SELECT porcentaje_dcto INTO descuento FROM Descuentos
WHERE DNI_cliente = DNI_cliente;

-- Se aplica el descuento al total si existe
IF descuento IS NOT NULL OR descuento > 0 THEN
    SELECT total_venta INTO total FROM Ventas
    WHERE venta_id = venta_id;

    -- Se aplica el descuento
    SET total = total - (total * descuento);

    -- Se actualiza el total de la venta
    UPDATE Ventas SET total_venta = total WHERE venta_id = venta_id;
END IF;
END //

```

### 3.4.3. Gestión de proveedores y pedidos

Similarmente con los procedimientos anteriores, estos procedimientos permiten agregar un proveedor o agregar un pedido y su detalle.

```

-- GESTION DE PROVEEDORES y PEDIDOS

-- Procedimiento para agregar un proveedor
CREATE PROCEDURE AgregarProveedor(
    IN nombre_proveedor VARCHAR(100),
    IN telefono VARCHAR(15),
    IN direccion VARCHAR(150),
    IN productos_proveedor TEXT
)
BEGIN
    INSERT INTO Proveedores (nombre_proveedor, telefono, direccion,
productos_proveedor)
VALUES (nombre_proveedor, telefono, direccion, productos_proveedor);
END //

-- Procedimiento para agregar un pedido
CREATE PROCEDURE AgregarPedido(
    IN proveedor_id INT,
    IN fecha_pedido DATETIME,
    IN estado_pedido VARCHAR(20)
)
BEGIN
    INSERT INTO Pedidos (proveedor_id, fecha_pedido, estado_pedido)
VALUES (proveedor_id, fecha_pedido, estado_pedido);
END //

-- Procedimiento para agregar un detalle de pedido
CREATE PROCEDURE AgregarDetallePedido(
    IN producto_id INT,
    IN pedido_id INT,
    IN cantidad_solicitada INT,
    IN cantidad_recibida INT
)
BEGIN
    INSERT INTO Detalle_Pedido (producto_id, pedido_id, cantidad_solicitada,
cantidad_recibida)
VALUES (producto_id, pedido_id, cantidad_solicitada, cantidad_recibida);
END //

```

### 3.4.4. Gestión de compras

Para esta sección también se tienen procedimientos de inserción, sin embargo, uno de los más importantes es el procedimiento para actualizar el stock luego de una compra. Es importante recalcar que una compra para esta empresa es un pedido que ya se pagó, dado que a veces hay muchas inconsistencias con la entrega de los pedidos en cuanto a la cantidad solicitada y la cantidad recibida, entonces no se puede tomar un pedido incompleto como una compra hasta que el pedido sea completo.

```
-- GESTION DE COMPRAS
```

```
-- Procedimiento para registrar una compra
```

```
CREATE PROCEDURE RegistrarCompra(  
    IN pedido_id INT,  
    IN fecha_compra DATE,  
    IN monto_total DECIMAL(10, 2)  
)  
BEGIN  
    INSERT INTO Compras (pedido_id, fecha_compra, monto_total)  
    VALUES (pedido_id, fecha_compra, monto_total);  
END //
```

```
-- Procedimiento para agregar un detalle de compra
```

```
CREATE PROCEDURE AgregarDetalleCompra(  
    IN producto_id INT,  
    IN compra_id INT,  
    IN cantidad_comprada INT,  
    IN precio_unitario DECIMAL(10, 2)  
)  
BEGIN  
    DECLARE subtotal DECIMAL(10, 2);  
    SET subtotal = cantidad_comprada * precio_unitario;  
    INSERT INTO Detalle_Compra (producto_id, compra_id, cantidad_comprada,  
    precio_unitario, subtotal)  
    VALUES (producto_id, compra_id, cantidad_comprada, precio_unitario,  
    subtotal);  
END //
```

```
-- Procedimiento para actualizar stock actual de un producto despues de una compra (Pedido completado)
```

```
CREATE PROCEDURE ActualizarStockCompra(  
    IN producto_id INT,  
    IN cantidad_comprada INT  
)  
BEGIN  
    UPDATE Productos SET stock_actual = stock_actual + cantidad_comprada WHERE  
    producto_id = producto_id;  
END //
```

```
-- Procedimiento para obtener el monto total de una compra
```

```
CREATE PROCEDURE CalcularTotalCompra(  
    IN compra_id INT  
)  
BEGIN  
    DECLARE total DECIMAL(10, 2);  
    SET total = (SELECT SUM(subtotal) FROM Detalle_Compra WHERE compra_id =  
    compra_id);  
    UPDATE Compras SET monto_total = total WHERE compra_id = compra_id;  
END //
```

```
-- Procedimiento para obtener discrepancias en las cantidades solicitadas y recibidas
```



```

CREATE PROCEDURE VerificarDiscrepancias (
    IN pedido_id INT
)
BEGIN
    SELECT * FROM Detalle_Pedido WHERE pedido_id = pedido_id AND
    cantidad_solicitada != cantidad_recibida;
END //

```

### 3.4.5. Gestión de pagos

A través de estos procedimientos se puede registrar un pago, actualizar el estado de un pago y ver el estado de un pago.

```

-- GESTION DE PAGOS

-- Procedimiento para registrar un pago
CREATE PROCEDURE RegistrarPago (
    IN venta_id_param INT,
    IN monto_pagado DECIMAL(10, 2),
    IN fecha_pago DATE,
    IN metodo_pago VARCHAR(20),
    IN estado VARCHAR(20)
)
BEGIN
    DECLARE saldo DECIMAL(10, 2);

    -- Calcular el saldo pendiente
    SET saldo = (SELECT total_venta FROM Ventas WHERE venta_id = venta_id_param)
    - monto_pagado;

    -- Insertar el pago
    INSERT INTO Pagos (venta_id, monto_pagado, fecha_pago, metodo_pago, estado,
    saldo_pendiente)
    VALUES (venta_id_param, monto_pagado, fecha_pago, metodo_pago, estado,
    saldo);
END //

-- Procedimiento para actualizar el estado de un pago
CREATE PROCEDURE ActualizarEstadoPago (
    IN venta_id INT,
    IN nuevo_estado VARCHAR(20)
)
BEGIN
    UPDATE Pagos SET estado = nuevo_estado WHERE venta_id = venta_id;
END //

-- Procedimiento para visualizar el estado de un pago
CREATE PROCEDURE VerEstadoPago (
    IN venta_id INT
)
BEGIN
    SELECT estado FROM Pagos WHERE venta_id = venta_id;
END //

-- Procedimiento para visualizar el saldo pendiente de un pago
CREATE PROCEDURE VerSaldoPendiente (
    IN venta_id INT
)
BEGIN
    SELECT saldo_pendiente FROM Pagos WHERE venta_id = venta_id;
END //

```

```

-- Procedimiento para actualizar el saldo pendiente de un pago
CREATE PROCEDURE ActualizarSaldoPendiente (
    IN venta_id INT,
    IN abono DECIMAL(10, 2)
)
BEGIN
    UPDATE Pagos SET saldo_pendiente = saldo_pendiente - abono WHERE venta_id =
venta_id;
END //

DELIMITER ;

```

**NOTA IMPORTANTE:** A la hora de colocar un procedimiento es necesario colocarle un delimitador distinto a ';' por lo tanto, si desea agregar los procedimientos a la base de datos puede hacerlo copiando de inicio a fin todos los procedimientos. Si se planea hacer de a un procedimiento o por bloques deberá agregar el delimitador al inicio así:

'DELIMITER //' Y al final de cada procedimiento colocar '//'. Cuando termine la inserción de todos los procedimientos termine con 'DELIMITER;'

### 3.5. Vistas y disparadores

Aunque por cuestiones de tiempo para esta entrega de esta versión no se logró manejar todo de manera más sencilla con vistas y disparadores, sino con consultas de selección, inserción, actualización y procedimientos, a continuación se presentan las vistas y disparadores. Las vistas son para tener tablas que contienen datos que el usuario necesita ver casi siempre o siempre. Los disparadores generan alertas o ejecuta funciones o procedimientos si se cumple una condición, es decir si se hace una selección, inserción, actualización o eliminación de un registro.

-- VISTAS

```

-- Vista para visualizar el registro de compra y precio de venta de cada producto
CREATE VIEW Registro_Compras_Ventas AS
SELECT p.nombre_producto, dc.cantidad_comprada, dc.precio_unitario,
p.precio_venta
FROM Productos p, Detalle_Compra dc
WHERE p.producto_id = dc.producto_id;

-- Vista para visualizar el stock actual y stock minimo de cada producto
CREATE VIEW Stock_Actual_Minimo AS
SELECT nombre_producto, stock_actual, stock_minimo
FROM Productos;

-- Vista para visualizar el total de ventas realizadas a cada cliente
CREATE VIEW Total_Ventas_Clientes AS
SELECT c.nombre_cliente, SUM(v.total_venta) AS total_ventas
FROM Ventas v, Clientes c
WHERE v.DNI_cliente = c.DNI_cliente
GROUP BY c.nombre_cliente;

-- Vista para visualizar el detalle de ventas por cliente para cada venta y que
productos se vendieron
CREATE VIEW Detalle_Ventas_Clientes AS
SELECT c.nombre_cliente, v.venta_id, dv.producto_id, dv.cantidad_vendida,
dv.precio_unitario, dv.subtotal

```

```

FROM Ventas v, Clientes c, Detalle_Venta dv
WHERE v.DNI_cliente = c.DNI_cliente AND v.venta_id = dv.venta_id;

-- Vista para historial de pedidos
CREATE VIEW Historial_Pedidos AS
SELECT p.nombre_producto, pr.nombre_proveedor, dp.cantidad_solicitada,
dp.cantidad_recibida, pe.fecha_pedido, pe.estado_pedido
FROM Productos p, Proveedores pr, Detalle_Pedido dp, Pedidos pe
WHERE p.producto_id = dp.producto_id AND pr.proveedor_id = pe.proveedor_id AND
dp.pedido_id = pe.pedido_id;

-- Vista para historial de compras
CREATE VIEW Historial_Compras AS
SELECT p.nombre_producto, dc.cantidad_comprada, dc.precio_unitario,
c.fecha_compra, c.monto_total
FROM Productos p, Detalle_Compra dc, Compras c
WHERE p.producto_id = dc.producto_id AND dc.compra_id = c.compra_id;

-- Vista para historial de pagos
CREATE VIEW Historial_Pagos AS
SELECT v.venta_id, c.nombre_cliente, p.monto_pagado, p.fecha_pago, p.metodo_pago,
p.estado, p.saldo_pendiente
FROM Ventas v, Clientes c, Pagos p
WHERE v.DNI_cliente = c.DNI_cliente AND v.venta_id = p.venta_id;

-- Vista para inventario de productos
CREATE VIEW Inventario_Productos AS
SELECT nombre_producto, precio_venta, stock_actual, stock_minimo, unidad_medida
FROM Productos;

-- Vista para visualizar los productos que se estan agotando
CREATE VIEW Productos_Agotandose AS
SELECT nombre_producto, stock_actual, stock_minimo
FROM Productos
WHERE stock_actual <= stock_minimo;

-- Disparadores

-- Alerta de stock minimo
DELIMITER //

CREATE TRIGGER Alerta_Stock_Minimo
AFTER INSERT ON Productos
FOR EACH ROW
BEGIN
    IF NEW.stock_actual <= NEW.stock_minimo THEN
        INSERT INTO Alertas (mensaje) VALUES ('El producto ' +
NEW.nombre_producto + ' esta por debajo del stock minimo');
    END IF;
END //

-- Actualizar stock actual despues de una venta
CREATE TRIGGER Actualizar_Stock_Venta
AFTER INSERT ON Detalle_Venta
FOR EACH ROW
BEGIN
    UPDATE Productos SET stock_actual = stock_actual - NEW.cantidad_vendida
    WHERE producto_id = NEW.producto_id;
END //

-- Actualizar stock actual despues de una compra
CREATE TRIGGER Actualizar_Stock_Compra
AFTER INSERT ON Detalle_Compra

```

```

FOR EACH ROW
BEGIN
    UPDATE Productos SET stock_actual = stock_actual + NEW.cantidad_comprada
    WHERE producto_id = NEW.producto_id;
END //

-- Actualizar saldo pendiente despues de un pago
CREATE TRIGGER Actualizar_Saldo_Pendiente
AFTER INSERT ON Pagos
FOR EACH ROW
BEGIN
    UPDATE Pagos SET saldo_pendiente = saldo_pendiente - NEW.monto_pagado
    WHERE venta_id = NEW.venta_id;
END //

-- Actualizar estado de pago despues de un pago
CREATE TRIGGER Actualizar_Estado_Pago
AFTER INSERT ON Pagos
FOR EACH ROW
BEGIN
    IF NEW.saldo_pendiente = 0 THEN
        UPDATE Pagos SET estado = 'completado'
        WHERE venta_id = NEW.venta_id;
    END IF;
END //

-- Atualizar estado de pedido despues de una compra
CREATE TRIGGER Actualizar_Estado_Pedido
AFTER INSERT ON Compras
FOR EACH ROW
BEGIN
    UPDATE Pedidos SET estado_pedido = 'completado'
    WHERE pedido_id = NEW.pedido_id;
END //

-- Actualizar total de venta despues de un descuento
CREATE TRIGGER Actualizar_Total_Venta
AFTER INSERT ON Ventas
FOR EACH ROW
BEGIN
    DECLARE descuento DECIMAL(5, 2);
    DECLARE total DECIMAL(10, 2);

    -- Se debe buscar el porcentaje de descuento del cliente
    SELECT porcentaje_dcto INTO descuento FROM Descuentos
    WHERE DNI_cliente = NEW.DNI_cliente;

    -- Se aplica el descuento al total si existe
    IF descuento IS NOT NULL OR descuento > 0 THEN
        SELECT total_venta INTO total FROM Ventas
        WHERE venta_id = NEW.venta_id;

        -- Se aplica el descuento
        SET total = total - (total * descuento);

        -- Se actualiza el total de la venta
        UPDATE Ventas SET total_venta = total WHERE venta_id = NEW.venta_id;
    END IF;
END //

DELIMITER ;

```

#### 4. ANEXOS

Se presentan a continuación, imágenes de la aplicación en funcionamiento:



The image shows the login screen of the 'Avícola El Manantial' application. It features a white login form centered on an orange gradient background. The form has a title 'Inicio de Sesión' and two input fields: 'DNI de usuario' and 'Contraseña'. Below the fields is a large orange button labeled 'Iniciar Sesión' and a smaller link labeled 'Regístrame'.

**Avícola El Manantial**

**Inicio de Sesión**

DNI de usuario

Contraseña

Iniciar Sesión

Regístrame



## Avícola El Manantial

### Producto nuevo

Nombre del producto

Tipo de producto

Precio del producto

Stock del producto

Stock mínimo del producto

Descripción del producto

Unidad de medida

## Avícola El Manantial

### Busca un producto

ID del producto

Buscar Producto

## Modifica un producto

ID del producto

5

Nombre del producto

huevo a blanco

Tipo de producto

alimento

Precio del producto

500,00

Stock del producto

1017

Stock mínimo del producto

600

Descripción del producto

huevos de gallina por unidad

Unidad de medida

unidad

Modificar Producto

## Historial de Pagos

 [Volver](#)

ID venta	Monto pagado	Fecha de pago	Forma de pago	Estado	Saldo pendiente
1	0.00	2024-11-07T05:00:00.000Z	efectivo	pendiente	0.00
2	0.00	2024-11-07T05:00:00.000Z	efectivo	pendiente	0.00
3	0.00	2024-11-07T05:00:00.000Z	efectivo	pendiente	0.00
4	0.00	2024-11-07T05:00:00.000Z	efectivo	pendiente	0.00
5	0.00	2024-11-07T05:00:00.000Z	efectivo	pendiente	0.00
6	0.00	2024-11-07T05:00:00.000Z	efectivo	pendiente	0.00
7	0.00	2024-11-07T05:00:00.000Z	efectivo	pendiente	0.00
8	0.00	2024-11-07T05:00:00.000Z	efectivo	pendiente	0.00
9	0.00	2024-11-07T05:00:00.000Z	efectivo	pendiente	0.00
10	0.00	2024-11-07T05:00:00.000Z	efectivo	pendiente	0.00

Ejecute la aplicación siguiendo el flujo de inicialización para comprobar el correcto y completo funcionamiento de la aplicación.

Repositorio del proyecto: <https://github.com/jpsz2004/Avicola-El-Manantial.git>