Universidad Autónoma del Estado de Hidalgo

REPORTE DE PROYECTO

Base de Datos Distribuida para la gestión de un Gimnasio

ALUMNO:

Castro Paredes José Ángel
Garcia Melo Ariana
Morales Vázquez Rubén
Pérez Romero Armando Oswaldo

1. Introducción

Descripción general del proyecto

El presente proyecto tiene como objetivo el desarrollo de una plataforma para la gestión integral de gimnasios, basada en una arquitectura de Bases de Datos Distribuidas. Esta solución permitirá a los administradores gestionar de manera eficiente clientes, membresías, clases, entrenadores, pagos y otros aspectos fundamentales del negocio.

La adopción de una base de datos distribuida garantiza la disponibilidad de la información en múltiples ubicaciones físicas, lo que optimiza el rendimiento, facilita la escalabilidad del sistema y mejora la tolerancia a fallos.

Dimensiones del Proyecto

1. **Almacenamiento Distribuido**
   La información se almacenará en múltiples nodos distribuidos estratégicamente, lo que permitirá la gestión eficiente de grandes volúmenes de datos. Cada gimnasio contará con su propio nodo de almacenamiento, mientras que la información crítica, como pagos y membresías, será replicada en otros nodos con el fin de garantizar la disponibilidad y la consistencia de los datos.

2. **Optimización del Rendimiento**
   Al distribuir la carga de trabajo entre varios servidores, se minimizarán los cuellos de botella y se mejorará el tiempo de respuesta del sistema. Esto es especialmente relevante durante los horarios de mayor demanda, como las horas de entrenamiento matutinas y vespertinas.

3. **Escalabilidad Horizontal**
   El diseño del sistema permitirá la expansión progresiva de la plataforma mediante la incorporación de nuevos gimnasios o la ampliación de los existentes sin afectar el rendimiento global. Esta escalabilidad se logrará a través de la adición dinámica de nuevos nodos dentro de la red distribuida.

4. **Seguridad y Consistencia de los Datos**
   Para garantizar la integridad y actualización de la información en todos los nodos, se implementarán mecanismos avanzados de replicación y sincronización de datos. Asimismo, se emplearán protocolos de seguridad robustos para la protección de la información sensible de los clientes, evitando accesos no autorizados o posibles vulnerabilidades.

5. **Enfoque de Diseño Top-Down**
   La arquitectura del sistema se desarrollará siguiendo un enfoque Top-Down, iniciando con la definición de los objetivos estratégicos del negocio y desglosándolos en subsistemas específicos. Este proceso incluirá la elaboración de esquemas globales, la fragmentación lógica de los datos y la asignación eficiente de los fragmentos a los nodos correspondientes.

Desafíos Potenciales y Estrategias de Mitigación

1. **Consistencia de los Datos**
   En un entorno distribuido, mantener la coherencia de los datos es un desafío clave. Para abordar este problema, se implementarán protocolos de consenso, como Paxos o Raft, junto con técnicas avanzadas de replicación para garantizar la integridad de la información.

2. **Rendimiento en Escenarios de Alta Demanda**
   Durante picos de uso, el sistema deberá procesar un gran volumen de transacciones concurrentes sin degradar su desempeño. Para mitigar este riesgo, se emplearán estrategias de balanceo de carga, mecanismos de caching y técnicas de sharding que optimicen el acceso a los datos y reducen la latencia.

3. **Escalabilidad sin Interrupciones**
   A medida que la red de gimnasios crezca, el sistema deberá permitir la incorporación de nuevos nodos sin afectar su operatividad. La arquitectura distribuida facilitará esta expansión de manera natural, permitiendo una integración fluida y sin interrupciones del servicio.

2. Marco teorico

Metodología de diseño SCRUM

Scrum al ser una metodología de desarrollo ágil tiene como base la idea de creación de ciclos breves para el desarrollo, que comúnmente se llaman iteraciones y que en Scrum se llamarán Sprints. Contando con 5 fases:

1. Concepto: Se define de forma general las características del producto y se asigna el equipo que se encargará de su desarrollo.
2. Especulación: en esta fase se hacen disposiciones con la información obtenida y se establecen los límites que marcarán el desarrollo del producto, tales como costes y agendas.
3. Exploración: Se incrementa el producto en el que se añaden las funcionalidades de la fase de especulación.
4. Revisión: El equipo revisa todo lo que se ha construido y se contrasta con el objetivo deseado.
5. Cierre: Se entregará en la fecha acordada una versión del producto deseado. Al tratarse de una versión, el cierre no indica que se ha finalizado el proyecto, sino que seguirá habiendo cambios, denominados "Mantenimiento".

Objetivo General

Desarrollar una plataforma integral de gestión de gimnasios basada en una arquitectura de bases de datos distribuidas, que permita a los administradores gestionar de manera eficiente clientes, membresías, clases, entrenadores y pagos, garantizando alta disponibilidad, escalabilidad y seguridad de la información.

Objetivos Especificos

1. Implementar una arquitectura de bases de datos distribuidas.
2. Optimizar el rendimiento del sistema.
3. Garantizar la escalabilidad del sistema.
4. Asegurar la integridad y seguridad de los datos.
5. Facilitar la gestión integral de gimnasios.
6. Mejorar la experiencia del usuario.
7. Garantizar la alta disponibilidad del sistema.
8. Cumplir con normativas y estándares.

Requerimientos Funcionales y No Funcionales

Funcionales

1. **Reservas de clases y equipos**

   Descripción: Implementar un sistema de reservas para clases grupales, entrenadores personales o equipos específicos (como bicicletas estáticas o máquinas de pesas).

   Requerimientos funcionales:

   - El sistema debe permitir a los usuarios reservar clases o equipos en horarios específicos.
   - El sistema debe enviar recordatorios automáticos (por correo o notificaciones push).
   - El sistema debe permitir a los administradores gestionar la disponibilidad de clases y equipos.

2. **Planes de entrenamiento personalizados** Descripción: Ofrecer a los usuarios planes de entrenamiento personalizados basados en sus objetivos (perder peso, ganar masa muscular, mejorar resistencia, etc.).

   Requerimientos funcionales:

   - El sistema debe permitir a los usuarios definir sus objetivos y preferencias.
   - El sistema debe generar planes de entrenamiento automáticamente o permitir a los entrenadores crearlos manualmente.
   - El sistema debe permitir el seguimiento del progreso del usuario en relación con su plan.

3. **Integración con pasarelas de pago y facturación electrónica**

   Descripción: Facilitar el proceso de pagos y facturación mediante integración con pasarelas de pago (como PayPal, Stripe o MercadoPago) y emisión automática de facturas electrónicas.

   Requerimientos funcionales:

   - El sistema debe permitir pagos en línea con diferentes métodos (tarjeta, PayPal, etc.).
   - El sistema debe generar facturas electrónicas automáticamente después de cada pago.
   - El sistema debe enviar comprobantes de pago por correo electrónico.

No Funcionales

1. **Seguridad**

   Descripción: Implementar medidas adicionales de seguridad, mediante contraseñas para usuarios y entrenadores.

   Requerimientos no funcionales:

   - El sistema debe requerir autenticación a usuarios y entrenadores.
   - El sistema debe registrar y auditar todos los accesos a la plataforma.

2. **Tiempo de respuesta**

   Descripción: Garantizar que el sistema responda rápidamente a las solicitudes de los usuarios, incluso durante picos de demanda.

   Requerimientos no funcionales:

- El sistema debe responder en menos de 2 segundos para el noventa y cinco porciento de las solicitudes.

3. **Capacidad de usuarios concurrentes**

   Descripción: Asegurar que el sistema pueda manejar un gran número de usuarios activos al mismo tiempo sin degradar su rendimiento.

   Requerimientos no funcionales:

   - El sistema debe soportar 1,000 usuarios concurrentes sin degradación del rendimiento.

Alcances y limitaciones

Alcances

- La plataforma permitirá gestionar clientes, membresías y entrenadores.
- Se implementará una arquitectura distribuida básica con tres o cuatro nodos para demostrar el concepto de replicación y consistencia de datos.
- Se desarrollará una interfaz web básica con funcionalidades esenciales (registro de clientes, asignación de membresías, etc.).
- Se implementará autenticación de usuarios con contraseñas encriptadas.

Limitaciones

- El proyecto se desarrollará con recursos limitados (hardware, software y tiempo), por lo que no se implementarán funcionalidades avanzadas como integración con wearables o apps de fitness.
- La arquitectura distribuida se implementará a pequeña escala (4 nodos) debido a limitaciones de infraestructura.
- El proyecto se desarrollará en un plazo limitado de un semestre, por lo que no se podrán implementar todas las funcionalidades deseadas.

3. Casos de Usuarios

3.0.1 Casos de Uso

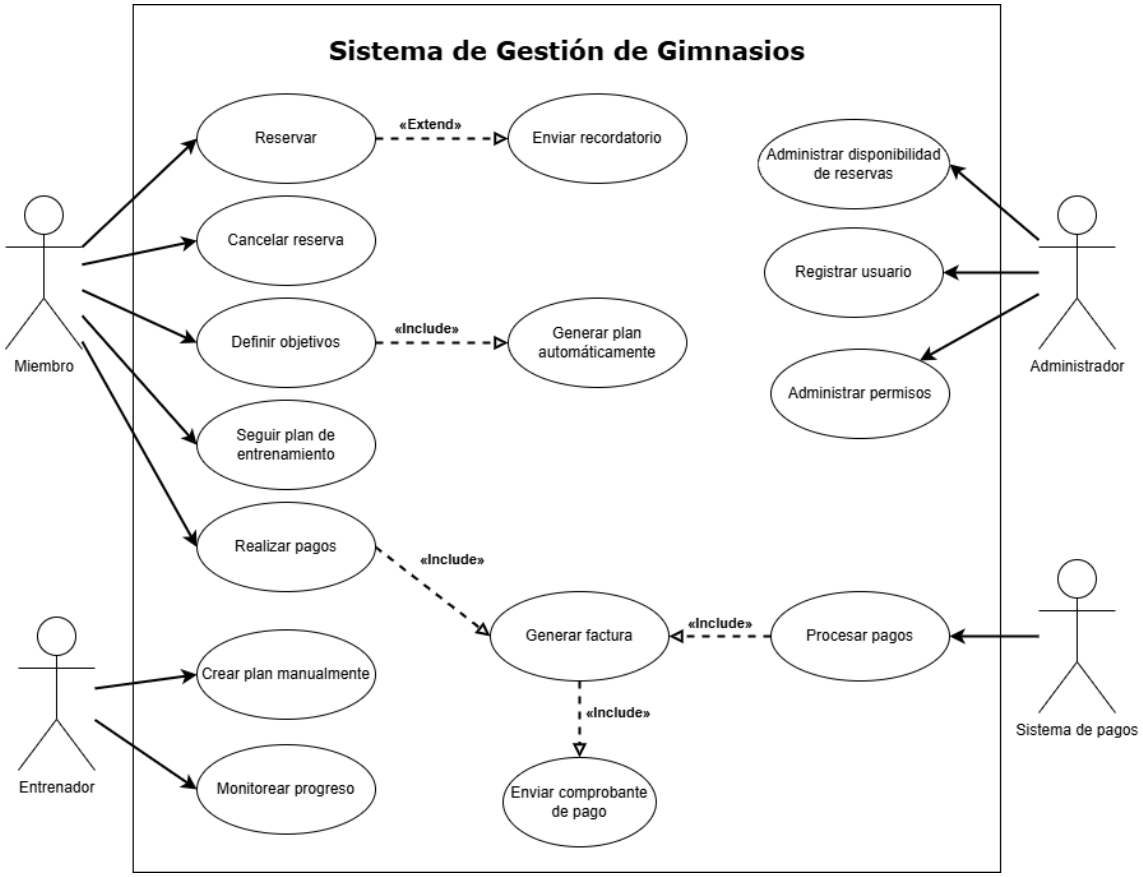

Figure 1: Diagrama de Casos de Uso del Sistema de Gestión de Gimnasios

La Figura 1 muestra el diagrama de casos de uso que representa los principales procesos funcionales del sistema de gestión de gimnasios, distinguiendo claramente las interacciones entre los usuarios del sistema y las funcionalidades disponibles. El modelo incluye dos actores principales: el usuario (cliente del gimnasio) y el administrador. El usuario puede realizar acciones como reservar y cancelar clases, definir objetivos, seguir su plan de entrenamiento, realizar pagos, crear planes manuales y monitorear su progreso. Algunas de estas acciones incluyen otros procesos, como en el caso de "Realizar pagos", que requiere generar una factura y procesar el pago, y que a su vez incluye el envío de un comprobante al cliente.

Asi mismo, al realizar una reserva, el sistema puede extender su funcionalidad para enviar un recordatorio, representado mediante una relación extend. También se ilustra que al definir objetivos, puede incluirse la generación automática de un plan de entrenamiento. Por otro lado, el administrador tiene acceso a funcionalidades internas del sistema como registrar nuevos usuarios, administrar los permisos asignados y gestionar la disponibilidad de reservas. El uso de relaciones include y extend en el diagrama permite modelar correctamente la lógica y dependencia entre los casos de uso, ofreciendo una visión clara del comportamiento esperado del sistema desde la perspectiva de los distintos tipos de usuario. Este diagrama sirve como base para la identificación de requerimientos y el diseño posterior del sistema distribuido.

3.0.2 Secuencial

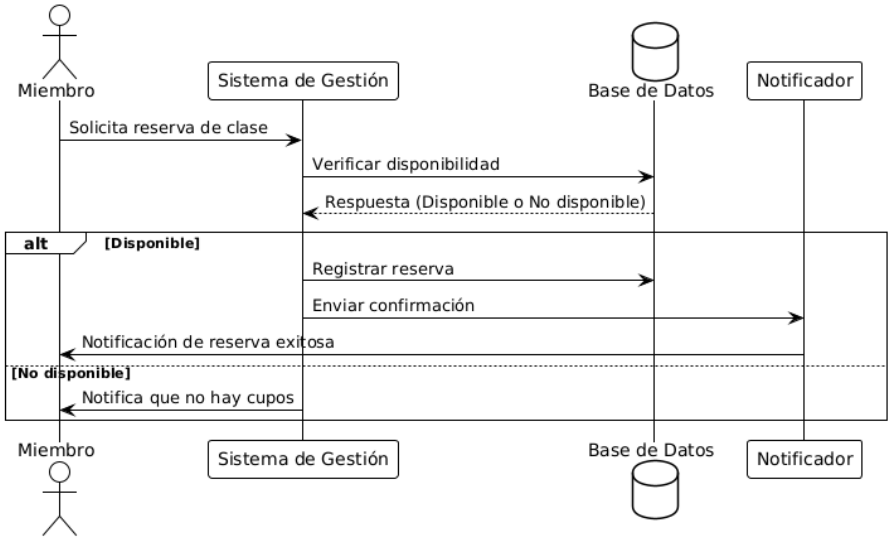

Figure 2: Diagrama de Secuencia para la Reserva de Clases

La Figura 2 representa el diagrama de secuencia correspondiente al proceso de reserva de clases dentro del sistema de gestión de gimnasios. El flujo inicia cuando el actor Miembro solicita una reserva al Sistema de Gestión, el cual consulta a la Base de Datos para verificar la disponibilidad de cupos en la clase solicitada. En caso de disponibilidad, el sistema registra la reserva y solicita al componente Notificador el envío de una confirmación al usuario. Si no hay cupos disponibles, el sistema responde directamente al miembro indicando la imposibilidad de completar la reserva. Este diagrama ilustra la secuencia precisa de interacciones y decisiones que ocurren en tiempo de ejecución durante este proceso fundamental del sistema.

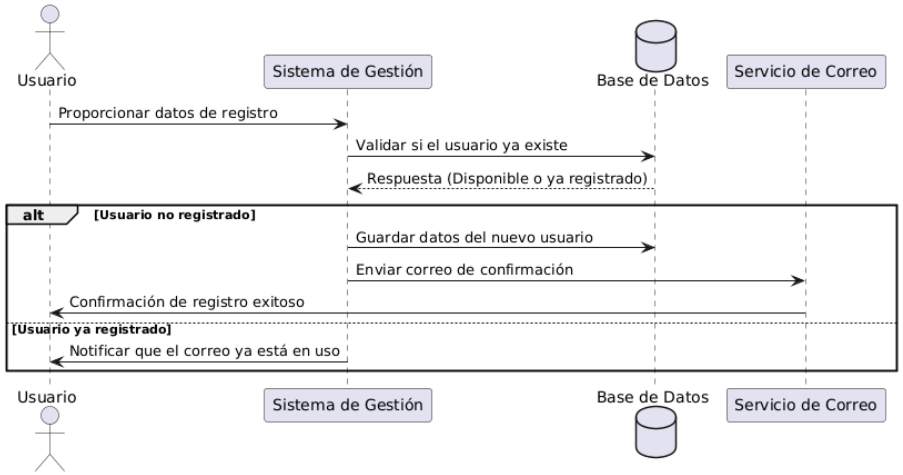

Figure 3: Diagrama de Secuencia para el Registro de un Nuevo Usuario

La Figura 3 ilustra el proceso de registro de un nuevo usuario en el sistema de gestión de gimnasios. El flujo comienza cuando el usuario proporciona sus datos al sistema, el cual consulta la base de datos para verificar si el correo ya está registrado. Si no existe un registro previo, se almacenan los datos y se solicita al servicio de correo el envío de una confirmación al usuario. En caso contrario, el sistema notifica al usuario que el correo electrónico ingresado ya está en uso, deteniendo el proceso de registro.

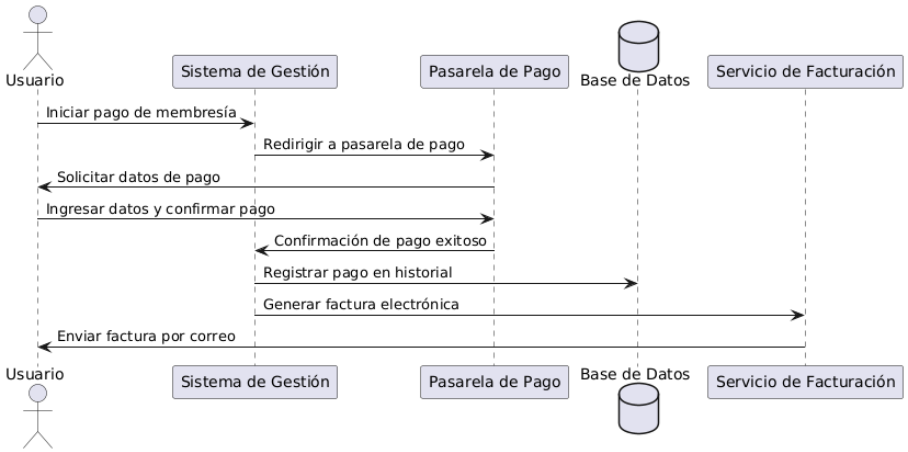

Figure 4: Diagrama de Secuencia para Procesar un Pago de Membresía

La Figura 4 presenta el diagrama de secuencia correspondiente al proceso de pago de una membresía dentro del sistema de gestión de gimnasios. El flujo inicia cuando el usuario solicita realizar el pago, siendo redirigido a la pasarela correspondiente. Tras el ingreso y confirmación de los datos por parte del usuario, la pasarela envía la confirmación al sistema. Este registra el pago en la base de datos y solicita al servicio de facturación la generación del comprobante electrónico, el cual es enviado al usuario como evidencia del proceso completado correctamente.

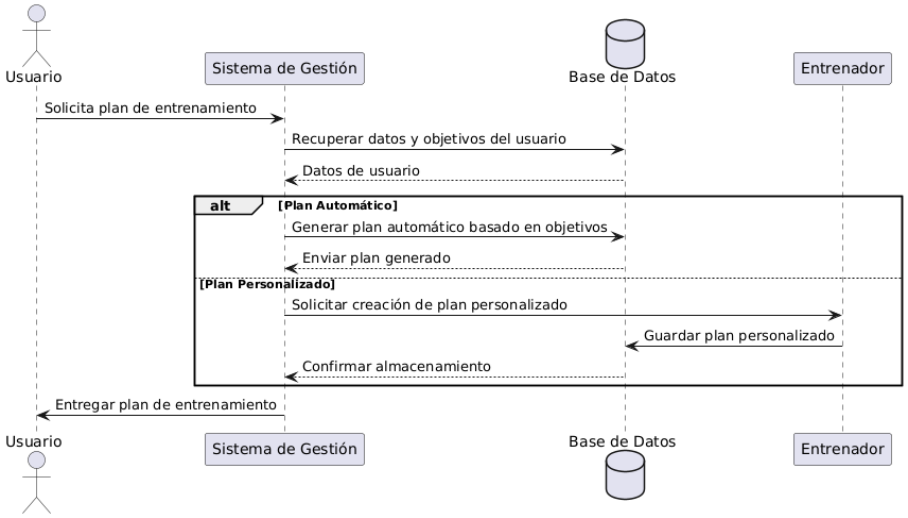

Figure 5: Diagrama de Secuencia para Generar Plan de Entrenamiento Personalizado

La Figura 5 muestra el flujo de interacción para la generación de un plan de entrenamiento dentro del sistema de gestión de gimnasios. El proceso se inicia cuando el usuario solicita su plan. El sistema recupera los datos y objetivos del usuario desde la base de datos y, según la modalidad seleccionada, se genera automáticamente el plan o se solicita al entrenador que lo cree manualmente. En ambos casos, el plan es almacenado en la base de datos y finalmente entregado al usuario, completando el proceso de forma personalizada o automatizada según corresponda.

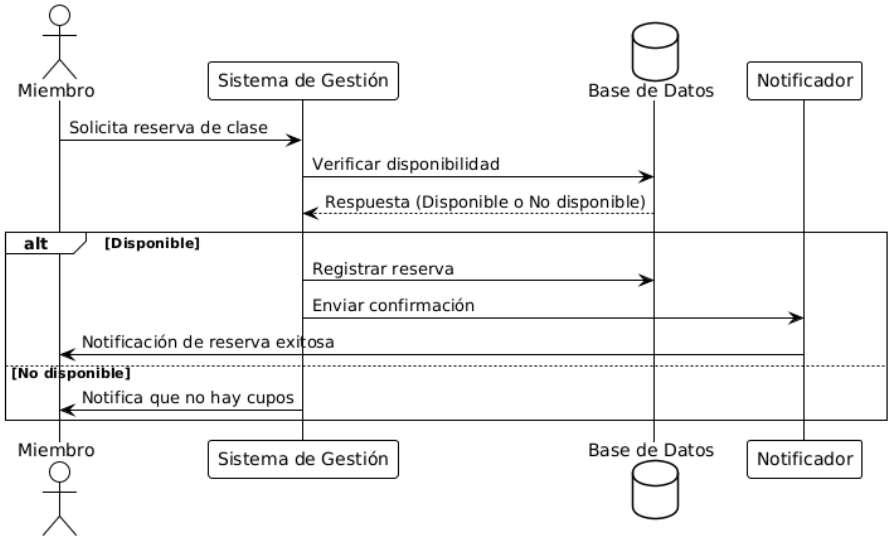

Figure 6: Diagrama de Secuencia para Registrar Asistencia a una Clase

La Figura 6 representa el proceso de registro de asistencia a una clase por parte del usuario. El procedimiento inicia cuando el usuario escanea un código QR o ingresa manualmente el código de su reserva. El sistema valida la reserva consultando la base de datos. Si la reserva es válida, se registra la asistencia y se notifica al usuario que esta ha sido confirmada. En caso contrario, el sistema informa que no se ha encontrado una reserva asociada, evitando el registro erróneo de asistencia.

4. Desarrollo

Modelo Enetidad Relación

	Usuario	Reserva	Entada	Membresía	Plan entrenamiento	Clase	Pago	Sucursal	Entrenador	Factura	Equipo
Usuario		X	X	X	X						
Reserva	X					X					
membresía	X						X				
Entrada	X							X			
Plan entrenamiento	X										
Clase		X							X		
Pago				X						X	
Sucursal			X								X
Entrenador						X					
Factura							X				
Equipo								X			

Figure 7: Modelo Entidad Relación

Modelo Enetidad Relación

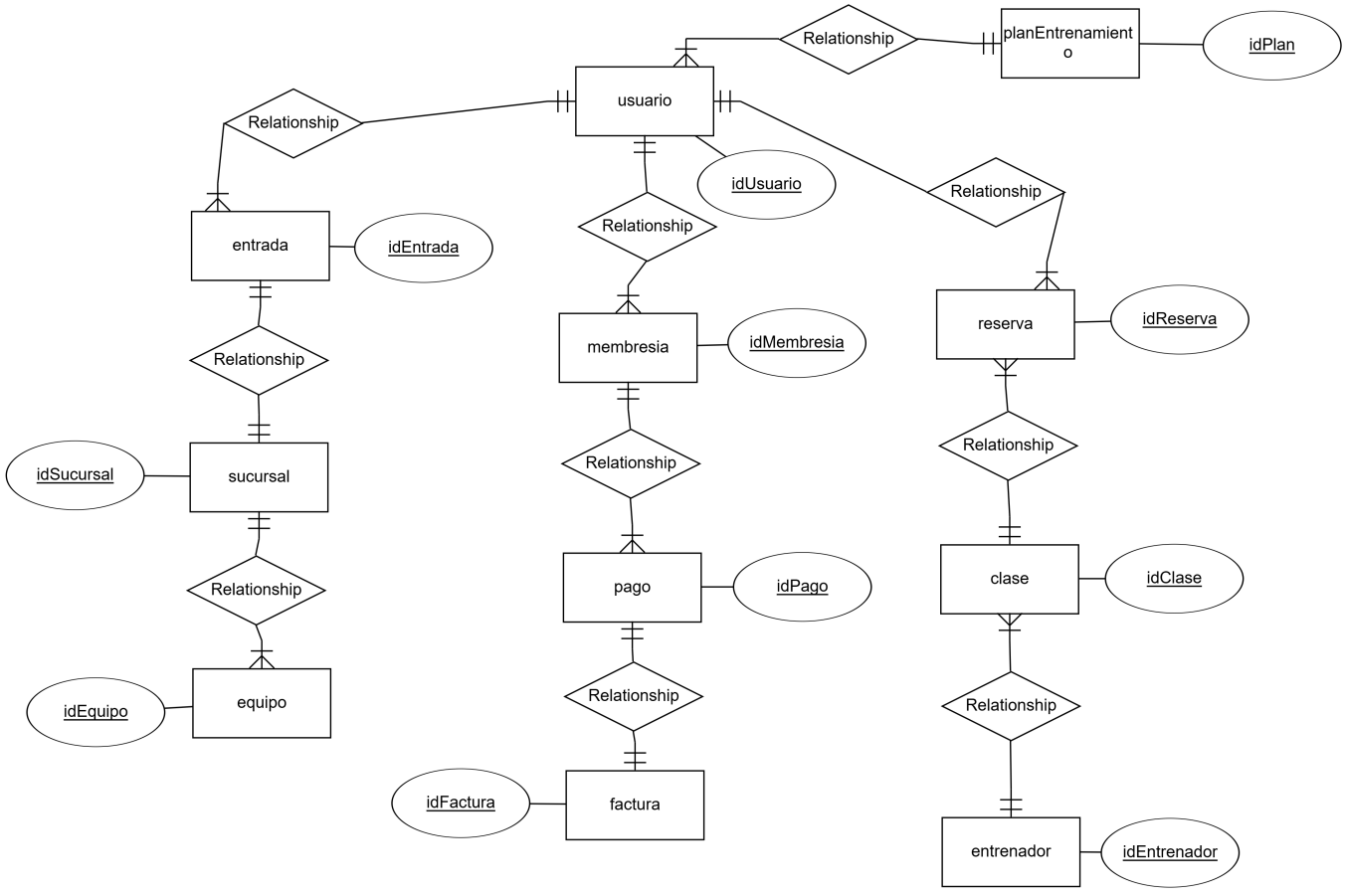

Figure 8: Modelo Entidad Relación

Modelo Relacional

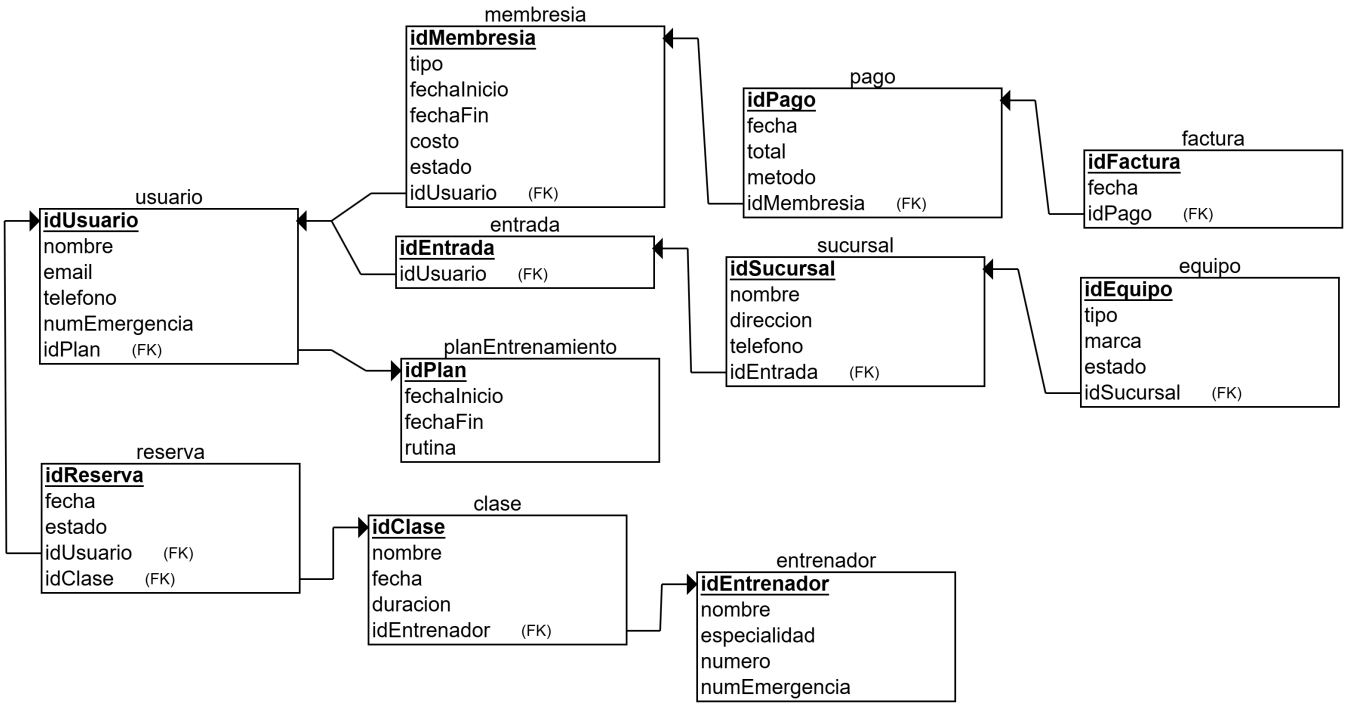

Figure 9: Modelo Relacional

Configuración para Nodos MySQL en LAN

4.4.1 Configuración de la Red LAN con Router WiFi

Se utilizo un router wi-fi de doble banda de la marca Steren como punto de acceso para la red LAN. Se conectaron todos los dispositivos a la red WiFi del router. El router asigna una IP estatica a partir de la dirección MAC del dispositivo.

NODO	IP ESTÁTICA	MÁSCARA DE SUBRED
Tulancingo	192.168.10.199	255.255.255.0
Pachuca	192.168.10.136	255.255.255.0
Siglo XXI	192.168.10.156	255.255.255.0
Servidor web	192.168.10.172	255.255.255.0

PUERTA DE ENLACE: 192.168.10.1

Figure 10: Asignación de redes IP

4.4.2 Configuración de MySQL Workbench y Servidor

En cada nodo se instala el MySQL server y workbench. y se habilitan las conexiones remotas editando el archivo my.ini parar que la aplicación acepte conexiones remotas.

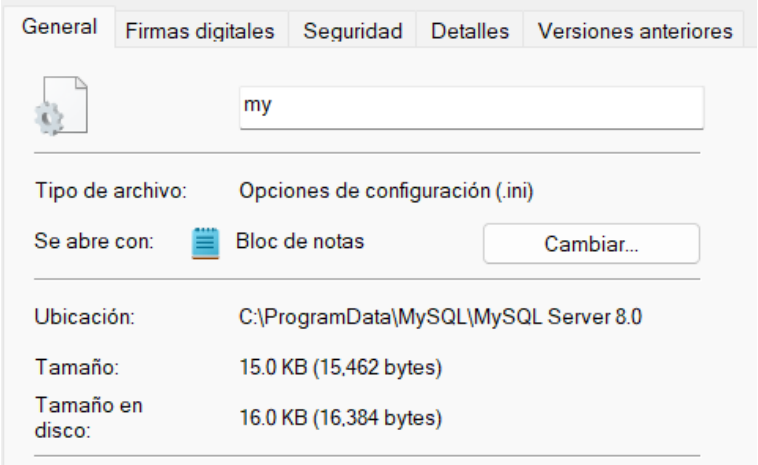

Figure 11: Ubicación de my.ini

El documento se abre con el block de notas y se modifica una linea, de bind-address = 127.0.0.1 por bind-address = 0.0.0.0 y se guardan los cambios.

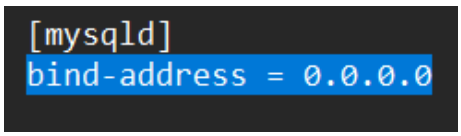

Figure 12: Modificación de my.ini

4.4.3 Crear usuario remoto con permisos

En cada servidor, ejecuta en MySQL:

```
1        CREATE USER 'admin'@'%' IDENTIFIED BY 'admin123';
2        GRANT ALL PRIVILEGES ON *.* TO 'admin'@'%' WITH GRANT OPTION;
3        FLUSH PRIVILEGES;
4
```

4.4.4 Abrir el puerto 3306 en el firewall del nodo

En todos los nodos se abre el Panel de Control > Firewall de Windows Defender > Configuración avanzada. En Reglas de entrada > Nueva regla. Selecciona Puerto, luego TCP, escribe 3306, y Permitir. Se asigna como nombre MySQL Remoto.

4.4.5 Probar la conexión desde otro nodo usando Workbench

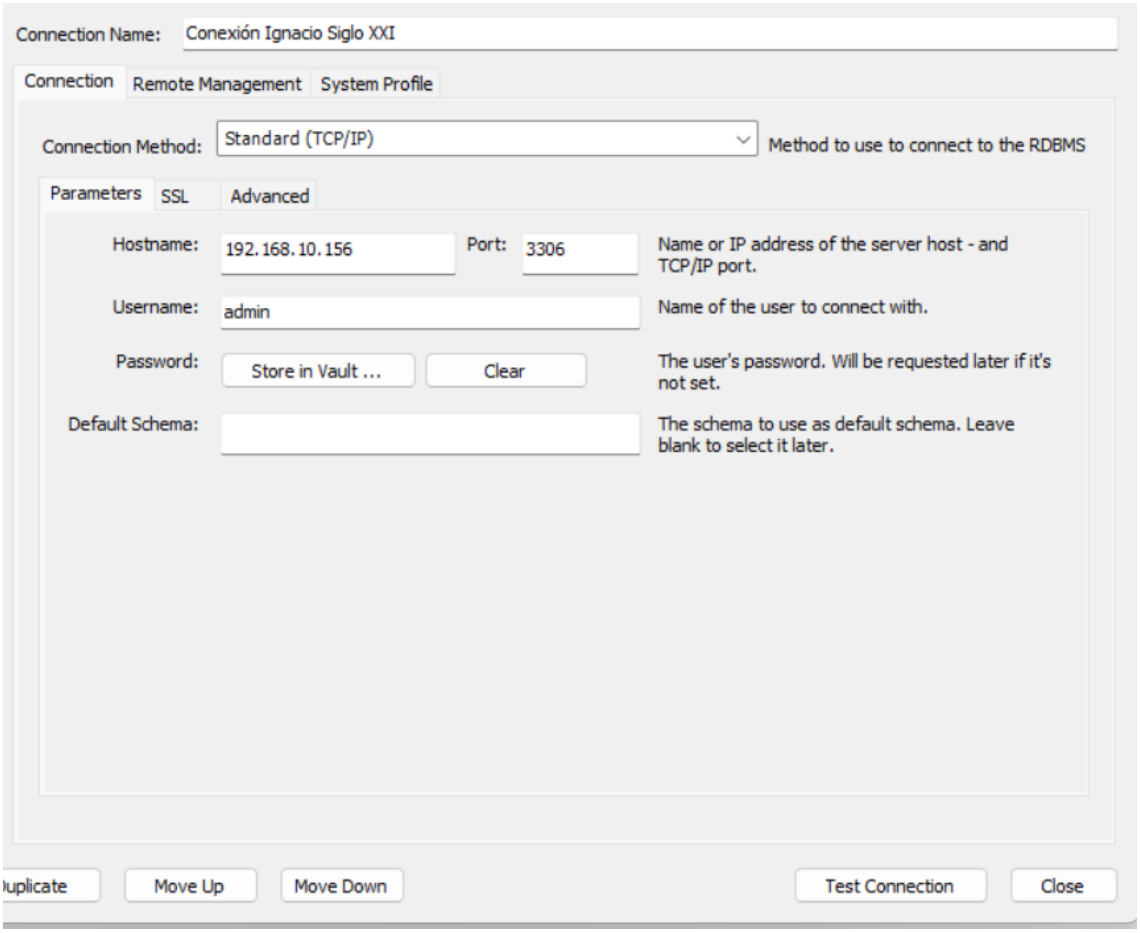

Figure 13: Crea una nueva conexión

4.4.6 Resultado de la conexión de los nodos

Figure 14: Muestra de la conexión desde el nodo Tulancingo

Desarrollo del codigo .py

```python
from flask import Flask, render_template, request, redirect
import mysql.connector

app = Flask(__name__)

# IPs de los nodos
nodos = {
    'pago': '192.168.10.136',
    'usuario': '192.168.10.136',
    'planEntrenamiento': '192.168.10.136',
    'membresia': '192.168.10.136'
}

def get_connection(tabla):
    return mysql.connector.connect(
        host=nodos[tabla],
        user='admin',
        password='admin123',
        database='ignaciopachuca'
    )

@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        tabla = request.form['tabla']
        try:
            conn = get_connection(tabla)
            cursor = conn.cursor()

            if tabla == 'pago':
                fecha = request.form['fecha']
                total = request.form['total']
                metodo = request.form['metodo']
                id_membresia = request.form['id_membresia']
                cursor.execute("""
                    INSERT INTO pago (fecha, total, metodo, idMembresia)
                    VALUES (%s, %s, %s, %s)
                """, (fecha, total, metodo, id_membresia))

            elif tabla == 'usuario':
                nombre = request.form['nombre']
                email = request.form['email']
                telefono = request.form['telefono']
                numEmergencia = request.form['numEmergencia']
                idPlan = request.form['idPlan']
                cursor.execute("""
                    INSERT INTO usuario (nombre, email, telefono, numEmergencia, idPlan)
                    VALUES (%s, %s, %s, %s, %s)
                """, (nombre, email, telefono, numEmergencia, idPlan))

            elif tabla == 'planEntrenamiento':
                fechaInicio = request.form['fechaInicio']
                fechaFin = request.form['fechaFin']
                rutina = request.form['rutina']
                cursor.execute("""
                    INSERT INTO planEntrenamiento (fechaInicio, fechaFin, rutina)
                    VALUES (%s, %s, %s)
                """, (fechaInicio, fechaFin, rutina))

            elif tabla == 'membresia':
                tipo = request.form['tipo']
                fechaInicio = request.form['fechaInicioM']
                fechaFin = request.form['fechaFinM']
                costo = request.form['costo']
                estado = request.form['estado']
                idUsuario = request.form['idUsuario']
```

```
                cursor.execute("""
                    INSERT INTO membresia (tipo, fechaInicio, fechaFin, costo, estado,
    idUsuario)
                    VALUES (%s, %s, %s, %s, %s, %s)
                """, (tipo, fechaInicio, fechaFin, costo, estado, idUsuario))

            conn.commit()
            cursor.close()
            conn.close()
            return redirect('/')
        except Exception as e:
            return f"Error: {str(e)}"
    return render_template('index.html')

if __name__ == '__main__':
    app.run(debug=True)
```

Para usar un equipo como host para la pagina web donde se haran las inserciones de registros, se hizo mediante el uso del comando "python app.py". El comando inicializa el servidor donde se hosteara la web.

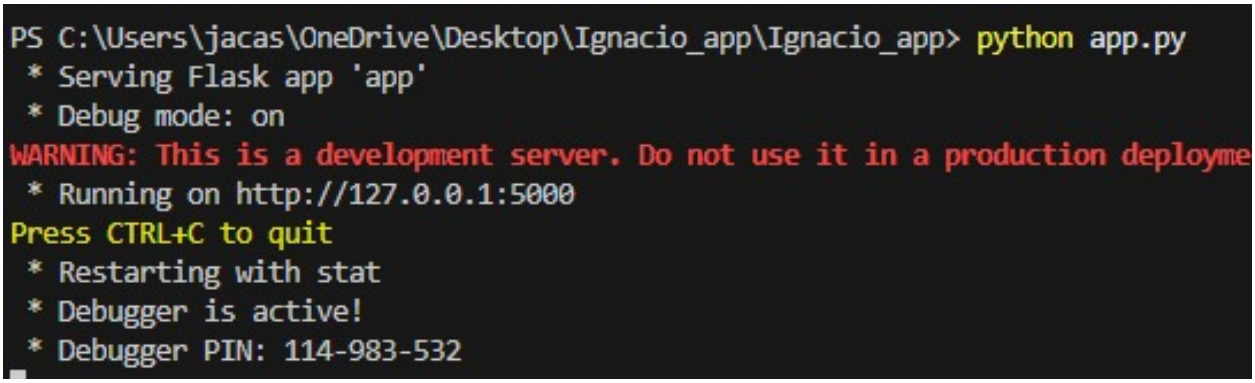

Figure 15: Comando python app.py

Desarrollo del codigo .html

```
<!DOCTYPE html>
<html>
<head>
    <title>Registro Distribuido</title>
</head>
<body>
    <h1>Registro de Entidades</h1>
    <form method="POST">
        <label>Selecciona entidad:</label>
        <select name="tabla" onchange="mostrarFormulario(this.value)">
            <option value="">-- Selecciona --</option>
            <option value="pago">Pago</option>
            <option value="usuario">Usuario</option>
            <option value="planEntrenamiento">Plan Entrenamiento</option>
            <option value="membresia">Membres a</option>
        </select>

        <div id="formularioPago" style="display:none;">
            <h3>Formulario Pago</h3>
            Fecha: <input type="date" name="fecha"><br>
            Total: <input type="number" name="total"><br>
            M todo: <input type="text" name="metodo"><br>
            ID Membres a: <input type="number" name="id_membresia"><br>
        </div>

        <div id="formularioUsuario" style="display:none;">
            <h3>Formulario Usuario</h3>
            Nombre: <input type="text" name="nombre"><br>
            Email: <input type="email" name="email"><br>
            Tel fono: <input type="text" name="telefono"><br>
            Emergencia: <input type="text" name="numEmergencia"><br>
            ID Plan: <input type="number" name="idPlan"><br>
        </div>

        <div id="formularioPlan" style="display:none;">
            <h3>Formulario Plan Entrenamiento</h3>
            Fecha Inicio: <input type="date" name="fechaInicio"><br>
            Fecha Fin: <input type="date" name="fechaFin"><br>
            Rutina: <input type="text" name="rutina"><br>
        </div>

        <div id="formularioMembresia" style="display:none;">
            <h3>Formulario Membres a</h3>
            Tipo: <input type="text" name="tipo"><br>
            Fecha Inicio: <input type="date" name="fechaInicioM"><br>
            Fecha Fin: <input type="date" name="fechaFinM"><br>
            Costo: <input type="number" name="costo"><br>
            Estado: <input type="text" name="estado"><br>
            ID Usuario: <input type="number" name="idUsuario"><br>
        </div>

        <br><button type="submit">Registrar</button>
    </form>

    <script>
        function mostrarFormulario(tabla) {
            document.getElementById("formularioPago").style.display = "none";
            document.getElementById("formularioUsuario").style.display = "none";
            document.getElementById("formularioPlan").style.display = "none";
            document.getElementById("formularioMembresia").style.display = "none";

            if (tabla === "pago") document.getElementById("formularioPago").style.display =
    "block";
            if (tabla === "usuario") document.getElementById("formularioUsuario").style.
    display = "block";
```

```
            if (tabla === "planEntrenamiento") document.getElementById("formularioPlan").
    style.display = "block";
            if (tabla === "membresia") document.getElementById("formularioMembresia").style.
    display = "block";
        }
    </script>
</body>
</html>
```

5. Conclusiones

El desarrollo de este proyecto, basado en una arquitectura de bases de datos distribuidas para la gestión integral de gimnasios, ha demostrado que puede llegar a ser una herramienta util y adaptable para su uso. A través de la implementación de múltiples nodos distribuidos, se logró la inserción de registros mediante el una pagina web realizada en una red de area local.

La adopción de metodologías ágiles como SCRUM permitió un desarrollo iterativo y flexible. Aunque el proyecto enfrentó limitaciones de recursos y sobre todo de tiempo, se sentaron las bases para futuras expansiones y posibles mejoras, como la integración de mas sucursales de ser necesario y también el poder ver el contenido de cualquier base de datos desde la propia pagina web.

En conclusión, este sistema no solo cumple con los objetivos iniciales y más basicos, sino que también establece un marco robusto para la escalabilidad del proyecto mismo. La arquitectura distribuida implementada asegura que el sistema pueda crecer junto con el negocio, adaptándose a nuevas demandas y tecnologías emergentes en el sector fitness.

Referencias Bibliográficas

References

[1] Grabowska, S.; Saniuk, S. (**2022**). Business models in the industry 4.0 environment—results of web of science bibliometric analysis. *J. Open Innov. Technol. Mark. Complex*, *8(1)*, 19.

[2] Tanenbaum, A.S.; Van Steen, M. (**2014**). *Distributed Systems: Principles and Paradigms*. Pearson Education.

[3] Bernstein, P.A.; Hadzilacos, V.; Goodman, N. (**1987**). *Concurrency Control and Recovery in Database Systems*. Addison-Wesley.

[4] Ongaro, D.; Ousterhout, J. (**2014**). In Search of an Understandable Consensus Algorithm. *Proceedings of the USENIX Annual Technical Conference*, 305–319.

[5] Schwaber, K.; Beedle, M. (**2002**). *Agile Software Development with Scrum*. Prentice Hall.

[6] Corbett, J.C.; Dean, J.; Epstein, M.; et al. (**2013**). Spanner: Google's Globally-Distributed Database. *ACM Transactions on Computer Systems*, *31(3)*, 8.

[7] Stonebraker, M. (**2010**). SQL Databases v. NoSQL Databases. *Communications of the ACM*, *53(4)*, 10–11.