GESTIÓN DE GIMNASIOS | 2024

Sistema Distribuido

SQL-Slayers

Barón Salinas Mauricio Valentín

Cristóbal Silverio Cristian

Gómez Daniel Aimar Jair

Hernández Reyes Magaly

Sánchez Carrasco Monserrat

Ramírez Suárez Gerardo

Índice

Definición de Proyecto

El proyecto en cuestión plantea el desarrollo de una base de datos para gestionar de manera eficiente las operaciones de una serie de gimnasios. Su enfoque principal está en la optimización de la administración diaria y en la mejora de la experiencia de los usuarios a través de un sistema que maneje de manera integral la información de membresías, pagos, empleados y el inventario de productos o equipos. El propósito final de este sistema es garantizar que todas las áreas operativas del gimnasio sean gestionadas de manera eficiente, con la capacidad de escalar a medida que el negocio crezca.

Se destacan cinco objetivos específicos que buscan cubrir aspectos claves del funcionamiento del gimnasio:

1. **Distribución geográfica** de los datos para gestionar múltiples sedes.
2. **Sincronización y consistencia** de la información entre esas sedes.
3. **Escalabilidad** del sistema para manejar el crecimiento del negocio.
4. **Implementación** de medidas de seguridad para proteger los datos.
5. **Garantizar la resiliencia y recuperación** en caso de desastres.

A su vez, el proyecto identifica algunas limitaciones, tales como la latencia de la red entre las distintas ubicaciones, la complejidad en la gestión de un sistema distribuido, los costos de infraestructura asociados, y posibles problemas de sincronización en redes de alta latencia.

Introducción

En la actualidad, la eficiencia de un gimnasio depende no solo de la calidad de sus instalaciones o de su personal, si no también de la correcta gestión de los datos de sus miembros, participantes y operaciones diarias. Los gimnasios de hoy en dia manejan una enorme cantidad de información que contiene el control de membresías, el seguimiento de sus pagos e información personal de sus integrantes. Por ello, sin un sistema bien organizado es fácil perder el control sobre los aspectos antes mencionados y ocasiona un experiencia negativa en los usuarios.

Este proyecto tiene la tarea de crear una base de datos que permita la excelente gestión de un gimnasio para optimizar la administración diaria, mejorar el servicio al cliente y aumentar la eficiencia operativa.

Planteamiento del problema

El gimnasio enfrenta dificultades para gestionar eficientemente las membresías, pagos y datos de usuarios. La falta de un sistema integral provoca desorganización, pérdida de información y dificultades en el seguimiento de las transacciones, lo que puede afectar la experiencia del cliente y la rentabilidad del negocio.

Objetivo General

Desarrollar una base de datos para una serie de gimnasios, buscando una capacidad de escalabilidad, que nos permita optimizar la administración y el control de todas las áreas operativas, incluyendo gestión de membresías, el manejo de inventario de equipos y productos como suplemnetos de igual manera el seguimiento de los empleados de cada sucursal.

La base de datos nos dara la integridad y disponibilidad de la informacion en tiempo real para facilitar la toma de decisiones estratégicas, mejorar la eficiencia operativa y asegurar un control efectivo de los recursos a medida que el negocio crezca.

Objetivos Especificos

- Desarrollar una base de datos distribuida que permita el almacenamiento de la información de los clientes, entrenadores, rutinas de entrenamiento, y pagos.
- Garantizar la comunicación entre los nodos de la base de datos para que la información esté sincronizada y disponible en todo momento, independientemente de la localización geográfica de los gimnasios o sucursales.
- Desarrollar una interfaz de usuario amigable, utilizando HTML, CSS y JavaScript, que permita a los usuarios acceder, visualizar y modificar datos.

Justificación

La necesidad de desarrollar una base de datos robusta para la gestión de un gimnasio surge de la creciente complejidad en la administración de los datos relacionados con los usuarios, sus membresías, los pagos y el manejo de inventario. En el contexto actual, donde la digitalización de los servicios se ha convertido en un estándar, contar con un sistema de gestión que permita optimizar las operaciones diarias no solo es una ventaja competitiva, sino también una necesidad fundamental para la sostenibilidad y crecimiento de un gimnasio.

Uno de los problemas principales que enfrentan los gimnasios es la desorganización y falta de control sobre los datos de sus miembros y transacciones financieras. La ausencia de un sistema cen-

tralizado de información puede dar lugar a la pérdida de datos, dificultades en la trazabilidad de las operaciones y errores en la gestión de inventarios, lo que puede perjudicar la experiencia del cliente y reducir la rentabilidad del negocio.

Por lo tanto, la implementación de una base de datos no solo resolvería estos problemas, sino que también ofrecería una serie de beneficios adicionales. Por ejemplo, permitiría un acceso rápido y seguro a la información en tiempo real, lo que facilitaría la toma de decisiones informadas. Además, la posibilidad de contar con un sistema que distribuya los datos geográficamente en múltiples ubicaciones permitiría gestionar de manera efectiva las distintas sedes de un gimnasio, asegurando que toda la información esté sincronizada y disponible para los administradores y usuarios en cualquier momento y lugar.

Asimismo, la escalabilidad del sistema es un aspecto crítico en este proyecto, ya que a medida que el gimnasio crezca y aumente la cantidad de miembros, empleados y equipos, la base de datos debe ser capaz de manejar este crecimiento sin comprometer la eficiencia operativa. La seguridad también es un factor esencial, especialmente cuando se trata de la protección de información personal y financiera de los usuarios. Las medidas de seguridad que se implementarán estarán destinadas a prevenir accesos no autorizados y garantizar la integridad de los datos.

Otro punto a considerar es la resiliencia del sistema. Dado que un gimnasio maneja información crítica de sus operaciones, es indispensable que la base de datos esté preparada para recuperar la información de manera rápida y efectiva en caso de cualquier eventualidad, como fallos técnicos o desastres naturales. Esto asegurará la continuidad de las operaciones sin afectar el servicio al cliente.

Alcances y Limitaciones

Alcances

1. **Distribución Geográfica de Datos**: Implementación de una base de datos distribuida que permita almacenar y gestionar datos en múltiples ubicaciones físicas —en nuestro caso, las sedes de gimnasio gestionadas por el cliente— para mejorar la disponibilidad y la redundancia.
2. **Sincronización y Consistencia de Datos**: Desarrollo de mecanismos que garantizen que los datos sean consistentes entre las distintas ubicaciones y que cualquier actualización se propague adecuadamente.
3. **Escalabilidad**: Diseño de la base de datos para que pueda escalar horizontalmente, añadiendo más nodos según sea necesario para manejar el aumento del volumen de datos y la carga de trabajo.
4. **Seguridad de Datos**: Implementación de medidas de seguridad que protejan los datos distribuidos contra accesos no autorizados y posibles ataques cibernéticos.
5. **Resiliencia y Recuperación ante Desastres**: Establecemiento de procedimientos y tecnologías para asegurar la recuperación rápida y efectiva en caso de fallos o desastres.

Limitaciones

1. **Latencia de Red**: La latencia de la red entre los nodos puede afectar el rendimiento del sistema, especialmente si los nodos están ubicados en regiones geográficas distantes.
2. **Complejidad de Gestión**: La administración y el mantenimiento de una base de datos distribuida pueden ser significativamente más complejos en comparación con una base de datos centralizada.
3. **Costos de Infraestructura**: Los costos asociados con el hardware, el software y la red para soportar una base de datos distribuida pueden ser elevados.
4. **Consistencia Eventual**: Algunas arquitecturas distribuidas optan por la consistencia eventual, lo que puede llevar a periodos en los que los datos no están sincronizados en todos los nodos.
5. **Problemas de Sincronización**: La sincronización de datos entre nodos puede enfrentar problemas, especialmente en redes con alta latencia o en escenarios de particionamiento de red.

Marco teórico

Nosotros enfocamos un marco teorico que influye en el proceso estructurado para guiar el desarrollo desde los elementos de entrada(información recopilada), hasta las herramientas que se utilizaran para la elaboración de ellas. Asi facilitando su implementación en las diferentes etapas del proceso.

Pressman promueve un enfoque iterativo e incrementalen e desarrollo de software, en este caso lo utilizaremos en el desarrollo de nuestra base de datos distribuida en base al proyecto del GYM, esto para que se contruya y mejore partes del proyecto de manera gradual. Esto tambien tomando en cuenta los aspectos de gestión de proyectos, estimación, planificación y control, asi tambien la detección temprana de los defectos y su corrección inmediata.

SCRUM

SCRUM es un marco ágil para la gestión de proyectos, especialmente en el desarrollo de software. Se basa en la colaboración, la autoorganización y la entrega incremental de productos. SCRUM utiliza roles específicos (como el Product Owner, Scrum Master y el equipo de desarrollo), así como eventos estructurados (como sprints, reuniones diarias y revisiones de sprint) para facilitar la planificación, ejecución y mejora continua del trabajo. Su objetivo es aumentar la flexibilidad y la adaptación a cambios en los requisitos del proyecto.

Un equipo SCRUM es un pequeño equipo de personas, que consta de:

- un Scrum Master,
- un propietario de producto (Product Owner) y
- desarrolladores.

Dentro de un equipo de Scrum, no hay sub-equipos ni jerarquías. Es una unidad cohesionada de profesionales enfocada en un objetivo a la vez, el objetivo del Producto.

Retraso: lista de prioridades de los requerimientos o características del proyecto que dan al cliente un valor del negocio. Es posible agregar en cualquier momento otros aspectos al retraso (ésta es la forma en la que se introducen los cambios). El gerente del proyecto evalúa el retraso y actualiza las prioridades según se requiera. Sprints: consiste en unidades de trabajo que se necesitan para alcanzar un requerimiento definido en el retraso que debe ajustarse en una caja de tiempo predefinida (lo común son 30 días). Durante el sprint no se introducen cambios (por ejemplo, aspectos del trabajo

retrasado). Así, el sprint permite a los miembros del equipo trabajar en un ambiente de corto plazo pero estable. Reuniones Scrum: son reuniones breves (de 15 minutos, por lo general) que el equipo Scrum efectúa a diario. Hay tres preguntas clave que se pide que respondan todos los miembros del equipo.

- ¿Qué hiciste desde la última reunión del equipo?
- ¿Qué obstáculos estás encontrando?
- ¿Qué planeas hacer mientras llega la siguiente reunión del equipo? Un líder del equipo, llamado maestro Scrum, dirige la junta y evalúa las respuestas de cada persona.

Metodología de Análisis

Guía el proceso de recopilación y evaluación de información para tomar decisiones informadas y resolver problemas, a partir de estrategias analíticas de la información preexistente (documentos, personal y necesidades).

Casos de Uso

Analiza el comportamiento del sistema en distintas condiciones en las que el sistema responde a una petición de alguno de sus participantes. En esencia, un caso de uso narra una historia estilizada sobre cómo interactúa un usuario final (que tiene cierto número de roles posibles) con el sistema en circunstancias específicas. Un caso de uso ilustra el software o sistema desde el punto de vista del usuario final.

Historias de Usuario

Las historias de usuario constan de un texto narrativo, un lineamiento de tareas o interacciones, una descripción basada en un formato o una representación diagramática, sin importar su forma. Suelen ser representaciones de características, donde se tiene en cuenta una necesidad concreta desde el punto de vista de un usuario bien definido.

Entrevista de requerimientos

Éste es el enfoque más directo, los miembros del equipo de software se reúnen con los usuarios para entender mejor sus necesidades, motivaciones, cultura laboral y una multitud de aspectos adicionales. Esto se logra en reuniones individuales o a través de grupos de enfoque.

Metodología de Diseño

Guía el proceso creativo y la resolución de problemas, desde la evaluación de diversas posibles soluciones. Su salida debe consistir en un activo potencial para el desarrollo del sistema, a partir de la generación de una estructura más sólida para el entendimiento del proyecto.

Definición del Problema

- Definición clara del problema a abordar
- Identificación de las necesidades y expectativas de los usuarios
- Definición de los objetivos del diseño

Desarrollo de Conceptos

- Proposición de ideas.
- Selección de las ideas más prometedoras
- Desarrollo en conceptos más concretos.
- Esbozo de prototipos iniciales o maquetas para visualizar y evaluar las soluciones propuestas.

Evaluación y Pruebas

- Los conceptos desarrollados se prueban y evalúan, a menudo mediante feedback de usuarios.
- Se identifican fallos y áreas de mejora, permitiendo iteraciones en el diseño.

Metodología de Desarrollo

Proporciona un marco estructurado en el que ya se puede planificar, ejecutar y gestionar el proyecto de manera eficiente.

Planificación

- Definición del alcance del proyecto
- Identificación de los objetivos
- Elaboración de un cronograma
- Definición de los recursos necesarios
- Asignación de roles y responsabilidades al equipo

Pruebas

- Desarrollo de pruebas para identificar errores y garantizar que el producto cumple con los requisitos establecidos, asegurando que el software sea robusto y confiable:
  - Pruebas unitarias,
  - de integración y
  - de aceptación del usuario.

Implementación

Una vez que el software ha sido probado y validado, se despliega en el entorno de producción. Esto puede incluir la migración de datos y la capacitación de usuarios, asegurando una transición fluida.

Mantenimiento

Después de la implementación, se requiere un seguimiento continuo para corregir errores, realizar actualizaciones y mejorar el sistema en función del feedback de los usuarios. Este componente es esencial para asegurar la longevidad y la relevancia del producto.

Evaluación y Mejora Continua

Se lleva a cabo una revisión del proceso y los resultados del proyecto. Esta evaluación permite identificar lecciones aprendidas y oportunidades de mejora, lo que contribuye a la optimización de futuras metodologías de desarrollo.

Análisis

Definición del Alcance del Proyecto

- **Objetivo**: Crear un sistema que gestione las operaciones de un gimnasio, incluyendo el registro de miembros, gestión de pagos, control administrativo de empleados y reportes de ganancias e inventario de suplementos.
- **Partes interesadas**: Administradores, miembros y empleados.

Entrevista con el Administrador

Visión de negocio

Entrevistador: Buenas tardes. Gracias por tomarse el tiempo para esta entrevista. Para empezar, ¿puede darme una visión general de su negocio y qué es lo que espera lograr con este nuevo sistema?

Administrador: Buenas tardes. Claro, tengo varios gimnasios en diferentes ubicaciones, y actualmente gestionamos todo manualmente o con sistemas muy básicos. Lo que realmente necesito es un sistema que me permita gestionar las inscripciones de los miembros, el seguimiento de su actividad y productividad, y también la administración de los empleados y horarios.

Entrevistador: Entiendo. Vamos a desglosar esto en partes. Primero, ¿puede describir cómo maneja actualmente el registro de miembros y qué aspectos le gustaría mejorar?

Administrador: En este momento, usamos un sistema de hojas de cálculo para el registro de nuevos miembros y el seguimiento de pagos. Esto es bastante ineficiente y propenso a errores. Me gustaría que el nuevo sistema permitiera registrar automáticamente nuevos miembros y gestionar sus pagos.

Registro de miembros y pagos

Entrevistador: Perfecto. Para el registro de miembros, ¿qué información específica necesita capturar?

Administrador: Necesitamos capturar información básica como:

- nombre,
- número de teléfono principal y
- número de teléfono alternativo.

También necesitamos registrar:

- el tipo de membresía,
- la fecha de inicio y
- los pagos realizados.

Seguimiento Administrativo

Entrevistador: En relación a la administración de empleados y horarios, ¿qué funcionalidades específicas está buscando?

Administrador: Necesito un módulo para gestionar:

- los horarios de los empleados, incluyendo:
  - las asignaciones de turno y
  - las solicitudes de vacaciones.

También sería útil poder hacer un seguimiento de:

- sus horas trabajadas y
- sus pagos.

Gestión de suplementos

Entrevistador: Perfecto. Para finalizar, ¿hay algún otro requerimiento o aspecto específico que no hayamos cubierto y que sea importante para usted?

Administrador: Sí, actualmente también servimos como distribuidor de suplementos, por lo que me gustaría que el sistema me permita:

- llevar un control de los suplementos vendidos y las ganancias, y
- el inventario de los suplementos que tenemos disponibles para venta.

Últimas palabras

Entrevistador: Entendido. Muchas gracias por la información detallada. Con estos requisitos, podemos empezar a definir el alcance del proyecto y trabajar en el diseño del sistema. ¿Hay algo más que quiera agregar antes de que terminemos?

Administrador: No, eso es todo. Gracias a usted.

Entrevistador: De nada. Le mantendremos informado sobre el avance del proyecto. Que tenga un buen día.

Administrador: Igualmente. Que tenga un buen día.

Analisis de Requerimientos

Funcionales

- **Gestión de clientes**: Registros, renovación y cancelación de membresías.
- **Pagos**: Procesamiento de pagos.
- **Gestión de empleados**: Registros, asisnación de rol y actividades a realizar y evaluación de desempeño.
- **Administración del inventario**: Control de cantidades existentes y vendidas de suplementos.

No Funcionales

- **Escalabilidad**: Capacidad del sistema para manejar un número creciente del usuario.
- **Disponibilidad**: Garantizar que el sistema esté disponible 24/7.
- **Seguridad**: Protección de datos personales y financieros de los miembros.
- **Rendimiento**: Respuesta rápida del sistema incluso en horas pico.

Historias de Usuario

Registro de Miembros

Como **administrador**, quiero **poder registrar nuevos miembros en el sistema**, para **automatizar el proceso de inscripción y tener un registro preciso de los datos de los miembros**.

- El sistema debe permitir ingresar información básica del miembro: nombre y número de teléfono
- El sistema debe registrar el tipo de membresía, fecha de inicio, fecha final y pagos realizados.

Gestión de Pagos

Como **administrador**, quiero **poder gestionar los pagos de los miembros dentro del sistema**, para **asegurarme de que los pagos se registren de manera precisa y automática**.

- El sistema debe registrar pagos realizados por los miembros.
- El sistema debe permitir consultar el estado de cuenta y el historial de pagos.

Gestión de Empleados y Horarios

Como **administrador**, quiero **poder gestionar los horarios de los empleados, incluyendo turnos**, para **llevar un orden y supervisar las actividades de los trabajadores**.

- El sistema debe hacer seguimiento de las horas trabajadas por cada empleado.
- El sistema debe gestionar los pagos de los empleados.

Gestión de inventario de suplementos

Como **administrador**, quiero **poder llevar un registro de la venta e inventario de suplementos** para **monitorear las ganancias y la disponibilidad de éstos**.

- El sistema debe permitir controlar el inventario de mis suplementos
- El sistema debe registrar las ganancias y precios de los suplementos.

Diseño

Arquitectura de Sistema de Base de Datos Distribuido

Componentes

- **Nodos:** Máquinas físicas o virtuales que participan en el sistema.
- **Red:** Infraestructura que conecta los nodos.
- **Gestor de Bases de Datos (DBMS):** Software que gestiona la base de datos en cada nodo. Puede ser uno solo o varios.

Tipos de Arquitectura

- **Arquitectura de Base de Datos Distribuida Homogénea:**
  - Todos los nodos utilizan el mismo DBMS y están en la misma red.
  - La gestión es más sencilla, pero requiere empezar desde cero.
- **Arquitectura de Base de Datos Distribuida Heterogénea:**
  - Los nodos pueden usar diferentes DBMS y sistemas operativos, y estar en distintas redes.
  - Ofrece flexibilidad, pero complica la gestión y la integración de datos.

Modelos

- **Replicación:** Los datos se copian en varios nodos para mejorar la disponibilidad y la recuperación ante fallos.
- **Fragmentación:** Los datos se dividen en fragmentos que se distribuyen entre los nodos. Puede ser horizontal (tuplas) o vertical (atributos).

Análisis de entidades

Entidades y atributos

- **Gimnasio**

  - idGimnasio (Llave primaria)
  - Dirección
  - Teléfono
  - Horario de Apertura
  - Horario de Cierre

- **Cliente**
  - idCliente (Llave primaria)
  - Nombre
  - Apellido Paterno
  - Apellido Materno
  - Teléfono

- **Membresía**
  - idMembresía (Llave primaria)
  - Tipo (Mensual, Anual, etc.)
  - Precio
  - Fecha de inicio
  - Fecha de fin
  - idCliente (Llave foránea)

- **Empleado**
  - idEmpleado (Llave primaria)
  - Nombre
  - Apellido Paterno
  - Apellido Materno
  - Tipo de empleado
  - Sueldo
  - Teléfono
  - idGimnasio (Llave foránea)

- **Suplemento**
  - idSuplemento (Llave primaria)
  - Nombre
  - Marca
  - Tipo
  - Presentación (Imagen)
  - Precio unitario

- **Venta**
  - Monto
  - Tipo

  - Fecha
  - Hora

- **Inventario-Gimnasio** (Intermediaria: existencia de un suplemento en un determinado gimnasio)
  - idInventarioGimnasio (Llave primaria)
  - Inventario
  - idSuplemento (Llave foránea)
  - idGimnasio (Llave foránea)

- **Detalle-Venta** (Intermediaria: una por cada elemento en una venta)
  - idDetalleVenta (Llave primaria)
  - idInventarioGimnasio (Llave foránea)
  - idMembresía (Llave foránea)
  - idVenta (Llave foránea)

- **idInfoEmpleado** (Distribuida)
  - idInfoEmpleado (Llave primaria)
  - idEmpleado (Llave foránea)
  - idGimnasio (Llave foránea)

- **idInfoMembresía** (Distribuida)
  - idInfoMembresía (Llave primaria)
  - idMembresía (Llave foránea)
  - idGimnasio (Llave foránea)

Relaciones entre Entidades

	Gimnasio	Empleado	Suplemento	Inventario	Membresía	Venta	Cliente
Gimnasio		X		X	X		
Empleado	X						
Suplemento				X			
Inventario	X		X			X	
Membresía	X					X	X
Venta				X	X		
Cliente					X		

Modelo Relacional

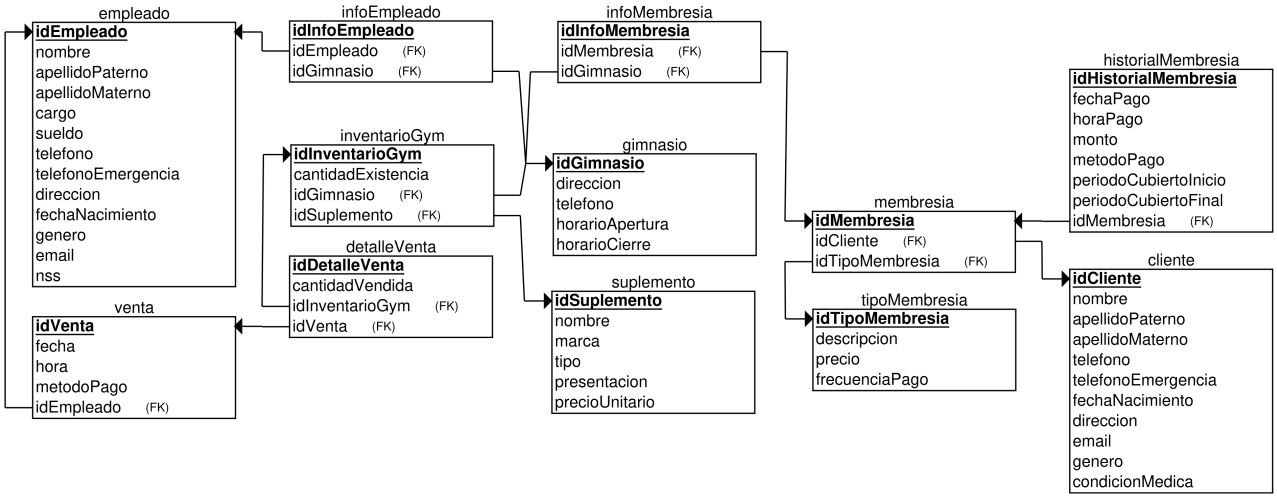

Figura 1 – Modelo Relacional

Desarrollo

Implementacion del SBDD

Estructura de backend del proyecto

Para implementar la estructura de datos del proyecto de base de datos distribuida en un entorno con Node.js, Docker, Express, y Mongoose, necesitaremos crear una arquitectura que permita gestionar múltiples entidades (como gimnasios, usuarios, empleados, etc.) y garantizar que los datos sean accesibles y sincronizados a través de las distintas sedes de manera eficiente.

```
./
├── package.json
├── package-lock.json
├── README.md
└── src
    ├── Controllers
    │   ├── cliente.Controller.js
    │   ├── detalleVenta.Controller.js
    │   ├── empleado.Controller.js
    │   ├── historialMembresia.Controller.js
    │   ├── info.txt
    │   ├── inventarioGym.Controller.js
    │   ├── membresia.Controller.js
    │   └── venta.Controller.js
    ├── info.txt
    ├── Middleware
    │   └── info.txt
    ├── Models
    │   ├── cliente.js
    │   ├── detalleVenta.js
    │   ├── empleado.js
    │   ├── gimnasio.js
    │   ├── historialMembresia.js
    │   ├── inventarioGym.js
```

```
    │   ├── membresia.js
    │   ├── suplumento.js
    │   ├── tipoMembresia.js
    │   └── venta.js
    ├── Routers
    │   ├── cliente.Router.js
    │   ├── detalleVenta.Router.js
    │   ├── empleado.Router.js
    │   ├── historialMembresia.Router.js
    │   ├── inventarioGym.Routers.js
    │   ├── membresia.router.js
    │   └── venta.Router.js
    └── server
        ├── app.js
        └── info.txt

7 directories, 32 files
```

Entidades como modelos

Cada archivo dentro de la carpeta `models` define un esquema de Mongoose que refleja las entidades del gimnasio, como se mencionó en el esquema conceptual.

Por ejemplo, tenemos en `gimnasio.js` la siguiente implementación:

```
import mongoose from "mongoose";
const { Schema, model } = mongoose;

const gimnasioSchema = new Schema({
  direccion: { type: String, required: true },
  telefono: { type: String, required: true },
  horarioApertura: { type: String, required: true },
  horarioCierre: { type: String, required: true },
  empleados: [{ type: Schema.Types.ObjectId, ref: "Empleado" }], // Relación
   ↪  inversa
});

export default model("Gimnasio", gimnasioSchema);
```

Funciones como controladores

Los controladores en la carpeta `Controllers` gestionan las operaciones de las rutas, ejecutando la lógica para manipular los datos en la base de datos y respondiendo a las peticiones HTTP.

Los controladores son útiles para concentrar todas las operaciones permitidas en la base de datos, gestionando de manera segura el acceso y manipulación de los datos.

Para implementar el de las ventas, por ejemplo, importamos la entidad y exportamos a partir de ella las funciones **asíncronas** correspondientes:

```
import venta from "../Models/venta.js"; // Para usar el modelo

// Funcion para registrar una nueva venta
export const registrarVenta = async (req, res) => {
  console.log("Registrando Nueva venta");
  const { fecha, hora, metodoPago, empleado, detallesVenta } = req.body; //
   ↪  Extraemos los datos a almacenar de la peticion

  // Vamos a validar de que estamos obteniendo todos los datos
  if (!fecha || !hora || !metodoPago || !empleado || !detallesVenta) {
    return res.status(400).json({
      message: "Todos los datos tienen que ser llenados...",
    });
  } // Si recibimos todos los datos

  const nuevaVenta = new venta({
    // Creamos la nueva venta
    fecha,
    hora,
    metodoPago,
    empleado,
    detallesVenta,
  });
  // Lo guardamos en nuestra db
  try {
    await nuevaVenta.save();
    res.status(201).json(nuevaVenta); // Responde de manera exitosa y devolvemos
 ↪  la venta que se registro
  } catch (error) {
    res.status(400).json({ message: error.message }); // Si algo falla, mandamos
 ↪  el error
  }
};
```

Relaciones como rutas de Express

Cada archivo en la carpeta `routers` define las rutas RESTful para interactuar con los modelos. Las rutas recibirán solicitudes HTTP (GET, POST, PUT, DELETE) y las gestionarán mediante los controladores.

En el caso de `cliente.Router.js`, se importa el `router` de Express, y los controladores de la entidad:

```
import { Router } from "express"; // Para poder utilizar Rutas
import {
  crearCliente,
  actualizarCliente,
  obtenerClientes,
  obtenerClientesID,
  eliminarCliente,
} from "../Controllers/cliente.Controller.js"; // Traemos el controlador

const router = Router(); // Para inicializar las rutas

router.get("/clientes", obtenerClientes); // definimos ruta para obtener all
router.get("/clientes/:id", obtenerClientesID); // para obtener por id
router.post("/clientes", crearCliente); // para crear
router.put("/clientes/:id", actualizarCliente); // para actualizar
router.delete("/clientes/:id", eliminarCliente); // para eliminar

export default router;
```

Configuración de la base de datos

El archivo `server/app.js` configura el servidor Express y conecta todas las rutas con el backend.

Para la configuración, necesitaremos algunas variables de entorno que `dotenv` leerá para configurar nuestra base de datos y la conexión a ésta. Por ejemplo, `MONGO_URL` debe tener un valor válido en nuestro archivo `.env` en la raíz del proyecto.

Una implementación posible de la configuración sería la siguiente:

```
import dotenv from "dotenv";
dotenv.config();
import bodyParser from "body-parser"; // Para poder pasar los datos a un formato
 ↪  más manejable
import express from "express"; // importamos los mudulos de express
import mongoose from "mongoose"; // Importar mongoose para conectar a MongoDB

import empleadoRouter from "../Routers/empleado.Router.js"; // Ahora incluye la
 ↪  extensión .js
import ventaRouter from "../Routers/venta.Router.js";
import detalleVentaRouter from "../Routers/detalleVenta.Router.js";
import clienteRouter from "../Routers/cliente.Router.js";
import membresiaRouter from "../Routers/membresia.router.js";
import inventarioGymRouter from "../Routers/inventarioGym.Routers.js";
import historialMembresia from "../Routers/historialMembresia.Router.js";
```

```
const app = express();
app.use(express.json());
app.use(bodyParser.json()); // Parseamos el body a formato JSON

// Conectar a MongoDB
mongoose.connect(process.env.MONGO_URL, {
  dbName: process.env.MONGO_DB_NAME,
});

// Creamos las rutas
app.use("/api", empleadoRouter);
app.use("/api", ventaRouter);
app.use("/api", detalleVentaRouter);
app.use("/api", clienteRouter);
app.use("/api", membresiaRouter);
app.use("/api", inventarioGymRouter);
app.use("/api", historialMembresia);

// Conexión a la base de datos de MongoDB
const db = mongoose.connection;
db.on("error", console.error.bind(console, "Error en la conexión a MongoDB:"));
db.once("open", () => {
  console.log("Conectado a MongoDB");
});

app.listen(process.env.PORT, () => {
  console.log(`Servidor escuchando en el puerto ${process.env.PORT}.`);
});

console.log("MONGO_URL:", process.env.MONGO_URL);
console.log("MONGO_DB_NAME:", process.env.MONGO_DB_NAME);
console.log("PORT:", process.env.PORT);
```

Configuración del contenedor de Docker

Para nuestro caso de estudio, usaremos Docker para obtener una mayor portabilidad, escalabilidad, aislamiento y mejor gestión de los recursos y dependencias de la base de datos.

Para configurar el contenedor a usar, generaremos un archivo `docker-compose.yml`, un archivo de configuración utilizado por Docker Compose, que facilita la definición, ejecución y gestión de contenedores de Docker.

Una posible implementación de este archivo es:

```
version: "3.8"
```

```yaml
services:
  mongo-db:
    image: mongo:8.0.1
    restart: always
    environment:
      MONGO_INITDB_ROOT_USERNAME: ${MONGO_USER}
      MONGO_INITDB_ROOT_PASSWORD: ${MONGO_PASS}
    volumes:
      - ./mongo:/data/db
    ports:
      - 27017:27017
```

Implementacion De Interfaces de Usuario

Estructura de `frontend` del proyecto

Para un desarrollo eficaz del sistema para gestionar gimnasios, se usaron diversas páginas HTML, hojas de estilo CSS, scripts JavaScript y recursos como imágenes, logos y otros archivos multimedia.

```
./
├── 6.html
├── 7.html
├── 8.html
├── css
│   ├── 6.css
│   ├── 7.css
│   ├── 8.css
│   ├── bootstrap-4.4.1.0.css
│   ├── bootstrap-4.4.1.css
│   ├── Empleado.css
│   ├── EmpleadoMostrar.css
│   ├── Index.css
│   ├── insetar imagen.css
│   ├── InterfazGe.css
│   ├── password.css
│   ├── PuntoV.css
│   └── style.css
├── eCommerceAssets
│   ├── images
│   │   └── 200x200.png
```

```
│   └── styles
│       └── eCommerceStyle.css
├── EmpleadoMostrar.html
├── estilos.css
├── Ganancia.html
├── Imagenes
│   ├── add-image.png
│   ├── Fondo1.jpg
│   ├── Fondo2.png
│   ├── fondo3.jpg
│   ├── fondo4.jpg
│   ├── fondo5.jpg
│   ├── gif 2.gif
│   ├── gif1.gif
│   ├── icon1.png
│   ├── icon2.png
│   ├── icon3.png
│   ├── icon4.png
│   ├── icon5.png
│   ├── Ima1.png
│   ├── ima2.png
│   ├── LOGO.png
│   ├── member.png
│   ├── miembro.png
│   ├── photo.png
│   ├── protein.png
│   ├── suplement.png
│   └── whey.png
├── Index.html
├── insertar imagen.html
├── Interfaz9.html
├── InterfazGe.html
├── js
│   ├── bootstrap-4.4.1.js
│   ├── Empleado.js
│   ├── EmpleadoMostrar.js
│   ├── index.js
│   ├── insertar imagen.js
│   ├── InterfazGe.js
│   ├── jquery-3.4.1.min.js
│   ├── popper.min.js
```

```
│   ├── punto de venta.js
│   └── script.js
├── Login.html
├── MenuAdmin.html
├── Password.html
├── Punto de Venta.html
├── README.md
└── script.js

7 directories, 63 files
```

Páginas de la aplicación (HTML)

Algunas de las páginas implementadas para frontend incluyen:

- **`Index.html`**: El archivo principal o de inicio de la aplicación.
- **`EmpleadoMostrar.html`**: Esta página muestra la información de los empleados.
- **`Ganancia.html`**: Una página muestra reportes de ganancias o estadísticas económicas del gimnasio.

Carpetas de CSS (Estilos)

La carpeta `css` contiene varios archivos CSS encargados de dar estilo a las páginas HTML. Entre ellos, encontramos:

- Archivos de **Bootstrap**, un framework CSS popular, que proporciona componentes y diseño responsivo.
- El archivo **`Index.css`**, que especifica los estilos para la página principal (index).
- El archivo **`EmpleadoMostrar.css`** que provee estilos adicionales para mostrar los datos de los empleados.

Lógica de la aplicación en JavaScript

La carpeta `js` contiene varios archivos JavaScript encargados de añadir interactividad y lógica a las páginas HTML. Entre ellos, encontramos:

- El script de **Bootstrap** para permitir el uso de sus componentes interactivos como menús desplegables, modales, etc.
- El script de **`index.js`** que permite la carga dinámica del contenido al inicio de la aplicación.

Pruebas Unitarias

Para garantizar que las funcionalidades individuales del sistema estén funcionando correctamente y sin errores, debemos recurrir a la verificación de pruebas unitarias. A través de ellas, podemos detectar y corregir posibles fallos antes de la etapa de producción.

Requisitos

Las pruebas unitarias deben cubrir los siguientes aspectos del proyecto:

1. **Validación de Formularios**: Se probará que las funciones de validación de los formularios, como los de inicio de sesión, inserción de imagen, y registro de miembros, funcionen correctamente.
2. **Funciones de Manejo de Datos**: Se probará que las funciones encargadas de manipular datos, como la creación, actualización y eliminación de empleados, se ejecuten correctamente.
3. **Funciones de Interactividad**: Se verificarán las funciones que gestionan la interactividad en las páginas, como la carga de imágenes, la navegación entre pantallas y la visualización de datos.

Implementacion

Para implementar, por ejemplo, la prueba unitaria `insertar_imagen.js` para insertar una imagen en frontend, podemos describir un caso, así como su resultado esperado. De no cumplirse, el script nos lo hará saber para actuar en concordancia.

```
describe("Validación de Formulario de Inserción de Imagen", () => {
  test("Debe mostrar un mensaje de error si el campo de imagen está vacío", () =>
  ↪  {
    const inputImagen = ""; // Simulamos un campo vacío
    const resultado = validarImagen(inputImagen); // Función que valida la imagen
    expect(resultado).toBe("Por favor, seleccione una imagen");
  });

  test("Debe permitir la carga de una imagen válida", () => {
    const inputImagen = "imagen.jpg"; // Simulamos un campo con una imagen válida
    const resultado = validarImagen(inputImagen);
    expect(resultado).toBe(true);
  });
});
```

Manual Técnico: *Instalación de Contenedor de Docker con MongoDB y Proyecto de Visual Studio Code*

Requisitos Previos

- Sistema operativo compatible con Docker (Windows, macOS o Linux)
- Visual Studio Code (VS Code) instalado
- Conocimientos básicos de Docker y VS Code

Instalación del Contenedor de Docker con MongoDB

1. Crea un nuevo archivo llamado `docker-compose.yml` en el directorio raíz de tu proyecto.
2. Copia y pega el siguiente contenido en el archivo `docker-compose.yml`:

```yaml
version: '3'

services:
  mongo:
    image: mongo
    environment:
      MONGO_INITDB_ROOT_USERNAME: mi_usuario
      MONGO_INITDB_ROOT_PASSWORD: mi_contraseña
    ports:
      - "27017:27017"
```

1. Abre una terminal o consola en el directorio donde se encuentra el archivo `docker-compose.yml`.
2. Ejecuta el comando `docker-compose up -d` para crear y iniciar el contenedor de Docker en segundo plano.
3. Verifica que el contenedor esté en ejecución ejecutando el comando `docker-compose ps`.

Conexión a la Base de Datos de MongoDB

1. Abre una terminal o consola en el directorio donde se encuentra el archivo `docker-compose.yml`.
2. Ejecuta el comando `docker-compose exec mongo mongo -u mi_usuario -p mi_contraseña` para acceder a la base de datos de MongoDB.
3. Introduce la contraseña de la base de datos cuando se te solicite.

Instalación del Proyecto de Visual Studio Code

1. Abre Visual Studio Code en el directorio donde se encuentra el archivo `docker-compose.yml`.
2. Crea un nuevo proyecto ejecutando el comando `mkdir mi_proyecto` en la terminal o consola.
3. Abre el proyecto en Visual Studio Code ejecutando el comando `code mi_proyecto`.
4. Instala las dependencias necesarias para el proyecto ejecutando el comando `npm install` o `yarn install`.
5. Configura la conexión a la base de datos de MongoDB en el archivo `settings.json` o `config.json` del proyecto.

Configuración de la Conexión a la Base de Datos de MongoDB

1. Abre el archivo `settings.json` o `config.json` del proyecto en Visual Studio Code.
2. Agrega la siguiente configuración de conexión a la base de datos de MongoDB:

```
{
  "database": {
    "host": "localhost",
    "port": 27017,
    "username": "mi_usuario",
    "password": "mi_contraseña",
    "database": "mi_base_de_datos"
  }
}
```

Guarda los cambios en el archivo de configuración.

Con esto hemos instalado un contenedor de Docker con MongoDB y un proyecto de Visual Studio Code, configurado la conexión a la base de datos de MongoDB logrando un entorno de desarrollo funcional.

Resultados y Conclusiones

Resultados

El proyecto logró cumplir con varios objetivos clave establecidos desde su inicio. A continuación, se detallan los resultados alcanzados en relación con los principales componentes del sistema:

- Gestionar múltiples sedes de gimnasios a través de una arquitectura distribuida, a partir de la implementación de un sistema de bases de datos distribuidas utilizando tecnologías como **MongoDB**.
- Garantizar la consistencia de los datos a través de las distintas ubicaciones a partir de la sincronización en tiempo real utilizando mecanismos de **replicación en MongoDB** y técnicas de consistencia eventual.

Conclusiones

El proyecto ha cumplido con sus objetivos principales, proporcionando una solución robusta, escalable y segura para gestionar de manera eficiente los gimnasios distribuidos. El sistema está preparado para afrontar el crecimiento del negocio y para ofrecer una experiencia de usuario mejorada, todo mientras garantiza la seguridad y la disponibilidad continua de los datos.