



IES Francisco de los Ríos

Alumno	Antonio Delgado Portero
Asignatura	Programación
Curso	1 DAM
Año	2024-2025
Título de la práctica (Enlace Repositorio)	<u>Proyecto 3 EVAL: Fitness360</u>

Proyecto 3 EVAL: Fitness360.....	1
1. Introducción.....	3
2. Tecnologías utilizadas.....	3
3. Arquitectura del sistema.....	4
4. Modelo de Datos.....	6
5. Casos de uso.....	8
6. Interfaces de usuario (GUI).....	9
7. Controladores.....	15
8. DAO (Data Access Object).....	16
9. Validaciones y control de errores.....	16
10. Pruebas y calidad.....	17
11. Instalación y despliegue.....	17
12. Lista de comprobación de requisitos técnicos.....	18
13. Patrones de diseño y componentes adicionales.....	31
14. Futuras mejoras.....	36
15. Backlog del proyecto.....	39
16. Conclusión.....	40
17. Aportaciones de ChatGPT.....	41

1. Introducción

Fitness360 es una aplicación de escritorio desarrollada en el lenguaje de programación Java, haciendo uso de la biblioteca JavaFX para la interfaz gráfica y JDBC para la gestión de la conexión con la base de datos MySQL. Su principal objetivo es ofrecer una solución tecnológica orientada a la gestión integral de rutinas de entrenamiento físico y dietas personalizadas para los usuarios de un gimnasio.

Esta aplicación está diseñada pensando en las necesidades tanto de los profesionales del fitness como entrenadores personales y dietistas, como de los clientes que asisten al centro. Los profesionales pueden crear, administrar y asignar planes de entrenamiento y alimentación personalizados a sus clientes, mientras que estos últimos pueden consultar, crear y seguir sus planes a través de una interfaz amigable, intuitiva y accesible.

Fitness360 busca mejorar la eficiencia en la gestión diaria de los entrenadores y dietistas, optimizando la comunicación con los usuarios y facilitando el seguimiento continuo del progreso físico de cada cliente. Además, al permitir la digitalización de datos y planes, se fomenta una mayor personalización y adaptación continua a las necesidades y evolución de cada usuario, contribuyendo así a mejorar su salud y bienestar general.

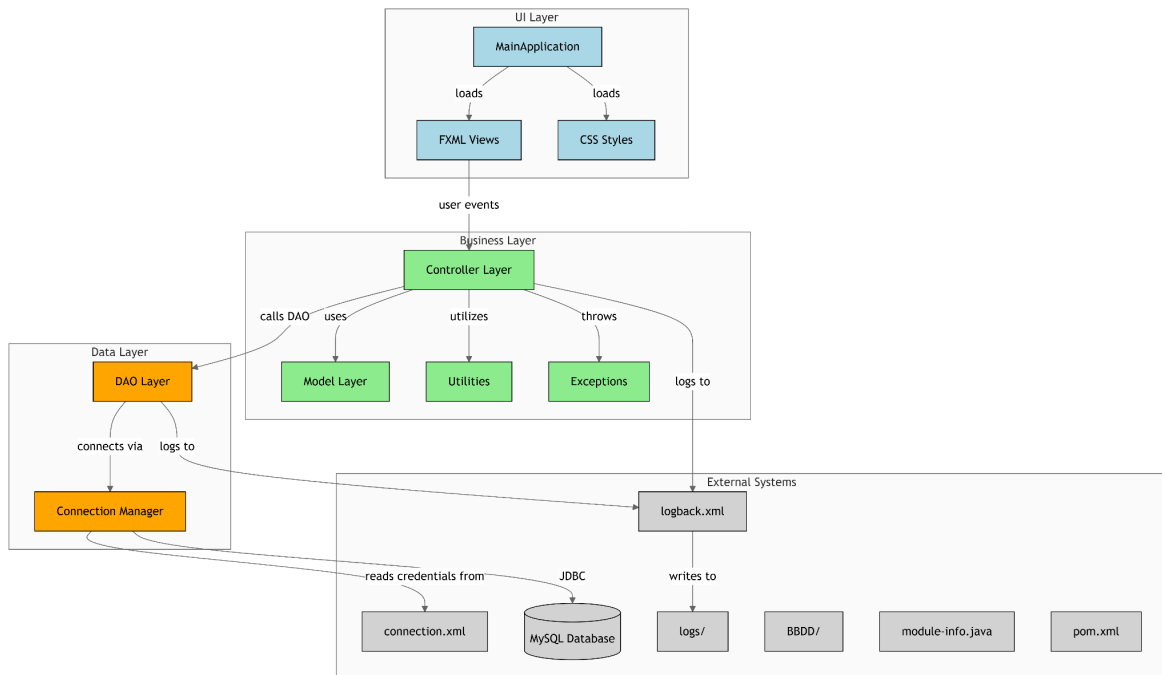
2. Tecnologías utilizadas

Para el desarrollo de Fitness360 se han empleado tecnologías modernas y robustas que aseguran estabilidad, rendimiento y facilidad de mantenimiento. A continuación, se describen las principales:

- **Java 17:** Versión actual del lenguaje Java utilizada para la programación de la lógica de negocio y la implementación de funcionalidades del sistema. La versión 17 es una versión LTS (Long-Term Support), garantizando soporte y actualizaciones a largo plazo.

- **JavaFX:** Framework para la creación de interfaces gráficas de usuario (GUI) en aplicaciones Java. Permite diseñar ventanas, formularios y controles visuales con gran flexibilidad, además de soportar un diseño declarativo mediante archivos FXML.
- **JDBC (Java Database Connectivity):** API que facilita la conexión y ejecución de operaciones SQL sobre bases de datos relacionales, en este caso con MySQL. JDBC proporciona métodos para la realización de consultas, inserciones, actualizaciones y eliminaciones de datos.
- **MySQL:** Sistema de gestión de bases de datos relacional (RDBMS) que se utiliza para almacenar toda la información relacionada con usuarios, rutinas, dietas, tarifas y progresos físicos. MySQL es una solución ampliamente utilizada por su rendimiento y escalabilidad.
- **IntelliJ IDEA:** Entorno de desarrollo integrado (IDE) utilizado para programar y depurar la aplicación. IntelliJ ofrece herramientas avanzadas para facilitar la escritura de código, gestión de proyectos, integración con control de versiones y ejecución de pruebas.

3. Arquitectura del sistema



El diseño de Fitness360 se basa en el patrón de arquitectura MVC (Modelo-Vista-Contrólador), el cual permite separar las responsabilidades y facilita el mantenimiento y evolución del código.

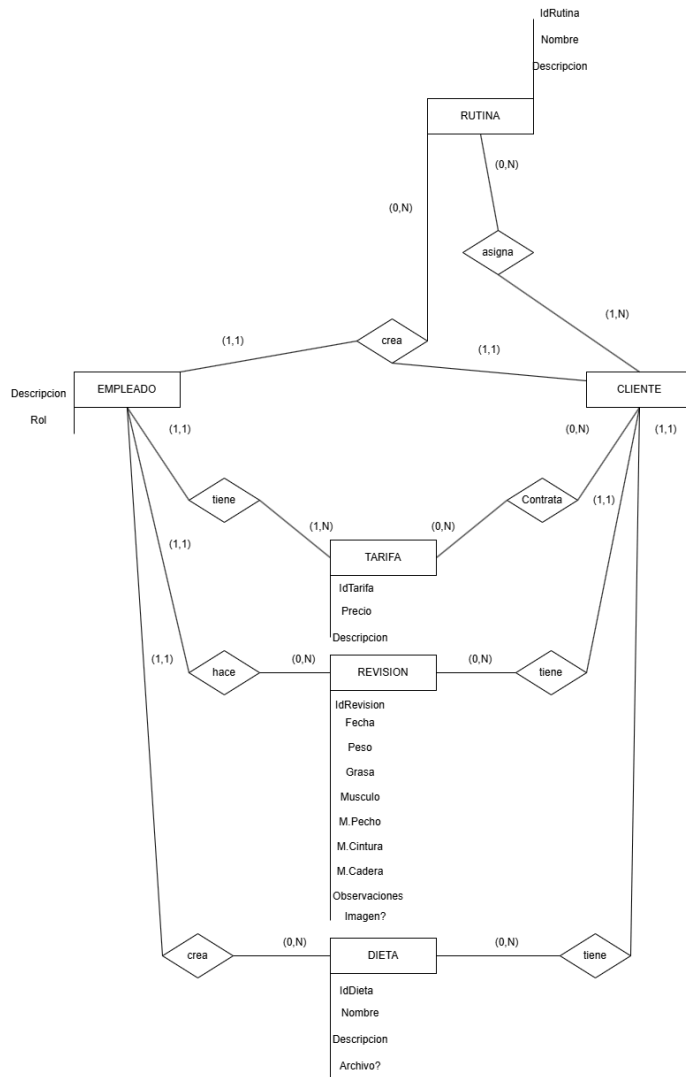
- **Modelo (Model):** Incluye las clases que representan las entidades del sistema y la lógica de negocio. Estas clases gestionan los datos, reglas y estructuras internas. Por ejemplo, las entidades Usuario, Rutina, Dieta o Revisión.
- **Vista (View):** Representa la interfaz gráfica que el usuario final utiliza para interactuar con el sistema. En Fitness360, las vistas están definidas mediante archivos FXML y se renderizan usando JavaFX.
- **Contrólador (Controller):** Gestiona la interacción entre el modelo y la vista, captura los eventos generados por el usuario (clics, entradas de texto, selecciones) y ejecuta la lógica correspondiente para actualizar la vista o modificar el modelo.

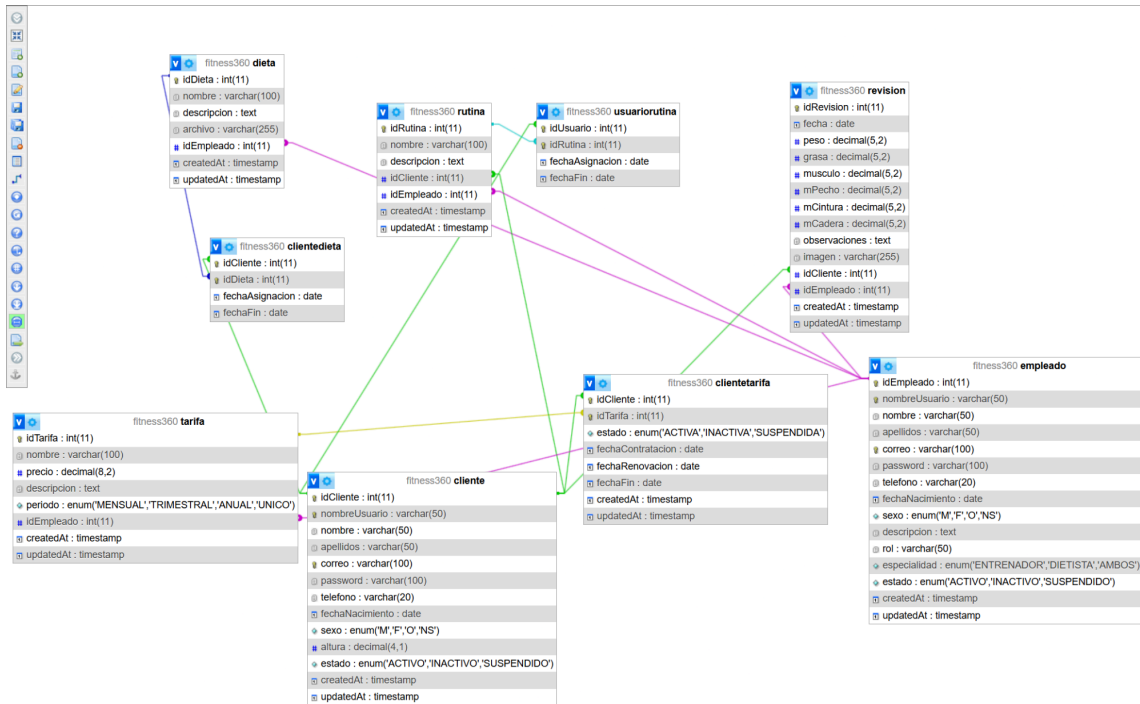
Estructura de paquetes

- **modelo:** Contiene las clases de entidad que representan los objetos del dominio, como UsuarioCliente, UsuarioEmpleado, Rutina, Dieta, etc. Cada clase incluye atributos, constructores, métodos getters/setters y posibles métodos adicionales para lógica específica.
- **dao:** Incluye las clases Data Access Object, encargadas de encapsular todas las operaciones de acceso a la base de datos. Esto incluye las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para cada entidad, así como métodos especializados para consultas específicas.
- **controlador:** Contiene los controladores JavaFX que gestionan la lógica de eventos y la interacción con las vistas, facilitando el flujo de la aplicación.
- **vistas:** Carpeta que almacena los archivos FXML, que definen el diseño visual y la estructura de las pantallas de la aplicación.
- **util:** Paquete para clases auxiliares que realizan funciones comunes y reutilizables como validadores de datos, gestores de encriptación de contraseñas, y utilidades para el manejo de fechas o cadenas.
- **baseDatos:** Contiene las clases encargadas de gestionar la conexión y configuración del acceso a la base de datos, garantizando que la conexión sea segura y eficiente.
- **exceptions:** Contiene las clases de excepciones personalizadas utilizadas en la aplicación para manejar errores específicos.

4. Modelo de Datos

La base de datos MySQL utilizada en Fitness360 está organizada en varias tablas principales que almacenan la información fundamental del sistema, así como tablas de relación que gestionan las asociaciones entre entidades.





Tablas principales:

- **Cliente:** Almacena los datos personales, credenciales y de acceso de los usuarios clientes que reciben servicios.
- **Empleado:** Contiene la información de los empleados que trabajan como entrenadores o dietistas o ambos.
- **Rutina:** Registra las rutinas de ejercicios creadas por los clientes o los empleados y que pueden ser asignadas a los clientes.
- **Dieta:** Registra los planes de alimentación desarrollados por los dietistas para ser asignados a los clientes.
- **Tarifa:** Define los diferentes planes económicos o paquetes que los clientes pueden contratar, relacionados con el tipo de servicio ofrecido.
- **Revisión:** Almacena datos sobre el progreso físico de los clientes, como peso corporal, porcentaje de grasa, masa muscular y otros parámetros relevantes para el seguimiento.

Tablas de relación:

- **Cliente Dieta:** Tabla intermedia que vincula a los clientes con las dietas asignadas, permitiendo una relación muchos a muchos si fuera necesario.
- **Usuario Rutina:** Relaciona a los clientes con las rutinas asignadas por los profesionales.
- **Cliente Tarifa:** Vincula a los clientes con las tarifas contratadas, permitiendo gestionar los diferentes planes económicos.

Estas tablas y relaciones aseguran que la información esté organizada de forma normalizada, evitando redundancias y facilitando consultas eficientes.

5. Casos de uso

La aplicación cubre una serie de funcionalidades clave que permiten a los diferentes perfiles de usuario interactuar con el sistema de manera efectiva:



- **Registro e inicio de sesión:** Los usuarios pueden crear una cuenta y acceder al sistema mediante un proceso seguro que utiliza contraseñas hasheadas para proteger la información sensible. Se implementan mecanismos para validar los datos y evitar accesos no autorizados.

- **Gestión de rutinas y dietas:** Los profesionales (entrenadores y dietistas) pueden crear, modificar y eliminar rutinas de ejercicios y planes alimenticios, ajustándose a las necesidades de sus clientes. Además los clientes también pueden crear, modificar y eliminar sus propias rutinas.
- **Asignación de planes:** Los profesionales pueden asignar las rutinas y dietas a los clientes desde una interfaz específica, estableciendo así planes personalizados y adaptados a cada usuario.
- **Consulta de planes:** Los clientes pueden acceder a sus rutinas y dietas asignadas, visualizando detalles y recomendaciones para su correcta ejecución y seguimiento.
- **Gestión de tarifas:** Los administradores o profesionales autorizados pueden crear y asignar diferentes tipos de planes económicos, permitiendo gestionar los contratos y pagos de los clientes.
- **Seguimiento físico:** Los clientes pueden registrar sus datos físicos periódicamente, lo que permite a los profesionales monitorizar el progreso y ajustar los planes según la evolución.

6. Interfaces de usuario (GUI)

La interfaz gráfica está diseñada para ofrecer una experiencia de usuario clara y sencilla, con un diseño consistente en todas las pantallas. A continuación se detallan todas las vistas FXML que componen la aplicación:

- **login-view.fxml:** Permite a los usuarios autenticarse en el sistema introduciendo su usuario y contraseña. Incluye campos para el nombre de usuario y contraseña, así como botones para iniciar sesión o acceder al registro.



The image shows a login form for Fitness360. It has a title 'Fitness360 - Login' at the top. Below the title, there are two input fields: one for 'Nombre de Usuario:' and one for 'Contraseña:'. The first input field has a placeholder text 'Ingrese su nombre de usuario' and the second has 'Ingrese su contraseña'. Below these fields is a green button labeled 'Iniciar Sesión'. At the bottom, there is a link '¿No tienes cuenta?' followed by a green button labeled 'Registrarse'.

- **registro-view.fxml:** Facilita la creación de nuevas cuentas, con validaciones para asegurar que los datos introducidos sean correctos y completos. Contiene campos para todos los datos personales necesarios y opciones para seleccionar el tipo de usuario.

Fitness360 - Registro de Usuario

Tipo de Usuario: ☒ Cliente ☐ Profesional

Nombre de Usuario:

Nombre:


Apellidos:

Correo Electrónico:

Contraseña:

Confirmar Contraseña:

Teléfono:

Fecha de Nacimiento: 

Sexo:

Altura (cm):

- **main-view-empleado.fxml:** Panel principal para profesionales (entrenadores y dietistas). Desde aquí pueden gestionar rutinas, dietas, tarifas y revisiones, con accesos rápidos a las funcionalidades principales. Utiliza un sistema de pestañas (TabPane) para organizar las diferentes secciones.

Fitness360 - Panel de Empleado
Bienvenido, [Nombre Usuario]

Mis Clientes | Mis Rutinas Creadas | Mis Dietas Creadas | Mis Tarifas | Revisiones Realizadas

Cientes asignados:

Nombre	Apellidos	Email	Teléfono	Fecha Alta
Tabla sin contenido				

- **main-view-cliente.fxml:** Panel principal para clientes donde pueden visualizar sus planes asignados y su historial de revisiones, permitiendo un seguimiento continuo y motivador. También utiliza un sistema de pestañas para organizar la información.

The screenshot shows the 'Fitness360 - Panel de Cliente' interface. At the top, there's a green header with the title and a welcome message 'Bienvenido, [Nombre Usuario]'. Below the header is a navigation bar with tabs: 'Mis Rutinas', 'Mis Dietas', 'Mis Revisiones', 'Mis Tarifas', and 'Contratar Entrenador'. Under 'Mis Rutinas', there's a section 'Rutinas disponibles:' with a 'Filtrar por' dropdown menu. To the right of this are three buttons: 'Crear Rutina' (green), 'Modificar Rutina' (green), and 'Eliminar Rutina' (red). Below these is a table with columns: 'Nombre', 'Descripción', 'Creador', 'FechaInicio', and 'FechaFin'. The table is currently empty, showing 'Tabla sin contenido'. At the bottom right, there is a 'Cerrar Sesión' button (red).

- **registro-rutina-view.fxml:** Interfaz para la creación y edición de rutinas de entrenamiento. Permite definir nombre, descripción y otros detalles de la rutina.

The screenshot shows the 'Fitness360 - Registro de Rutina' interface. At the top, there's a title 'Fitness360 - Registro de Rutina'. Below the title, there's a 'Creado por:' section with two radio buttons: 'Cliente' (selected) and 'Empleado'. Below this, there's a 'Nombre de la Rutina:' label followed by a text input field with placeholder text 'Ingrese nombre de la rutina'. Below that, there's a 'Descripción:' label followed by a larger text area with placeholder text 'Ingrese descripción detallada de la rutina'. Below the description area, there's a 'Cliente Creador:' label followed by a dropdown menu with the text 'Seleccione el cliente creador'. At the bottom, there are two buttons: 'Registrar' (green) and 'Cancelar' (red).

- **registro-dieta-view.fxml:** Interfaz para la creación y edición de planes de alimentación. Incluye campos para nombre, descripción y detalles específicos de la dieta.

The screenshot shows a web form titled "Fitness360 - Registro de Dieta". It contains the following fields: "Nombre de la Dieta:" with a text input field containing the placeholder "Ingrese nombre de la dieta"; "Descripción:" with a large text area containing the placeholder "Ingrese descripción detallada de la dieta"; and "Asignar a Cliente:" with a dropdown menu containing the placeholder "Seleccione el cliente al que asignar la dieta". At the bottom, there are two buttons: a green "Registrar" button and a red "Cancelar" button.

- **registro-tarifa-view.fxml:** Permite crear y modificar las tarifas disponibles en el sistema, definiendo precios, periodos y servicios incluidos.

The screenshot shows a web form titled "Fitness360 - Registro de Tarifa". It contains the following fields: "Nombre de la Tarifa:" with a text input field containing the placeholder "Ingrese nombre de la tarifa"; "Descripción:" with a large text area containing the placeholder "Ingrese descripción detallada de la tarifa"; "Precio (€):" with a text input field containing the placeholder "Ingrese el precio de la tarifa"; and "Periodo:" with a dropdown menu containing the placeholder "Seleccione el periodo de la tarifa". At the bottom, there are two buttons: a green "Registrar" button and a red "Cancelar" button.

- **registro-revision-view.fxml:** Interfaz para registrar y actualizar las revisiones físicas de los clientes, incluyendo campos para peso, porcentaje de grasa corporal y otros indicadores.

Fitness360 - Registro de Revisión

Cliente:

Fecha:

Peso (kg):

Grasa corporal (%):

Masa muscular (%):

Medida de pecho (cm):

Medida de cintura (cm):

Medida de cadera (cm):

Observaciones:

- **asignar-view.fxml:** Vista especializada para asignar rutinas o dietas a uno o varios clientes simultáneamente. Incluye una tabla con selección mediante checkboxes.

Asignar a Clientes

Clientes disponibles:

Sele...	Nombre	Apellidos	Email
Tabla sin contenido			

7. Controladores

Los controladores JavaFX actúan como puente entre la interfaz gráfica y la lógica de negocio, manejando eventos y actualizando la información mostrada. A continuación se detallan todos los controladores implementados en la aplicación:

- **LoginController:** Gestiona el proceso de autenticación, validando credenciales y controlando los accesos. Verifica el tipo de usuario (cliente o empleado) y redirige a la vista correspondiente. Implementa medidas de seguridad como el hash de contraseñas y bloqueo tras intentos fallidos.
- **RegistroController:** Controla la creación de nuevas cuentas y la validación de datos durante el registro. Incluye validaciones para todos los campos del formulario y gestiona el proceso de alta en la base de datos, asegurando la integridad de la información.
- **MainViewClienteController:** Maneja la interfaz principal del cliente, cargando los planes y revisiones correspondientes. Gestiona las diferentes pestañas (rutinas, dietas, revisiones, tarifas) y proporciona funcionalidades para visualizar, filtrar y gestionar la información personal del cliente.
- **MainViewEmpleadoController:** Controla la vista principal del profesional, facilitando la gestión integral de planes y clientes. Permite crear, modificar y eliminar rutinas, dietas y tarifas, así como gestionar los clientes asignados y sus revisiones. Incluye funcionalidades de búsqueda y filtrado avanzadas.
- **RegistroRutinaController:** Gestiona la creación y edición de rutinas de entrenamiento. Permite definir nombre, descripción y detalles específicos de la rutina, así como asignarla a clientes si se accede desde el perfil de empleado.
- **RegistroDietaController:** Controla la creación y edición de planes de alimentación. Facilita la definición de dietas personalizadas con nombre, descripción y detalles nutricionales, permitiendo también su asignación a clientes.
- **RegistroTarifaController:** Maneja el registro y modificación de tarifas en el sistema. Permite definir precios, periodos (mensual, trimestral, anual) y servicios incluidos, así como gestionar su estado (activa/inactiva).

- **RegistroRevisionController:** Controla el registro y actualización de revisiones físicas de los clientes. Permite introducir y validar datos como peso, porcentaje de grasa corporal, masa muscular y otros indicadores relevantes para el seguimiento.
- **AsignarController:** Controlador especializado para la asignación de rutinas o dietas a múltiples clientes simultáneamente. Implementa una interfaz con selección mediante checkboxes que permite al empleado seleccionar varios clientes y asignarles un plan específico en una sola operación, optimizando el flujo de trabajo.

8. DAO (Data Access Object)

El patrón DAO se implementa para aislar la lógica de acceso a datos y facilitar la reutilización y el mantenimiento del código. Destacan las siguientes clases:

- **GenericDAO:** Clase abstracta que define operaciones CRUD genéricas aplicables a diferentes entidades.
- **UsuarioClienteDAO, UsuarioEmpleadoDAO:** Gestionan las operaciones específicas para los usuarios clientes y empleados.
- **RutinaDAO, DietaDAO, TarifaDAO, RevisionDAO:** Clases que manejan las operaciones sobre las tablas correspondientes.
- **ClienteRutinaDAO, ClienteDietaDAO, ClienteTarifaDAO:** Manejan las tablas intermedias y las relaciones entre clientes y planes o tarifas.

9. Validaciones y control de errores

Para garantizar la calidad y seguridad de la aplicación, se implementan múltiples mecanismos de validación y control:

- **Validación de campos:** Se verifica que no existan campos vacíos, que los correos tengan formato válido y que las contraseñas cumplan con requisitos mínimos de seguridad.
- **Control de acceso:** Se restringen funciones según el rol del usuario para evitar accesos no autorizados.

- **Excepciones personalizadas:** Se definen clases propias para manejar errores específicos como `CampoVacioException` o `EmailInvalidoException`, facilitando un manejo más claro y específico. Estas excepciones heredan de una excepción padre llamada `ValidacionException` que esta hereda de `runTimeException` y no de `Exception` por los siguientes motivos:
 - Son errores del usuario, no del sistema: No representan fallos externos (como fallo de conexión o archivo no encontrado), sino que el usuario escribió mal o dejó un campo vacío.
 - No quieres forzar throws en todos los métodos que validan: Sería muy molesto tener que escribir throws `CampoVacioException` en todos tus controladores.
 - Facilita la propagación de errores a la capa visual (controladores JavaFX): Pueden capturarlas de forma centralizada y mostrar alertas sin necesidad de modificar todas las firmas de métodos.
 - Es lo que hace la mayoría de frameworks modernos (Spring, Jakarta, etc.): Las validaciones usan `RuntimeExceptions` (como `IllegalArgumentException`) y se capturan cerca de la UI o con interceptores.
- **Manejo de errores SQL:** Se capturan excepciones derivadas de la base de datos, como errores de duplicidad o problemas de conexión, mostrando mensajes claros al usuario.
- **Registro de logs:** Se utiliza la librería `SLF4J` para registrar eventos y errores en archivos de log, facilitando la auditoría y depuración.

10. Pruebas y calidad

El proyecto se somete a diversas pruebas para asegurar su correcto funcionamiento y robustez:

- **Pruebas funcionales:** Se realizan comprobaciones manuales de cada caso de uso para verificar que la aplicación responde correctamente.

- **Pruebas de validación:** Se testean las reglas de negocio y restricciones para asegurar que no se introducen datos erróneos o inconsistentes.
- **Pruebas de integridad:** Se verifica que las relaciones y restricciones de la base de datos se mantienen correctamente, evitando inconsistencias.
- **Pruebas de carga:** Se simulan múltiples entradas y usuarios para evaluar el rendimiento y estabilidad bajo condiciones de uso realistas.
- **Pruebas de interfaz gráfica:** Se prueba la visualización en distintas resoluciones y tamaños de pantalla para garantizar una experiencia consistente.

11. Instalación y despliegue

Para ejecutar Fitness360 en un entorno local o de producción, se deben seguir los siguientes pasos y cumplir con los requisitos:

Requisitos previos

- **Java JDK 17:** Debe estar instalado para ejecutar la aplicación y compilar el código fuente.
- **MySQL Server:** Servidor de base de datos activo y accesible.

Pasos para la instalación

- **Crear la base de datos:** Ejecutar el script SQL (Script.sql) para crear la estructura de tablas y relaciones necesarias.
- **Poblar con datos de prueba (opcional):** Ejecutar el archivo insert.sql para insertar datos iniciales que faciliten las pruebas.
- **Configurar conexión:** Editar el archivo connection.xml con los parámetros de conexión al servidor MySQL, incluyendo host, puerto, usuario y contraseña.
- **Ejecutar la aplicación:** Puede hacerse directamente desde IntelliJ IDEA o mediante la ejecución del archivo JAR generado, asegurando que el entorno Java está correctamente configurado.

12. Lista de comprobación de requisitos técnicos

1. Uso de herencia, restricciones con abstract y llamada a constructores de la clase padre

En Fitness360, la herencia se implementa principalmente en las clases de usuario:

- La clase base Usuario define atributos y métodos comunes para todos los usuarios del sistema.
- Las clases UsuarioCliente y UsuarioEmpleado extienden de Usuario, añadiendo atributos y comportamientos específicos.

Ejemplo de herencia y llamada al constructor padre en UsuarioCliente:

```
public class UsuarioCliente extends Usuario {  
    private double altura;  
    private List<ClienteRutina> rutinasAsignadas;  
  
    public UsuarioCliente() {  
        super(); // Llamada al constructor de la clase padre  
    }  
  
    public UsuarioCliente(int id, String nombreUsuario, String  
nombre, String apellidos,  
                        String correo, String password, String  
telefono,  
                        Date fechaNacimiento, Sexo sexo, Estado
```

```

estado,
                                double altura, Date createdAt, Date
updatedAt) {
    // Llamada al constructor de la clase padre con
    parámetros
    super(id, nombreUsuario, nombre, apellidos, correo,
password,
                                telefono, fechaNacimiento, sexo, estado,
createdAt, updatedAt);
    // Inicialización de atributos propios
    this.altura = altura;
}

// Resto de la clase...
}

```

Aunque el proyecto no utiliza clases abstractas explícitamente, implementa interfaces como GenericDAO que definen métodos que deben ser implementados por las clases DAO concretas, proporcionando un comportamiento similar al de las clases abstractas.

2. Relaciones utilizadas (1:N o N:M)

El proyecto implementa ambos tipos de relaciones:

- **Relaciones 1:N:**
 - Un empleado puede crear múltiples dietas o rutinas o tarifas.
 - Un cliente puede tener múltiples revisiones
- **Relaciones N:M:**
 - Clientes y dietas: Un cliente puede tener varias dietas asignadas, y una dieta puede ser asignada a varios clientes
 - Clientes y rutinas: Un cliente puede tener varias rutinas, y una rutina puede ser asignada a varios clientes

- Clientes y tarifas: Un cliente puede contratar varias tarifas, y una tarifa puede ser contratada por varios clientes
- Las relaciones N:M se implementan mediante tablas intermedias como ClienteDieta, ClienteRutina y ClienteTarifa.

3. Construcción de los DAO y control en relaciones N:M

Los DAO se implementan siguiendo el patrón DAO (Data Access Object):

- Se define una interfaz GenericDAO<T> con métodos CRUD genéricos
- Cada entidad tiene su propia clase DAO que implementa esta interfaz
- Para las relaciones N:M, se crean DAOs específicos que gestionan las tablas intermedias

Ejemplo de control en relaciones N:M en ClienteDietaDAO:

```
public class ClienteDietaDAO implements GenericDAO<ClienteDieta>
{
    // Consultas SQL para gestionar la relación N:M
    private static final String SQL_INSERT =
        "INSERT INTO ClienteDieta (idCliente, idDieta,
fechaAsignacion, fechaFin) VALUES (?, ?, ?, ?)";
    private static final String SQL_DELETE =
        "DELETE FROM ClienteDieta WHERE idCliente = ? AND
idDieta = ?";

    // Método para insertar una relación
    @Override
    public ClienteDieta insert(ClienteDieta entity) {
        try (Connection conn = ConnectionDB.getConnection();
            PreparedStatement stmt =
conn.prepareStatement(SQL_INSERT)) {
            stmt.setInt(1, entity.getId());

```

```

        stmt.setInt(2, entity.getDieta().getIdDieta());
        stmt.setDate(3, (Date) entity.getFechaAsignacion());
        stmt.setDate(4, (Date) entity.getFechaFin());
        stmt.executeUpdate();
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
    return entity;
}

// Método para eliminar una relación
@Override
public boolean delete(ClienteDieta entity) {
    try (Connection conn = ConnectionDB.getConnection();
        PreparedStatement stmt =
conn.prepareStatement(SQL_DELETE)) {
        stmt.setInt(1, entity.getCliente().getId());
        stmt.setInt(2, entity.getDieta().getIdDieta());
        stmt.executeUpdate();
        return true;
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}

// Otros métodos...
}

```

4. Funcionamiento de mostrar, insertar y eliminar

- **Mostrar:** Los datos se recuperan de la base de datos mediante los métodos `getAll()`, `getById()` o métodos personalizados de los DAOs, y se muestran en la interfaz utilizando componentes JavaFX como `TableView`.
- **Insertar:** Los controladores recogen los datos de los formularios, validan la información, crean nuevos objetos y utilizan los DAOs para persistirlos en la base de datos.

Ejemplo de inserción en `RegistroDietaController`:

```
public boolean registrarDieta() {
    try {
        // Validar campos

        Utilidades.validarCampoNoVacio(nombreDietaField.getText(),
        "nombre de la dieta");

        // Crear dieta
        Dieta dieta = new Dieta();
        dieta.setNombre(nombreDietaField.getText().trim());

        dieta.setDescripcion(descripcionDietaField.getText().trim());
        dieta.setCreador(empleadoAutenticado);

        // Insertar la dieta
        Dieta dietaRegistrada = dietaDAO.insert(dieta);

        // Si se ha seleccionado un cliente, asignar la dieta
        if (clienteAsignado != null) {
            ClienteDieta clienteDieta = new ClienteDieta();
```

```

        clienteDieta.setCliente(clienteAsignado);
        clienteDieta.setDieta(dietaRegistrada);
        clienteDieta.setFechaAsignacion(new
java.sql.Date(System.currentTimeMillis()));

        clienteDietaDAO.insert(clienteDieta);
    }
    return true;
} catch (Exception e) {
    // Manejo de errores
    return false;
}
}

```

- **Eliminar:** Los controladores identifican el objeto a eliminar, solicitan confirmación al usuario y utilizan los DAOs para eliminarlo de la base de datos.

5. Funcionamiento de actualizar

La actualización de datos sigue estos pasos:

1. Se recupera el objeto existente de la base de datos
2. Se modifican sus atributos con los nuevos valores
3. Se utiliza el método update() del DAO correspondiente para persistir los cambios

Ejemplo de actualización en RegistroDietaController:

```

private void manejarRegistro() {
    if (validarCampos()) {
        // Si estamos editando una dieta existente
        if (this.dieta != null) {
            try {

```

```

        // Actualizar los datos de la dieta existente

dieta.setNombre(nombreDietaField.getText().trim());

dieta.setDescripcion(descripcionDietaField.getText().trim());

        // Actualizar la dieta en la base de datos
dietaDAO.update(dieta);
        logger.info("Dieta actualizada correctamente:
{}", dieta.getNombre());
    } catch (Exception e) {
        // Manejo de errores
    }
} else {
    // Crear una nueva dieta
    registrarDieta();
}
}
}

```

En algunos casos, como en las relaciones N:M, la actualización se implementa eliminando la relación existente y creando una nueva:

```

@Override
public boolean update(ClienteDieta entity) {
    try {
        delete(entity);
        insert(entity);
        return true;
    } catch (Exception e) {
        return false;
    }
}

```

```
}
```

6. Carga de datos en columnas de TableView con CellData

En Fitness360, se utilizan dos enfoques principales para cargar datos en las columnas de TableView:

Uso de setCellValueFactory con lambda expressions y SimpleStringProperty

Este enfoque permite una mayor flexibilidad y control sobre cómo se muestran los datos, incluyendo la posibilidad de formatear valores, combinar campos o mostrar datos calculados. Es especialmente útil cuando se necesita transformar los datos antes de mostrarlos.

Ejemplo de carga de columnas en MainViewClienteController:

```
// Configuración de columnas para la tabla de tarifas
contratadas
colNombreMiTarifa.setCellValueFactory(cellData ->
                                                                    new
SimpleStringProperty(cellData.getValue().getTarifa().getNombre()
)
);

// Formateo de valores numéricos con formato específico
colPrecioMiTarifa.setCellValueFactory(cellData ->
                                                                    new SimpleStringProperty(String.format("%.2f",
cellData.getValue().getTarifa().getPrecio()) + " €")
);
```

```
// Manejo de valores que podrían ser nulos
colPeriodoMiTarifa.setCellValueFactory(cellData -> {
    Periodo periodo =
cellData.getValue().getTarifa().getPeriodo();
    return new SimpleStringProperty(periodo != null ?
periodo.toString() : "Sin especificar");
});

// Formateo de fechas con formato personalizado
colFechaContratacionMiTarifa.setCellValueFactory(cellData -> {
    java.util.Date fecha =
cellData.getValue().getFechaContratacion();
    return new
SimpleStringProperty(Utilidades.formatearFechaEspanol(fecha));
});
```

Uso de PropertyValueFactory

Este enfoque más simple se basa en la convención de nombres de JavaFX para vincular automáticamente las propiedades de los objetos con las columnas de la tabla. Es menos flexible pero requiere menos código cuando los nombres de las propiedades coinciden con los nombres de los campos.

```
// Ejemplo de uso de PropertyValueFactory
TableColumn<Usuario, String> nombreColumn = new
TableColumn<>("Nombre");
nombreColumn.setCellValueFactory(new
PropertyValueFactory<>("nombre"));
```

7. Estructura de interfaces con TabPane

La aplicación utiliza TabPane para organizar la interfaz en pestañas temáticas, lo que permite una navegación clara y una separación lógica de las funcionalidades.

Definición del TabPane en FXML

En el archivo main-view-cliente.fxml, el TabPane se define de la siguiente manera:

```
<TabPane fx:id="tabPane" prefHeight="200.0" prefWidth="200.0"
tabClosingPolicy="UNAVAILABLE" BorderPane.alignment="CENTER">
  <tabs>
    <Tab fx:id="tabRutinas" text="Mis Rutinas">
      <content>
        <!-- Contenido de la pestaña Rutinas -->
      </content>
    </Tab>
    <Tab fx:id="tabDietas" text="Mis Dietas">
      <content>
        <!-- Contenido de la pestaña Dietas -->
      </content>
    </Tab>
    <Tab fx:id="tabRevisiones" text="Mis Revisiones">
      <content>
        <!-- Contenido de la pestaña Revisiones -->
      </content>
    </Tab>
    <!-- Más pestañas... -->
  </tabs>
</TabPane>
```

Estructura común de las pestañas

Cada pestaña sigue una estructura similar:

1. Un contenedor principal (AnchorPane)
2. Un layout vertical (VBox) para organizar los elementos
3. Una barra superior (HBox) con controles de filtrado y acciones
4. Una tabla (TableView) que muestra los datos principales

5. Opcionalmente, controles adicionales específicos de la funcionalidad

Ejemplo de estructura de una pestaña:

```
<Tab fx:id="tabRutinas" text="Mis Rutinas">
  <content>
    <AnchorPane minHeight="0.0" minWidth="0.0"
prefHeight="180.0" prefWidth="200.0">
      <children>
        <VBox prefHeight="511.0" prefWidth="800.0"
AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0"
AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">
          <children>
            <!-- Barra superior con filtros y botones -->
            <HBox alignment="CENTER_LEFT" prefHeight="50.0"
prefWidth="200.0" spacing="10.0">
              <children>
                <Label text="Rutinas disponibles:"
styleClass="section-header" />
                <ComboBox fx:id="comboFiltroRutinas"
prefWidth="150.0" promptText="Filtrar por" />
                <Region prefHeight="31.0" prefWidth="68.0"
HBox.hgrow="ALWAYS" />
                <Button fx:id="btnCrearRutina"
mnemonicParsing="false" text="Crear Rutina" />
                <Button fx:id="btnModificarRutina"
layoutX="697.0" layoutY="20.0" mnemonicParsing="false"
text="Modificar Rutina" />
                <Button fx:id="btnEliminarRutina"
layoutX="587.0" layoutY="20.0" styleClass="delete-button"
mnemonicParsing="false" text="Eliminar Rutina" />
              </children>
            </HBox>
          </children>
        </VBox>
      </children>
    </AnchorPane>
  </content>
</Tab>
```

```

        <!-- Tabla principal de datos -->
        <TableView fx:id="tablaRutinas" prefHeight="418.0"
prefWidth="815.0" VBox.vgrow="ALWAYS">
            <columns>
                <TableColumn fx:id="colNombreRutina"
prefWidth="150.0" text="Nombre" />
                <TableColumn fx:id="colDescripcionRutina"
prefWidth="250.0" text="Descripción" />
                <!-- Más columnas... -->
            </columns>
        </TableView>
    </children>
</VBox>
</children>
</AnchorPane>
</content>
</Tab>

```

Manejo del TabPane en el controlador

En el controlador MainViewClienteController, cada pestaña tiene sus propios componentes y métodos asociados:

```

// Declaración del TabPane y sus pestañas
public TabPane tabPane;
public Tab tabRutinas;
public Tab tabDietas;
public Tab tabRevisiones;
public Tab tabMisTarifas;
public Tab tabContratarEntrenador;

```



```
// Inicialización y carga de datos
public void cargarDatosCliente() {
    logger.debug("Iniciando carga de datos del cliente");
    cargarRutinas();
    cargarDietas();
    cargarRevisiones();
    cargarMisTarifas();
    cargarEntrenadores();
    logger.info("Datos del cliente cargados correctamente");
}

// Métodos específicos para cada pestaña
private void cargarRutinas() {
    // Código para cargar datos en la pestaña de rutinas
}

private void cargarDietas() {
    // Código para cargar datos en la pestaña de dietas
}

// Más métodos...
```

Esta estructura modular facilita la organización del código y la separación de responsabilidades, haciendo que la aplicación sea más mantenible y escalable.

13. Patrones de diseño y componentes adicionales

El proyecto Fitness360 implementa varios patrones de diseño y componentes adicionales que mejoran su estructura y funcionalidad:

1. Patrón Singleton

Se implementa el patrón Singleton en la clase Sesion, que gestiona la sesión del usuario en la aplicación. Este patrón garantiza que solo exista una instancia de la sesión en toda la aplicación, proporcionando un punto de acceso global a ella.

```
public class Sesion {  
    // Instancia única de la clase (patrón Singleton)  
    private static Sesion instance;  
  
    // Constructor privado para evitar instanciación directa  
    private Sesion() { }  
  
    // Método para obtener la instancia única  
    public static Sesion getInstance() {  
        if (instance == null) {  
            instance = new Sesion();  
        }  
        return instance;  
    }  
  
    // Resto de la clase...  
}
```

Esta clase mantiene el estado de autenticación y el usuario actual, proporcionando métodos para verificar si el usuario es un cliente o un empleado, y para obtener el usuario autenticado.

2. Enumeraciones para tipos de datos específicos

El proyecto utiliza varias enumeraciones para representar tipos de datos específicos:

- **Especialidad:** Representa las posibles especialidades de los empleados (ENTRENADOR, DIETISTA, AMBOS).

- **Estado:** Define los posibles estados de un usuario (ACTIVO, INACTIVO).
- **EstadoTarifa:** Indica los posibles estados de una tarifa (ACTIVA, INACTIVA).
- **Periodo:** Representa los posibles periodos de tiempo para tarifas o suscripciones (MENSUAL, TRIMESTRAL, ANUAL, UNICO).
- **Sexo:** Define los posibles valores para el sexo de un usuario (MASCULINO, FEMENINO, OTRO).

Estas enumeraciones mejoran la legibilidad del código y evitan errores al restringir los valores posibles para ciertos atributos.

3. Clases auxiliares para la interfaz de usuario

La clase ClienteSeleccionable es un wrapper que se utiliza en las interfaces de asignación para permitir seleccionar clientes mediante checkboxes en tablas. Esta clase combina un objeto UsuarioCliente con una propiedad observable (SimpleBooleanProperty) que indica si el cliente está seleccionado.

```
public class ClienteSeleccionable {  
    private final UsuarioCliente cliente;  
    private final SimpleBooleanProperty seleccionado;  
    private final boolean yaAsignado;  
  
    // Constructor y métodos...  
}
```

4. Controlador de asignación

El AsignarController es un controlador especializado para la asignación de rutinas o dietas a clientes. Permite a los empleados seleccionar múltiples clientes y asignarles una rutina o dieta específica.

Este controlador trabaja con la vista `asignar-view.fxml` y utiliza la clase `ClienteSeleccionable` para gestionar la selección de clientes en una tabla con checkboxes.

5. Uso de expresiones lambda

El proyecto Fitness360 hace un uso extensivo de expresiones lambda, una característica introducida en Java 8 que permite escribir código más conciso y funcional. Las expresiones lambda se utilizan en diferentes partes del proyecto para mejorar la legibilidad.

6. Configuración de TableView con expresiones lambda

Uno de los usos más frecuentes de las expresiones lambda en el proyecto es la configuración de las columnas de TableView mediante el método `setCellValueFactory`. Este enfoque permite definir de manera concisa cómo se extraen y formatean los datos para mostrarlos en las tablas.

Ejemplo de configuración simple de una columna:

```
// Configuración de columna para mostrar el nombre de una tarifa
colNombreMiTarifa.setCellValueFactory(cellData ->
    new SimpleStringProperty(cellData.getValue().getTarifa().getNombre()
    )
);
```

Ejemplo de configuración con lógica adicional para formateo:

```
// Formateo de valores numéricos con formato específico
colPrecioMiTarifa.setCellValueFactory(cellData ->
    new SimpleStringProperty(String.format("%.2f",
    cellData.getValue().getTarifa().getPrecio()) + " €")
);
```

```
);
```

Ejemplo de manejo de valores potencialmente nulos:

```
// Manejo de valores que podrían ser nulos
colPeriodoMiTarifa.setCellValueFactory(cellData -> {
    Periodo periodo =
cellData.getValue().getTarifa().getPeriodo();
    return new SimpleStringProperty(periodo != null ?
periodo.toString() : "Sin especificar");
});
```

7. Manejo de eventos con expresiones lambda

Las expresiones lambda también se utilizan para definir manejadores de eventos de forma concisa, lo que facilita la lectura y mantenimiento del código.

Ejemplo de configuración de eventos para botones:

```
// Configuración de eventos para botones
btnCancelar.setOnAction(event -> cerrarVentana());
btnAsignar.setOnAction(event -> asignarSeleccionados());
Ejemplo de configuración de eventos con lógica más compleja:
```

```
// Configuración de evento con lógica adicional
empleadoRadio.setOnAction(e -> {
    alternarCamposCreador();
    empleadoComboBox.setDisable(false);
    empleadoComboBox.getSelectionModel().selectFirst();
});
Switch expressions con lambda
```

En las enumeraciones del proyecto, se utilizan expresiones lambda en combinación con `switch expressions` (introducidas en Java 12) para proporcionar representaciones de cadena más legibles para los valores enumerados.

Ejemplo en la enumeración `Periodo`:

```
@Override
public String toString() {
    return switch (this) {
        case MENSUAL -> "Mensual";
        case TRIMESTRAL -> "Trimestral";
        case ANUAL -> "Anual";
        case UNICO -> "Unico";
    };
}
```

El uso de expresiones lambda en el proyecto Fitness360 demuestra cómo las características modernas de Java pueden mejorar significativamente la calidad y mantenibilidad del código, permitiendo expresar operaciones complejas de manera más concisa y legible.

14. Futuras mejoras

A continuación, se presentan diversas mejoras y funcionalidades adicionales que podrían implementarse en futuras versiones de Fitness360 para enriquecer la experiencia de usuario y ampliar las capacidades del sistema:

1. Gestión avanzada de rutinas y dietas

- **Asignación de fecha de fin:** Implementar la posibilidad de establecer una fecha de finalización al asignar rutinas y dietas a los clientes, permitiendo una mejor planificación temporal de los programas de entrenamiento y alimentación.
- **Gestión de estados:** Añadir estados para rutinas y dietas (activa, completada, pausada, cancelada) que permitan un seguimiento más preciso del progreso del cliente.
- **Agregar ejercicios a las rutinas:** Desarrollar un sistema para añadir ejercicios específicos a las rutinas, incluyendo series, repeticiones, descansos y notas técnicas.
- **Planificación semanal de dietas:** Permitir la creación de planes alimenticios detallados por días de la semana, incluyendo diferentes comidas y cantidades específicas.
- **Historial de rutinas y dietas:** Mantener un registro histórico de todas las rutinas y dietas que ha seguido un cliente, facilitando el análisis de su evolución a largo plazo.

2. Mejoras en la gestión de usuarios

- **Modificación del perfil de usuario:** Permitir a los usuarios actualizar su información personal, preferencias y configuración desde su panel principal.
- **Niveles de acceso para empleados:** Implementar diferentes niveles de permisos para los empleados (administrador, entrenador senior, entrenador junior, etc.) con acceso a distintas funcionalidades.
- **Sistema de notificaciones:** Crear un sistema de alertas y notificaciones para informar a los usuarios sobre cambios en sus rutinas, próximas revisiones o vencimiento de tarifas.
- **Gestión de ausencias:** Permitir a los clientes notificar ausencias temporales y a los empleados gestionar sus periodos de vacaciones o baja.

3. Funcionalidades para tarifas y pagos

- **Limitaciones en la contratación de tarifas:** Implementar restricciones basadas en el perfil del usuario, historial o disponibilidad del centro para la contratación de determinadas tarifas.
- **Sistema de pagos integrado:** Incorporar pasarelas de pago para facilitar la contratación y renovación de tarifas directamente desde la aplicación.
- **Descuentos y promociones:** Permitir la creación de ofertas temporales, códigos promocionales o descuentos por fidelidad.
- **Facturación automática:** Generar y enviar facturas electrónicas a los clientes tras cada pago o renovación de tarifa.
- **Recordatorios de renovación:** Notificar a los clientes cuando se aproxima la fecha de vencimiento de su tarifa actual.

4. Seguimiento y análisis

- **Gráficos de progreso:** Implementar visualizaciones gráficas del progreso físico de los clientes basadas en los datos de las revisiones.
- **Objetivos personalizados:** Permitir establecer metas específicas para cada cliente y realizar un seguimiento de su cumplimiento.
- **Análisis comparativo:** Ofrecer comparativas entre el progreso actual y periodos anteriores o entre diferentes clientes con perfiles similares.
- **Exportación de datos:** Permitir la exportación de informes y datos en diferentes formatos (PDF, Excel, CSV) para su análisis externo.

5. Integración y expansión

- **Aplicación móvil complementaria:** Desarrollar una versión para dispositivos móviles que permita a los clientes consultar sus rutinas y dietas, registrar actividades y comunicarse con sus entrenadores desde cualquier lugar.
- **Integración con wearables:** Conectar la aplicación con dispositivos como pulseras de actividad o relojes inteligentes para recopilar datos de entrenamiento automáticamente.
- **Calendario de actividades:** Implementar un sistema de reserva para clases grupales o sesiones con entrenadores personales.

- **Foro o comunidad:** Crear un espacio donde los usuarios puedan compartir experiencias, resolver dudas o participar en retos colectivos.
- **Contenido multimedia:** Incorporar vídeos demostrativos de ejercicios, tutoriales y consejos nutricionales para enriquecer la experiencia educativa.

6. Mejoras técnicas

- **Optimización del rendimiento:** Mejorar la eficiencia de las consultas a la base de datos y la carga de la interfaz gráfica para una experiencia más fluida.
- **Sincronización en la nube:** Implementar un sistema de respaldo y sincronización que permita acceder a los datos desde diferentes dispositivos.
- **Internacionalización:** Añadir soporte para múltiples idiomas y formatos regionales.
- **Accesibilidad:** Mejorar la compatibilidad con tecnologías de asistencia para usuarios con diversidad funcional.
- **Modo oscuro:** Implementar una alternativa de interfaz con fondo oscuro para reducir la fatiga visual.

Estas mejoras potenciales representan un camino de evolución para Fitness360, permitiendo que la aplicación crezca en funcionalidad y valor para sus usuarios, manteniendo siempre el enfoque en la usabilidad, personalización y eficiencia que caracterizan al proyecto actual.

15. Backlog del proyecto

El backlog del proyecto Fitness360 detalla la planificación y seguimiento de las tareas necesarias para completar el desarrollo de la aplicación.

Entrega Final: Martes 21 de mayo de 2025

- Fecha Límite de Desarrollo: Lunes 20 de mayo de 2025
- Inicio: Martes 23 de abril de 2025

Sprint 1: 23 al 27 de abril

Objetivo: Preparar entorno, base de datos y modelo inicial

- Crear estructura del proyecto Java con paquetes: model, dao, util, controller, view
- Crear la base de datos con el script SQL completo
- Implementar todas las clases modelo con sus atributos
- Configurar conexión JDBC con MySQL
- Crear clase DBUtil para gestionar la conexión a base de datos
- Crear primeras pruebas de conexión y consultas básicas

Sprint 2: 28 de abril al 4 de mayo

Objetivo: Crear DAOs e implementar CRUD básico

- DAO para Usuario, Cliente, Empleado
- DAO para Dieta, Rutina, Tarifa, Revisión
- DAO para tablas intermedias (Cliente_Dieta, etc.)
- CRUD en consola para probar inserción, lectura, modificación y eliminación

Sprint 3: 5 al 10 de mayo

Objetivo: Integrar JavaFX y crear las primeras interfaces

- Crear estructura de carpetas para FXML, css, etc.
- Ventana de Login + validación con hash de contraseña
- Registro de nuevo usuario con selección de tipo (cliente o profesional)
- Ventana principal según el tipo de usuario
- Carga de datos desde la BD con JavaFX (por ejemplo, tabla de rutinas asignadas)

Sprint 4: 11 al 15 de mayo

Objetivo: Funcionalidades avanzadas

- Asignar rutinas o dietas por parte del profesional
- Mostrar las revisiones asociadas a un cliente
- Crear nueva revisión (por parte del profesional)

Sprint Final: 16 al 20 de mayo

Objetivo: Pulir, probar y preparar la entrega

- Pruebas completas con distintos perfiles
- Validación de formularios y manejo de errores
- Documentación del código (comentarios + guía uso)
- Preparar entrega final: ZIP del proyecto + script BD

16. Conclusión

El proyecto Fitness360 se presenta como una solución integral, modular y escalable para la gestión de planes personalizados de entrenamiento y alimentación en centros de fitness. La elección de tecnologías consolidadas como Java, JavaFX y MySQL garantiza una base sólida y de fácil mantenimiento.

El diseño mediante MVC permite separar las capas del sistema, facilitando la comprensión y evolución futura del software. La seguridad se ve reforzada con el uso de contraseñas encriptadas y validaciones exhaustivas, mientras que el manejo adecuado de excepciones contribuye a una experiencia de usuario fluida y sin fallos inesperados.

Fitness360 no solo cubre las funcionalidades básicas de gestión, sino que también ofrece posibilidades de expansión, como la integración con aplicaciones móviles, visualización gráfica de progresos y conexión con dispositivos wearables.

Este proyecto sienta las bases para un sistema completo que puede adaptarse a las necesidades cambiantes de profesionales y usuarios del sector fitness, aportando valor a la salud y bienestar de sus usuarios.

17. Aportaciones de ChatGPT

Durante el desarrollo de este proyecto, ChatGPT ha aportado un soporte fundamental en diversas áreas técnicas y conceptuales, lo que ha facilitado un avance más fluido y sólido en la construcción de Fitness360. Entre las principales contribuciones destacan:

1. Explicación y soporte en la implementación del sistema de logging con Logback

ChatGPT ha explicado con detalle cómo integrar y configurar Logback, una potente biblioteca para la gestión de logs en aplicaciones Java, dentro del proyecto Fitness360. Se ha proporcionado orientación sobre:

La configuración del archivo logback.xml para definir distintos niveles de log (INFO, DEBUG, ERROR) y su almacenamiento en archivos de texto.

La manera de incluir logs contextuales y personalizar formatos para facilitar la posterior revisión y depuración del sistema.

La importancia de registrar eventos clave, errores SQL, excepciones personalizadas y flujos de acceso para mantener un historial completo y útil en entornos de producción y desarrollo.

Cómo usar SLF4J como fachada para Logback, permitiendo una implementación flexible y desacoplada del sistema de logging.

Esta aportación ha mejorado notablemente la capacidad de monitoreo, diagnóstico y mantenimiento del proyecto.

2. Apoyo en la implementación y manejo avanzado de tablas en JavaFX

Uno de los retos habituales al trabajar con interfaces gráficas en JavaFX es la correcta implementación y manipulación de tablas (TableView), para mostrar datos dinámicos y actualizables de la base de datos.

ChatGPT ha aportado explicaciones detalladas sobre:

La configuración de columnas usando TableColumn y la manera correcta de enlazarlas con las propiedades de los objetos del modelo.

El uso de ObservableList para mantener sincronizada la tabla con los cambios en los datos, permitiendo que la vista se actualice automáticamente sin necesidad de recargar la tabla manualmente.

La diferenciación entre métodos tradicionales y enfoques recomendados para manejar datos con propiedades observables en JavaFX.

3. Diferencia entre PropertyValueFactory y SimpleStringProperty para enlazar datos en JavaFX

Un punto técnico muy importante tratado fue la distinción y utilidad de los dos mecanismos más comunes para enlazar datos con columnas en una TableView:

PropertyValueFactory: Es una clase genérica que facilita la vinculación entre la columna y una propiedad del objeto modelo basada en convenciones de nombres de métodos getter. Permite que la tabla acceda automáticamente a los datos, siempre que el objeto tenga un método getX() correspondiente. Esta forma es sencilla y rápida, pero depende de que el objeto tenga métodos convencionales y no maneja automáticamente actualizaciones dinámicas en la vista.

SimpleStringProperty (y otras propiedades observables): Representan valores observables que permiten que JavaFX detecte cambios automáticamente. Cuando una clase de modelo devuelve propiedades observables (por ejemplo, SimpleStringProperty en lugar de un simple String), la tabla puede actualizarse en tiempo real si el valor cambia, lo que facilita la reactividad y mejora la experiencia de usuario. Esta técnica requiere un poco más de código, ya que implica definir las propiedades con sus métodos get(), set(), y property() correspondientes.

ChatGPT ha explicado cuándo es mejor usar uno u otro enfoque, dependiendo de la necesidad de reactividad de los datos y la complejidad del modelo, ayudando a tomar decisiones informadas para la implementación de la interfaz gráfica en el proyecto.

4. Explicación detallada del uso de JOIN y LEFT JOIN en SQL para consultas complejas

Otra de las aportaciones fundamentales de ChatGPT ha sido la explicación del funcionamiento y las diferencias prácticas entre los tipos de unión JOIN (específicamente INNER JOIN) y LEFT JOIN al momento de realizar consultas SQL entre múltiples tablas relacionadas.

Gracias a esta explicación, se ha podido implementar de forma correcta la recuperación de información relacionada entre entidades del modelo, como por ejemplo:

- Mostrar el nombre del entrenador o dietista asignado a un cliente.

- Listar las rutinas o dietas junto con los datos del profesional que las creó.
- Visualizar rutinas que podrían no estar aún asignadas a un empleado, usando LEFT JOIN para que aparezcan igualmente en los resultados.

Resumen de conceptos explicados:

JOIN (o INNER JOIN): Sólo devuelve las filas que tienen coincidencias en ambas tablas. Es útil cuando se desea trabajar únicamente con datos que tienen relaciones complejas entre tablas. Por ejemplo, al obtener clientes que sí tienen asignado un entrenador.

```
SELECT cd.*,
       c.idCliente, c.nombreUsuario, c.nombre, c.apellidos,
       c.correo,
       c.password, c.telefono, c.fechaNacimiento, c.sexo,
       c.altura,
       c.estado, c.createdAt, c.updatedAt,
       d.idDieta, d.nombre as nombreDieta, d.descripcion as
descripcionDieta,
       d.createdAt as createdAtDieta, d.updatedAt as
updatedAtDieta
FROM ClienteDieta cd
JOIN Cliente c ON cd.idCliente = c.idCliente
JOIN Dieta d ON cd.idDieta = d.idDieta
```

LEFT JOIN: Devuelve todas las filas de la tabla de la izquierda, incluso si no tienen coincidencia en la tabla derecha. Las columnas de la derecha aparecerán como NULL si no hay relación. Es útil para mostrar, por ejemplo, todos los usuarios aunque aún no tengan asignado un profesional.

```
private static final String SQL_FIND_BY_USUARIO =
    "SELECT ur.*, " +
    "c.idCliente, c.nombreUsuario, c.nombre,
    c.apellidos, c.correo, " +
```

```

        "c.password, c.telefono, c.fechaNacimiento,
c.sexo, c.altura, " +
        "c.estado, c.createdAt, c.updatedAt, " +
        "r.nombre AS nombreRutina, r.descripcion AS
descripcionRutina, " +
        "r.createdAt AS createdAtRutina, r.updatedAt
AS updatedAtRutina, " +
        "e.idEmpleado, e.nombre AS nombreEmpleado,
e.apellidos AS apellidosEmpleado " +
        "FROM UsuarioRutina ur " +
        "JOIN Cliente c ON ur.idUsuario =
c.idCliente " +
        "JOIN Rutina r ON ur.idRutina = r.idRutina "
+
        "LEFT JOIN Empleado e ON r.idEmpleado =
e.idEmpleado " +
        "WHERE ur.idUsuario = ?";

```

Gracias a estas explicaciones, se ha podido realizar un modelado de base de datos más flexible, con consultas SQL que manejan correctamente los casos de relaciones opcionales (por ejemplo, usuarios sin rutinas o sin dietas asignadas), lo que mejora la robustez y funcionalidad de la aplicación.