ILERNA

GESTIÓNGYM

Elena Mesa Requena
Optativa DAM
2º DAM

Documentación: Gestión Gym

Autor: Mesa Requena Elena

1. Resumen del proyecto

La aplicación GestiónGym está diseñada para uso exclusivo del gestor del gimnasio. Permite administrar clientes, aparatos, reservas de sesiones de 30 minutos, facturación mensual y control de pagos. Los clientes no acceden a la aplicación: las operaciones se realizan por el administrador.

2. Requisitos funcionales

- FR1 - Gestión de clientes: Alta, baja y modificación de clientes (nombre, DNI/NIF, email, teléfono, fecha de alta).
- FR2 - Gestión de aparatos: Alta, baja y modificación de aparatos (ID, tipo, descripción). Aunque haya aparatos iguales, cada uno se trata como distinto.
- FR3 - Reserva de sesiones: Reservar una sesión de 30 minutos para un cliente en un aparato concreto en una fecha y franja horaria (lunes a viernes). Validar solapamientos para ese aparato. Los minutos válidos para hora_inicio son 00 o 30.
- FR4 - Consulta de ocupación: Para un día determinado (lunes–viernes) listar todas las sesiones por aparato con cliente y franja horaria. El listado de ocupación puede mostrar huecos libres además de ocupados.
- FR5 - Generación de recibos mensuales: Generar recibos para todos los clientes con la cuota fija mensual (fecha del recibo, periodo, importe, estado “pendiente” inicialmente).
- FR6 - Registro de pagos: Registrar que un cliente ha pagado un recibo (fecha de pago, medio de pago, referencia).
- FR7 - Listado de morosos: Obtener listado de clientes con recibos impagados (filtro por mes o acumulado).
- FR8 - Listados y exportación: Exportar listados (ocupación diaria, morosos, clientes, facturación) a CSV.
- FR9 - Autenticación mínima: Usuario administrador (para la interfaz de gestión) con acceso protegido por contraseña (puede ser simple para la práctica).
- FR10 - Asistente de Reservas (Wizard): Sistema de reservas guiado en dos pasos (selección de cliente -> detalles) con validación visual de disponibilidad.
- FR11 - Buscadores en Tiempo Real: Filtrado instantáneo de clientes y pagos mediante introduccion de texto (nombre o DNI) sin recargas.
- FR12 - Exportación PDF: Generación de informes de ocupación y listados de morosos en formato PDF profesional.

3. Requisitos no funcionales

- NFR1: Implementación en Python.
- NFR2: Persistencia con SQLite.
- NFR3: Interfaz Gráfica de Usuario (GUI) moderna implementada en Python utilizando CustomTkinter y el patrón MVC (Modelo-Vista-Controlador).
- NFR4: Documentación y pruebas mínimas incluidas.
- NFR5: Código con estilo PEP8 y comentado.

4. Casos de uso

Actores

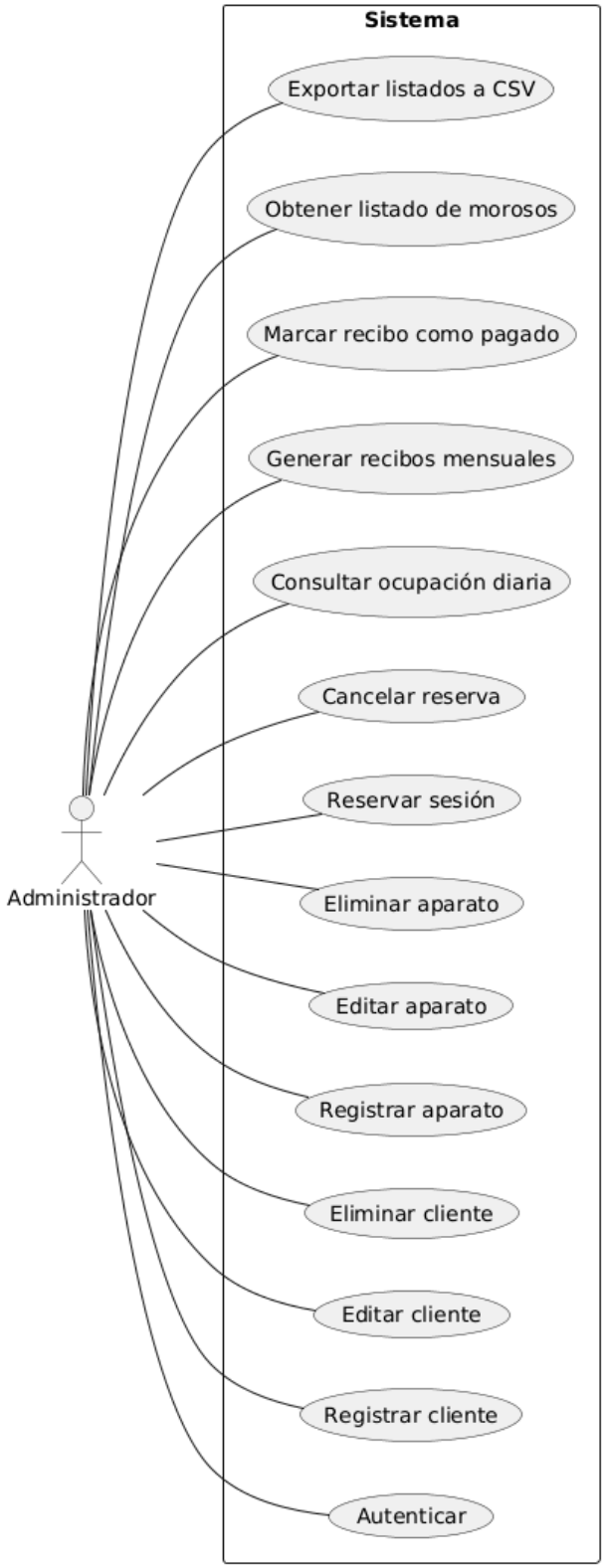
Administrador: gestiona clientes, aparatos, reservas, genera recibos y consulta morosos.

Sistema: tareas automáticas (generación masiva de recibos).

Lista de casos de uso (títulos)

- CU01: Autenticar administrador
- CU02: Registrar cliente
- CU03: Editar cliente
- CU04: Eliminar cliente
- CU05: Registrar aparato
- CU06: Editar aparato
- CU07: Eliminar aparato
- CU08: Reservar sesión
- CU09: Cancelar reserva
- CU10: Política de cancelación.
- CU11: Consultar ocupación diaria
- CU12: Generar recibos mensuales
- CU13: Marcar recibo como pagado
- CU14: Obtener listado de morosos
- CU15: Exportar listados a CSV

Diagrama de Casos de Uso (PlantUML)



5. Modelo E-R y normalización

Entidades principales

```
- Cliente(cliente_id, dni, nombre, apellido, email, telefono, fecha_alta)
- Aparato(aparato_id, codigo, tipo, descripcion)
- Sesion(sesion_id, aparato_id, cliente_id, fecha, hora_inicio, duracion,
created_by)
- Recibo(recibo_id, cliente_id, periodo_anyo, periodo_mes, fecha_generacion,
importe, estado)
- Pago(pago_id, recibo_id, fecha_pago, metodo, referencia)
- Usuario(usuario_id, username, password_hash, rol)

Relaciones principales:
- Cliente 1:N Sesion
- Aparato 1:N Sesion
- Cliente 1:N Recibo
- Recibo 1:1 Pago
- Usuario 1:N Sesion (el gestor crea las reservas)
```

Reglas/consideraciones

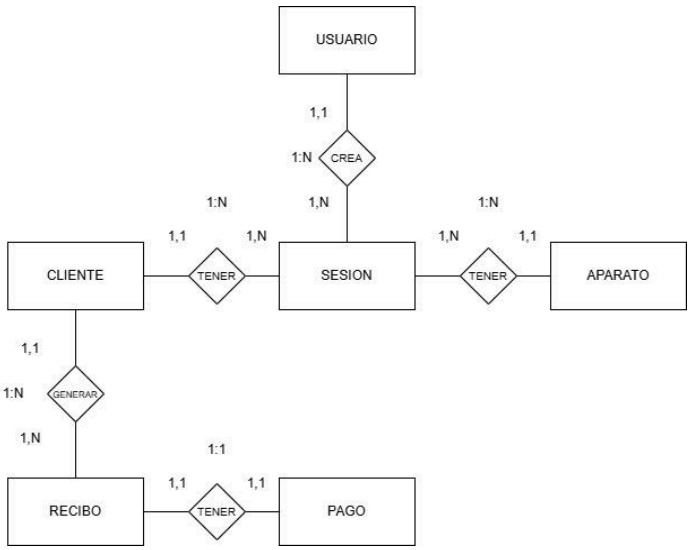
Cada sesión dura 30 minutos. Guardamos hora_inicio (por ejemplo 14:00) y duracion implícita de 30 minutos.

Mismo tipo de aparato pero instancias distintas se consideran distintos aparatos (cada uno con aparato_id).

Minutos válidos: 00 o 30; horario 00:00–23:30; días L–V (validado en aplicación).

Normalización

Todas las tablas están en 3ª Forma Normal (3NF): atributos atómicos, claves primarias simples, no hay dependencias transitivas.



6. Esquema SQL (DDL) - SQLite

```
PRAGMA foreign_keys = ON;

CREATE TABLE Cliente (
  cliente_id INTEGER PRIMARY KEY AUTOINCREMENT,
  dni TEXT UNIQUE NOT NULL,
  nombre TEXT NOT NULL,
  apellido TEXT NOT NULL,
  email TEXT,
  telefono TEXT,
  fecha_alta DATE NOT NULL
);

CREATE TABLE Aparato (
  aparato_id INTEGER PRIMARY KEY AUTOINCREMENT,
  codigo TEXT UNIQUE,
  tipo TEXT NOT NULL,
  descripcion TEXT
);

CREATE TABLE IF NOT EXISTS Usuario (
  usuario_id INTEGER PRIMARY KEY AUTOINCREMENT,
  username TEXT UNIQUE NOT NULL,
  password_hash TEXT NOT NULL,
  rol TEXT NOT NULL CHECK (rol IN ('admin'))
);

CREATE TABLE IF NOT EXISTS Sesion (
  sesion_id INTEGER PRIMARY KEY AUTOINCREMENT,
  aparato_id INTEGER NOT NULL,
  cliente_id INTEGER NOT NULL,
  fecha DATE NOT NULL,
  hora_inicio TEXT NOT NULL,
  duracion INTEGER NOT NULL DEFAULT 30,
  created_by INTEGER,
  FOREIGN KEY (aparato_id) REFERENCES Aparato(aparato_id) ON DELETE CASCADE,
  FOREIGN KEY (cliente_id) REFERENCES Cliente(cliente_id) ON DELETE CASCADE,
  FOREIGN KEY (created_by) REFERENCES Usuario(usuario_id) ON DELETE SET NULL,
  UNIQUE (aparato_id, fecha, hora_inicio),
  CHECK (substr(hora_inicio,4,2) IN ('00','30')),
  CHECK (CAST(substr(hora_inicio,1,2) AS INTEGER) BETWEEN 0 AND 23),
  CHECK (duracion = 30)
);

CREATE TABLE Recibo (
  recibo_id INTEGER PRIMARY KEY AUTOINCREMENT,
  cliente_id INTEGER NOT NULL,
  periodo_anyo INTEGER NOT NULL,
  periodo_mes INTEGER NOT NULL, -- 1..12
  fecha_generacion DATE NOT NULL,
  importe REAL NOT NULL,
  estado TEXT NOT NULL CHECK (estado IN ('pendiente','pagado')),
  FOREIGN KEY (cliente_id) REFERENCES Cliente(cliente_id) ON DELETE CASCADE,
  UNIQUE (cliente_id, periodo_anyo, periodo_mes)
```

```
);

CREATE TABLE Pago (
  pago_id INTEGER PRIMARY KEY AUTOINCREMENT,
  recibo_id INTEGER NOT NULL,
  fecha_pago DATE NOT NULL,
  metodo TEXT,
  referencia TEXT,
  FOREIGN KEY (recibo_id) REFERENCES Recibo(recibo_id) ON DELETE CASCADE
);

CREATE UNIQUE INDEX IF NOT EXISTS uq_pago_recibo ON Pago(recibo_id);
CREATE INDEX IF NOT EXISTS idx_sesion_aparato_fecha ON Sesion(aparato_id,
fecha, hora_inicio);
CREATE INDEX IF NOT EXISTS idx_recibo_cliente_periodo ON Recibo(cliente_id,
periodo_anyo, periodo_mes);
```

7. Interfaz en Python (diseño y esqueleto)

Enfoque Implementado: La aplicación sigue una arquitectura MVC (Modelo-Vista-Controlador) para separar la lógica de la interfaz.

- Modelo (model/): Gestiona la base de datos y las operaciones CRUD (ej: cliente.py, sesion.p).
- Vista (view/): Implementada con CustomTkinter, ofrece una interfaz moderna con modo Claro/Oscuro. Incluye navegación lateral y paneles dedicados para cada módulo.
- Controlador (controller/): Intermediario que procesa las acciones del usuario y actualiza la vista (ej: sesion_controller.py).

Tecnologías Clave:

- customtkinter: Para la interfaz gráfica principal.
- tkcalendar: Para selectores de fecha (DateEntry).
- reportlab: Para la generación de informes PDF.
- ttk.Treeview: Para visualización de datos en tablas.

Ejemplo de main.py (esqueleto)

```
from model.conexion import crear_tablas

from controller.aparato_controller import inicializar_aparatos_por_defecto

from view.app import App

def main():

    # 1. Inicialización de BD y datos semilla

    crear_tablas()
```

```
    inicializar_aparatos_por_defecto()

    # 2. Lanzamiento de la Interfaz Gráfica

    app = App()

    app.mainloop()

if __name__ == "__main__":

    main()
```

Ejemplo corto de funciones clave

```
# db.py
import sqlite3
from pathlib import Path

DB_PATH = Path('gestiongym.db')

def get_conn():
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    conn.execute('PRAGMA foreign_keys = ON;')
    return conn

def init_db():
    conn = get_conn()
    with open('schema.sql', 'r', encoding='utf-8') as f:
        conn.executescript(f.read())
    conn.commit()
    conn.close()

# cli.py (fragmento)
from db import get_conn

def main_menu():
    while True:
        print('\n== GestiónGym ==')
        print('1. Gestionar clientes')
        print('2. Gestionar aparatos')
        print('3. Reservas')
        print('4. Cobros y recibos')
        print('5. Listados / Exportar')
        print('0. Salir')
        opt = input('> ')
        if opt == '0':
            break
        # mapear a funciones…

# Ejemplo de login:
import getpass, hashlib
from db import get_conn

def login():
```

```
    user = input('Usuario: ').strip()
    pwd = getpass.getpass('Contraseña: ')
    h = hashlib.sha256(pwd.encode()).hexdigest()
    with get_conn() as conn:
        row = conn.execute('SELECT usuario_id, username FROM Usuario WHERE
username=? AND password_hash=?', (user, h)).fetchone()
    if row:
        print(f'Bienvenido, {row[1]}')
        return dict(usuario_id=row[0], username=row[1])
    print('Credenciales incorrectas')
    return None

# Ejemplo de creación de sesión (reserva):
def crear_sesion(conn, aparato_id, cliente_id, fecha, hora, current_user_id):
    conn.execute('INSERT INTO Sesion(aparato_id, cliente_id, fecha,
hora_inicio, created_by) VALUES(?,?,?,?,?)',
                 (aparato_id, cliente_id, fecha, hora, current_user_id))
```

8. Estructura del repositorio y README

```
GestiónGym_MesaRequenaElena/

├─ main.py                  # Punto de entrada de la aplicación

├─ requirements.txt         # Dependencias (customtkinter, reportlab, etc.)

├─ README.md                # Instrucciones de uso

├─ gestiongym.db            # Base de datos SQLite

├─ model/

│  ├─ conexion.py           # Gestión de conexión SQLite

│  ├─ cliente.py

│  ├─ aparato.py

│  └─ (...)

├─ view/

│  └─ app.py                # Clase principal de la GUI (Ventanas y tabs)

├─ controller/

│  ├─ cliente_controller.py

│  ├─ sesion_controller.py  # Lógica de reservas

│  └─ (...)

├─ resources/

│  └─ logo.png              # Recursos gráficos

└─ utils/                   # Utilidades varias
```

Contenido sugerido para README.md: Descripción del proyecto, requisitos (versión de Python), cómo inicializar (crear BD con `python db.py --init` o ejecutar `schema.sql`), cómo ejecutar la aplicación, estructura del repo, autor y fecha de entrega (10 de diciembre 23:59).

9. Plan de entregables y checklist

Documento de requisitos y diseño, código implementado (CRUD, reservas, recibos, pagos), tests unitarios, README y schema.sql, datos de ejemplo, repositorio en GitHub.

Sugerencia de milestones: Día 1–2 (Modelado y DDL + estructura), Día 3–6 (CRUD y reservas), Día 7–9 (recibos y pagos), Día 10–12 (pruebas, documentación y subida).

Checklist: Usuario creado, validadores de slot/día activos, cancelación implementada, diagramas incrustados, índices creados.

10. Tests y datos de prueba

```
- Probar creación de clientes y aparatos
- Reservar sesiones en el mismo aparato en horas distintas -> OK
- Intentar reservar sesión en el mismo aparato y misma franja -> debe fallar
(UNIQUE)
- Generar recibos mensuales y marcar algunos como pagados -> comprobar listado
de morosos
- Exportar CSV y validar columnas
```

Anexos

PlantUML para diagrama de casos de uso (incluido arriba).

SQL DDL (schema.sql) ya provisto (copiar al fichero schema.sql).

15. Posibles mejoras futuras

1. Versión Web/Móvil: Migrar el backend a una API REST (Flask/FastAPI) para permitir acceso desde dispositivos móviles.
2. Control de Acceso Físico: Integración con lectores NFC o tornos para validar la entrada al gimnasio automáticamente.
3. Notificaciones Automáticas: Envío de correos electrónicos a clientes recordándoles sus reservas o avisando de impagos.
4. Sistema de Socios: Implementar niveles de suscripción con diferentes privilegios de reserva.

16. Conclusiones

La aplicación GestiónGym digitaliza la administración de un gimnasio, automatizando reservas, facturación y control de pagos. El diseño modular en Python, con SQLite y validación de datos, permite un mantenimiento sencillo y extensible. La trazabilidad de acciones a través de la relación Usuario–Sesión permite saber qué gestor realizó cada reserva, garantizando control y responsabilidad en la administración diaria.

17. Bibliografía

- Documentación oficial de Python: https://docs.python.org/3/
- Documentación de SQLite: https://sqlite.org/docs.html
- Documentación PlantUML: https://plantuml.com/
- PEP 8 – Guía de estilo de código Python: https://peps.python.org/pep-0008/