Documentación proyecto final: Power Core Gym

Carlos Daniel Correa Morales

Mileidys Claret Cortes Santos

Docente: Elkin Lopez

Bootcapm: Programación - Explorador

Talento Tech

15 de octubre del 2025

Introducción

Power Core Gym es una aplicación web completa para la gestión de membresías de un gimnasio. El sistema permite a los usuarios registrarse, iniciar sesión, seleccionar planes de membresía y gestionar su perfil.

Tecnologías Utilizadas

- **Backend**: Flask (Python)
- **Base de Datos**: PostgreSQL
- **Frontend**: HTML5, CSS3, JavaScript (Vanilla)
- **ORM**: SQLAlchemy
- **Seguridad**: Werkzeug (hash de contraseñas)

Arquitectura del Sistema

Patrón de Diseño

El sistema utiliza una arquitectura **MVC (Modelo-Vista-Controlador)**:

- **Modelo** (models.py): Define la estructura de datos y lógica de negocio
- **Vista** (Templates HTML): Presenta la información al usuario
- **Controlador** (app.py): Maneja las peticiones y coordina modelo/vista

Flujo de Datos

Arquitectura del sistema

Cliente (Navegador)
↓
JavaScript (script.js)
↓
API REST (app.py)
↓
SQLAlchemy ORM
↓
PostgreSQL Database

Requisitos del Sistema

Software

- **Python** 3.8+
- **PostgreSQL** 12+
- **pip** (gestor de paquetes de Python)
- **Navegador web moderno** (Chrome, Firefox, Edge)

Dependencias Python

- Flask 2.3.0
- Flask-SQLAlchemy 3.0.0
- psycopg2-binary 2.9.6
- Werkzeug 2.3.0

Instalación y Configuración

Paso 1: Preparar el Entorno

- **Crear directorio del proyecto:**

bash

```
mkdir power-core-gym

cd power-core-gym
```

- **Crear entorno virtual:**

bash

```
python -m venv venv
```

- **Activar entorno virtual:**

Windows:

bash

```
venv\Scripts\activate
```

Linux/Mac:

bash

```
source venv/bin/activate
```

Paso 2: Instalar Dependencias

bash

```
pip install Flask==2.3.0
pip install Flask-SQLAlchemy==3.0.0
pip install psycopg2-binary==2.9.6
pip install Werkzeug==2.3.0
```

Paso 3: Configurar PostgreSQL

1. **Crear la base de datos:**

sql

```
CREATE DATABASE init_db;
```

2. **Configurar credenciales en** app.py:

python

```
app.config['SQLALCHEMY_DATABASE_URI']='postgresql://postgres:TU_PASSWORD@localhost/init_db'
```

IMPORTANTE: Reemplaza TU_PASSWORD con tu contraseña real de PostgreSQL.

Paso 4: Estructura de Carpetas

```
power-core-gym/
│
├── app.py                      # Aplicación principal Flask
├── models.py                   # Modelos de base de datos
├── init_db.sql                 # Script de inicialización (vacío)
│
├── static/                     # Archivos estáticos
│   ├── estilos.css             # Hojas de estilo
│   └── script.js               # JavaScript
│
└── templates/                  # Plantillas HTML
    ├── Power_core-index.html
    └── registro.html
```

Paso 5: Iniciar la Aplicación

```
bash
python app.py
```

El servidor iniciará en: http://localhost:3000

Estructura de Archivos

app.py - Aplicación Principal

Propósito: Controlador principal que maneja rutas y lógica de backend.

Componentes principales:

```
python
# Configuración
app.config['SECRET_KEY'] = 'clave_secreta'
app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://
...'
# Inicialización
db.init_app(app)
# Rutas principales
@app.route('/')                          # Página principal
@app.route('/registro')                   # Página de registro
@app.route('/api/login', methods=['POST'])   # API login
@app.route('/api/registro', methods=['POST']) # API registro
```

Funcionalidades clave:

- Gestión de sesiones de usuario
- Validación de datos
- Comunicación con base de datos

- Respuestas JSON para AJAX

`models.py` - Modelos de Datos

Propósito: Define la estructura de la tabla de usuarios y métodos asociados.

Modelo Usuario:

```
python
class Usuario(db.Model):
 __tablename__ = 'usuarios'

# Campos principales
id = db.Column(db.Integer, primary_key=True)
nombre_perfil = db.Column(db.String(50), unique=True, nullable=False)
nombre = db.Column(db.String(100), nullable=False)
apellido = db.Column(db.String(100), nullable=False)
email = db.Column(db.String(120), unique=True, nullable=False)
password_hash = db.Column(db.String(255), nullable=False)
plan = db.Column(db.String(20))

 # Métodos
def set_password(self, password)      # Hashear contraseña
def check_password(self, password)    # Verificar contraseña
def to_dict(self)                     # Convertir a JSON
```

estilos.css - Estilos

Estructura:

1. **Estilos Generales**: Reset CSS, fuentes, colores base
2. **Animaciones**: Keyframes para efectos visuales
3. **Navegación**: Barra superior con menús
4. **Carrusel**: Slider de promociones
5. **Características**: Secciones de contenido
6. **Planes**: Tarjetas de membresía
7. **Autenticación**: Formularios de login/registro
8. **Responsive**: Media queries para móviles

Animaciones principales:

```
css
@keyframes deslizarDesdeIzquierda { ... }
@keyframes zoomIn { ... }
@keyframes pulso { ... }
```

```
@keyframes brillo { ... }
```

script.js - JavaScript

Módulos principales:

1. **Carrusel**: Navegación automática de slides
2. **Animaciones Scroll**: Intersection Observer
3. **Formularios**: Validación y envío AJAX
4. **Autenticación**: Login/registro
5. **Perfil de Usuario**: Gestión de cuenta
6. **Easter Egg**: Código Konami para cupones

Funciones clave:

```javascript
moverCarrusel(direccion)          // Navegación carrusel
cambiarFormulario(tipo)           // Alternar login/registro
mostrarMensaje(mensaje, tipo)     // Notificaciones toast
verificarNombrePerfil()           // Validación en tiempo real
```

Base de Datos

Esquema de la Tabla usuarios

Campo	Tipo	Restricciones	Descripción
id	INTEGER	PRIMARY KEY	Identificador único
nombre_perfil	VARCHAR(50)	UNIQUE, NOT NULL	Nombre de usuario
nombre	VARCHAR(100)	NOT NULL	Nombre real
apellido	VARCHAR(100)	NOT NULL	Apellido
email	VARCHAR(120)	UNIQUE, NOT NULL	Correo electrónico
telefono	VARCHAR(20)		Número telefónico
fecha_nacimiento	DATE		Fecha de nacimiento
genero	VARCHAR(20)		Género del usuario
password_hash	VARCHAR(255)	NOT NULL	Contraseña hasheada
foto_perfil	VARCHAR(500)	DEFAULT	URL de foto
plan	VARCHAR(20)		Plan de membresía
fecha_registro	DATETIME	DEFAULT NOW	Fecha de registro
activo	BOOLEAN	DEFAULT TRUE	Estado del usuario

Índices

```sql
CREATE UNIQUE INDEX idx_nombre_perfil ON usuarios(nombre_perfil);
CREATE UNIQUE INDEX idx_email ON usuarios(email);
```

Frontend

Páginas Principales

1. Power_core-index.html - Página Principal

Secciones:

- **Navegación**: Menú superior con enlaces
- **Carrusel**: 3 slides con promociones
- **Equipamiento**: Galería de instalaciones
- **Entrenadores**: Tarjetas de staff
- **Clases**: Tipos de entrenamientos
- **Planes**: Opciones de membresía
- **Footer**: Información de contacto

Características interactivas:

- Carrusel automático con navegación manual
- Animaciones al hacer scroll
- Smooth scroll en enlaces
- Menú de usuario (si está logueado)

2. registro.html - Autenticación

Componentes:

- **Formulario Login**: Email y contraseña
- **Formulario Registro**: Datos completos del usuario
- **Validaciones**: En tiempo real
- **Botones sociales**: Google y Facebook (placeholder)
- **Panel lateral**: Imagen motivacional

Validaciones implementadas:

- Email válido
- Contraseñas coincidentes
- Longitud mínima de contraseña (6 caracteres)
- Nombre de perfil único
- Edad mínima (16 años)

Backend - API

Endpoints Disponibles

1. `GET /` **- Página Principal**

```
Retorna: Power_core-index.html
```

2. `GET /registro` **- Página de Registro**

```
Parámetros opcionales:
- plan: string (smart|fit|black)
Retorna: registro.html
```

3. `POST /api/verificar-nombre-perfil` **– Verificar Disponibilidad**

Request:

```
json
{
"nombre_perfil": "usuario123"
}
```

Response:

```
json
{
"disponible": true,
"mensaje": "Nombre de perfil disponible"
}
```

4. `POST /api/login` **– Iniciar Sesión**

Request:

```
json
{
 "email": "usuario@example.com",  "password": "mipassword"

 }
```

Response (éxito):

```
json
{
"success": true,
"mensaje": "¡Inicio de sesión exitoso!",
"usuario": {
"id": 1,
"nombre_perfil": "usuario123",
"nombre": "Juan",
"apellido": "Pérez",
"email": "usuario@example.com",
"plan": "black"
  }
}
```

Response (error):

```
json
{
"success": false,
"mensaje": "Email o contraseña incorrectos"
}
```

5. `POST /api/registro` **– Registrar Usuario**

Request:

```
json
{
"nombre_perfil": "usuario123",
"nombre": "Juan",
"apellido": "Pérez",
"email": "usuario@example.com",
"telefono": "+57 300 123 4567",
"fecha_nacimiento": "1990-01-15",
"genero": "masculino",
"password": "mipassword",
"plan": "black"
}
```

Response (éxito):

```
json
{
"success": true,
"mensaje": "¡Registro exitoso! Bienvenido a Power Core",
"usuario": { ... }
}
```

6. `GET /api/usuario-actual` – Usuario Logueado

Response (logueado):

```
json
{  "logueado": true,
"usuario": { ... }
}
```

Response (no logueado):

```
json
{
"logueado": false
}
```

7. `POST /api/cambiar-password` – Cambiar Contraseña

Request:

```
json
{
"password_actual": "viejapass",
"password_nueva": "nuevapass",
"password_confirmar": "nuevapass"
}
```

8. `POST /api/actualizar-foto` – Actualizar Foto de Perfil

Request:

```
json
{
"foto_url": "https://example.com/foto.jpg"
}
```

9. `POST /api/logout` – Cerrar Sesión

Response:

```
json
{
"success": true,
"mensaje": "Sesión cerrada"
}
```

Funcionalidades Principales

1. Sistema de Registro

Flujo:

1. Usuario completa formulario
2. JavaScript valida datos en frontend
3. Se verifica disponibilidad de nombre de perfil
4. Se envía petición POST a /api/registro
5. Backend valida y hashea contraseña
6. Se crea registro en base de datos
7. Se inicia sesión automáticamente

Validaciones:

- Nombre de perfil único (mínimo 3 caracteres)
- Email válido y único
- Contraseñas coincidentes (mínimo 6 caracteres)
- Todos los campos requeridos completos
- Edad mínima de 16 años

2. Sistema de Autenticación

Inicio de sesión:

Javascript

```
// Frontend
fetch('/api/login', {
method: 'POST',
headers: { 'Content-Type': 'application/json' },
body: JSON.stringify({ email, password })})
```

Backend:

python

```
# Verificar credenciales
usuario = Usuario.query.filter_by(email=email).first()
if usuario.check_password(password):
session['user_id'] = usuario.id
```

1. **Gestión de sesiones:**

- Uso de Flask sessions con cookies seguras

- Verificación de sesión en cada request protegido
- Timeout automático de sesión

2. Gestión de Perfil

Funcionalidades:

- Ver información personal
- Cambiar foto de perfil
- Actualizar contraseña
- Ver plan actual

Modales implementados:

- Modal de perfil (información completa)
- Modal de cambio de contraseña
- Menú desplegable de usuario

3. Selección de Planes

Planes disponibles:

1. **Plan Smart** - $59.900/mes
    - Acceso 6 AM - 4 PM
    - Sede principal
    - 5 clases grupales/mes
2. **Plan Fit** - $89.900/mes
    - Acceso 5 AM - 10 PM
    - Sede principal
    - Clases ilimitadas
3. **Plan Black** - $159.900/mes (destacado)
    - Acceso 24/7
    - Todas las sedes
    - Entrenador personal
    - Sauna y vapor

Características Especiales

1. Código Konami (Easter Egg)

Secuencia: ↑↑↓↓←→←→BA

Funcionalidad:

- Al ingresar la secuencia, se activan cupones secretos
- Muestra modal con 6 cupones exclusivos
- Animación de confetti
- Copiar código al portapapeles

Cupones incluidos:

- POWERCORE2025: 1 mes gratis Plan Black
- KONAMI50: 50% descuento por 3 meses
- TRAINER5: 5 sesiones con entrenador
- AMIGO2X1: 2×1 en planes
- KITGRATIS: Kit deportivo gratis
- ULTRA6: 6 meses al precio de 4

Implementación:

```
javascript
const secuenciaKonami = [38, 38, 40, 40, 37, 39, 37, 39, 66, 65];

document.addEventListener('keydown', function(e) {
 codigoKonami.push(e.keyCode);
if (codigoKonami.join(',') === secuenciaKonami.join(',')) {
activarCupones();
    }
});
```

2. Verificación en Tiempo Real

Nombre de perfil:

- Verifica disponibilidad mientras el usuario escribe
- Timeout de 500ms para evitar requests excesivos
- Feedback visual inmediato (verde/rojo)

```javascript
nombrePerfilInput.addEventListener('input', function() {
clearTimeout(timeoutVerificacion);
timeoutVerificacion = setTimeout(function() {
      fetch('/api/verificar-nombre-perfil', { ... })
   }, 500);
});
```

3. Animaciones y Efectos

Intersection Observer:

- Detecta cuando elementos entran en viewport
- Activa animaciones de entrada
- Optimizado para rendimiento

Animaciones CSS:

- Deslizamiento desde laterales
- Zoom in
- Pulso
- Brillo
- Rotación suave

4. Responsive Design

Breakpoints:

- Desktop: > 1024px
- Tablet: 768px - 1024px
- Mobile: < 768px
- Small mobile: < 480px

Adaptaciones:

- Navegación colapsable en móviles
- Carrusel touch-friendly
- Formularios en columna única
- Imágenes optimizadas

Seguridad

1. Hash de Contraseñas

Werkzeug Security:

```
python
from werkzeug.security import generate_password_hash, check_password_hash
# Al registrar
password_hash = generate_password_hash(password)
# Al verificar
check_password_hash(password_hash, password_ingresada)
```

Características:

- Algoritmo PBKDF2 con SHA256
- Salt automático
- Múltiples iteraciones
- Irreversible

2. Validación de Datos

Frontend:

- Validación HTML5 (required, type, pattern)
- Validación JavaScript personalizada
- Sanitización de inputs

Backend:

- Validación de tipos de datos
- Verificación de campos requeridos
- Prevención de duplicados
- Escape de SQL (SQLAlchemy ORM)

3. Sesiones Seguras

```
python
app.config['SECRET_KEY'] = 'clave_secreta_compleja'
```

Recomendaciones:

- Cambiar SECRET_KEY en producción
- Usar variable de entorno

- Generar clave aleatoria fuerte

3. Protección CSRF

Implementado:

- Tokens en formularios (Flask-WTF recomendado)
- Verificación de origin en AJAX
- SameSite cookies

4. SQL Injection

Prevención:

- Uso de SQLAlchemy ORM (parametrizado)
- Nunca concatenar SQL manualmente
- Validación de tipos

Troubleshooting

Problema 1: Error al Conectar Base de Datos

Error:

```
sqlalchemy.exc.OperationalError: could not connect to server
```

Solución:

1. **Verificar que PostgreSQL esté corriendo:**

```
bash
# Windows
services.msc → buscar PostgreSQL
# Linux
sudo systemctl status postgresql
```

2. **Verificar credenciales en app.py:**

```
python
'postgresql://usuario:password@localhost/init_db'
```

3. **Verificar que la base de datos exista:**

```
sql
\l  -- Listar bases de datos en psql
```

Problema 2: Módulos No Encontrados

Error:

```
ModuleNotFoundError: No module named 'flask'
```

Solución:

```
bash
# Activar entorno virtual
source venv/bin/activate  # Linux/Mac
venv\Scripts\activate     # Windows
# Reinstalar dependencias
pip install -r requirements.txt
```

Problema 3: Puerto Ya en Uso

Error:

```
OSError: [Errno 48] Address already in use
```

Solución:

```
bash
# Encontrar proceso usando puerto 3000
lsof -i :3000  # Linux/Mac
netstat -ano | findstr :3000  # Windows
# Matar proceso
kill -9 PID  # Linux/Mac
taskkill /PID PID /F  # Windows
# O cambiar puerto en app.py
app.run(port=5000)
```

Problema 4: Tablas No Creadas

Error:

```
sqlalchemy.exc.ProgrammingError: relation "usuarios" does not exist
```

Solución:

```
python
# En Python shell
from app import app, db
with app.app_context():
    db.create_all()
```

Problema 5: Session KeyError

Error:

```
KeyError: 'user_id'
```

Solución:

```
python
# Usar .get() en lugar de acceso directo
user_id = session.get('user_id')
if user_id:
# Usuario logueado
```

- Verificar que SECRET_KEY esté configurada
- Limpiar cookies del navegador
- Verificar que la sesión se esté guardando correctamente

Mantenimiento

Backup de Base de Datos

Crear backup:

```
bash
pg_dump -U postgres init_db > backup.sql
```

Restaurar backup:

```
bash
psql -U postgres init_db < backup.sql
```

Actualizar Dependencias

```
bash
pip list --outdatedpip
install --upgrade flask
pip freeze > requirements.txt
```

Logs y Monitoreo

Habilitar logs:

python
```
import logging
logging.basicConfig(level=logging.DEBUG)
```

Log personalizado:

python
```
app.logger.info('Usuario registrado: %s', usuario.nombre_perfil)
app.logger.error('Error en login: %s', str(e))
```

Limpieza de Sesiones Expiradas

python
```
# Implementar limpieza periódica
from datetime import datetime, timedelta

def limpiar_sesiones_viejas():
fecha_limite = datetime.utcnow() - timedelta(days=30)
usuarios_inactivos = Usuario.query.filter(
   Usuario.ultima_actividad < fecha_limite
).all()
 # Implementar lógica de limpieza
```

Comandos Útiles

Desarrollo

bash
```
# Iniciar servidor de desarrollo
python app.py
# Consola interactiva de Flask
flask shell
# Crear base de datos
flask db init
flask db migrate
flask db upgrade
```

PostgreSQL

bash
```
# Conectar a PostgreSQL
psql -U postgres
# Listar bases de datos\l
# Conectar a base de datos
\c init_db# Listar tablas\dt
# Describir tabla
\d usuarios
```

```
# Ejecutar query
SELECT * FROM usuarios;
```

Git (Control de Versiones)

```
bash
git init
git add .
git commit -m "Implementación inicial Power Core Gym"
git branch -M main
git remote add origin https://github.com/usuario/power-core-gym.git
git push -u origin main
```

Conclusión

Power Core Gym es una aplicación web completa y funcional que demuestra:

- **Backend robusto** con Flask y PostgreSQL
- **Frontend moderno** con animaciones y responsive design
- **Seguridad** con hash de contraseñas y validaciones
- **UX optimizada** con feedback en tiempo real
- **Código mantenible** con separación de responsabilidades

Próximas Mejoras Sugeridas

1. **Integración de pagos** (Stripe, PayPal)
2. **Panel administrativo** para gestionar usuarios
3. **Sistema de reservas** para clases
4. **Notificaciones por email**
5. **Upload de imágenes** para fotos de perfil
6. **Dashboard de métricas** para usuarios
7. **Chat en vivo** con soporte
8. **App móvil** nativa (React Native, Flutter)