

# UNIVERSIDAD DE GUADALAJARA

## CENTRO UNIVERSITARIO DE LOS VALLES



Lopez\_Avelino\_Manuel\_Adrian  
Desarrollo de bases de datos web  
LIC. Tecnologías de la información

## Cambios Implementados

### 1. Reversión a PyMySQL Funcional

Se eliminó Flask-SQLAlchemy y se restauró la arquitectura original del proyecto que usa PyMySQL directamente para las operaciones CRUD de los juegos. Esto se hizo para resolver los persistentes errores de `RuntimeError` y las importaciones circulares que impedían la ejecución. Se restauraron los archivos `bd.py` y `controlador_juegos.py`.

### 2. Integración de Autenticación con Flask-Login

Se añadió la librería Flask-Login para gestionar las sesiones de usuario (inicio de sesión, cierre de sesión) y proteger las rutas. Esto incluyó:

- Configuración de `LoginManager` en `app.py`.
- Implementación de la función `load_user` para cargar usuarios desde la (simulada) base de datos.
- Uso de funciones como `login_user()`, `logout_user()`, y `current_user`.

### 3. Estructura Modular con Blueprints

Se creó un Blueprint llamado `auth` para agrupar todas las rutas y la lógica relacionadas con la autenticación (`/login`, `/register`, `/logout`). Esto organiza el código y lo separa de la lógica principal de los juegos. La estructura ahora incluye una carpeta `auth/` con `__init__.py`, `forms.py`, y `routes.py`.

### 4. Modelo de Usuario Híbrido

Se creó un archivo `models.py` con una clase `User` que hereda de `UserMixin` (requerido por Flask-Login). Esta clase incluye métodos para hashear (`set_password`) y verificar (`verify_password`) contraseñas usando `werkzeug.security`, aunque no es un modelo ORM completo. También se crearon métodos estáticos (`get_by_id`, `get_by_username`) que llaman al `controlador_usuarios.py` para interactuar con la base de datos PyMySQL.

### 5. Formularios con Flask-WTF

Se definieron formularios específicos para el registro (`RegistrationForm`) y el inicio de sesión (`LoginForm`) en `auth/forms.py`, utilizando validadores como `DataRequired`, `Length`, `EqualTo` y una validación personalizada para evitar nombres de usuario duplicados. Se aseguró la protección CSRF mediante `form.hidden_tag()` en las plantillas.

## 6. Protección de Rutas

Se añadió el decorador `@login_required` a las rutas del CRUD de juegos (`/agregar_juego`, `/guardar_juego`, `/eliminar_juego`, etc.) para asegurar que solo los usuarios autenticados puedan realizar estas acciones.

## 7. Ajustes en Plantillas

- Se crearon las plantillas `login.html` y `register.html` dentro de la carpeta `templates/auth/`.
- Se modificó la plantilla base (`maestra.html` o `base.html`) para:
  - Mostrar mensajes flash (`get_flashed_messages()`).
  - Mostrar condicionalmente los enlaces de "Ingresar"/"Registro" o "Cerrar Sesión" en la barra de navegación, dependiendo del estado de `current_user.is_authenticated`.

## 8. Dependencias

Se instalaron las librerías faltantes como `Flask-Login`, `Werkzeug`, `Flask-WTF`, y `email_validator`.

De acuerdo. Aquí tienes la descripción de las modificaciones que implementamos en tu proyecto, separadas por el componente principal que afectan.

### 1. Migración de Arquitectura: PyMySQL a Flask-Login

El cambio más significativo fue la reversión del ORM (SQLAlchemy) a la arquitectura de PyMySQL funcional para eliminar los persistentes errores de `RuntimeError` y luego la integración de un sistema de autenticación moderno.

- Abandono de SQLAlchemy: Se eliminó `Flask-SQLAlchemy` debido a los errores de Importación Circular (`RuntimeError`) que causaban fallos en la inicialización de la aplicación.
- Retorno a PyMySQL: Se restauró la conexión directa a la base de datos mediante `bd.py` y el controlador `juegos.py`, el cual accede a los datos como tuplas (índices) en lugar de objetos.

### 2. Implementación de Autenticación de Usuarios

Se añadió un sistema de autenticación completo y modular basado en `Flask-Login`.

- Modelo de Usuario Híbrido (`models.py`): Se creó la clase `User` que hereda de `UserMixin`. Esta clase es esencialmente un objeto de Python que simula un

modelo ORM y contiene métodos para el cifrado y verificación de contraseñas (`set_password`, `verify_password`) usando Werkzeug.

- Controlador de Usuarios (`controlador_usuarios.py`): Se crearon las funciones CRUD para la tabla de users (`obtener_usuario_por_username`, `insertar_usuario`, etc.). Estas funciones utilizan PyMySQL para interactuar directamente con la base de datos.
- Sistema de Blueprints: Se creó un Blueprint `auth` (`auth/__init__.py`) para manejar las rutas `/auth/register` y `/auth/login`, aislando la lógica de autenticación del código principal de los juegos.

### 3. Correcciones de Seguridad y Base de Datos

Se resolvieron varios errores críticos de base de datos y cifrado:

- `OperationalError` (Acceso Denegado): Se solucionó al verificar y corregir la URI de conexión en `app.py`, asegurando que la aplicación use la contraseña correcta y el puerto 3307 que fue verificado.
- `DataError` (Columna Demasiado Larga): Se resolvió el error que impedía el registro (Datos demasiado largos para `password_hash`). Se aumentó el tamaño de la columna `password_hash` a `VARCHAR(256)` en la tabla `users` de MySQL para almacenar el *hash* de contraseña completo generado por Werkzeug.
- `ProgrammingError` (Columna Desconocida): Se corrigió en el controlador (`controlador_juegos.py`) al sincronizar los nombres de las columnas SQL (`titulo`, `genero`, `plataforma`) con las variables que utiliza el código Python.

### 4. Estructura y Vistas

- Protección de Rutas: Se implementó el decorador `@login_required` en todas las rutas críticas de gestión de juegos (`/agregar_juego`, `/eliminar_juego`, etc.) en `app.py`.
- Ajuste de Plantillas: Se crearon y ajustaron las plantillas `auth/login.html` y `auth/register.html` para que las llamadas `render_template` funcionen correctamente con la estructura de Blueprint.

De acuerdo. A continuación, se presenta una descripción de las modificaciones implementadas en su proyecto, organizadas por el componente principal afectado. 1.

#### Migración de Arquitectura: PyMySQL a Flask-Login

La modificación más significativa consistió en la reversión del ORM (SQLAlchemy) a una arquitectura basada en PyMySQL funcional para subsanar los errores

persistentes de RuntimeError, seguida de la integración de un sistema de autenticación moderno.

- Abandono de SQLAlchemy: Se prescindió de Flask-SQLAlchemy debido a los errores de Importación Circular (RuntimeError) que provocaban fallos en la inicialización de la aplicación.
- Retorno a PyMySQL: Se restableció la conexión directa a la base de datos mediante bd.py y controlador\_juegos.py, el cual accede a los datos como tuplas (índices) en lugar de objetos.

## 2. Implementación de Autenticación de Usuarios

Se incorporó un sistema de autenticación completo y modular fundamentado en Flask-Login.

- Modelo de Usuario Híbrido (models.py): Se creó la clase User que hereda de UserMixin. Esta clase opera como un objeto Python que simula un modelo ORM y contiene métodos para el cifrado y la verificación de contraseñas (set\_password, verify\_password) utilizando Werkzeug.
- Controlador de Usuarios (controlador\_usuarios.py): Se desarrollaron las funciones CRUD para la tabla de users (obtener\_usuario\_por\_username, insertar\_usuario, etc.). Estas funciones interactúan directamente con la base de datos mediante PyMySQL.
- Sistema de Blueprints: Se estableció un Blueprint auth (auth/init.py) para gestionar las rutas /auth/register y /auth/login, aislando la lógica de autenticación del código principal de los juegos.

## 3. Correcciones de Seguridad y Base de Datos

Se solventaron varios errores críticos relacionados con la base de datos y el cifrado:

- OperationalError (Acceso Denegado): Se resolvió mediante la verificación y corrección de la URI de conexión en app.py, asegurando que la aplicación utilizara la contraseña y el puerto 3307 correctos, los cuales fueron previamente validados.
- DataError (Columna Demasiado Larga): Se subsanó el error que impedía el registro (Datos demasiado largos). A continuación, se presenta una descripción de las modificaciones implementadas en su proyecto, organizadas por el componente principal afectado.

## 1. Migración de Arquitectura: PyMySQL a Flask-Login

La modificación más significativa consistió en la reversión del ORM (SQLAlchemy) a una arquitectura basada en PyMySQL funcional para subsanar los errores persistentes de RuntimeError, seguida de la integración de un sistema de autenticación moderno.

- Abandono de SQLAlchemy: Se prescindió de Flask-SQLAlchemy debido a los errores de Importación Circular (RuntimeError) que provocaban fallos en la inicialización de la aplicación.
- Retorno a PyMySQL: Se restableció la conexión directa a la base de datos mediante `bd.py` y `controlador_juegos.py`, el cual accede a los datos como tuplas (índices) en lugar de objetos.

## 2. Implementación de Autenticación de Usuarios

Se incorporó un sistema de autenticación completo y modular fundamentado en Flask-Login.

- Modelo de Usuario Híbrido (`models.py`): Se creó la clase `User` que hereda de `UserMixin`. Esta clase opera como un objeto Python que simula un modelo ORM y contiene métodos para el cifrado y la verificación de contraseñas (`set_password`, `verify_password`) utilizando Werkzeug.
- Controlador de Usuarios (`controlador_usuarios.py`): Se desarrollaron las funciones CRUD para la tabla de `users` (`obtener_usuario_por_username`, `insertar_usuario`, etc.). Estas funciones interactúan directamente con la base de datos mediante PyMySQL.
- Sistema de Blueprints: Se estableció un Blueprint `auth` (`auth/init.py`) para gestionar las rutas `/auth/register` y `/auth/login`, aislando la lógica de autenticación del código principal de los juegos.

## 3. Correcciones de Seguridad y Base de Datos

Se solventaron varios errores críticos relacionados con la base de datos y el cifrado:

- `OperationalError` (Acceso Denegado): Se resolvió mediante la verificación y corrección de la URI de conexión en `app.py`, asegurando que la aplicación utilizara la contraseña y el puerto 3307 correctos, los cuales fueron previamente validados.
- `DataError` (Columna Demasiado Larga): Se subsanó el error que impedía el registro (Datos demasiado largos para `password_hash`). Se incrementó el

tamaño de la columna `password_hash` a `VARCHAR(256)` en la tabla `users` de MySQL para alojar el *hash* de contraseña completo generado por Werkzeug.

- `ProgrammingError` (Columna Desconocida): Se corrigió en el controlador (`controlador_juegos.py`) al sincronizar los nombres de las columnas SQL (`titulo`, `genero`, `plataforma`) con las variables empleadas por el código Python.

#### 4. Estructura y Vistas

- Protección de Rutas: Se implementó el decorador `@login_required` en todas las rutas críticas de gestión de juegos (`/agregar_juego`, `/eliminar_juego`, etc.) en `app.py`.
- Ajuste de Plantillas: Se crearon y ajustaron las plantillas `auth/login.html` y `auth/register.html` para asegurar la correcta funcionalidad de las llamadas `render_template` con la estructura del Blueprint. para `password_hash`). Se incrementó el tamaño de la columna `password_hash` a `VARCHAR(256)` en la tabla `users` de MySQL para almacenar el *hash* de contraseña completo generado por Werkzeug.
- `ProgrammingError` (Columna Desconocida): Se corrigió en el controlador (`controlador_juegos.py`) al sincronizar los nombres de las columnas SQL (`titulo`, `genero`, `plataforma`) con las variables empleadas por el código Python.

#### 4. Estructura y Vistas

- Protección de Rutas: Se implementó el decorador `@login_required` en todas las rutas críticas de gestión de juegos (`/agregar_juego`, `/eliminar_juego`, etc.) en `app.py`.
- Ajuste de Plantillas: Se crearon y adaptaron las plantillas `auth/login.html` y `auth/register.html` para asegurar el correcto funcionamiento de las llamadas a `render_template` con la estructura de Blueprint.

