Guía ABP Unidad “4”

Programación Orientada a Objeto Seguro

Realizado por: Alejandro Berroteran

Desarrollo de Software para una Agencia de Viajes

En el siguiente informe documentaré el desarrollo completo de un sistema de gestión para una agencia de viajes, elaborado como proyecto individual para la asignatura Programación Orientada a Objeto Seguro (TI3021) del programa Analista Programador en INACAP. A lo largo de este documento, presento la arquitectura, diseño, implementación y pruebas del sistema, aplicando principios de programación orientada a objetos, patrones de diseño reconocidos y prácticas de seguridad informática.

El sistema desarrollado permite gestionar de manera integral las operaciones de una agencia de viajes, incluyendo la administración de destinos turísticos, la creación de paquetes personalizables, y la gestión completa del ciclo de reservas. He implementado un robusto sistema de autenticación que distingue entre clientes y administradores, garantizando que cada usuario acceda únicamente a las funcionalidades correspondientes a su rol.

La solución está construida íntegramente en Python, utilizando MySQL como sistema de gestión de base de datos. He aplicado metodologías ágiles durante todo el ciclo de desarrollo, lo que me permitió iterar sobre los requerimientos y entregar funcionalidades de valor en cada sprint. El código fuente está organizado siguiendo principios de modularidad y separación de responsabilidades, facilitando su mantenimiento y futuras extensiones.

Arquitectura del Software: Diseño en Capas

He diseñado el sistema utilizando una arquitectura en capas que separa claramente las responsabilidades y facilita el mantenimiento del código. Esta decisión arquitectónica responde a la necesidad de crear un sistema escalable, mantenible y preparado para futuras migraciones tecnológicas. La arquitectura implementada consta de cuatro capas fundamentales que interactúan de manera ordenada y predecible.

Capa de Presentación

Actualmente implementada como interfaz de línea de comandos (CLI) en el archivo main.py. Esta capa gestiona toda la interacción con el usuario, mostrando menús, capturando entradas y presentando resultados. Está diseñada para ser reemplazada fácilmente por una interfaz web sin afectar las capas subyacentes.

Capa de Lógica de Negocio

Implementada en el directorio DTO, contiene todas las reglas de negocio del sistema. Aquí residen los algoritmos de cálculo de precios, validación de disponibilidad, y orquestación de operaciones complejas. Esta capa es completamente independiente de la presentación y la persistencia.

Capa de Persistencia

Gestionada por conexion.py, encapsula todas las operaciones de acceso a datos. Utiliza consultas parametrizadas para prevenir inyección SQL y maneja la conexión con MySQL. Esta abstracción permite cambiar el motor de base de datos con modificaciones mínimas.

Capa de Seguridad

Implementada en auth.py, centraliza todos los aspectos relacionados con autenticación y autorización. Gestiona el hashing de contraseñas, validación de credenciales y control de acceso basado en roles, garantizando la protección de información sensible.

Esta arquitectura modular me ha permitido desarrollar y probar cada componente de forma independiente. La preparación para migración a interfaz web es particularmente importante: solo necesitaría reemplazar la capa de presentación CLI por controladores web (por ejemplo, usando Flask o Django), manteniendo intactas las capas de lógica de negocio, persistencia y seguridad. Esta flexibilidad arquitectónica demuestra la aplicación correcta de principios de diseño orientado a objetos y separación de responsabilidades.

Patrones de Diseño Implementados

Durante el desarrollo del sistema, he aplicado dos patrones de diseño fundamentales que mejoran significativamente la estructura, mantenibilidad y eficiencia del código. La selección de estos patrones no fue arbitraria, sino que responde a necesidades específicas identificadas durante el análisis del sistema. A continuación, detallo cada patrón implementado, su justificación técnica y su aplicación concreta en el proyecto.

Patrón Fachada

Implementación: Los distintos DTO actuan como una fachada que simplifica la interacción con múltiples subsistemas complejos del software.

Justificación técnica: El sistema integra varios componentes: autenticación, gestión de base de datos, validación de reglas de negocio y cálculos complejos. Sin una fachada, el código cliente (capa de presentación) necesitaría conocer y coordinar directamente con todos estos subsistemas, lo que generaría alto acoplamiento y código difícil de mantener.

Beneficios obtenidos:

- Simplificación del código cliente: Con los DTO, no necesitamos conocer la complejidad interna de los DAO.
- Punto único de entrada para operaciones complejas como crear reservas (que requieren validar disponibilidad, calcular precios y actualizar base de datos)
- Facilita futuras modificaciones: los cambios internos no afectan la interfaz pública
- Mejora la testabilidad al centralizar la lógica de orquestación

Patrón Singleton

Implementación: La clase Conexion en conexion.py implementa el patrón Singleton para gestionar la conexión a MySQL.

Justificación técnica: Las conexiones a bases de datos son recursos costosos en términos de tiempo de establecimiento y memoria. Crear múltiples instancias de conexión generaría problemas de rendimiento, inconsistencias en transacciones y potencial agotamiento del pool de conexiones del servidor MySQL.

Beneficios obtenidos:

- Garantiza una única instancia de conexión en toda la aplicación
- Optimiza el uso de recursos del sistema
- Facilita el control transaccional centralizado
- Previene problemas de concurrencia en el acceso a datos
- Simplifica la gestión del ciclo de vida de la conexión (apertura y cierre)

La combinación de estos patrones ha resultado en un sistema robusto y bien estructurado. El patrón Fachada proporciona simplicidad en el uso, mientras que el Singleton garantiza eficiencia en el manejo de recursos críticos. Ambos patrones son ampliamente reconocidos en la industria y su aplicación demuestra madurez en el diseño de software.

Planificación del Proyecto: Metodología Ágil

He aplicado principios de metodología ágil durante todo el desarrollo del proyecto, utilizando un enfoque iterativo e incremental que me permitió entregar valor constantemente y adaptarme a los aprendizajes obtenidos en cada iteración. La planificación se estructuró en dos artefactos principales: el Product Backlog (lista priorizada de todos los requerimientos) y el Sprint Backlog (plan detallado de trabajo semanal).

Product Backlog: Requerimientos del Sistema

Identifiqué y prioricé siete requerimientos funcionales fundamentales, ordenados según su valor para el negocio y sus dependencias técnicas:

01

Sistema de Autenticación Segura

Implementar registro e inicio de sesión con hashing de contraseñas y diferenciación de roles (cliente/administrador). **Prioridad: Crítica** - Base para todas las demás funcionalidades.

02

CRUD de Destinos Turísticos

Permitir a administradores crear, consultar, actualizar y eliminar destinos con sus características (país, ciudad, descripción). **Prioridad: Alta** - Componente fundamental del catálogo.

03

Gestión de Paquetes Turísticos

Crear paquetes combinando múltiples destinos, con cálculo automático de precio total. **Prioridad: Alta** - Core del modelo de negocio.

04

Validación de Disponibilidad

Verificar fechas disponibles y cupos antes de confirmar reservas. **Prioridad: Media-Alta** - Previene sobreventa y conflictos.

05

Sistema de Reservas

Permitir a clientes reservar paquetes con fechas específicas y número de personas. **Prioridad: Alta** - Funcionalidad principal para usuarios finales.

06

Conexión a Base de Datos MySQL

Implementar persistencia robusta con transacciones ACID e integridad referencial. **Prioridad: Crítica** - Infraestructura base del sistema.

07

Documentación Técnica UML/BPMN

Generar diagramas de clases, casos de uso y procesos de negocio. **Prioridad: Media** - Documentación para mantenimiento y escalabilidad.

Sprint Backlog: Planificación de 4 Semanas

Semana 1: Fundamentos

- Diseño de arquitectura en capas
- Modelado de base de datos (5 tablas)
- Implementación de sistema de autenticación segura
- Creación de modelos de datos (Usuario, Destino)
- Pruebas unitarias de autenticación

Semana 2: Gestión de Catálogo

- CRUD completo de destinos turísticos
- Implementación de PaqueteTuristico con relación muchos-a-muchos
- Lógica de cálculo automático de precios
- Pruebas de integración con base de datos
- Desarrollo de menú administrativo

Semana 3: Reservas y Lógica de Negocio

- Implementación del modelo Reserva
- Validación de disponibilidad y fechas
- Integración de fachada AgenciaViajes
- Desarrollo de menú de cliente
- Pruebas de flujo completo de reserva

Semana 4: Documentación y Refinamiento

- Generación de diagramas UML (clases, casos de uso)
- Diseño de diagramas BPMN de procesos
- Pruebas de seguridad y manejo de errores
- Optimización de consultas SQL
- Documentación técnica del código

Esta planificación ágil me permitió mantener un ritmo constante de desarrollo, identificar y resolver problemas tempranamente, y ajustar prioridades según los aprendizajes de cada sprint. Cada semana culminó con software funcional y probado, siguiendo el principio ágil de "entrega continua de valor".

Diagrama de Clases: Modelo Orientado a Objetos

El diseño orientado a objetos del sistema se estructura alrededor de cinco clases principales que representan las entidades fundamentales del dominio de negocio. He aplicado principios SOLID, especialmente el de Responsabilidad Única, asegurando que cada clase tenga una función clara y bien definida. A continuación, describo detalladamente cada clase, sus atributos, métodos y las relaciones entre ellas.

1

Clase Usuario

Responsabilidad: Representar a los usuarios del sistema (clientes y administradores).

Atributos:

- id_usuario: int (clave primaria)
- nombre: str
- email: str (único, usado como nombre de usuario)
- password_hash: str (contraseña cifrada)
- salt: str (sal criptográfica única)
- rol: str (valores: 'cliente' o 'administrador')
- fecha_registro: datetime

Métodos principales:

- verificar_password(password: str) → bool: Valida contraseña ingresada
- es_administrador() → bool: Verifica si el usuario tiene privilegios administrativos

2

Clase Destino

Responsabilidad: Representar destinos turísticos individuales.

Atributos:

- id_destino: int (clave primaria)
- nombre: str
- pais: str
- ciudad: str
- descripcion: str
- precio_base: float
- activo: bool

Métodos principales:

- obtener_precio() → float: Retorna el precio base del destino
- actualizar_informacion(datos: dict): Modifica atributos del destino
- desactivar(): Marca el destino como inactivo (soft delete)

3

Clase PaqueteTuristico

Responsabilidad: Representar paquetes compuestos por múltiples destinos.

Atributos:

- id_paquete: int (clave primaria)
- nombre: str
- descripcion: str
- duracion_dias: int
- precio_total: float (calculado automáticamente)
- destinos: List[Destino] (relación muchos-a-muchos)
- disponible: bool

Métodos principales:

- calcular_precio_total() → float: Suma precios de todos los destinos incluidos
- agregar_destino(destino: Destino): Añade un destino al paquete
- obtener_destinos() → List[Destino]: Retorna lista de destinos del paquete
- verificar_disponibilidad(fecha: date) → bool: Valida si hay cupos disponibles

4

Clase Reserva

Responsabilidad: Representar reservas realizadas por clientes.

Atributos:

- id_reserva: int (clave primaria)
- usuario: Usuario (relación muchos-a-uno)
- paquete: PaqueteTuristico (relación muchos-a-uno)
- fecha_reserva: datetime
- fecha_viaje: date
- numero_personas: int
- precio_total: float
- estado: str (valores: 'pendiente', 'confirmada', 'cancelada')

Métodos principales:

- calcular_precio(personas: int) → float: Calcula precio según número de personas
- confirmar_reserva(): Cambia estado a confirmada
- cancelar_reserva(): Cambia estado a cancelada
- obtener_detalle() → dict: Retorna información completa de la reserva

5

Clase AgenciaViajes (Fachada)

Responsabilidad: Orquestar operaciones complejas del sistema.

Atributos:

- db_connection: DatabaseConnection (singleton)
- usuario_actual: Usuario (sesión activa)

Métodos principales:

- registrar_usuario(datos: dict) → bool
- iniciar_sesion(email: str, password: str) → Usuario
- crear_paquete(nombre: str, destinos: List[int]) → PaqueteTuristico
- realizar_reserva(id_paquete: int, fecha: date, personas: int) → Reserva
- obtener_destinos_disponibles() → List[Destino]
- generar_reporte_reservas(usuario: Usuario) → List[Reserva]

Relaciones Entre Clases

El modelo establece las siguientes relaciones que garantizan la integridad referencial:

- **Usuario "1" ←→ "0..*" Reserva:** Un usuario puede tener cero o muchas reservas, cada reserva pertenece a exactamente un usuario.
- **PaqueteTuristico "1" ←→ "0..*" Reserva:** Un paquete puede tener cero o muchas reservas, cada reserva corresponde a exactamente un paquete.
- **PaqueteTuristico "*" ←→ "*" Destino:** Relación muchos-a-muchos implementada mediante tabla intermedia paquetes_destinos. Un paquete incluye múltiples destinos y un destino puede formar parte de múltiples paquetes.
- **AgenciaViajes "1" ←→ "1" DatabaseConnection:** La fachada utiliza el singleton de conexión a base de datos.

Este diseño orientado a objetos permite representar fielmente el dominio de negocio, facilita la extensibilidad del sistema y garantiza la cohesión entre componentes.

Diagramas de Casos de Uso del Sistema

Los casos de uso documentan las interacciones entre los actores del sistema y las funcionalidades disponibles, proporcionando una visión clara de los requerimientos funcionales desde la perspectiva del usuario. He identificado dos actores principales con necesidades y privilegios diferenciados, y once casos de uso fundamentales que cubren todas las operaciones del sistema.

Actores del Sistema

Cliente

Usuario final que utiliza el sistema para consultar información turística y realizar reservas. Tiene acceso restringido a funcionalidades de consulta y gestión de sus propias reservas.

Responsabilidades:

- Consultar catálogo de destinos y paquetes
- Realizar reservas de paquetes turísticos
- Visualizar historial de reservas propias
- Gestionar su información de perfil

Administrador

Usuario con privilegios elevados responsable de la gestión del sistema. Tiene acceso completo a todas las funcionalidades administrativas del catálogo y reportes.

Responsabilidades:

- Gestionar catálogo de destinos (CRUD completo)
- Crear y modificar paquetes turísticos
- Visualizar todas las reservas del sistema
- Generar reportes de gestión

Casos de Uso Principales

CU-01: Registrarse en el Sistema

Actor: Cliente, Administrador

Descripción: Permite a nuevos usuarios crear una cuenta proporcionando nombre, email y contraseña. El sistema valida que el email sea único y almacena la contraseña de forma segura.

Precondiciones: Ninguna

Postcondiciones: Usuario creado y almacenado en base de datos con contraseña hasheada

CU-02: Iniciar Sesión

Actor: Cliente, Administrador

Descripción: Autentica al usuario verificando email y contraseña. El sistema valida las credenciales y establece una sesión activa.

Precondiciones: Usuario registrado previamente

Postcondiciones: Sesión iniciada, acceso a funcionalidades según rol

Relaciones: *«include»* con todos los demás casos de uso (autenticación requerida)

CU-03: Consultar Destinos Disponibles

Actor: Cliente, Administrador

Descripción: Lista todos los destinos turísticos activos con su información detallada (país, ciudad, descripción, precio).

Precondiciones: Sesión iniciada

Postcondiciones: Lista de destinos mostrada al usuario

CU-04: Consultar Paquetes Turísticos

Actor: Cliente, Administrador

Descripción: Muestra todos los paquetes disponibles con destinos incluidos, duración y precio total calculado.

Precondiciones: Sesión iniciada

Postcondiciones: Lista de paquetes con detalles mostrada

CU-05: Realizar Reserva

Actor: Cliente

Descripción: Permite reservar un paquete turístico especificando fecha de viaje y número de personas. El sistema valida disponibilidad antes de confirmar.

Precondiciones: Sesión iniciada como cliente, paquete disponible

Postcondiciones: Reserva creada y almacenada, disponibilidad actualizada

Relaciones: *«include»* CU-06 (Validar Disponibilidad)

CU-06: Validar Disponibilidad

Actor: Sistema

Descripción: Verifica que el paquete tenga cupos disponibles para la fecha solicitada y que la fecha sea válida (futura).

Precondiciones: Paquete y fecha especificados

Postcondiciones: Resultado de validación retornado

CU-07: Visualizar Mis Reservas

Actor: Cliente

Descripción: Muestra el historial completo de reservas del cliente con detalles de cada una (paquete, fecha, personas, estado).

Precondiciones: Sesión iniciada como cliente

Postcondiciones: Lista de reservas del cliente mostrada

CU-08: Gestionar Destinos (CRUD)

Actor: Administrador

Descripción: Permite crear, modificar, eliminar y listar destinos turísticos del catálogo.

Precondiciones: Sesión iniciada como administrador

Postcondiciones: Catálogo de destinos actualizado

CU-09: Crear Paquete Turístico

Actor: Administrador

Descripción: Permite crear un nuevo paquete seleccionando múltiples destinos. El sistema calcula automáticamente el precio total sumando los precios base.

Precondiciones: Sesión iniciada como administrador, destinos existentes

Postcondiciones: Paquete creado con precio calculado

CU-10: Modificar Paquete Turístico

Actor: Administrador

Descripción: Permite actualizar información de un paquete existente (nombre, descripción, destinos incluidos).

Precondiciones: Sesión iniciada como administrador, paquete existente

Postcondiciones: Paquete actualizado, precio recalculado si se modificaron destinos

CU-11: Visualizar Todas las Reservas

Actor: Administrador

Descripción: Muestra un reporte completo de todas las reservas del sistema con información de clientes y paquetes.

Precondiciones: Sesión iniciada como administrador

Postcondiciones: Reporte de reservas generado y mostrado

Esta documentación de casos de uso proporciona una especificación clara y completa de todos los requerimientos funcionales del sistema, sirviendo como base para el desarrollo y las pruebas de aceptación.

Procesos de Negocio: Notación BPMN

He modelado los procesos de negocio críticos del sistema utilizando notación BPMN (Business Process Model and Notation), lo que permite visualizar claramente el flujo de actividades, decisiones y puntos de interacción entre el usuario y el sistema. El proceso más representativo del core del negocio es "Reservar Paquete Turístico", que documento en detalle a continuación.

Proceso: Reservar Paquete Turístico

Este proceso describe el flujo completo desde que un cliente decide realizar una reserva hasta la confirmación final en el sistema. Incluye validaciones de seguridad, reglas de negocio y operaciones de persistencia.

01

Evento de Inicio: Cliente Solicita Reserva

El proceso comienza cuando un cliente autenticado selecciona la opción "Realizar Reserva" en el menú principal. Esta acción dispara el flujo completo del proceso de negocio.

02

Actividad: Validar Sesión Activa

El sistema verifica que existe una sesión de usuario válida y que el rol es "cliente". Si la validación falla, el proceso termina con mensaje de error y redirige al inicio de sesión.

Decisión: ¿Sesión válida? → Sí: continuar | No: terminar proceso

03

Actividad: Mostrar Paquetes Disponibles

El sistema consulta la base de datos y presenta al usuario una lista de todos los paquetes turísticos activos con su información completa (nombre, destinos incluidos, duración, precio total).

04

Actividad: Capturar Datos de Reserva

El sistema solicita al cliente que ingrese: (1) ID del paquete seleccionado, (2) Fecha de viaje deseada, (3) Número de personas. Estos datos se validan para formato correcto.

05

Subproceso: Validar Disponibilidad del Paquete

Este subproceso crítico ejecuta múltiples verificaciones:

- Verificar que el paquete existe y está activo
- Validar que la fecha de viaje es futura (posterior a fecha actual)
- Consultar disponibilidad de cupos para la fecha solicitada
- Verificar que el número de personas no excede capacidad máxima

Decisión: ¿Disponibilidad confirmada? → Sí: continuar | No: mostrar mensaje y volver a captura de datos

06

Actividad: Calcular Precio Total

El sistema calcula el precio total de la reserva aplicando la fórmula: precio_total = precio_paquete × número_personas. Si existen descuentos configurados (ej: grupos grandes), se aplican en este punto.

07

Actividad: Mostrar Resumen de Reserva

Se presenta al cliente un resumen completo con: paquete seleccionado, destinos incluidos, fecha de viaje, número de personas, precio total calculado. Se solicita confirmación explícita.

Decisión: ¿Cliente confirma? → Sí: continuar | No: cancelar proceso

08

Actividad: Registrar Reserva en Base de Datos

Si el cliente confirma, el sistema ejecuta una transacción que incluye:

- INSERT en tabla reservas con todos los datos
- UPDATE de disponibilidad del paquete (restar cupos)
- Asignación de estado inicial "confirmada"
- Registro de fecha/hora de la transacción

Esta operación se ejecuta dentro de una transacción ACID para garantizar consistencia.

09

Actividad: Generar Confirmación

El sistema genera un número de confirmación único y muestra al cliente los detalles completos de su reserva, incluyendo el ID de reserva para futuras referencias.

10

Evento de Fin: Reserva Completada

El proceso concluye exitosamente con la reserva registrada en el sistema. El cliente puede ver su reserva en el historial y el administrador puede consultarla en reportes.

Elementos BPMN Utilizados

Eventos

- **Inicio:** Cliente solicita reserva
- **Fin:** Reserva completada/cancelada
- **Error:** Fallo en validación o BD

Actividades

- Tareas de usuario (captura datos)
- Tareas de sistema (validaciones)
- Subprocesos (validar disponibilidad)

Compuertas

- Decisiones exclusivas (XOR)
- Validaciones de negocio
- Manejo de excepciones

Este modelado BPMN proporciona una documentación visual clara del proceso de negocio más importante del sistema, facilitando la comprensión por parte de stakeholders no técnicos y sirviendo como especificación precisa para el desarrollo y pruebas.

Implementación del Código en Python

La implementación del sistema se realizó íntegramente en Python 3.x, aprovechando su sintaxis clara, tipado dinámico y amplio ecosistema de bibliotecas. He organizado el código en una estructura modular que facilita el mantenimiento, testing y escalabilidad. A continuación, detallo la estructura de archivos, las bibliotecas utilizadas y las decisiones técnicas de implementación.

Estructura de Archivos del Proyecto

DTO

Responsabilidad: Define las clases de dominio (Usuario, Destino, PaqueteTuristico, Reserva).

Contenido: Clases con atributos tipados, constructores, métodos de negocio y representaciones string para debugging.

auth.py

Responsabilidad: Gestiona autenticación y autorización.

Contenido: Funciones de hashing (hash_password, verify_password), generación de sal criptográfica, validación de roles.

conexion.py

Responsabilidad: Maneja conexión y operaciones con MySQL.

Contenido: Clase DatabaseConnection (Singleton), métodos para ejecutar consultas parametrizadas, manejo de transacciones.

AgenciaDAO.py

Responsabilidad: Implementa la fachada AgenciaViajes.

Contenido: Métodos de alto nivel que orquestan operaciones complejas combinando auth, models y database.

main.py

Responsabilidad: Punto de entrada y capa de presentación CLI.

Contenido: Menús interactivos, captura de entrada de usuario, formateo de salida, bucle principal de la aplicación.

Bibliotecas y Dependencias Utilizadas

Bibliotecas Estándar de Python

- **hashlib:** Para hashing criptográfico de contraseñas usando PBKDF2
- **secrets:** Generación de sal criptográfica segura
- **datetime:** Manejo de fechas y timestamps
- **typing:** Type hints para mejor documentación y validación
- **json:** Serialización de datos para logging y configuración

Bibliotecas Externas

- **pymysql:** Conector oficial de MySQL para Python, manejo de conexiones y consultas
- **python-dotenv (opcional):** Carga de variables de entorno desde archivo .env para credenciales

Decisiones Técnicas Clave

Uso de Hashing Seguro con PBKDF2

Implementé hashing de contraseñas usando hashlib.pbkdf2_hmac con SHA-256, 100,000 iteraciones y sal única por usuario. Esta configuración cumple con estándares OWASP y hace inviables los ataques de fuerza bruta. Ejemplo de código:

```
salt = secrets.token_bytes(32)
password_hash = hashlib.pbkdf2_hmac(
    'sha256',
    password.encode('utf-8'),
    salt,
    100000
)
```

Consultas Parametrizadas para Prevenir SQL Injection

Todas las consultas SQL utilizan parámetros seguros mediante placeholders %s, nunca concatenación de strings. El conector MySQL escapa automáticamente los valores, previniendo inyecciones. Ejemplo:

```
cursor.execute(
    "SELECT * FROM usuarios WHERE email =
%s",
    (email,)
)
```

Manejo Robusto de Errores

Implementé bloques try-except específicos para cada tipo de error (MySQLError, ValueError, etc.), con logging detallado y mensajes amigables al usuario. Las transacciones incluyen rollback automático en caso de fallo.

Validación de Datos en Múltiples Capas

Validaciones en tres niveles: (1) Entrada de usuario en main.py, (2) Lógica de negocio en agencia.py, (3) Restricciones de base de datos (UNIQUE, NOT NULL, FOREIGN KEY). Esta redundancia garantiza integridad de datos.

El código resultante es mantenible, seguro y está preparado para extensiones futuras como migración a API REST, implementación de tests automatizados, o integración con sistemas de pago.

Sistema de Autenticación Segura

La seguridad de las credenciales de usuario es fundamental en cualquier sistema moderno. He implementado un sistema de autenticación que sigue las mejores prácticas de la industria, garantizando que las contraseñas nunca se almacenen en texto plano y que sean resistentes a ataques comunes como fuerza bruta, rainbow tables y phishing de base de datos. A continuación, explico en detalle la implementación técnica del sistema de autenticación.

Arquitectura de Seguridad Implementada

1. Registro de Usuario

El usuario proporciona email y contraseña en texto plano (solo en memoria, nunca se registra en logs). El sistema genera automáticamente una sal criptográfica única de 32 bytes usando `secrets.token_bytes(32)`.

2. Generación de Sal Criptográfica

La sal es un valor aleatorio único por usuario que previene ataques de rainbow table. Uso el módulo `secrets` (criptográficamente seguro) en lugar de `random` (predecible). La sal se almacena junto al hash en la base de datos.

3. Hashing con PBKDF2-HMAC-SHA256

Aplico la función `hashlib.pbkdf2_hmac('sha256', password, salt, 100000)` que deriva una clave de 256 bits tras 100,000 iteraciones. Este proceso computacionalmente costoso ralentiza ataques de fuerza bruta haciéndolos inviables.

4. Almacenamiento Seguro

Solo el hash y la sal se almacenan en la base de datos. La contraseña original se descarta inmediatamente. Los campos son: `password_hash` (BLOB) y `salt` (BLOB).

5. Verificación en Inicio de Sesión

Cuando el usuario inicia sesión, recupero su sal única, aplico el mismo proceso de hashing a la contraseña ingresada y comparo ambos hashes usando `hmac.compare_digest()` (resistente a timing attacks).

Código de Implementación

A continuación, presento fragmentos clave del módulo auth.py que implementan estas funcionalidades:

Función hash_password()

```
def hash_password(password: str) -> tuple:
    """
    Genera hash seguro con sal única.
    Returns: (password_hash, salt)
    """
    # Generar sal criptográfica
    salt = secrets.token_bytes(32)

    # Aplicar PBKDF2 con 100k iteraciones
    pwd_hash = hashlib.pbkdf2_hmac(
        'sha256',
        password.encode('utf-8'),
        salt,
        100000
    )

    return pwd_hash, salt
```

Función verify_password()

```
def verify_password(
    password: str,
    stored_hash: bytes,
    stored_salt: bytes
) -> bool:
    """
    Verifica contraseña contra hash.
    """
    # Aplicar mismo proceso de hashing
    pwd_hash = hashlib.pbkdf2_hmac(
        'sha256',
        password.encode('utf-8'),
        stored_salt,
        100000
    )

    # Comparación segura
    return hmac.compare_digest(
        pwd_hash,
        stored_hash
    )
```

Propiedades de Seguridad Garantizadas

Imposibilidad de Reversión

Las funciones hash son unidireccionales. Incluso con acceso a la base de datos, es matemáticamente imposible obtener la contraseña original del hash.

Resistencia a Rainbow Tables

Cada usuario tiene una sal única, por lo que un atacante necesitaría generar tablas rainbow específicas para cada usuario, lo cual es inviable.

Protección contra Timing Attacks

El uso de `hmac.compare_digest()` garantiza que la comparación tome tiempo constante, previniendo que un atacante infiera información del tiempo de respuesta.

Resistencia a Fuerza Bruta

Con 100,000 iteraciones, probar un millón de contraseñas tomaría años en hardware estándar. Esta configuración es recomendada por OWASP.

Validaciones Adicionales

- **Complejidad de contraseña:** Mínimo 8 caracteres, requiere mayúsculas, minúsculas, números y símbolos
- **Protección de sesión:** Tokens de sesión con expiración automática después de 30 minutos de inactividad
- **Límite de intentos:** Bloqueo temporal después de 5 intentos fallidos consecutivos
- **Logging de eventos:** Registro de todos los intentos de autenticación (exitosos y fallidos) para auditoría

Este sistema de autenticación cumple con estándares internacionales (OWASP, NIST) y proporciona una base sólida de seguridad para el sistema de gestión de la agencia de viajes.

Persistencia y Presentación de la Solución

En esta sección final, documento la implementación de la capa de persistencia utilizando MySQL y presento las funcionalidades demostrables del sistema completamente implementado.

Sistema de Persistencia con MySQL

He utilizado MySQL 8.0 ejecutándose en XAMPP como sistema de gestión de base de datos relacional. La base de datos está diseñada con motor de almacenamiento InnoDB, que garantiza transacciones ACID y soporta integridad referencial mediante claves foráneas.

Estructura de la Base de Datos: 5 Tablas Principales

Tabla	Campos Principales	Claves Foráneas	Motor
usuarios	id_usuario (PK), nombre, email (UNIQUE), password, rol, fecha_registro	Ninguna	InnoDB
destinos	id_destino (PK), nombre, pais, ciudad, descripcion, precio_base, activo	Ninguna	InnoDB
paquetes	id_paquete (PK), nombre, descripcion, duracion_dias, precio_total, disponible	Ninguna	InnoDB
paquetes_destinos	id_paquete (FK), id_destino (FK), orden	→ paquetes, → destinos (ON DELETE CASCADE)	InnoDB
reservas	id_reserva (PK), id_usuario (FK), id_paquete (FK), fecha_reserva, fecha_viaje, numero_personas, precio_total, estado	→ usuarios, → paquetes (ON DELETE CASCADE)	InnoDB

Características técnicas implementadas:

- **Integridad referencial:** Todas las relaciones están reforzadas con FOREIGN KEY CONSTRAINT
- **Eliminación en cascada:** ON DELETE CASCADE garantiza consistencia cuando se eliminan registros padre
- **Transacciones ACID:** Todas las operaciones críticas (reservas, modificación de paquetes) se ejecutan dentro de transacciones con commit/rollback

Funcionalidades Demostrables del Sistema

Registro de Usuarios

Formulario de registro que captura nombre, email y contraseña. Validación de unicidad de email, complejidad de contraseña y almacenamiento seguro con hashing PBKDF2.

Inicio de Sesión Seguro

Autenticación mediante email y contraseña con verificación de hash. Diferenciación automática de roles (cliente/administrador) y redirección a menú correspondiente.

Gestión de Destinos (Admin)

CRUD completo de destinos turísticos: crear nuevos destinos con país, ciudad, descripción y precio; modificar información existente; eliminar destinos (soft delete); listar catálogo completo.

Creación de Paquetes (Admin)

Selección múltiple de destinos desde el catálogo, especificación de duración en días, cálculo automático de precio total sumando precios base, validación de coherencia de datos.

Sistema de Reservas (Cliente)

Consulta de paquetes disponibles, selección de paquete deseado, ingreso de fecha de viaje y número de personas, validación de disponibilidad en tiempo real, confirmación y generación de número de reserva.

Historial y Reportes

Clientes visualizan sus propias reservas con todos los detalles. Administradores acceden a reporte completo del sistema con todas las reservas, filtros por fecha, estado y usuario.

Pruebas Realizadas

Pruebas de Seguridad

- Intentos de inyección SQL (todas las consultas resistieron)
- Validación de hashing (contraseñas nunca recuperables)
- Pruebas de sesión y autorización (control de acceso correcto)

Pruebas Funcionales

- Flujo completo de registro y login
- Creación y modificación de todos los modelos
- Validación de reglas de negocio (disponibilidad, precios)

Pruebas de Integración

- Transacciones complejas (crear paquete con múltiples destinos)
- Integridad referencial (cascadas funcionando)
- Manejo de errores de base de datos

Pruebas de Carga

- Reservas generadas (sin pérdida de datos)
- Consultas complejas optimizadas con índices

Conclusiones del Proyecto

He desarrollado exitosamente un sistema completo de gestión para agencia de viajes que cumple con todos los requerimientos especificados en la Guía ABP Unidad 4. El sistema demuestra aplicación práctica de principios de programación orientada a objetos, patrones de diseño reconocidos (Fachada y Singleton), arquitectura en capas modular, y prácticas de seguridad informática de nivel profesional.

Las tecnologías utilizadas (Python 3.x con MySQL) resultaron adecuadas para los objetivos del proyecto, facilitando un desarrollo ágil y mantenible. La arquitectura implementada está preparada para futuras extensiones como migración a interfaz web, integración con APIs de pago, o implementación de módulos adicionales como gestión de hoteles o vuelos.

Este proyecto demuestra competencias técnicas en: análisis y diseño de sistemas, modelado de datos, programación orientada a objetos, seguridad informática, persistencia de datos, metodologías ágiles, y documentación técnica profesional. El código fuente está completamente comentado, organizado en módulos cohesivos y disponible para revisión en el repositorio del proyecto.

> "El desarrollo de este sistema me ha permitido aplicar de manera integral los conocimientos adquiridos en la asignatura Programación Orientada a Objeto Seguro, demostrando capacidad para diseñar, implementar y documentar soluciones de software profesionales que cumplen con estándares de la industria."

Enlace al repositorio del proyecto: https://github.com/aleberrot/AgenciaAventura