

Proyecto final

Integrantes: Alien Herrera, Agustin Samperi, Mateo Peralta, Juan Cruz Ambrosini.



EL TEMPLO

¿En qué consiste nuestro proyecto?

El proyecto consiste en el desarrollo de una aplicación de reserva de hoteles, la cual permite gestionar y visualizar las reservaciones tanto desde la perspectiva de un administrador como de un cliente. Esta solución tiene como objetivo facilitar el proceso de reservas de habitaciones en diferentes hoteles, brindando una interfaz amigable y funcional para ambos tipos de usuarios.

Los clientes pueden crear una cuenta o iniciar sesión si ya tienen una, lo que les permite acceder al sistema para elegir un hotel, seleccionar una habitación y establecer las fechas de inicio y fin de su estancia.

Por otro lado, los administradores tienen acceso a una vista más completa del sistema, pudiendo gestionar tanto a los clientes como los hoteles.

A través de esta estructura, la aplicación de reservas de hoteles busca ofrecer una plataforma eficiente para usuarios de diferentes roles, optimizando tanto el proceso de reserva para los clientes como la gestión para los administradores.

¿Qué herramientas utilizamos?

Java: Lenguaje de programación principal utilizado para construir la lógica y el funcionamiento central del proyecto.

Java Swing: Biblioteca gráfica de Java utilizada para crear la interfaz de usuario del proyecto.

JDBC (Java Database Connectivity): API de Java utilizada para conectar y ejecutar operaciones en la base de datos SQLite. JDBC permite la comunicación entre la aplicación Java y SQLite, facilitando la realización de consultas SQL para almacenar, recuperar y gestionar los datos.

SQLite: Base de datos ligera y basada en archivos que almacena la información del proyecto.

¿Cómo es la estructura del proyecto?


Estructura de paquetes y clases:

AppLogic: Es el paquete donde reside la lógica principal del sistema.

DAO (Data Access Object): Este paquete contiene las clases responsables de la interacción con la base de datos. Cada clase DAO maneja una entidad específica. Gestionan las operaciones CRUD (Create, Read, Update, Delete) para la base de datos.

- **HabitacionesDAO:** Clase que gestiona las operaciones de las habitaciones en la base de datos.
- **HotelDAO:** Clase que gestiona las operaciones de los hotel en la base de datos.
- **PersonaDAO:** Clase que gestiona las operaciones de las personas en la base de datos.
- **ReservaDAO:** Gestiona las reservas en la base de datos.

Relaciones entre clases: Aquí se implementa el patrón **DAO (Data Access Object)**, que permite desacoplar la lógica de acceso a datos de la lógica de negocio. El paquete **DAO** contiene las clases concretas que interactúan directamente con la base de datos.



DAO.Interface: Este subpaquete define las interfaces para los objetos DAO, lo que permite una abstracción clara entre la lógica de negocio y el acceso a datos.

Modo de empleo: Las interfaces permiten establecer un contrato que las clases DAO deben seguir, lo que asegura que todas las implementaciones cumplirán con un conjunto específico de operaciones.

DTO (Data Transfer Object): Este paquete contiene las clases que actúan como contenedores de datos. Estas clases facilitan la transferencia de datos entre capas del sistema.

- **ClienteDTO:** Clase que representa los datos de un cliente.
- **HabitacionDTO, HotelDTO, ReservaDTO:** Clases análogas para representar habitaciones, hoteles y reservas, respectivamente.

Relaciones entre clases: Las clases DTO son utilizadas para encapsular y transportar la información entre las capas del sistema, lo que permite una comunicación eficiente y clara. Las capas de presentación (GUI) y los servicios trabajan con estos DTO para intercambiar información sin necesidad de manipular directamente las entidades del modelo.

Exceptions: Aquí se gestionan las excepciones personalizadas que pueden ocurrir durante la ejecución del programa.

- **ServiceExceptions:** Excepciones relacionadas con los servicios del sistema.
- **DAOExceptions:** Excepciones específicas para operaciones con la base de datos.

Relaciones entre clases: Las excepciones como **DAOExceptions** o **ServiceExceptions** se utilizan para manejar errores específicos en cada capa. Las clases en los paquetes DAO, Service y GUI pueden capturar y gestionar estas excepciones para asegurar un comportamiento estable del sistema.

GUI: Este paquete contiene las clases relacionadas con la interfaz gráfica de usuario (GUI).

- **GUIMain:** Clase principal que maneja la interacción con el usuario.

Relaciones entre clases: La GUI utiliza los servicios para realizar las operaciones necesarias. Por ejemplo, cuando un cliente selecciona un hotel y realiza una reserva, la clase GUI delega la lógica al **ServiceReserva**, que luego interactúa con los DAO para almacenar la información en la base de datos.

Model: En este paquete se encuentran las clases que representan las entidades de negocio del sistema.

- **Persona** : Esta clase es abstracta, que implementa los atributos que necesitan Admin y Cliente
- **Admin**: Clase que hereda de Persona, define los atributos que posee un admin.
- **Cliente**: Clase que hereda de Persona. Define los atributos y comportamientos de un clientes.
- **Habitación**: Modelo para la habitación que posee cada Hotel.
- **Hotel**: Representa un hotel en el sistema con sus diferentes atributos.
- **Reserva**: Reserva contiene los diferentes atributos que va a tener una reserva, estos atributos van a estar asociados al IdCliente, idHotel, idHabitacion, etc.

Relaciones entre clases : Estas clases son utilizadas por la capa de servicios para implementar la lógica de negocio. Las instancias de estas clases representan los datos que son almacenados en la base de datos y gestionados por los DAO.

Service: Este paquete contiene las clases que implementan la lógica de negocio.

- **ServiceCliente**: Implementa la lógica de negocio relacionada con los clientes.
- **ServiceHabitacion**: Gestiona la lógica de las habitaciones.
- **ServiceHotel**: Controla la lógica de los hoteles.
- **ServiceReserva**: Implementa la lógica de reservas.

Relaciones entre clases: Los servicios actúan como un intermediario entre los DAO y las clases de la GUI (Interfaz de Usuario). Cuando el usuario realiza una operación (como una reserva), las

clases de servicio se encargan de coordinar las acciones necesarias, como validar los datos, interactuar con la base de datos a través de los DAO, y devolver los resultados a la GUI.

Util: Paquete de utilidades donde se encuentran clases auxiliares o comunes.

- **ConnectionBD:** Clase que maneja la conexión a la base de datos.
- **Rol:** Clase que gestiona los roles de usuarios en el sistema.
- **ValidarDatos:** Clase que valida los datos de entrada en la aplicación.

Relaciones entre clases: Estas clases utilitarias son utilizadas por varias capas del sistema para tareas comunes. Por ejemplo, las clases DAO pueden utilizar **ConnectionBD.java** para conectarse a la base de datos, mientras que **ValidarDatos.java** puede ser utilizada por las clases de servicio antes de procesar las solicitudes del usuario y del administrador.

Arquitectura del proyecto:

El proyecto sigue una **arquitectura en capas**, donde cada capa tiene una responsabilidad específica, y las capas superiores interactúan con las inferiores:

1. **Capa de presentación (GUI):** Es la capa más externa y se comunica directamente con los usuarios. Esta capa envía las solicitudes de los usuarios a los servicios.
2. **Capa de lógica de negocio (Service):** Aquí se implementan las reglas de negocio. Los servicios gestionan las solicitudes de la GUI y se encargan de coordinar las interacciones con la capa de acceso a datos.
3. **Capa de acceso a datos (DAO):** Las clases DAO manejan la comunicación con la base de datos. Las clases de servicio interactúan con los DAO para realizar las operaciones de lectura, escritura, actualización y eliminación de datos.
4. **Capa de utilidades (Util):** Proporciona soporte técnico, como la conexión a la base de datos o la validación de datos.

Esta organización garantiza una separación clara de responsabilidades, lo que facilita la mantenibilidad y escalabilidad del sistema. Además, los **DTO** facilitan la transferencia de datos entre las capas sin acoplar las entidades de dominio directamente a la GUI.

Nuestro UML:

Link [UML](#)

Aclaración: Para poder ver en totalidad el UML debe abrir el archivo con draw.io.