DESAFÍO 2.

INTEGRANTES**:**

FABIANA DIAZ ANAYA

DANIELA HAYBIB CARVAJALINO MUÑOZ

DOCENTES:

DOCTOR ANÍBAL GUERRA

DOCTOR AUGUSTO SALAZAR.

INFORME SOBRE DEL PROYECTO UDEASTAY - DESAFIO 2.

UNIVERSIDAD DE ANTIOQUÍA.

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

17 DE MAYO DE 2025

MEDELLÍN

# INTRODUCCIÓN.

Este informe describe el análisis, diseño y solución del Desafío N°2 de Informática II. El objetivo es crear un sistema de reservas para la plataforma UdeAStay, que permite gestionar estadías hogareñas en Antioquia, diferenciando las operaciones según el tipo de usuario: huésped o anfitrión.  
  
Debido a las restricciones del desafío, no se permite el uso de herencia. Por tanto, las clases Huesped y Anfitrion se implementan de forma completamente independiente, aunque comparten ciertos atributos como nombre y correo. La lógica común se gestiona desde clases auxiliares o desde la clase principal del sistema.  
  
Además, no se permite el uso de estructuras STL, por lo cual se usan arreglos dinámicos implementados manualmente, y se tiene un control riguroso de memoria con new y delete.

# OBJETIVOS.

* Implementar las clases: Huesped, Anfitrion, Alojamiento, Reservacion y Sistema.
* Crear menús diferenciados para huéspedes y anfitriones.
* Desarrollar funciones como búsquedas, reservas, cancelaciones, y registro de alojamientos.
* Manejar datos mediante archivos externos (entrada y salida).
* Crear estructuras dinámicas propias para manipulación de datos.
* Controlar el uso de memoria, evitar fugas y evaluar la eficiencia de búsquedas y filtros.
* Implementar el seguimiento de reservaciones activas e históricas sin STL ni herencia.

# CLASES IMPLEMENTADAS.

1. Huésped: nombre, correo, contraseña, arreglo de Reservación\*
2. Anfitrión: nombre, correo, contraseña, arreglo de Alojamiento\*
3. Alojamiento: código, dirección, precioNoche, puntuación, capacidad, fechas, estado
4. Reservación: código, fechas, activo, punteros a Huesped y Alojamiento
5. Sistema: menú principal, validaciones, carga y escritura de archivos, filtrado
6. Validador: validaciones de entrada
7. Fecha (opcional): para comparar fechas de reservas

# RELACIONES ENTRE CLASES.

* Huesped contiene una lista de Reservacion\*
* Anfitrion contiene una lista de Alojamiento\*
* Cada Reservacion conecta a un Huesped y un Alojamiento mediante punteros
* El Sistema accede a todas las entidades y gestiona flujos

# ESTRUCTURA DE ARCHIVOS.

1. main.cpp – menú principal y control general
2. clases.hpp – definición de clases
3. clases.cpp – implementación de métodos
4. sistema.cpp – operaciones generales y flujo de control
5. utilidades.cpp – funciones auxiliares de validación

**Archivos de texto:**

* usuarios.txt
* alojamientos.txt
* reservas.txt
* historial\_reservas.txt

# ENFOQUE Y METODOLOGÍA.

Dado que no se puede usar ni herencia ni STL, se diseñaron clases independientes, evitando duplicación mediante el uso de funciones auxiliares compartidas. Se implementaron estructuras de datos personalizadas (arreglos dinámicos) y mecanismos de conteo de memoria para controlar eficiencia y evitar errores. El sistema se diseñó pensando en modularidad, validación estricta, y representación fiel de las entidades del dominio.

# CONCLUSIONES.

La eliminación de herencia implicó duplicar ciertos atributos en clases como Huesped y Anfitrion, pero también permitió observar la importancia de una estructura clara y modular.  
  
El sistema cumple con los requisitos del desafío, implementando manejo de usuarios, alojamientos y reservas sin STL ni herencia, y aplicando programación orientada a objetos con control explícito de memoria. Se deja abierta la posibilidad de mejorar el diseño mediante plantillas, polimorfismo por funciones libres o funciones miembro en común si en el futuro se relajan las restricciones.

# DIAGRAMA DE CLASES

