**ANÁLISIS DESAFIO 2**

### **a. Análisis del problema y consideraciones para la solución propuesta**

**El proyecto plantea el desarrollo de una aplicación orientada a objetos para la gestión de un sistema de estadías hogareñas denominado UdeAStay. El sistema debe permitir la administración de alojamientos, anfitriones, huéspedes y reservaciones, diferenciando claramente entre dos tipos de usuarios: administradores (anfitriones) y huéspedes.**

**Las funcionalidades clave incluyen autenticación, consulta y manipulación de datos de alojamientos, y operaciones de reservación. Se exige el uso de C++ sin el uso de bibliotecas STL, salvo el tipo string, y el manejo de archivos debe hacerse mediante lectura línea por línea, promoviendo un enfoque formativo.**

**La solución debe modelar adecuadamente relaciones entre las entidades utilizando principios fundamentales de POO, como encapsulamiento, modularidad y reutilización.**

### **b. Esquema de tareas y fases**

1. **Análisis del enunciado**
   * **Identificación de actores y entidades clave.**
   * **Determinación de funcionalidades requeridas.**
2. **Diseño del modelo de clases**
   * **Clases Administrador, Huésped, Lugar, Reservacion.**
   * **Relaciones jerárquicas y de composición.**
3. **Diseño del flujo de programa**
   * **Pantalla de inicio.**
   * **Autenticación (login).**
   * **Menús específicos por tipo de usuario.**
4. **Implementación inicial de login**
   * **Verificación de credenciales en archivos .txt.**
   * **Creación dinámica de objetos según el tipo de usuario.**
5. **Implementación progresiva de funcionalidades**
   * **Funcionalidades administrativas.**
   * **Funcionalidades para el huésped.**
6. **Validaciones y pruebas**
   * **Control de errores de entrada.**
   * **Validación de formatos y restricciones.**

### **c. Algoritmos implementados**

1. **Autenticación**
   * **Comparación de documento y contraseña contra registros en archivo admin.txt y usuarios.txt.**

1. **Creación de objetos**
   * **Se construyen objetos Administrador y Usuario al encontrar coincidencia en login.**
2. **Carga de datos**
   * **Lectura línea a línea desde archivos .txt, separando campos por delimitador ;.**
3. **Menús dinámicos**
   * **Se presentan distintas opciones según el rol autenticado.**

### **d. Problemas de desarrollo afrontados**

* **Restricción del uso de STL: se optó por punteros dinámicos para manejar listas de objetos, lo cual incrementa la complejidad del manejo de memoria.**
* **Diseño modular sin clases avanzadas: se tuvo que trabajar de forma manual con múltiples archivos .cpp y .h para mantener separación lógica.**
* **Validación de datos sin estructuras dinámicas: se resolvió con validaciones manuales al leer los archivos.**
* **Sincronización en trabajo colaborativo: se creó un flujo de trabajo en Git con ramas separadas para cada perfil (administrador/usuario).**

### **e. Evolución de la solución y consideraciones para la implementación**

**Inicialmente, se diseñó una estructura básica con autenticación y menú según rol. Luego, se fue incrementando el detalle, adicionando clases con atributos relevantes y planificando su asociación mediante archivos auxiliares (como lugares.txt y reservas.txt).**

**Consideraciones claves:**

* **A futuro puede ser útil incorporar punteros inteligentes para evitar fugas de memoria.**
* **El uso de un delimitador (;) en los archivos de texto mejora la separación de campos complejos como nombres compuestos.**
* **Se planea una sola función de login que discrimine internamente el tipo de usuario.**