Desafio #2: UdeaStay

Nikolas Geovanny Ortega Suarez   
  
Keiner torres

Informatica 2

Augusto Enrique Salazar

ingeniería en telecomunicaciones

Universidad de Antioquia

17/07/2025

**Análisis y desarrollo de solución(13 de mayo/ 17 de mayo):**En el desafío 2 un cliente nos propone un proyecto de nombre UdeaStay la cual es una pagina de inmobiliaria y alquiler de alojamientos. Esta debe contar con diferentes características especificas basada en el correcto funcionamiento y uso por parte de un cliente y de un anfitrión (en este caso el anfitrión tendría las cualidades de un admin en la página), cada uno disfrutara de distintas características aunque parecidas como credenciales que los identificaran, vale la aclaración de que todo dato sobre alojamientos huéspedes y anfitriones será dado por el clientes( los profesores)   
  
  
**características importantes:**

Del proyecto se esperan funcionalidades tales que:

* Acceso mediante credenciales tanto de huéspedes como de anfitriones
* Gestión eficiente de los datos
* Búsqueda y creación de reservas (para los huéspedes)
* Cancelación de reservas
* Mantenimiento de un archivo histórico de reservaciones
* Medición del rendimiento del código(para métodos puramente académicos)

**Limitantes o retos:**

El equipo toma como retos o limitaciones situaciones como el no saber el formato en el que vendrán los archivos del dataset proporcionado por los profesores, pensamos que es uno de los retos  
  
otro reto en cuanto al desarrollo es el limitante de uso de la STL para estructuras de datos, es decir estamos supeditados al uso de arreglos dinámicos para el manejo de los diferentes datos del programa haciendo uso de la memoria dinámica siempre viéndolo desde un punto de vista de eficiencia y no redundancia  
  
  
**consideraciones personales:**

Como decisión de grupo y decisión en vista de los requerimientos, procuraremos evitar el uso de temascomo herencia que a pesar de ayudar a la eficiencia delprograma para evitar la redundancia, no es un tema que manejemos en u totalidad además no lo hemos visto en clase, del mismo modo descartamos cualquier uso de métodos virtuales en vita de que no son necesarios y al igual que la herencia son temas que no se han manejado, procuramos mantener básica la arquitectura del programa obviamente cumpliendo los estamentos propuestos por los profesores haciendo solo uso de string como una útil y necesaria herramienta   
  
  
**aclaraciones:**

El diagrama de clases propuesto y su orden obedecen a una visión inicial del problema, esta sujeto a posibles cambios ya que ver el programa y su complejidad completa si una mínima etapa de desarrollo puede ser contraproducente o así lo asumimos, esperamos señirnos a lo propuesto y que lo cambios sean de bajo impacto por ejemplo no implementar mas clases sino usar las ya propuestas que naces de nuestro análisis inicial

**Diagrama de clases (INICIAL):**  
**Diagrama

El contenido generado por IA puede ser incorrecto.**

**Actualización (23/05/2025):**

El desarrollo del desafio se ha dado utilizando QT y el repositorio de git hub, como decisión del arquitecto del diagrama de clases se separo el desarrollo en dos Branch, una enfocada en las clases: anfitrión, huésped y reservaciones.  
la otra enfocada en las clases: fecha, alojamientos y SistemaUdeaStay (que sería el cerebro de todo), esto con el fin de tener un trabajo conjunto pero con reglas establecidas, y mantener la equidad de trabajo.  
  
en el diagrama de clases hasta el momento se hizo un cambio en la clase fecha, que ahora incluye un método “CalcularNombreDiaSemana” a fin de cumplir con el formato de "nombre Día, día 'de' nombreMes 'del' año" )

Se ha definido el formato de los archivos de datos basados en el modelo CSV basado en la separación lógica por medio de comas, considerando 5 archivos:

* Alojamientos.csv
* Anfitriones.csv
* Huéspedes.csv
* Reservaciones.csv
* Histórico.csv

**Respecto al no uso de la STL:**

Para manejar los datos se prohíbe la STL por ende optamos por el uso de arreglos dinámicos, además de el uso de componentes de la biblioteca estándar que no son para manejo de datos por si mismas como **string** para el manejo de cadenas, **iomanip** para formateo y  **sstream** para la construcción de los strings.  
  
  
**Actualización (05/06/2025)**

A continuación, se detallan las actualizaciones y la evolución del proyecto desde el análisis inicial, reflejando las decisiones de diseño concretas, la implementación de la arquitectura base y la resolución de los desafíos técnicos encontrados durante el desarrollo.

**Evolución del Diseño y Decisiones de Implementación**

Basado en la visión inicial del proyecto, el diseño se ha refinado y concretado en varios aspectos clave para asegurar la robustez y el cumplimiento de los requisitos.

* **Definición del Formato de Persistencia de Datos** En respuesta al reto inicial sobre la incertidumbre del formato de los datos, se ha tomado la decisión de diseño de implementar un formato **CSV (Valores Separados por Comas)** para todos los archivos de persistencia: Alojamientos.csv, Anfitriones.csv, Huespedes.csv, Reservaciones.csv y Historico.csv. Cada archivo cuenta con una línea de cabecera que define las columnas, facilitando su comprensión y mantenimiento. Para manejar datos que puedan contener comas, como nombres o direcciones, se ha adoptado el estándar de encapsular dichos campos entre comillas dobles.
* **Arquitectura del Sistema y Refinamiento de Clases** Para darle una identidad propia al proyecto y evitar nombres genéricos, la clase central SistemaUdeAStay fue renombrada a GestorUdeaStay. Se mantuvo la decisión de diseño de **no utilizar herencia** entre las clases Anfitrion y Huesped, favoreciendo la simplicidad y la composición sobre la herencia para este caso. La evolución del diseño también implicó la adición de los getters necesarios (como getId(), getDocumento() y getContrasena()) a las clases Anfitrion y Huesped para facilitar la interacción segura de datos con el gestor principal, una necesidad que se hizo evidente durante la implementación de la funcionalidad de login.
* **Especialización de la Persistencia de Datos** Se tomó una decisión de diseño fundamental respecto a la actualización de datos en el almacenamiento permanente. Se determinó que los datos de las entidades Alojamiento, Anfitrion y Huesped son mayormente estáticos después de la carga inicial, ya que el desafío no incluye funcionalidades de usuario para su modificación. En consecuencia, se eliminaron los métodos toFileString() de estas clases. La responsabilidad de la persistencia de datos dinámicos recae exclusivamente en la clase Reservacion, cuyo estado cambia constantemente (creación, anulación, archivado). De esta manera, al finalizar el programa, GestorUdeaStay solo necesita reescribir el archivo de reservaciones activas, optimizando el proceso.
* **Adaptación de la Funcionalidad de Medición de Recursos** Se aclaró el requisito de la medición de rendimiento. En lugar de ser un reporte automático mostrado al final de cada tarea, se implementó como una **opción de menú bajo demanda** para el usuario. Esto permite una inspección del estado del sistema en cualquier momento de la ejecución. Adicionalmente, se definió que el contadorIteracionesGlobal sea **acumulativo** durante toda la sesión del programa, ofreciendo un panorama completo del costo computacional.

**Desafíos de Desarrollo y Soluciones Aplicadas**

El proceso de implementación desde el diseño inicial presentó varios desafíos técnicos, cuya resolución ha fortalecido la robustez del sistema.

* **Resolución de Errores de Compilación y Enlace** Durante el desarrollo inicial, se enfrentaron dos tipos de errores de compilación significativos:
  1. **Error de Enlazador (undefined reference):** Este error surgió al intentar crear arreglos dinámicos de objetos (ej. new Huesped[cupo]). El diagnóstico reveló que esta operación requiere que la clase posea un **constructor por defecto** público. La solución fue implementar los constructores por defecto faltantes en las clases Huesped y Reservacion, asegurando que inicialicen sus atributos a un estado válido.
  2. **Error de Calificadores const (discards qualifiers):** Se produjo al llamar métodos que modificaban el estado del objeto GestorUdeaStay (como incrementarContadorIteraciones) desde métodos declarados como const. Se resolvió aplicando el principio de **"const-correctness"**: se eliminó el calificador const de las firmas de los métodos que legítimamente necesitaban alterar el estado del objeto, como guardarReservacionesActivasEnArchivo() y finalizarSistema().
* **Diagnóstico y Solución de Errores en Tiempo de Ejecución**
  1. **"Crash" por Archivos No Encontrados:** El programa terminaba abruptamente al inicio. La causa fue identificada como un error de "No se pudo abrir el archivo...", lo que provocaba que el sistema intentara operar con datos no cargados. Se diagnosticó que el **directorio de trabajo** del IDE no coincidía con la ubicación de los archivos fuente. La solución fue establecer como procedimiento estándar la colocación de todos los archivos .csv en el directorio de compilación (build/.../debug), asegurando que el programa los encuentre consistentemente al ejecutarse.
  2. **Fallo Persistente en el Login:** Este fue el desafío más complejo. A pesar de ingresar credenciales visiblemente correctas, el login fallaba. Se implementó un **protocolo de depuración sistemático**, añadiendo cout en tres etapas clave: la carga de datos, la búsqueda de usuarios por ID y la comparación de contraseñas. Este proceso reveló que la causa raíz era un problema de integridad de datos, específicamente **espacios en blanco ocultos** leídos desde los archivos CSV que no eran eliminados por el parseador inicial. La solución definitiva fue reemplazar el parseador simple por una **función parsearLineaCSVInterno más robusta**, la cual incluye una subrutina trim() que limpia automáticamente los espacios de cada campo, garantizando la integridad de los datos en memoria y el éxito de las comparaciones de strings.