# Escuela Superior Politécnica Del litoral.

Tarea #3

## Diseño Software.

Grupo**: HomeStay**

**Integrantes:**

Alfonzo Yagual Diego Alexander

Patiño Castro Dhamar Amelie

Reyes Sandoval Daniela Nicole

Valle Franco Marlo Carlos

Paralelo 1

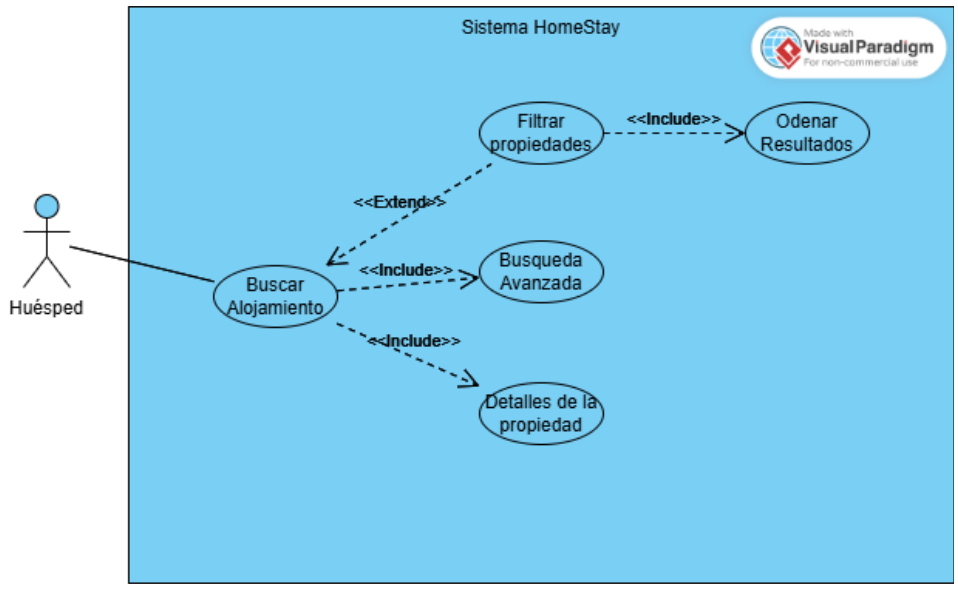
Prof: Ing. Jurado Mosquera David Alonso

2024 - 2025

# Contenido

# Diagrama de Casos de Usos.

## Buscar Alojamiento



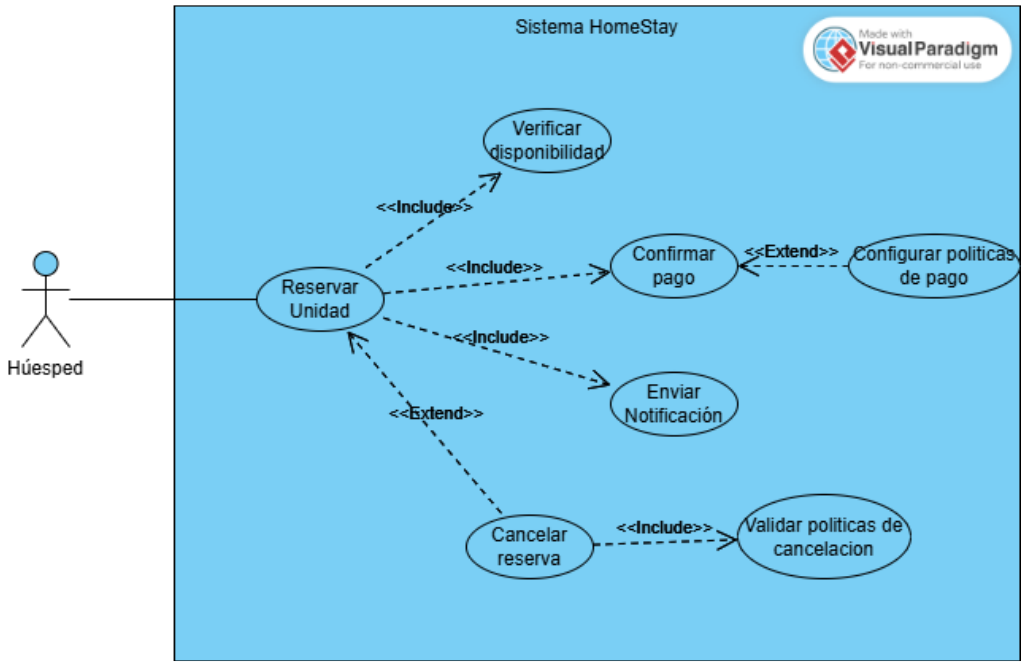

Actores: Huésped y Sistema HomeStay

Funcionalidad principal: Búsqueda y filtrado de propiedades

Flujos clave: Iniciar búsqueda → Aplicar filtros → Mostrar resultados → Seleccionar propiedad

Incluye manejo de errores como "No hay propiedades disponibles" y "Problemas de conexión"

## Reservar Unidad



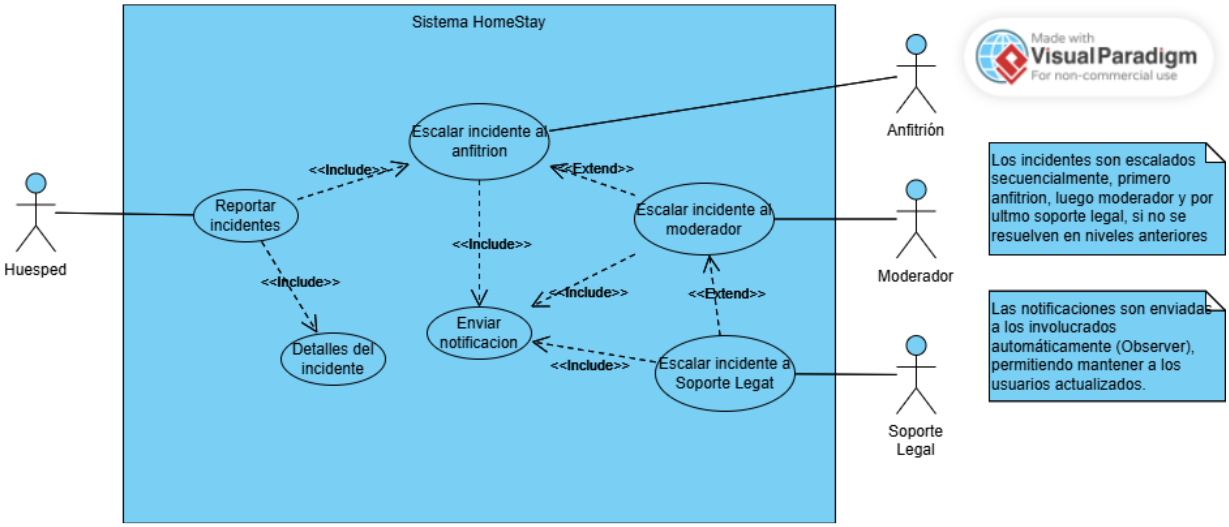

Actores: Huésped y Sistema HomeStay

Funcionalidad principal: Gestión del proceso de reserva

Flujos clave: Verificar disponibilidad → Procesar pago → Enviar notificaciones → Gestionar políticas

Incluye flujos alternativos para manejar errores de pago y cancelaciones

## C) Reportar Incidentes



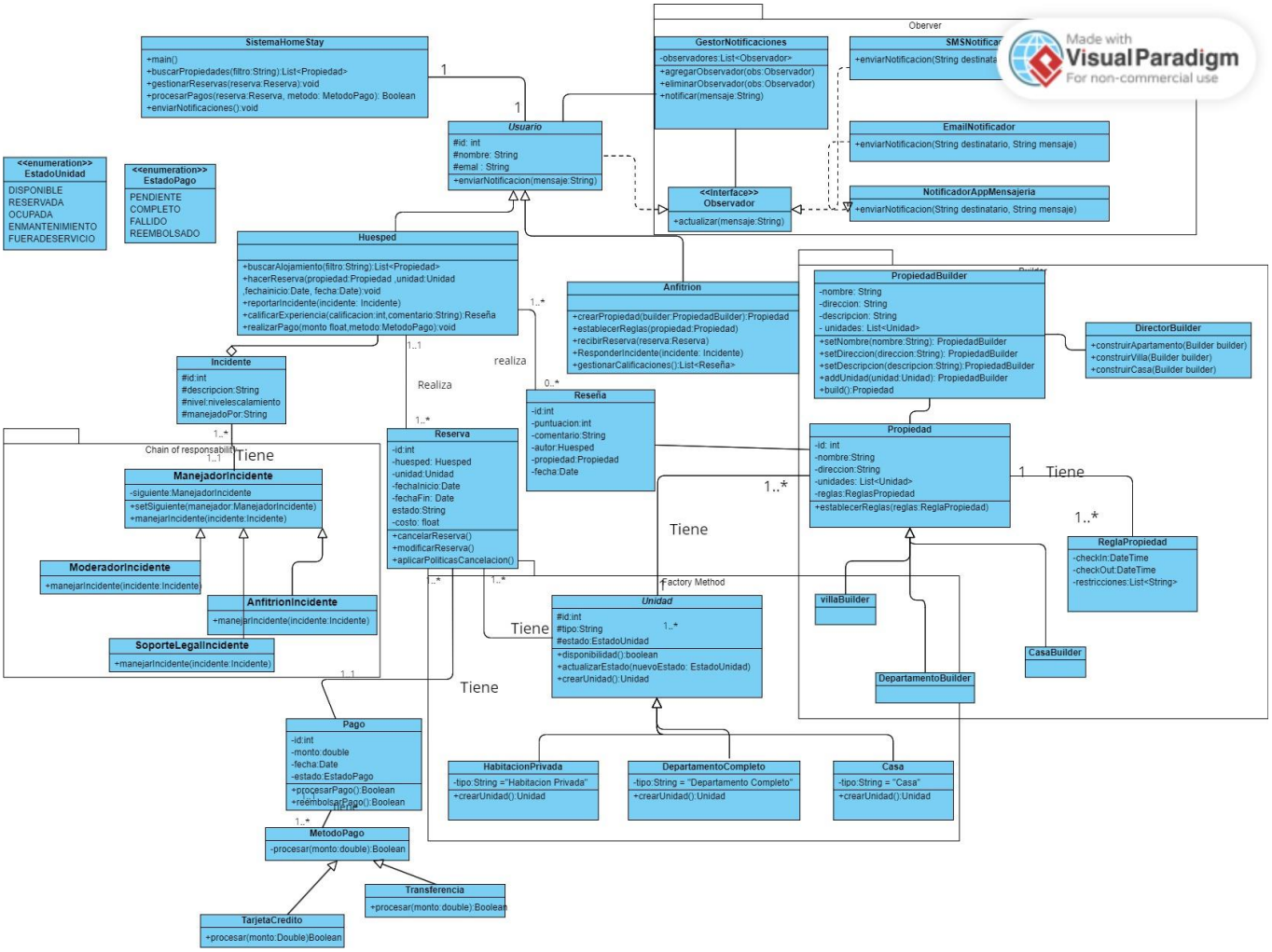

Actores: Huésped, Anfitrión, Moderador y Soporte Legal

Funcionalidad principal: Sistema de escalamiento de problemas

Flujos clave: Reportar → Escalar a anfitrión → Escalar a moderador → Escalar a soporte legal

Incluye resolución en diferentes niveles según la gravedad del incidente

# DIAGRAMA DE CLASES




## A) Patrón Factory Method

Clases principales: PropiedadBuilder

Relaciones: Creación de diferentes tipos de propiedades

Propósito: Facilitar la creación de distintos tipos de alojamiento

Extensibilidad: Permite agregar nuevos tipos de propiedades sin modificar código existente

## B) Patrón Observer

Clases principales: Observador y GestorNotificaciones

Relaciones: Sistema de notificaciones entre componentes

Propósito: Mantener informados a los usuarios sobre eventos importantes

Implementación: Mediante interfaz para asegurar consistencia

## C) Patrón Chain of Responsibility

Clases principales: ManejadorIncidente y sus subclases

Relaciones: Cadena de manejo de incidentes

Propósito: Gestionar el escalamiento de problemas

Estructura: Jerarquía de manejadores con responsabilidades específicas

## D) Patrón Builder

Clases principales: PropiedadBuilder

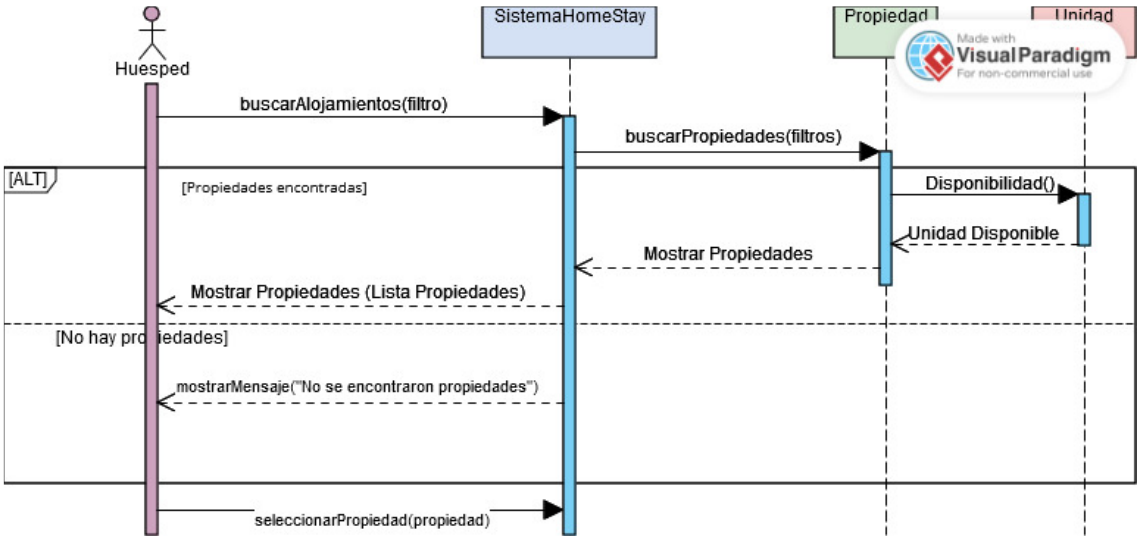
Relaciones: Construcción paso a paso de propiedades

Propósito: Simplificar la creación de propiedades complejas

Flexibilidad: Permite diferentes configuraciones de propiedades

# DIAGRAMAS DE SECUENCIA

## A) Buscar Alojamiento

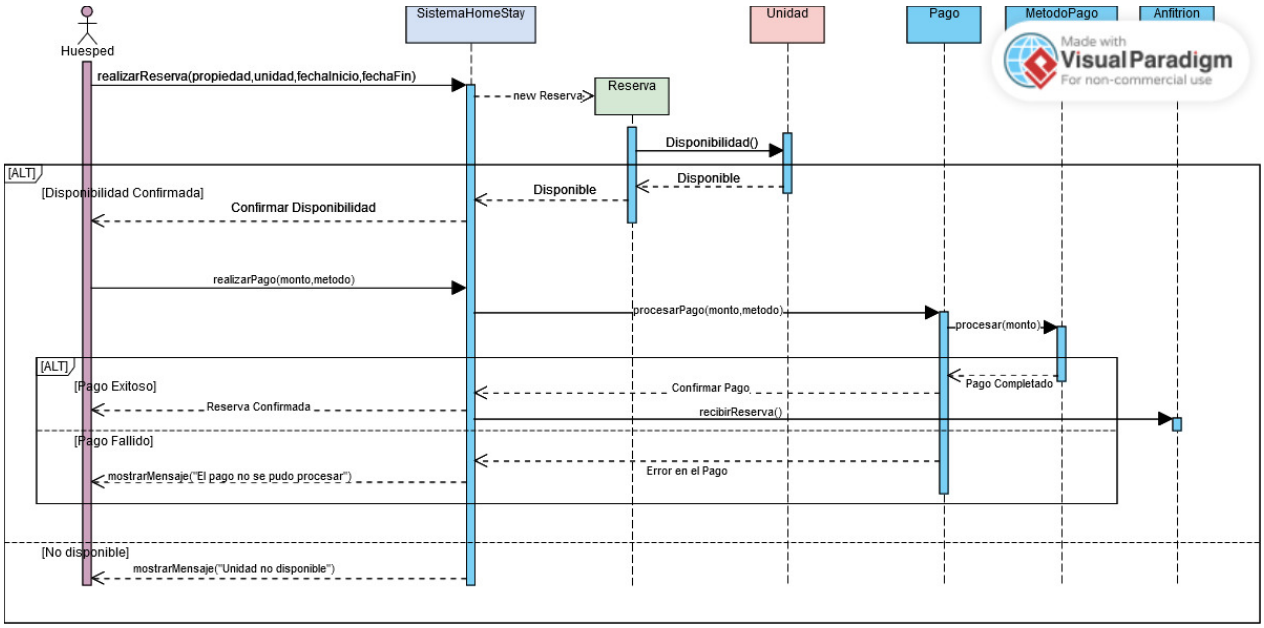



Secuencia: Usuario → Sistema → Base de datos → Resultados

Interacciones: Búsqueda, filtrado y presentación de resultados

Temporalidad: Muestra el orden de las operaciones

## B) Seleccionar Unidad

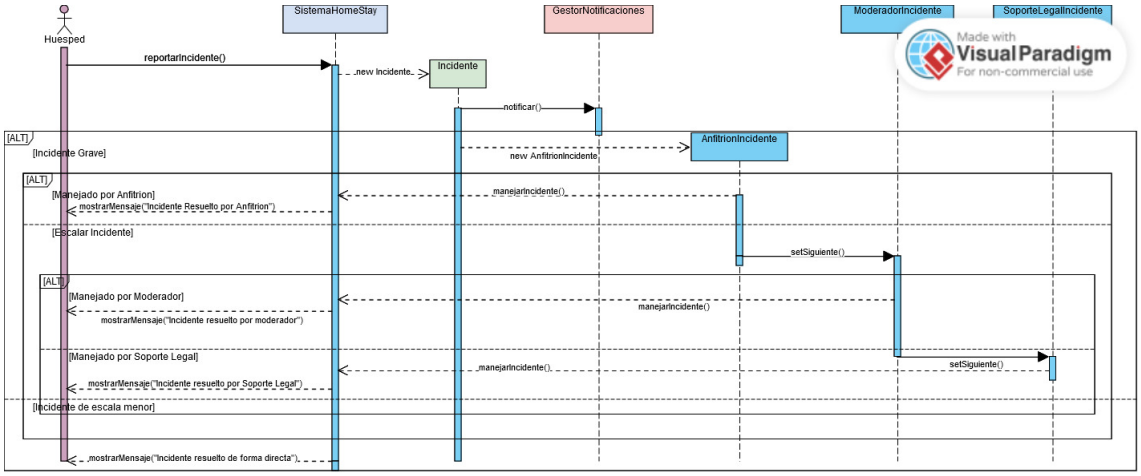



Secuencia: Usuario → Sistema → Unidad → Confirmación

Interacciones: Selección, verificación y reserva

Temporalidad: Proceso de selección paso a paso

## C) Reportar Incidente




Secuencia: Usuario → Sistema → Niveles de soporte

Interacciones: Reporte y proceso de escalamiento

Temporalidad: Flujo de resolución de incidentes

# Informe de Evaluación y Mejora del Proyecto

## 1. Flexibilidad del Diseño Original para Incorporar Cambios

El diseño original del sistema HomeStay, implementado con los patrones **Factory Method**, **Builder**, **Observer** y **Chain of Responsibility**, muestra una buena capacidad de adaptación. Sin embargo, los cambios recientes, como la incorporación de clases faltantes en los patrones **Builder** y **Factory Method**, y la funcionalidad interactiva con entradas por teclado, revelaron lo siguiente:

- **Fortalezas:**
  - **Factory Method:** La creación centralizada de tipos específicos de objetos permitió agregar nuevas clases de forma sencilla y sin afectar el código existente, cumpliendo el principio de abierto/cerrado (OCP).
  - **Builder:** La modularidad en la construcción de objetos complejos facilitó la extensión de configuraciones adicionales sin introducir inconsistencias.
  - **Chain of Responsibility:** La jerarquía bien estructurada permitió gestionar incidentes con un flujo claro, y la extensión del flujo interactivo demostró ser fluida.
  - **Observer:** Su uso para las notificaciones proporcionó un sistema desacoplado, flexible para agregar más canales de comunicación.
- **Limitaciones:**
  - El diseño original no contemplaba un enfoque interactivo, lo que requirió adaptar métodos y asegurarse de que la funcionalidad persistiera.
  - Algunas clases y métodos no estaban completamente implementados, lo que redujo la funcionalidad inicial y exigió revisiones para soportar la interacción.

## 2. Beneficios y Limitaciones de los Patrones de Diseño

**Beneficios:**

- **Factory Method:**
  - Proporcionó flexibilidad para integrar nuevas subclases para tipos específicos de alojamiento.
  - Centralizó la lógica de creación, mejorando la mantenibilidad.
- **Builder:**
  - Facilitó la creación de configuraciones detalladas de reservas y propiedades.
  - Permitió construir objetos paso a paso, manteniendo la claridad del código.
- **Observer:**
  - Simplificó la notificación de eventos importantes a múltiples usuarios sin alterar las clases base.

- **Chain of Responsibility:**
  - Garantizó un flujo organizado de resolución de incidentes, asegurando que los problemas escalen hasta el nivel adecuado.

**Limitaciones:**

- **Factory Method y Builder:** Dependían de configuraciones explícitas, lo que añadió complejidad al manejar objetos dinámicos.
- **Observer:** En casos con múltiples observadores simultáneos, la gestión de notificaciones podría ser menos eficiente sin un mecanismo centralizado de procesamiento.
- **Chain of Responsibility:** Requirió ajustes adicionales para soportar funcionalidad interactiva y garantizar un manejo adecuado si todos los niveles fallaban.

**3. Propuestas de Mejora**

Con base en la experiencia al implementar cambios y hacer el sistema interactivo, se proponen las siguientes mejoras:

1. **Centralización de Configuraciones Dinámicas:**
   - Implementar una clase o módulo que administre configuraciones comunes para patrones como Builder y Factory Method, permitiendo ajustar dinámicamente los parámetros requeridos.
2. **Manejo de Errores y Registro:**
   - Agregar un sistema de registro centralizado para capturar los errores y los pasos de escalado en la cadena de responsabilidad, lo que facilitará el monitoreo y la depuración.
3. **Optimización del Patrón Observer:**
   - Incorporar un gestor de eventos asíncronos para manejar grandes volúmenes de notificaciones sin afectar el rendimiento.
4. **Interactividad en el Builder:**
   - Extender los métodos de Builder para admitir configuraciones dinámicas desde entradas de usuario, adaptándose mejor a requisitos personalizados.
5. **Simulación y Pruebas Automatizadas:**
   - Crear un módulo para simular diferentes escenarios de uso y pruebas automatizadas para garantizar que los nuevos cambios sean compatibles con los patrones existentes.
6. **Documentación y Diagramas Actualizados:**
   - Actualizar los diagramas de clases, secuencia y casos de uso para reflejar los cambios recientes y facilitar futuras iteraciones.