Facultad de Ingeniería en Electricidad y Computación

Sistema de Alquiler de Hospedaje HomeStay

Tarea Grupal #1

Estrada Lara Windsor Alexander

Defranc Rojas Francisco Etienne

Arenas Tituaña Juan Pablo

Gonzalez Prieto Lenier

Escuela Superior Politécnica del Litoral

Diseño de Software - Paralelo 1

Jurado Mosquera David Alonso

01 de noviembre de 2025

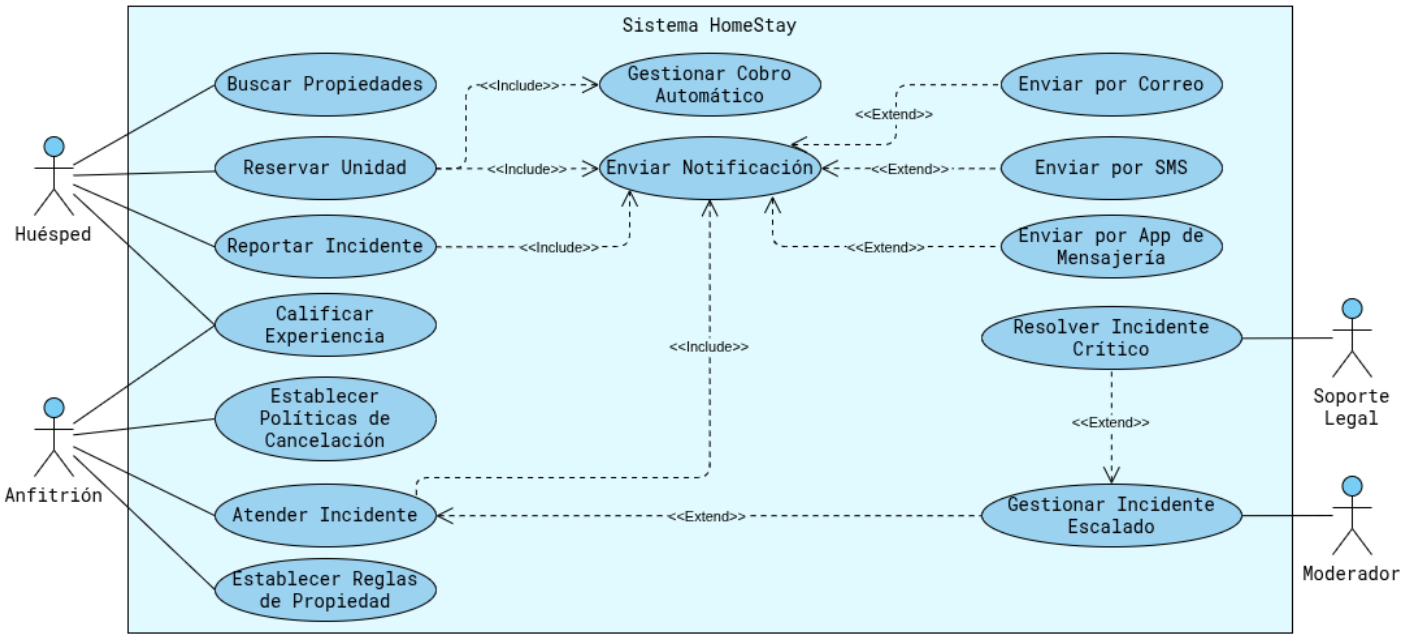
Índice

Sección A: Casos de Uso

Asunciones del Proyecto

- El sistema operará a nivel nacional.
- Las propiedades pueden tener múltiples unidades independientes.
- Los estados de las unidades (disponible, reservada, ocupada, en mantenimiento, fuera de servicio) cubren todos los escenarios posibles.
- Los criterios de búsqueda (ubicación, precio, tipo de alojamiento, servicios) son suficientes para satisfacer las necesidades de los usuarios.
- Los huéspedes y anfitriones interactúan directamente con el sistema para gestionar reservas, reseñas e incidentes.
- Los moderadores y el equipo legal son actores externos que intervienen solo en escalamientos de incidentes.
- Las reservas están sujetas a disponibilidad y políticas de cancelación definidas por el anfitrión.
- Es posible implementar múltiples canales de notificación (email, SMS, apps de mensajería) sin cambiar el funcionamiento del sistema.

Diagrama de Casos de Uso



Descripción de los Casos de Uso Seleccionados

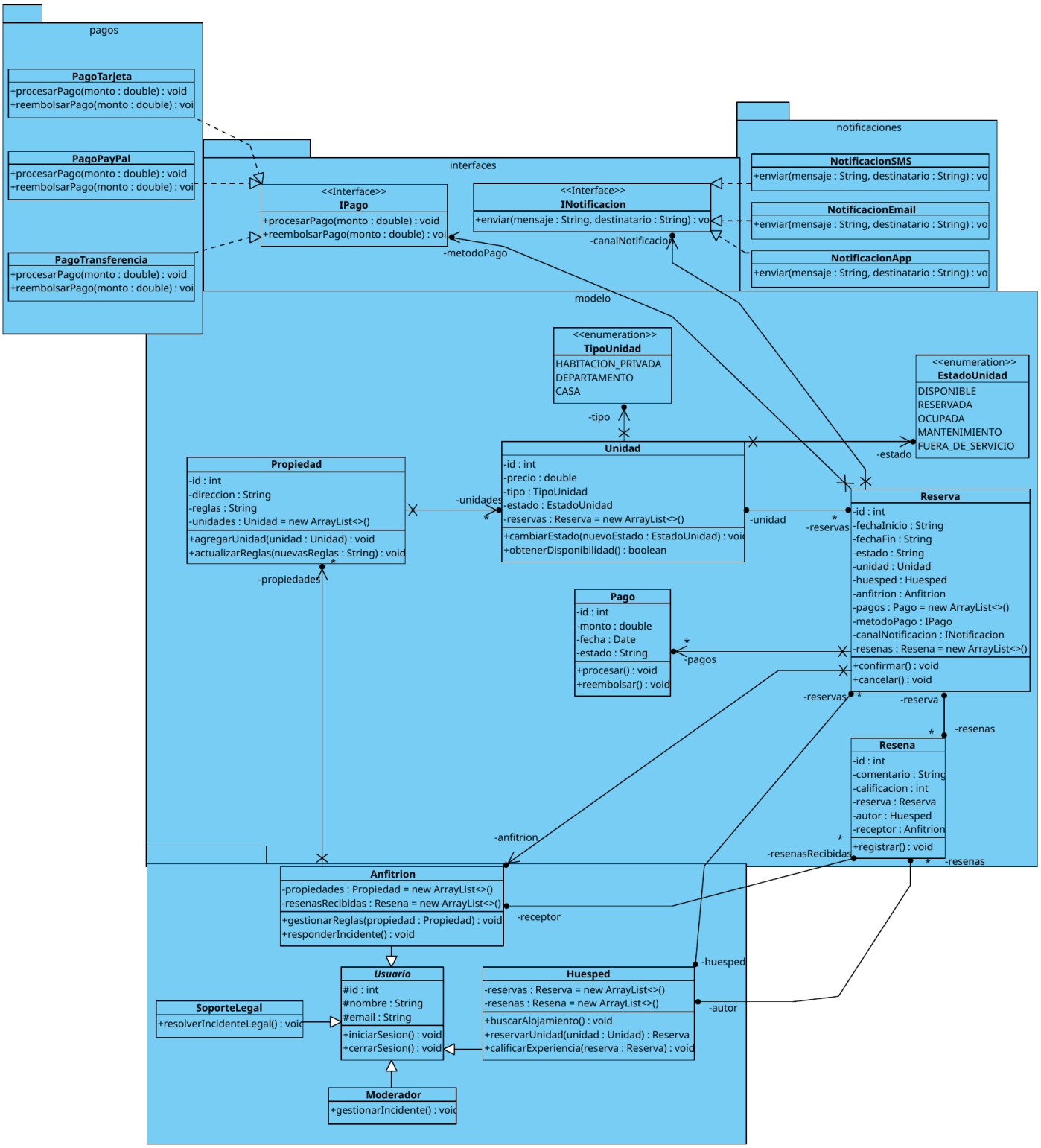
CDU 01 - Buscar Alojamiento	
Actores	Huésped
Precondiciones necesarias	El huésped debe estar registrado en el sistema.
Flujo Principal	El huésped accede al sistema. Ingresa criterios de búsqueda (ubicación, fechas, precio, tipo de unidad). El sistema filtra y muestra resultados disponibles. El huésped selecciona una unidad para ver detalles.
Flujos Alternativos	Si no hay resultados, el sistema sugiere ampliar criterios.
Postcondiciones	El huésped visualiza las opciones de alojamiento disponibles.

CDU 02 - Reservar Unidad	
Actores	Huésped, Sistema de Notificaciones, Anfitrión.
Precondiciones necesarias	El huésped debe haber iniciado sesión y seleccionado una unidad disponible.
Flujo Principal	El huésped selecciona fechas y unidad. El sistema valida disponibilidad. El sistema solicita datos de pago. El huésped ingresa la información. El sistema procesa el pago (Ref. Include). El sistema confirma la reserva. El sistema envía notificación al huésped y al anfitrión.
Flujos Alternativos	Si el pago falla, se notifica al huésped y se cancela la reserva. Si la unidad ya no está disponible, se sugiere otra opción.
Postcondiciones	La unidad queda en estado de “reservada” y el huésped recibe confirmación.

CDU 03 - Calificar Experiencia	
Actores	Huésped, Anfitrión, Sistema de Notificaciones.
Precondiciones necesarias	El huésped debe haber completado una estancia.
Flujo Principal	El huésped accede a su historial de reservas. Selecciona la estancia finalizada. Ingresa una calificación y comentario. El sistema registra la reseña. El sistema extienda la acción para “Enviar notificación al anfitrión.”
Flujos Alternativos	Si el huésped no deja reseña, el sistema puede enviar un recordatorio.
Postcondiciones	La reseña queda registrada y el anfitrión recibe notificación.

CDU 04 - Reportar Incidente	
Actores	Huésped, Anfitrión, Moderador, Soporte Legal.
Precondiciones necesarias	El huésped debe tener una reserva activa o finalizada.
Flujo Principal	El huésped reporta un incidente desde la plataforma. El sistema notifica al anfitrión. El anfitrión atiende el incidente.
Flujos Alternativos	Si no se resuelve, el caso se “extiende” a un Moderador. Si el Moderador no logra resolverlo, se escala a Soporte Legal.
Postcondiciones	El huésped puede cancelar el reporte si el problema se resuelve directamente. El incidente queda registrado como estado “resuelto” o “escalado”.

Sección B: Diagrama de Clases



Single Responsibility Principle (SRP)

Cada clase cumple con SRP porque tiene una única responsabilidad bien definida y no mezcla lógicas ajenas: los usuarios gestionan únicamente sus roles, las entidades de dominio representan conceptos específicos (propiedad, unidad, reserva, pago, reseña) y las interfaces encapsulan contratos mínimos. Esto asegura que cualquier cambio en una funcionalidad afecte solo a la clase correspondiente, evitando dependencias innecesarias.

Clases:

- **Huésped:** Solo gestiona sus reservas y reseñas.
- **Anfitrión:** Se limita a administrar propiedades y reseñas recibidas.
- **Propiedad:** Se centra en sus reglas y en contener unidades.
- **Unidad:** Se ocupa únicamente de su estado y disponibilidad.
- **Reserva:** Administra el ciclo de vida de la reserva (confirmar, cancelar).
- **Pago:** Registra transacciones, sin gestionar reservas.
- **Reseña:** Almacena calificaciones y comentarios, sin intervenir en pagos ni reservas.

Relaciones:

- **Propiedad – Unidad:** Asegura que la gestión de unidades esté delegada en la clase correcta.
- **Huésped – Reserva:** Refleja que solo el huésped crea y gestiona sus reservas.
- **Anfitrión – Propiedad:** Muestra que solo el anfitrión administra propiedades.

Open/Closed Principle (OCP)

El sistema está abierto a extensión y cerrado a modificación porque las dependencias hacia interfaces permiten añadir nuevas funcionalidades sin alterar el código existente. Así, se puede incorporar nuevos métodos de pago o canales de notificación sin modificar "Reserva", y se pueden crear nuevos tipos de usuario sin alterar la clase base "Usuario".

Clases:

- "**Reserva**" depende de "**IPago**" y "**INotificacion**", lo que permite añadir nuevos métodos de pago o notificación sin modificarla.
- **Usuario:** Al ser abstracta, permite crear nuevos tipos de usuario sin alterar su definición.

Relaciones:

- **Reserva – IPago:** Abierta a nuevos métodos de pago.
- **Reserva – INotificacion:** Abierta a nuevos canales de notificación.
- **Usuario – subclases:** Abierto a nuevos roles de usuario.

Liskov Substitution Principle (LSP)

Se cumple porque las subclases y las implementaciones de interfaces pueden sustituir a sus supertipos sin alterar el comportamiento esperado. Esto garantiza que cualquier instancia de una subclase pueda usarse en lugar de su superclase sin romper la lógica del sistema.

Clases:

- "**Huésped**", "**Anfitrión**", "**Moderador**", "**SoporteLegal**" heredan de "**Usuario**" y pueden sustituirlo en cualquier contexto.
- "**PagoTarjeta**", "**PagoPayPal**", "**PagoTransferencia**" implementan "**IPago**" y pueden sustituirse entre sí.
- "**NotificacionEmail**", "**NotificacionSMS**", "**NotificacionApp**" implementan "**INotificacion**" y son intercambiables.

Relaciones:

- **Usuario – subclases:** Garantiza que cualquier subclase pueda usarse donde espere un "**Usuario**".
- **IPago – implementaciones:** Asegura que todas las implementaciones piedad sustituirse en "**Reserva**".
- **INotificacion —- implementaciones:** Asegura que todas las notificaciones puedan sustituirse en "**Reserva**".

Interface Segregation Principle (ISP)

El principio se cumple porque las interfaces son específicas y ligeras, evitando que las clases implementen métodos innecesarios. Cada interfaz define un contrato mínimo y claro, lo que asegura que las dependencias sean precisas y no sobrecarguen a las clases que las implementan.

Clases:

- **IPago:** Define solo lo necesario para procesar y reembolsar pagos.
- **INotificacion:** Define únicamente el envío de mensajes.
- **Reserva:** Depende de ambas interfaces, pero cada una cumple un rol específico.

Relaciones:

- **Reserva – IPago:** Muestra que solo depende de lo necesario para procesar pagos.
- **Reserva – INotificacion:** Muestra que solo depende de lo necesario para enviar notificaciones.

Dependency Inversion Principle (DIP)

El sistema cumple con DIP porque las clases de alto nivel dependen de abstracciones y no de implementaciones concretas. Esto permite cambiar las implementaciones sin modificar la lógica de negocio, manteniendo el sistema flexible y desacoplado.

Clases:

- "**Reserva**" depende de "**IPago**" e "**INotificacion**" (abstracciones)
- Las implementaciones ("**PagoTarjeta**", "**NotificacionEmail**", etc.) dependen de las interfaces, no al revés.

Relaciones:

- **Reserva – IPago:** Inversión de dependencia hacia la abstracción.
- **Reserva – INotificacion:** Inversión de dependencia hacia la abstracción.
- **Implementaciones – Interfaces:** Asegura que las clases concretas dependen de contratos, no de clases de alto nivel.

Sección C: Diagramas de Secuencia

Diagramas de Secuencia

Los 4 diagramas de secuencia que representen la ejecución de los casos de uso seleccionados en la Sección A, detallando los mensajes enviados entre objetos, la creación de instancias y las llamadas a métodos, asegurando consistencia con el diagrama de clases.

DIAGRAMA CDU-01

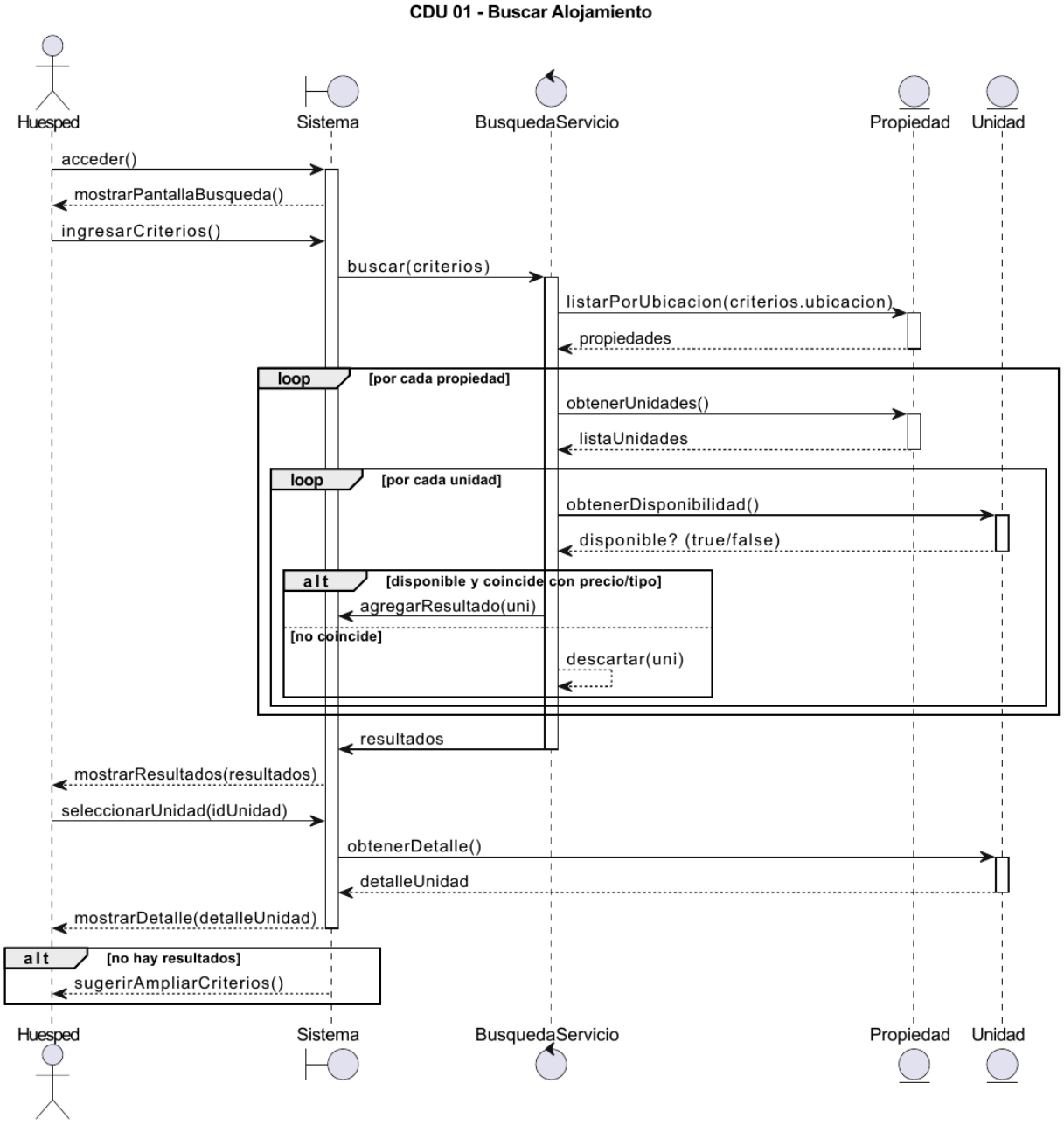


DIAGRAMA CDU-02

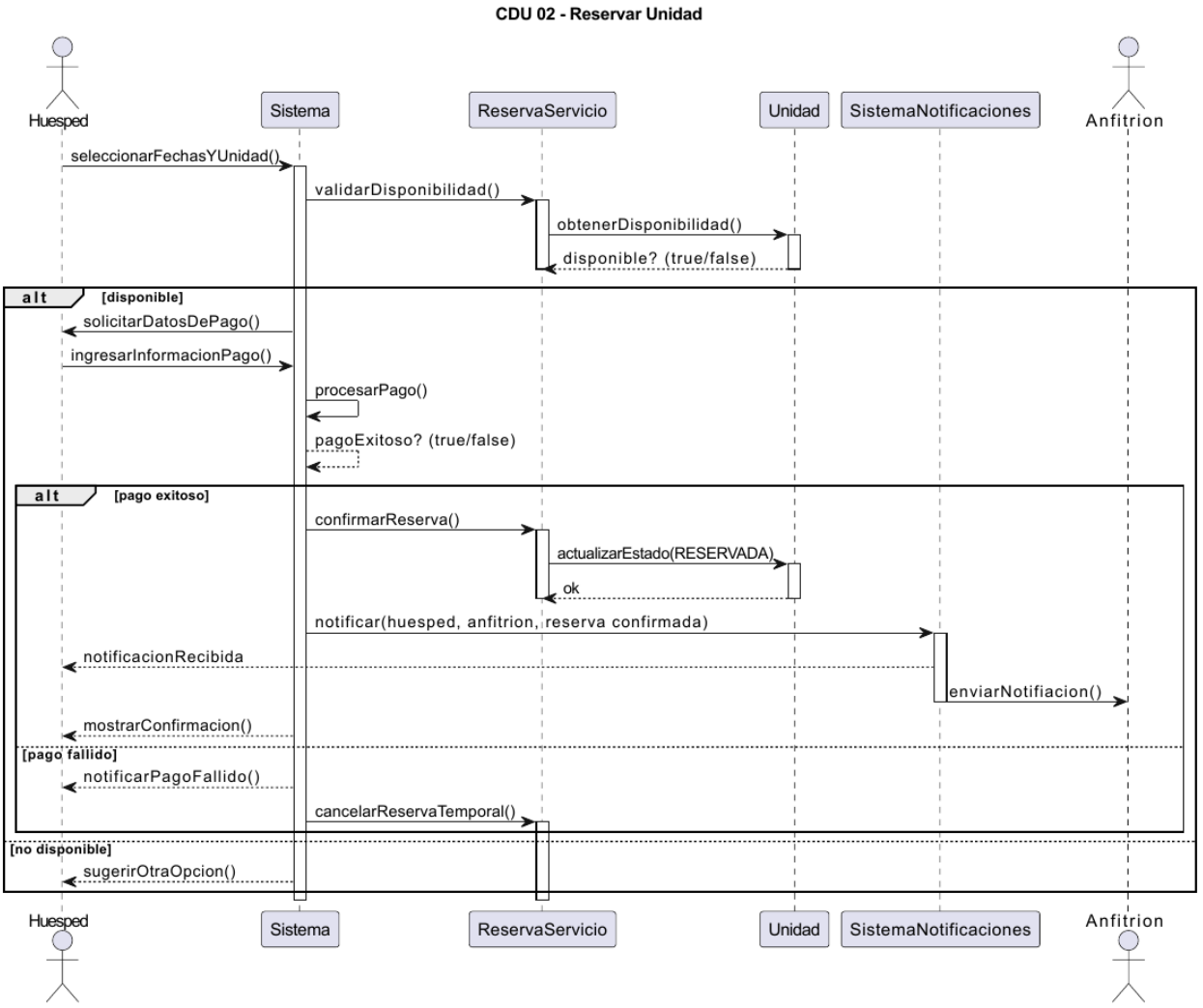


DIAGRAMA CDU-03

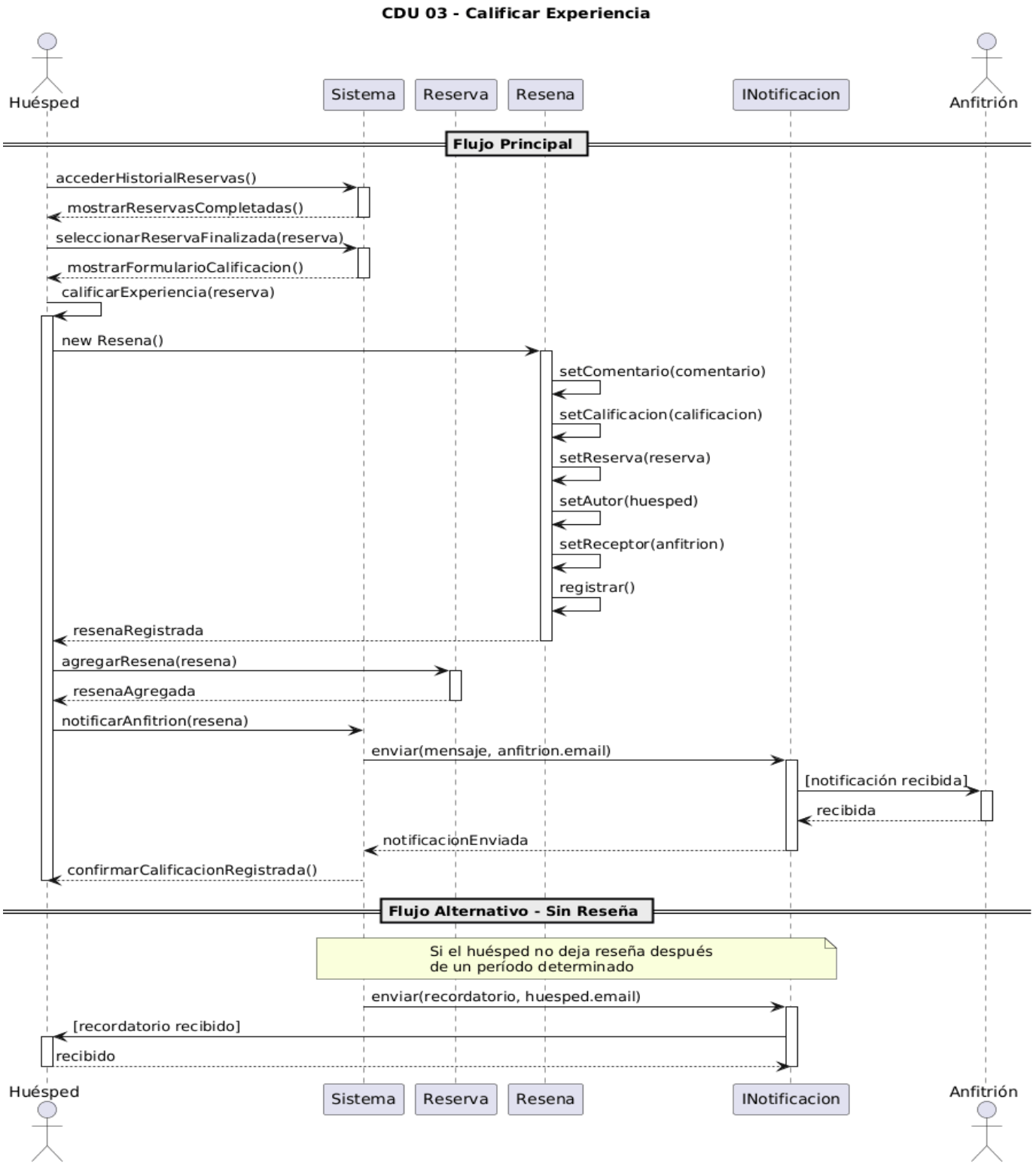
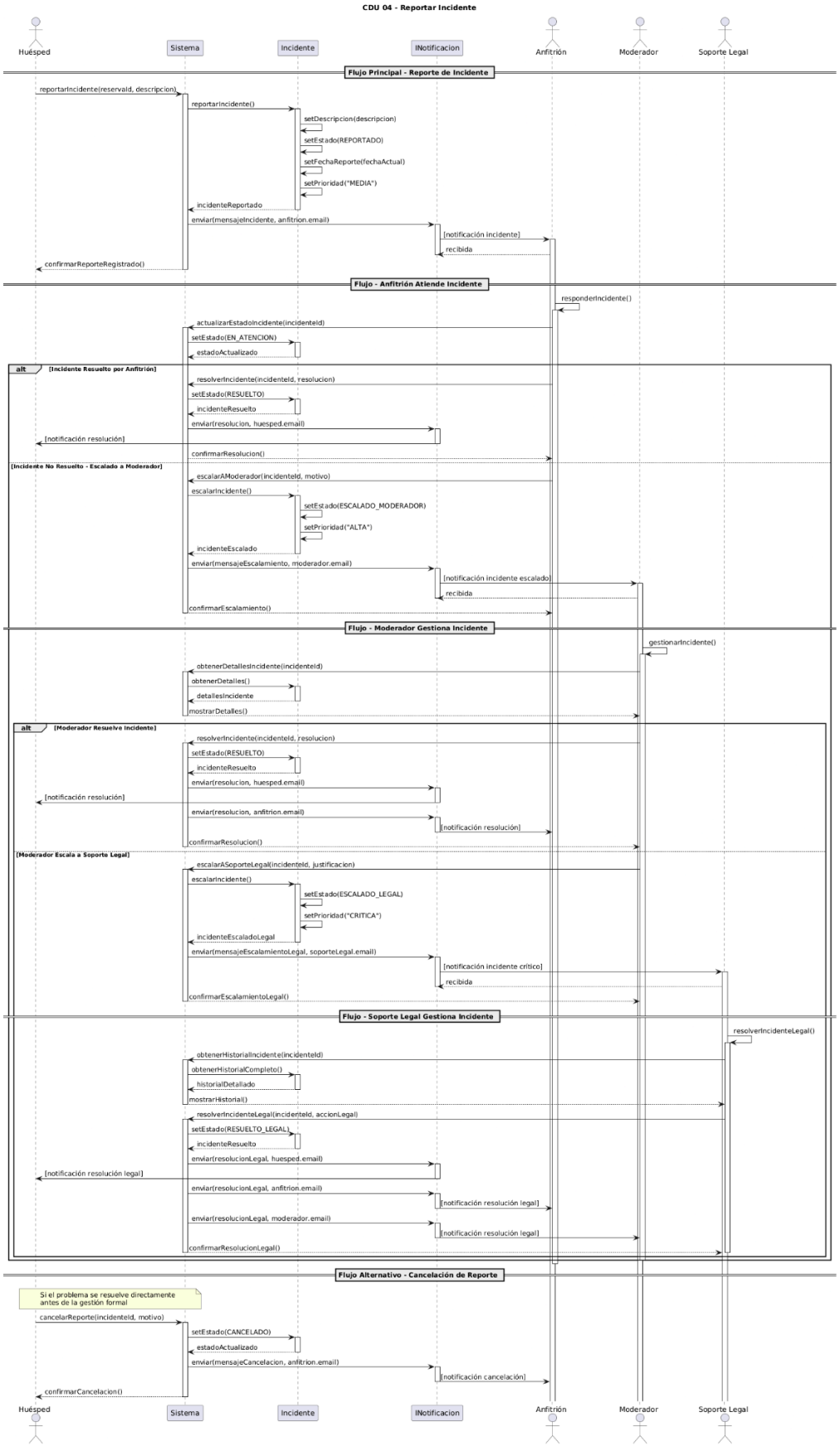


DIAGRAMA CDU-04



Sección D: Generación del Código

Enlace al Repositorio de Github: https://github.com/Winareku/Tarea01-SOLID_UML

Organización de Clases

El código está estructurado en los siguientes paquetes para separar responsabilidades:

- com.espol.usuarios: Contiene las clases que representan a los diferentes tipos de usuarios del sistema.
- com.espol.modelo: Contiene las clases del dominio principal del negocio, como Propiedad, Unidad, Reserva, etc.
- com.espol.pagos: Implementa diferentes estrategias para procesar pagos.
- com.espol.notificaciones: Implementa diferentes estrategias para enviar notificaciones.
- com.espol.interfaces: Define los contratos (interfaces) para los sistemas de pagos y notificaciones, permitiendo la extensibilidad.

Commits de realización de Carpetas de Clases por cada Integrante