|  |
| --- |
| Goblanch! |
| GB Inventory System (0.4.2) |
| Documentación técnica y de configuración |

|  |
| --- |
| GONZALO BLANCH DOMÍNGUEZ  22-10-2025 |

Contenido

[**1.** **Introducción** 1](#_Toc212056216)

[1.1. Características principales 2](#_Toc212056217)

[1.2. Objetivos 2](#_Toc212056218)

[2. Arquitectura del sistema 3](#_Toc212056219)

[2.1 Flujo de datos 3](#_Toc212056220)

[2.2 Componentes principales 4](#_Toc212056221)

[2.2.1 Dominio 4](#_Toc212056222)

[2.2.2 Aplicación 4](#_Toc212056223)

[2.2.3 Infraestructura 5](#_Toc212056224)

[2.3 Ventajas de diseño 5](#_Toc212056225)

[3. Estructura del proyecto 5](#_Toc212056226)

[3.1 Estructura general 6](#_Toc212056227)

[3.2 Descripción por carpeta 7](#_Toc212056228)

[3.3 Reglas de organización 10](#_Toc212056229)

# **Introducción**

GB Inventory System es un sistema de inventario modular diseñado para su integración en proyectos desarrollados con Unity.

Su principal objetivo es ofrecer una infraestructura reutilizable, desacoplada de la interfaz y configurable desde el editor, permitiendo la creación y modificación de inventarios sin alterar el código base.

El sistema se fundamenta en tres capas principales, con responsabilidades separadas:

|  |  |  |
| --- | --- | --- |
| Capa | Descripción | Dependencias |
| Domain | Contiene la lógica del inventario, los modelos de datos, las políticas de apilamiento (stacking) y los filtros de compatibilidad entre ítems y slots. No depende del motor ni de clases de Unity. | Ninguna |
| application | Expone una interfaz de uso (IInventoryService) que actúa como punto de acceso a las operaciones del inventario. Gestiona las acciones de alto nivel (añadir, mover, usar, dividir) y coordina el modelo con los efectos y políticas de uso. | Domain |
| Infrastructure | Gestiona la integración con Unity. Proporciona los ScriptableObjects de configuración (ítems, tipos, perfiles del slot), los proveedores de datos (Providers) y el InventoryInstaller, encargado de inicializar y cablear las dependencias. | Domain, Application |

## 1.1. Características principales

* Arquitectura desacoplada y extensible basada en principios de *Clean Archtecture*.
* Configuración a través de *ScriptableObjects,* permitiendo la creación de inventarios sin intervención de programación.
* Sistema de slots configurables: cada slot define sus propias reglas de apilamiento, tipos permitidos y filtros por etiquetas.
* Compatibilidad con un sistema opcional de fases de uso.
* Soporte para efectos de ítems mediante un registro (*EffectRegistry*) que asocia claves de efecto (*EffectKey*) con implementación de *IItemEffect*.
* Implementación independiente de la interfaz de usuario, lo que permite su integración en juegos 2D, 3D o sistemas de UI personalizados.

## 1.2. Objetivos

1. Centralizar la lógica del inventario en un núcleo común reutilizable entre proyectos.
2. Permitir la configuración completa del comportamiento del inventario desde el editor.
3. Facilitar el testeo unitario y la validación de reglas de apilamiento y compatibilidad.
4. Separar las responsabilidades de diseño (configuración de datos) y desarrollo (lógica).
5. Mantener la independencia entre el sistema y la capa de presentación (*UI o HUD*).

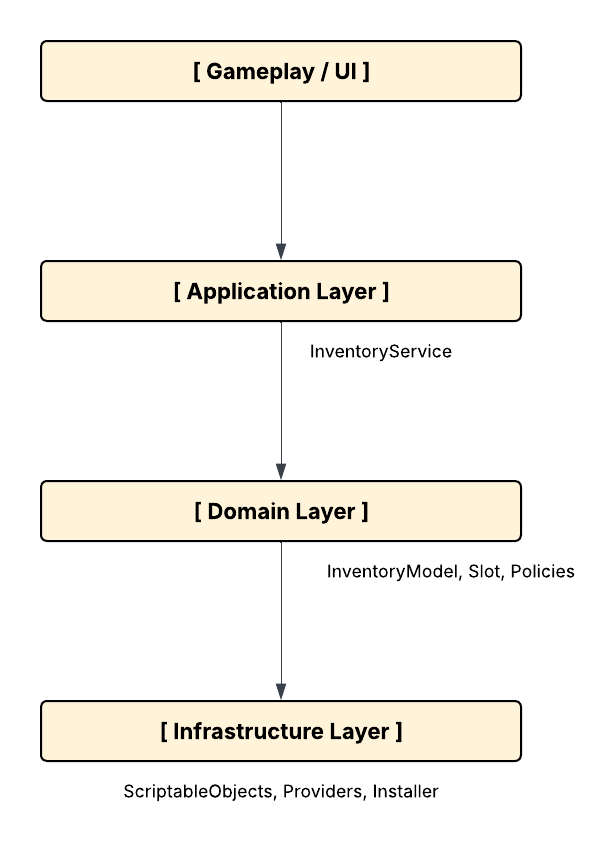


Diagrama 1: Arquitectura del sistema.

# Arquitectura del sistema

GB Inventory System implementa una arquitectura en capas que separa la lógica de la interfaz de uso y la integración con *Unity*.

El diseño sigue un patrón *Clean Architecture / Domain-Driven Design (DDD)* adaptado al entorno de *Unity*, con el objetivo de garantizar el desacoplamiento, testabilidad y extensibilidad.

## 2.1 Flujo de datos

El flujo de ejecución típico entre capas es el siguiente:

1. El gameplay o la UI invocan una acción a través del *InventoryService* (por ejemplo, *TryAdd, TryUse, TrySplit, etc.*).
2. *Application* valida y dirige la operación hacia el modelo de dominio (*InventoryModel*), aplicando las políticas de apilamiento y filtrado configuradas.
3. *Domain* ejecuta la lógica del inventario (añadir ítems, fusionar stacks, mover slots, validar capacidad).
4. Si la operación implica efectos o uso de ítems:
   1. *Application* consulta el *IEffectRegistry* para resolver la acción asociada.
   2. Si el sistema de fases está activo, se consulta la política de uso (*IUsagePolicy*) para validar si el ítem puede usarse en la fase actual.
5. *Infrastructure* provee la información necesaria (definición de ítems, perfiles de slot, configuraciones de efectos) mediante *Providers* conectados a *ScriptableObjects*.

El resultado es un flujo de arriba hacia abajo (desde *gameplay* hasta dominio) y de datos configurables hacia arriba (desde los *ScriptableObjects* hasta el modelo).

## 2.2 Componentes principales

### 2.2.1 Dominio

* ***InventoryModel:*** núcleo lógico del sistema. Contiene los slots y gestiona las operaciones de adición, división, movimiento y capacidad.
* ***Slot:*** representa una posición del inventario y mantiene un *Stack* (definición + cantidad).
* ***SlotProfile:*** determina las restricciones del slot (tipos y etiquetas permitidas, si admite *stacking* y el máximo).
* ***Políticas:***
  + ***IStackingPolicy:*** define los límites globales o por tipo de item.
  + ***ISlotFilterPolicy:*** valida la compatibilidad de un ítem con un slot.
  + ***IUsagePhasePolicy:*** valida si un ítem puede usarse en la fase actual (opcional).
* ***Entidades de apoyo:***
  + ***ItemMeta:*** contiene metadatos del ítem (TypeId, Tags, etc.).
  + ***UseResult:*** estructura de salida con resultado de un uso (*Success, ConsumeOne, Message*)

### 2.2.2 Aplicación

* ***InventoryService:*** fachada de acceso al dominio. Expone una interfaz de alto nivel (*IInventoryService)* para *gameplay y UI.*
* ***Coordina con:***
  + ***IInventory*** (modelo interno).
  + ***IEffectRegistry*** (efectos).
  + ***IUsagePhasePolicy*** (fases, si existen).

### 2.2.3 Infraestructura

* ***ScriptableObjects*** de configuración:
  + *ItemDefinition, ItemTypeDefinition, SlotProfileDefinition.*
* ***Databases:*** agrupan las definiciones (*ItemDatabase, SlotProfileDatabase).*
* ***Providers:*** transforman los datos del SO en estructuras de dominio (*ItemMeta, SlotProfile, etc.).*
* ***EffectRegistry:*** gestiona el registro y la resolución de efectos (*effectKey 🡪 IItemEffect)*.
* ***InventoryInstaller:*** configura el sistema y construye el sistema completo en escena.

## 2.3 Ventajas de diseño

* **Separación de responsabilidades:** cada capa cumple una función específica y no depende de detalles de implementación de las demás.
* **Reutilización:** la capa de dominio puede integrarse en distintos juegos sin modificaciones.
* **Testabilidad:** el dominio y la aplicación pueden probarse con tests unitarios sin cargar Unity.
* **Extensibilidad:** se pueden añadir nuevas políticas, efectos o fases sin alterar las clases existentes.
* **Configurabilidad:** los diseñadores pueden definir nuevos ítems o reglas directamente desde el editor.

# Estructura del proyecto

GB Inventory System se organiza en una jerarquía de carpetas que refleja la separación por capas definida en su arquitectura.

Cada carpeta tiene una función específica dentro del flujo del sistema y contiene solo los elementos correspondientes a su responsabilidad.

Esta estructura permite mantener el código modular, legible y fácilmente exportable como paquete.

## 3.1 Estructura general

1. Assets/

2. └── InventorySystem/

3. ├── Domain/

4. │ ├── Abstractions/

5. │ ├── Entities/

6. │ ├── Policies/

7. │ └── ValueObjects/

8. │

9. ├── Application/

10. │ ├── Abstractions/

11. │ ├── Services/

12. │ └── Contexts/

13. │

14. ├── Infrastructure/

15. │ ├── Definitions/

16. │ │ └── ScriptableObjects/

17. │ ├── Providers/

18. │ ├── Effects/

19. │ └── Installers/

20. │

21. └── Tests/

22. ├── Domain/

23. ├── Application/

24. └── Infrastructure/

25.

## 3.2 Descripción por carpeta

**Domain**

|  |  |  |
| --- | --- | --- |
| Carpeta | Contenido | Descripción |
| Domain | Código puro C# (sin dependencias de Unity). | Contiene la lógica central del inventario y los contratos principales. Todas las clases de esta capa deben ser serializables y testeables sin ejecutar Unity. |
| Domain/Abstractions | Interfaces base del sistema. | Define los contratos: IInventory, ISlot, IStack, IItemEffect, IEffectRegistry, IStackingPolicy, ISlotFilterPolicy, IUsagePhasePolicy. |
| Domain/Entities | Clases principales del modelo. | Incluye InventoryModel (núcleo del sistema) y Slot. |
| Domain/Policies | Implementaciones de las políticas del dominio. | Contiene las reglas de stacking (SimpleStackingPolicy), filtrado (SimpleSlotFilterPolicy) y uso por fase (DefaultUsagePhasePolicy). |
| Domain/ValueObjects | Estructuras de datos inmutables. | Contiene SlotProfile, ItemMeta, UseResult y otros objetos de valor utilizados en la lógica. |

**Application**

|  |  |  |
| --- | --- | --- |
| Carpeta | Contenido | Descripción |
| Application | Capa intermedia que expone el sistema al gameplay. | Contiene la fachada InventoryService y las interfaces públicas de acceso. |
| Application/Abstractions | Interfaces de servicios. | Define IInventoryService (interfaz de alto nivel para el inventario). |
| Application/Services | Implementaciones concretas. | Incluye InventoryService, encargado de coordinar el modelo de dominio, el registro de efectos y las políticas de fase. |
| Application/Contexts | Contextos o adaptadores de juego. | Contiene clases como SimpleTurnContext o TurnContextProvider, que representan el estado de fase o turno actual. |

**Infrastructure**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Carpeta | | Contenido | | Descripción |
| Infrastructure | | Código dependiente de Unity. | | Gestiona la configuración mediante ScriptableObjects, los proveedores de datos y la inicialización del sistema. |
| Infrastructure/Definitions/ScriptableObjects | | Definiciones configurables desde el editor. | | Contiene ItemDefinition, ItemTypeDefinition, SlotProfileDefinition y sus bases de datos (ItemDatabase, SlotProfileDatabase). |
| Infrastructure/Providers | | Adaptadores entre los ScriptableObjects y el dominio. | | Clases como SoItemMetadataProvider, SoSlotProfileProvider y SoItemEffectInfoProvider, que convierten los datos de Unity en estructuras lógicas. |
| Infrastructure/Effects | | Implementaciones y registro de efectos. | | Contiene EffectRegistry y las clases concretas de efectos (TestEffect, HealEffect, etc.). |
| Infrastructure/Installers | Inicializadores del sistema. | | InventoryInstaller crea e inyecta las dependencias (modelo, políticas, registro de efectos) al iniciar la escena. | |

**Tests**

|  |  |  |
| --- | --- | --- |
| Carpeta | Contenido | Descripción |
| Tests | Pruebas unitarias y de integración. | Contiene los conjuntos de pruebas para cada capa. |
| Tests/Domain | Pruebas del modelo y las políticas. | Valida el comportamiento del InventoryModel, stacking, capacidad y perfiles. |
| Tests/Application | Pruebas del servicio. | Evalúa la interfaz pública (IInventoryService) y los flujos con efectos y fases. |
| Tests/Infrastructure | Pruebas de integración con ScriptableObjects. | Comprueba la carga de datos y el funcionamiento de los proveedores (Providers). |

## 3.3 Reglas de organización

1. **Separación estricta de dependencias;**
   1. El *Domain* nunca debe hacer referencia a *UnityEngine* ni a clases fuera de su capa.
   2. *Application* solo depende del *Domain.*
   3. *Infrastructure* es la única capa que depende de Unity y de las demás capas.
2. **Nombres consistentes:**
   1. Los nombres de clases y archivos deben reflejar su función.
   2. Ejemplo: *SimpleSlotFilterPolicy 🡪* Implementa *ISlotFilterPolicy*.
   3. Ejemplo: *SoItemMetadataProvider 🡪* Proveedor basado en *ScriptableObject.*
3. **Carpetas paralelas entre capas:** siempre que se añada una nueva funcionalidad (por ejemplo, persistencia o UI), se recomienda replicar esta estructura en un nuevo módulo:

1. Assets/InventorySystem/

2. ├── Persistence/

3. ├── UI/

4. └── Networking/

5.

1. **Ficheros asmdef:** cada carpeta raíz (*Domain, Application, Infrastructure, Tests*) debe tener su propio archivo *.asmdef.* Esto facilita el control de dependencias y la compilación independiente por capa.

# Componentes principales del sistema

GB Inventory System se compone de varios módulos funcionales distribuidos entre las capas de *Domain, Application e Infrastructure*.

Cada módulo cumple un rol específico dentro de la arquitectura, garantizando la separación entre lógica, interfaz de uso y configuración.

A continuación, se describen los componentes más relevantes del sistema.

## 4.1 Dominio (*Domain*)

Contiene la lógica principal del inventario, sus entidades, políticas y estructuras de datos inmutables. Todo el código en esta capa está diseñador para funcionar sin dependencias de Unity, facilitando su testeo automatizado.

### 4.1.1 *InventoryModel*

1. **Ubicación**: Domain/Entities/InventoryModel.cs

**Responsabilidad:** es el núcleo del sistema. Gestiona los slots del inventario y todas las operaciones internas sobre ellos.

**Funciones principales:**

1. TryAdd(definitionId, count, **out** slotIndex, **out** reason)

2. Inserta un ítem en el inventario, apilando si es posible.

3. TrySplit(slotIndex, count, **out** newSlotIndex, **out** reason)

4. Divide una pila en dos slots distintos.

5. TryMove(srcSlots, destSlots, **out** reason)

6. Mueve o combina ítems entre slots.

7. TryClear(slotIndex, **out** reason)

8. Vacía el contenido de un slot.

9. TrySetSlotProfile(slotIndex, profileId, **out** reason)

10. Asigna un perfil de configuración a un slot.

11. TrySetCapacity(newCapacity, **out** reason) / IncreaseCapacity(delta, **out** reason)

12. Ajusta dinámicamente la capacidad **del** inventario.

**Dependencias:**

1. IStackingPolicy – controla los límites globales de apilado.

2. ISlotFilterPolicy – determina qué ítems puede aceptar cada slot.

**Salida principal:**

* *Slots* (lista de *ISlot*): vista de solo lectura con todos los slots actuales del inventario.