Inventory System Overview & Reflection

System Functionality
The inventory system is a modular Unity-based solution for managing items, equipment, and UI interactions. It combines ScriptableObjects for item data (ItemSO) with JSON persistence for cross-session continuity. Key features include:

Core Components:

InventoryManager: Central controller handling item addition/removal, slot validation, and synchronization between UI and the data model (InventoryModel).

DataManager: Serializes/deserializes inventory data (item IDs, quantities, equipment states) to JSON, ensuring seamless save/load functionality.

UIInventorySlot/UIInventoryItem: Slots validate item types (e.g., weapons in equipment slots), while items enable drag-and-drop interactions via Unity's event system.

UIDragAndDrop: Manages item movement, validating drops into slots or the game world, and updating the InventoryModel accordingly.

Workflow:

Initialization: Loads saved data to reconstruct the UI, populating slots with items from InventoryModel.

Item Interactions: Drag-and-drop swaps items between slots or drops them into the environment. Consumables trigger stat changes (e.g., health/hunger via GameplayManager).

Equipment: Enforces type restrictions (e.g., helmets in head slots) using EquipmentType enums.

Integration:

ItemsController: Provides item metadata (name, image) via ItemSO, enabling dynamic UI updates.

Gameplay Link: Consumables affect CharacterStats, while dropped items spawn in the game world as interactable objects.

Personal Reflection
This task was an engaging challenge to demonstrate practical Unity skills while aligning with the company's technical expectations. The interview process clarified mutual goals, leaving no ambiguities about the job scope.

The main challenges were improving JSON persistence (initially had issues with slot-ID mapping) and reworking a player controller after not using one for a while. However, iterative testing resolved these issues. Using Unity Store assets streamlined UI polish, emphasizing the importance of aesthetic integration alongside functionality.

I’m proud of the system’s decoupled design—separating UI logic from data management ensured scalability. The drag-and-drop mechanic, while initially complex, became robust through event-driven updates.

In summary, the system achieves:

Fluid UI interactions with equipment validation.

Reliable JSON-based persistence.

Simple gameplay integration.