# Thinking Process and Organizing

- The first step was to organize the specifications in order of architecture, which has to be implemented first to base the next system.
- Then I explored technologies and assets that would help me develop such systems.
- In the third step, with things explored and confirmed, I started the process of breaking everything down into smaller chunks and creating a to-do list with brief descriptions of what to do and what to use, to get the thinking out of the way and into mode. developer
- lastly and being the most complicated part, creating the scenery and making the look more pleasant.

# About my performance

While this was something I had never done before, I think I managed to do well, on the first day I used a lot of my time exploring, planning, and organizing things to start development right away. Thanks to the time spent in the beginning, I was able to make a quick development. Even after finishing, I already see Optimization points to be made that are already noted. What started as a test will turn into a study project after the results come in.

# Project Full Documentation

### Premade scripts used as a basis:

- CustomEventManager.cs
- GameEvent.cs
- CanvasGroupExtensions.cs (Depends on DOTween)
- PlayerMovement.cs

**Event System:** This system is based on CustomEventManager.cs and GameEvent.cs. By making classes inherit from the "GameEvent" script, communication between systems can be established without the need for direct references, thereby decoupling one system from another.

**Script "InteractionEvents.cs":** Contains various classes for communication between systems. Each class is named explicitly to avoid confusion. Some events included are:

- EnteredInteractionRangeEvent
- ExitedInteractionRangeEvent
- RequestInteractionEvent
- NotInInteractionRangeEvent
- BuyItemFromShopEvent
- PlayerSellItemEvent

- PlayerEquipItemEvent
- RequestCurrentAmountOfCoinsEvent
- CurrentAmountOfCoinsResponseEvent
- OpenShopEvent
- CloseShopEvent
- OpenInventoryEvent
- CloseInventoryEvent

Changing Clothes System: Consists of 2 scripts:

- BodyItemManager.cs: Controls player's equipped items and facilitates item exchange by providing data to "BodySectionModel" scripts.
- BodySectionModel.cs: Responsible for changing the appearance of specific body parts based on information received from "BodyItemManager".

Item System: This system comprises 2 scripts:

- ItemLibrary.cs: A Scriptable Object that aggregates items, used to create collections like player and store item inventories.
- ItemObject.cs: Base Scriptable Object for creating items used in the game.
- ItemObjectTorso.cs: An extension of ItemObject used for implementing torso-specific behavior, utilizing an additional sprite for the arm separate from the chest sprite.

Item Buttons System: This set of scripts visually represents items in the game and includes:

- ItemButtonPool.cs: Performs object pooling for UI item buttons.
- ItemButtonUI.cs: Base script controlling and displaying items in the inventory view. Determines buy ability, and triggers buy/sell actions.
- ItemButtonUIPlayer.cs: Extension of ItemButtonUI, adds item equipping functionality and enables/disables the sell button based on whether the store is open.

Inventory System: Based on two main scripts, one for the player and one for the shopkeeper:

- ShopUIController.cs
- PlayerInventoryUIController.cs Responsible for opening/closing the UI for their respective owners and populating the inventory view with items. Updates inventory based on sales/purchases.

Interaction System: Comprised of two scripts:

- InteractionHandler.cs: Contains trigger logic enabling player interaction with the store.
- InteractionEnums.cs: Stores the interaction type Enum, currently including Environment, Store, and others.

Economy System: Implemented through a single script:

- EconomyUIController.cs: Manages player's coins during transactions, sends item values from player's reserve to enable/disable purchase buttons.

Remaining scripts:

- ScenarioController.cs: Component used to enable/disable store walls when the player enters it.
- ApplicationQuit.cs: Component used in a button to exit the game.