#BluprintFunctionLibrary it's something that allows you to create a static function that then you can use across different blueprints, and if you have some sets of helper functions that you want to implement we can place them in a child class of BluprintFunctionLibrary and we can have as many as we want library (in one of the child classes).

#Unreal Engine creates a DefaultObject for every UClass that you create. It's always there and exist for everyone (for server, for client).

```
const UItemStaticData* UActionGameStatics::GetItemStaticData(TSubclassOf<
UItemStaticData> ItemDataClass)
{
	if (ItemDataClass)
	{
		return GetDefault<UItemStaticData>(ItemDataClass);
	}

	return nullptr;
}
```

(`UActionGameStatics` is a child class of `UBlueprintFunctionLibrary`)

By having this we will be always ready to get our static data object that then we can use to get our basic item parameters. Worth noticing that it's `const` and `Static`) also `BlueprintPure.`

```
UFUNCTION(BlueprintCallable, BlueprintPure)
static const UItemStaticData* GetItemStaticData(TSubclassOf< UItemStaticData>
ItemDataClass);
```

#even though we declare it as a replicated property it's replicated for people who connected right now and they can equip it. And for people who connect later to the game that property is still there unless you explicitly add some conditions calls on the beginning when they just join. So we need a UPROPERTY and we going to track when to receive it.
You'll need to define replication conditions for the inventory data. For example, you might replicate the entire inventory to a newly connected client when they join the game. Subsequently, you can replicate changes to the inventory, such as adding or removing items, only to relevant clients.

```
UPROPERTY(ReplicatedUsing = OnRep_Equipped)
bool bEquipped = false;

UFUNCTION()
void OnRep_Equipped();
```

#in inventory list we going to use the approach of fast array serialization. This approach works with two classes.
First one represents an array item and must be derived from fast array serializer item.
Second one represents fast array serializer which going to be just the array that will be serialized.

We need to add this in order for this to consider for data serialization.

```
bool NetDeltaSerialize(FNetDeltaSerializeInfo& DeltaParams)
```

#if we want to have an inventory that stay after player death we should add our inventory component (derived from actor component) as a DefaultSubobject to PlayerState or PlayerController.
But we are adding it to PlayerCharacter because we want to inventory to reset after death and lose all the items.
Also don't forget:

```
InventoryComponent->SetIsReplicated(true);
```

#We left Tick in Inventory component because of debugs purposes.