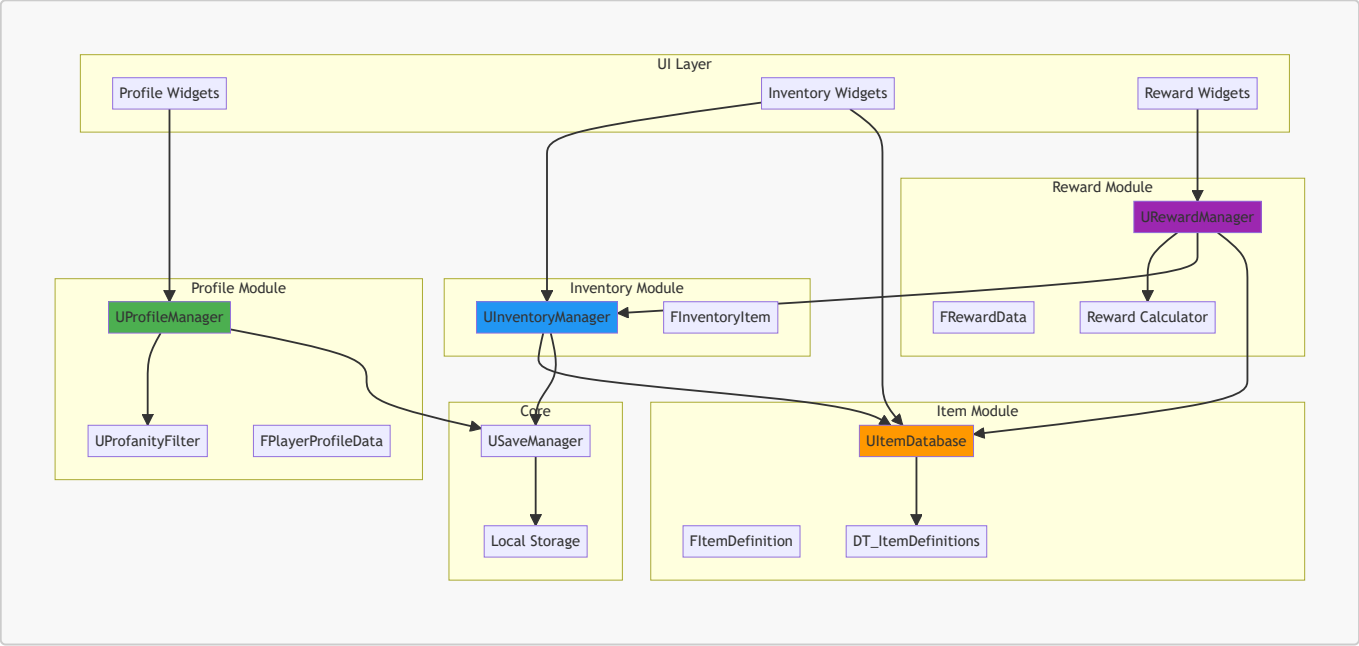
Profiles & Inventory Design

Feature ID: `profiles-inventory`
Date: 2025-12-15
Status: Design Complete
Scope: Offline MVP

Architecture Overview

Modular Architecture

Hệ thống được tách thành 4 modules độc lập để dễ nâng cấp và mở rộng:



Module Responsibilities

Module	Responsibility	Dependencies
Profile	Player info, name, avatar, stats	SaveManager, ProfanityFilter
Inventory	Item storage, quantity management	SaveManager, ItemDatabase
Item	Item definitions, properties, lookup	DataTable
Reward	Reward calculation, distribution	InventoryManager, ItemDatabase

Module 1: Profile Module

UProfileManager (C++)

Purpose: Quản lý Player Profile data độc lập

```
UCLASS()
class PROTOTYPERACING_API UProfileManager : public UObject
{
    GENERATED_BODY()

public:
    void Initialize(UProfanityFilter* Filter, USaveManager* Save);

    // Name Management
    UFUNCTION(BlueprintCallable, Category = "Profile")
    bool SetPlayerName(const FString& NewName, FString& OutError);

    UFUNCTION(BlueprintPure, Category = "Profile")
    FString GetPlayerName() const;

    // Avatar Management
    UFUNCTION(BlueprintCallable, Category = "Profile")
    void SetAvatar(const FString& AvatarID);

    UFUNCTION(BlueprintPure, Category = "Profile")
    FString GetAvatar() const;

    UFUNCTION(BlueprintPure, Category = "Profile")
    TArray<FAvatarInfo> GetAvailableAvatars() const;

    // Stats Management
    UFUNCTION(BlueprintCallable, Category = "Profile")
    void UpdateRaceStats(const FRaceResultData& RaceResult);

    UFUNCTION(BlueprintCallable, Category = "Profile")
    void AddOnlineTime(float DeltaSeconds);

    // Data Access
    UFUNCTION(BlueprintPure, Category = "Profile")
    FPlayerProfileData GetProfileData() const;

    // Events
    UPROPERTY(BlueprintAssignable, Category = "Profile")
    FOnProfileUpdated OnProfileUpdated;

private:
    UPROPERTY()
    FPlayerProfileData ProfileData;

    UPROPERTY()
    UProfanityFilter* ProfanityFilter;

    UPROPERTY()
    USaveManager* SaveManager;

    bool ValidateName(const FString& Name, FString& OutError);
    void GeneratePlayerID();
};
```

UProfanityFilter (C++)

Purpose: Lọc từ ngữ thô tục - module độc lập có thể reuse

```
UCLASS()
class PROTOTYPERACING_API UProfanityFilter : public UObject
{
    GENERATED_BODY()

public:
    void Initialize();

    UFUNCTION(BlueprintCallable, Category = "Filter")
    bool ContainsProfanity(const FString& Text) const;

    UFUNCTION(BlueprintCallable, Category = "Filter")
    FString FilterText(const FString& Text, const FString& Replacement =
TEXT("*")) const;

    UFUNCTION(BlueprintCallable, Category = "Filter")
    void AddCustomBadWord(const FString& Word);

private:
    TSet<FString> VietnameseBadWords;
    TSet<FString> EnglishBadWords;
    TSet<FString> CustomBadWords;
    TMap<TCHAR, TArray<TCHAR>> LeetSpeakMap;

    void LoadBadWordLists();
    FString NormalizeText(const FString& Text) const;
};
```

FPlayerProfileData

```
USTRUCT(BlueprintType)
struct FPlayerProfileData
{
    GENERATED_BODY()

    // Identity
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    FString PlayerID;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    FString PlayerName = TEXT("Racer");

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    FString AvatarID = TEXT("avatar_default");
```

```
    // Race Stats
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    float OnlineTimeSeconds = 0.0f;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    float TopSpeed = 0.0f;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    float TotalRaceTimeSeconds = 0.0f;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    int32 TotalRaces = 0;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    int32 TotalWins = 0;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    int32 FirstPlaceCount = 0;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    int32 SecondPlaceCount = 0;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    int32 ThirdPlaceCount = 0;

    // Unlock Info
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    int32 CarsUnlocked = 0;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    int32 TracksUnlocked = 0;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    int32 CitiesUnlocked = 0;

    // Economy
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    int64 TotalEarned = 0;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    int64 TotalSpent = 0;

    // Timestamps
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    FDateTime CreatedAt;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    FDateTime LastModified;

    // Helpers
    float GetWinRate() const { return TotalRaces > 0 ? (float)TotalWins /
TotalRaces * 100.0f : 0.0f; }
```

```
    float GetAveragePosition() const;
};
```

Module 2: Inventory Module

UInventoryManager (C++)

Purpose: Quản lý item storage độc lập

```
UCLASS()
class PROTOTYPERACING_API UInventoryManager : public UObject
{
    GENERATED_BODY()

public:
    void Initialize(UItemDatabase* ItemDB, USaveManager* Save);
    void InitializeDefaultItems();

    // Item Access
    UFUNCTION(BlueprintPure, Category = "Inventory")
    TArray<FInventoryItem> GetAllItems() const;

    UFUNCTION(BlueprintPure, Category = "Inventory")
    TArray<FInventoryItem> GetItemsByType(EItemType Type) const;

    UFUNCTION(BlueprintPure, Category = "Inventory")
    FInventoryItem GetItem(const FString& ItemID) const;

    UFUNCTION(BlueprintPure, Category = "Inventory")
    bool HasItem(const FString& ItemID, int32 MinQuantity = 1) const;

    UFUNCTION(BlueprintPure, Category = "Inventory")
    int32 GetItemCount(const FString& ItemID) const;

    // Item Management
    UFUNCTION(BlueprintCallable, Category = "Inventory")
    bool AddItem(const FString& ItemID, int32 Quantity = 1, const FString& Source
= TEXT(""));

    UFUNCTION(BlueprintCallable, Category = "Inventory")
    bool RemoveItem(const FString& ItemID, int32 Quantity = 1);

    UFUNCTION(BlueprintCallable, Category = "Inventory")
    void SetItemEquipped(const FString& ItemID, bool bEquipped);

    UFUNCTION(BlueprintCallable, Category = "Inventory")
    void SetItemFavorite(const FString& ItemID, bool bFavorite);

    // Bulk Operations
    UFUNCTION(BlueprintCallable, Category = "Inventory")
```

```
    void AddItems(const TArray<FItemQuantity>& Items, const FString& Source =
TEXT(""));

    // Events
    UPROPERTY(BlueprintAssignable, Category = "Inventory")
    FOnInventoryUpdated OnInventoryUpdated;

    UPROPERTY(BlueprintAssignable, Category = "Inventory")
    FOnItemAdded OnItemAdded;

    UPROPERTY(BlueprintAssignable, Category = "Inventory")
    FOnItemRemoved OnItemRemoved;

private:
    UPROPERTY()
    TMap<FString, FInventoryItem> Items;

    UPROPERTY()
    UItemDatabase* ItemDatabase;

    UPROPERTY()
    USaveManager* SaveManager;

    static const int32 MAX_ITEMS = 999;
    static const int32 MAX_UNIQUE_ITEMS = 200;

    bool CanAddItem(const FString& ItemID, int32 Quantity) const;
};
```

FInventoryItem

```
USTRUCT(BlueprintType)
struct FInventoryItem
{
    GENERATED_BODY()

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    FString ItemID;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    int32 Quantity = 1;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    FDateTime AcquiredDate;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    FString AcquisitionSource = TEXT("default");

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    bool bIsEquipped = false;
```

```
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    bool bIsFavorite = false;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    int32 UsageCount = 0;
};
```

Module 3: Item Module

UItemDatabase (C++)

Purpose: Item definitions và lookup - read-only database

```
UCLASS()
class PROTOTYPERACING_API UItemDatabase : public UObject
{
    GENERATED_BODY()

public:
    void Initialize();

    // Item Lookup
    UFUNCTION(BlueprintPure, Category = "Items")
    FItemDefinition GetItemDefinition(const FString& ItemID) const;

    UFUNCTION(BlueprintPure, Category = "Items")
    bool ItemExists(const FString& ItemID) const;

    UFUNCTION(BlueprintPure, Category = "Items")
    bool IsStackable(const FString& ItemID) const;

    UFUNCTION(BlueprintPure, Category = "Items")
    EItemType GetItemType(const FString& ItemID) const;

    UFUNCTION(BlueprintPure, Category = "Items")
    EItemRarity GetItemRarity(const FString& ItemID) const;

    // Queries
    UFUNCTION(BlueprintPure, Category = "Items")
    TArray<FItemDefinition> GetAllItems() const;

    UFUNCTION(BlueprintPure, Category = "Items")
    TArray<FItemDefinition> GetItemsByType(EItemType Type) const;

    UFUNCTION(BlueprintPure, Category = "Items")
    TArray<FItemDefinition> GetItemsByRarity(EItemRarity Rarity) const;

    UFUNCTION(BlueprintPure, Category = "Items")
    TArray<FItemDefinition> GetItemsForCar(const FString& CarGroup) const;
```

```
private:
    UPROPERTY()
    UDataTable* ItemDefinitionsTable;

    TMap<FString, FItemDefinition> ItemCache;

    void BuildCache();
};
```

FItemDefinition (Data Table Row)

```
USTRUCT(BlueprintType)
struct FItemDefinition : public FTableRowBase
{
    GENERATED_BODY()

    // Basic Info
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Basic")
    FString ItemID;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Basic")
    FText DisplayName;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Basic")
    FText Description;

    // Classification
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Classification")
    EItemType ItemType = EItemType::Other;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Classification")
    EItemRarity Rarity = EItemRarity::Common;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Classification")
    bool bIsStackable = true;

    // Visual
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Visual")
    TSoftObjectPtr<UTexture2D> Icon;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Visual")
    FLinearColor RarityColor;

    // For CCV (Visual Items)
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Visual Item")
    FString CarGroup; // Vios, Supra, M3

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Visual Item")
    FString PartType; // Bumper, Window, Light, etc

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Visual Item")
```

```
    TSoftObjectPtr<UStaticMesh> MeshAsset;

    // For CCP (Performance Items)
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Performance Item")
    int32 UpgradeLevel = 0; // 4, 5, 6

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Performance Item")
    FPerformanceModifiers StatModifications;

    // For LC (Loot Crates)
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Loot Crate")
    TArray<FLootTableEntry> LootTable;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Loot Crate")
    float RewardMultiplier = 1.0f;

    // Economy
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Economy")
    int32 PurchasePrice = 0;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Economy")
    int32 SellPrice = 0;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Economy")
    float DropRate = 0.0f;
};
```

Enums

```
UENUM(BlueprintType)
enum class EItemType : uint8
{
    CCV     UMETA(DisplayName = "Car Customize Visual"),
    CCP     UMETA(DisplayName = "Car Customize Performance"),
    LC      UMETA(DisplayName = "Loot Crate"),
    Ticket  UMETA(DisplayName = "Ticket"),
    Currency UMETA(DisplayName = "Currency"),
    Other   UMETA(DisplayName = "Other")
};

UENUM(BlueprintType)
enum class EItemRarity : uint8
{
    Common      UMETA(DisplayName = "Common"),
    Uncommon    UMETA(DisplayName = "Uncommon"),
    Rare        UMETA(DisplayName = "Rare")
};
```

Module 4: Reward Module

URewardManager (C++)

Purpose: Tính toán và phân phối rewards

```
UCLASS()
class PROTOTYPERACING_API URewardManager : public UObject
{
    GENERATED_BODY()

public:
    void Initialize(UInventoryManager* InvMgr, UItemDatabase* ItemDB);

    // Race Rewards
    UFUNCTION(BlueprintCallable, Category = "Rewards")
    FRaceRewardResult ProcessRaceRewards(const FRaceResultData& RaceResult);

    // Loot Crate
    UFUNCTION(BlueprintCallable, Category = "Rewards")
    FLootCrateResult OpenLootCrate(const FString& CrateItemID);

    // Daily Rewards
    UFUNCTION(BlueprintCallable, Category = "Rewards")
    FDailyRewardResult ClaimDailyReward(int32 DayIndex);

    // Generic Reward
    UFUNCTION(BlueprintCallable, Category = "Rewards")
    void GrantRewards(const TArray<FRewardEntry>& Rewards, const FString& Source);

    // Events
    UPROPERTY(BlueprintAssignable, Category = "Rewards")
    FOnRewardsGranted OnRewardsGranted;

private:
    UPROPERTY()
    UInventoryManager* InventoryManager;

    UPROPERTY()
    UItemDatabase* ItemDatabase;

    TArray<FRewardEntry> CalculateRaceRewards(int32 Position, int32 TotalRacers,
float RaceTime);
    TArray<FRewardEntry> RollLootTable(const TArray<FLootTableEntry>& LootTable);
};
```

Reward Data Structures

```
USTRUCT(BlueprintType)
struct FRewardEntry
{
    GENERATED_BODY()
```

```
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    FString ItemID;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    int32 Quantity = 1;
};

USTRUCT(BlueprintType)
struct FRaceRewardResult
{
    GENERATED_BODY()

    UPROPERTY(BlueprintReadWrite)
    TArray<FRewardEntry> Items;

    UPROPERTY(BlueprintReadWrite)
    int32 CurrencyEarned = 0;

    UPROPERTY(BlueprintReadWrite)
    int32 XPEarned = 0;

    UPROPERTY(BlueprintReadWrite)
    bool bSuccess = false;
};

USTRUCT(BlueprintType)
struct FLootCrateResult
{
    GENERATED_BODY()

    UPROPERTY(BlueprintReadWrite)
    TArray<FRewardEntry> Items;

    UPROPERTY(BlueprintReadWrite)
    bool bSuccess = false;

    UPROPERTY(BlueprintReadWrite)
    FString ErrorMessage;
};

USTRUCT(BlueprintType)
struct FLootTableEntry
{
    GENERATED_BODY()

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    FString ItemID;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    float DropChance = 0.0f; // 0.0 - 1.0

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    int32 MinQuantity = 1;
```

```
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    int32 MaxQuantity = 1;
};
```

Core: Save Manager

USaveManager (C++)

Purpose: Centralized save/load cho tất cả modules

```
UCLASS()
class PROTOTYPERACING_API USaveManager : public UObject
{
    GENERATED_BODY()

public:
    void Initialize();

    // Profile Data
    UFUNCTION(BlueprintCallable, Category = "Save")
    void SaveProfileData(const FPlayerProfileData& Data);

    UFUNCTION(BlueprintCallable, Category = "Save")
    FPlayerProfileData LoadProfileData();

    // Inventory Data
    UFUNCTION(BlueprintCallable, Category = "Save")
    void SaveInventoryData(const TArray<FInventoryItem>& Items);

    UFUNCTION(BlueprintCallable, Category = "Save")
    TArray<FInventoryItem> LoadInventoryData();

    // Full Save/Load
    UFUNCTION(BlueprintCallable, Category = "Save")
    void SaveAll();

    UFUNCTION(BlueprintCallable, Category = "Save")
    void LoadAll();

    // Utility
    UFUNCTION(BlueprintPure, Category = "Save")
    bool HasSaveData() const;

    UFUNCTION(BlueprintCallable, Category = "Save")
    void DeleteSaveData();

private:
    static const FString SAVE_SLOT_NAME;
    static const int32 SAVE_VERSION;
```

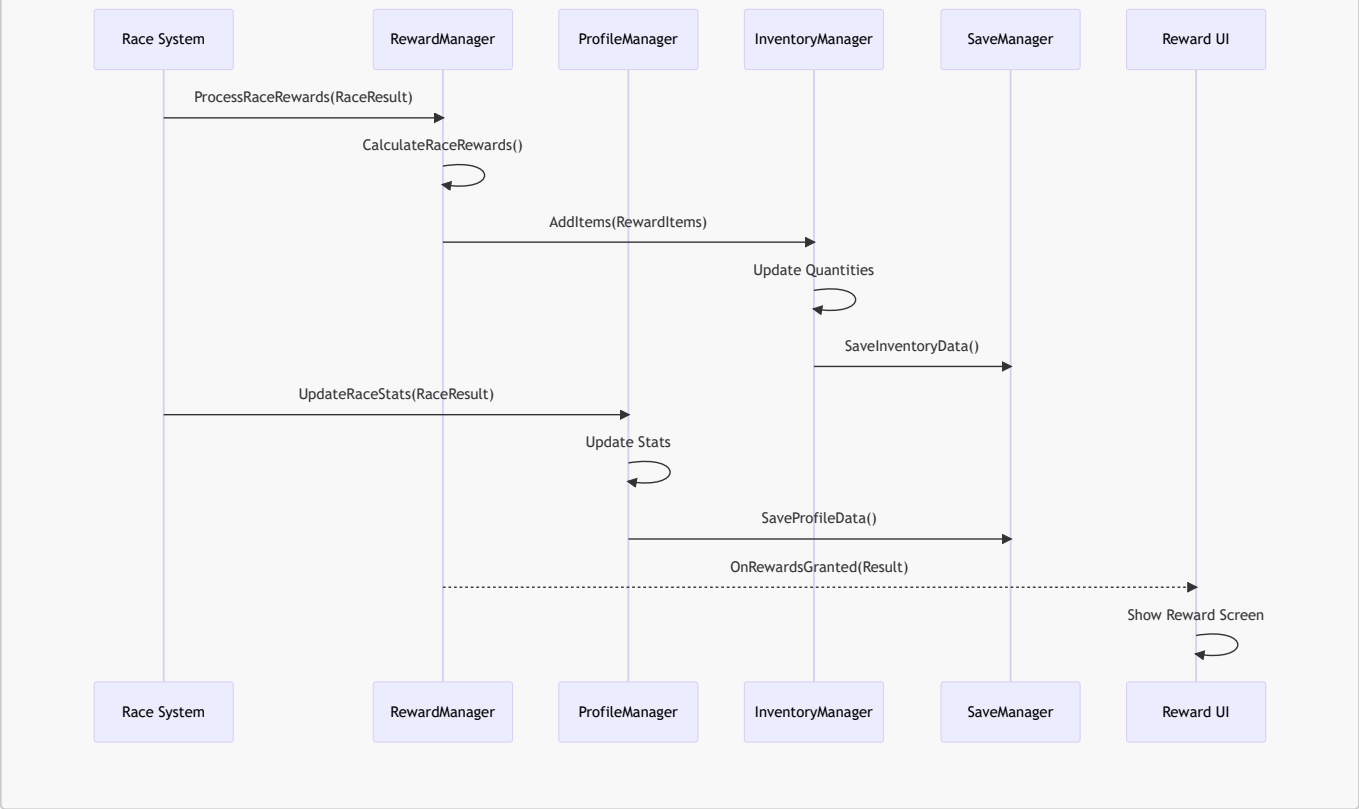
```
    UPROPERTY()
    UProfileInventorySaveGame* CurrentSaveGame;

    void CreateNewSaveGame();
    bool ValidateSaveData();
    void MigrateSaveData(int32 FromVersion);
};
```

USaveGame Class

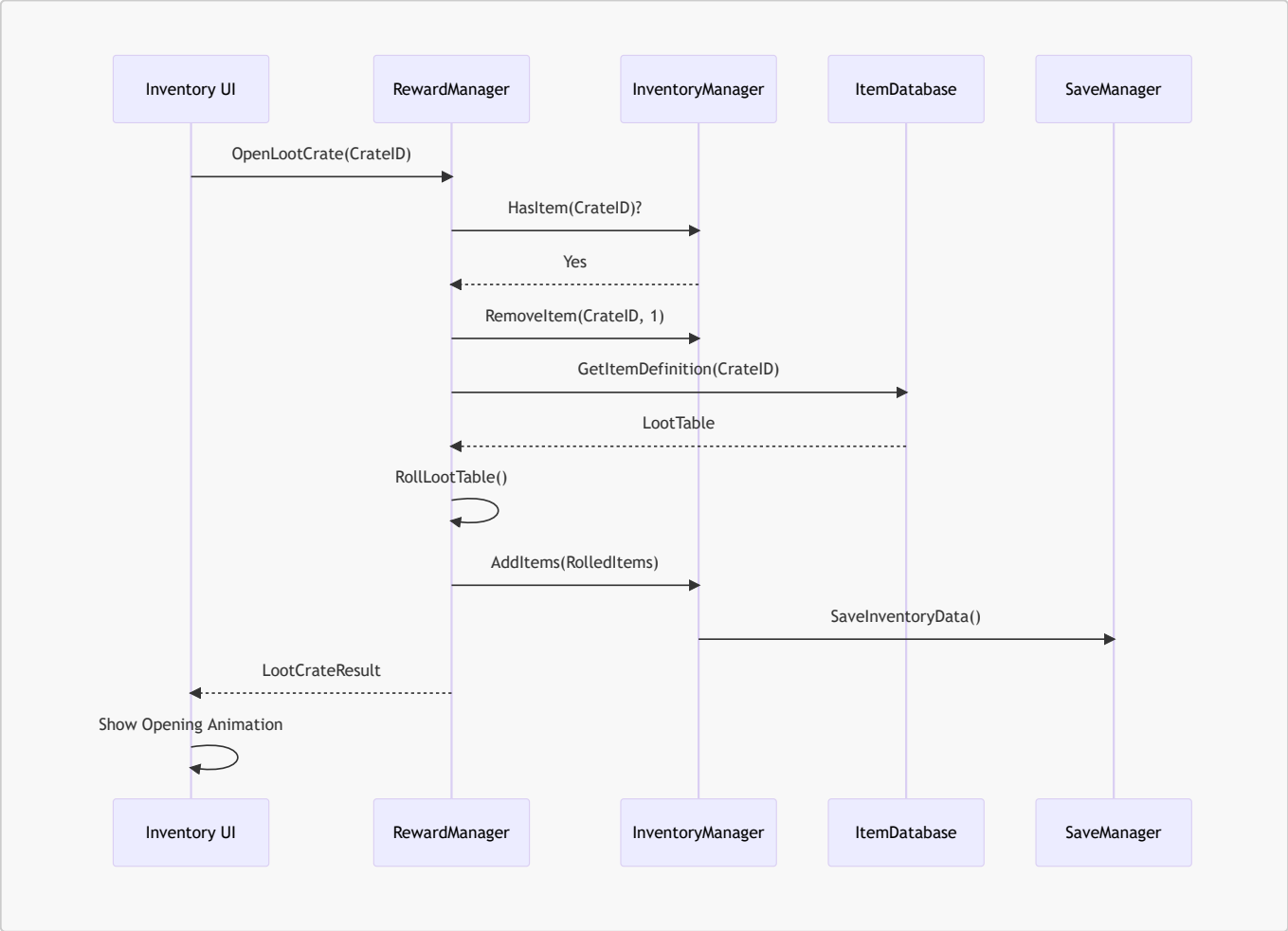
```
UCLASS()
class PROTOTYPERACING_API UProfileInventorySaveGame : public USaveGame
{
    GENERATED_BODY()

public:
    UPROPERTY()
    int32 SaveVersion = 1;

    UPROPERTY()
    FPlayerProfileData ProfileData;

    UPROPERTY()
    TArray<FInventoryItem> InventoryItems;

    // For future Nakama sync
    UPROPERTY()
    FDateTime LastSyncTime;

    UPROPERTY()
    bool bPendingSync = false;

    UPROPERTY()
    FString SyncChecksum;
};
```

Module Interaction Flow

Race Complete Flow



Loot Crate Flow



Profanity Filter Implementation

Bad Words Lists

```
void UProfanityFilter::LoadBadWordLists()
{
    // Vietnamese
    VietnameseBadWords = {
        TEXT("đụ"), TEXT("địt"), TEXT("lồn"), TEXT("cặc"), TEXT("buồi"),
        TEXT("đéo"), TEXT("đĩ"), TEXT("cave"), TEXT("chó"), TEXT("ngu"),
        TEXT("khốn"), TEXT("đần"), TEXT("nát"), TEXT("thằng chó"),
        TEXT("con đĩ"), TEXT("đồ ngu"), TEXT("mẹ mày"), TEXT("đm"),
        TEXT("vcl"), TEXT("vl"), TEXT("cc"), TEXT("clgt"), TEXT("đkm")
    };

    // English
    EnglishBadWords = {
        TEXT("fuck"), TEXT("shit"), TEXT("ass"), TEXT("bitch"),
        TEXT("damn"), TEXT("crap"), TEXT("dick"), TEXT("pussy"),
        TEXT("bastard"), TEXT("whore"), TEXT("slut"), TEXT("nigger")
    };

    // Leetspeak mappings
    LeetSpeakMap = {
        {'a', {'4', '@'}},
        {'e', {'3'}},
        {'i', {'1', '!'}},
        {'o', {'0'}},
        {'s', {'5', '$'}},
        {'t', {'7'}}
    };
}
```

Default Items Configuration (MVP)

```
void UInventoryManager::InitializeDefaultItems()
{
    // Car Customize Visual (CCV) - All unlocked, quantity 1 (non-stackable)
    // 54 items (18 per car × 3 cars)
    for (const FItemDefinition& Item : ItemDatabase-
>GetItemsByType(EItemType::CCV))
    {
        AddItem(Item.ItemID, 1, TEXT("default"));
    }

    // Car Customize Performance (CCP) - All unlocked, quantity 999
    for (const FItemDefinition& Item : ItemDatabase-
>GetItemsByType(EItemType::CCP))
    {
        AddItem(Item.ItemID, 999, TEXT("default"));
    }
```

```
    // Loot Crates (LC) - Default quantity 999
    for (const FItemDefinition& Item : ItemDatabase-
>GetItemsByType(EItemType::LC))
    {
        AddItem(Item.ItemID, 999, TEXT("default"));
    }
}
```

Future Nakama Compatibility

Tất cả data structures được thiết kế để compatible với Nakama:

1. **PlayerID**: Generated locally, map với Nakama User ID
2. **Timestamps**: CreatedAt, LastModified cho sync tracking
3. **SaveVersion**: Handle migration khi update
4. **bPendingSync**: Flag cho offline changes cần sync
5. **SyncChecksum**: Verify data integrity

References

- Requirements
- Planning
- UserProfile_Inventory_V5.md
- Items_V5.md