

# *Repository in Github*

---

<https://github.com/roboticsjoe2020/OpenAIAPI>

`git clone` <https://github.com/roboticsjoe2020/OpenAIAPI> *yourprojectfolder*

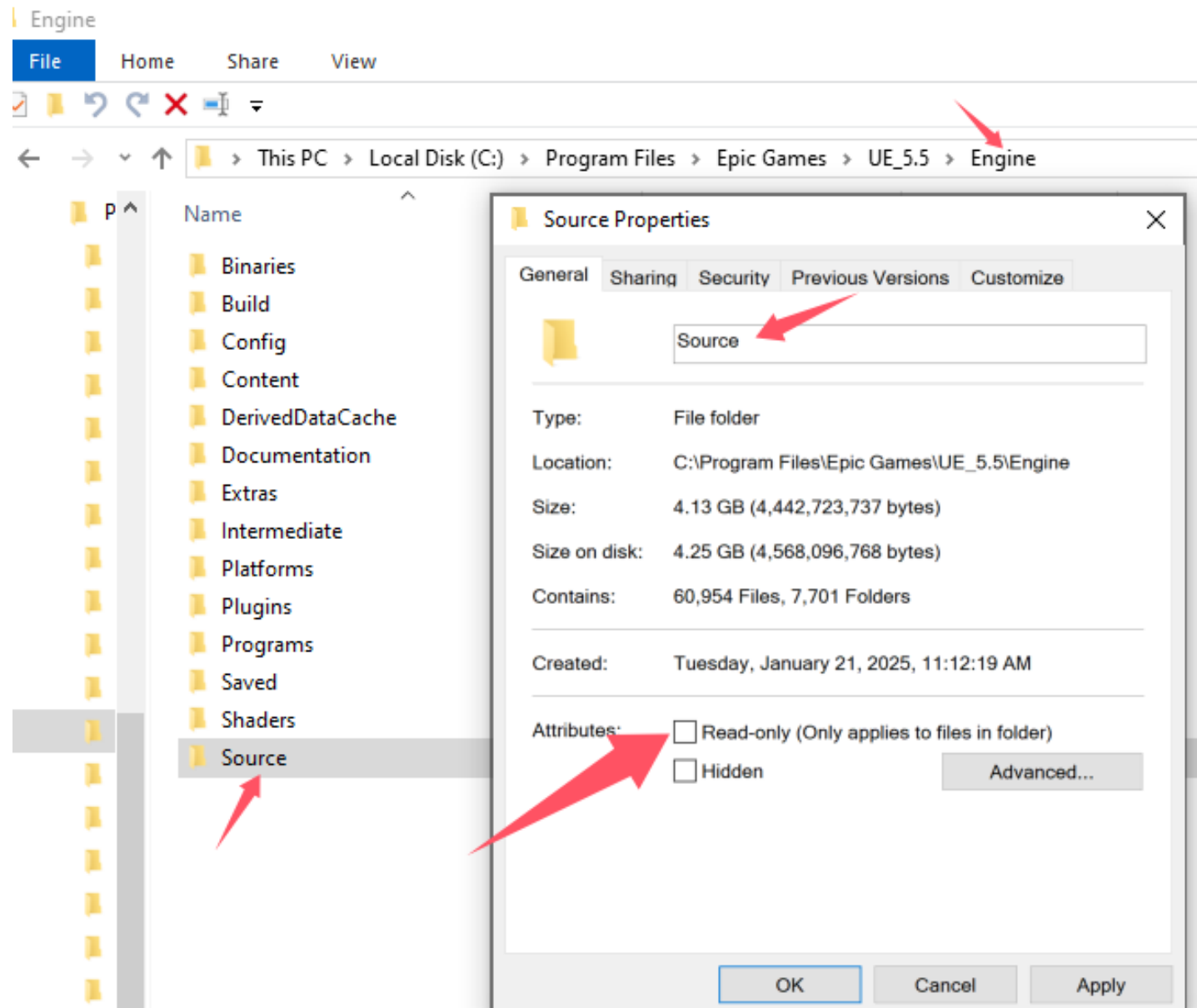
*Or directly download*

*Endpoint:* <https://api.openai.com/v1/completions>

***API Key:***

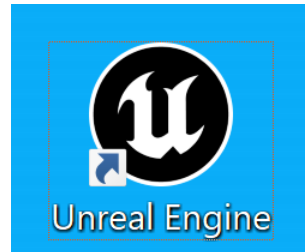
# Practical Procedures (1)

Make Sure 'Source' Folder R/W

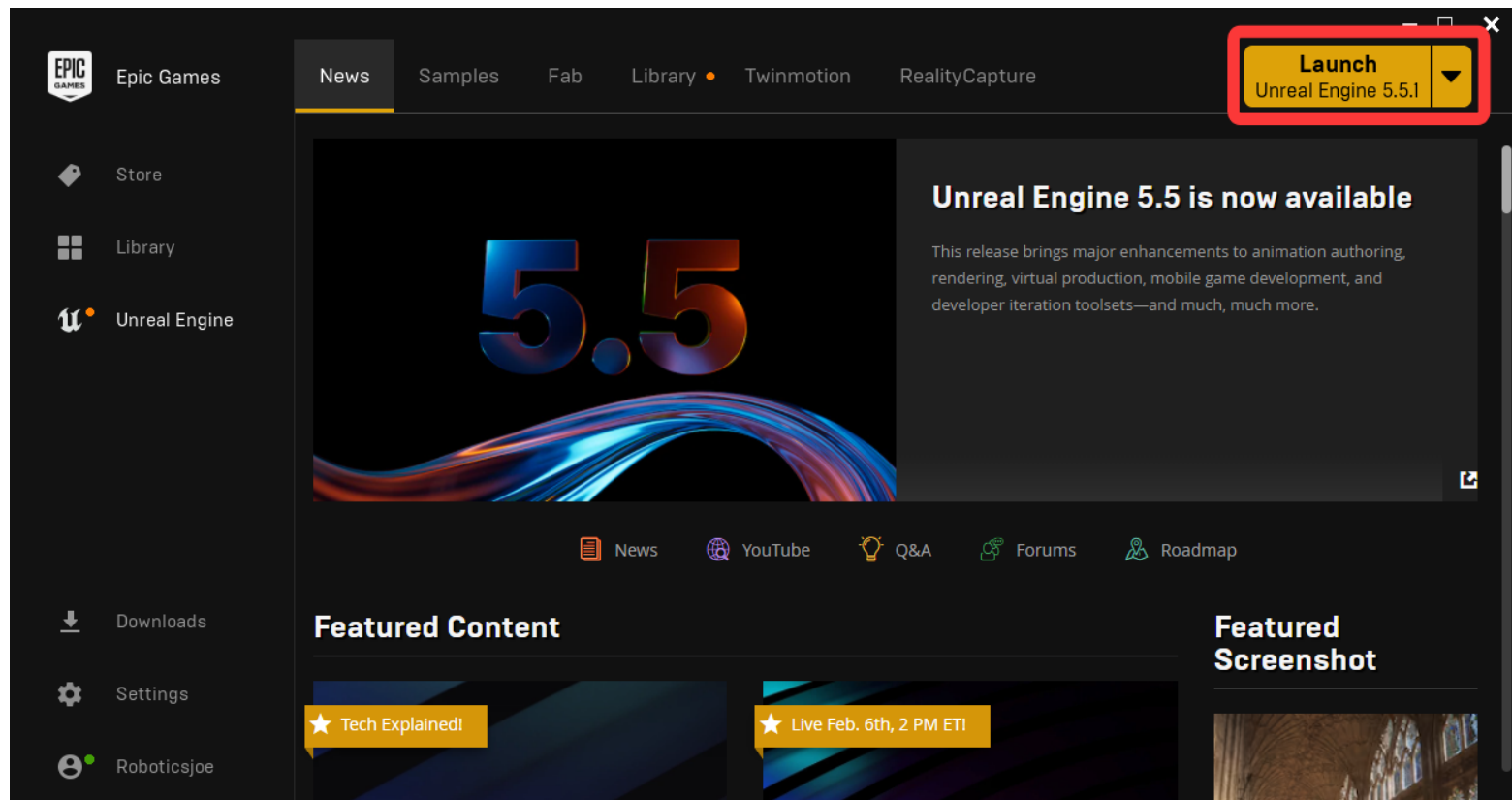


# Practical Procedures (2)

Directly Launch Unreal Engine



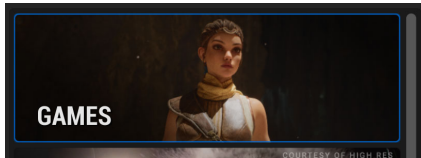
Or Through Epic Game Launcher



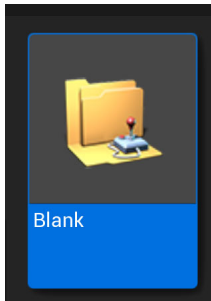
# Practical Procedures (3)

## Create a New C++ Project:

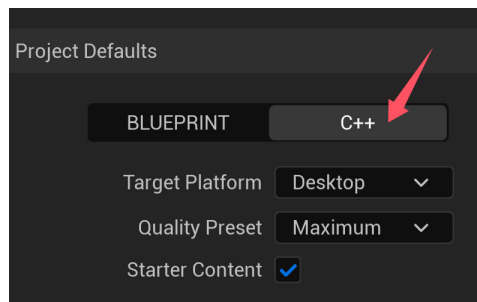
→ Click 'GAMES'



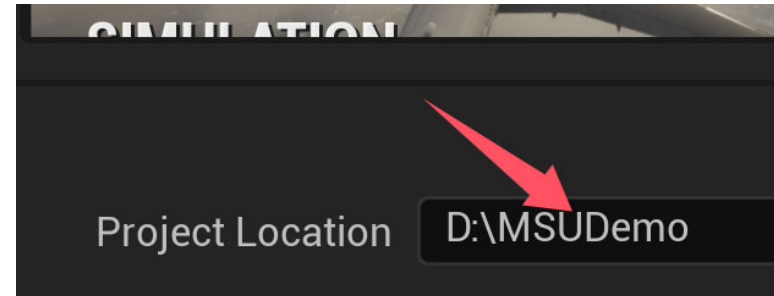
→ Choose 'Blank'



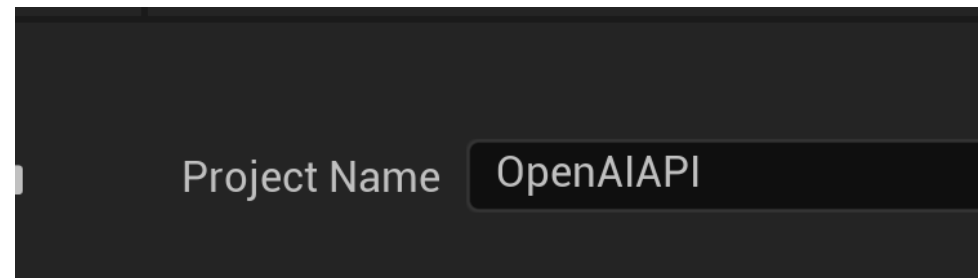
→ Choose C++ as the template



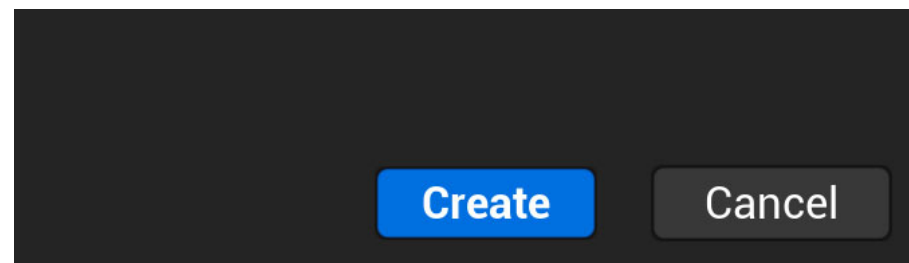
→ Project Location (Your Local Drive)



→ Name your project (e.g., OpenAI API).

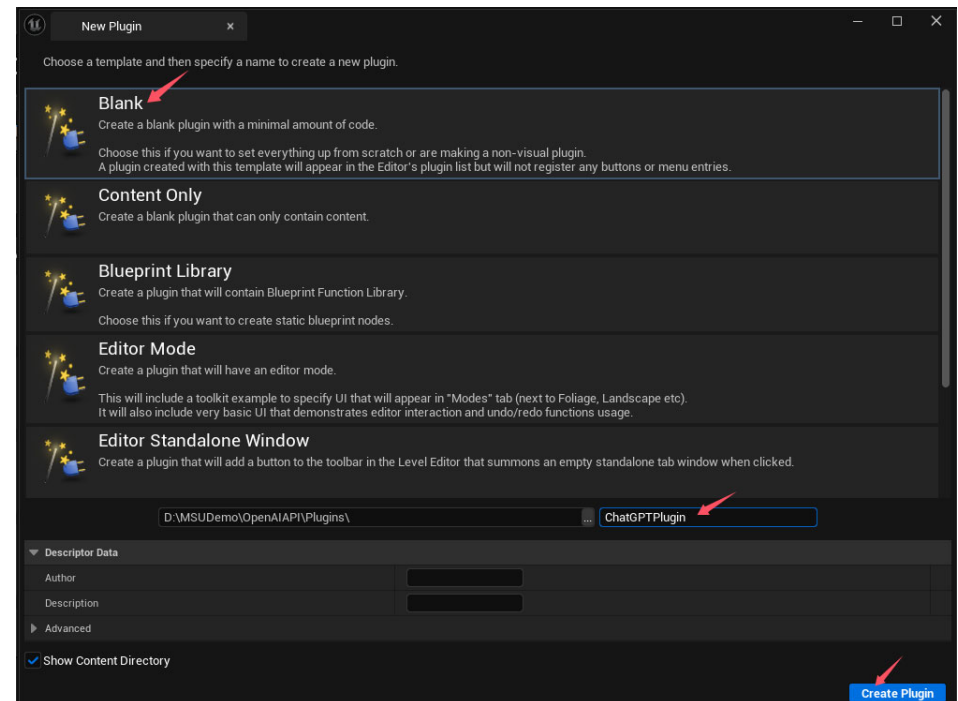
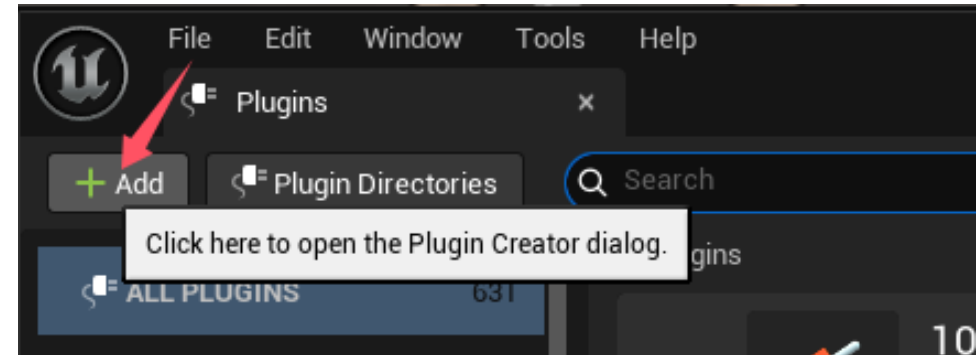
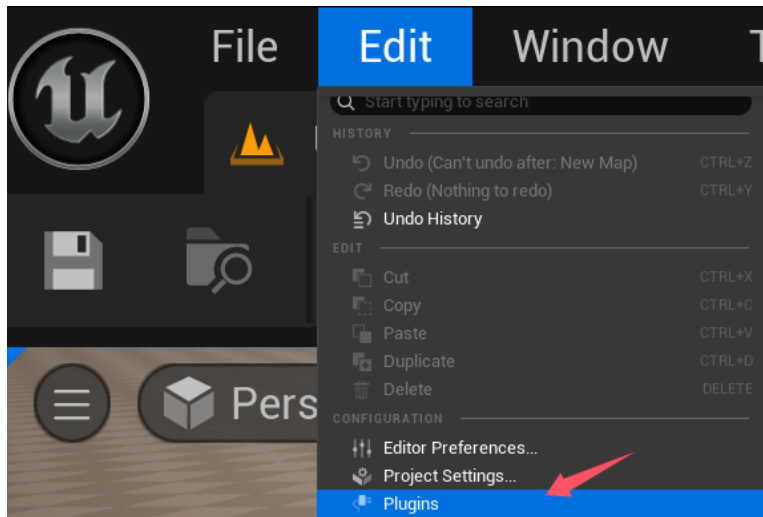


→ Click 'Create'.



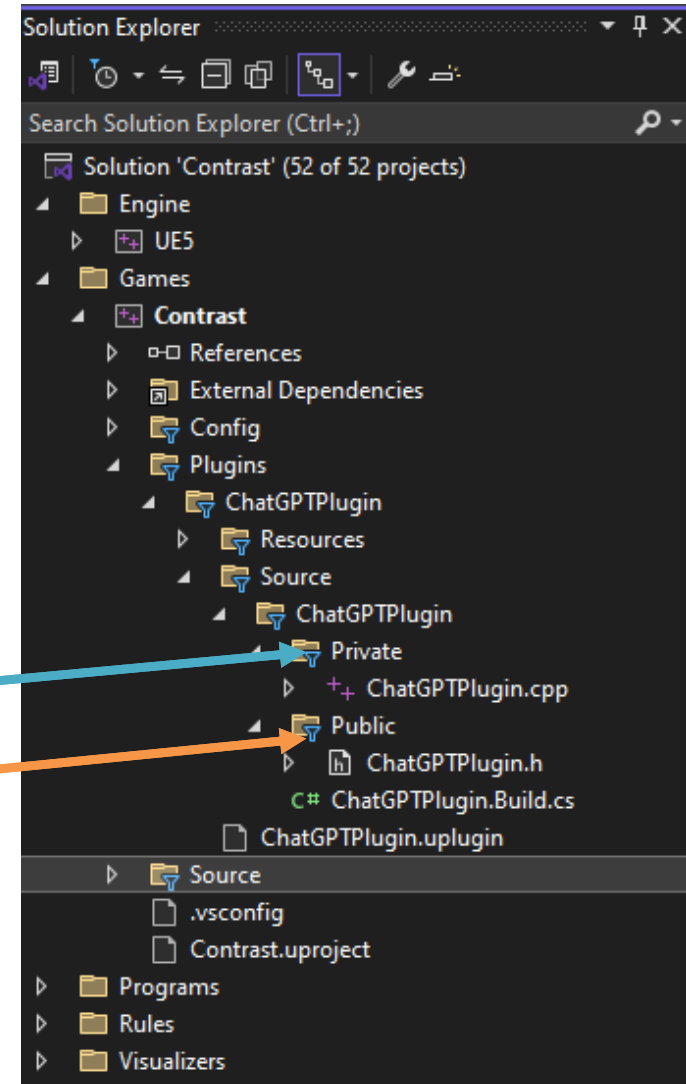
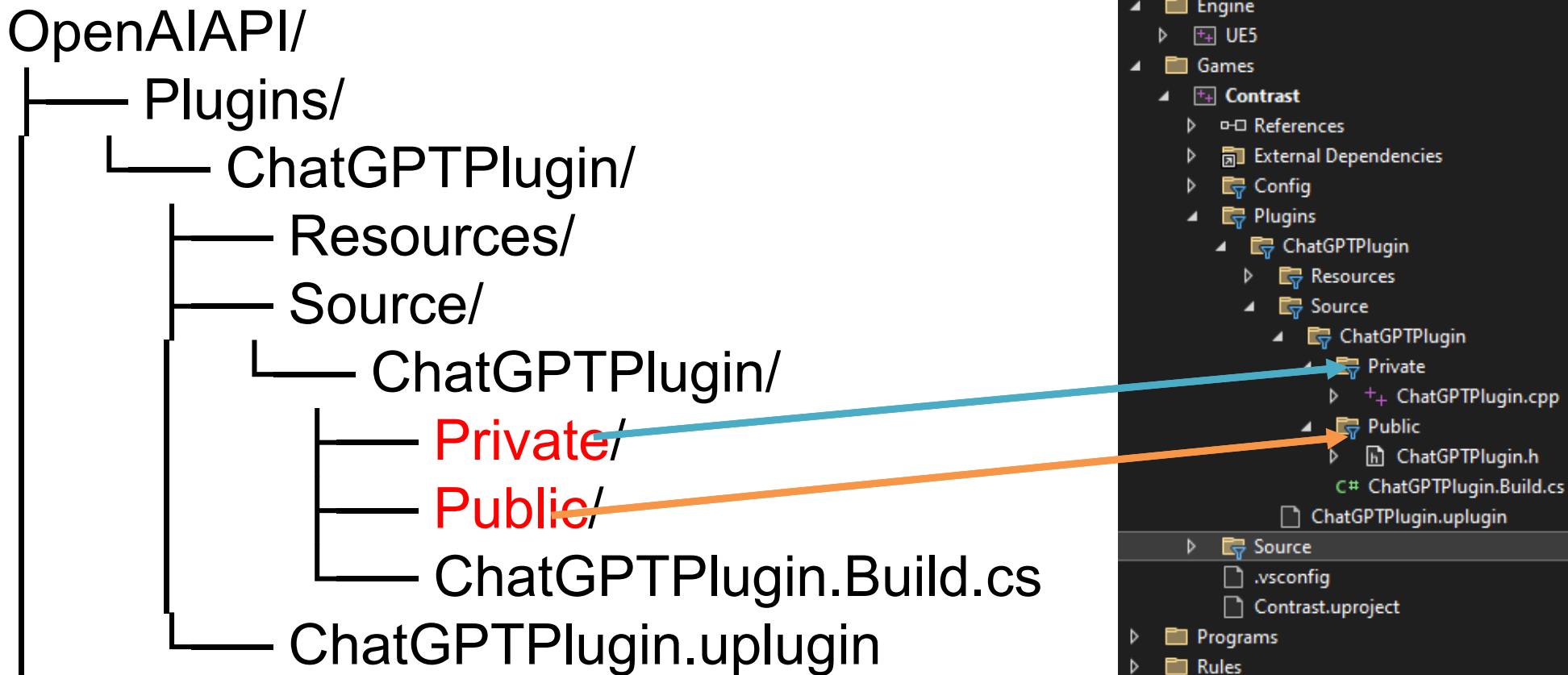
# Practical Procedures (4)

## Create a New C++ plugin in Unreal editor



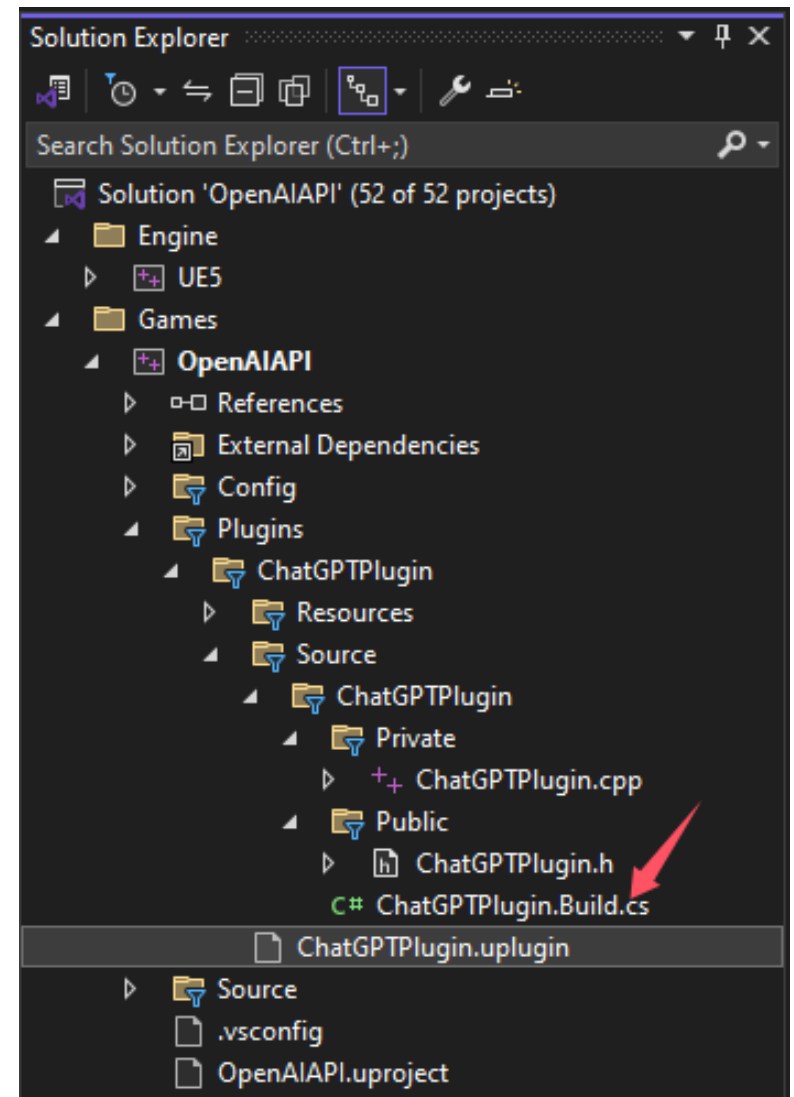
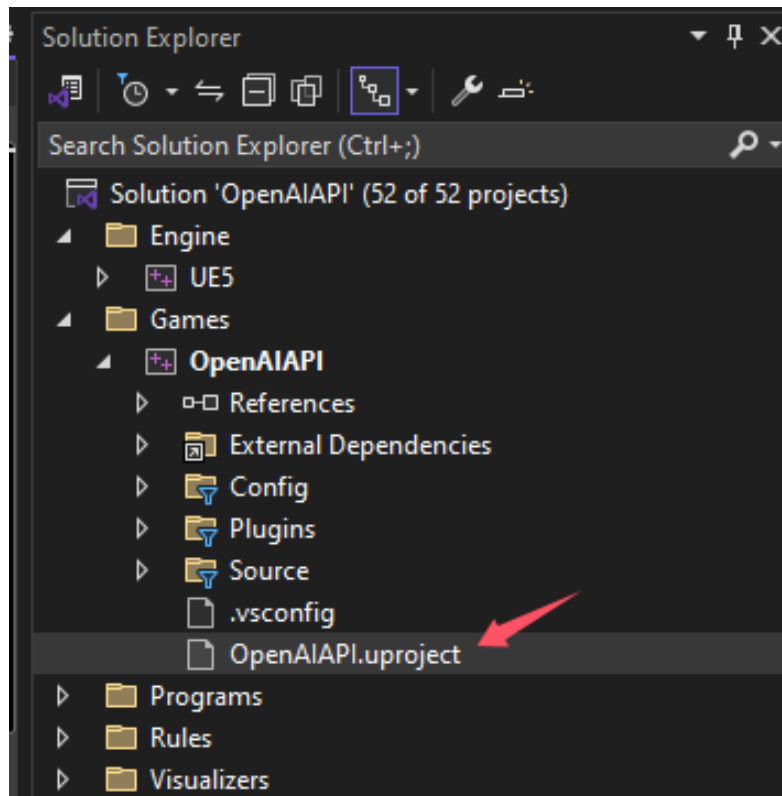
## ***Practical Procedures (5)***

## After creating the plugin, verify the folder structure:



# Practical Procedures (6)

Go to Visual Studio and open project and define the project and plugin metadata

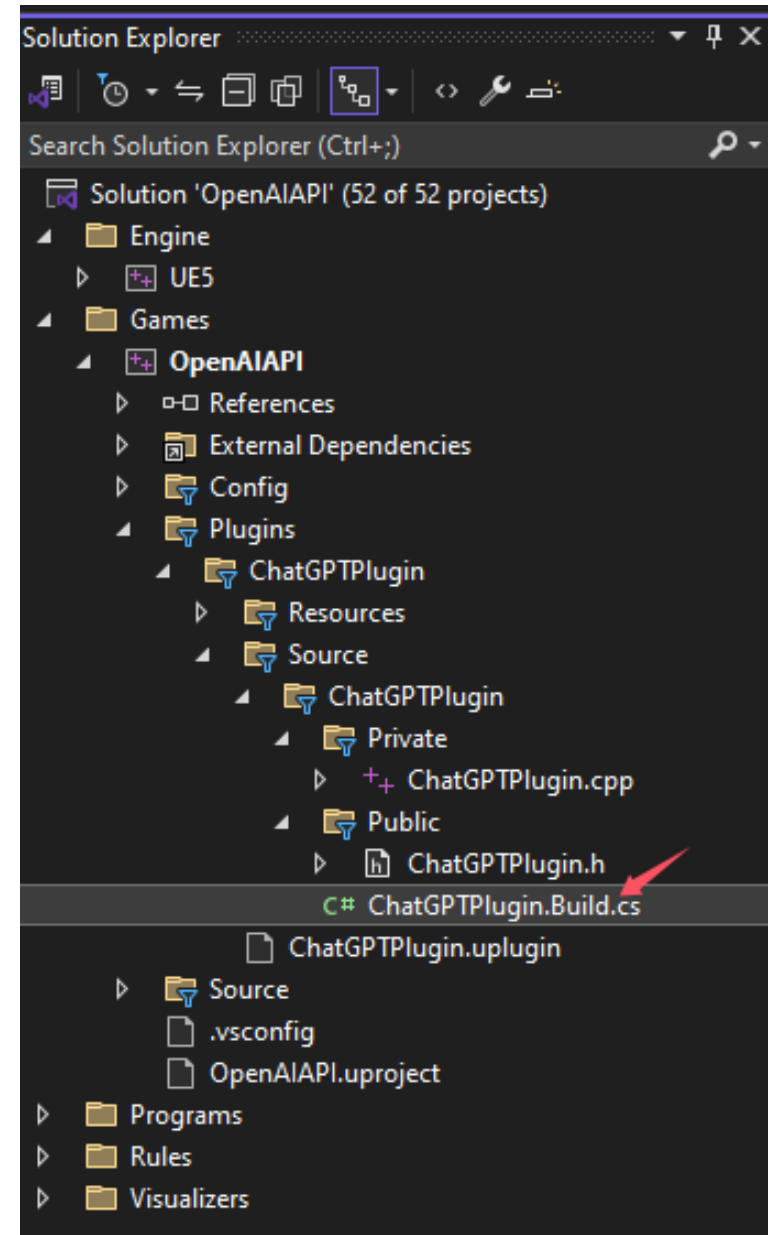


Use default setup for both

# Practical Procedures (7)

Add required dependencies (Crucial Step) in **ChatGPTPlugin.Build.cs**

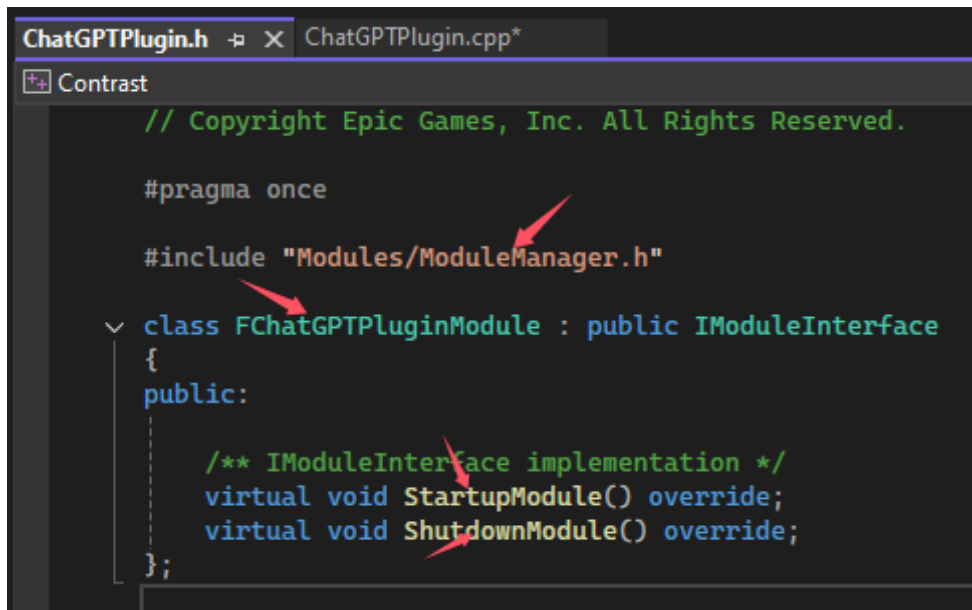
```
PrivateIncludePaths.AddRange(  
    new string[] {  
        // ... add other private include paths re  
    }  
);  
  
PublicDependencyModuleNames.AddRange(  
    new string[]  
    {  
        "Core",  
        // ... add other public dependencies that  
    }  
);  
  
PrivateDependencyModuleNames.AddRange(  
    new string[]  
    {  
        "CoreUObject",  
        "Engine",  
        "Slate",  
        "SlateCore",  
        // ... add private dependencies that you  
        "HTTP",           // Add HTTP module  
        "Json",           // Add Json module  
        "JsonUtilities"  
    }  
);
```





# Practical Procedures (8)

## How Unreal Engine Manages Modules: Header?



```
ChatGPTPlugin.h  X ChatGPTPlugin.cpp*
+ Contrast
// Copyright Epic Games, Inc. All Rights Reserved.

#pragma once

#include "Modules/ModuleManager.h"

class FChatGPTPluginModule : public IModuleInterface
{
public:

    /** IModuleInterface implementation */
    virtual void StartupModule() override;
    virtual void ShutdownModule() override;
};
```

□ Unreal Engine, modules (like FChatGPTPluginModule) managed by the module system (**IModuleInterface**). Engine starts → automatically loads and initializes all registered modules.

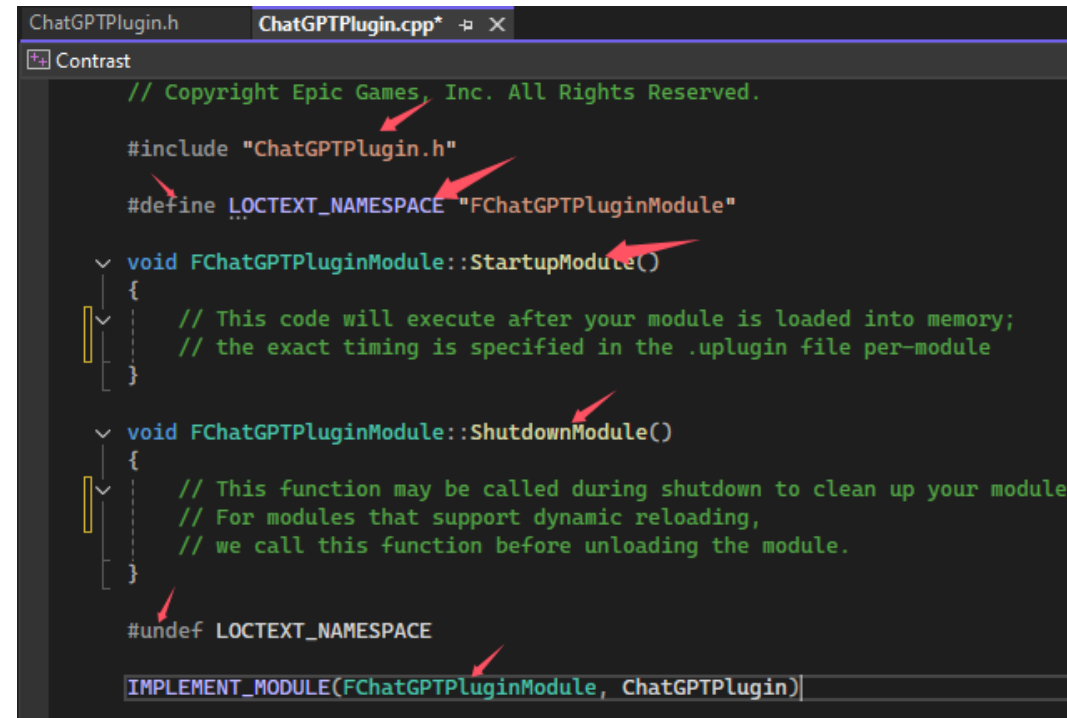
- The declaration (ChatGPTPlugin.h) is part of the module header.
- The implementation (ChatGPTPlugin.cpp) is instantiated at runtime by Unreal Engine.
- StartupModddule() and ShutdownModule declared here.

### □ Keypoints

- '**FChatGPTPluginModule**' declared normally as a class.
- No instance created here.
- Unreal Engine will instantiate this module at runtime and manage its lifecycle.

# Practical Procedures (9)

## How Unreal Engine Manages Modules: Implementation file?



```
// Copyright Epic Games, Inc. All Rights Reserved.

#include "ChatGPTPlugin.h"

#define LOCTEXT_NAMESPACE "FChatGPTPluginModule"

void FChatGPTPluginModule::StartupModule()
{
    // This code will execute after your module is loaded into memory;
    // the exact timing is specified in the .uplugin file per-module
}

void FChatGPTPluginModule::ShutdownModule()
{
    // This function may be called during shutdown to clean up your module.
    // For modules that support dynamic reloading,
    // we call this function before unloading the module.
}

#undef LOCTEXT_NAMESPACE

IMPLEMENT_MODULE(FChatGPTPluginModule, ChatGPTPlugin)
```

❑ Defines the module and registers it using Unreal's **IMPLEMENT\_MODULE** macro

### ❑ Keypoints

- **IMPLEMENT\_MODULE(FChatGPTPluginModule, ChatGPTPlugin)** macro tells Unreal Engine:
  - "This is the module class for the plugin."
  - "Instantiate it at runtime when Unreal Engine loads the plugin."

- *FChatGPTPluginModule is not manually instantiated in ChatGPTPlugin.h or ChatGPTPlugin.cpp because Unreal Engine does this automatically.*
- *Function FModuleManager::Get().LoadModuleChecked<FChatGPTPluginModule>("ChatGPTPlugin") can be used to get a reference to the instantiated module if needed.*

# Practical Procedures (10)

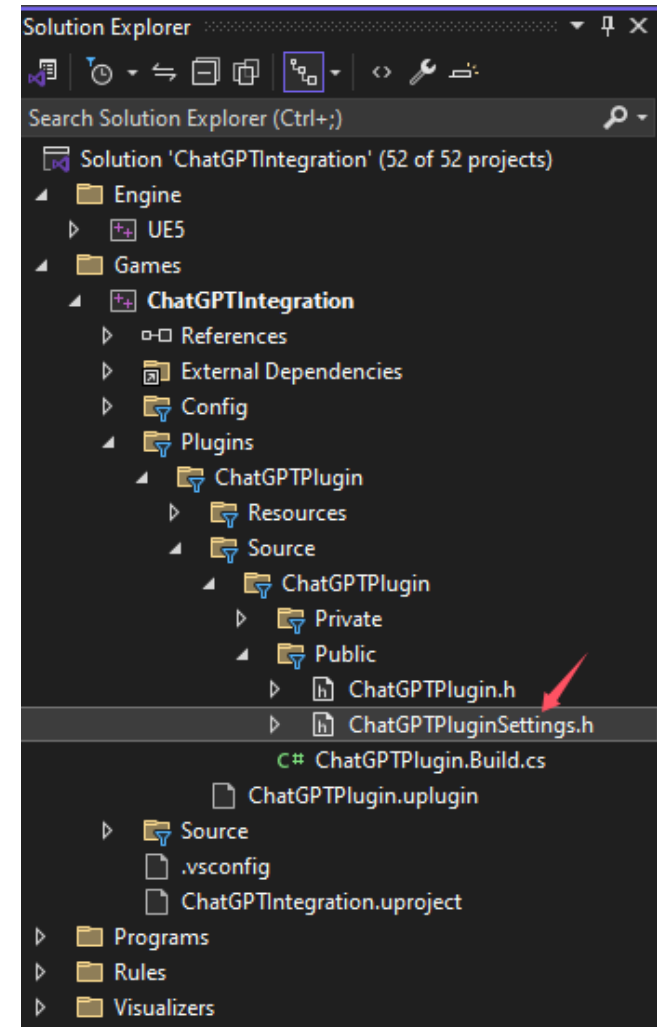
## Implement core functionality in C++: header- **ChatGPTPluginSettings.h** (1)

ChatGPT Plugin Settings for API configuration:

- Dynamic API Key and Endpoint: Allows for runtime configuration through Unreal Editor's *Project Settings*.
- Error Handling: Clear error messages for missing or invalid API keys and endpoints.
- Modular Parsing: Separate *ParseResponse* for clean JSON parsing.
- Performance: Minimized memory overhead by relying on settings when needed.

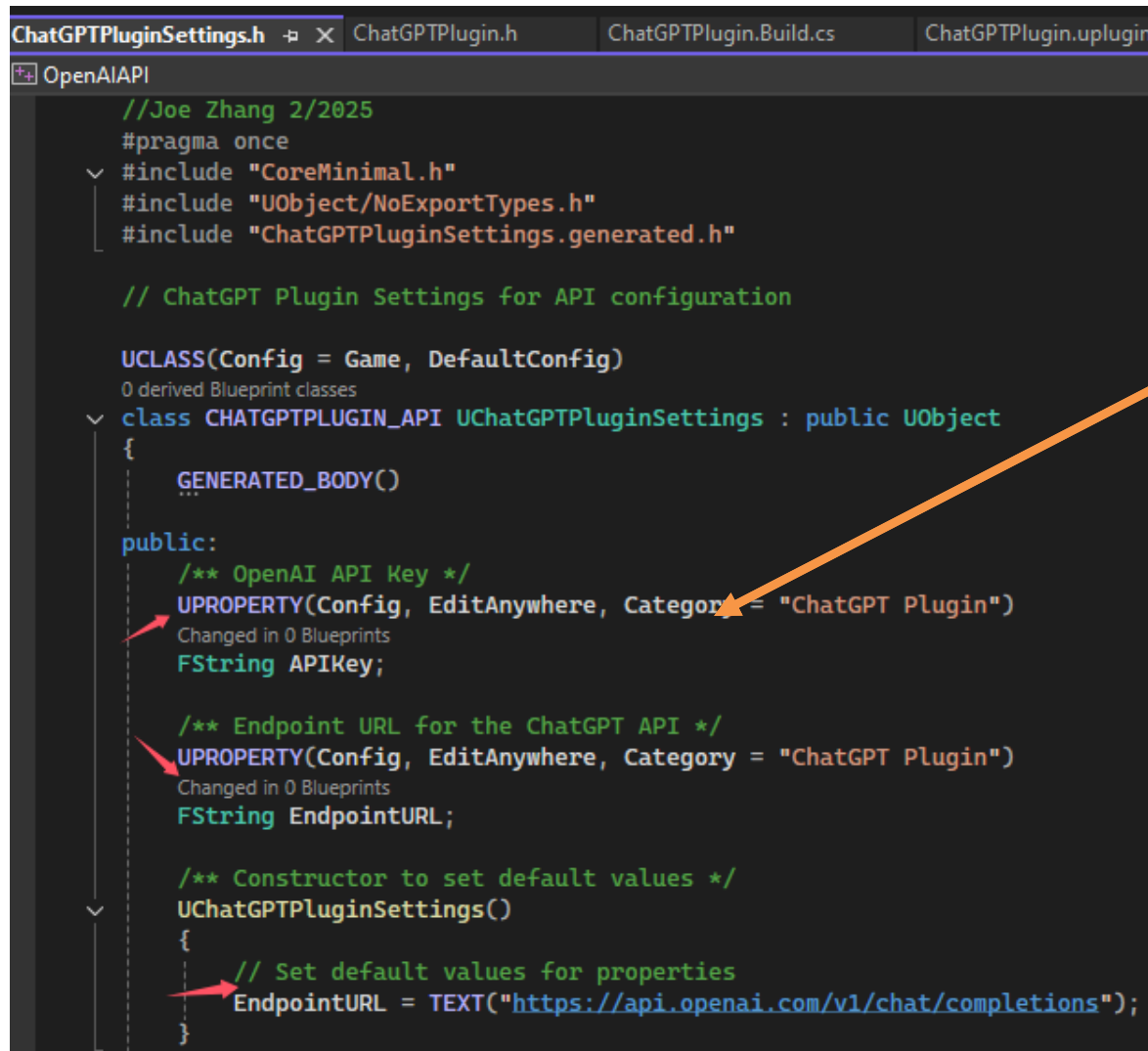
```
/** OpenAI API Key */
UPROPERTY(Config, EditAnywhere, Category = "ChatGPT Plugin")
Changed in 0 Blueprints
FString APIKey;

/** Endpoint URL for the ChatGPT API */
UPROPERTY(Config, EditAnywhere, Category = "ChatGPT Plugin")
Changed in 0 Blueprints
FString EndpointURL;
```



# Practical Procedures (11)

## Implement core functionality in C++: header- ChatGPTPluginSettings.h (2)



```
//Joe Zhang 2/2025
#pragma once
#include "CoreMinimal.h"
#include "UObject/NoExportTypes.h"
#include "ChatGPTPluginSettings.generated.h"

// ChatGPT Plugin Settings for API configuration

UCLASS(Config = Game, DefaultConfig)
0 derived Blueprint classes
class CHATGPTPLUGIN_API UChatGPTPluginSettings : public UObject
{
    GENERATED_BODY()

public:
    /** OpenAI API Key */
    UPROPERTY(Config, EditAnywhere, Category = "ChatGPT Plugin")
    FString APIKey;

    /** Endpoint URL for the ChatGPT API */
    UPROPERTY(Config, EditAnywhere, Category = "ChatGPT Plugin")
    FString EndpointURL;

    /** Constructor to set default values */
    UChatGPTPluginSettings()
    {
        // Set default values for properties
        EndpointURL = TEXT("https://api.openai.com/v1/chat/completions");
    }
}
```

UPROPERTY(Config, EditAnywhere, Category="ChatGPT Plugin"):

- The property will appear under a category labeled "*ChatGPT Plugin*" in the details panel.
- Best for grouping all plugin-related settings under a "ChatGPT Plugin" heading.

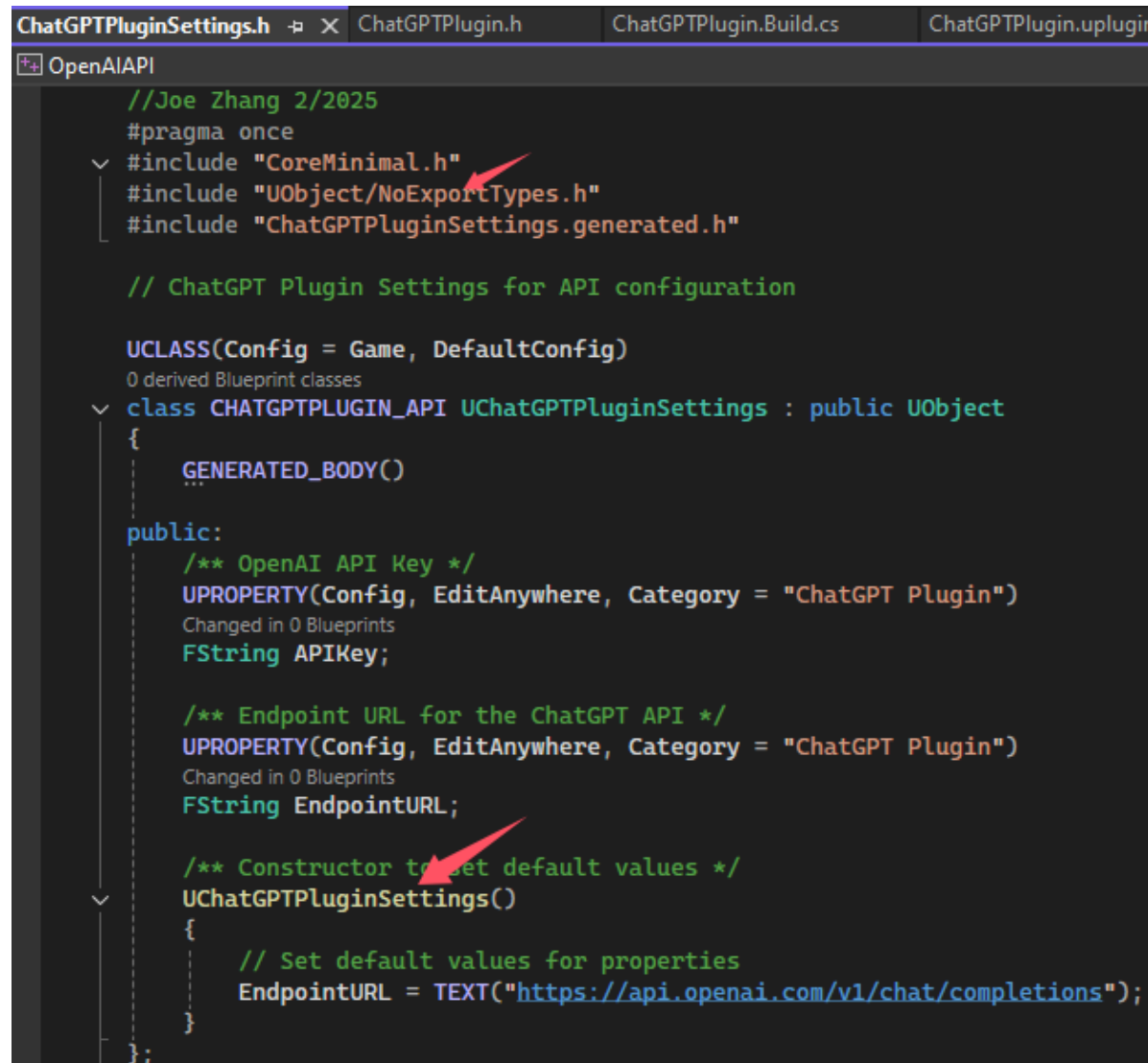
**EditAnywhere**: Enable visible and editable in the details panel in Unreal Editor;

**Config**: property should be loaded from or saved to a configuration file (e.g., DefaultGame.ini) and changes to this property in the Unreal Editor are persisted in the configuration file.

- **Category**: Defines how the property is organized in the details panel, and properties with the same category are grouped together in the editor.

# Practical Procedures (12)

## Implement core functionality in C++: header- ChatGPTPluginSettings.h (3)



```
//Joe Zhang 2/2025
#pragma once
#include "CoreMinimal.h"
#include "UObject/NoExportTypes.h"
#include "ChatGPTPluginSettings.generated.h"

// ChatGPT Plugin Settings for API configuration

UCLASS(Config = Game, DefaultConfig)
0 derived Blueprint classes
class CHATGPTPLUGIN_API UChatGPTPluginSettings : public UObject
{
    GENERATED_BODY()

public:
    /** OpenAI API Key */
    UPROPERTY(Config, EditAnywhere, Category = "ChatGPT Plugin")
    Changed in 0 Blueprints
    FString APIKey;

    /** Endpoint URL for the ChatGPT API */
    UPROPERTY(Config, EditAnywhere, Category = "ChatGPT Plugin")
    Changed in 0 Blueprints
    FString EndpointURL;

    /** Constructor to set default values */
    UChatGPTPluginSettings()
    {
        // Set default values for properties
        EndpointURL = TEXT("https://api.openai.com/v1/chat/completions");
    }
};
```

### Why Use the Constructor?

- **Compatibility:** Unreal Engine automatically calls the constructor when creating instances of `UObject`-derived classes, ensuring your default values are properly initialized.
- **Reflection safety:** Properties managed by UPROPERTY must adhere to Unreal's property system. Direct initialization in the header is not guaranteed to work properly with reflection and serialization.
- **Ease of future modifications:** If you need to adjust the default value, it's centralized in the constructor, reducing potential conflicts or overlooked changes.

# Practical Procedures (13)

## Core functionality in C++: header- **ChatGPTPlugin.h (1)**

Function Name	Purpose
StartupModule()	Initializes the plugin when loaded.
ShutdownModule()	Cleans up resources when the plugin is unloaded.
SendMessage()	Sends a user query to ChatGPT and handles the response.
CreateHttpRequest()	Creates and configures an HTTP request for the API.
OnResponseReceived()	Processes the API's HTTP response and triggers the callback.
ParseResponse()	Extracts useful data from the API's JSON response.

- Public functions (StartupModule, ShutdownModule, SendMessage) handle initialization, shutdown, and API interaction.
- Private functions (CreateHttpRequest, OnResponseReceived, ParseResponse) manage HTTP communication and response parsing.
- Ensures a ***modular, structured, and efficient API*** interaction within Unreal Engine.



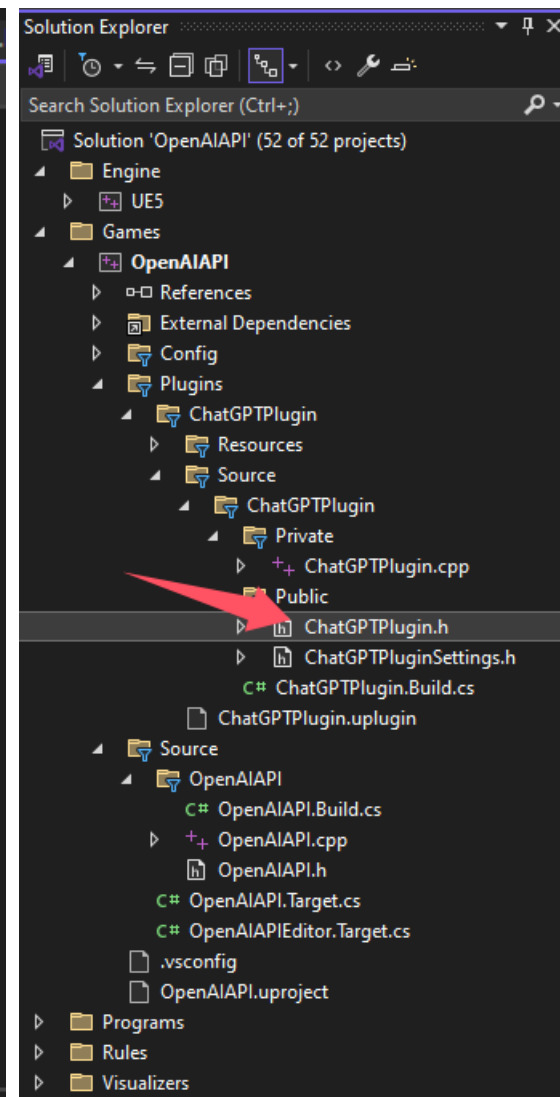
# Practical Procedures (14)

## Core functionality in C++: header- ChatGPTPlugin.h (2)

```
ChatGPTPluginSettings.h  ChatGPTPlugin.h  ChatGPTPlugin.Build.cs  ChatGPTPlugin.uplugin  OpenAIAP...
OpenAIAP...
//Joe Zhang, 2/2025
#pragma once
#include "CoreMinimal.h"
#include "HttpModule.h"
#include "Interfaces/IHttpRequest.h"
#include "Interfaces/IHttpResponse.h"
#include "Json.h"
#include "JsonUtilities.h"
#include "ChatGPTPluginSettings.h"

// The main module class for the ChatGPT plugin
class CHATGPTPLUGIN_API FChatGPTPluginModule : public IModuleInterface
{
public:
    /** IModuleInterface implementation */
    virtual void StartupModule() override;
    virtual void ShutdownModule() override;
    /** Sends a message to the ChatGPT API */
    void SendMessage(const FString& Message, TFunction<void(const FString&)> Callback);

private:
    /** Helper function to create an HTTP request */
    TSharedPtr<IHttpRequest> CreateHttpRequest(const FString& Url);
    /** Parses the HTTP response (forward declaration) */
    void ParseResponse(const FString& Reply, TFunction<void(const FString&)> Callback);
    /** Handles the HTTP response */
    void OnResponseReceived(FHttpRequestPtr Request, FHttpResponsePtr Response,
        bool bWasSuccessful, TFunction<void(const FString&)> Callback);
}
```



# Practical Procedures (15)

## Implement core functionality in C++: Implementation file - **ChatGPTPlugin.cpp (1)**

Function Name	Purpose
StartupModule()	Initializes the plugin and registers settings.
ShutdownModule()	Cleans up the module when the plugin is unloaded.
SendMessage()	Retrieves the API key & endpoint, then sends an HTTP request.
CreateHttpRequest()	Creates and configures the HTTP request with headers.
OnResponseReceived()	Handles the HTTP response and passes data to ParseResponse().
ParseResponse()	Extracts relevant text from the JSON response and invokes the callback.

- StartupModule and ShutdownModule handle initialization and cleanup.
- SendMessage sends a request to ChatGPT.
- CreateHttpRequest configures HTTP requests.
- OnResponseReceived processes API responses.
- ParseResponse extracts useful content from JSON.
- Ensures clean, modular, and efficient communication with OpenAI's API in Unreal Engine.



# Practical Procedures (15)

## Implement core functionality in C++: Implementation file - ChatGPTPlugin.cpp (2)

```
ChatGPTPluginSettings.h  ChatGPTPlugin.h  ChatGPTPlugin.Build.cs  ChatGPTPlugin.uplugin  OpenAIAPI.Build.cs  ChatGPTPlugin.cpp

OpenAIAPI
{
    #include "ChatGPTPlugin.h"
    #include "ChatGPTPluginSettings.h"
    #include "ISettingsModule.h"

    #define LOCTEXT_NAMESPACE "FChatGPTPluginModule"

    void FChatGPTPluginModule::StartupModule()
    {
        // Register plugin settings
        if (ISettingsModule* SettingsModule = FModuleManager::GetModulePtr<ISettingsModule>("Settings"))
        {
            SettingsModule->RegisterSettings("Project", "Plugins", "ChatGPT Plugin",
                LOCTEXT("ChatGPTPluginSettingsName", "ChatGPT Plugin"),
                LOCTEXT("ChatGPTPluginSettingsDescription", "Configure the ChatGPT plugin."),
                GetMutableDefault<UChatGPTPluginSettings>());
        }

        UE_LOG(LogTemp, Log, TEXT("ChatGPT Plugin has started.));

    }

    void FChatGPTPluginModule::ShutdownModule()
    {
        // Unregister plugin settings
        if (ISettingsModule* SettingsModule = FModuleManager::GetModulePtr<ISettingsModule>("Settings"))
        {
            SettingsModule->UnregisterSettings("Project", "Plugins", "ChatGPT Plugin");
        }

        UE_LOG(LogTemp, Log, TEXT("ChatGPT Plugin has shut down.));

    }
}
```

# Practical Procedures (16)

## Implement core functionality in C++: Implementation file - ChatGPTPlugin.cpp (3)

```
void FChatGPTPluginModule::SendMessage(const FString& Message, TFunction<void(const FString&)> Callback)
{
    const UChatGPTPluginSettings* Settings = GetDefault<UChatGPTPluginSettings>();
    if (!Settings || Settings->APIKey.IsEmpty() || Settings->EndpointURL.IsEmpty())
    {
        UE_LOG(LogTemp, Error, TEXT("API Key or Endpoint URL is not set in the plugin settings.));
        Callback(TEXT("Error: API Key or Endpoint URL is not set.));
        return;
    }

    FString Url = Settings->EndpointURL;

    // Create the HTTP request
    TSharedPtr<IHttpRequest> HttpRequest = CreateHttpRequest(Url);

    if (!HttpRequest.IsValid())
    {
        UE_LOG(LogTemp, Error, TEXT("Failed to create HTTP request.));
        Callback(TEXT("Error: Failed to create HTTP request.));
        return;
    }

    // Create JSON payload
    TArray<TSharedPtr<FJsonValue>> MessagesArray;
    TSharedPtr<FJsonObject> UserMessage = MakeShareable(new FJsonObject());
    UserMessage->SetStringField(TEXT("role"), TEXT("user"));
    UserMessage->SetStringField(TEXT("content"), Message);
    MessagesArray.Add(MakeShareable(new FJsonValueObject(UserMessage)));

    TSharedPtr<FJsonObject> JsonObject = MakeShareable(new FJsonObject());
    JsonObject->SetStringField(TEXT("model"), TEXT("gpt-3.5-turbo"));
    JsonObject->SetArrayField(TEXT("messages"), MessagesArray);
    JsonObject->SetNumberField(TEXT("max_tokens"), 150);
    JsonObject->SetNumberField(TEXT("temperature"), 0.7);

    // Serialize JSON payload
    FString Payload;
    TSharedRef<TJsonWriter<>> Writer = TJsonWriterFactory<>::Create(&Payload);
    FJsonSerializer::Serialize(JsonObject.ToSharedRef(), Writer);

    HttpRequest->SetContentAsString(Payload);

    // Set up the response handler
    HttpRequest->OnProcessRequestComplete().BindRaw(this, &FChatGPTPluginModule::OnResponseReceived, Callback);
    HttpRequest->ProcessRequest();
}
```

# Practical Procedures (17)

Implement core functionality in C++: Implementation file - **ChatGPTPlugin.cpp (4)**

```
//Creates and configures the HTTP request with headers.
TSharedPtr<IHttpRequest> FChatGPTPluginModule::CreateHttpRequest(const FString& Url)
{
    TSharedPtr<IHttpRequest> Request = FHttpModule::Get().CreateRequest();
    const UChatGPTPluginSettings* Settings = GetDefault<UChatGPTPluginSettings>();
    if (!Settings || Settings->APIKey.IsEmpty())
    {
        UE_LOG(LogTemp, Error, TEXT("API Key is not set.));
        return nullptr;
    }

    Request->SetVerb(TEXT("POST"));
    Request->SetURL(Url);
    Request->SetHeader(TEXT("Content-Type"), TEXT("application/json"));
    Request->SetHeader(TEXT("Authorization"), TEXT("Bearer ") + Settings->APIKey);
    return Request;
}
```

# Practical Procedures (18)

## Implement core functionality in C++: Implementation file - ChatGPTPlugin.cpp (5)

```
//Handles the HTTP response and passes data to ParseResponse().
void FChatGPTPluginModule::OnResponseReceived(FHttpRequestPtr Request, FHttpResponsePtr Response, bool bWasSuccessful, TFunction<void(const FString&)> Callback)
{
    if (bWasSuccessful && Response.IsValid())
    {
        FString Reply = Response->GetContentAsString();
        ParseResponse(Reply, Callback);
    }
    else
    {
        UE_LOG(LogTemp, Error, TEXT("HTTP request failed.));
        Callback(TEXT("Error: Unable to reach ChatGPT API.));
    }
}

void FChatGPTPluginModule::ParseResponse(const FString& Reply, TFunction<void(const FString&)> Callback)
{
    TSharedPtr<FJsonObject> JsonResponse;
    TSharedPtr<TJsonReader<>> Reader = TJsonReaderFactory<>::Create(Reply);

    if (FJsonSerializer::Deserialize(Reader, JsonResponse))
    {
        const TArray<TSharedPtr<FJsonValue>>* Choices;
        if (JsonResponse->TryGetArrayField(TEXT("choices"), Choices))
        {
            for (const auto& Choice : *Choices)
            {
                const TSharedPtr<FJsonObject>* Message;
                if (Choice->AsObject()->TryGetObjectField(TEXT("message"), Message))
                {
                    FString Content = (*Message)->GetStringField(TEXT("content"));
                    Callback(Content);
                    return;
                }
            }
        }

        Callback(TEXT("Error: Unable to parse ChatGPT response.));
    }
}
```

# Practical Procedures (19)

Implement core functionality in C++: Implementation file - **ChatGPTPlugin.cpp (6)**

***Don NOT forget***

```
//Cleans up the localization namespace to prevent unintended carryover into other files
#undef LOCTEXT_NAMESPACE

//Registers the module with Unreal Engine so it can be properly loaded and managed.
IMPLEMENT_MODULE(FChatGPTPluginModule, ChatGPTPlugin)
```

If `#undef LOCTEXT_NAMESPACE` is missing and another file forgets to define a new `LOCTEXT_NAMESPACE`, any `LOCTEXT` calls in that file might incorrectly inherit the namespace from this file. This could result in:

- Wrong translations being applied.
- Debugging headaches when the wrong text gets localized.

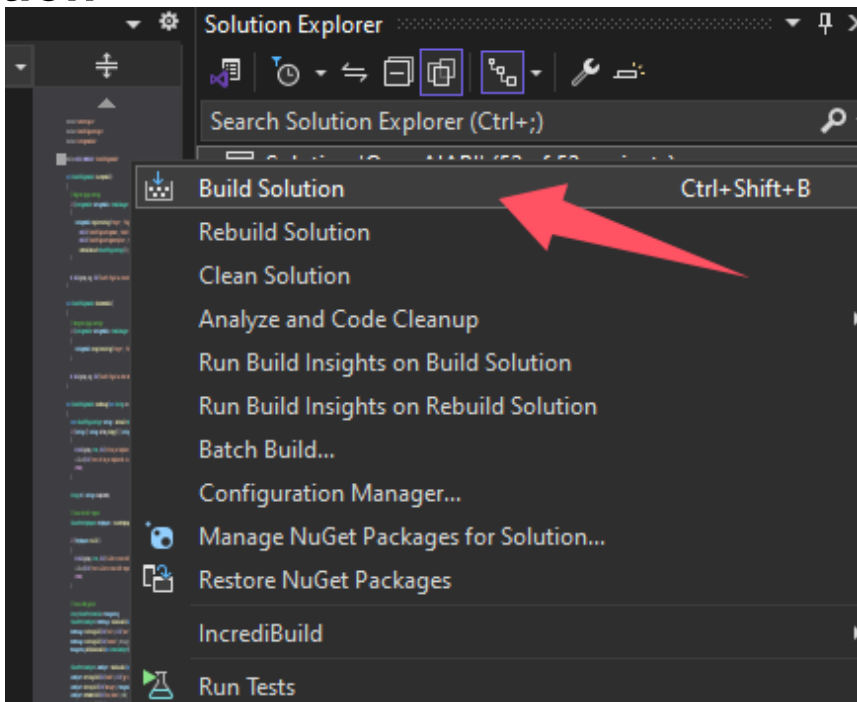
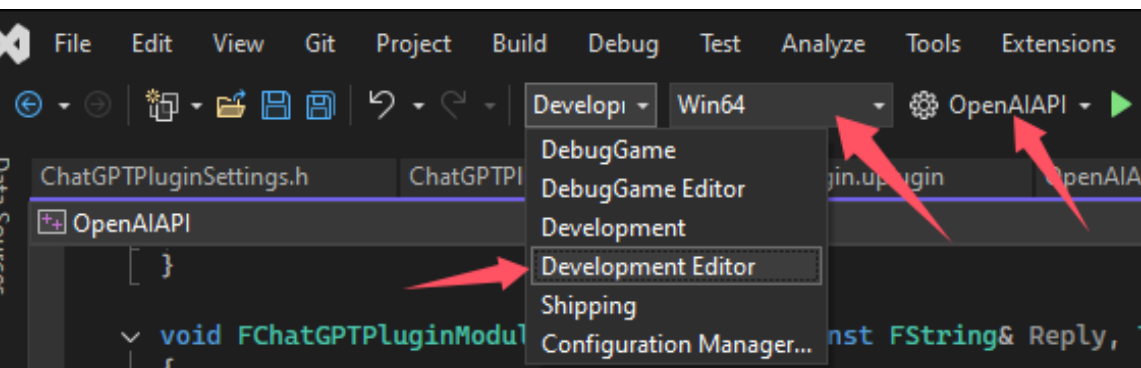
What Happens If You Forget `IMPLEMENT_MODULE`?

- Your plugin module will NOT be recognized or loaded by Unreal Engine!
  - Unreal won't call `StartupModule()`, meaning your plugin won't initialize.
  - Unreal won't call `ShutdownModule()`, meaning cleanup won't happen.

***Always use `IMPLEMENT_MODULE` at the end of your module's .cpp file.***

# Practical Procedures (20)

## Build and Get First ChatGPT Plugin: Configuration



```
51>----- Rebuild All started: Project: CookedEditor.Automation, Configuration: Development Any
51>CookedEditor.Automation -> C:\Program Files\Epic Games\UE_5.5\Engine\Binaries\DotNET\Automat
52>----- Rebuild All started: Project: LiveLinkHub.Automation, Configuration: Development Any
52>LiveLinkHub.Automation -> C:\Program Files\Epic Games\UE_5.5\Engine\Binaries\DotNET\Automat
===== Rebuild All: 51 succeeded, 0 failed, 1 skipped =====
===== Rebuild completed at 8:37 PM and took 01:35.078 minutes =====
```

# *Practical Procedures (21)*

## **Build and Get First ChatGPT Plugin: Directory Structure after Building**

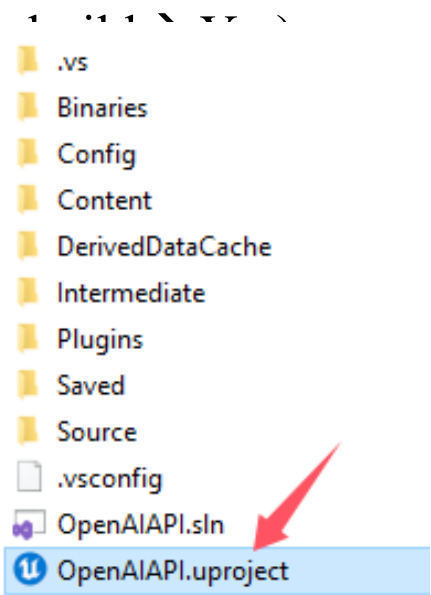
```
ChatGPTPlugin/  
├── Binaries/  
├── Content/      <-- This folder stores plugin assets  
├── Source/       <-- Source code files  
├── Resources/    <-- Icon and other metadata  
└── ChatGPTPlugin.uplugin
```



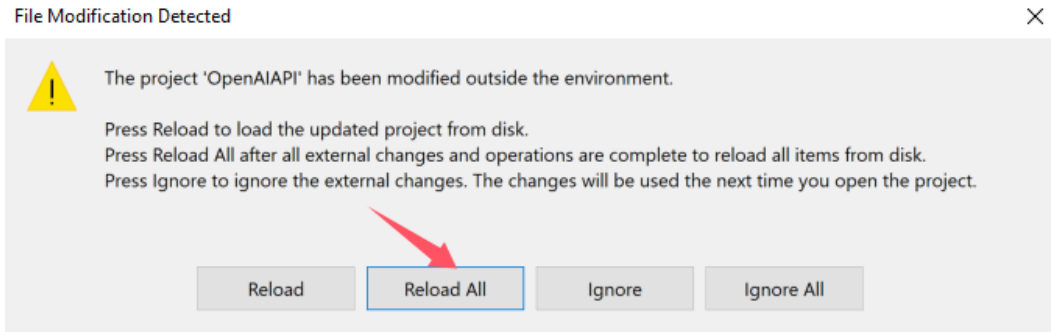
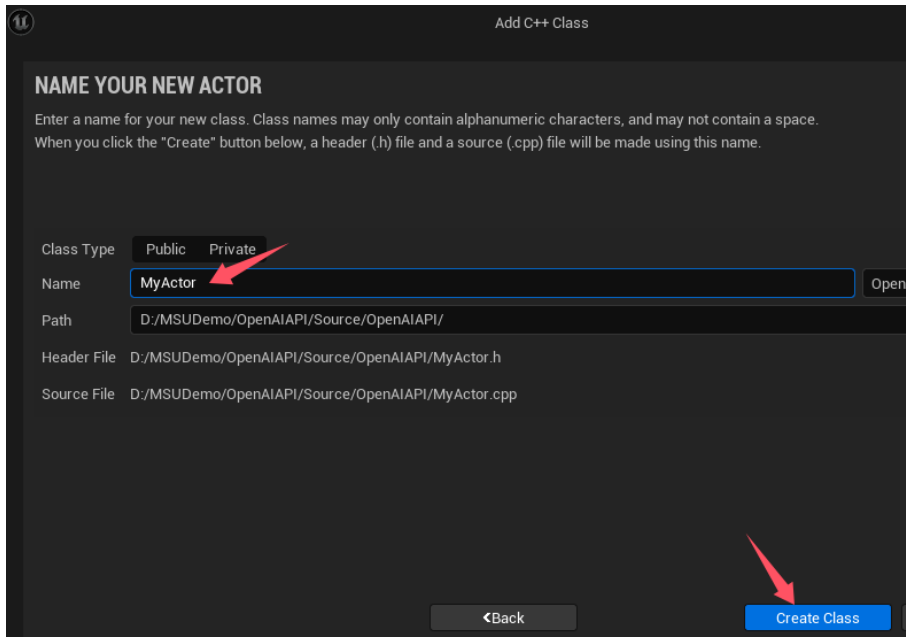
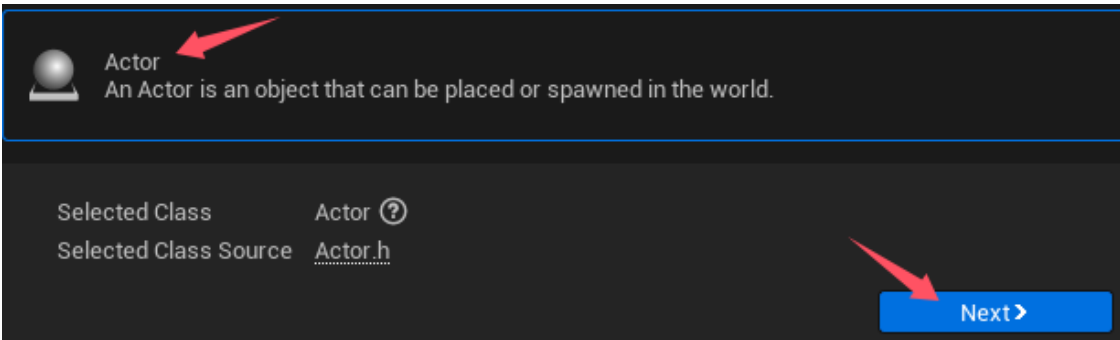
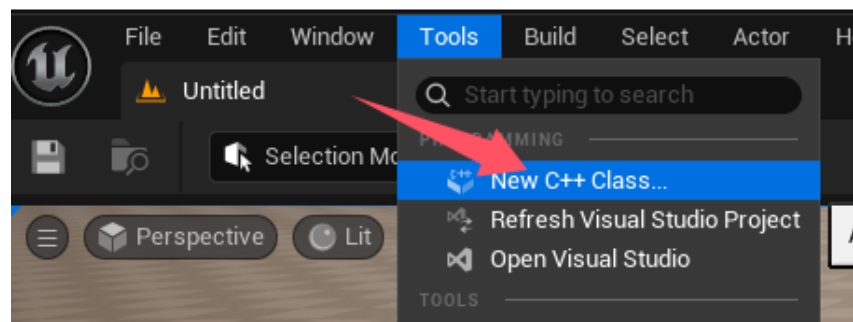
# Practical Procedures (22)

## Test the Plugin: Create a New Actor

→ Open Unreal Editor (if promoted to



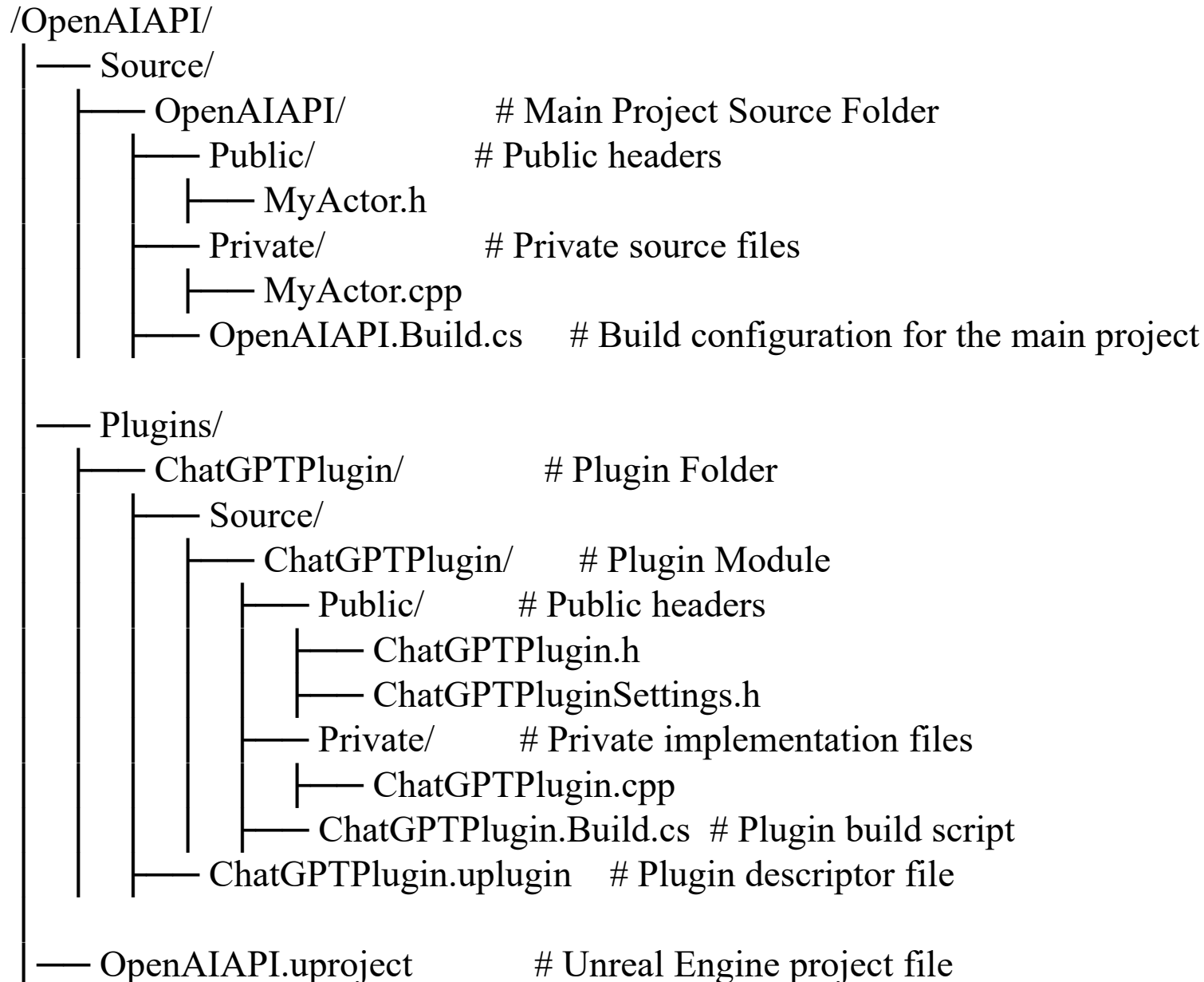
→ Create a test actor





# Practical Procedures (23)

## Test the Plugin: Directory Structure with New Actor

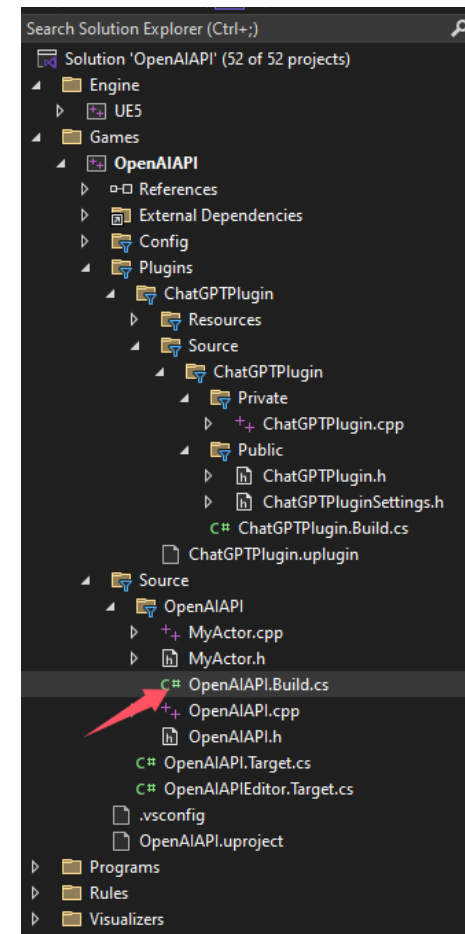


# Practical Procedures (24)

## Test the Plugin: Add new plugin dependency in OpenAI.API.Build.cs

```
class OpenAI.API : ModuleRules
{
public:
    OpenAI.API(ReadOnlyTargetRules Target) : base(Target)
    {
        PCHUsage = PCHUsageMode.UseExplicitOrSharedPCHs;

        PublicDependencyModuleNames.AddRange(new string[] { "Core", "CoreUObject", "Engine", "InputCore", "EnhancedInput", "ChatGPTPlugin" });
        PrivateDependencyModuleNames.AddRange(new string[] { });
    }
}
```



# Practical Procedures (25)

## Test the Plugin: MyActor.h (1)

- AMyActor is an actor class that tests the functionality of ChatGPTPlugin in Unreal Engine.
- It sends a request to the ChatGPT API when the game starts and logs the response.

Function	Purpose
AMyActor()	Constructor – Initializes default values.
BeginPlay()	Called when the game starts, triggers TestChatGPT().
TestChatGPT()	Sends a test message to ChatGPT and logs the response.
Tick(float)	Called every frame (not currently modified).

Included Headers	Purpose
"CoreMinimal.h"	Essential Unreal Engine types.
"ChatGPTPlugin/Public/ChatGPTPlugin.h"	Access to FChatGPTPluginModule.
"ChatGPTPlugin/Public/ChatGPTPluginSettings.h"	Access to API settings.
"GameFramework/Actor.h"	Base class for actors in Unreal Engine.

# Practical Procedures (26)

## Test the Plugin: MyActor.h (2)

```
OpenAIAPI.Build.cs  MyActor.cpp  MyActor.h  ↵  ✕

+ OpenAIAPI
  // Joe Zhang 2/2025

  #pragma once

  #include "CoreMinimal.h"
  #include "ChatGPTPlugin/Public/ChatGPTPlugin.h"
  #include "ChatGPTPlugin/Public/ChatGPTPluginSettings.h"
  #include "GameFramework/Actor.h"
  #include "MyActor.generated.h"

  //AMyActor is a test actor class designed to verify the functionality of the ChatGPT plugin.

  UCLASS()
  0 derived Blueprint classes
  class OPENAIAPI_API AMyActor : public AActor
  {
      GENERATED_BODY()

      public:
          // Constructor: Sets default values for this actor's properties
          AMyActor();

      protected:
          // Called when the game starts or when spawned
          virtual void BeginPlay() override;

          //Sends a test message to the ChatGPT plugin and logs the response.
          void TestChatGPT();

      public:
          // Called every frame
          virtual void Tick(float DeltaTime) override;
  };
};
```

# Practical Procedures (27)

## Test the Plugin: MyActor.cpp (1)

Implements the logic for AMyActor, as a test actor to verify the functionality of plugin

- AMyActor initializes at game start, checks the ChatGPT API key and endpoint, and sends a test request to OpenAI's API.
- The response is logged in Unreal Engine's output console.

Function	Purpose
AMyActor()	Constructor – Initializes default values and enables Tick().
BeginPlay()	Called when the game starts, retrieves API settings, and calls TestChatGPT().
TestChatGPT()	Sends a predefined prompt to ChatGPT and logs the response.
Tick(float)	Currently does nothing, but can be used for updates.

Included Headers	Purpose
"MyActor.h"	Includes the class definition.
"ChatGPTPlugin/Public/ChatGPTPlugin.h"	Allows access to FChatGPTPluginModule for sending messages.
"ChatGPTPlugin/Public/ChatGPTPluginSettings.h"	Retrieves API key and endpoint settings.

# Practical Procedures (28)

## Test the Plugin: MyActor.cpp (2)

```
OpenAPI.Build.cs  MyActor.cpp  MyActor.h
OpenAPI

// Joe Zhang

#include "MyActor.h"

// Constructor: Sets default values
AMyActor::AMyActor()
{
    PrimaryActorTick.bCanEverTick = true; // Enable Tick() if needed
}

// Called when the game starts or when spawned
void AMyActor::BeginPlay()
{
    Super::BeginPlay();

    // Access plugin settings
    const UChatGPTPluginSettings* Settings = GetDefault<UChatGPTPluginSettings>();

    if (!Settings || Settings->APIKey.IsEmpty())
    {
        UE_LOG(LogTemp, Error, TEXT("ChatGPT API Key is not set in the settings.));
        return;
    }

    if (Settings->EndpointURL.IsEmpty())
    {
        UE_LOG(LogTemp, Error, TEXT("ChatGPT Endpoint URL is not set in the settings.));
        return;
    }

    // Test the ChatGPT plugin functionality
    TestChatGPT();
}

// Sends a test message to the ChatGPT plugin and logs the response
void AMyActor::TestChatGPT()
{
    FString Prompt = TEXT("Can you briefly introduce Montclair State University?");

    // Get the plugin module instance
    FChatGPTPluginModule& ChatGPTModule = FModuleManager::GetModuleChecked<FChatGPTPluginModule>("ChatGPTPlugin");

    // Send a test message to ChatGPT
    ChatGPTModule.SendMessage(Prompt, [](const FString& Reply)
    {
        // Log the reply in the Output Log
        UE_LOG(LogTemp, Log, TEXT("ChatGPT Reply: %s"), *Reply);
    });
}

// Called every frame
void AMyActor::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);
}
```

# Practical Procedures (29)

**Test the Plugin: Try to Generate Visual Studio project files if build failed**

› OpenAIAPL

Name

- Content
- Intermediate
- Plugins
- Saved
- Source
- .vsconfig

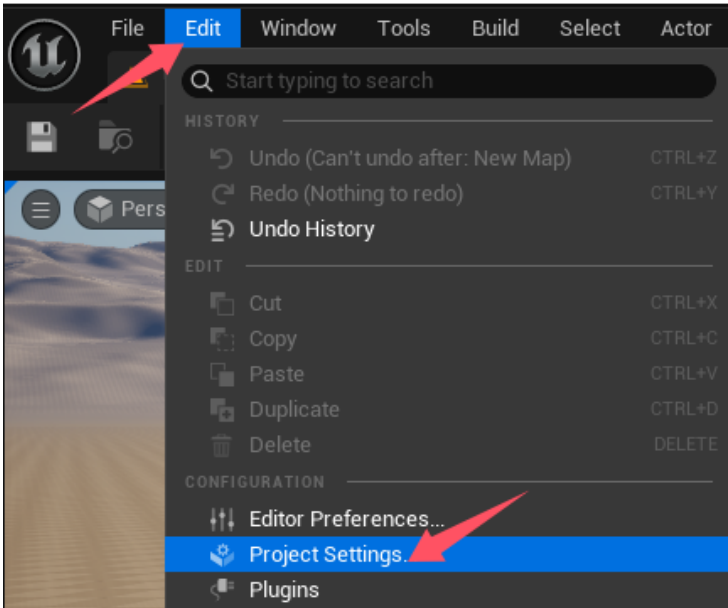
OpenAIAPLproject

Open

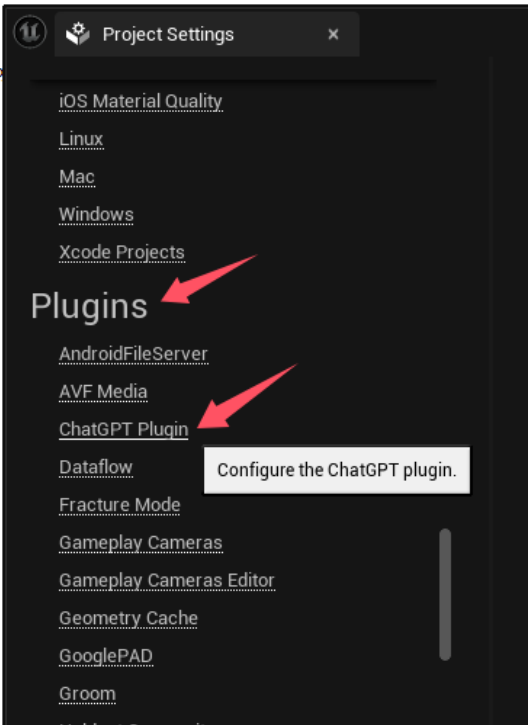
- Launch game
- Generate Visual Studio project files
- Switch Unreal Engine version...
- Share with Skype
- Edit with PyCharm Community Edition

# Practical Procedures (30)

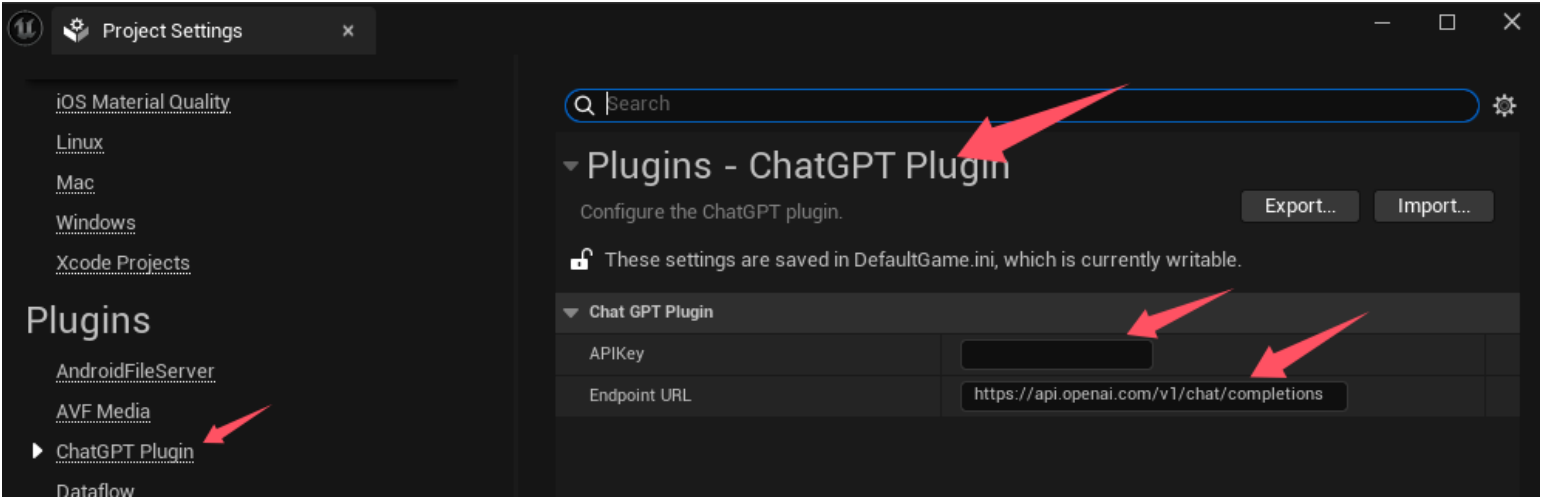
Run Unreal Editor Again  
→ Edit → Project Settings



→ Plugins → ChatGPTPlugin



→ Input API key and URL

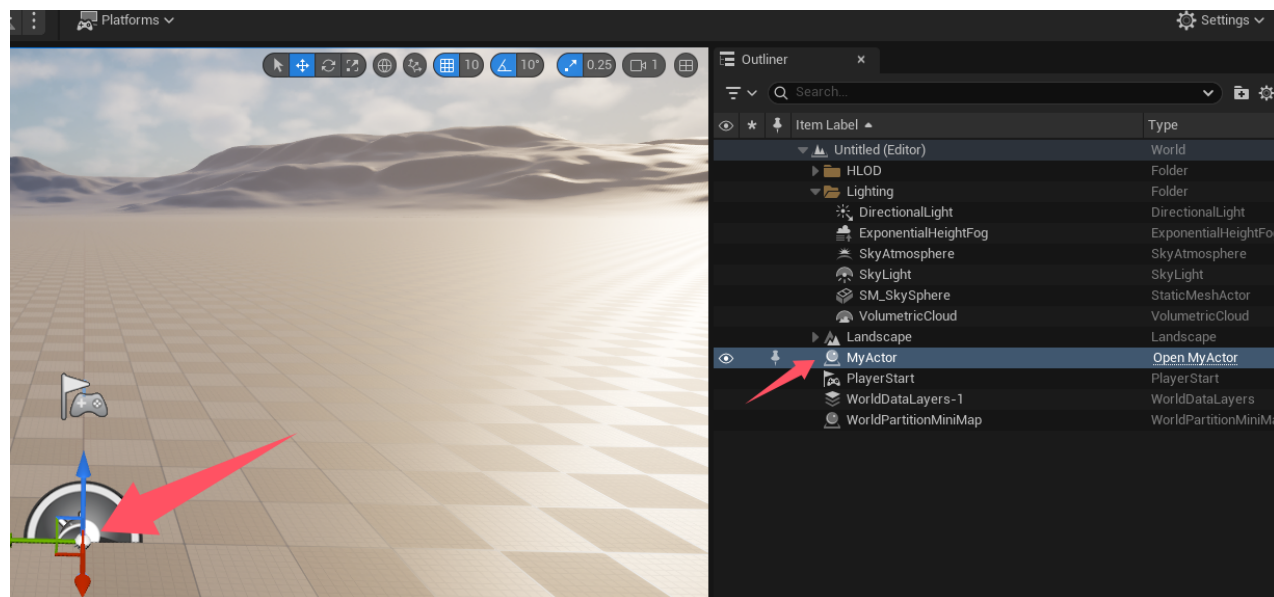
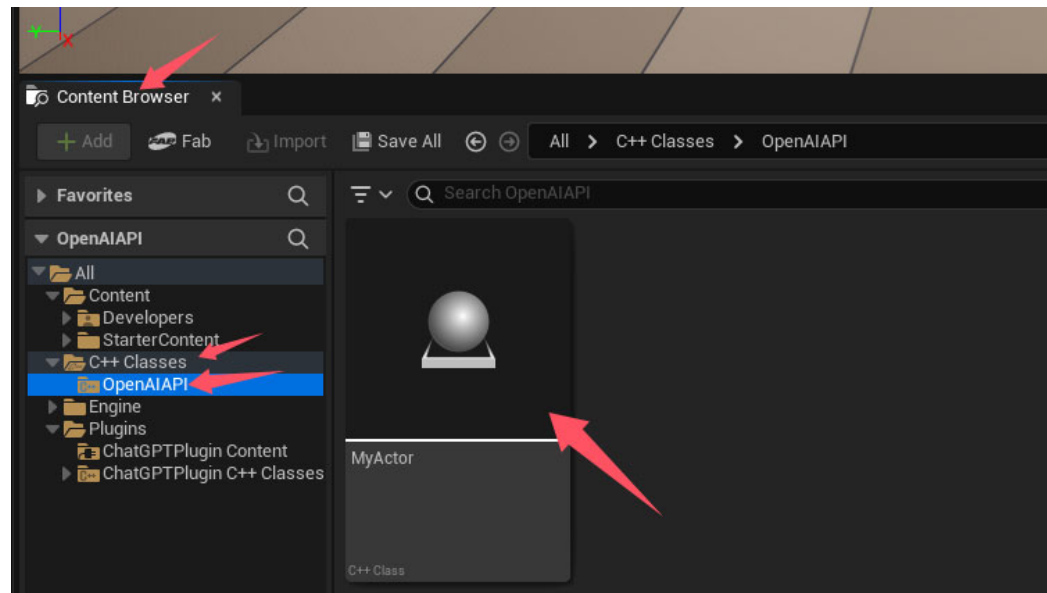




# Practical Procedures (31)

**Place MyActor in the Level:**

**→ Content Browser → C++ Classes → OpenAI API → MyActor → Drag MyActor to the Level**



# Practical Procedures (32)

Play this Level and Check Output Log:

