Integration Flow

Creating the AIS project involves setting up the overall solution structure, defining the Azure Functions, and ensuring the separation of concerns in accordance with the architecture provided. Below is a detailed outline for generating the required Azure Integration Services project, including project structures, class definitions, and function outlines.  
  
### Solution Structure  
  
The following hierarchy reflects the AIS architecture:  
  
```  
AIS.Integrations.sln  
│  
├── AIS.Integrations.Core.Common // Common classes and interfaces  
│ └── Core.Common.csproj  
│  
├── AIS.Integrations.Functions // Azure Functions project  
│ ├── FunctionsApp.csproj  
│ ├── Function PO-Main.cs (HTTP Trigger)  
│ ├── Function InitializeErrorVariables.cs  
│ ├── Function AuthorizeUser.cs  
│ ├── Function GetLastRunTimestamp.cs  
│ ├── Function PostLastRunTimestamp.cs  
│ ├── Function POReceiptQueryData.cs  
│ ├── Function POQueryData.cs  
│ ├── Function TransformPOReceiptData.cs  
│ ├── Function DistinctPOsQuery.cs  
│ ├── Function PODataFilter.cs  
│ └── Shared/Models/  
│ ├── ErrorModel.cs  
│ └── PODataModel.cs  
│   
└── AIS.Integrations.Adapters // Adapters for external services  
 ├── Adapters/HTTP/  
 │ ├── HttpClientConfiguration.cs  
 │ └── CustomHttpClient.cs  
 └── Adapters/SQL/  
 ├── SqlDatabaseConfiguration.cs  
 └── DataAccess.cs  
```  
  
### Project Files  
  
#### 1. Core.Common.csproj  
  
```xml  
<Project Sdk="Microsoft.NET.Sdk">  
 <PropertyGroup>  
 <TargetFramework>net6.0</TargetFramework>  
 <RootNamespace>AIS.Integrations.Core.Common</RootNamespace>  
 </PropertyGroup>  
</Project>  
```  
  
#### 2. FunctionsApp.csproj  
  
```xml  
<Project Sdk="Microsoft.NET.Sdk.Functions">  
 <PropertyGroup>  
 <TargetFramework>net6.0</TargetFramework>  
 <RootNamespace>AIS.Integrations.Functions</RootNamespace>  
 </PropertyGroup>  
 <ItemGroup>  
 <PackageReference Include="Microsoft.Azure.WebJobs.Extensions.Http" Version="4.\*" />  
 <PackageReference Include="Microsoft.NET.Sdk.Functions" Version="3.\*" />  
 <PackageReference Include="Newtonsoft.Json" Version="13.\*" />  
 </ItemGroup>  
</Project>  
```  
  
### Functions Implementation  
  
#### 3. Function: PO-Main.cs  
  
```csharp  
using Microsoft.AspNetCore.Mvc;  
using Microsoft.Azure.WebJobs;  
using Microsoft.Azure.WebJobs.Extensions.Http;  
using Microsoft.AspNetCore.Http;  
using Microsoft.Extensions.Logging;  
using System.Threading.Tasks;  
  
namespace AIS.Integrations.Functions  
{  
 public static class PO\_Maint  
 {  
 [FunctionName("PO-Main")]  
 public static async Task<IActionResult> Run(  
 [HttpTrigger(AuthorizationLevel.Function, "post", Route = null)] HttpRequest req,  
 ILogger log)  
 {  
 // Initialize error variables  
 await InitializeErrorVariables();  
  
 // Authorize User  
 await AuthorizeUser();  
  
 // Get Last Run Timestamp  
 await GetLastRunTimestamp();  
  
 // Call PO Receipt Data  
 await POReceiptQueryData();  
  
 // Check for distinct POs and proceed accordingly  
 // Logic goes here...  
  
 return new OkResult();  
 }  
 }  
}  
```  
  
#### 4. Function: InitializeErrorVariables.cs  
  
```csharp  
using Microsoft.Azure.WebJobs;  
using Microsoft.Extensions.Logging;  
  
namespace AIS.Integrations.Functions  
{  
 public static class InitializeErrorVariables  
 {  
 [FunctionName("InitializeErrorVariables")]  
 public static void Run(ILogger log)  
 {  
 // Initialize the variables...  
 log.LogInformation("Error variables initialized.");  
 }  
 }  
}  
```  
  
#### 5. Function: AuthorizeUser.cs  
  
```csharp  
using Microsoft.Azure.WebJobs;  
using Microsoft.Extensions.Logging;  
  
namespace AIS.Integrations.Functions  
{  
 public static class AuthorizeUser  
 {  
 [FunctionName("AuthorizeUser")]  
 public static async Task<string> Run(ILogger log)  
 {  
 // Implementation for Authorizing User  
 log.LogInformation("User authorized.");  
 return "Authorization Token";  
 }  
 }  
}  
```  
  
#### 6. Function: GetLastRunTimestamp.cs  
  
```csharp  
using Microsoft.Azure.WebJobs;  
using Microsoft.Extensions.Logging;  
  
namespace AIS.Integrations.Functions  
{  
 public static class GetLastRunTimestamp  
 {  
 [FunctionName("GetLastRunTimestamp")]  
 public static async Task<string> Run(ILogger log)  
 {  
 // Fetch last run timestamp logic...  
 log.LogInformation("Last run timestamp retrieved.");  
 return "LastRunTimestamp";  
 }  
 }  
}  
```  
  
#### 7. Function: PostLastRunTimestamp.cs  
  
```csharp  
using Microsoft.Azure.WebJobs;  
using Microsoft.Extensions.Logging;  
  
namespace AIS.Integrations.Functions  
{  
 public static class PostLastRunTimestamp  
 {  
 [FunctionName("PostLastRunTimestamp")]  
 public static async Task Run(ILogger log)  
 {  
 // Post last run timestamp logic...  
 log.LogInformation("Last run timestamp posted.");  
 }  
 }  
}  
```  
  
### Models  
  
#### 8. ErrorModel.cs  
  
```csharp  
namespace AIS.Integrations.Shared.Models  
{  
 public class ErrorModel  
 {  
 public string ProcessName { get; set; }  
 public string SourceSystem { get; set; }  
 // Additional properties...  
 }  
}  
```  
  
#### 9. PODataModel.cs  
  
```csharp  
namespace AIS.Integrations.Shared.Models  
{  
 public class PODataModel  
 {  
 public string Id { get; set; }  
 public string Name { get; set; }  
 // More properties as needed...  
 }  
}  
```  
  
### Adapters Directory  
  
In the `Adapters/HTTP` and `Adapters/SQL` folders, implement the necessary configurations to facilitate communication with HTTP and SQL services. You could use the HttpClient and Database SDKs as required.  
  
### Final Steps  
  
1. \*\*Add Project and References\*\*: Ensure all projects are included in the solution with the correct references. The Functions project should reference the Core.Common project and any necessary NuGet packages.  
2. \*\*Configuration Settings\*\*: Create a `local.settings.json` for local development and environment-based configurations including HTTP base URIs and database connection strings.  
3. \*\*Deployment\*\*: Use Azure Functions CLI or Azure Portal to deploy the functions into Azure after ensuring all the required settings and configurations are set properly.  
  
### Conclusion  
  
This outline provides a structured start toward creating the AIS project based on the requirements derived from the Southbound integration flow. Each Azure Function adheres to the principles of Clean Architecture, ensuring proper separation of concerns and maintainable code.