Integration Flow

To create an Azure Integration Services (AIS) project based on the provided architecture, we'll outline the project structure, components, and code snippets necessary for implementation. This includes Azure Functions, configuration, and the necessary references.  
  
### Project Structure  
  
Here's how the proposed Azure Functions project structure would look:  
  
```  
AIS.Integrations/  
├── AIS.Integrations.Functions/  
│ ├── FunctionApp.csproj  
│ ├── Startup.cs  
│ ├── PO\_Maintain.cs  
│ ├── POReceiptQuery.cs  
│ ├── POQuery.cs  
│ ├── CompanyReferenceQuery.cs  
│ └── VendorAddressCodeQuery.cs  
├── AIS.Integrations.Common/  
│ ├── CommonConfig.cs  
│ ├── ErrorHandler.cs  
│ └── Constants.cs  
└── local.settings.json  
```  
  
### Project File: `FunctionApp.csproj`  
  
```xml  
<Project Sdk="Microsoft.NET.Sdk.Functions">  
 <PropertyGroup>  
 <TargetFramework>net6.0</TargetFramework>  
 <AzureFunctionsVersion>v3</AzureFunctionsVersion>  
 <RootNamespace>AIS.Integrations.Functions</RootNamespace>  
 </PropertyGroup>  
 <ItemGroup>  
 <PackageReference Include="Microsoft.Azure.Functions.Extensions" Version="1.1.0" />  
 <PackageReference Include="Microsoft.Extensions.Http" Version="5.0.0" />  
 <PackageReference Include="MediatR" Version="9.0.0" />  
 <PackageReference Include="Newtonsoft.Json" Version="13.0.1" />  
 <PackageReference Include="Microsoft.ApplicationInsights.AspNetCore" Version="2.17.0" />  
 </ItemGroup>  
</Project>  
```  
  
### Configuration: `local.settings.json`  
  
```json  
{  
 "IsEncrypted": false,  
 "Values": {  
 "AzureWebJobsStorage": "UseDevelopmentStorage=true",  
 "FUNCTIONS\_WORKER\_RUNTIME": "dotnet",  
 "Salesforce\_Config": "<Salesforce API Credentials and Configuration>",  
 "AphriaCommon:Api:ReconnectionDelay": "1000",  
 "AphriaCommon:Api:ReconnectionAttempts": "5",  
 "sqlserver:Process": "<SQL Server Process>"  
 }  
}  
```  
  
### Azure Functions  
  
#### 1. PO\_Maintain.cs  
  
```csharp  
using Microsoft.Azure.WebJobs;  
using Microsoft.Extensions.Logging;  
using System.Net.Http;  
using System.Threading.Tasks;  
  
namespace AIS.Integrations.Functions  
{  
 public class PO\_Maintain  
 {  
 private readonly HttpClient \_httpClient;  
  
 public PO\_Maintain(HttpClient httpClient)  
 {  
 \_httpClient = httpClient;  
 }  
  
 [FunctionName("PO\_Maintain")]  
 public async Task Run(  
 [HttpTrigger(AuthorizationLevel.Function, "get", Route = "po/maintain")] HttpRequestMessage req,  
 ILogger log)  
 {  
 // Initialize Error Handler  
 var errorInitializer = new ErrorHandler();  
 await errorInitializer.InitializeError();  
  
 // Authorize User  
 var authorization = await \_httpClient.GetAsync("/api/v1.0/authorization/authorize");  
 authorization.EnsureSuccessStatusCode();  
  
 // Call helper functions for sub-flows  
 await POReceiptQuery.Run();  
 await POQuery.Run();  
 }  
 }  
}  
```  
  
#### 2. POReceiptQuery.cs  
  
```csharp  
using Microsoft.Azure.WebJobs;  
using Microsoft.Extensions.Logging;  
using System.Threading.Tasks;  
  
namespace AIS.Integrations.Functions  
{  
 public class POReceiptQuery  
 {  
 public static async Task Run()  
 {  
 // Implement query logic for PO Receipts here  
 // e.g., Call Salesforce API to retrieve PO receipt data  
 // Update LastRunTimestamp  
 }  
 }  
}  
```  
  
#### 3. POQuery.cs  
  
```csharp  
using Microsoft.Azure.WebJobs;  
using Microsoft.Extensions.Logging;  
using System.Threading.Tasks;  
  
namespace AIS.Integrations.Functions  
{  
 public class POQuery  
 {  
 public static async Task Run()  
 {  
 // Implement query logic for Purchase Orders here  
 // e.g., Call Salesforce API to query purchase orders  
 }  
 }  
}  
```  
  
#### 4. CompanyReferenceQuery.cs  
  
```csharp  
using Microsoft.Azure.WebJobs;  
using Microsoft.Extensions.Logging;  
using System.Threading.Tasks;  
  
namespace AIS.Integrations.Functions  
{  
 public class CompanyReferenceQuery  
 {  
 public static async Task Run()  
 {  
 // Implement query logic for company references here  
 // e.g., Query Salesforce configurations  
 }  
 }  
}  
```  
  
#### 5. VendorAddressCodeQuery.cs  
  
```csharp  
using Microsoft.Azure.WebJobs;  
using Microsoft.Extensions.Logging;  
using System.Threading.Tasks;  
  
namespace AIS.Integrations.Functions  
{  
 public class VendorAddressCodeQuery  
 {  
 public static async Task Run()  
 {  
 // Implement logic to find Vendor Address based on PO data  
 }  
 }  
}  
```  
  
### Common Utilities  
  
#### CommonConfig.cs  
  
```csharp  
namespace AIS.Integrations.Common  
{  
 public static class CommonConfig  
 {  
 public static string SalesforceConnectionString => "<Salesforce Connection String>";  
 }  
}  
```  
  
#### ErrorHandler.cs  
  
```csharp  
using Microsoft.Extensions.Logging;  
using System.Threading.Tasks;  
  
namespace AIS.Integrations.Common  
{  
 public class ErrorHandler  
 {  
 public async Task InitializeError()  
 {  
 // Implement logic to initialize error and log to Application Insights  
 }  
 }  
}  
```  
  
### Deployment  
  
Deploy the project structure to Azure Functions using Azure CLI or Azure DevOps pipeline, ensuring that all dependencies and settings are configured correctly.  
  
### Notes  
- Ensure the necessary APIs are accessible and correctly set up in Azure Portal.  
- Configure the Application Insights for monitoring and logging purposes.  
- Use dependency injection as needed to manage HTTP client instances effectively.  
  
This project structure provides a modular and scalable approach to implementing the AIS architecture based on the provided MuleSoft integration design. Each function is purpose-driven and can be tested independently to ensure compliance with the overall architecture standards.