Integration Flow

To generate an Azure Integration Services (AIS) project based on the provided architecture, I'll outline the project structure and components. The project will consist of Azure Functions, connectors, and related resources that correspond to the AIS architecture. Below are the specifications for each component:  
  
### Project Structure  
  
```  
AIS.Integrations/  
 ├── AIS.Integrations.API/  
 │ ├── FunctionApps/  
 │ │ ├── POAuthorizationFunction.cs  
 │ │ ├── GetLastRunTimestampFunction.cs  
 │ │ ├── POReceiptQueryFunction.cs  
 │ │ ├── POQueryFunction.cs  
 │ │ ├── PutPOtoERPFunction.cs  
 │ │ ├── PostLastRunTimestampFunction.cs  
 │ │ ├── CompanyReferenceFunction.cs  
 │ │ ├── VendorAddressQueryFunction.cs  
 │ │   
 │ ├── local.settings.json  
 │ └── AIS.Integrations.API.csproj  
 │   
 ├── AIS.Integrations.Core.Common/  
 │ ├── Models/  
 │ │ ├── POData.cs  
 │ │ ├── POReceiptData.cs  
 │ │ ├── CompanyReference.cs  
 │ │ ├── VendorAddress.cs  
 │ │   
 │ ├── Interfaces/  
 │ │ ├── IErrorLogger.cs  
 │ │   
 │ └── AIS.Integrations.Core.Common.csproj  
 │   
 ├── AIS.Integrations.Salesforce/  
 │ └── AIS.Integrations.Salesforce.csproj  
 │   
 ├── AIS.Integrations.SQLServer/  
 │ └── AIS.Integrations.SQLServer.csproj  
 │   
 ├── AIS.Integrations.ERP/  
 │ └── AIS.Integrations.ERP.csproj  
 │   
 └── AIS.Integrations.sln  
```  
  
### Project Components  
  
1. \*\*Function App\*\*:  
 - Each of the listed functions will be implemented in separate C# files within the `FunctionApps` folder.  
 - Example code structure of POAuthorizationFunction.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.Net.Http;  
 using System.Threading.Tasks;  
  
 public static class POAuthorizationFunction  
 {  
 [FunctionName("POAuthorizationFunction")]  
 public static async Task<IActionResult> Run(  
 [HttpTrigger(AuthorizationLevel.Function, "get", Route = "authorize")] HttpRequest req,  
 ILogger log)  
 {  
 // Logic for authorization  
 return new OkObjectResult("User authorized");  
 }  
 }  
 ```  
  
2. \*\*Common Models and Interfaces\*\*:  
 - The `Models` folder holds the data model classes such as `POData`, `POReceiptData`, `CompanyReference`, `VendorAddress`.  
 - The `Interfaces` folder contains common interfaces for error logging and other shared functionalities.  
  
3. \*\*Connectors\*\*:  
 - Separate projects for Salesforce, SQL Server, and ERP connections will be included, each implementing API-specific classes and methods for interfacing with those APIs.  
  
4. \*\*Configuration\*\*:  
 - The `local.settings.json` file is to hold configuration values such as connection strings for SQL Server, Salesforce credentials, ERP API endpoints, etc.  
  
### Example local.settings.json  
  
```json  
{  
 "IsEncrypted": false,  
 "Values": {  
 "AzureWebJobsStorage": "YourStorageConnectionString",  
 "FUNCTIONS\_WORKER\_RUNTIME": "dotnet",  
 "Salesforce:ClientId": "YourClientId",  
 "Salesforce:ClientSecret": "YourClientSecret",  
 "ERP:ApiEndpoint": "YourERPApiEndpoint",  
 "SQLServer:ConnectionString": "YourSQLServerConnectionString"  
 },  
 "Host": {  
 "CORS": "\*",  
 "LocalHttpPort": 7071,  
 "Experimentation": {  
 "PreloadEnabled": false  
 }  
 }  
}  
```  
  
### Example AIS.Integrations.API.csproj  
  
```xml  
<Project Sdk="Microsoft.NET.Sdk.Functions">  
 <PropertyGroup>  
 <TargetFramework>net6.0</TargetFramework>  
 <AzureFunctionsVersion>v3</AzureFunctionsVersion>  
 </PropertyGroup>  
 <ItemGroup>  
 <PackageReference Include="Microsoft.Azure.Functions.Extensions" Version="1.1.0" />  
 <PackageReference Include="Microsoft.Extensions.Http" Version="5.0.0" />  
 <PackageReference Include="Newtonsoft.Json" Version="12.0.3" />  
 </ItemGroup>  
</Project>  
```  
  
### Solution File (AIS.Integrations.sln)  
  
Create a new solution file that includes all projects:  
```shell  
dotnet new sln -n AIS.Integrations  
dotnet sln add AIS.Integrations.API/AIS.Integrations.API.csproj  
dotnet sln add AIS.Integrations.Core.Common/AIS.Integrations.Core.Common.csproj  
dotnet sln add AIS.Integrations.Salesforce/AIS.Integrations.Salesforce.csproj  
dotnet sln add AIS.Integrations.SQLServer/AIS.Integrations.SQLServer.csproj  
dotnet sln add AIS.Integrations.ERP/AIS.Integrations.ERP.csproj  
```  
  
### Final Notes  
- This structure allows for a clear separation between different parts of the integration solution, adhering to principles such as SOLID and Clean Code.  
- Each Azure Function acts as a microservice, allowing for independent scaling and updates.  
- Ensure to implement appropriate error handling and logging within the functions to maintain observability.  
  
You can proceed to create this project structure by implementing the outlined components in a local or cloud-based development environment.