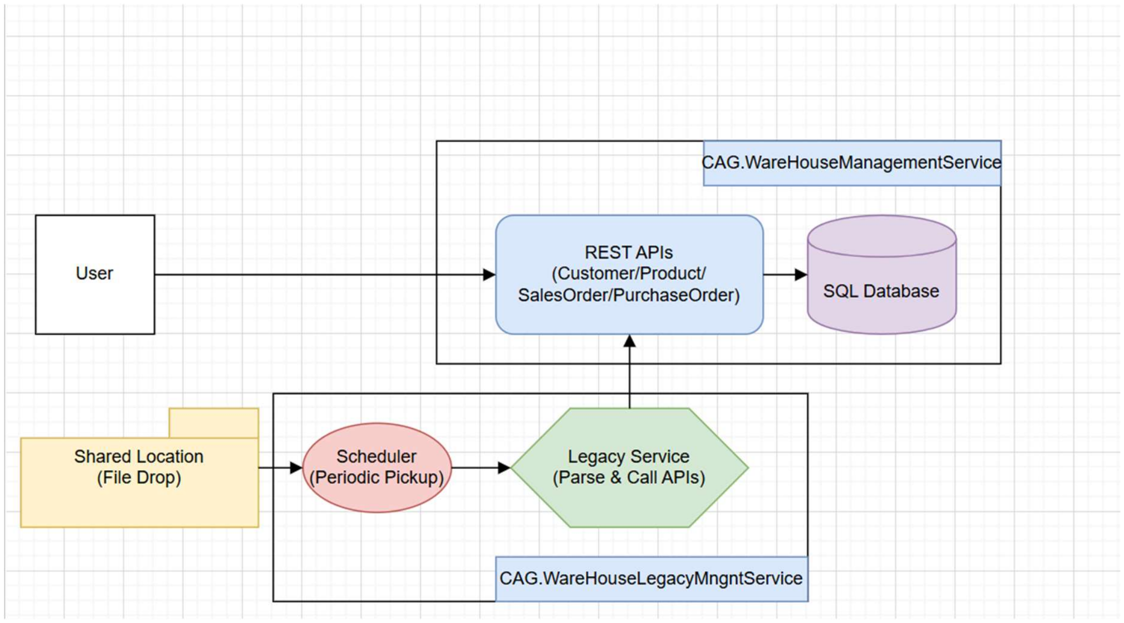
CAG – WareHouseManagementSystem

Functional Background:

CAG-WareHouseManagement System, is designed to enable

- Real-time ingestion via RESTful APIs for Products, Purchase Orders (POs) & Sales Orders (SOs).
- Scheduled polling of legacy data files and ingesting data for Products, Purchase Orders (POs) & Sales Orders (SOs).

Flow Diagram:


CAG.WareHouseManagementService
User
REST APIs (Customer/Product/ SalesOrder/PurchaseOrder)
SQL Database
Shared Location (File Drop)
Scheduler (Periodic Pickup)
Legacy Service (Parse & Call APIs)
CAG.WareHouseLegacyMngntService

data_ingestion_flow_fixed.drawio

REST API Swagger Specification:

CAG.WarehouseManagementSystem.yml

Technical Patterns/Designs implemented:

CAG.WareHouseManagement Service REST API design pattern :
Microservices design implemented with Rest Controller pattern + Repository pattern.

Feature	Design Strategy
Microservice design pattern	1) Created two microservices - a) CAG.WareHouseManagementService microservice for REST API calls b) CAG.WareHouseLegacyMngntService microservice to manage Legacy files- as file processing requires different CPU/Memory
Dependency Injection - Autofac	1) Implement a custom easy to use strategy in program.cs. 2) All classes just need to inherit Iscoped/Itransient interfaces - and they will be registered

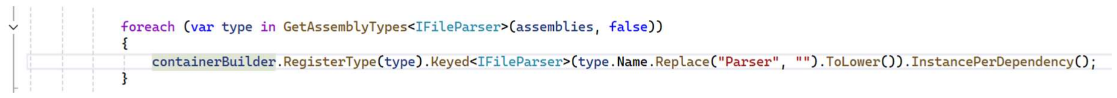
Repository Pattern - Generic	IRepository<T> supports regular CRUD scross entities
Configurable Data Source	1) For Dev testing - InMemory SQLLite, for real-world SQL source - 2) auto configred via DatabaseType in appsetttings.json
Swagger Integration - Swashbuckle	Swagger UI for Dev testing
Centralised Exception Handling-Filters	CagExceptionFilter & custom business exception
Configurable logging via Microsoft.Extensions.Logging	appsettings.json has logging config
DTOs & AutoMapper	Mapping logic simplified
EagerLoading	To link PurchaseOrder to list of orders

CAG.WareHouseLegacyMngnt Service Scheduled Poller over File System:

Periodcally poll a configurable file folder location, read files and ingest the information to CAG via RestAPI(developed above).

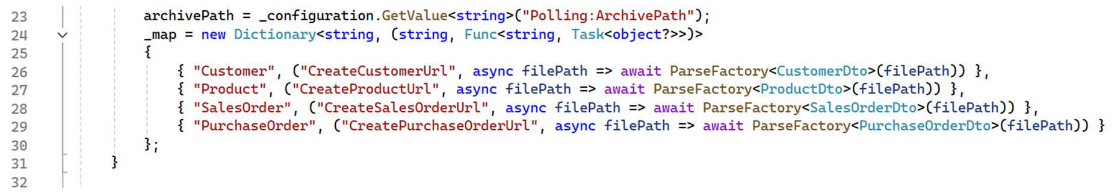
File Parsing strategy - Extensible

Implmented using Strategy pattern + Factory pattern(via autofac key based registration in Program.cs).
To implement a new file format(Example Json), just create a class JsonParser implementing interface IFileParser.
Everything will work seamlessly.



```
foreach (var type in GetAssemblyTypes<IFileParser>(assemblies, false))
{
    containerBuilder.RegisterType(type).Keyed<IFileParser>(type.Name.Replace("Parser", "").ToLower()).InstancePerDependency();
}
```

Entity Type strategy – Extensible
We identify the Entity to be updated via FileName – Example "Customer_1.xml".
Implemented using strategy pattern + factory pattern via Dictionary
To support a new Entity
(1) Create a Dto class
(2) Add an entry in the dictionary in FileProcessService
Everything will work seamlessly.



```
        archivePath = _configuration.GetValue<string>("Polling:ArchivePath");
        _map = new Dictionary<string, (string, Func<string, Task<object?>>)>
        {
            { "Customer", ("CreateCustomerUrl", async filePath => await ParseFactory<CustomerDto>(filePath)) },
            { "Product", ("CreateProductUrl", async filePath => await ParseFactory<ProductDto>(filePath)) },
            { "SalesOrder", ("CreateSalesOrderUrl", async filePath => await ParseFactory<SalesOrderDto>(filePath)) },
            { "PurchaseOrder", ("CreatePurchaseOrderUrl", async filePath => await ParseFactory<PurchaseOrderDto>(filePath)) }
        };
    }
```

Feature	Design Strategy
Cron scheduler Job via Quartz lib	Created FilePollingJob - with configurable Cron scheduling expression via appsettings.json
Retry via polly	File operation - retriable using Polly
Exception Handling & Logging	appsettings.json has logging config
File archival	Archived processed file to stop re-processing
HttpClient	To call the Rest APIs