# Feed-products: POC

# -Where to find your code

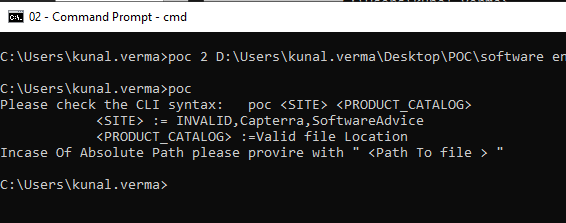
GitHub: <https://github.com/kunalverma94/POC>

Video: <https://raw.githubusercontent.com/kunalverma94/POC/master/coding/Demo.mp4>

(Inside Repo)SQL: <https://raw.githubusercontent.com/kunalverma94/POC/master/database/POCanswer.sql>

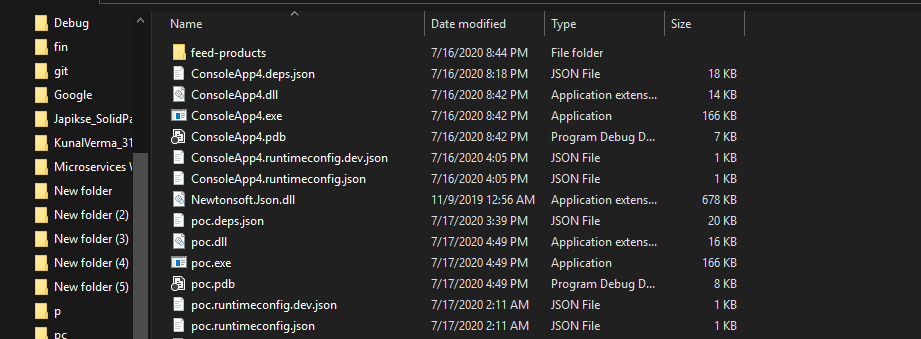
# Prerequisites:

* **Syntax:**

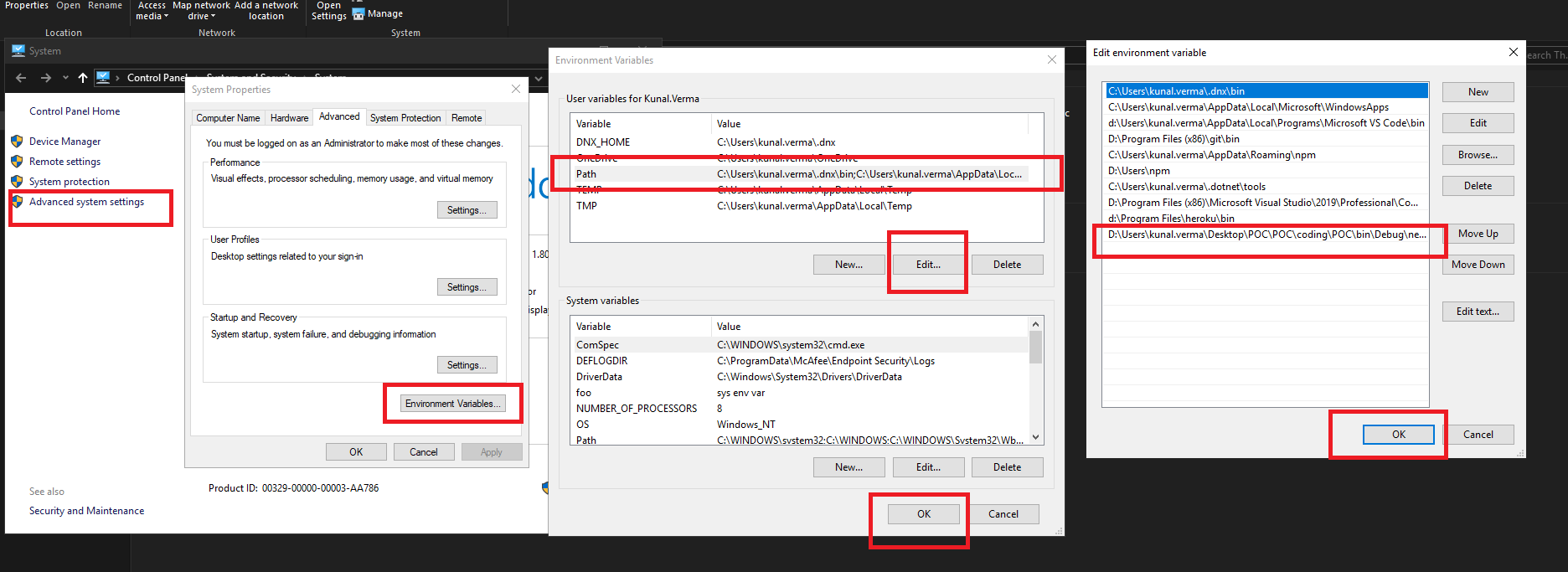
****

* Dotnet Core Setup and Installation :
  + Linux: Follow instructions here
    - <https://docs.microsoft.com/en-us/dotnet/core/install/linux>
  + Windows : <https://dotnet.microsoft.com/download>

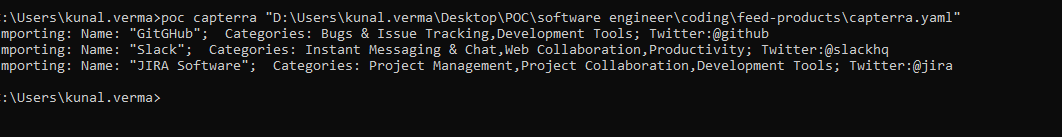
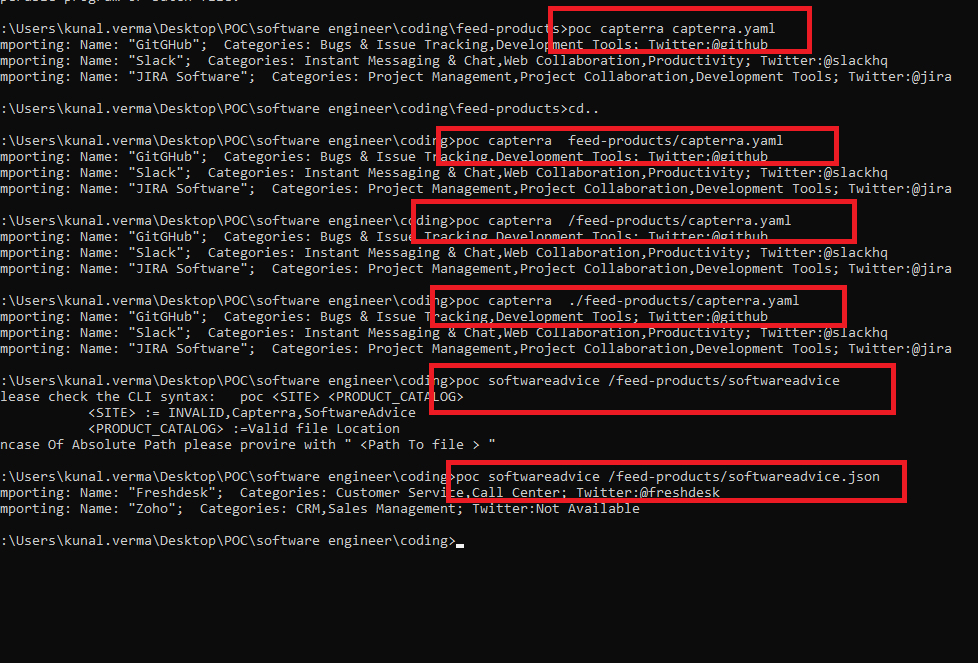
# - Installation steps & How to run your code / tests

1. Build /get the executable from application .
   1. Available AT \bin\Debug\netcoreapp3.1\
2. 

Add it in the Environment Root Folder .



Now you It in following :

1. Absolute Path
   1. 
2. Relative Path
3. 

# Proof of Concept:

The application developed in .net Core 3.1.

For Maintainability and Scalability following design patterns SOLID principals have been added as Of current Implementation

1. Factory Pattern
2. Builder Pattern
3. Adapter Pattern
4. Repository Like Pattern
5. Dependency Injection (IOC)

Further Development May involve:

1. Distributed of DTO ,modules into NuGet or standard DLL .
2. More level of abstract or modularity could be implemented .
3. Adding Comments and Summary Tags for more readability.
4. Automation
5. CLI commands enrichment
6. Providing more flags like
   1. –source : Web, local, sql, network
   2. –format :csv, json, yaml

Adding New CLients

For each Client add the Service Provider for the respective.

Add DTO and Business Adapter to provide Singularity and necessary Abstraction and Separation.

Hook or Inject the Data Repository with The Data Reader Of desired type.

It can be local, web , Mongo any

Add necessary serialize It can be any csv xml yaml json …

Features:

* NO strong Dependency:
  + Interface backs all the models.
  + Makes unit and Module testability easy.
  + Implementation guidelines
  + The comments could be added only in the interface and are reflected by inherit doc supplying more readability .
* Abstraction:
  + The products are modeled into an instance Of IProduct .
  + An Adapter pattern ensures for providing all IProducts with same level of business maturity.
* IOC : Dependency Injection
  + Supplies Lean and clean way of initialization and injection of instances.
* MOQ :
  + Testing and Mocking done using MOQ library for testability
* Modularity:
  + 3 level
    - Service Layer
    - Data Layer
    - Reader Layer (lower level dealing with files, networks and stuff)
* Tries to Support “O” For open close
  + Provided as much implementation and separation for addition of new provide, source, and products service.
* All the basic appsetting and const variables provided in Appsetting static class To avoided and encourage placing all const fields ,to avoid bugs and redundancy.

Thank really enjoyed working on assignment and was a nice Learning Experience .And Took an approach in building a CLI based applications dealing with the args on fly.