**DynamicETL\_Validator:**

* **Description:**
  + Project provides Web APIs used to validate ETLs using the Flask library in python.
  + APIs are functions in python. They support dependency injection similar to Unity containers and ASP.net patterns in C#.
* **App configuration:**
  + Some app attributes can be configured in ScriptArgs/DynamicETL\_Validator.json:
    - "apijson": JSON path to API configuration (string, folder must exist, points to json file).
    - "debug": Put True if want to run in debug mode (Boolean).
    - "logpath": Path to output application log file (string, must exist).
  + Apis are configured using .json file with following arguments (currently is Configs/APIArgs.json but can be any file with following attributes, with path in “apijson”):
    - "AppConfig" : Stores attributes regarding the application itself. Dictionary with following attributes:
      * "AppName" : Name of application (string).
      * "HostName" : Hostname for application (ex: 127.0.0.1) (string, must be valid hostname).
    - "APIs" : Stores configurations for all APIs that may be injected into target functions (dictionary with { “APIName” -> { “Config” -> “Val” } }.
      * “APIName” string must match name of configuration section object, case sensitive.
      * Argument dictionary keys and values match configuration section object constructor arguments, case sensitive.
* **Current APIs:**
  + **ETLSummaryReportAPI:** Using POSTrequest to /ETLSummaryReport/GenerateReport/ route, generate report detailing the following attributes:
    - Required**:**
      * "etl": Name of ETL performed (string).
      * "tablename": Destination table for ETL (string).
      * "database": Database storing table (string).
      * "sourcedata": { “Column” -> [Data] } containing data from source (file, another table, etc) (JSON object).
      * "insertdata": { “Column” -> [Data] } containing data that was inserted into destination (JSON object).
      * "server": Name of server containing database and table (string).
      * "status": Indicates whether ETL was completed or not (string).
      * "starttime": Start date + time for ETL (string, datetime).
      * "endtime": End date + time for ETL (string, datetime).
    - Optional Arguments:
      * "preops": All pre operations completed (list, string).
      * "inputops": All input operations completed (list of strings, string).
      * "postops": All post operations completed (list of strings, string).
* **Adding New APIs:**
  + 1. Create new python module with name of API in the APIs folder.
  + 2. Add all functions you want to serve as controllers.
    - Can optionally add dependency injection by putting “@inject” decorator right above the function definition, and specifying which parameters you want injected with pattern “variablename : Type” in function definition (see ETLSummaryReportAPI for example).
  + 3. Register the new endpoint in the register\_endpoints() function in APIs\RegisterEndpoints.py.
    - Each item in endpoints list requires the following arguments in dictionary:
      * "func": Function object to use as controller (callable).
      * "name": Identifier for endpoint (string, must be unique).
      * "route" : URL route for endpoint (string, must start with backslash and be unique).
      * “methods” : REST methods to use for endpoint (list, must be one/more of ['POST', ‘GET’, ‘PUT’, ‘DELETE’]).
      * Optional arguments are:
        + "inject" : Put true if want to use dependency injection for controller. Function must use @inject decorator if True (boolean).

(Optional steps: if controller requires configurations)

* + 4. Define new app configuration section object for API in /Configs/ folder.
  + 5. Add this new app configuration section as property (using @property decorator) to ValidatorConfig (look for “# properties”).
    - **NOTE:** The underlying property must be public (‘does not start with “\_\_”’) and must have same name as both the string section name in .json file and as object.
  + 6. In APIArgs.json (or whichever file is used), add new configuration as section in “APIs” section with { “APIName” -> { “Config” -> “Val” } }. Name must match configuration section object name, case sensitive.
  + 7. Import new app configuration object in AllConfigs.py.