infoSecur

Documentation and User Guide

8/7/2021

Table of Contents

User Guide

Introduction

infoVia offers a unique and innovative approach to efficiently and effectively providing appropriate data to those demanding speed and security simultaneously. Using technologies already evident and readily available within organizations of all sizes, ***infoSecur*** provides a fresh approach applicable to all industries which solves many of the challenges of delivering data to those who need to know with controls at the cell level directly from the existing database platform.

Having been implemented in industries ranging from manufacturing, health care, services, education, insurance, and government, ***infoSecur*** leverages leading practices developed in ultra-secure environments like global high-tech industries & the United States Department of Defense.

infoSecur uses basic relational database capabilities and ANSI-SQL. No fancy technology is necessary, and a basic SQL skill set is all that is required for personnel to implement or select from the platform. Coding complexity is reduced 50-90% for applications, reports, and dashboards downstream. ***infoSecur*** is platform-agnostic, easily portable, and even works on "big data" technologies with federation and schema-on-read tools.

infoSecur is easy to use…and fast to learn and implement. Often, secure objects are available to users within hours of initial set-up. Development staff needs only basic SQL skills. Plus, with minimal added technical overhead, it is both scalable and performant – often lowering query times significantly. Your users will love it.

Capabilities

Automatic Query Filtering at the Cell Level

By defining classifications of data and roles for users, ***infoSecur*** provides the means to seamlessly present data to all users in an organization, both internally and externally, where all data is filtered by consistent policies and presented with "cell-level" security (meaning both rows and columns are filtered per defined policies). Downstream applications, reports, or models automatically benefit from the consistent, streamlined provisioning approach.

Multi-Tenancy

Housing data for multiple tenants, or data consumers, within a single system and schema creates economies of scale but presents security challenges. ***infoSecur*** allows a seamless way to provide this data with the confidence of filtering across multiple types of consumers. A user from one organization can never see another organization's data without being explicitly granted access. Yet a single, consistent SQL query interface is

provided for all users. Even external user can query through dashboards or reports knowing their data is securely held and provided.

Masking and Obfuscation

By leveraging basic ANSI-SQL capabilities, data columns are masked at query time based on simply-maintained policies. Never will PII or Financial information or other sensitive data make it to users unless that need is defined and explicitly granted.

Portals

Combining capabilities into a solution for providing data external to the organization consistently and safely is a key value proposition. ***infoSecur*** is leveraged by developers to make sensitive data easily and securely available to end users that may reside outside the client domain, thus reducing the sets of data being passed between organizations in hard or soft copies by orders of magnitude. No more custom reports delivered by mail, fax, or email.

Peace of Mind

User access is easy. There are no new technologies or architecture layers to learn and control. Your users will want to interface with ***infoSecur*** data and adoption tends to grow organically.

infoSecur is easy to administer. Excess data governance bureaucracy is eliminated. Data provisioning policies are defined at the enterprise level and maintained in a consistent and efficient manner - typically by a small, part-time administration staff using any of the three interface options – SQL, GUI, or API.

Best of all, ***infoSecur*** helps insure that data sensitivity policies are managed centrally and consistently applied by all downstream consumers of data. Breaches, and the high stress and costs thereof, are of lower risk, so administrators can sleep at night knowing their organization's sensitive data is protected.

Definitions

Term	Definition
User	Identity of a connection actor, usually a specific, named user or persona
Role	Role played by a user for reuse. Many users can associate to a role, and a user may have multiple roles simultaneously.
Category	A contextual item depicting sensitive data, usually associated to a column or columns in the database
Category Value	A domain value relating to the Category. The set of Category Values exist to populate drop-down lists in the User Interface during the maintenance of the Class.
Class	A set of Category/Value pairs that make up a Policy grouping
Class Element	Each pair of Category/Value elements associated to the Class
Policy	Role Policy (typical scenario): Class or Classes assigned to a Role User Policy (exception scenario): Class or Classes assigned to a User

DDL/Table Definitions

infoSecur Base Objects

DB

- Sagum

Tables

- **SecurCategory** – Defines the set of Category concepts. Categories are the context concepts that make your data sensitive. Categories are the building blocks of infoSecur Policies.
- **SecurCategoryValue** – Defines the distinct domain values for each of the Categories.
- **SecurClass** –a distinct set of Category/Value combinations that together make a specific policy when assigned to Users and/or Roles. It allows for modularization and reuse of policy definitions across one or more Users and/or Roles.
- **SecurUserRole** – Associates a User to a Role or Roles, defining the intermediate step in the User->Role->Class assignment trio.
- **SecurRolePolicy** – Associates a Class to a Role or Roles. This is the final step in creating a policy by completing the User->Role->Class assignment trio.
- **SecurUserClass** (optional) - if necessary and usually by exception only, allows specific Users to associate directly to a Class rather than through a

User->Role->Class assignment. It is normally recommended to instead use SecurRolePolicy to associate Roles to Classes and SecurUserRole to associate Users to Roles but as an exception SecurUserClass provides a more direct approach for managing policies for certain Users without Role assignments being necessary.

- **SecurImpersonate** - If necessary, allows for one User to emulate the security policies of another user for a period of time and/or connection scope.
    - Global Impersonation – for all connections in the environment for the period of time defined, any connection for the Impersonating user will emulate the Policies of the Impersonated user
    - Local Impersonation – scope limited to the current connection only, the Impersonating user will emulate the Policies of the Impersonated user for only the duration of the current connection.
- **SecurDomain** – defines the set of metadata that relate together in a multi-Tenancy scenario. The default Domain value is '1' and usually is sufficient for most organizations until complexity or multi-tenancy dictates the need to introduce other Domain values of 2, 3, etc.

Audit Tables

Audit tables exist on platforms where change tracking is not inherently available on the platform. They are populated by triggers on the I/U/D transactions against their parent table. I.e., inserts/updates/deletes against SecurUserPolicy are logged to SecurUserPolicy_audit

- SecurUserPolicy_audit
- SecurUserRole_audit
- SecurRolePolicy_audit
- SecurClassElement_audit
- SecurImpersonate_audit

Procedures

Note: Procedures are not yet fully implemented. They are conceptually useful for clients who wish to leverage a SQL interface (instead of using the infoSecur API) to manage infoSecur metadata, but not through direct table inserts/updates.

Category/Class Definition

- insCategory (CategoryName varchar(1000))
- insCategoryValue (CategoryName varchar(1000), CategoryValue varchar(1000))

- addRuleToClass (ClassID char(50), CategoryName varchar(1000) , CategoryValue varchar(1000), SecurType varchar(50))
- removeRuleFromClass(ClassID char(50), CategoryName varchar(1000) , CategoryValue varchar(1000), SecurType varchar(50))

- grantPolicy (SecurURID nvarchar(128), SecurRoleOrUser char(1), ClassID c har(50), , SecurType varchar(50))
- revokePolicy (SecurURID nvarchar(128), SecurRoleOrUser char(1), ClassID char(50), SecurType varchar(50))

User/Role Maintenance

- insRole
- delRole
- grantUserRole (SecurUserID nvarchar(128), SecurRoleID nvarchar(128))
- revokeUserRole
-
- allowImpersonate – define what URID a given User is allowed to impersonate
- doImpersonate – cause a User to impersonate another for the duration of the session or until proc is called again

Views

- SecurUser
  - ***FROM [TABLEOWNER].[edw_SecurUserRoles]***
  - ***JOIN [TABLEOWNER].[edw_SecurRoleClass]***
  - ***ON edw_SecurUserRoles.Role = edw_SecurRoleClass.Role***
  - ***where***
  - ***edw_SecurUserRoles.Username = /*note: Strips off the domain. Remove if not necessary*/***
  - ***(SELECT SUBSTRING(SUSER_NAME(), CHARINDEX('\',SUSER_NAME())+1,LEN(SUSER_NAME())))***
- SecurUser_<Category>
  - Over SecurUser filtering for a specific category.
    - Note: There are likely 3-10 of these, 1 for each Category in an organization
- SecurImpersonate
  - If not using the native DBMS impersonation capability, writes a row to a temp or globally temporary table for the current connection. Update SecurUser to use the impersonated UserID instead of the default connection.
- SecurRoleClass (view)
  - Pivot or not??? Unpivoted in the form of Label/Data pairs is most flexible in that it can accommodate all type of categories without impacting the schema and can always be pivoted (perhaps virtually) if necessary.

User Interfaces

- Role definitions
- Role grants/revokes
- Category definitions
    - New Category
    - Add Value to Category
- Policy Definition
    - Class Definitions – creating reusable groupings of Category/Value combinations
    - Mapping Class(es) to Users and/or Roles

Security Interfaces

- Data Warehouse Extracts – [ODBC, JDBC, JSON, XML, CSV/Text]
    - Export all definitions and mappings
    - Export *changed* definitions and mappings
- Application Programmer Interfaces (APIs) – [REST]
    - For a User, return associated Classification Set (direct or indirect thru User's Role
    - For a User, return associated Role(s)
    - For a Role and Classification Set, return Authorization Flag (Boolean)
    - For a Role, Return all authorized Classification Sets
    - For a Classification Set, Return all authorized Roles

Auditing

- Classification Reports
- Authentication Reports
- Access Monitoring & Alerts
- History of edits to Roles, Classes, & Users
- History of Impersonation events

Objectives & Principles

Transparency

With Policies in metadata, you can see what you have, organize your policies, easily audit them, and even report on them. Plus, your users trust it when they can see it.

Cell-Level Control

Managing access doesn't have to be hard. Apply policies to both rows and columns simultaneously. The result: cell-level security that is easy to maintain.

Ubiquity

You can apply the metadata to your actual data to create data that is available on a need-to-know basis to any user. All your structured and semi-structured data can

be secured this way. Even entire documents in your specified repositories can be managed with *infoSecur* policies by leveraging tagging and folder structures.

Consistency
When users connect to data sources that implement *infoSecur*, you can be assured that they only see what they need to see, and no more. With *infoSecur* at the basis of all data access points, the all consumption points (applications, dashboards, reports, and APIs) are enforcing the same, consistent security policies.

Ease of Maintenance
With *infoSecur* policies in one place, managed centrally and defined consistently, the administration becomes easier. Less people involved helps ensure quality and consistency of access.

Performance & Scalability
When implemented correctly, queries can perform faster with *infoSecur* security than without. Plus the solution scales with your data platform so it is never a bottleneck. Your data is available at the speed of business!

Approach & Mindset

- Meta data driven is a mindset. Similar to code generation, policy enforcement can attain consistency and administration is easy with metadata.
- New way of doing things. Make a few number of sharable objects, Simplify the data provisioning with reusable objects.
- Need management support that it is the correct strategy and approach. Avoid falling back into the old mindset of managing secure data provisioning via set-based chaotic means.
- Think about infoSecur policies as the “master policies” that drive your enterprise and apply them accordingly, automating to the extent possible.

Enterprise Integration & Deployment

What other areas can we use this in our company? To ultimately achieve enterprise consistency, consider implementing *infoSecur* policies in your other data access points. Some examples include:

- Source Systems
- BI Tools
- Document Repositories
- Virtualization Technologies

- Big Data / Data Lake

Intellectual Property

1. The Licensee acknowledges and agrees that the Proprietary Rights in and to the Software and Documentation belong exclusively to the Company and Licensee shall not dispute such ownership.
2. The Licensee may not copy the Software or Documentation, in whole or in part, in any visual or machine readable form, except to the extent that such copying is necessary for the Licensee's own backup purposes.
3. The Licensee undertakes that it will:
    a. not reproduce, translate, reverse-engineer, adapt, vary or modify the Software, nor communicate the same to any person, except as permitted by this Agreement;
    b. not transfer, assign or grant a security interest in the Software, Documentation, or the Licensee's rights under this Agreement;
    c. properly maintain all copyright notices on the Software and Documenta tion; and
    d. notify the Company in writing immediately if it has knowledge of the existence of any circumstances in which any person(s) may have unauthorized knowledge, possession or use of the Software or Documentation.

Governance/Administration Guide

Policy Management

User & Role Management

Users and Roles maintain the basis of the user side of the infoSecur equation. Users define who can interact with data. Roles define types or hierarchies of Users.

Often an adopting organization already has user and role definitions defined. Some platforms leverage LDAP or Active Directory for these definitions. In that case, it may be preferrable to "reference" the existing definitions rather than redefining and maintaining them in the infoSecur Sagum schema. Perhaps a hybrid approach is beneficial as well. For these instances, infoSecur supports "virtualized references" to those other definitions. See "infoSecur Active Directory Integration" document for additional details.

Leverage the three (3) Sagum views below to virtualize User/Role information. Usually these virtualized references are implemented with a UNION or UNION ALL statement so that any additional domain values defined in the Sagum User or Role tables are also included.

- Sagum.vUser - optionally add virtual **User** domain values from another source query
- Sagum.vRole - optionally add virtual **Role** domain values from another source query
- Sagum.vUserRole - optionally add virtual **User/Role associations** from another source query

Category & Category Value Management

- Identify Categories
- Assign Values to Categories
- Virtualizing Category Values

Often an adopting organization already has Category and Value definitions defined. Similar to the virtualized reference case mentioned in the *User & Role Management* section, it may be preferrable to "reference" the existing definitions rather than redefining and maintaining them in the infoSecur Sagum schema. Perhaps a hybrid approach is beneficial as well. For these instances, infoSecur supports "virtualized references" to those other definitions of Category and Value definitions.

Leverage the two (2) Sagum views below to virtualize Category/Value information. Usually these virtualized references are implemented with a UNION or UNION ALL statement so that any additional domain values defined in the Sagum User or Role tables are also included.

- Sagum.vCategory - optionally add virtual **Category** domain values from another source query
- Sagum.vCategoryValue - optionally add virtual **CategoryValue** domain values from another source query

Class Management

- Class Definition
  - A Class is a grouping of Category/Value pairs that together make up a Policy when assigned to a User and/or Role.
- Class Purpose
  - Class Definitions serve as the building blocks or Policies. A Class contains all of the atomic Category/Value pairs to define a single, grouped set of matching criteria to identify a row or column in the data platform. Classes are reusable, and can be assigned to one or more Roles to complete the Policy definition.
- Singular Classes
  - The simplest Class contains only elements of a single Category. This is a Singular Class. In this case, the value of a single column can evaluate the applicability of the Class to the data row.
- Plural Classes
  - More complex Classes contain combinations of Category/Value pairs from multiple Category definitions. These are necessary if the value of multiple

columns must be evaluated simultaneously to determine the applicability of the Class to the data row.

Class Assignment (to Roles or Users)

- To Roles – One or more Classes can be assigned to a Role. This completes the policy definition. Any User associated to that Role is then allowed to see the data rows/columns evaluated to match the Class(es) that are associated to the Role.
- To Users (exception process) - In some exception scenarios, Role definions are unnecessary and/or redundant. It is then acceptable to assign one or more Classes directly to a User rather than indirectly through a Role association.

Policy Implementation

Architecture

Row-Level Policies

Singular Policies – Classes can be joined to a data set by a single column in the data table.

Plural Policies – Most policies are complex and require joining to the data set by multiple columns in the data table.

Column-Level Policies

Typical Scenarios

- Clear Text/Value – this least-restrictive policy allows the column value to be seen in it's unmasked state
- Masked Text/Value – a somewhat-restrictive policy allows the column value to be seen with masking of some of its original characters. For numeric values, masking can include rounding up or down or adding a random number within a range to the value.
- No Text/Value – the most-restrictive policy causes the column value to be eliminated from visibility and replaced with a null value in the result set

Atypical Scenarios

- Summarized Data Access – Using aggregation, some policies could be implemented which restrict the detailed, row-level access to data but allow for data to instead be displayed in aggregate instead to protect the identity of the population members while still reporting useful summarized data to defined Roles/Users.
- Population Threshold Enforcement – In order to further protect the identity of individuals within a population by discouraging disaggregate information identification (DII), a data set that is summarized (as defined above) may wish to

require elevated viewing access when a particular aggregated row of data represents a small number of individuals in the population. That small number is defined by a Population Threshold parameter, which may be globally applied or applied to a specific data set depending on the legal/compliance requirements.

Secure a single column

Create a CASE statement in the secur view

Example: Secure all columns in a table/object

Build object with all columns applicable to a single column-level policy

Assign a Token Keyword

Outer Join to the object where CurrentUserPolicy has the Token Keyword assigned

Plural Policies mixing Row- & Column-level security

IF (condition) then Mask/Obfuscate column

Installing *infoSecur*

Task	Script (if applicable)	Role	Time Required
Create DB and Sagum (or alternative) Schema		DBA	<10 minutes
Install infoSecur tables on Schema	Sagum InfoSecur <DB> <version>.sql	DBA or Admin	15 minutes
Install infoSecur base views and code	Sagum vViews <DB> <version>.sql	DBA or Admin	15 minutes
Seed Data	Sagum SeedData <DB> <version>.sql	DBA or Admin	15 minutes
Determine Client Custom Requirements	*See Methodology section*	Security Analyst	120 minutes
Create Initial Client Policies	UI, API, or direct Sagum Table inserts		60 minutes
Create Client Helper Views from Templates	Sagum infoSecur User View Templates <DB> <version>.sql	Developer or Admin	60 minutes
Create Client User Views from Templates	Sagum infoSecur User View Templates <DB> <version>.sql	Developer or Admin	120 minutes, then ongoing
Test Client User Views		Security Analyst, Developer, or Tester	As needed

Train Analyst/Admin team on Maintenance		All	100 minutes, then as needed

Methodology

1. Determine if multiple Domain subsets are required. Default to Domain=1
2. Determine What Data is Sensitive based on context items
3. Define context Items in business terms. These are the **Categories**
4. Collect or define the domain values for each context item. These are the **Category Values**.
5. Determine User domain values.
    a. If applicable, virtualize the user domain in Sagum.vUser view.
    b. Otherwise, add items to Sagum.SecurUser table.
6. Determine Client Roles
    a. If applicable, virtualize the user domain in Sagum.vRole view.
    b. Otherwise, add items to Sagum.SecurRole table.
7. Determine User Role mappings.
    a. If applicable, virtualize the user/role relationship in Sagum.vUserRole view.
    b. Otherwise, add items to Sagum.SecurUserRole table.
8. Create Classes by adding Category/Value combinations
    a. Singular classes are simplest. They require row or column restrictions based on one or more values of a **single** Category within a Class. (Example 1: Salesperson with Role X can view Sales for Region Y or Z. Example 2: Also, Salesperson with Role X can see Product A but not Product B.)
    b. Plural classes are more complex. Use them when a combination of Categories is necessary to create a Policy Class. (Example: Salesperson with Role X can view Sales for Region Y and only for Product A but in Region Z for only Product B.)
        i. Note: Plural classes require Pivot style helper views because comparison of multiple columns simultaneously within a row is necessary.
9. Determine Client Policies, i.e., Role to Class Mappings
    a. Use the infoSecur UI or API
    b. or, add items to Sagum.SecurPolicy table.
10. Test/Review the Policies in effect for users in varying roles
    a. `SELECT * FROM [DB].[Sagum].[vCurrentUserPolicy]`
    b. Impersonate or Log In as a different user to see impact on vCurrentUserPolicy results
    c. Manipulate the Class/Policy metadata to see the result impact
11. Determine the database objects that contain definitions with sensitive Category data.

12. Create User Secur Views. Leverage infoSecur templates to “join” infoSecur policies for each of the above database objects depending on the use case requirements

Development Guide

Use Cases / Scenarios

Grouped Policies

Policy’s may contain a single Category or many Categories, depending on their complexity. There are two GROUPED keywords, assigned to Class definitions in order to identify their complexity. They are Singular, Plural, and Range defined as:
Singular Policy – the simplest of policy definitions, Singular Policies contain Classes which only define a single named Category Name in their definition. For example, if Category called “Color” existed, then a Singular Category could specify one or more values for Color, i.e., Color=Red, White, Blue but no values for any Category other than Color.
Plural Policy - a more complex form of Policy, where knowledge of more than one Category is necessary in order to evaluate a Policy. For example, if a Category called “Color” and another called “Size” existed, then a Plural Category could specify one or more values for Color and Size in coordination, i.e., Color=Red, Size = XL, such that the Policy only applies to data rows “where Color = ‘Red’ and Size = ‘XL’ ”
Range Policy – Occasionally, the need to define data Classes that span a range of values are necessary. Instead of applying the client data value in a column with an equal (=) relationship, there is a “between” relationship in effect where a “min” and “max” value is specified evaluated inclusively. For example, for a Category called “Age”, a Range policy would include Age = 5 (min) and Age = 10 (max) so that the policy data filters only those data rows “where Age >= 5 and Age <= 10”.

Template Stubs

Sagum.ClassPivot

Sagum.RangePivot

UserSchema.RowFilter

```
--now filter rows
WHERE EXISTS (SELECT 1 FROM [IV_DW].[infoSecur].[vCurrentUserPolicy_
Singular] AS pol WHERE pol.MatchType = 'equal' and pol.CategoryName
= 'TxnType' and pol.CategoryValue = txn.TYPE)
--and remove any with exceptions
```

UserSchema.ColFilter

```
LEFT OUTER JOIN [IV_DW].[infoSecur].[vSalesTxnFINANCIAL_Singular] AS
 fin
ON txn.TRANS_DATE = fin.TRANS_DATE and txn.AMOUNT = fin.AMOUNT /*not
e, in this example these may not be unique due to the sample's lack
of a Primary Key*/
```

UserSchema.Deny

Exception

```
WHERE
NOT EXISTS (SELECT 1 FROM [IV_DW].[infoSecur].[vCurrentUserPolicy_Cl
assPivot] AS exc WHERE exc.TxnMatchType = 'equal' and exc.TxnType =
txn.TYPE and exc.Exception = 'Y')
```

Not equals

```
WHERE exc.TxnMatchType = 'not equal' and exc.TxnType <> txn.TYPE
```

Examples

```
--Build the relationship and define the switch keyword with
the vLookup view

--Examples:
--Staff/Student
--Parent(Guardian)/Child
--Org Hierarchy Supervisor/Subordinate
--Class/Student, Class/Teacher
--Club/Student, Club/Advisor
--Session/Course/Class(Section)/Staff(Teacher)/Student
--AcadCredential/Level


--Other scenario templates pending...

/*Eventual Hierarchy: (Eventual Dependent/Subordinate) */
--Org Hierarchy Supervisor to EventualSubordinate

/*Relationships that skip N levels: (Eventual Dependent/Subordinate) */
--All Rooms at a Site given: {Room, Facility, Location, Site}
--All Students enrolled in a Session or Course via a Class(Section) given:
{Session/Course/Class(Section)/Staff(Teacher)/Student}

/*GRANT or REVOKE schema or table to DB role or user*/
--may want to use infoSecur to manage GRANT or REVOKE
```

Integration

API – direct to Angular

SQL

Virtualization

ActiveDirectory/LDAP – see “Active Directory Integration”

Customization

infoSecur API and UI Setup Guide

V1.9 - 6/7/2021

Development Setup

- Install GIT
  - https://git-scm.com/download/win
- Create github account and get access to following repositories:
  - Backend API - https://github.com/info-via/InfoSecurApi.git

    - Frontend UI - https://github.com/info-via/info-secur.git
    - From your chosen directory
        - run 'git clone <git url from above>' to initially pull down the code to your local development environment.
        - Run 'git pull' to get updates pushed to the repository from other developers.
- Install Node
    - https://nodejs.org/en/download/
- Install .NET Core 5.0 SDK
    - https://dotnet.microsoft.com/download
- Install Angular Using Command Line
    - npm install -g @angular/cli
- Install Sql Server Express 2019
    - https://www.microsoft.com/en-us/sql-server/sql-server-downloads

Run Locally On Laptop/Desktop

Database

- Enable SQL Server Authentication in SSMS for your database server
- Create Database and add login with sysadmin role. User should use SQL Server Authentication.
- Run MsSqlDDL.sql script found inside the Backend API git repository. The script assumes it is running in the context of the database just created.

API

- Database Connection String
    - Create System Environment Variable named "INFOSECUR_CONNECTION" and set the value to your ODBC connection string.
        - Example
            - "Driver={ODBC Driver 17 for SQL Server};Server=MSI;Database=InfoSecure;UID=username;PWD=password;"
- Edit <InfoSecurAPI Directory>/Properties/launchSettings.json
    - Set DATABASE_TYPE to "MsSql"
    - Update DATABASE_NAME, SCHEMA, SECUR_SCHEMA, and SERVER with correct values.
    - DATABASE_ROLE and DATABASE_WAREHOUSE can be left blank.

```
"profiles": {
  "Development": {
    "commandName": "Project",
    "launchBrowser": true,
    "launchUrl": "infosecure",
    "applicationUrl": "https://localhost:5001;http://localhost:5000",
    "environmentVariables": {
      "ASPNETCORE_ENVIRONMENT": "Development",
      "CONNECTION_TYPE": "Odbc",
      "DATABASE_TYPE": "MsSql",
      "DATABASE_NAME": "InfoSecure",
      "SCHEMA": "Sagum",
      "SECUR_SCHEMA": "SagumSecur",
      "DATABASE_ROLE": "",
      "DATABASE_WAREHOUSE": "",
      "SERVER": "MSI"
    }
  },
```

- From the command line, in the application root directory, run:
  - 'dotnet run --launch-profile Development'

Front End UI

- In the application root directory, run:
  - 'npm install'
  - 'ng serve'
  - Browse to localhost:4200

Set Up Admin Role

- From the UI Home Page, click the "Register" button, and create a new user you will use to administer the app. This user will have rights to edit the impersonation tables and assign roles to new users.
- To give admin rights to this newly created user, run the final command against your database from the MsSqlDDL.sql script found in the InfoSecureApi code repository. Substitute <admin_username> with the username you just created. Example command for Sql Server:

```
INSERT INTO [SagumSecur].[SecurRegisteredUser_SecurRegisteredUserRole]
SELECT a.[ID], b.[ID] FROM [SagumSecur].[SecurRegisteredUser] a
FULL JOIN [SagumSecur].[SecurRegisteredUserRole] b on 1=1
WHERE a.Username = 'my_user_name' AND b.[Role] = 'admin'
;
```

- You will need to logout and log back into the app for the admin rights to apply in the UI.

Deploy and Run On Local IIS

Front End UI

- Assuming the frontend and backend projects are within the same directory, edit the "outputPath" setting inside angular.json to be "../InfoSecureApi/wwwroot"

```
"architect": {
  "build": {
    "builder": "@angular-devkit/build-angular:browser",
    "options": {
      "outputPath": "../InfoSecureApi/wwwroot",
```

- From the command line, in the application root directory run:
  - ‘ng build --prod’
  - This will publish the front build to the “outputPath” we specified previously
- At this point you should also be able to run both backend and frontend from the API by running ‘dotnet run’ and navigate to localhost:5000

API

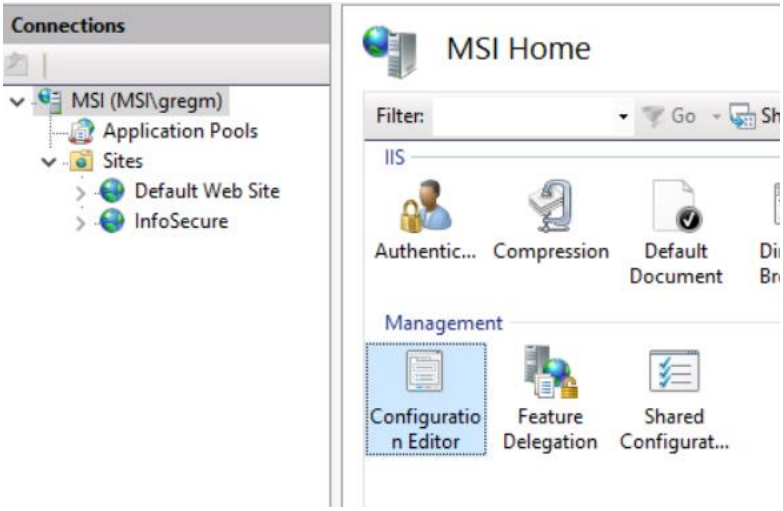
- From the command line, in the app root directory, run:
  - ‘dotnet publish --configuration Release’

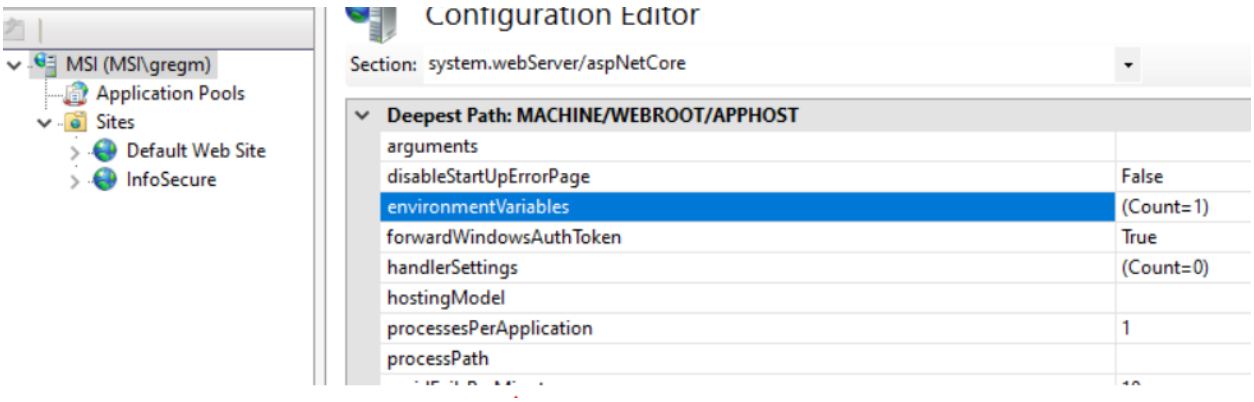
IIS

- Enable IIS on Windows 10:
  - Control Panel > Windows Features > Internet Information Services
    - Also enable Dynamic Content Compression under Internet Information Services > World Wide Web Services > Dynamic Content Compression
  - Press OK
- Download Dotnet core hosting bundle 5.0
  - https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/iis/?view=aspnetcore-5.0#install-the-net-core-hosting-bundle
- Open IIS Manager
- Left Click ‘Default Web Site’
- Click on ‘Basic Settings’ on right hand side to view Physical path
- Copy and paste the published files

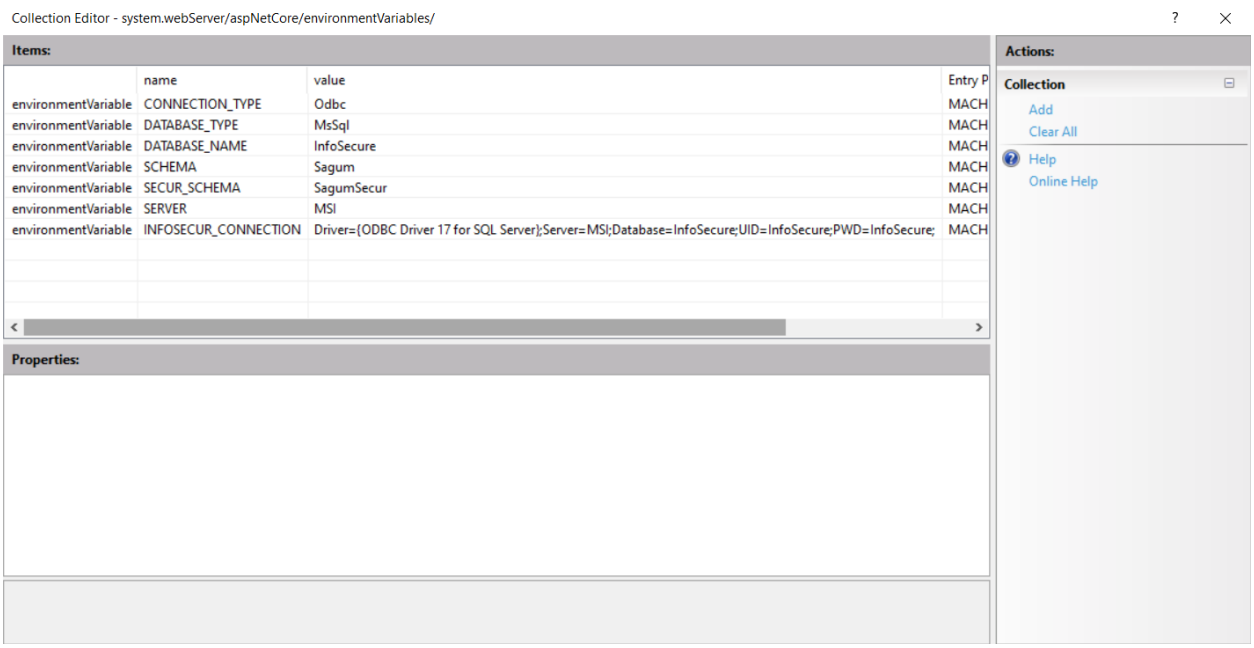
from InfoSecureApi/bin/Release/netcoreapp5.0/publish/ to the physical path identified in the ‘Basic Settings’ from IIS.

- Edit the environment variables by left clicking the server (top left) > Open Configuration Editor > environmentVariables





- Set the following environment variables in IIS. To edit environment variables, Left click the server (top left) > Open Configuration Editor > environmentVariables
    - “CONNECTION_TYPE”: “ODBC”
    - “DATABASE_TYPE”: “MsSql”
    - “DATABASE_NAME”: “<Your_Database_Name>”
    - “SCHEMA”: “Sagum”
    - “SECUR_SCHEMA”: “SagumSecur”
    - “SERVER”: “<Your_Server>”
    - “INFOSECUR_CONNECTION”: “<Your_Connection_String>”
- The database names and schema names may differ from the values shown above.



- Restart the server and the website
- Browse

SSL

- In open powershell as Admin
- Create self signed cert
    - New-SelfSignedCertificate -CertStoreLocation Cert:\LocalMachine\My -DnsName "localhost" -FriendlyName "LocalInfoSecurCert" -NotAfter (Get-Date).AddYears(1)
- Open certificate manager and install the newly created certificate as a trusted certificate
- In the IIS Manager
    - Add new website
    - Give it a name, select physical path
    - Set binding to https
    - Select newly created cert
    - Browse to site

Optimization

On very large data sets and instances where performance is of the essence, it may be necessary to tokenize classes by assigning the class value directly into the rows of the client data, rather than evaluating the matches between the Class and the client columns for each individual query execution. In that case, the user-facing views would join to the CurrentUserPolicy view by the ClassID instead of the individual class elements. Of course, any time the data row changes OR the Class definition changes, then a re-evaluation of the ClassID assignment for the row/row(s) needs executed immediately. This is triggered by the load process and/or the class maintenance trigger(s).

Support

The Client may notify the Company of a Fault at any time during the Support Hours by contacting the Company email: support@info-via.com

We are here to help you. Please contact us support@info-via.com

See "Support Services Agreement"