Requirements for myQOI

myQOI will play a crucial role in identifying households that are experiencing degraded internet quality. Since the beginning of Covid, remote learning has played a big role for all levels of education. All school entities were quick in finding technology to help remote learning, but household internet was not ready for the added usage.

myQOI will run network quality tests with Ookla and gather hardware information. With this information, it will allow entities discover who is experiencing degraded internet; thus, allowing the entities to act.

We currently have two Azure Functions with one API Gateway in Azure. The SQL DB is on premises with a hybrid connection to Azure. One Azure Function (QOI_Config) gets the schedule from the database and passes it along to the application. The second Azure Function (QOI_API) receives the data from the application and inserts it into the database. This adds a layer of protection for the database because the clients are not directing connecting to it. We are also using Key Vault and managed identities to secure the API key. This also allows us to easily refresh the API token since it lives in one place.

Functional Requirements
The application needs local System/Admin authorization
API intergration that will send schedule configuration settings.
The application needs to run in the background without user interaction
The application needs to gather hardware information that is listed under the WMI section
Integration with Ookla to run a network quality test and gather the information under the Ookla section
Secure API keys with Azure Key Vault
API Integration that will store the data sent in a database
The application needs to gather GPS coordinates
The application needs to be deployable with management systems like SCCM or Intune
The application should have a time randomizer before the network speed test

WMI Data
Local IP Address
Mac Addresss
Make
Model
Serial Number
Login User
DNS
Operating System

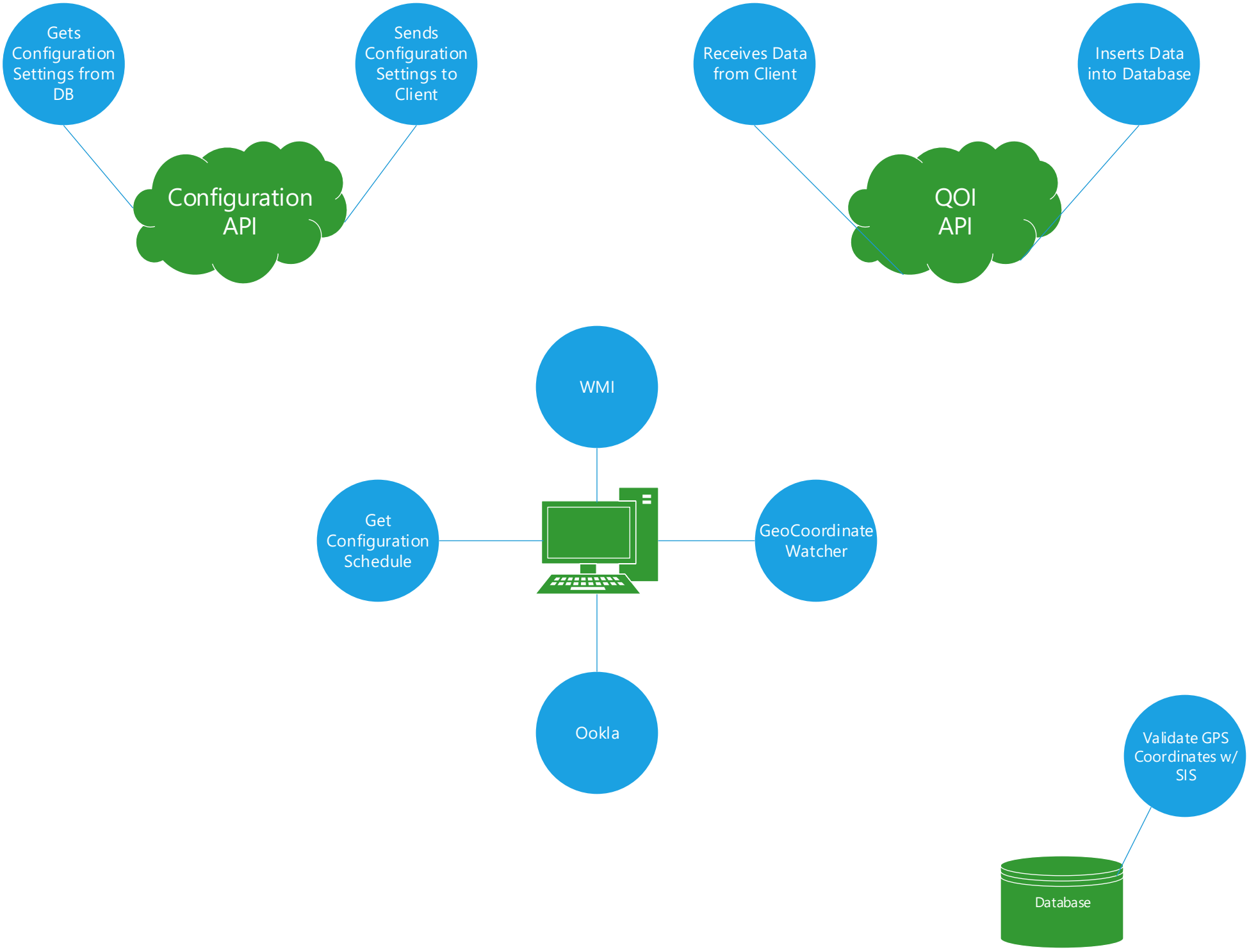
Ookla Data
Download Speed
Upload Speed
Latency
ISP
Public IP Address

GeoCoordinateWatcher Data
Latitude
Longitude

*Windows Management Instrumentation (WMI) is the infrastructure for management data and operations on Windows-based operating systems.

*Ookla is a global leader in providing network quality test. The Ookla's Speedtest service measures data throughput along with latency against a nearby server.

*GeoCoordinateWatcher is a Microsoft class that will provide GPS coordinates using two methods. The first method will try to use GPS if it's available on the device. The second method will use Wi-Fi Triangulation.


Gets Configuration Settings from DB
Sends Configuration Settings to Client
Configuration API
Receives Data from Client
Inserts Data into Database
QOI API
WMI
Get Configuration Schedule
GeoCoordinate Watcher
Ookla
Validate GPS Coordinates w/ SIS
Database

- **Get Configuration Schedule (Client)**

  Purpose – This feature will get the desired run time from QOI_Config API

  Behavior – The application will first need to get the API key from Azure Key Vault to get the configuration settings from the API. It will then store the configuration settings in a local configuration file.

  Case 1 – ***The application can communicate with Azure to get the API key***

  This is the desired scenario because it can successfully communicate with Azure

  Case 2 – ***The application is not able to get the API to get configuration setting***

  In this case, the application should still attempt to run a network speed test with Ookla. The fail attempt with Azure could be permission issues, or issues with Microsoft Services.

  The application should store the data locally until the next successful attempt to get the API key.

- **Perform Network Speed Test with Ookla (Client)**

  Purpose – This feature will find the nearest server and run a speed test against it

  Behavior - The application will get a list of the nearest servers and then determine which server has the lowest latency. Once the application determines the best server, it will then perform the speed test.

  Case 1 – ***The application can perform a network speed test***

  In this case, we will get our desired data. Ookla will send us our public IP address, internet service provider (ISP), download speed, upload speed, and latency.

  Case 2 – ***The application does not have internet access***

  In this case, the application will not be able to run a network speed test and gather the necessary data.

  Case 3 - ***Ookla is not available***

  This scenario will be rare since Ookla has thousands of servers that are available to perform a speed test. This could skew our data if the nearest available server is further away

- ## Get Coordinates with GeoCoordinateWatcher (Client)

  Purpose – This feature will use the GeoCoordinateWatcher class that is available in Windows

  Behavior – The application will first store the state of 4 registry keys: [insert reg keys]. Then set the registry keys to allow GeoCoordinateWatcher access to the coordinates.

  Case 1 – *GeoCoordinateWatcher has permissions to obtain the coordinates using onboard GPS*

  GeoCoordinateWatcher will get the coordinates using onboard GPS if it's available. This is the preferred method and most accurate

  Case 2 – *GeoCoordinateWatcher has permission to obtain the coordinates with Wi-Fi triangulation*

  GeoCoordinateWatcher will use the second method and obtain coordinates using Wi-Fi triangulation. This method will be less accurate than the onboard GPS method. In this scenario, we could verify the accuracy using a student information system and compare the address that is on file.

  Case 3 – *GeoCoordinateWatcher doesn't have permissions to obtain coordinates*

  This scenario will occur if GeoCoordinateWatcher doesn't have the appropriate registry permissions. The program will then pass on null values to the database.

- ## Get hardware and user information using WMI (Client)

  Purpose – This feature will use Microsoft's built in Windows Management Instrumentation to obtain hardware and user information.

  Behavior – The application will be using three WMI classes. The first one is Win32_ComputerSystem. This class will allow us to gather the Manufacturer, Model, Username, and DNS hostname. The second WMI class is Win32_BIOS; this will allow us to gather the Serial Number. The final WMI class we use is Win32_OperatingSystem. With this class we can gather Operating System and Operating System Architecture.

  Case 1 – *The application can obtain the hardware and user information*

  The process will query the WMI classes listen above and store the data into an object.

  Case 2 – *The application can't obtain the hardware and user information*

  This case should be rare because these are built in WMI classes and Microsoft uses them. If the application can't get the information than it will pass on null values. We can have a process that searches the database for null values and troubleshoot the issue. One scenario I've seen is that custom made computers could have null values in WMI classes.

- **Get Configuration Settings from Database (Backend)**

  Purpose – This process in the QOI_Config will obtain configuration settings from the specified database.

  Behavior – The QOI_Config API is expecting to receive a GET call from the clients. This will trigger the API to obtain the desired run time.

  Case 1 – ***The API is available and responds with the desired data***

  Case 2 – ***The API is available, but not able to read from the database***

  Case 3 – ***The API is not available***

- **Send Configuration Settings to Client (Backend)**

  Purpose – This feature will keep clients up to date with desired schedule settings

  Behavior – This API will return the desired scheduled run time to the client. The client will then have a process to update its local configuration file.

  Case 1 – ***The API is available and responds with the desired data***

  Case 2 – ***The API is available, but not able to send the data to the client***

  Case 3 – ***The API is not available***

- **Receive Client Data from Client to API (Backend)**

  Purpose – The QOI API will accept the client data and prepare it.

  Behavior – The API will be expecting the following data: IP Address, Mac Address, Make, Model, Serial Number, Download Speed, Upload Speed, Latency, Time of Execution, User, DNS (Hostname), Operating System, Operating System Architecture, Internet Service Provider (ISP), Public IP, Network Interface Type, Latitude, and Longitude.

  Case 1 – ***The API is available***

  In this scenario, the client can send the API Client Data. The API will then process it and continue with the next process. The next process will insert the data into the database.

  Case 2 – ***The API received unexpected/missing data***

  This scenario can occur if the client has an outdated version that is not sending the complete data set. To help avoid this, we can create versions within the API Management or make sure all clients are up to date before updating the API.

Case 3 – *The API is not available*

Since the API is on Azure, it should have an 99.9% availability. Where the API might not be available is when the content filter that is running on the district might block it. One of the requirements would be to allow Azure or AWS resources access on the district network.

- ## Send Client Data to Database from the API (Backend)

Purpose – This process will use the client data it received and insert the data into the database

Behavior – The API is expecting to have the SQL connection setting store on the application configuration in Azure. This will help secure the connection to the database because only the API will have access to it.

Case 1 – *The API successfully inserts the client data into the database*

The API will use the System.Data.SQLClient namespace to create a connection to the specified database. The data parameters are added using the built in "AddwithValue" method to prevent SQL injection. The API will then execute the SQL statement and insert the client data.

Case 2 - ***The API cannot access the database***

If the API cannot access the database, it can be for a couple reasons. If the database is in Azure, we must ensure that the database is allowed to communicate with other Azure resources.

If the database is on premises, we must ensure that the hybrid connection with Azure is established and working.

Case 3 – *SQL Connection Settings is not available or changed*

If the SQL DNS or authentication is modified, we have to also update the SQL Connection Settings in the Azure Function Application Settings.