NDSU Microsoft Capstone 2020 Web API

**Index**

1. **Overview**
2. **Background**
3. **Services**
4. **Controllers**
5. **Classes**
6. **Authentication**

**Overview**

The goal of the Microsoft capstone was to create a web app that could host analytics for NDSU’s FarmBeats kits. TO do this we created a Web App, PowerBI reports, and a Web API. This is the documentation for how to use the Web API as well as some background and some information I gathered while working on this Web API. If you are looking for documentation on how my requests work skip ahead to the services section. If you want to know how to call certain http requests skip ahead to the controller section.

**Background** This section is dedicated to the process it took me to getting the Web API to work and includes several of the resources I used. In this section I will go over: first the research preview, then the public preview, how the two previews interact, how the azure is setup, and finally the resources I used. For context if you don’t know this project was split into two parts the first part was before NDSU got access to FarmBeats Public Preview and the second was us working with the public preview.

**The Research Preview:** The research preview is probably the most confusing of all the parts to FarmBeats. This is where all of your data will be stored. It will be stored in a NoSQL database in an Azure Table Storage called ndsufarmbeatsstorage. It can be found in the Azure portal1 under the ndsupafarmbeatsrg resource group. However I do not recommend using the web portal for looking at the data because several tables will not show up. If you need to look at the data directly download the azure storage app2 and you should be able to find all the tables there. The data for each sensor box will get its own table called AgSensorBox(ID). To be clear the id will be the id with the format F#### it is not the skuid that is a different id. To find out what each channel carrelates to for data check the table AgSensorBoxSKU(SKUID). This will correlate each sensor boxes channels with the corresponding data. For example on box 2019 channel 0 is the BME280 Temperature. In order to get my API to work with the research preview I did have to add 2 tables one to store a list of all sensors so I could query the correct tables, and one to list the geolocation of each sensor for our PowerBI maps. These have since been deleted. The final thing I wanted to talk about for the research preview is if you need to query the tables directly, be careful because azure table storage has been mostly replaced by CosmosDB servers. Most of the commands work with azure table storage but you do not need to connect to a CosmosDB server you can set your connection string directly to the tables you want to query. There should also be included a research preview Web API included in the Github. Fair warning however the output is not properly mapped with the channels and what type of data it is in the included research preview Web API.

**The Public Preview:** The Public Preview is far more user friendly than the research preview since the data access point in the public preview is a Web API3. There is also a website4 included in the public preview to manage sensors. Our Web App is separate and is mostly used for analytics purposes. The Web API has swagger documentation which is fairly thorough and you can reference here3.For our purposes we only care about the get calls for sensor and sensor model, and the post request for telemetry. The get request for sensor is useful for it allows us to gather a list of all sensors so that we can query all of their data later. The sensor model lets us know which types of sensors we have on each sensor so that when new sensors are added you don’t have to change the backend code. And the telemetry request returns two lists one filled with the times and one filled with the data. I then take this data and convert it into list of entries of all the data a sensor collects every set time interval.

**How they work together:** I wanted to include this section to let the user know that the public preview itself doesn’t store any data. All it does its take the data from the research preview and then transforms it through the FarmBeats Web API. There are also scripts to add new sensor models and setup scripts for new sensors.

**How the Azure is set up:** For my Web API to work I had to do some set up with Azure. This is important since all of our authorization goes through Microsoft’s Azure Tenant system. The Public Previews Web API by default had authentication set up for any NDSU user in the ad.ndsu.edu tenant. In order to get access tokens to this Web API I had to register it in azures active directory and then point the Public Previews Web API to the application id of the registration. This means in order to get access to the Public Preview Web API you need to get the application ID of the registration not the application id of the Web API itself.

**Resources:**

1. Portal.azure.com
2. <https://azure.microsoft.com/en-us/features/storage-explorer/>
3. [https://ndsufarmbeats-api.azurewebsites.net/swagger](https://nam06.safelinks.protection.outlook.com/?url=https%3A%2F%2Fndsufarmbeats-api.azurewebsites.net%2Fswagger&data=02%7C01%7Cdfroslie%40microsoft.com%7Ce08f8d587bb545f5450508d7dc0c6a08%7C72f988bf86f141af91ab2d7cd011db47%7C1%7C0%7C637219818207296536&sdata=DjdPHmEAv8M6KUh02wPA96kq9BIHn0IC2Kkpw9ruvWk%3D&reserved=0)
4. [https://ndsufarmbeats.azurewebsites.net](https://nam06.safelinks.protection.outlook.com/?url=https%3A%2F%2Fndsufarmbeats.azurewebsites.net%2F&data=02%7C01%7Cdfroslie%40microsoft.com%7Ce08f8d587bb545f5450508d7dc0c6a08%7C72f988bf86f141af91ab2d7cd011db47%7C1%7C0%7C637219818207301533&sdata=7Cm%2BIeiLoIoHGFP8lO%2BckPKjGa%2FqXSFSQI9GR%2BdkEnY%3D&reserved=0)

**Services**

My services are broken down into several methods that work together to return all transformed data. All of my services require an authorization being passed as a string parameter. This is the just the access token generated by OpenId. The first method id the GetSensor method which returns a list of all sensors that map onto the sensormapping class. I then use this list of sensors to query all the telemetry data for each sensor in the GetDataList method. Then I get all the times for the entries with the GetTimeList method. Finally I make all these calls and transform the data in the GetApiData method.

**GetSensor(string authtoken)**: This method takes in a string argument that is the access token it needs to make the post request to the FarmBeats Web API. I then then create a Post request and add an authorization header using the before mentioned token. I then execute the request and map the json response to the sensor mapping class by using the JsonSerializer classes Deserialize method. I then convert those new sensor mapping objects into a list of sensor objects and return that list.

**getDataList(string AuthToken, String id, string value):**This method takes in an authtoken to authenticate the request, the id of the sensor we are querying, and the datatype that we would like to query(ex. Ambient Temperature). It then generates a post request to the FarmBeats Web API’s telemetry request. Then it converts the json response into an object of the data mapping class. It then creates a list of type double that it stores all of the data of that sensor channel and returns it.

**getTimeList(string AuthToken, String id):** This method takes in an authtoken to authenticate the request and the id of the sensor we are querying. It then generates a post request to the FarmBeats Web API’s telemetry request. Then it converts the json response into an object of the data mapping class. It then creates a list of type DateTime that it stores all of the time data of that sensor channel and returns it.

**GetApiData(string AuthToken):** This method takes in an authtoken to authenticate the requests. This service first calls GetSensor in order to generate a list of all the sensors set up. Then in a foreach loop that will run for every sensor in this list, it will generate sensor data entries. It does this by calling the getDataList for each datatype and then stores that in a local double list. I then loop for the amount of entries in the lists, and create a new Sensor object which will be a new data entry to add to the return list. Then I will populate the data in the sensor using the various lists and by copying over the sensor information from the current sensor generated by the GetSensor method. I then add this Sensor object to a list and return it.\

**UpdateLocation(string AuthToken, string SensorName, double longitude, double latitude):** This method takes in an authtoken to authorize the requests, a string sensorname to find whatever sensor’s location you are trying to change, and a longitude and latitude that are the new values for the updated location.

**Controllers**

Currently there are only two requests in the controller

**Get:** This is the generic get all data for the PowerBI reports only works if called from the web API web page directly, because otherwise you can’t sign in and you can’t generate an access token.

**Post{authtoken}:** This is for querying the data from any other site since you need an access token to query the FarmBeats Web API.

**Classes**

There are currently three classes

**DataMapping:** This is the class that the json response from the telemetry request from the FarmBeats Web API maps onto. It contains two lists one of all the timestamps for all of the sensor updates and one of all the values of those updates which is contained in a nested class called property.The property class contains three things: a string with the name of the values(ex. Soil temperature) called name, a string with the type of the values(ex. Double) called type, and a list of all the data called values.

**SensorMapping:** This is the class that the json response from the Sensor request from the FarmBeats Web API maps onto. It contains a list of objects of a nested class called item and is named items. The item class contains: a string for an id called id, a DateTime for when the sensor was set up called createdAt, a DateTime for when the sensor was last changed called lastModifiedAt, A string for the hardware id called hardwareId, a string for the models id called sensorModelId, a string with the device id called deviceId, a string with the sensors name called name, a string with a description called description, and objects created by nested classes: properties, and location, properties is empty and can be safely ignored, and location contains three values: float longitude which contains the longitude, float latitude which contains the latitude, and float elevation which contains the elevation.

**Sensor:** This class is what we transform the data into for the json output of the getapidata service. It includes all possible datatypes as well as: an int for a skuid, a string id for the id passes on from the FarmBeats Web API, a string for the name of the sensor, and a string for the model id so we can tell what layout the channels from the sensor are in.

**Authentication**

I used OpenID for my authentication by redirecting the user to an azure tenant login page. Whenever you open the webpage to query the Web API you have to sign in with you ad.ndsu.edu account. For websites making this call they have to use the post request and pass an authorization token.