GIS 590 Final Report

Data Bridge from the Sensor Management System

to a PostgreSQL database

with Web Mapping

Paul Schrum

7 May, 2018

Contents

[I. Background 1](#_Toc513389517)

[II. Problem and Solution 1](#_Toc513389518)

[III. Methods 1](#_Toc513389519)

[a. Analysis Step 1](#_Toc513389520)

[b. System Setup 2](#_Toc513389521)

[c. Data Development 3](#_Toc513389522)

[d. Geospatial Analysis 3](#_Toc513389523)

[e. Web service, map, and application development 3](#_Toc513389524)

[IV. Results 3](#_Toc513389525)

[V. Discussion 4](#_Toc513389526)

[VI. Future Research 6](#_Toc513389527)

[VII. Appendices 7](#_Toc513389528)

[a. Final Product 7](#_Toc513389529)

[b. Project Data and Supporting Files 8](#_Toc513389530)

[i. Local file structure 8](#_Toc513389531)

[ii. Geodatabase 9](#_Toc513389532)

[c. Table of Map Doc, Service Name, Function 9](#_Toc513389533)

[d. REST Services 10](#_Toc513389534)

[e. AGOL Project Folder 10](#_Toc513389535)

[f. Scripts 11](#_Toc513389536)

[g. SQL 11](#_Toc513389537)

[h. Passwords 11](#_Toc513389538)

[i. Contact Information 12](#_Toc513389539)

[j. References 12](#_Toc513389540)

[k. Acknowledgements 12](#_Toc513389541)

# Background

The US Army Corps of Engineers' [Field Research Facility](http://www.frf.usace.army.mil/) in Duck, North Carolina (FRF), maintains multiple sensor stations at various locations in the Atlantic Ocean. To keep track of important information about these sensors, FRF maintains a database with all essential information about stations, sensors, and service times in a MongoDB database.

In December 2017, FRF staff suggested that Mr. Schrum, for his Capstone Project, help FRF by developing a solution to a recently emerged challenge regarding their sensor database, the Sensor Management System (SMS).

# Problem and Solution

The Field Research Facility has a database of all their stations and the sensors on those stations. The database is implemented as [Document Database](https://en.wikipedia.org/wiki/Document-oriented_database)[[1]](#footnote-1) (MongoDB), which is a relatively new type of database technology. Since document database have a fundamentally different data architecture, performing queries on them requires new skillsets. Yet not many of FRF's staff possess this skillset at the required proficiency level.

But they do have high proficiency is using Structured Query Language (SQL) to analyze and explore databases. Thus FRF personnel would like to have this data available for running queries using the Structured Query Language (SQL), which requires bridging it to a Relational Database Management System (RDBMS).

As the solution to this problem, FRF asked Mr. Schrum to develop a system which can read a .json data dump from their SMS database (built upon MongoDB) and write that data to a PostgreSQL database for SQL-based analysis and for public facing Web Mapping.

# Methods

## Analysis Step

For this particular project, a major early step was to identify how the data would be moved out of MongoDB and moved into PostgreSQL.

For the first half of the semester, Mr. Schrum understood that his Python module would be opening the MongoDB database and reading it for content. In preparation for this, he installed MongoDB on his VCL. During a coordination phone call with the client, the contact person indicated that this was not what they were envisioning.

Rather, the client expected that their personnel would have their customized MongoDB database, SMS, open on their server, and they would perform a Mongo Data Dump, which generates a text file with a .json extension.

At several times there was confusion about which MongoDB collections (the equivalent of a RDBMS table) would be bridged to PostgreSQL. Ultimately the client determined that at this stage they only want two collections to be bridged, Stations and Gages.

The .json files (resulting from the data dump) were found not to be json entities, but rather each .json file contains multiple text rows, and each text row is itself a valid .json object. Based on this realization, reading and parsing the .json files proceeded.

Each type of .json file is bridged to a PostgreSQL table. In other words, the data in Stations.json is bridged to the PostgreSQL table Stations. Each field in the .json file is written to the field in PostgreSQL using the same name. The single exception to this is Mongo's field named '\_id'. When the attempt was made to create this field is PostgreSQL, it always changed the field name to 'f'. To work around this, the '\_id' filed in the .json file is named 'mongo\_id' in the PostgreSQL database.

The question arose as to what was the best approach is to open the PostgreSQL database and write to it while preserving the Geospatial integrity of the data. (SMS has the station and gage locations in fields 'lat' and 'lng'.) Using a Python PostgreSQl module was considered for this. For example, there is a package, Psycopg,[[2]](#footnote-2) which may be used for this.

Ultimately Mr. Schrum realized that Arc already has this problem taken care of. The key to this is that when we handle a table by working with an ArcMap Layer that has a PostgreSQL table as its Data Source, Arc takes care of properly interacting with PostgreSQL behind the scenes. So by working with Layers instead of the actual tables, we can have all code work through arcpy in general and Insert Cursors in particular.

## System Setup

* The VCL's IP address is 152.1.13.83.
* The http server is Internet Information Services. Only Default Web Site is running.
* No additional html files have been put into wwwroot.
* The Enterprise Geodatabase is PostgreSQL.
* All files from the student are found in C:\SMSbridge, including GIS590-Passwords.txt.
* The project is also maintained as a Github project at <https://github.com/PaulSchrum/SMSbridge2PostGreSQL>, although the directory structure is a little different. Passwords are not stored in this repository because it is open to the public. It is recommended that continued development by FRF take place by Paul transferring ownership of this repo to Mike Forte.
* The one and only map file is C:\SMSbridge\SRSmap.mxd.
* The URL for the REST Services is <http://152.1.13.83/arcgis/rest/services/FRFsensors/SRSmap/MapServer>.
* The ArcGISOnline folder is Schrum\_Capstone.
  + The Web Map is "FRF Sensors".
  + The Web Application is "FRF Sensors App".

## Data Development

All data comes from FRF personnel in the form of the .json files. The most recent delivery of that data is in C:\SMSbridge\SourceCode\Data\SMS\_json\_v20180426.

No preprocessing is required for the grader to execute the module. End users will be server administrators at FRF, and they will preprocess the data by dumping the MongoDB collections to .json files.

Data is populated into the tables by running the primary module, "C:\SMSbridge\SourceCode\BridgeMongo2PostGreSQL.py". Argument 1 is the full name of the Map Document containing a Stations layer and a Gages layer. Argument 2 is the full path to the directory containing the .json files.

## Geospatial Analysis

Due to the minimal nature of the data being used for development, only a toy GP Widget is provided. It simply buffers gages at a length specified by the user.

The tool runs, but it does not show the results on the map. Time did not allow for debugging this issue.

## Web service, map, and application development

The REST Service is <http://152.1.13.83/arcgis/rest/services/FRFsensors/SRSmap/MapServer>.

The Web Map is [FRF Sensors](http://ncsu.maps.arcgis.com/home/item.html?id=596005fc1c504af9a9ffd1d3e0ea1358).

The Web Application is [FRF Sensors App](http://ncsu.maps.arcgis.com/home/item.html?id=6225fb8de8b24899b5ebfe939b06356f).

No applications were developed to run through the web app. The primary deliverable of the project is an application which runs from the command line and is accessible only to FRF system administrators. The Python module is BridgeMongo2PostGreSQL.py. The code listing is provided in Appendix E, but it is also available on [my Github page](https://github.com/PaulSchrum/SMSbridge2PostGreSQL).

# Results

A well-tested Python module was developed which

1. allows an administrative user to populate (and refresh) the Stations and Gages tables from a simple command line call.
2. may be automated to run once per day without needing to be run by a human.
3. may be easily extended to work with other data types as desired.

A web map and web app were developed which show the stations and gages in a public facing web site. The public has the ability via the web app to download the data as a csv file. Together, these help FRF meet its mandate to make its data publicly available.

# Discussion

There are some key aspects of the module which should be pointed out.

The module does not create any new feature class tables or map layers. These must be created manually from a map document which has the PostgreSQL database connected as a data source. The layers are also created by hand. The layers must be named "Station"[[3]](#footnote-3) and "Gages" or the Python module will not find them and an exception will be raised. These layers must have the respective feature class tables set as their data sources. The only necessary fields are OBJECTID and @SHAPE, which are required for all feature classes anyway.

In accordance with the client's wish, the module does not seek to do any kind of incremental update. Rather, every time it is run, it deletes all rows from each feature class and rewrites all rows using an Insert Cursor. The module also deletes and re-creates all fields (attributes) except OBJECTID and @SHAPE.

There is a relationship between Gages and Stations in which Gages holds the Mongo\_id of the Station that it is associated with. But the relationship is not created in the database. This decision was made because such relationships are really only enforced when adding or deleting rows to the database, and we are assuming that MongoDB has already guaranteed referential integrity of that relationship when users work with data via the SMS application. This may be changed later if it becomes necessary.

There are a total of four required Modules which must exist together in the same directory:

* BridgeMongo2PostGreSQL.py
* frfObjectBase.py
* station.py
* gage.py

Only BridgeMongo2PostGreSQL.py should be run in operations. (The others may be run during testing and debugging.) The last three modules contain classes of the same name as the modules. Station and gage [inherit](https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)) from frfObjectBase. frfObjectBase should never be instantiated. The Object Inheritance Diagram is shown in Figure 1.

frfObjectBase

Gage

Station

Figure 1: Object Inheritance Diagram showing which classes inherit from frfObjectBase.

If subsequent developers at FRF wish to add additional feature class tables, they should create new modules with classes which inherit from frfObjectBase in the same pattern as is found in the station class.

Adding or removing fields in the bridge process (running the Python module) is done by adding them to the module-level variable, requiredFields, currently found on Line 10 of Station.py and Gage.py. The definition of requiredFields is shown in Listing 1.

1. requiredFields = [('mongo\_id', 'STRING', '31'),
2. ('stationId', 'STRING', '31'),
3. ('gageNumber', 'STRING', '31'),
4. ('gageName', 'STRING', '31'),
5. ('gageType', 'STRING', '63'),
6. ('lat', 'STRING', '31'),
7. ('lon', 'STRING', '31'),
8. ('owner', 'STRING', '63'),
9. ('serialNumber', 'STRING', '63'),
10. ('barCode', 'STRING', '31'),
11. ('manufacturer', 'STRING', '31'),
12. ('model', 'STRING', '31'),
13. ('firmwareVersion', 'STRING', '31'),
14. ('depth', 'STRING', '31'),
15. ('description', 'STRING', '255'),
16. ('createdAt', 'STRING', '31'),
17. ('createdBy', 'STRING', '31'),
18. ]

Listing 1: Portion of the definition of the requiredFields variable in Station.py.

The contents of the requiredFields variable dictate the field names, types, and lengths which will be added to the feature class table. If more or less are needed, they should be added or removed here.

Each row is a tuple (between the parentheses). For each tuple the first value if the name of the field. The second value is the type of the field (in SQL language). For STRING types, the third field is the string length of the field.

# Future Research

As directed by the client, only two tables are being populated by the current implementation, yet FRF has at least seven Collections which could be bridged to PostgreSQL using this module. Implementing the other five could be useful to carry out.

The client has indicated that the readings of the various gages are not currently included in SMS, and that that data is being curated at multiple locations within the facility. Yet this data is supposed to be downloadable for the public. Some of it is, but perhaps not all of it, likely none of it is available via a web map. An excellent future project would be to add all of the historic gage reading data to the website for public consumption. When this kind of data becomes part of the system, meaningful geoprocessing services may be devised and implemented.

Immediate Plans.

The understanding is that after the transition time in June, Field Research Facility personnel will be using the source code and instructions from this project to establish essentially the same system on their own on-premises server.

# Appendices

## Final Product

* URL: <http://ncsu.maps.arcgis.com/apps/webappviewer/index.html?id=6225fb8de8b24899b5ebfe939b06356f>
* Screen Shot

Figure 2 shows a screen shot of the Web Application created for this project.

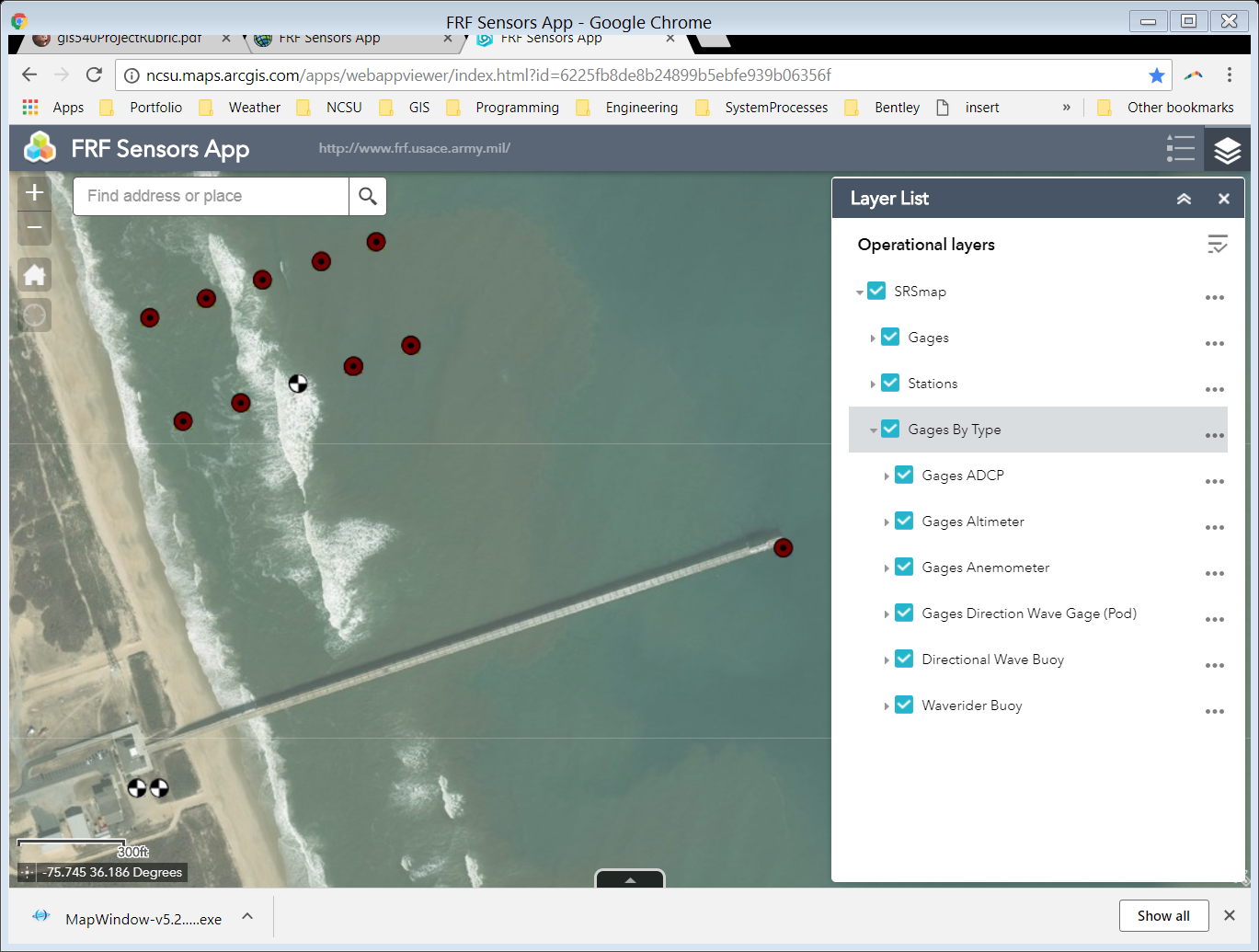


Figure 2: Screenshot showing the FRF Web Application

## Project Data and Supporting Files

### Local file structure

Figure 3 shows a screen capture of the expanded project folder in Windows Explorer.

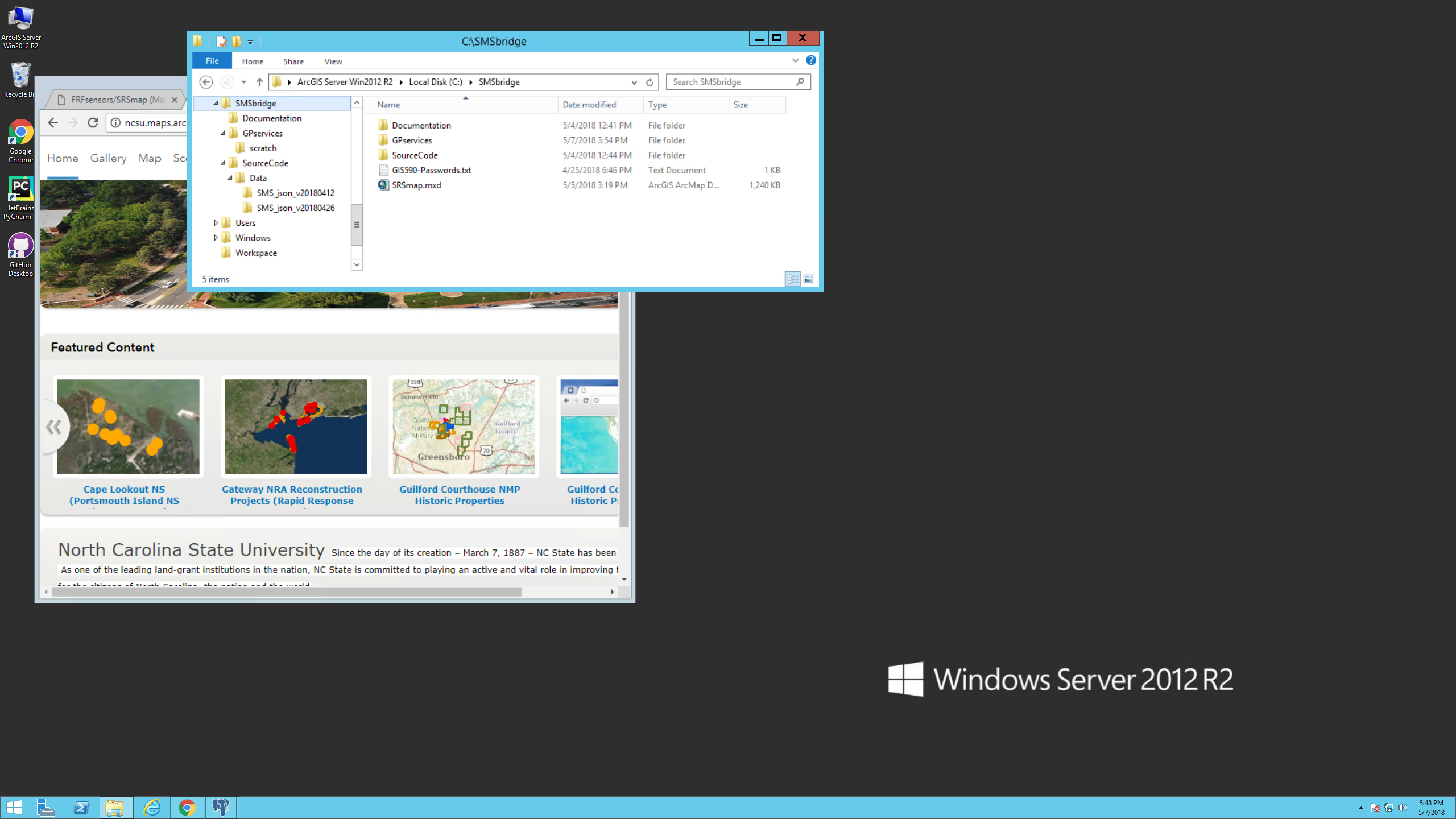


Figure 3: File Structure of the project on the server

Table 1 provides definitions for each element in the file structure.

Table 1: Description of each element in the file structure of Figure 3

|  |  |
| --- | --- |
| File or Directory | Description |
| SMSbridge | Root directory of all project files |
| GIS590-Passwords.txt | Text file containing information on all logins and their passwords. |
| SRSmap.mxd | The only map document used for this project. The REST Service is published from this file. |
| SourceCode | Directory containing all Python source code modules. (See Table 3, below.) |
| GPservices | Directory for all Geoprocessing services, including mxd files and a toolbox. |
| scratch | Directory for intermediate work to be saved by the process. |
| SMS\_json\_v20180412 | Folder containing all .json files from the client sent on 12 April 2018 |
| SMS\_json\_v20180426 | Folder containing all .json files from the client sent on 26 April 2018 |

### Geodatabase

Figure 4 shows a screen capture of the Enterprise Geodatabase using ArcCatalog.

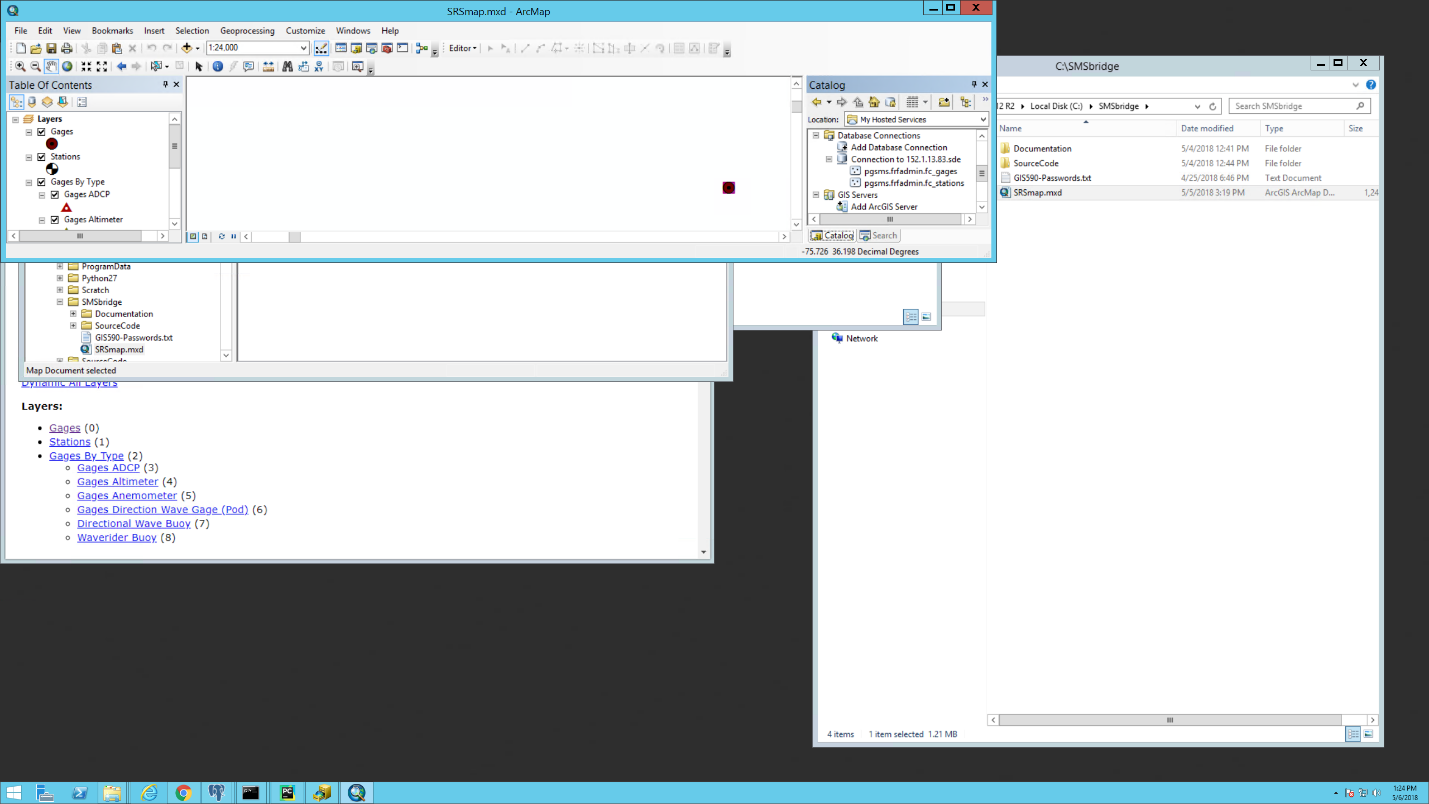


Figure 4: Screen capture showing the expanded Geodatabase.

All project data is found in two files: [stations.json](https://github.com/PaulSchrum/SMSbridge2PostGreSQL/blob/master/Data/SMS_json_v20180426/stations.json) and [gages.json](https://github.com/PaulSchrum/SMSbridge2PostGreSQL/blob/master/Data/SMS_json_v20180426/gages.json). The links are to the github repository version of the files. They also reside on the server at C:\SMSbridge\SourceCode\Data\SMS\_json\_v20180426.

The above files are what the bridge reads to repopulate the feature class tables. They are the results of MongoDB data dumps, and they are how the station and gage data comes to the project for processing.

The source code .py modules would also be considered supporting files. These are shown in Table 3.

Table 3: Summary of Python Modules and their descriptions.

|  |  |
| --- | --- |
| Python Module | Description |
| BridgeMongo2PostGreSQL.py | The primary module. (Run this.) All data bridging operations take place in this module. |
| frfObjectBase.py | Base class from which station.py and gage.py inherit. It contains executable code which is common to all other classes (so the code can be maintained in a single location.) |
| station.py | Contains the Station class, which is used by the bridge to read Stations from stations.json and write them to the stations feature class table. |
| gage.py | Contains the Gage class, which is used by the bridge to read Stations from gages.json and write them to the gages feature class table. |

## Table of Map Doc, Service Name, Function

Table 4 shows the map document and the service name, type, and function associated with the project.

Table 4: Listing of Map Document and associated Service information

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Map Document | Service Name | Service Type | Function | Model or Script |
| SRSmap.mxd | SRSmap | Web Service | Display layers for map | None |
| GPserviceMap.mxd | SRSMap | GP Service | Runs Buffer Gages | BufferGages |

## REST Services

Figure 5 shows a screen shot of the REST Service.

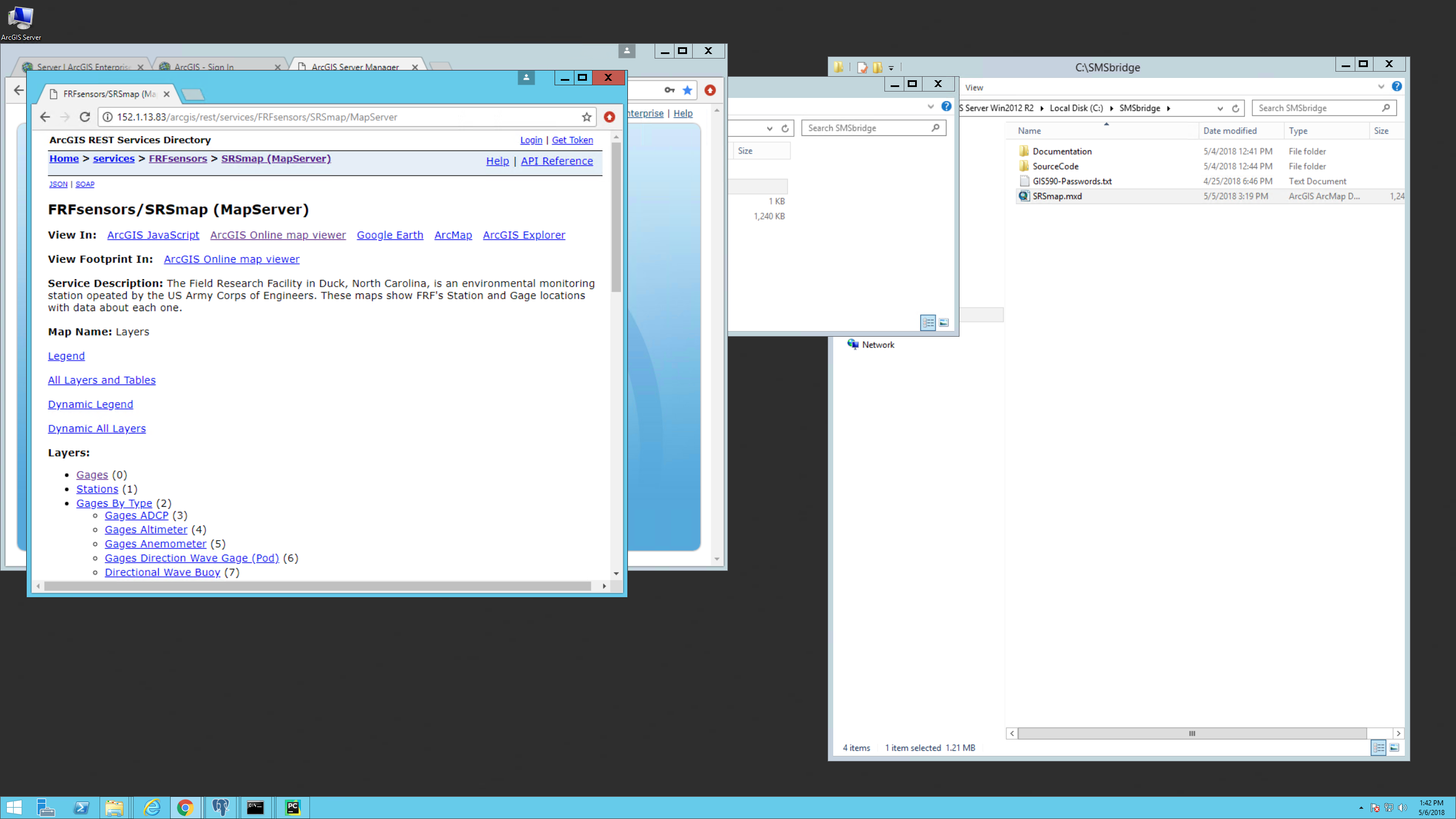


Figure 5: Screen capture showing the REST Service.

Figure 6 shows a screen shot of the BufferGages task.

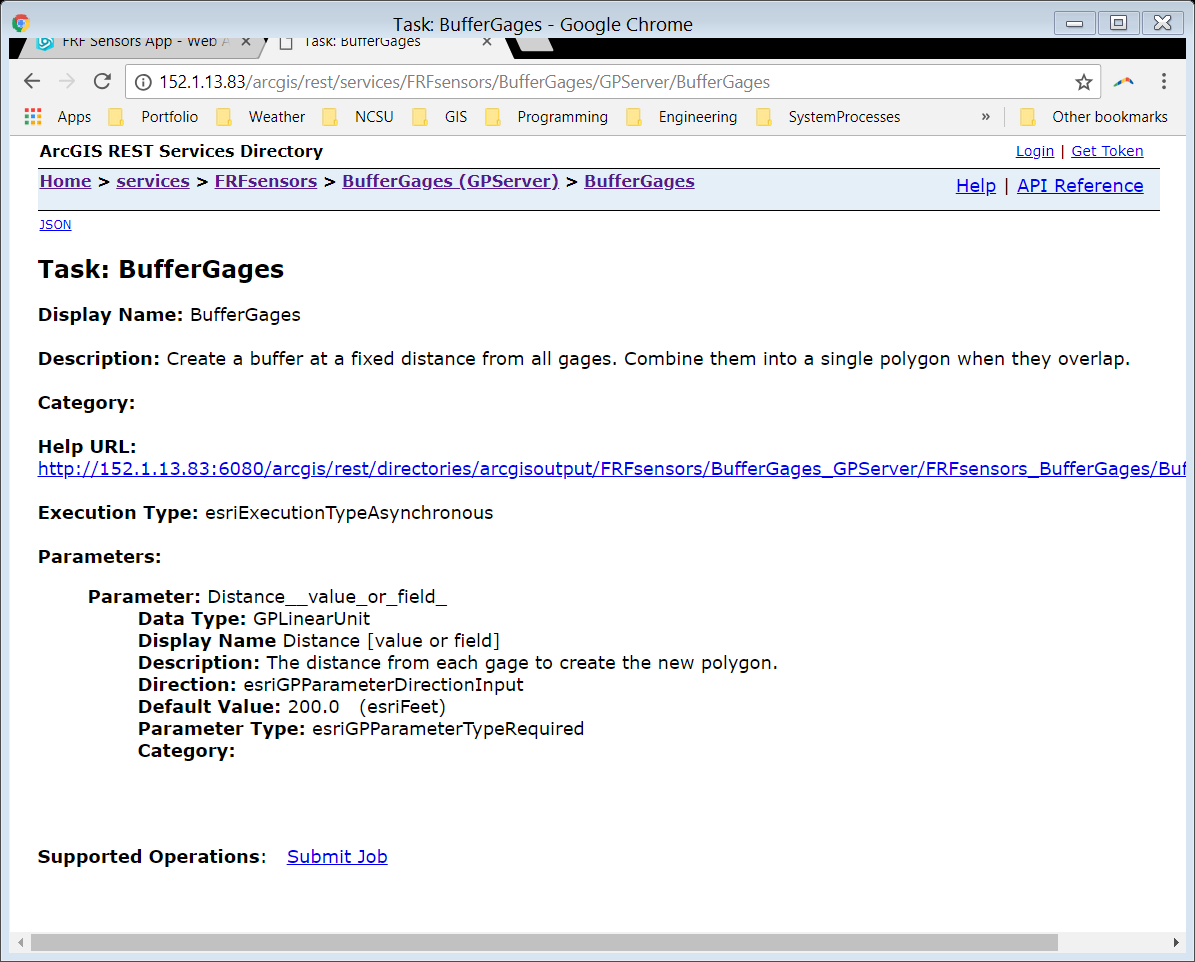


Figure 6: Screen capture showing the BufferGages Task GP Service

## AGOL Project Folder

Figure 7 shows a screen capture of ArcGIS Online project folder. Note there is nothing "beyond the obvious single web map/app."

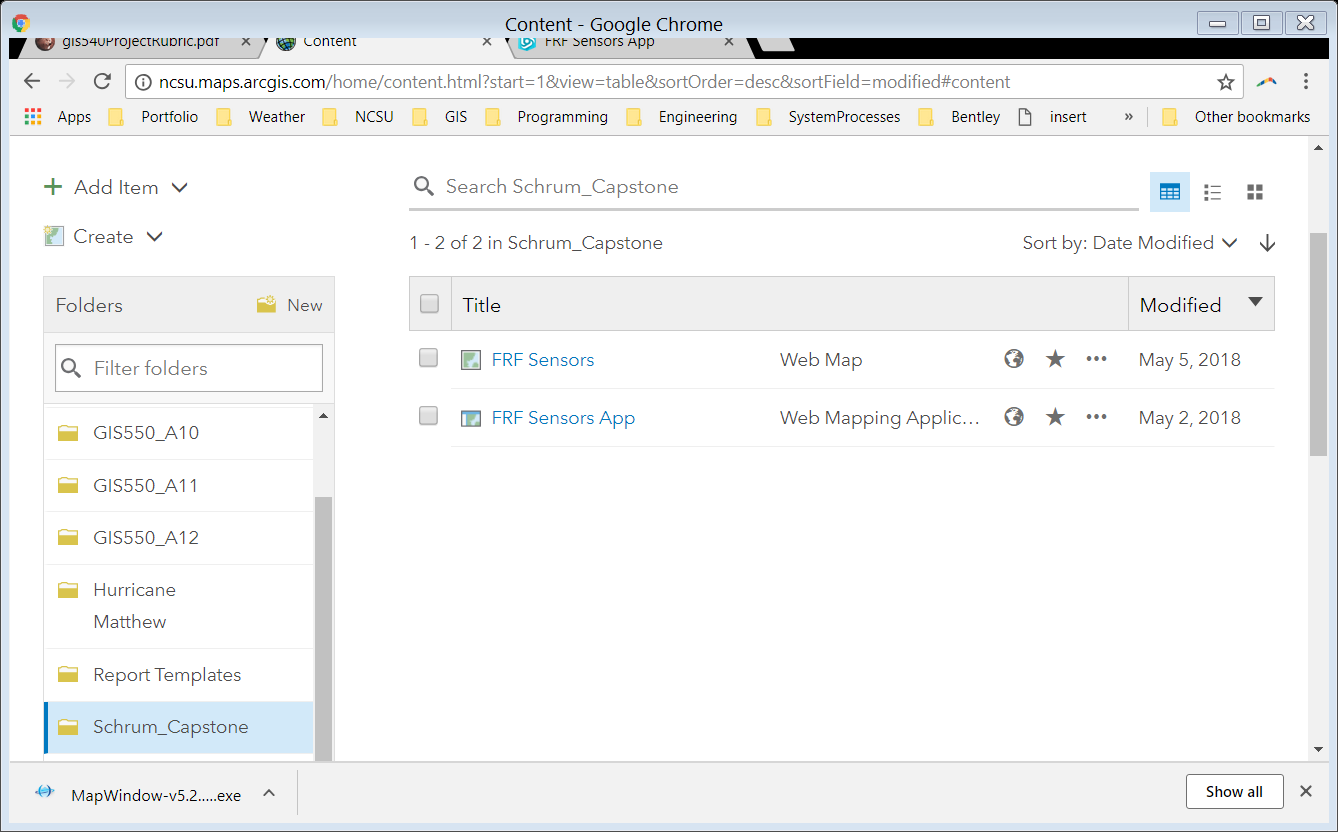


Figure 7: Screen capture showing the ArcGIS Online folder for the project.

## Scripts

There are no Models for this project. There is a single Python module intended to be executed at the command line by a system administrator while on the server. There are three other modules which are not directly executed, but are called by the primary module when it is run.

The scripts are long, so will not be posted here. They are available on the server at

C:\SMSbridge\SourceCode

They are also available at <https://github.com/PaulSchrum/SMSbridge2PostGreSQL>.

Note, the directory structure in the github repository differs slightly from that on the server.

## SQL

No SQL is used for this project. All populating of tables with values is handled in Python modules using Insert Cursors. This is also true for the operation to delete and add fields.

## Passwords

Passwords are not listed in this document as it is being saved in a public github repository.

The passwords file is available on the server at "C:\SMSbridge\GIS590-Passwords.txt".

This file has also been saved securly to Mr. Schrum's GDrive and is available to instructors upon request.

## Contact Information

Student:

Paul Schrum

[ptschrum@ncsu.edu](mailto:ptschrum@ncsu.edu)

919-745-9173

Client:

Field Research Facility

Contact: Michael Forte

[mffort@ncsu.edu](mailto:mffort@ncsu.edu)

252-261-6840 ext. 228

## References

All data was provided by the client through email communications.

The Rainfall Intensity layer is provided as a REST Service by the National Weather Service.

Server infrastructure development and deployment was conducted using methods originally described by Bill Slocumb, but some help from GIS 590 notes, and a lot of help from the TA, Juliana Quist, plus review of key lectures by Dr. James McCarter in the Fall 2016 semester of GIS 550. Thank you Julia.

## Acknowledgements

I (Paul) deeply thank Dr. Charlynne Smith and Juliana Quist for all the help they gave me this semester, Mike Forte of the Field Research Facility for working with me so well and patiently listening to me ramble on about stuff over the phone, and Dr. James McCarter for help he gave me this semester although I have been long gone from his GIS 550 class.

The user guide and the operations manual are embedded in the body of the report and have not been copied out to their own sections.

1. <https://en.wikipedia.org/wiki/Document-oriented_database>, accessed 5 May 2018. [↑](#footnote-ref-1)
2. <http://initd.org/psycopg/>, accessed 5 May 2018. [↑](#footnote-ref-2)
3. The layer named "Station" should be changed to "Stations". This must be done in both the Map Document as well as BridgeMongo2PostGreSQL.py. [↑](#footnote-ref-3)