AquaWatch Mobile Documentation

As of: May 11, 2024

0.0: Intro

Authors:

Blue Jelly: Ardin Kraja, Meryl Mizell, Erin Sorbella, Kenji Okura and Lulu Moquete
Data Divas: Lizi Imedashvili, Victor Lima, and Kenji Okura

Contact info:

Kenji Okura (okurakeng@gmail.com or kenj.o on Discord)

1.0 About the App:

This app should be able to educate users on water quality metrics, such as salinity, pH, and turbidity. These metrics are used to describe the “health” of local bodies of water, such as lakes, rivers, and ponds. Understanding these metrics can be helpful in allowing a user to make informed choices about performing recreational activities in that body of water.

Additionally, a core component of the app is being able to visualize these water quality metrics. The app is able to deliver monthly water quality reports, which consists of graphs of metrics such as salinity throughout a given month, and a “grade” known as WQIs on the water quality for that month. Furthermore, the user will be able to investigate that relationship between weather and water quality, seeing how weather conditions such as rainfall can impact water quality parameters such as turbidity through graphs.

Lastly, the app will contain features that contextualize the importance of water quality data. Such features include a list of animals that can be found at a chosen body of water, and how water quality can impact their living.

Our code is hosted publicly on GitHub, access the repository here.

How to run?

The steps to test the app are included in the README.md documentations. Please refer to them for more information.

2.0 Functional Requirements:

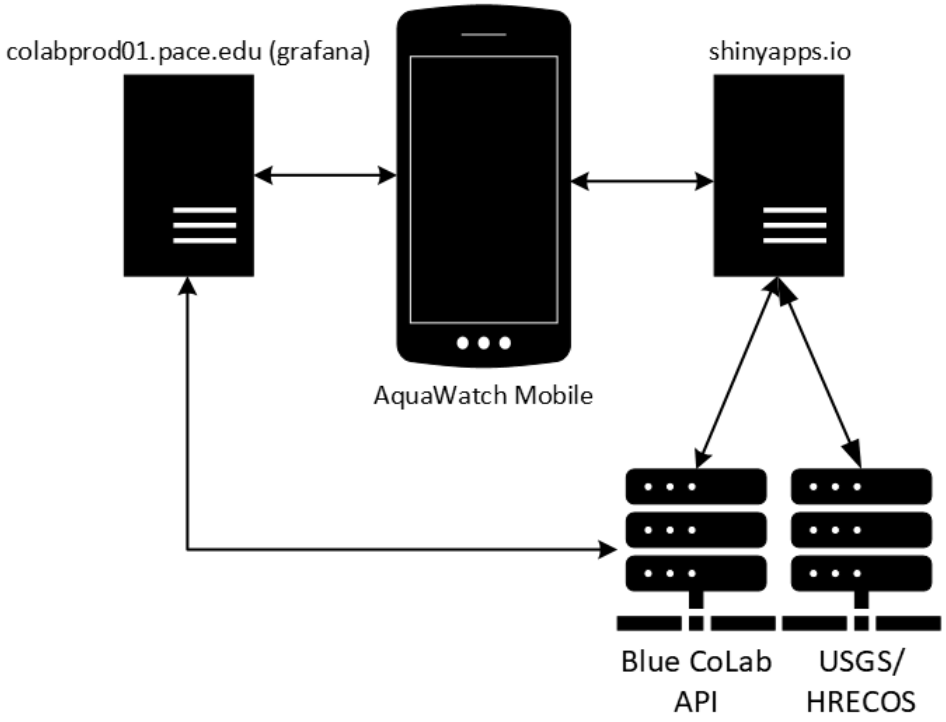


Figure 2-0: Block Diagram, showing basic interfaces.

The above diagram shows major components of our app and related infrastructure, that is:

(1) *AquaWatch Mobile App*: This is the app…
(2) *shinyapps.io*: This is the service used to host the data visualization server in Python.
(3) *Grafana*: This is a service used to display current water data. For more information, please approach the Blue CoLab WQI team.
(4) *Blue CoLab API & USGS/HRECOS APIs*: These are the APIs used to get water quality information. For more information, please view the documentation of each organization.

We will dive into more detail for each in later sections. In essence we try to describe what we did - but not how we did it.

2.1 AquaWatch Mobile App:

The following section will dive into detail about the app and its features. The exact code implementation can be found in section 3.1. Our goal with our section is to communicate what the app is supposed to do - not how to do it.

2.1.0 AquaWatch Mobile App Interface

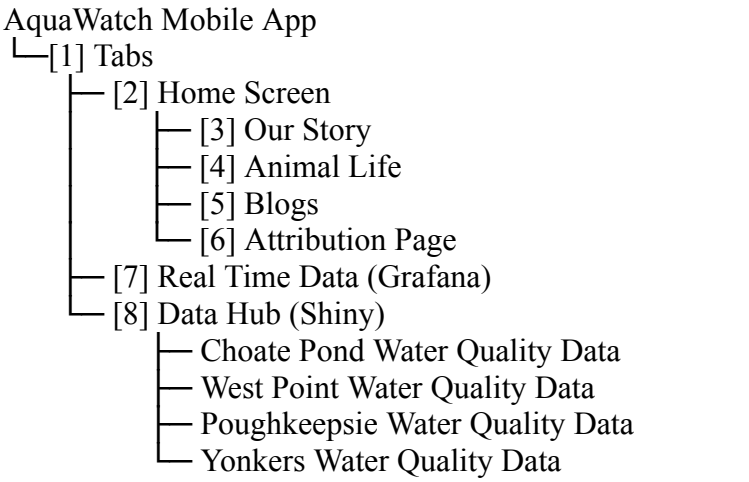


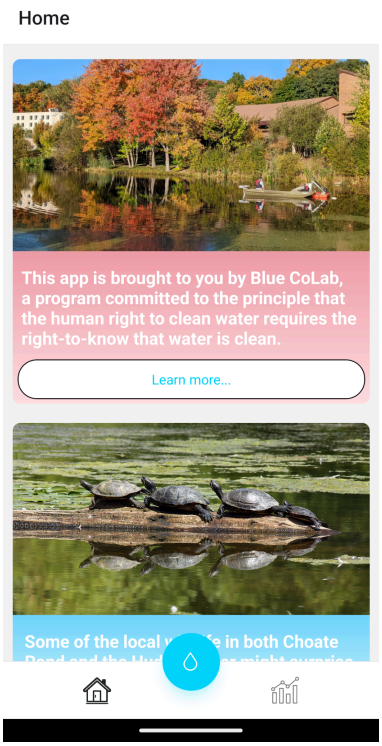
Figure 2-1: App Interface

The primary interface used by users is the AquaWatch Mobile app. The app is organized into subsections for (3) Our Story (4) Animal Life (5) Blog Page (6) Attributions (7) Attributions (8) Monthly Data.

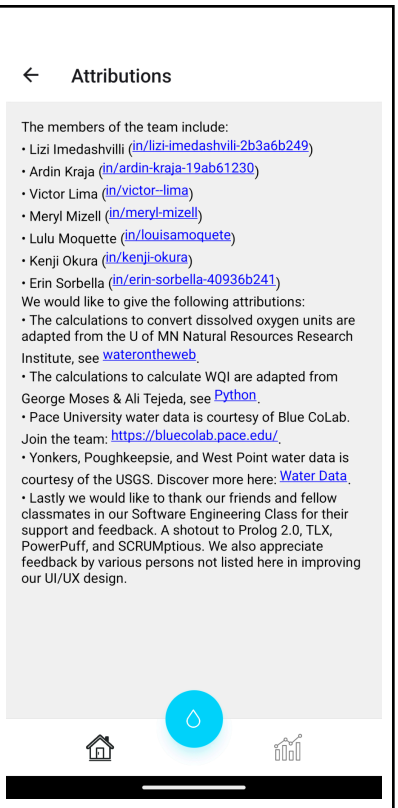
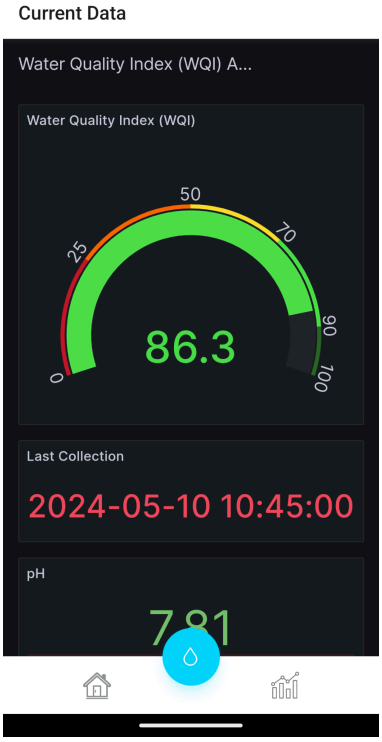
Within (8) 4 secondary interfaces via drop downs show different options on which water source to look at. 3 of these options are different points in the Hudson River while one of them looks at Choate Pond on the Pace University campus. See section 2.2.

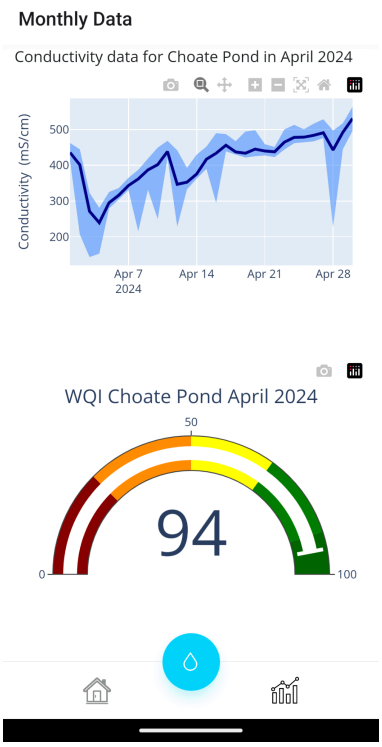
2.1.1 AquaWatch Mobile App Interface/Functionality - Descriptions

Table 2-0: Screen Interfaces Descriptions

ID	Name	Description	Screenshot
[2]	Home Screen	Contains buttons to go to the following sub-pages. We navigate to those pages via buttons. [2] Home Screen ├─ [3] Our Story ├─ [4] Animal Life ├─ [5] Blogs └─ [6] Attribution Page	
[3]	Our Story	Provides the background info on Blue CoLab. It is a website that's embedded into our app. The contents of which can be found here: https://bluecolab.pace.edu/about-us-2/.	[omitted: see https://bluecolab.pace.edu/about-us-2/]

[4]	Animal Life	Displays what animal life will be doing at the time of year user views the page.	
[5]	Blogs	Displays Blue CoLab blogs. It is a website that's embedded into our app. The contents of which can be found here: https://bluecolab.blogs.pace.edu/blog-app/	[omitted: see https://bluecolab.blogs.pace.edu/blog-app/]

[6]	Attribution Page	Contains appropriate attributions to teams and APIs used.	
[7]	Real Time Data	Realtime data via Grafa	

[8]	Historic Data	Historic Data [8] Data Hub (Shiny) ├─ Choate Pond Water Quality Data ├─ West Point Water Quality Data ├─ Poughkeepsie Water Quality Data └─ Yonkers Water Quality Data	

2.2 shinyapps.io

This is the service that hosts our data visualization service. The exact code implementation can be found in section 3.2. We will communicate our goals with the data visualizations in this section. For the sake of clarity we will call this "data visualization" interface instead.

2.2.0 Data Visualization Interface

Data Visualization Interactive UI (Water)
- [1] Interactable Graph
- [2] Monthly Summaries and WQI
- [3] Interface
  - Water Parameters Dropdown
  - Location 1 Location Dropdown (required)
  - Location 1 Year Dropdown (required)
  - Location 1 Month Dropdown (required)
  - Location 2 Location Dropdown (optional)
  - Location 2 Year Dropdown (optional)
  - Location 2 Month Dropdown (optional)

Figure 2.2: Interface for Water (4a, 4b, 4c, 4d)

The above gives the general requirements of the interface of our data visualizations.

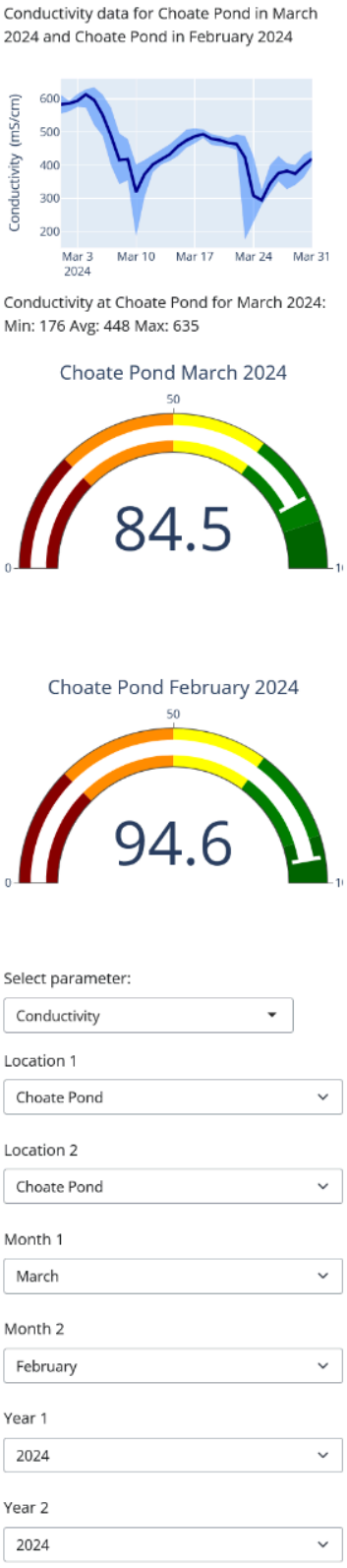
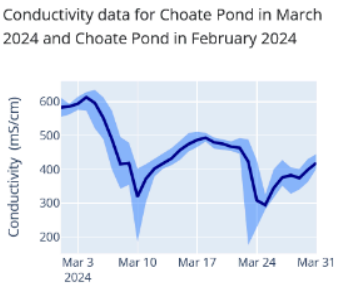
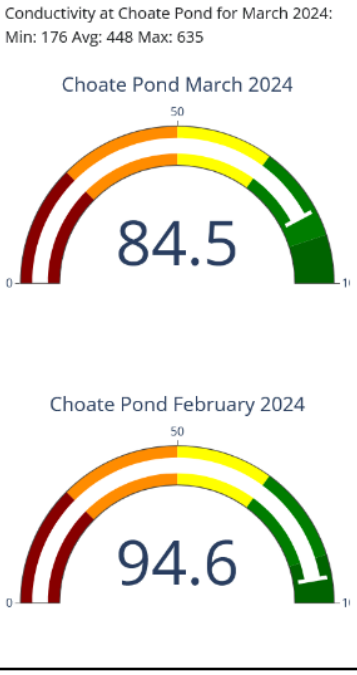
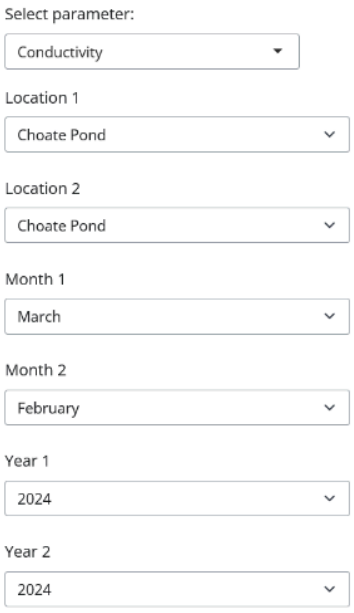


Figure 2.3: "Current" Data Viz Interface

2.1.1 Data Viz Interface/Functionality - Description

Table 2-1: Data Viz Interfaces Descriptions

ID	Name	Description	Screenshot
[1]	Data Viz	The data viz should show the following noted on the right. The graph shows the daily min, max, and average of the day's data in that month. *title* - title *description of data* - Display what data is being displayed *the actual data* - whether it be for two months/location or one month, display the data as a line graph *ribbons* - ribbons indicate the min and max, i.e. the range of the parameter for a particular day. The line is the average. *x/y-axis labels* - indicate units and dates *error points* - indicate if data is out of range	
[2]	Monthly Summary and WQI	Shows the summary of the month's data overall. Shows the min, max, and average of the whole month. Also includes a WQI gauge.	

[3]	Interface	Interface to control the above displays. One dropdown for the parameter chosen. Location #1 - Drop down for the location #1, where, what year, what month. Location #2 - Drop down for the location #2, where, what year, what month.	

2.1.2 Data Viz Interface/Functionality - Backend

~~The current version of the data viz is capable of taking url parameters to autofill dropdowns.~~

2.1.3 Data Viz Interface/Functionality - API Usage

The current version of the data viz uses Blue CoLab and USGS/HRECOS data.

2.1.4 Data Viz Interface/Functionality - Misc.

Other miscellaneous functions include:

- ~~Basic data cleaning to remove outliers~~
- Converting Celsius to Fahrenheit

2.3 Blue CoLab API & USGS/HRECOS APIs

API used. We used Blue CoLab API cause, well that's our job. We used USGS/HRECOS APIs because why not.

Learn more here for Blue CoLab API: lkeeley@pace.edu
Learn more here for USGS: https://ny.water.usgs.gov/maps/hrecos/ :)

2.4 Conclusion

Next section will cover how to read our code.

3.0 The Code

An overview of libraries used and our code. For detailed understanding of our code we wrote please refer to the documentation within it - but this serves as a general overview.

3.1 AquaWatch Mobile App:

Our mobile app fundamentally uses Expo as a foundation to build the app. We also add additional packages, as documented, to enhance the app experience.

3.1.0 Software Requirements

1. React/React Native (18.2.0, 0.72.6, Metra Open Source) - Serves as our front end alongside other packages. Various packages include:
    a. react-native-webview (13.2.2, Thibault Malbranche) - WebView component to embed website.
    b. react-native-snap-carousel (^3.9.1, Benoît Delmaire) - Create cool carousels used in wildlife pages.
    c. expo-linear-gradient (~12.3.0, Expo) - Gradients for the cards
    d. react-navigation/bottom-tabs (^6.5.20) - Additional navigation
    e. react-navigation/native (^6.1.17) - Additional navigation
    f. react-navigation/stack (^6.3.29") - Additional navigation

2. Expo (~49.0.10, 650 Industries, Inc.) - Required as of now to display applications on phones. For Expo related requirements, see this.

3. moment (^2.29.4, JS Foundation) - Used to parse and format dates.

4. node.js/node - Used to manage all packages.

All required libraries should automatically download when running the *npm i* in the app directory.

3.1.1 Required Accounts

- Google Developer Account
- Expo Developer Account

3.1.2 Local Testing Steps

1. The official local testing can be found here.
2. One time steps:
    a. Download node.js
    b. Run *npm i* in a terminal in the directory where the app is located, it installs all needed node_packages and expo. It may take a few minutes to install everything.
    c. Download Expo Go on your phone.
    d. If you want to test in an emulator, follow these steps.
3. Testing:
    a. Run *npx expo start* in the terminal in the directory where the app is located (should be installed with node). Scan the QR Code.
    b. If your phone is on the same network as the computer running the app, the Expo Go app will display the app. (Even at Pace, you must be logged into WiFi with the same Pace Account Type (only Student Type, or only Employee account type).

c. Anytime you make changes in your computer, it should be reflected on the phone but…
d. After making any changes if no changes are reflected, pressing 'r' with the terminal open should refresh the phone. Otherwise, try running *npx expo start* and scanning QR code again.
    i. Note: If you add any new libraries, you will have to rerun *npx expo start.*

3.1.3 Official Deployment Steps

1. The official deployment steps can be found here and here. With unofficial deployment steps here.
2. One time steps - Steps you only have to do once per computer:
    a. In a command line enter: *npm install -g eas-cli*
    b. In a command line enter: *eas login* and login.
    c. Don't have to run: *eas build:configure* (as we already have a eas.json)
3. Uploading the build directly to app store:
    a. Merge all changes into the *app-deploy-branch* branch in GitHub.
    b. Update the app.json file, increment the version code and version. You must update the version code or else Google Play will reject build.
    c. In a command line enter: *eas build -p android --profile preview*
        i. Note: In general build times should take about 10 minutes. The queue time varies, late nights queues are much shorter.
    d. Navigate to the link given in the command prompt. Wait for the build to finish.
    e. Download the app bundle build once complete.
    f. Play Console: Create a new release, upload the new bundle for it, create a release notes. Submit it for review by Google Play.
4. Uploading Updates:
    a. We just upload the app bundle every time.

3.2 shinyapps.io:

Internally our app uses shinyapp.io to host the data visualizations.

3.2.0 Software Requirements

1. Python (3.11.* only) - Language used for the shiny app. Shiny does not yet support Python 3.12.
    a. Suggested Dev Environment: VS Code
        i. Suggested VS Code Extensions:
            1. Python (linting, debugging etc.)
            2. Shiny (simplifies running app locally)
        ii. Learn more here.
    b. NOTE: When installing Python, make sure to check "Add python.EXE to PATH".

2. Additional Packages (not already part of Python)
    a. shiny (0.8.1, Winston Chang) - Tool to create interactive web interface for accessing data visualizations. This is where the drop downs come from.

b. shinywidgets (0.3.1, Carson Sievert) - Extends Shiny with custom input widgets.
c. plotly (5.20.0, Chris P) - Enables interactive plots and charts.
d. pandas (2.2.1) - Data analysis for Python.
e. dataretrieval (1.0.6) - Library used to access USGS water data.
f. rsconnect_python (1.22.0) - Package used to interact with and deploy to Posit Connect (shinyapps.io).

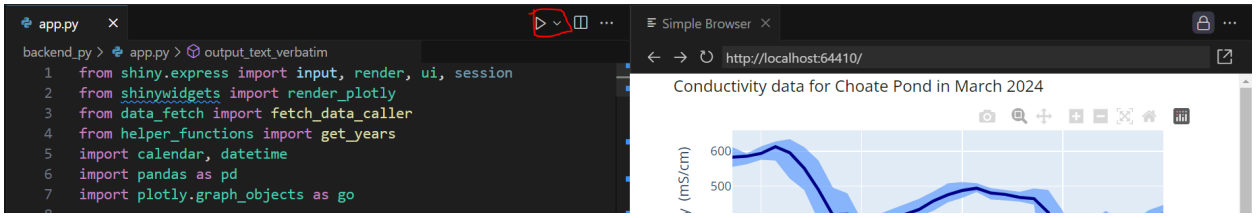
All required libraries are in the requirements.txt. They should automatically install by running *pip install -r requirements.txt* in backend_py directory. Otherwise run *pip install [name_of_packages]*

3.2.2 Required Accounts

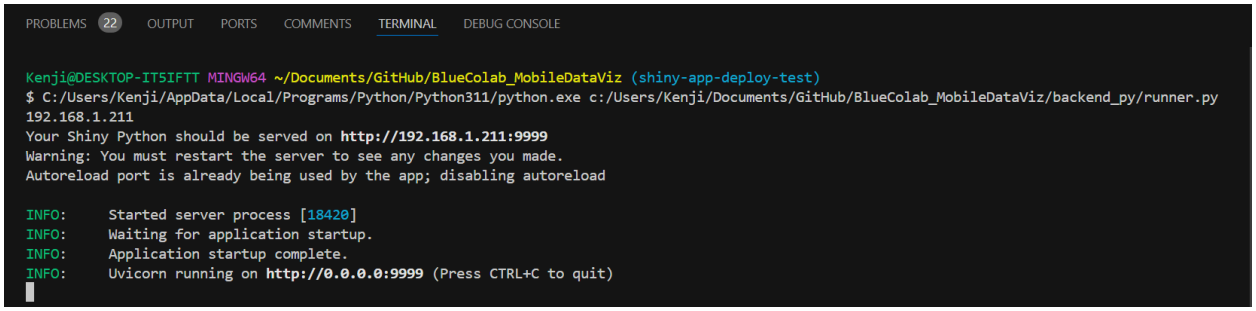
- shinyapps.io

3.2.2 Local Testing Steps

1. The official local steps can be found here.
2. One time steps - Steps you only have to do once per computer:
    a. Install Python and download all required packages. See the requirements.txt file.
    b. Install VS Code
    c. Install Shiny Extension in VS Code.
3. Open app.py, and click on the Run (Play) Button.



4. Mobile Device Testing. Run the runner.py file.



    a. Once you run it, you should see the above. Go to the link noted after "Your Python should be served on: http://xxx.xxx.x.xxx:9999".
    b. If your phone is on the same network as the computer running the Python, you should be able to navigate to it. (Even at Pace, you must be logged into WiFi with the same Pace Account Type (only Student Type, or only Employee account type).
    c. Note that this is not an official way of testing app and may be buggy.

3.2.3 Official Deployment Steps

1. The official deployment steps can be found here.
2. One time steps - Steps you only have to do once per computer:
    a. In a command line enter: *pip install rsconnect-python* to install rsconnect python which will upload the shiny code into the online server.
    b. Log in to shinyapps.io with Blue CoLab credentials (contact Leanne or Cronin).
    c. Navigate to Account > Tokens > Show (on a token) > With Python > Copy to Clipboard > Copy what it tells you to copy
    d. Paste the copied text into a command line and run. It should like something like this: *rsconnect add --account aquawatchmobile --name aquawatchmobile --token token --secret secret*
        i. Note: If you get a command not found error, you may not have added Python to your path.
3. Deploying:
    d. Run: *rsconnect deploy shiny /path/to/app --name aquawatchmobile --title aquawatchmobilepy*
    e. For the above, put the path to where backend_py is located on your computer.

4.0 Appendix A: Resources

1. GitHub
2. Expo Documentation
3. USGS HRECOS Documentation
4. Shiny For Python Documentation
5. shinyapps.io

5.0 Appendix B: Ideas for Future Teams

Deploy the app kids. Ask Victor.

- Add spinner that they added literally day of presentation: https://github.com/posit-dev/py-shiny/pull/918