Lab Five - Databases, Visualization, and Putting it All Together

**Note: This is the final lab of the semester. There are two parts to this lab, and you will, tentatively, be given three weeks to complete both parts.*

Part One: Databases and Data Visualization

Useful Links:
http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html
https://docs.mongodb.com/manual/administration/install-on-linux/
https://docs.mongodb.com/manual/
http://scikit-learn.org/stable/modules/svm.html

When devices respond to changes in the environment or read in inputs from the physical world data is generated. We can create systems or execute functions based on this data to achieve a desirable output or result. Sometimes we need to collect large amounts of data for long periods of time from a large variety of sources. In these cases it would be in our best interest to store our data in an organized fashion so that we can later process and analyze our dataset. In cases like these, it is very useful to use a database.

For this portion of the lab, we will be using the MongoDB database service. Database services, like MySQL, provide a table-based relational database that can be accessed through the SQL language, which can be rigid and inflexible for certain applications. MongoDB is based off of NoSQL, which essentially gets rid of the rigid tabular structure of SQL, allowing the user to set up any organizational structure (though the two structures are actually very similar in their core). Since the ESP8266 does not have an operating system, and thus cannot install any of the existing database software, we will host the database on an Amazon AWS server, and access it using HTTP commands on the ESP8266. We can store almost anything on a database, and will focus on storing sensor values from our smartwatch.

The tasks to complete are as follows.

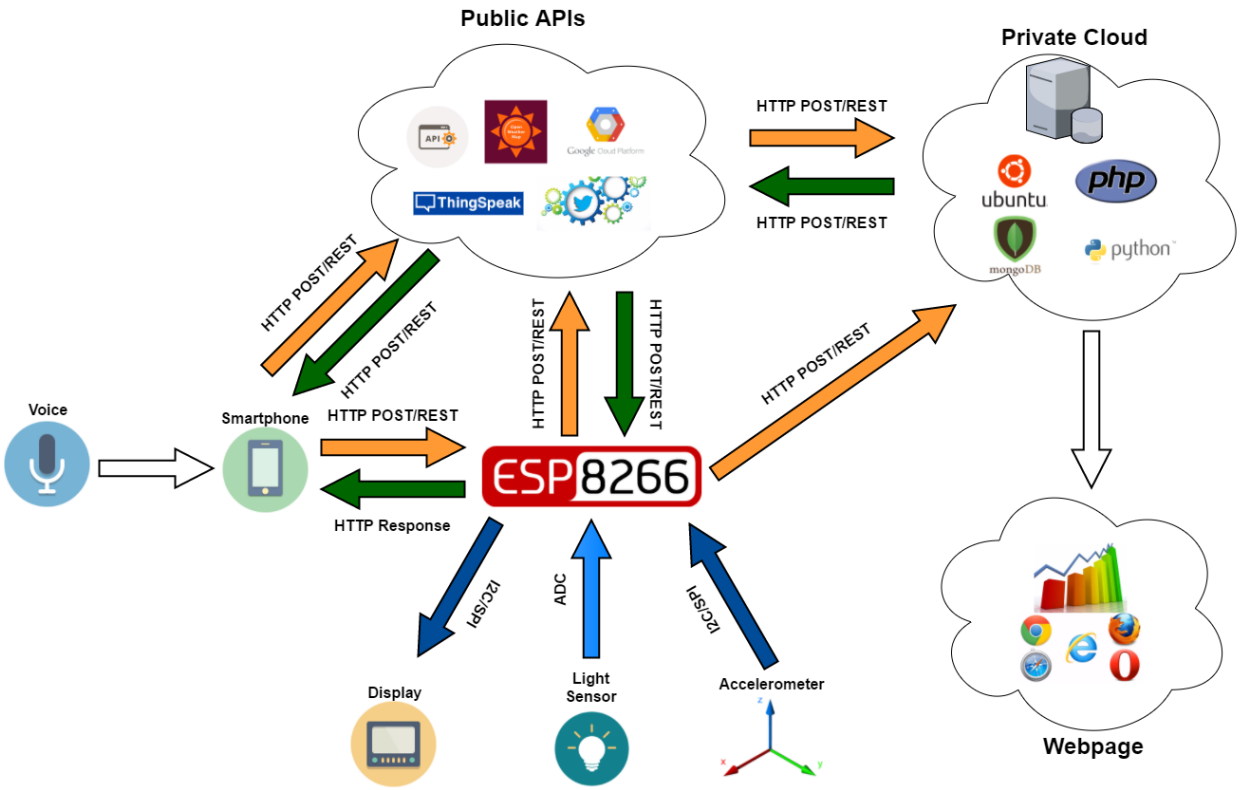
1. Launch an EC2 Linux server instance. Make sure to create a security group for your instance that at least allows the server to accept any inbound HTTP, HTTPS, and SSH requests from Columbia IP addresses. The easiest way to do this is to allow any IP address to connect to your server via these methods by inputting “0.0.0.0/0” into the source fields, though in practice you may not want to do this for security reasons.
2. Access the EC2 server and install MongoDB.
    a. More information on how to set up an EC2 instance can be found: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html
    b. More information on how to install MongoDB can be found: https://docs.mongodb.com/manual/administration/install-on-linux/
    c. Mongodb Documentation: https://docs.mongodb.com/manual/
3. Create the database using an appropriate structure.

4. Create and assign your own gestures to recognize the letters “COLUMBIA”.
5. Setup a neural network on the EC2 server which can recognize these gestures. There are 2 steps involved in setting up the neural network:
    a. Training: Send the accelerometer data from Huzzah to EC2 server and store it in the database. Once you think you have enough training samples, pull the data from the database, and use it to train the neural network (SVM is recommended). Store the trained network in a filehttps://scikit-learn.org/stable/modules/svm.html.
    b. Testing: Use a few more data samples for testing and see if you are getting the expected results. If not repeat the training process with better data samples.

    Be sure to store all the data samples (Training and testing) in proper databases.
6. Obtain the word “COLUMBIA”, one letter at a time and display it on OLED.

Part Two: Creating the Smartwatch

We now have all of the individual components that will combine to form our smartwatch. The only thing left to be done is to create the system. The diagram and sections below provide a summary of what the final system should contain.



Voice Commands:

The smartphone application and smartwatch should respond to the following voice commands:

- Display weather: temperature and description (e.g. mostly cloudy)
- Send spoken tweets

- Display current time on smartwatch

Smartphone Application:

- Interface with Google Speech API to translate voice commands into text.
- Send the voice commands to the smartwatch
- Voice command features:
    - Display weather: Display the temperature and description on the application itself; also send the command to the smartwatch.
    - Send spoken tweets: In addition to sending tweets to the smartwatch, the application should also display the spoken tweet on the application itself.
    - Display current time on smartwatch: Send the command to the smartwatch.

Embedded Server/ESP8266:

- Interface with the OLED display to display the time and allow users to set the time.
- Alarm: Users should be able to set an alarm through the OLED display; once the alarm goes off, a visual and audio notification (using the piezzo) should play until the user interacts with the display again.
- Receive and process voice commands from the smartphone application
- Voice command features:
    - Display weather: After receiving this command from the smartphone, obtain the weather in the area around your geolocation and display it on the screen.
    - Send spoken tweets: After receiving this command, the custom spoken tweet should be tweeted on the account linked to the project. Additionally, display the tweet on the smartwatch.
    - Display the current time on smartwatch: After receiving this command, the watch should switch to the current time menu and display it to the user on the OLED.
- Allow users to display the weather information obtained from the previous weather voice command, through the OLED display.
- Allow users to display the last tweet sent through the OLED display.
- Read values from the light sensor; adjust screen brightness/contrast with respect to the ambient light around. For example, the display should get brighter if it the reading of the sensor increases, and it should get dimer if it the reading of the light sensor decreases.
- Include gesture recognition mode to recognize any letter in the word “COLUMBIA”.

Cloud Server/Database:

- Receive and store accelerometer data from the smartwatch in a database.
- Setup and train a neural network that can recognize the gestures through the data received from huzzah.

**Notes: The above sections provide an outline of what should go into the final system. By this point, some of you may have experienced one of the constraints of small embedded systems: memory. While putting the system together, be mindful of your implementation to optimize your code to use less RAM. Additionally, strive to think about certain strategies (off-loading tasks*

from one part of the system to other parts, reduce RAM usage, more efficient implementations, modularization, etc.) that could drastically reduce the amount of memory your program requires; the guidelines above only disclose the features that should be present in the final system, not how to go about implementing them.

EECS 4764 Lab Five Checkpoints:

Part One: Databases and Data Visualization

1. Establish a connection between the smartwatch and the AWS server and continuously send accelerometer data to the AWS database at a frequency of at least 1 Hz.

2. Setup a neural network on the AWS server that can recognize any letter in the word “COLUMBIA”.

3. Obtain the word “COLUMBIA” within 10 tries and display the word one letter at a time on OLED.

Part Two: Creating the Smartwatch

1. Features outlined for each of the subsystems should be functioning discretely.
    a. Voice Commands
    b. Smartphone Application
    c. Embedded Server/ESP8266
    d. Cloud Server/Database
2. All discrete components are combined and fully functioning.