Lab Six - 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 two weeks to complete both parts, instead of the usual one week.*

Part One: Databases and Data Visualization

Useful Links:

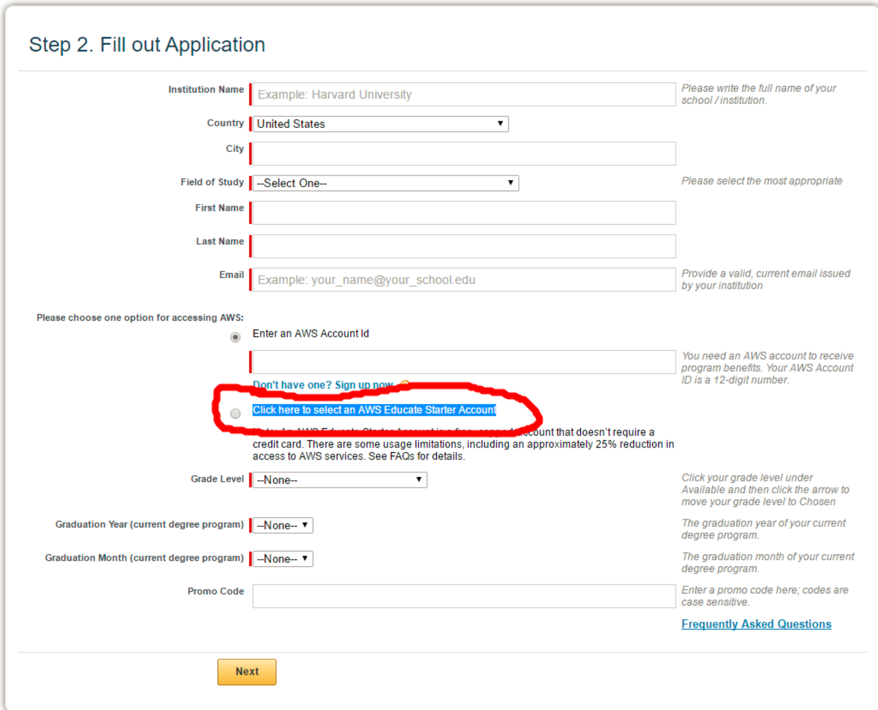
https://www.awseducate.com/Application
http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html
https://docs.mongodb.com/v3.0/administration/install-on-linux/
https://docs.mongodb.com/manual/
http://flask.pocoo.org/docs/0.11/installation/
http://flask.pocoo.org/docs/0.11/

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.

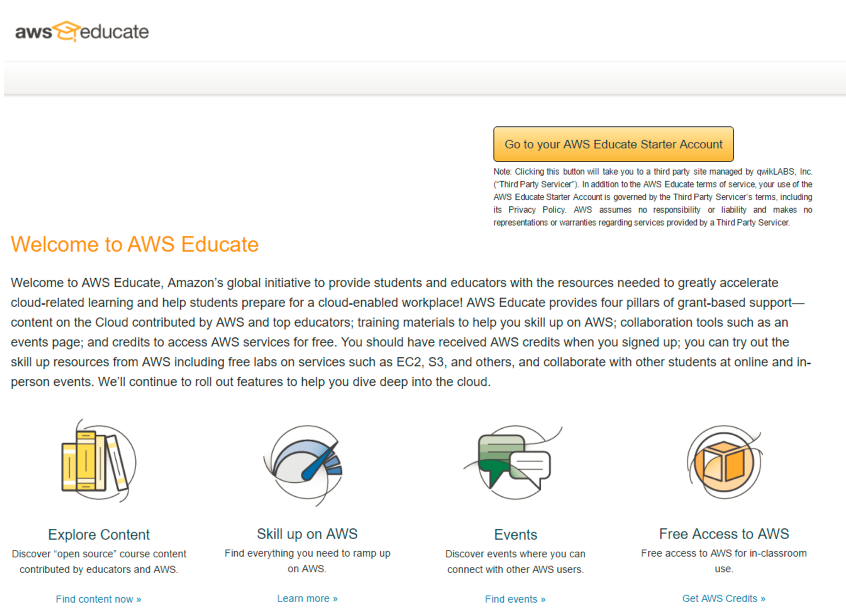
For this portion of the lab, we will be using the MongoDB database service. Database services like MySQL, provide a table-based relational database that can be accessed through the SQL language, which can be rigid and inflexible for certain applications. MongoDB is based off of NoSQL, which essentially 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. Sign up for an AWS Educate Starter Account through the following link: https://www.awseducate.com/Application.
2. Make sure to choose the student option. Once you get past the first screen you will be taken to the screen shown on the following page. Make sure to select “Click here to to select an AWS Educate Starter Account”. This way, you can complete registration without having to input payment information.

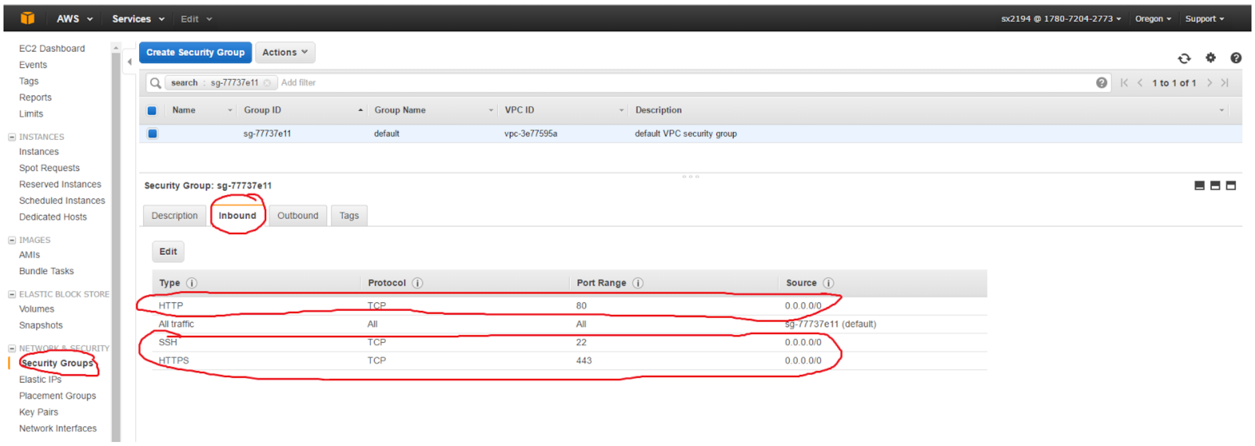


3. Once you register and gain access to your account, you will be brought to the following page.



4. Click on "Go to your AWS Educate Starter Account". You will be brought to an educational website with labs. Select "AWS Starter Account 75 Lab", and from here you can access the AWS Management console, just like in a regular AWS account.

5. 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.



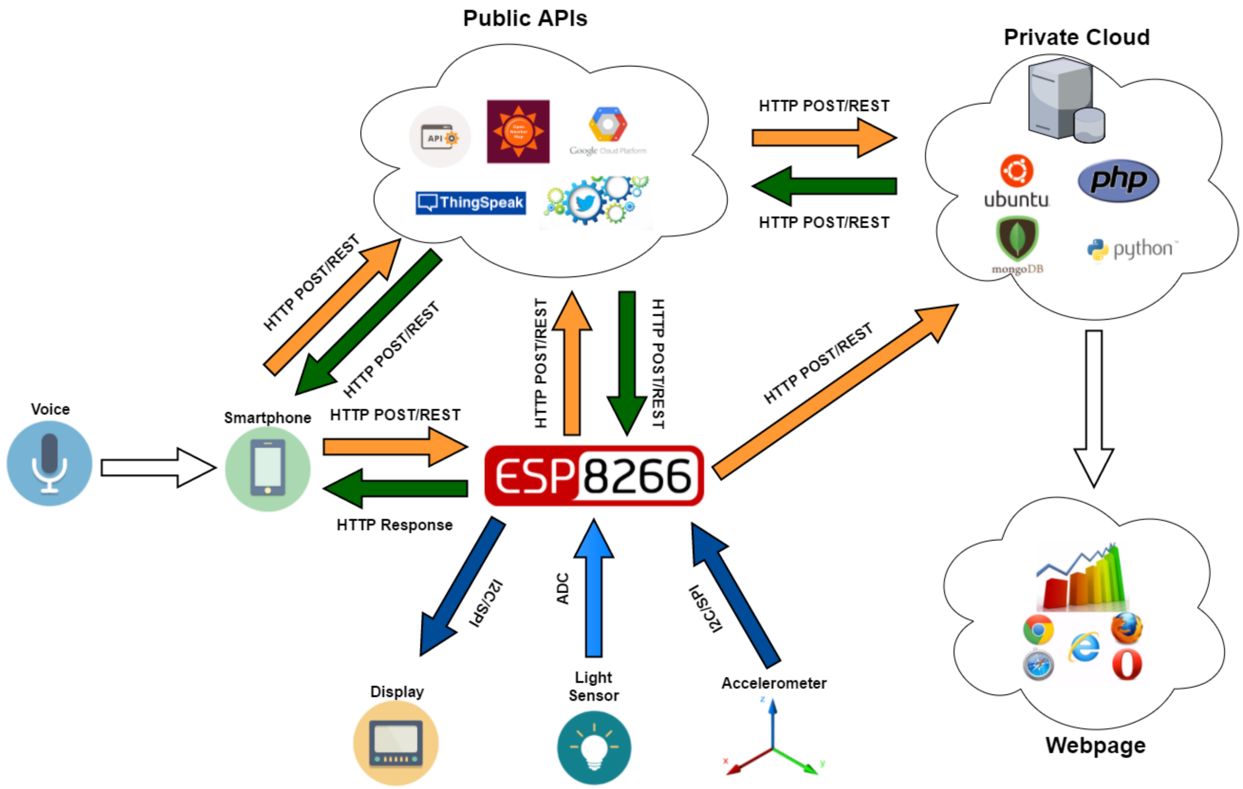
6. Access the EC2 server and install MongoDB.
    a. More information on how to set up an EC2 instance can be found: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html
    b. More information on how to install MongoDB can be found: https://docs.mongodb.com/v3.0/administration/install-on-linux/
    c. Mongodb Documentation: https://docs.mongodb.com/manual/
7. Create the database using an appropriate structure. The database will store data from all three axes of the accelerometer.
8. Send accelerometer data from the smartwatch to the AWS database at a frequency of at least 1 Hz. To set up a server on the EC2 instance, you can use Python Flask.
    a. Installation Instructions:
    ```
    sudo apt-get install python-pip
    sudo pip install flask
    ```
    b. Create a flask app and set it up with the apache server. You should configure and setup the apache server and the link the flask app to it.
    c. The apache server can be installed with the following commands:
    ```
    sudo apt-get update
    sudo apt-get install apache2
    ```
    d. The apache server displays html pages by default. To server dynamic content using the flask app you need to setup mod_wsgi through which you will link your flask app to the apache server. The installation of which can be done by :
    ```
    sudo apt-get install libapache2-mod-wsgi
    ```
    e. You will then have to appropriately configure apache and mod_wsgi.
    f. Documentation: http://flask.pocoo.org/docs/0.11/

    The easiest way to do this is to design RESTful interfaces on your server which will accept GET/POST commands, so that you can send your values using GET / POST.

9. Finally, create an html webpage on that will allow you to fetch and visualize the data by invoking the Google Charts API, any other nice visualization API, or make your own visualization to display the data in a nice graph. A relatively straightforward API to use is Highcharts. The visualization should be real-time. The documentation for Highcharts can be found here: http://www.highcharts.com/docs

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 do is to create the system. The diagram and sections below provides 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 dimmer if it is bright out, and it should get brighter if it is dim out.
- Read data from the x, y, z axes on the accelerometer. Send these values to the cloud database at a rate of at least 1 Hz.

Cloud Server/Database:

- Receive and store accelerometer data from the smartwatch in a database.
- Create a webpage that allows users to view the accelerometer data in the database. The webpage and graph should look nice, and it would be great if the graph updated in real-time as the server receives data from the smartwatch.

**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.*

Group Members:

EECS 4764 Lab Six Checkpoints:

Part One: Databases and Data Visualization

1. Establish a connection between the smartwatch and the AWS server. It is recommended to set up a Flask server on AWS.

2. Continuously send accelerometer data to the AWS database at a frequency of at least 1 Hz.

3. Display accelerometer data on an html webpage with nice visualizations. Ideally you should be using Highcharts, and the graph should be updating in real-time as it receives new data points.

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.