

# Lab Six - Databases, Gesture Recognition, 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 Gesture Recognition

### Useful Links:

<https://www.awseducate.com/Application>

[http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2\\_GetStarted.html](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/>

<https://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. 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 select an AWS Educate Starter Account". This way, you can complete registration without having to input payment information.

**Step 2. Fill out Application**

Institution Name  Please write the full name of your school / institution.

Country

City

Field of Study  Please select the most appropriate

First Name

Last Name

Email  Provide a valid, current email issued by your institution

Please choose one option for accessing AWS:

☒ Enter an AWS Account ID  You need an AWS account to receive program benefits. Your AWS Account ID is a 12-digit number.

☐ [Click here to select an AWS Educate Starter Account](#) Click here to select an AWS Educate Starter Account. You don't need an AWS account that doesn't require a credit card. There are some usage limitations, including an approximately 25% reduction in access to AWS services. See FAQs for details.

Grade Level  Click your grade level under Available and then click the arrow to move your grade level to Chosen

Graduation Year (current degree program)  The graduation year of your current degree program.

Graduation Month (current degree program)  The graduation month of your current degree program.

Promo Code  Enter a promo code here; codes are case sensitive.

[Frequently Asked Questions](#)

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

**awseducate**

Note: Clicking this button will take you to a third party site managed by quickLABS, Inc. ("Third Party Service"). In addition to the AWS Educate terms of service, your use of the AWS Educate Starter Account is governed by the Third Party Service's terms, including its Privacy Policy. AWS assumes no responsibility or liability and makes no representations or warranties regarding services provided by a Third Party Service.

**Welcome to AWS Educate**

Welcome to AWS Educate, Amazon's global initiative to provide students and educators with the resources needed to greatly accelerate cloud-related learning and help students prepare for a cloud-enabled workplace! AWS Educate provides four pillars of grant-based support—content on the Cloud contributed by AWS and top educators; training materials to help you skill up on AWS; collaboration tools such as an events page; and credits to access AWS services for free. You should have received AWS credits when you signed up; you can try out the skill up resources from AWS including free labs on services such as EC2, S3, and others, and collaborate with other students at online and in-person events. We'll continue to roll out features to help you dive deep into the cloud.

**Explore Content**  
Discover "open source" course content contributed by educators and AWS.

[Find content now »](#)

**Skill up on AWS**  
Find everything you need to ramp up on AWS.

[Learn more »](#)

**Events**  
Discover events where you can connect with other AWS users.

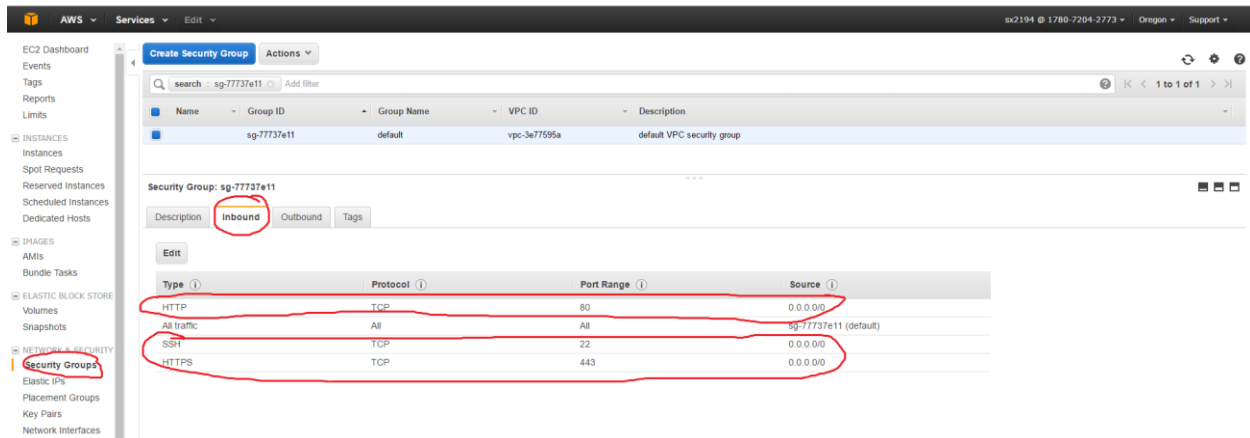
[Find events »](#)

**Free Access to AWS**  
Free access to AWS for in-classroom use.

[Get AWS Credits »](#)

- 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.

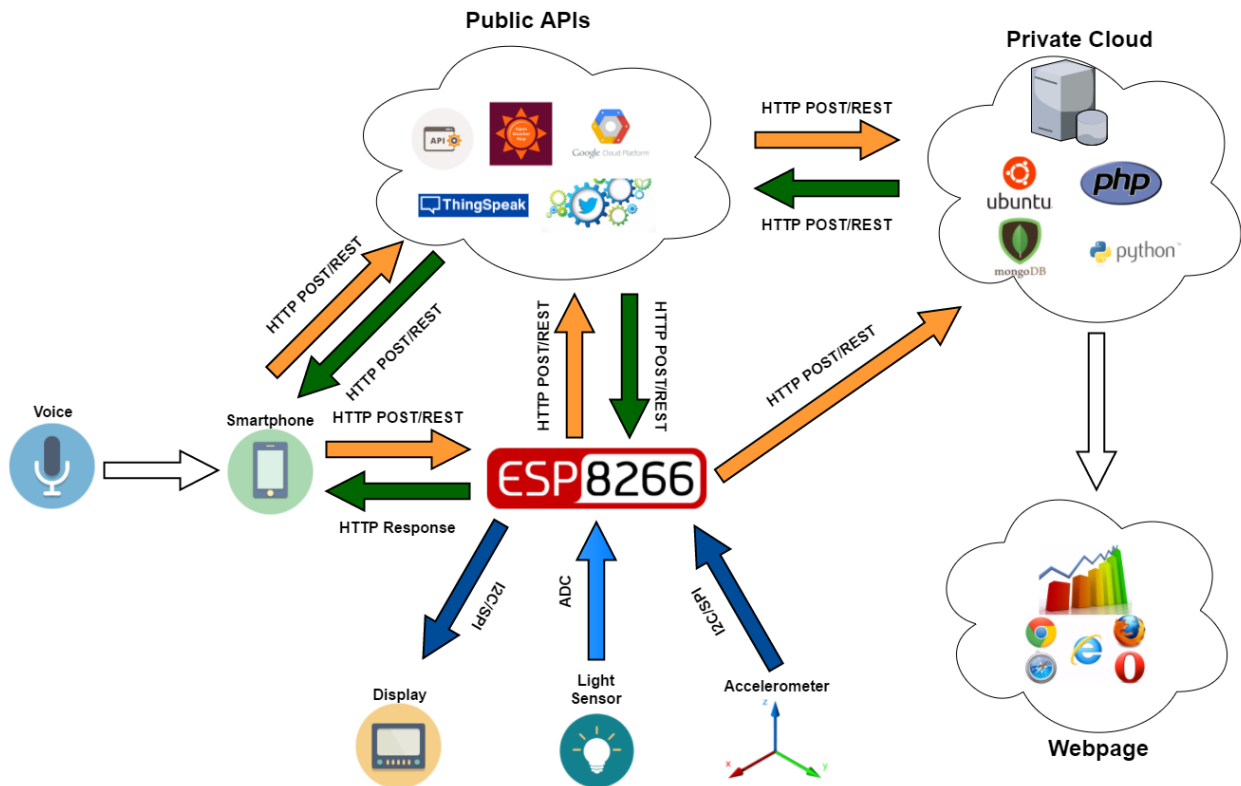
5. Launch an EC2 Linux server instance. Make sure to create a security group for your instance that at least allows the server to accept any inbound HTTP, HTTPS, and SSH requests from Columbia IP addresses. The easiest way to do this is to allow any IP address to connect to your server via these methods 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](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.
8. Create and assign your own gestures to recognize the letters “COLUMBIA”.
9. 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 file <https://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.
  - c. Be sure to store all the data samples (Training and testing) in proper databases.
10. 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 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 piezo) 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.
- 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 Six Checkpoints:**

### **Part One: Databases and Gesture Recognition**

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.