Lab Four - Embedded Servers

As the name implies, a server is a component or device that provides a service or function to a client device. Any web page you can access through the Internet browser and applications, like games and video services, are hosted on servers for users and clients to request and interact with. All a basic server does is take in requests from a client and sends back a response.

In this lab, we will make our smartwatch application (more specifically the ESP8266) into a server that will take requests from a smartphone app through HTTP. More specifically, the smartphone app will send commands or requests to the smartwatch server, and the smartwatch will execute these commands.

Part One: Smartphone Application

Useful links:

https://developer.android.com/develop/index.html
http://www.instructables.com/id/How-To-Create-An-Android-App-With-Android-Studio/
https://developer.android.com/guide/topics/manifest/manifest-intro.html
https://developer.android.com/reference/android/app/Activity.html
https://developer.android.com/guide/topics/ui/declaring-layout.html
https://developer.android.com/guide/topics/ui/controls/button.html
http://android4beginners.com/2013/06/lesson-1-how-to-display-and-format-a-text-in-android-apps/
https://developer.android.com/training/basics/network-ops/connecting.html
https://developer.android.com/reference/org/json/JSONObject.html
https://www.tutorialspoint.com/android/android_json_parser.htm
https://cloud.google.com/speech/

Create an Android smartphone application that will interface with the Google Speech API. The application should allow users to speak a voice command and send that command to the ESP8266 server via HTTP.

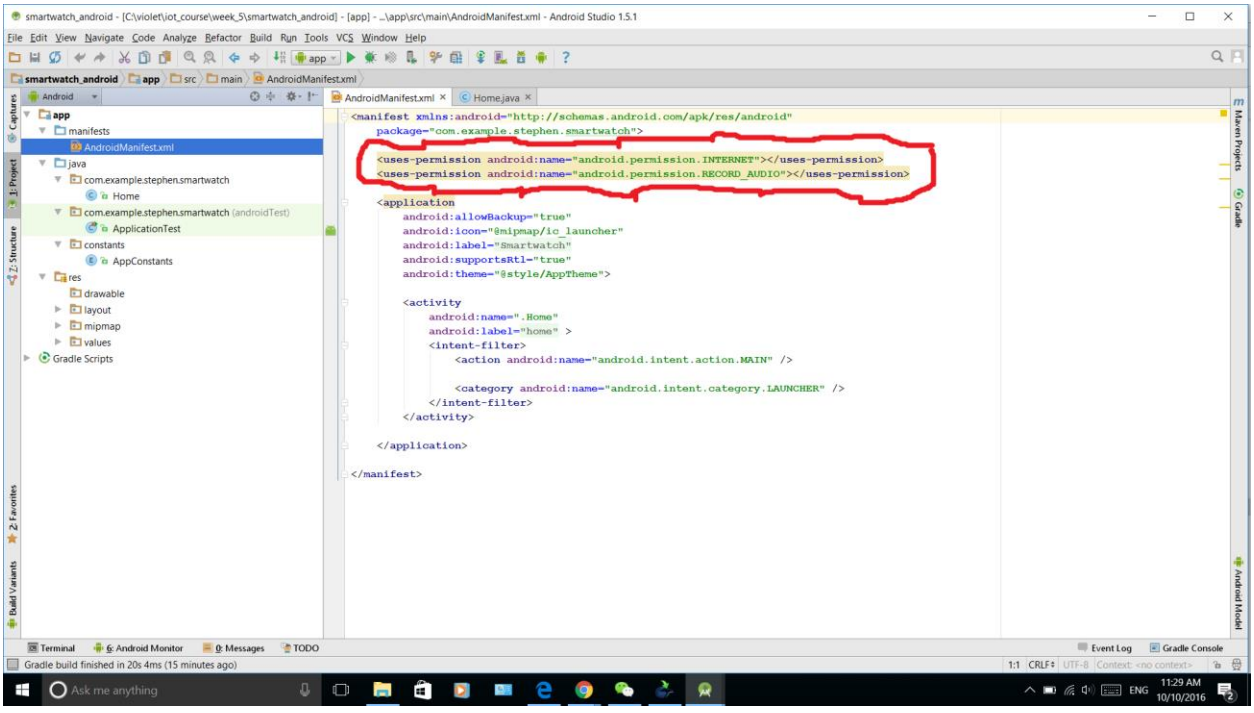
To download Android Studio, go to https://developer.android.com/studio/index.html and download the version appropriate for your operating system.

You can look on https://developer.android.com/develop/index.html to learn more about how to create an Android application; there are more links listed in the “useful links” section above. Once you have set up Android Studio, you can now create an application that interfaces with the Google Speech API. More information about the Google Speech API can be found here: https://cloud.google.com/speech/.

General Tips for Running and Debugging Android applications:

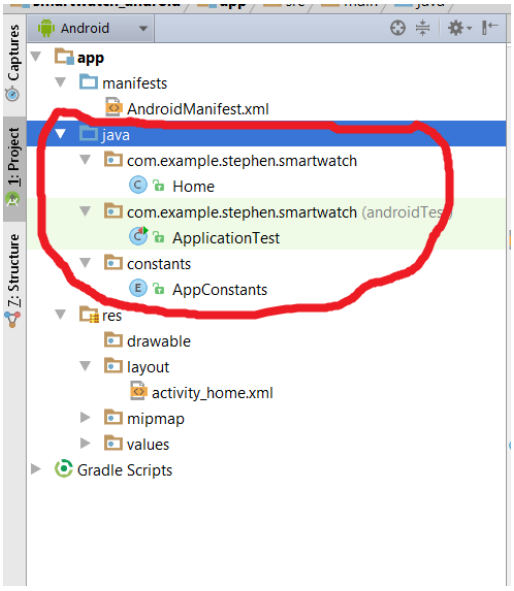
Developer Options:
In order to download applications to your device from Android Studio, you must enable developer options and USB Debugging on your device. To do this, go to **Settings->About Phone** and tap **Build number** about 7 times until you get a message saying that developer options has been enabled. Once this is enabled, you should see a new menu item called **Developer options** appear in your Settings screen. Go into this menu and enable **USB debugging**.

Android Manifest
The AndroidManifest.xml file contains information that specifies how the application is setup (permissions the application requires, what activities are within the application, which activity is the main activity, etc.). For this lab, you must add the Internet permission and the speech recording permission because the application will record your voice and communicate to Google Speech API and the HUZZAH through the Internet. The image provided below details what needs to be added.



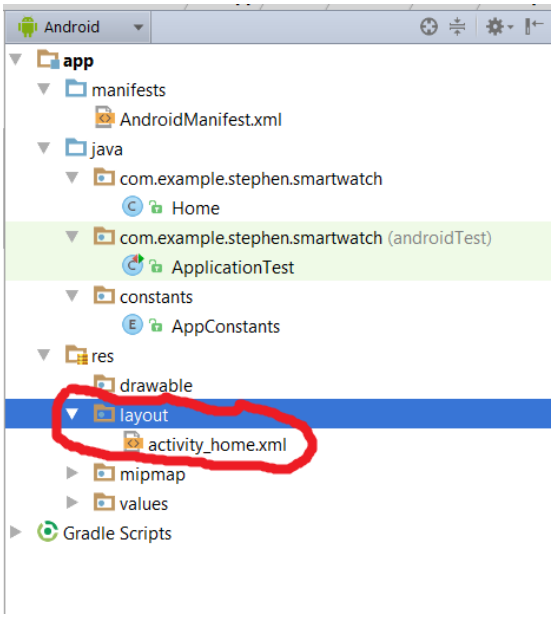
Activities and Source Code

Android applications run and allow users to interact with them through activities. You can specify what your application does through activities, specified in **app->java**. Essentially, the files in this folder contain the “source code” that specifies how your application functions.



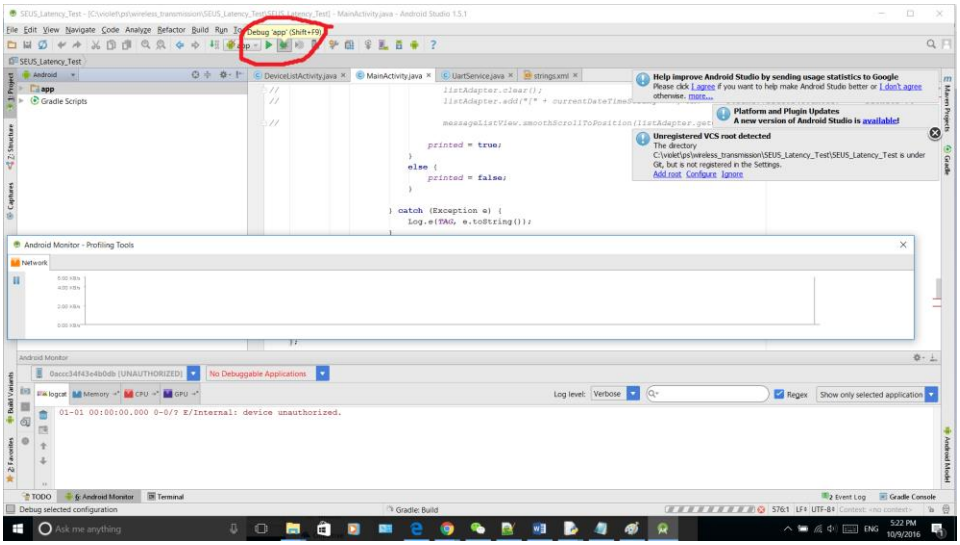
Activity Layouts

The java source files specifies how your application functions, but any smartphone application also has a user interface that users can interact with. User interfaces can be created in .xml files located in **res->layout**. These screens must be linked to an activity specified in **app->java**.

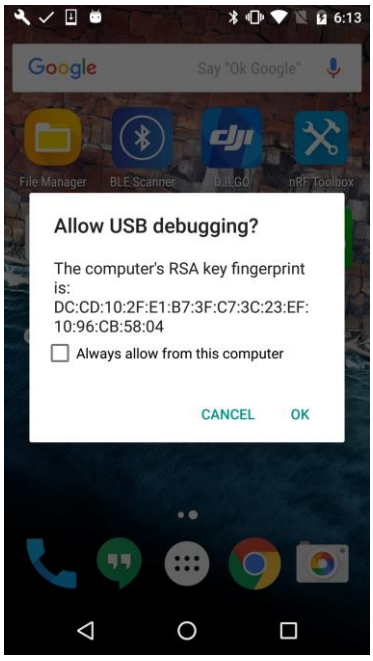


Downloading the Application to your Device

Once you have enabled USB debugging on your phone and you have created your application, connect your phone to the computer. Click on the debug button, as shown below:



Once you click on the debug button, Android Studio will launch something called the “Android Debug Bridge” or ADB, which essentially allows you to transfer your application from Android Studio to your device. When the ADB launches, your phone will receive a notification, like the one shown below. You must accept in order for you to be able to load your application onto your phone.



Part Two: Smartwatch Embedded Server

Useful links:
https://ngrok.com/
https://ngrok.com/docs

1. Once the smartwatch application is done, we must make sure the ESP8266 or smartwatch application can receive these requests. On the ESP8266 server, write a script that will allow the server to receive and process HTTP requests.
2. Configure the smartwatch display to turn off and turn on, display the time, and display any other message on the OLED whenever it receives the corresponding command from the smartphone. In other words, interface the smartphone app with the smartwatch embedded server. In order to access a server via HTTP on the Internet, the server must have a global IP address, which the ESP8266 does not have by default; it only has a local IP. To get around this, use ngrok to create a tunnel connection to the ESP8266. The ngrok software is available here: https://ngrok.com/. More information on how to use ngrok can be found here: https://ngrok.com/docs.

Submission Guidelines

Since you are required to create an Android Studio project, you may instead of zipping up the entire file project folder, upload to Github or Bitbucket, and provide a link/share your repository.

EECS 4764 Lab Five Checkpoints:

1. Create smartphone application that interfaces with the Google Speech API and sends translated commands to the smartwatch. Display the translated command on the smartphone application.

2. Configure the smartwatch to be a server and be able to connect to it via a smartphone application, web browser, etc.

3. Connect the smartphone application with the smartwatch; the commands to turn off/on the display, display the time, and display any other spoken message on the OLED display should be included.
    a. Other specifications: When the time is displaying, it should behave like a normal clock; that is, the time on the OLED should change corresponding to the actual time that has passed in the real-world whenever the time display is left on.

b. Smartphone Specifications: In a system like this, there should be 2-way communication between devices; when a command is sent to the ESP8266, the ESP should send back a response to the smartphone. The response should be displayed on the user interface of the application, and it should provide information about whether or not the command was successfully received and interpreted by the smartwatch.