Enterprise Smart Outlet: Android Development

Aaron Koeppel

Advisor: Professor Aaron Keen

California Polytechnic State University

Computer Science Department

June 9, 2015

Abstract

This project consists of one part of a larger multidisciplinary project, Enterprise Smart Outlet. The purpose of the Enterprise Smart Outlet is to create an electrical outlet which would allow enterprise users to monitor and change their power consumption habits on a large scale. The goal of this project is to allow consumers to see their power consumption per outlet in real time, and then use that information to inform their future habits, helping them conserve electrical energy and money. The Enterprise Smart Outlet Android application provides an interface to display the real time data of all smart outlets connected and the ability to enable or disable each outlet individually.

Contents

List of Figures

1 Introduction

1.1 Project Background

The market for "smart home" accessories has been growing for the past few years. Although the market is currently limited to home automation enthusiasts, the Enterprise Smart Outlet will branch off of the home automation market and expand into the enterprise market. The enterprise market has much more room to grow than the home automation market because businesses and organizations will pay for the extra amenities such as automation that in the long run (10-15 years) will pay for itself by saving electricity.

Furthermore, because Title 24 for California has recently passed, this opens up a mandatory adoption of our product into new buildings. Title 24 is California's efficiency code of regulations to help conserve energy and reduce carbon emissions within California. Regulations that went into effect July 1, 2014 require that all newly constructed or renovated buildings within California must adhere to new environmental standards. These standards require more compliance, creating stricter environmental standards and regulations for building construction and renovation.

One aspect of California's Title 24 that affects our project deals with the mandatory installation of controlled electrical outlets. The revision has numerous requirements, but one requirement in particular is that for every uncontrolled (always-on) electrical outlet there must be one controlled outlet within six feet. This demand on renovated buildings will create a much larger demand for Smart Outlets.

1.2 Project Goals

In order to create a full, marketable product, the Enterprise Smart Outlet team felt it was best to develop both an Android and iOS application. As the developer of the Android application, I was responsible for displaying data on all outlets connected to our server and providing the ability to activate and deactivate each outlet. The user is able to get real time data from each outlet updated at a regular interval.

2 Technical Specifications

2.1 Implementation Languages

2.1.1 Android Application

The application was written in Java and XML using the Android SDK. I used the Android Studio IDE to code and enhance my layouts using its live layout editor [5]. The data is transferred from server to client in JSON format. JSON (JavaScript Object Notation) is a lightweight data-interchange format [8]. I targeted Android SDK version 21 (Android 5.0) to give my app a modern feel but made sure it was compatible with SDK version 16 (Android 4.1) so it would be accessible to a large market. Currently Android 5.0+ (Lollipop) has a 9.6% market share, Android 4.4 (KitKat) has a 40.3% market share, and Android 4.1-4.3 (Jelly Bean) has a 41% market share [1]. Android 4.0.x has only a 4.2% market share [1]. By targeting Android 4.1+, my application is accessible to 90.9% of Android phones and I did not have to sacrifice modern features of the API. To make application side data persistent, I used Realm, a mobile database that runs directly inside phones, tablets or wearables [11]. Realm was a much simpler solution than SQLite, another mobile database option which has been the standard choice for the past fifteen years.

2.1.2 Servers

The application and outlets require two servers to run. The first server is an HTTP server which handles requests from the app to provide the outlet data. The second is an FTP server that handles data transmissions from outlets and sending commands to the outlets to activate or deactivate. Both of these servers were implemented in Node.js, a platform for server-side and networking applications [7]. Both servers have connections to a MySQL database. MySQL is the world's most popular open source relational database management system [10].

2.2 Hardware

2.2.1 Android Application

The Android application was tested and run on a Samsung Galaxy S3 running Android version 4.4.4.

2.2.2 Servers

The HTTP and FTP servers are hosted locally on a Raspberry Pi, a low cost, versatile computer [3] running Raspbian, a flavor of Linux based on Debian optimized for the Raspberry Pi hardware [2].

2.2.3 Router

In order to connect the phones, server, and outlets, a router is needed to provide a local WiFi network.

3 User Guide

3.1 Splash Screen

When a user first starts the application, they will see the splash screen as shown in Figure 1. After three seconds, the user will be taken to the main activity of the app.

Figure 1: Splash Screen

3.2 Main Screen

After the splash screen, the user will see the list of outlets as shown in Figure 2.

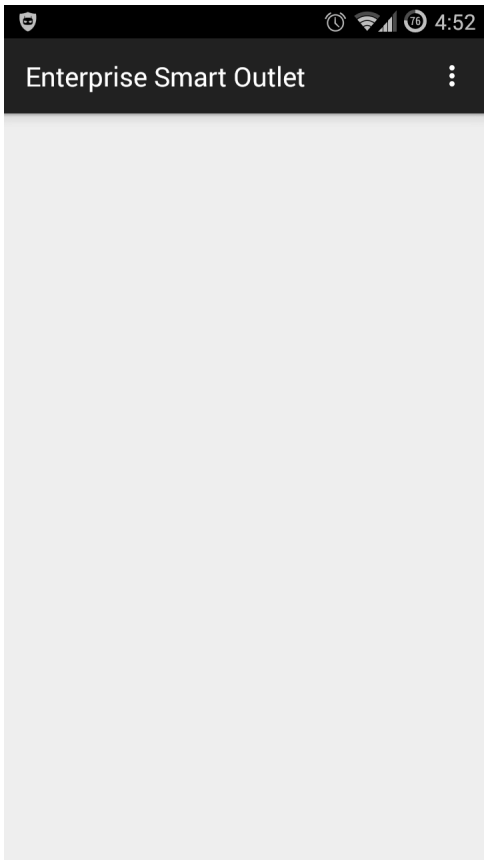

Figure 2: Empty Main Screen

This is what the app looks like if there are no outlets added to the server yet. Once outlets have been connected to the server, the list view will update and show the added outlets, as shown in Figure 3.

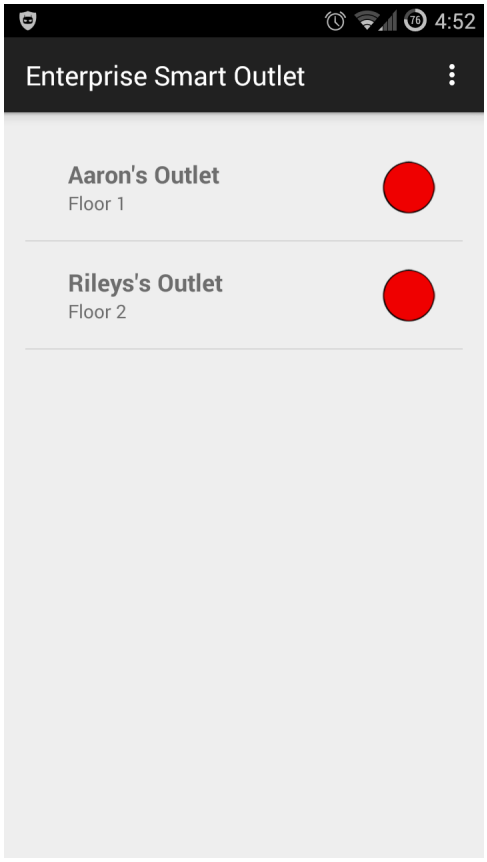

Figure 3: Deactivated Outlets Main Screen

The list view of outlets displays three pieces of data. The bolded top text is the name of the outlet. The subtext is the group the outlet is associated with. The colored circle is the status of the outlet.

There are three possible statuses for the outlet:

- red – The outlet is deactivated.
- green – The outlet is activated but not currently in use.
- yellow – The outlet is activated and in use.

To refresh the list of outlets, pull down on the list as shown in Figure 4.

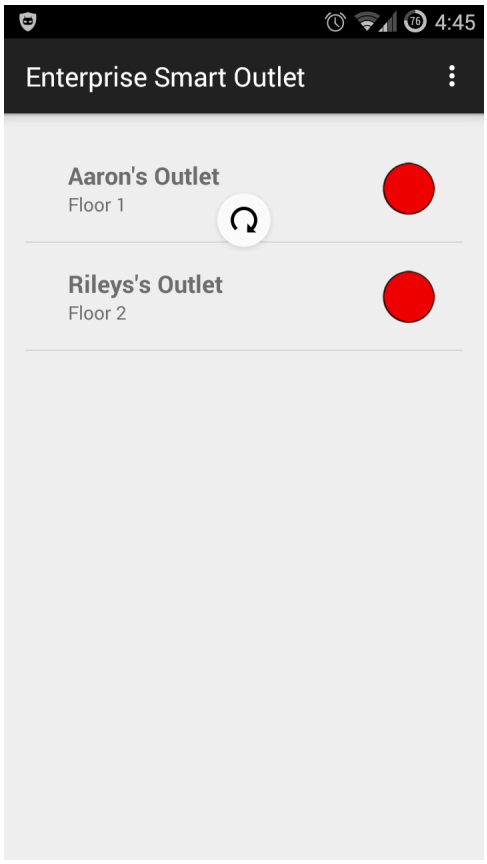

Figure 4: Refreshing Outlets Main Screen

Once you release the refresh listener, it will display the current data in the database for the outlets as shown in Figure 5.

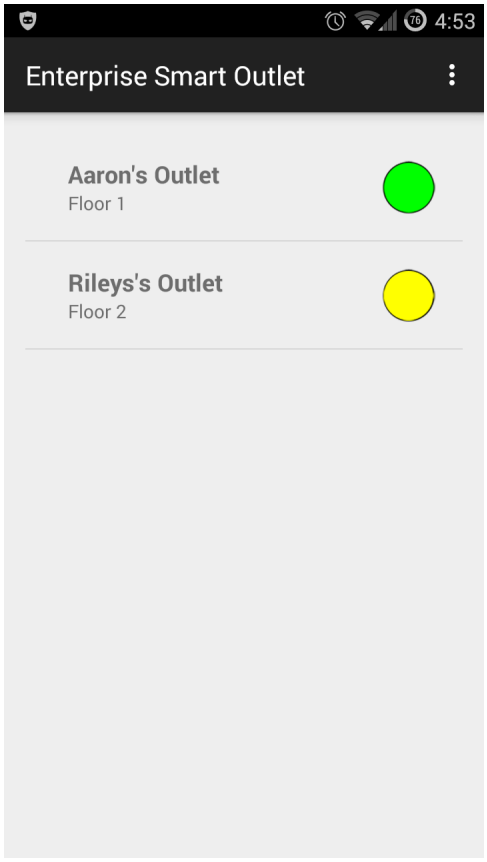

Figure 5: Activated Outlets Main Screen

The data for the outlets has been updated and the status icon of each outlet reflects the outlet's current status.

3.3 Detail Screen

The detailed view shows information specific to one outlet.

There is an assortment of data presented on this screen:

- Name – The name of the outlet.
- Group – The group the outlet is associated with.
- Voltage – The most recent voltage reading from the outlet.
- Current – The most recent current reading from the outlet.
- Status – If the outlet is inactive, in use, or active but not in use.
- Last Contact – The last time the outlet contacted the server.

There is also a toggle button to activate or deactivate the outlet from the app.

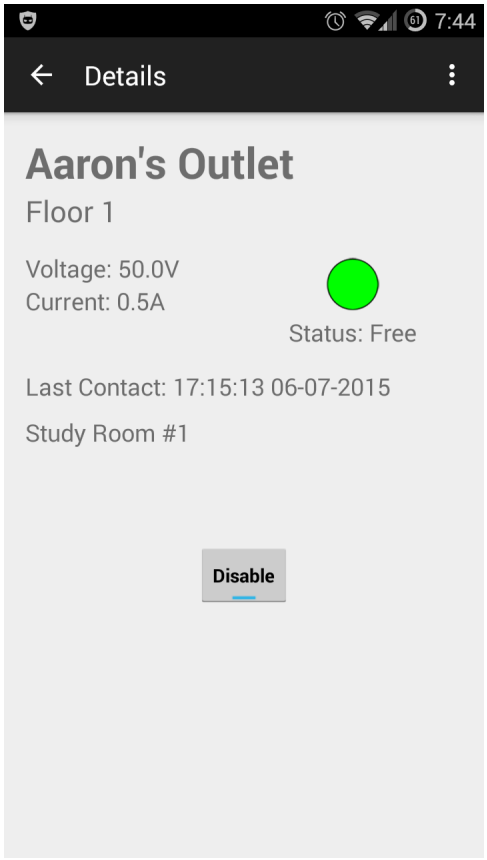

Figure 6: Detailed Screen of Free Outlet

Figure 6 shows the outlet from the list view that was active but not in use. When the toggle button is clicked, it sends a command to the server to send a command to the outlet to activate or deactivate depending on its status.

4 Technical Details

Figure 7 shows an overview of the connections between app, server, and outlet needed to function. The requests and transfers of data are explained further in this section.

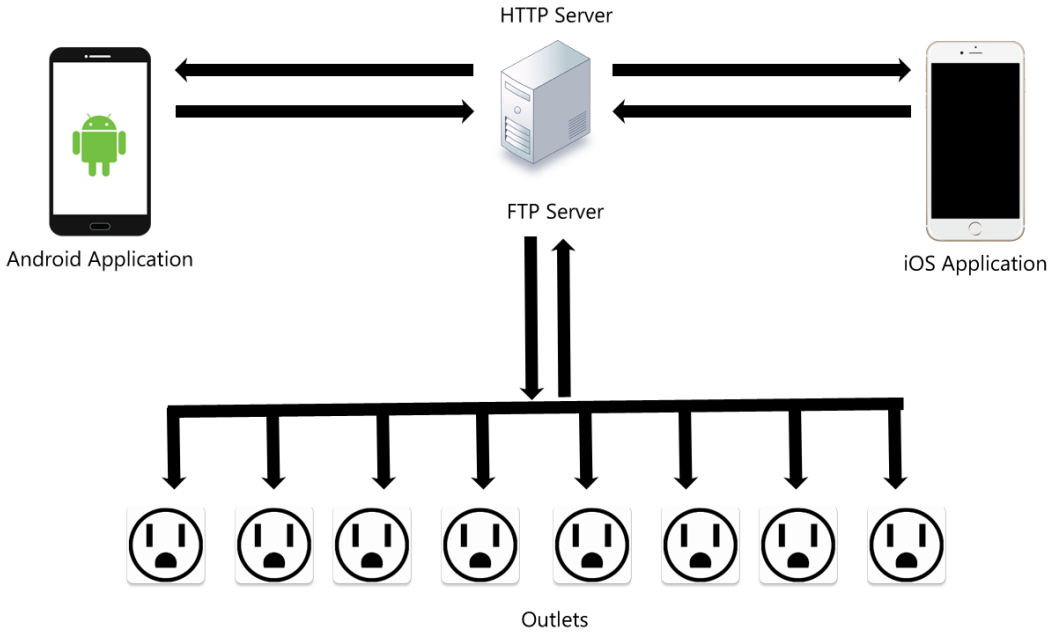

Figure 7: Enterprise Smart Outlet Diagram

4.1 Application Details

The following sub-sections explain how the backend for each screen works. An Intent is called to switch between screens, starting the activity of the next screen to be displayed. In Android, an intent is an abstract description of an operation to be performed [4].

4.1.1 Application Splash Screen

On the creation of the instance of the application, the default action of the application is displayed. In the manifest of the app, the default action is set to be the splash

screen. The splash screen displays the logo of the app for three seconds before transitioning to the main activity of the app; the list view of outlets.

4.1.2 Application Main Screen

For the main screen of the app, I display the list of outlets connected to the server. On the start of the post-splash screen process, I send a GET request to the server in the form of an AsyncTask to get an updated list of outlets on the app. The URL is built app side depending on the IP of the server. A future update to the app would use mDNS to obtain the IP of the server. Multicast DNS resolves a host name by sending an IP multicast query message that asks the host having that name to identify itself [6]. If the request does not return anything, nothing happens. However, if the server returns JSON data on the outlets, the Realm database is cleared and the JSON is sent to another process to be parsed and formatted into Realm outlet objects. The Realm outlet objects are stored in the Realm database. More details on the server side of this process in Section 4.2.1.

Once I have Outlet objects, I use a custom list adapter that extends RealmBaseAdapter to format my outlet data. RealmBaseAdapter is an abstract utility class for binding UI elements to Realm data [12]. I bind the outlet name and group to the list item and, depending on the status of the outlet, display a red, yellow, or green status icon. The parameters for the colors are described in Section 3.2. The text items are displayed in TextViews and the image is displayed in an ImageView.

4.1.3 Application Detail Screen

When a list item is clicked, I set up an intent to show the detail screen and display all of the information on the outlet. I attach the MAC address of the outlet to the intent and query the Realm database from the detail fragment to get the rest of the information on the outlet. The outlet name, outlet group, voltage, current, last contact, description, and status texts are bound to TextViews. The status icon is displayed in an ImageView. At the bottom of the detail view is a ToggleButton. This ToggleButton sends a GET request to the server in the form of an AsyncTask to either enable or disable the outlet depending on its current status. The URL is built app side and includes a query to the server to either activate or deactivate the MAC address of the outlet. Currently, the app will always switch from sending an enable to disable or disable to enable after a button press. I originally had it based on the status of the outlet. When it was based on the status of the outlet, it required the server to receive the request, successfully send the activate or deactivate command to the outlet, receive updated data from the outlet saying its voltage and current

values were lower, then update the database and send that updated data back to the app. This process required the app to be waiting for new data and because we could not get current and voltage readings from the outlet, I made the decision to set the toggle press to be how I described. More details on the server side of this process in Section 4.2.1.

4.2 Server Details

4.2.1 HTTP Server

The HTTP server is the interface between the MySQL database and the mobile applications and between the mobile applications and a FTP client socket to an outlet. Before the HTTP server is started, a connection is made to the MySQL database so there is never a situation where you hit the server before that connection is made. The request the mobile application makes to get outlet data is simply querying the database for all outlet data and sorting the outlets based on their group in ascending order. Currently, the Android app does not support refreshing individual outlets because there wasn't a need for it with the amount of outlets we were testing with. However, the server does have a query for a specific MAC address that would return data on just that outlet. Once the database query has been formatted, it is executed and the resulting row or rows of data are directly converted to JSON using the function JSON.stringify() [9]. The JSON is packed into the response to the request.

The request to activate or deactivate an outlet is a bit more complicated. The request takes the form of a query to activate or deactivate the MAC address of the outlet. If it is an activate query, the command sent to the outlet is a '1'; if it is a deactivate query, the command sent to the outlet is a '0'. The MAC address is used to query an IP lookup table in the MySQL database and with the IP address returned, an FTP socket client is created to send the command to the outlet. If the outlet successfully received the command, then the status of the outlet in the database is changed to reflect the successful transfer.

4.2.2 FTP Server

The FTP server is the interface between the outlets and the MySQL database. As mentioned in Section 4.1.2, we do not use any form of DNS so we used static IPs for the server and outlets. The server must be running before the outlet is turned on to capture the add string that they send when they start up. The outlet sends a string that contains its MAC address, name, group, and description to the servers' IP on

startup to make sure it is always there. When the server receives this command, it adds the outlet to the table database. It also adds the MAC address of the outlet and the IP address of the outlet to a separate table to be used when sending commands from the HTTP server to the outlet. The server is also capable of receiving update strings from the outlets to update their voltage and current values, but as mentioned before, we never got those values.

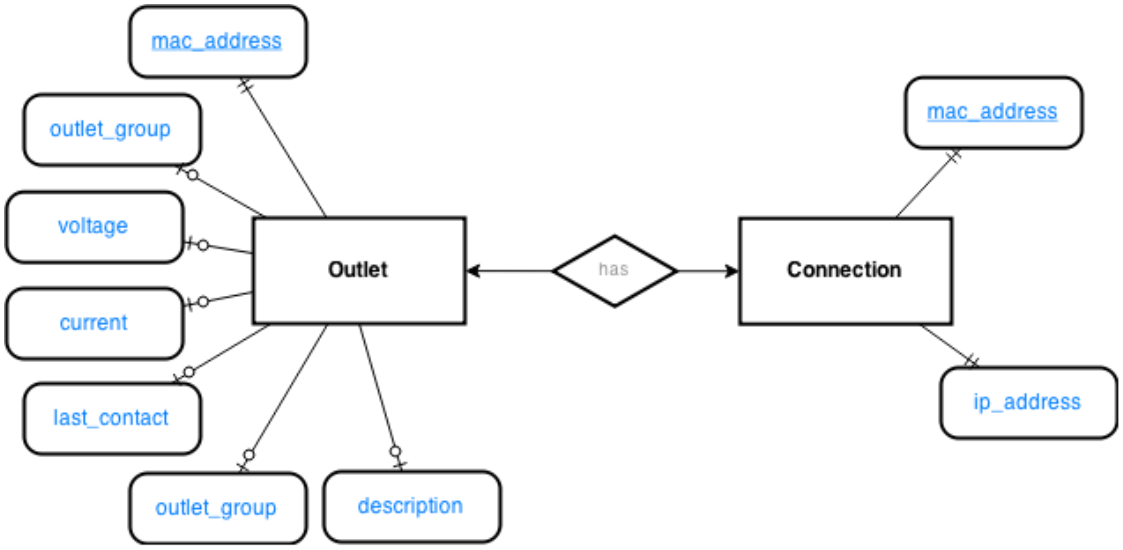

Figure 8: Enterprise Smart Outlet Entity Relationship Diagram

As shown in Figure 8, the Entity Relationship model for the database is simple, but gets the job done. There is a one-to-one relationship between Outlet and Connection. Normally, a one to one relationship would not call for a separate table, but I felt the IP address should not be in the same table as less protected information.

5 Conclusion and Future Work

As someone who previously had limited experience writing Android apps and no experience with servers of any kind, I learned a great amount on the development process of an Android app and using different networking protocols. I also found it humbling to work with teammates who have a different discipline than my own. Working with EE major David Faltemier on getting the FTP connection between server and outlet to work was an interesting experience because I didn't know how the WiFi chip was programmed, and he had no idea how my Node.js server worked. I enjoyed working with iOS developer Riley McGovern on the design of our apps and the functionality we desired. It was also the first time where the content of the project dictated the language I implemented in. Classes normally specify the language they want a project implemented in. For this project, I had the freedom to find the language that made server implementation easiest. I decided on Node.js after doing some research of my own and consulting my advisor, Professor Keen.

Regarding future work on the project, the EE majors are all going their separate ways with seemingly no interest to continue with development on the hardware. I enjoyed my work on the Android application and servers but with no updates to the hardware, there is no point in updating the software to support that hardware.

References

[1] AppBrain. (2014) Top android sdk versions. [Online]. Available: http://www.appbrain.com/stats/top-android-sdk-versions

[2] R. P. Foundation. (2015) Welcome to raspbian. [Online]. Available: http://www.raspbian.org/

[3] ——. (2015) What is a raspberry pi? [Online]. Available: https://www.raspberrypi.org/

[4] Google. (2015) Intent. [Online]. Available: https://developer.android.com/reference/android/content/Intent.html

[5] ——. (2015) The official android ide. [Online]. Available: https://developer.android.com/sdk/index.html

[6] A. Inc. (2012) Multicast dns. [Online]. Available: http://files.multicastdns.org/draft-cheshire-dnsext-multicastdns.txt

[7] I. Joyent. (2015) Node.js. [Online]. Available: https://nodejs.org/

[8] JSON. (1999) Ecma-404 the json data interchange standard. [Online]. Available: http://json.org/

[9] MDN. (2015) Json.stringify(). [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify

[10] Oracle. (2015) Mysql. [Online]. Available: http://www.oracle.com/us/products/mysql/overview/index.html

[11] Realm. (2014) Introducing realm a mobile-first database. [Online]. Available: https://realm.io/news/introducing-realm/

[12] ——. (2015) Realmbaseadapter. [Online]. Available: https://realm.io/docs/java/0.79.0/api/io/realm/RealmBaseAdapter.html