Senior Project Paper

Da Hydro App

Fall 2015

# **Table of Contents**

# Introduction Pg2

# Planning and Analysis Pg2

# Implementation Pg4

# Trials and Tribulations Pg8

# What I’ve Learned Pg9

Introduction

I graduated high school over a decade ago in 2002. Before that I had been interested in networking and technology in general. Just after graduating I went out and earned a CCNA and enrolled in college at NMU. Network Computing sounded like the ideal major for me so I selected it. After a couple semesters I thought it wasn’t the right pick for me and changed my major to Mechanical Engineering. I changed majors once again after that and finally decided it was best for me to get on with life and stop going to school since I couldn’t settle on a specific path and it was costing me a lot of money that I didn’t have.

In 2011, I was working in a factory and slowly working my way up the ranks. It wasn’t a bad gig, but I wanted something more for myself and for my family and I realized that it was time to go back to school. I was still interested in technology and computers so I went to the University of Wisconsin, Green Bay and studied Computer Science full time while trying to also work full time. I failed miserably at doing this and my wife and I were offered a place to live rent free if I moved back home to the U.P. and finished school at NMU. So here I am, a few years later, and all that stands between me and my degree is this Senior Project.

Planning and Analysis

I’d been trying to think of a topic for my project for well over a year and I had a very difficult time settling on this project. My original plan was to make a program to assist in brewing beer. Mainly because it was listed as one of the options on the Senior Project webpage. I gave the idea a lot of thought and eventually decided that it might be too difficult to buy all of the equipment needed for brewing beer, not to mention teach myself how to use it all. Not long after that I was trying to think of a project that would benefit somebody else other than myself and asked my sister if she was interested in me doing something for her store in Ishpeming. She has a hydroponics store that is located in the Country Village in Ishpeming called Da Hydro Shop.

I asked my sister if she thought she needed any kind of website or anything like that and she wasn’t interested in anything involving e-commerce. We threw a few ideas back and forth and then tried looking at it from a wider perspective. What exactly were the stores weaknesses and was there any way that we could attempt to remedy those weaknesses? The first weakness that I thought of was that the store didn’t really have much in the way of advertising. If you don’t already know about the store, the only way to learn of it is see it as you drive by or by word of mouth from current customers. The store has a Facebook page, but it is rarely updated so people just dismiss it.

So it seemed like doing some kind of online web app would be a good idea. It would allow people to learn about the store by using the internet. It’s not straight up advertising but it might bring a few more customers in over a period of time. It also wasn’t going to cost the store anything to have me develop an app for them. So there wasn’t any risk involved there. My sister gave me a few ideas about what the app should allow customers to do and the major thing that we came up with was to allow the customers to put together a bunch of products that are available in her store and basically build a “grow room” from scratch. From there I decided to play on some of the strengths that her store already has and allow the app to assist in those strengths. Their customer service is what makes them competitive with online stores and other stores in the area so I thought that a good idea would be to allow customers to ask questions directly through the app. Then the store could respond to the questions via email. I determined that customers wouldn’t have to log in to the app because having a log in will deter some people from using it at all. But there had to be a way to distinguish regular users from the store owners and employees so there needed to be a login for the store’s owners.

Between all of the thoughts going back and forth between us, I decided that I really wanted to learn how to make an application for mobile devices. I asked her if she would be okay if the app was transitioned from a web app to a mobile app and she was all for the idea. I think she liked it even more, in fact.

So I had a good idea of what I wanted to do but before I started coding anything for the app I had to do a bit of project design and I had to learn how to do Android development. A lot of the project design took place in my head but I did manage to jot down some use cases and UI mock ups. It was beneficial for me to get some of my thoughts in order before starting but I wish I had been a little more organized about it and fleshed it out even a little more.

For my use cases I came up with three actors: Owners, Customers, and the Point Of Sale system. The use cases that I originally came up with were the following:

Owners:

* Send Notifications
* View Customer Messages
* Log In

Customers:

* Browse Items
* Send Messages
* Build a Room
* Compare Items

Point Of Sale:

* Send current inventory to App System
* Send item prices to App System

The use cases were beneficial in conceptualizing what the program really needed to do and even how it should look and feel. They also helped to focus my concentration in one place at a time instead of trying to do everything at once.

There ended up being an Android Programming class offered at the same time that I was going to be doing my senior project. The class helped a lot with learning Android development, but if I hadn’t taken the class I think that I would’ve started on the project a bit earlier instead of waiting to learn about how to do things in class. Because of waiting to learn what I needed in the class, it didn’t give me a whole lot of time to finish my project and I think my project suffered for it. I also didn’t want to just make an application based entirely off of what was learned in class. But with time running out there wasn’t a lot for me to learn on my own other than that.

Implementation

Implementation was obviously the hardest and most time consuming part of the process. There were several hiccups and just about nothing went according to plan which is probably a common occurrence in senior projects. The store uses a software called QuickBooks POS Basic for its sales and inventory management. The important part of that title is the word Basic. Since it was the Basic version and not the Pro version there were absolutely no networking capabilities with the software. It would have cost about $2000 to upgrade the software, which was out of the question. When I submitted my proposal I wasn’t aware of this fact and thought I would be able to use the QuickBooks Software Development Kit to integrate my app with their system. I had no such luck.

So I came up with a solution that wouldn’t allow for as much dynamic content in the app as I’d hoped but it would still allow for the app to function at some basic level. The POS system allowed the inventory and prices to be exported to an excel spreadsheet, so I had my sister export the inventory and I then converted it to a CSV file. Android has a folder in its project structure that allows you to store assets. An asset in Android is a raw data file, such as a text or music file, which can be used by your application. In order to use the asset, you need to create an AssetManager which treats assets like a file system. You can then call the AssetManager.open() method specifying the name of the file as the argument to the method. The following code demonstrates how this is accomplished.

**private** AssetManager **mManager**;

.

.

.

String csvFile = **"CSVPOS.csv"**;  
InputStream inStream = **null**;  
**try** {  
 inStream = **mManager**.open(csvFile);  
} **catch** (IOException e) {  
 e.printStackTrace();  
}  
BufferedReader buffer = **new** BufferedReader(**new** InputStreamReader(inStream));

I hadn’t learned about AssetManagers in the Android Programming class so it was nice to find something on my own that I could use in the project. After reading the file in line-by-line, I then separate it on its commas and put the values into a SQLite database that is supported by Android. So this method limits the abilities for the app to update the inventory on the fly but it still shows what items are normally in the store and what items would need to be ordered.

Storing the values into a database table is actually quite convenient. It is accomplished by using a ContentValues object. This object holds key-value pairs, the key being a table column name and the value being the field that you want to enter into the table under that column. After storing your values, you call an insert() on the database object giving it the arguments of a table name, something called a nullColumnHack, and finally the ContentValues object. You can specify null for the nullColumnHack, which is what I did, otherwise if you know the name of a column that you might not be setting a value for, you can specify that column’s name for this argument and it will automatically enter NULL wherever you didn’t provide a value for that column.

I handled all of my database transactions by using a DatabaseHelper class which extends SQLiteOpenHelper. This allows me to use the same database from any Activity that needs it. I accomplish this with the following line of code:

SQLiteDatabase mDatabase = **new** DatabaseHelper(**this**).getWritableDatabase();

You have to specify the context when opening the database. A context basically just tells new objects what is currently going on in the application. So by specifying ‘this’ as the context I am telling the DatabaseHelper which Activity I am in.

I used the item table from the database in a couple different places. The first thing I used it for was my comparison activity. I wanted to use drop down lists for selecting which items to compare. To accomplish this I used Spinner objects. A Spinner is populated by using a SpinnerAdapter. I created methods that provide an ArrayAdapter<String>, which is a subclass of SpinnerAdapter, for my Spinners. The ArrayAdapters are given a context, a resource ID for the layout, and finally a List of objects to display.

ArrayAdapter<String> spinnerAdapter = **new** ArrayAdapter<>(CompareActivity.**this**,  
 android.R.layout.***simple\_spinner\_item***, itemTypes);

Running a query from the database to load the ArrayList for itemTypes in the above example is pretty straight forward. The database object has a method called rawQuery() which allows for passing a String containing a SQLite query. The method returns a Cursor for the rows of results from the query. The following snippet is my implementation of loading the ArrayAdapter with the first column of each row in the results:

Cursor cursor = mDatabase.rawQuery(sql, **null**);  
**try** {  
 **for** (cursor.moveToFirst(); !cursor.isAfterLast(); cursor.moveToNext()) {  
 itemTypes.add(cursor.getString(0));  
 }  
 ArrayAdapter<String> spinnerAdapter = **new** ArrayAdapter<>(ByorActivity.**this**,  
 android.R.layout.***simple\_spinner\_item***, itemTypes);  
 spinnerAdapter.setDropDownViewResource(android.R.layout.***simple\_spinner\_dropdown\_item***);  
 **return** spinnerAdapter;  
} **finally** {  
 cursor.close();  
}

However, I think that there may have been a simpler way of implementing the Spinners. There is another subclass of SpinnerAdapter called CursorAdapter. A CursorAdapter would have allowed me to load the adapter directly from the database object rather than querying the database and storing the results into an array and then populating the adapter with them. I will likely end up changing the code to use CursorAdapters instead.

What I thought was kind of neat about the spinners I use, though, is that some of them are populated depending on the value of another. For instance, I let the customer decide, with a spinner, whether they want to see all items, or just items available at the store. Within this spinner’s OnItemSelectedListener anonymous inner class I populate the other spinners with their items. The way that I accomplish this is by changing part of the String for the SQL query based on which item is selected:

**if**(optionSelected.equals(***ALLITEMS***)){  
 **mSqlArgs** = **"' GROUP BY itemName"**;  
}  
**else**{  
 **mSqlArgs** = **"' AND quantity > 0 GROUP BY itemName"**;  
}

The String above is concatenated onto the end of the rest of the SQL query which is specified in another spinner’s OnItemSelectedListener:

**itemSql** = **"SELECT itemName from "** + ***ITEMTABLE*** + **" WHERE department='"** + **mTypeSelected** + **mSqlArgs**;

I only use two tables in my database. The first is for the inventory items and the second is a much smaller table that holds the owners’ usernames and md5 encrypted passwords for when they log in. When the owner logs in I hash the password input to md5 and check it against the md5 encrypted password that is stored in the database. I actually used code that I found on stack overflow to generate the md5 hash. The code uses a MessageDigest which turns a byte sequence with any length to a byte sequence with a fixed length. The method then formats the bytes as a hexadecimal String. The following three lines are the bit of code that converts the bytes into a fixed length:

MessageDigest md = MessageDigest.*getInstance*(**"md5"**);  
md.update(passwd.getBytes());  
  
**byte**[] bytes = md.digest();

It was nice to find somebody else’s code here and use it rather than spending a lot of time figuring it out on my own when there was already an easy way to do it.

It was also nice to use somebody else’s library for networking with Android. I ended up using a library called Volley which turns out to be very nice. It handles all of the networking on a separate thread than the UI so I don’t have to worry about locking up the UI thread when it is connecting to a server or sending and retrieving data. So, using volley, I make http requests to get and post the app’s messages and notifications from a PHP script on a webserver. I also really didn’t want to turn in this app without it having some sort of networking capabilities since I am a Network Computing major.

To make a request using volley, you have to have a RequestQueue that you add request’s to. The requests that I make in my app are StringRequests, but volley also supports JSONObjectRequests as well. A StringRequest requires a request method such as GET or POST, a URL as a String, a Response.Listener object, and a Response.ErrorListener object. Apparently anonymous inner classes are used a lot in Android development. This is one of many fine examples of when they are used. Behold, the code for a volley StringRequest:

StringRequest request = **new** StringRequest(Request.Method.***POST***, ***URL***,  
 **new** Response.Listener<String>() {  
 @Override  
 **public void** onResponse(String s) {  
 **if**(!s.isEmpty())  
 Toast.*makeText*(getApplicationContext(), **"Notification From DaHydro Shop: "** + s, Toast.***LENGTH\_LONG***).show();  
 }  
 },  
 **new** Response.ErrorListener() {  
 @Override  
 **public void** onErrorResponse(VolleyError volleyError) {  
 Toast.*makeText*(getApplicationContext(), **"Error: "** + volleyError.getMessage(), Toast.***LENGTH\_LONG***).show();  
  
 }  
 }  
);

So this request then gets added to the RequestQueue and I can forget about it and let Volley do the rest. When volley gets a response, the onResponse() method will be triggered and my code for displaying the Notification from the owners will be executed.

The server side of these connections was implemented using php and are quite simple. They were set up strictly for testing the app out right now and if the app is deployed there will need to be a more permanent solution. I created a new database on the department’s Euclid server with a couple of tables. One for messages and one for notifications. The following is my table structure for the messages:

Message Table:

+-------------+--------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-------------+--------------+------+-----+---------+-------+

| email | varchar(50) | NO | | NULL | |

| messageText | varchar(255) | NO | | NULL | |

+-------------+--------------+------+-----+---------+-------+

I then made two php files for each table that the app can get requests from. One php file for updating the table and one for just reading from the table. The following is some example php code for reading from the database and returning a value:

$db = mysqli\_connect("localhost", "mkinnune", "", "hydroshop") or die("Failed to connect to database" . mysqli\_connect\_error());

@$index = $\_REQUEST['index'];

$result = mysqli\_query($db, "select \* from message;");

$messageText;

$email;

if(mysqli\_num\_rows($result) < $index+1){

for($i = 0; $i < mysqli\_num\_rows($result); $i++)

{

$row\_array = mysqli\_fetch\_row($result);

$email = $row\_array[0];

$messageText = $row\_array[1];

echo($email . " - ");

echo($messageText . "\n");

}

}else{

$j = 0;

for($i = 0; $i < mysqli\_num\_rows($result)-$index; $i++)

{

$row\_array = mysqli\_fetch\_row($result);

$j++;

}

for($i = 0; $i < mysqli\_num\_rows($result)-$j; $i++)

{

$row\_array = mysqli\_fetch\_row($result);

$email = $row\_array[0];

$messageText = $row\_array[1];

echo($email . " - ");

echo($messageText . "\n");

}

}

}

You can specify parameters when using the POST method with Volley. So when requesting from this php file, you specify how many of the results you would like to see which is then stored into the index variable in the script. Querying a database in php is actually quite similar to doing it in java. The query returns a cursor holding the rows of data from the query result. Then you can loop through the rows and get the fields that you need. Inserting into the database from the app is quite easy as well. You use the parameters for POST in volley to send the data to the server and then in the php script you store the parameters as variables and do an insert to the database like so:

mysqli\_query($db, "insert into message values(\"$email\", \"$messageText\");");

Aside from all of the java code, and the little bit of php and sql, there was a lot of xml used. XML is used in the resources for Android. They are essentially the code for the View portion of the MVC pattern. There really isn’t much that is interesting about it, but I thought I should mention it anyways since it is something that I set out to learn with this project. So for each Activity I have a resource layout file which creates the view in Java with one line:

setContentView(R.layout.***activity\_main***);

The R.layout.activity\_main is actually an int that refers to the xml file with the specified ID of activity\_main. Within the xml file is where all of the attributes related to the layout of the UI are stored. So for each Button, or TextView, or Spinner, there will need to be a layout that has those components specified with their relation to other components along with their size and other attributes. The following is an example of the xml for a Spinner:

<**Spinner  
 android:id="@+id/item\_select\_spinner"  
 android:prompt="@string/onhand\_text"  
 android:layout\_width="fill\_parent"  
 android:layout\_height="wrap\_content"**>  
</**Spinner**>

The xml portion of the project is definitely the easiest to write and shouldn’t contain any of the program logic. It is strictly for views. It is similar to html and css for a website in this regard.

Trials and Tribulations

I mentioned earlier about some of the problems that I came across. The most challenging was the problem that the POS system was of almost no use to me at all when I was hoping to integrate my app with it. I probably spent a week or two trying to find a work-around and I am not pleased with what I ended up deciding to do. If I had been able to overcome this issue in some other way I might have been able to get myself the points to put me into the A range on my grading scale.

After that issue, somewhere in the middle of my project, and after spending a few days trying, I decided that I didn’t have enough time to do a Graphical Interface with the Build Your Own Room function. I think it was something that would require a lot of creativity and a lot more time than I actually had. It might be something that I am able to do in the future someday, but right now it just wasn’t worth the 3 points. I had started to implement it with a Canvas and some shapes depending some specs that the user would have to input such as the size of the room and the orientation of lights. It turned out to look pretty bad though and I realized that it wasn’t my forte.

Another issue that I didn’t even attempt to solve was the item descriptions. The stores POS doesn’t keep any descriptions of the items in its tables and in the beginning I was planning on maybe getting them from a website of a company that they get most of their products from. But by the time I was able to get to figuring out how to accomplish something like this, I was pretty much out of time. I still haven’t looked into it and it might be something that is of a large enough scale to be another project in and of itself.

I had some other hiccups that I was able to eventually get done though. One of which was the messaging to the owners. I didn’t really know how to approach it at first and I thought that I could maybe just have a fileserver with a text file that I might be able to read and write to from the app. Then, with a little push in the right direction, I found it was a lot easier to use some knowledge that I already had from my Networking and Advance Web classes and use PHP and MySQL on a webserver. I didn’t have the resources to set up a permanent server in the store yet (I might still do so in the future), so I used a temporary resource, Euclid.

The final major hiccup I had was that I didn’t know any Android development before-hand. Java is something that I was comfortable with, but Android has so many quirks and built in tools that it’s almost like it is its own language. Taking the Android class was really helpful in this regard. But I ended up waiting to start coding my project until I felt the class gave me a good enough base and I should have started much sooner, even without necessarily knowing what I was doing. I could have learned as I went along.

What I’ve Learned

I’m not sure that I would have taken the Android Programming course if not for this project so it’s hard to say that this project taught me how to program in Android or not. But there are definitely other things that I learned from the project that aren’t strictly Android things.

One thing that I learned is how hard and time consuming a project like this can be, no matter what language or technology is being used. 1000 lines of code is nothing to laugh about or dismiss. I don’t believe that my project in particular is very complex but it was still pretty tough to reach that milestone. I also wish I had given myself a little more time to get there.

Which leads me into another items of importance, and that is to start coding earlier, even if you don’t necessarily know what you are doing. I think that sitting at the computer and typing some code in can get your brain working and it’s very possible to find out how to do something that you might not have already known how to do. If I had done more of this I would have had a lot more time to maybe complete another item or two on my proposal.

Finally, there were plenty of other more technical and code specific items that I learned. For instance, without Android I don’t think I would know about anonymous inner classes and I think that they are pretty interesting and they make a lot of sense now where they didn’t when I first started out. I also learned how to combine some different technologies to accomplish a single task, like using sql in a java application. It was also beneficial to find how to use other people’s code in my own applications. I spent a lot of time reading developer documents and stack overflow when I didn’t really know what to do and it is something that I am sure I will continue to do in the future.